DevOps TROUBLESHOOTING

Linux® Server Best Practices

KYLE RANKIN

DevOps Troubleshooting

This page intentionally left blank

DevOps Troubleshooting

Linux® Server Best Practices

Kyle Rankin

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Cataloging-in-Publication Data is on file with the Library of Congress.

ISBN-13: 978-0-321-83204-7
ISBN-10: 0-321-83204-3

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, November 2012

Editor-in-Chief
Mark Taub

Executive Editor
Debra Williams Cauley

Development Editor
Michael Thurston

Managing Editor
John Fuller

Project Editor
Elizabeth Ryan

Copy Editor
Rebecca Rider

Indexer
Richard Evans

Proofreader
Diane Freed

Technical Reviewer
Bill Childers

Publishing Coordinator
Kim Boedigheimer

Compositor
Kim Arney

This book wouldn't be possible without the support of my wife, Joy, who once again helped me manage my time so I could complete the book, only this time while carrying our first child, Gideon. I'd also like to dedicate this book to my son, Gideon, who so far is easier to troubleshoot than any server.

This page intentionally left blank

Contents

Preface xiii

Acknowledgments xix

About the Author xxi

CHAPTER 1 Troubleshooting Best Practices 1

Divide the Problem Space 3
Practice Good Communication When Collaborating 4
Conference Calls 4
Direct Conversation 5
Email 6
Real-Time Chat Rooms 7
Have a Backup Communication Method 8
Favor Quick, Simple Tests over Slow, Complex Tests 8
Favor Past Solutions 9
Document Your Problems and Solutions 10
Know What Changed 12
Understand How Systems Work 13
Use the Internet, but Carefully 14
Resist Rebooting 15

CHAPTER 2 Why Is the Server So Slow? Running Out of CPU, RAM, and Disk I/O 17

System Load 18
What Is a High Load Average? 20
Diagnose Load Problems with top 20
Make Sense of top Output 22
Diagnose High User Time 24
Diagnose Out-of-Memory Issues 25
Diagnose High I/O Wait 27
Troubleshoot High Load after the Fact 29
Configure sysstat 30
View CPU Statistics 30

View RAM Statistics 31
View Disk Statistics 32
View Statistics from Previous Days 33

CHAPTER 3 Why Won't the System Boot? Solving Boot Problems 35

The Linux Boot Process 36
The BIOS 36
GRUB and Linux Boot Loaders 37
The Kernel and Initrd 38
/sbin/init 39
BIOS Boot Order 45
Fix GRUB 47
No GRUB Prompt 47
Stage 1.5 GRUB Prompt 48
Misconfigured GRUB Prompt 49
Repair GRUB from the Live System 49
Repair GRUB with a Rescue Disk 50
Disable Splash Screens 51
Can't Mount the Root File System 51
The Root Kernel Argument 52
The Root Device Changed 52
The Root Partition Is Corrupt or Failed 55
Can't Mount Secondary File Systems 55

CHAPTER 4 Why Can't I Write to the Disk? Solving Full or Corrupt Disk Issues 57

When the Disk Is Full 58
Reserved Blocks 59
Track Down the Largest Directories 59
Out of Inodes 61
The File System Is Read-Only 62
Repair Corrupted File Systems 63
Repair Software RAID 64

CHAPTER 5 Is the Server Down? Tracking Down the Source of Network Problems 67

Server A Can't Talk to Server B 68
Client or Server Problem 69
Is It Plugged In? 69

Is the Interface Up? 70
Is It on the Local Network? 71
Is DNS Working? 72
Can I Route to the Remote Host? 74
Is the Remote Port Open? 76
Test the Remote Host Locally 76
Troubleshoot Slow Networks 78
DNS Issues 79
Find the Network Slowdown with traceroute 80
Find What Is Using Your Bandwidth with iftop 81
Packet Captures 83
Use the tcpdump Tool 84
Use Wireshark 88

CHAPTER 6 Why Won't the Hostnames Resolve? Solving DNS Server Issues 93

DNS Client Troubleshooting 95
No Name Server Configured or Inaccessible Name Server 95
Missing Search Path or Name Server Problem 97
DNS Server Troubleshooting 98
Understanding dig Output 98
Trace a DNS Query 101
Recursive Name Server Problems 104
When Updates Don't Take 107

CHAPTER 7 Why Didn't My Email Go Through? Tracing Email Problems 119

Trace an Email Request 120
Understand Email Headers 123
Problems Sending Email 125
Client Can't Communicate with the Outbound Mail Server 126
Outbound Mail Server Won't Allow Relay 130
Outbound Mail Server Can't Communicate with the Destination 131
Problems Receiving Email 135
Telnet Test Can't Connect 136
Telnet Can Connect, but the Message Is Rejected 137
Pore Through the Mail Logs 138

CHAPTER 8	**Is the Website Down? Tracking Down Web Server Problems**	**141**
	Is the Server Running?	143
	Is the Remote Port Open?	143
	Test the Remote Host Locally	144
	Test a Web Server from the Command Line	146
	Test Web Servers with Curl	146
	Test Web Servers with Telnet	148
	HTTP Status Codes	149
	1xx Informational Codes	150
	2xx Successful Codes	150
	3xx Redirection Codes	151
	4xx Client Error Codes	152
	5xx Server Error Codes	153
	Parse Web Server Logs	154
	Get Web Server Statistics	158
	Solve Common Web Server Problems	163
	Configuration Problems	163
	Permissions Problems	164
	Sluggish or Unavailable Web Server	166
CHAPTER 9	**Why Is the Database Slow? Tracking Down Database Problems**	**171**
	Search Database Logs	172
	MySQL	173
	PostgresSQL	173
	Is the Database Running?	174
	MySQL	174
	PostgresSQL	175
	Get Database Metrics	177
	MySQL	177
	PostgresSQL	179
	Identify Slow Queries	182
	MySQL	182
	PostgresSQL	183

CHAPTER 10 It's the Hardware's Fault! Diagnosing Common Hardware Problems 185

The Hard Drive Is Dying 186
Test RAM for Errors 190
Network Card Failures 191
The Server Is Too Hot 192
Power Supply Failures 194

Index 197

This page intentionally left blank

Preface

DevOps describes a world where developers, Quality Assurance (QA), and systems administrators work more closely together than in many traditional environments. Although DevOps is already recognized as a boon to rapid software deployment and automation, an often-overlooked benefit of the DevOps approach is the rapid problem solving that occurs when the whole team can collaborate to troubleshoot a problem on a system. Unfortunately, developers, QA, and sysadmins have gaps in their troubleshooting skills that they often resolve by blaming each other for problems on the system. This book aims to bridge those gaps and guide all groups through a standard set of troubleshooting practices that they can apply as a team to some of the most common Linux server problems.

Although the overall topics covered in the book are traditionally the domain of sysadmin, in a DevOps environment, developers and QA also find themselves troubleshooting network problems, setting up web servers, and diagnosing high load, even if they may not have a background in Linux administration. What makes this book more than just a sysadmin troubleshooting guide is the audience and focus. This book assumes the reader may not be a Linux sysadmin, but instead is a talented developer or QA engineer in a DevOps organization who may not have much system-level Linux experience. That said, if you are a sysadmin, you won't be left out either. Included are troubleshooting techniques that can supplement the skills of even senior sysadmin—just written in an accessible way.

In a traditional enterprise environment without DevOps principles, troubleshooting is as dysfunctional as development is. When there is a server problem, if you can even get developers and sysadmin on the same call, you can expect everyone to fall into their traditional roles—the sysadmin will only look at server resources and logs; the developers will wait for the inevitable blame to be heaped on them for their "bloated" or "buggy" code, at which point they will complain about the unstable, underpowered server; or maybe everyone will redirect the blame at QA for not finding the problem before it hit production. All the while, the actual problem is not any closer to being solved.

In a DevOps organization, cooperation between all the teams is stressed, but when it comes to troubleshooting, often people still fall into their traditional roles even if there's no blame game. Why? Well, even if everyone wants to work together, without the same troubleshooting skills and techniques, everyone may still be waiting on everyone else to troubleshoot their part. The goal of this book is to get every member of your DevOps team on the same page when it comes to Linux troubleshooting. When everyone has the same Linux troubleshooting skills, the QA team will better be able to diagnose problems before they hit production, developers will be better at tracking down why that latest check-in doubled the load on the system, and sysadmins can be more confident in their diagnoses, so when a problem strikes, everyone can pitch in to help.

This book is broken into ten chapters based on some of the most common problems you'll face on Linux systems, and the chapters are ordered so that techniques you learn in some of the earlier chapters (particularly about how to diagnose high load and how to troubleshoot network problems) can be helpful as you get further into the book. That said, I realize you may not read this book cover-to-cover, but instead you will probably just turn to the chapter that's relevant to your particular problem. So when topics in other chapters are helpful, I will point you to them.

- **Chapter 1: Troubleshooting Best Practices** Before you learn how to troubleshoot specific problems, it may be best to learn an overall approach to troubleshooting that you can apply to just about any kind of problem, even outside of Linux systems. This chapter talks about general troubleshooting principles that you will use when you try specific troubleshooting steps throughout the rest of the book.
- **Chapter 2: Why Is the Server So Slow? Running Out of CPU, RAM, and Disk I/O** This chapter introduces troubleshooting principles that you will apply to one of the most common problems you'll have

to solve: Why is the server slow? Whether you are in QA and are trying to figure out why the latest load test is running much slower; you are a developer trying to find out if your program is I/O bound, RAM bound, or CPU bound; or you are a sysadmin who isn't sure whether a load of 8, 9, or 13 is OK, this chapter will give you all the techniques you need to solve load problems.

- **Chapter 3: Why Won't the System Boot? Solving Boot Problems** Any number of different problems can stop a system from booting. Whether you have ever thought about the Linux boot process or not, this chapter helps you track down boot problems by first walking you through a healthy Linux boot process, and then discussing what it looks like when each stage in that boot process fails.
- **Chapter 4: Why Can't I Write to the Disk? Solving Full or Corrupt Disk Issues** Just about anyone who has used Linux for a period of time has run across a system where they can't write to the disk. It could be that you are a developer who enabled debugging in your logs and you accidentally filled the disk, or you could simply be the victim of file system corruption. In either case, this chapter helps you track down what directories are using up the most space on the system and how to repair corrupted file systems.
- **Chapter 5: Is the Server Down? Tracking Down the Source of Network Problems** No matter where you fit in a DevOps organization, network troubleshooting skills are invaluable. Sometimes it can be difficult to track down networking problems because they often impact a system in strange ways. This chapter walks you through how to isolate and diagnose a network problem step-by-step by testing problems on different network layers. This chapter also lays the groundwork for troubleshooting techniques for specific network services (such as DNS) covered in the rest of the book.
- **Chapter 6: Why Won't the Hostnames Resolve? Solving DNS Server Issues** DNS can be one of the trickier services to troubleshoot because even though so much of the network relies on it, many users are unfamiliar with how it works. Whether you are a web developer who gets DNS service for your site on a web GUI via your registrar, or a sysadmin in charge of a full BIND instance, these DNS troubleshooting

techniques will prove invaluable. This chapter will trace a normal, successful DNS request and then elaborate on the DNS troubleshooting covered in Chapter 5 with more specific techniques for finding problems in DNS zone transfers, caching issues, and even syntax errors.

- **Chapter 7: Why Didn't My Email Go Through? Tracing Email Problems** Email was one of the first services on the Internet and still is an important way to communicate. Whether you are tracing why your automated test emails aren't being sent, why your software's email notifications are stuck, or why mail delivery is down for your entire company, this chapter helps you solve a number of email problems, including misconfigured relay servers and DNS-related mail server issues. This chapter even shows you how to send an email "by hand" with telnet.
- **Chapter 8: Is the Website Down? Tracking Down Web Server Problems** So many of the applications we interact with on a daily basis are based on the Web. In fact, if you are a software developer, there's a good chance web programming is at least a part of what you develop, and if you are a sysadmin, you are likely responsible for at least one web server. Web server troubleshooting is a large topic, but for the purposes of this chapter, you only learn about the common problems you are likely to run into with two of the most popular web servers today: Apache and Nginx. This chapter discusses how to pull server status and how to identify the cause of high server load as well as other common debugging techniques.
- **Chapter 9: Why Is the Database Slow? Tracking Down Database Problems** Just like much of the software you use on a daily basis is on the Web, much of the software you use stores its data in some sort of database. This chapter is similar to Chapter 8, only its focus is on troubleshooting problems with two popular open source database servers: MySQL and PostgresSQL. As with Chapter 8, it discusses how to pull load metrics from these databases and how to identify problem queries as well as other causes of high load.
- **Chapter 10: It's the Hardware's Fault! Diagnosing Common Hardware Problems** With all this focus on software, we should also discuss one of the most common causes of server problems: hardware

failures. The problem with hardware failures is that often hardware doesn't fail outright. Instead, segments of RAM have errors, hard drive sectors fail, or Ethernet cards drop random packets. What's worse, these failures often cause software problems that are almost impossible to track down. This chapter discusses how to troubleshoot some common hardware failures, from bad RAM, to failing hard drives, to dying network cards. This chapter contains hardware troubleshooting techniques you can apply anywhere—from a production rackmount server to your personal laptop.

This page intentionally left blank

Acknowledgments

THANKS TO DEBRA for advocating for this book, from the first time the idea came up all the way through to it becoming a real book. Thanks also to Trotter and Bill for all of their feedback along the way. Finally, thanks to all of the broken systems I've worked on through the years that helped me hone my troubleshooting skills.

This page intentionally left blank

About the Author

Kyle Rankin is a senior systems administrator and DevOps engineer; the current president of the North Bay Linux Users' Group; author of *The Official Ubuntu Server Book*, *Knoppix Hacks*, *Knoppix Pocket Reference*, *Linux Multimedia Hacks*, and *Ubuntu Hacks*; and a contributor to a number of other books. Rankin is an award-winning columnist for *Linux*™ *Journal* and has written for *PC Magazine*, TechTarget websites, and other publications. He speaks frequently on open source software, including at SCALE, OSCON, Linux World Expo, Penguicon, and a number of Linux Users' Groups.

This page intentionally left blank

CHAPTER 5

Is the Server Down? Tracking Down the Source of Network Problems

MOST SERVERS ARE ATTACHED to some sort of network and generally use the network to provide some sort of service. Many different problems can creep up on a network, so network troubleshooting skills become crucial for anyone responsible for servers or services on those servers. Linux provides a large set of network troubleshooting tools, and this chapter discusses a few common network problems along with how to use some of the tools available for Linux to track down the root cause.

Network troubleshooting skills are invaluable for every member of a DevOps team. It's almost a given that software will communicate over the network in some way, and in many applications, network connectivity is absolutely vital for the software to function. When there is a problem with the network, everyone from the sysadmin, to the QA team, to the entire development staff will probably take notice. Whether your networking department is a separate group or not, when your entire DevOps team works together on diagnosing networking problems, you will get a better overall view of the problem. Your development team will give you the deep knowledge of how your software operates on the network; your QA team will explain how the application behaves under unusual circumstances and provide you with a backlog of networking bug history; and your sysadmin will provide you with an overall perspective of how networked applications work under Linux. Together you will be able to diagnose networking problems much faster than any team can individually.

Server A Can't Talk to Server B

Probably the most common network troubleshooting scenario involves one server being unable to communicate with another server on the network. This section will use an example in which a server named dev1 can't access the web service (port 80) on a second server named web1. Any number of different problems could cause this, so we'll run step by step through tests you can perform to isolate the cause of the problem.

Normally when troubleshooting a problem like this, you might skip a few of these initial steps (such as checking the link), since tests further down the line will also rule them out. For instance, if you test and confirm that DNS works, you've proven that your host can communicate on the local network. For this example, though, we'll walk through each intermediary step to illustrate how you might test each level.

Client or Server Problem

One quick test you can perform to narrow down the cause of your problem is to go to another host on the same network and try to access the server. In this example, you would find another server on the same network as dev1, such as dev2, and try to access web1. If dev2 also can't access web1, then you know the problem is more likely on web1, or on the network between dev1, dev2, and web1. If dev2 can access web1, then you know the problem is more likely on dev1. To start, let's assume that dev2 can access web1, so we will focus our troubleshooting on dev1.

Is It Plugged In?

The first troubleshooting steps to perform are on the client. You first want to verify that your client's connection to the network is healthy. To do this you can use the `ethtool` program (installed via the `ethtool` package) to verify that your link is up (the Ethernet device is physically connected to the network). If you aren't sure what interface you use, run the `/sbin/ifconfig` command to list all the available network interfaces and their settings. So if your Ethernet device was at eth0

```
$ sudo ethtool eth0
Settings for eth0:
     Supported ports: [ TP ]
     Supported link modes:   10baseT/Half 10baseT/Full
                             100baseT/Half 100baseT/Full
                             1000baseT/Half 1000baseT/Full
     Supports auto-negotiation: Yes
     Advertised link modes:  10baseT/Half 10baseT/Full
                             100baseT/Half 100baseT/Full
                             1000baseT/Half 1000baseT/Full
     Advertised auto-negotiation: Yes
     Speed: 100Mb/s
     Duplex: Full
     Port: Twisted Pair
     PHYAD: 0
     Transceiver: internal
```

```
Auto-negotiation: on
Supports Wake-on: pg
Wake-on: d
Current message level: 0x000000ff (255)
Link detected: yes
```

Here, on the final line, you can see that `Link detected` is set to `yes`, so dev1 is physically connected to the network. If this was set to `no`, you would need to physically inspect dev1's network connection and make sure it was connected. Since it is physically connected, you can move on.

NOTE `ethtool` has uses beyond simply checking for a link. It can also be used to diagnose and correct duplex issues. When a Linux server connects to a network, typically it autonegotiates with the network to see what speeds it can use and whether the network supports full duplex. The `Speed` and `Duplex` lines in the example `ethtool` output illustrate what a 100Mb/s, full duplex network should report. If you notice slow network speeds on a host, its speed and duplex settings are a good place to look. Run `ethtool` as in the previous example, and if you notice `Duplex` set to `Half`, then run

```
$ sudo ethtool -s eth0 autoneg off duplex full
```

Replace `eth0` with your Ethernet device.

Is the Interface Up?

Once you have established that you are physically connected to the network, the next step is to confirm that the network interface is configured correctly on your host. The best way to check this is to run the `ifconfig` command with your interface as an argument. So to test eth0's settings, you would run

```
$ sudo ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:17:42:1f:18:be
          inet addr:10.1.1.7  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::217:42ff:fe1f:18be/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:1 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:229 (229.0 B)  TX bytes:2178 (2.1 KB)
          Interrupt:10
```

Probably the most important line in this is the second line of output, which tells us our host has an IP address (`10.1.1.7`) and subnet mask (`255.255.255.0`) configured. Now, whether these are the correct settings for this host is something you will need to confirm. If the interface is not configured, try running `sudo ifup eth0` and then run `ifconfig` again to see if the interface comes up. If the settings are wrong or the interface won't come up, inspect /etc/network/interfaces on Debian-based systems or /etc/sysconfig/network_scripts/ifcfg-<interface> on Red Hat-based systems. It is in these files that you can correct any errors in the network settings. Now if the host gets its IP through DHCP, you will need to move your troubleshooting to the DHCP host to find out why you aren't getting a lease.

Is It on the Local Network?

Once you see that the interface is up, the next step is to see if a default gateway has been set and whether you can access it. The `route` command will display your current routing table, including your default gateway:

```
$ sudo route -n
Kernel IP routing table
Destination     Gateway      Genmask          Flags Metric Ref     Use Iface
10.1.1.0        *             255.255.255.0    U     0      0        0 eth0
default         10.1.1.1     0.0.0.0           UG    100    0        0 eth0
```

The line you are interested in is the last line, which starts with `default`. Here you can see that the host has a gateway of 10.1.1.1. Note that the `-n` option was used with `route` so it wouldn't try to resolve any of these IP addresses into hostnames. For one thing, the command runs more quickly, but more important, you don't want to cloud your troubleshooting with any potential DNS errors. If you don't see a default gateway configured here, and the host you want to reach is on a different subnet (say, web1, which is on 10.1.2.5), that is the likely cause of your problem. To fix this, either be sure to set the gateway in /etc/network/interfaces on Debian-based systems or /etc/sysconfig/network_scripts/ifcfg-<interface> on Red Hat-based systems, or if you get your IP via DHCP, be sure it is set correctly on the DHCP server and then reset your interface with the following on Debian-based systems:

```
$ sudo service networking restart
```

The following would be used on Red Hat-based systems:

```
$ sudo service network restart
```

On a side note, it's amazing that these distributions have to differ even on something this fundamental.

Once you have identified the gateway, use the `ping` command to confirm that you can communicate with the gateway:

```
$ ping -c 5 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=64 time=3.13 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=64 time=1.43 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=64 time=1.79 ms
64 bytes from 10.1.1.1: icmp_seq=5 ttl=64 time=1.50 ms
--- 10.1.1.1 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4020ms
rtt min/avg/max/mdev = 1.436/1.966/3.132/0.686 ms
```

As you can see, we were able to successfully ping the gateway, which means that we can at least communicate with the 10.1.1.0 network. If you couldn't ping the gateway, it could mean a few things. It could mean that your gateway is blocking ICMP packets. If so, tell your network administrator that blocking ICMP is an annoying practice with negligible security benefits and then try to ping another Linux host on the same subnet. If ICMP isn't being blocked, then it's possible that the switch port on your host is set to the wrong VLAN, so you will need to further inspect the switch to which it is connected.

Is DNS Working?

Once you have confirmed that you can speak to the gateway, the next thing to test is whether DNS functions. Both the `nslookup` and `dig` tools can be used to troubleshoot DNS issues, but since you need to perform only basic testing at this point, just use `nslookup` to see if you can resolve web1 into an IP:

```
$ nslookup web1
Server: 10.1.1.3
Address: 10.1.1.3#53
```

```
Name: web1.example.net
Address: 10.1.2.5
```

In this example DNS is working. The web1 host expands into `web1.example.net` and resolves to the address `10.1.2.5`. Of course, make sure that this IP matches the IP that web1 is supposed to have! In this case, DNS works, so we can move on to the next section; however, there are also a number of ways DNS could fail.

No Name Server Configured or Inaccessible Name Server If you see the following error, it could mean either that you have no name servers configured for your host or they are inaccessible:

```
$ nslookup web1
;; connection timed out; no servers could be reached
```

In either case you will need to inspect /etc/resolv.conf and see if any name servers are configured there. If you don't see any IP addresses configured there, you will need to add a name server to the file. Otherwise, if you see something like the following, you need to start troubleshooting your connection with your name server, starting off with `ping`:

```
search example.net
nameserver 10.1.1.3
```

If you can't ping the name server and its IP address is in the same subnet (in this case, 10.1.1.3 is within the subnet), the name server itself could be completely down. If you can't ping the name server and its IP address is in a different subnet, then skip ahead to the Can I Route to the Remote Host? section, but only apply those troubleshooting steps to the name server's IP. If you can ping the name server but it isn't responding, skip ahead to the Is the Remote Port Open? section.

Missing Search Path or Name Server Problem It is also possible that you will get the following error for your `nslookup` command:

```
$ nslookup web1
Server: 10.1.1.3
```

```
Address: 10.1.1.3#53
** server can't find web1: NXDOMAIN
```

Here you see that the server did respond, since it gave a response: `server can't find web1`. This could mean two different things. One, it could mean that web1's domain name is not in your DNS search path. This is set in /etc/resolv.conf in the line that begins with `search`. A good way to test this is to perform the same `nslookup` command, only use the fully qualified domain name (in this case, web1.example.net). If it does resolve, then either always use the fully qualified domain name, or if you want to be able to use just the hostname, add the domain name to the search path in /etc/resolv.conf.

If even the fully qualified domain name doesn't resolve, then the problem is on the name server. The complete method for troubleshooting all DNS issues is covered in Chapter 6, but here are some basic pointers. If the name server is supposed to have that record, then that zone's configuration needs to be examined. If it is a recursive name server, then you will have to test whether or not recursion is working on the name server by looking up some other domain. If you can look up other domains, then you must check if the problem is on the remote name server that does contain the zones.

Can I Route to the Remote Host?

After you have ruled out DNS issues and see that web1 is resolved into its IP 10.1.2.5, you must test whether you can route to the remote host. Assuming ICMP is enabled on your network, one quick test might be to ping web1. If you can ping the host, you know your packets are being routed there and you can move to the next section, Is the Remote Port Open? If you can't ping web1, try to identify another host on that network and see if you can ping it. If you can, then it's possible web1 is down or blocking your requests, so move to the next section. If you can't ping any hosts on the remote network, packets aren't being routed correctly. One of the best tools to test routing issues is `traceroute`. Once you provide `traceroute` with a host, it will test each hop between you and the host. For example, a successful `traceroute` between dev1 and web1 would look like this:

```
$ traceroute 10.1.2.5
traceroute to 10.1.2.5 (10.1.2.5), 30 hops max, 40 byte packets
1 10.1.1.1 (10.1.1.1) 5.432 ms 5.206 ms 5.472 ms
2 web1 (10.1.2.5) 8.039 ms 8.348 ms 8.643 ms
```

Here you can see that packets go from dev1 to its gateway (10.1.1.1), and then the next hop is web1. This means it's likely that 10.1.1.1 is the gateway for both subnets. On your network you might see a slightly different output if there are more routers between you and your host. If you can't ping web1, your output would look more like the following:

```
$ traceroute 10.1.2.5
traceroute to 10.1.2.5 (10.1.2.5), 30 hops max, 40 byte packets
1 10.1.1.1 (10.1.1.1) 5.432 ms 5.206 ms 5.472 ms
2 * * *
3 * * *
```

Once you start seeing asterisks in your output, you know that the problem is on your gateway. You will need to go to that router and investigate why it can't route packets between the two networks. Instead you might see something more like

```
$ traceroute 10.1.2.5
traceroute to 10.1.2.5 (10.1.2.5), 30 hops max, 40 byte packets
1 10.1.1.1 (10.1.1.1) 5.432 ms 5.206 ms 5.472 ms
1 10.1.1.1 (10.1.1.1) 3006.477 ms !H 3006.779 ms !H 3007.072 ms
```

In this case, you know that the ping timed out at the gateway, so the host is likely down or inaccessible even from the same subnet. At this point, if you haven't tried to access web1 from a machine on the same subnet as web1, try pings and other tests now.

NOTE If you have one of those annoying networks that block ICMP, don't worry, you can still troubleshoot routing issues. You just need to install the `tcptraceroute` package (`sudo apt-get install tcptraceroute`), then run the same commands as for `traceroute`, only substitute `tcptraceroute` for `traceroute`.

Is the Remote Port Open?

So you can route to the machine but you still can't access the web server on port 80. The next test is to see whether the port is even open. There are a number of different ways to do this. For one, you could try `telnet`:

```
$ telnet 10.1.2.5 80
Trying 10.1.2.5...
telnet: Unable to connect to remote host: Connection refused
```

If you see `Connection refused`, then either the port is down (likely Apache isn't running on the remote host or isn't listening on that port) or the firewall is blocking your access. If `telnet` can connect, then, well, you don't have a networking problem at all. If the web service isn't working the way you suspected, you need to investigate your Apache configuration on web1. Troubleshooting web server issues is covered in Chapter 8.

Instead of `telnet`, I prefer to use `nmap` to test ports because it can often detect firewalls. If `nmap` isn't installed, use your package manager to install the `nmap` package. To test web1, type the following:

```
$ nmap -p 80 10.1.2.5
Starting Nmap 4.62 ( http://nmap.org ) at 2009-02-05 18:49 PST
Interesting ports on web1 (10.1.2.5):
PORT STATE SERVICE
80/tcp filtered http
```

Aha! `nmap` is smart enough that it can often tell the difference between a closed port that is truly closed and a closed port behind a firewall. Normally when a port is actually down, `nmap` will report it as closed. Here it reported it as filtered. What this tells us is that some firewall is in the way and is dropping the packets to the floor. This means you need to investigate any firewall rules on the gateway (10.1.1.1) and on web1 itself to see if port 80 is being blocked.

Test the Remote Host Locally

At this point, we have either been able to narrow the problem down to a network issue or we believe the problem is on the host itself. If we think the problem is on the host itself, we can do a few things to test whether port 80 is available.

Test for Listening Ports

One of the first things you should do on web1 is test whether port 80 is listening. The `netstat -lnp` command will list all ports that are listening along with the process that has the port open. You could just run that and parse through the output for anything that is listening on port 80, or you could use `grep` to show only things listening on port 80:

```
$ sudo netstat -lnp | grep :80
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN 919/apache
```

The first column tells you what protocol the port is using. The second and third columns are the receive and send queues (both are set to 0 here). The column you want to pay attention to is the fourth column, as it lists the local address on which the host is listening. Here the `0.0.0.0:80` tells us that the host is listening on all of its IPs for port 80 traffic. If Apache were listening only on web1's Ethernet address, you would see `10.1.2.5:80` here.

The final column will tell you which process has the port open. Here you can see that Apache is running and listening. If you do not see this in your `netstat` output, you need to start your Apache server.

Firewall Rules

If the process is running and listening on port 80, it's possible that web1 has some sort of firewall in place. Use the `iptables` command to list all of your firewall rules. If your firewall is disabled, your output will look like this:

```
$ sudo /sbin/iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Notice that the default policy is set to ACCEPT. It's possible, though, that your firewall is set to drop all packets by default, even if it doesn't list any rules. If that is the case you will see output more like the following:

```
$ sudo /sbin/iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

On the other hand, if you had a firewall rule that blocked port 80, it might look like this:

```
$ sudo /sbin/iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
REJECT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:80 reject-with
  ➥icmp-port-unreachable

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Clearly, in the latter case you would need to modify the firewall rules to allow port 80 traffic from the host.

Troubleshoot Slow Networks

In a way, it's easier to troubleshoot network problems when something doesn't work at all. When a host is unreachable, you can perform the troubleshooting steps discussed earlier until the host is reachable again. When the network is just slow, however, sometimes it can be a bit tricky to track down why. This section discusses a few techniques you can use to track down the cause of slow networks.

DNS Issues

Although DNS is blamed more often than it should be for network problems, when DNS does have an issue, it can often result in poor network performance. For instance, if you have two DNS servers configured for a domain and the first one you try goes down, your DNS requests will wait 30 seconds before they time out and go to the secondary DNS server. Although this will definitely be noticeable when you run tools like `dig` or `nslookup`, DNS issues can cause apparent network slowdowns in some unexpected ways; this is because so many services rely on DNS to resolve hostnames to IP addresses. Such issues can even affect your network troubleshooting tools.

`Ping`, `traceroute`, `route`, `netstat`, and even `iptables` are great examples of network troubleshooting tools that can degrade during DNS issues. By default, all of these tools will attempt to resolve IP addresses into hostnames if they can. If there are DNS problems, however, the results from each of these commands might stall while they attempt to look up IP addresses and fail. In the case of `ping` or `traceroute`, it might seem like your `ping` replies are taking a long time, yet when they do finally come through, the round-trip time is relatively low. In the case of `route`, `netstat`, and `iptables`, the results might stall for quite some time before you get output. The system is waiting for DNS requests to time out.

In all of the cases mentioned, it's easy to bypass DNS so your troubleshooting results are accurate. All of the commands we discussed earlier accept an `-n` option, which disables any attempt to resolve IP addresses into hostnames. I've just become accustomed to adding `-n` to all of the commands I introduced you to in the first part of this chapter unless I really do want IP addresses resolved.

NOTE Although we'll get into this more in Chapter 8, DNS resolution can also affect your web server's performance in an unexpected way. Some web servers are configured to resolve every IP address that accesses them into a hostname for logging. Although that can make the logs more readable, it can also dramatically slow down your web server at the worst times—when you have a lot of visitors. Instead of serving traffic, your web server can get busy trying to resolve all of those IPs.

Find the Network Slowdown with traceroute

When your network connection seems slow between your server and a host on a different network, sometimes it can be difficult to track down where the real slowdown is. Especially in situations where the slowdown is in latency (the time it takes to get a response) and not overall bandwidth, it's a situation `traceroute` was made for. `traceroute` was mentioned earlier in the chapter as a way to test overall connectivity between you and a server on a remote network, but `traceroute` is also useful when you need to diagnose where a network slowdown might be. Since `traceroute` outputs the reply times for every hop between you and another machine, you can trace down servers that might be on a different continent or gateways that might be overloaded and causing network slowdowns. For instance, here's part of a `traceroute` between a server in the United States and a Chinese Yahoo server:

```
$ traceroute yahoo.cn
traceroute to yahoo.cn (202.165.102.205), 30 hops max, 60 byte packets
 1  64-142-56-169.static.sonic.net (64.142.56.169)  1.666 ms  2.351 ms  3.038 ms
 2  2.ge-1-1-0.gw.sr.sonic.net (209.204.191.36)  1.241 ms  1.243 ms  1.229 ms
 3  265.ge-7-1-0.gw.pao1.sonic.net (64.142.0.198)  3.388 ms  3.612 ms  3.592 ms
 4  xe-1-0-6.ar1.pao1.us.nlayer.net (69.22.130.85)  6.464 ms  6.607 ms  6.642 ms
 5  ae0-80g.cr1.pao1.us.nlayer.net (69.22.153.18)  3.320 ms  3.404 ms  3.496 ms
 6  ae1-50g.cr1.sjc1.us.nlayer.net (69.22.143.165)  4.335 ms  3.955 ms  3.957 ms
 7  ae1-40g.ar2.sjc1.us.nlayer.net (69.22.143.118)  8.748 ms  5.500 ms  7.657 ms
 8  as4837.xe-4-0-2.ar2.sjc1.us.nlayer.net (69.22.153.146)  3.864 ms  3.863 ms  3.865 ms
 9  219.158.30.177 (219.158.30.177)  275.648 ms  275.702 ms  275.687 ms
10  219.158.97.117 (219.158.97.117)  284.506 ms  284.552 ms  262.416 ms
11  219.158.97.93 (219.158.97.93)  263.538 ms  270.178 ms  270.121 ms
12  219.158.4.65 (219.158.4.65)  303.441 ms *  303.465 ms
13  202.96.12.190 (202.96.12.190)  306.968 ms  306.971 ms  307.052 ms
14  61.148.143.10 (61.148.143.10)  295.916 ms  295.780 ms  295.860 ms
...
```

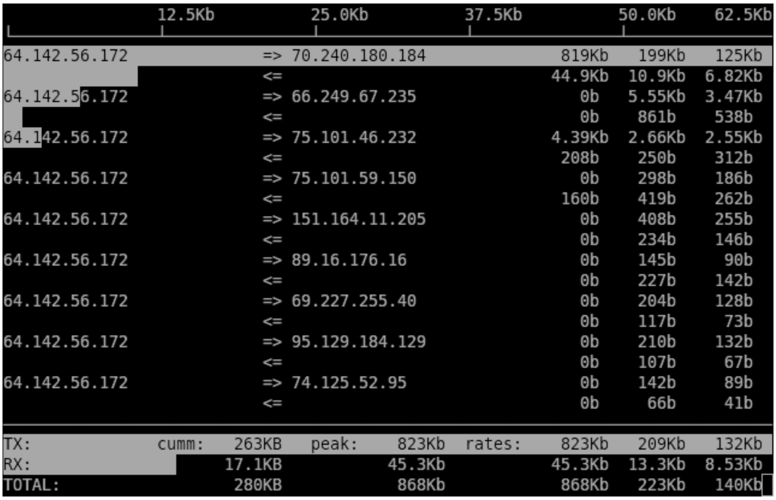
Without knowing much about the network, you can assume just by looking at the round-trip times that once you get to hop 9 (at the 219.158.30.177 IP), you have left the continent, as the round-trip time jumps from 3 milliseconds to 275 milliseconds.

Find What Is Using Your Bandwidth with iftop

Sometimes your network is slow not because of some problem on a remote server or router, but just because something on the system is using up all the available bandwidth. It can be tricky to identify what process is using up all the bandwidth, but there are some tools you can use to help identify the culprit.

`top` is such a great troubleshooting tool that it has inspired a number of similar tools like `iotop` to identify what processes are consuming the most disk I/O. It turns out there is a tool called `iftop` that does something similar with network connections. Unlike `top`, `iftop` doesn't concern itself with processes but instead lists the connections between your server and a remote IP that are consuming the most bandwidth (Figure 5-1). For instance, with `iftop` you can quickly see if your backup job is using up all your bandwidth by seeing the backup server IP address at the top of the output.

`iftop` is available in a package of the same name on both Red Hat- and Debian-based distributions, but in the case of Red Hat-based distributions,



```
                12.5Kb          25.0Kb          37.5Kb          50.0Kb    62.5Kb
64.142.56.172              => 70.240.180.184               819Kb   199Kb   125Kb
                           <=                              44.9Kb  10.9Kb  6.82Kb
64.142.56.172              => 66.249.67.235                0b      5.55Kb  3.47Kb
                           <=                              0b      861b    538b
64.142.56.172              => 75.101.46.232                4.39Kb  2.66Kb  2.55Kb
                           <=                              208b    250b    312b
64.142.56.172              => 75.101.59.150                0b      298b    186b
                           <=                              160b    419b    262b
64.142.56.172              => 151.164.11.205               0b      408b    255b
                           <=                              0b      234b    146b
64.142.56.172              => 89.16.176.16                 0b      145b    90b
                           <=                              0b      227b    142b
64.142.56.172              => 69.227.255.40                0b      204b    128b
                           <=                              0b      117b    73b
64.142.56.172              => 95.129.184.129               0b      210b    132b
                           <=                              0b      107b    67b
64.142.56.172              => 74.125.52.95                 0b      142b    89b
                           <=                              0b      66b     41b

TX:             cumm:   263KB   peak:   823Kb   rates:     823Kb   209Kb   132Kb
RX:                     17.1KB          45.3Kb             45.3Kb  13.3Kb  8.53Kb
TOTAL:                  280KB           868Kb              868Kb   223Kb   140Kb
```

Figure 5-1 Sample iftop output

you might have to find it from a third-party repository. Once you have it installed, just run the `iftop` command on the command line (it will require root permissions). Like with the `top` command, you can hit Q to quit.

At the very top of the `iftop` screen is a bar that shows the overall traffic for the interface. Just below that is a column with source IPs followed by a column with destination IPs and arrows between them so you can see whether the bandwidth is being used to transmit packets from your host or receive them from the remote host. After those columns are three more columns that represent the data rate between the two hosts over 2, 10, and 40 seconds, respectively. Much like with load averages, you can see whether the bandwidth is spiking now, or has spiked some time in the past. At the very bottom of the screen, you can see statistics for transmitted data (TX) and received data (RX) along with totals. Like with `top`, the interface updates periodically.

The `iftop` command run with no arguments at all is often all you need for your troubleshooting, but every now and then, you may want to take advantage of some of its options. The `iftop` command will show statistics for the first interface it can find by default, but on some servers you may have multiple interfaces, so if you wanted to run `iftop` against your second Ethernet interface (eth1), type `iftop -i eth1`.

By default `iftop` attempts to resolve all IP addresses into hostnames. One downside to this is that it can slow down your reporting if a remote DNS server is slow. Another downside is that all that DNS resolution adds extra network traffic that might show up in `iftop`! To disable network resolution, just run `iftop` with the `-n` option.

Normally `iftop` displays overall bandwidth used between hosts, but to help you narrow things down, you might want to see what ports each host is using to communicate. After all, if you knew a host was consuming most of your bandwidth over your web port, you would perform different troubleshooting than if it was connecting to an FTP port. Once `iftop` is launched, press P to toggle between displaying all ports and hiding them. One thing you'll notice, though, is that sometimes displaying all the ports can cause hosts you are interested in to fall off the screen. If that happens, you can also hit either S or D to toggle between displaying ports only from the source or only from the destination host, respectively. Showing only source ports can be useful when you run `iftop` on a server, since for many services, the destination host uses random high ports that don't necessarily identify what service is being used, but the ports on your server are more likely to correspond to a service on your machine. You can then follow up with the `netstat -lnp` command referenced earlier in this chapter to find out what service is listening on that port.

Like with most Linux commands, `iftop` has an advanced range of options. What we covered should be enough to help with most troubleshooting efforts, but in case you want to dig further into `iftop`'s capabilities, just type `man iftop` to read the manual included with the package.

Packet Captures

Although the techniques mentioned in this chapter should help you troubleshoot a wide range of networking problems, some problems are so subtle or low-level that the only way to track them down is to dig down into the protocol itself and examine individual packets as they go back and forth. Because of the low-level and tedious nature of analyzing packet dumps, you should try to use it as a last resort. That said, this type of troubleshooting can be quite effective, particularly to identify hosts on your local network that are misbehaving, hosts with misconfigured network settings, or debugging communications between your own client and server software. Packet dumps are less effective for troubleshooting if you are unfamiliar with the protocols you are examining since you can't tell correct traffic from errors, or if you allow yourself to get buried in volumes of packets and can't see the problem for all of the normal traffic.

When you capture packets, it's most effective if you can capture them on both sides of a communication, especially if there is a router or firewall between two hosts. If a machine between the two hosts is the cause of the problem, you're more likely to detect it when you can see whether packets sent from host A arrive on host B exactly as they are sent. For instance, if you see host B send a reply back to host A that never gets there, you can be confident that the problem is somewhere in between the two hosts.

A great example of where packet captures come into play occurred some time back when I was troubleshooting a host that seemed to have trouble communicating with a different server. Connections would sometimes just die out, yet at other times things seemed relatively fine, if slow. Nothing can be trickier to troubleshoot than an intermittent problem. After a series of different troubleshooting steps, we captured packets both from the problem host and the destination server.

What we discovered in the packet dump was that a misconfigured router had been trying to apply NAT (Network Address Translation) rules to our destination server incorrectly and had sent reply packets back to our host while the destination server was trying to reply to us directly. Our host was seeing the same reply twice, but from two different MAC addresses. What happened was a race where each time we tried to set up a TCP handshake, sometimes the destination server won the race and replied back, but other times the router replied back first; upon seeing that reply, our host tried to re-initiate the handshake. Depending on who won the race, the communication would continue or get reset. If we weren't able to analyze the individual packets going back and forth, we may have never discovered the duplicate packets.

Use the tcpdump Tool

The main packet capture tool we will discuss is `tcpdump`. This is an old and proven command-line packet capture tool, and although there are more modern tools out there, `tcpdump` is a program that you should be able to find on any Linux system. Because of how `tcpdump` works, you will need to run it with root privileges on your machine. By default, it will scan through your network interfaces and choose the first suitable one; then it will capture, parse, and output information about the packets it sees. Here's some example output from `tcpdump` with the `-n` option (so it doesn't convert IP addresses to hostnames and slow things down):

```
$ sudo tcpdump -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
19:01:51.133159 IP 208.115.111.75.60004 > 64.142.56.172.80: Flags [F.], seq 753858968, ack
  ➥1834304357, win 272, options [nop,nop,TS val 99314435 ecr 1766147273], length 0
```

```
19:01:51.133317 IP 64.142.56.172.80 > 208.115.111.75.60004: Flags [F.], seq 1, ack 1, win
  ➥54, options [nop,nop,TS val 1766147276 ecr 99314435], length 0
19:01:51.157772 IP 208.115.111.75.60004 > 64.142.56.172.80: Flags [.], ack 2, win 272,
  ➥options [nop,nop,TS val 99314437 ecr 1766147276], length 0
19:01:51.224021 IP 72.240.13.35.45665 > 64.142.56.172.53: 59454% [1au] AAAA? ns2.example.
  ➥net. (45)
19:01:51.224510 IP 64.142.56.172.53 > 72.240.13.35.45665: 59454*- 0/1/1 (90)
19:01:51.256743 IP 201.52.186.78.63705 > 64.142.56.172.80: Flags [.], ack 1833085614, win
  ➥65340, length 0
```

NOTE Whenever you are done capturing packets, just hit Ctrl-C to exit `tcpdump`. As `tcpdump` exits, it tells you how many packets it was able to capture and how many the kernel dropped.

The output of `tcpdump` can be a bit tricky to parse at first, and I won't go over all the columns, but let's take two lines from the preceding output and break them down:

```
19:01:51.224021 IP 72.240.13.35.45665 > 64.142.56.172.53: 59454% [1au] AAAA? ns2.example.
  ➥net. (45)
19:01:51.224510 IP 64.142.56.172.53 > 72.240.13.35.45665: 59454*- 0/1/1 (90)
```

The first line tells you that at 19:01:51, the host 72.240.13.35 on port 45665 sent a packet to 64.142.56.172 on port 53 (DNS). If you wanted to dig further in that line you could see that the source host sent a request for the AAAA record (an IPv6 IP address) for ns2.example.net. The second line tells you that also at 19:01:51 the host 64.142.56.172 on port 53 replied back to host 72.240.13.35 on port 45665, presumably with an answer to the query.

Since the first column is a datestamp for each packet, it makes it simple to see how long communication takes between hosts. This can be particularly useful for protocols that have set timeouts (like 30-second timeouts for DNS requests) since you can watch the timeout occur and see the source host resend its request. The next major column shows the IP and port for the source host. The > in the line can be treated like an arrow that lets you know that the direction of communication is from the first IP to the second. Finally, the next column tells you the destination IP and port followed by some extra flags, sequence numbers, and other TCP/IP information for that packet that we won't get into here.

Filtering Tcpdump Output Since by default `tcpdump` captures all of the packets it sees, it usually bombards you with a lot of noise that doesn't help with your troubleshooting. What you want to do is pass `tcpdump` some filtering rules so it only shows you packets that you are interested in. For instance, if you were troubleshooting problems between your host and a server with a hostname of web1, you could tell `tcpdump` to only show packets to or from that host with

```
$ sudo tcpdump -n host web1
```

If you wanted to do the opposite, that is, show all traffic except anything from web1, you would say

```
$ sudo tcpdump -n not host web1
```

You can also filter traffic to and from specific ports. For instance, if you wanted to just see DNS traffic (port 53) you would type

```
$ sudo tcpdump -n port 53
```

If you wanted to capture all of your web traffic on either port 80 or port 443, you would type

```
$ sudo tcpdump -n port 80 or port 443
```

You can actually get rather sophisticated with `tcpdump` filters, but it's often easier to just capture a certain level of `tcpdump` output to a file and then use `grep` or other tools to filter it further. To save `tcpdump` output to a file, you can use a command-line redirect:

```
$ sudo tcpdump -n host web1 > outputfile
```

If you want to view the packets on the command line while they are being saved to a file, add the `-l` option to `tcpdump` so it buffers the output, and then use `tee` to both display the output and save it to a file:

```
$ sudo tcpdump -n -l host web1 | tee outputfile
```

Raw Packet Dumps Although you might think that `tcpdump` already provides plenty of difficult-to-parse output, sometimes all that output isn't enough—you want to save complete raw packets. Raw packets are particularly useful since they contain absolutely all of the information about communication between hosts, and a number of tools (such as Wireshark, which we'll discuss briefly momentarily) can take these raw packet dumps as input and display them in a much-easier-to-understand way.

The simplest way to save raw packet dumps is to run `tcpdump` with the `-w` option:

```
$ sudo tcpdump -w output.pcap
```

Like with other `tcpdump` commands, hit Ctrl-C to stop capturing packets. You can also use all of the same filtering options we've discussed so far when capturing raw packets. With raw packet dumps, you are getting the complete contents of the packets as best as `tcpdump` and your disk can keep up. So if someone is transferring a 1Gb file from your server, you might just capture the whole file in your packet dump. You may want to open up a second command-line session just so you can keep an eye on the size of the output file.

`tcpdump` provides a few options you can use to manage the size of output files. The first option, `-C`, lets you specify the maximum size of the output file (in millions of bytes) before it moves on to a second one. So, for instance, if you wanted to rotate files after they grow past ten megabytes, you can type

```
$ sudo tcpdump -C 10 -w output.pcap
```

The first output file will be named output.pcap.1, and once it gets to ten megabytes, `tcpdump` will close it and start writing to output.pcap.2, and so on, until you either kill `tcpdump` or you run out of disk space. If you want to be sure that you won't run out of disk space, you can also add the `-W` option, which lets you limit the number of files `tcpdump` will ultimately create. Once `tcpdump` reaches the last file, it will start from the beginning and overwrite the first file in the set. So, for instance, if you want `tcpdump` to rotate to a new file after ten megabytes and want to make sure `tcpdump` only uses fifty megabytes of disk space, you could limit it to five rotated files:

```
$ sudo tcpdump -C 10 -W 5 -w output.pcap
```

Once you have these packet captures, you can use `tcpdump` to replay them as though they were happening in real time with the `-r` option. Just specify your raw packet output file as an argument. You can specify filters and other options like `-n` just as if you were running `tcpdump` against a live stream of traffic:

```
$ sudo tcpdump -n -r output.pcap
```

The `tcpdump` program is full of useful options and filters beyond what I've mentioned here. The man page (type `man tcpdump`) not only goes over all of these options and filters, but it also provides a nice primer on TCP packet construction, so it's worth looking through if you want to dig deeper into `tcpdump`'s abilities.

Use Wireshark

Although `tcpdump` is a handy tool for packet capture, when you actually need to parse through and analyze raw packets, the `-r` option sometimes doesn't cut it. Luckily some tools make the process simpler. One of the best tools for raw packet analysis is Wireshark. It is a desktop application that provides a lot of sophisticated tools for packet analysis that are way beyond the scope of this book. At a basic level, though, Wireshark provides you with a much easier way to view your raw packet dumps and pinpoint obvious problems.

The Wireshark package should be packaged and available for major Linux distributions, and it even has clients for Windows and Mac systems. Once installed, you can launch it via your desktop environment or just type `wireshark` on the command line. If you type `wireshark` followed by your raw packet file, it will go ahead and open it up as it starts.

As Figure 5-2 shows, Wireshark separates its GUI into a few sections. The main pane below the toolbar displays basic packet information like you might find in default `tcpdump` output. What's useful about Wireshark is that its columns are a bit simpler to read, plus it color-codes packets based on protocol and will even highlight error packets in red. The color coding in this main pane makes it a bit simpler to filter through your traffic and identify possible problems.

Once you click on a particular packet in the main pane, the pane below it shows all of the detailed information in the different layers of the packet.

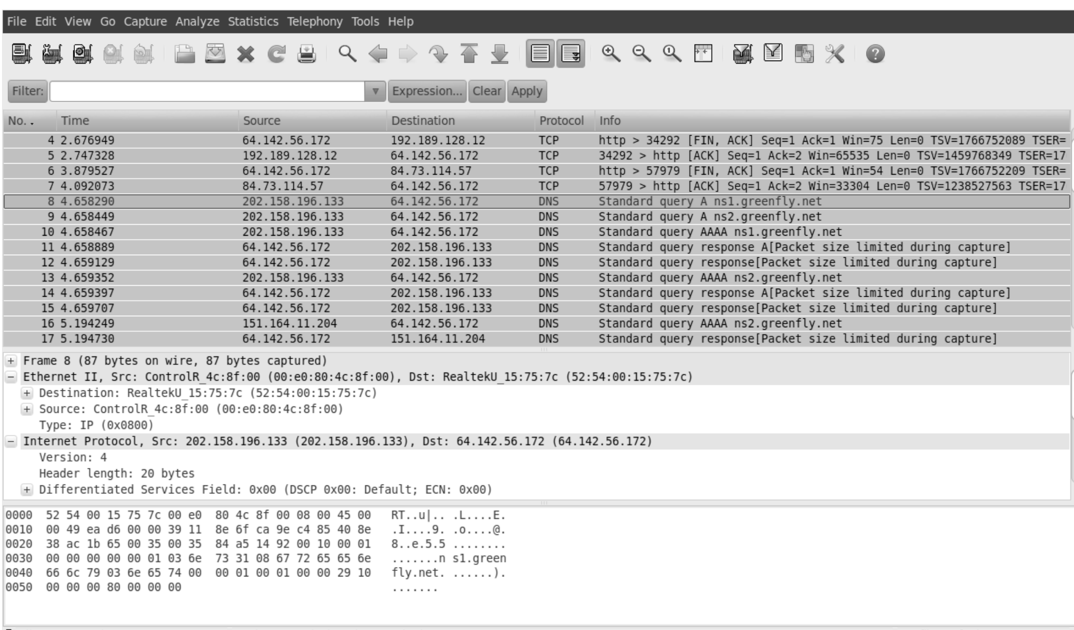


Figure 5-2 Default Wireshark window

In that pane you can drill down to display IP headers, the data section of the packet, and everything in between. Once you do, click on and expand a particular section of a packet; at the very bottom of the window is a separate pane that will show you both the hex and ASCII representation of that data.

Wireshark has a ton of features, including the ability to capture packets in its own right, and is a complicated and powerful-enough tool that it could be a subject for its own book. Since this is a book about troubleshooting and not TCP/IP itself, this section just mentions a few basic features that will help you with troubleshooting.

Along the top toolbar you'll see an input box and a button labeled Filter. As with `tcpdump`, you filter packet dumps so you only see packets that match your criteria. Unlike `tcpdump`, Wireshark uses a completely different syntax for filters. So, for instance, if you want to see only packets related to host 192.168.0.1, type this in the filter and press Enter:

```
ip.addr == 192.168.0.1
```

To display only packets related to DNS (port 53), type

```
tcp.port == 53 || udp.port == 53
```

The filtering syntax for Wireshark is pretty extensive, but if you click on the button labeled Filter, a window pops up that gives you a good list of examples to get you started. From there you can also click a Help button that gives you more complete documentation on how to construct your own filter rules.

Another useful feature in Wireshark is the ability to pick a complete stream of communication between two hosts out of a large number of packets. Although you can certainly do this yourself by hand, you can also just select one sample packet you are interested in, then click Analyze → Follow TCP Stream. If it's a UDP or SSL stream, those options will be visible instead. Once you select that menu, a new window pops up (Figure 5-3),

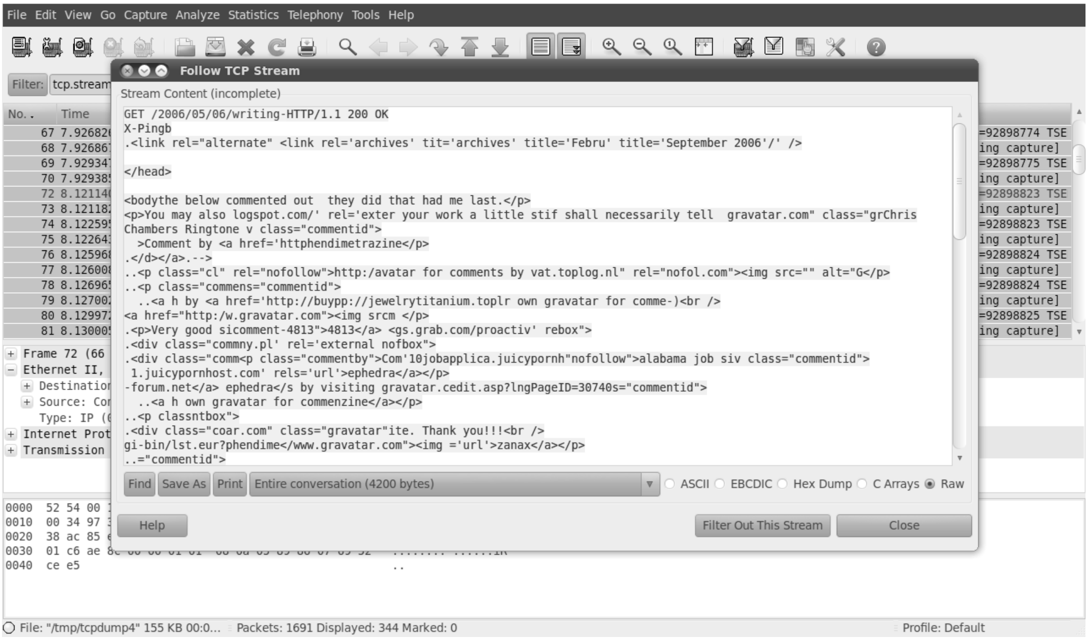


Figure 5-3 Wireshark following an HTTP stream filled with blog spam

and if it is able to piece together any content from that stream, it is displayed. In either case, when you close the Follow TCP Stream window, the main Wireshark window will have automatically filtered out all of the packets except for those related to this particular stream.

This page intentionally left blank

Index

(pound sign), comment indicator, 43
1xx informational codes, 150
2xx successful codes, 150–151
3xx redirection codes, 151–152
4xx client error codes, 152–153
5xx server error codes, 153–154

A

Active threads, metrics, 177
Apache
  displaying web server statistics, 158–162
  validating web server configuration, 163–164
apache2ctl command, 163–164

B

BIOS (Basic Input Output System), 36–37
BIOS boot order, 45–47
Blame, establishing
  human actions, 11
  postmortems, 11
  technology, 14
Boot issues. *See also* GRUB issues; Linux boot process; *specific issues*.
  root file system won't mount, 51–55
  secondary file system won't mount, 55–56

C

Changes
  rolling back, 13
  tracking, 12–13
Chat rooms as a communication method, 7–8
Client error codes, 152–153
Communication methods
  backup methods, 8
  chat rooms, 7–8
  conference calls, 4–5
  direct conversation, 5–6
  email, 6–7
Conference calls as a communication method, 4–5
Conversation as a communication method, 5–6
Copy failure, 62–63
CPU statistics, displaying. *See also* top command.
  idle time, 23
  iostat program, 28
  sysstat package, 30–31
CPU time
  system, 23
  user, 23–25
CPU-bound load average, 20
curl command
  parsing web server logs, 154–157
  testing web servers, 146–148

D

Database metrics. *See also* Metrics.
  active threads, 177
  database statistics, 180–181
  flush tables, 178
  MySQL, 177–179
  open tables, 178

Database metrics, *continued*
opens, 178
pg_stat_activity table, 179–180
pg_stat_all_tables table, 181–182
pg_stat_database table, 180–181
PostgresSQL, 179–182
queries per second, 178
questions from clients, 177
server process stats, 179–180
statistics, per table, 181–182
uptime, 177
Database metrics, slow queries
identifying, 182–184
MySQL, 182–183
PostgresSQL, 183–184
statistics on, 178
Database servers. *See also* Logs, databases.
MySQL, 174–175
PostgresSQL, 175–177
testing, 174–175
df command, 59
dig command
displaying TTL values, 109–110
DNS troubleshooting, 72–74, 95–97
recursive DNS resolution, 102–104
recursive name servers, 106
+trace argument, 102–104
zone transfer issues, 113–117
Directories, space usage, 59–61
Disk issues
corrupted file systems, repairing, 63–64
disk full, 58–61
hard drive failure, 186–190
label problems, diagnosing, 54
large .swp files, 61
large /tmp files, 61
out of inodes, 61–62
read-only file system, 62–63
reserved blocks, 59
software RAID, repairing, 64–66
space usage, displaying, 59–61
statistics, displaying, 32
Dividing the problem space, 3–4
dmesg command, 62
DNS (Domain Naming System)
caches, flushing, 111–112
caching, 108–112
inaccessible, 73, 95–97
missing search path, 97–98
not configured, 73, 95–97
overview, 94–95
recursive name servers, 95–98
testing, 72–74
troubleshooting, 95
DNS servers, troubleshooting
dig output, 98–101
DNS caches, flushing, 111–112
DNS caching, 108–112
recursive DNS resolution, 102–107
recursive name servers, 104–107
tracing DNS queries, 101–104
TTL (Time To Live) values, 108–112
update not taking, 107–117
zone syntax errors, 112
zone transfer issues, 113–117
Documenting troubleshooting activities, 10–12
du command, 60
duck command, 60
Duplex issues, diagnosing, 70

E

Email
as a communication method, 6–7
greylisting, 130
headers, 123–125
spam reduction, 130, 132–133
tracing requests, 125

Email, receiving
logs, examining, 138–140
telnet cannot connect, 136–137
telnet connects, message rejected, 137–138
Email, sending
error codes, 129–130
outbound server can't communicate with destination, 131–135
outbound server won't allow relay, 130–131
overview, 125–126
sending a test email, 127–129
unable to communicate with outbound server, 126–130
Error codes. *See also* HTTP status codes.
client error, 152–153
email, 129–130
informational, 150
overview, 149–150
redirection, 151–152
server error, 153–154
successful, 150–151
Error logs. *See* Logs.
/etc/init.d directory, 40–41
/etc/rc.local directory, 41
/etc/rc*n*.d directory, 41
/etc.rcS.d directory, 41
ethtool program, 69–70
Extended-status command, 178–179

F

File systems
corrupted, repairing, 63–64
read-only, 62–63
Files
listing by size, 61
space usage, displaying, 59–61
unable to save or copy, 62–63
Firewalls
detecting, 76
rules, displaying, 77–78, 145–146
5xx server error codes, 153–154
Flush tables, metrics, 178
Folders. *See* Directories.
4xx client error codes, 152–153
fsck command, 63–64

G

grep command
parsing web server logs, 156
searching for email ID, 132
testing MySQL, 175
Greylisting email, 130
GRUB boot loader, 37–38
GRUB issues
configuration file, editing, 54
disabling splash screens, 51
version, displaying, 47
GRUB issues, prompt
misconfigured prompt, 49
no prompt, 45–47, 47–48
stage 1.5 prompt, 48–49
GRUB issues, repairs
from the live system, 49–50
with a rescue disk, 50–51

H

Hard drive. *See* Disk issues.
Hardware interrupts, displaying, 23
Headers, email, 123–125
High load average, definition, 20
Hostnames, converting from IP addresses. *See* DNS (Domain Naming System).
HTTP status codes
1xx informational codes, 150
2xx successful codes, 150–151
3xx redirection codes, 151–152

HTTP status codes, *continued*
- 4xx client error codes, 152–153
- 5xx server error codes, 153–154
- overview, 149–150

I

ICMP, blocked packets, 72
ifconfig command, 70–71
iftop command, 81–83
Informational error codes, 150
init scripts
- directory for, 41
- respawning, 42–45
- upstart scripts, 42–45

initrd (initial RAM disk), 38–39
Inodes
- definition, 61
- running out of, 61–62
- usage, displaying, 61–62

Internet, targeted searches, 14–15
intramfs file, 38–39
I/O wait time
- diagnosing, 27–29
- displaying, 23

I/O-bound load average, 20
iostat program, 27–29
iotop command, 81–83
IP addresses, converting to hostnames. *See* DNS (Domain Naming System).
iptables command
- displaying firewall rules, 77–78, 145–146
- troubleshooting DNS issues, 79

L

LILO boot loader, 37
Linux boot process. *See also* GRUB issues.
- BIOS (Basic Input Output System), 36–37
- BIOS boot order, 45–47
- GRUB boot loader, 37–38
- initrd (initial RAM disk), 38–39
- intramfs file, 38–39
- LILO boot loader, 37
- Linux kernel, 38–39

Linux boot process, /sbin/init program
- /etc/init.d directory, 40–41
- /etc/rc.local directory, 41
- /etc/rc*n*.d directory, 41
- /etc.rcS.d directory, 41
- init scripts, 41–45
- overview, 39
- runlevels, 40
- single-user mode, 40
- startup scripts, 40–41
- system init scripts, 41
- System V init, 39–42
- upstart scripts, 42–45
- user-editable script, 41

Linux kernel, 38–39
Listening ports, displaying, 143, 144. *See also* Port 80.
lm-sensors package, 193
Load. *See* System load.
log_min_duration_statement setting, 183–184
Logs, databases
- high server load, 173–174
- MySQL, 173
- PostgresSQL, 173
- searching, 172–174

Logs, email, 138–140
Logs, web server
- enabling DNS resolution, 158
- parsing, 154–157

log_slow_queries variable, 182–183
long_query_time variable, 182–183

M

mdadm command, 64–66
Memory. *See* RAM.

Memtest86+ tool, 190–191
Metrics. *See also* Database metrics; Statistics; System load; *specific metrics*.
 CPU idle time, 23
 hardware interrupts, 23
 I/O wait, 23
 nice CPU time, 23
 software interrupts, 23
 steal time, 23
 system CPU time, 23
 user CPU time, 23
mke2fs tool, 64
mysql command, 174
MySQL databases
 database servers, 174–175
 logs, 173
 metrics, 177–179
 slow queries, 182–183
 testing, 175
mysqladmin command, 177, 183

N

Narrowing the problem, 3–4
netstat command
 displaying listening ports, 77, 144
 troubleshooting DNS issues, 79
Network card failure, 191–192
Network interfaces
 configuration, checking, 70–71
 displaying, 69–70
Networks
 connections, checking, 69–70
 settings, displaying, 69–70
Networks, slow
 bandwidth consumption, tracing, 81–83
 DNS issues, 79
 finding the slowdown, 80
 packet captures, 83–88
Nginx
 displaying web server statistics, 158–162
 validating web server configuration, 163–164
nginx command, 164
Nice CPU time, displaying, 23
nmap program, 76
nosplash option, 51
nslookup tool, 72–74, 95–97

O

1xx informational codes, 150
OOM (out-of-memory) killer, 26–27
Open tables, metrics, 178
Out-of-memory issues, 25–27

P

Packet captures
 overview, 83–84
 raw packet dumps, 87–91
 replaying captured packets, 88
 tcpdump tool, 84–88
 Wireshark program, 88–91
Partitions, duplicate names, 52
Past solutions, favoring, 9–10
Performance
 slow or no server response. *See* System load.
 troubleshooting slow networks, 78–83
perl command, 156–157
pg_stat_activity table, 179–180
pg_stat_all_tables table, 181–182
pg_stat_database table, 180–181
ping command
 DNS troubleshooting, 96
 testing local gateway, 72
 troubleshooting DNS issues, 79
Port 80, testing. *See also* Listening ports.
 servers, 76, 77–78
 web servers, 143–146

PostgresSQL databases
- database servers, 175–177
- logs, 173
- metrics, 179–182
- slow queries, 183–184
- testing, 176

Postmortems, 10–12
Pound sign (#), comment indicator, 43
Power supply failure, 194–195
Processes
- displaying, 29. *See also* top command.
- RAM consumption, 25

Processes, killing
- OOM (out-of-memory) killer, 26–27
- top command, 21

ps command
- testing MySQL, 175
- testing PostgresSQL, 176

Q

Queries per second, metrics, 178
Questions from clients, metrics, 177

R

RAID (Redundant Array of Inexpensive Disks)
- failure detection, 64–66
- repairing, 64–66

RAM
- DIMM failure, identifying, 191
- statistics, displaying, 31–32
- testing, 190–191
- usage, diagnosing, 25–27

RAM-bound load average, 20
Raw packet dumps, 87–91
Rebooting, 15
Recursive DNS resolution, 102–104
Recursive name servers, 95–98
Redirection error codes, 151–152
Remote host
- routing to, 74–75
- testing locally, 76–78

Remote ports, testing, 76, 77–78
Rescue disk, repairing GRUB issues, 50–51
Reserved blocks, 59
Respawning init scripts, 42–45
Rolling back changes, 13
Root file system won't mount
- duplicate partition names, 52
- root device changed, 52–55
- root kernel argument, 52
- root partition corrupt or failed, 55
- UUID changed, 54–55

route command
- displaying current route table, 71–72
- troubleshooting DNS issues, 79

Routing table, displaying, 71–72
Runlevels, 40

S

sar tool, 31
Save failure, 62–63
/sbin/ifconfig command, 69–70
/sbin/init program
- /etc/init.d directory, 40–41
- /etc/rc.local directory, 41
- /etc/rc*n*.d directory, 41
- /etc.rcS.d directory, 41
- init scripts, 41–45
- overview, 39
- runlevels, 40
- single-user mode, 40
- startup scripts, 40–41
- system init scripts, 41
- System V init, 39–42
- upstart scripts, 42–45
- user-editable script, 41

SBL (Spam Blackhole List), 132–133

Scripts
- init, 41–45
- startup, 40–41
- system init, 41
- upstart, 42–45
- user-editable, 41

Secondary file system won't mount, 55–56
sensors command, 193–194
Server error codes, 153–154
Servers. *See also specific servers*.
- process statistics, 179–180
- slow or no response. *See* System load.
- too hot, 192–194

Servers, cannot communicate
- blocked ICMP packets, 72
- client problem *versus* server, 69
- default gateway, pinging, 71–72
- DNS, testing, 72–74
- DNS inaccessible, 73
- DNS not configured, 73
- firewall rules, displaying, 77–78
- firewalls, detecting, 76
- within the local network, 71–72
- missing search path, 73–74, 97–98
- network connection, checking, 69–70
- network interface, checking, 70–71
- port 80, testing, 76, 77–78
- remote host, routing to, 74–75
- remote host, testing locally, 76–78
- remote port, testing, 76, 77–78
- routing table, displaying, 71–72

Single-user mode, 40
SMART tools, 186–190
smartctl command, 189
smartd daemon, 189
Software interrupts, displaying, 23
Sorting
- files, by size, 61
- top command output, 26

Space usage, displaying, 59–61
Spam Blackhole List (SBL), 132–133
Spam reduction, 130, 132–133
Speed. *See* Performance.
Splash screens, disabling, 51
Startup scripts, 40–41
Statistics, data files, 30–31. *See also* Metrics; *specific statistics*.
Statistics, displaying
- CPU, 30–31
- disk, 32
- RAM, 31–32
- for specific days, 33

Status codes. *See* Error codes; HTTP status codes.
status command, 177
Steal time, displaying, 23
Successful error codes, 150–151
.swp files, size issues, 61
sysstat package
- CPU statistics, displaying, 30–31
- disk statistics, displaying, 32
- installing, 30
- RAM statistics, displaying, 31–32
- run frequency, modifying, 30

System CPU time, displaying, 23
System init scripts, 41
System load, diagnosing
- after the fact, 29–33
- high I/O wait, 27–29
- high user time, 24–25
- out-of-memory issues, 25–27
- RAM usage, 25–27
- top command, 20–24

System load, load average
- CPU-bound, 20
- high, 20
- I/O-bound, 20
- overview, 19
- RAM-bound, 20

System load, overview, 18–19
System operations, understanding, 13–14
System V init, 39–42

T

tcpdump tool
- filtering output of, 86
- output file size, managing, 87
- packet captures, 84–88
- parsing output, 85
- replaying captured packets, 88
- saving output to a file, 86–87

tcptraceroute package, 75
telnet
- cannot connect, 136–137
- connects, message rejected, 137–138
- displaying listening ports, 143
- sending a test email, 127–129
- testing a remote port, 76
- testing web servers, 148–149

Testing
- database servers, 174–175
- DNS (Domain Naming System), 72–74
- local gateway, 72
- MySQL, 175
- port 80, 76, 77–78, 143–146
- PostgresSQL, 176
- quick *versus* slow, 8–9
- remote hosts locally, 76–78
- remote port, 77–78
- simple *versus* complex, 8–9
- web servers, 146–149

3xx redirection codes, 151–152
Time To Live (TTL) values, 108–112
/tmp files, size issues, 61
top command
- overview, 20–22
- tracing bandwidth consumption, 81–83

top command, output
- example, 21
- interpreting, 22–24
- sorting, 26

+trace argument, 102–104
traceroute command
- finding network slowdowns, 80
- routing to a remote host, 74–75
- troubleshooting DNS issues, 79

Tracing
- DNS queries, 101–104
- email requests, 120–123, 125

Tracking changes, 12–13
Troubleshooting, favoring past solutions, 9–10. *See also specific problems.*
TTL (Time To Live) values, 108–112
2xx successful codes, 150–151

U

Upstart scripts, 42–45
Uptime, metrics, 177
uptime command, 18–19
User CPU time, 23–25
User-editable script, 41

V

vi editor, 155

W

watch command, 162
Web servers
- configuration problems, 163–164
- logs, enabling DNS resolution, 158
- permission problems, 164–165

server status pages, 168–169
sluggish performance, 166–168
statistics, displaying, 158–162
Web servers, unavailable
CPU-bound load, 166–168
displaying firewall rules, 145–146
high load, 166–168
I/O-bound load, 166–168
port 80, testing, 143–146
RAM-bound load, 166–168
testing from the command line, 146–149
Wireshark program, 88–91

Z

Zone syntax errors, 112
Zone transfer issues, 113–117