2

VPN Fundamentals

THIS CHAPTER COVERS THE IMPORTANT ISSUES you need to be aware of before choosing and deploying a VPN solution. It describes the conventions used in the examples throughout this book. It also describes various related concepts in the context of VPNs, such as firewalls, routing, and netmasks.

Our Conventions

In this book, we describe a number of VPN technologies and packages. For each specific VPN package or setup, we provide examples corresponding to the following three basic VPN scenarios you might have: network-network, host-network, and host-host.

In the *network-network* scenario, two subnets are connected using a VPN tunnel. Each subnet should consist of a gateway/router and a host. The gateway for each subnet should have two network interfaces: one to the outside world and one to the gateway's internal subnet. You can use this scenario to securely connect to your company's branch overseas, for example.

The *host-network* scenario can be interpreted as a degenerate case of the network-network scenario. The *host-network* scenario is used when one of the subnets to be connected consists of just one host. To illustrate, you use the

host-network scenario when telecommuting or when you want to securely connect to a corporate network while on the road.

The *host-host* scenario is formed by further reducing the host-network scenario so that it consists of only a pair of hosts that want to connect to each other. For example, you might want to use this scenario to establish a secure tunnel between two hosts that belong to different networks. This scenario is different from others in that only these two hosts can send secured traffic to each other. If more machines need to communicate securely, a host-network or network-network setup would be more appropriate.

Gateways vs. Networks

In this book, we say that VPNs have the capability to connect networks, for example the host-network or network-network VPN scenarios. All VPNs have two hosts that handle the encryption/decryption of the VPN traffic, the endpoints of the VPN. When one or both of these hosts allows access to a network of machines rather than to just the single host, we call the host a *gateway*.

The concept of a gateway is already standard networking terminology. For example, the router that connects a business to its ISP is a gateway, as could be a firewall through which all traffic passes. In VPN terminology, a gateway is simply a VPN endpoint that sits in front of a network that has access to the VPN.

In other literature, you might see our three VPN senarios referred to as host-to-host, host-to-gateway, and gateway-to-gateway. The former explicitly mentions the endpoint of the VPN—be it a host or gateway—while the latter two describe what you want to connect—a host or a network. The two forms of terminology are functionally equivalent, but we prefer the "network" form because it is easier to see what you are trying to connect—one or more networks—instead of specifying how this connection is implemented.

For instance, if we use a VPN package and configure it to connect 192.168.1.0/24 and 192.168.2.0/24, we are connecting two networks. Each of the networks will have a gateway (say, 192.168.1.1 and 192.168.2.1) that will forward traffic for its respective network.

Sample Scenario

To help you understand the scenarios just described, we will consider a simple network-network scenario (see Figure 2.1), which is similar to those used in the examples throughout the book.

As you can see, the network-network scenario shown in Figure 2.1 consists of two networks, one in Chicago and one in Atlanta.

The Chicago network is 192.168.1.0/24, with the VPN server/gateway (Bears) located on the internal network on IP address 192.168.1.1. Several hosts, such as Cubs, Bulls, and Blackhawks, are on the internal Chicago network. Atlanta has a similar configuration, with Falcons as its VPN server/gateway.

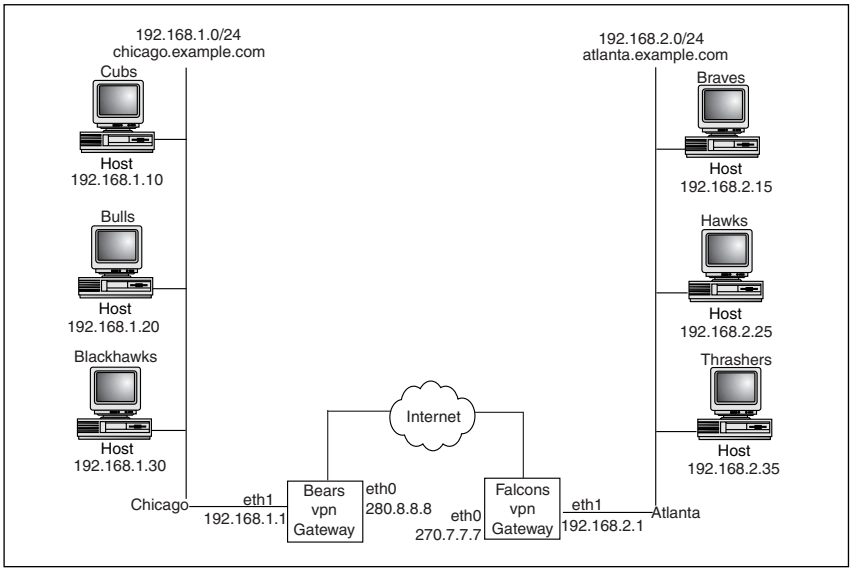

Figure 2.1 A network-network scenario example.

Both networks use addresses in the private-network range, as specified in RFC 1918. (We detail the RFC 1918 networks later in this chapter.) The IP addresses on the outside (280.8.8.8 and 270.7.7.7) are fictitious Internet-routable IP addresses that the two machines will use for their actual communication.

Our External IP Addresses

You might notice something fishy with our two external IP addresses: 280.8.8.8 and 270.7.7.7. These addresses are not legal; the legal range for any byte in an IP address is 0 to 255. We did not want to use actual routable IP addresses, lest you accidentally type them in and attempt to establish a VPN between you and some unsuspecting party. The examples we provide will do one of the following:

- Use private IPs as external IPs, explicitly stating that you should replace them with real routable Internet IPs.
- Use w.x.y.z addresses as external IPs, where w is greater than 255 and is thus illegal on the Internet.

Our network-network scenario can be converted into the host-network one by removing the eth1 interface and the entire 192.168.1.0/24 network from Bears and connecting Bears to Falcons. Similarly, the host-network scenario can be converted into host-host by removing the eth1 interface and the 192.168.2.0/24 network from Falcons and having Falcons and Bears be the only two machines involved in the VPN.

Considerations

Before deciding what kind of VPN is right for you, you need to first define your requirements, conduct a risk analysis, and review collateral issues such as long-term support and operational management needs. In the next few sections, we will describe various issues for you to consider.

Key Distribution

Key distribution among your VPN clients and servers should be one of the first security concerns you review. If your keys are compromised, the security of your system is compromised. When we talk about key distribution, we are concerned with two types of keys: symmetric and asymmetric.

It is crucial to be able to distribute symmetric keys securely. Ideally, you should use a secure out-of-band channel or physically access both systems and set keys up yourself.

In most cases, however, this is not an option. If you have to distribute symmetric keys remotely, make sure you at least use SFTP, SCP, or SSL/TLS. (Telnet and FTP are insecure. If you use them, you are potentially giving your keys away to attackers.) You could even go so far as to hire a professional courier or have one of your employees travel to the remote site with the keys on a floppy disk.

Asymmetric keys (which consist of both a public and private key pair) are a different matter. The public key can be transmitted without secrecy, such as with FTP or even email. Public keys by themselves do not offer an attacker anything that can be leveraged to break into your VPN. The secret key should not be transmitted to the remote end at all because it is only needed for decryption.

Although it is completely fine to transfer public keys in nonsecure manners, it is crucial for you to verify that the keys have not been tampered with when they are received. This can be done simply by installing the key as appropriate to your VPN software and then calling up the administrator at the other end of the VPN link and reading the key aloud. If you transmit your key in a way that allows an attacker to modify it, the attacker could perform a man-in-the-middle attack, defeating the security of your VPN.

In short, reliable transmission of the public key is important, though secret transmission is irrelevant. A secret key should never be stored on the remote endpoint of the VPN. If you are not familiar with asymmetric cryptography, see our glossary (Appendix C) for explanation.

Scalability

VPNs, like all pieces of your network infrastructure, must be able to support the traffic you have and be able to scale with the traffic you will support in the future. If you are implementing a VPN only to connect central and branch offices and have a limited growth plan, you are probably not concerned much with scalability because most VPN technologies will provide what you need. If you are rolling out an enterprise-wide VPN service and are supporting many dial-in users, scalability will be near the top of your list of VPN requirements.

Three main issues help define scalability:

- Capability to handle more connections
- Ease of maintenance and support
- Cost

All three of these issues will depend on the VPN you choose and its design. The topology you use can also greatly affect scalability.

Star Topology

We will now examine a sample network-network VPN that has been built using the star topology (see Figure 2.2).

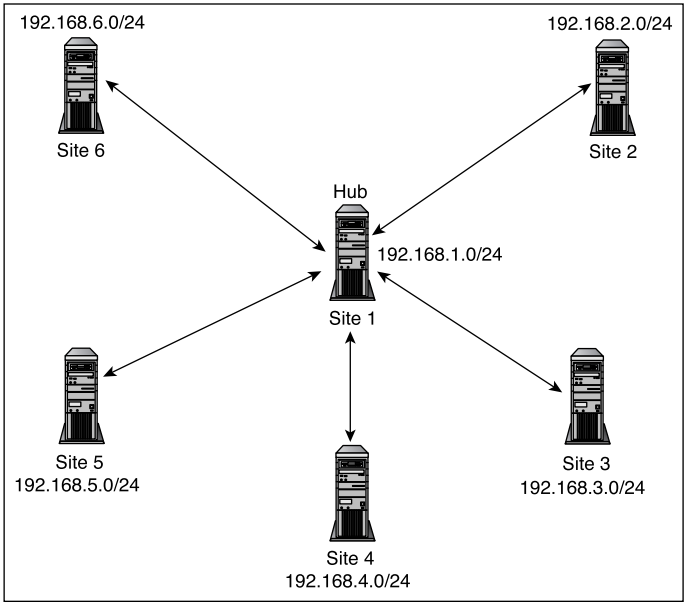

Figure 2.2 The star topology for VPNs.

In the star topology, you have a VPN connection from each remote site to the main VPN hub, Site One. The VPN hub must be able to support *n* VPN connections, where *n* is the number of remote sites. Each remote site that wants to communicate securely must send its traffic through the VPN hub in the center.

The main benefit of this model is that adding new sites is straightforward because all configuration changes are made at a central location, the VPN hub. The drawbacks are as follows:

- There is a single point of failure. If the hub goes down, all the spokes can no longer communicate.
- If performance on the hub suffers, performance for all VPN traffic suffers.
- Two nodes that might be geographically close still have to route through the hub to communicate.

Most of the drawbacks can be mitigated by adding more hubs to distribute the load and provide fault tolerance.

Full Mesh Topology

Next, we will review a VPN built using the full mesh topology. In the fully meshed design, each site communicates directly with every other site. Figure 2.3 shows our sample network built as a fully meshed VPN.

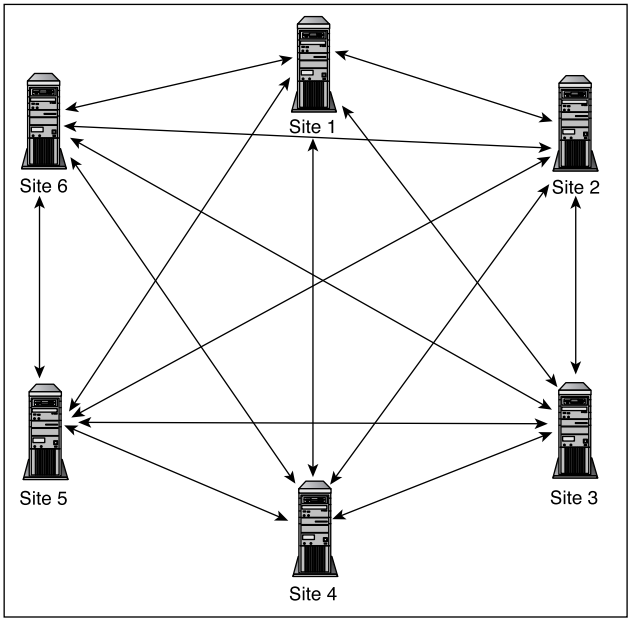

Figure 2.3 The full mesh topology.

Again, this VPN model has several benefits and one main drawback. The benefits are as follows:

- There is no more single point of failure. Sites do not rely on a hub for intra-VPN communication. VPN connectivity is lost to the problematic network, not all networks.
- Overall performance is not limited to a single system.
- Geographically close sites can now communicate directly.

The drawback is increased maintenance. If you add a new node, you would typically create VPN associations with every other node.

Although this topology has only one drawback, it is a big one. Instead of managing your keys at a central site, you must now manage them on every node. Again, if you have 1,000 nodes, this can be a formidable task.

> **Note**
> Some VPN solutions, such as `tinc` (see Chapter 10, "*tinc*"), handle routing for you automatically. This probably will not help much with a 1,000-node setup, but it can still save you a lot of time and effort.

Issues

From both of the preceding examples, we see that adding nodes can be a lot of work. Both star and fully meshed topologies need a scalable way to distribute keys in a secure fashion.

Instead of placing keys on each server, one solution some VPNs have implemented is retrieving key information from a centralized source. FreeS/WAN (see Chapter 6, "FreeS/WAN") supports what is called "opportunistic encryption" and is able to pull key information from DNS.

Another issue that might be a problem is routing. You need to be able to propagate routing information (perhaps by running a common IGP routing protocol such as RIP or OSPF within your networks) or be stuck hard-coding routes manually.

Interoperability

VPN software—even products based on open standards and RFCs—has notoriously had problems with interoperability. For example, administrators frequently find that different standards-compliant commercial solutions don't actually work well together or at all. Your interoperability needs can make or break your VPN decision.

First, you must decide whether you will be connecting to VPNs not controlled by you. If you are connecting to devices over which you have no control, it is best to choose a VPN that is standards based for maximum interoperability.

FreeS/WAN is a good choice for interoperability. IPSec is an industry standard for traffic encryption and has been implemented by many different vendors. Although some of the implementations have only partial support for some of its features, usually both sides of the VPN can be configured to use a set of commonly shared features.

FreeS/WAN itself does not use 56-bit DES for encryption; it supports only triple (168-bit) DES as it is currently bundled. Although this was done by the developers to make FreeS/WAN more secure, it can cause compatibility issues with other implementations. Fortunately, most vendors have implemented IPSec to include support for triple DES as well. This is just one example of the possible snags you will find as your try to connect one protocol implementation to another.

Overall, the issue of interoperability boils down to this question: "Will I ever need to communicate with VPN implementations not under my control?" If so, stick with a standards-based solution. If not, interoperability becomes a nonissue. Just make sure to plan far enough ahead so that you are not constantly rethinking your VPN.

Cross-Platform Availability

Another thing to consider is whether your VPN package will need to run on multiple platforms.

Some VPN packages rely on a common interface that is available for many platforms. For example, the universal TUN/TAP driver provides such an interface, which is used by cIPe (see Chapter 9, "cIPe"). As a result, cIPe is less coupled to the underlying architecture, and as soon as the driver is ported to a new platform, cIPe can then be added relatively quickly.

Cost

Fortunately, because Linux is a free OS, costs incurred from building a Linux-based VPN are usually smaller than for other commercial solutions. The costs you will encounter are for the hardware and the Linux installation media, although the installation can be downloaded free from a distribution mirror.

Other operating systems require licensing for the OS as well as per-seat licensing for the VPN software. Linux has none of these licensing requirements. Not only is Linux VPN software free, a number of free firewall and security packages can be used in conjunction with the VPN software to further secure your site. If you were using similar software on commercial OSs, it could get expensive very quickly.

Although Linux VPNs are cheap, client VPN packages for WinTel platforms are scarce, and as a result, they might not be the best choice for end user VPNs (unless your end users are UNIX savvy, which might rule out all but your engineers). Linux VPNs do provide a great solution for network-network VPNs, where end users and their inferior OSs are not directly involved.

VPN and Firewall Interaction

VPNs enable you to set up secure communications between endpoints and are just one weapon in your security arsenal. Firewalls, an older and more established security technology, are common in almost every environment. A VPN should be integrated into your security policy and that means making sure your VPN and firewall play nicely with each other.

Types of Firewalls

Three main types of firewalls are in common use today: packet filters, application gateways, and stateful inspection firewalls. They are described in the following sections.

Packet Filters

A packet filter is the simplest form of firewall. A packet filter firewall will compare any IP packet that attempts to traverse the firewall against its access control list (ACL). If the packet is allowed, it is sent through. If not, the packet filter can either silently drop the packet (`DENY` in `ipchains` speak) or send back an ICMP error response (`REJECT`).

Packet filters only look at five things: the source and destination IP addresses, the source and destination ports, and the protocol (UDP, TCP, and so on). These tests are very fast because each packet contains all the data (in the packet headers) necessary to make its determination. Due to its simplicity and speed, a packet filter can be enabled on your routers, eliminating the need for a dedicated firewall.

One problem with packet filters is that they generally do not look deeply enough into the packet to have any idea what is actually being sent in the packet. Though you might have configured a packet filter to allow inbound access to port 25, the Simple Mail Transfer Protocol (SMTP) port, a packet filter would never know if some other protocol was used on that port. For example, a user on one system might run his Secure Shell (SSH) daemon on that port, knowing that the traffic would be allowed by the packet filter, and be able to SSH through the firewall against policy.

Another problem with packet filters is that they are not effectively able to handle protocols that rely on multiple dynamic connections. The FTP protocol, for example, opens a command channel on which the various commands such as `USER`, `RECV`, and `LIST` are sent. Whenever data is transferred between the hosts, such as files or the `LIST` output, a separate connection is established. You would need to have an ACL that would allow these data connections through for FTP to work. However, packet filters do not read the FTP command channel to know when such an ACL should be permitted.

Application Gateways

An application gateway goes one step beyond a packet filter. Instead of simply checking the IP parameters, it actually looks at the application layer data. Individual

application gateways are often called *proxies*, such as an SMTP proxy that understands the SMTP protocol. These proxies inspect the data that is being sent and verify that the specified protocol is being used correctly.

Let's say we were creating an SMTP application gateway. It would need to keep track of the state of the connection: Has the client sent a `HELO/ELHO` request? Has it sent a `MAIL FROM` before attempting to send a `DATA` request? As long as the protocol is obeyed, the proxy will shuttle the commands from the client to the server.

The application gateway must understand the protocol and process both sides of the conversation. As such, it is a much more CPU-intensive process than a simple packet filter. However, this also lends it a greater element of security. You will not be able to run the previously described SSH-over-port-25 trick when an application gateway is in the way because it will realize that SMTP is not in use.

Additionally, because an application gateway understands the protocols in use, it is able to support tricky protocols such as FTP that create random data channels for each file transfer. As it reads the FTP command channel, it will see (and rewrite, if necessary) the data channel declaration and allow the specified port to traverse the firewall only until the data transfer is complete.

Often there is a protocol that is not directly understood by your application gateway but that must be allowed to traverse the firewall. SSH and HTTPS are two simple examples. Because they are encrypted end to end, an application gateway cannot read the traffic actually being sent.

In these cases, there is usually a way to configure your firewall to allow the appropriate packets to be sent without interference by the firewall. You might hear this called a *plug*, which comes from the `plug-gw`, a part of the Firewall Toolkit (FWTK) that was used to connect a client and server directly when the protocol was not supported.

It can be difficult to integrate application gateways into your standard routing hardware due to the processing overhead. Some newer high-end routers are able to function as application gateways, but you'll need a lot of CPU power for acceptable performance.

Note

Even application gateways can be fooled if you are crafty enough. For example, you could tunnel any arbitrary protocol over SMTP: The client could send data as the `DATA` portion of the transaction, and the server could respond in the resulting error/success code message. The nature and ubiquity of the HTTP protocol makes it even easier; SOAP and .NET are just two "accepted" examples of tunneling other protocols across HTTP. Httptunnel, available at `www.httptunnel.com`, is a good freeware tool capable of tunneling any protocol across HTTP.

Stateful Inspection Firewalls

Stateful inspection firewalls are a middle ground between application gateways and packet filters. Rather than truly reading the whole dialog between client and server, a

stateful inspection firewall will read only the amount necessary to determine how it should behave.

Take the SMTP `DATA` command, for example. When this command is sent, the client will send the data (the text of the email) ending with a line containing a single ".". The server then responds with a success or error code. An application gateway will need to be reading all the data that is sent and looking for the "." and error code. A stateful inspection firewall, however, will realize that the client is sending data until the server responds. Thus, it will simply forward the client's packets without inspection until the server responds. Simply put, a stateful inspection firewall understands the manner in which stateful protocols must conduct themselves and manages that traffic accordingly within the confines of its rulebase.

By not reading all the data sent, a stateful inspection firewall achieves a significant performance gain over an application gateway while maintaining the higher level of security and protocol support. Our VPN traffic, however, will be encrypted end to end. As such, there will be very little that a stateful inspection firewall will need to look at in our VPN datastream—it can't inspect the actual data anyway. Because of this, there is no functional difference between a stateful inspection firewall and an application gateway for our VPN traffic. There is, however, a solid performance boost from using a stateful inspection firewall. Because our VPN is already introducing latency due to the overhead of encryption, the more performance you can get with your firewall, the better.

Stateful Inspection Concerns

Stateful inspection firewalls have suffered from occasional faults. For example, to support the FTP protocol, stateful inspection firewalls would look at the beginning of a packet to see whether a data channel was requested. An attacker could trick the server into sending a large error message containing a data channel request that aligned perfectly at the beginning of a second packet. The firewall would then open a channel dictated by the attacker. A program called `ftpd-ozone` (available at `www.monkey.org/~dugsong/`) can exploit this vulnerability. This particular problem has been fixed by firewall vendors, but there might be more lurking around.

Common Linux Firewalls

Several firewalls can run on Linux. We will briefly describe them here, but it is beyond the scope of this book to cover them in depth. See New Riders' *Linux Firewalls, Second Edition* by Robert Ziegler (2002) for more information on firewalls.

- **The Firewall Toolkit (FWTK).** This was the first publicly available application gateway suite and was the basis of the commercial firewall Gauntlet. It is a set of userland applications that support various protocols such as STMP, HTTP, `telnet`, and X11. It is still available at `www.fwtk.org`, though it has not been officially supported in years.

- **IPF.** A Linux packet filter for 2.0 kernels. If you are still using a 2.0 kernel, you should really upgrade.
- **IPChains.** The Linux packet filter for 2.2 kernels. Though it is a simple packet filter, you can support some protocols via kernel modules. The `ip_masq_ftp` module enables you to support the FTP protocol, for example. The problem with IPChains is that the kernel packet filters are handled before the modules can see packets, meaning you must allow inbound access to ports that potentially could be required by the kernel modules.
- **IPTables.** The Linux firewall software for 2.4 kernels, also known as Netfilter. IPTables supports both packet filtering and application gateway support together. Taking the FTP protocol as our example, the data channels that are used are accepted as `RELATED` packets. This makes your firewall much cleaner because you do not need to leave holes in your firewall for packets that might or might not be used by modules.
- **IPFilter.** Created by Darren Reed, IPFilter is the default kernel packet filter of NetBSD and FreeBSD, it and runs on many other UNIX-like systems. IPFilter only runs on the very old Linux 2.0 kernels, so if you want to use IPFilter, we suggest that you use a BSD machine instead. It is available at `http://coombs.anu.edu.au/~avalon/`.

Note

Darren Reed modified the IPFilter license (or clarified it, depending on your position) to deny modifications to the source, rendering it not completely open source. Because of this, it was removed from OpenBSD. Reed later allowed modifications when used as part of NetBSD and FreeBSD. OpenBSD is creating a new packet filter, PF, from scratch to replace IPFilter. Yes, now and then there doth be strife in the open-source community ...

- **Dante.** Though normally included as part of a larger commercial firewall system, Dante is available as open source under a BSD-style license. It is a circuit-level firewall/proxy that is largely transparent to users. It is available at `www.inet.no/dante/`.
- **T.REX.** Open sourced in April 2000, T.REX is a sophisticated application gateway firewall that includes intrusion-detection features, strong authentication support, and extensive logging. Version 2 of T.REX was not yet available at the time this was written; though precompiled versions are available on CD. You can obtain T.REX at `www.opensourcefirewall.com`.

Placing Your Firewall

We strongly suggest that you use a firewall as part of your security infrastructure. Using a VPN in conjunction with your firewall, however, requires careful planning and configuration. Several different configurations are available when using both a firewall

and a VPN server. Each has its pros and cons, and we'll do our best to help you pick the option that makes sense for you.

VPN Server on a Firewall

The solution that feels most natural is to install your VPN software on your firewall itself. Many commercial firewalls include VPN components as an extra option. If you are working in a mixed commercial and freeware VPN environment, you will likely end up supporting this configuration.

As seen in Figure 2.4, we have a single point of entry into our network, which serves three purposes:

- The firewall allows outbound access to the Internet.
- The firewall prevents inbound access from the Internet.
- The VPN service encrypts traffic to remote clients or networks.

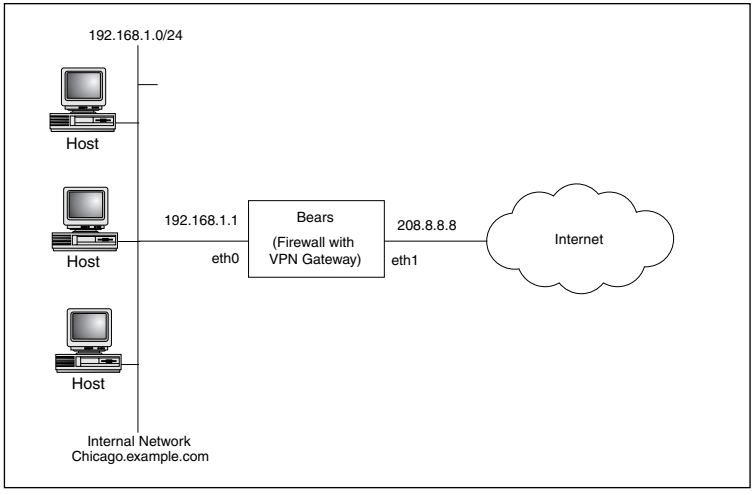

Figure 2.4 The VPN server on a firewall.

The pros of putting your VPN on your firewall are as follows:

- You have one place controlling all your security, meaning fewer machines to manage.
- You can create firewall rules that apply to your VPN traffic using the same tools you already use to manage your firewall.

The cons of putting your VPN on your firewall are as follows:

- You have one place controlling all your security. You'd better make sure this machine is extremely secure.

- You must configure your routes carefully to make sure the traffic goes through the appropriate interfaces (`eth0` vs. `vpn1`, for example.)
- Improper configuration in your firewall rules could allow traffic from the Internet to get through to the inside by slipping through using VPN addresses.
- Your Internet and VPN traffic will compete for resources on the machine, so a larger machine is likely necessary.

VPN Server Parallel to Firewall

Another topology that seems logical is to use your VPN parallel to your firewall. Your internal machines will still point to the firewall as their default route. The firewall will have a route to the VPN-accessible networks via the VPN server and will inform clients to send packets to the VPN machine when appropriate. (The "Routing" section later in this chapter discusses how this works.) You can see this topology in Figure 2.5.

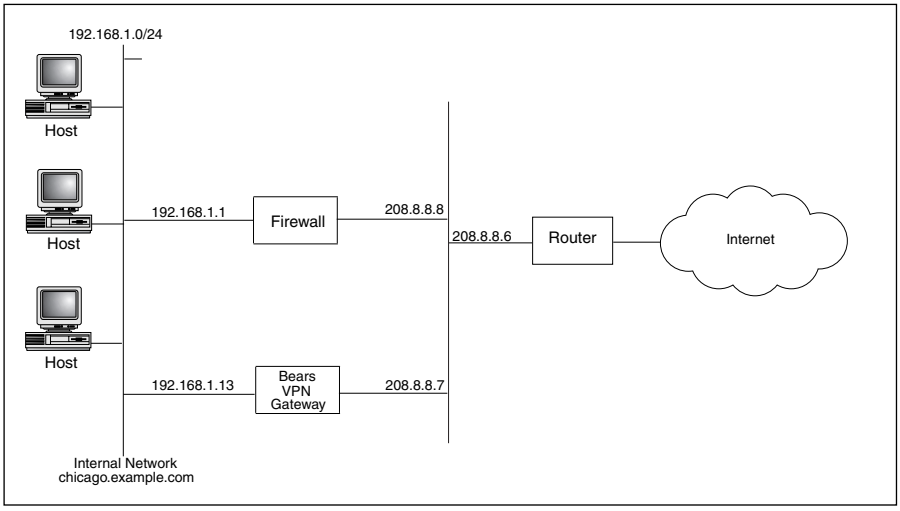

Figure 2.5 The VPN server parallel to a firewall.

If you prefer, you can place a router between the internal network and the VPN and firewall machines. You would configure the router to know the networks accessible through the VPN and would set up your routing rules there instead of on the firewall.

The pros of putting your VPN server parallel to your firewall are as follows:

- Your VPN traffic is not going through the firewall at all, so you do not need to modify your firewall configuration to support the VPN packets. Because some VPN protocols are not supported by all firewalls, this might be your only option.
- You can scale much easier. If you find that a VPN server is under too much load, you can add machines and distribute the VPNs that can be established

between them. If you are connecting multiple remote networks, you can have one VPN machine per network.

- If you are supporting roaming users in a host-network configuration, you can simply add VPN servers and use round-robin DNS to distribute the load between the VPN servers.

The cons of putting your VPN server parallel to your firewall are as follows:

- Your VPN server is directly attached to the Internet. You had better be very sure that it is well secured; otherwise, an attacker could break in and have direct access to your internal network.
- You now have two machines that communicate with the Internet, and you must make sure that both are correctly configured to only pass legitimate traffic, thereby increasing the support workload and associated costs.
- Cost, of course, increases incrementally with the addition of new servers and related support staff.

VPN Server Behind Your Firewall

Another location for your VPN server is to put it behind your firewall completely, attached to the internal network. As shown in Figure 2.6, our VPN server is not directly reachable from the Internet at all, and all packets must reach it through our dedicated firewall.

As with the previous topology, you will need to add a route to your firewall that redirects VPN traffic from internal machines to the VPN server. You will also need to configure your firewall to pass the encrypted VPN traffic directly to your VPN server.

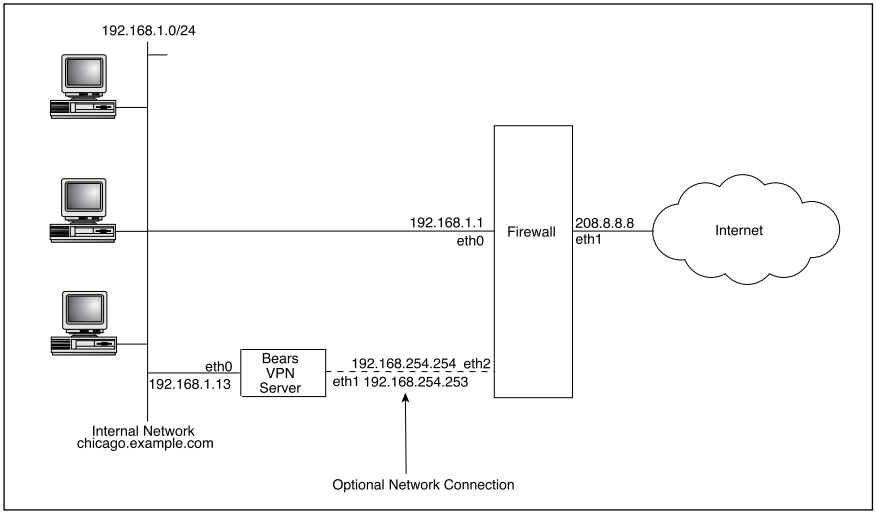

Figure 2.6. The VPN server behind a firewall.

The pros of putting your VPN server behind your firewall are as follows:

- The VPN is completely protected from the Internet by the firewall.
- There is only one machine controlling all access to and from the Internet.
- Network restrictions for VPN traffic are located only on the VPN server, which can make writing rulesets easier.

The cons of putting your VPN server behind your firewall are as follows:

- All the VPN traffic must travel through the firewall as well, adding a degree of latency.
- The firewall will need to shuttle VPN traffic from the Internet to the VPN directly. Because it cannot inspect this traffic (it is encrypted, after all), it will require a simple packet filter ACL or application plug proxy.
- Getting your firewall to pass the encrypted VPN traffic to the VPN server is sometimes tricky. Some firewalls do not know what to do with IP protocols other than ICMP, TCP, and UDP. This means it might be harder or impossible to support VPNs that use different IP protocols, such as ESP packets for IPSec VPNs or GRE packets for PPTP VPNs.
- All the VPN traffic travels through the same LAN wire twice, once from client to VPN server in the clear and once from the VPN to the firewall encrypted. This can degrade LAN performance.

One option to decrease latency would be to connect a second network card (`eth1`) on the VPN server directly to a network card on the firewall (`eth2`) with a crossover cable. You could use a hub or switch and create an actual network segment if you prefer, but because you'd only have the two hosts here, a crossover cable would work as well and would be faster because no extra equipment is involved. This enables you to route the encrypted VPN traffic directly over this wire and through the firewall rather than backtracking over the existing LAN segment. This means the VPN traffic won't be competing for LAN resources.

The direct VPN-to-firewall link will need to either be a point-to-point link or be given its own network segment. If you prefer the latter, we suggest a small network such as a /252, which can hold only two hosts, plenty for our purposes. We could allocate 192.168.254.254 to the firewall and 192.168.254.253 to the "external" VPN interface, and the network itself would be 192.168.254.252/252.

Configuring Your VPN with Its Own Firewall

In any of the configurations previously described, it is possible to restrict what traffic can traverse the VPN connection. This might be desired when the networks or hosts you are connecting are not of the same trust level.

For the case in which your Internet firewall and VPN server are the same machine, this can be achieved simply by adding new rules to the firewall using your existing firewall software.

For cases when you have a separate VPN server, you might either install a separate firewall machine in front of your VPN server or simply rely on Linux kernel packet filters on the VPN server itself. For example, if you want to allow only mail-related traffic to traverse your VPN, you might implement kernel restrictions on your VPN server, such as the following:

```
# Allow only SMTP, POP, IMAP and POPs/IMAPs through our VPN:
for port in 25 109 110 143 993 995
do
  ipchains -A output —destination-port $port -i vpn1 -j ACCEPT
  ipchains -A  input      —source-port $port -i vpn1 -j ACCEPT
done
ipchains -A output -i vpn1 -j DENY
ipchains -A  input -i vpn1 -j DENY
```

These rules assume that your VPN traffic goes through the virtual interface `vpn1`. This is a very simple example. You can create ACLs tailored for your VPN using whatever tools (IPchains, IPtables, IPfilter, and so on) you are comfortable with.

Networking Issues

You will need to address a fair number of issues when designing your network topology. If you do not plan, you will have a good deal of trouble getting things working correctly. A solid network topology map, complete with all the machines, business functions by IP address, routes, other network settings, and the driving business need are critical to establishing a successful VPN setup.

Dedicated vs. Dynamic IP Addresses

Because you are connecting machines across the Internet, the IP addresses of these machines are dictated largely by your Internet service provider. Your ISP will allocate to you one or more IP addresses that you will have available for Internet access.

If you have a dedicated connection such as a T1 or greater, you will likely be given an IP subnet by default. On the other hand, if you have DSL or a dial-up modem, most ISPs prefer to dole out IP addresses with the Dynamic Host Configuration Protocol (DHCP). This means your machine might have a different IP address each time you connect to the Internet.

It is not too much trouble for roaming clients to be given nonstatic IP addresses. When they dial up from their hotel, for example, they will simply point to the VPN server—say, bears.example.com—and establish a connection. Unfortunately, supporting roaming users with nonstatic IP addresses means your VPN server cannot restrict the IP of the connecting machine.

Your VPN server, however, must certainly have a dedicated IP address; otherwise, your remote networks and users will never be able to find it. In our example, bears.example.com must resolve to the IP address on which your VPN server is listening. Although you could survive by changing your DNS entries each time your VPN

server gets a new IP address, this is largely a hack. It also introduces an unnecessary level of complexity, and is time consuming.

Unless you want to spend a lot of time maintaining your DNS records, we highly suggest that you get a dedicated IP address from your ISP for all of your VPN servers.

Internal Subnets

If you are creating a host-network or network-network VPN, you will need to have different network IP ranges for each network you want to connect. In our sample network described at the beginning of this chapter, we have two interconnected networks. Chicago is 192.168.1.0/24 and Atlanta is 192.168.2.0/24.

We did not pick these networks entirely at random. These are a set of networks reserved for private internal use, specified in RFC 1918. By using only these networks, you are guaranteed not to have a conflict with a network that exists on the Internet. The "private" networks are as follows:

Network	Number of Networks	Hosts Per Network
10.0.0.0/8	1	16777214
172.16.0.0/16 through 172.31.0.0/16	16	65534
192.168.0.0/24 through 192.168.255.0/24	256	254

We chose the 192.168.x.y networks for Chicago and Atlanta because they are small and only need to support about 20 hosts each, easily fitting within the 254 maximum allowed. We could have used subnets of the 10 or 172 networks instead, but we are lazy and would rather use networks that do not require custom subnetting.

Unless you have large blocks of IP addresses allocated to you—which is very rare these days—you should consider using RFC 1918 networks that are dedicated for internal use while using network address translation (NAT) on your gateway to the Internet. It is customary to use similar network numbers for all your internal networks, as we have in our sample network. You might even want to use conventions for your networks—for example, using 10.x.y.z for standard internal networks, 192.168.x.y for VPN links and dedicated lines, and so on.

The important thing is to make sure you pick networks that do not conflict with each other and to use the smallest subnetworks possible such that you do not run out of IP ranges.

Netmasks

If you are new to networking, you are probably wondering what all these /24 and /8 things are. These are called Classless Interdomain Routing (CIDR) blocks, and the network 192.168.1.0/24 is written in CIDR notation.

To understand this, let's start with a history lesson. In the beginning (we love saying that), the IP space was broken into the following blocks:

[0-127].x.y.z	Class A
	16777214 hosts per network
[128-191].x.y.z	Class B
	65534 hosts per network
[192-223].x.y.z	Class C
	254 hosts per network
[224-239].x.y.z	Class D
	Available for multicast network protocols only
[240-255].x.y.z	Class E
	Reserved for future use

There were 128 Class A blocks, each capable of supporting 16777214 hosts. There's no way a single network with 16777214 hosts could have any hope of good performance. Thus, networks were broken up into subnets. Hosts were put onto different subnets based on function or location, and routers connected the various subnets.

Let's say we take the Class A network 10.x.y.z and break it up into 256 Class B networks of the form 10.[0-255].y.z. Each network now has 65534 possible hosts (still a relatively unreasonable number from a management perspective).

By default, a host that saw that its IP address was 10.1.1.1 assumed it was on a full Class A and assumed that if it wanted to communicate with 10.2.2.2, they were on the same physical wire. To be able to correctly understand that 10.2.2.2 was on a different network, the concept of subnet masks was formed.

A *subnet mask* is simply a dotted quad (like an IP address) that signifies how much of an IP address is the network portion and how much is the host portion. The Class A networks have a default subnet mask of 255.0.0.0, the Class B networks have a default subnet mask of 255.255.0.0, and the Class C networks have a default subnet mask of 255.255.255.0.

To determine the network portion of an IP address, you can perform a bitwise (or Boolean) AND of the IP and netmask. To break our huge Class A network into 256 smaller networks, the administrators use a netmask of 255.255.0.0 instead of the default. Thus, the host 10.2.2.2 is on the 10.2.x.y network, and the host 10.1.1.1 is on the 10.1.x.y network.

A different way of viewing the subnet mask is to represent it in binary form. The common netmasks would be represented as follows:

Subnet Mask	Binary Form	CIDR Notation
255.0.0.0	11111111.00000000.00000000.00000000	/8
255.255.0.0	11111111.11111111.00000000.00000000	/16
255.255.255.0	11111111.11111111.11111111.00000000	/24

The CIDR notation at the end is simply a shortcut you can use to specify an IP address and its netmask in one quick form.

Thus, our machine 10.1.1.1 with a netmask 255.255.0.0 could be written as 10.1.1.1/16. The number after the slash is simply the number of 1s in the binary form of the subnet mask.

You can pick any subnet that is more restrictive than the default (that is, you could not pick a subnet of 255.0.0.0 for 172.x.y.z) when designing your networks. Most people find it easiest to use one of the standard subnets /8, /16, or /24.

This makes recognizing the network portion easier because the network portion always falls at a byte boundary.

For example, 192.168.5.0/24 is clearly on a different network than 192.168.10.0/24 simply by reading the first three numbers.

> **Addresses and Subnets That Are Unavailable**
>
> Under ordinary circumstances, the first and last IP addresses of any subnet are not available for use by an actual host. The first is reserved to identify the network itself, and the last is used for the broadcast address of the subnet. Thus, on the 172.20.5.0/24 network, you can neither use 172.20.5.0 (this network) nor 172.20.5.255 (all hosts on this network).
>
> Additionally, some network equipment might not let you use the first subnet at all, because it could cause confusion associated with having network and subnet addresses that are indistinguishable. If your hardware does not support using the first subnet, the entire 172.20.0.0/24 network is also off limits.
>
> The most notable instances of this router behavior are older versions of Cisco's IOS. Unless you configured your router with the `ip subnet-zero` option, the first subnet was off limits. Cisco IOS >= 12.0 includes `ip subnet-zero` by default. If you are using IOS prior to 8.3, the configuration option is named `service subnet-zero`.

In general, you should pick the smallest subnet possible that enables you to host your current and expected hosts. Usually, this means using a /24 subnet.

There are two special cases in which you might use subnets that are even more restrictive. If you are creating a network with only two hosts, such as a dedicated line between two hosts, you might want to use a /30:

IP address	Netmask	CIDR Notation
192.168.254.253	255.255.255.252	/30
192.168.254.254	255.255.255.252	/30

Here we have a network with only two hosts and no IPs left unused. Don't forget that the first and last are reserved for network and broadcast.

The other special case is a point-to-point link. In this case, you do not create a specific network at all; instead, you use a /32 (host) subnet. Take the following `ifconfig` output, for example:

```
$ ifconfig ppp0
ppp0      Link encap:Point-to-Point Protocol
          inet addr:192.168.254.254  P-t-P:192.168.254.253  Mask:255.255.255.255
```

Here we have a local address 192.168.254.254/32 that is connected via a point-to-point link to 192.168.254.253/32. This is commonly used for dial-up connections, but some of our VPNs will use this style of networking as well.

Network Conflicts

Sometimes you will find yourself in a position in which you have networks to connect that overlap. If this is because you planned poorly, you are now feeling the pain.

More commonly, this is caused by two companies merging. They both might have correctly picked networks from those available per RFC 1918, but they happened to pick the same ones. Ugh.

What do you do in this situation? Well, it's a nasty one. Changing all the IP addresses on one of the offending networks is certainly the cleanest option, though it might be time consuming. If you already allocate IP addresses with DHCP, you will be patting yourself on the back. You can simply change the DHCP server to use the new network and have machines reboot; they'll automatically be reconfigured. You must then only visit hard-coded machines and change them manually.

If changing IP addresses automatically is not an option, you can try to use network address translation (NAT) or port address translation (PAT) on your VPN servers to rewrite the outbound address. Let's say Atlanta and Chicago both use the same network addresses, 192.168.2.0/24.

If you configure Bears to rewrite all the Chicago addresses as if it came from itself, using a nonconflicting address (say, 192.168.3.1), Chicago machines will be able to access Atlanta machines even though they actually have the same network IP address conflicts. You'll need to perform some DNS trickery to supply Chicago hosts with fictitious IP addresses for the Atlanta hosts.

In short, it is possible, but you're going to need a network guru, some heavy planning, and many aspirin. In the end, you might just decide it's better to renumber one of the networks and save yourself the maintenance headaches of this horrible hack.

DNS for VPN

One often-overlooked requirement of a functioning VPN is DNS. For any host-network or network-network VPN, you will be enabling access to machines that are not available on the Internet at large. Unless you want to access machines only by their IP address, you want to have DNS work cleanly.

The easiest way to accomplish this is to create a new domain name for your internal networks. Let's say our company owns example.com, which we use for our external systems. We could create chicago.example.com and atlanta.example.com as internal domain names. We then would run a DNS server internally to support those domains.

Let's assume we install a DNS server on the internal machines Cubs and Braves. We can make Cubs authoritative for the chicago.example.com domain and Braves authoritative for the atlanta.example.com domain. We set up each machine to be secondary

for the internal domains it does not serve, which will enable them to send updates cleanly between them.

You then configure Cubs and Braves to relay all other queries to an external DNS server (say, one at your ISP), making sure you have recursive queries allowed from internal addresses. You configure all your internal machines to use Cubs and Braves as their DNS servers, preferring whichever is on the local network to avoid sending DNS traffic across the VPN.

Let's say a user on Bulls wants the IP address for thrashers.atlanta.example.com. Cubs already knows the answer because it is a secondary DNS server for the domain. Should Bulls request the IP for `www.buildinglinuxvpns.net`, Cubs will forward the query to an external DNS server and return the answer to Bulls when it is received.

This situation works seamlessly for all hosts on networks that are connected via dedicated VPNs. The only tricky situation is supporting roaming users. Because those VPNs are created only periodically and the users might want to be connected to the Internet without using the VPN, you cannot hard-code their DNS setting to use the internal DNS servers.

If possible, configure their machines to use the internal DNS servers only when the VPN is active. This can be done by munging the `ip-up` script when using a PPP-related VPN, for example, or by any other method you desire to rewrite `/etc/resolv.conf` when the VPN is established.

The worst-case scenario (next to remembering IP addresses, that is) is to simply point first to an internal DNS server and then to an external server, as seen here:

```
$ cat /etc/resolv.conf
search chicago.example.com example.com
nameserver 192.168.1.10          # cubs
nameserver 345.6.7.8             # My ISP
```

The internal DNS server Cubs (192.168.1.10) is first in the list. If the VPN is available, Cubs will handle your DNS requests for both internal and external domains. If the VPN is not up, the DNS request to Cubs will fail, and your machine will then query 345.6.7.8. This machine will be able to respond for all Internet addresses but not the internal chicago.example.com addresses. This is not a problem, however, because the internal addresses aren't available except when the VPN is running anyway.

You will experience some name-resolution lag when using such a setup when the VPN is not established. DNS queries contact hosts in the order specified in `/etc/resolv.conf`, only moving onto the next in the list after determining that the first server isn't responding. Some resolver libraries try to consider this and will stop asking a nonresponding server for a while.

Routing

As has been alluded to in the previous sections, we must make sure our hosts send their traffic to the correct hosts.

In the case in which you have a simple network connected to the Internet, from the viewpoint of one of the machines on the network, you usually have only two classes of hosts that are accessible.

The first class comprises those on the internal network, which can be contacted directly. The second class comprises all other hosts on the Internet, which you contact via your Internet firewall or gateway/router.

To set up routing for the hosts on our Chicago network, you need to do two things. First, you must assign IP addresses to their network interfaces. Second, you must add a default route to the gateway/router using the following command:

```
cubs# route add default gw 192.168.1.1
```

The preceding command shows how you can manually add routes. How you permanently set up a default route depends on your Linux distribution.

For example, Red Hat machines use the file `/etc/sysconf/network`, Debian uses `/etc/network/interfaces`, and so on. You can view your routing table with the `netstat` command. For example, the routing table on Cubs might look like the following:

```
cubs# netstat -rn
Destination Gateway      Genmask         Flags   MSS Window  irtt Iface
default     192.168.1.1  255.255.255.0     UG     0     0     0    eth0
192.168.1.0 0.0.0.0      255.255.255.0     U      0     0     0    eth0
127.0.0.0   0.0.0.0      255.0.0.0         U      0     0     0    lo
```

Based on this routing table, all traffic originating from Cubs destined for 192.168.1.0/24 (the internal network) is sent directly to the `eth0` interface. All other traffic originating from Cubs is sent using the default route to the gateway, 192.168.1.1. The gateway then performs NAT, if necessary, and forwards the traffic to its destination on the Internet.

Explicit VPN Routing Configuration

We've discussed three potential VPN and firewall topologies. In the case in which your VPN and firewall are the same machine, it should be obvious that your default route should point to the firewall, which is also conveniently the VPN server. This machine should be smart enough to know where packets should go, and thus the client should need no configuration changes.

So, what do you need to do when the firewall/gateway and VPN server are separate machines? You could configure the client machines to explicitly route VPN packets to the VPN server. If our remote VPN network were 192.168.2.0/24 and our VPN server were 192.168.1.13, we would run the following:

```
cubs# route add -net 192.168.2.0/24 192.168.1.13

cubs# netstat -rn

Destination Gateway      Genmask         Flags   MSS Window  irtt Iface
default     192.168.1.1  255.255.255.0     UG     0     0     0    eth0
192.168.1.0 0.0.0.0      255.255.255.0     U      0     0     0    eth0
```

```
192.168.2.0 192.168.1.13 255.255.255.0     UG     0     0     0    eth0
127.0.0.0   0.0.0.0      255.0.0.0         U      0     0     0    lo
```

Creating these VPN-related routes explicitly means you can clearly see exactly where your packets should go. It is a burden to set these up on each client, however.

The plain fact is that if you set up your firewall/gateway correctly, you have absolutely no need to explicitly define VPN-related routes on your client. We will see how this works in the next section.

Although explicitly defined VPN-related routes are not needed when you have a functioning VPN, they can be very helpful when setting up your VPN for the first time or when you need to debug problems.

If you are setting up a new VPN to connect to a remote network that is already available by other means (an existing VPN, a leased line, and so on), the only way to test the new VPN is by configuring static routes on one or more machines to test.

In the general case, however, you should avoid these static routes because they can only cause you headaches later should you change your network topology and find yourself fixing the routes on each and every client.

Centralized VPN Routing Configuration

If you prefer to configure your clients with a default route only rather than explicitly hard-coding each and every VPN-related route, we have good news for you: It's a cinch. Leave your client with the default gateway route and teach your default gateway the actual routes that should be used.

Let's say our firewall, 192.168.1.1, has the following routing table:

```
firewall# netstat -rn

Destination Gateway      Genmask         Flags   MSS Window  irtt Iface
default     280.8.8.7    255.255.255.0     UG     0     0     0   eth0
280.8.8.0   0.0.0.0      255.255.255.248   U      0     0     0   eth0
192.168.1.0 0.0.0.0      255.255.255.0     U      0     0     0   eth1
127.0.0.0   0.0.0.0      255.0.0.0         U      0     0     0   lo
192.168.2.0 192.168.1.13 255.255.255.255   UG     0     0     0   eth1
```

Our firewall machine has local interfaces `eth0` (the Internet segment) and `eth1` (the internal segment). It has a default route to the Internet via 280.8.8.6 (the router to the ISP) and a route for our Atlanta network, 192.168.2.0/24, through 192.168.1.13, our internal VPN server.

The clients all point to our firewall as the default gateway, so when Cubs wants to establish a TCP connection to braves.atlanta.example.com (192.168.2.15) through the VPN, the following packets will be sent:

```
Packet #   source   dest        tcpdump info

1          cubs     firewall    04:43:55.731166 < cubs.1600 > braves.www: S
➥669411963:669411963(0) win 32120 <mss 1460,sackOK,timestamp 79542679
➥0,nop,wscale 0> (DF)
```

```
2          firewall cubs        04:43:55.912572 > firewall > cubs: icmp: redirect
           braves to host bears [tos 0xc0]
3          cubs     bears       04:43:55.931166 < cubs.1600 > braves.www: S
           669411963:669411963(0) win 32120 <mss 1460,sackOK,timestamp 79542679
           0,nop,wscale 0> (DF)
4          bears    cubs        04:43:56.104105 > braves.www > cubs.1600: S
           1678687708:1678687708(0) ack 669411964 win 1460 <mss 1460> (DF)
5          cubs     bears       04:43:56.106602 < cubs.1600 > braves.www: .
           1:1(0) ack 1 win 32120 (DF)
6          cubs     bears       04:43:56.120659 < cubs.1600 > braves.www: P
           1:8(7) ack 1 win 32120 (DF)
7          bears    cubs        04:43:57.373603 > braves.www > cubs.1600: P
           1:200(199) ack 8 win 2920 (DF)
```

The preceding list is the output of `tcpdump` (a packet sniffer) watching the packets. Here's an explanation of each packet:

- Packet 1: Cubs sends a TCP packet to the firewall (its default gateway) with the final destination of Braves (192.168.2.15).
- Packet 2: The firewall, which knows that the 192.168.2.0/24 network is served by a different machine (Bears) tells Cubs "Hey, send data destined for braves to the machine bears instead." This is called an ICMP REDIRECT.
- Packet 3: Cubs resends the TCP packet to Bears with the final destination Braves (192.168.2.15).
- Subsequent packets: Cubs will talk to Bears directly for all communication to Braves. Bears handles sending this through the VPN.

Note

For your Linux VPN hosts to support ICMP REDIRECT messages arriving from a firewall, use the following:

```
root@fox # echo "1" > /proc/sys/net/ipv4/conf/all/accept_redirects
```

You can enable ICMP Redirects on the firewall as follows:

```
root@fox # echo "1" > /proc/sys/net/ipv4/conf/all/send_redirects
```

The ICMP REDIRECT is only needed for the first packet you want to send to a machine over the VPN. For the life of that connection, the client will automatically send the packets through the VPN server. There is no performance overhead to this configuration, save the one single ICMP REDIRECT at the beginning of each connection. As you can see from the timestamps in the `tcpdump` output earlier, this lag is hardly even worth mentioning.

We suggest that you configure all hosts to have a single default gateway and teach that gateway the VPN-related routes. This makes client configuration much easier because they need only the default route set up. It also means you have only one place you need to set up this route. If your firewall or gateway can exchange routing information with other routing hardware (via protocols such as RIP, OSPF, and so on), this makes the single point of configuration even more appealing.

VPN Server Behind a Firewall and NAT Traversal

If you use a routable IP address for the external interface (the one that is not on 192.168.1.0/24) of the Chicago VPN server, you can use the explicit or centralized routing configurations described earlier.

If you can't use a routable IP for the Chicago VPN server outside interface, you'll most likely need to get your VPN server to work over a NAT. Getting VPN solutions to work over NAT can be tricky.

For instance, for an IPSec-based solution, the easiest way is to have it work in tunnel mode. This is because one-to-one NAT mapping can be established (that is, the private IP address of the VPN server will be mapped to the routable IP address of the firewall). In transport mode, one-to-one mapping is a problem because packets can be coming from different IP addresses. Although a number of techniques have been proposed to deal with the problem, as of now, IPSec still does not have an official standard defining IPSec interoperation with NAT. (Check `www.ietf.org/html.charters/ipsec-charter.html` for more up-to-date information.)

Overall, NAT traversal is very implementation dependent and can be affected by a number of factors such as the transport protocols your VPN solution uses and so on. Your best bet is probably to consult the documentation and mailing lists for your particular VPN solution.

Trust Level

Trust is important in an environment in which security is a concern. (In a paranoid world, this would mean *every* environment.) Which systems you trust should be considered carefully, and even if you do trust a system, be sure to trust it as minimally as possible.

The principle of least privilege should be used on any system that is providing a service out to the Internet. This means don't give people outside of your network access to servers or services they do not need. In regard to VPNs, this means don't run any services that aren't absolutely necessary. Ideally, your machine that provided VPN access should be running only the VPN-related services, thus limiting the services that attackers could try to exploit.

Realistically, you will probably have to be running a service or two for management access to the box. To maintain security, these services should be of the encrypted variety: SSH instead of Telnet, SCP instead of FTP, and so on.

Further, access to these services should be limited to a small range of IPs, at least limited only to your local LAN. This could be done through TCP wrappers or your firewall but preferably should be done on both. A little bit of redundancy can buy you some additional time for damage control if someone manages to break in.

This section so far has discussed trust as it applies to outsiders trying to get in. We will next discuss trust as it applies to site-to-site and end-user VPN connections. You need to apply the principle of least privilege to these connections as well. The intent is

not to mistrust the users on your VPN (although you might) but to minimize the possible business impact should one of the users' machines be compromised.

Limiting access to users can be done with the firewall of your choice on the VPN machine. There are two basic ways to limit access: by IP or by the inbound interface on the VPN server. The latter solution is a little more basic and does not provide as much control over the traffic. Most VPNs provide a local interface that is a logical entryway to the VPN. A firewall rule can be placed on this interface, limiting destination traffic. If you choose to limit by IP, you can restrict entire networks. This is useful if you have provisioned a certain range of IPs for a specific group of users.

For example, if you have given your sales team the IP network of 192.168.6.0/24 and it needs access to only one server, 192.168.1.100, you could use `ipchains/iptables` on your VPN server or firewall to restrict the sales network to access only that server. If you are concerned about security, you could further limit the access by application type. If in this example the sales team only uses that server for POP and SMTP connections, you could limit the access to port 110 and port 25.

Again, you will need to decide how much trust you will need for both your users and your Internet-facing servers. Hopefully, this section has imparted to the reader the importance of limiting trust as much as possible.

Logging

Logging is one of the most powerful techniques you can use to keep a constant eye on your system. In addition to system logs, we recommend that you set up your VPN server to log all connections to your VPN, whether successful or not.

Usually, you can specify different debugging levels to your VPN software when starting it or put them into configuration files. For example, if you use FreeS/WAN, you can use the following options in its `ipsec.conf` to determine the amount of debugging info that is written:

```
[…]
   # Debug-logging
   # See ipsec_klipsdebug(8), ipsec_pluto(8) for details
   klipsdebug=esp
   plutodebug=all
```

We suggest that you replicate your logs on a machine removed from VPN activity. Having the logs from your VPN server stored on a different machine ensures that if the VPN server is compromised, the attackers will not be able to cover their tracks easily by modifying log files.

For example, on our system, we use the following lines in `/etc/syslog.conf` to log all authentication events, including successes and failures. The events are logged both locally and to a dedicated logger machine, Blackhawks (192.168.1.30):

```
auth.debug                      /var/log/messages
auth.debug                      @192.168.1.30
authpriv.debug                  /var/log/secure
authpriv.debug                  @192.68.1.30
```

The `auth` facility is used for authentication events viewable by everyone. `Authpriv` is used for private authentication events, meaning authentication events that might have privileged or sensitive information in them. That is why `authpriv` events are sent to `/var/log/secure`, which is usually set so that only root can view it. The facilities you need to log depend on your VPN software. See `syslogd(1)` man page for more details.

> **Note**
> Although this is a good practice to keep your logs safe, all the remote messages will be stored in a single file on the logging machine. To make things a little easier, you can look into `syslog-ng`. This new implementation of `syslogd` enables you to log to a destination based on pattern matching. Thus, all logs from a specific remote machine can be kept in a separate file. More info can be found at `www.balabit.hu/en/products/syslog-ng/`.

Performance

VPN performance depends on many factors, including the type of VPN package you are using, the operating system you are using, the hardware it runs on, and so forth. This section will discuss techniques you can use to measure server and network performance of your VPN components.

Server Performance

Server performance should be periodically reviewed to make sure the machine is not throttling under the load of the VPN. The easiest way to gather information on how your server is performing is to use tools like `top` and `vmstat`.

The `top` command can be used to give you a real-time estimate of what each process on your system is doing. The following is part of a listing from the output of `top`:

```
  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
  485 root       9   0  3096 2500  2436 S     5.0  4.0   0:00 VPN-PROC
23713 root      18   0  1016 1016   812 R     0.9  1.6   0:00 top
23699 root      10   0  1068 1052   884 S     0.1  1.6   0:00 sshd
    1 root       8   0   160  128   108 S     0.0  0.2   0:04 init
    2 root       9   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root       9   0     0    0     0 SW    0.0  0.0   0:55 kswapd
    4 root       9   0     0    0     0 SW    0.0  0.0   0:00 kreclaimd
    5 root       9   0     0    0     0 SW    0.0  0.0   0:09 bdflush
    6 root       9   0     0    0     0 SW    0.0  0.0   0:03 kupdated
```

Here we can see that the system is relatively idle. The process `VPN-PROC` represents a VPN process; watching it as clients begin connecting to your machine can give you an idea of how many clients your server can handle.

To collect utilization statistics, you can use `cron` to schedule the `vmstat` command to run at regular intervals during the day.

Network Performance

In most cases, VPNs will perform worse than an identical network without a VPN because VPNs must encrypt and decrypt the data. (The decrease in performance can potentially be mitigated if your VPN solution compresses traffic—for example, the IPCOMP protocol in IPSec.) When testing your VPN solution, it is a good idea to see how it performs over time.

Some VPN technologies give you different choices that might affect performance. You might find that one encryption cipher outperforms others on your hardware. In general, the weaker the encryption, the faster it is. However, you might find that some strong ciphers outperform weaker ciphers. (We discuss cipher-related issues in Appendix B, "Selecting a Cipher.")

Over time, your VPN performance will change, perhaps because new users are added, network needs are different, or your hardware starts failing. It would behoove you to have an idea of how your VPN performs now and has performed in the past, before your users start complaining.

To view your current network usage, you can use a number of tools. Our favorite is the Multi Router Traffic Grapher (MRTG). We will also describe how you can use `ttcp` to gather some simple network performance measurements.

MRTG

Before you bring your VPN into production, you should first understand the amount of bandwidth currently in use for user traffic. An excellent lower-level tool for monitoring bandwidth is MRTG.

MRTG is a free tool that produces graphs on your current network traffic in addition to weekly, monthly, and yearly graphs. It uses SNMP to gather utilization statistics from interfaces on servers and routers. MRTG shows only bandwidth utilization; it does not show a breakdown of utilization by traffic type, such as SMTP traffic or web traffic.

Tightening up SNMP Security

Although SNMP on a gateway device can be considered a security risk, it can be useful to have it enabled for monitoring certain types of events. To tighten the security on SNMP, you can take the following steps:

1. Make your community string read-only. This prevents outsiders from making changes to devices even if they have the community string.
2. Create a firewall (or access list) entry, preventing SNMP traffic from entering from the Internet.
3. Make sure to change the default community strings. They are often "public" for the read-only string and "private" for the read-write strings. In SNMP, these strings are treated as passwords and should be changed regularly, just like any other password.

After you have gotten an idea of your normal bandwidth utilization, you should try connecting to your Linux server with a single VPN client. Just as you did when measuring the server performance, you will use this client connection as your baseline bandwidth requirement.

Again, as with server performance, make sure you do this for each type of connection that your server will be initiating. Once the server is in production, MRTG can be used to monitor your VPN interfaces, such as an ipsec, tun, or ppp. This will give you a good idea of your current utilization trends.

ttcp

To get a rough estimate of what kind of overhead the VPN will have on your traffic, you can use the `ttcp` command and compare throughput.

Using VTun as an example, we first measure the normal throughput. We will send some data (in this case, the Linux kernel sources) over and measure its performance. First, we test the connection between the two VPN machines, Bears and Falcons, using their public IP addresses. This tests the actual connection of the machines without any of the VPN overhead. The non-VPN IP addresses we are using are 280.8.8.8 and 270.7.7.7.

```
bears# ./ttcp -t 270.7.7.7 < ../linux-2.4.12.tar.bz2
ttcp-t: buflen=8192, nbuf=2048, align=16384/0, port=5001  tcp  -> 192.168.1.2
ttcp-t: socket
ttcp-t: connect
ttcp-t: 21508430 bytes in 26.00 real seconds = 807.94 KB/sec +++
```

The last line of the output gives us a throughput of 807.94 KB/s. Next, we run the same tests but use the VPN IP addresses this time, 192.168.1.1 and 192.168.2.1. By using these addresses, we will be sending data over the VPN.

```
bears# ./ttcp -t 192.168.2.1 < ../linux-2.4.12.tar.bz2
ttcp-t: buflen=8192, nbuf=2048, align=16384/0, port=5001  tcp  -> 10.0.0.2
ttcp-t: socket
ttcp-t: connect
ttcp-t: 21508430 bytes in 29.59 real seconds = 709.78 KB/sec +++
```

As you can see, the throughput this time is 709.78 KB/s. There is a throughput loss of almost 100 KB/s due to VPN overhead. The numbers you receive in your testing will vary, but you get the idea. You can get `ttcp` from `www.mentortech.com/learn/tools/tools.shtml`.

Summary

One of the essential things we wanted to convey in this chapter is the importance of planning for your VPNs. When building VPNs, we recommend that you always keep in mind fundamental considerations, particularly scalability, interoperability, and key distribution. You also need to have a clear understanding of what topology is the best for

your scenario, how packets are going to be routed, what parts of your network are protected, what are the possible points of failure, and so forth.

Once your VPN has been deployed, it is important to know how to maintain it. In this chapter, we described two common maintenance tasks: performance measurements and logging. In addition, you might want to consider other areas related to maintenance, including upgrades, adding new users, and defining security policies. The rest of this book will focus on the details of various VPN solutions and protocols. Use the knowledge from this chapter with the VPN solutions we describe to effectively deploy and maintain your VPN.