4
Implementation

At this point, you should have a clear operational policy that governs how your users may make use of network services. You should also have good network monitoring tools installed and collecting data, so you can tell at a glance precisely how your network is actually being used. With these major components in place, you are now ready to make changes to the network configuration that bring actual utilisation in line with policy. The ***implementation phase*** of network building closes the feedback loop that allows policy to be consulted upon and revised, and network services to be implemented and changed, based on information provided by the monitoring systems.

This chapter will show you the essential technical components that should be in place in virtually every network connected to the Internet. These techniques will allow you to limit, prioritise, and optimise the flow of information between the Internet and your users.

While these technical constraints are necessary to maintain the health of the network, there is one management technique that is often overlooked, yet nearly always makes the greatest impact on network utilisation: **communication with your users**. If your users don't understand that their actions directly impact the performance of the network, how can they be expected to know that the network is overutilised and not "just broken?" When users are frustrated with network performance, and met with indifference (or outright contempt) by network administrators, they tend to try to find ways around the rules in order to "get their work done." More often than not, these workarounds will consume even more bandwidth, causing problems for users and administrators alike. For example, an enterprising user might discover that they can bypass a slow and badly configured network proxy by using an anonymising proxy server found somewhere on the Internet. While this may show improvement in performance for the user, that traffic is not being cached locally, so bandwidth is wasted. As news of this "fix" spreads among the users, even more bandwidth

is wasted, making the entire network slower for everyone. If the network administrators had been monitoring the performance of the proxy server, or listening to user complaints, then the problem could have been addressed much earlier.

Of course, the vast majority of users are not malicious. Often they simply do not understand the repercussions of their actions, and may be completely caught off-guard when they realise that they are consuming significantly more bandwidth than other users. For example, when network administrators at Carnegie Mellon approached one user about excessive traffic coming from their computer, their response was: "I don't understand--what's bandwidth? How do I reduce it? What am I doing wrong? What's going on?!" (Read the full story in the **Case Studies** chapter, page **235**).

While technical solutions are indeed important, they must go hand-in-hand with education and a responsive network management team. No technical solution can help a network if all of the users insist on consuming as much bandwidth as possible without regard for others. Likewise, it is impossible to effectively manage a network without communicating with your users. You can only provide the best possible service to your users by understanding their needs. The best way to understand those needs, and to make your users appreciate the reality of limited resources, is through clear and honest communication.

The importance of user education

The actions of your users will ultimately determine your bandwidth utilisation. Therefore, your users should be informed of the rights and obligations that govern network use. Having an operational policy does little good if your users are not informed about its scope or details. While this could mean sending out blanket information that is targeted at all users, you may find that you have better results when personally contacting individual users.

But how can you realistically expect to personally speak with every user on a 1000+ node network? Fortunately, it is rarely necessary to give **every** user a personal interview. You can do wonders by simply starting with the biggest "bandwidth hogs" and proceed from there.

The 5/50 rule

More often than not, the majority of users make light use of network resources. They occasionally check their email and browse a few web pages without making excessive use of the network. However, there are always a few users who will use all available bandwidth, both inbound and outbound, by running servers, peer-to-peer file sharing programs, video streams, and other bandwidth intensive applications. Those are the circumstances where the 5/50 rule

comes into play. By focusing on the 5% of users who typically consume 50% of the bandwidth, you can make a huge impact on network performance with less effort.

Targeting the top 5% means identifying and contacting people in order to inform them of their high utilisation. Unless you have a clear idea of who is using your network, there is no way to identify the source of the problem (see chapter three: **Monitoring & Analysis** for ways to reliably identify individual users). Once you have identified problem users, you should inform them that their usage deviates from that which is stated as acceptable by the network access policy.

It may become apparent that such users have legitimate needs differing from those of the typical user, in which case you could make arrangements to specifically support them. Often, users have no idea that they are causing problems (particularly if their computer has been infected by a virus, or if it is running a peer-to-peer file sharing application). Of course, a technical solution can solve the problem by simply denying access to the user once utilisation exceeds a particular threshold. But it is more likely that informing the user of the problem, and showing them how to fix it, will lead that user to take greater responsibility for their own network activity in the future.

Providing feedback to users about network load

If users know there is network congestion, they tend to cut down on their usage, which in turn makes the network experience better for everyone. Without that feedback, users tend to think that the network is "broken" and may inadvertently increase the network load. A classic example of this can be seen when a user becomes impatient with a slowly loading web page and will repeatedly click the reload key. This action submits more network requests, so that the network moves even more slowly, causing the user to continue clicking the reload key, ensuring that the vicious cycle continues. That is, it continues until the user gives up in disgust and complains that the network is "broken."

Here are a few methods you can use to communicate the state of the network to your users.

Public utilisation graphs

One obvious method for showing network congestion to your users is to simply publish your utilisation graphs on a web server. Even when the Internet connection is experiencing heavy use, the monitoring server will be available since it resides on the local network. This will show one and all that the lag is caused by too much network access, and not by a misconfiguration of the servers. By including individual user data, as well as aggregated throughput statistics, you can encourage people to remind their peers that bandwidth is limited. Some

sites even post a "top ten" or "top 100" list of excessive bandwidth users, aggregated daily or weekly. If your name appears on that list, you can be sure that your friends and colleagues will have a few words for you. This method can be even more effective than receiving a phone call from the network administrator.

You can set the utilisation graphs as the default home page (or perhaps make it a single click away) for computers installed in public labs. You might even set up a computer with a continuous display of the current network utilisation and post it in a highly visible place. This helps to serve as a constant reminder for users that their actions have a direct impact on their colleagues' ability to access Internet resources.

Using a captive portal

A ***captive portal*** allows you to "capture" a user's browsing session and redirect them to a web page where they may be asked to perform some task. Wireless hotspots often use captive portals to capture the client's session and redirect them to a site where they can enter their credit card details or other credentials.

A real world example of effective use of a captive portal is the implementation at the University of KwaZulu-Natal. All users at the institution were being authenticated to several Squid proxy servers. The Squid log files were then in turn being imported into a MySQL database. This resulted in a comprehensive database that could be interrogated to derive statistics. Queries run against the database showed that during work hours, up to 20% of the bandwidth was being used by just 20 users out of a total of roughly 12000. That is, 0.2% of the user base was consuming 20% of the bandwidth for the entire university! In addition, the majority of the sites were clearly not of academic content. The university had a policy about bandwidth usage, however its users were either ignoring the policy or simply had not read it. A decision was made that the top 20 users needed to be shown the policy, and if they continued to ignore it that action would be taken against them.

A captive portal mechanism was implemented to notify the top 20 users automatically. Squid has a feature called redirection, which enables you to redirect a user to a web page if they match a specific Squid ***Access Control List*** (***ACL***). Every night a script was run that compiled a list of the top 20 users. These users were then added to a special ACL contained in a text file on the Squid servers. When the user tried to browse the Internet the next morning, they were redirected to a PHP based web page that highlighted the relevant sections of the policy. When they clicked on the OK, button the script would remove them from the ACL and they could continue browsing. If a user was shown this message more than twice in a 14 day period, they were then disabled from browsing the Internet during work hours for a week.

The effect of this over the course of the first month was very interesting. The top 20 users reduced their work hours bandwidth usage from 16-20% to around 6%, but after hours usage increased. Clearly a lot of the "abusers" had now moved their browsing habits to after hours where there was less policing. Did the condition of the Internet line change? No it did not, but what did change is the the number of users and legitimate sites visited. These both showed an increase, which indicated that the latent demand for bandwidth had absorbed the bandwidth freed up by the abusive top users.

If you are keeping detailed logs of network usage, you can interrupt bandwidth hogs before they cause real problems. By using the redirect functionality of Squid (page **184**), it is simple to redirect "errant users" to a page where they are reminded of, and must to agree to, the network access policy. Once they have done this, Squid then allows them to continue browsing. To do this, you will need to set up the following:

- Configure Squid to authenticate and log traffic per user (page **186**)
- Each night, compile list of top users. If any individual user exceeds the "excessive" usage threshold, add this user's name to a Squid ***Access Control List*** (***ACL***) (page **269**).
- The redirect function in Squid will match the ACL the next time the user browses the web, and the redirect page will be shown instead of the requested page.
- When the user clicks the "I Agree" button, they are removed from the ACL and can then browse normally.
- Traffic continues to be logged, and the process begins again.

This is just one example of a creative technical solution that, when combined with social reinforcement, can change users' behaviour.

General good practices

There are several techniques that your users can implement on their own to help keep bandwidth utilisation to a minimum. While there is never a guarantee that users will completely comply with the techniques in this list, making them aware of these techniques will empower them to start making bandwidth management part of their daily routine. These techniques of ***network etiquette*** aren't really rules, so much as guidelines for being a good net neighbor.

Optimise your web browser

Every web browser includes options that will limit bandwidth usage. Some of these options include:

1. **Disable bandwidth-hungry plugins like Java and Flash.** Unless a particular site requires Java for access, it can simply be disabled. Since the vast majority of Java and Flash applications are simple animations, games, and videos, there is rarely any need to download them except for entertainment purposes. Use the HTML version of sites that include both HTML and Flash options. Note that many sites require the use of JavaScript, which is significantly smaller and faster than Java, and can usually be enabled without any noticeable speed penalty.

2. **Disable automatic updates.** While keeping your browser software up-to-date is vitally important from a security point of view, updating in an ad-hoc and automatic way may waste significant bandwidth at peak times. While this can be sped up considerably using a good caching proxy (page **135**) or a local software update repository (page **144**), simply disabling automatic updates immediately reduces background bandwidth usage. You must remember to manually update the software when the network is not busy in order to apply security patches and feature updates.

3. **Increase the size of the local cache.** If there are sufficient resources on the local machine, increase the browser size. More is generally better, but a cache of several hundred megabytes is usually reasonable.

4. **Disable pop-ups.** Pop-up windows are nearly always unwanted advertisements containing large images or flash movies, and will automatically consume significant amounts of unintentionally requested bandwidth. Pop-ups can be disabled in all modern browsers. Well-coded sites that require pop-up windows for functionality will still work, and the user can always allow pop-ups on a case by case basis.

5. **Use ad blocking software.** By blocking ads before you download them, you can save bandwidth and reduce user frustration. Free and commercial ad blocking packages are available for every browser. For Mozilla Firefox, try AdBlock Plus: *https://addons.mozilla.org/firefox/1865/*

6. **Install anti-spyware tools.** Malicious sites may install spyware programs that consume bandwidth and introduce security problems. These attacks nearly always come through the web browser. Using a spyware detection and removal tool (such as Lavasoft AdAware) will keep these problems to a minimum. *http://www.lavasoftusa.com/software/adaware/*

7. **Disable images.** For many kinds of online work, graphical images may not be required. Since graphic files are considerably larger than HTML code, disabling the display of images (even temporarily) can significantly improve response time and reduce bandwidth use. If possible, configure your browser to only display graphics when explicitly requested.

8. **Use Mozilla Firefox instead of Internet Explorer.** Although it is the default browser on most Windows boxes, IE is a notorious attack point for spyware and viruses, and is widely considered obsolete since the release

of Mozilla Firefox (*http://www.mozilla.com/firefox/*). Since it is an open source project, Mozilla Firefox has a very large and flexible set of extensions that allow you to configure and optimise just about every aspect of how the browser works. One popular extension is FireTune (*http://www.totalidea.com/content/firetune/firetune-index.html*), which groups many common optimisation options into a simple, easy to understand menu. Other extensions provide excellent content filtering, presentation, and download optimisation features.

Of course, the most effective bandwidth optimisation tool is simply refraining from requesting information that you don't need. Ask yourself if it's really appropriate to try to stream videos from YouTube during peak network times (even if the video is really funny...). The more you request from a busy network, the longer everyone will have to wait for their requests to be filled. Be considerate of your fellow network users, and your network will be healthier and faster.

Optimise your email

Web browsing and email are the most commonly used services on the Internet. Just as you can optimise your web browser to minimise bandwidth usage, there are many steps you can take as a user to make your email work better too.

1. **Don't use web mail.** Sites such as Hotmail, Gmail, and Yahoo! Mail use significantly more bandwidth than do traditional email services. With graphics, advertisements, and an HTML interface, an individual email may represent thousands of times the number of bytes of the equivalent text email. If your network provides email services (via Mozilla Thunderbird, MS Outlook, or another email client), then use them. If email service is not provided for you, ask your network administrator if it makes sense to set it up for your organisation.

   Some web mail services, such as Gmail and Fastmail, allow access via POP3 and IMAP from a standard mail program. This is much faster and more efficient than using their web interfaces.

2. **Send emails in plain text, not HTML.** HTML emails are bigger than their equivalents in plain text, so it is preferable to send plain text emails to reduce bandwidth requirements. Most email clients let you set plain text as the default format, and/or on a per email basis. As well as reducing the amount of bandwidth needed, you'll also benefit from your email being less likely to be treated as spam by the recipient.

3. **Limit the use of attachments.** While it is possible to send colossally huge files through email, the protocol wasn't really designed for this use. Instead of sending a large file to a group of people, post the file to a web server

and send the link instead. This will not only speed up your own email, but it will save significant bandwidth for everyone receiving the message. If you **must** include an attachment, make it as small as possible. You can use a compression utility to reduce the size of the file (WinZip is one popular commercial tool, but many free alternatives exist. One list of WinZip alternatives is *http://free-backup.info/free-winzip.html*). If the attachment is a photograph, reduce the image size to something reasonable, such as 1024x768 or 800x600. This size is usually fine for on-screen viewing, and is significantly smaller than a raw 3+ Megapixel image.

In addition, websites exist which allow files to be uploaded and included in the email as a link. This allows recipients to choose whether or not to download the file. One such service is Dropload, *http://www.dropload.com/*.

4. **Filter your email on the server, not the client.** If your network admin provides email services, they should also provide spam and virus filtering. While it is possible to do this kind of filtering within your mail client, it is far more efficient to do it on the server side (since junk messages can be discarded before they are downloaded). If your network offers filtering services, be sure these are enabled on the server.

5. **Use IMAP instead of POP.** The IMAP protocol makes much more efficient use of network bandwidth by using a caching system that only downloads messages when needed. The POP protocol downloads all email messages as they are received, whether they are opened or not. If possible, use IMAP instead of POP. Your network administrator should be able to tell you if an IMAP server is available. This is especially critical when accessing your email over a low-bandwidth link.

6. **Don't send spam.** Junk email (commonly referred to as ***spam***) comes in many guises. While we are all familiar with the "get rich quick" and "improve your love life" variety, other kinds are more subtle. One example of this is the so-called "boycott" email, where someone sends you a petition for a popular cause (such as lowering gasoline prices or changing an unjust law). You are instructed to "sign" the petition with your email address, and send it to six of your friends. These messages are completely fabricated by spammers who then use the collected email addresses as targets for more spam. These kinds of scams are easily avoided if you ask yourself, "what will really happen if I send this message with my email address to total strangers?" The answer: wasted bandwidth, with the promise of more wasted bandwidth to follow.

Use of collaborative tools vs. Word attachments

One popular phenomenon among inexperienced users is the proliferation of Microsoft Word file attachments in normal email. While programs like Word and OpenOffice are excellent choices for desktop publishing, they are nearly always a waste of bandwidth and effort when used for normal communications.

Consider the case of a group of ten people collaborating on a research project. If the group uses a large cc: list of email addresses, and includes Word attachments in the development process, they are inevitably going to run into several problems:

- Each message now takes up ten, one hundred, or even thousands of times the storage space of a simple email. This makes everyone's email take longer to upload and download, and wastes storage space on the server and everyone's personal computer.
- The work flow of each user is interrupted as they must now download the attachment and open it in a separate program. They must download every attachment sent to the group, even if they do not need it for their work.
- Every user needs to have the same version of the program installed (in this case, Word) or they will not be able to read the message. By contrast, plain email can be read by hundreds of different email clients.
- Maintaining a consistent set of revisions is difficult or impossible, since users tend to work asynchronously. Some may reply with revisions in email, while others may respond with new Word documents. Someone on the team will need to make sense of all of these revisions and maintain a single authoritative copy of the work. This problem is known in project management circles as ***versionitis***.
- If the list of team members is maintained by hand, it is easy for people to be accidentally (or intentionally) removed from the list, resulting in general confusion.
- If any user has problems with their own computer (e.g., it is lost, damaged, stolen, or infected by viruses), then any work they have not yet sent to the list is permanently lost.

There is simply no need to indulge in these bad habits! You can be more effective and save bandwidth by using collaborative tools designed to make group projects easy. Most organisations provide mailing list services free to their users. This makes management of the group discussion much simpler than trying to maintain a list by hand. Mailing list software will often provide restrictions on the type and size of file attachments, discouraging this bad habit. If the discussion group is very large, web-based forum software can be implemented that will allow any number of participants.

For project development, one very popular collaborative tool is the ***wiki***. A wiki is a web site that allows any user to edit the contents of any page. The wiki can be protected by a simple password, an access control list (based on IP address or other identifying information), or simply left open for public access. All changes to the wiki are recorded, allowing users to view all revisions made to the work. This effectively cures the revision control problem, provides excellent

backups for the work, and makes efficient use of available bandwidth. Attachments (such as graphics, slideshows, and even Word documents) can be uploaded to any page and downloaded on demand. The resulting work can be easily searched and linked to, just as with any web page. Users can be notified of changes to the wiki via email or using ***Really Simple Syndication*** (***RSS***).

This book was developed using a wiki in conjunction with a mailing list. Trying to manage a team of a dozen people scattered all over the world by using attachments in email would have been nearly impossible, and would certainly have been a waste of bandwidth.

If you are a user, ask your network administrator what tools are available for collaborative work. If you are a network admin, see the list of resources at the end of this chapter for many freely available software packages you can install on your own network.

One last word about attachments

Consider what happens when you send a file attachment to a large group of people. The flow of information may look something like Figure 4.1:

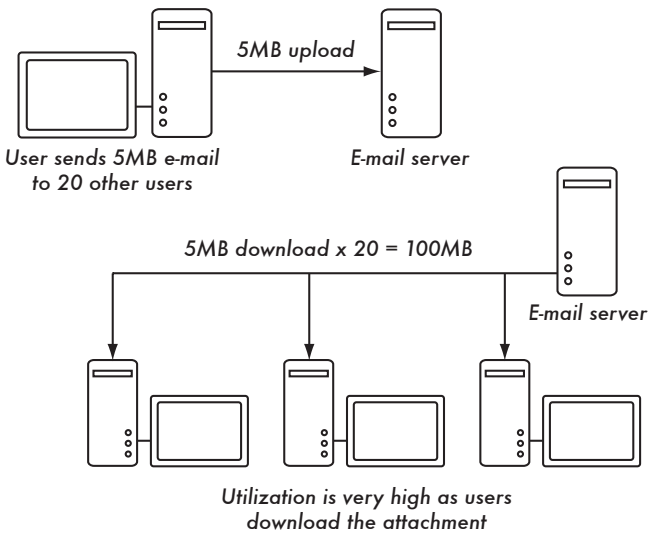

Figure 4.1: A single unnecessary email attachment can quickly add up to hundreds of megabytes of wasted bandwidth as users forward it to each other.

A single 5 Megabyte attachment can easily blossom to several hundred megabytes of wasted information as it is forwarded between people on the network. If the original sender had sent a hyperlink instead of the original file, considerable bandwidth would have been saved, while people could still easily view the

file when needed. Your users should consider this the next time they want to send an attachment to their friends or colleagues.

Download managers & peer-to-peer clients

The word "client" is not exactly accurate when applied to ***peer-to-peer*** programs (such as BitTorrent, Gnutella, KaZaA, and eDonkey2000). These programs work by turning every computer into both a client and a server, where users exchange information directly with each other. While this can be a tremendous boon to publishers who use these protocols to avoid high bandwidth costs, they can be a nightmare for network administrators. These programs often maximise both inbound and outbound bandwidth, reducing the performance of other network services to a crawl. They will often aggressively attempt to circumvent firewall restrictions, and can even disguise some of their protocol information in order to avoid filtering. These programs continue to utilise outbound bandwidth even when the user has finished downloading, serving copies of the retrieved data to the Internet at large.

So-called ***download managers*** can include peer-to-peer technology (sometimes referred to as ***swarming***) in order to "speed up" user downloads. Examples of some of these types of download managers include FlashGet, GetRight, Download Accelerator Plus, LimeWire Download Manager, DownloadThemAll (DTA), and GetBot.

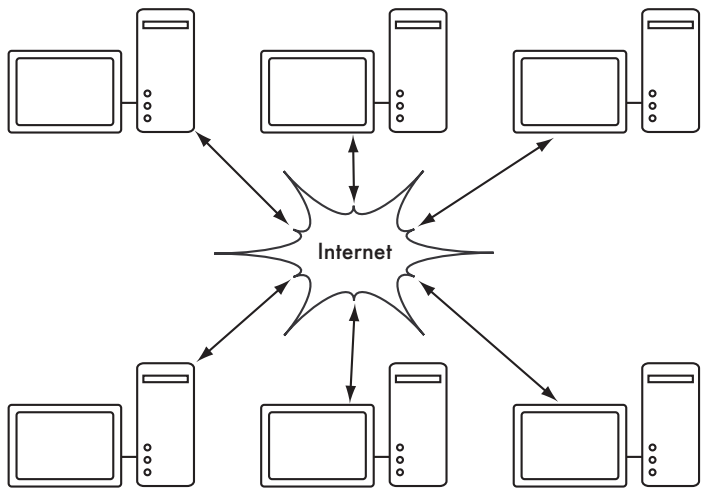

Figure 4.2: Peer-to-peer programs work by turning every node into a server that uploads data to the others. This can be a boon on very fast and cheap networks, but it is disastrous on slow and expensive connections such as VSAT.

The lure of these programs can be great for your users, since they promise fast downloads of music, movies, and other data. Your operational policy should be

very clear about the use of peer-to-peer applications and download managers, with regard to both legitimate and potentially infringing uses.

Users are often unaware that these programs introduce a heavy toll on a network. They may not realise that their download manager is consuming resources even when they are not using it. While many of these services can be blocked or limited at the firewall, the best policy is simply not to use them on low-bandwidth links. If they must be used, then use them to schedule large downloads when the network is less busy, which can improve network performance for everyone. Make sure your users understand the impact that these programs will have on the network.

Essential services

Now that your users have a clear understanding of your usage policy, as well as an idea of general good practices, you can implement hard restrictions on the flow of network traffic. The primary method for making your Internet connection more responsive is to ensure that network usage never quite reaches the available capacity. However large or complex your network installation, you will likely make use of a few essential kinds of technology to ensure that your Internet connection is used in an appropriate and efficient manner.

The most basic class of tool in the bandwidth management toolkit is the ***firewall***. A firewall can be thought of as a filter for network traffic. By making good choices in your firewall configuration, you can allow legitimate traffic through, and drop traffic destined for inappropriate services before it is transmitted to your ISP. This allows the available bandwidth to be used exclusively for its intended purpose. For example, by blocking access to peer-to-peer services, considerable bandwidth can be saved, making email and the web servers more responsive.

Firewalls can also add considerable security to your network by preventing access to machines on your private network from the Internet. It is generally accepted as standard practice to have at least one firewall in place if you have a connection to the Internet.

Caches exist everywhere in the computing field. The term cache means literally, to store. Most people are familiar with the web browser cache, which keeps a local copy of content retrieved from the Internet. This local copy can be retrieved and reused faster than making the same request from the Internet. By using a cache, images and web pages can be displayed much more quickly without making use of the Internet connection. Of course, information stored in the cache may or may not be used again, so caches are only beneficial when the cost of storing the information is less than the cost of retrieving the informa-

tion again. This cost is measured both in system resources (disk space and RAM) and in time.

In addition to the cache available in your web browser, caching services can be set up to be used by an entire network. Using a ***site-wide web cache*** can save considerable bandwidth, since one user's visit to a popular site (such as *www.yahoo.com*) will cause a local copy to be saved and automatically served to other users who visit the same site. Other network services, such as DNS lookups, may also be cached, considerably improving the "feel" of a connection while saving Internet bandwidth for other uses.

Caches are typically implemented through use of a ***proxy server***. This is a kind of buffer or middleman between a computer and the Internet resources it requests. A client makes a request of the proxy server, which then contacts the web server (or other server) on its behalf. The response is sent back to the client as if it had come from the original server. A ***socks proxy*** is a typical example of this kind of server. Proxies can provide a controlled and secure way to access resources that lie outside your firewall by requiring authentication from the client. While proxies typically also include caching services, this is not required. You may wish to make use of a proxy server to provide access control and an audit trail to monitor your users' usage patterns.

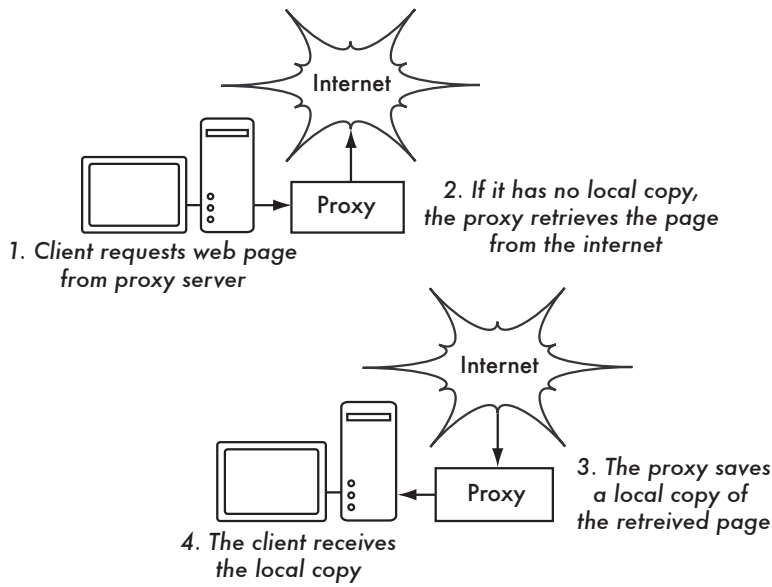

Figure 4.3: Proxy servers make requests on behalf of a client program. They may also implement authentication or caching.

Mirrors can be thought of as a kind of manually updated caching server. Whole copies of popular websites and data stores can be stored locally, and updated when network utilisation is low (after working hours, or in the middle of

the night). Users can then use the local mirror rather than requesting information directly from the Internet. This can save considerable bandwidth, and is much more responsive from a user's point of view, particularly at peak utilisation times. However, users must be aware of the existence of the mirror, and know when to use it, otherwise it will waste bandwidth by downloading information that is never used.

Email can become one of the most heavily abused services on a network, even though legitimate email service itself may use relatively little bandwidth. Unless your email servers are properly configured, they may allow unauthenticated users to send messages to any destination. This is referred to as an ***open relay***, and such servers are often abused by spammers to send huge numbers of messages to the Internet at large. In addition, ***spam*** and ***viruses*** can clog your legitimate email services unless you employ an effective form of ***content filtering***. If your mail service is slow, or unreasonably difficult to use, then your users might turn to ***webmail*** services (such as Hotmail, Yahoo mail, or Gmail) and waste even more bandwidth. If you offer email services to your users, then these services must be properly configured to make the best possible use of your Internet connection. Spam and viruses should be filtered before they cross your Internet line (page **174**), open relays should be closed (page **166**), and web mail services should be avoided.

By implementing these essential services (firewall, caching, mirrors, and proper email configuration) you will make a significant impact on your bandwidth utilisation. These basic services should be considered mandatory for any network connected to the Internet. For more advanced topics (such as bandwidth shaping, fairness queueing, and protocol tweaks) see chapter six, **Performance Tuning**.

Firewall

The word ***firewall*** refers to a physical barrier in a building or vehicle designed to limit damage in the event of a fire. It prevents fire on one side of the wall from spreading to the other. In a car, the firewall is generally a solid plate that seals off the fuel tank or engine compartment from the passenger compartment. In buildings, the firewall may be made of concrete or metal, and it seals off different sections of the building. This provides protection from fire and possibly the total collapse of the structure.

The logical network firewall functions in a similar manner. Although, instead of protecting your network against fire, it protects against undesirable traffic. For example, it may deny access to peer-to-peer file sharing services on the Internet, or to prevent unauthorised connections to servers inside your organisation. The firewall can filter both inbound and outbound traffic. In order to do

this most effectively, the firewall needs to be located at the border where your network meets the Internet.

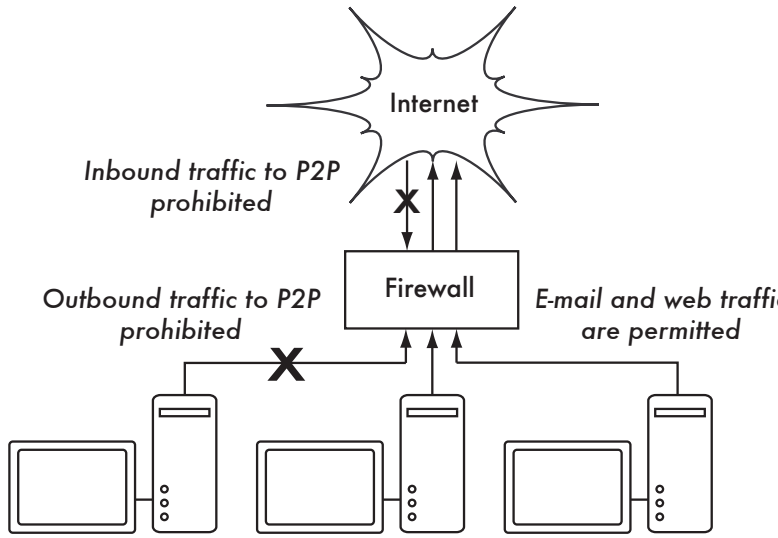

Figure 4.4: Firewalls are most effective at the border between your network and the external world (e.g. the Internet).

Users may also choose to implement their own ***personal firewall***. This provides a "last line of defense" for an individual computer by blocking access to all (or most) services. Linux, BSD, Windows XP, and Mac OS X all have built-in firewall support. There are also a number of third party firewall packages available for Windows; ***ZoneAlarm*** and ***Norton Firewall*** are two popular commercial packages.

A personal firewall is a good idea if properly implemented, but for most users it may seem like an inconvenience because some services (such as Voice-over-IP) will not work properly unless the software is configured to allow such access. Since users often disable their firewall when they are having trouble, it is not a good idea to rely exclusively on personal firewalls to protect your network.

Firewalls can make filtering decisions based on any of the network layers from two and above (see chapter three: **Monitoring & Analysis** for a discussion of layered network models) but traditionally are grouped into two classes. ***Packet filters*** operate at the Internet layer by inspecting source and destination IP addresses, port numbers, and protocols.

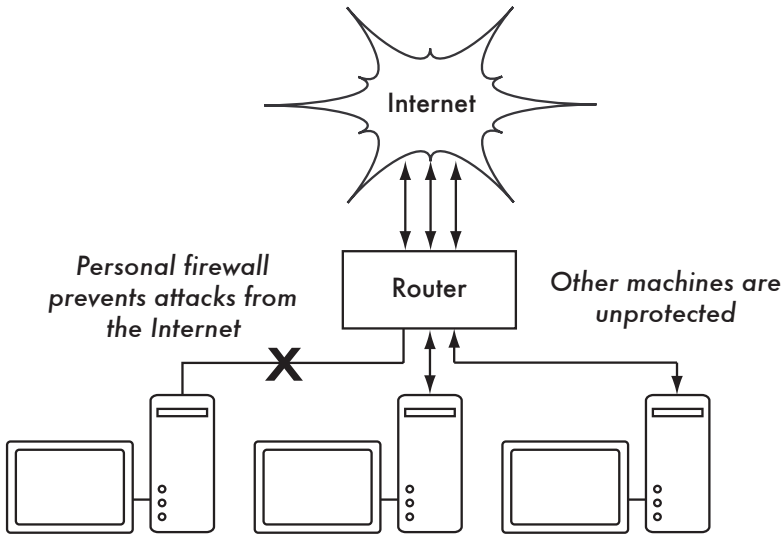

Figure 4.5: Personal firewalls can provide a small measure of protection, but should not be relied upon to protect a large organisation.

Application firewalls operate at the top layer, and make filtering decisions based on the application protocol being spoken. They tend to be more flexible than simple packet filters, but they tend to require more in the way of system resources. A packet filter may block all communications on port 80, while an application firewall can block HTTP traffic on any port. One example of an application firewall is L7-filter, *http://l7-filter.sourceforge.net/*.

Access philosophy

There is a very old joke floating around the Internet that goes something like this:

> *In France, everything is permitted, except what is explicitly forbidden.*
> *In Germany, everything is forbidden, except what is explicitly permitted.*
> *In Russia, everything is forbidden, including what is explicitly permitted.*
> *In Italy, everything is permitted,* ***especially*** *what is explicitly forbidden.*

When building firewalls, you may choose to implement either of the first two models as a general policy: the "French" model (everything that is not expressly forbidden is permitted) or the "German" model (everything that is not expressly permitted is forbidden). While the first approach may seem easier from a network administrator's point of view, it is far less secure, and can be more difficult to maintain over the long term. It is much safer to err on the side of denying traffic first, and make exceptions for legitimate traffic as the need arises. If you are already monitoring your network extensively (as detailed in

chapter three), then you should have a good idea of which services your users need to access.

For a solid network firewall, you will need to implement the following four rules as a standard policy:

1. Allow already established and related connection traffic.
2. Allow TCP/IP SYN packets to the services you wish to permit.
3. Allow UDP packets to the services you wish to permit.
4. Deny ALL other traffic, and optionally log denied traffic to disk.

This configuration works well for the vast majority of networks connected to the Internet. If your organisation requires support for other protocols as well (such as GRE, which is required for VPN services such as PPTP), you can add those exceptions just before step four. You may consider logging denied traffic to disk in order to debug problems and detect attempts to circumvent your firewall. But this can quickly fill up your disk on a very busy network or in the event of a denial of service attack. Depending on your circumstances, you may wish to only enable logging when you need to debug firewall problems.

Here are some examples of how to set up a good default firewall in Linux and BSD.

Building a firewall in Linux

The firewall implemented in modern Linux kernels is called ***netfilter***. Netfilter is extremely powerful, flexible and complex, and a whole book could easily be written on netfilter alone. We will only cover the very basics here. Full documentation is available at *http://www.netfilter.org/*.

Netfilter consists of kernel code that filters packets, and userspace programs to control the kernel code. The interface that most people are familiar with is the **`iptables`** command, which allows you to list and change the firewall rules from the command line.

Netfilter rules are divided into sections called ***tables***. The default table is the ***filter*** table, but there are additional tables used for Network Address Translation and other purposes. Each table contains a number of processing phases, called ***chains***. Each chain in turn contains ***rules*** that determine the fate of any packet that enters the chain. Each table and chain is used during a different phase of the filtering process, allowing for very flexible packet matching.

There are three chains defined by the **filter** table:

- The **INPUT** chain is used for every packet destined for the firewall itself. For example, packets bound for a web server or SSH server running on the firewall itself must first traverse the INPUT chain.
- The **OUTPUT** chain is used for each packet generated by the firewall itself. For example, web requests or SSH connections made from the firewall itself first pass through the OUTPUT chain.
- The **FORWARD** chain is read for each packet passing through the firewall that was not generated the firewall itself, or destined for it. This is where the majority of filtering rules are inserted on firewall machines.

The **nat** table (for Network Address Translation) defines these chains:

- The **PREROUTING** chain is read for each packet passing through the firewall, not generated by or destined for it.
- The **OUTPUT** chain is used for packets generated by the firewall itself, but is executed before the OUTPUT stage of the filter table.
- The **POSTROUTING** chain is read for each packet passing through the firewall, not generated by or destined for it.

Figure 4.6 shows the path that a packet takes as it passes through the netfilter system when using NAT.

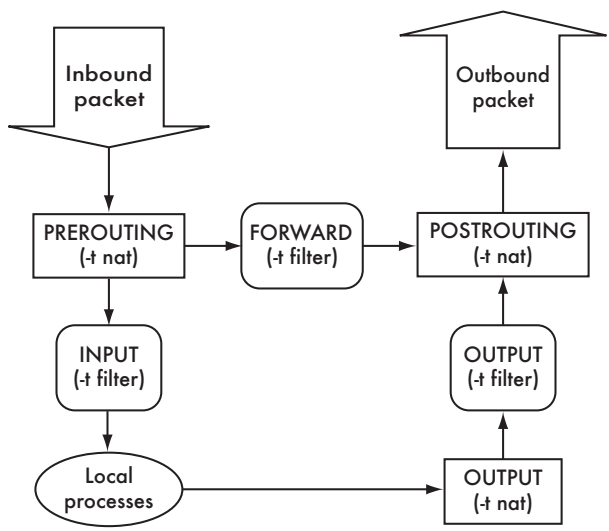

Figure 4.6: The netfilter process is applied to packets according to this schematic.

Each chain contains a number of rules. The kernel starts reading from the top of the chain, beginning with the first rule. It checks the conditions on each rule

in turn, and if they match the packet, it executes the associated ***target*** on that packet.

Some targets cause the packet processing to finish immediately. These targets imply that a final decision has been made on what to do with the packet, such as whether to accept (allow) or drop (discard) it. The ***ACCEPT***, ***DROP***, and ***REJECT*** targets are examples of such terminating targets. Other targets cause a side effect, but allow processing to continue. For example, the ***LOG*** target writes some information about the packet to the system logs, but the packet will continue to pass down the chain until it reaches a rule with a terminating target.

On most systems, support for netfilter is not built into the kernel, and must be loaded as a kernel module. You should at least load the **`ip_tables`**, **`iptable_filter`**, and **`ipt_state`** modules, as well as any other advanced features you may want to use (such as **`iptable_nat`** and **`ipt_MASQUERADE`** for NAT). Because these commands affect the system's security, only the root user can run them. The following commands will load the basic modules:

```
# modprobe ip_tables
# modprobe iptable_filter
# modprobe ipt_state
```

To load these at boot time, add the appropriate lines to **`/etc/modules`**.

Before configuring a firewall, you should make sure that the kernel is allowed to act as a router, and will forward packets between interfaces. This is disabled by default for safety. You can enable it temporarily with the following command:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

This setting will be lost when the system is rebooted. Most systems have an **`/etc/sysctl.conf`** file which defines default values for kernel variables like this. You can make the change permanent by adding the following line:

```
net.ipv4.ip_forward = 1
```

and remove any existing line that refers to **`net.ipv4.ip_forward`**.

The normal state of each chain is empty, but some Linux distributions configure a simple firewall automatically during installation. You can list the rules in each chain with the following **`iptables`** commands:

```
# iptables -L INPUT -n
# iptables -L FORWARD -n
# iptables -L OUTPUT -n
```

Each rule has a ***match condition***, which specifies the packets that match the rule, and a ***target***, which is activated for each packet that matches the condi-

tions. For example, **`-p tcp -j DROP`** matches TCP packets and discards them.

Until you are more familiar with netfilter, you should only configure your firewall from the physical console of the machine, and never over the network. It is very easy to accidentally block your own access to the machine when configuring netfilter remotely.

The chains in the filter table, INPUT, OUTPUT, and FORWARD, are required to make a final decision for every packet that passes through the system. Therefore, they have a ***policy*** which is applied to each packet that does not match any rule in the chain. This can simply be thought of as the default target for the entire chain. The policy must be set to ACCEPT or DROP, and the default is ACCEPT. It is normally considered that a default ACCEPT policy is not secure, and that you should change the default policy to be DROP. This is known as ***deny by default***, and fits with the "German" network security model. The following commands change the default policy for each chain to DROP:

```
# iptables -P INPUT DROP
# iptables -P OUTPUT DROP
# iptables -P FORWARD DROP
```

To create your own firewall, you should delete any existing rules in the filter chain, using the following command:

```
# iptables -F
```

The filter table is used so often that it is implied when no explicit table is used. This is functionally equivalent to the command:

```
# iptables -t filter -F
```

...but is shorter to type. The **`-F`** option stands for ***flush***. You can also Flush individual chains:

```
# iptables -F INPUT
# iptables -F OUTPUT
# iptables -F FORWARD
```

Now all traffic is blocked on the firewall, regardless of its source or destination. To allow packets to pass through the firewall, you need to add rules which match them using the ACCEPT target. The **`iptables -A`** command appends a new rule to the end of a chain. With a deny by default policy, it is common practice to allow packets that are part of, or associated with, an established connection. Since the connection was established in the first place, we assume that subsequent packets in the same connection are also allowed.

The following commands allow such traffic into and through the firewall:

```
# iptables -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
```

With established connections permitted, you can now specify the new TCP connections that should be permitted into the firewall. These should correspond to services running on the firewall itself that should be permitted. In this case, we will allow access to SSH (port 22), HTTP (port 80), SMTP (port 25), and TCP DNS (port 53) services on the firewall. Note that no new connections may be made through the firewall yet, as these rules only open connections for services on the firewall itself.

```
# iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
# iptables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
# iptables -A INPUT -p tcp --dport 25 -m state --state NEW -j ACCEPT
# iptables -A INPUT -p tcp --dport 53 -m state --state NEW -j ACCEPT
```

These rules each have multiple conditions that must be met in order to be accepted. For example, the first rule matches only if:

- The protocol of the packet is TCP, **and**
- The TCP destination port is 22 (SSH), **and**
- The state is NEW.

When a rule contains multiple conditions, they must all match before the target will be executed.

The rules above also make use of netfilter's ***stateful inspection*** features, also known as ***connection tracking***. The rules match on the state associated with the packet. The state is not part of the packet as transmitted over the Internet. Instead, they are determined by the firewall itself, after comparing the packet with each connection that it knows about:

- If the packet is part of an existing connection, its state is ESTABLISHED; otherwise:
- If the packet is related to an existing connection, such as an ICMP destination unreachable reply to a TCP packet sent earlier, its state is RELATED; otherwise:
- Its state is NEW.

Every NEW or RELATED packet adds a temporary entry to the connection tracking table, and every ESTABLISHED packet extends the life of the entry. If a connection is established through the firewall, but it sees no activity for some

period of time, the entry in the table will expire, and further packets will be regarded as NEW again, rather than ESTABLISHED.

You cannot use the **--sport** and **--dport** options (which match the source and destination ports) unless you specify the protocol with **-p tcp** or **-p udp**. It is not possible to write a single rule that matches both TCP and UDP packets and specifies port numbers.

Some types of UDP traffic are very useful. For example, DNS requests are normally made over UDP rather than TCP. If the firewall runs a DNS server, then you will probably want to allow UDP port 53 traffic into it, with the following command:

```
# iptables -A INPUT -p udp --dport 53 -j ACCEPT
```

If your users use a DNS server out on the Internet, then you will need to give them access to it. Replace 10.20.30.40 with the DNS server's IP address in the following command:

```
# iptables -A FORWARD -p udp --dport 53 -d 10.20.30.40 -j ACCEPT
```

If you use globally routable IP addresses throughout your organisation, you may want to allow your users to have direct access to the Internet. This can be enabled with this command:

```
# iptables -A FORWARD -j ACCEPT
```

Alternatively, you may wish to force them to use a proxy to access the Internet. The proxy server should be the only computer that's allowed to make connections out onto the Internet. If the proxy server runs on the firewall itself, then the firewall must be allowed to make outgoing connections:

```
# iptables -A OUTPUT -j ACCEPT
```

If the proxy server is another computer inside your network, you will need a rule like this, replacing 10.10.10.20 with the proxy server's IP address:

```
# iptables -A FORWARD -s 10.10.10.20 -j ACCEPT
```

The options **-s** and **-d** in the commands above match source and destination addresses respectively. You can specify a range of addresses with a network mask, for example **-s 10.10.50.0/24** matches all source addresses from 10.10.50.0 to 10.10.50.255.

You can make your rules more explicit by specifying interfaces. The **-i** option matches the incoming interface, and **-o** matches the outgoing interface. You cannot specify **-o** for rules in the INPUT chain, because packets passing

through INPUT will not leave the firewall, so they have no outbound interface. Similarly, you cannot specify **-i** for rules in the OUTPUT chain, but you can specify either or both for rules in the FORWARD chain. For example, if you want to allow traffic from 10.10.10.30 through the firewall, but only if it comes in on the eth0 interface and leaves on eth1:

```
# iptables -A FORWARD -s 10.10.10.30 -i eth0 -o eth1 -j ACCEPT
```

You may wish to allow ICMP packets in both directions, to allow the **ping** and **traceroute** commands to work to and from the firewall itself:

```
# iptables -A INPUT -p icmp -j ACCEPT
# iptables -A OUTPUT -p icmp -j ACCEPT
```

And you may wish to allow users on your inside network (e.g. eth0) to run **ping** and **traceroute** to the outside world (e.g. eth1) as well:

```
# iptables -A FORWARD -i eth0 -o eth1 -p icmp -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -p icmp -j ACCEPT
```

All the **iptables** commands above start with the **-A** option, which appends the specified rule to the end of the specified chain. Other useful options are:

- **-L <chain>** lists the rules in the specified chain, with the following useful options:

  -n stops the iptables command from trying to resolve IP addresses to hostnames. If running **iptables -L** seems to hang, you may be waiting for DNS resolution. Add **-n** to see the list immediately using just IP addresses.

  -v lists all details of the rules, including input and output interfaces and byte and packet counters

  --line-numbers gives the rule number next to each rule, which is useful for the **-D** and **-I** options
- **-D <chain> <rule>** deletes the first rule matching the specification from the specified chain
- **-D <chain> <number>** deletes the specified rule number from the specified chain
- **-I <chain> <number> <rule>** inserts the specified rule before the specified rule number in the chain

The packet and byte counters, shown with **iptables -L -v**, are very useful for debugging and optimising rules. The packet counter is increased by one every time a packet matches the rule. If you create a rule and it doesn't seem to be working, check the packet counter to see whether it is being matched. If

not, then either the specification is wrong, or an earlier rule is capturing the packets that you wanted to match. You can try moving the rule higher up the chain (delete and re-insert it), or removing conditions until it starts to match.

If your firewall has very heavy traffic, you should optimise your rules so that the ones with the highest packet counters are higher up the chains. In other words, the rules should be in descending order by packet count. However, you should be careful not to violate your security policy by doing so. Rules which might match the same packets under any circumstances should not be reversed in order without careful thought.

It is also a good idea to add a LOG rule at the end of each chain, so that you can see from your firewall logs (**`/var/log/messages`**) which packets are not matching any rules. This can be useful for debugging and for spotting attacks on your firewall and network. You may wish to limit the rate of logging with the limit match, to avoid filling up your logs too fast:

```
# iptables -A INPUT -m limit --limit 10/min -j LOG
```

Other useful targets include ***REJECT***, which is like DROP in that the packet is not allowed to pass through the firewall. However, while DROP is silent, REJECT sends an ICMP destination unreachable message back to the originator. REJECT is more polite, because it tells the sender what happened to their packet, but DROP is more secure because it provides an attacker with less information and makes it much slower to scan your machine for open ports. You may want to consider dropping ICMP echo-request packets from outside your network for security reasons:

```
# iptables -A INPUT -p icmp --icmp-type echo-request -j DROP
```

You should ensure that this rule comes before any rule which allows ICMP packets in the INPUT chain (otherwise it will have no effect).

You can use NAT to rewrite the source address of packets forwarded by your firewall, to make them appear to have come from the firewall itself. This is called ***masquerading***, and is very useful if your internal network uses a private IP address range. For example, if your internal network uses 192.168.1.0/24, and the external interface is eth1, then you could use the following command:

```
# iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth1 -j MASQUERADE
```

Once you are finished adding exceptions, test each of the rules from inside and outside your network. You can make connections manually, or use a security scanning tool such as ***nmap*** (*http://insecure.org/nmap/*).

The ruleset will be lost when the system is rebooted unless you save it first. On Debian (and Ubuntu) systems, you will need to install the iptables **initscript**:

```
# zcat /usr/share/doc/iptables/examples/oldinitscript.gz > /etc/init.d/iptables
# chmod a+x /etc/init.d/iptables
# ln -s /etc/init.d/iptables /etc/rcS.d/S39iptables
```

No other distribution makes it this difficult to use iptables. Normally you just have to make sure that iptables is started at boot:

```
# chkconfig iptables on
```

Whenever you update your ruleset and want to save your changes, use the following command on Debian/Ubuntu:

```
# /etc/init.d/iptables save active
```

And on other distributions:

```
# /etc/init.d/iptables save
```

You can discard the current ruleset and reload the last saved one with:

```
# /etc/init.d/iptables restart
```

You can find more about netfilter from the following resources:

- iptables manual page. EIther run **man iptables** or see:
  http://www.linuxguruz.com/iptables/howto/maniptables.html
- Packet Filtering HOWTO:
  http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO-7.html
- Network Address Translation HOWTO :
  http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html
- Netfilter documentation: *http://www.netfilter.org/documentation/*

BWM Tools

BWM Tools (*http://freshmeat.net/projects/bwmtools/*) is a Linux utility which provides a wrapper to **iptables**, allowing all of its features to be configured using an easy XML syntax. Firewalling, bandwidth shaping, logging, and graphing are all supported.

BWM Tools consists of two main utilities. **bwm_firewall** is used for building the firewall rules, and **bwmd** is used for traffic shaping. BWM Tools works by queueing packets in userspace, where they are inspected, queued, rate limited, and given the go-ahead to pass. Live bandwidth graphs can be generated from

any flow by using RRDtool file formats along with **`rrd-cgi`**, which provides for a good overall health check on networks.

More information about configuring and installing BWM Tools can be found at *http://bwm-tools.pr.linuxrulz.org/doc/*.

Below is a simple stateful firewalling example that allows SSH, SMTP, and DNS traffic to be received.

```
<firewall>
  # Global configuration and access classes
  <global>
    <modules>
      # Track FTP connections
      <load name="ip_nat_ftp"/>
      <load name="ip_conntrack_ftp"/>
    </modules>

    # Setup traffic classification classes
    <class name="ssh">
      <address proto="tcp" dport="22" />
    </class>

    <class name="smtp">
      <address proto="tcp" dport="25"/>
    </class>

    <class name="http">
      <address proto="tcp" dst-port="80"/>
    </class>

    <class name="dns_tcp">
      <address proto="tcp" dst-port="53"/>
    </class>

    <class name="dns_udp">
      <address proto="udp" dst-port="53"/>
    </class>

    <class name="new_connections">
      <address proto="tcp" cmd-line="--syn -m state --state NEW"/>
    </class>

    <class name="valid_traffic">
      <address cmd-line="-m state --state ESTABLISHED,RELATED"/>
    </class>
  </global>

  # Access list
  <acl>
    # Filter table
    <table name="filter">
      # Allow only SYN packets here...
```

```
      <chain name="new_connections">
        <rule target="ACCEPT">
          ssh;
          smtp;
          http;
          dns_tcp;
        </rule>
      </chain>

      # Deny-all default policy
      <chain name="INPUT" default="DROP">
        # Accept valid traffic, and UDP as its stateless
        <rule target="ACCEPT>
          valid_traffic;
          dns_udp;
        </rule>
        # Match SYN packets (tcp) and jump to new_connections
        <rule target="new_connections">
          new_connections;
        </rule>
      </chain>

      # Accept output  from ourselves
      <chain name="OUTPUT" default="ACCEPT">
      </chain>

      # Allow valid forwarded traffic, if we had any
      <chain name="FORWARD" default="DROP">
        <rule target="ACCEPT>
          valid_traffic;
        </rule>
      </chain>
    </table>
  </acl>
</firewall>
```

Shorewall

Shorewall (*http://shorewall.net/*) is a tool that can make setting up a firewall easier than using pure **`iptables`** commands. It is not a daemon or service, but is simply a tool that is used to configure netfilter.

Rather than dealing with the sometimes confusing tables, chains, and rules of netfilter, Shorewall abstracts the firewall configuration into a number of easy-to-read files that define interfaces, networks, and the sort of traffic that should be accepted. Shorewall then uses these files to generate the applicable netfilter rules.

There is an excellent configuration guide and basic introduction to networking concepts at: *http://shorewall.net/shorewall_setup_guide.htm*

Building a firewall in BSD

There are three firewalling systems in BSD, which are not compatible with each other. ***IPFW*** is the oldest and most efficient, but does not have as many features as the others. ***IPF*** was created to enhance IPFW, but the author decided to restrict its license, and so it is not completely free. ***PF*** was created as a free replacement for IPF. PF offers the most features, including packet shaping, but is claimed to be less efficient than IPFW. We will give simple examples of using IPFW and PF.

You can find out more about the relative benefits of each firewall here:

- *http://lists.freebsd.org/pipermail/freebsd-ipfw/2004-December/001583.html*
- *http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls.html*

In order to create a useful firewall, you will need to enable packet forwarding through your machine. This can be done with the following command:

```
sysctl net.inet.ip.forwarding=1
```

To make this change permanent, assuming that packet forwarding is disabled, add the following line to **`/etc/sysctl.conf`**:

```
net.inet.ip.forwarding=1
```

To build a firewall with **IPFW**, first enable IPFW functionality in the kernel. The procedure varies between different BSDs. The method described here is for FreeBSD.

```
# cd /usr/src/sys/i386/conf/
# cp GENERIC MYFIREWALL
```

Open MYFIREWALL with your favorite text editor. At the bottom of the file add these lines:

```
options         IPFIREWALL
options         IPFIREWALL_VERBOSE
options         IPFIREWALL_FORWARD
options         IPFIREWALL_VERBOSE_LIMIT=100
options         IPFIREWALL_DEFAULT_TO_ACCEPT
```

Save your work and compile the kernel with the following commands:

```
# /usr/sbin/config MYFIREWALL
# cd ../compile/MYFIREWALL
# make cleandepend
# make depend
# make
# make install
```

To ensure that the firewall is activated at boot time, edit the file **/etc/rc.conf** and add the following lines:

```
gateway_enable="YES"
firewall_enable="YES"
firewall_type="myfirewall"
firewall_quiet="NO"
```

Now reboot the computer to ensure that the firewall is active.

Edit the **/etc/rc.firewall** file and adapt it to your needs. You can apply the new rules with the command **sh /etc/rc.firewall**.

For more information on configuring IPFW, please refer to the manual page (**man ipfw**) and the FreeBSD Website:

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-ipfw.html

To activate **PF**, you should first enable it in the kernel. The procedure varies between different BSDs. The method described here is for FreeBSD.

```
# cd /usr/src/sys/i386/conf/
# cp GENERIC MYFIREWALL
```

Then add the following at the bottom of MYFIREWALL:

```
device pf
device pflog
device pfsync
```

Save the file and then recompile your kernel.

To ensure that PF is activated at boot time and logging is enabled, add this to **/etc/rc.conf**:

```
gateway_enable="YES"
pf_enable="YES"
pf_rules="/etc/pf.conf"
pf_flags=""
pflog_enable="YES"
pflog_logfile="/var/log/pflog"
pflog_flags=""
```

Now reboot the machine to ensure that PF is loaded. You should be able to run the following commands:

- **pfctl -e** to enable PF

- **`pfctl -s`** all to show list all firewall rules and status
- **`pfctl -d`** to disable PF

Edit the **`/etc/pf.conf`** file and uncomment all the lines that apply to your setup. You can apply the new rule set with the command **`pfctl -f /etc/pf.conf`**.

You can find out more about configuring PF in the manual page (**`man pf.conf`**) and the OpenBSD website:

http://www.openbsd.org/faq/pf/

Transparent bridging firewall in FreeBSD

Chances are good that if you are low on bandwidth, you are also short on IP addresses. Many organisations only have a few public IP addresses - perhaps one for the router and another for the proxy server. So how do you introduce a firewall without having to restructure the network or use yet another valuable IP address?

One way to do this is to use a ***transparent bridging firewall***. Such a firewall does not require an IP address, and is able to to protect your LAN, route traffic, and be integrated seamlessly into the network. As your network grows, you can add capacity by simply adding more network cards. FreeBSD has a special feature in its kernel that allows it to function as a bridge, after which you can use any of the firewall programs available in FreeBSD (including ***IPFW***, ***PF***, or ***IPF***). To build a transparent firewall with ***IPFW***, first enable IPFW and bridging functionality in the kernel.

```
# cd /usr/src/sys/i386/conf/
# cp GENERIC MYFIREWALL
```

Open MYFIREWALL with your favorite text editor. At the bottom of the file add these lines:

```
options         IPFIREWALL                    #firewall
options         IPFIREWALL_VERBOSE            #enable logging to syslogd(8)
options         IPFIREWALL_FORWARD            #transparent proxy support
options         IPFIREWALL_VERBOSE_LIMIT=100  #limit verbosity
options         IPFIREWALL_DEFAULT_TO_ACCEPT  #allow everything by default
options         BRIDGE
```

Save your work and compile the kernel with the following commands:

```
# /usr/sbin/config MYFIREWALL
# cd ../compile/MYFIREWALL
# make cleandepend
# make depend
# make && make install
```

To ensure that the firewall is activated at boot time, edit the file **/etc/rc.conf** and add the following lines:

```
gateway_enable="YES"
firewall_enable="YES"
firewall_type="myfirewall"
firewall_quiet="NO"
```

Next, we need to tell FreeBSD which interfaces will be bridged together. This is done by editing **/etc/sysctl.conf** to include the following:

```
net.link.ether.bridge.enable=1
net.link.ether.bridge.config=if1,if2
net.link.ether.bridge.ipfw=1
```

Replace if1 and if2 with your network interfaces. Finally, you can edit **/etc/rc.firewall** and insert your desired filtering rules.

To activate bridging with ***PF***, you should first enable PF and bridging support in the kernel.

```
# cd /usr/src/sys/i386/conf/
# cp GENERIC MYFIREWALL
```

Then add the following at the bottom of **MYFIREWALL**:

```
device pf
device pflog
device pfsync
options BRIDGE
```

Save the file and then recompile your kernel. To ensure that PF is activated at boot time and logging is enabled, add this to **/etc/rc.conf**:

```
gateway_enable="YES"
pf_enable="YES"
pf_rules="/etc/pf.conf"
pf_flags=""
pflog_enable="YES"
pflog_logfile="/var/log/pflog"
pflog_flags=""
```

For PF, you also need to create a bridge interface with the names of the network cards you want to use. Add the following to **/etc/rc.conf**, replacing **xl0** and **xl1** with the network devices on your computer:

```
cloned_interfaces="bridge0"
ifconfig_bridge0="addm xl0 addm xl1 up"
```

To ensure the two network interfaces are activated at boot, add the following to **/etc/rc.local**:

```
ifconfig xl0 up
ifconfig xl1 up
```

Then activate filtering on the two interfaces by adding these lines to **/etc/sysctl.conf**:

```
net.link.bridge.pfil_member=1
net.link.bridge.pfil_bridge=1
net.inet6.ip6.forwarding=1
net.inet.ip.forwarding=1
```

Finally, you can build a transparent bridging firewall using IPF. IPF support is already active on a FreeBSD install, but bridging is not. To use IPF, first enable bridging in the kernel:

```
# cd /usr/src/sys/i386/conf/
# cp GENERIC MYFIREWALL
```

Add these lines to **MYFIREWALL**:

```
options BRIDGE
options IPFILTER
options IPFILTER_LOG
options IPFILTER_DEFAULT_BLOCK
```

Then rebuild the kernel as described above. To activate IPF at boot time, edit **/etc/rc.conf** and add these lines:

```
ipfilter_enable="YES"             # Start ipf firewall
ipfilter_program="/sbin/ipf"
ipfilter_rules="/etc/ipf.rules"   # loads rules definition text file
```

Now edit **/etc/sysctl.conf** in order to create the bridged interfaces. Add the following lines, replacing **xl1** and **fxp0** for your Ethernet devices.

```
net.link.ether.bridge.enable=1
net.link.ether.bridge_ipf=1
net.link.ether.bridge.config=xl1:0,fxp0:0,xl0:1,rl0:1
net.inet.ip.forwarding=1
```

You can also create separate VLANs with this method. If you have four network cards, you can bridge them in pairs to create two separate networks. In the example on the next page, **xl1:0** and **fxp0:0** will bridge one network segment while **xl0:1** and **rl0:1** will bridge another.

```
net.link.ether.bridge.enable=1
net.link.ether.bridge_ipf=1
net.link.ether.bridge.config=xl1:0,fxp0:0,xl0:1,rl0:1
net.inet.ip.forwarding=1
```

Transparent bridging firewall in Linux

The Linux kernel also supports bridging and firewalling. In version 2.4, this required an optional patch, but in version 2.6 it is enabled by default. You only need to make sure that the **BRIDGE_NETFILTER** option is enabled. To check this, run the **make menuconfig** command in the kernel source directory and select the following options:

- Device Drivers
- Networking support
- Networking options
- Network packet filtering
- Bridged IP/ARP packets filtering

Ensure that the last option is enabled, with an asterisk (*) in the box. If you had to change any options, recompile your kernel with the **make** command, and install it with **make install**. Finally, reboot the machine to load the new kernel.

Bring down the network devices that you want to bridge together, e.g. eth0 and eth1:

```
# ifconfig eth0 down
# ifconfig eth1 down
```

Create a new bridge interface with the following command:

```
# brctl addbr br0
```

Add the network devices that you want to bridge together to the bridge device:

```
# brctl addif br0 eth0
# brctl addif br0 eth1
```

Bring all the interfaces up:

```
# ifconfig br0 up
# ifconfig eth0 up
# ifconfig eth1 up
```

Manually assign an IP address to the bridge, e.g.:

```
ifconfig br0 10.20.30.40 netmask 255.255.255.0
```

Now you should be able to write **iptables** firewall rules that control traffic through the bridge by specifying **-i br0** and/or **-o br0**. For example, to allow all traffic through the bridge:

```
# iptables -A FORWARD -i br0 -o br0 -j ACCEPT
```

To block all packets on port 80 from passing through the bridge:

```
# iptables -A FORWARD -i br0 -o br0 -p tcp --dport 80 -j DROP
```

You can also use **physdev-in** and **physdev-out** to match the actual physical device on which the packet entered or left the firewall:

```
iptables -A FORWARD -i br0 -o br0 -m physdev --physdev-in eth0 \
  --physdev-out eth1 -p tcp --dport 80 -j DROP
```

The above rule will drop packets to TCP port 80 (HTTP) going from eth0 to eth1, but not the other way around. The bridge configuration will be lost when you reboot your machine, so you may wish to add the commands to create the bridge to **/etc/rc.d/rc.local**, or the equivalent file on your distribution.

Note that once you add an interface to a bridge with **brctl addif**, netfilter will see packets through that interface as coming from or going to the bridge device (br0) instead of the real physical interface. You will need to adjust any existing firewall rules that refer to real physical devices, such as **-i eth0** or **-o eth1**, replacing the device name with **br0**.

Summary

Firewalls are an important way to control access to and from your network. You could think of them as being like locks on the doors of a house: they are necessary, but not sufficient for high security. Also, firewalls normally work on IP packets and connections, but there are times when you want more control over what can be done at the application layer, for example blocking certain websites and other content. This can be done through the use of an application layer firewall, such as a web proxy with access control (e.g. Squid).

Caching

It takes time to retrieve information from sources on the Internet. The amount of time it takes depends on a variety of factors, such as the distance to the destination and how many other people are requesting data at the same time. Greater distances between the server and client mean longer delays when

sending and receiving data, since (as far as we know) electrical signals are bounded by the speed of light. When you consider the route a single packet may need to travel in order to reach California from Nairobi (up 35 000 Km to a satellite, 35 000 Km back down again, then across the ocean, possibly through another satellite trip, and across an entire continent halfway around the world, and then back again) it's no wonder that there is an upper limit to how fast information can travel across the net.

While we can't do much about changing the speed of light, we can definitely address the problem of too many people requesting information at once. Much of the information requested from the Internet is web traffic. If we can keep a local copy of data retrieved from the web, and intercept subsequent requests for the same information and serve the local copy instead, we can free up significant bandwidth for other uses, and greatly improve the overall feel of the network. Serving images from a local cache may take a few milliseconds, as compared with hundreds (or thousands) of milliseconds to retrieve the same data from distant Internet sites.

Web traffic is the most obvious service that can benefit from a cache, but just about any service that doesn't deal with realtime data can be cached. Prerecorded video streams can be cached, but live video cannot. Voicemail systems can be cached, but ***Voice over IP*** (***VoIP***) cannot. One service that should definitely be cached is DNS. Since virtually every other service makes use of hostname-to-IP address lookups, caching DNS will help "speed" up nearly everything else on the network, while saving bandwidth for other uses.

In this section, we will see specific examples that illustrate how to implement both web and DNS caching on your network.

Web caching

As mentioned earlier, ***caching web proxies*** work in much the same way as does a local browser cache. By saving requested information to a local disk, data is then served on subsequent requests to anyone on the network who needs it. How much can you really save with a web cache such as ***Squid***? While this varies depending on your traffic profile, you can expect to save, on average, between 20% and 30% of your bandwidth. As your users request more cacheable data, and the more those requests overlap, the more bandwidth you will save.

There are literately hundreds of web caches made available by the open source community and commercial vendors. Squid is undoubtedly the most popular web caching software available. It is mature, robust, very fast, and completely open source. In this book we will mainly concentrate on the Squid proxy server, but if you are running another proxy server the general concepts will still apply.

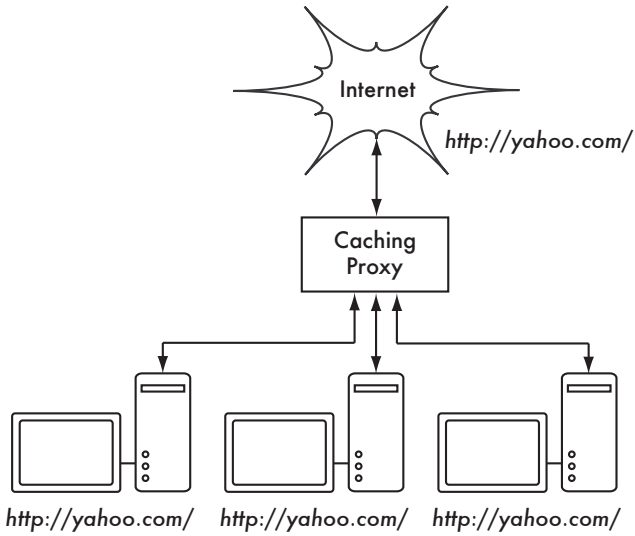

Figure 4.7: Caching proxy servers act as a shared web browser disk cache for all of the users in your network.

Besides saving Internet bandwidth, ***authenticating cache servers*** can provide you with increased control over the ways in which the network is used. When network usage is logged per user, you can then implement various kinds of behaviour modifying techniques, such as quotas or billing. The cache server also gives you a central place from which to watch the browsing habits of your users. The logs that are produced by the proxy server are useful when combined with a log file analysis tool such as Webalizer or Analog (page **81**). Other features include:

- **Bandwidth limiting / traffic shaping**. By implementing delay pools in Squid (page **189**), you can prioritise traffic in ways that make use of network resources in an equitable manner.
- **Redirection and content filtering**. This allows you to block advertisements, resample images, or make other changes to the content before it reaches the client. You can also redirect users to a page of your choice based on certain criteria.
- **Advanced Access Control List (ACL) processing**. ACLs allow you to perform firewall-like features at the application layer (for example, blocking some users while letting others through based on their credentials, IP address, or even time of day).
- **Peer caching**. This allows large sites to use multiple caches and share the cached files between them. By using peer caching you can scale your cache services to networks of any size.

When properly implemented, cache servers can have a significant impact on your bandwidth usage (even before traffic shaping and other access modification tools have been applied).

The caching server and the Firewall

The more a cache server is used, the more effective it will become. Your web cache is not fully effective unless it is used consistently throughout your organisation. If your users can easily bypass the proxy, they will certainly do so for a variety of reasons. They may not like the idea of all of their network requests being logged and monitored. If you have implemented delay pools or other access restrictions, they may notice that bypassing the proxy increases their performance. This is equivalent to cutting in line at a crowded theater. It may work for one user, but it is cheating, and can't work for everyone.

A proxy server necessarily goes hand-in-hand with a good firewall (page **114**). Configuring the firewall to only allow web access via the proxy is one way to ensure that the caching proxy is the only way that users can access the web. Of course, having a firewall doesn't do much good if it isn't properly configured.

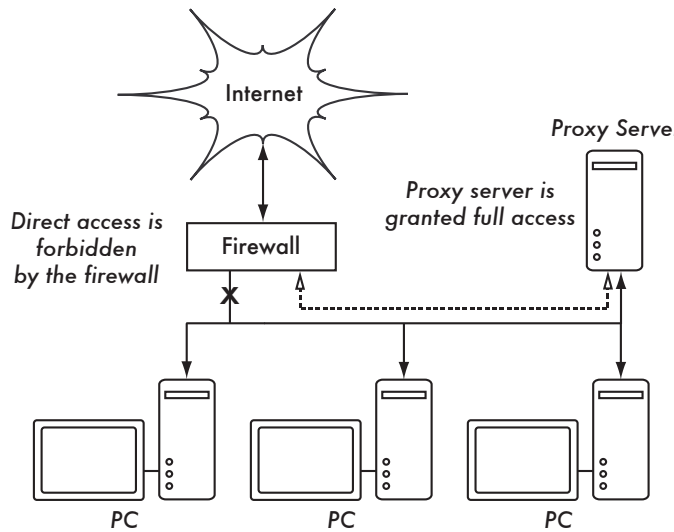

Figure 4.8: Firewalls should be used to enforce use of the caching proxy. With a good firewall in place, the only way to access web content is via the proxy.

It is a common mistake to install a firewall for security, but to allow outbound access to the HTTP/FTP ports (TCP ports 80 and 21). Although you are protecting your network with the firewall, you are not forcing your users to use the cache server. Since your users are not forced to use the cache server, users that access the web directly are potentially wasting bandwidth. This also be-

comes a problem when you authenticate users on your cache server and rely on those logs to determine network usage.

Smaller networks can often install a cache server on the same server as the firewall. In larger networks, the cache server is placed somewhere on the internal network. On very busy networks, multiple cache servers may be used.

Transparent versus manual caches

A ***transparent cache*** is a caching server that works without any explicit proxy configuration on the client. All web traffic is silently redirected to the cache, and responses are returned to the client. This configuration is sometimes called an ***interception caching*** server.

With a traditional cache, the client is configured to use the cache server by either manually setting the proxy settings in the browser preferences, or by using an automatic configuration URL (page **140**). Transparent caches require no configuration on the client side. As far as the client knows, they are receiving web pages directly from the originating servers.

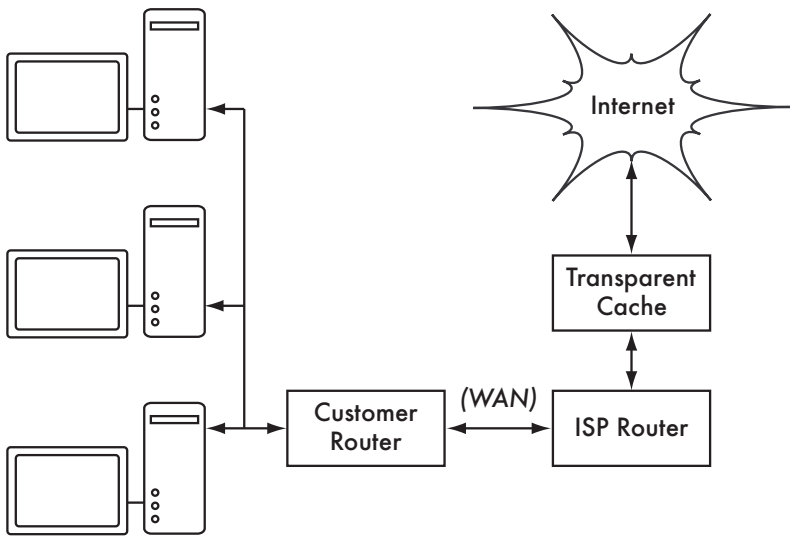

Figure 4.9: Transparent caches can be used by ISPs to save bandwidth by locally caching requests made by their customers.

Transparent caching can make problem solving difficult, as the user may not be aware that their requests are being served by the cache server and not from the original website. Transparent caching also makes proxy authentication impossible, since no credentials are presented to the cache server. This removes the ability to log and create "accountability" reports. While you can still log based on source IP address, reports at the user level can only be achieved by

using authentication. The only real advantage in using a transparent cache is that the clients do not need to be configured. For installations with casual users who bring their own equipment (as is the case in cafes or public labs) this can help to reduce support requests. Transparent caches are often used as upstream caches in the ISP environment.

Transparent caching can be an involved process as it depends on specific features in your firewall and kernel. To learn how to setup a transparent cache with Squid, and various other firewalls, see the documentation at: *http://www.squid-cache.org/Doc/FAQ/FAQ-17.html*

Running Squid

Squid is the premier open source web caching proxy server originally developed by Duane Wessels. Squid has been around for over ten years and is probably the most popular web caching server in use today.

To quote from the Squid wiki;

> *"Squid is a high-performance proxy caching server for web clients, supporting FTP, gopher, and HTTP data objects. Unlike traditional caching software, Squid handles all requests in a single, non-blocking, I/O-driven process. Squid keeps meta data and especially hot objects cached in RAM, caches DNS lookups, supports non-blocking DNS lookups, and implements negative caching of failed requests. Squid supports SSL, extensive access controls, and full request logging. By using the lightweight Internet Cache Protocol, Squid caches can be arranged in a hierarchy or mesh for additional bandwidth savings. Squid consists of a main server program squid, an optional Domain Name System lookup program dnsserver (Squid nowadays implements the DNS protocol on its own by default), some optional programs for rewriting requests and performing authentication, and some management and client tools.*"

You can download Squid from *http://www.squid-cache.org/*. It is also available as a standard package in most Linux distributions. A Windows port is also available at *http://www.acmeconsulting.it/SquidNT/*. For most installations, the default Squid build included in your system package is probably sufficient. In some circumstances you may want to rebuild Squid from source.

Some options, such as delay pools and a few of the authentication helpers, require a recompile. You may also want to recompile Squid, for large installations, to increase the number of available file descriptors (the default is 1024). You should monitor your Squid box with the ***cachemgr*** interface to determine if you have sufficient file descriptors. Detailed instructions on how to increase your file descriptors are available in the Squid FAQ at *http://www.squid-cache.org/Doc/FAQ/FAQ-11.html#ss11.4*.

A Squid configuration file can be as simple as four lines:

```
http_port 3128
cache_mgr your@email.address.here
acl our_networks src 192.168.1.0/24
http_access allow our_networks
```

Change the network ACL to reflect your IP address range, add a contact email address, and that's it. Save the file as **`/etc/squid.conf`**.

When you run Squid for the first time, you will have to initialise the cache directory. To do this, run Squid with the **`-z`** switch:

```
# squid -z
```

Finally, run **`squid`** with no other parameters to launch the Squid daemon.

```
# squid
```

Congratulations, you now have a caching web proxy! All that's left now is to configure your users' browsers to use the proxy. If you only administer a few computers, you can do that manually. For larger installations, it can be easier to enable automatic proxy configuration (see the next section).

While this example will get you started, Squid can be fine-tuned to optimise networks of just about any size. Full documentation is available at *http://www.visolve.com/squid/*. For more advanced features, see the **Performance Tuning** chapter on page **177**. Many more tips and techniques are also available on the Squid Wiki at *http://wiki.squid-cache.org/*.

Automatic web proxy configuration

As mentioned earlier, the main advantage of transparent proxies is that the cache can be used automatically without changing browser settings. The trade-off for this convenience is that some proxy features, such as authentication, cannot be used. One alternative to using transparent proxies is an ***automatic proxy configuration***. This technique allows browsers to automatically detect the presence of the proxy, and works with all proxy features (including authentication).

When configured for automatic proxy discovery, the browser requests a file that defines which proxy to use for a given URL. The file contains javascript code that defines the function **`FindProxyForURL`**. It looks like this:

```
function FindProxyForURL(url, host)
        {
            ...
        }
```

This script should return a string that defines the proxy server to be used (in the format **`PROXY host:port`**), or if a SOCKS proxy is used, it should return the string **`SOCKS host:port`**. The string **`DIRECT`** should be returned if a proxy is not needed. This function is run for every request, so you can define different values for different URLs or hosts. There are also several functions that can be called in the script to help determine what action to take. For example, **`isPlainHostName(host)`** returns true if there is no domain specified for the given host, and **`isResolvable(host)`** returns true if the host name resolves to an IP address.

The file containing the **`FindProxyForURL`** function is usually given a **`.pac`** extension (short for ***Proxy Auto Configuration***) and is placed in a directory on a local web server.

Here is an example **`config.pac`** file that returns **`DIRECT`** for plain host names and local sites, and returns a proxy for everything else:

```
function FindProxyForURL(url, host)
    {
        if (isPlainHostName(host) ||
            dnsDomainIs(host, ".mydomain.com"))
            return "DIRECT";
        if (isInNet(host, "10.1.0.0", "255.255.0.0"))
            return "DIRECT";
        else
            return "PROXY proxy.mydomain.com:8080";
    }
```

You may also need to tell your web server to associate **`.pac`** with the proper mime type. Make sure there is an entry like this in the **`mime.types`** file for your web server:

```
application/x-ns-proxy-autoconfig pac
```

A full description of **`FindProxyForURL`** can be found online at *http://wp.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html*.

Now that we have a working **`config.pac`**, how do we tell our clients to use it? There are three ways to do it: by direct configuration in the browser, with a DNS entry, or via DHCP.

Most browsers have a place in their preferences or settings where the URL to the **`config.pac`** can be defined. In Firefox this is the ***Automatic proxy configuration URL***, and in Internet Explorer it is called ***Use automatic configuration script***. To configure the location manually, check these boxes and insert the URL to the **`config.pac`** script in the appropriate boxes. While this may seem counterintuitive (why do you need to manually define a URL for automatic configuration?) this method still gives you one significant benefit: once your

browsers are configured, they never need to be changed again. The entire network can be configured to use new proxy settings by changing the single **config.pac** file.

The DHCP and DNS methods allow you to define the location of the automatic configuration file without hard coding a URL on every browser. This is implemented by the ***Web Proxy Auto Discovery*** (***WPAD***) protocol. The protocol provides a number of methods for generating a URL that refers to a Proxy Auto Configuration file. This option is enabled in Firefox with the ***Auto-detect proxy settings for this network*** setting. In Internet Explorer, the option is called ***Automatically detect settings in IE***.

The DHCP method is tried first, followed by a series of DNS lookups. To use the DHCP method, define the URL to the **config.pac** using DHCP option 252. In ISC's dhcpd, the configuration looks like this:

```
# Config for ISC DHCP server
option wpad code 252 = text;
option wpad "http://myserver.mydomain/config.pac";
```

When clients request a DHCP lease, they are also given the URL pointing to the **config.pac**. Your clients may need to be rebooted (or their leases released and renewed) in order to pick up the change.

If no DHCP option 252 is provided, the WPAD protocol next attempts to find the proxy server using DNS. If the client has a domain name of *mydomain.com*, the browser will try to look up *wpad.mydomain.com*. If the DNS request is successful, the browser makes an HTTP connection to that address on port 80 and requests the file **/wpad.dat**. This file should have the same contents as the **config.pac** file. To use the DNS method, you will need to configure your DNS server to reply to requests for *wpad.mydomain.com* with the IP address of your web server. You should copy (or rename) the **config.pac** file to **wpad.dat** and place it in the root directory of your web server. Make sure your web server returns the proper mime type for .dat files with an entry like this in your **mime.types**:

```
application/x-ns-proxy-autoconfig dat
```

Now that your browsers can automatically find the proxy server, you can make network-wide changes in a single place. Be sure that the web server that serves the configuration file is always available. If that file cannot be found, then your clients will not be able to find the proxy, and they won't be able to browse.

DNS caching

DNS caching is used to save bandwidth on DNS queries and improve response times. Improvements to DNS can make every other service (including web browsing and email) "feel" faster, since virtually all services make use of host lookups. Each DNS record has a ***Time To Live*** (***TTL***) that defines how long the entry is valid. All name server responses include a TTL along with the record requested. DNS caches keep a record in memory or on disk until the TTL expires, thus saving bandwidth by not having to do queries for frequently requested records.

Most DNS servers will act as a caching proxy. Here are examples of how to implement a cache using dnsmasq, BIND, and djbdns.

dnsmasq

Dnsmasq is a lightweight, easy to configure DNS forwarder and DHCP server. It is available for BSD and most Linux distributions, or from *http://freshmeat.net/projects/dnsmasq/*. The big advantage of dnsmasq is flexibility: it acts as both a caching DNS proxy and an authoritative source for hosts and domains, without complicated zone file configuration. Updates can be made to zone data without even restarting the service. It can also serve as a DHCP server, and will integrate DNS service with DHCP host requests. It is lightweight, stable, and flexible. Bind is likely a better choice for large networks (more than a couple of hundred nodes), but the simplicity and flexibility of dnsmasq makes it attractive for small to medium sized networks.

Setting up Dnsmasq is pretty easy. If you already have **`/etc/hosts`** and **`/etc/resolv.conf`** set up, run dnsmasq and point other computers to the server's IP address (using DHCP or manually configuring the DNS server). For more complex configuration, the **`/etc/dnsmasq.conf`** file contains documentation explaining all the various variables.

BIND (named)

The ***Berkeley Internet Name Domain*** (***BIND***) is capable of serving zones, acting as a slave, performing caching and forwarding, implementing split horizon (page **212**), and doing just about anything else that is possible with DNS. BIND is used by a vast majority of the Internet community to serve DNS, and considered to be stable and robust. It is provided by the Internet Software Consortium at *http://www.isc.org/sw/bind/*. Virtually all versions of Linux and BSD include a package for BIND.

To run BIND as a caching DNS server, make sure there is an entry for the root zone in your **named.conf**:

```
# Hints file, listing the root nameservers
zone "." {
        type hint;
        file "root.cache";
};
```

That's all there is to it. Make sure your clients use this server for DNS instead of your ISP's DNS server, and you're good to go.

djbdns

DJBDNS is a DNS server implemented by D. J. Bernstein. It is focused on security, and is claimed to be free of security holes. Unfortunately, djbdns has a restrictive license which prevents patching and redistribution of patched executables, making it difficult for distribution maintainers to package it. To use djbdns, you will need to download it from *http://cr.yp.to/djbdns.html* and compile it yourself.

Once djbdns has been built and installed, it can be configured as follows to provide DNS caching. First, create the users and configuration we need:

```
$ useradd Gdnscache
$ useradd Gdnslog
$ dnscache-conf Gdnscache Gdnslog /etc/dnscache 127.0.0.1
```

Next, install it as a service:

```
$ ln -s /etc/dnscache /service
$ sleep 5
$ svstat /service/dnscache
```

DNS queries made on this server should now be cached.

Mirroring

Mirroring is a technique used to synchronise files between hosts. This can be used to store local copies of software updates on a server, provide redundancy for load distribution between servers, or even perform backup. Mirroring saves bandwidth by providing a local copy of popular data. Users request information from the mirror rather than directly from the site on the Internet. Mirrors are updated at off-peak times to minimise the impact on other services.

Software Updates

Most packages or operating systems require software updates. If you look at the output from your proxy log file analysis, it will become apparent that one or two software update sites contribute significantly to your bandwidth consumption. While a caching proxy will take some of the load off, you can proactively populate a local mirror with software updates, and redirect your users there. Some common applications that update themselves are:

- **Windows.** Windows updates are a common problem, since machines asynchronously decide when to update themselves, and download redundant copies of the same patches. Fortunately, you can to set up a ***Microsoft Windows Server Update Services*** (***WSUS***) mirror on your local network. Clients can then be made to point to the local server for updates through a simple registry entry. Full details on setting up your own Windows Update Server can be found at *http://www.microsoft.com/wsus*
- **Adobe software.** If you have Adobe Acrobat installed on several machines, it will frequently look for and download updates. Keep local, up to date copies and encourage users to disable automatic updates.
- **Linux distributions.** Some Linux distributions can download vast amounts of updates automatically at regular intervals. You should decide on a few distributions that you want to support and try to keep a local mirror, even if it is only used for updates. Data repositories for Linux distributions can be mirrored locally using rsync or wget, and configured to be used by system update tools (such as yum and apt).

There are many other software packages that "call home" to look for updates. Some of these include anti-virus software, most shareware, and even the Mac OS X operating system. By watching your proxy logs (page **81**), you can get a clear idea of what information is frequently requested, and make local copies to remove some of the load on the Internet connection.

Some services lend themselves well to mirroring, while others do not. Large software repositories (such as Linux and BSD distributions, public FTP servers, software update sites, etc.). are ideal for local mirroring. Dynamic sites (such as news sites and web mail services) cannot be mirrored, but may benefit from caching (page **135**). While there isn't a standard method for mirroring arbitrary sites, several tools can help you create local copies of Internet content.

GNU wget

GNU ***wget*** is a free utility for non-interactive download of files from the web. It supports HTTP, HTTPS, and FTP, as well as retrieval through HTTP proxies. By limiting the transfer rate to any value, wget can shape usage to fit the available bandwidth. It can be run manually or in the background as a non-

interactive process, downloading as many web pages as you like from a particular site. This allows you to start a retrieval and disconnect from the system, letting wget finish the work.

Wget supports a full-featured recursion mechanism, through which it is possible to retrieve large parts of the web automatically. The level of recursion and other mirroring parameters can be specified. It respects the ***robot exclusion standard*** (specified by the file **`robots.txt`** in the root directory of a web server).

Wget has been designed for robustness over slow or unstable network connections. If a download fails due to a network problem, it will keep retrying until the whole file has been retrieved. If the server supports resumed downloads, it will instruct the server to continue the download from where it left off. Wget can even convert the links in downloaded HTML files to point to local files. This feature facilitates off-line viewing.

There are quite a few command-line options for Wget. From a terminal, **`man wget`** or **`wget --help`** will list all of them. Here are some examples to get you started.

This will create a mirror of *www.yahoo.com*, retrying each request once when errors occur. The result will have the same directory structure as the original website. Results are logged to the file **`wget.log`**.

```
wget -r -t1 http://www.yahoo.com/ -o wget.log
```

You can schedule wget to run at any time using cron. This crontab will mirror the Xemacs ftp server every Sunday at midnight:

```
0 0 * * 0 wget --mirror ftp://ftp.xemacs.org/pub/xemacs/ -o /home/me/xfer.log
```

The **`--mirror`** option turns on recursion and time stamping, sets infinite recursion depth and keeps ftp directory listings.

Wget is so popular that it is included by default in many Linux distributions. You can download the latest version from *http://www.gnu.org/software/wget/*.

Curl

Like wget, ***curl*** is a command line tool for downloading data from URLs, but is far more flexible. From the *http://curl.haxx.se/* website:

> *curl is a command line tool for transferring files with URL syntax, supporting FTP, FTPS, TFTP, HTTP, HTTPS, TELNET, DICT, FILE and LDAP. curl supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, user+password authentication (Basic,*

Digest, NTLM, Negotiate, kerberos...), file transfer resume, proxy tunneling and a busload of other useful tricks.

One difference between wget and curl is that curl assumes that you want to use standard out for the retrieved data. To retrieve a web page and save it to **`file.html`**, use the **`-o`** switch:

```
curl -o file.html http://www.yahoo.com
```

Unfortunately, one feature that curl does **not** have is recursive downloading. But this is supported using a wrapper script for curl, such as **`curlmirror`** (*http://curl.haxx.se/programs/curlmirror.txt*). Run **`curlmirror -?`** for a list of options. Curl can be downloaded from the main web site, and is included in many modern Linux distributions.

HTTrack

HTTrack (*http://www.httrack.com*) is a website mirroring and offline browser utility. It allows you to recursively download and store entire websites, rewriting URLs to facilitate offline browsing.

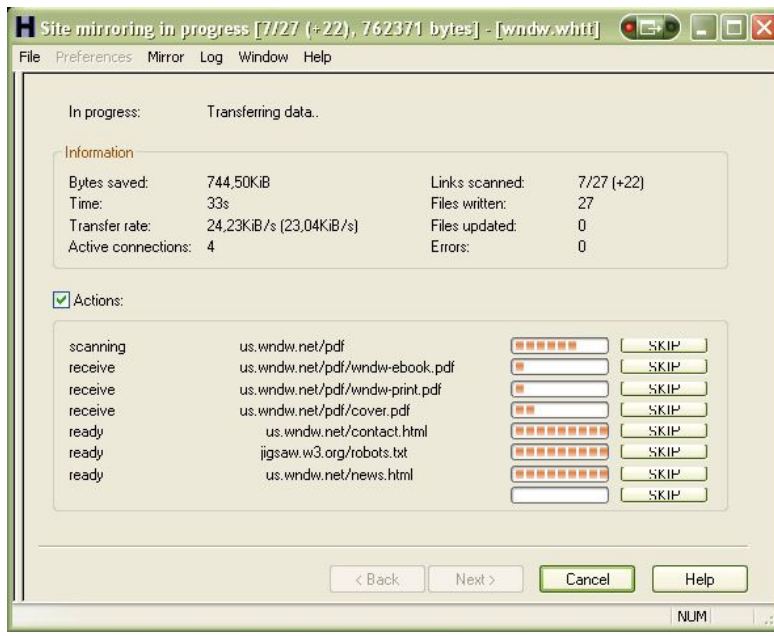

Figure 4.10: HTTrack can save entire websites for offline browsing.

It can also update an existing mirrored site without downloading the entire contents again, as well as resume interrupted downloads. There are versions available for Windows 9x/NT/2000/XP, as well as Linux, BSD, and Mac OS X.

HTTrack has a friendly, intuitive interface and an extensive help system. It is much easier to run than other mirroring software, but lacks some features (such as download scheduling). It is open source and released under the GPL.

rsync

There are far more efficient ways to mirror content than over HTTP. If you have shell access on a remote file server, you can use ***rsync*** instead. This lets you synchronise files and directories between systems much faster, without tying up public web server resources. You can run rsync over its own TCP port, or tunnel it over SSH for added security.

It can be quicker to run rsync rather than using a web mirroring utility, since rsync can transfer only changes within a file rather than the entire file. This can help when mirroring large files, such as compressed backups, or file archives. You can only use rsync on systems that run a public rsync server, or servers on which you have shell access.

Just about all versions of Linux and BSD include rsync, and it will also run on Windows and Mac OS X. You can download it at *http://samba.org/rsync/*.

Email

Email was the most popular application of the Internet before Tim Berners-Lee invented the World Wide Web, and is still extremely popular today. Most Internet users have at least one email address. Email's long history of development has made it cheap, fast, resilient, and almost universally supported. All modern computers, mobile phones, PDAs, and many other electronic devices have email support, making it nearly as ubiquitous as the telephone. In many organisations, email has even surpassed the telephone as the standard means of communication, where it is used for everything from internal information exchange to customer-facing sales and support. Many companies provide free email services (e.g. Microsoft, Yahoo, and Google).

Email Basics

Email senders use a program such as Thunderbird or Outlook (or even a Web browser) to create their message. When they click the "Send" button, the message is passed on to a local ***Mail Transfer Agent*** (***MTA***), often located in their company or at their Internet Service Provider. The MTA's task is to deliver the email to the recipient's MTA via the Internet.

Email is usually transferred across the Internet using a protocol known as the ***Simple Mail Transfer Protocol*** (***SMTP***). The current version is defined in RFC2821 (*http://www.rfc.net/rfc2821.html*). Earlier RFCs related to email date

back as far as 1980, and primordial electronic messaging systems existed for years before that.

When the recipient's MTA receives an email, it often cannot be delivered directly to the recipient's computer. This could be because the recipient is not online, their mail client is not running, or they do not wish to store their mail on their own machine. Therefore, the recipient's MTA usually delivers the email to a storage service instead. This is known as a ***Mail Delivery Agent*** (***MDA***). The client can collect their mail from the MDA whenever they like, using protocols such as POP3 and IMAP. The program that ultimately retrieves the mail and displays it to the user is called the ***Mail User Agent*** (***MUA***).

Email Security

Unfortunately, SMTP is a simple protocol that does not offer much security. Due to its popularity and widespread support, it has proved impossible to replace. This lack of security has made it trivial for unscrupulous users to send vast amounts of ***unsolicited bulk email*** (a.k.a. ***spam***). The senders are often called ***spammers***. Most users do not want to receive spam, and it wastes their time and bandwidth.

It is in your best interests to limit the amount of junk mail that passes through your mail server. Your organisation is ultimately responsible for the email that emanates from your network. If you allow the entire world to relay mail through your email server, you will not only add to the growing spam problem, but you can easily get your mail server or IP address range blacklisted. If this happens, you will not be able to send or receive email at all, as major mail servers around the world will refuse to communicate with you. Acting as a spam relay may even violate the terms of service of your ISP, and could get you disconnected. If your bandwidth is charged by the byte, you are likely to receive a bill for all of the spam that came from your network before it was shut off!

There are four critical steps to keep in mind when running an Internet email service from your network. These steps are:

1. **Choose a good MTA.** There are many available, so you should choose one that is secure, fast, and flexible enough to support mail services at your organisation. (page **151**)
2. **Eliminate all open relays.** Users should be authenticated (by IP address, username and password, or some other credentials) before your server accepts the email for relaying. (page **166**)
3. **Implement anti-spam and anti-virus measures for inbound (and, if possible, outbound) email.** For inbound email, this will help reduce complaints from your users. For outbound email, this can help you to find malicious users and machines infected with viruses. (page **152**)

4. **Closely monitor the volume and profile of mail sent from your network.** While they can catch more than 99% of unwanted email traffic, no email filter is completely infallible. A sudden increase in email volume, or mail being sent at unusual hours, can indicate that spam is getting around your filter and leaving your network. (page **82**)

By following these four steps, you can go a long way towards reducing the bandwidth spent on handling email, and keeping your network off the blacklists. But since email is so widely abused, there are a few other subtle points to keep in mind. If your organisation uses dynamic IP space (page **37**), you may run into unexpected trouble with your mail server. Some organisations that compile blacklists also collect lists of known dynamic IP ranges. These IP lists are then published in the form of a ***DNS Black List*** (***DNSBL***) and can be used by large ISPs for rejecting mail. Since dynamic IPs are relatively anonymous, they are often havens for prolific spammers.

It is also common practice for large ISPs to block all traffic destined for port 25 (the SMTP port) originating from their dynamic IP space, in an effort to prevent outbound spam. Whenever possible, use a static IP address for your email server. If this is not possible, you should configure your email server to relay all outbound mail to the SMTP server at your ISP. This is required by some ISPs in order to track email abuse.

Choosing an MTA

On UNIX-like operating systems such as Linux and FreeBSD, there are four popular MTAs: ***Sendmail***, ***qmail***, ***Postfix***, and ***Exim***. All of these are quite powerful and reliable, but choosing the right one for your needs is vital, as there are some important differences between them.

- **Sendmail** (*http://www.sendmail.org/*) is the oldest MTA still in common use. When it was released in the early 1980s, it was responsible for delivering almost all of the e-mail sent over the Internet. Its popularity has declined due to its reputation for being difficult to configure, as well as the discovery of a number of critical security flaws over the years. Despite this, it is still included by default with many UNIX operating systems. If properly configured, Sendmail is efficient and has many useful features, but unless you are already familiar with the program or it has been set up for you already, you should probably consider one of the other, more easy-to-use MTAs. Sendmail is being completely re-written and the new version, known as Sendmail X, may resolve some of the issues with the original program.
- **qmail** (*http://www.qmail.org/*) was written specifically because of security concerns with Sendmail. It has an unusual design that has proven very successful at repelling attacks, and is also known for being very fast, stable, and minimal (that is, it has a relatively small set of features compared to the other

MTAs). The author of the program has not updated it since 1997, and the licensing conditions prevent any modifications from being officially released, although unofficial patches are available. Because of its unconventional nature, it is not recommended that you use qmail unless you are already familiar with it or are willing to put the effort into learning more about it. For more information, see *http://www.lifewithqmail.org/*.

- **Postfix** (*http://www.postfix.org/*) was developed by a security expert at IBM. It has a significantly better security record than Sendmail, and is considered easier to use than qmail. Like qmail, and unlike Sendmail and Exim, Postfix emphasises security over features. However, it has a more conventional design than qmail and its configuration is easier to understand. It is the only MTA besides Sendmail to support Sendmail's mail filters (***milters***). It is also well documented and has good community support. Postfix is very efficient in memory usage, speed, and bandwidth.
- **Exim** (*http://www.exim.org/*) was developed at Cambridge University (UK) before the release of qmail or Postfix, to provide them with a replacement for Sendmail. The configuration file is easy to read and understand, and the documentation is very good. Exim is extendable and easy to configure, making it simple to do clever and strange things with mail systems. The author has never claimed that Exim is secure, and early versions had many security problems. But since 2000 its security record has been quite good.

In summary, Postfix and Exim are likely to be the most appropriate choices for inexperienced users. Postfix seems to have a better security record, while Exim is more flexible. Both are relatively easy to use. Postfix may have better performance for larger sites. Exim is recommended for smaller sites, people new to mail systems, and sites that need strange/unusual mail configurations. For exceptionally large sites, qmail may provide better performance, but it comes with a very high learning curve.

If you are stuck with a Windows server, there are a few free mail servers that you can try. All of these include MTA and MDA functionality (in other words, they can receive, send, and store mail):

- **Mercury Mail** (*http://www.pmail.com/*) is a feature-rich mail server. It was designed to work with the Pegasus Mail client (which is an MUA), but it is also very standards-compliant and should work with most MUAs. Two versions are available: one for Novell NetWare and the other for Windows.
- **Macallan** (*http://macallan.club.fr/MMS*) runs on Windows XP and 2000. It provides all the basic mail server features, and includes built-in webmail. The free version supports up to 128 mailboxes.
- **MailEnable** (*http://www.mailenable.com/*) is a Windows mail server that supports SMTP and POP3. The free version does not support IMAP. It has a

mailing list manager that makes it easy to administer subscription-based distribution lists.

- **BaSoMail** (*http://www.baso.no/*) is a compact and easy-to-use SMTP/POP3 server for Windows. It does not support IMAP.

Securing your mail server

The war on spam is constantly evolving as developers find new and clever ways to outwit the spammers, and spammers find devious and subtle ways to outwit the developers. A strong email server implementation will include the following components:

- **A secured MTA installation**. The choice of MTA is up to you, but it should authenticate outbound email before sending to eliminate the possibility of open relays. Follow the installation instructions for your MTA to be sure that authentication is properly enabled (page **152**). You can test your own email server for holes using abuse.net's **Mail relay testing tool**, available at *http://www.abuse.net/relay.html* . Another popular testing tool is the ORDB at *http://www.ordb.org/submit/*, provided by the Open Relay Database.
- A spam filtering package, such as **SpamAssassin** or **DSPAM**. SpamAssassin provides a way to reduce, if not completely eliminate, unsolicited junk mail from your incoming email. It uses a scoring system based on local and network defined rules to identify messages which appear to be spam. It then adds headers to the message that identify the total spam "score," allowing the messages to be easily filtered by the user's mail client. DSPAM approaches the problem differently, by relying almost completely on automated language analysis and deobfuscation techniques instead of user-contributed rulesets and blacklists. While we use SpamAssassin for the examples in this book, both packages are open source and are worth considering. Spamassassin can be downloaded from *http://spamassassin.apache.org/*. DSPAM is available at *http://dspam.nuclearelephant.com/*.
- **ClamAV** (*http://www.clamav.net/*). Clam Antivirus is fast and robust open source antivirus package. It includes a multi-threaded daemon (for speed) and a simple command line scanner. Its virus definition files are automatically updated from the Internet, keeping it up to date as new viruses are discovered. When used in combination with an MTA, ClamAV allows you to scan for viruses and other harmful content embedded in email, before it is delivered to the user.
- **Amavisd-new** (*http://www.ijs.si/software/amavisd/*). This package acts as an interface between your MTA and content filters such as SpamAssassin and ClamAV. It works with every popular MTA including Postfix, Sendmail, Qmail, and Exim.
-

From the AMaViS website:

> *amavisd-new is a high-performance interface between mailer (MTA) and content checkers: virus scanners, and/or SpamAssassin. It is written in Perl for maintainability, without paying a significant price for speed. It talks to MTA via (E)SMTP or LMTP, or by using helper programs.*

By combining amavisd-new with your content filters (SpamAssassin and ClamAV), it is possible to build a robust email solution capable of handling very large email loads, even on relatively modest hardware. All of these components are open source.

You should combine all of these, if possible, to defend your mail server and your users against spam and viruses.

Greylisting

While blacklists provide explicit criteria used to refuse mail, and whitelists explicitly permit mail, ***greylists*** provide the functionality of both. Each time a given mailbox receives an email from an unknown IP address, the mail is automatically rejected with a "try again later" message. This happens at the SMTP layer, and is transparent to the sender or recipient. Since most spamming software is quite simple and does not comply with established RFCs, will not try again later and will simply drop the message. This can provide an immediate and dramatic reduction in spam without using content filtering.

If the MTA tries again after the greylisting period (typically 30 minutes), a combination of the sender's email address, IP address, and the recipient's email address is whitelisted for 3 days. If email from the same sender and IP address is sent to the same recipient is within 3 days, it is sent through without delay and the whitelisting period is reset for a further 3 days.

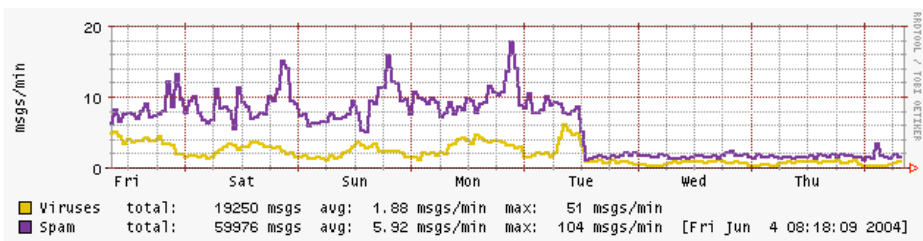

Figure 4.11: Greylisting can show an immediate improvement in deterring spam, while consuming modest resources.

While some users report very good results using greylists, others have had mixed results.

The author of one package, exim-greylist, writes on his website:

> *I no longer maintain this because I no longer use greylisting on my server, it proved to be little efficient when having a lot of other SPAM checks and often deferred valid email for me and my users. I'm leaving this here for reference only.*

If you decide to implement greylisting, you should evaluate its performance to determine how effectively it combats the sort of spam that is directed at your network.

Greylisting must be implemented in the MTA, and is typically achieved by using third party software. The following is a list of freely available, open source greylisting software packages for various MTAs.

For **Postfix**:

- GLD (*http://www.gasmi.net/gld.html*) - Greylist server with MySQL database by Salim Gasmi.
- SQLgrey (*http://sqlgrey.sourceforge.net/*) - Greylist policy server with auto-whitelisting in Perl with support for PostgreSQL, MySQL and SQLite storage by Lionel Bouton.
- GPS (*http://mimo.gn.apc.org/gps/*) - Greylist policy server in C++ using MySql, Postgres, or SQLite by Michael Moritz.
- PostGrey (*http://isg.ee.ethz.ch/tools/postgrey/*) - Greylist policy server in Perl by David Schweikert.
- Policyd (*http://policyd.sourceforge.net/*) - Policy server in C which provides greylisting, sender (envelope, SASL, or host/ip) based throttling (messages and/or volume per hour) and spam traps by Cami Sardinha.
- TumGreySPF (http://www.tummy.com/Community/software/tumgreyspf/) - Greylisting and SPF policy server by Sean Reifschneider. This uses the file system (instead of a database file) for storing greylist data and configuration information.

For **Sendmail**:

- milter-greylist (*http://hcpnet.free.fr/milter-greylist/*) - milter-greylist is a stand-alone milter written in C that implements the greylist filtering method.
- SMFS (*http://smfs.sourceforge.net/*) - Lightweight, fast, and reliable Send-mail milters for SPAM and virus filtering. SMFS includes support for greylists, SPF, SpamAssassin, and other filtering techniques.

For **Exim**:

- greylistd (*http://www.tldp.org/HOWTO/Spam-Filtering-for-MX/exim.html*) - a Python implementation by Tor Slettnes.
- Direct support in Exim configuration file (no external programs) using MySQL (*http://theinternetco.net/projects/exim/greylist*) or Postgres (*http://raw.no/personal/blog/tech/Debian/2004-03-14-15-55_greylisting*)
- exim-greylist (*http://johannes.sipsolutions.net/Projects/ex*), advanced greylisting including statistics and manual whitelisting capability with Exim 4 and MySQL. No longer maintained.

For **Qmail**:

- qgreylist (*http://www.jonatkins.com/page/software/qgreylist*)
- denysoft_greylist (*http://www.openfusion.com.au/labs/dist/denysoft_greylist*)

DomainKeys

DomainKeys are designed not only to verify the authenticity of an email sender, but also to ensure that the email has not been tampered with in transit. Under the DomainKeys system, the sender generates two cryptographic keys: the public key and the private key. Anything encoded with the public key can be decoded using the private key, and vice versa. The public key is published in the sender's DNS records.

The sender signs all of their messages with a signature that has been generated based on the contents of the email, and encoded using the sender's private key. When the email arrives, the receiver can look up the public key in the DNS record for the domain from which the email claims to be. If the sender is authentic and really has access to the private key, then the public key can decode the encrypted signature, and hence the receiver can be confident that the sender is authentic and that the email has not been tampered with in transit.

More details on how to set up DomainKeys with various MTAs can be found at *http://antispam.yahoo.com/domainkeys*. Yahoo! is one of the chief proponents of DomainKeys.

Sender Policy Framework (SPF)

Sender Policy Framework (***SPF***) is designed to fight email forgery, which is often used in scam emails. SPF allows you to verify that mail apparently from a

particular domain (say, a financial institution or government office) was sent from an authorised mail server. SPF verifies the path that email took to arrive at your server, and can discard email that originated at unauthorised MTAs before the message is transmitted, thus saving bandwidth and reducing spam. It works by publishing DNS records that indicate valid mail servers for your domain. Before mail is accepted, the receiving MTA does a DNS lookup to verify that the sender is an authorised email server for the domain. If not, it terminates the connection. If SPF records for the domain match the sender's MTA, the mail is accepted.

While this can do little to help with email sent from bogus addresses (or domains that do not publish SPF records), it promises to reduce email forgeries as more organisations implement it. SPF documentation is available from the main site at *http://www.openspf.org/*.

As with greylisting and domain keys, SPF is implemented in your MTA. The following is a list of packages that provide SPF support for various MTAs.

For **Postfix**:

- libspf2-1.2.x Patch (*http://www.linuxrulz.org/nkukard/postfix/*) - Patch to Postfix to allow native SPF support by Nigel Kukard.
- TumGreySPF (*http://www.tummy.com/Community/software/tumgreyspf/*) - Greylisting and SPF policy server by Sean Reifschneider. This uses the file system (instead of a database file) for storing greylist data and configuration information.

For **Sendmail**:

- SMFS (http://smfs.sourceforge.net/) - Lightweight, fast and reliable Sendmail milters for SPAM and virus filtering.
- SPFMilter (http://www.acme.com/software/spfmilter/)

Resources

- DomainKeys entry on Wikipedia, (*http://en.wikipedia.org/wiki/DomainKeys*)
- Firewalls in FreeBSD,
  http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls.html
- MTA comparisons:
  http://shearer.org/MTA_Comparison
  http://www.python.org/cgi-bin/faqw-mm.py?req=show&file=faq04.002.htp

- Netfilter Packet Filtering HOWTO: *http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO-7.html*
- Netfilter documentation: *http://www.netfilter.org/documentation/*
- Network Address Translation HOWTO : *http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html*
- Squid documentation, *http://www.visolve.com/squid/*
- Squid FAQ, *http://www.squid-cache.org/Doc/FAQ/FAQ-17.html*
- Squid Wiki, *http://wiki.squid-cache.org/*
- Wessels, Duane. *Squid: The Definitive Guide*. O'Reilly Media (2004). *http://squidbook.org/*