5
Networking Hardware

In the last couple of years, an unprecedented surge in interest in wireless networking hardware has brought a huge variety of inexpensive equipment to the market. So much variety, in fact, that it would be impossible to catalog every available component. In this chapter, we'll look at the sort of features and attributes that are desirable in a wireless component, and see several examples of commercial and DIY gear that has worked well in the past.

Wired wireless

With a name like "wireless", you may be surprised at how many wires are involved in making a simple point-to-point link. A wireless node consists of many components, which must all be connected to each other with appropriate cabling. You obviously need at least one computer connected to an Ethernet network, and a wireless router or bridge attached to the same network. Radio components need to be connected to antennas, but along the way they may need to interface with an amplifier, lightning arrestor, or other device. Many components require power, either via an AC mains line or using a DC transformer. All of these components use various sorts of connectors, not to mention a wide variety of cable types and thicknesses.

Now multiply those cables and connectors by the number of nodes you will bring online, and you may well be wondering why this stuff is referred to as "wireless". The diagram on the next page will give you some idea of the cabling required for a typical point-to-point link. Note that this diagram is not to scale, nor is it necessarily the best choice of network design. But it will introduce you to many common interconnects and components that you will likely encounter in the real world.

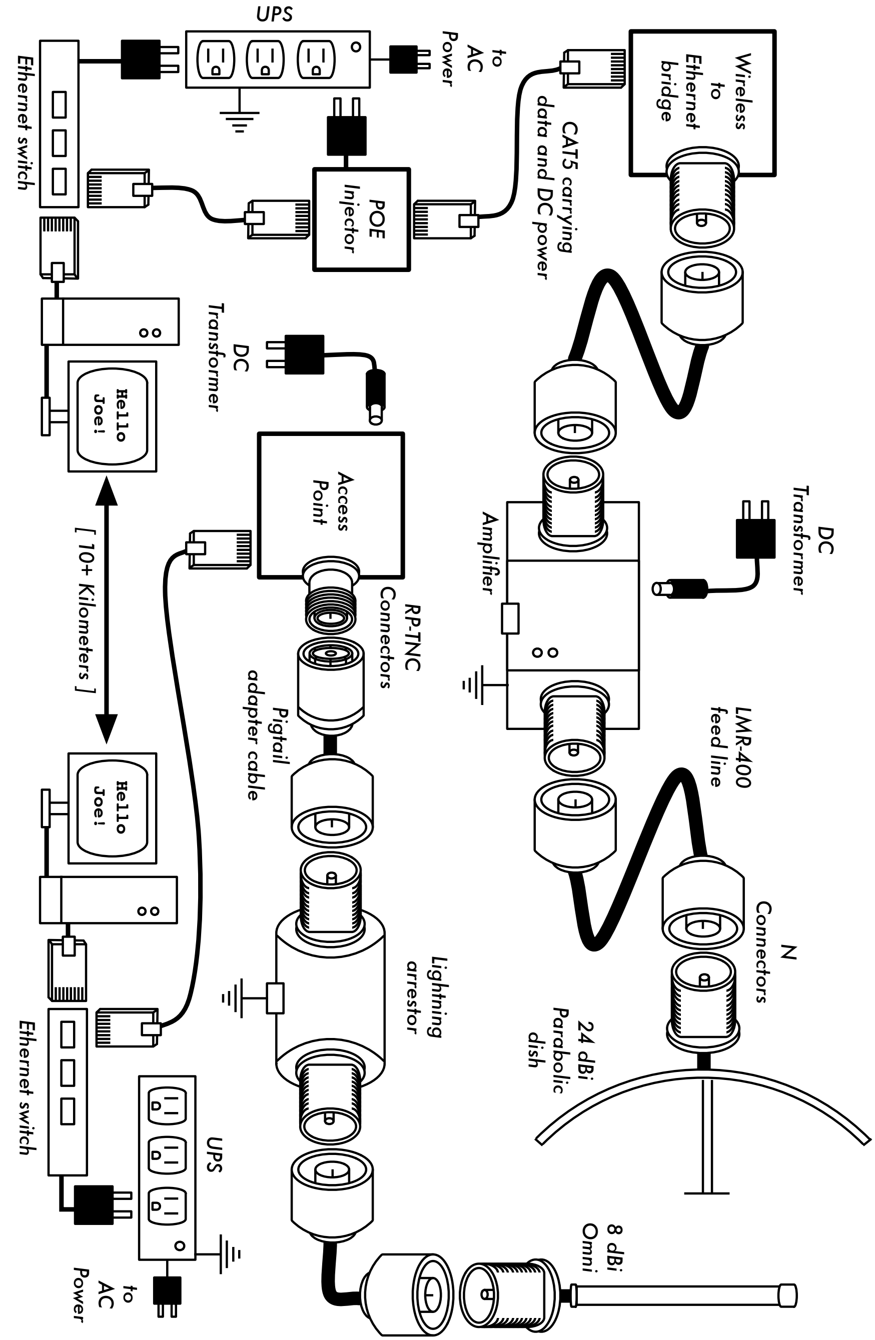


Figure 5.1: Component interconnects.

While the actual components used will vary from node to node, every installation will incorporate these parts:

1. An existing computer or network connected to an Ethernet switch.
2. A device that connects that network to a wireless device (a wireless router, bridge, or repeater).
3. An antenna that is connected via feed line, or is integrated into the wireless device itself.
4. Electrical components consisting of power supplies, conditioners, and lightning arrestors.

The actual selection of hardware should be determined by establishing the requirements for the project, determining the available budget, and verifying that the project is feasible using the available resources (including providing for spares and ongoing maintenance costs). As discussed in **Chapter 1**, establishing the scope of your project is critical before any purchasing decisions are made.

Choosing wireless components

Unfortunately, in a world of competitive hardware manufacturers and limited budgets, the price tag is the single factor that usually receives the most attention. The old saying that "you get what you pay for" often holds true when buying high tech equipment, but should not be considered an absolute truth. While the price tag is an important part of any purchasing decision, it is vital to understand precisely what you get for your money so you can make a choice that fits your needs.

When comparing wireless equipment for use in your network, be sure to consider these variables:

- **Interoperability.** Will the equipment you are considering work with equipment from other manufacturers? If not, is this an important factor for this segment of your network? If the gear in question supports an open protocol (such as 802.11b/g), then it will likely interoperate with equipment from other sources.
- **Range.** As we saw in **Chapter 4**, range is not something inherent in a particular piece of equipment. A device's range depends on the antenna connected to it, the surrounding terrain, the characteristics of the device at the other end of the link, and other factors. Rather than relying on a semi-fictional "range" rating supplied by the manufacturer, it is more useful to know the ***transmission power*** of the radio as well as the ***antenna gain*** (if

an antenna is included). With this information, you can calculate the theoretical range as described in **Chapter 3**.

- **Radio sensitivity.** How sensitive is the radio device at a given bit rate? The manufacturer should supply this information, at least at the fastest and slowest speeds. This can be used as a measure of the quality of the hardware, as well as allow you to complete a link budget calculation. As we saw in **Chapter 3**, a lower number is better for radio sensitivity.
- **Throughput.** Manufacturers consistently list the highest possible bit rate as the "speed" of their equipment. Keep in mind that the radio symbol rate (eg. 54 Mbps) is never the actual throughput rating of the device (eg. about 22 Mbps for 802.11g). If throughput rate information is not available for the device you are evaluating, a good rule of thumb is to divide the device "speed" by two, and subtract 20% or so. When in doubt, perform throughput testing on an evaluation unit before committing to purchasing a large amount of equipment that has no official throughput rating.
- **Required accessories.** To keep the initial price tag low, vendors often leave out accessories that are required for normal use. Does the price tag include all power adapters? (DC supplies are typically included; power over Ethernet injectors typically are not. Double-check input voltages as well, as equipment is often provided with a US-centric power supply). What about pigtails, adapters, cables, antennas, and radio cards? If you intend to use it outdoors, does the device include a weatherproof case?
- **Availability.** Will you be able to easily replace failed components? Can you order the part in large quantity, should your project require it? What is the projected life span of this particular product, both in terms of useful running time in-the-field and likely availability from the vendor?
- **Other factors.** Be sure that other needed features are provided for to meet your particular needs. For example, does the device include an external antenna connector? If so, what type is it? Are there user or throughput limits imposed by software, and if so, what is the cost to increase these limits? What is the physical form factor of the device? How much power does it consume? Does it support POE as a power source? Does the device provide encryption, NAT, bandwidth monitoring tools, or other features critical to the intended network design?

By answering these questions first, you will be able to make intelligent buying decisions when it comes time to choose networking hardware. It is unlikely that you will be able to answer every possible question before buying gear, but if you prioritize the questions and press the vendor to answer them before committing to a purchase, you will make the best use of your budget and build a network of components that are well suited to your needs.

Commercial vs. DIY solutions

Your network project will almost certainly consist of components purchased from vendors as well as parts that are sourced or even fabricated locally. This is a basic economic truth in most areas of the world. At this stage of human technology, global distribution of information is quite trivial compared to global distribution of goods. In many regions, importing every component needed to build a network is prohibitively expensive for all but the largest budgets. You can save considerable money in the short term by finding local sources for parts and labor, and only importing components that must be purchased.

Of course, there is a limit to how much work can be done by any individual or group in a given amount of time. To put it another way, by importing technology, you can exchange money for equipment that can solve a particular problem in a comparatively short amount of time. The art of building local telecommunications infrastructure lies in finding the right balance of money to effort needed to be expended to solve the problem at hand.

Some components, such as radio cards and antenna feed line, are likely far too complex to consider having them fabricated locally. Other components, such as antennas and towers, are relatively simple and can be made locally for a fraction of the cost of importing. Between these extremes lie the communication devices themselves.

By using off-the-shelf radio cards, motherboards, and other components, you can build devices that provide features comparable (or even superior) to most commercial implementations. Combining open hardware platforms with open source software can yield significant “bang for the buck” by providing custom, robust solutions for very low cost.

This is not to say that commercial equipment is inferior to a do-it-yourself solution. By providing so-called “turn-key solutions”, manufacturers not only save development time, but they can also allow relatively unskilled people to install and maintain equipment. The chief strengths of commercial solutions are that they provide ***support*** and a (usually limited) ***equipment warranty***. They also provide a ***consistent platform*** that tends to lead to very stable, often interchangeable network installations.

If a piece of equipment simply doesn’t work or is difficult to configure or troubleshoot, a good manufacturer will assist you. Should the equipment fail in normal use (barring extreme damage, such as a lightning strike) then the manufacturer will typically replace it. Most will provide these services for a limited time as part of the purchase price, and many offer support and warranty for an extended period for a monthly fee. By providing a consistent platform, it is simple to keep spares on hand and simply "swap out" equipment that fails in the field, without the need for a technician to configure equipment on-site. Of course, all of this comes at comparatively higher initial cost for the equipment compared to off-the-shelf components.

From a network architect's point of view, the three greatest hidden risks when choosing commercial solutions are ***vendor lock-in***, ***discontinued product lines***, and ***ongoing licensing costs***.

It can be costly to allow the lure of ill-defined new "features" drive the development of your network. Manufacturers will frequently provide features that are incompatible with their competition by design, and then issue marketing materials to convince you that you simply cannot live without them (regardless of whether the feature contributes to the solution of your communications problem). As you begin to rely on these features, you will likely decide to continue purchasing equipment from the same manufacturer in the future. This is the essence of vendor lock-in. If a large institution uses a significant amount of proprietary equipment, it is unlikely that they will simply abandon it to use a different vendor. Sales teams know this (and indeed, some rely on it) and use vendor lock-in as a strategy for price negotiations.

When combined with vendor lock-in, a manufacturer may eventually decide to discontinue a product line, regardless of its popularity. This ensures that customers, already reliant on the manufacturer's proprietary features, will purchase the newest (and nearly always more expensive) model. The long term effects of vendor lock-in and discontinued products are difficult to estimate when planning a networking project, but should be kept in mind.

Finally, if a particular piece of equipment uses proprietary computer code, you may need to license use of that code on an ongoing basis. The cost of these licenses may vary depending on features provided, number of users, connection speed, or other factors. If the license fee is unpaid, some equipment is designed to simply stop working until a valid, paid-up license is provided! Be sure that you understand the terms of use for any equipment you purchase, including ongoing licensing fees.

By using generic equipment that supports open standards and open source software, you can avoid some of these pitfalls. For example, it is very difficult to become locked-in to a vendor that uses open protocols (such as TCP/IP over 802.11a/b/g). If you encounter a problem with the equipment or the vendor, you can always purchase equipment from a different vendor that will interoperate with what you have already purchased. It is for these reasons that we recommend using proprietary protocols and licensed spectrum **only** in cases where the open equivalent (such as 802.11a/b/g) is not technically feasible.

Likewise, while individual products can always be discontinued at any time, you can limit the impact this will have on your network by using generic components. For example, a particular motherboard may become unavailable on the market, but you may have a number of PC motherboards on hand that will perform effectively the same task. We will see some examples of how to use these generic components to build a complete wireless node later in this chapter.

Obviously, there should be no ongoing licensing costs involved with open source software (with the exception of a vendor providing extended support or some other service, without charging for the use of the software itself). There have occasionally been vendors who capitalize on the gift that open source programmers have given to the world by offering the code for sale on an ongoing licensed basis, thereby violating the terms of distribution set forth by the original authors. It would be wise to avoid such vendors, and to be suspicious of claims of "free software" that come with an ongoing license fee.

The disadvantage of using open source software and generic hardware is clearly the question of support. As problems with the network arise, you will need to solve those problems for yourself. This is often accomplished by consulting free online resources and search engines, and applying code patches directly. If you do not have team members who are competent and dedicated to designing a solution to your communications problem, then it can take a considerable amount of time to get a network project off the ground. Of course, there is never a guarantee that simply "throwing money at the problem" will solve it either. While we provide many examples of how to do much of the work yourself, you may find this work very challenging. You will need to find the balance of commercial solutions and the do-it-yourself approach that works for project.

In short, always define the scope of your network first, identify the resources you can bring to bear on the problem, and allow the selection of equipment to naturally emerge from the results. Consider commercial solutions as well as open components, while keeping in mind the long-term costs of both.

When considering which equipment to use, always remember to compare the expected useful distance, reliability, and throughput, in addition to the price. Be sure to include any ongoing license fees when calculating the overall cost of the equipment. And finally, make sure that the radios you purchase operate in an unlicensed band where you are installing them, or if you must use licensed spectrum, that you have budget and permission to pay for the appropriate licenses.

Professional lightning protection

Lightning is a natural predator of wireless equipment. There are two different ways lightning can strike or damage equipment: direct hits or induction hits. Direct hits happen when lightning actually hits the tower or antenna. Induction hits are caused when lightning strikes near the tower. Imagine a negatively charged lightning bolt. Since like charges repel each other, that bolt will cause the electrons in the cables to move away from the strike, creating current on the lines. This can be much more current than the sensitive radio equipment can handle. Either type of strike will usually destroy unprotected equipment.

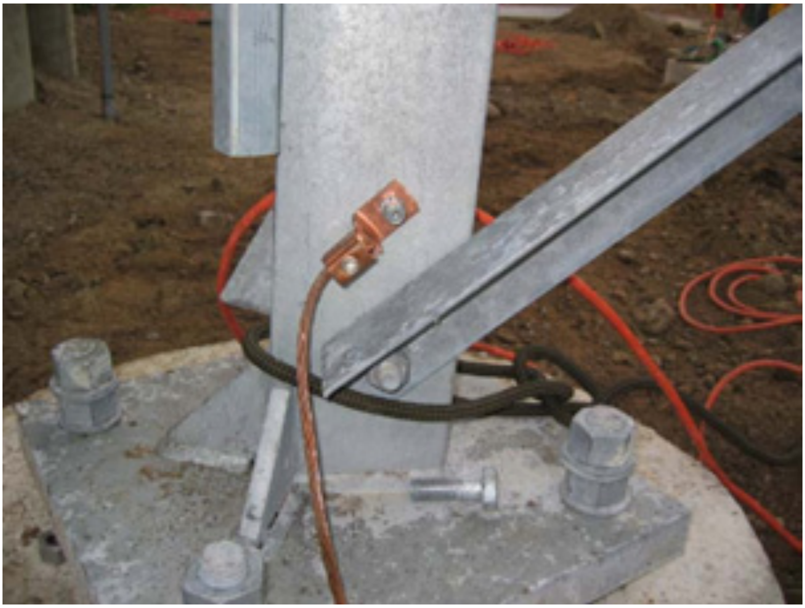


Figure 5.2: A tower with a heavy copper grounding wire.

Protecting wireless networks from lightning is not an exact science, and there is no guarantee that a lightning strike will not happen, even if every single precaution is taken. Many of the methods used will help prevent both direct and induction strikes. While it is not necessary to use every single lightning protection method, using more methods will help further protect the equipment. The amount of lightning historically observed within a service area will be the biggest guide to how much needs to be done.

Start at the very bottom of the tower. Remember, the bottom of the tower is below the ground. After the tower foundation is laid, but before the hole is backfilled, a ring of heavy braided ground wire should have been installed with the lead extending above ground surfacing near a tower leg. The wire should be American Wire Gauge (AWG) #4 or thicker. In addition, a backup ground or earthing rod should be driven into the ground, and a ground wire run from the rod to the lead from the buried ring.

It is important to note that not all steel conducts electricity the same way. Some types of steel act as better electrical conductors then others, and different surface coatings can also affect how tower steel handles electrical current. Stainless steel is one of the worst conductors, and rust proof coatings like galvanizing or paint lessen the conductivity of the steel. For this reason, a braided ground wire is run from the bottom of the tower all the way to the top. The bottom needs to be properly attached to the leads from both the ring and the backup ground rod. The top of the tower should have a lightning rod attached, and the top of that needs to be pointed. The finer and sharper the point, the more effective the rod will be. The braided ground wire from the bottom needs to be terminated at this grounding rod. It is very important to be sure that the ground wire is connected to the actual metal. Any sort of coating, such as paint, must be removed before the wire is attached. Once the connection is made, the exposed area can be repainted, covering the wire and connectors if necessary to save the tower from rust and other corrosion.

The above solution details the installation of the basic grounding system. It provides protection for the tower itself from direct hits, and installs the base system to which everything else will connect.

The ideal protection for indirect induction lightning strikes are gas tube arrestors at both ends of the cable. These arrestors need to be grounded directly to the ground wire installed on the tower if it is at the high end. The bottom end needs to be grounded to something electrically safe, like a ground plate or a copper pipe that is consistently full of water. It is important to make sure that the outdoor lightning arrestor is weatherproofed. Many arresters for coax cables are weatherproofed, while many arresters for CAT5 cable are not.

In the event that gas arrestors are not being used, and the cabling is coax based, then attaching one end of a wire to the shield of the cable and the other to the ground wire installed on the towers will provide some protection. This can provide a path for induction currents, and if the charge is weak enough, it will not affect the conductor wire of the cable. While this method is by no means as good of protection as using the gas arrestors, it is better then doing nothing at all.

Building an access point from a PC

Unlike consumer operating systems (such as Microsoft Windows), the GNU/Linux operating system gives a network administrator the potential for full access to the networking stack. One can access and manipulate network packets at any level from the data-link layer through the application layer. Routing decisions can be made based on any information contained in a network packet, from the routing addresses and ports to the contents of the data segment. A Linux-based access point can act as a router, bridge, firewall, VPN concentrator, application server, network monitor, or virtually any other networking role you can think of. It is freely available software, and requires no licensing fees. GNU/Linux is a very powerful tool that can fill a broad variety of roles in a network infrastructure.

Adding a wireless card and Ethernet device to a PC running Linux will give you a very flexible tool that can help you deliver bandwidth and manage your network for very little cost. The hardware could be anything from a recycled laptop or desktop machine to an embedded computer, such as a Linksys WRT54G or Metrix networking kit.

In this section we will see how to configure Linux in the following configurations:

- As a wireless access point with Masquerading/NAT and a wired connection to the Internet (also referred to as a wireless gateway).
- As a wireless access point that acts as a transparent bridge. The bridge can be used either as a simple access point, or as a repeater with 2 radios.

Consider these recipes as a starting point. By building on these simple examples, you can create a server that fits precisely into your network infrastructure.

Prerequisites

Before proceeding, you should already be familiar with Linux from a users perspective, and be capable of installing the Gnu/Linux distribution of your choice. A basic understanding of the command line interface (terminal) in Linux is also required.

You will need a computer with one or more wireless cards already installed, as well as a standard Ethernet interface. These examples use a specific card and driver, but there are a number of different cards that should work equally well. Wireless cards based on the Atheros and Prism chipsets work particularly well. These examples are based on Ubuntu Linux version 5.10 (Breezy Badger), with a wireless card that is supported by the HostAP or MADWiFi drivers. For more information about these drivers, see *http://hostap.epitest.fi/* and *http://madwifi.org/* .

The following software is required to complete these installations. It should be provided in your Linux distribution:

- Wireless Tools (iwconfig, iwlist commands)
- iptables firewall
- dnsmasq (caching DNS server and DHCP server)

The CPU power required depends on how much work needs to be done beyond simple routing and NAT. For many applications, a 133MHz 486 is perfectly capable of routing packets at wireless speeds. If you intend to use a lot of encryption (such as WEP or a VPN server), then you will need something faster. If you also want to run a caching server (such as Squid) then you will need a computer with plenty of fast disk space and RAM. A typical router that is only performing NAT will operate will with as little as 64MB of RAM and storage.

When building a machine that is intended to be part of your network infrastructure, keep in mind that hard drives have a limited lifespan compared to most other components. You can often use solid state storage, such as a flash disk, in place of a hard drive. This could be a USB flash drive (assuming your PC will boot from USB), or a Compact Flash card using a CF to IDE adapter. These adapters are quite inexpensive, and will make a CF card appear act like standard IDE hard drive. They can be used in any PC that supports IDE hard drives. Since they have no moving parts, they will operate for many years through a much wider range of temperatures than a hard disk will tolerate.

Scenario 1: Masquerading access point

This is the simplest of the scenarios, and is especially useful in situations where you want a single access point for an office setting. This is easiest in a situation where:

1. There is an existing dedicated firewall and gateway running Linux, and you just want to add a wireless interface.

2. You have an old refurbished computer or laptop available, and prefer to use that as an access point.

3. You require more power in terms of monitoring, logging and/or security than most commercial access points provide, but don't want to splurge on an enterprise access point.

4. You would like a single machine to act as 2 access points (and firewall) so that you can offer both a secure network access to the intranet, as well as open access to guests.

Initial setup

Start of with an already configured computer running GNU/Linux. This could be an Ubuntu Server installation, or Fedora Core. The computer must have at least 2 interfaces for this to work, and at least one of these interfaces should be wireless. The rest of this description assumes that your cabled Ethernet port (eth0) is connected to the Internet, and that there is a wireless interface (wlan0) that will provide the access point functionality.

To find out if your chipset supports master mode, try the following command as root:

```
# iwconfig wlan0 mode Master
```

...replacing wlan0 with the name of your interface.

If you get an error message, then your wireless card doesn't support access point mode. You can still try the same setup in Ad-hoc mode, which is supported by all chipsets. This requires that you to set all the laptops that are connecting to this "access point" into Ad-hoc mode as well, and may not work quite the way you are expecting. It is usually better to find a wireless card that will support AP mode. See the HostAP and MADWiFi websites mentioned earlier for a list of supported cards.

Before continuing, make sure dnsmasq is installed on your machine. You can use the graphical package manager of your distribution to install it. In Ubuntu you can simply run the following as root:

```
# apt-get install dnsmasq
```

Setting up the interfaces

Set up your server so that eth0 is connected to the Internet. Use the graphical configuration tool that came with your distribution.

If your Ethernet network uses DHCP, you could try the following command as root:

```
# dhclient eth0
```

You should receive an IP address and default gateway. Next, set your wireless interface to Master mode and give it a name of your choice:

```
# iwconfig wlan0 essid “my network” mode Master enc off
```

The **`enc off`** switch turns off WEP encryption. To enable WEP, add a hex-key string of the correct length:

```
# iwconfig wlan0 essid "my network" mode Master enc 1A2B3C4D5E
```

Alternately, you can use a readable string by starting with "s:"

```
# iwconfig wlan0 essid "my network" mode Master enc "s:apple"
```

Now give your wireless interface an IP address in a private subnet, but make sure it is not the same subnet as that of your Ethernet adapter:

```
# ifconfig wlan0 10.0.0.1 netmask 255.255.255.0 broadcast 10.0.0.255 up
```

Setting up masquerading in the kernel

In order for us to be able to translate addresses between the two interfaces on the computer, we need to enable masquerading (NAT) in the linux kernel. First we load the relevant kernel module:

```
# modprobe ipt_MASQUERADE
```

Now we will flush all existing firewall rules to ensure that the firewall is not blocking us from forwarding packets between the two interfaces. If you have an existing firewall running, make sure you know how to restore the existing rules later before proceeding.

```
# iptables -F
```

Enable the NAT functionality between the two interfaces

```
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Finally we need to enable the kernel to forward packets between interfaces:

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

On Debian-based Linux distributions such as Ubuntu, this change can also be made by editing the file **/etc/network/options**, and be sure that **`ip_forward`** is set to **`yes`**:

```
ip_forward=yes
```

and then restarting the network interfaces with:

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

```
or
```

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

Setting up the DHCP server

At this point we actually should have a working access point. It can be tested by connecting to the wireless network "my network" with a separate machine and giving that machine an address in the same address range as our wireless interface on the server (10.0.0.0/24 if you followed the examples). If you have enabled WEP, be sure to use the same key that you specified on the AP.

In order to make it easier for people to connect to the server without knowing the IP address range, we will set up a DHCP server to automatically hand out addresses to wireless clients.

We use the program dnsmasq for this purpose. As the name indicates, it provides a caching DNS server as well as a DHCP server. This program was developed especially for use with firewalls performing NAT. Having a caching DNS server is especially helpful if your Internet connection is a high-latency and/or low-bandwidth connection, such as a VSAT or dial-up. It means that many DNS queries can be resolved locally, saving a lot of traffic on the Internet connection, and also making the connection feel noticeably faster for those connecting.

Install dnsmasq with your distributions package manager. If dnsmasq is not available as a package, download the source code and install it manually. It is available from *http://www.thekelleys.org.uk/dnsmasq/doc.html*.

All that is required for us to run dnsmasq is to edit a few lines of the dnsmasq configuration file, **`/etc/dnsmasq.conf`**.

The configuration file is well commented, and has many options for various types of configuration. To get the basic DHCP server up and running we just need to uncomment and/or edit two lines.

Find the lines that starts:

```
interface=
```

...and make sure it reads:

```
interface=wlan0
```

...changing wlan0 to match name of your wireless interface. Then find the line that starts with:

```
#dhcp-range=
```

Uncomment the line and edit it to suit the match addresses being used, i.e.

```
dhcp-range=10.0.0.10,10.0.0.110,255.255.255.0,6h
```

Then save the file and start dnsmasq:

```
# /etc/init.d/dnsmasq start
```

That's it, you should now be able to connect to the server as an access point, and get an IP address using DHCP. This should let you connect to the Internet through the server.

Adding extra security: Setting up a Firewall

Once this is set up and tested, you can add extra firewall rules using whatever firewall tool is included in your distribution. Some typical front-ends for setting up firewall rules include:

- ***firestarter*** - a graphical client for Gnome, which requires that your server is running Gnome
- ***knetfilter*** – a graphical client for KDE, which requires that your server is running KDE
- ***Shorewall*** – a set of scripts and configuration files that will make it easier to setup an iptables firewall. There are also frontends for shorewall, such as webmin-shorewall
- ***fwbuilder*** - a powerful, but slightly complex graphical tool that will let you create iptables scripts on a machine separate from your server, and then transfer them to the server later. This does not require you to be running a graphical desktop on the server, and is a strong option for the security conscious.

Once everything is configured properly, make sure that all settings are reflected in the system startup scripts. This way, your changes will continue to work should the machine need to be rebooted.

Scenario 2: Transparent Bridging access point

This scenario can either be used for a two-radio repeater, or for an access point connected to an Ethernet. We use a bridge instead of routing when we want both interfaces on the access point to share the same subnet. This can be particularly useful in networks with multiple access points where we prefer to have a single, central firewall and perhaps authentication server. Because all clients share the same subnet they, can easily be managed with a single DHCP server and firewall without the need for DHCP relay.

For example, you could setup a server as the first scenario, but use two wired Ethernet interfaces instead of one wired and one wireless. One interface would be your Internet connection, and the other would connect to a switch. Then connect as many access points as you require to the same switch, set them up as transparent bridges, and everyone will pass through the same firewall and use the same DHCP server.

The simplicity of bridging comes at a cost of efficiency. Since all clients share the same subnet, broadcast traffic will be repeated throughout the network. This is usually fine for small networks, but as the number of clients increases, more wireless bandwidth will be wasted on broadcast network traffic.

Initial setup

The initial setup for a bridging access point is similar to that of a masquerading access point, without the requirement of dnsmasq. Follow the initial setup instructions from the previous example.

In addition, the ***bridge-utils*** package is required for bridging. This package exists for Ubuntu and other Debian-based distributions, as well as for Fedora Core. Make sure it is installed and that the command **`brctl`** is available before proceeding.

Setting up the Interfaces

On Ubuntu or Debian the network interfaces are configured by editing the file **`/etc/network/interfaces`**.

Add a section like the following, but change the names of interfaces and the IP addresses accordingly. The IP address and netmask must match that of your existing network. This example assumes you are building a wireless repeater with two wireless interfaces, wlan0 and wlan1. The wlan0 interface will be a client to the “office” network, and wlan1 will create a network called “repeater”.

Add the following to **`/etc/network/interfaces`**:

```
auto br0
iface br0 inet static
  address 192.168.1.2
  network 192.168.1.0
  netmask 255.255.255.0
  broadcast 192.168.1.255
  gateway 192.168.1.1
  pre-up ifconfig wlan 0 0.0.0.0 up
  pre-up ifconfig wlan1 0.0.0.0 up
  pre-up iwconfig wlan0 essid “office” mode Managed
  pre-up iwconfig wlan1 essid “repeater” mode Master
  bridge_ports wlan0 wlan1
  post-down ifconfig wlan1 down
  post-down ifconfig wlan0 down
```

Comment out any other sections in the file that refer to wlan0 or wlan1 to make sure that they don't interfere with our setup.

This syntax for setting up bridges via the **interfaces** file is specific to Debian-based distributions, and the details of actually setting up the bridge are handled by a couple of scripts: **/etc/network/if-pre-up.d/bridge** and **/etc/network/if-post-down.d/bridge**. The documentation for these scripts is found in **/usr/share/doc/bridge-utils/**.

If those scripts don't exist on your distribution (such as Fedora Core), here is an alternative setup for **/etc/network/interfaces** which will achieve the same thing with only marginally more hassle:

```
iface br0 inet static
  pre-up ifconfig wlan 0 0.0.0.0 up
  pre-up ifconfig wlan1 0.0.0.0 up
  pre-up iwconfig wlan0 essid “office” mode Managed
  pre-up iwconfig wlan1 essid “repeater” mode Master
  pre-up brctl addbr br0
  pre-up brctl addif br0 wlan0
  pre-up brctl addif br0 wlan1
  post-down ifconfig wlan1 down
  post-down ifconfig wlan0 down
  post-down brctl delif br0 wlan0
  post-down brctl delif br0 wlan1
  post-down brctl delbr br0
```

Starting the bridge

Once the bridge is defined as an interface, starting the bridge is as simple as typing:

```
# ifup -v br0
```

The “-v” means verbose output and will give you information to what is going on.

On Fedora Core (i.e. non-debian distributions) you still need to give your bridge interface an ip address and add a default route to the rest of the network:

```
#ifconfig br0 192.168.1.2 netmask 255.255.255.0 broadcast 192.168.1.255
#route add default gw 192.168.1.1
```

You should now be able to connect a wireless laptop to this new access point, and connect to the Internet (or at least to the rest of your network) through this box.

Use the **brctl** command to see what your bridge is doing:

```
# brctl show br0
```

Scenario 1 & 2 the easy way

Instead of setting up your computer as an access point from scratch, you may wish to use a dedicated Linux distribution that is specially tailored for this purpose. These distributions can make the job as simple as booting from a particular CD on a computer with a wireless interface. See the following section, “Wireless-friendly operating systems” for more information.

As you can see, it is straightforward to provide access point services from a standard Linux router. Using Linux gives you significantly more control over how packets are routed through your network, and allows for features that simply aren’t possible on consumer grade access point hardware.

For example, you could start with either of the above two examples and implement a private wireless network where users are authenticated using a standard web browser. Using a captive portal such as Chillispot, wireless users can be checked against credentials in an existing database (say, a Windows domain server accessible via RADIUS). This arrangement could allow for preferential access to users in the database, while providing a very limited level of access for the general public.

Another popular application is the prepaid commercial model. In this model, users must purchase a ticket before accessing the network. This ticket provides a password that is valid for a limited amount of time (typically one day). When the ticket expires, the user must purchase another. This ticketing feature is only available on relatively expensive commercial networking equipment, but can be implemented using free software such as Chillispot and phpMyPrePaid. We will see more about captive portal technology and ticketing systems in the **Authentication** section in **Chapter 6**.

Wireless-friendly operating systems

There are a number of open source operating system that provide useful tools for working with wireless networks. These are intended to be used on repurposed PCs or other networking hardware (rather than on a laptop or server) and are fine-tuned for building wireless networks. Some of these projects include:

- **Freifunk**. Based on the OpenWRT project (*http://openwrt.org/*), the Freifunk firmware brings easy OLSR support to MIPS-based consumer access points, such as the Linksys WRT54G / WRT54GS / WAP54G, Siemens SE505, and others. By simply flashing one of these APs with the Freifunk firmware, you can rapidly build a self-forming OLSR mesh. Freifunk is not currently available for x86 architecture machines. It is maintained by Sven Ola of the Freifunk wireless group in Berlin. You can download the firmware from *http://www.freifunk.net/wiki/FreifunkFirmware* .

- **Pyramid Linux**. Pyramid is a Linux distribution for use on embedded platforms that evolved out of the venerable Pebble Linux platform. It supports several different wireless cards, and has a simple web interface for configuring networking interfaces, port forwarding, WifiDog, and OLSR. Pyramid is distributed and maintained by Metrix Communication LLC, and is available at *http://pyramid.metrix.net/*.
- **m0n0wall**. Based on FreeBSD, m0n0wall is a very tiny but complete firewall package that provides AP services. It is configured from a web interface and the entire system configuration is stored in a single XML file. Its tiny size (less than 6MB) makes it attractive for use in very small embedded systems. Its goal is to provide a secure firewall, and as such does not include userspace tools (it is not even possible to log into the machine over the network). Despite this limitation, it is a popular choice for wireless networkers, particularly those with a background in FreeBSD. You can download m0n0wall from *http://www.m0n0.ch/ .*

All of these distributions are designed to fit in machines with limited storage. If you are using a very large flash disk or hard drive, you can certainly install a more complete OS (such as Ubuntu or Debian) and use the machine as a router or access point. It will likely take a fair amount of development time to be sure all needed tools are included, without installing unnecessary packages. By using one of these projects as a starting point for building a wireless node, you will save yourself considerable time and effort.

The Linksys WRT54G

One of the most popular consumer access points currently on the market is the Linksys WRT54G. This access point features two external RP-TNC antenna connectors, a four port Ethernet switch, and an 802.11b/g radio. It is configured through a simple web interface. While it is not designed as an outdoor solution, it can be installed in a large sprinkler box or plastic tub for relatively little cost. As of this writing, the WRT54G sells for about $60.

Back in 2003, network hackers realized that the firmware that shipped with the WRT54G was actually a version of Linux. This led to a tremendous interest in building custom firmware that extended the capabilities of the router significantly. Some of these new features include client radio mode support, captive portals, and mesh networking. Some popular alternative firmware packages for the WRT54G are DD-Wrt (*http://www.dd-wrt.com/*), OpenWRT (*http://openwrt.org/*), Tomato (*http://www.polarcloud.com/tomato*) and Freifunk (*http://www.freifunk.net/*).

Unfortunately, in the fall of 2005, Linksys released version 5 of the WRT54G. This hardware revision eliminated some RAM and flash storage on the motherboard, making it very difficult to run Linux (it ships with VxWorks, a much smaller operating system that does not allow easy customization). Linksys also released the WRT54GL, which is essentially the WRT54G v4 (which runs Linux) with a slightly bigger price tag.

A number of other Linksys access points also run Linux, including the WRT54GS and WAP54G. While these also have relatively low price tags, the hardware specifications may change at any time. It is difficult to know which hardware revision is used without opening the packaging, making it risky to purchase them at a retail store and practically impossible to order online. While the WRT54GL is guaranteed to run Linux, Linksys has made it known that it does not expect to sell this model in large volume, and it is unclear how long it will be offered for sale.

Fortunately, wireless hackers have now been able to install custom firmware on the notoriously difficult WRT54G version 5 and 6, and the latest revisions as well(v7 and v8). For details on getting alternate firmware installed on a v5 or v6 access point see: *http://www.scorpiontek.org/portal/content/view/27/36/*

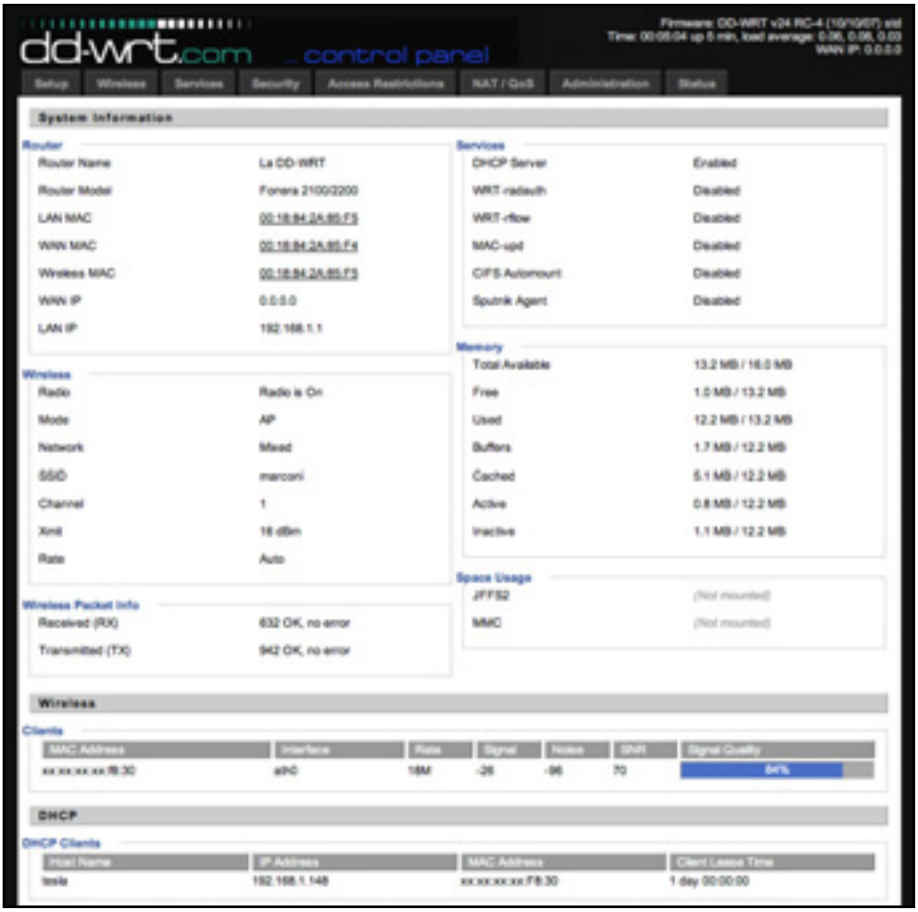
For more information about the current state of Linksys wireless router hacking, see *http://linksysinfo.org/*

DD-WRT

One popular alternate firmware for the Linksys family of access point hardware is DD-WRT (*http://www.dd-wrt.com/*). It includes several useful features, including radio client mode, adjustable transmission power, various captive portals, QoS support, and much more. It uses an intuitive web-based configuration tool (unencrypted or via HTTPS), and also provides SSH and telnet access.

Several versions of the firmware are available from the DD-WRT website. The general procedure for upgrading is to download the version of the firmware appropriate for your hardware, and upload it via the router's "firmware update" feature. Specific installation details vary according to the hardware version of your router. In addition to Linksys hardware, DD-WRT will run on Buffalo, ASUS, the La Fonera, and other access points.

For specific instructions for your hardware, see the installation guide on the DD-WRT wiki at *http://www.dd-wrt.com/wiki/index.php/Installation.* The default login for a fresh DD-WRT installation is **root** with the password **admin**.



Firmware: DD-WRT v24 RC-4 (10/10/07) std
Time: 00:05:04 up 5 min, load average: 0.06, 0.06, 0.00
WAN IP: 0.0.0.0

System Information

Router

Router Name	La DD-WRT
Router Model	Fonera 2100/2200
LAN MAC	00:18:84:2A:85:F5
WAN MAC	00:18:84:2A:85:F4
Wireless MAC	00:18:84:2A:85:F5
WAN IP	0.0.0.0
LAN IP	192.168.1.1

Wireless

Radio	Radio is On
Mode	AP
Network	Mixed
SSID	marconi
Channel	1
Xmit	16 dBm
Rate	Auto

Wireless Packet Info

Received (RX)	632 OK, no error
Transmitted (TX)	942 OK, no error

Services

DHCP Server	Enabled
WRT-radauth	Disabled
WRT-rflow	Disabled
MAC-upd	Disabled
CIFS Automount	Disabled
Sputnik Agent	Disabled

Memory

Total Available	13.2 MB / 16.0 MB
Free	1.0 MB / 13.2 MB
Used	12.2 MB / 13.2 MB
Buffers	1.7 MB / 12.2 MB
Cached	5.1 MB / 12.2 MB
Active	0.8 MB / 12.2 MB
Inactive	1.1 MB / 12.2 MB

Space Usage

JFFS2	(Not mounted)
MMC	(Not mounted)

Wireless

Clients

MAC Address	Interface	Rate	Signal	Noise	SNR	Signal Quality
xx:xx:xx:xx:xx:f8:30	ath0	18M	-26	-96	70	84%

DHCP

DHCP Clients

Host Name	IP Address	MAC Address	Client Lease Time
tesla	192.168.1.148	xx:xx:xx:xx:xx:F8:30	1 day 00:00:00

Figure 5.3: The DD-WRT (v23) control panel