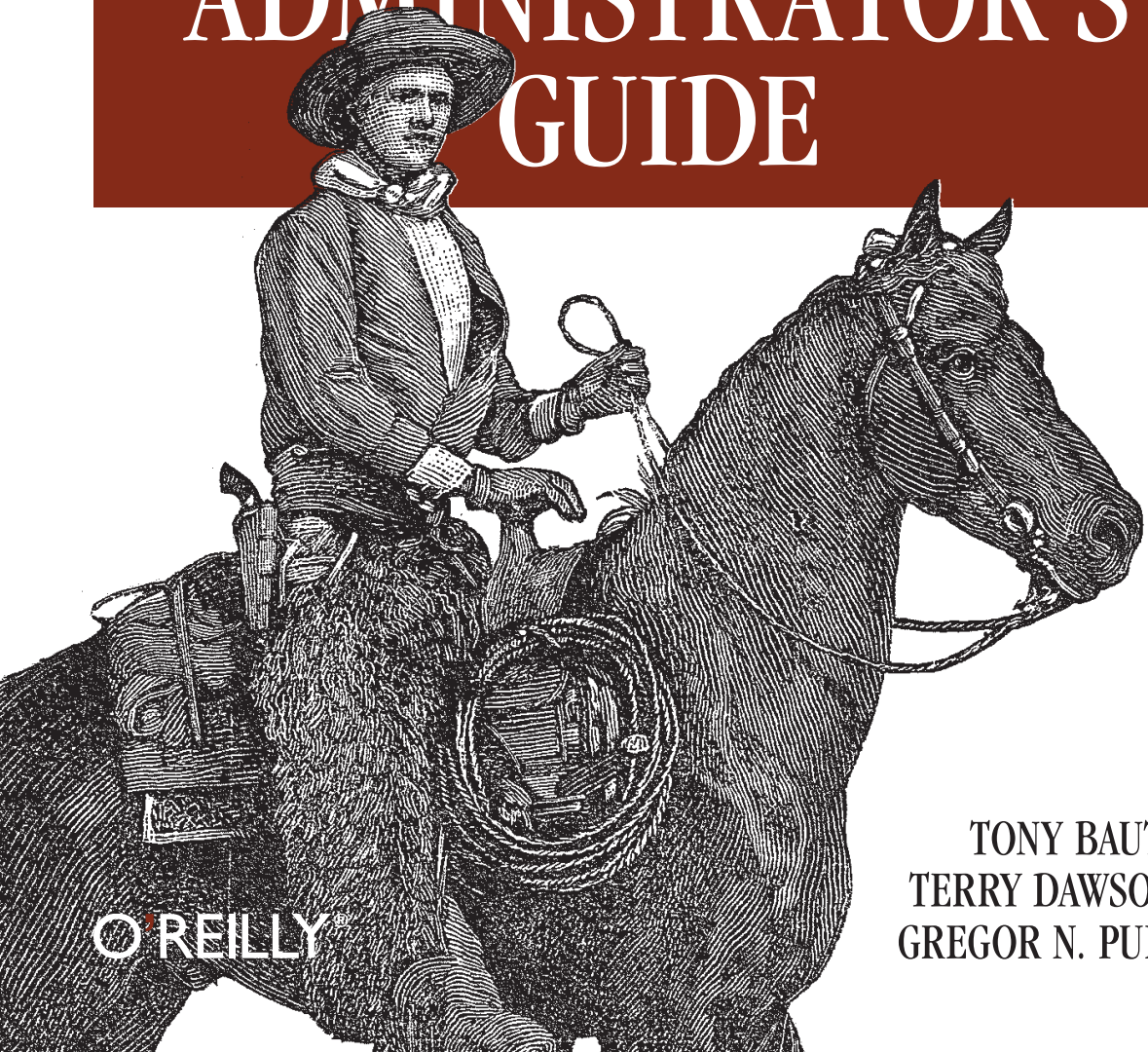
INFRASTRUCTURE, SERVICES, AND SECURITY

3rd Edition

# LINUX NETWORK ADMINISTRATOR'S GUIDE

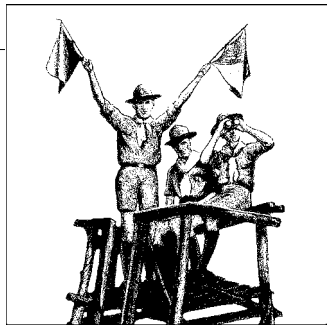


TONY BAUTTS,
TERRY DAWSON &
GREGOR N. PURDY

O'REILLY®

CHAPTER 18

# Wireless Networking



Wireless networking is a promising and increasingly popular technology, offering a wide range of benefits compared to traditional wired technology. These advantages range from increased convenience to users and decreased deployment cost to ease of network installation. A new wireless deployment can save substantial amounts of money since there is no need for additional cables, jacks, or network switches. Adding new users to a network can be as easy as plugging in a wireless card and powering up a machine. Wireless networking has also been used to deliver network access to areas where there is little or no traditional network infrastructure.

Perhaps the biggest impact of wireless networking can be seen within its widespread acceptance among consumers. The most obvious example of this popularity can be seen with new laptop systems, where nearly every unit is shipped with integrated 802.11b or g. The practical benefits have consequently insured good sales, allowing manufacturers to lower the equipment costs. At the time of this writing, the price of client wireless cards is comparable to that of traditional Ethernet adapter cards.

These benefits, however, do not come without some disadvantages, the most severe of these being the security issues.

## History

Wireless LANs are based on spread spectrum technology, initially developed for military communications by the U.S. Army during World War II. Military technicians considered spread spectrum desirable because it was more resistant to jamming. Other advances at this time allowed an increase in the radio data rate. After 1945, commercial enterprises began to expand on this technology, realizing its potential benefits to consumers.

Spread spectrum technology evolved into the beginnings of the modern wireless LAN in 1971 with a University of Hawaii project called AlohNet. This project

allowed seven computers around the various islands to communicate bidirectionally with a central hub on Oahu.

The university research on AlohNet paved the way for the first generation of modern wireless networking gear, which operated at the 901–928 MHz frequency range. Primarily used by the military, this phase of wireless development saw only limited consumer use, due to crowding within this frequency and the relatively low speed.

From this point, the 2.4 GHz frequency was defined for unlicensed use, so wireless technology began to emerge in this range and the 802.11 specification was established. This specification evolved into the widely accepted 802.11b standard, and continues to evolve into faster, more secure implementations of the technology.

## The Standards

The standards based around wireless networking for PCs are established by the Institute of Electrical and Electronics Engineers (IEEE). LAN/MAN technology has been broadly assigned number 802, which is then broken down into working groups. Some of the most active wireless working groups include 802.15, designed for wireless personal area networks (Bluetooth), 802.16 which defines support for broadband wireless systems, and finally, 802.11, assigned to wireless LAN technology. Within the 802.11 definition, there are more specific definitions that are assigned letters. Here is a list of the most important 802.11 wireless LAN definitions:

*802.11a*
: This definition provides wireless access on the 5 GHz band. It offers speeds of up to 54 MBps, but has not caught on, perhaps due to relatively higher priced equipment and short range.

*802.11b*
: This is still the standard to which most people refer when talking about wireless networking. It establishes 11 MBps speeds on the 2.4 GHz band, and can have a range extending more than 500 meters.

*802.11g*
: This standard has been established to provide higher data rates within the 2.4 GHz band and provides added security with the introduction of WiFi Protected Access, or WPA. 802.11g devices are now being deployed in place of 802.11b devices and have nearly reached mainstream acceptance.

*802.11i*
: While still in the development phase, this standard seeks to resolve many of the security issues that have plagued 802.11b and provide a more robust system of authentication and encryption. At the time of this writing, the specification has not been finalized.

*802.11n*

802.11n is being touted as the high-speed answer to current wireless network speed shortcomings. With an operational speed of 100 Mbps, it will roughly double existing wireless transfer speeds, while maintaining backward compatibility with b and g. At the time of this writing, the specification is not complete; however, several vendors have released "pre-n" products, based on the early drafts of the specification.

# 802.11b Security Concerns

When the IEEE created the 802.11b standard, they realized that the open nature of wireless networking required some kind of data integrity and protection mechanism and thus created Wired Equivalent Privacy (WEP). Promised by the standard to provide encryption at the 128-bit level, users were supposed to be able to enjoy the same levels of privacy found on a traditional wired network.

Hopes for this kind of security, however, were quickly dashed. In a paper called "Weaknesses in the Key Scheduling Algorithm of RC4" by Scott Fluhrer, Itsik Mantin, and Adi Shamir, the weaknesses in the key generation and implementation of WEP were described in great detail. Although this development was a theoretical attack when the paper was written, a student at Rice University, Adam Stubblefield, brought it into reality and created the first WEP attack. Although he has never made his tools public, there are now many similar tools for Linux that will allow attackers to break WEP, making it an untrustworthy security tool.

Still, it should be acknowledged that staging a WEP attack requires a considerable amount of time. The success of the attack relies upon the amount of encrypted data the attacker has captured. Tools such as AirSnort require approximately 5 to 10 million encrypted packets. A busy wireless LAN, which is constantly seeing the maximum amount of traffic, can still take as long as 10 hours to crack. Since most networks do not run at capacity for this long, it can be expected that the attack would take considerably longer, stretching out to a few days for smaller networks.

However, for true protection from malicious behavior and eavesdropping, a VPN technology should be used, and wireless networks should never be directly connected to internal, trusted networks.

## Hardware

Different manufacturers use a slightly different architecture to provide 802.11b functionality. There are two major chipset manufacturers, Hermes and Prism, and within each, hardware manufacturers have made modifications to increase security or speed. For example, the USRobotics equipment, based on the Prism chipset, now offers 802.11b at 22 MBps, but it will not operate at these speeds without the DLink 802.11b 22 MBps hardware. However, they are interoperable at the 11 MBps speed.

### 801.11g versus 802.11b on Linux

Due to new chipsets and manufacturer differences, 802.11g support on Linux has been somewhat difficult. At the time of writing, support for 802.11g devices under Linux was still emerging and was not yet as stable and robust as the 802.11b support. For this reason, this chapter focuses on 802.11b drivers and support. Mainstream Linux support for g devices, however, is not far off. With the work of groups such as Prism54.org, which is developing g drivers, and Intel's announcement that it will release drivers for its Centrino chipset, full support is less than a year away.

### Chipsets

As mentioned, there are two main 802.11b chipsets, Hermes and Prism. While initially Hermes cards were predominant due to the popularity of Lucent WaveLAN (Orinoco) cards, a majority of card makes today use Prism's prism2 chipset. Some well-known Prism cards are those from D-Link, Linksys, and USR. You'll get roughly the same performance with either card, and they are interoperable when operating within the 802.11b standard, meaning that you can connect a Lucent wireless card to a D-Link access point, and vice versa. A brief listing of major card manufacturers and their chipsets follows. If your card is not listed, check your operation manual or the vendor web site.

- Hermes chipset cards:

  Lucent Orinoco Silver and Gold Cards
  Gateway Solo
  Buffalo Technologies

- Prism 2 Chipset cards:

  Addtron
  Belkin
  Linksys
  D-Link
  ZoomMax

## Client Configuration

802.11b networks can be configured to operate in several different modes. The two main types you're likely to encounter are infrastructure mode (sometimes referred to as managed mode) and ad-hoc. Infrastructure mode is the most common and uses a hub-and-spoke architecture in which multiple clients connect to a central access point, as shown in Figure 18-1. The ad-hoc wireless network mode is a peer-to-peer network, in which clients connect to each other directly, as shown in Figure 18-2. Infrastructure mode deployments are an effective means to replace wires on a traditional network, making them ideal for office environments where wireless clients need access to servers that are connected to the wired network. Ad-hoc networks are

beneficial to those who simply wish to transfer files between PCs, or do not require access to any servers outside of the wireless network.

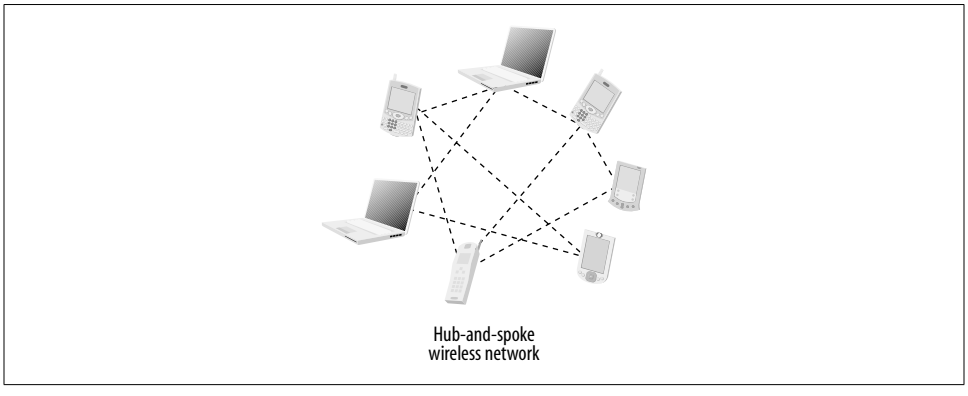



*Figure 18-1. Hub-and-spoke wireless network*

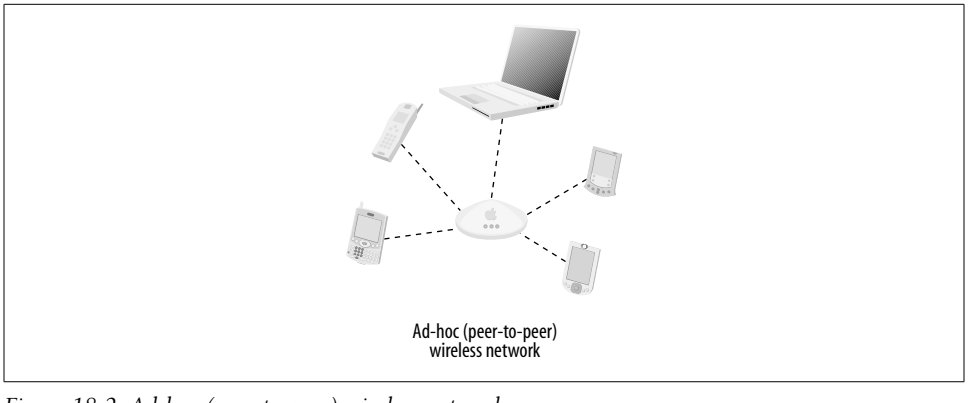



*Figure 18-2. Ad-hoc (peer-to-peer) wireless network*

802.11b networks operate on a predetermined set of frequencies known as a channel. The specification allows for 14 separate channels, though in North America users are limited to the first 11 and in Europe, the first 13. Only Japanese users have access to the full range of channels. In North America, the eleven channel span from 2400 to 2483 MHz, with each channel set to 22 MHz wide. Clearly there is some overlap between the channels, so it is important to conduct a site survey before selecting a channel to save future headaches by avoiding possible interference from other wireless networks.

As it is the most commonly used mode, this section will focus primarily on the infrastructure mode, which works on a hub-and-spoke model. The access point is the hub, and the clients are the spokes. An access point can either be a packaged unit bought from a store, or be built from a Linux machine running HostAP, which we'll discuss later.

An 802.11b network utilizes an access point that transmits a signal known as a *beacon*. This is basically just a message from the access point informing any nearby clients that it is available for connections. The beacon contains a limited amount of information that is necessary for the client to connect. The most important content of the beacon is the ESSID, or the name of the access point. The client, on seeing an access point, sends a request to associate. If the access point allows this, it grants an associate frame to the client, and they attach. Other types of authentication can also occur at this point. Some access points allow only clients with a prespecified MAC address, and some require further authentication. Developers are working on the standard 802.1x in an effort to establish a good authentication framework. Unfortunately, however, this is unlikely to prove effective, as there are already known vulnerabilities for it.

### Drivers

A Linux wireless driver can either be built into your kernel when you compile, or created as a *loadable kernel module* (LKM). It is recommended that you create a kernel module, since drivers may require frequent updates. Building a new module is much easier and less time consuming than rebuilding the entire kernel. Either way, you need to enable the wireless extensions in the kernel configuration. Most distributions now come with this enabled; however, if you are upgrading from scratch, the configuration looks like this:

```
# Loadable module support
#
CONFIG_MODULES=y
# CONFIG_MODVERSIONS is not set
CONFIG_KMOD=y
```

You may also notice that `MODVERSIONS` has been disabled. Having this option disabled makes it easier to compile modules separate from the kernel module tree. While not a requirement, this can save time when trying to patch and recompile kernel modules. Whichever chip architecture your card is using, if you're using a PC card, or even some PCI cards, pcmcia-cs, the Linux PCMCIA card manager, will be able to detect your card and will have the appropriate driver installed for you. There are a number of drivers available, but only two are currently and actively maintained.

The Orinico_cs drivers, written by David Gibson, are generally recognized as the best for the Hermes cards. They are actively developed and patched, and work with most wireless applications. The Orinoco_cs drivers have been included in the Linux kernel since Version 2.4.3 and have been a part of the pcmcia-cs since version 3.1.30. The driver included with your distribution may not be as current, so you may wish to upgrade.

Confusingly enough, some prism2 cards are now supported in the Orninoco_cs drivers. That number is increasing with each new release, so despite the name of the

driver, it is beginning to be a solid option for prism2 users, and may emerge as a standard in the future.

However, should your card not be supported by the Orinoco_cs driver, Prism cards are also supported by the linux-wlan-ng driver from Absolute Value Systems. This is the best known and maintained client driver for this chipset at the moment. It is also included with most Linux distributions and supports PCI, USB, and PCMCIA versions of Prism 2.x and 3.0 cards.

Once you have installed the driver of your choice and everything is working, you'll need to install the Linux Wireless Extension Tools, a collection of invaluable configuration tools written by Jean Tourrilhes, found at his site: http://*www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html*.

### Using the Linux Wireless Exension Tools

Linux Wireless Extension Tools are very useful for configuring every aspect of your wireless networking devices. If you need to change any of the default wireless options or want to easily configure ad-hoc networks, you will need to familiarize yourself with these tools. They are also required for building a Linux access point, which will be discussed later in the chapter.

The toolkit contains the following programs:

*iwconfig*
: This is the primary configuration tool for wireless networking. It will allow you to change all aspects of your configuration, such as the ESSID, channel, frequency, and WEP keying.

*iwlist*
: This program lists the available channels, frequencies, bit rates, and other information for a given wireless interface.

*iwspy*
: This program collects per node link quality information, and is useful when debugging connections.

*iwpriv*
: With this program, you can modify and manipulate parameters specific to your driver. For example, if you are using a modified version of the Orinoco_cs driver, *iwpriv* will allow you to place the driver into promiscuous mode.

## Linux Access Point Configuration

A very useful tool in the Linux wireless arsenal is HostAP, a wireless driver written by Jouni Malinen. It allows prism2 card users to turn their wireless cards and Linux servers into access points. Since there are many inexpensive access points on the market, you might be asking yourself why you'd ever want to turn a server into an

access point. The answer is simply a matter of functionality. With most inexpensive dedicated access points, there is little functionality other than simply serving up the wireless network. There is little option for access control and firewalling. This is where Linux provides immeasurable advantages. With a Linux-based access point, you will be able to take advantage of Netfilter, RADIUS, MAC authentication, and just about any other type of Linux-based software you may find useful.

### Installing the HostAP driver

In order to install HostAP, your system must have the following:

- Linux Kernel v2.4.20 or higher (kernel patches for 2.4.19 are included)
- Wireless extensions toolkit
- Latest HostAP driver, found at *http://hostap.epitest.fi/releases*

### Obtaining and building the HostAP driver

While RPM and .deb packages may be available, it's likely that you will have to build HostAP from scratch in order to have the most recent version of the driver. Untar the source to a working directory. HostAP will also look for the Linux kernel source code in */usr/src/linux*. Some distributions, such as Red Hat, place kernel source in */usr/src/linux-2.4*. In that case, you should make a symbolic link called *linux* that points at your kernel source directory. Preparing the source for installation is fairly straightforward and looks like this:

```
[root@localhost root]# tar xzvf hostap-0.0.1.tar.gz
hostap-0.0.1/
hostap-0.0.1/COPYING
hostap-0.0.1/ChangeLog
hostap-0.0.1/FAQ
.
.
hostap-0.0.1/Makefile
hostap-0.0.1/README
hostap-0.0.1/utils/util.h
hostap-0.0.1/utils/wireless_copy.h
```

Unlike many packages, HostAP has no configuration script to run before building the source. You do, however, need to choose which modules you would like to build. HostAP can currently support PCMCIA, PLX, or PCI devices. USB devices are not compatible, though support may be added in the future. For this example, we'll be building the PC Card version.

```
[root@localhost root]# make pccard
gcc -I/usr/src/linux/include  -O2 -D__KERNEL__ -DMODULE -Wall -g -c  -I/usr/src/
linux/arch/i386/mach-generic -I/usr/src/linux/include/asm/mach-default -fomit-frame-
pointer -o driver/modules/hostap_cs.o driver/modules/hostap_cs.c
gcc -I/usr/src/linux/include  -O2 -D__KERNEL__ -DMODULE -Wall -g -c  -I/usr/src/
linux/arch/i386/mach-generic -I/usr/src/linux/include/asm/mach-default -fomit-frame-
pointer -o driver/modules/hostap.o driver/modules/hostap.c
```

```
.
.
Run 'make install_pccard' as root to install hostap_cs.o

[root@localhost root]# make pccard_install
Installing hostap_crypt*.o to /lib/modules/2.4.20-8/net
mkdir -p /lib/modules/2.4.20-8/net
.
.
Installing /etc/pcmcia/hostap_cs.conf
[root@localhost hostap-0.0.1]#
```

After compiling, take note of the *hostap_cs.conf* file that's now installed in */etc/pcmcia*. It is the module configuration file and tells the module to load when seeing a matching card. The list comes with configurations for a number of popular cards, but if yours isn't listed, you will need to add it. This is an easy process, and entries are generally only three lines long:

```
card "Compaq WL100 11Mb/s WLAN Card"
   manfid 0x0138, 0x0002
   bind "hostap_cs"
```

To determine the exact make of your card, this command can be used:

```
[root@localhost etc]# cardctl ident
Socket 0:
  no product info available
Socket 1:
  product info: "Lucent Technologies", "WaveLAN/IEEE", "Version 01.01", ""
  manfid: 0x0156, 0x0002
  function: 6 (network)
[root@localhost etc]#
```

After these steps, you can either use *modprobe* to install your device, or reboot, and it will automatically load. You can check your syslog for the following message, or something similarly reassuring, to confirm that it has been loaded properly:

```
hostap_crypt: registered algorithm 'NULL'
hostap_cs: hostap_cs.c 0.0.1 2002-10-12
  (SSH Communications Security Corp, Jouni Malinen)
hostap_cs: (c) Jouni Malinen
PCI: Found IRQ 12 for device 00:0b.0
hostap_cs: Registered netdevice wlan0
prism2_hw_init()
prism2_hw_config: initialized in 17775 iterations
wlan0: NIC: id=0x8013 v1.0.0
wlan0: PRI: id=0x15 v1.0.7
wlan0: STA: id=0x1f v1.3.5
wlan0: defaulting to host-based encryption as a workaround for
  firmware bug in Host AP mode WEP
wlan0: LinkStatus=2 (Disconnected)
wlan0: Intersil Prism2.5 PCI: mem=0xe7000000, irq=12
wlan0: prism2_open
wlan0: LinkStatus=2 (Disconnected)
```

### Configuring HostAP

As discussed earlier, the *iwconfig* program is necessary to configure HostAP. First, you have to tell HostAP that you wish to use it in infrastructure mode. This is done with the following command:

```
vlager# iwconfig wlan0 mode Master
```

Next, the ESSID must be set. This will be the name of the access point, seen by all of the clients. In this example, we'll call ours "pub":

```
vlager#  iwconfig wlan0 essid pub
```

Then you should set the IP address as follows:

```
vlager# iwconfig wlan0 10.10.0.1
```

Selecting a channel is an important step, and as mentioned earlier, a site survey should be conducted to find the least congested channel available:

```
vlager# iwconfig channel 1
```

Now, once this has been completed, you can check to make sure that it's all been entered properly. The following command will produce:

```
vlager# iwconfig wlan0
wlan0     IEEE 802.11-DS  ESSID:"pub"  Nickname:" "
          Mode:Managed  Frequency:2.457GHz  Access Point:00:04:5A:0F:19:3D
          Bit Rate=11Mb/s   Tx-Power=15 dBm   Sensitivity:1/3
          Retry limit:4   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality:21/92  Signal level:-74 dBm  Noise level:-95 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:2960
          Tx excessive retries:1  Invalid misc:0   Missed beacon:0
```

The configuration of HostAP is complete. You should now be able to configure clients to connect to your Linux server through the wireless network.

### Additional options

As you get more comfortable with HostAP, you may wish to configure some additional options, such as MAC filtering and WEP configurations. It is a good idea to implement one or both of these security measures, since the default configuration results in an open access point that can be sniffed or used by anyone within range. Using WEP will make sniffing more difficult but not impossible, and will also make unauthorized use a more complex process. MAC address filtering provides another good way to keep out unwanted guests. Again, it is important to note that because of flaws in the 802.11b protocol, neither of these steps will guarantee a safe and secure computing environment. In order to secure a wireless installation properly, traditional security methods, like VPNs, should be used. A VPN provides both confidentiality and authentication, since after all, a client on a wireless LAN should be classified in the same way as a client from the Internet—untrusted.

Enabling WEP is simple and accomplished through the *iwconfig* command. You can choose whether you wish to use a 40-bit or 104-bit key. A 40-bit WEP key is configured by using 10 hexadecimal digits:

```
# iwconfig wlan0 key 1234567890
```

A 104-bit WEP key is configured with 26 hexadecimal digits, grouped in four, separated by a dash:

```
# iwconfig wlan0 key 1000-2000-3000-4000-5000-6000-70
```

Using the following command will confirm your key:

```
# iwconfig wlan0
```

It is also important to note that a WEP key can also be configured with an ASCII string. There are a number of reasons why this method isn't particularly good, but perhaps the most important is that ASCII keys don't always work when entered on the client side. However, should you decide you'd like to try, ASCII key configuration is accomplished by specifying *s:* followed by the key in the *iwconfig* command, as follows:

For 40-bit keys, 5 characters, which equate to a 10-digit hexadecimal, are requred:

```
# iwconfig wlan0 key s:smile
```

For 104-bit keys, 13 characters, which equate to the 26 hexadecimal digits in the earlier example, are required:

```
# iwconfig wlan0 key s:passwordtest3
```

If you wish to disable WEP, it can be done with:

```
# iwconfig wlan0 key off
```

HostAP also provides another useful feature that allows clients to be filtered by MAC address. While this method is not a foolproof security mechanism, it will provide you with a certain amount of protection from unauthorized users.

There are two basic ways to filter MAC addresses: you can either allow the clients in your list, or you can deny the clients in your list. Both options are enabled with the *iwpriv* command. The following command will enable MAC filtering, allowing the clients in the MAC list:

```
# iwpriv wlan0 maccmd 1
```

The MAC filtering command `maccmd` offers the following options:

`maccmd 0`
: Disables MAC filtering

`maccmd 1`
: Enables MAC filtering and allows specified MAC addresses

`maccmd 2`
: Enables MAC filtering and denies specified MAC addresses

*maccmd 3*
: Flushes the MAC filter table

To begin adding MAC addresses to your list, *iwpriv* is again used as follows:

```
# iwpriv wlan0 addmac 00:44:00:44:00:44
```

This command adds the client with the MAC address **00:44:00:44:00:44**. Now this user will be allowed to participate in our wireless network. However, should we decide that we don't want to allow this MAC at some point in future; it can be removed with the following command:

```
# iwpriv wlan0 delmac 00:44:00:44:00:44
```

Now this MAC has been removed, and will no longer be able to associate. If you have a large list of client MAC addresses and wish to remove them all, you can flush the MAC access control list by invoking:

```
# iwpriv wlan0 maccmd 3
```

This clears the access control list, and you will need to either disable filtering or re-enter the valid MAC addresses you wish to authorize.

## Troubleshooting

Because of their wireless nature, 802.11b networks can be much more prone to problems than traditional wired networks. There are a number of issues you may face when planning a wireless deployment.

The first is that of the signal strength. You want to make sure that your signal is strong enough to reach all of your clients, and yet not so strong that you're broadcasting to the world. The signal strength can be controlled with an antenna, through access point placement and some software controls. Experiment with different configurations and placements to see what works the best in your environment.

Interference may also be an issue. Many other devices now share the same 2.4 GHz frequency used by 802.11b. Cordless phones, baby monitor, and microwave ovens may all cause certain amounts of interference with your network. Neighboring access points operating on the same channel, or close to the channel you have selected, can also interfere with your network. While it's not likely that this particular issue will cause an outage, there will certainly be performance degradation. It is recommended, again, that experimentation be conducted prior to any major deployment.

Besides the physical issues, there are a number of software issues that are fairly common. Most issues are caused by driver or card incompatibility. The best way to avoid these kinds of problems is to know precisely which hardware you're using. Being able to identify your chipset will make finding the correct driver much easier.

Card identification is accomplished with the *cardctl* command, or by looking at the system log:

```
[root@localhost etc]# cardctl ident
Socket 0:
  no product info available
Socket 1:
  product info: "Lucent Technologies", "WaveLAN/IEEE", "Version 01.01", ""
  manfid: 0x0156, 0x0002
  function: 6 (network)
[root@localhost etc]#
```

In this example, we're using the easily identifiable WaveLAN card. Of course, this only works after successful configuration and module loading.

## Bridging Your Networks

Once the HostAP software and drivers have been properly configured, a useful next step is to grant the client's access to your wired LAN. This is done via bridging, which requires software located at *http://bridge.sourceforge.net*. Some distributions have ready-to-install packages containing all necessary tools and kernel modifications. Red Hat RPMs contain both. Debian users can just enter *apt-get install bridge-utils*. Check your distribution for specifics.

The bridging software is controlled with a program called *brctl*. It is the main configuration tool for the software and has quite a few options. We'll be using only a few of the available choices, but for a more comprehensive listing, check the brctl manpages, which install with the software.

The first step in bridging is to create our virtual bridging interface by typing:

```
vlager#  brctl addbr br0
```

This command creates an interface on the machine that is used to bridge the two connections. You'll see it when running *ifconfig*:

```
vlager# ifconfig br0
br0       Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

Additionally, you can tell by looking at the system log whether the bridge has been enabled:

```
Jan 22 13:17:54 vlager kernel: NET4: Ethernet Bridge 008 for NET4.0
```

The next step is to add the two interfaces that you wish to bridge. In this example, we'll be bridging `wlan0`, our wireless interface, and `eth0`, our wired Ethernet interface. First, however, it is important to clear the IP addresses from your interfaces. Since we're bridging the networks at layer two, IP addresses are not required.

```
Vlager#  ifconfig eth1 0.0.0.0 down
Vlager#  ifconfig wlan0 0.0.0.0 down
```

Once the IP addresses have been removed, the interfaces can be added to the bridge.

```
vlager# brctl addif br0 wlan0
vlager# brctl addif br0 eth1
```

The `addif` option adds interfaces that you'd like to have bridged. If you have another wired or wireless interface to add to the bridge, do so now in the same way.

The final step in the bridging process is to bring up the interfaces. You can also decide at this point whether you wish to assign an IP to your bridging interface. Having an IP makes it possible to remotely manage your server; however, some would argue that not having an IP makes the device more secure. For most purposes, however, it is helpful to have a management IP. To enable the bridge, you will need to enable the bridge interface, as well as both hardware interfaces.

```
vlager#  ifconfig br0 10.10.0.1 up
vlager#  ifconfig wlan0 up
vlager#  ifconfig eth0 up
```

With the successful completion of these commands, you will now be able to access your bridge using the IP **10.10.0.1**. Of course, this address must be one that is accessible from either side of the bridge.

Now you may wish to configure all of this to load at startup. This is done in a different way on just about every Linux distribution. Check the documentation specific to your distribution regarding the modification and addition to startup scripts.