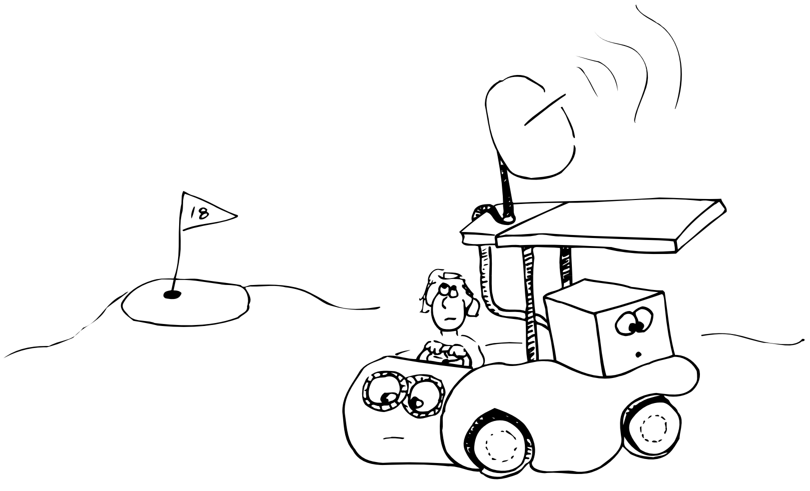
19 SLIP and PPP



19.1 Introduction

A typical UNIX environment consists of hosts connected with some type of network hardware, such as Ethernet. On top of this hardware, hosts communicate using the TCP/IP protocols discussed in Chapter 14. These protocols facilitate file transfer among hosts (**`rcp`** and **`ftp`**), interactive logins (**`rlogin`** and **`telnet`**), and file sharing (NFS).

You may find yourself wanting these same network services in places where an Ethernet connection isn't readily available: an engineer's home, a remote office in Guam, or perhaps on a notebook computer that you carry when you travel. This chapter discusses the software that is available to help connect machines that are out of reach of your LAN. You might want to review Chapter 13, *Configuring the Kernel,* and Chapter 8, *Serial Devices,* before continuing.

Almost every UNIX machine, from mainframe to notebook, provides one or more serial ports. Because serial ports are so flexible and so widely supported, they provide the standard hardware interface used to connect "outlying" machines. Two machines' serial ports can be wired together directly or connected via telephone using inexpensive, high-speed modems.

SLIP (Serial Line Internet Protocol) and PPP (Point-to-Point Protocol) are protocols that allow the transmission of network packets over serial lines. SLIP and PPP are called "serial line encapsulation protocols" because they specify how packets must be encoded for transmission on a slow (and often unreliable) serial line. This chapter discusses how SLIP and PPP work, and how to use them to connect a machine to a network. We'll also discuss the differences between the two.

19.2 How SLIP and PPP are Different from UUCP

One of the first questions most people ask about SLIP and PPP is, "What do SLIP and PPP offer me that systems like UUCP do not?" The answer is full network connectivity, exactly like you'd get on an Ethernet. You can use **`rlogin`**, **`rsh`**, **`ftp`**, **`rcp`**, **`telnet`**, NFS, and even X Windows over a SLIP or PPP connection. Furthermore, multiple connections to (and from) remote hosts can be active simultaneously.

By comparison, UUCP is a batched, store-and-forward protocol that provides a relatively limited set of capabilities and commands. However, UUCP is a little easier to set up and is pre-installed on most systems.

19.3 Performance Issues

SLIP and PPP provide all the functionality of Ethernet, but at *much* slower speeds. Normal office LANs operate at 10 Mb/s or 10,000 Kb/s. A dial-up connection operates at about 14 Kb/s.[1] To put this in perspective, it takes about 12.5 minutes to transfer a one-megabyte file across a SLIP line. This is usually suitable for home use, as well as for connections to remote offices that support only a handful of employees. A dial-up SLIP or PPP connection is *not* suitable as the only network connection for your 500-member engineering group based in Midwest City, Oklahoma.

19.4 SLIP and PPP Compared

SLIP and PPP are different and independent serial line encapsulation protocols. But they are similar in spirit and function and thus fit nicely within a single chapter. As with **`vi`** and **`emacs`**, there are advantages and disadvantages to both. SLIP is usually the best choice if you are connecting a house or a small field office.

Both SLIP and PPP require you to add a driver to your kernel to manage communication between the serial interface and the network portion of the kernel. Both systems also provide user-level commands that set up and manage connections to remote hosts.

SLIP

SLIP is the hot-rod, "bare bones" encapsulation protocol. It can be traced back to TCP/IP implementations of the early 80s and was actually imple-

1. SLIP and PPP are normally used at speeds over 9,600 bps. Technically, they can be used on slower links, but they become insufferably slow.

mented and released to the world on 4.2BSD systems and Sun workstations by Rick Adams in 1984.[2] Today, SLIP is available on a large variety of machines and operating systems, from UNIX and VMS to DOS, Macintosh, and even X terminals.

See page 426 for an explanation of Internet RFCs.

The original SLIP standard, RFC1055, defined the sequence of characters used to frame IP packets for transmission across a serial line: nothing more, nothing less. SLIP as defined in RFC1055 makes no effort to minimize the number of bytes sent across a low-speed serial line; whatever would normally be contained in an IP packet is transmitted. Since IP was designed to solve general network connectivity problems, it is not as efficient as it could be for the case in which only one or a few hosts are on the "remote" end of the link.

RFC1144 is a later SLIP standard (commonly referred to as "compressed SLIP" or "CSLIP") which tries to minimize the number of bytes that are actually transmitted over the serial line. CSLIP uses the following methods to reduce traffic and improve interactive response time:

- *TCP header compression* takes advantage of the small number of active connections over the link and the many fields of the header that can be predicted in the normal case (no dropped packets). To reduce the amount of data transmitted, the header for each connection is stored on both sides of the link, and only changes are sent. The header that normally accompanies each packet is reconstructed on the receiving side.
- *Type-of-service queueing* marks each packet coming from a user program to indicate whether it's "interactive." Modern versions of both **telnet** and **rlogin** request this option. Interactive packets are inserted at the front of the transmission queue, improving interactive response time when other programs (such as **ftp**) are contending for the serial line.

See Chapter 14 for more information about ICMP.

- *ICMP filtering* avoids the transfer of ICMP packets, which are used by **ping** to determine if a remote host is alive. Such inquiries, if done by many other sites, can consume a significant percentage of the bandwidth of a slow serial line.

Most implementations of CSLIP are backward-compatible with the original SLIP protocol.

Because SLIP is conceptually simpler than PPP, it is easier to debug when problems arise. A pure CSLIP connection will usually perform *slightly* better than an equivalent PPP connection.

2. The idea for SLIP debuted in a TCP/IP package called UNET by 3com and Ford Aerospace.

PPP

Designed by committee, PPP is the "everything *and* the kitchen sink" encapsulation protocol. PPP allows the transmission of "multi-protocol" packets over a single link. It is currently described in RFC1331. PPP is more flexible than SLIP, which only handles IP packets.

PPP has three main components:

- A method for encapsulating datagrams over serial links
- A Link Control Protocol (LCP) for establishing, configuring, and testing the data-link connection
- A family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols

These components, complete with state tables that rival the best finite-state automata final exams, are explained in detail in the RFC; we won't discuss them further in this chapter. As with SLIP, there are a number of PPP implementations (both commercial and free) available for a wide variety of machines.

PPP does offer some interesting features beyond those of SLIP. In particular, PPP can encapsulate packets from many protocols simultaneously over a single serial line. Thus, PPP would be a good choice if you needed to exchange TCP/IP and DECnet packets between two sites. PPP also has built-in (but optional) error correction.

What PPP gains in bells and whistles, it loses in simplicity and performance over slow lines. That is why we make the general recommendation that, when possible, SLIP should be used on dial-up lines.

19.5 Basic Connection Model

In order to connect a remote host to a network with SLIP or PPP, you need to be concerned with three things:

- Your host's kernel must be able to send IP packets across a serial line as specified by the SLIP or PPP protocol standard.
- You must have a user-level program that allows you to establish and maintain SLIP or PPP connections.
- There must be a host on the other end of the serial line that understands the protocol you are using.

Making your Host Speak SLIP or PPP

See page 262 for more information about **`ifconfig`**.

The most fundamental requirement for a connection is that your host be capable of sending and receiving SLIP or PPP packets. In the case of a UNIX host, this generally involves adding a module to your kernel that takes network packets (normally stored in a chain of kernel data structures called *mbufs*) and places them in the serial device output queue, and vice-versa. This module is usually placed in the kernel so that it appears to be just another network interface and can be manipulated with standard tools such as **`ifconfig`**. Kernel modules that perform this task are discussed later in this chapter.

Controlling SLIP and PPP Links

There are three common ways to manage a serial IP link:

Static	You configure a serial port as a network interface. This option can be used when the connection between the two machines is a serial cable or a dedicated link.
Dial-up	You use a command to dial a modem, log in to a remote host, and start the remote SLIP or PPP protocol engine. If this procedure succeeds, the serial port is then configured as a network interface. This option normally leaves the link up for a long time, which makes it best suited for a phone line dedicated to this purpose.
Dynamic	A daemon watches your serial "network" interfaces to see when traffic is queued for them. When someone tries to send a packet, the daemon automatically dials a modem to establish the connection, transmits the packet, and if the line goes back to being idle, disconnects the line after a reasonable amount of time. Dynamic dial-up is often used if a phone line is shared between voice and data, or if the connection involves long distance or connect-time charges. It's a common myth that dynamic dial-up is only available with PPP.

Programs to implement all of these schemes are included with most versions of SLIP and PPP.

Finding a Host to Talk to

See Chapter 20 for more information about the Internet.

If you're setting up a link between two remote sites within your company, or between home and work, you can simply install the SLIP or PPP software on both ends. However, if your intent is to use SLIP or PPP to obtain an Internet connection, you'll probably need to talk to an Internet service provider. Many service providers offer dial-up connections to the public at a reasonable cost. See page 437 for a list of Internet providers in the United States.

19.6 Network Concerns

Because a SLIP or PPP link is similar to an Ethernet connection, some of the administrative chores performed on a LAN must also be performed for a serial network connection. Since most links are similar to one another, sites often use boilerplate configurations that are modified only slightly for each connection.

On the other hand, since a SLIP or PPP link is not *really* an Ethernet connection, a few facilities should be used differently, or not at all. UNIX programs that assume the network is fast and reliable can easily run into trouble on a loaded 9,600 bps serial link.

Address Assignment

See page 260 for more information about assigning IP addresses.

Just as you assign an IP address to a new host on your Ethernet, you need to assign an IP address to each SLIP or PPP interface. There are a number of ways to assign addresses to these links (including assigning no addresses at all). We'll discuss only the simplest method here.

Think of a SLIP or PPP link as a network of its own. That is, a network of exactly two hosts, often called a "point-to-point" network. You need to assign a network number to the link just like you would assign a network number to a new Ethernet segment, according to whatever rules are used at your site. You can pick any two host addresses on that network, and assign one to each end of the link. Other local customs, such as the interface subnet mask, should be applied as well. Each host then becomes a "gateway" to the point-to-point network as far as the rest of the world is concerned.

This method is conceptually simple, but has the flaw of wasting a network number on each SLIP or PPP link. Check the documentation of the package you're using for other supported addressing conventions.

Routing

See page 252 for more information about routing.

Since SLIP and PPP turn the server into an IP router, you need to be concerned with IP routing just as you would on a "real" gateway, such as a machine that connects two Ethernets. The purpose of routing is to direct packets though gateways so that they can reach their ultimate destinations. There are a number of different ways to configure routing.

A run-of-the-mill SLIP or PPP client host should have a default route that forwards packets to its server. Likewise, the server needs to be known to the other hosts on its network as the gateway to the leaf machine.

Many SLIP and PPP packages handle these routing chores automatically.

Security

See Chapter 23 for more information about security.

Security concerns are introduced whenever you add a host to a network. Since a host connected via SLIP or PPP is a real member of the network, you need to treat it as such: verify that there are no accounts without passwords or with insecure passwords, that all appropriate vendor security fixes are installed, that **/.rhosts** and **/etc/hosts.equiv** files are not overly permissive, and so on.

NFS

See Chapter 17 for a general description of NFS.

One frequently-asked question is, "Can I use NFS with SLIP or PPP?" The answer is, "Maybe." Standard NFS uses the UDP protocol for packet transport. UDP does not guarantee reliable delivery and does not have TCP's congestion control algorithms. Congestion control becomes *extremely* important in the case of low-speed lines. Ergo, standard NFS performs poorly at best over SLIP and PPP. The solution is to use TCP-based NFS, which is now available from a few major vendors.

Perhaps the biggest problem with NFS over SLIP and PPP is that most vendors ship their machines with UDP checksums turned off (for backward compatibility with a time long past), and thus will accept packets that arrive corrupted. If you do not turn on UDP checksums in your kernel and an NFS packet arrives corrupted, it may corrupt your files. This is one case where PPP outshines SLIP, in that it is possible to have PPP do link-layer error checking. The solution, again, is to use TCP-based NFS, which doesn't have this problem.

X Windows

Since X Windows is based on TCP/IP, it is possible to run X applications over a SLIP or PPP connection. The X Windows protocol has a fairly high overhead, and therefore performance across SLIP/PPP is often less than stellar. Simple font-based applications such as **xterm** perform acceptably, but applications that use bitmap graphics do not. If you're looking into a serial connection for the sole purpose of remote X capability, your best bet is a protocol that is optimized for X over serial lines, such as XRemote from NCD.

19.7 Flavors of SLIP

There are several freely-available implementations of SLIP. A collection of them is available via anonymous **ftp** from ftp.uu.net, in the directory **networking/ip/slip**. The **cslip-2.7.tar.Z** distribution is maintained by Craig Leres at Lawrence Berkeley Labs, and is the best reference copy of UNIX SLIP code. If you're interested in SLIP for DOS, look in the **networking/ip/ka9q** directory on ftp.uu.net.

19.8 Flavors of PPP

Because of PPP's complexity, you may find that commercial implementations of PPP are more viable than any freely-available version. The best-supported package is the one sold by Morning Star Technologies in Columbus, Ohio. It is available for a variety of systems.

Fewer versions of free PPP are available than free SLIP, but there is still a decent selection. Look in the **networking/ip/ppp** directory at ftp.uu.net. We think **dp-3.0.tar.Z** by Kirk Smith at Purdue University is probably your best bet.

19.9 WALKTHROUGH: INSTALLING SLIP ON SUNOS

By now, you have a basic understanding of what SLIP and PPP are and what they can do for you. This section puts hand-waving aside and walks through an installation step by step. We will describe the installation of SLIP on a Sun IPX workstation that runs SunOS 4.1.3 and uses a Telebit T3000 V.32bis modem.

Before starting, collect the following items:

- Telebit T3000 modem
- Telebit T3000 modem manual
- RS-232 cable (male-to-male, straight through)
- Internet addresses you will use for the link
- Login name and password you will use for the link
- Phone number for the remote site

This package is also included on the CD-ROM.

Step 1: Obtain the **cslip-2.7** distribution. It's available via **ftp** from ftp.uu.net as **networking/ip/slip/cslip/cslip-2.7.tar.Z**. The package includes the kernel SLIP module for SunOS and the **tip**-with-SLIP program that we'll use to establish the connection.

Step 2: **uncompress** and un-**tar** the file. You may want to print out the **README** file for easy reference as you work.

Step 3: Following the instructions in the **README** under "Kernel Configuration," copy the SLIP include files to the appropriate directories:

```
% cp common/net/slcompress.h /sys/net
% cp common/net/slip.h /sys/net
% cp sunos4/net/if_slvar.h /sys/net
% cp common/net/slcompress.h /usr/include/net
% cp common/net/slip.h /usr/include/net
```

Step 4: Copy the source files for the SLIP kernel modules into the kernel source directory:

```
% cp common/net/slcompress.c /sys/net
% cp sunos4/net/if_sl.c /sys/net
```

These files will be compiled along with the rest of the kernel files when a new kernel containing SLIP is built.

Step 5: Edit **/sys/conf.common/files.cmn** to add the lines for the SLIP modules:

```
net/if_sl.c             optional sl INET
net/slcompress.c        optional sl INET
```

This file is used by **config** when generating a **Makefile** to control the compilation of a new kernel. These lines tell **config** to include the SLIP kernel modules.

Step 6: Edit **/sys/sun/str_conf.c** to include the lines that describe the streams used by SLIP. You'll see that the file is broken into three parts: include directives, external declarations, and an initialized structure. Add the three sets of additions to their appropriate sections:

```
#include "sl.h"
...
#if NSL > 0
extern struct streamtab if_slinfo;
#endif
...
#if NSL > 0
    { "slip", &if_slinfo },
#endif
```

These changes make SLIP a valid kernel module that can be used in association with a TTY stream to implement the SLIP protocol.

Step 7: Edit **/sys/sun4c/conf/***machine* (where *machine* is the name of your kernel configuration file) to add a line that declares the SLIP device within the kernel:

```
pseudo-device   slN  init slattach
```

N is the number of SLIP interfaces to add. Even if you're planning to use only one of your serial ports for a SLIP link, it's always a good idea to configure two SLIP devices in the kernel. In case of trouble, you can hook a cable between the two serial ports on the back of your system and try to get a SLIP connection running between them.

Step 8: Build a new kernel. This is done by first running **config** on your kernel configuration file (*machine*), to generate a **Makefile** in the directory **/sys/sun4c/***machine*. You can then use **make** in that directory to compile a new kernel.

Step 9: Install the new kernel. You'll need to copy the new kernel into the root directory, make a backup copy of the running kernel (in case something goes awry), and then name your new kernel **/vmunix** so it will be used the next time the system is booted. Be sure you know how to boot your old kernel before you reboot.

Step 10: Reboot the machine.

Step 11: **cd** to the **tip** directory of the **cslip-2.7** package. This directory contains a version of **tip** that Doug Kingston modified to know about login scripts and SLIP, hence the nickname **tip**-with-SLIP.

Step 12: Edit the **Makefile** in the **tip** directory to reflect the type of modem you'll be using. Since we're using a Telebit T3000 for this example, we'd edit the file to contain a line defining TELEBIT as a preprocessor constant. Many other modems are supported; they are described in the comment lines of the **Makefile**.

Step 13: Compile **tip**-with-SLIP using **make**.

Step 14: Install **tip**-with-SLIP. You may want to place it in a directory with other local executables, such as **/usr/local/bin**.

See Chapter 8 for more information about serial connectors.

Step 15: Connect the modem to your workstation. In this example, we'd use a 25-pin straight-through male-to-male RS-232 cable. Since the IPX really uses a mini DIN-8 connector for its serial ports, we'd also need a mini DIN-8 to DB-25 converter cable. We'll assume below that you connect the modem to the **ttya** port.

Step 16: Configure the machine to use hardware flow control on the serial port. Software flow control (XON/XOFF) interferes with the normal operation of SLIP, and so it should be avoided at all costs. Use the **eeprom** command to set the flow control parameter of the port:

```
eeprom ttya-mode=38400,8,n,1,h
```

The last argument, **h**, indicates hardware flow control.

See page 116 for more information about soft carrier.

Step 17: Turn off "soft carrier" on the port. Sun ships their machines configured to ignore loss of the carrier detect (CD) line from a modem. While this sometimes makes things work better "out of the box," it can cause disasters when a modem is connected. To turn off soft carrier, delete the word `local` after the `on` column for **ttya** in **/etc/ttytab**:

```
ttya    "/usr/etc/getty std.38400"    dialup      on
```

You must reboot to make the changes from steps 16 and 17 take effect.

See Chapter 8 for more about the ***/etc/remote*** *file.*

Step 18: Add an entry to **/etc/remote** so that you can use **tip** to connect to the modem and configure its onboard registers. Since you're just using the "standard" mode of **tip** to do this configuration, you can use either your system's original **tip** or the **tip** you just compiled. For this stage, you just need an entry that specifies the port and the baud rate at which you'll be talking to the modem:

```
telebit:dv=/dev/ttya:br#38400
```

We'll get to the fancy uses of **tip** a bit later.

Step 19: Use **tip** to connect to the modem and configure its registers. A good configuration for a T3000 would be

```
AT &F &C1 &D2 S0=0 S2=128 S7=65 S10=25
AT S51=6 S58=2 S68=2 S180=2 S181=1 S225=0
AT M0 V1 E0 X0 Q0 &W &W1
```

A brief explanation of the meaning of each command code is given in Table 19.1. You'll have to adapt the settings to your particular modem. The ones that really matter are S58 and S68, which set hardware flow control on the modem.

Table 19.1 Guide to command codes for example modem configuration[a]

Code	Meaning
AT	Tells the modem to listen to you
&F	Returns to factory default parameters, a sane baseline
&C1	Uses DCD (pin 8 of the RS-232 port) to indicate carrier detect
&D2	Disconnects when DTR (pin 20) is lost (**tip** dies or host crashes)
S0=0	Turns off auto-answer
S2=128	Disables the escape-to-command-mode sequence
S7=65	Waits 65 seconds for a connection, reasonable on many modems
S10=25	Waits 2.5 seconds before "loss of carrier," good on noisy lines
S51=6	Locks modem interface speed at 38,400 bits per second
S58=2	Uses RTS/CTS (hardware) flow control
S68=2	Uses RTS/CTS (hardware) flow control
S180=2	Uses V.42 error correction
S181=1	Requests error correction but doesn't require it
S255=0	Reloads profile A at power on or when reset
M0	Turns off the speaker
V1	Requests verbose result codes (required by **tip**)
E0	Turns off command echo (required by **tip**)
X0	Uses standard result codes
Q0	Returns result codes
&W	Writes to non-volatile profile A
&W1	Writes to non-volatile profile B

a. Some lines have been shaded to improve readability.

Step 20: Create an **/etc/remote** entry to be used by **tip**-with-SLIP to establish a SLIP connection. You'll need to know the phone number of the site that you'll be calling, your IP address on the SLIP link, and the IP address of the host on the other end of the link. If you're connecting to an Internet service provider, they can supply you with these addresses.

See page 256 for information about subnetting.

If you're connecting to another site that you administer, you'll need to allocate a pair of numbers on a network that you dedicate to the SLIP link. If you're going to install a lot of SLIP links, you can use subnetting to create network numbers that allow for only four hosts, thus reducing the number of class C network numbers you must apply for.

The **/etc/remote** fields used by **tip**-with-SLIP are defined in the file **README.SLIP** in the **tip** directory. The `ls` parameter specifies a file containing a login script, discussed below. The `cc` parameter specifies a command to execute once the connection is established. Usually, the **sliplogin** program (also included with **cslip**) is used. **sliplogin** reads **/etc/slip.hosts** to determine IP addresses and other connection attributes. See the man page for **sliplogin** in the **sliplogin** directory for more information. Here's a complete sample configuration:

```
slip:\
  :ls=/usr/local/slip/etc/annex.login:\
  :cc=/etc/sliplogin Shost2:br#38400:\
  :st=slip:rt:at=telebit:dv=/dev/ttya:du:pn=5551000:
```

Step 21: Create a login script for use by **tip**-with-SLIP. This file is also described in **README.SLIP** in the **tip** directory. It is basically an expect-send script that is used by **tip**-with-SLIP to log in to the remote host and start the SLIP server. In this example, the remote host will be a terminal server that asks for our user name and our password, and then expects us to send the command **slip**.

```
send \r
recv 10/again username:
goto login
label again
send \r
recv 10/error username:
label login
send \dloginname\r
recv 10/error word:
send \dpassword\r
recv 10/error annex:
send \d\d\rslip\r\r
done
label error
fail
```

Step 22: Establish a SLIP connection and test. Use **tip**-with-SLIP with the **-s** flag to dial the modem, log in, and start the SLIP protocol.

```
% tip -s slip
[Logging In]
[SLIP Running]
% ping -s xor.com
PING xor.com: 56 data bytes
64 bytes from xor.com (192.108.21.1): seq=0. time=241ms
...
```

You can use **tip**-with-SLIP's **-v** flag to debug your script. Once the link is up, you should be able to connect to the host of your choice.

Now you are ready to being using your SLIP connection for day-to-day communication. Using commands like **ping**, you can easily write scripts that monitor the status of the SLIP line and restart **tip**-with-SLIP when necessary (if you did not turn on the CSLIP option that disables **ping**). Keep in mind that SLIP provides a real network connection and that the security of hosts attached via SLIP must be carefully monitored.

19.10 Configuring PPP on Solaris 2.4

Solaris 2.4 includes dial-up PPP as part of the standard distribution. Third-party versions of PPP for Solaris (such as the version from Morning Star Technologies) generally offer more features and are better supported. This section discusses the generic version as distributed by Sun.

The PPP package that is integrated into Solaris 2.4 is "asynchronous PPP," since it is designed to handle connections over standard serial lines (such as dial-up modems). It's an official part of the Solaris operating system, so you don't need to perform all the sticky steps of installing a kernel PPP module. You can verify that you have PPP available with the following command:

```
pkginfo | grep ppp
```

If PPP is installed, you should get a response similar to:

```
system   SUNWapppr   PPP/IP Async PPP configuration files
system   SUNWapppu   PPP/IP Async PPP login service
system   SUNWpppk    PPP/IP and IPdialup Device Drivers
```

If PPP is not already installed, you'll need to install it as a Solaris package. See the manual page for **pkgadd** for more details. Table 19.2 lists the files used to configure and manage Solaris PPP.

Table 19.2 Files Involved with PPP on Solaris 2.4

File	Purpose
`/etc/init.d/asppp`	Boot-time startup script for dial-up PPP
`/usr/sbin/aspppd`	Daemon that manages PPP links
`/etc/asppp.cf`	Config file containing list of connections
`/usr/sbin/aspppls`	Login shell for dial-in connections
`/var/adm/log/asppp.log`	PPP activity log file
`/tmp/.asppp.fifo`	Hook into **aspppd** for dial-in connections

Solaris PPP and UUCP cooperate to share dial-out modems. To set up a PPP connection to a remote site, you should first add the modem and the site to the **Systems**, **Dialers**, and **Devices** files in the **/etc/uucp** directory. Details on this procedure can be found in Chapter 30.

Once the modem and remote site have been set up in the UUCP files (including a login script for the remote site in **/etc/uucp/Systems**), you need to edit **/etc/asppp.cf** to configure the connection's IP address and associate it with a **Systems** entry.

Here's an example **/etc/asppp.cf** that illustrates a link to "sliphub" (192.225.32.1) from "myhost" (192.225.32.2):

```
# set IP addresses of the pseudo-interface
ifconfig ipdptp0 plumb 192.225.32.2 192.225.32.1 up

# dynamic dialup parameters for pseudo-interface
path
  interface ipdptp0
  peer_system_name sliphub # Same as in Systems file
  inactivity_timeout 600   # time out if idle 10 minutes
```

Once this is in place, you can start the PPP daemon manually with

```
/etc/init.d/asppp start
```

This step should only be necessary the first time. On subsequent reboots, the PPP daemon will be started by **init**. If all goes well (check **/var/adm/log/asppp.log**), you should be able to reach the remote site with commands such as **telnet** and **ftp**.

19.11 Installing SLIP and PPP on Other Architectures

SLIP and PPP are available for almost every major version of UNIX. Installation instructions and features vary widely. In some cases, the OS vendors ship SLIP or PPP either as "unsupported" software or as part of their "networking" distribution. In other cases, vendors sell them as separate products. Unfortunately, vendors' implementations are often poorly documented or just plain broken, which is why other companies also support a SLIP/PPP package for those machines.

19.12 Dial-In Hubs

You may find that once you begin offering SLIP or PPP connections to users at home, you have more requests than you have serial ports. A number of terminal servers offer SLIP/PPP capability, including the Telebit Netblazer, the Xylogics Annex, and the Livingston Portmaster. These products often provide a convenient and easily-maintainable source of serial ports complete with the SLIP or PPP software already installed. They allow you to establish a dial-in "pool" of modems that offer SLIP and PPP service to off-site users.