20

Configuring PPP

Two protocols support connection to the Internet via modem: *SLIP* (*Serial Line Internet Protocol*) and *PPP* (*Point to Point Protocol*). As the name suggests, SLIP supports only IP. It is an older, less rugged protocol. Its only advantage is that it may be available where PPP isn't. If you have the choice, always take PPP: it differs from SLIP in being able to handle multiple protocols simultaneously, and it's also used on many DSL links (*PPP over Ethernet* or *PPPoE*). In this chapter, we'll look only at PPP.

PPP can perform a number of functions:

- It dials and establishes a phone connection if necessary. Strictly speaking, this isn't part of the PPP specification, but it is supported by most PPP implementations.
- It performs *authentication* to ensure that you are allowed to use the connection.
- It performs *negotiation* to decide what kind of protocol to use over the link. You might think, ''that's OK, I'm just using IP,'' but in fact there are a number of different ways to transmit IP datagrams over a PPP link. In addition, the other end may be able to handle non-Internet protocols such as X.25, SNA and Novell's IPX.
- It can perform *line quality monitoring* to ensure that the modems are able to understand each other.

FreeBSD provides two versions of PPP:

- Traditional BSD implementations of IP are located in the kernel, which makes for more efficiency. The corresponding implementation of PPP is referred to as *kernel PPP*. We'll look at it on page 355.
- Although kernel PPP is more efficient, it's also frequently more difficult to debug. As a result, FreeBSD also supplies an implementation known as *user PPP* or *iijppp*, after the *Internet Institute of Japan*, which supplied the original base code. It uses the *tunnel driver* to pass IP packets up to a user process. It's easier to configure and debug, and though it's not as efficient, the difference is not usually a problem. We'll look at this implementation on page 348.

If you have a DSL link, you don't have a choice of version: currently, only User PPP supports PPPoE.

Quick setup

The following sections go into some detail about how PPP works. It's not completely necessary to know it all to set up PPP. If you're in a hurry, you can move on to the configuration summaries on page 348 for user PPP, or page 359 for kernel PPP.

How PPP works

The following steps are necessary to set up a PPP connection:

- Set up a serial connection between the two systems. This could be a direct wire connection, but normally it's a dialup modem or an ISDN or DSL link.
- For a modem link, establish connection, traditionally called *dialing* the other end. The modems then set up a link and assert *DCD* (*Data Carrier Detect*) to tell the machines to which they are connected that the modem connection has been established.
- Start PPP. PPP selects a network interface to use for this connection.
- The two PPP processes negotiate details like IP address, protocol, and authentication protocols.
- Establish routes to the systems at the other end of the link.

On the following pages, we'll look at these points in detail.

The interfaces

Most network interfaces are dedicated to networking. For example, an Ethernet adapter can't be used for anything else. Serial lines are different: you could also use them to connect a mouse or even a remote terminal. There's another difference, too: you access serial lines via their device names. You access network interfaces via the *ifconfig* program, because they don't usually have device names—in technical jargon, they're in a separate *name space* from files. How do we solve this conflict?

The solution may seem a little surprising: PPP uses two different devices for each connection. You decide which serial line you want to use, and the software chooses a network interface for you, though you can override this choice if you're using user PPP. For example, your serial line might be called */dev/cuaa0*, */dev/cuaa1* or */dev/cuaa2*, while your interface will be called *tun0* or *tun1* (for user PPP), or *ppp0* or *ppp1* (for kernel PPP). It's possible to connect to a DSL line without PPP, but when you use PPPoE, you also have two devices, the Ethernet interface and *tun0* (Kernel PPP does not support PPPoE).

The tunnel device uses a device interface called */dev/tun**n***, where ***n*** is a digit, to read and write to the other side of the corresponding network interface.

User PPP runs in user space, so it *does* require a device name for the network interface, for example */dev/tun0*. It uses this device to read and write to the back end of the tunnel interface.

Dialing

If you're running a PPP connection over a dial-up link, you'll need to establish a telephone connection, which is still called *dialing*. That's a modem function, of course, and it's not defined in the PPP standard.

User PPP includes both built-in dialing support and external dialing support, while kernel PPP supplies only the latter. In practice, the only difference is the way your configuration files look. We'll look at these when we discuss the individual implementations.

You don't need to dial for a DSL connection.

Negotiation

Once the connection is established and the PPP processes can talk to each other, they negotiate what PPP features they will use.[1] The negotiation is successful if the two sides can agree on a functional subset of the features both would like to have.

For each feature of the link, PPP negotiation can perform up to two actions. User PPP uses the following terms to describe them, viewed from the local end of a link:

1. Years ago, you might have first have had to perform a normal UNIX login ("login authentication"). This was usually handled by the dialing script ("chat script"). Microsoft didn't support this kind of authentication, so it's practically obsolete now, though there's nothing wrong with the idea.

- To *enable* a feature means: ‘‘request this feature.’’
- To *disable* a feature means: ‘‘do not request this feature.’’
- To *accept* a feature means: ‘‘if the other side requests this feature, use it.’’
- To *deny* a feature means: ‘‘if the other side requests this feature, refuse it.’’

Negotiation is successful if each end accepts all the features that the other end has enabled. In some cases, however, PPP systems have an alternative. For example, if you accept PAP and deny CHAP, a router may first request CHAP, and when you deny it, it may then request PAP. You do this by enabling both PAP and CHAP in your PPP configuration files.

Who throws the first stone?

The first step in negotiation is to decide which side starts. One of them starts the negotiation, and the other one responds. If you configure your end incorrectly, one of these things can happen:

1. You both wait for the other end to start. Nothing happens. After a while, one of you times out and drops the connection.
2. You both fire away and place your demands, and listen for the other one to reply. The software should recognize that the other end is talking, too, and recover, but often enough both ends give up and drop the connection.
3. One side initiates negotiations before the other, and things work normally despite the misconfiguration. This is the most difficult kind to recognize: sometimes the connection will work, and sometimes it won’t, apparently dependent on the phase of the moon.

In general, systems with login authentication also initiate the negotiation. ISPs with *PAP* or *CHAP* authentication tend to expect the end user to start first, because that’s the way Microsoft does it. It’s easier for debugging to assume that the other end will start. If it doesn’t, and you have an external modem, you’ll notice that there is no traffic on the line, and that the line has dropped. Then you can switch to active mode negotiation.

It makes more sense for the called system to start the negotiation: the calling system is ready to use the link immediately, but the called system often takes a certain amount of time execute its PPP server program. A common cause of problems is when the server machine is busy and it takes a while to invoke the PPP process. In this case the caller sends its initial configuration data and the called system’s tty device may echo it back, resulting in a lot of confusion at the caller’s end. User PPP can typically survive about three reflections of this type before getting too confused to recover.

Typical features that require negotiation are:

- *What kind of authentication?* Login authentication doesn’t count here, because it’s not part of PPP. You may choose to offer *CHAP* or *PAP* negotiation. You may also require the other end to authenticate itself. You can accept both *CHAP* and *PAP* authentication—that way, you can accept whichever the other end asks for. If the other end is an ISP, you will probably not be able to authenticate him, but you should check with the ISP.

A common configuration problem is when a user enables some form of authentication without first agreeing this with the ISP. For example, very few ISPs perform authentication from their end (to prove to you that they're really the ISP you dialed). You can specify this type of authentication in your configuration file, but if the ISP refuses to authenticate, you will never establish a connection.

- *LQR*, *Link Quality Requests*, give you an overview of your line quality, *if* your modem doesn't use error correction. If it does use error correction, it will hide any LQR problems. Occasionally LQR packets can confuse a PPP implementation, so don't enable it if you don't intend to use it.
- *Data and header compression*. You have a choice here: modern modems offer various kinds of data compression, and so do the PPP implementations. As we saw on page 331, modem compression increases the data throughput, but also increases the latency. If your ISP supports the same kind of data compression as your PPP software, you might find that it improves matters to disable modem data compression. Both implementations support *Van Jacobson*, *deflate* and *Predictor 1* compression, and kernel PPP also supports *BSD compression*.

  Which do you choose? *Van Jacobson* compression works at the TCP level. It compresses only the headers (see page 280 for more details), and the other compression schemes work at the frame level. You can always enable Van Jacobson compression. As far as the others are concerned, use whatever the other side offers. In case of doubt, enable all available compression types and allow PPP to negotiate the best combination.

  Compression negotiation is handled by the *Compression Control Protocol*, usually known as *CCP*. It uses its own protocol number so that it can be distinguished from other protocols that the remote system might offer, such as IP, X.25, SNA and IPX.
- *IP addresses*. In many cases, the server machine allocates a dynamic IP address. We'll look at the implications below.
- *Proxy ARP*. Some systems can't understand being at the other end of a PPP link. You can fool them by telling the router to respond to *ARP* requests for machines at the other end of the link. You don't need this subterfuge in FreeBSD.

Authentication

Nearly every PPP link requires some kind of identification to confirm that you are authorized to use the link. On UNIX systems, the authentication traditionally consisted of the UNIX *login* procedure, which also allows you to dialup either to a shell or to a PPP session, depending on what user ID you use. Login authentication is normally performed by the dial-up chat script.

Microsoft has changed many things in this area. Their platforms don't normally support daemons, and in some cases not even multiple users, so the UNIX login method is difficult to implement. Instead, you connect directly to a PPP server and perform authentication directly with it. There are two different authentication methods currently available, *PAP* (*Password Authentication Protocol*) and *CHAP* (*Challenge Handshake Authentication Protocol*). Both perform similar functions. From the PPP point of view, you just need to know which one you are using. Your ISP should tell you this information, but a surprising number don't seem to know. In case of doubt, accept either of them.

Just to confuse matters, Microsoft has implemented authentication protocols of its own, such as MS LanMAN, MS CHAP Version 1 (also known as CHAP type `0x80`) and MS CHAP Version 2, also known as CHAP type `0x81`. User PPP supports both kinds.

If you're using *PAP* or *CHAP*, you need to specify a system name and an authentication key. These terms may sound complicated, but they're really just a fancy name for a user name and a password. We'll look at how to specify these values when we look at the individual software.

How do you decide whether you use *PAP* or *CHAP*? You don't need to—accept both and let the other end decide which kind to use.

Which IP addresses on the link?

After passing authentication, you may need to negotiate the addresses on the link. At first sight, you'd think that the IP addresses on the link would be very important. In fact, you can often almost completely ignore them. To understand this, we need to consider what the purpose of the IP addresses is.

An IP address is an address placed in the source or the destination field in an IP packet to enable the software to route it to its destination. As we saw in Chapter 17, *Configuring the local network*, it is not necessarily the address of the interface to which the packet is sent. If your packet goes through 15 nodes on the way through the Internet, quite a normal number, it will be sent to 14 nodes whose address is not specified in the packet.

The first node is the router at the other end of the PPP link. This is a point-to-point link, so it receives all packets that are sent down the line, so you don't need to do anything special to ensure it gets them. This is in marked contrast to a router on a broadcast medium like an Ethernet: on an Ethernet you must specify the IP address of the router for it to receive the packets.

> On an Ethernet, although the IP address in the packets doesn't mention the router, the Ethernet headers do specify the Ethernet address of the router as the destination address. Your local system needs the IP address to determine the Ethernet address with the aid of *ARP*, the *Address Resolution Protocol*.

In either case, except for testing, it's very unlikely that you will ever want to address a packet directly to the router, and it's equally unlikely that the router would know what to do with most kinds of packets if they are addressed to itself. So we don't really need to care about the address.

What if we set up the wrong address for the other end of the link? Look at the router `gw.example.com` in the reference network on page 294. Its PPP link has the local address 139.130.136.133, and the other end has the address 139.130.136.129. What happens if we get the address mixed up and specify the other end as 139.130.129.136? Consider the commands we might enter if we were configuring the interface manually (compare with page 300):

```
# ifconfig tun0  139.130.136.133  139.130.129.136  netmask 255.255.255.255
# route add default 139.130.129.133
```

Figure 20-1: Configuring an interface and a route

You need to specify the netmask, because otherwise *ifconfig* chooses one based on the network address. In this case, it's a class B address, so it would choose 255.255.0.0. This tells the system that the other end of the link is 139.130.129.136, which is incorrect. It then tells the system to route all packets that can't be routed elsewhere to this address (the default route). When such a packet arrives, the system checks the routing table, and find that 139.130.129.136 can be reached by sending the packet out from interface *tun0*. It sends the packet down the line.

At this point any memory of the address 139.130.129.136 (or, for that matter, 139.130.136.129) is gone. The packet arrives at the other end, and the router examines it. It still contains only the original destination address, and the router routes it accordingly. In other words, the router never finds out that the packet has been sent to the incorrect "other end" address, and things work just fine.

What happens in the other direction? That depends on your configuration. For any packet to get to your system from the Internet, the routing throughout the Internet must point to your system. Now how many IP addresses do you have? If it's only a single IP address (the address of your end of the PPP link), it must be correct. Consider what would happen if you accidentally swapped the last two octets of your local IP address:

```
# ifconfig tun0  139.130.133.136  139.130.129.136
```

If gw sends out a packet with this source address, it does not prevent it from getting to its destination, because the source address does not play any part in the routing. But when the destination system replies, it sends it to the address specified in the source field, so it will not get back.

So how can this still work? Remember that routers don't change the addresses in the packets they pass. If system bumble sends out a packet, it has the address 223.147.37.3. It passes through the incorrectly configured system gw unchanged, so the reply packet gets back to its source with no problems.

In practice, of course, it doesn't make sense to use incorrect IP addresses. If you don't specify an address at either end of the link, PPP can negotiate one for you. What this does mean, though, is that you shouldn't worry too much about what address you get. There is one exception, however: the issue of *dynamic addressing*. We'll look at that below.

The net mask for the link

As we saw on page 290, with a broadcast medium you use a net mask to specify which range of addresses can be addressed directly via the interface. This is a different concept from *routing*, which specifies ranges of addresses that can be addressed indirectly via the interface. By definition, a point-to-point link only has one address at the other end, so the net mask must be `255.255.255.255`.

Static and dynamic addresses

Traditionally, each interface has had a specific address. With the increase in the size of the Internet, this has caused significant problems: a few years ago, people claimed that the Internet was running out of addresses. As a solution, Version 6 of the Internet Protocol (usually called *IPv6*) has increased the length of an address from 32 bits to 128 bits, increasing the total number of addresses from 4,294,967,296 to 3.4×10^{38}—enough to assign multiple IP addresses to every atom on Earth (though there may still be a limitation when the Internet grows across the entire universe). FreeBSD contains full support for IPv6, but unfortunately that’s not true of most ISPs, so at present, IPv6 is not very useful. This book doesn’t discuss it further.

ISPs don’t use IPv6 because they have found another ‘‘solution’’ to the address space issue: *dynamic IP addresses*. With dynamic addresses, every time you dial in, you get a free IP address from the ISP’s address space. That way, an ISP only needs as many IP addresses as he has modems. He might have 128 modems and 5000 customers. With static addresses, he would need 5000 addresses, but with dynamic addresses he only needs 128. Additionally, from the ISPs point of view, routing is trivial if he assigns a block of IP addresses to each physical piece of hardware.

Dynamic addresses have two very serious disadvantages:

1. IP is a peer-to-peer protocol: there is no master and no slave. Theoretically, any system can initiate a connection to any other, as long as it knows its IP address. This means that your ISP could initiate the connection if somebody was trying to access your system. With dynamic addressing, it is absolutely impossible for anybody to set up a connection: there is no way for any other system to know in advance the IP address that you will get when the link is established.

   This may seem unimportant—maybe you consider the possibility of the ISP calling you even dangerous—but consider the advantages. If you’re travelling somewhere and need to check on something on your machine at home, you can just connect to it with *ssh*. If you want to let somebody collect some files from your system, there’s no problem. In practice, however, very few ISPs are prepared to call you, though that doesn’t make it a bad idea.

2. Both versions of PPP support an *idle timeout* feature: if you don’t use the link for a specified period of time, it may hang up. Depending on where you live, this may save on phone bills and ISP connect charges. It only disconnects the phone link, and not the TCP sessions. Theoretically you can reconnect when you want to continue, and the TCP session will still be active. To continue the session, however, you need to

have the same IP address when the link comes up again. Otherwise, though the session isn't dead, you can't reconnect to it.

Setting a default route

Very frequently, the PPP link is your only connection to the Internet. In this case, you should set the *default route* to go via the link. You can do this explicitly with the *route add* command, but both versions of PPP can do it for you.

When you set your default route depends on what kind of addressing you're using. If you're using static addressing, you can specify it as one of the configuration parameters. If you're using dynamic addressing, this isn't possible: you don't know the address at that time. Both versions have a solution for this, which we'll look at when we get to them.

Autodial

A PPP link over modem typically costs money. You will normally pay some or even all of the following charges:

- Telephone call setup charges, a charge made once per call. Unlike the other charges, these make it advantageous to stay connected as long as possible.
- Telephone call duration charges. In some countries, you pay per time unit (for example, per minute), or you pay a fixed sum for a variable unit of time.
- ISP connect charges, also per time unit.
- ISP data charges, per unit of data.

Typically, the main cost depends on the connection duration. To limit this cost, both PPP implementations supply methods to dial automatically and to disconnect when the line has been idle for a predetermined length of time.

The information you need to know

Whichever PPP implementation you decide upon, you need the following information:

- Which physical device you will use for the connection. For a modem, it's normally a serial port like */dev/cuaa0*. For PPPoE, it's an Ethernet adapter, for example *xl0*.
- If it's a modem connection, whom are you going to call? Get the phone number complete with any necessary area codes, in exactly the format the modem needs to dial. If your modem is connected to a PABX, be sure to include the access code for an external line.
- The user identification and password for connection to the ISP system.
- The kind of authentication used (usually CHAP or PAP).

In addition, some ISPs may give you information about the IP addresses and network masks, especially if you have a static address. You should have collected all this information in the table on page 323.

Setting up user PPP: the fast track

This chapter contains a lot of information about PPP setup. If you're in a hurry, and you have a ''normal'' PPP connection, the following steps may be enough to help you set it up. If it doesn't work, just read on for the in-depth explanation.

- Edit */etc/ppp/ppp.conf*. Find these lines lines:

  ```
  papchap:
   (comments omitted)
   set phone PHONE_NUM                           only for modem connections
   set authname USERNAME
   set authkey PASSWORD
  ```

  Replace the texts PHONE_NUM, USERNAME and PASSWORD with the information supplied by the ISP. If you're using PPPoE, remove the `set phone` line.

- Still in */etc/ppp/ppp.conf*, check that the device is correct. The default is */dev/cuaa1*. If you're connecting to a different serial line, change the device name accordingly. If you're running PPPoE, say over the Ethernet interface *xl0*, change it to:

  ```
   set device PPPoE:xl0
  ```

- Modify */etc/rc.conf*. First, check the PPP settings in */etc/defaults/rc.conf*. Currently they are:

  ```
  # User ppp configuration.
  ppp_enable="NO"         # Start user-ppp (or NO).
  ppp_mode="auto"         # Choice of "auto", "ddial", "direct" or "dedicated".
                          # For details see man page for ppp(8). Default is auto.
  ppp_nat="YES"           # Use PPP's internal network address translation or NO.
  ppp_profile="papchap"   # Which profile to use from /etc/ppp/ppp.conf.
  ppp_user="root"         # Which user to run ppp as
  ```

  Don't change this file: just add the following line to */etc/rc.conf*:

  ```
  ppp_enable=YES          # Start user-ppp (or NO).
  ```

- If you have a permanent connection (in other words, you don't ever want to disconnect the line), you should also add the following line to */etc/rc.conf*:

  ```
  ppp_mode=ddial          # Choice of "auto", "ddial", "direct" or "dedicated".
  ```

  This tells PPP not to disconnect at all.

- After this, PPP will start automatically on boot and will connect whenever necessary. If you are not planning to reboot, you can start PPP immediately with the following command:

```
    # /usr/sbin/ppp -quiet -auto papchap
```

If that works for you, you're done. Otherwise, read on.

Setting up user PPP: the details

The user PPP configuration files are in the directory */etc/ppp*. In addition to them, you probably want to modify */etc/rc.conf* to start PPP and possibly to include global Internet information. The main configuration file is */etc/ppp/ppp.conf*. It contains a number of multi-line entries headed by a label. For example, the `default` entry looks like:

```
default:
 set log Phase Chat LCP IPCP CCP tun command
 ident user-ppp VERSION (built COMPILATIONDATE)

 # Ensure that "device" references the correct serial port
 # for your modem. (cuaa0 = COM1, cuaa1 = COM2)
 #
 set device /dev/cuaa1                 device to use

 set speed 115200                      connect at 115,200 bps
 set dial "ABORT BUSY ABORT NO\\sCARRIER TIMEOUT 5 \
           \"\" AT OK-AT-OK ATE1Q0 OK \\dATDT\\T TIMEOUT 40 CONNECT"
 set timeout 180                       # 3 minute idle timer (the default)
 enable dns                            # request DNS info (for resolv.conf)
```

Let's look at this entry in detail.

- Note the format: labels begin at the beginning of the line, and other entries must be indented by one character.
- The line `default:` identifies the default entry. This entry is always run when PPP starts.
- The `set log` line specifies which events to log. This can be helpful if you run into problems.
- The `ident` line specifies what identification the system will present to the other end of the connection. You don't need to change it.
- The `set device` line specifies the device that PPP should use to establish the connection, in this case the second serial port, */dev/cuaa1*. For PPPoE connections, use the name of the Ethernet interface, prepended by the text `PPPoE`.

  ```
  set device PPPoE:xl0
  ```

- For modem connections, the `set speed` line sets the speed of the link between the modem and the computer. Some older PCs had problems at 115,200 bps, but you shouldn't have any need to change it any more, especially since the next lower speed for conventional PC hardware is 57,600 bps, which is too slow to use the full bandwidth when compression is enabled.
- Also for modems only, `set dial` describes a *chat script*, a series of responses and commands to be exchanged with the modem.

- `enable dns` tells PPP to get information about name servers when setting up the link. If the remote site supplies this information, you don't need to set it manually. You should remove this line if you're running a local name server, which I strongly recommend. See Chapter 21, *The Domain Name Service*, for more details.

The default entry alone does not supply enough information to create a link. In particular, it does not specify who to call or what user name or password to use. In addition to the default entry, you need an entry describing how to connect to a specific site. The bare minimum would be the first three `set` lines of the `papchap` entry in *ppp.conf*:

```
papchap:
 #
 # edit the next three lines and replace the items in caps with
 # the values which have been assigned by your ISP.
 #

 set phone PHONE_NUM
 set authname USERNAME
 set authkey PASSWORD

 set ifaddr 10.0.0.1/0 10.0.0.2/0 255.255.255.0 0.0.0.0
 add default HISADDR                    # Add a (sticky) default route
```

PPP calls this entry a *profile*. `papchap` is the profile supplied in the default installation. You can change the name, for example to the name of your ISP. This is particularly useful if you connect to more than one ISP (for example, with a laptop). In these examples, we'll stick with `papchap`.

As the comment states, replace the texts `PHONE_NUM`, `USERNAME` and `PASSWORD` with your specific information. If you are using PPPoE, replace the `set phone` line with a `set device` line as discussed above.

The last two lines may or may not be needed. The line `set ifaddr` specifies addresses to assign to each end of the link, and that they can be overridden. This line is seldom needed, even for static addressing: the ISP will almost always allocate the correct address. We'll look at this issue again below when we discuss dynamic addresses.

Finally, the last line tells *ppp* to set a default route on this interface when the line comes up. `HISADDR` is a keyword specifying the other end of the link. This is the only way to specify the route for dynamic addressing, but it works just as well for static addressing. If your primary connection to the Internet is via a different interface, remove this entry.

Negotiation

As we saw on page 342, you need to decide who starts negotiation. By default, user PPP starts negotiation. If the other end needs to start negotiation, add the following line to your */etc/ppp/ppp.conf*:

```
set openmode passive
```

User PPP uses four keywords to specify how to negotiate:

- To *enable* a feature means: “request this feature.”
- To *disable* a feature means: “do not request this feature.”
- To *accept* a feature means: “if the other side requests this feature, accept it.”
- To *deny* a feature means: “if the other side requests this feature, refuse it.”

We’ll see examples of this in the following sections.

Requesting LQR

By default, user PPP disables LQR, because it has been found to cause problems under certain circumstances, but it accepts it for modem lines. If you want to enable it, include the following line in your dial entry:

```
enable lqr
```

Authentication

The configuration file syntax is the same for PAP and CHAP. Normally, your ISP assigns you both system name and authorization key. Assuming your system name is *FREEBIE*, and your key is *X4dWg9327*, you would include the following lines in your configuration entry:

```
set authname FREEBIE
set authkey  X4dWg9327
```

User PPP accepts requests for PAP and CHAP authentication automatically, so this is all you need to do unless you intend to authenticate the other end, which is not normal with ISPs.

/etc/ppp/ppp.secret

The PPP system name and authentication key for PAP or CHAP are important data. Anybody who has this information can connect to your ISP and use the service at your expense. Of course, you should set the permissions of your */etc/ppp/ppp.conf* to `-r--------` and the owner to `root`, but it’s easy and costly to make a mistake when changing the configuration. There is an alternative: store the keys in the file */etc/ppp/ppp.secret*. Here’s a sample:

```
# Sysname       Secret Key      Peer's IP address
oscar           OurSecretKey    192.244.184.34/24
FREEBIE         X4dWg9327       192.244.184.33/32
gw              localPasswdForControl
```

There are a few things to note here:

- As usual, lines starting with # are comments.
- The other lines contain three values: the system name, the authentication key, and possibly an IP address.
- The last line is a password for connecting to the *ppp* process locally: you can connect to the process by starting:

  ```
  # telnet localhost 3000
  ```

  The local password entry matches the host name. See the man page *ppp(8)* for further details.

Dynamic IP configuration

If you have to accept dynamic IP addresses, user PPP can help. In fact, it provides fine control over which addresses you accept and which you do not. To allow negotiation of IP addresses, you specify how many bits of the IP addresses at each end are of interest to you. For static addresses, you can specify them exactly:

```
set ifaddr 139.130.136.133  139.130.136.129
```

You can normally maintain some control over the addressing, for example to ensure that the addresses assigned don't conflict with other network connections. The addresses assigned to you when the link comes up are almost invariably part of a single subnet. You can specify that subnet and allow negotiation of the host part of the address. For example, you may say ''I don't care what address I get, as long as the first three bytes are `139.130.136`, and the address at the other end starts with `139`.'' You can do this by specifying the number of bits that interest you after the address:

```
set ifaddr 139.130.136.133/24  139.130.136.129/8
```

This says that you would prefer the addresses you state, but that you require the first 24 bits of the local interface address and the first eight bits of the remote interface address to be as stated.

If you really don't care which address you get, specify the local IP address as 0:

```
set ifaddr 0 0
```

If you do this, you can't use the `-auto` modes, because you need to send a packet to the interface to trigger dialing. Use one of the previous methods in this situation.

Running user PPP

After setting up your PPP configuration, run it like this:

```
$ ppp
Working in interactive mode
Using interface: tun0
ppp ON freebie> dial papchap                   this is the name of the entry in ppp.conf
Dial attempt 1 of 1
Phone: 1234567                                 the phone number
dial OK!                                       modem connection established
login OK!                                      authentication complete
ppp ON freebie> Packet mode.                   PPP is running
ppp ON freebie>
PPP ON freebie>                                and the network connection is complete
```

You'll notice that the prompt (ppp) changes to upper case (PPP) when the connection is up and running. At the same time, *ppp* writes some messages to the log file */var/log/ppp.log*:

```
Sep  2 15:12:38 freebie ppp[23679]: Phase: Using interface: tun0
Sep  2 15:12:38 freebie ppp[23679]: Phase: PPP Started.
Sep  2 15:12:47 freebie ppp[23679]: Phase: Phone: 1234567
Sep  2 15:13:08 freebie ppp[23679]: Phase: *Connected!
Sep  2 15:13:11 freebie ppp[23679]: Phase: NewPhase: Authenticate
Sep  2 15:13:11 freebie ppp[23679]: Phase:  his = c223, mine = 0
Sep  2 15:13:11 freebie ppp[23679]: Phase:  Valsize = 16, Name = way3.Adelaide
Sep  2 15:13:11 freebie ppp[23679]: Phase: NewPhase: Network
Sep  2 15:13:11 freebie ppp[23679]: Phase: Unknown protocol 0x8207
Sep  2 15:13:11 freebie ppp[23679]: Link:  myaddr = 139.130.136.133  hisaddr = 139.1
30.136.129
Sep  2 15:13:11 freebie ppp[23679]: Link: OsLinkup: 139.130.136.129
Sep  2 15:14:11 freebie ppp[23679]: Phase: HDLC errors -> FCS: 0 ADDR: 0 COMD: 0 PRO
TO: 1
```

You'll notice a couple of messages that look like errors. In fact, they're not: `Unknown protocol 0x8207` means that the other end requested a protocol that *ppp* doesn't know (and, in fact, is not in the RFCs. This is a real example, and the protocol is in fact Novell's IPX). The other message is `HDLC errors -> FCS: 0 ADDR: 0 COMD: 0 PROTO: 1`. In fact, this relates to the same "problem."

How long do we stay connected?

The following entries in */etc/defaults/rc.conf* relate to user ppp:

```
# User ppp configuration.
ppp_enable="NO"         # Start user-ppp (or NO).
ppp_mode="auto"         # Choice of "auto", "ddial", "direct" or "dedicated".
                        # For details see man page for ppp(8). Default is auto.
ppp_nat="YES"           # Use PPP's internal network address translation or NO.
ppp_profile="papchap"   # Which profile to use from /etc/ppp/ppp.conf.
ppp_user="root"         # Which user to run ppp as
```

Now our PPP connection is up and running. How do we stop it again? There are two possibilities:

- To stop the connection, but to leave the *ppp* process active, enter `close`:

  ```
  PPP ON freebie> close
  ppp ON freebie>
  ```

- To stop the connection and the *ppp* process, enter `q` or `quit`:

  ```
  PPP ON freebie> q
  #
  ```

  There are a couple of problems with this method: first, a connection to an ISP usually costs money in proportion to the time you are connected, so you don't want to stay connected longer than necessary. On the other hand, you don't want the connection to drop while you're using it. User PPP approaches these problems with a compromise: when the line has been idle for a certain time (in other words, when no data has gone in either direction during this time), it disconnects. This time is called the *idle timeout*, and by default it is set to 180 seconds. You can set it explicitly:

  ```
   set timeout 300
  ```

  This sets the idle timeout to 300 seconds (5 minutes).

Automating the process

Finally, setting up the connection this way takes a lot of time. You can automate it in a number of ways:

- If you have a permanent connection, you can tell user PPP to stay up all the time. Use the `-ddial` modifier:

  ```
  $ ppp -ddial papchap
  ```

  Again, papchap is the name of the PPP profile. This version dials immediately and keeps the connection up regardless of whether traffic is passing or not.

- If you want to be able to connect to the Net automatically whenever you have something to say, use the `-auto` modifer:

  ```
  $ ppp -auto papchap
  ```

  In this case, user PPP does not dial immediately. As soon as you attempt to send data to the Net, however, it dials automatically. When the line has been idle for the idle timeout period, it disconnects again and waits for more data before dialing. This only makes sense for static addresses or when you know that no IP connections remain alive after the line disconnects.

- Finally, you can just write

  ```
  $ ppp -background papchap
  ```

The -background option tells user PPP to dial immediately and stay in the background. After the idle timeout period, the user PPP process disconnects and exits. If you want to connect again, you must restart the process.

Actions on connect and disconnect

If you don't have a permanent connection, there are some things that you might like to do every time you connect, like flush your outgoing mail queue. User PPP provides a method for doing this: create a */etc/ppp/ppp.linkup* with the same format as */etc/ppp/ppp.conf*. If it exists, PPP looks for the profile you used to start PPP (papchap in our examples) and executes the commands in that section. Use the exclamation mark (!) to specify that the commands should be performed by a shell. For example, to flush your mail queue, you might write:

```
papchap:
 ! sendmail -q
```

Similarly, you can create a file */etc/ppp/ppp.linkdown* with commands to be executed when the link goes down. You can find sample files in the directory */usr/share/examples/ppp*.

If things go wrong

Things don't always work "out of the box." You could run into a number of problems. We'll look at the more common ones on page 361.

Setting up kernel PPP

It makes more sense to run PPP in the kernel than in user space: in the kernel it's more efficient and theoretically less prone to error. The implementation has fewer features than user PPP, and it's not quite as easy to debug, so it is not used as much.

The configuration files for kernel PPP are in the same directory as the user PPP configuration files. You can also set up your own *~/.ppprc* file, though I don't recommend this: PPP is a system function and should not be manipulated at the user level.

Kernel PPP uses a daemon called *pppd* to monitor the line when it is active. Kernel PPP interface names start with *ppp* followed by a number. You need one for each concurrent link. You don't need to specifically build a kernel for the *ppp* interface: FreeBSD Release 5 loads the PPP module */boot/kernel/if_ppp.ko* dynamically and adds interfaces as required. This also means that you can no longer check for ppp support with the *ifconfig* command. The interface won't be there until you need it.

Kernel PPP used to provide a number of build options to enable some features, including the compression options described below. The options are still there, but they're set by default, so you don't need to do anything there either.

When kernel PPP starts, it reads its configuration from the file */etc/ppp/options*. Here is a typical example:

```
# Options file for PPPD
defaultroute                                 set the default route here when the line comes up
crtscts                                      use hardware flow control
modem                                        use modem control lines
deflate 12,12                                use deflate compression
predictor1                                   use predictor 1 compression
vj-max-slots 16                              Van Jacobson compression slots
user FREEBIE                                 our name (index in password file)
lock                                         create a UUCP lock file
```

This is quite a short file, but it's full of interesting stuff:

- The `defaultroute` line tells the kernel PPP to set the default route via this interface after it establishes a connection.
- The `crtscts` line tells it to use hardware flow control (necessary to prevent loss of characters). You could also specify `xonxoff`, which uses software flow control, but hardware flow control is preferable.
- The `modem` line says to monitor the modem *DCD* (Carrier detect) line. If the connection is lost without proper negotiation, the only way that kernel PPP can know about it is because of the drop in *DCD*.
- The line *deflate* tells kernel PPP to request *deflate* compression, which can increase the effective bandwidth.
- `predictor1` tells PPP to use *Predictor 1* compression where possible.
- `vj-max-slots` specifies how many slots to use for *Van Jacobson* header compression. Having more slots can speed things up.
- The `user` line tells kernel PPP the user ID to use. If you don't specify this, it takes the system's name.
- `lock` tells kernel PPP to create a UUCP-style lock on the serial line. This prevents other programs, such as *getty*, from trying to open the line while it is running PPP.

None of these options are required to run *pppd*, though you'll probably need a `user` entry to establish connection. It's a good idea to set the indicated options, however.

Authentication

We've seen that */etc/ppp/options* contains a user name, but no password. The passwords are stored in separate files, */etc/ppp/chap-secrets* for *CHAP*, or */etc/ppp/pap-secrets* for *PAP*. The format of either file is:

username systemname password

To match any system name, set *systemname* to `*`. For example, to authenticate the *FREEBIE* we saw on page 351, we would enter the following in the file:

```
FREEBIE * X4dWg9327
```

In addition, you should add a `domain` line to specify your domain for authentication purposes:

```
domain example.org
```

Dialing

Kernel PPP does not perform dialing, so you need to start a program that does the dialing. In the following example, we use *chat*, a program derived from UUCP intended exactly for this purpose. Some people use *kermit*, which is in fact a complete communications program for a PC protocol, to perform this function, but this requires manual intervention. *chat* does the whole job for you.

Chat scripts

chat uses a *chat script* to define the functions to perform when establishing a connection. See the man page *chat(8)* for further details. The chat script consists primarily of alternate *expect strings*, which *chat* waits to receive, followed by *send* strings, which *chat* sends when it receives the *expect* string.

In addition to these strings, the chat script can contain other commands. To confuse things, they are frequently written on a single line, though this is not necessary: *chat* does not pay any attention to line breaks. Our chat script, which we store in */etc/ppp/dial.chat*, looks more intelligible written in the following manner:

```
# Abort the chat script if the modem replies BUSY or NO CARRIER
ABORT BUSY
ABORT 'NO CARRIER'
# Wait up to 5 seconds for the reply to each of these
TIMEOUT 5
'' ATZ
OK ATDT1234567
# Wait 40 seconds for connection
TIMEOUT 40
CONNECT
```

This script first tells *chat* to abort dial-up on a `BUSY` or `NO CARRIER` response from the modem. The next line waits for nothing (`''`) and resets the modem with the command `ATZ`. The following line waits for the modem to reply with `OK`, and dials the ISP.

Call setup can take a while, almost always more than five seconds for real (analogue) modems, so we need to extend the timeout, in this case to 40 seconds. During this time we must get the reply `CONNECT` from the modem.

Who throws the first stone?

On page 342 we saw how to specify whether we should start negotiating or whether we should wait for the other end to start. By default, kernel PPP starts negotiation. If you want the other end to start, add the keyword `passive` in your */etc/ppp/options* file.

Dynamic IP configuration

By default, kernel PPP performs dynamic address negotiation, so you don’t need to do anything special for dynamic IP. If you have static addresses, add the following line to */etc/ppp/conf*:

```
139.130.136.133:139.130.136.129
```

These are the addresses that you would use on machine `gw.example.org` to set up the PPP link in the middle of Figure 16-7 on page 294. The first address is the local end of the link (the address of the *ppp**n*** device), and the second is the address of the remote machine (`free-gw.example.net`).

Running kernel PPP

To run *pppd*, enter:

```
# pppd /dev/cuaa1 115200 connect 'chat -f /etc/ppp/dial.chat'
```

This starts kernel PPP on the serial line */dev/cuaa1* at 115,200 bps. The option `connect` tells kernel PPP that the following argument is the name of a program to execute: it runs *chat* with the options `-f /etc/ppp/dial.chat`, which tells *chat* the name of the chat file.

After you run *pppd* with these arguments, the modem starts dialing and then negotiates a connection with your provider, which should complete within 30 seconds. During negotiation, you can observe progress with the *ifconfig* command:

```
$ ifconfig ppp0
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
            at this point, the interface has not yet started
$ ifconfig ppp0
ppp0: flags=8810<POINTOPOINT,RUNNING,MULTICAST> mtu 1500
            now the interface has been started
$ ifconfig ppp0
ppp0: flags=8811<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1500
            inet 139.130.136.133 --> 139.130.136.129 netmask 0xffffffff
            now the connection has been established
```

Automating the process

You can automate connection setup and disconnection in a number of ways:

- If you have a permanent connection, you can tell kernel PPP to stay up all the time. Add the following line to */etc/ppp/options*:

  ```
  persist
  ```

  If this option is set, kernel PPP dials immediately and keeps the connection up regardless of whether traffic is passing or not.

- If you want to be able to connect to the Net automatically whenever you have something to say, use the `demand` option:

  ```
  demand
  ```

  In this case, kernel PPP does not dial immediately. As soon as you attempt to send data to the net, however, it dials automatically. When the line has been idle for the idle timeout period, it disconnects again and waits for more data before dialing.

- Finally, you can start kernel PPP without either of these options. In this case, you are connected immediately. After the idle timeout period, kernel PPP disconnects and exits. If you want to connect again, you must restart the process.

Timeout parameters

A number of options specify when kernel PPP should dial and disconnect:

- The `idle` parameter tells kernel PPP to disconnect if the line has been idle for the specified number of seconds, and if `persist` (see above) has not been specified. For example, to disconnect after five minutes, you could add the following line to the */etc/ppp/options* file:

  ```
  idle 300
  ```

- The `active-filter` parameter allows you to specify which packets to count when determining whether the line is idle. See the man page for more details.

- The `holdoff` parameter tells kernel PPP how long to wait before redialing when the line has been disconnected for reasons other than being idle. If the line is disconnected because it was idle, and you have specified `demand`, it dials as soon as the next valid packet is received.

Configuration summary

To summarize the examples above, we'll show the kernel PPP versions of the user PPP examples on page 348. As before, we assume that the reference network on page 294 uses *CHAP* authentication, and we have to initiate. The */etc/ppp/options* looks like:

```
# Options file for PPPD
defaultroute                          set the default route here when the line comes up
crtscts                               use hardware flow control
modem                                 use modem control lines
domain example.org                    specify your domain name
persist                               stay up all the time
deflate 12,12                         use deflate compression
predictor1                            use predictor 1 compression
vj-max-slots 16                       Van Jacobson compression slots
user FREEBIE                          name to present to ISP
139.130.136.133:139.130.136.129       specify IP addresses of link
```

/etc/ppp/dial.chat is unchanged from the example on page 357:

```
# Abort the chat script if the modem replies BUSY or NO CARRIER
ABORT BUSY
ABORT 'NO CARRIER'
# Wait up to 5 seconds for the reply to each of these
TIMEOUT 5
'' ATZ
OK ATDT1234567
# Wait 40 seconds for connection
TIMEOUT 40
CONNECT
```

/etc/ppp/chap-secrets contains:

```
FREEBIE * X4dWg9327
```

With kernel PPP, there's no need to disable *PAP*: that happens automatically if it can't find an authentication for *FREEBIE* in */etc/pap-secrets*.

The change for dynamic addressing is even simpler. Remove the line with the IP addresses from the */etc/ppp/options* file:

```
# Options file for PPPD
defaultroute                          set the default route here when the line comes up
crtscts                               use hardware flow control
modem                                 use modem control lines
domain example.org                    specify your domain name
persist                               stay up all the time
deflate 12,12                         use deflate compression
predictor1                            use predictor 1 compression
vj-max-slots 16                       Van Jacobson compression slots
user FREEBIE                          name to present to ISP
```

Actions on connect and disconnect

If you don't have a permanent connection, there are some things that you might like to do every time you connect, like flush your outgoing mail queue. We've seen that user PPP provides a method for doing this with the */etc/ppp/ppp.linkup* and */etc/ppp/ppp.linkdown* files. Kernel PPP supplies similar functionality with */etc/ppp/auth-up* and */etc/ppp/auth-down*. Both of these files are shell scripts. For example, to flush your mail queue, you might put the following line in */etc/ppp/auth-up*:

```
sendmail -q
```

Things that can go wrong

Setting up PPP used to be a pain. Two things have made it easier than it used to be. Firstly, the widespread adoption of dialup Internet connections has consolidated the procedure, so one size fits nearly everybody. Secondly, the software has had some of the rough edges taken off, so now it almost works out of the box. Still there are a number of things that can go wrong.

Problems establishing a connection

The first thing you need to do is to dial the connection. If you have an external modem, you can follow the process via the indicator LEDs. The following steps occur:

- First, the `OH` LED (“off hook”) goes on, indicating that the modem is dialing. If this doesn’t happen, check the cables and that you’re talking to the right device.
- Next you should see a brief flicker of the `RD` and `TD` LEDs. If that doesn’t happen, you may also have cable problems, or it could be a problem with the chat script.
- When the `CD` (or `DCD`) LED goes on, you have a connection to the remote system. If you don’t get that, check the phone number.
- If you get this far, but you still don’t get a connection, check the system log files. It’s most likely to be an authentication failure. See page 353 for an example of the messages from user PPP. Kernel PPP is much less verbose.