Morning Star PPP

Version 1.0

User Guide

Revision 1.15
August 22, 1991

Copyright © 1991 Morning Star Technologies, Inc.
All rights reserved

Published by
Morning Star Technologies, Inc.
1760 Zollinger Road
Columbus, Ohio 43221
USA

+1 614 451 1883 (voice)
+1 800 558 7827 (voice)
+1 614 459 5054 (FAX)
marketing@morningstar.com (e-mail)

Morning Star PPP
(Version 1.0)
User Guide (version 1.15)

1. Introduction to PPP

Morning Star PPP provides an easy and flexible means of creating wide-area TCP/IP based networks to transfer data between UNIX[1] systems, and between UNIX systems and others that also implement the Internet standard Point to Point Protocol. Users of Morning Star PPP can connect their isolated UNIX systems to established TCP/IP networks, or can connect geographically-separated networks. Morning Star PPP can also run on a hub machine to provide dialin access to multiple remote systems.

Since they continue to use the UNIX host's native TCP/IP implementation and related utilities, users require no additional training to make full use of the new link. Because of the transparency of TCP/IP internetwork routing, many may even be unaware that remote facilities are not connected to the local network.

By contrast to network routers that require an expensive leased line to provide a constant connection, Morning Star PPP can establish the link when traffic demand warrants a call over standard telephone lines. When the traffic has finished, Morning Star PPP will hang up the connection to save on telephone costs.

The UNIX system administrator has complete control over the traffic allowed to traverse the link. Morning Star PPP can even turn a UNIX system into a selective-isolation 'firewall' gateway, enhancing the security provided to other systems on the local network.

Morning Star PPP implements HDLC address/control field and protocol field compression, as well as TCP header compression, to reduce transmission overhead to a minimum. This improves both throughput and interactive response on typical modem connections. Its interface to the UNIX host's TCP/IP implements Van Jacobson's TCP 'fast queue' scheme, whereby interactive packets have priority for transmission on a congested link.

2. Using this Manual

The portions of this manual following this introductory User Guide have been organized in the style of the standard Berkeley UNIX manual pages. The interface to the host UNIX system's IP implementation is described in Section 4. Section 5 describes the format of the configuration files. The Morning Star PPP daemon itself is described in Section 8.

3. Related Manuals

The Morning Star PPP daemon is described in detail in the UNIX-style manual page **pppd**(8). Its configuration files are described in **ppp.devices**(5), **ppp.dialers**(5), **ppp.filter**(5), and **ppp.systems**(5). The IP tunnel driver is described in **tun**(4).

Hardware installation, when necessary, is described in the *Hardware Installation Guide* for the Morning Star *SnapLink*[2] SCSI communications interface. Not all Morning Star PPP installations require additional hardware, so if you are using a serial port-only product, you won't have received a copy of any *Hardware Installation Guide*.

The authoritative references on TCP/IP and the Point-to-Point Protocol are the Internet Requests for Comments series, particularly RFCs 1171[3] and 1172[4]. These are occasionally updated by the Point-to-Point Protocol Working Group of the Internet Engineering Task Force (IETF), and are available from the Network Information Center. See the file **rfc-index** in the Morning Star PPP distribution directory for

[1] UNIX is a registered trademark of AT&T.

[2] SnapLink is a registered trademark of Morning Star Technologies, Inc.

[3] Perkins, D., 'Point-to-Point Protocol for the transmission of multi-protocol datagrams over Point-to-Point links', RFC 1171, Carnegie Mellon University, July 1990.

[4] Perkins, D. and Hobby, R., 'The Point-to-Point Protocol (PPP) Initial Configuration Options', RFC 1172, Carnegie Mellon University and the University of California at Davis, July 1990.

instructions on acquiring an up-to-date copy.

TCP header compression is described in RFC 1144[5], and reduces the number of protocol overhead bytes required for typical packets in a TCP data connection.

Fast queueing for interactive TCP packets is described by its implementation in Van Jacobson's header-compressed SLIP[6].

4. Sample Configuration

We'll go through the steps necessary to configure a simple but illustrative installation. Our system, *robin*, has a Telebit[7] T1600 asynchronous dialup modem on its /dev/ttya, which it uses for its on-demand connection to *lark*. It is also connected to an adjacent machine, *sparrow*, via a short null-modem cable on its /dev/ttyb, which it uses for a connection that is constantly active.

4.1. Unpacking the distribution

You received the software distribution as a **compress**(1)ed **tar**(1) archive. Create a working directory and unpack it:

```
% ls mstppp*
mst-sparc-ppp.tar.Z
% mkdir /usr/local/src/ppp
% cp mstppp.tar.Z /usr/local/src/ppp
% cd /usr/local/src
% zcat ppp/mstppp.tar.Z | tar xvf –
...lots of output...
%
```

4.2. Installing the software and documentation

Become the superuser, and run the installation shell script:

```
% su
Password: (enter your root password here)
# cd ppp
# ./Install
```

If necessary, the **Install** script will ask you questions about your system. It will place the daemon into **/usr/etc/pppd** and put the manual pages into the appropriate locations. It will create a shell script named **ppp.uninstall**, to be used in case you need to remove the software from your system.

4.3. Building and installing a kernel

The contents of the file /usr/local/src/ppp/README will describe how to configure and build a new kernel, move it into place, and reboot your system to ensure that it runs correctly. If your system supports dynamically loadable device drivers, you may need only to reboot for them to become available.

The README file also describes how to configure the asynchronous terminal interfaces on your machine, if any special arrangements are necessary.

---

[5] Jacobson, V., 'Compressing TCP/IP Headers for Low-Speed Serial Links', RFC 1144, Lawrence Berkeley Laboratory, February 1990.

[6] Jacobson, V., ftp.ee.lbl.gov:cslipbeta.tar.Z, if_sl.c, UTSL.

[7] Telebit is a trademark of Telebit Corporation.

4.4. Arranging for PPP connections

One tunnel driver must be installed in the kernel for each remote host with which we will establish a link, no matter which end of the link initiates the connection and no matter how many simultaneous links will be in operation. To make configuration easier, if you say 'pseudo-device tun' in the kernel configuration file, it will be the same as saying 'pseudo-device tun16'. This will allow you to grow up to sixteen neighbors before rebuilding your kernel with a specification like 'pseudo-device tun17'.

The number of PPP neighbors a system can have is independent of the number of serial ports it contains, though it is constrained to that many simultaneous connections. Outbound PPP links (initiated by this system) are independent of inbound PPP links (initiated by a neighbor system). If a link might be initiated upon demand in either direction, then each system must arrange for both an outbound and an inbound connection.

4.4.1. Arranging for outbound PPP connections

Whenever *robin* is booted, one PPP daemon must be started for each outbound link. In this case, since *robin* will call *lark* on demand, we add the following lines to the appropriate spot in *robin*'s boot-time shell script (**/etc/rc.local** on our example machine):

```
if [ −f /usr/etc/pppd ]; then
        if [ −f /usr/adm/pppd.log ]; then
                mv /usr/adm/pppd.log /usr/adm/OLDpppd.log
        fi
        pppd robin:sparrow auto up
                (echo −n ' pppd(sparrow)')      >/dev/console
        pppd robin:lark auto idle 150
                (echo −n ' pppd(lark)')         >/dev/console
fi
```

The daemons started here will look through **/etc/ppp.systems** to find entries for their destinations.

4.4.2. Arranging for inbound PPP connections

An account must be created for each machine that will initiate PPP connections to *robin*. In this case, since we expect *lark* to dial us on demand, we add an entry to *robin*'s **/etc/passwd** file:

```
# cat >> /etc/passwd
Plark::105:20:Lark's PPP login:/tmp:/usr/local/etc/ppplogin
^D
# passwd Plark
New password: some password
Retype new password: some password
#
```

where **/usr/local/etc/ppplogin** is an executable shell script that looks like

```
# cat > /usr/local/etc/ppplogin
#!/bin/sh

PATH=/usr/bin:/usr/etc:/bin

mesg n
stty −tostop
exec pppd `hostname`: idle 130 crtscts
^D
# chmod 755 /usr/local/etc/ppplogin
#
```

The PPP user's login shell script can, of course, be located anywhere and be named almost anything; we

just use the above for purposes of illustration. On a NeXT[8] workstation, the name of the user's login shell script may not contain the letter 'r', because of an undocumented feature that would cause /bin/sh to consider itself to be a security-restricted shell.

In most cases, all inbound PPP logins can use the same ppplogin script. If you want to ensure that only a particular host can use a particular login, then have that login account use a private **/usr/local/etc/ppplogin-***host* that looks something like

```
# cat > /usr/local/etc/ppplogin-sparrow
#!/bin/sh

PATH=/usr/bin:/usr/etc:/bin

mesg n
stty −tostop
exec pppd ‘hostname‘:sparrow idle 130 crtscts
^D
# chmod 755 /usr/local/etc/ppplogin-sparrow
#
```

4.5. Creating the PPP configuration files

Now we can start building our configuration files. See **ppp.devices**(5), **ppp.dialers**(5), **ppp.filter**(5), **ppp.systems**(5), and **pppd**(8) for a description of all the configuration options, and see the **Examples/** subdirectory in the software distribution for some example configurations. (Each complete entry in **ppp.dialers** and **ppp.systems** must be typed on a single line. The **ppp.dialers** example here is broken to fit on the paper.)

```
# cd /etc
# cat > ppp.devices
TB-V32        cua     38400 crtscts
^D
# cat > ppp.dialers
TB-V32        ABORT NO\sCARRIER ABORT BUSY TIMEOUT 5 "" AT
        OK-AT-OK ATZ OK \c TIMEOUT 30 "" ATX1S111=0DT\T
        CONNECT \c \n
^D
# cat > ppp.filter
default  bringup  !ntp !3/icmp !who
           pass       !who
           keepup   !ntp !3/icmp !who
           log        !nntp tcp/syn/recv
^D
# cat > ppp.systems
lark Any;50 ACU 38400 5551212 "" "" ogin: Probin word: \qmypasswd
sparrow Any;50 cub 38400 0 "" \r\r ogin: Probin word: \qmypasswd
^D
# chmod 644 ppp.devices ppp.dialers ppp.filter
# chmod 600 ppp.systems
```

[8] NeXT is a trademark of NeXT Computer, Inc.

4.6. Testing the installation

Our PPP configuration is now complete, so we can go ahead and start it up for testing:

```
# pppd robin:lark auto
#
```

And make sure it got started:

```
# cat /usr/adm/pppd.log
8/22-17:48:54-5997 Morning Star Technologies PPP
8/22-17:48:54-5997 Version 1.0 [22-Aug-1991 17:44:28]
8/22-17:48:54-5997 du0: pppd robin:lark auto
#
```

And bring up the link:

```
# telnet lark
Trying lark ...
Connected to lark.
Escape character is '^]'.


SunOS 4.1 UNIX (lark) (ttyp3)


lark login: ^DConnection closed by foreign host.
#
```

And watch how the link was initiated:

```
# cat /usr/adm/pppd.log
8/22-17:48:54-5997 Morning Star Technologies PPP
8/22-17:48:54-5997 Version 1.0 [22-Aug-1991 17:44:28]
8/22-17:48:54-5997 du0: pppd robin:lark auto
8/22-17:49:45-5997 tcp 137.175.2.11/1204 –> 131.187.1.131/telnet 40 syn bringup
8/22-17:49:45-5997 Dialing lark (cua/38400/5551212/TB-V32)
8/22-17:49:23-5997 Call succeeded
8/22-17:49:37-5997 PPP connected to lark
8/22-17:50:34-5997 tcp 137.175.2.2/smtp <- 129.221.8.2/2547 40 syn
#
```

This log file snippet shows that our *telnet* session to *lark* was the cause for the link being initiated. It also shows that the 'log' line in **ppp.filter** is operating as expected, and that the first incoming TCP connection was to deliver mail.

Morning Star PPP is installed, configured, running and connected on both of our links. When the link is up, users should be able to access *robin* from a remote machine via *rlogin*. You should also be able to *telnet*, *ftp*, etc. between the two. If either machine is connected to a local area network, you can set set up IP routing on the two networks so that hosts on either can communicate with hosts on the other, using the machines on the ends of the PPP link as routers.

5. IP Routing tips

The UNIX host's IP implementation sees MST PPP as a point to point network connection between two known addresses. If neither endpoint resides on an IP-based local area network (LAN), packets will simply flow in both directions as soon as the PPP connection is established. When the PPP link connects a remote host to a LAN, or provides a connection between two LANs, or connects a host or a LAN to the worldwide Internet, the systems involved must be concerned with routing.

In the examples below, we'll describe how to establish static routes when the machines boot. This is cumbersome, because any topology change requires that all the machines even remotely involved be

reconfigured by editing their boot-time shell scripts. An alternative to static routes would be to use some automated system for updating all the hosts' routing tables. This is usually implemented as a daemon running on all the hosts involved. The RIP protocol[9] used by *routed* is generally considered inadequate for describing transient point to point links. You would do better to install the Gate Daemon[10]. If you use *gated*, you should probably invoke *pppd* with the **netmask** argument set to the same value as used on the LAN.

5.1. Connecting a host to a LAN

There are two reasonable approaches for connecting a set of standalone hosts to a LAN via PPP.

5.1.1. Separate Network

The method that will cause the least confusion is to assign a network or subnet for use by all the remote machines. For example, suppose the LAN is network 134.19.5.0 and is populated by hosts *alpha* (134.19.5.22) and *bravo* (134.19.5.41). A modem is attached to one of *alpha*'s serial ports, and *alpha*'s *ppplogin* shell script is as described in the first example above. A laptop SPARCstation clone named *charlie* wishes to connect to the network.

Designate subnet 45 for remote machines. Assign *charlie* the IP address 134.19.45.17. The MST PPP daemon on *charlie* would be started from */etc/rc.local* as

```
/usr/etc/pppd 134.19.45.17:134.19.5.22 auto idle 180
```

or, if all the names can be resolved locally,

```
/usr/etc/pppd charlie:alpha auto idle 180
```

The next line in *charlie*'s */etc/rc.local* would set up an IP route through the dialin gateway:

```
/usr/etc/route add net 134.19.0.0 alpha 1
```

or, if the LAN were also connected to the Internet,

```
/usr/etc/route add default alpha 1
```

Similarly, the PPP daemon on *alpha* would be started as

```
/usr/etc/pppd alpha:charlie auto idle 180
```

bravo's /etc/rc.local would contain a line like

```
/usr/etc/route add net 134.19.45.0 alpha 1
```

5.1.2. Proxy ARP

Proxy ARP[11] can be used to trick the normal ARP implementation[12] used by hosts on a broadcast LAN into sending packets for several IP addresses to the same LAN address. Assign *charlie* an address on the same subnet as the other machines on the LAN, such as 134.19.5.42. Then run a Proxy ARP daemon on some host directly connected to the LAN, configured to respond to ARP requests for *charlie*'s IP address by returning *alpha*'s Ethernet[13] address. No other routing should be necessary on the LAN, but *charlie*'s routes should be set up as described above.

---

[9] Hedrick, C.L., 'Routing Information Protocol', RFC 1058, Rutgers, June 1988.

[10] gated.cornell.edu:pub/gated/gated-2.0.1.10.tar.Z or gated-alpha.tar.Z.

[11] ftp.cis.ohio-state.edu:pub/proxyarp/proxyarpd.shar.Z

[12] Plummer, D.C., 'Ethernet Address Resolution Protocol', RFC 826, Symbolics, November 1982.

[13] Ethernet is a trademark of Xerox Corporation.

5.2. Connecting two LANs

Suppose *delta* (192.9.200.42) and *echo* (192.9.200.66) are on one LAN, with *delta* supporting a modem. Also suppose *foxtrot* (134.19.14.29) and *golf* (134.19.14.102) are on another LAN across town, with *foxtrot* supporting a modem. *foxtrot*'s PPP daemon should be started as

```
/usr/etc/pppd foxtrot:delta auto idle 130
```

and *foxtrot* should have a route like

```
/usr/etc/route add net 192.9.200 delta 1
```

golf should have a route like

```
/usr/etc/route add net 192.9.200 foxtrot 1
```

Similarly, *delta*'s PPP daemon should be started as

```
/usr/etc/pppd delta:foxtrot auto idle 130
```

and *delta* should have a route like

```
/usr/etc/route add net 134.19.14.0 foxtrot 1
```

or, depending upon the structure of the LAN occupied by *foxtrot* and *golf*, maybe

```
/usr/etc/route add net 134.19.0.0 foxtrot 1
```

echo should have a route like

```
/usr/etc/route add net 134.19.14.0 delta 1
```

or, again depending upon the structure of the LAN occupied by *foxtrot* and *golf*, perhaps

```
/usr/etc/route add net 134.19.0.0 delta 1
```

5.3. Connecting a host or LAN to the Internet

If your LAN is connected to the Internet, or if you have arranged an account at a local Point Of Presence (POP) of an IP connectivity vendor, then you should arrange for your *default* route to point through the gateway at the other end of the PPP connection. If *hotel* supported a modem to call a POP, it would start its PPP daemon like

```
/usr/etc/pppd hotel:pop.foo.net auto idle 240
```

and would arrange a route as

```
/usr/etc/route add default pop.foo.net 1
```

Any hosts on the LAN 'behind' *hotel* would set a route like

```
/usr/etc/route add default hotel 1
```

A machine's default route should point to the next machine along its path 'outward' to the Internet. If *hotel* were a remote machine dialing into one machine in a complex corporate internet, its default route should point to that hub machine, expecting that the hub will deal with the issues of routing *hotel*'s packets to their destination, whether on the LAN or elsewhere on the corporate Internet or onto the Internet.

6. Finer Points of Configuring PPP

Here we'll offer suggestions, hopefully helpful ones, about subtle points of configuring and using **pppd**.

6.1. Active vs. Passive

pppd can be used on either end of a connection, either as the calling system or as the answering system. A **pppd** that's being used as the calling system uses the **auto** option on the command line, and one that's answering does not use the **auto** option. Its default behavior as the calling system is to wait until the

answering system initiates a protocol negotiation. Its default behavior as the answering system is to immediately begin a protocol negotiation as soon as it can open the line, anticipating that another PPP implementation will be waiting there, ready to begin the negotiation. This combination of behaviors seems to bring links up as quickly as possible.

In some situations, it is better to modify these defaults. For example, when a Morning Star PPP is dialing into a machine running another implementation, you may find that the answering system uses an initial Passive Open, so that both ends of the PPP circuit are waiting for the other end to start. In this case, it may be necessary to start the daemon with the **active** option along with the **auto** option.

Some PPP implementations expect to be the active initiator in the negotiation process, and the negotiation progresses much more slowly if both ends are active. If a system running such a PPP dials into a system running Morning Star PPP, the answering **pppd**'s default active behavior may cause problems. In this case, it may be necessary for the ppplogin script to invoke **pppd** with the **passive** option.

The current **pppd** implementation's **active** and **passive** options control the finite state machine's behavior while negotiating both LCP and IPCP options. This is appropriate for most environments in which **pppd** is currently used, but future versions will provide separate controls for the behavior of each negotiation phase.

6.2. The Idle Timer

The **idle** option may be used for both the calling and the answering invocation of **pppd**. Whichever specifies the shorter interval will initiate the line shutdown. But how should idle timer values be selected, and should they ever not be used?

When **pppd** is talking to a system running PPP software that does not implement on-demand auto-dialing, it should probably not take the link down and expect the sessions to stay intact, as they would if the other end were running MST PPP. These system include those running most free PPP implementations, and MS-DOS PCs[14] running FTP Software's PC/TCP (see below).

Idle timer values should be selected based on the scarcity of resources and the cost of maintaining a connection. The calling system should set a relatively brief idle timeout if the connection carries a monetary cost per minute that it is open - e.g. a long-distance telephone call. On the other hand, if the telephone billing scheme is based on the number of calls rather than their duration, the calling system would probably prefer to use a longer idle timer, or none at all.

The answering system should set a relatively brief timeout if it can accommodate a limited number of incoming connections because of e.g. a lack of sufficient modems to service all the incoming requests. The answering system might also use a shorter idle-timeout value if it were acting as a hub and providing its services via a toll-free WATS (800) number. On the other hand, if it were providing its services via a 976 or 900 number scheme, it may not want to specify any timeout interval at all, since bills for the calling system's connect time would put funds in the answering system's pocket.

7. Observing Pppd's Progress

pppd records its actions in two places: the argument vector and the log file.

7.1. The Argument Vector

As **pppd** runs, it updates its argument vector. You can see its progress by typing 'ps ax | grep ppp'. The first field inside the '[' is the state of the connection: dialing, connecting, disconnecting, up, down, or a description of the reason for a connection failure. The second field is the name of the serial interface device assigned to this connection. If an inbound **pppd** is using **/dev/ttya**, the space would say **ttya**. If an outbound **pppd** is using **/dev/cub**, the space would say **cub**. The third and fourth fields of the argument vector, separated by a slash ('/'), are the recent reception and transmission rates, respectively, of data over the modem, in bits per second. The fifth field is the idle timer contents: the number of seconds that have passed since the link last carried a packet described in the *keepup* line. The trailing ']' indicates the end of

[14] DOS is a International Business Machines Corporation. MS-DOS is a registered trademark of Microsoft Corporation.

the status segment of the argument vector.

The remaining fields are the actual arguments with which **pppd** was invoked.

7.2. The Log File

The log file (default **/usr/adm/pppd.log**) also contains progress reports about the daemon's progress. Higher debugging levels provide more detailed logging output. At extremely high debugging levels such as 7 or 8, the log file will contain a record of each byte sent or received. Such extreme detail should be avoided for normal operations, since the log file can consume prodigious amounts of disk space, and since the very act of logging can occasionally cause protocol timeout problems. Lower debugging levels can show progress through PPP's finite state machine, or allow observation of the daemon's progress through the chat scripts in **ppp.systems** and **ppp.dialers**.

8. Observations about on-demand dialup IP

8.1. Application behavior

The scheme used to support on-demand dialups in Morning Star PPP depends upon network- and higher-layer protocol timeouts being typically longer than the time needed to allocate a modem, dial it, complete the call, login to the remote system, start up the PPP protocol handler there, and establish LCP and IPCP connectivity. This is a reasonable assumption, since a local call will typically require about 25 seconds between the daemon's receipt of the first 'bringup' packet and the establishment of IP-level connectivity (of that, more than 20 are consumed with dialing the call and training the modems). This scheme works even if the first remote modem is busy and the dialing pppd must try a second telephone number, so long as the dialer script is carefully written to include appropriate ABORT and TIMEOUT strings. The user's application will simply wait for the connection to be established and continue as if the connection had always been in place. This even works well when a link times out with active TCP sessions in place, such as a *telnet* connection to the remote end. When the user types, or when an application writes output, a new call is placed from either end and the session is re-established without TCP or the application above it noticing anything was amiss.

Unfortunately, if the connection cannot be re-established, TCP will time out and the application may fail ungracefully. This most often happens during times of resource contention (no free modems to place or answer the call) or acute connectivity problems (telephone system call-routing breakdown or extreme service degradation). The effect will be the same as if the machine at the 'other' end had crashed ungracefully, since the application will only be able to tell that it is unable to exchange packets with its partner.

8.2. Dealing with bandwidth limitations

The Internet's TCP/IP protocol suite is remarkably flexible and robust, with a wide variety applications for use over media ranging from multigigabit data superhighways to long-delay satellite channels to local area networks to dialup modems to amateur radio to carrier pigeons[15]. While by definition all IP-based applications may be transparently used over links traversing any combination of media, some applications are better suited to media with the relatively low bandwidth and moderate latency and delay characteristics of asynchronous dialup connections.

8.2.1. Applications with few problems on thin wires

PPP over current-technology dial modems[16] provides satisfactory performance for most typical TCP/IP and UDP/IP wide-area applications. Especially with TCP header compression, users will experience useful file transfer throughput (V.32/V.42bis yields 1.7−2.8 Kbytes/sec for a typical large binary data file) and reasonable interactive response in applications such as *telnet*, *rlogin*, and the like. Plenty of bandwidth is available for network infrastructure applications such as SMTP electronic mail, NNTP network news, NTP

[15] Waitzman, D., 'Standard for the transmission of IP datagrams on avian carriers', RFC 1149, BBN, April 1990.

[16] CCITT V.32 (9600 baud) or V.32bis (14400 baud) carrier, with V.42 error correction and V.42bis data compression, and the host serial interface latched at 38400 baud

clock synchronization, domain name service, etc.

8.2.2. Applications with some problems on thin wires

The speed limitation of the dialup medium becomes more apparent when larger amounts of data are transferred, particularly for interactive applications such as *xterm* or other display-oriented programs. While large bitmap editors and CAD packages will operate correctly, users may find that the modem simply isn't able to keep up with their demands for interactive responsiveness.

8.2.3. Applications to avoid on thin wires

Applications that are extreme users of communications bandwidth, such as Sun's Network File System (NFS[17]), should be avoided over dialup links except in cases of dire necessity. Not only is NFS able to easily overwhelm a modem link, it is a UDP-based application, so the user will not enjoy the packet overhead reduction benefits of TCP header compression. While performance may be marginally acceptable for read-only file access, NFS's synchronous write semantics will make file creation and writing quite painful. Attempting to execute a program stored on a remotely-mounted filesystem will have this same problem, since it will be swapped to the same file from which it was read.

Fortunately, there are usually other approaches, more appropriate for wide-area networking over low-bandwidth links, to solving the problems that NFS addresses. Use Berkeley's *rdist* system to keep the important segments of two sites' file structures in reasonable synchrony. Also, use a package like ange-ftp.el[18] in your GNU Emacs to automatically manage FTP connections that are transparently associated with your editing buffers. This gives the appearance of the ability to edit files while they reside on a remote machine, without needing to use NFS to mount the remote filesystems locally.

If, even after the above attempts at dissuasion, you still think that you absolutely must use NFS, set the *rsize* and *wsize* parameters on the mount request to something like 1400 (to fit safely within the usual PPP MRU/MTU of 1500, even with possible special-character escaping as per the *asyncmap*), and increase the *timeo* timeout parameter to accommodate network delays.

9. Cabling

Most systems and modems use the EIA RS-232[19] signalling conventions for their asynchronous serial ports. In the tables below, there is no difference between the pins referred to as 'DTE pins' or 'Modem pins', nor between 'Local pins' and 'Remote pins', except notational convenience. If a cable has the same physical connector type on each end (both DB-25 or both MiniDIN-8), it is completely symmetrical and may be flipped end-for-end with no effect.

9.1. Asynchronous Modem cables

If your system is to be connected to an asynchronous modem, and if the system uses DB-25 connectors, the pins on the system (DTE) end of the cable should be connected straight through to the same-numbered pins on the DB-25 modem end.

---

[17] NFS is a trademark of Sun Microsystems, Inc.

[18] Written by Andy Norman <ange@hplb.hpl.hp.com>, available from ftp.cis.ohio-state.edu:pub/gnu/emacs/elisp-archives/packages/ange-ftp.el.Z

[19] Electronic Industries Association, EIA Standard RS-232-C, 'Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange', August 1969.

In order of importance for proper operation, the pins you should connect are

DTE DB-25 pin	**RS-232 Signal**	**Modem DB-25 pin**	**Direction**
1	Protective Ground	1	both
7	GND (Signal Ground)	7	both
2	TD (Transmit Data)	2	from DTE
3	RD (Receive Data)	3	from Modem
6	DSR (Data Set Ready)	6	from Modem
20	DTR (Data Terminal Ready)	20	from DTE
4	RTS (Request To Send)	4	from DTE
5	CTS (Clear To Send)	5	from Modem
8	DCD (Data Carrier Detect)	8	from Modem
22	RI (Ring Indicator)	22	from Modem

NeXTstations' serial ports have MiniDIN-8 connectors. A cable to connect a NeXTstation to a DB-25 asynchronous modem looks like[20]

DTE MiniDIN-8 pin	**RS-232 Signal**	**Modem DB-25 pin**	**Direction**
1	DTR	20	from DTE
2	DCD	8	from Modem
3	TD	2	from DTE
4	GND	7	both
5	RD	3	from Modem
6	RTS	4	from DTE
7	*N/C*	*N/C*	*N/C*
8	CTS	5	from Modem
N/C	DSR	6	from Modem
N/C	RI	22	from Modem

9.2. Asynchronous 'Null-Modem' cables

If your system is to be connected directly to another system, the pins on the local system's end of the cable should be connected with the symmetric-function pins on the remote system's end. This is commonly called a null-modem cable.

[20] 'Network and System Administration', Chapter 13 'Attaching Peripheral Devices', section on 'Serial Port Connector and Cable Specifications', pps 186−188, NeXT Computer Inc., 1990.

In order of importance for proper operation, the pins you should attach between two systems with DB-25 connectors are

Local DB-25 pin	**Local RS-232 Signal**	**Remote DB-25 pin**	**Remote RS-232 Signal**
1	Protective Ground	1	Protective Ground
7	GND	7	GND
2	TD	3	RD
3	RD	2	TD
6	DSR	20	DTR
20	DTR	6	DSR
4	RTS	5	CTS
5	CTS	4	RTS

NeXTstations' serial ports have MiniDIN-8 connectors. A cable to connect a NeXTstation to another system that has DB-25 connectors looks like

Local MiniDIN-8 pin	**Local RS-232 Signal**	**Remote DB-25 pin**	**Remote RS-232 Signal**
1	DTR	8	DCD
2	DCD	20	DTR
3	TD	3	RD
4	GND	7	GND
5	RD	2	TD
6	RTS	5	CTS
7	*N/C*	*N/C*	*N/C*
8	CTS	4	RTS
N/C	DSR	6	DSR
N/C	RI	22	RI

A cable to connect a pair of NeXTstations looks like

Local MiniDIN-8 pin	**Local RS-232 Signal**	**Remote MiniDIN-8 pin**	**Remote RS-232 Signal**
1	DTR	2	DCD
2	DCD	1	DTR
3	TD	5	RD
4	GND	4	GND
5	RD	3	TD
6	RTS	8	CTS
7	*N/C*	7	*N/C*
8	CTS	6	RTS

10. Flow control

If your UNIX system supports it, use out-of-band 'hardware' (RTS/CTS) flow control between your system and your modem, or between two systems over a null-modem cable. On a Sun system, specify 'crtscts' either on the *pppd* command line, or in the Optional Parameters field of the line in **ppp.devices**(5). On a NeXTstation, specify 'cufa' in the Device field in **ppp.devices**(5).

If, for whatever reason, you are forced to use in-band 'software' (XON/XOFF, ^Q/^S) flow control, then specify 'xonxoff' on the *pppd* command line or in **ppp.devices**(5), and use 'cua' on a NeXT.

The default is to use no flow control.

SunOS 4.1.1 implements hardware flow control on the native CPU serial ports only for transmission[21]. Workstations running this OS respect the modem's use of CTS to indicate its readiness to receive characters from the host, but do not use RTS to throttle the modem's data stream to the host. This means that when presented with a heavy incoming traffic load, the Sun's native CPU serial ports will be overloaded, and the console will report a series of occasional 'zs1: silo overflow' errors. This is unfortunate but unavoidable by PPP, introduces no errors into the data being transferred across the network, and does not measurably affect the PPP link's throughput or responsiveness.

11. Modem parameters

11.1. General Recommendations

You should connect your modem to your computer at the highest asynchronous serial speed that both can support (typically 38400 baud, as found in /usr/include/sys/ttydev.h), and allow the modem's internal carrier speed matching capability to make a usable connection with whatever type of modem is in use at the other end. This will allow you to take maximum advantage of the modem's data compression capabilities.

If your modem supports error correction, you should enable its use. This will allow line noise problems to be corrected at as low a level in the protocol stack as possible. If the modem has both MNP[22] level 4 and CCITT V.42, choose the latter so that you can also use CCITT data compression.

If your modem supports data compression, you probably should enable its use unless your tests show your application has preferable performance attributes without it. This will allow you to pass as much information across the PPP line as possible. If the modem has both MNP5 and CCITT V.42bis, choose the latter because it has a higher maximum compression ratio and because it behaves better with precompressed data streams.

11.2. Specific examples

In the file **Examples/ppp.dialers.ex** you'll find dialer descriptions for a few types of modems. Since your modems will likely be used for purposes besides PPP (e.g. UUCP or interactive users), it's best to set them up for efficiency in all possible uses. The PPP-specific register settings in ppp.dialers ensure that an otherwise general-purpose modem will work as well as possible with PPP, for the duration of the PPP session.

In the examples below, we'll show how to configure a typical modern modem, including a picture of its working register settings and a command string that will set it up correctly. We'll also describe what each of those register settings mean, so that if you use a different type of modem, you should be able to set it up similarly.

We find that a Telebit T1600 works well with the following register settings:

```
OK
at&v
T1600 – Version LA1.00 – Active Configuration
 B1  E1  L3  M0  Q2  T   V1  X12  Y0
&C1 &D3 &G0 &J0  &L0 &Q0 &R3 &S1  &T4 &X0
S000:1    S001=0    S002=43   S003=13   S004=10   S005=8    S006=2    S007:120
S008=2    S009=6    S010=14   S011=70   S012=50   S018=0    S025=5    S026=1
S038=0    S041=0    S045=0    S046=0    S047=4    S048=0    S050=0    S051:6
S056=17   S057=19   S058:2    S059:15   S060=0    S061:0    S062=15   S063:2
S064=0    S068=255  S069=0    S090=0    S093=8    S094=1    S100=0    S102=0
S104=0    S105=1    S111:0    S112=1    S180=2    S181=1    S183=25   S190=1
S253=10   S254=255  S255=255
OK
```

[21] See the Output Modes section of **termio**(4) and the IOCTLS section of **zs**(4s).

[22] MNP is a trademark of Microcom Inc.

You can make a T1600 work like this by giving it the following command:

at&fs0=1s7=120s51=6s58=2s59=15s61=0s63=2s111=255l3m0q2tx12&c1&d3&s1&w

This accomplishes the following:

&f	Loads factory default settings.
s0=1	Answer after one ring.
s7=120	Wait 120 seconds for a valid carrier tone to be sent from the remote modem.
s51=6	Latch the DTE interface at 38400 bps.
s58=2	Use full-duplex RTS/CTS flow control. When RTS is OFF, the modem will not send data to the local DTE. When RTS is ON the modem sends data to the local DTE.
s59=15	Use the following result code suffixes: Possible link protocol suffixes: nothing, REL, or LAPM Possible compression suffixes: nothing or COMP
s61=0	Local reaction to a break signal on the serial interface is the same as specified in S63.
s63=2	When a break signal is transmitted by the local DTE during an error controlled connection, the modem will discard any buffered data and send the break.
s111=255	Use the file transport protocol spoofing mechanism specified by the remote modem. If both modems have S111=255, no protocol is spoofed.
l3	Sets the speaker volume high.
m0	Turns off the speaker.
q2	Modem reports result codes when originating a call, but does not return result codes when answering a call.
t	Modem uses DTMF tone dialing.
x12	Result codes include the following: BUSY, DIALING, ERROR, NO CARRIER, NO DIALTONE, OK, RING, RRING, CONNECT 300, CONNECT 1200, CONNECT 2400, CONNECT 4800, CONNECT 7200, CONNECT 9600, CONNECT 12000, CONNECT 14400, CONNECT 19200, and CONNECT 38400.
&c1	When a carrier is detected from the remote modem, the DCD signal is turned ON after the CONNECT result code is sent to the DTE. The DCD signal is turned OFF when the carrier is dropped.
&d3	The modem resets and enters command mode when the DTR signal is switched from ON to OFF. This causes the modem to recall the current user configuration parameters from nonvolatile memory.
&s1	DSR is ON after the answer tone is detected and stays ON throughout the connection.
&w	Saves the current configuration settings to nonvolatile RAM.

If you have some other sort of modem that's not described here, you should configure it with similar functions (e.g. RTS/CTS flow control) using that modem's own command set and register semantics.

12. Interoperability with other PPP implementations

12.1. The Telebit NetBlazer[23]

Morning Star PPP has been tested with version 1.20 of Telebit's NetBlazer software. We were unable to configure the NetBlazer to negotiate the use of RFC 1144 'VJ' TCP header compression, though we are assured that future versions of the NetBlazer software will incorporate this feature. It will probably not

[23] NetBlazer is a trademark of Telebit Corporation.

require the use of **pppd**'s 'old' argument.

12.1.1. Dialing into a NetBlazer from a UNIX system

This would be the arrangement to allow several remote workstations to dial into a NetBlazer serving as a hub for a LAN, or when connecting a remote workstation or LAN to the Internet through an IP connectivity vendor's POP.

12.1.1.1. Configuring the NetBlazer to receive the call from the UNIX system

The NetBlazer may be set up as if the UNIX machine were just another remote 'dynamic-interface' NetBlazer, using the PPP encapsulation protocol in 'packet mode'.

Configure the NetBlazer as described in chapter 3 of the NetBlazer Reference Manual, noting particularly the discussion of the **ipdial** command in the section on Setting Up Remote IP Destinations. When queried for 'Name of dial IP interface to add', respond with the name of the UNIX system. Morning Star PPP does not implement a 'crypto handshake' to exchange with the NetBlazer. Use PPP rather than SLIP, and specify 1500 for the MTU. When asked to enter the password for the dial-in user for use by the UNIX system, note that password for later inclusion in the UNIX system's **/etc/ppp.systems** entry that describes how to dial into this NetBlazer.

12.1.1.2. Configuring the UNIX system to call the NetBlazer

In your UNIX system's **/etc/ppp.systems**, the username provided by the login portion of the chat script will be the name of the UNIX system itself. The password will be the one provided when arranging for the dial-in user on the NetBlazer.

In your UNIX system's **/etc/rc.local**, start the Morning Star PPP daemon with the 'active' argument, because the NetBlazer starts in the LCP Passive-Open state.

```
pppd `hostname`:remote-netblazer auto active idle 130
```

12.1.2. Dialing into a UNIX system from a NetBlazer

This would be the arrangement to allow a hub NetBlazer to connect on demand to several remote workstations.

12.1.2.1. Configuring the UNIX system to receive the call from the NetBlazer

In order to accommodate a NetBlazer dialing into a UNIX system running Morning Star PPP, the following changes must be made to the UNIX system. Make an entry in **/etc/gettytab** that's a duplicate of the entry you would normally have used, except with the addition of the **:p8:** parameter (see **gettytab**(5)), like this:

```
h|std.38400|38400-baud:sp#38400:
w|P.38400|38400-baud:sp#38400:p8:
```

Then change /etc/ttytab to use the new entry, like this:

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

This change in parity may have an adverse effect on the ability of other users with other machines and other applications to dial into this port. Users of Morning Star PPP can compensate by including the pair

```
"" P_ZERO
```

in the **ppp.systems** line that instructs their system to dial into this system. Future releases of the NetBlazer software may include the ability to talk to a typical 7-bit UNIX *getty*, so the **:p8:** switch would become unnecessary, and all normal UNIX dialins would work as expected.

The login shell script for the NetBlazer's incoming connection must tell the NetBlazer what it expects to hear ('Packet mode enabled') before starting the PPP LCP option negotiation phase of the connection. Since the NetBlazer does not perform IPCP 'IP-Addresses' negotiation, the UNIX PPP must tell the

NetBlazer what IP address it should be.

```
# cat > /usr/local/etc/ppplogin-netblazer
#!/bin/sh

PATH=/usr/bin:/usr/etc:/bin

mesg n
stty −tostop
echo Packet mode enabled
exec pppd ‘hostname‘:remote-netblazer idle 130 crtscts
^D
# chmod 755 /usr/local/etc/ppplogin-netblazer
#
```

The name *remote-netblazer* in the above shell script should be replaced with the IP address of the NetBlazer that will be connecting, or its name if its address can be found in the local hosts table or can be resolved by the local name server before the NetBlazer link is available.

12.1.2.2. Configuring the NetBlazer to call the UNIX system

Add the following entry to the NetBlazer's CHAT.TXT file:

```
# For a UNIX system running Morning Star Technologies PPP
#  no crypto challenge/response, longer timeouts, short expect strings
#  login shell script must ‘echo Packet mode enabled‘ before ‘exec pppd‘
:unix
e1,20,30,2,in:
u2,5,100,3
s3,5,100,4,
e4,10,30,5,word:
o99,Packet mode enabled
p5,5,100,6
s6,5,100,7,
e7,20,30,99,Packet mode enabled
o32,in:

s30,5,100,31,
e31,20,100,32,in:
u32,5,100,33
s33,5,100,34,
e34,10,100,35,word:
o99,Packet mode enabled
p35,5,100,36
s36,5,100,37,
e37,20,100,99,Packet mode enabled
o100,in:

r99,0
r100,1
```

When configuring the NetBlazer's dynamic interface to call the UNIX system as described above in the section on **Configuring the NetBlazer to receive the call from the UNIX system**, specify ‘unix’ for the chat script, rather than the default ‘ics’.

12.2. FTP Software Inc's PC/TCP[24] Network Software for DOS

Morning Star PPP interoperates with version 2.05 of FTP Software Inc's PC/TCP PPP (PC-227) implementation.

12.2.1. Getting ready for a DOS PC to dial into a UNIX system

The login script used by the PC should specify the IP addresses of both the UNIX system and the PC, and should specify **passive** for option negotiation purposes. It should not specify an idle timer because PC/TCP is unable to re-open the connection.

12.2.2. Getting a DOS PC ready to dial into a UNIX system

The PC should be configured as described in FTP Software's manual. No special arrangements are required in order to talk to a UNIX system running MST PPP.

13. Getting Help

Call Morning Star Technologies's technical support staff at +1 800 558 7827 or +1 614 451 1883, or address electronic mail to them as

Support@MorningStar.Com

Join the electronic mailing list of Morning Star PPP users to share hints, tips, critiques, feature suggestions, questions about configuration quirks, and anything else related to the product. MST will announce new releases and newly-supported hardware on the list. Our engineering, support, and marketing departments are well represented on the list, so that we can listen to your concerns and offer help too.

To send a message to the other members of the list, address it to

PPP-users@MorningStar.Com

To request addition to the list, write to

PPP-users-request@MorningStar.Com.

14. Credits

pppd was written by Karl F. Fox <karl@MorningStar.Com>, with design and implementation assistance from Robert A. Sutterfield <bob@MorningStar.Com>, Kim L. Toms <kim@MorningStar.Com>, and Steven K. Wilson <steve@MorningStar.Com>. The documentation was written by Robert A. Sutterfield <bob@MorningStar.Com>. Parts of the PPP protocol engine are derived from code written by Drew D. Perkins <ddp@andrew.cmu.edu>, now <ddp@interstream.com>.

15. Copyright Information

This software and associated documentation is Copyright 1991 Morning Star Technologies, all rights reserved. It contains software developed at Carnegie Mellon University which is Copyright 1989 Carnegie Mellon University, all rights reserved.

[24] PC/TCP and FTP Software are registered trademarks of FTP Software, Inc.

NAME

tun – IP network tunnel driver

CONFIG

pseudo-device tun[*n*]

SYNOPSIS

#include <sys/tun.h>
open("/dev/tun*n***", mode);**

DESCRIPTION

When IP packets are written to **/dev/tun***n*, they will be received by the kernel's IP layer on the network interface **tun***n*. When the kernel's IP layer sends packets to the IP interface **tun***n*, they will be available for reading on **/dev/tun***n*.

Instead of having hardware and an associated kernel interface that support network functions, the **tun** driver allows a network interface to be implemented as a user-space process. While talking to the same set of tunnel drivers on the same system, different network interface processes can implement different IP encapsulation methods, such as RFC877 for use over CCITT X.25-based public data networks, or RFC1171 PPP for use over dedicated lines and dialup modems.

The **tun** driver provides support for a pair of devices collectively known as an *IP tunnel*. The two devices comprising a tunnel are known as the *inbound* and *outbound sides*, similar to the pairing between **/dev/tty***n* (the inbound terminal) and **/dev/cu***n* (the outbound 'auto-call unit' available on many systems). The outbound side's minor device number is that of the inbound side plus 128, though they together appear to IP as one interface. If both the inbound and outbound sides of a tunnel device are open, packets received from IP are delivered to only the inbound side.

If a TCP packet received from IP is part of a telnet, rlogin, or FTP command stream, it will be put in a *fast queue*. All packets in the fast queue are delivered to the user before any packets in the normal queue.

IOCTLS

A few special **ioctl**s are provided, to supply the functionality needed by applications programs to emulate real hardware interfaces. The complete list of supported **ioctl**s is:

SIOCSIFADDR Sets the interface address.

SIOCSIFFLAGS Allows changing all the usual **if** flags, as well as

IFF_POINTOPOINT Configure this interface as one end of a point-to-point network.

IFF_BROADCAST Configure this interface for a broadcast-capable network.

SIOCGIFMTU Returns the Maximum Transmission Unit (in octets) for a packet sent over this interface.

SIOCSIFMTU Sets the Maxumum Transmission Unit (in octets) for a packet sent over this interface. Larger packets will be sent in fragments.

TUIOSADRMD Set or clear 'address mode', in which packets read are prefaced with four octets containing the destination IP address in network byte order. The third argument is a pointer to an integer containing either a zero or a one, indicating whether 'address mode' should be cleared or set, respectively. If both 'address mode' and 'packed puffer mode' are set, each packet's destination address will come first, followed by the packet's length, followed by the packet itself.

TUIOGADRMD Get the current status of 'address mode'.

TUIOSPKBMD Set or clear 'packed buffer mode' where multiple packets are encoded in single read/write buffers. The third argument is a pointer to an integer containing either a zero or a one, indicating whether 'packed buffer mode' should be cleared or set, respectively. If set, each packet is preceded by two octets representing the next packet's length in octets. If cleared, packets will be delivered one per read(3) from the tunnel device. If both 'address mode' and 'packed puffer mode' are set, each packet's destination address will come first, followed by the packet's length, followed by the packet itself.

TUIOGPKBMD Get the current status of ‘packed buffer mode’.

TUIOSNAME Set the interface name (may only be invoked by the superuser).

TUIOGNAME Get the interface name.

FIONBIO Set or clear non-blocking mode for I/O operations.

EXAMPLES

```
#include <sys/tun.h>
int tun_fd = -1, len;
char *packet;

tun_fd = open("/dev/tun0", O_RDWR);
ioctl(tun_fd, TUIOSNAME, "du");
len = read(tun_fd, packet, size);
write(tun_fd, packet, len);
```

ERRORS

If a packet is delivered to the interface for an address family other than AF_INET, EAFNOSUPPORT will be returned.

FILES

/dev/tun0 through **/dev/tun127** ‘inbound’ tunnel devices
/dev/tun128 through **/dev/tun255** ‘outbound’ tunnel devices

SEE ALSO

if(4n), ppp.devices(5), ppp.dialers(5), ppp.filter(5), ppp.systems(5), pppd(8), RFC1171, RFC1172, RFC1144, x25-ddn(7), x25d(8), RFC877.

COPYRIGHT INFORMATION

Copyright © 1991 Morning Star Technologies Inc.; all rights reserved.

NAME

ppp.devices – PPP physical device description file format

DESCRIPTION

The file **/etc/ppp.devices** associates dialer types with physical devices and speeds. It is examined when *pppd* needs to originate a call to a neighboring machine. If no suitable speed is found, or if all devices associated with that speed are busy, *pppd* will try again later.

FORMAT

Entries are one to a line; blank lines are ignored. Comments begin with a '#' and extend to the end of the line. Upper/lower case distinctions are significant. Fields on a line are separated by horizontal white space (blanks or tabs).

Each entry must contain three or more fields, in this order:

dialer The name of the modem dialing procedure description, found in **/etc/ppp.dialers**, to use with this device

device The name of the device in the **/dev** directory (**ttya**, **cua**, etc.)

speed This field is a string to be matched against the speed field of entries in **/etc/ppp.systems** when the **/etc/ppp.systems** device field is set to ACU. Speeds must either be valid async baud-rate numbers (as found in <sys/ttydev.h>) or must begin with them (2400, 38400, 19200-PEP, etc.)

optional parameters

Any special handling for this device. Currently supported values include

crtscts Specifies that the line be conditioned for out-of-band EIA RS-232 'hardware' (RTS/CTS) flow control. The default is to use no flow control. For an outbound connection, this may be specified either in **/etc/ppp.devices** or on the *pppd* command line.

xonxoff Specifies that the line be conditioned for in-band ('software') flow control, using the characters DC3 (^S, XOFF, ASCII 0x13) to stop the flow and DC1 (^Q, XON, ASCII 0x11) to resume. The default is to use no flow control. For an outbound connection, this may be specified either in **/etc/ppp.devices** or on the *pppd* command line.

EXAMPLE

```
#
#           /etc/ppp.devices - PPP devices file
#
#Dialer     device    speed       Optional parameters
TB-PEP      cua       19200-PEP   crtscts
TB-V32      cub       38400       crtscts
```

CAVEATS

The **crtscts** flag is unsupported on the daemon's command line or in **/etc/ppp.filter** on some systems. On NeXTstations, use the devices **/dev/cufa** and **/dev/ttydfa** for dialout and dialin connections, respectively.

SEE ALSO

tun(4), ppp.dialers(5), ppp.filter(5), ppp.systems(5), pppd(8), RFC1171, RFC1172, RFC1144.

COPYRIGHT INFORMATION

Copyright © 1991 Morning Star Technologies Inc.; all rights reserved.

NAME

ppp.dialers – PPP dialer description file format

DESCRIPTION

The file **/etc/ppp.dialers** describes how to dial each type of modem attached to the UNIX® system that is to be made available for for outbound PPP calls. It is examined when *pppd* needs to originate a call to a neighboring machine.

When *pppd* selects a line from **/etc/ppp.systems**, it uses the 'speed' field to select an entry in **/etc/ppp.devices**, from which it uses the 'dialer' field to select an entry in **/etc/ppp.dialers**. *Pppd* then interprets the 'chat script' field from that dialer description.

FORMAT

Entries are one to a line; blank lines are ignored. Comments begin with a '#' and extend to the end of the line. Upper/lower case distinctions in the dialer field are significant for matching purposes, as are strings in the chat script. Fields on a line are separated by horizontal white space (blanks or tabs).

Each entry must contain two fields, in this order:

dialer — The name of this dialer, to be matched against the dialer field in **/etc/ppp.devices**

chat script — A description of the conversation that *pppd* holds with the modem. It takes the form of a word to expect the 'remote' end to send, followed by a word to send in response. After each 'send' string that doesn't end with **\c**, *pppd* sends a carriage return character (ASCII 0x0d).

CHAT SCRIPT PARTICULARS

Certain special words may be used in the chat script to control the behavior of *pppd* as it attempts to dial.

ABORT *abort-string* — If *pppd* sees *abort-string* while executing the remainder of the chat script, abort the dialing attempt and note the failure in the log file.

TIMEOUT *timeout-time* — While executing the current chat script, wait *timeout-time* seconds for a response before considering the dialing attempt to have timed out.

The expect-send couplet of **"" WORD** sets the line parity accordingly:

P_ZERO — Set the parity bit to zero.

P_ONE — Set the parity bit to one.

P_EVEN — Transmit characters with even parity. This is the default.

P_ODD — Transmit characters with odd parity.

In the midst of either an 'expect' string or a 'send' string, ^*x* gets translated into the appropriate control character, and *x* gets translated into *x*. Other special sequences are:

\s — Send or receive a space character (ASCII 0x20)

\t — Send or receive a horizontal tab character (ASCII 0x09)

\n — Send or receive a linefeed character (ASCII 0x0a)

\r — Send or receive a carriage return character (ASCII 0x0d)

\c — Don't append a carriage return character after sending this string (send only)

\d — Delay for two seconds before proceeding (send only)

\q — Don't print succeeding send strings (e.g. for a password) in any debugging output. Subsequent **\q** sequences toggle 'quiet' mode.

\T — Insert the telephone number (found in the fifth field of **/etc/ppp.systems**) here

EXAMPLE

Because of the restrictions of physical media, the following example may represent single lines from **/etc/ppp.dialers** on multiple lines of your output medium. In the actual file itself, each entry must be contained on a single line, each of which may contain up to 1023 characters.

```
#
#       /etc/ppp.dialers - PPP dialers file
#
#Dialer  Chat script
TB-PEP  ABORT NO\sCARRIER ABORT BUSY TIMEOUT 5
        "" AT OK-AT-OK ATZ OK \c TIMEOUT 60 ""
        ATX1S50=255S111=0S58=2DT\T CONNECT\sFAST
TB      ABORT NO\sCARRIER ABORT BUSY TIMEOUT 5
        "" AT OK-AT-OK ATZ OK \c TIMEOUT 30 ""
        ATX1S111=0S58=2DT\T CONNECT \c \n
TB-V32  ABORT NO\sCARRIER ABORT BUSY TIMEOUT 5
        "" AT OK-AT-OK ATZ OK \c TIMEOUT 30 ""
        ATX1S111=0S58=2DT\T CONNECT \c \n
```

SEE ALSO

tun(4), ppp.devices(5), ppp.filter(5), ppp.systems(5), pppd(8), RFC1171, RFC1172, RFC1144.

COPYRIGHT INFORMATION

Copyright © 1991 Morning Star Technologies Inc.; all rights reserved.

NAME

ppp.filter – PPP packet filter specification file format

DESCRIPTION

This file describes how on-demand PPP links are to be maintained. Normally, any packet causes the link to be brought up to its destination; any packet is allowed to traverse the link; and any packet is sufficient to keep the idle timer from causing the link to be shut down. This is rarely appropriate behavior, so the filter file allows individual selectivity based on the type of the packet and its source or destination. These selection criteria may be specified for any of the three phases of operation: bringing up the link, passing packets on the link, and shutting down the link because of inactivity. Packet logging detail may also be selected using the same criteria.

FORMAT

Comments begin with a '#' and extend to the end of the line; blank lines, or lines beginning with a '#', are ignored. Upper/lower case distinctions are ignored in hostname specifications, but are significant elsewhere. Fields on a line are separated by horizontal white space (blanks or tabs).

If a line begins with a hostname or IP address or the special word 'default', that line is considered to be the beginning of a new set of filtering specifications. If a newline is followed by white space, that line is a continuation of the filtering specification already in progress.

There are four keywords to describe the actions taken by *pppd* in response to a particular packet:

bringup Describes those packets that will cause a call to be placed and a connection initiated. If a packet would not be passed, it will also not cause the link to be brought up.

pass Describes those packets that will be allowed to traverse the link once it is established. If a packet would not be passed, it will also not cause the link to be brought up. Any packet that is not passed is optionally logged, then discarded.

keepup Describes those packets that will keep the idle timer from expiring.

log Describes those packets to be noted in the log file.

After the action keywords come words, separated by white space, describing packets that fit the criteria for that action. Each list of words is processed in the order they are found in the file. As soon as a pattern or a condition is found that matches the packet in question, *pppd* takes the indicated action and ignores the rest of the list.

Words may be names or numbers describing entries in **/etc/services**, or names or IP addresses of hosts or networks, or the special keyword 'all', which is the default for all actions except 'log', where the default is '!all'. If a network is specified, either by name or by address, then the corresponding subnet mask must also be specified if it is of a different size than the default for that class of network. Qualifications within a word are separated by slashes ('/'). The sense of each word may be individually negated by prefixing its string with an exclamation mark ('!').

In the 'log' filter specification, the special keyword 'trace' causes the contents of the indicated type of packet to be written to the log file.

Since TCP data streams are opened when the initiator sends a SYN packet to the intended recipient, *pppd* can discern between outbound (sent from this host) and inbound (coming from the other end of the link) uses of TCP applications such as telnet or FTP. The special keyword 'syn' allows filtering or logging these session starters. Qualifying it with 'recv' or 'send' allows sessions to be started or logged only if they are initiated in the indicated direction.

EXAMPLE

The following **/etc/ppp.filter** file describes the default behavior of *pppd*, either in the absence of a filter specification file or in the case of an empty file:

```
#
#       /etc/ppp.filter - PPP configuration file binding packet
#                         types to actions.
#       Describes the default behavior of the daemon
```

```
#
default     bringup all pass all keepup all log !all
#
```

If any action is unspecified in the filter file, the default is no restrictions of that sort and no packet logging.

The following **/etc/ppp.filter** file instructs the daemon that a connection to any neighbor except the host 'backbone' be brought up in response to any packet except for those generated by NTP, ICMP Destination Unreachable, and *rwhod*. If those are the only types of packets flowing across the link, it will not be kept up, but all packets are allowed to cross the link while it is up.

In the special case of the host 'backbone' (perhaps a server belonging to a network connectivity vendor), only telnet and FTP sessions, SMTP electronic mail, NNTP network news, and domain name service queries are considered sufficient cause to bring the link up or to keep it up if otherwise idle.

Once the link is up, all the above plus NTP clock chimes and ICMP messages may flow across the link. No packets to or from a particular host, nor any packets except domain name service queries and responses for any host on subnet 42 of the class B network 137.175, are ever allowed to cross the link, nor do they cause the link to be initiated. We allow telnet and FTP sessions only if they are initiated in the outbound direction.

We log occurrences of various ICMP problem messages (Unreachable, Time Exceeded, and IP header problems), including the complete contents of the latter reports. We log all telnet and FTP sessions, including inbound attempts (though they will fail because they are excluded in the 'pass' specification above). We also log the first packet of any electronic mail message flowing over this link on its way to or from a particular host.

```
#
#           /etc/ppp.filter - PPP configuration file binding packet
#                             types to actions.
#
backbone    bringup   smtp nntp 53/udp 53/tcp telnet ftp !all
            pass      !131.119.250.104
                      53/udp/137.175.42.0/255.255.255.0
                      53/tcp/137.175.42.0/255.255.255.0
                      !137.175.42.0/0xffffff00
                      53/udp 53/tcp
                      !telnet/syn/recv !ftp/syn/recv
                      smtp nntp ntp icmp telnet ftp
                      !all
            keepup    smtp nntp 53/udp 53/tcp telnet ftp !all
            log       3/icmp 11/icmp 12/icmp/trace
                      telnet/syn ftp/syn
                      smtp/syn/terminus.netsys.com
#
default     bringup   !ntp !3/icmp !who
            keepup    !ntp !3/icmp !who
#
```

RECOMMENDATIONS

Simpler filter specifications allow *pppd* to start up quicker and run faster, with less processing overhead for each packet, and are more likely to be gotten right. The silly 'backbone' example above is severe overkill for the sake of illustration, and evolved over a period of several weeks. It took the authors several tries to get it right. Start with a simple filter specification and add each special case only as the need presents itself. Then test carefully to ensure that your change had only the desired effect.

Be very careful with traffic logging and even more careful with packet content tracing. Make the selection criteria very narrow, or the log file will grow extremely large in a short period of time. Also, if the daemon is running on a diskless workstation or if the log file is on a NFS-mounted file system, increased amounts of

logging information will drastically impede the daemon's ability to process packets. Remember, NFS write(2)s are synchronous.

If you specify host names, be sure that their addresses are available locally, even with the connection down. If you find that you must bring up a connection to resolve a domain name, consider using that host's IP address (decimal numbers separated by periods) in both **/etc/ppp.filter** and **/etc/ppp.systems** instead.

If you want to specify all Domain Name Service traffic, use both 53/tcp and 53/udp instead of saying 'domain'. Some DNS traffic uses each transport, and simply saying 'domain' would be ambiguous. Similarly, some systems' older **/etc/services** files, as distributed by the manufacturer, list NTP as a TCP service. When the current UDP NTP implementation was installed on your system, the administrator may have left the old 123/tcp entry along with the correct 123/udp. The correct solution is to remove the 123/tcp entry from **/etc/services**. A workaround would be to specify 123/udp in **/etc/ppp.filter**.

DEC ULTRIX 4.2 and some other systems may have no entry for FTP's data socket in their /etc/services file. If you want to log the bulk data connections as well as the control connections, you'll need to either add an entry to /etc/services, or use '20/tcp' explicitly in **/etc/ppp.filter**. The former is preferable because it will cause the log file entry to contain the symbolic name ('ftp-data') rather than the socket/protocol notation.

SEE ALSO

tun(4), ppp.devices(5), ppp.dialers(5), ppp.systems(5), services(5), pppd(8), RFC792, RFC1171, RFC1172, RFC1144.

COPYRIGHT INFORMATION

Copyright © 1991 Morning Star Technologies Inc.; all rights reserved.

NAME

ppp.systems – PPP neighboring systems description file format

DESCRIPTION

The file **/etc/ppp.systems** describes how to connect with neighboring systems via PPP.

FORMAT

Entries are one to a line; blank lines are ignored. Comments begin with a '#' and extend to the end of the line. Upper/lower case distinctions are ignored in hostname specifications, but are significant elsewhere. Fields on a line are separated by horizontal white space (blanks or tabs).

Each entry must contain six fields, in the following order:

name — The name or IP address of the destination machine

when — A string that indicates the days of the week and the times of day when the system can be called (for example, MoTuTh0800-1740). The day portion may be a list containing any of Su, Mo, Tu, We, Th, Fr or Sa. The day may also be Wk for any weekday (same as MoTuWeThFr) or Any for any day (same as SuMoTuWeThFrSa).

You can indicate hours in a range (for example, 0800-1230). If you do not specify a time, calls will be allowed at any time.

Note that a time range that spans 0000 is permitted. For example, 0800-0600 means that all times are allowed except times between 6 AM and 8 AM.

Multiple date specifications that are separated by a vertical bar (|) are allowed. For example, Any0100-0600|Sa|Su means that the system can be called any day between 1 AM and 6 AM or any time on Saturday and Sunday.

The entire (sequence of) days and times may be followed by a semicolon and the number of seconds before a failed call may be retried. For example, Any;60 means call any time, but wait at least 60 seconds after a failure has occurred before trying to call again. If a call retry fails, *pppd* will double the delay before trying again. If no beginning call retry delay time is specified, 10 seconds is assumed.

The call retry delay time may be followed by a hyphen and the maximum number of seconds to wait before trying to call again. If a call repeatedly fails, the retry time will double with each call until it reaches this value, after which the call will be retried every time the maximum number of seconds passes. If no beginning call retry maximum backoff interval is specified, 3600 seconds is assumed.

device — If set to ACU, any device in **/etc/ppp.devices** with a matching speed may be used. The device's dialer chat script will be executed first, followed by the **/etc/ppp.systems** chat script. If set to the name of a device in the **/dev** directory (**ttya**, **cua**, etc.), then **/etc/ppp.devices** and **/etc/ppp.dialers** will not be consulted, and only the **/etc/ppp.systems** chat script will be executed.

speed — The baud rate. If the device field is ACU, the speed field will be string matched against entries in **/etc/ppp.devices**. Speeds must either be valid baud-rate numbers or must begin with them (2400, 38400, 19200-PEP, etc.).

phone number — The value to replace the **\T** escape sequence in the dialer script

chat script — A description of the conversation that *pppd* holds with the remote machine.

CHAT SCRIPT PARTICULARS

A chat script takes the form of a word to expect the remote end to send, followed by a word to send in response. After each 'send' string that doesn't end with **\c**, *pppd* sends a carriage return character (ASCII 0x0d).

Chat scripts are 'expect send expect send ...' or 'expect-send-expect send ...', where the send following the hyphen is executed if the preceding expect fails.

Certain special words may be used in chat script 'send' strings to control the behavior of *pppd* as it attempts to dial.

ABORT *abort-string* If *pppd* sees *abort-string* while executing the remainder of the chat script, abort the dialing attempt and note the failure in the log file.

TIMEOUT *timeout-time* While executing the current chat script, wait *timeout-time* seconds for a response before considering the dialing attempt to have timed out.

The expect-send couplet of **"" WORD** sets the line parity accordingly:

P_ZERO Set the parity bit to zero.

P_ONE Set the parity bit to one.

P_EVEN Transmit characters with even parity. This is the default.

P_ODD Transmit characters with odd parity.

In the midst of either an 'expect' string or a 'send' string, ^*x* gets translated into the appropriate control character, and *x* gets translated into *x*. Other special sequences are:

\s Send or receive a space character (ASCII 0x20)

\t Send or receive a horizontal tab character (ASCII 0x09)

\n Send or receive a linefeed character (ASCII 0x0a)

\r Send or receive a carriage return character (ASCII 0x0d)

\d Delay for two seconds before proceeding (send only)

\c Don't append a carriage return character after sending this string

\q Don't print succeeding send strings (e.g. passwords) in any debugging output

EXAMPLE

In the example below, we call host 'everyone' using a Telebit PEP modem with its DTE interface set at 19200 bps. We call host 'nobody' using a V.32/V.42/V.42bis modem that's capable of driving a 38400 DTE, and we are connected to host 'someone' via a direct cable attached to /dev/ttya.

If we are unsuccessful at connecting with 'someone' we will try again in two seconds. If that attempt fails, we will wait four seconds before the next attempt; then eight, then sixteen, then thirty two, then forty seconds. We will continue attempting to contact 'someone' every forty seconds. Our retry intervals and maximum backoff values for 'everyone' and 'nobody' are the default '10-3600'.

The notation "" "" means to expect nothing, then send nothing (followed by a carriage return). The implicit carriage return is often useful for eliciting a response from a remote system.

```
#
#       /etc/ppp.systems - PPP systems file
#
everyone Any ACU 19200-PEP 5551212  login: Pwe word: \qfoObar
nobody Any ACU 38400 5551213 "" "" gin: Pthey word: \qbaZz1ng
someone Any;2-40 cua 38400 0 ogin:--ogin: Pthem word: \qmeumBle
#
```

RECOMMENDATIONS

The default retry time and backoff (i.e. Any;10-3600) are appropriate in the case of a dedicated line between two systems, over which the PPP connection must normally be reestablished as quickly as possible after an interruption. Moderate call retry times, such as 60 seconds, work well on systems that call each other on dialup modems, to avoid deadlocks waiting for telephone busy signals from each calling the other at the same times. Because of the difference between the behaviors of originating and answering modems, the 60-second clocks will usually start ticking at different times, allowing one side to call the other without interference. Alternatively, different call retry times may be specified at either end of a link to help keep opposing ends of a link from calling the other end simultaneously.

If you specify host names, be sure that their addresses are available locally, even with the connection down. If you find that you must bring up a connection to resolve a domain name, consider using that host's IP address (decimal numbers separated by periods) in both **/etc/ppp.filter** and **/etc/ppp.systems** instead.

SEE ALSO

tun(4), ppp.devices(5), ppp.dialers(5), ppp.filter(5), pppd(8), RFC1171, RFC1172, RFC1144.

COPYRIGHT INFORMATION

Copyright © 1991 Morning Star Technologies Inc.; all rights reserved.

NAME

pppd – PPP daemon

SYNOPSIS

pppd [**nooptions**] [**noaccomp**] [**noprotcomp**] [**novjcomp**] [**noasyncmap**] [**asyncmap** async-map] [**noipaddress**] [**netmask** subnet-mask] [**nomagic**] [**nomru**] [**mru** mru-size] [device] [comm-speed] [**crtscts**] [**xonxoff**] [local:[remote]] [**auto**] [**up**] [**nodetach**] [**active**] [**passive**] [**old**] [**log** log-file] [**filter** filter-file] [**idle** idle-time] [**timeout** restart-time] [**debug** debug-level]

DESCRIPTION

pppd is a daemon process used in UNIX® systems to manage a connection to another host using PPP, the Point to Point Protocol. It uses the UNIX host's native serial ports. It communicates with the UNIX kernel's own TCP/IP implementation via the Morning Star IP tunnel driver. (see *tun*(4))

OPTIONS

nooptions Disable all LCP and IPCP options

noaccomp Disable HDLC Address and Control Field compression.

noprotcomp Disable LCP Protocol Field Compression.

novjcomp Disable RFC1144 Van Jacobson TCP header compression.

noasyncmap Disable LCP Async Control Character Map negotiation.

asyncmap async-map Set the desired Async Control Character Map to async-map, expressed in hexadecimal (default 0xA0000).

noipaddress Disable IPCP IP Address negotiation.

netmask subnet-mask Set the subnet mask of the interface to subnet-mask, expressed either in hexadecimal (e.g. 0xffffff00) or in dotted-quad notation (e.g. 255.255.255.0). The default subnet mask will be appropriate for the network (class A, B, or C), assuming no subnetting.

nomagic Disable LCP Magic Number negotiation.

nomru Disable LCP Maximum Receive Unit negotiation (demand 1500).

mru mru-size Set LCP Maximum Receive Unit value to mru-size for negotiation (default 1500).

device Communicate over the named device (default /dev/tty).

comm-speed Set communications rate to comm-speed bits per second.

crtscts Condition the line for out-of-band EIA RS-232 'hardware' (RTS/CTS) flow control. The default is to use no flow control. For an outbound connection, this may be specified either in **/etc/ppp.devices** or on the *pppd* command line.

xonxoff Condition the line for in-band ('software') flow control, using the characters DC3 (^S, XOFF, ASCII 0x13) to stop the flow and DC1 (^Q, XON, ASCII 0x11) to resume. The default is to use no flow control. For an outbound connection, this may be specified either in **/etc/ppp.devices** or on the *pppd* command line.

local:remote The address of this machine, optionally followed by the expected address for the the remote machine. Can be specified either as symbolic names (if the IP addresses are known locally) or as literal IP addresses.

Both addresses must be specified when running as a daemon in 'autocall' mode. If only 'local:' is specified when receiving an incoming call, the remote address will be discovered during IPCP IP-Addresses negotiations.

auto Start in 'autocall' mode and detach from the controlling terminal to run as a daemon. Initiate a connection in response to a packet specified in the 'bringup' category in filter-file. Requires both local and remote addresses.

up If used with **auto**, bring the link up immediately rather than waiting for traffic.

nodetach Don't detach from the controlling terminal in 'autocall' mode.

active Begin in the LCP Active-Open state (the default unless in 'autocall' mode).

passive Begin in the LCP Passive-Open state (the default in 'autocall' mode).

old Enable backward compatibility with older implementations of PPP by using 0x37 (as specified in the typographical error in RFC1172 section 5.2) to negotiate VJ TCP Header Compression in IPCP. The default is to use the correct value of 0x2d.

log log-file Direct logging messages to log-file (default: **/usr/adm/pppd.log**).

filter filter-file Look in filter-file for packet filtering and link management information (default: **/etc/ppp.filter**).

idle idle-time If in 'autocall' mode, turn off the link when idle-time seconds pass without passing a packet specified in the 'keepup' category in filter-file (default: never).

timeout restart-time Set the LCP and IPCP option negotiation restart timer (default 3 seconds).

debug debug-level Set the debugging verbosity to debug-level (default 0).

SIGNALS

Upon reception of the following signals, *pppd* takes the indicated actions:

SIGINT Disconnect gracefully from an active session. If in 'autocall' mode, re-read the filter file and reset the call retry delay timer and call retry backoff interval. Exit if not in 'autocall' mode.

SIGHUP Disconnect abruptly from an active session. If in 'autocall' mode, re-read the filter file. Exit if not in 'autocall' mode.

SIGTERM Disconnect gracefully from an active session, then exit.

SIGUSR1 Re-read the filter file, and increment the verbosity level for debugging information written to the log file:

1 Dial chat

2 All IP messages

3 Interface changes, etc.

4 LCP, IPCP, UPAP and other finite state machine messages

5 More detailed procedure call messages

6 All PPP messages (without framing)

7 Timers, even more detailed procedure call messages

8 Characters read or written

SIGUSR2 Re-read the filter file, and reset the debugging verbosity level to 0 (the default).

EXAMPLE

To run a pair of daemons on 'oursystem', one maintaining a constant link with 'backbonesystem' and the other prepared to initiate outbound calls to a neighboring machine named 'theirsystem', add the following to **/etc/rc.local**:

```
#
if [ -f /usr/etc/pppd ]; then
        if [ -f /usr/adm/pppd.log ]; then
                mv /usr/adm/pppd.log /usr/adm/OLDpppd.log
        fi
        pppd oursystem:backbonesystem auto up
                        (echo -n ' pppd(backbonesystem)') > /dev/console
        pppd oursystem:theirsystem auto idle 120
                        (echo -n ' pppd(theirsystem)') > /dev/console
```

```
fi
```

To allow a PPP implementation running on ‘theirsystem’ to dial into ‘oursystem’, insert the following into /etc/passwd on ‘oursystem’:

```
TheirPPP:??:105:20:Their System's PPP:/tmp:/usr/etc/ppplogin
```

where **/usr/etc/ppplogin** is an executable shell script that looks something like

```
#!/bin/sh
#
PATH=/usr/bin:/usr/etc
#
mesg n
stty -tostop
exec pppd ‘hostname‘:theirsystem idle 130
```

RECOMMENDATIONS

Use host names in **/etc/rc.local** if they are known locally. If a call would be required to resolve a host name, use its literal IP address instead.

SEE ALSO

tun(4), ppp.devices(5), ppp.dialers(5), ppp.filter(5), ppp.systems(5), RFC1171, RFC1172, RFC1144.

CREDITS

Parts of the protocol engine were written by Drew D. Perkins, ddp@andrew.cmu.edu, now ddp@interstream.com.

COPYRIGHT INFORMATION

This daemon contains code that is Copyright © 1989 Carnegie Mellon University.

Copyright © 1991 Morning Star Technologies Inc., all rights reserved.