Mpd 3.18 User Manual

Archie Cobbs
<archie@freebsd.org>

May 4, 2004

Abstract

This manual covers the use of **mpd**, a multi-link PPP daemon for FreeBSD based on `netgraph(4)`. Mpd is highly configurable and extendable as well as being very fast. It supports many combination of drivers and protocols including asynchronous serial (e.g., modems), synchronous serial, Frame Relay, PPTP, and others. This manual documents mpd version **3.18**. This version relies on the Netgraph networking system to combine the advantages of user-level configuration and negotiation flexibility with kernel-level speed of data handling. You must be running FreeBSD version 3.4 or higher.

1. Introduction

1.1 Overview

Mpd is a netgraph(4) based implementation of the multi-link PPP protocol for FreeBSD. It is designed to be both fast and flexible. It handles configuration and negotiation in user land, while routing all data packets strictly in the kernel. It supports several of the numerous PPP sub-protocols and extensions, such as:

- Multi-link PPP capability
- PAP, CHAP, and MS-CHAP authentication
- RADIUS authentication and accounting
- DES and MPPE encryption
- Stac*, and MPPC* compression
- Point-to-Point Tunnelling Protocol (PPTP)

Mpd also includes many additional features:

- Dial-on-demand with idle timeout
- Multiple active connections running simultaneously
- Dynamic demand based link management (also known as ''rubber bandwidth'')
- Powerful chat scripting language for asynchronous serial ports
- Pre-tested chat scripts for automatic configuration of several common modems and ISDN terminal adapters
- Support for the Netgraph kernel networking system
- Support for tunnelling PPP over TCP, UDP, and PPTP
- Clean device-type independent design
- Comprehensive logging

Mpd supports the following device types:

- **Modem.** The modem device type supports asychronous serial connections, including modems, ISDN terminal adapters, and null-modem connections. Mpd includes a fully event-driven scripting language for modem identification, setup, manual server login, etc.
- **Netgraph.** Netgraph is highly modular kernel networking system, supporting synchronous serial connections, Cisco HDLC, Frame Relay, and other protocols. Mpd supports running PPP over any netgraph connection.
- **TCP.** Mpd can tunnel PPP frames over a TCP connection. Frames are encoded in the same was as asychronous serial connections.
- **UDP.** Mpd can tunnel PPP frames over a UDP connection. Each frame is encapsulated in a UDP datagram packet.
- **PPTP.** Mpd can connect over the Internet using the Point-to-Point Tunnelling Protocol (PPTP). This protocol is supported by the popular Microsoft Dial-Up Networking client.
- **PPPoE.** Mpd can connect over an Ethernet port using the PPP-over-Ethernet (PPPoE) protocol.

Mpd was originally developed at Whistle Communications, Inc. for use in the Whistle InterJet. It is based on the original `iij-ppp` user-mode PPP code, though it has pretty much been completely rewritten since then. Mpd is now hosted on sourceforge.net MPD Project Page.

* These protocols require source files that are proprietary and/or export-controlled and cannot be included in the normal distribution. For MPPC, you need proprietary compression from Hi/Fn (formerly Stac).

1.2 Mpd Layers

Mpd consists of several **layers**, corresponding to the layered nature of the PPP protocol. The two main concepts in mpd are *bundles* **(section 4.2 , page 20)** and *links* **(section 4.5 , page 28)**. A **link** is a single physical point-to-point connection between the local machine and a remote peer machine, such as a modem connection or a synchronous serial connection. A **bundle** is a collection of one or more links, all connecting to the same remote peer, that together form a single multi-link PPP connection whose effective bandwidth is the sum of the bandwidths of the individual links.

The bundle layer in effect lies just above the link layer. The bundle layer handles the task of making multiple physical links appear as a single virtual link. Bundles and layers exist regarless of whether the link(s) are actually connected and transmitting packets.

At the bundle layer you configure multi-link PPP settings, authentication credentials for the bundle (i.e., login and password), and the link management policy. The link management policy determines whether mpd tries to keep all the links connected all the time, or whether it adds and subtracts links depending on demand, and if so, according to what parameters.

At this time, mpd requires the links that constitute each bundle to be declared when the bundle is created; after that, they cannot be changed. A future revision may lift this restriction.

With each bundle is a corresponding *interface* **(section 4.3 , page 23)** layer, which corresponds directly to a netgraph interface accessible via `ifconfig(8)`, such as `ng0`. The netgraph interface is where IP packets ultimately appear after travelling across the link. The interface layer handles configuring the interface, bringing it up or down as appropriate, assigning IP addresses, setting up static routes, and configuring proxy-ARP. The interface layer is also responsible for implementing dial-on-demand and idle timeout functionality.

Each bundle has a corresponding *IP Control Protocol (IPCP)* **(section 4.4 , page 26)** layer, which manages the IP specific configuration of the bundle. This layer handles the negotiation of local and remote IP addresses and Van Jacobson TCP header compression, as well as other optional IP related information such as DNS servers and NBNS servers.

Each bundle also has corresponding *compression* **(section 4.6 , page 31)** and *encryption* **(section 4.7 , page 32)** layers, which allow you to enable and configure compression and encryption for data sent and received over the bundle.

Individual links in a bundle are configured at the *link* **(section 4.5 , page 28)** layer. There are two parts to the link configuration, the device independent part and the device specific part. Device independent parameters include the authentication type (PAP or CHAP), whether authentication is enabled in either direction, keep-alive packets, and various other link specific parameters.

Beneath the link layer is the **device** layer. Configuration of device specific parameters happens at this layer. Each link has a specific *type* corresponding to one of the supported device types in mpd. The type dictates how the device dependent part of the link is configured and what its capabilities are.

While the combination of all the various layers presents a large number of configuration options, mpd tries to have reasonable defaults for everything.

1.3 Organization of this manual

This remainder of manual is separated into five sections:

- **Installation.** How to download, compile, and install mpd.
- **Running mpd.** How to invoke mpd from the UNIX command line, and how to use mpd's own command line interface.
- **Configuration.** How to configure the various link-type independent components of mpd.
- **Device types.** The devices types supported by mpd and how to configure each device type.
- **References.** References to the standards to which mpd was written.

1.4 Change history

Changes since version 3.17:

- BugFix: an empty authname was passed to the iface up/down scripts in client-mode.
- BugFix: fix a mem-leak in the PPTP-Ctrl handling.
- BugFix: handle IP-Address-Assignement correctly, if the RADIUS server doesen't send the Framed-IP-Address attribute. Submitted by: Damir Bikmukhametov, `boco at ufanet dot ru`
- BugFix: fix a SEGV when no bundles are defined and user types "quit".
- BugFix: fix a SEGV when no bundles are defined and the telnet console was opened (in this case the tcp-wrapper option can't be checked).
- BugFix: fix build errors with newer versions of libradius.
- Added a new option for PPTP links for disabling the windowing mechanism specified by the protocol. Disabling this will cause Mpd to violate the protocol, possibly confusing other PPTP peers, but often results in better performance. The windowing mechanism is a design error in the PPTP protocol; L2TP, the successor to PPTP, removes it. You need a recent version of FreeBSD (NGM_PPTPGRE_COOKIE >= 1082548365) in order to get this feature.

  ```
  set pptp disable windowing
  ```

Changes since version 3.16:

- BugFix: keep-alives were b0rked, and every other function wich relys on valid link-stats.
- Added sample configs for ADSL-Austria.
- Fixed missing command line flags in man page and manual.

Changes since version 3.15:

- Added `set bundle max-logins` *num* option at bundle level for limiting the max. amount of concurrent logins with the same username. Original idea submitted by Andrey V. Elsukov `<bu7cher@land.ru>`.
- Mpd has now its own link-stats with 64 bit counters. This is a workaround for the 32 bit limitation of ng_ppp_link_stat.
- RADIUS enhancements:
  - Set the RAD_ACCT_INPUT_GIGAWORDS and RAD_ACCT_OUTPUT_GIGAWORDS attributes when sending accounting requests.
  - Use the RAD_ACCT_INTERIM_INTERVAL attribute.
- Implemented PPPoE server functionality
- Added ability to configure ipfw firewall for clients using vendor specific RADIUS attributes. Submitted by Alexander Motin `<mav@alkar.net>`.

- Added the possibility of choosing link by it's number (same as RAD_NAS_PORT) instead of name

Changes since version 3.14:

- Implemented a workaround for weird crashes when using RADIUS and PAP. Thanx to Alexey Popov for helping debugging this problem.
- Added the possibility of sending periodic RADIUS Accounting updates. Submitted by Vitaliy Akimov <vx@bigline.net>.
- Fixed usage of RAD_FRAMED_MTU, it is now interpreted as an upper limit and not as absolute value.
- `set iface enable tcpmssfix` option added. Submitted by Sergey Korolew <dsAT-bittu.org.ru>.
- New command line flag `-a` for specifying the telnet console IP address. Original idea submitted by Andrey V. Elsukov <bu7cher@land.ru>.

Changes since version 3.13:

- Added session timeout, configurable at iface level.
- Added new configuration params: `chap-md5 chap-msv1 chap-msv2` at link level for explicitly configuring the different CHAP methods. The formerly `chap` parameter is now an alias for `chap-md5 chap-msv1 chap-msv2`. MS-CHAPv1 is now disabled and denied by default. A better fallback mechanism through the different auth-methods was implemented.
- More RADIUS fixes/features:
  - Use also the RAD_SESSION_TIMEOUT.
  - Fixed RAD_ACCT_TERMINATE_CAUSE.
  - Fixed MPPE-Types negotiation.
  - Session time is now calculated per link.
  - Use RAD_FRAMED_ROUTE attribute.
  - When no MPPE-Info's are returned by the server, but MPPE-Keys are, then implicitely allow encryption with all types (workaround for Microsoft IAS).
- Reset link statistics when the link goes down
- `set iface radius-route` option added. Submitted by Alexander Motin <mav@alkar.net>.
- DNS servers added to interface up script. Submitted by Josh Elsasser <jre@vine-yard.net>.
- Fix bug in multi-session PPPoE support caused by incorrect application of the original patch. Submitted by OGAWA Takaya <t-ogawa@triaez.kaisei.org>.
- Fix problem where IP address assignment from `mpd.secrets` would linger and affect subsequent connections on the same bundle.

Changes since version 3.12:

- Support for PPPoE node re-use added. Patch submitted by Gleb Smirnoff <glebius@cell.sick.ru>. See also FreeBSD PR <ports/48138>.
- Multi-session PPPoE support added. Patch submitted by Hideyuki Nishiyama <hnishi@mx5.harmonix.ne.jp>

- More RADIUS fixes from Michael Bretterklieber <mbretter@jawa.at>
- After configuring the IP address, add a route to the local IP address via `lo0`. This prevents packets destined for the local machine from needlessly travelling across the link and back. Patch submitted by Fabio Vilan <fabio@isec.com.br>

Changes since version 3.11:

- Fixed bug in client-side MPPE key generation.

Changes since version 3.10:

- Support for authentication and IP address assignment via RADIUS was added. Thanks to Michael Bretterklieber <mbretter@jawa.at> for submitting the patches.
- Support for finding passwords using an external authentication script was added. Thanks to Gregory Bond <gnb@itga.com.au> for submitting the patches.
- Fixed build failure when MICROSOFT_CHAP is not enabled.
- Enabled support for DES encryption by default in the build.
- Fixed bug where LCP echo reply packets were being sent with garbled payload contents.
- Fixed bug with `set iface mtu` where an inadvertent `set iface up-script` command would also be invoked.
- Removed LCP_MRU_MARGIN hack which should no longer be needed. Note: you may need to explicitly use `set iface mtu` and/or `set link mtu` to get the previous MTU behavior for your configuration.
- Fixed bug where outgoing PPTP requests were not being initiated with the configured local source IP address. Note: using multiple PPTP IP addresses at the same time for incoming connections still does not work properly; workaround: use multiple MPD processes.
- Log the MTU being used each time an interface is configured.
- New command added: `set udp origination`.
- Link with libcrypto instead of the obsolete libdes.
- Added new command bundle option `noretry`.
- Install the PostScript version of the documentation as well.
- Bumped the max length for an up or down script to 128 (from 32).

Changes since version 3.9:

- Fixed more bugs when computing the interface MTU
- Added the `set iface mtu` command.
- Added troubleshooting chapter to the documentation.
- Peer authname is appended as an extra parameter to the interface up and down scripts.
- Fixed broken links to IETR RFC's in the references chapter of the documentation.
- Fixed bugs that only showed up on non-i386 hardware (mostly alignment bugs).
- Added new GCC -W flags and fixed lots of GCC warnings.

Changes since version 3.8:

- Added the `set link mtu` command.

- Fixed typo in description of `set bundle period` command.
- Fixed incorrect description of `set ng hook` command.
- Updated HTML links to on-line Netgraph documents.
- Added HTML link to Microsoft update for Windows 95/98.
- When displaying bundles and links, show the links LCP state as well.

Changes since version 3.7:

- Now setting `max-redial` to -1 means mpd will never redial. This is useful for dial-in servers.
- Increased the default `FD_SETSIZE` to 2048 so we can handle lots of bundles, and made it compile-time adjustable.
- Allow multiple PPTP connections from the same IP address when this is OK to do (no explicit peer IP addresses configured). This allows multiple incoming connections through a remote NAT device.
- Added support for 56-bit MPPE encryption.
- Allow the inner and outer PPTP IP addresses to be the same, relying on the kernel to return an error if it needs to. This reverts the change put into version 3.3.
- Fixed bug in MTU calculation when doing MPPC/MPPE.
- Added the `set pptp phonenum` *number* command.
- Added documentation for the `delayed-ack` and `always-ack` PPTP device configuration options.
- Minor tweaks to the sample config files.

Changes since version 3.6:

- Fix a bug where the MPPE keys were being incorrectly generated when MS-CHAP was used to authenticate in both directions.

Changes since version 3.5:

- Fix MS-CHAPv2 acknowledge reply to workaround Win9x bug when trying to negotiate MPPE encryption.

Changes since version 3.4:

- Fix an install problem on FreeBSD-current.
- Display the peer's authname and EID in "show bundle" command.

Changes since version 3.3:

- Support for MS-CHAPv2 when we are the server. Always request MS-CHAPv2 first, then accept MS-CHAPv1 if the peer requests that instead.
- Fix for possible core dump in ECP code.
- Fix for bug in event library when an invalid file descriptor is passed.
- Disallow allow PPTP peer inside and outside IP addresses from being the same; we know this won't work.
- Fixed bug in PAP code where the PAP response packet would be improperly formatted.
- Accept EINPROGRESS error from ng_ksocket connect message. This happens on FreeBSD-current for some reason.

- Accept PPTP messages with non-zero values in reserved fields. They violate the specification but some peers send them anyway.
- Remove bogus extra field in definition of PPTP StopCtrlConnReply.

Changes since version 3.2:

- There is no longer a fixed limit to the number of bundles and/or links that may be created.
- Better random number generation using `srandomdev(4)`.
- Miscellaneous minor bug fixes.
- Support for authenticating to the peer using Microsoft CHAP Version 2 has been added. **Note:** When authenticating the peer, only Version 1 is requested.
- Hostnames are allowed in the configuration files.

The latter two are thanks to Franco Venturi `<fventuri@mediaone.net>`

2. Installation

2.1 Getting mpd

Mpd is supplied as a **FreeBSD port**, under the name `net/mpd`. Therefore, the simplest way to get mpd onto your machine is to install the mpd package or port. For information on how to do this, see the **FreeBSD ports web page**.

Mpd can also be compiled and installed manually. This is useful for example if you want to change the set of device types supported by mpd.

2.2 Installing mpd

If you use the FreeBSD package or port, mpd will be automatically installed into `/usr/local/sbin/mpd`. with the default set of device types compiled into it. In order to run, mpd needs a few configuration files. The directory `/usr/local/etc/mpd` is created to hold them. In it you will find some sample configuration files.

Before you can run mpd, you must set up your site-specific configuration files. These files are:

`mpd.conf`
: This file defines one or more **configurations**. When mpd is invoked, you specify the name of a configuration on the command line. This configuration (which merely consists of a series of mpd commands) is loaded. If no configuration is given, the first one listed in `mpd.conf` is loaded.

  Each configuration defines one or more bundles. Each bundle is given a name, assigned a system interface and one or more links that make up the bundle. A bundle is defined with the `new` command. Subsequent commands in the configuration configure the various layers for that bundle.

  Generally speaking, the commands in `mpd.conf` only configure the device independent information for a bundle.

`mpd.links`
: This file contains definitions of the various links that are available on the system. Different links may be part of different bundles in different configurations. Therefore, the commands in this file configure information that is link specific and independent of any particular configuration. For example, a certain modem may require a special init string, no matter how or in what bundle the modem is to be used. The commands in this file are executed when the bundle is defined (i.e., as a result of a `new` command). So the information configured for a link by that link's `mpd.links` entry can be overriden by subsequent commands in `mpd.conf`.

`mpd.secret`
: This file contains login, password pairs. This is where mpd looks to find all authentication information. This file should be readable only by `root`.

`mpd.script`
: This file contains chat scripts for modem devices.

2.3 Building mpd

If you choose to build mpd yourself to customize it, the process is straightforward. First, edit the `Makefile` to define (or undefine) the various device types and options you want to support. Then type `make depend all` to rebuild the binary.

The various build-time definitions in the Makefile are below:

`MPD_CONF_DIR`
: The default configuration directory where mpd looks for `mpd.conf`, `mpd.links`, etc.

`PHYSTYPE_MODEM`

`PHYSTYPE_TCP`

`PHYSTYPE_UDP`

`PHYSTYPE_NG_SOCKET`

`PHYSTYPE_PPTP`
: Define these to include support for the corresponding device type.

`COMPRESSION_PRED1`

`COMPRESSION_STAC`

`COMPRESSION_MPPC`

`COMPRESSION_DEFLATE`
: These enable support for the corresponding compression types. Note that STAC and MPPC compression require proprietary source files that are not included. DEFLATE is unimplemented. All of these are not very well tested.

`ENCRYPTION_DES`

`ENCRYPTION_MPPE`
: These enable support for the corresponding encryption types.

`MICROSOFT_CHAP`
: Enables support for Microsoft style CHAP (versions 1 and 2). Note that this option is implied by the `ENCRYPTION_MPPE` option.

`SYSLOG_FACILITY`
: Mpd normally logs via `syslog(3)` using the facility `LOG_DAEMON`. You can customize the facility here.

3. Running Mpd

3.1 Invoking mpd

Mpd is invoked by first setting up your configuration files, and then running

```
mpd [ options ] [ configuration ]
```

If *configuration* is supplied, then mpd looks in `mpd.conf` for a matching label, and runs the corresponding commands (up to the first blank line). If *configuration* is not supplied, mpd looks for the configuration named `default`. If the configuration is not found, no commands are executed from `mpd.conf`.

Mpd can either be run in interactive mode or in the background as a daemon. In interactive mode mpd accepts commands typed at the console.

Mpd understands the following command line options. Each option has a short, single character form, as well as an equivalent long form.

`-b --background`
: Run as a background daemon.

`-a --console-address` *ip-address*

`-c --console-port` *port*
: With either of these options, mpd will listen for and accept TCP connections on the given IP address and port. Once connected, you can enter commands just as if they were typed in interactive mode. Note that this is a potential security hole, as any user on the system can connect. Mpd allows you to require a login and password if so desired, but **beware**: by default, no login is required.

: The default IP address is `0.0.0.0` (i.e., listen on all interfaces) and the default port is 5005.

`-d --directory` *dir*
: This option allows you to specify a different directory for the mpd configuration files other than the default `/usr/local/etc/mpd`.

`-f --file` *filename*
: Specify an initial configuration file other than the default, `mpd.conf`.

`-p --pidfile` *filename*
: Mpd will open and lock *filename* and write its process ID before starting. If another mpd process is already running, mpd will not start. The default is `/var/run/mpd.pid`.

`-k --kill`
: With this option, pd will attempt to kill any existing mpd process before beginning execution. The previous process must have written its process ID in the PID file `/var/run/mpd.pid` (or the file you specify with the `--pidfile` option).

`-s --syslog-ident` *ident*
: Mpd normally logs via `syslog(3)` with the identifier `mpd`. This option allows you to change that identifier.

`-v --version`
: Displays the version number of mpd and exits.

`-h --help`
: Displays a usage message and exits.

Mpd responds to the following signals while it is running:

SIGTERM
: Attempt to gracefully shut down all active connections and exit.

SIGUSR1
: This signal causes mpd to initiate a connection with the currently active bundle. If mpd is already attempting to connect, this signal has no effect.

SIGUSR2
: This signal has the opposite effect, namely, it causes mpd to close the currently active bundle. If the bundle is already in a closed state, this signal has no effect.

3.2 Configuration file format

Mpd uses four different configuration files when it executes. These files are described below.

3.2.1 General properties

For all of the configuration files, the following things are true:

- Any line whose first character is a number sign (`#`) is considered a comment and is ignored.
- Strings may be quoted using double quotes. The string is decoded according to normal C language rules.

3.2.2 mpd.conf and mpd.links

The `mpd.conf` and `mpd.links` files have the same simple structure. An *entry* consists of a *label* followed by a sequence of mpd commands. A label begins at the first column and ends with a colon character. Commands are usually indented with a tab character (this is not required) and follow the label immediately on the next and subsequent lines. An entry ends at the first blank line after the label.

Here is an example `mpd.conf` that contains a single configuration `config1` that defines a single bundle `myisp`, containing two links `modem1` and `modem2`:

```
#
# mpd.conf configuration file
#

config1:
        new myisp modem1 modem2
        set bundle authname "mylogin1"
        set iface enable on-demand
        set iface addrs 1.2.3.4 5.6.7.8
        set iface route default
        set iface idle 300
        link modem1
        set modem var $Telephone "123-4567"
        link modem2
        set modem var $Telephone "123-4567"
```

The corresponding `mpd.links` file might look like this:

```
#
# mpd.links configuration file
#

# my internal modem which is on COM3
modem1:
        set link type modem
        set modem device /dev/cuaa2

# my external modem which is on COM2
modem2:
        set link type modem
        set modem device /dev/cuaa1
```

Note that the telephone number to dial for each link is set in `mpd.conf` instead of `mpd.links`, because this is specific to `config1`. On the other hand, the serial port device is a property of the link and independent of the configuration; hence this is set for each link in `mpd.links`.

3.2.3 mpd.secret

This file contains login, password pairs, one entry per line. Each entry may have an optional third argument, which is an IP address with optional netmask width. This is used when netgotiating IP addresses with the corresponding peer. We restrict the allowable IP addresses we'll assign to the peer to lie within the specified range.

In the example below, we define two ISP accounts that we use to connect to the Internet. Also, we have three friends who are allowed to connect to us, and we want to restrict the IP addresses that we'll let them have. Finally, the last user's password is retrieved by an external program.

```
#
# mpd.secrets configuration file
#

# my two ISP accounts
mylogin1    password1
mylogin2    "Xka \r\n"

# my three friends
bob         "akd\"ix23"   192.168.1.100
jerry       "33dk88kz3"   192.168.1.101
phil        "w*d9m&_4X"   192.168.1.128/25

# An external password access program
gregory     "!/usr/local/etc/mpd/get_passwd.sh"
```

Here `bob` and `jerry` must negotiate `192.168.1.100` and `192.168.1.101`, respectively, while `phil` can ask for any address from `192.168.1.128` through `192.168.1.255`.

The leading "!" means that the password for user `gregory` is not stored in the `mpd.secret` file directly. Instead, the named program is run with the username being authenticated as an additional argument (so in this case the command line would be "`/usr/local/etc/mpd/get_passwd.sh gregory`"). The command string may include initial, fixed arguments as well. This program should print the plaintext password for the named user as a single line to standard output, and then exit. `Mpd` will block for this operation, so the program should respond and exit quickly. If there is an error, the command should print an empty line, or just not print anything.

Once authentication has completed, this script is run a second time, this time with two extra arguments. The first extra argument is either "-y" or "-n" depending on whether this user was properly authenticated or not. The second extra argument is the username, so in this example if Gregory entered his password correctly the command would be

"/usr/local/etc/mpd/get_passwd.sh -y gregory".

As a special case, if the username in the mpd.secret file is "*", then this line must be last as it matches any username. Then it is up to the external program to determine whether the username is valid. This wildcard matching only works for "!" lines.

This external password program is intented to allow links to other authentication types, such as one-time passwords or hardware authentication dongles. The second invocation of the program is intended to perform tasks such as incrementing the password generation for one-time password schemes (when called with "-y"), or disabling hardware dongles after a number of failures (when called with "-n").

The total length of the executed command must be less than 128 characters. The program is run as the same user who runs mpd, which is usually root, so the usual care should be taken with scripts run as root, e.g., make sure the script is not world-readable or world-writable. Standard input and standard error are inherited from the parent mpd process. Note that any additional arguments will be visible to users on the local machine running ps(1).

3.2.4 mpd.script

This file has a format similar to mpd.conf and mpd.links, but it contains chat script commands instead of mpd commands. Each line contains a single command, and labels identify starting points and/or subroutines in the script. Unlike mpd.conf and mpd.links, a blank line *does not* end the processing of commands.

See *Chat Scripting* (section 6. , page 40) for a complete decscription of chat scripting.

3.3 Mpd command line interface

When run in interactive mode, or after connecting to the telnet port, mpd presents a command line interface at the console. Here mpd commands can be entered to configure the various layers and check the status of the defined bundles and links. The commands available at the command line interface are exactly the same ones available in the mpd.conf and mpd.links config files.

At any given time, there is always an *active bundle* and an *active link* within that bundle. These are shown in the prompt. Here is an example of running mpd with a configuration in mpd.conf that defines a bundle myisp containing two links modem1 and modem2:

```
$ mpd config1
Multi-link PPP for FreeBSD, by Archie L. Cobbs.
Based on iij-ppp, by Toshiharu OHNO.
mpd: pid 1419, version 3.0b1 (root@jan.whistle.com 10:37  7-Jan-1999)
[myisp] ppp node is "mpd771-myisp"
[myisp] using interface ng0
[myisp:modem1]
```

By the time the prompt appears, mpd has already read in and executed all the commands associated with the configuration config1, which defines the bundle myisp. When the bundle is created mpd logs a message showing the name of the associated netgraph ng_ppp(4) node and system interface (mpd will automatically select one if not explicitly told which one to use). In this case mpd771-myisp is the netgraph node and ng0 is the interface. The prompt [myisp:modem1] shows the currently active bundle (myisp) and link (modem1). Any bundle specific commands issued at this point will apply to myisp. Similarly, any link specific commands will apply to modem1.

To change the active link or bundle, use the link or bundle command:

```
[myisp:modem1] link modem2
[myisp:modem2]
```

Now any commands entered apply to `myisp` or `modem2`, as appropriate. This technique works the same way in `mpd.conf`. Also, if changing the link requires changing bundles as well, mpd will do that too.

The available commands are described in the next chapter. When mpd is running the `help` can be used to get more information on a specific command:

```
[myisp:modem2] help show
Commands available under "show":
 bundle   : Bundle status                ccp      : CCP status
 events   : Current events               ipcp     : IPCP status
 iface    : Interface status             routes   : IP routing table
 layers   : Layers to open/close         link     : Link status
 lcp      : LCP status                   mem      : Memory map
 mp       : Multi-link status            proto    : Protocol summary
 types    : Supported device types       version  : Version string
[myisp:modem2]
```

Mpd allows you to use only a prefix of a command, as long as the command is unambigous:

```
[myisp:modem2] b
Defined bundles:
        Bundle         Links
        ------         -----
        myisp          modem2[DOWN]
[myisp:modem2] s
s: ambiguous command
```

4. Configuring Mpd

4.1 General mpd commands

This chapter describes general mpd commands that don't apply to a specific layer.

`bundle [` *name*]
: This command changes the currenly active bundle to the bundle named *name*. The currently active link is also changed, to the first link defined the bundle. If no *name* is given, this command lists all of the defined bundles and their constituent links. The new bundle will be reflected in the prompt.

`link` *name*

`link [`*number*]
: This command changes the currenly active link to the link named *name*. If this link does not belong to the currently active bundle, then the currently active bundle is changed to the bundle that owns link *name*. The new link will be reflected in the prompt. Link *number* (RAD_NAS_PORT) can be specified in hex form in square brackets in place of name.

`load` *configuration*
: This command causes mpd to read the entry for *configuration* out of `mpd.conf` and execute the corresponding commands, just as if *configuration* had been specified on the command line. Execution stops at the first blank line.

`show [` *item*]
: This command displays various status information. The valid values for *item* are:

bundle
: Show status information about the currently active bundle.

link
: Show status information about the currently active link.

iface
: Show status information about the interface layer associated with the currently active bundle.

routes
: Show the current IP routing table.

mp
: Show status information about the multi-link protocol associated with the currently active bundle.

ipcp
: Show status information about the IP control protocol associated with the currently active bundle.

ccp
: Show status information about the compression control protocol associated with the currently active bundle.

ecp
: Show status information about the encryption control protocol associated with the currently active bundle.

lcp
: Show status information about the link control protocol associated with the currently active link.

radius
: Information relating to RADIUS authentication.

proto
: Show protocol statistics for the currently active bundle and link.

types
: Show the device types supported by this version of mpd.

layers
: Show the names of the layers that are valid as arguments to the `open` and `close` commands.

events
: Show all pending events (for debugging mpd).

mem
: Show distribution of dynamically allocated memory (for debugging mpd).

version
: Show the version of mpd running.

`set ...`

This command sets various configuration information. Further details are contained in later sections in this chapter. The valid completions for `set` are:

`set bundle ...`
: Set bundle layer configuration parameters for the currently active bundle.

`set link ...`
: Set link layer configuration parameters for the currently active link.

`set iface ...`
: Set interface layer configuration parameters for the currently active bundle.

`set ipcp ...`
: Set IP control protocol configuration parameters for the currently active bundle.

`set ccp ...`
: Set compression control protocol configuration parameters for the currently active bundle.

`set login [` *name*]
: Sets the login name required for telnet connections to be *name*, or eliminates any required login if *name* is omitted. The password for *name* is found by looking up *name* in `mpd.secrets`.

`set radius ...`
: Configures RADIUS authentication:

    `set radius server` *name secret* [*auth-port* [*acct-port*]]
    : Configure RADIUS server parameters. Multiple RADIUS servers may be configured by repeating this command.

    `set radius timeout` *seconds*
    : Set the timeout for completion of RADIUS requests.

`set radius retries` *#retries*
: Set the number of retries for RADIUS requests.

`set radius config` *pathname*
: Set the pathname to the `radius.conf(5)`, if different from the default `/etc/radius.conf`.

`set radius me` *IP*
: Send the given IP in the RAD_NAS_IP_ADDRESS attribute to the server.

`set radius acct-update` *seconds*
: Set the accounting update interval.

`set debug` *level* . Sets the `netgraph(3)` debugging level. Default is zero.

`set modem` . Sets device specific configuration parameters for the currently active link, which must have type **modem**.

`set ng` . Sets device specific configuration parameters for the currently active link, which must have type **netgraph**.

`set tcp` . Sets device specific configuration parameters for the currently active link, which must have type **tcp**.

`set udp` . Sets device specific configuration parameters for the currently active link, which must have type **udp**.

`set pptp` . Sets device specific configuration parameters for the currently active link, which must have type **pptp**.

`set pppoe` . Sets device specific configuration parameters for the currently active link, which must have type **pppoe**.

`open [` *layer* `]` This command causes an OPEN event to be sent to the corresponding layer or finite state machine (FSM). The valid layers are:

iface
: Interface layer. Opening this layer configures the interface and installs any routes. If dial-on-demand is enabled, nothing else happens until there is outgoing traffic. Otherwise, the lower layers are opened as well in order to establish a connection.

bund
: Bundle layer. This layer represents the multi-link bundle "virtual link" and is normally opened and closed automatically by the interface layer.

ipcp
: IP control protocol FSM. This is normally opened and closed automatically.

ccp
: Compression control protocol FSM. This is normally opened and closed automatically.

lcp
: Link control protocol FSM. This is normally opened and closed automatically.

phys
: Device layer. This is normally opened and closed automatically.

Normally, only the `iface` layer should be explicitly opened or closed. This is because the other layers are programmed to automatically open and close when appropriate. Note that the **SIGUSR1** and **SIGUSR2** signals are an alternative way to open and close the currently active bundle.

`close [` *layer* `]`
: This command causes a CLOSE event to be sent to the corresponding layer or finite state machine (FSM).

`log [` +/-*flag* ... `]`
: Mpd supports several different logging flags. Every line logged by mpd is tagged with one (or more) of these flags. By turning the flags on or off, that particular type of logging is enabled or disabled. This allows you to customize logging verbosity.

Without any arguments, the `log` command shows the current set of logging flags. To enable a logging flag, add the +*flag* argument. To disable a logging flag, add the −*flag* argument.

These are the various flags supported by mpd and their corresponding categories. Those with a * are turned on by default:

auth*
: Link authentication events

bund*
: Bundle layer events

ccp
: Compression events and negotiation

ccp2
: Compression more verbosity

ccp3
: Compression even more verbosity

chat*
: Modem chat script

chat2
: Modem chat script more verbosity. Very useful when debugging chat scripts.

console*
: Log to the console as well as the log file

echo
: Echo/reply packets for all FSM's

frame
: Dump all frames transmitted and received

fsm*
: All FSM events (except echo and resets)

iface*
: Interface layer events

ipcp*
: IP control protocol events and negotiation

lcp*
: Link control protocol events and negotiation

link*
: Link layer events

mp
: Multi-link events and negotiation

mp2
: Multi-link more verbosity

phys*
: Device layer events

pptp*
: PPTP device layer high level events

pptp2
: PPTP more verbosity

pptp3
: PPTP even more verbosity

radius*
: RADIUS authentication events

`help [` *command* `]` This gives a brief description of the supplied command, or if an incomplete command is given, lists the available alternatives.

`null` This command does nothing.

`exit` This command exits the console, but does not quit the mpd process. This command is useful for disconnecting a telnet connection.

`quit` Close down all connections and quit the mpd process.

4.2 Bundle layer commands

This chapter describes commands that configure the bundle layer. All of these commands (except `new`) apply to the currently active bundle, i.e., the bundle shown at the command line prompt.

`new [ -i` *iface* `]` *bundle link1* `[` *link2* ... `]`

This command creates a new bundle named *bundle*. The bundle will be associated with system interface *iface*, which should be the name of a netgraph interface, i.e., `ng0`, `ng1`, etc. If an interface is specified but does not exist yet, mpd will create it (and any lower numbered `ng*` interfaces as well). If no *iface* is specified, mpd will create a new one. Specifying an interface explicitly is advised, as otherwise a new interface will be created on your system every time mpd is invoked.

The links that constitute the bundle are created and named *link1*, *link2*, etc. For each link, there must be a corresponding entry with the same name (i.e., label) in `mpd.links`. This entry must at the very least define the type of the link (via the `set link type` command).

`set bundle period` *seconds*

set bundle hiwat *percent*

set bundle lowat *percent*

set bundle min-con *seconds*

set bundle min-dis *seconds*" These commands are only meaningful when bandwidth management is enabled. Mpd will attempt to bring up a new link or take down an existing link when the utilization goes above *hiwat* or below *lowat* percent, respectively. The utilization is sampled every *period* divided by 6 seconds, and is averaged over the past *period* seconds.

Mpd will not attempt to reconnect a link unless it has been disconnected for at least *min-dis* seconds. Similarly, mpd will not bring down a link unless it has stayed connected for at least *min-con* seconds. These latter two parameters help prevent oscillations, but cause slower response time.

`set bundle authname` *login*
: This command sets the authentication login name associated with the bundle (in multi-link PPP, though each link is authenticated individually, they all must use the same login name). The *login* must have a corresponding entry in `mpd.secrets`. The *login* and password are used when the peer requires us to authenticate ourselves.

`set bundle password` *password*
: This command is normally not necessary. It causes mpd to *not* lookup the password corresponding to *login* in `mpd.secrets`, but rather to use *password* instead. If you're too lazy to set up `mpd.secrets` and are only dialing out, you can use this command instead.

`set bundle retry` *seconds*
: This sets the finite state machine (FSM) retry interval for all FSM's on this bundle. The default is 2 seconds. This command should only be used in exceptional situations, such as doing PPP over high-latency satellite links.

`set bundle accept` *option ...*

set bundle deny *option ...*

set bundle enable *option ...*

set bundle disable *option ...*

set bundle yes *option ...*

set bundle no *option ...*" These commands configure various bundle options. Most options are *bi-directional* in that they can be independently enabled and disabled in each direction.

The **enable** and **disable** commands determine whether we want the corresponding option. The **accept** and **deny** commands determine whether we will allow the peer to request the corresponding option.

The `yes` command is the same as **enable** and **accept**. The `no` command is the same as **disable** and **deny**.

The options available at the bundle layer are:

`bw-manage`
: This command enables *bandwidth management*, in which mpd attempts to dynamically adjust the number of connected links depending on the traffic load.

The proper functioning of bandwidth management depends on the correct latency and bandwidth information configured for each link; see the **set link latency** and **set link bandwidth** commands.

The default for this option is off -- mpd connects the links ''all or none.''

`multilink`
: This command enables multi-link PPP on the bundle. This option is required in both directions if there is more than one link in the bundle. However, multi-link PPP is sometimes useful on single links when the link MTU is low; multi-link PPP allows arbitrarily long packets to go over a link in fragments.

  This option defaults to `enable` and `accept` unless there is only one link in the bundle, in which case it defaults to `disable` and `deny` (i.e., normal non-multilink PPP).

`shortseq`
: This option is only meaningful if multi-link PPP is negotiated. It proscribes shorter multi-link fragment headers, saving two bytes on every frame. It defaults to `enable` and `accept`.

`compression`
: When compression support is compiled into mpd, this option must be enabled in order to activate it.

`encryption`
: When encryption support is compiled into mpd, this option must be enabled in order to activate it.

`crypt-reqd`
: Require encryption. When encryption support is compiled into mpd, this option causes mpd to drop the link if encryption cannot be succesfully negotiated in both directions. This means in effect, ''use encryption or don't connect at all.''

`round-robin`
: When multilink PPP is enabled and there are two or more links connected, this option enables round robin packet scheduling across the links. That is, instead of breaking each packet into fragments and delivering the individual fragments over all the links in the bundle, each packet is delivered whole over single link. The link is chosen by rotating in round-robin fashion over all the links in the bundle.

  This option normally would result in higher latency and possible delays due to TCP packets being received out of order; however, it may help in certain situations of high packet loss or broken peer equipment.

  The default for this option is off.

`noretry`
: This tells mpd to not attempt to reconnect a bundle whose last link has closed. This is needed for dialin-only bundles, where mpd doesn't know how to initiate an outgoing connection.

`radius-auth`
: Enable authentication via RADIUS. All authentication methods are supported (PAP, CHAP, MS-CHAPv1, MS-CHAPv2). When using IAS as RADIUS server you can not use CHAP, because CHAP needs a plaintext password on the server, and Windows servers only store the NT-HASH. Password changing is currently not supported.

`radius-fallback`
: This option is only relevant when `radius-auth` is enabled. If disabled, authentication is only attempted via RADIUS. Otherwise, if authentication via RADIUS fails, mpd will attempt to authenticate using the traditional method using `mpd.secrets`.

`radius-acct`
: Enable accounting via RADIUS. Accounting is implemented at bundle-level (in BundJoin() and BundLeave()). Accounting is done per link. There are some minor problems:

  - If user enters `quit` on the console, Account-stop will not be sent.
  - If CCP fails but is required, Account-start and Account-stop are sent, but they shouldn't be.

  Note: Its possible to use RADIUS-accounting without RADIUS-authentication.

  The following attributes are set when doing accounting requests:

  `RAD_FRAMED_IP_ADDRESS`

  `RAD_ACCT_STATUS_TYPE`

  `RAD_ACCT_SESSION_ID`

  `RAD_ACCT_MULTI_SESSION_ID`

  `RAD_USER_NAME`

  `RAD_ACCT_LINK_COUNT`

  `RAD_ACCT_AUTHENTIC`

  `RAD_ACCT_TERMINATE_CAUSE`

  `RAD_ACCT_INPUT_OCTETS`

  `RAD_ACCT_INPUT_PACKETS`

  `RAD_ACCT_OUTPUT_OCTETS`

  `RAD_ACCT_OUTPUT_PACKETS`

  `RAD_ACCT_INPUT_GIGAWORDS`

  `RAD_ACCT_OUTPUT_GIGAWORDS`

  `RAD_ACCT_SESSION_TIME`

`tcp-wrapper`
: With this option mpd uses `/etc/hosts.allow` everytime a connection is made from the client. This is useful if you would like to restrict access to mpd. This option affects all TCP connections, including PPTP and the TCP device layer.

  The console is also affected, but a bug requires this option to be set consistently on all bundles for it to work reliably.

`max-logins` *num*
: Limit the max. amount of concurrent logins with the same username. If set to zero, then this feature is disabled.

4.3 Interface layer commands

This chapter describes commands that configure the interface layer. All of these commands apply to the currently active bundle.

Note that while most of the time mpd is used for transmitting IP traffic, it is designed to support other (currently unimplemented) protocols such as AppleTalk, IPX, etc. This is why the Interface layer (which is protocol independent) is distinct from the *IP Control Protocol (IPCP) layer* (section 4.4 , page 26) which is specific to IP.

`set iface addrs` *local-ip remote-ip*
: This command is only required when dial-on-demand is enabled. Normally, mpd configures the interface with the IP addresses that were negotiated when connecting the link. Since dial-on-demand configures the interface before actually connecting, mpd has to be told initial local and remote IP addresses to give the interface. These addresses don't have to correspond to the ''real'' ones; in fact, both addresses can be completely fictional. If and when different addresses are negotiated later, mpd will automatically renumber the interface and adjust the routes.

`set iface route` *address[/width]*
: This command associates a route with the bundle. Whenever the interface is configured up, all associated routes are added. A route of **default** indicates the default route. Otherwise, the route is a network address with optional netmask width (e.g., **192.168.1.0/24**). If the netmask width is omitted, a single host route is assumed (i.e., a width of 32).

  Routes are automatically removed when the interface is brought down.

`set iface mtu` *value*
: This command sets an upper bound on the MTU that will be configured on the interface when the bundle comes up. This value is not used at all during link negotiation; there are separate bundle and link commands for configuring that. Even if a higher bundle MTU is negotiated, this limit will still apply.

  This command is useful when you want to manually restrict the MTU of the interface for other reasons, e.g., if you're also doing IPSec. The default value is 1500.

`set iface idle` *seconds*
: Sets the idle timeout value for the bundle. If no qualifying outgoing packets are transmitted for *seconds* seconds, the connection is brought down. An idle timeout of zero disables this feature.

  Mpd attempts to be intelligent about what a qualifying packet is. Idle time should only apply to packets that are generated locally and represent real local activity of some sort. For example, NTP traffic, ICMP echo replies, and TCP reset packets do not count as outgoing demand.

  When the idle timeout occurs, if the dial-on-demand option is enabled, mpd goes back into dial-on-demand mode. Otherwise, the interface is brought down and all associated routes removed.

`set iface session` *seconds*
: Sets the session timeout value for the bundle.

`set iface up-script` *script*

`set iface down-script` *script*

  Mpd can optionally run a user program every time the interface is brought up or down. The **up-script** is called like this:

```
script interface inet local-ip remote-ip authname [ dns1 server-ip ] [ dns2 server-ip ]
```

  The **down-script** is called like this:

```
script interface inet authname
```

`set iface enable` *option ...*

set iface disable *option ...*" Enable and disable the various interface layer options for the bundle.

The options available at the interface layer are:

`on-demand`
: This option causes the interface to operate in dial-on-demand mode, allowing you to have a ''virtual full time'' connection. An OPEN event causes the interface to be configured, but the actual connection of the link is delayed until qualifying outgoing traffic is detected. Moreover, after an idle timeout, the interface is not brought down; further traffic will bring the link back up again.

`proxy-arp`
: When this option is enabled, if after link negotiation the peer's IP address is determined to lie on a local subnet, then mpd will arrange for the local machine to install a proxy ARP entry for the remote machine's IP address.

For example, suppose the local machine lies on a LAN with address 192.168.1.10, and another machine will be connecting via mpd and using the LAN address 192.168.1.20. Then these commands would set up proxy ARP for the remote machine:

```
set iface enable proxy-arp

set ipcp 192.168.1.10/32 192.168.1.20/32
```

`tcpmssfix`
: This option causes mpd to adjust outgoing TCP data so that the requested segment size is not greater than the amount allowed by the interface MTU.

This is necessary in many setups to avoid problems caused by routers that drop ICMP Datagram Too Big messages. Without these messages, the originating machine sends data, it passes the rogue router then hits a machine that has an MTU that is not big enough for the data. Because the IP Don't Fragment option is set, this machine sends an ICMP Datagram Too Big message back to the originator and drops the packet. The rogue router drops the ICMP message and the originator never gets to discover that it must reduce the fragment size or drop the IP Don't Fragment option from its outgoing data.

The code is based on tcpmssd wich was written by Ruslan Ermilov <ru@FreeBSD.org> based on work done by Patrick Bihan-Faou <patrick@mindstep.com>.

This option is disabled by default.

`radius-mtu`
: This option causes mpd to use the MTU from the RADIUS server. It has the same effect as `set iface mtu`.

`radius-idle`
: This option causes mpd to use the idle timeout from the RADIUS server. It has the same effect as `set iface idle`.

`radius-session`
: This option causes mpd to use the session timeout from the RADIUS server. It has the same effect as `set iface session`.

`radius-route`
: This option causes mpd to use the routes list from the RADIUS server. It has the same effect as `set iface route`.

`radius-acl`
: This option causes mpd to use the Access Control Lists (ACLs) taken from the RADIUS server. This ACLs may include ipfw rules, dummynet pipes and dummynet queues. To use this feature you should add to your RADIUS server such dictionary:

```
#----------------------------------------------------------
# dictionary.mpd

VENDOR          mpd             12341

ATTRIBUTE       mpd-rule        1       string          mpd
ATTRIBUTE       mpd-pipe        2       string          mpd
ATTRIBUTE       mpd-queue       3       string          mpd
#----------------------------------------------------------
```

After this you can write in your RADIUS configuration something like:

```
mpd-pipe += "1=bw 10Kbyte/s",
mpd-pipe += "5=bw 20Kbyte/s",
mpd-rule += "1=pipe %p1 all from any to any in",
mpd-rule += "2=pipe %p5 all from any to any out",
mpd-rule += "100=allow all from any to any",
```

When mpd receives these parameters it will

```
ipfw(8)
```

to create firewall rules, pipes and queues with unique numbers starting from 10000. To the end of each rule will be added "via ngX" to make the rule apply only to that client's networking interface.

As a result of this example we would get these commands executed:

```
ipfw pipe 10000 config bw 10Kbyte/s
ipfw pipe 10001 config bw 20Kbyte/s
ipfw add 10000 pipe 10000 all from any to any in via ng0
ipfw add 10001 pipe 10001 all from any to any out via ng0
ipfw add 10002 allow all from any to any via ng0
```

When the link goes down, all created rules will be removed.

4.4 IPCP layer commands

This chapter describes commands that configure the IP Control Protocol (IPCP) layer. All of these commands apply to the currently active bundle.

`set ipcp ranges` *local/width remote/width*
: This command determines what IP addresses mpd will allow to be negotiated at the local and remote ends of the link. For each endpoint, we have a target address and a netmask width. The *width* determines how flexible we are, i.e., how close the actual negotiated address must be to the target address. A *width* of 32 means they must match exactly; a *width* of zero means any address is suitable. For example, `192.168.1.17/25` means that IP address `192.168.1.17` is desired, but any IP address in the range `192.168.1.0` through `192.168.1.128` is acceptable.

By convention, the *local* address may be `0.0.0.0` to request that the remote server assign us an IP address. Of course, for this to work the remote side must know *a* *priori* what our local IP address should be.

The *remote* address should *not* be `0.0.0.0`. This is so if the peer requests `0.0.0.0`, we have some address to give him. The *width* may of course be zero.

If the two sides cannot agree on the IP address assignments after repeated negotiation attempts, then the connection will fail. This is manifested with the error message ''IPCP: not converging.''

`set ipcp dns` *primary* [*secondary*]
: Some PPP clients request DNS server information from their remote peer. This commands enables mpd to have an answer for any such clients. This command is especially useful for supplying information to PPTP clients. One or two DNS server IP addresses may be given. An address of `0.0.0.0` erases that entry.

`set ipcp nbns` *primary* [*secondary*]
: Some MIcrosoft PPP clients request NetBIOS name server (NBNS) information from their remote peer. This commands enables mpd to have an answer for any such clients. This command is especially useful for supplying information to PPTP clients. One or two NBNS server IP addresses may be given. An address of `0.0.0.0` erases that entry.

`set ipcp accept` *option ...*

`set ipcp deny` *option ...*

`set ipcp enable` *option ...*

`set ipcp disable` *option ...*

`set ipcp yes` *option ...*

`set ipcp no` *option ...*

These commands configure various IPCP options. The **vjcomp** option is *bi-directional* in that it can be independently enabled and disabled in each direction.

The **enable** and **disable** commands determine whether we want the corresponding option. The **accept** and **deny** commands determine whether we will allow the peer to request the corresponding option.

The `yes` command is the same as **enable** and **accept**. The `no` command is the same as **disable** and **deny**.

The options available at the IPCP layer are:

`vjcomp`
: This option enables Van Jacobson TCP header compression, which saves several bytes per TCP data packet. You almost always want this option.

Default **enable** and **accept**.

`req-pri-dns`

`req-sec-dns`

`req-pri-nbns`

`req-sec-nbns`

Enabling these options causes mpd to request primary and/or secondary DNS and/or NBNS servers from the remote peer during negotiation.

If any DNS servers are supplied by the peer, they will appear as parameters to the script specified by the `set iface up-script` command, if any.

Currently, mpd does not use the NBNS values for anything; they just appear in the log. A future revision may actually do something with them.

`radius-ip`
: When using RADIUS for authentication, enable IP address assignment via RADIUS as well.

4.5 Link layer commands

This chapter describes commands that configure the link layer. All of these commands apply to the currently active link, i.e., the link shown at the command line prompt.

These are only the type-independent link layer commands. To configure properties specific to a particular link type, see the chapter describing that type.

`set link type` *type*
: This command tells mpd what the link type for this link is. This should generally be the first command in the `mpd.links` entry for each link.

`set link latency` *microseconds*

`set link bandwidth` *bits-per-second*
: These commands are relevant when multi-link PPP is active. They affect the way in which packets are chopped up into fragments before being sent over the various links that make up the bundle.

  To motivate the idea, imagine a bundle that had a modem link and a 1.5Mbps T1 link. If mpd sent each packet in two equal sized fragments over these links, then by the time the modem got around to transmitting the first byte of its fragment, the T1 link would have probably already sent the whole other fragment. Clearly this is not very good. By factoring in the latency and bandwidth parameters for each link, mpd can distribute the fragments in a more intelligent way.

  Mpd attempts to distribute bytes over the links so that (if the configured parameters are accurate) the last byte of each fragment arrives at the peer at the same time on each link. This minimizes latency. However, if you only care about maximizing throughput, simply set all of the latency values to zero.

  If all of your links are of the same type and speed (which is often the case), then they should be configured with the same values (or just not configured at all, since all links default to the same values anyway). Then mpd will distribute packets in equal sized fragments over the links.

`set link mtu` *numbytes*

`set link mru` *numbytes*
: The `set link mtu` command sets the maximum transmit unit (MTU) value for the link. This is the size of the largest single PPP frame (minus PPP header) that this link will transmit, unless the peer requests an even lower value. The default value is 1500 bytes.

  The `set link mru` command sets maximum receive unit (MRU) value for the link, which is the size of the largest single PPP frame (minus PPP header) that this link is capable of receiving. The default value is 1500 bytes.

  If PPP multilink is negotiated on a link, then these values are less important, because multilink allows PPP frames themselves to be fragmented, so a PPP frame can always pass through no matter how small the MTU is in a particular direction.

  Otherwise, mpd is responsible for making sure that the MTU configured on the system networking interface is low enough so that the largest transmitted IP packet does not exceed the peer's negotiated MRU after it becomes a PPP frame. This includes e.g. PPP encryption and/or compression overhead.

However, mpd does not account for overhead that occurs "outside" of the PPP frame. For example, when using link types such as PPTP that encapsulate PPP frames within IP packets, a large outgoing "inner" IP packet can result in a fragmented "outer" IP packet, resulting in suboptimal performance. In this situation it may be useful to set the link MTU to a lower value to avoid fragmentation.

`set link accmap` *value*

This sets the desired asynchronous control-character map for the link at the local end. This option is only relevant for the asynchronous link types (i.e., **modem** and **tcp**). It determines which control characters need to be escaped.

The *value* is expressed as a 32-bit hex value; the default is `0x000a0000`, which escapes the Control-S and Control-Q characters.

`set link ident` *string*

This enables the sending of an identification string to the peer via the LCP Ident code. The Ident string is sent when the link is brought up. This is useful for deubgging, etc. and is meant to be human-readable. However, it confuses some broken PPP implementations.

Setting an empty string disables this feature; this is the default.

`set link fsm-timeout` *seconds*

This command is analogous to the same command at the bundle layer, but it applies to link-layer FSM's such as Link Control Protocol (LCP). The default is two seconds; normally this value shouldn't be changed.

`set link keep-alive` *seconds max*

This command enables the sending of LCP echo packets on the link. The first echo packet is sent after *seconds* seconds of quiet time (i.e., no frames received from the peer on that link). After *seconds* more seconds, another echo request is sent. If after *max* seconds of doing this no echo reply has been received yet, the link is brought down.

If *seconds* is zero, echo packets are disabled. The default values are five second intervals with a maximum no-reply time of fourty.

This feature is especially useful with modems when the carrier detect signal is unreliable. However, in situations where lines are noisy and modems spend a lot of time retraining, the *max* value may need to be bumped up to a more generous value.

`set link max-redial` *num*

When a link fails to connect, mpd automatically retries the connection immediately. This command limits the number of consecutive retries. After *num* attempts, mpd will give up.

When there is another open event, new dial-on-demand traffic, etc. mpd will try again, starting over at zero.

The default value is zero, which means unlimited retries.

If `max-redial` is set to -1, then mpd will never redial. This setting should be used with links that are dedicated for dial-in.

`set link accept` *option ...*

`set link deny` *option ...*

`set link enable` *option ...*

`set link disable` *option ...*

`set link yes` *option ...*

`set link no` *option ...*

These commands configure various link options. Most options are *bi-directional* in that they can be independently enabled and disabled in each direction.

The **enable** and **disable** commands determine whether we want the corresponding option. The **accept** and **deny** commands determine whether we will allow the peer to request the corresponding option.

Note that when talking about the authentication options PAP and CHAP, when you **enable** an option you're saying you are going to require a login and password from the peer. When you **accept** an option you're saying you will allow the peer to require a login and password from us.

The `yes` command is the same as **enable** and **accept**. The `no` command is the same as **disable** and **deny**.

The options available at the link layer are:

`pap`

PAP style authentication. Note that this style of authentication is insecure, since the password crosses the link in plaintext.

Default **disable** and **accept**.

`chap`

CHAP style authentication. This style of authentication is safer than PAP, because only a hash of the password is passed over the link. Mpd supports MD5 style CHAP and Microsoft style CHAP versions 1 and 2. Mpd will prefer MD5 CHAP over Microsoft CHAP, unless the link is a pptp link.

This parameter is an alias for `md5-chap` `ms-chapv1` `ms-chapv2`

`chap-md5`

Traditional CHAP MD5 style authentication.

Default **disable** and **accept**.

`chap-msv1`

Microsoft CHAP style authentication.

Default **disable** and **deny**.

`chap-msv2`

Microsoft CHAP style authentication Version 2.

Default **disable** and **accept**.

`acfcomp`

Address and control field compression. This option only applies to asynchronous link types. It saves two bytes per frame.

Default **enable** and **accept**.

`protocomp`

Protocol field compression. This option saves one byte per frame for most frames.

Default **enable** and **accept**.

`magicnum`

`check-magic`

The `magicnum` option enables using a magic number for the local end of the PPP link. This causes a unique number to be included in each LCP packet we send, which helps detect loopback conditions.

The `check-magic` option causes mpd to verify that the peer's magic number is correct in all received LCP frames.

Some old broken PPP implementations don't handle magic numbers correctly, so these options need to be disabled in these cases.

Default for both options is **enable**.

Note that the two most common reasons for seeing ''loopback condition detected'' on a modem link are:

- The modem is in command mode and is echoing back all of our frames.
- The PPP server is not in PPP mode, but is giving a shell prompt or somesuch and echoing back all of our frames.

`passive`

Enables passive mode for this link. This is useful on some full time connections. See RFC 1661 for more information about this option.

Default **disable**.

`callback`

Enables PPP callback. If the remote peer wants to, we will hangup immediately after connecting and wait to be called back.

This option is currently unimplemented.

Default **disable**.

`no-orig-auth`

Normally, if PAP or CHAP is enabled, we require the peer to authenticate to us at the beginning of each connection. This option temporarily disables this requirement if we are the one who originated the connection and the peer rejects our request for a login.

This is useful when the same link is used for both dial-in and dial-out.

Default **disable**.

4.6 Compression layer commands

This chapter describes commands that configure the Compression Control Protocol (CCP) layer. All of these commands apply to the currently active bundle.

`set ccp accept` *option ...*

`set ccp deny` *option ...*

`set ccp enable` *option ...*

`set ccp disable` *option ...*

`set ccp yes` *option ...*

`set ccp no` *option ...*

These commands configure various CCP options. Each direction of traffic (transmit and receive) is independent from the other. All options default to **disable** and

deny.

The **enable** and **disable** commands determine whether we desire the corresponding option on received data. The **accept** and **deny** commands determine whether we will allow the peer to request the corresponding option (which we must enable on our transmitted data).

The `yes` command is the same as **enable** and **accept**. The `no` command is the same as **disable** and **deny**.

The options available at the CCP layer are listed below. Note that the STAC and MPPC options require proprietary files that are not included with the normal mpd distribution.

`stac`
: This option enables STAC compression.

`mppc`
: This option is required in order to enable the Microsoft Point-to-point compression (MPPC) and/or Microsoft Point-to-point encryption (MPPE) protocols. However, this option does not actually enable MPPC or MPPE; you must use additional options (below) for that.

  This option is required because of the way Microsoft implemented MPPC and MPPE as additional, optional sub-components of the MPPC protocol.

`mpp-compress`
: Enables MPPC compression.

`mpp-e40`
: Enables 40-bit MPPE encryption.

`mpp-e56`
: Enables 56-bit MPPE encryption. Note: 56 bit MPPE encryption is only supported in FreeBSD 4.5 or later.

`mpp-e128`
: Enables 128-bit MPPE encryption.

`mpp-stateless`
: Enables stateless mode. This mode requires slightly more CPU time and is somewhat less secure, but allows faster recovery in the face of dropped packets.

`radius`
: When enabled, MPPE types (40, 56 or 128 bit) and MPPE policies (whether encryption is required or optional) are controlled by the RADIUS server. In order to get MPPE to work you have also to enable mppc in `mpd.conf`.

4.7 Encryption layer commands

This chapter describes commands that configure the Encryption Control Protocol (ECP) layer. All of these commands apply to the currently active bundle.

`set ecp key` *string*

: Sets the encryption key.

`set ecp accept` *option ...*

`set ecp deny` *option ...*

`set ecp enable` *option ...*

`set ecp disable` *option ...*

`set ecp yes` *option ...*

`set ecp no` *option ...*

These commands configure various ECP options. Each direction of traffic (transmit and receive) is independent from the other. All options default to **disable** and **deny**.

The **enable** and **disable** commands determine whether we desire the corresponding option on received data. The **accept** and **deny** commands determine whether we will allow the peer to request the corresponding option (which we must enable on our transmitted data).

The `yes` command is the same as **enable** and **accept**. The `no` command is the same as **disable** and **deny**.

The options available at the ECP layer are:

`des`

This option enables DES encryption.

5. Device Types

5.1 Modem type commands

This chapter describes commands that are specific to modem type links. These commands apply to the currently active link, and are only valid if the currently active link has type **modem**.

`set modem device` *devname*
: Modem links operate over serial ports. This command tells mpd which serial port to use for this link, and is required. The *devname* should be the pathname of a serial device, e.g., `/dev/cuaa0`.

`set modem var` *$variable string*
: Pre-sets the chat script variable *$variable* to contain the string *string*. See *the chapter on chat scripting* (section 6. , page 40) for more information on chat variables.

`set modem speed` *speed*
: This commands sets the initial serial port speed when opening the serial device, e.g., 9600, 57600, 115200. Note that you can always change the serial port speed later from within the chat script.

`set modem script` *connect-script*
: This command tells mpd which chat script to run in order to initiate a PPP connection for this link. The *connect-script* should correspond to a label in `mpd.script`. Mpd will jump to this label after opening the serial port device. If *connect-script* is not set, then the chat phase of the connection is skipped (e.g., if you have a direct null modem connection).

`set modem idle-script` *idle-script*
: This command tells mpd what to do with the serial port when the link is not connected. If no *idle-script* is set, then mpd will leave the serial port closed when not connected. Otherwise, when the link is disconnected mpd runs the *idle-script* chat script. If this script returns `failure`, then mpd restarts the script.

  Otherwise, (i.e., if it returns `success`) mpd examines the contents of the `$IdleResult` chat variable to decide what to do next. If it equals `answer` then mpd will assume that an incoming call has been answered and immediately enter PPP negotiation. If it equals `ringback` then mpd will close the serial port, reopen it, and initiate an outgoing connection normally (i.e., using *connect-script*).

  If `$IdleResult` contains anything else or is not set, then mpd acts as if `failure` had occurred and simply restarts the script.

`set modem watch` *+/-signal ...*
: Mpd normally tracks the carrier detect signal on the serial port and drops the connection when this signal is lost. You can disable this behavior with **set modem watch -cd**. Also, mpd can do the same thing with the DSR signal, though the default is to ignore DSR. To enable watching of the DSR signal, use **set modem watch +dsr**.

5.2 Netgraph device type commands

This chapter describes commands that are specific to netgraph type links. These commands apply to the currently active link, and are only valid if the currently active link has type **ng**.

Note that this device type has no direct way to detect a physical layer disconnection (i.e., what would be analogous to loss of carrier detect) except by receiving a write error. One way to achieve this is to insure that when the physical layer goes down, the connection to the Netgraph hook is broken.

`set ng node` *nodepath*

`set ng hook` *hook*

These commands configure which Netgraph node, and which hook on that node, mpd is supposed to connect to. The node is found via the absolute Negraph address *nodepath* and must have a free hook named *hook*. Both of these commands are required.

The hook should be connectable directly to a link hook of the `ng_ppp(3)` netgraph node type. That is, it should be prepared to transmit and receive PPP frames starting with the address and control fields (unless compressed), followed by the PPP protocol number and information fields, but containing no checksum field.

5.3 TCP device type commands

Note: The TCP device type is currently broken and will not compile (it needs to be updated to support netgraph).

This chapter describes commands that are specific to TCP type links. These commands apply to the currently active link, and are only valid if the currently active link has type **tcp**.

`set tcp mode [` *active* | *passive*]

Set the link for active or passive mode. In active mode, the local side initiates all connections. In passive mode, the local side waits for incoming connections (in a future revision bi-directional connections will be supported).

`set tcp self` *ipaddr* [*port*]

Sets the local IP address and port for the TCP connection. If none is specified, any suitable local IP address is used.

`set tcp peer` *ipaddr* [*port*]

Sets the peer IP address and port for the TCP connection. If none is specified, any peer may connect to us.

5.4 UDP device type commands

This chapter describes commands that are specific to UDP type links. These commands apply to the currently active link, and are only valid if the currently active link has type **udp**. It is suggested that the `set iface mtu` command be used when this link type is in use to avoid IP fragmentation.

Note: Technically this device type is broken, because PPP links are not allowed to re-order frames, yet UDP packets can be delivered out of order. Therefore, if you use this link type you should not enable multi-link PPP, PPP encryption, or PPP compression. Otherwise, out of order frames will be dropped, leading to performance degradation.

`set udp self` *ipaddr* [*port*]

Sets the local IP address and port for the UDP connection. If none is specified, any suitable local IP address is used.

`set udp peer` *ipaddr* [*port*]

Sets the peer IP address and port for the UDP connection. If none is specified, any peer may talk to us.

`set udp origination local`

`set udp origination remote`" This command is only required if MPPE encryption is used on the bundle. MPPE requires knowledge of which peer "originated" the connection and which peer "answered" the connection. Since UDP is connectionless, there's no way for mpd to determine this reliably. Therefore one peer must be configured with `local` origination and the other with with `remote` origination. The default is unknown.

5.5 PPTP device type commands

This chapter describes commands that are specific to PPTP type links. These commands apply to the currently active link, and are only valid if the currently active link has type **pptp**.

The PPTP protocol can be most easily understood as just another link layer type, where the link layer medium just happens to be an IP connection. So, for example, instead of configuring a telephone number to dial as you would with a modem, you configure an IP address to connect to. Everything else that's above the link layer functions exactly the same. Hence PPTP allows you to ''tunnel'' PPP frames over IP.

Note that PPTP connections are initiated by a TCP connection from one machine to another, and that servers usually listen to TCP port 1723 (and this is the default for *port* in the commands below). PPTP also uses the GRE protocol, which has IP protocol number 47. Your firewall may need to be adjusted to allows this type of IP packet.

Currently, mpd will not allow you to create a bundle containing more than one PPTP link.

`set pptp mode [` *active* | *passive*]
: Set the link for active or passive mode. In active mode, the local side initiates all connections. In passive mode, the local side waits for incoming connections (in a future revision bi-directional connections will be supported).

`set pptp self` *ipaddr* [*port*]
: Sets the local IP address and port for the PPTP connection.

`set pptp peer` *ipaddr* [*port*]
: Sets the peer IP address and port for the PPTP connection. This command applies to both incoming and outgoing connections. For outgoing connections, this command is required in order to specify where to connect to. For incoming connections, this command is optional; if not given, mpd accepts incoming connections from any host. Otherwise, only connections from the stated IP address (and, optionally, port) are allowed.

`set pptp phonenum` *number*
: Sets the called telephone number to use when initiating a PPTP connection. For most VPN applications this is ignored, but in certain cases an actual phone number is required. The default is the empty string.

`set pptp enable` *option ...*

`set pptp disable` *option ...*

: Enable and disable PPTP device type options for the link.

The following options are supported:

`originate`
: This option enables the initiation of PPTP connections to the peer. If this option is disabled, mpd will not initiate any PPTP connections. You must configure the peer's IP address with the **set pptp peer** command.

`incoming`
: This option enables the acceptance of incoming PPTP connections. If this option is disabled, mpd will not accept any incoming connections.

`outcall`
: In PPTP, each individual PPP connection between two IP hosts (there may be several, and these are not to be confused with the *single* TCP connection between any two IP hosts), is initiated as either an incoming or an outgoing call. This is to allow,

for example, making an actual outgoing telephone call via a remote access server, as well as the more common use of forwarding of an incoming telephone call from an access server to a remote PPTP server.

When this option is enabled, mpd will initiate outgoing calls; otherwise mpd will initiate incoming calls. Although it seems that incoming would be more correct, the default is outgoing, as this is consistent with the behavior of the Microsoft PPTP dial-up adapter client.

Mpd will accept either type of call on an incoming PPTP connection.

`delayed-ack`
: Enable delayed ACK's. This can improve throughput on reliable links. Default is on.

`always-ack`
: Always include ACK even if already sent. This can improve throughput on unreliable links. Default is off.

`windowing`
: Enables the windowing mechanism specified by the protocol. Disabling this will cause Mpd to violate the protocol, possibly confusing other PPTP peers, but often results in better performance. The windowing mechanism is a design error in the PPTP protocol; L2TP, the successor to PPTP, removes it. You need a recent version of FreeBSD (NGM_PPTPGRE_COOKIE >= 1082548365) in order to get this feature. Default is on.

5.5.1 PPTP setup hints

If what you want to do is setup a PPTP server for remote clients to connect to your LAN, you would want something like this for each `mpd.links` entry:

```
#
# Act like a PPTP server allowing clients to connect
#
Pptp0:
    set link type pptp
    set pptp enable incoming
    set pptp disable originate
```

You may also need to enable proxy-ARP on each interface if you're assigning local Ethernet LAN addresses, as well as configuring DNS and NBNS server information for client bundles using the `set ipcp dns` and `set ipcp nbns` commands.

If you want multiple *simultaneous* clients to be able to connect, you should define (and load) multiple bundles, one for each potential client. Each bundle should negotiate a different peer proxy-ARP IP address to avoid conflicts. For example:

```
#
# mpd.conf: allow up to 2 clients
#
default:
    load Pptp1
    load Pptp2

Pptp1:
    new -i ng0 Pptp1 Pptp1
    ...
    set ipcp ranges 1.1.1.1/32 1.1.1.101/32
    ...

Pptp2:
    new -i ng1 Pptp2 Pptp2
    ...
    set ipcp ranges 1.1.1.1/32 1.1.1.102/32
    ...
```

If instead you want to set up a peer-to-peer symmetrical tunnelling arrangement (i.e., normal IP routing using PPP over PPTP), you might want something like this (here `1.1.1.1` is the local IP address and `2.2.2.2` is the remote IP address):

```
#
# Tunnelling PPP traffic over a PPTP connection with peer 2.2.2.2
#
Pptp1:
    set link type pptp
    set pptp self 1.1.1.1
    set pptp peer 2.2.2.2
    set pptp enable incoming originate
```

On the remote machine, you'd have the same entry with the `self` and `peer` addresses reversed.

Note that if you are connecting to an NT server, your authentication name must include the NT domain name. For example:

```
    set bundle authname "DOMAIN\\username"
```

For updating Windows 95 and 98 clients so they work properly, see VPN Update for Windows 98 and Dial-Up Networking 1.3 Available

5.6 PPPoE device type commands

This chapter describes commands that are specific to PPPoE type links. These commands apply to the currently active link, and are only valid if the currently active link has type **pppoe**.

Your kernel must be compiled with ''options NETGRAPH'' to netgraph-enable the Ethernet interfaces (if your system is new enough to have the `/modules/ng_ether.ko` KLD installed, then this is no longer true; you only need load this KLD).

PPPoE requires two pieces of configuration information: the name of the Ethernet interface to use, and a service name, which should be supplied by your ISP. If you do not have a service name, then the empty string (the default) often suffices.

`set pppoe iface` *interface-name*
: Set the name of the Ethernet interface to use. PPPoE does not affect normal use of the interface.

`set pppoe service` *service-name*
: Sets the requested PPPoE service name. Setting service name to "*" on link with allowed incoming connections allows to accept incoming connection requests with

any service name specified.

`set pppoe enable` *option ...*

`set pppoe disable` *option ...*

Enable and disable PPPoE device type options for the link.

The following options are supported:

`originate`
This option enables the initiation of PPPoE connections to the peer. If this option is disabled, mpd will not initiate any PPPoE connections.

`incoming`
This option enables the acceptance of incoming PPPoE connections. If this option is disabled, mpd will not accept any incoming PPPoE connections. You must configure iface and service parameters before using this option.

6. Chat Scripting

Mpd includes a powerful chat scripting language for use with links of type **modem**, i.e., asynchronous serial ports. The language is interpreted and executed dynamically and supports fully event driven processing with timeouts and regular expression pattern matching.

Much of what is described below is more readily understood by looking at the `mpd.script` file that is included with mpd.

6.1 Script file format and execution

The syntax for `mpd.script` is similar to the other mpd configuration files. Lines beginning with a hash sign are ignored. A **label** sits on its own line, starting in the first column and ends with a colon character.

Chat commands are indented with tab characters and written one per line. Contrary to `mpd.conf` and `mpd.links`, the processing of chat commands *does not* stop when a blank line is encountered.

Commands are executed sequentially. Execution continues without stopping until there is an error or one of the following commands is encountered:

- `success` The script returns success.
- `failure` The script returns failure.
- `wait` Execution of the script is halted until an **event** occurs.

Events are described in the next section.

6.2 Events

An **event** is either the expiration of a timer, or the matching of some pattern in the input. Events that haven't happened yet are **pending events**. Pending events are created with the `timer` and `match` commands, respectively.

Each pending event has an associated label (called the **target label**) of the event. When and if the event occurs, execution resumes starting at the target label. Events only occur during the execution of a `wait` command.

Events are grouped into named **sets**. Events in the same set are usually related, in the sense that they represent alternate outcomes to the same situation.

A set of pending events may be explicitly **cancelled** (i.e., forgotten) before they occur with the `cancel` command. All events in the set are cancelled.

The other way a set of pending events may be cancelled is implicitly, when any event in the set occurs. Execution resumes at the target label associated with the event that occurred, and all other events in that set are immediately cancelled.

6.2.1 Timer events

Timer events are created with the `timer` command. The timer event occurs when the designated time has elapsed, starting from the time at which the `timer` statement was executed. Further execution continues starting with the target label specified by the `timer` command.

Timer events (like all events) can only occur during the execution of a `wait` command.

6.2.2 Match events

Match events are created with the `match` and `regex` commands. A match event occurs when the pattern specified in the `match` or `regex` command has been matched in the input. Characters are only input during the execution of `wait` commands. In order for a pattern to match, the first matching character of the pattern must be read *after* the event is created (i.e., during a subsequent `wait` command).

When the pattern is matched, chat script execution continues starting at the target label. When multiple patterns match on the same input character, mpd chooses the first one defined to be the one that matches.

The pattern is either plain text or an extended regular expression (see below).

6.3 Sets

A set is a collection of pending timer and/or match events. When any one event occurs, or upon execution of the appropriate `cancel` command, all events in the set are cancelled.

Multiple sets may be active at any time. The occurrence of an event from one set has no effect on events in other sets.

6.3.1 Special sets and targets

The `timer`, `match` and `regex` commands may specify (by omission) the **default set**. This set is exactly like any other set, except that it is automatically cancelled after the occurrence of *any* event. It may be explicitly named by using the empty string, (ie, `""`).

The `timer`, `match` and `regex` commands may also specify (by omission) the **default target**. This target, whose name is also the empty string, has the special propery that it always refers to ''the statement immediately following the next wait statement.'' That is, if the target of any event is the default target, the occurrence of that event just causes execution to continue starting with the statement following the wait statement.

Finally, when `all` is used with the ''cancel'' command, *all* sets (i.e., all pending events) are cancelled.

6.4 Variables

Variables hold simple strings. They are global and come in two types: **temporary** and **permanent**. Permanent variables persist over multiple invocations of the chat script, while temporary variables are forgotten each time the script completes.

Variables are specified with a dollar sign, then a letter, followed by letters, digits, and underscores. If the first letter is upper case the variable is permanent; otherwise it is temporary. The variable name may be optionally enclosed in curly braces.

Examples:

```
$initString
$My_variable_234
${i_am_safely_followed_by_a_letter}
```

6.4.1 Special variables

These variables have special meaning in chat scripts:

`$Login`
: Authorization login name as set by `set bundle authname`.

`$Password`
: The password corresponding to `$Login`.

`$modemDevice`
: The serial port device in use as set by the `set modem device` command, e.g., `/dev/cuaa0`. Changing this string does not change the port being used.

`$Baudrate`
Always equals the current baudrate (e.g., ''57600''). This is a read/write variable; setting it changes the baudrate. Attempting to set an invalid baudrate causes the script to fail.

`$matchedString`
When a `match` event occurs this string contains the input string that matched the pattern.

`$matchedString0`

`$matchedString1`

`$matchedString2`

`...`

When a `regex` event occurs or the `if match` or `if !match` commands are executed, and the regular expression pattern contains parenthesized subexpressions, these strings will be equal to the substrings matching each subexpression. `$matchedString0` is equal to the entire string, while `$matchedString1` equals the first subexpression match, `$matchedString2` the second, etc.

`$IdleResult`
This variable is used to return the result of an **idle script**; If the value is `answer`, mpd assumes an incoming call has been answered. If the value is `ringback`, mpd will initiate an outgoing connection. See the chapter on *Modem type commands* (section 5.1 , page 34) for more information.

`$$`
Always expands to a single dollar sign.

6.5 Script commands

`set` `$`*variable-name string*
Set the `$`*variable-name* equal to *string*. The *string* is variable expanded first.

`match` *name string label*

`match` *string label*

`match` *string*

Add a new pending event to event set *name* such that when *string* is seen in the input, resume execution at *label*. The match must be exact. All arguments are variable expanded.

If no *name* is given, or if *name* is the empty string, add the new event to the default set.

If no *label* is given, or if *label* is the empty string, the event causes execution to resume starting with the statement following the `wait` command (i.e., use the default target).

`regex` *name pattern label*

`regex` *pattern label*

`regex` *pattern*

Same as the `match` command, but matches an extended regular expression instead of an exact string. If *pattern* is not a valid extended regular expression, the script fails.

Regular expressions are matched on a line-by-line basis; the input matching the pattern cannot span more than one line. A line is terminated with either a newline or a carriage return-newline pair (the latter is more common). An end-of-line should be matched using a dollar sign rather than trying to match these characters directly, as mpd elides them from the input when testing the regular expression.

See `re_format(7)` for more information about extended regular expressions.

`timer` *name seconds label*

`timer` *seconds label*

`timer` *seconds*

Add a new pending event to set *name* such that when *seconds* seconds have elapsed, resume execution at *label*. All arguments are variable expanded.

If no *name* is given, or if *name* is the empty string, add the new event to the default set.

If no *label* is given, or if *label* is the empty string, the event causes execution to resume starting with the statement following the `wait` command (i.e., use the default target).

`wait`

`wait` *seconds*

In the first form, wait indefinitely for any event to happen. When an event happens, continue execution starting at the target label for that event.

The second form is equivalent to:

```
timer seconds

wait
```

`if` *string1* == *string2*
command "

`if` *string1* != *string2*
command "

Executes *command* if the two variable expanded strings are equal or not equal.

`if` *string* match *pattern*
command "

`if` *string* !match *pattern*
command "

Tests *string* against the regular expression *pattern* and executes *command* if *string* matches or does not match. Both *string* and *pattern* are variable expanded first.

`print` *string*
Ouput the *string* to the serial port. Variables are expanded, and the normal mpd C-style character escapes may be used.

`cancel` *name1* [*name2* ...]
Cancel all pending events in the sets *name1*, *name2*, etc. It is not an error to cancel a set which is already empty.

`goto` *label*
Jump to label *label*.

`call` *label*
: Call a subroutine at *label*.

`return`
: Return from subroutine.

`success`
: Exit the script successfully.

`failure`
: Exit the script unsuccessfully.

`log` *string*
: Print *string*, variable expanded, to the log file under logging level CHAT.

6.6 Errors

Any errors in the script cause the script to exit unsuccessfully. Examples of errors include:

- A `call` or `goto` to a non-existent label
- A `return` without a corresponding `call`
- Executing `wait` without any events pending
- Execution falling off the end of the script
- Trying to assign `$Baudrate` an invalid baudrate
- Trying to match with an invalid regular expression

6.7 The included mpd.script

The `mpd.script` that ships with mpd can usually be used as it is without modification for typical PPP applications. This script defines the following labels:

`DialPeer`
: This script causes mpd to attempt to identify the attached modem or ISDN Terminal Adapter device, configure the device, and dial a remote peer. This the label you want to use for the `set modem script` when acting as a dialup client.

  The variables that are used by this script are:

  `$Telephone`
  : The telephone number to dial.

  `$DialPrefix`
  : Dialing prefix to get an outside line, if any.

  `$ConnectTimeout`
  : Time in seconds to wait for the modems to connect. Default 45 seconds.

  `$SpeakerOff`
  : Set this to `yes` if you want to silence the modem speaker while dialing.

  `$Serial230K`
  : If your serial port is capable of running at 230K baud, set this to `yes`. Note: normal PC hardware is not capable of 230K.

  In addition, the following variables are required to be set for ISDN terminal adapters:

`$TA_Bonding`
: If the terminal adapter should do 2 B-channel multi-link PPP, set this to `yes`. Note: a terminal adapter doing multi-link PPP still *appears* to mpd as a single link PPP connection and should be configured as such.

`$TA_NoDoubleTelno`
: When `$TA_Bonding` is `yes`, don't double the dialed number. Normally, in such situations mpd dials using the command `ATDT${Telephone}&${Telephone}`. If your `$Telephone` string already contains both numbers, then set `$TA_NoDoubleTelno` to `yes`.

`$TA_56K`
: If you need to restrict the B channels to to 56K, set this to `yes`. Only necessary for some North American ISDN lines.

`$TA_VoiceCall`
: If you need to place a voice mode call, set this to `yes`. Only necessary for some North American ISDN lines. Not all terminal adapters support this.

`$TA_AuthChap`
: Some older terminal adapters require you to specify *a priori* whether CHAP or PAP authentication will be requested by the remote side. Set to `yes` to tell the terminal adapter to use CHAP.

`$TA_SwitchType`
: Set this to your ISDN line's switch type. Should be one of `NI-1`, `DMS-100`, `5ESS P2P`, or `5ESS MP`. Only required for North American ISDN.

`$TA_Dirno1`

`$TA_Dirno2`

`$TA_SPID1`

`$TA_SPID2`
: Set these to your ISDN line's directory numbers and SPID's for both B channels. Only required for North American ISDN.

`AnswerCall` This is an **idle script** that waits for an incoming call and then answers it. The variables that are used by this script are:

`$ConnectTimeout`
: Time in seconds to wait for the modems to connect. Default 45 seconds.

`$RingTimeout`
: How long to wait for a RING before giving up and trying again. Default 10 minutes.

`Ringback` This is also an idle script that waits for an incoming call, but instead of answering the call it ignores it and initiates an outgoing connection. This is useful for remotely bringing up a dialup client machine.

`$RingbackTimeout`
: How long before giving up (reset and try again). Default: 60 minutes.

`$RingStoppedTime`

For analog modems, we have to wait for the ringing to stop before trying to dial out, otherwise we'll inadvertently answer the incoming call. This value is set to the minimum time to wait before declaring that the ringing has stopped. Default: 8 seconds.

7. Troubleshooting

7.1 Troubleshooting

Some tips for troubleshooting. (This section is under construction).

For further help, subscribe to the `freebsd-net@freebsd.org` mailing list and ask your question there.

Packets won't flow.
: Make sure you have set **gateway_enable="YES"** in `/etc/rc.conf`, otherwise your FreeBSD box will not route packets. Alternately, invoke `sysctl -w net.inet.ip.forwarding=1` for immediate effect.

  Also, check your firewall settings. Mpd will create new interfaces which may need to be incorporated into your firewall rules. If you're doing PPTP, you need to allow TCP port 1723 and IP protocol 47 (GRE).

It doesn't work, and there are strange netgraph-related errors in the log.
: Make sure you have all the required netgraph KLD's loaded. You can check them by doing `kldstat -v | grep ng_` .

  Usually these are loaded on demand. If not, you can load them manually using `kldload(8)`.

  The following node types are or may be needed:

  - `ng_ppp`
  - `ng_socket`
  - `ng_iface`
  - `ng_bpf`
  - `ng_vjc`
  - `ng_mppc`
  - `ng_pptpgre`
  - `ng_ksocket`
  - `ng_tty`
  - `ng_pppoe`

Poor performance with Mac OS X PPTP clients.
: Upgrade to a version of FreeBSD newer than 4.7-RELEASE. Or apply this patch manually.

My configuration doesn't seem to be working as expected.
: Make sure your `mpd.conf` and `mpd.links` files are formatted properly. Labels must be left-justified, and blank lines must exist between sections to separate them.

I'm getting no valid response from the RADIUS server.
: Double-check the shared secret in the RADIUS servers config and the one specified in `mpd.conf` or `radius.conf`. Check the ports of the RADIUS server: they should be 1812 (authentication) and 1813 (accounting), wich are also the default ports of mpd. In the past unofficial port numbers (1645 and 1646) were used for RADIUS, but these are deprecated.

I'm getting "Operation not permitted" errors trying to run mpd.
: This is often caused by a combination of (a) using netgraph in KLD module form instead of statically compiling the various modules into the kernel via kernel

config(8) options, and (b) increasing your kernel security level (see the init(8) man page), which prevents KLD modules from being loaded. Change one of these things to fix the problem.

I've configured a link for dial-in, and sometimes after disconnecting mpd gets into some kind of infinite loop." This is because mpd is trying to "redial," which of course is not appropriate for dial-in links. Use `set link max-redial -1` to disable the redial function.

I've configured lots and lots of bundles. But after creating 100 of them (or some other power of 10) mpd starts getting "Address already in use" errors." This is because the names assigned to the netgraph nodes are too long, being truncated, and the conflicting with existing names. The maximum length of a netgraph node name is 15 characters. The names are derived from the bundle names assigned in `mpd.conf`. So try shortening your bundle names.

I'm using mpd as PPTP server for Windows XP clients, when accessing bigger websites, or listing larger ftp directories, then the connection seems to hang." Windows XP insists on a very low MTU (usualy 1396 Bytes), this needs fragmentation, if bigger packets should be transmited over the link. Fragmentation is negotiated at ICMP level, if there is a bad router somewhere in the network, who drops such packets, then the connection seems to hang. The first thing what you can try is reducing mpd's MTU value, by setting: `set link mtu 1300` and `set bundle disable multilink`, this should help in most cases. For TCP connections it's possible enabling the TCP-MSS-Fix: `set iface enable tcpmssfix` (available since mpd-3.15).

8. Internals

8.1 ToDo

A list of open tasks for MPD. This is a good startpoint if you wan't to support actively the development of MPD. After completing a task post it to MPD's maling list.

callback (client & server)
: Currently MPD doesen't support callback, but it should.

NAT
: MPD has currently no support for NAT, but it would be great, if he can support this. We intend not simply using libalias for this, instead a new netgraph node ng_nat would be the right way for doing this.

EAP
: Extensible Authentication Protocol. RFC 2284 EAP

Unimplemented Protocols
: lcp.c contains a list of protocols (gLcpConfOpts), each protocol with "false" is not implemented.

MPD GUI
: A GUI which connects to the terminal port of mpd and is able to configure MPD. If possible platform independent.

MPD Client-GUI
: A GUI which is able to initiate client-side connections. This should add the possibility for end-users (not admins) to start MPD connections from other clients (usualy Windows clients).

8.2 Authentication

This chapter describes implementation specific details of MPD.

Authentication protocols - a short overview
: Currently MPD supports these four authentication protocols: MD5-CHAP, MS-CHAPv1, MS-CHAPv2, and PAP.

  The traditional CHAP-MD5 needs the plaintext password stored on the server. The password hash is computed like this: md5(id + password + challenge), where the id is incremented after each authentication attempt. The challenge is generated by the server and then sent to the client (peer). The peer sends the hash to the server and the server genrates himself the hash using the plaintext password. If both hash are the same, the authentication succeeds.

  MS-CHAP doesen't need plaintext passwords on the server, but does need the hashed password either as NT-Hash or as LAN-Manager-Hash (the LAN-Manager-Hash is weak and shouldn't be used). MS-CHAPv1 uses DES as hashing algorithm and is weak, therefore don't use it! MS-CHAPv2 uses a peer challenge and a server challenge and uses SHA1 as hashing algorithm, so it's much more secure then MS-CHAPv1. MS-CHAPv2 requires the NT-Hash be available.

  When using PAP, the password is sent as plaintext over the network, therefore PAP should be avoided, if the link between the peers is not secure. On the other hand, PAP allows using any kind of password database on the server, even if the passwords are stored in non-reversible hashed format.

  Usualy UNIX systems have a different non-revertable hashing algorithm for passwords, therefore it is not possible to use the traditional UNIX password database if you want to use any CHAP algorithm, with the exception that FreeBSD versions 5.1

and later support the NT-Hash format in the password database (configurable via login.conf: `passwd_format=nth`). However MPD doesen't currently support authentication against the UNIX password database.

Note: in order for MPPE to work, MS-CHAPv1 or MS-CHAPv2 is mandatory, because the MPPE keys are generated using the results from authentication.

Authentication protocol negotiation

Starting with MPD 3.14, MPD behaves more intelligently when negotiating authentication protocols. MPD searches an internal list of protocols, from most to least secure, until a mutually agreeable protocol is found. If the link is a PPTP link, then MS-CHAP is most preferrable, otherwise MD5-CHAP is most preferrable.

9. References

9.1 References

The PPP protocol is documented in a slew of RFC's. Some of these are listed below.

RFC 1332 The PPP Internet Protocol Control Protocol (IPCP)

RFC 1334 PPP Authentication Protocols

RFC 1570 PPP LCP Extensions

RFC 1661 The Point-to-Point Protocol (PPP)

RFC 1662 PPP in HDLC-like Framing

RFC 1877 PPP Internet Protocol Control Protocol Extensions for Name Server Addresses

RFC 1962 The PPP Compression Control Protocol (CCP)

RFC 1968 The PPP Encryption Control Protocol (ECP)

RFC 1969 The PPP DES Encryption Protocol (DESE)

RFC 1973 PPP in Frame Relay

RFC 1974 PPP Stac LZS Compression Protocol

RFC 1978 PPP Predictor Compression Protocol

RFC 1979 PPP Deflate Protocol

RFC 1990 The PPP Multilink Protocol (MP)

RFC 1994 PPP Challenge Handshake Authentication Protocol (CHAP)

RFC 2427 Multiprotocol Interconnect over Frame Relay

RFC 2433 Microsoft PPP CHAP Extensions

RFC 2865 Remote Authentication Dial In User Service (RADIUS)

RFC 2866 RADIUS Accounting

RFC 2548 Microsoft Vendor-specific RADIUS Attributes

CONTENTS

1. Introduction 2
   1.1 Overview 2
   1.2 Mpd Layers 3
   1.3 Organization of this manual 3
   1.4 Change history 4

2. Installation 9
   2.1 Getting mpd 9
   2.2 Installing mpd 9
   2.3 Building mpd 9

3. Running Mpd 11
   3.1 Invoking mpd 11
   3.2 Configuration file format 12
      3.2.1 General properties 12
      3.2.2 mpd.conf and mpd.links 12
      3.2.3 mpd.secret 13
      3.2.4 mpd.script 14
   3.3 Mpd command line interface 14

4. Configuring Mpd 16
   4.1 General mpd commands 16
   4.2 Bundle layer commands 20
   4.3 Interface layer commands 23
   4.4 IPCP layer commands 26
   4.5 Link layer commands 28
   4.6 Compression layer commands 31
   4.7 Encryption layer commands 32

5. Device Types 34
   5.1 Modem type commands 34
   5.2 Netgraph device type commands 34
   5.3 TCP device type commands 35
   5.4 UDP device type commands 35
   5.5 PPTP device type commands 36
      5.5.1 PPTP setup hints 37
   5.6 PPPoE device type commands 38

6. Chat Scripting 40
   6.1 Script file format and execution 40
   6.2 Events 40
      6.2.1 Timer events 40
      6.2.2 Match events 40
   6.3 Sets 41
      6.3.1 Special sets and targets 41
   6.4 Variables 41
      6.4.1 Special variables 41
   6.5 Script commands 42
   6.6 Errors 44
   6.7 The included mpd.script 44

7. Troubleshooting 47

7.1 Troubleshooting 47

8. Internals 49
8.1 ToDo 49
8.2 Authentication 49

9. References 51
9.1 References 51