Mpd 4.3 User Manual

Archie Cobbs, Michael Bretterklieber, Alexander Motin

`<mpd-users@lists.sourceforge.net>`

Á×ÇÕÓÔÁ 29, 2007

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 **4.3**. 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 4.x 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, MS-CHAP and EAP authentication
- PPP compression and encryption
- IPCP and IPV6CP parameter negotiation

Mpd have support for many link types:

- **Serial port 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.
- **Point-to-Point Tunnelling Protocol (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.
- **Layer Two Tunnelling Protocol (L2TP)** Mpd can connect over the Internet using the Layer Two Tunnelling Protocol (L2TP). This protocol is supported by the popular Microsoft Dial-Up Networking client.
- **PPP over Ethernet (PPPoE)** Mpd can connect over an Ethernet port using the PPP-over-Ethernet (PPPoE) protocol.
- **PPP over TCP** Mpd can tunnel PPP frames over a TCP connection. Frames are encoded in the same was as asychronous serial connections.
- **PPP over UDP** Mpd can tunnel PPP frames over a UDP connection. Each frame is encapsulated in a UDP datagram packet.
- **PPP over specified netgraph(4) node** 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.

Mpd also includes many additional features:

- IPv4 and IPv6 support
- RADIUS authentication and accounting
- NetFlow traffic accounting
- Network address translation (NAT)
- 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 several common modems and ISDN TAs
- Clean device-type independent design
- Comprehensive logging
- Telnet and HTTP control interfaces.

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.

1.2 Mpd Layers

Mpd consists of several **layers**, corresponding to the layered nature of the PPP protocol. There are two types of layers hierarchy connected with two different operation modes as "ppp terminator" and "ppp repeater".

In "ppp terminator" mode such hierarchy used: *Interface* **(section 4.7 , page 28)** -> *NCPs* **(section 4.8 , page 30)** -> *Compression* **(section 4.11 , page 35)** -> *Encryption* **(section 4.12 , page 37)** -> *Bundle* **(section 4.5 , page 25)** -> *Links* **(section 4.9 , page 31)** -> *Physical devices* **(section 4.10 , page 35)**.

In "ppp repeater" mode such hierarchy used: *Repeater* **(section 4.6 , page 27)** -> *Physical devices* **(section 4.10 , page 35)**.

A *link* **(section 4.9 , page 31)** is a single point-to-point connection between the local machine and a remote peer machine, implemented by some kind of *physical device* **(section 4.10 , page 35)**, such as a serial modem connection or a virtual PPTP connection.

A *bundle* **(section 4.5 , page 25)** 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.

At the bundle layer you configure multi-link PPP settings 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 4.7 , page 28)** 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 several corresponding NCP layers as *IP Control Protocol (IPCP)* **(section 4.8 , page 30)** and **IPv6 Control Protocol (IPv6CP)** layers, which manages the IP specific configuration of the interface. This layers handle the negotiation of local and remote addresses and 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 4.11 , page 35)** and *encryption* **(section**

4.12 , page 37) 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 4.9 , page 31)** layer. Link 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 *physical devices* **(section 4.10 , page 35)** layer. Configuration of device type specific parameters happens at this layer. Each device has a specific *type* corresponding to one of the supported device types in mpd. The type dictates how the device dependent part is configured and what it's capabilities are.

A *repeater* **(section 4.6 , page 27)** is a collection of two physical devices. It repeats all connections coming from one physical device to another without any modification. This technology is also known as LAC (L2TP Access Concentrator) or PAC (PPTP Access Concentrator).

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 4.2.2:

- New features:
    - Added L2TP local hostname configuration.
    - Added L2TP length and dataseq options.
    - L2TP local hostname and secret at server side is now configurable depending on client address.
    - Reimplemented RADIUS Class attribute support.
    - Added PPPoE AC-name specification for the server side.
    - Added IP accounting with ng_ipacct node support.
    - Added configure script for better system features detection.
    - 'show version' command now shows compiled-in system features.
    - 'session ...' and 'msession ...' commands to select link/bundle by their current session IDs added.
- Bugfixes:

    - Fixed race condition on PPTP tunnel creation/death.
    - Fixed crash when stdout redirected to /dev/null.
    - Fixed memory leak in proxy-arp.
    - Fixed Dial-on-Demand functionality broken in 4.2.
    - Do not set ACCM for a Sync links.
    - Fixed Sync mode detection for L2TP links.
- Performance improvements:
    - Added support for 64bit ng_ppp counters where available.

Changes since version 4.2.1:

- Bugfixes:
    - Fixed build and stack overflow on FreeBSD 5.x.
    - Fixed startup script dependencies.

Changes since version 4.2:

- Bugfixes:
    - Fixed default route support bug.
    - Fixed memory leak in L2TP link creation.

Changes since version 4.1:

- New features:
    - Implemented link repeater functionality (aka LAC/PAC). New "phys" and "repeater" layers added.
    - PPTP now supports listening on multiple different IPs.
    - L2TP now supports tunnel authentication with shared secret.
    - Implemented traffic filtering using ng_bpf.
    - Implemented fast traffic shaping/rate-limiting using ng_car.
    - Added workaround for Windows 2000 PPPoE MRU negotiation bug.
    - Implemented minimal client side of auth-driven callback (w/o number specification).
    - Restored control console on stdin.
    - Added multiline console command history.
    - Added new 'ext-auth' auth backend as full-featured alternative to 'radius-auth'.
    - Added support for some new ng_nat features.
    - Implemented PPTP/L2TP SetLinkInfo sending to PAC/LAC.
    - NetFlow generation for both incoming and outgoing packets same time is now supported. NOTE: To have more then 1000 interfaces with NetFlow in 6-STABLE you may need to increase NG_NETFLOW_MAXIFACES constant in netflow.h and rebuild ng_netflow kernel module.
    - Added mpd-drop-user vendor specific accounting reply attribute support.
- Changes:

- 'set link type ...' command is deprecated now. Use 'set phys type ...' instead.
- -a, -n, -N, and -t bundle options are deprecated now. Use 'set iface enable ...' instead.
- ng_tee, ng_nat, ng_netflow and other netgraph nodes between ng_ppp anf ng_iface now created when NCP (IPCP/IPV6CP) goes up instead of startup time.
- Auth subsystem refactored to avoid incorrect cross-level dependencies.
- Physical device level refactored to remove link and bundle levels dependencies.
- While accepting calls PPTP, L2TP, TCP and UDP links are now trying to use link with most specific peer address configured.
- Removed setting up local IPv4 address routing to loopback. /usr/sbin/ppp does not doing it.

- Bugfixes:
  - Fixed thread-safety related crash in accounting.
  - Fixed assertion in PPTP on control connection fail while answering.
  - Fixed assertion in L2TP on control message sending failure.
  - Fixed broken L2TP outcall mode.
  - Updated chat scripts to detect incoming modem calls speed.
- Performance improvements:
  - Calls to ifconfig and route programs replaced by internal functions.
  - Where possible system() calls replaced by fork()+execv() to avoid shell execution.
  - Added connect requests storm overload protection. Mpd will drop incoming requests when message queue reach some defined length.

Changes since version 4.1rc2:

- Changes:
  - Default value of link's max-redial parameter changed to -1.
  - Bundle's noretry option is enabled by default now.
- Bugfixes:
  - Better up/down reason tracking.

Mpd version was bumped from 4.0rc2 to 4.1rc2 due to large number of changes done since 4.0b4 and FreeBSD ports version number conflict.

Changes since version 4.0rc1:

- Bugfixes:
  - Idle timeout fixed.
  - Fixed bug with 'set l2tp self ' specified at the server side.
  - Device type check for device-specific commands added.
  - IPCP reject is not fatal by itself now.
  - Up/down-script will now be called not for the whole interface, but for each of negotiated protocols. Proto parameter should be checked in the script!

- Fixed ng_ppp link bandwidth configuration.

Changes since version 4.0b5:

- New features:
  - Integrated Web server added.
  - NAT support by ng_nat(4) added.
  - L2TP (RFC 2661) device type implemented.
  - UDP device type was completely rewritten. Now it:
    - does not require manual 'open' command on the server side, it behaves just like any other device type;
    - allows many connections to the same server UDP port;
    - allows not to specify peer address/port for incoming connections (so it will work through different NATs and firewalls);
    - allows not to specify self address/port for outgoing connections (so it is easier to configure);
  - TCP device type was completely rewritten. It has some minor issues due to limitation of ng_ksocket module, but now IT WORKS! :)
  - Compression Predictor-1 (RFC 1978) added.
  - Compression Deflate (RFC 1979) added.
  - Encryption DESE (RFC 1969) support was reimplemented.
  - Encryption DESE-bis (RFC 2419) support added.
  - New command 'show phys' added.
  - New command 'show summary' added.
  - Support for ipfw tables added to RADIUS ACL's.
  - New commands 'set global start...' added..
  - Added support of calling/called numbers (mostly for PPTP/L2TP).
- Changes:
  - "lcp" layer in open/close commands replaced by "link".
  - Auth configuration (set auth ...) moved from bundle layer to lcp. It works per link now.
  - MPPE policy option moved from auth layer to ccp.
- Bugfixes:
  - Fixed a few bugs on amd64 and sparc64 platforms.
  - Phys layer was made stateless to remove race condition.
  - Link layer changed to remove race conditions on LinkDown().
  - Fixed race condition in accepting PPPoE connections.
  - Link up/down reason recording is now more accurate.
  - Complete link shutdown procedure on auth failure implemented.

    - Fixed several small PPTP level processing issues.
    - Removed limitation about PPTP which must be in the bundle alone.
    - Fixed MSCHAP auth which was broken in 4.0b5.
    - Fixed memory leak in PAP and CHAP auth on the client side.
    - Fixed some CCP negotiation issues.
    - Fixed threads-related crash in internal auth.
    - Fixed crash on incoming when no free PPTP link found.
    - Bug in "rubber bandwidth" algorithm fixed.
    - Bug and possible crash fixed in DoD code.
    - Fixed bug in AUTHPROTO negotiation.
    - Fixed bug in RAD_MICROSOFT_MS_CHAP2_SUCCESS handeling. Needs testing.

Changes since version 4.0b4:

- New features:
    - IPv6 support:
        - IPV6CP support added, NCPs and IFACE calls was rewritten to support many NCPs.
        - Console now supports IPv6.
        - UDP and TCP link types now support IPv6.
        - PPTP link type is ready to support IPv6, but requires ng_pptpgre(4) to support IPv6.
        - NetFlow export over IPv6 is supported.
        - The following features don't yet support IPv6: TcpMSSFix, NetFlow, Tee, DialOnDemand.
    - TCP link type now compiles and works (but isn't yet ready for production usage).
    - NetFlow data generation on outgoing interface is supported.
    - Added a possibility to use an existing ng_netflow(4) node.
    - Added a possibility to specify network interface names instead of IP addresses.
    - Added more log levels to decrease log file size.
- Changes:
    - Default argument of open/close commands changed from iface to lcp.
- Bugfixes:
    - Fixed races between startup process and client connecting.
    - Fixed a few crashes in console.
    - Incoming call processing significantly reworked to fix some aspects of multilink server functionality.
    - The shutdown of mpd is now much more graceful: the netgraph nodes are closed, the accounting RADIUS packets for closing links are sent, new connections aren't accepted during shutdown.

  - Fixed races in filling of RADIUS packets. In particular, RAD_NAS_PORT value in the RADIUS could be wrong.
  - RADIUS support rewritten to use poll(2) instead of select(2), allowing to create a bigger number of links.
  - Fixed a problem with identifying correct interface for proxy-arp when alias addresses are used.
  - Fixed memory leaks and crashes when more than 256 PPTP bundles are in use.
  - Fixed crash in PPPoE when more than 64 parent Ethernet interfaces used.
- Performance improvements:
  - Message and PPPoE subsystems reworked to decrease number of open files per bundle.

Changes since version 4.0b3:

- BugFix: fix crash in processing of MS domain name from RADIUS server.
- New feature: automatic creation, configuring and attaching of ng_netflow(4) node.
- ng_tee(4) now can be inserted on a per bundle basis.
- New feature: on FreeBSD 6.0 and higher ng_tcpmss(4) is utilized if doing TCP MSS fixup.
- BugFix: tcpmssfix now works for both incoming and outgoing TCP segments.
- New options: update-limit-in, update-limit-out.
- Fixed loss of statistics when -t options is used.
- Fixed chat scripting, modem links not broken anymore.

Changes since version 4.0b2:

- BugFix: make PPPoE interface control events recurring, PPPoE is not broken anymore.
- Added a new `startup` section to the config-file, wich is loaded once at startup.
- Added a new `global` config space for all the global settings.
- BugFix: don't generate new challenges, while retransmitting them.
- Fix `va_args` bug on certain non-i386 platforms.
- Auto-load `ng_ether` for PPPoE connections; fix default path for undefined service.
- Rewrite the console-stuff. Multiple telnet connections are now allowed. There is no input-console anymore, must use telnet instead.
- BugFix: The directly configured password wasn't taken into account when using PAP.
- Disallow empty usernames safely.

Changes since version 4.0b1:

- Fixed a race-condition wich caused a dead-lock.
- RADIUS
  - Fixed several race-conditions when sending accounting requests.
  - Use the username from the access-accept packet (if present) for accounting requests.

Changes since version 3 (most of this work was sponsored by SURFnet SURFnet):

- Design changes: Mpd uses now a thread-based event system using libpdel, these libpdel parts are now integrated:
  - typed_mem(3)
  - pevent(3)
  - alog(3)

  Mpd uses a "Giant Mutex" for protecting its resources.
- Major new features:
  - Implemented the Extensible Authentication Protocol RFC 2284 (EAP). Currently only EAP-MD5 is supported (client and server side). EAP negotiaton can be enabled at link level.
  - Implemented OPIE (One-time Passwords In Everything).
  - Implemented authentication against systems password database `master.passwd`.
  - utmp/wtmp logging.
- Rewrites of the authentication subsystem:
  - Make authentication and accounting requests asynchronous using paction(3).
  - Authentication backends are acting now independently from the rest of Mpd, using some internal structs as interface.
  - The `mpd.secrets` file is now used as one authentication backends of many, it has no special role anymore, i.e. it could be disabled.
  - Generate a session-id at bundle and link level for using with accounting requests.
- RADIUS related changes:
  - **IMPORTANT**: Mpd needs now an enhanced libradius, here are the patchsets: `4-STABLE` `5-CURRENT`
  - Remember and send the RAD_STATE attribute.
  - Message-Authenticator support.
  - EAP Proxy Support.
- 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
  ```
- Added a new commandline option `-t` for adding ng_tee into the netgraph.

  Submitted by: Gleb Smirnoff, glebius at cell dot sick dot ru
- Removed configuration parameters:
  - bundle: `radius-fallback`
  - iface: `radius-session`, `radius-idle`, `radius-mtu`, `radius-route`, `radius-acl`

  - ipcp: `radius-ip`

  Moved configuration parameters:

  - bundle to auth: `radius-auth`, `radius-acct`, `authname`, `password`, `max-logins`
  - radius to auth: `acct-update`
  - ccp to auth: `radius` and renamed to `mppc-pol`

  New configuration parameters:

  - link: `keep-ms-domain`, this prevents Mpd from stripping the MS-Domain, this is can be useful when using IAS as RADIUS server.
  - radius: `message-authentic`, this adds the Message-Authenticator attribute to the RADIUS request.
  - auth: `internal`, controles the usage of the `mpd.secrets` file (internal authentication backend).
  - auth: `opie`, enables/disables the OPIE authentication backend.
  - auth: `system`, enables/disables authentication against systems password database.
  - auth: `utmp-wtmp`, enables/disables utmp/wtmp logging. database.
  - auth: `timeout`, configureable timeout for the authentication phase.
  - eap: `radius-proxy`, this causes Mpd to proxy all EAP requests to the RADIUS server, Mpd only makes the initial Identity-Request (this saves one round-trip), every other requests are forwarded to the RADIUS server. This adds the possibility supporting every EAP-Type of the RADIUS server, without implementing each EAP-Type into Mpd.
  - eap: `md5`, EAP-Type MD5, it's the same as CHAP-MD5, but inside EAP frames.

- Removed defines `ENCRYPTION_MPPE` and `COMPRESSION_MPPC`, they are now built in.
- Get rid of `IA_CUSTOM` define.
- BugFix: Fixed a mem-leak in the pptp-ctrl stuff.

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/mpd4`. 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/mpd4` 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`

`PHYSTYPE_L2TP`

`PHYSTYPE_PPPOE`

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

`COMPRESSION_PRED1`

`COMPRESSION_DEFLATE`

: These enable support for the corresponding compression types.

`ENCRYPTION_DES`

: These enable support for the corresponding encryption types.

`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.

`-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 first link. 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 first link. If the link 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 auth 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/cuad2

# my external modem which is on COM2
modem2:
        set link type modem
        set modem device /dev/cuad1
```

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.

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.

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 50) for a complete decscription of chat scripting.

3.3 Mpd command line interface

Mpd provides to user a command line interface on stdin and TCP port. 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 daemon for FreeBSD

process 26177 started, version 4.2 (root@orphanage.alkar.net 15:37 19-Jun-2007)
[myisp] ppp node is "mpd771-myisp"
[myisp] using interface ng0
```

By that time, 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. In this case `mpd771-myisp` is the netgraph node and `ng0` is the interface.

To control mpd runtime you can connect to mpd's Telnet console.

```
> telnet 0 5005
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
Multi-link PPP daemon for FreeBSD

Username: Admin
Password:
Mpd pid 1419, version 4.2 (root@orphanage.alkar.net 15:37 19-Jun-2007)

[modem1]
```

The prompt `[modem1]` shows the currently active 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:

```
[modem1] link modem2
[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:

```
[modem2] help show
Commands available under "show":
 bundle   : Bundle status              repeater : Repeater status
 ccp      : CCP status                 ecp      : ECP status
 eap      : EAP status                 events   : Current events
 ipcp     : IPCP status                ipv6cp   : IPV6CP status
 iface    : Interface status           routes   : IP routing table
 layers   : Layers to open/close       phys     : Physical device status
 link     : Link status                auth     : Auth status
 radius   : RADIUS status              lcp      : LCP status
 nat      : NAT status                 mem      : Memory map
 mp       : Multi-link status          console  : Console status
 web      : Web status                 global   : Global settings
 types    : Supported device types     version  : Version string
 summary  : Daemon status summary
[modem2]
```

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

```
[modem2] b
Defined bundles:
        Bundle         Links
        ------         -----
        myisp          modem2[DOWN]
[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 of 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* | *[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.

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

`phys [` *name*]
: This command changes the currenly active phys to the phys named *name*. If this phys does not belong to the currently active bundle/repeater, then the currently active bundle/repeater is changed to the bundle/repeater that owns phys *name*. If no *name* is given, this command lists all of the defined physes and their bundles/repeaters. The new phys will be reflected in the prompt.

`msession [` *msesid*]
: This command changes the currenly active bundle to the bundle holding session with multilink session ID *msesid*.

`session [` *sesid*]
: This command changes the currenly active link to the link holding session with session ID *sesid*.

`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.

  repeater
  : Show status information about the currently active repeater.

phys
: Show status information about the currently active physical device.

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.

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

nat
: Show information about NAT on the currently active bundle.

auth
: Show status information about Authentication.

eap
: Show status information about EAP.

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.

summary
: Show status summary

console
: Show console summary

web
: Show web server summary

`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 nat ...`
: Set NAT 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 auth ...`
: Configures the authentication subsystem.

`set radius ...`
: Configures RADIUS.

`set eap ...`
: Configures the EAP.

`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 l2tp ...`
: Sets device specific configuration parameters for the currently active link, which must have type **l2tp**.

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

`set console ...`
: Sets console specific configuration parameters

`set web ...`
: Sets web server specific configuration parameters

`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. If dial-on-demand is disabled there no reason to open this layer manually.

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

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

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

link
: Link control protocol FSM. This is default layer for open/close commands. Opening of this layer initiates outgoing call by this link.

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

Normally, only the `iface` and `link` layers 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 first link.

`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 Global commands

This chapter describes options that affect Mpd's behavior globally.

`set global startrule` *num*

set global startpipe *num*

set global startqueue *num*

set global starttable *num*" These commands define initial values for autonumeration of ipfw rules, pipes, queues and tables. Defaults are 10000, 10000, 10000 and 32 accordingly.

`set global enable` *option ...*

set global disable *option ...*" These commands configure various global options.

The **enable** and **disable** commands determine whether or not we want the corresponding option.

The options available are:

`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.

4.3 Console commands

This chapter describes commands for configuring Mpd's console. The console can be accessed via IPv4 or IPv6 using telnet. Multiple telnet connections are allowed.

`set console open`
: Opens the console, i.e., creates the listening TCP socket.

`set console close`
: Closes the console, i.e., closes the listening TCP socket. Only the listener is affected; active sessions are not closed.

`set console port` *port*

set console ip *ip*" Sets the credentials for the console-listener. After changing one of these options, the console must be closed and re-opened for the changes to take effect.

`set console user` *username password*
: This configures which users are allowed to connect to the console. This command may be invoked multiple times with different usernames.

`set console enable` *option ...*

set console disable *option ...*" These commands configure various console options.

The options available for the console are:

`logging`
: This options enables writing of logging messages to the console.

4.4 Web server commands

This chapter describes commands for configuring Mpd's web server.

`set web open`
: Opens the web server, i.e., creates the listening TCP socket.

`set web close`
: Closes the web server, i.e., closes the listening TCP socket.

`set web port` *port*

set web ip *ip*" Sets the credentials for the web-listener. After changing one of these options, the web must be closed and re-opened for the changes to take effect.

`set web user` *username password*
: This configures which users are allowed to connect to the web server. This command may be invoked multiple times with different usernames.

`set web enable` *option ...*

set web disable *option ...*" These commands configure various web server options.

The **enable** and **disable** commands determine whether we want the corresponding option.

The options available for the web are:

`auth`
: This option enables basic authorisation on web server. Default is enabled.

4.5 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 physival device associated with link (via the `set phys 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 disconnect a link unless previous one was disconnected at least *min-dis* seconds before. Similarly, mpd will not open a link unless previous was opened at least *min-con* seconds before. These two parameters help prevent oscillations, but cause slower response time.

`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).

`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.

`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`.

`ipcp`
: Enable/Disable IPCP negotiation. Default is enabled.

`ipv6cp`
: Enable/Disable IPV6CP negotiation. Default is disabled.

`compression`
: Enable/Disable CCP (Compression Control Protocol) negotiation. Default is disabled.

`encryption`
: Enable/Disable ECP (Encryption Control Protocol) negotiation. Default is disabled.

`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.''

  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.

  The default for this option is on.

4.6 Repeater layer commands

This chapter describes commands that configure the repeater layer. All of these commands (except `rnew`) apply to the currently active repeater.

Repeater repeats all connections, coming from one physical device, to another without any processing. This technology is also known as LAC (L2TP Access Concentrator) and PAC (PPTP Access Concentrator).

For correct operation one device should be configured for accepting incoming connections while another for making outgoing connections.

`rnew` *name phys1 phys2*
: This command creates a new repeater named *name*.

  Two physical devices that constitute the repeater are created and named *phys1* and *phys2*. For each physical device, 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 physical device (via the `set phys type ...` command).

4.7 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 4.8 , page 30) 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 incoming or outgoing packets are transmitted for *seconds* seconds, the connection is brought down. An idle timeout of zero disables this feature.

  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 one of network protocols at the interface is brought up or down. The **up-script** is called like this:

script interface proto local-ip remote-ip authname [dns1 *server-ip*] [dns2 *server-ip*]

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

script interface proto 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 ranges 192.168.1.10/32 192.168.1.20/32
```

`tcpmssfix`
: This option causes mpd to adjust incoming and outgoing TCP SYN segments so that the requested maximum 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.

`tee`
: If enabled, a ng_tee(4) node will be inserted into the bundle graph, right after the interface node. The tee node can be useful for debugging or traffic accounting.

`nat`
: If enabled, a ng_nat(4) node will be inserted into the bundle graph. The ng_nat(4) node performs network address translation (NAT) of packets passing through it. ng_nat(4) node type exists since FreeBSD 6.0.

`netflow-in`

`netflow-out`
: If enabled, the bundle will participate in NetFlow generation. See *the NetFlow chapter* (section 4.16 , page 40) for more information. The netflow-out option is like netflow-in, but generates NetFlow data for outgoing packets instead of incoming.

`ipacct`
: If enabled, the bundle will participate in IP accounting with ng_ipacct node.

4.8 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. This compression ineffective for TCP connections with enabled modern extensions like time stamping or SACK, which modify TCP options between sequential packets.

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.

4.9 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, which physical device shown at the command line prompt.

`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 debugging, 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.

  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.

  If `max-redial` is set to 0, then mpd will redial infinitely.

  The default value is -1.

`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**.

`eap`

Extensible Authentication Protocol.

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 request. If the remote peer can/wants to, it will hangup immediately after connecting and call us back.

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**.

`keep-ms-domain`

Normally, if using MS-CHAP, the MS-Domain is stripped and only the plain username is used. Under certain circumstances the MS-Domain should be kept, for instance if IAS is used as RADIUS server.

Default **disable**.

4.10 Phys layer commands

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

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

`set phys type` *type*

This command tells mpd what the type for this physical device is. This should generally be the first command in the `mpd.links` entry for each physical device.

4.11 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.

To use CCP you should enable compression at the bundle level. To allow compression to be negotiated you should enable here some of compression protocols.

`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.

`pred1`
: This option enables Predictor-1 (RFC 1978) compression. Predictor-1 compression is much faster then Deflate.

  This option require ng_pred1 Netgraph node type, which is present since FreeBSD 6.2-STABLE of 2007-01-28. If there no ng_pred1 in system, this algorithm is supported in user-level, but will consume more CPU power. To reduce packet loss in user-level on fast links it is recommended to increase values of net.graph.recvspace and net.graph.maxdgram sysctl variables.

`deflate`
: This option enables Deflate (RFC 1979) compression. Deflate compression usually gives better compression ratio then Predictor-1.

  This option require ng_deflate Netgraph node type, which is present since FreeBSD 6.2-STABLE of 2007-01-28.

`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.

  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.

`mpp-compress`
: Enables MPPC compression. This is the only compression method supported by Microsoft Windows RAS.

  This option requires compression support from ng_mppc node which is usually disabled as it require some Hi/Fn proprietary files.

`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.

`mppe-policy`

If enabled, Mpd uses the MPPE-Types and MPPE-Policy info from the authentication backend.

Default **disable**.

4.12 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.

To use ECP you should enable encryption at the bundle level. To allow encryption to be negotiated you should enable here some of encryption protocols.

Note: In PPP, encryption should be handled by the ECP rather than CCP. However, Microsoft combined both compression and encryption into their ''compression'' algorithm (MPPC), which is confusing.

`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:

`dese-bis`

This option enables DESE-bis (rfc 2419) encryption. This algorithm implemented in user-level, so require much CPU power on fast (>10Mbit/s) links.

`dese-old`

This option enables DESE (rfc 1969) encryption. This algorithm implemented in user-level, so require much CPU power on fast (>10Mbit/s) links.

Note: DESE protocol is deprecated. Because of data padding to the next 8 octets boundary, required by block nature of DES encryption, dese-old option can have interoperability issues with other protocols which work over it. As example, it is incompatible with Predictor-1 and Deflate compressions.

4.13 Auth commands

This chapter describes commands that configure the Authentication subsystem of LCP layer. All of these commands apply to the currently active link. Mpd currently supports authentication against (tried in this order) RADIUS, systems password database (`master.passwd`), OPIE and its integrated `mpd.secrets` file.

`set auth authname` *login*
: This command sets the authentication login name associated with the link (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 auth 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 auth max-logins` *num*
: Limit the max. amount of concurrent logins with the same username. If set to zero, then this feature is disabled.

`set auth acct-update` *seconds*
: Enables periodic accounting updates, if set to a value greater then zero.

`set auth timeout` *seconds*
: Sets the timeout for the whole authentication process. It defaults to 40 seconds. Under some circumstances the value should be changed; it usually depends on the authentication backend and protocol. E.g. when using EAP with a slow RADIUS server this value should be increased.

`set auth enable` *option ...*

set auth disable *option ...*"

The options available are:

`internal`
: Enables authentication against the `mpd.secrets` file.

  Default **enable**.

`radius-auth`
: Enable authentication via RADIUS.

  Default **disable**.

`radius-acct`
: Enable per link accounting via RADIUS.

  Default **disable**.

`ext-auth`
: Enable authentication by calling external authentination script. This method pretended to be a fullfeatured alternative to the **radius-auth**.

  Path to that script must be specified using **set auth extauth-script ...** command.

  Default **disable**.

`system`

Enables authentication against the systems password database. This options can only be used with PAP and MS-CHAP, but not with CHAP-MD5. If you intend to use this with MS-CHAP, then the passwords in the `master.passwd` must be NT-Hashes. You can enable this by putting `:passwd_format=nth:` into your `/etc/login.conf`, but you need at least FreeBSD 5.2.

Default **disable**.

`opie`

Enables authentication using OPIE. When using PAP there is nothing more todo. For all other authentication protocols you have to put the username into the `mpd.secrets` file, but the specified password is then interpreted as secret pass phrase. This is needed, because Mpd must be aware of the plaintext password when using CHAP. The (windows) endusers could generate their actual responses themselfs using Winkey.

IMPORTANT: Disable the internal authentication when using OPIE and CHAP, because otherwise users are also able to authenticate with their secret pass phrase.

Default **disable**.

`utmp-wtmp`

Enable accounting via utmp/wtmp.

Default **disable**.

4.14 EAP commands

This chapter describes commands that configure the EAP (Extensible Authentication Protocol). Mpd supports natively only the EAP-Type MD5; other EAP-Types may be used in conjunction with a RADIUS server. All of these commands apply to the currently active link.

`set eap accept` *option ...*

`set eap deny` *option ...*

`set eap enable` *option ...*

`set eap disable` *option ...*

`set eap yes` *option ...*

`set eap no` *option ...*

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

`md5`

EAP-Type MD5. It's the same as CHAP-MD5, except that is framed inside EAP packets.

`radius-proxy`

Causes Mpd to proxy all EAP requests to the RADIUS server.

4.15 RADIUS commands

This chapter describes commands that configure the RADIUS authentication backend. All of these commands apply to the currently active bundle.

`set radius server` *name secret* [*auth-port* [*acct-port*]]
: Configure RADIUS server parameters. Multiple RADIUS servers may be configured by repeating this command, and up to 10 servers may be specified.

`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 me` *IP* | *ifname* | *hostname*
: Send the given IP in the RAD_NAS_IP_ADDRESS attribute to the server

`set radius v6me` *IPv6*
: Send the given IP in the RAD_NAS_IPV6_ADDRESS attribute to the server

`set radius enable message-authentic`
: Adds the Message-Authenticator attribute to the RADIUS request. The Message-Authenticator is an HMAC-MD5 checksum of the entire Access-Request packet using the shared secret as the key. This should protect the RADIUS server against online dictionary attacks. This is mandatory when using the EAP-RADIUS-Proxy and it's implicitly added to the request by Mpd.

`set radius enable peer-as-calling`
: Forces mpd to send remote tunnel address in Calling-Station-Id instead of addres supplied by remote peer via tunnel (for PPTP and L2TP). Can be disabled for trusted peers. Default is enabled.

4.16 NetFlow commands

This chapter describes commands for configuring NetFlow data export on Mpd's interfaces. NetFlow is generated by `ng_netflow(4)` node. To enable NetFlow on a particular interface `netflow-in` or `netflow-out` options should be used.

Note: This configuration is global for the daemon, so it must be done before first bundle get connected.

Note: This feature is available only on FreeBSD 6.0-RELEASE and newer.

`set netflow export` *ip port*
: Set destination IP address and port, where NetFlow export datagrams will be sent. This options is important to have working NetFlow.

`set netflow source` *ip port*
: Set optional local IP and port binding, to make NetFlow datagrams originate from a specific source.

`set netflow timeouts` *inactive active*
: Modify inactive and active timeout values. For more information, see `ng_netflow(4)` manual page.

`set netflow node` *nodename*
: Use existing `ng_netflow(4)` node instead of creating a new one.

`set netflow hook` *number*
: Use `ng_netflow(4)` node's hooks starting from specified number. For detailed information about naming of hooks, see `ng_netflow(4)` manual page.

4.17 NAT commands

This chapter describes commands for configuring Network Address Translation on Mpd's interfaces. NAT is implemented by `ng_nat(4)` node. To enable it on a particular interface `nat` option should be used.

Note: ng_nat node is available only on FreeBSD 6.0-RELEASE and newer. Some additional options like setting target address implemeted in later versions.

`set nat address` *ip*
: Set alias IP address. If not specified, interface IP will be used.

`set nat target` *ip*
: Set target IP address for unknown incoming packets. If not specified, packet will not be modified.

`set nat enable` *option ...*

`set nat disable` *option ...*

: Enable and disable nat options.

The following options are supported:

`log`
: Enable statistics logging.

  Default is disabled.

`incoming`
: Allow unknown incoming packets to pass.

  Default is enabled.

`same-ports`
: Try to use same ports while aliasing.

  Default is enabled.

`unreg-only`
: Enables aliasing only for unregistered networks.

  Default is disabled.

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/cuad0`.

`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 50) 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

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**.

Due to limitations of ng_ksocket module this device type have two issues:

- when there are no free incoming link, new connection will anyway be acceped and then dropped instead of not accepting it,
- tcp connection status is not monitored, so you should use 'set link keep-alive' to detect broken links.

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

Sets the local IP address and port for the TCP connection. Port field is required for accepting incoming connections.

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

Sets the peer IP address and port for the TCP connection. Address and port fields are required for outgoing connections. For incoming connections it is not required, but can limit who can connect to us.

`set tcp enable` *option ...*

`set tcp disable` *option ...*

Enable and disable TCP device type options for the link.

The following options are supported:

`originate`

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

`incoming`

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

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 if you want to get something better channel utilization.

Note: Technically this device type is not completely correct, 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. Port specification is required for incoming connection accepting.

`set udp peer` *ipaddr* [*port*]
: Sets the peer IP address and port for the UDP connection. Address and port specification is required for connection originating. For incoming connections it is not required, but can limit who can connect to us.

`set udp enable` *option ...*

`set udp disable` *option ...*

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

The following options are supported:

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

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

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.

Complete PPTP network topology looks like:

```
client <- some link type -> PAC <- PPTP tunnel -> PNS
```

PAC is physical level repeater, which receives PPP connection of some type and forwards it to PNS using PPTP protocol. PNS is ppp endpoint, which receives ppp frames via PPTP tunnel and processes them.

In simple case, when physical conversion is not required, topology can be simplified to:

```
client (PAC emulator) <- PPTP tunnel -> PNS
```

Mpd is able to operate in both PAC and PNS modes. As PAC mpd supports both simple case PAC emulator and complete PAC topologies. Complete PAC can be configured by joining two physical devices using mpd's repeater functionality.

`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 callingnum` *number*

`set pptp callednum` *number*
: Sets the calling and 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 outgoing telephone call (by PNS) via a remote access server (PAC), as well as the more common use of forwarding of an incoming telephone call from an access server (PAC) to a remote PPTP server (PNS).

: When this option is enabled, mpd will initiate outgoing calls (PNS); otherwise mpd will initiate incoming calls (PAC). 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 auth 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 L2TP device type commands

This chapter describes commands that are specific to L2TP type links. These commands apply to the currently active link, and are only valid if the currently active link has type **l2tp**.

The L2TP protocol utilizes UDP datagrams on port 1701 (and this is the default for *port* in the commands below) to create and maintain virtual tunnel between IP peers. One or more independent PPP connections (sessions) can be carried inside this tunnel.

Complete L2TP network topology looks like:

```
client <- some link type -> LAC <- L2TP tunnel -> LNS
```

LAC is physical level repeater, which receives PPP connection of some type and forwards it to LNS using L2TP protocol. LNS is ppp endpoint, which receives ppp frames via L2TP tunnel and processes them.

In simple case, when physical conversion is not required, topology can be simplified to:

```
client (LAC emulator) <- L2TP tunnel -> LNS
```

Mpd is able to operate in both LAC and LNS modes. As LAC mpd supports both simple case LAC emulator and complete LAC topologies. Complete LAC can be configured by joining two physical devices using mpd's repeater functionality.

Windows L2TP client uses IPSec encryption for the additional tunnel security. So, to let it connect you must configure IPSec on your MPD router or disable IPSec on Windows by setting registry DWORD value ProhibitIpSec at the key HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RasMan\Parameters\ into "1".

`set l2tp self` *ipaddr* [*port*]
: Sets the local IP address and port for the L2TP connection. There is known implementation bug if this option is not set while accepting incoming connections on the router with multiple IPs and clients are connecting not to the nearest address of this router.

`set l2tp peer` *ipaddr* [*port*]
: Sets the peer IP address and port for the L2TP 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 l2tp hostname` *name*
: Sets the L2TP tunnel local hostname. For server side, only one unique hostname supported for every pair of listening IP (set l2tp self ...) and peer ip (set l2tp peer ...). If several hostnames defined, only the first matching will be used for all incoming connections.

`set l2tp secret` *secret*
: Sets the L2TP tunnel secret. Used to authenticate tunnel connection and encrypt important control packets avpairs. For server side, only one unique secret supported for every pair of listening IP (set l2tp self ...) and peer ip (set l2tp peer ...). If several secrets defined, only the first matching will be used for all incoming connections.

NOTE: This options is not related with usual PPP authentication. Windows client does not support tunnel authentication.

`set l2tp callingnum` *number*

`set l2tp callednum` *number*
: Sets the calling and called telephone number to use when initiating a L2TP 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 l2tp enable` *option ...*

`set l2tp disable` *option ...*

Enable and disable L2TP device type options for the link.

The following options are supported:

`originate`
: This option enables the initiation of L2TP connections to the peer. If this option is disabled, mpd will not initiate any L2TP connections. You must configure the peer's IP address with the **set l2tp peer** command.

`incoming`
: This option enables the acceptance of incoming L2TP connections. If this option is disabled, mpd will not accept any incoming connections.

`outcall`
: Inside L2TP tunnel, each individual PPP connection (there may be several), is initiated as either an incoming or an outgoing call. This is to allow, for example, making an outgoing telephone call (by LNS) via a remote access server (LAC), as well as the more common use of forwarding of an incoming telephone call from an access server (LAC) to a remote L2TP server (LNS).

When this option is enabled, mpd will initiate outgoing calls (LNS); otherwise mpd will initiate incoming calls (LAC).

`hidden`
: When L2TP tunnel secret is configured it is possible hide (encrypt) some control data for additional protection.

Default is disabled.

`length`
: By default L2TP uses header Length field for control packets, but not for data

packets. This option enables Length field for data packets. This feature enabled may be useful on links where packets padding may happend. Disabling it reduces overhead by 2 bytes per packet.

Default is disabled.

`dataseq`
: By default L2TP requires header sequence fields for control packets, but not require them for data packets. This option enables sequence fields for data packets. This feature enabled may be useful on links where packets reordering may happend but it is intolerable. Disabling it reduces overhead by 4 bytes per packet.

Default is enabled.

5.7 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.

Note: There is a bug in Windows 2000 PPPoE implementation which makes connection impossible due to LCP mru negotiation failure. Other Windows version have no this bug.

`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 acname` *name*
: Set name of this PPPoE access concentrator to be sent to clients. If not specified, local hostname will be used.

`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 auth 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/cuad0`. 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 42) 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.

For further help, ask to the `freebsd-net@freebsd.org` mailing list or sourceforge.net mpd project forum and mailing list.

It does not works and I don't understand why.

Mpd has extensive logging system supporting syslog. To direct logs into a file add lines

```
!mpd
*.*                                         /var/log/mpd.log
```

into your /etc/syslog.conf file, create /var/log/mpd.log file and send SIGHUP to syslogd to reread config.

To enable all possible logging you can use 'log +all' command.

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_async`
- `ng_bpf`
- `ng_car`
- `ng_deflate`
- `ng_ether`
- `ng_iface`
- `ng_ksocket`
- `ng_l2tp`
- `ng_mppc`
- `ng_ppp`
- `ng_pppoe`

- `ng_pptpgre`
- `ng_nat`
- `ng_netflow`
- `ng_pred1`
- `ng_socket`
- `ng_tcpmss`
- `ng_tee`
- `ng_tty`
- `ng_vjc`

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, while other lines must not.

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` and `set bundle yes noretry` to disable the redial function.

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).

Problems with MPPE stateless when connecting to Cisco devices.
Upgrade the IOS of your device, see: `CSCdu30944 MPPE rejects stateless Fixed in 12.3(11.4)`

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 supports only simpliest LCP callback without number negotiation, without CBCP support and only as client.

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

8.2 Authentication

This chapter describes implementation specific details of MPD.

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

  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.

  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.

  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.

  EAP is an Extensible Authentication Protocol. Mpd supports natively only the EAP-Type MD5; other EAP-Types may be used in conjunction with a RADIUS server.

  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.

8.3 RADIUS authentication

This chapter describes implementation specific details of MPD.

RADIUS internals

Mpd supports both user authentication and session accounting using RADIUS. RADIUS-Accounting and RADIUS-Authentication are independant so it is possible to use them in any combination.

All authentication methods are supported with RADIUS (PAP, CHAP, MS-CHAPv1, MS-CHAPv2, EAP). Password changing is currently not supported.

RADIUS attributes supported by mpd:

N	Name	Access		Accounting		
		Req	Resp	Req	Resp	
1	User-Name	+	+	+	-	
2	User-Password	+	-	-	-	
3	CHAP-Password	+	-	-	-	
4	NAS-IP-Address	+	-	+	-	
5	NAS-Port	+	-	+	-	
6	Service-Type	+	-	+	-	
7	Framed-Protocol	+	-	+	-	
8	Framed-IP-Address	-	+	+	-	
9	Framed-IP-Netmask	-	+	+	-	
12	Framed-MTU	-	+	-	-	
18	Reply-Message	-	+	-	-	
22	Framed-Route	-	+	-	-	
24	State	+	+	+	-	
25	Class	-	+	+	-	
27	Session-Timeout	-	+	-	-	
28	Idle-Timeout	-	+	-	-	
30	Called-Station-Id	+	-	+	-	
31	Calling-Station-Id	+	-	+	-	
32	NAS-Identifier	+	-	+	-	
40	Acct-Status-Type	-	-	+	-	
42	Acct-Input-Octets	-	-	+	-	
43	Acct-Output-Octets	-	-	+	-	
44	Acct-Session-Id	-	-	+	-	
45	Acct-Authentic	-	-	+	-	
46	Acct-Session-Time	-	-	+	-	
47	Acct-Input-Packets	-	-	+	-	
48	Acct-Output-Packets		-	-	+	-
49	Acct-Terminate-Cause	-	-	+	-	
50	Acct-Multi-Session-Id	-	-	+	-	
51	Acct-Link-Count	-	-	+	-	
52	Acct-Input-Gigawords	-	-	+	-	
53	Acct-Output-Gigawords	-	-	+	-	
60	CHAP-Challenge	+	-	-	-	
61	NAS-Port-Type	+	-	+	-	
85	Acct-Interim-Interval	-	+	-	-	
95	NAS-IPv6-Address	+	-	+	-	
99	Framed-IPv6-Route	-	+	-	-	
	Microsoft VSA (311)					
1	MS-CHAP-Response	+	-	-	-	
2	MS-CHAP-Error	-	+	-	-	
7	MS-MPPE-Encryption-Policy	-	+	-	-	
8	MS-MPPE-Encryption-Types	-	+	-	-	
10	MS-CHAP-Domain	-	+	-	-	
11	MS-CHAP-Challenge	+	-	-	-	
12	MS-CHAP-MPPE-Keys	-	+	-	-	
16	MS-MPPE-Send-Key	-	+	-	-	
17	MS-MPPE-Recv-Key	-	+	-	-	
25	MS-CHAP2-Response	+	-	-	-	
26	MS-CHAP2-Success	-	+	-	-	
	mpd VSA (12341)					
1	mpd-rule	-	+	-	-	
2	mpd-pipe	-	+	-	-	
3	mpd-queue	-	+	-	-	
4	mpd-table	-	+	-	-	
5	mpd-table-static	-	+	-	-	
6	mpd-filter	-	+	-	-	
7	mpd-limit	-	+	-	-	
154	mpd-drop-user	-	-	-	+	

Mpd allows RADIUS server to terminate user session by setting vendor specific mpd-drop-user attribute to nonzero value in accounting start/update reply packet.

RADIUS ACL's

Mpd can use the Access Control Lists (ACLs) given by the RADIUS server. This ACLs may include ipfw rules, pipes, queues and tables and also mpd internal traffic filtering/shaping/limiting features. That two sets are redundant. ipfw proposed as standard and universal solution, while internal filter/shaper/limiter based on ng_bpf+ng_car expected to work faster with big number of active links.

To use this features 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
ATTRIBUTE       mpd-table       4       string          mpd
ATTRIBUTE       mpd-table-static       5       string          mpd
ATTRIBUTE       mpd-filter      6       string          mpd
ATTRIBUTE       mpd-limit       7       string          mpd
ATTRIBUTE      mpd-drop-user  154     integer        mpd
#----------------------------------------------------------
```

ipfw

You can write in your RADIUS configuration something like:

```
mpd-table += "1=10.0.0.1",
mpd-table += "1=10.0.0.15",
mpd-pipe += "1=bw 10Kbyte/s",
mpd-pipe += "5=bw 20Kbyte/s",
mpd-rule += "1=pipe %p1 all from any to table(%t1) in",
mpd-rule += "2=pipe %p5 all from table(%t1) to any out",
mpd-rule += "100=allow all from any to any",
```

When mpd receives these parameters it will call ipfw(8) to create firewall rules, pipes and queues with unique numbers starting from 10000 (configurable via 'set global start...'). 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 table 32 add 10.0.0.1
ipfw table 32 add 10.0.0.15
ipfw pipe 10000 config bw 10Kbyte/s
ipfw pipe 10001 config bw 20Kbyte/s
ipfw add 10000 pipe 10000 all from any to table(32) in via ng0
ipfw add 10001 pipe 10001 all from table(32) 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.

internal (ng_bpf/ng_car)

Mpd can create complex per-interface traffic filtering/limiting engines inside netgraph when it is requested by mpd-filter and mpd-limit RADIUS attributes.

mpd-filter attribute is a packet filter declaration for using in mpd-limit. mpd-filter consists of two main parts: match/nomatch verdict and the condition. tcpdump (libpcap) expression syntax used for conditions.

mpd-filter: *match | nomatch {condition}* mpd-limit attribute is an action which should be done for packet. It consists of two main parts: filter and action.

mpd-limit: *{filter} {action}*

Filter can be or "all" (any packet) or "fltX" (packets matching to specified mpd-filter).

filter: *any | fltX*

Action can be: "pass" (stop processing and pass packet), "deny" (stop processing and drop packet), "rate-limit" (do Cisco-like rate-limit), "shape" (do simple RED aware traffic shaping).

Actions "rate-limit" and "shape" can have optional "pass" suffix to stop processing after doing this action.

action: *pass* | *deny* | *rate-limit {rate(bits/s)}* [*{normal burst(bytes)}* [*{extended burst(bytes)}*]] [*pass*] | *shape {rate(bits/s)}* [*{burst(bytes)}*] [*pass*]

As example you can write in your RADIUS configuration something like:

```
mpd-filter += "1#1=nomatch src net 10.0.0.0/24",
mpd-filter += "1#2=match src net 10.0.0.0/10",
mpd-filter += "2#1=match dst net 10.0.0.0/16",
mpd-filter += "2#2=match dst net 11.0.0.0/8",
mpd-limit += "in#1=flt1 pass",
mpd-limit += "in#2=flt2 shape 64000 4000 pass",
mpd-limit += "in#3=all deny",
mpd-limit += "out#1=flt2 pass",
mpd-limit += "out#2=all rate-limit 1024000 150000 300000",
mpd-limit += "out#3=all pass",
```

As result, one ng_bpf node will be created to implement traffic filters and several (two for this example) ng_car nodes for traffic shaping and rate-limiting. Incoming traffic from 10.0.0.0/10 except 10.0.0.0/24 will be passed, traffic to 10.0.0.0/16 and 11.0.0.0/8 will be shaped to 64Kbits/s, all other will be denied. Outgoing traffic to 10.0.0.0/16 and 11.0.0.0/8 will be passed, all other will be limited to 1024Kbit/s.

8.4 External authentication

This chapter describes implementation specific details of MPD.

Mpd supports authentication by calling external authentination script. Path to that script must be specified using **set auth extauth-script ...** command.

On invocation this script receives on stdin set of request attribute:value pairs terminated by empty line and must generate response in same format on stdout.

Currently supported request attributes:

```
USER_NAME
USER_PASSWORD - for PAP only
NAS_PORT
NAS_PORT_TYPE
CALLING_STATION_ID
CALLED_STATION_ID
PEER_ADDR
PEER_PORT
```

Currently supported response attributes:

```
RESULT - one of SUCCESS, UNDEF (if valid USER_PASSWORD returned) or FAIL
USER_NAME
USER_PASSWORD
FRAMED_IP_ADDRESS
FRAMED_ROUTE
FRAMED_IPV6_ROUTE
FRAMED_MTU
SESSION_TIMEOUT
IDLE_TIMEOUT
REPLY_MESSAGE
MS_CHAP_ERROR
MPD_RULE
MPD_PIPE
MPD_QUEUE
MPD_TABLE
MPD_TABLE_STATIC
MPD_FILTER
MPD_LIMIT
```

For most of attributes description look their RADIUS alternatives.

8.5 Hints for developers

This chapter describes describes some hints for developers.

Source-Code-Style
: There is not so much to say about that, just take a look at the existing source files.

  The Tab-Width is eight with an indent of two. Please make a space between operators and operands.

NgFuncGetStats
: NEVER call NgFuncGetStats() with the *clear* parameter set to true, because there are other functions (echo requests/replies, bandwidth management) wich relies on increasing link-stats. Mpd carries a copy of the netgraph link-stats at link-level, just use these instead. You can call LinkUpdateStats() for updating the internal stats-struct.

New Authentication-Backends
: Authentication backends must run independently from the rest of Mpd, i.e. you must not access any Mpd resource, because the authentication process is started in its own thread (you have to take care about thread-safety). An `AuthData` object is passed to your authenticating function which carries a copy of all required information. If your backend provides other parameters, like MTU, IP, etc. then put these at the appropriate place into `AuthData` or `Auth`.

  If you can't avoid reading from Mpd's internal data, then acquire the Giant Mutex:

```
[...]
  pthread_mutex_lock(&gGiantMutex);
  [do whatever]
  pthread_mutex_unlock(&gGiantMutex);
[...]
```

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 2284 EAP

RFC 2419 The PPP DES Encryption Protocol, Version 2 (DESE-bis)

RFC 2427 Multiprotocol Interconnect over Frame Relay

RFC 2433 Microsoft PPP CHAP Extensions

RFC 2516 A Method for Transmitting PPP Over Ethernet (PPPoE)

RFC 2548 Microsoft Vendor-specific RADIUS Attributes

RFC 2637 Point-to-Point Tunneling Protocol (PPTP)

RFC 2661 Layer Two Tunneling Protocol (L2TP)

RFC 2865 Remote Authentication Dial In User Service (RADIUS)

RFC 2866 RADIUS Accounting

RFC 2869 RADIUS Extensions

RFC 3579 RADIUS EAP

9.2 Credits

- EAP, RADIUS enhancements, basework for Mpd-4 was sponsored by SURFnet SURFnet.

CONTENTS