Mpd 5.8 User Manual

Archie Cobbs, Michael Bretterklieber, Alexander Motin

<mpd-users@lists.sourceforge.net>

2016-01-06

Abstract

This manual covers the use of **mpd** version **5.8**, a multi-link PPP daemon for FreeBSD based on `netgraph(4)`. You must be running FreeBSD version 5.x or higher to use it, and it is advised to have the latest system version to get access to all mpd features.

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 as it handles configuration and negotiation in user land, while routing all data packets strictly in the kernel.

Mpd has unified support for many link types:

- **modem** to connect using different asychronous serial connections, including modems, ISDN terminal adapters, and null-modem. Mpd includes event-driven scripting language for modem identification, setup, manual server login, etc.
- **pptp** to connect over the Internet using the Point-to-Point Tunnelling Protocol (PPTP). This protocol is supported by the most OSes and hardware vendors.
- **l2tp** to connect over the Internet using the Layer Two Tunnelling Protocol (L2TP). L2TP is a PPTP successor supported with modern clients and servers.
- **pppoe** to connect over an Ethernet port using the PPP-over-Ethernet (PPPoE) protocol. This protocol is often used by DSL providers.
- **tcp** to tunnel PPP session over a TCP connection. Frames are encoded in the same was as asychronous serial connections.
- **udp** to tunnel PPP session over a UDP connection. Each frame is encapsulated in a UDP datagram packet.
- **ng** to connect using different devices supported by netgraph. Netgraph is highly modular kernel networking system, supporting synchronous serial connections, Cisco HDLC, Frame Relay, and other protocols.

It supports numerous PPP sub-protocols and extensions, such as:

- Multi-link PPP
- PAP, CHAP, MS-CHAP and EAP authentication
- traffic compression (MPPC, Deflate, Predictor-1)
- traffic encryption (MPPE, DESE, DESE-bis)
- IPCP and IPV6CP parameter negotiation

Depending on configured rules and connection parameters mpd can operate as usual PPP client/server or forward connection unmodified to other host using any supported link type providing LAC/PAC/TSA functionality for building distributed access networks.

Mpd also includes many additional features:

- IPv4 and IPv6 support
- Telnet and HTTP control interfaces.
- Different authentication and accounting methods (RADIUS, PAM, script, file, ...)
- NetFlow traffic accounting
- Network address translation (NAT)

- Dial-on-demand with idle timeout
- 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

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 been completely rewritten since then. Mpd is now hosted on sourceforge.net MPD Project Page.

1.2 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.

1.3 Change history

Changes since version 5.7:

- New features:
  - Added JSON format output into the web console.
  - Added 'set l2tp pmask ...' command.
  - Added 'set pppoe mac-format ...' command.
  - Added 'set pppoe max-payload ...' command from RFC 4638.
  - Added 'set radius src-addr ...' command.
  - Added 'set iface keep-timeout' options.
  - Added 'set console auth' options.
  - Added 'agent-cid' global option to control display PPPoE ADSL-Agent-Circuit-Id option in 'show session' command. Default is disabled.
  - Added 'session-time' global option to control display session time in seconds in 'show session' command. Default is disabled.
  - Using 'peer_addr' in ACL tell to use mpd-table with the peer negotiated IP address.
  - Added more wildcards, passed from ACL's.
  - Added more wildcards, passed from 'set iface description ...' command or 'mpd-iface-descr' RADIUS attribute.

  - Added ‘Filter-Id‘ RADIUS attribute.
  - Added support for Backtrace Access Library.
  - Added support for LibreSSL Library.
- Changes:
  - Rename ‘quit‘ command to ‘shutdown‘.
  - ‘authname ...‘ command can be case insensitive.
- Bugfixes:
  - Restore send mac address to RADIUS server in unformatted value.
  - Fix long living bug with ECP.
  - Fix ability to use both IPv4 and IPv6 addresses on the same interface.

Changes since version 5.6:

- New features:
  - Added global ‘qthreshold‘ option.
  - Added ‘unset radius server ...‘ command.
  - Added ‘unset nat ...‘ command.
  - Added ‘Class‘ CoA attribute from RFC 2865.
  - New command ‘show netflow‘ added.
- Changes:
  - NAT rules may be added/deleted without shutdowning interface.
  - NetFlow can export IPv6 data.
  - Interface description may be construct from predefined variables in bundle template.
- Bugfixes:
  - Restore ‘show sessions‘ as unprivileged command.
  - Fix infinite event loop when STDIN redirected to /dev/null after it recently got non-blocking mode support.
  - Fix invalid output of ‘show nat‘ command in some times.
  - Fix some possible memory leaks.

Changes since version 5.5:

- New features:
  - Added ‘mpd-iface-name‘ RADIUS attribute.
  - Added ‘mpd-iface-descr‘ RADIUS attribute.
  - Added ‘mpd-iface-group‘ RADIUS attribute.
  - Added ‘mpd-peer-ident‘ RADIUS attribute.
  - Added ‘set iface name ...‘ command.
  - Added ‘set iface description ...‘ command.
  - Added ‘set iface group ...‘ command.

    - Added support for NetFlow v9 export.
    - Added ‘set l2tp|pptp|tcp|udp resolve-once ...‘ command. They allow to resolve peer address every time on reconnect.
- Changes:
    - Remove dependency from libpdel library. Import required files into the MPD tree.
- Bugfixes:
    - Fix invoke ‘set iface up|down-script‘ without arguments.
    - Fix ‘show eap‘ command
    - Fix build on older FreeBSD versions.
    - Fix several memory leaks.
    - Fix building without SYSLOG_FACILITY option.
    - Fix byte order in ports in ‘set nat red-port‘.
    - Fix some potential crashes because of NULL dereferences.

Changes since version 5.4:

- New features:
    - Added ‘set link redial-delay ...‘ command.
    - Print global filters on ‘show iface|customer‘ commands.
    - Added protocol/port forwarding support for NAT.
    - Added utmpx support on 9-CURRENT.
- Bugfixes:
    - Fix memory leaks on PPTP and RADIUS on some reason.
    - Really make RESULT a mandatory option in ext-auth.

Changes since version 5.3 (most of this work was sponsored by JSC ’Ufanet’):

- New features:
    - Added built-in RADIUS server, supporting RFC 3576: Dynamic Authorization Extensions to RADIUS.
    - Added Disconnect-Request extension support from RFC 3576.
    - Added CoA-Request extension support from RFC 3576.
    - Added ‘authname ...‘ command to choose active link by peer auth name.
    - Added support for DSL Forum vendor-specific Circuit-ID/Remote-ID PPPoE tags and respective RFC 4679 RADIUS VSA.
    - Peer address argument added to interface up/down scripts.

Changes since version 5.2:

- New features:
    - Added ’drop’ link action and ’set link action clear’ command.
    - Added ability to receive link action from AAA in auth reply. It allows AAA to select bundle/repeater configuration for specific user or session.

  - Added global traffic filters support to reduce auth reply size. 'set global filter ...' commands.
  - Added ability to include other local or remote config files. 'load ...' command able to accept configuration file path/URI as first argument.
  - Added support for new ng_netflow node features to improve bidirectional accounting performance.
  - Added 'acct-mandatory' auth option to control accounting start errors handeling. Default is enabled.
- Changes:
  - Improved build modularization to allow more customized builds.
  - Reduced memory usage by more effective ACL memory allocation.
  - Allowed MRRU less then 1500 bytes. RFC claims that 1500 must be supported, but lower values are acceptable.
- Bugfixes:
  - Fix possible crash on nonterminated ident string receive.
  - Fix memory leaks on auth failures.
  - Change NCPs join/leave sequences to avoid ENXIO errors on connect.
  - Use separate socket for getting CCP node ID to avoid fake reports.

Changes since version 5.1:

- New features:
  - Added 'set radius identifier' command.
  - Added '$CallingID' and '$CalledID' modem chat variables. Their values will be reported to the auth backend.
  - Added tunnel related RADIUS attributes of RFC2868 support.
  - 'set auth max-logins' feature can now be case insensitive.
  - Added force ability to the 'set iface addrs' command.
  - IPCP/IPv6CP now closing on interface address assign error or up-script error.
  - Accounting start error now closes link.
  - PPPoE peer address format changed to more traditional.
  - Link peer-as-calling option default changed to disabled. PPTP and L2TP users are advised to check configurations!
  - Some of RADIUS accounting update log messages moved from radius to radius2 log level.
- Bugfixes:
  - Fix PPTP peer address reporting for real LAC/PAC mode.
  - Fix auth thread busy check.
  - Fix incorrect L2TP self address used for outgoing calls when several different addresses configured.

Changes since version 5.0:

- New features:
  - Added support for NS-related RADIUS attributes from RFC 2548.
  - Added global max-children option.
  - Added link, bundle, iface and iface-index RADIUS VSA.
  - Added 'set link mrru ...' command. Set default MRRU to 2048 and maximum to 4096 bytes.
  - Added USER_NT_HASH and USER_LM_HASH ext-auth attributes for MS-CHAP authentication.
  - Added mpd-input-acct/mpd-output-acct RADIUS attributes to allow sending typed traffic accounting using standard RADIUS attributes.
  - Added support for local side IP management using IP pools.
  - Added support for auth/acct-only RADIUS servers. It allows to specify different servers for authentication and accounting in mpd configuration file.
  - Added support for the new ng_pptpgre node design, supporting multiple calls per node. It improves performance, when multiple calls active between two IPs.
- Changes:
  - peer-as-calling and report-mac options moved from radius to link to improve LAC operation.
- Bugfixes:
  - Fixed incorrect link creation error handeling.
  - Added workaround for some incorrect PAP implementations.
  - Changed processing of NAK on multilink options. NAK enables rejected options back.
  - Added missing multilink parameters check in BundJoin().
  - Fixed sending of incoming traffic typed accounting on accounting stop.
  - Fixed using correct proxy-arp MAC when more then one interface matches.
  - Fixed some L2TP and PPPoE errors processing.
  - Fixed TCP and UDP link type nodes naming.

Changes since version 5.0rc2:

- New features:
  - Sending LCP Time-Remaining packet implemented.
- Bugfixes:
  - Fixed MPPC options loss on link disconnect.
  - Fixed crash on PPTP CDN sending error.
  - Fixed incorrect IPCP options reject processing.
  - Fixed MP SHORTSEQ option.
  - Fixed packet order on accepting outgoing PPTP call.

Changes since version 5.0rc1:

- New features:
  - 'auth2' log level added.
- Changes:
  - Always prefer MS-CHAP to others to get encryption keys.
- Bugfixes:
  - Fixed bug in tcpmssfix when compression or encryption is used.
  - Fixed build on FreeBSD 5.x.
  - Fixed build without PPTP or L2TP support.
  - Fixed netflow node creation.

Changes since version 5.0b4:

- New features:
  - 'show pptp' and 'show l2tp' commands added.
- Bugfixes:
  - Rewritten ippool to avoid races on IPCP renegotiation.
- Changes:
  - Rewritten message engine using internal circular queue instead of system pipe.
  - L2TP/PPTP tunnel shutdown is now delayed for better LAC/PAC interoperation.

Changes since version 5.0b3:

- New features:
  - If Framed-Netmask RADIUS attribute != 255.255.255.255 mpd will create Framed-IP-Address/Framed-Netmask route to the client side.
  - Added reporting peer MAC address and interface to AAA. Added NAS-Port-Id RADIUS attribute support.
  - New 'iface' command added.
  - Added IPv6 support for Tee and DialOnDemand.
  - 'set iface addrs' now able to set IPv6 addresses.
  - ACCT_INTERIM_LIM_RECV and ACCT_INTERIM_LIM_XMIT attributes added to ext-auth.
- Bugfixes:
  - Fixed /32 routes processing.
  - Fixed crash on repeater shutdown.
  - Fixed 'create link ' command syntax check.
  - Fixed redial delay.
  - Many small tunings and fixes.
- Performance improvements:
  - Netgraph management completely rewritten. Now 6 sockets per daemon used to communicate with netgraph instead of 4 sockets per link before. This gives significant performance benefit due to reduced pevent engine overhead.

- Internal memory management rewritten.

Changes since version 5.0b1:

- New features:
  - Implemented type-differentiated traffic accounting based on mpd-limit traffic filters.
  - Added 'set link max-children ...' command for DoS protection.
  - Implemented user privilege levels "admin"/"operator"/"user".
  - Web console rewritten and allows now execute any commands allowed by privileges. Added plain-text command interface.
  - New 'show sessions' and 'show customer' commands added.
  - Implemented one-shot operation mode to allow mpd to be used in complicated dial setups.
  - Acct-Session-Id attribute now present in auth request.
  - Show to auth real PPPoE session name received from peer.
- Changes:
  - Rewritten PPPoE, L2TP, TCP and UDP link types to fulfill new dynamic design.
  - MPPC related options moved from 'set ccp' to the new 'set mppc' command.
  - 'set bundle retry' command renamed to 'set bundle fsm-timeout'.
  - Number of auth retries increased to 5.
  - PPTP windowing is disabled by default.
  - Improved unified command error reporting.
  - Users list is now global and the same for console and web.
- Bugfixes:
  - Fixed memory leak on link/bundle shutdown.
  - Fixed reference (memory) leak on console close.
  - Fixed netflow setup errors handeling.
  - Improved IfaceIp[v6]IfaceUp() errors handeling.
  - Restore link MRU to default after use. Should help with some EAP-TLS cases.
  - MPPC now automaically disables unusable subprotocols. For example, it is impossible to use MPPE encryption without MSCHAP.
  - Fixed FSM instantiation to fix LCP keep-alives.
  - Fixed 'set eap ...' context.
  - Implemented PAP-ACK packet retransmit.
  - 'show mem' command now returns output to console instead of stdout.
  - Many small fixes.

Changes since version 4:

- Design changes:

  - Removed static link - bundle relations. Links now choose their bundles using negotiated parameters when they reach NETWORK phase. The benefit of it is simple and complete client and server multilink operation. Also it gives ability to implement more complicated LAC, PAC and TSA setups then it was possible before.
  - Implemented template based dynamic link/bundle creation. It allows significantly reduce amount of configuration required to operate big access servers.

    Link may be autocreated by incoming call request from device or by DoD/BoD request from bundle. Bundle may be autocreated by the link reached NETWORK phase.
  - To simplify configuration link and phys layers separated since version 4.2 are now rejoined again into a single link layer.
- New features:
  - Added PAM authentication and accounting.
  - Added dynamic IP addresses pools support.
  - Added new 'ext-acct' accounting backend as full-featured alternative to 'radius-acct'.
- Changes:
  - Massive changes in configuration commands. You should read the manual and examples for the new configuration techniques.
  - FreeBSD 4.x and old DragonFly releases are not supported anymore.

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):

- 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.secret` 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.secret` 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/mpd5`. 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/mpd5`. 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/mpd5` 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, 'default' configuration is loaded.

  Each configuration defines one or more bundles, links and repeaters. They can be defined with the `create` command. Subsequent commands in the configuration configure the various layers for that item.

`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. Run 'configure' script and 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`, 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.

`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 ]
```

On startup mpd run commands from the `startup` label. After that, if *configuration* is supplied, mpd looks in `mpd.conf` for a matching label, and runs the corresponding commands. If *configuration* is not supplied, mpd looks for the configuration named `default`.

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`.

`-o` `--one-shot`
: This option forces mpd terminate itself after the last link disappear. Link templates are not counted as links.

`-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.

`-m` `--pam-service` *service*
: Mpd normally invoke `pam(3)` with the service `mpd`. This option allows you to change that service.

`-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 three 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

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 indented with a tab character and follow the label on the next and subsequent lines.

Here is an example `mpd.conf` that contains a single configuration `client` that defines a single bundle template `B1` and one link `L1`:

```
#
# mpd.conf configuration file
#

client:
	create bundle template B1

	create link static L1 modem
	set modem device /dev/cuau0
	set modem speed 115200
	set modem script DialPeer
	set modem idle-script AnswerCall
	set modem var $DialPrefix "DT"
	set modem var $Telephone "1234567"
	set link no pap chap eap
	set link accept pap
	set auth authname "MyLogin"
	set auth password "MyPassword"
	set link max-redial 0
	set link action bundle B1
	open
```

Commands are independant and executed one by one as if they were entered using console. Most command operation depends on the active context, which includes currently active link, bundle and repeater. Information about active context is shown at the console command prompt. Some commands, like `link`, `bundle`, `repeater` and `create` can change this context.

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.secret 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`, 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.

See *chat* (section 6. , page 72) 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` config file.

At any given time, there is always exists an *active context* shich includes active link, bundle and repeater. These name 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 5.0 (root@orphanage.alkar.net 13:16 22-Sep-2007)
[modem1]
```

By that time, mpd has already read in and executed all the commands associated with the configuration `config1`, which defines the bundle `myisp`.

To control mpd runtime you also 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 5.0a1 (root@orphanage.alkar.net 13:16 22-Sep-2007)

[modem1]
```

The prompt `[modem1]` shows the currently active link (`modem1`). Any link specific commands issued at this point will apply to `modem1`.

To change the active link, bundle or repeater, use the `link`, `bundle` or `repeater` 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        device   : Physical device status
 link     : Link status                 auth     : Auth status
 radius   : RADIUS status               lcp      : LCP status
 nat      : NAT status                  mem      : Memory map
 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.

`create bundle [ template | static ]` *name* [*template*]
: Create new bundle with specified name, optionally based on specified template. If 'template' option is specified bundle will not operate itself, but will be used to create new bundles on demand. If 'static' option is specified, it will be a permanent bundle which will not disapear after operation complete. Maximum name length is 10 characters for templates and 15 characters for the rest of bundles.

`create link [ template | static ]` *name* (*type* | *template*)
: Create new link with specified name and type, optionally based on specified template. If 'template' option is specified link will not operate itself, but will be used to create new links on demand. If 'static' option is specified, it will be a permanent link which will not disapear after operation complete. Maximum name length is 10 characters for templates and 15 characters for the rest of links.

`destroy bundle [` *name*]

destroy link [*name*]" Destroy existing link/bundle. Note, that if link is in use it may not disapear after command return, it may take some time to properly close it.

`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.

`iface [` *iface*]
: This command changes the currenly active bundle to the bundle using interface *iface*.

`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*.

`authname [` *name*]
: This command changes the currenly active link to the link holding session with peer auth name *name*. If CI argument is present login comparasion will be case insensitive.

`load [` *file*] *label*
: This command causes mpd to read the entry for *label* out of `mpd.conf` or other file if specified and execute the corresponding commands. *file* can be relative or absolute file path or http/https/ftp URL. Note, that remote file access may be less reliable.

`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.

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

  routes
  : Show the current IP routing table.

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

  ippool
  : Show status information about configures IP pools.

  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.

l2tp
: Show active L2TP tunnels.

pptp
: Show active PPTP tunnels.

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

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

version
: Show running mpd version and supported features.

sessions [*param value*]
: Show active sessions conforming specified param/value. Available params: iface, ip, bundle, msession, link, session, user, peer.

customer
: Show active customer details.

summary
: Show status summary.

console
: Show console summary.

web
: Show web server summary.

user
: Show defined console users.

netflow
: Show status information about NetFlow.

`set ...`

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

`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 pptptimeout` *seconds*

set global l2tptimeout *seconds*" Set PPTP/L2TP tunnels inactivity shutdown delay.

Default is 10 seconds.

`set global pptplimit` *num*

set global l2tplimit *num*" Limit maximum number of calls per PPTP/L2TP tunnel. When limit is reached new tunnel will be created for the next call. This option violates PPTP RFC which requires only one tunnel existing for the PAC/PNS pair, but it can increase busy tunnels stability.

Defaults are 100 (10 for L2TP before FreeBSD 6.3-STABLE and 7.0-RELEASE).

`set global max-children` *num*
: This option specifies global maximum number of links, created using template, that could exist at the same time.

The default value is 10000.

`set global qthreshold` *min max*
: This option specifies global message queue limit thresholds.

The default values are 64 and 256.

`set global filter` *num* add *fltnum flt*

set global filter *num* clear" These commands define or clear traffic filters to be used by rules submitted by *radius* (section 4.10.1 , page 49) during authentication.

`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:

`one-shot`
: This option forces mpd terminate itself after the last link disappear. Link templates are not counted as links.

The default is disable.

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

The default is disable.

`agent-cid`
: Display PPPoE *ADSL-Agent-Circuit-Id* option in *show session* command.

The default is disable.

`session-time`
: Display session time in seconds in *show session* command.

The default is disable.

`set user` *username password*
: [*admin* | *operator* | *user*]" This command configures which users are allowed to connect to the console. It may be invoked multiple times with different usernames.

`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

`unset ...`

This command allows to unsets some of configuration information set using `set` command.

`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.

device
: 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:

auth*
: Link authentication events

bund*
: Bundle layer events

ccp*
: Compression events and negotiation

chat*
: Modem chat script

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

echo
: Echo/reply packets for all FSM's

ecp*
: Encryption events and negotiation

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

ipv6cp*
: IPv6 control protocol events and negotiation

lcp*
: Link control protocol events and negotiation

link*
: Link layer events

phys*
: Device layer events

radius*
: RADIUS authentication events

rep*
: Repeater layer events

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

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

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

4.2 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". Operation mode is getting chosen dynamically depending on link configuration and user auth name (see 'set link action ...' command).

In "ppp terminator" mode such hierarchy used: *interface* **(section 4.9 , page 44)** -> *ipcp* **(section 4.7 , page 43)** -> *compression* **(section 4.6 , page 41)** -> *encryption* **(section 4.5 , page 40)** -> *bundle* **(section 4.4 , page 38)** -> *links* **(section 4.3 , page 32)**

In "ppp repeater" mode different hierarchy used: *links* **(section 4.3 , page 32)** -> **Repeater** -> *links* **(section 4.3 , page 32)**

A *links* **(section 4.3 , page 32)** is a single point-to-point connection between the local machine and a remote peer machine, implemented by some kind of physical device, such as a serial modem connection or a virtual PPTP connection.

Link parameters include whether authentication is enabled in either direction, the authentication type (PAP or CHAP) used, keep-alive packets, multilink negotiation options and various other link specific parameters.

Lower half of the link layer is the physical devices 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 *bundle* **(section 4.4 , page 38)** 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.

With each bundle is a corresponding *interface* **(section 4.9 , page 44)** layer, which corresponds directly to a system network interface accessible via `ifconfig(8)`, such as `ng0`. 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 *ipcp* **(section 4.7 , page 43)** and *ipv6cp* **(section 4.8 , page 44)**, which manages the protocol 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 4.6 , page 41)** and *encryption* **(section 4.5 , page 40)** layers, which allow you to enable and configure compression and encryption for data sent and received over the bundle.

A **repeater** is a collection of two physical devices (links). It is getting created when there is need to forward connection coming from one physical device to another without modification. This technology is also known as LAC (L2TP Access Concentrator) and 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.

4.3 Link layer

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

`set link action (bundle|forward)` *name* [*regexp*]

`set link action drop [` *regexp*]

List of such command describes how incoming calls should be processed. "bundle" means that connection should be processed locally with specified bundle. "forward" means that connection should be forwarded using repeater to the specified link. "drop" means that connection should be dropped. Optional "regexp" parameneter defines regular expression which will be checked against peer auth name.

Actions from list are checked in order of definition until regexp match will be found. Actions processed at three points. First time they are checked just after connection acception, second time just after receiving peer's auth during LCP negotiation and third time when link is authenticated. As during first check there is no peer auth name known yet, check will be skipped if there are more then one action specified for link or if action has regexp specified.

`set link action clear`

Clears link actions list.

`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*

`set link mrru` *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 up to MRRU bytes 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 forty.

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. 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 redial-delay` *seconds*
: This command defines time between connection retries.

  The default value is 1.

`set link max-children` *num*
: This template option specifies maximum number of links, created using this template, that could exist at the same time. Value 0 disables template.

  The default value is 10000.

`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 Microsoft CHAP over MD5 CHAP to get encryption keys.

  This option is an alias for `chap-md5 chap-msv1 chap-msv2`

`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. For details see *eap* (section 4.3.1 , page 37).

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

`incoming`

This option enables the acceptance of incoming connections. If this option is disabled, mpd will not accept incoming connections using this link. To avoid races it is advised to enable it after all other link options are configured.

The default is **disable**.

`multilink`

This command enables multi-link PPP on the link. 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.

The default is **disable** (i.e., normal non-multilink PPP).

`shortseq`

This option is only meaningful if multi-link PPP is negotiated. It proscribes shorter multi-link fragment headers, saving two bytes on every frame.

The default is **enable** and **accept**.

`acfcomp`

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

The default is **enable** and **accept**.

`protocomp`

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

The default is **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**.

`time-remain`
: Send Time-Remaining LCP packet to the peer if AAA returned session timeout.

  Default **disable**.

`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 enabled for untrusted peers.

  Default is **disable**.

`report-mac`
: Forces mpd to send peer MAC address and interface in Calling-Station-Id.

  Default **disable**.

4.3.1 EAP

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.

The default is disable and accept.

`radius-proxy`

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

The default is disable.

4.4 Bundle layer

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.

`set bundle period` *seconds*

set bundle hiwat *percent*

set bundle lowat *percent*

set bundle min-con *seconds*

set bundle min-dis *seconds*

set bundle links *link1* [...]" These commands are only meaningful when bandwidth management is enabled. Mpd will attempt to bring up a new link from the list specified 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 points, 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 fsm-timeout` *seconds*

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

`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.''

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

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

  The default is disable.

`ipcp`
: Enable/Disable IPCP negotiation. For details see *ipcp* (section 4.7 , page 43).

  The default is enable.

`ipv6cp`
: Enable/Disable IPV6CP negotiation.

  The default is disable.

`compression`
: Enable/Disable CCP (Compression Control Protocol) negotiation. For details see *compression* (section 4.6 , page 41).

  The default is disable.

`encryption`
: Enable/Disable ECP (Encryption Control Protocol) negotiation. For details see *encryption* (section 4.5 , page 40).

  The default is disable.

`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 is disable.

4.5 Encryption (ECP) layer

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.

The default is disable.

`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.

The default is disable.

4.6 Compression (CCP) layer

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.

`pred1`

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

This option requires ng_pred1 Netgraph node type, which is present since FreeBSD 6.2-STABLE of 2007-01-28. If there is no ng_pred1 in system, this algorithm is supported in user-level, but will consume more CPU power.

The default is disable.

`deflate`

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

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

The default is disable.

`mppc`

This option enables MPPC compression/encryption subprotocol. For details see *mppc* (section 4.6.1 , page 41).

The default is disable.

4.6.1 MPPC protocol

Mpd implements Microsoft Point-to-point compression (MPPC) CCP subprotol. To enable it, 'mppc' option should be enabled at the CCP layer.

MPPC CCP subprotocol consists of MPPC compression and MPPE encryption parts. To make MPPC CCP actually do something you should enable some of them using options below.

`set mppc accept` *option ...*

`set mppc deny` *option ...*

`set mppc enable` *option ...*

`set mppc disable` *option ...*

`set mppc yes` *option ...*

`set mppc no` *option ...*

These commands configure various MPPC options.

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

  Note: This option requires ng_mppc node to be built with compression support, which is disabled by default, requiring external compressor code. Open-source implementation of this code, based on RFC2118, could be found at http://mavhome.dp.ua/MPPC/. But MPPC compression algorithm itself covered by US patent, so you may need to contact Hi/Fn Inc. to obtain their proprietary implementation. If kernel support is not detected, compression will not be negotiated. Use 'show version' command to get actual status.

  The default is disable.

`e40`
: Enables 40-bit MPPE encryption.

  The default is disable.

`e56`
: Enables 56-bit MPPE encryption.

  The default is disable.

`e128`
: Enables 128-bit MPPE encryption.

  Note: in order for MPPE encryption to work, MS-CHAPv1 or MS-CHAPv2 auth is mandatory, because the MPPE keys are generated using the authentication results. If MS-CHAP auth is not used by link, encryption will not be negotiated.

  The default is disable.

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

  The default is disable.

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

  The default is disable.

4.7 IPCP layer

This chapter describes commands that configure the IP Control Protocol (IPCP) layer. To enable IPCP, `ipcp` option should be enabled at the bundle layer. All of these commands apply to the currently active bundle.

`set ipcp ranges` (*local/width* | ippool *pool*) (*remote/width* | ippool *pool*)
: 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.

  It is also possible to specify ippool name to use for assigning remote ip. In such case width 32 is assumed.

  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.

4.8 IPv6CP layer

Mpd implements IPv6 Control Protocol (IPv6CP) for IPv6 link level addresses negotiation. To enable IPv6CP, `ipv6cp` option should be enabled at the bundle layer.

Now there is no additional configuration options present at this layer.

4.9 Interface layer

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 *ipcp* (section 4.7 , page 43) which is specific to IP.

`set iface name [` *name* `]`
: This command changes interface name from default ngX to specified one. If name argument is not specified, original ngX name is restored. Note that inside Netgraph original ngX name is always used.

`set iface description [` *description* `]`
: This command changes interface description.

  Template may contain conversion specifications:

```
%% expands to single % sign;
%a for interface local address;
%A for peer local address;
%i for system interface index;
%I for interface name;
%l for name of bundle's first link
%M for peer MAC address of bundle's first link
%o for local outer ("physical") address of bundle's first link
%O for peer outer ("physical") address of bundle's first link
%P for peer outer ("physical") port of bundle's first link
%S for interface status (DoD/UP/DOWN)
%t for type of bundle's first link (pppoe, pptp, l2tp etc.)
%u for self auth name (or dash if self auth name not used)
%U for peer auth name (or dash if peer has not authenticated)
```

`set iface group [` *group*]
: This command add interface to specific group.

`set iface addrs [!]`*local-ip* [!]*remote-ip*
: This command is usually 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.

  Also this command may be used to force specified addresses usage instead of negotiated ones. It may be useful in some specific cases, for example, to avoid routing loop with misconfigured PPTP server. In such case '!' mark specifies IPs to be forced.

`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 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.

  The default is 0.

`set iface session` *seconds*

Sets the session timeout value for the bundle. An session timeout of zero disables this feature.

The default is 0.

`set iface up-script` *script*

`set iface down-script` *script*

Mpd can optionally run a user program every time one of network protocols (IPCP/IPv6CP) 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 ]
peer-address
```

If up-script exit status is not 0, mpd will kill respective protocol.

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

```
script interface proto local-ip remote-ip authname peer-address
```

`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.

The default is disable.

`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
```

The default is disable.

`keep-timeout`

When this option is enabled, we are not clear ''session timeout'' counter. This is useful together with CoA requests.

The default is disable.

`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@mind-step.com>.

The default is disable.

`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.

The default is disable.

`nat`

Enable NAT for this bundle. See *nat* (section 4.14 , page 62) for more information.

The default is disable.

`netflow-in`

`netflow-out`

`netflow-once`

If enabled, the bundle will participate in NetFlow generation. See *netflow* (section 4.12.1 , page 59) for more information. The netflow-out option is like netflow-in, but generates NetFlow data for outgoing packets instead of incoming. If netflow-once enabled - packet will be accounted only once while in kernel.

The default is disable.

`ipacct`

If enabled, the bundle will participate in IP accounting with ng_ipacct node. See *ipacct* (section 4.12.2 , page 60) for more information.

The default is disable.

4.10 Authentication, Authorization and Accounting (AAA)

Mpd currently supports authentication against (tried in this order) *extauth* (section 4.10.2 , page 55), *radius* (section 4.10.1 , page 49), PAM, systems password database (`master.passwd`), OPIE and internal `mpd.secret` file.

This chapter describes commands that configure the Authentication subsystem of LCP layer. All of these commands apply to the currently active link.

`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* may have a corresponding entry in `mpd.secret`. 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.secret`, but rather to use *password* instead. If you're too lazy to set up `mpd.secret` and are only dialing out, you can use this command instead.

`set auth max-logins` *num* [CI]
: Limit the max. amount of concurrent logins with the same username. If set to zero, then this feature is disabled. If CI argument is present login comparasion will be case insensitive.

`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 extauth-script` *script*

set auth extacct-script *script*" Sets scripts names for external authentication and accounting.

`set auth enable` *option ...*

set auth disable *option ...*"

The options available are:

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

  Default **enable**.

`radius-auth`
: Enable authentication via RADIUS. For details see *radius* (section 4.10.1 , page 49).

  Default **disable**.

`radius-acct`
: Enable per link accounting via RADIUS. For details see *radius* (section 4.10.1 , page 49).

  Default **disable**.

`ext-auth`
: Enable authentication by calling external script. This method pretended to be a fullfeatured alternative to the **radius-auth**. For details see *extauth* (section 4.10.2 , page 55).

  Default **disable**.

`ext-acct`
: Enable accounting by calling external script. This method pretended to be a fullfeatured alternative to the **radius-acct**. For details see *extauth* (section 4.10.2 , page 55).

  Default **disable**.

`pam-auth`
: Enables authentication using PAM service "mpd". This options can only be used with PAP.

  Default **disable**.

`pam-acct`
: Enable accounting using PAM service "mpd".

  Default **disable**.

`system-auth`
: 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**.

`system-acct`
: Enable accounting via utmp/wtmp.

  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.secret` 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**.

`acct-mandatory`
: Makes accounting start mandatory. If enabled, on accounting start failure connection will be dropped.

  Default **enable**.

4.10.1 RADIUS

This chapter describes RADIUS authentication backend. 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.

All of these commands apply to the currently active link.

`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. If one of auth/acct ports specified as 0, it will not be used for requests of that type.

`unset radius server` *name* [*auth-port* [*acct-port*]]
: Deletes cpecific RADIUS server from pool.

`set radius src-addr` *ipaddr*
: Configure IP address on the multihomed host that is used as a source address for all requests.

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

  The default is 5 second.

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

  The default is 3 retries.

`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 identifier` *name*
: Send the given name in the RAD_NAS_IDENTIFIER attribute to the server. If not set the local hostname is used.

`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.

RADIUS internals

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	-	+	+	-	
11	Filter-Id	-	+	-	-	
12	Framed-MTU	-	+	-	-	
13	Framed-Compression	-	+	-	-	
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	+	-	+	-	
64	Tunnel-Type		+	-	+	-
65	Tunnel-Medium-Type	+	-	+	-	
66	Tunnel-Client-Endpoint	+	-	+	-	
67	Tunnel-Server-Endpoint	+	-	+	-	
85	Acct-Interim-Interval	-	+	-	-	
87	NAS-Port-Id		+	-	+	-
88	Framed-Pool		-	+	-	-
90	Tunnel-Client-Auth-ID	+	-	+	-	
91	Tunnel-Server-Auth-ID	+	-	+	-	
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	-	+	-	-	
28	MS-Primary-DNS-Server	-	+	-	-	
29	MS-Secondary-DNS-Server	-	+	-	-	
30	MS-Primary-NBNS-Server	-	+	-	-	

```
31  MS-Secondary-NBNS-Server  -       +       -       -

    DSL Forum VSA (3561)
1   ADSL-Agent-Circuit-Id     +       -       +       -
2   ADSL-Agent-Remote-Id      +       -       +       -

    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                 -       +       -       -
8   mpd-input-octets          -       -       +       -
9   mpd-input-packets         -       -       +       -
10  mpd-output-octets         -       -       +       -
11  mpd-output-packets        -       -       +       -
12  mpd-link                  +       -       +       -
13  mpd-bundle                -       -       +       -
14  mpd-iface                 -       -       +       -
15  mpd-iface-index           -       -       +       -
16  mpd-input-acct            -       +       -       -
17  mpd-output-acct           -       +       -       -
18  mpd-action                -       +       -       -
19  mpd-peer-ident            +       -       +       -
20  mpd-iface-name            -       +       -       -
21  mpd-iface-descr           -       +       -       -
22  mpd-iface-group           -       +       -       -
154 mpd-drop-user             -       -       -       +
```

To use mpd VSA you should add such dictionary to your RADIUS server:

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

VENDOR          mpd             12341

BEGIN-VENDOR   mpd

ATTRIBUTE      mpd-rule       1       string
ATTRIBUTE      mpd-pipe       2       string
ATTRIBUTE      mpd-queue      3       string
ATTRIBUTE      mpd-table      4       string
ATTRIBUTE      mpd-table-static       5       string
ATTRIBUTE      mpd-filter     6       string
ATTRIBUTE      mpd-limit      7       string
ATTRIBUTE      mpd-input-octets       8       string
ATTRIBUTE      mpd-input-packets      9       string
ATTRIBUTE      mpd-output-octets      10      string
ATTRIBUTE      mpd-output-packets     11      string
ATTRIBUTE      mpd-link       12      string
ATTRIBUTE      mpd-bundle     13      string
ATTRIBUTE      mpd-iface      14      string
ATTRIBUTE      mpd-iface-index        15      integer
ATTRIBUTE      mpd-input-acct 16      string
ATTRIBUTE      mpd-output-acct        17      string
ATTRIBUTE      mpd-action     18      string
ATTRIBUTE      mpd-peer-ident 19      string
ATTRIBUTE      mpd-iface-name 20      string
ATTRIBUTE      mpd-iface-descr        21      string
ATTRIBUTE      mpd-iface-group        22      string
ATTRIBUTE      mpd-drop-user  154     integer

END-VENDOR     mpd
#----------------------------------------------------------
```

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.

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...'). %rX, %pX, %qX, %tX and %aX macroses will be expanded within mpd-rule and mpd-queue. To the end of each rule will be added "via ngX" to make the rule apply only to that client's networking interface.

Allowed macroses:

```
%rX IPFW rule pool
%pX IPFW pipe pool
%qX IPFW queue pool
%tX IPFW table pool
%a1 peer negotiated IP address
%a2 self negotiated IP address
```

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.

Note: As soon as mpd executes ipfw commands using shell, shell's special characters like "(" and ")" must be slashed.

You can specify *mpd-table += "1=peer_addr"* to use mpd-table with the peer negotiated IP address.

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: "" (do nothing, just account), "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 dst net 10.0.0.0/24",
mpd-filter += "1#2=match dst net 10.0.0.0/8",
mpd-filter += "2#1=nomatch src net 10.0.0.0/24",
mpd-filter += "2#2=match src net 11.0.0.0/8",
mpd-limit += "in#1=flt1 pass",
mpd-limit += "in#2#Biz=all shape 64000 4000",
mpd-limit += "out#1=flt2 pass",
mpd-limit += "out#2#Biz=all rate-limit 1024000 150000 300000",
```

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 to 10.0.0.0/8 except 10.0.0.0/24 will be passed, other traffic will be shaped to 64Kbits/s. Outgoing traffic from 10.0.0.0/8 except 10.0.0.0/24 will be passed, all other will be limited to 1024Kbit/s. Also traffic that passed mpd-limit rules marked "Biz" will be accordingly accounted and present with that name in AAA accounting requests.

4.10.2 External authentication

Mpd supports authentication and accounting by calling external scripts. Pathes to that scripts must be specified using **set auth extauth-script ...** and **set auth extacct-script ...** commands.

On invocation extauth/extacct scripts receive on stdin set of request attribute:value pairs terminated by empty line and must generate response in same format on stdout.

Supported attributes:

Name	Access		Accounting		
	Req	Resp	Req	Resp	
USER_NAME	+	+	+	-	
AUTH_TYPE	+	-	-	-	
USER_PASSWORD	+	+	-	-	
USER_NT_HASH	-	+	-	-	
USER_LM_HASH	-	+	-	-	
LINK	+	-	+	-	
NAS_PORT	+	-	+	-	
NAS_PORT_TYPE	+	-	+	-	
CALLING_STATION_ID	+	-	+	-	
CALLED_STATION_ID	+	-	+	-	
SELF_NAME	+	-	+	-	
PEER_NAME	+	-	+	-	
SELF_ADDR	+	-	+	-	
PEER_ADDR	+	-	+	-	
PEER_PORT	+	-	+	-	
PEER_MAC_ADDR	+	-	+	-	
PEER_IFACE	+	-	+	-	
PEER_IDENT	+	-	+	-	
RESULT	-	+	-	-	
FRAMED_IP_ADDRESS	-	+	+	-	
PRIMARY_DNS_SERVER	-	+	-	-	
SECONDARY_DNS_SERVER	-	+	-	-	
PRIMARY_NBNS_SERVER	-	+	-	-	
SECONDARY_NBNS_SERVER	-	+	-	-	
FRAMED_ROUTE	-	+	-	-	
FRAMED_IPV6_ROUTE	-	+	-	-	
FRAMED_MTU	-	+	-	-	
FRAMED_COMPRESSION	-	+	-	-	
FRAMED_POOL	-	+	-	-	
SESSION_TIMEOUT		-	+	-	-
IDLE_TIMEOUT	-	+	-	-	
ACCT_INTERIM_INTERVAL	-	+	-	-	
ACCT_INTERIM_LIM_RECV	-	+	-	-	
ACCT_INTERIM_LIM_XMIT	-	+	-	-	
REPLY_MESSAGE	-	+	-	-	
MS_CHAP_ERROR	-	+	-	-	
IFACE	-	-	+	-	
IFACE_INDEX	-	-	+	-	
BUNDLE	-	-	+	-	
ACCT_STATUS_TYPE	-	-	+	-	
ACCT_SESSION_ID		+	-	+	-
ACCT_MULTI_SESSION_ID	-	-	+	-	
ACCT_LINK_COUNT		-	-	+	-
ACCT_TERMINATE_CAUSE	-	-	+	-	
ACCT_SESSION_TIME	-	-	+	-	
ACCT_INPUT_OCTETS	-	-	+	-	
ACCT_INPUT_PACKETS	-	-	+	-	
ACCT_OUTPUT_OCTETS	-	-	+	-	
ACCT_OUTPUT_PACKETS	-	-	+	-	
MPD_RULE	-	+	-	-	
MPD_PIPE	-	+	-	-	
MPD_QUEUE	-	+	-	-	
MPD_TABLE	-	+	-	-	
MPD_TABLE_STATIC	-	+	-	-	
MPD_FILTER	-	+	-	-	
MPD_LIMIT	-	+	-	-	
MPD_INPUT_OCTETS	-	-	+	-	
MPD_INPUT_PACKETS	-	-	+	-	
MPD_OUTPUT_OCTETS	-	-	+	-	
MPD_OUTPUT_PACKETS	-	-	+	-	
MPD_ACTION	-	+	-	-	
MPD_DROP_USER	-	-	-	+	
MPD_IFACE_NAME	-	+	-	-	

```
MPD_IFACE_DESCR                      -       +       -       -
MPD_IFACE_GROUP                      -       +       -       -
```

Attribute USER_PASSWORD provided in Access-Request only for PAP auth. Attributes USER_NT_HASH and USER_LM_HASH in Access-Reply used for MS-CHAP auth only.

In reply script may specify any combination of attributes it needs. The only mandatory attribute in Access-Response is RESULT. RESULT must be one of SUCCESS, UNDEF or FAIL. RESULT UNDEF means that mpd should authenticate user itself using USER_PASSWORD/USER_NT_HASH attribute supplied by script.

For description of most attributes look their RADIUS alternatives.

4.11 Dynamic Authorization

After session has been first time authorized by AAA subsystem, Mpd provides you several ways to affect it's further operation. Process of affecting established session called dynamic authorization.

There are two types of dynamic authorization activities exist: Disconnect (terminate session, causing it's graceful shutdown) and Change of Authorization, CoA (changing session operation parameters, such as speed, ACLs and so on, on-flight)

Mpd provides several different control interfaces, that can be used to implement dynamic authorization.

Control consoles
: The basic method of controlling mpd is it's STDIN and TCP *console* **(section 4.15 , page 63)**.

  You can disconnect any session by connecting to console, selecting required session with any command for changing current context, such as: *link*, *bundle*, *session*, *msession* and so on, and using *close* command.

Web server
: Mpd provides two *web* **(section 4.16 , page 64)** interfaces: human (text/html) and binary (text/plain).

  Human web interface allows you disconnect specified session just by clicking on respective *[Close]* link on the "Current status summary" web page on mpd built-in web server.

  Binary web interface provides API for executing any of control console commands via HTTP request. For example, to disconnect session on link named L125 you may use such HTTP request: */bincmd?link%20L125&close*

RADIUS accounting
: Mpd provides simple, but non-standard method of disconnecting session using *radius* **(section 4.10.1 , page 49)** accounting reply. To disconnect arbitrary session you may just include *mpd-drop-user* attribute with nonzero value into any accounting reply packet.

  This method considered not to be completely reliable, as AAA receives no acknowledge that accounting reply packet was received by mpd. The only thing guarantied, is that on packet loss mpd will retry accounting sending for specified number of times before giveup.

Built-in RADIUS server
: RFC 3576: "Dynamic Authorization Extensions to RADIUS" defines standard way to implement dynamic authorization. It defines two additional RADIUS request types:

Disconnect-Request and CoA-Request, to be sent from AAA server to dedicated UDP port on NAS with regular RADIUS protocol.

To have this function working, mpd should be built with *libradius* library, having RADIUS server functionality (FreeBSD 7/8-STABLE after 2009-10-30).

This chapter describes commands that configure mpd's built-in RADIUS server. All of these commands are executed in global context.

`set radsrv open`
: Opens the RADIUS server, i.e., creates the listening UDP socket.

`set radsrv close`
: Closes the RADIUS server, i.e., closes the listening UDP socket.

`set radsrv self` *ip* [*port*]
: Sets the credentials for the RADIUS-listener. After changing one of these options, the RADIUS server must be closed and re-opened for the changes to take effect.

  The default is '0.0.0.0 3799'.

`set radsrv peer` *ip secret*
: Defines additional AAA server, allowed to contact this NAS. After changing one of these options, the RADIUS server must be closed and re-opened for the changes to take effect.

`set radsrv enable` *option ...*

: set radsrv disable *option ...*" These commands configure various RADIUS server options.

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

  The options available for the RADIUS server are:

`coa`
: This option enables CoA-Request support on RADIUS server.

  The default is enable.

`disconnect`
: This option enables Disconnect-Request support on RADIUS server.

  The default is enable.

Dynamic authorization RADIUS server receives three groups of attributes: NAS identification (to be sure that request got to the right server), session identification (to identify session that should be affected) and session parameters (to describe new session state to set). NAS and session identification attributes are native part of any Disconnect or CoA request, while session parameters could be used only with CoA. At least one session identification attribute must be present in request. If there are several identification attributes present, session should match all of them to be affected.

NAS identification attributes supported by mpd:

```
N   Name
4   NAS-IP-Address
```

Session identification attributes supported by mpd:

```
N   Name
1   User-Name
5   NAS-Port
8   Framed-IP-Address
30  Called-Station-Id
31  Calling-Station-Id
44  Acct-Session-Id
50  Acct-Multi-Session-Id

    mpd VSA (12341)
12  mpd-link
13  mpd-bundle
14  mpd-iface
15  mpd-iface-index
```

Session parameters attributes supported by mpd:

```
N   Name
24  State
25  Class
27  Session-Timeout
28  Idle-Timeout
85  Acct-Interim-Interval

    mpd VSA (12341)
1   mpd-rule
2   mpd-pipe
3   mpd-queue
4   mpd-table
5   mpd-table-static
7   mpd-filter
8   mpd-limit
16  mpd-input-acct
17  mpd-output-acct
```

Received in CoA session parameters replace existing ones. If some parameter is not received, it keeps it's previous value for standard attributes, and getting cleared for mpd's VSAs.

Nots, that CoA request always restarts Session and Idle timers for matching interfaces, and restarts Accounting Update timer for matching links, if new value received.

4.12 Traffic accounting

Mpd supports several methods of traffic accounting:

4.12.1 NetFlow

NetFlow is a protocol for exporting aggregated traffic information with UDP packets flow. NetFlow is generated by `ng_netflow(4)` node. To enable NetFlow on a particular interface `netflow-in` or `netflow-out` options should be used. Best practice is to enable netflow on every interface in system in only one direction: incoming for "ingress" flow or outgoing for "egress" flow.

Note: Netflow 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. NetFlow v9 supprted only on FreeBSD 9.0-RELEASE and newer.

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

`set netflow version` *ver*
: Choose NetFlow version to export. Supported options are 5 or 9.

  Default is 5.

`set netflow mtu` *bytes*
: Set export interface MTU to build packets of specified size (NetFlow v9-specific).

  Default is 1500.

`set netflow template` *time packets*
: Modify time and packet timeouts to announce data flow templates (NetFlow v9-specific).

  Defaults are 600 and 500.

`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.

4.12.2 IP Accounting

IP Accounting accumulates traffic information in kernel buffer. This buffer can be periodically readed and flushed by some other program. IP Accounting implemented by `ng_ipacct` node. To enable IP Accounting on a particular interface `ipacct` options should be used.

4.12.3 Accounting Update

Mpd provides summary traffic statistics in every accounting Update/Stop call of radius-acct or ext-acct.

For radius-acct it provides such attributes:

`Acct-Input-Packets`
: 32bit input packets counter,

`Acct-Output-Packets`
: 32bit output packets counter,

`Acct-Input-Octets`
: lower 32bit of input bytes counter,

`Acct-Output-Octets`
: lower 32bit of output bytes counter,

`Acct-Input-Gigawords`
: higher 32bit of input bytes counter,

`Acct-Output-Gigawords`
: higher 32bit of output bytes counter.

For ext-acct it provides:

`ACCT_INPUT_PACKETS`
: 64bit input packets counter,

`ACCT_OUTPUT_PACKETS`
: 64bit output packets counter,

`ACCT_INPUT_OCTETS`
: 64bit input bytes counter,

`ACCT_OUTPUT_OCTETS`
: 64bit output bytes counter.

NOTE: as Accounting Update is implemented on a link layer it accounts traffic via single physical link (without link overhead). So it accounts traffic after compression, encryption and multilink were applied to it.

4.12.4 Typed Accounting Update

In addition to standard Accounting Update stats, mpd is able to provide accounting information for any specified traffic types. Traffic types may be defined using optional third component of mpd-limit's left part.

In accounting requests mpd can provide such attributes: for radius-acct:

`mpd-input-packets`
: traffic type : 64bit input packets counter,

`mpd-output-packets`
: traffic type : 64bit output packets counter,

`mpd-input-octets`
: traffic type : 64bit of input bytes counter,

`mpd-output-octets`
: traffic type : 64bit of output bytes counter,

for ext-acct:

`MPD_INPUT_PACKETS`
: traffic type : 64bit input packets counter,

`MPD_OUTPUT_PACKETS`
: traffic type : 64bit output packets counter,

`MPD_INPUT_OCTETS`
: traffic type : 64bit input bytes counter,

`MPD_OUTPUT_OCTETS`
: traffic type : 64bit output bytes counter.

NOTE: as Typed Accounting Update is implemented on interface layer instead of link as usual Accounting Update, same data will be reported for every link of multilink connection. Some special care should be taken by AAA to not account same traffic twice in multilink case.

For example, specification:

```
mpd-filter += "1#1=match dst net 10.0.0.0/8",
mpd-limit += "in#1#Local=flt1 pass",
mpd-limit += "in#2#Biz=all shape 64000 4000",
```

, or for ext-auth:

```
MPD_FILTER:1#1=match dst net 10.0.0.0/8
MPD_LIMIT:in#1#Local=flt1 pass
MPD_LIMIT:in#2#Biz=all shape 64000 4000
```

will instruct mpd to account traffic of two types: "Local" and "Biz". Traffic going from client to the 10.0.0.0/8 network will be accounted as "Local", all other incoming traffic after shaping will be accounted as "Biz". In accounting request mpd will provide to RADIUS something like:

```
mpd-input-packets = Local:213213
mpd-input-octets = Local:32132132
mpd-input-packets = Biz:21321
mpd-input-octets = Biz:3213213
```

, and for ext-acct:

```
ACCT_INPUT_PACKETS:Local:213213
ACCT_INPUT_OCTETS:Local:32132132
ACCT_INPUT_PACKETS:Biz:21321
ACCT_INPUT_OCTETS:Biz:3213213
```

To simplify AAA integration mpd able to return specified traffic types using standatd RADIUS attributes. Required traffic type can be specified using mpd-input-acct/mpd-output-acct attributes.

4.13 IP address pools

Mpd implements dynamic IP addresses pool management. When user is authenticated and auth backend have not provided IP address, it can be taken from pool. To make it work you should create one or more address pools with 'set ippool add ...' command and define which pool should be used with 'set ipcp ranges ... ippool ...' command, radius-auth Framed-Pool attribute or ext-auth FRAMED_POOL attribute.

`set ippool add` *pool first last*
: This command creates new IP address pool if it not exists and adds specified address range to it.

4.14 Network Address Translation (NAT)

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 red-addr` *alias_addr local_addr*
: Redirect traffic for public (alias) IP address to a machine on the local network.

`set nat red-proto` *proto alias_addr local_addr* [*remote_addr*]
: Redirect traffic of specified protocol for public (alias) IP address (optionally from specified remote address) to a machine on the local network.

`set nat red-port` *proto alias_addr alias_port local_addr local_port* [*remote_addr remote_port*]
: Redirect incoming connections arriving to given public (alias) address and port, (optionally from specified remote address) to a host and port of a machine on the local network.

`unset nat red-addr` *alias_addr local_addr*
: Deletes specific *set nat red-addr ...* rule.

`unset nat red-proto` *proto alias_addr local_addr* [*remote_addr*]
: Deletes specific *set nat red-proto ...* rule.

`unset nat red-port` *proto alias_addr alias_port local_addr local_port* [*remote_addr remote_port*]
: Deletes specific *set nat red-port ...* rule.

`set nat enable` *option ...*

`set nat disable` *option ...*

: Enable and disable nat options.

The following options are supported:

`log`
: Enable statistics logging.

: Default is disable.

`incoming`
: Allow unknown incoming packets to pass.

: Default is enable.

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

: Default is enable.

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

: Default is disable.

4.15 Control console

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 self` *ip* [*port*]
: 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.

The default is ’127.0.0.1 5005’.

`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 current console.

  The default is enable for stdout and disable for the rest.

`auth`
: This options enables autorized login to console. This is a read-only value.

  The default is enable.

4.16 Web server

Mpd provides HTTP interface for monitoring and control purposes. 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 self` *ip* [*port*]
: 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.

  The default is ’127.0.0.1 5006’.

`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.

  The default is enable.

You can send any set of command allowed by privileges via WEB server for mpd infrastructure integration. Depending on URL used mpd supports two response formats: text/html (/cmd?command1&...) and text/plain (/bincmd?command1&...). Also you can see output ‘show summary‘ command in JSON format, typing ‘/json‘ in URL.

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/cuau0`.

`set modem var` *$variable string*
: Pre-sets the chat script variable *$variable* to contain the string *string*. See *chat* (section 6. , page 72) 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.

The following options are supported:

`resolve-once`
: Enables resolving peer address only once, on startup, or on manual typing in CLI.

  The default is enable.

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.

The following options are supported:

`resolve-once`
: Enables resolving peer address only once, on startup, or on manual typing in CLI.

  The default is enable.

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 a 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:

`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 allows to make an outgoing phone call (by PNS) via a remote access server (PAC), as well as in more common case forward an incoming phone 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.

  The default is enable.

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

  The default is disable.

`windowing`
: Enables the windowing mechanism specified by the protocol. Disabling this will cause Mpd to violate the protocol, possibly confusing some PPTP peers, but usually results in better performance. The windowing mechanism is a design error in the PPTP protocol; L2TP, the successor to PPTP, removes it.

  The default is disable.

`resolve-once`
: Enables resolving peer address only once, on startup, or on manual typing in CLI.

  The default is enable.

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 a 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 pmask` *mask*
: Check peer hostname, related to wildcard *mask*. Wildcard can contain any shell-like mask, such as "*.myhost.com" Peer can set self hostname with *set l2tp hostname* command.

`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:

`outcall`
Inside L2TP tunnel, each individual PPP connection (there may be several), is initiated as either an incoming or an outgoing call. This allows to make an outgoing phone call (by LNS) via a remote access server (LAC), as well as in more common case forward an incoming phone 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).

The default is disable.

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

The default is disable.

`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.

The default is disable.

`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.

The default is enable.

`resolve-once`
Enables resolving peer address only once, on startup, or on manual typing in CLI.

The default is enable.

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**.

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.

DSL Forum vendor-specific Circuit-ID/Remote-ID PPPoE tags extraction supported in server mode.

`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.

  Since FreeBSD 6.3-STABLE/7.0-STABLE of 2008-01-30 it is possible to use "AC-Name\Service-Name" syntax to request specified access concentrator for outgoing connection.

  The default is "*".

`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 mac-format` *format*
: Set RADIUS server attribute 31 MAC format as:

```
unformatted    aabbccddeeff
unix-like      aa:bb:cc:dd:ee:ff
cisco-like     aabb.ccdd.eeff
ietf           aa-bb-cc-dd-ee-ff
```

  The default is "unformatted".

`set pppoe max-payload` *size*
: Set PPP-Max-Payload PPPoE tag (RFC4638). This option works when mpd5 acts as PPPoE client only. This option takes precedence over both `set link mtu` and `set link mru`options, if PPPoE server agrees with our setting.

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.

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* (section 5.1 , page 65) 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.

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

My configuration doesn't seem to be working as expected.
Make sure your mpd.conf file 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 (usually 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`.

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.

L2TP auth proxying
: Currently MPD acting as LAC requires parameter renegotiation on every tunnel hop. It would be good to implement auth proxying to speedup negotiation.

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.

  Usually 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 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 fore. 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 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 2118 Microsoft Point-To-Point Compression (MPPC) Protocol

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 2868 Attributes for Tunnel Protocol Support

RFC 2869 RADIUS Extensions

RFC 3576 Dynamic Authorization Extensions to RADIUS

RFC 3579 RADIUS EAP

9.2 Credits

- EAP, RADIUS enhancements, basework for Mpd4 was sponsored by SURFnet.
- L2TP enhancements were sponsored by Scom.ca Internet Services Inc.
- Built-in RADIUS server support was sponsored by JSC 'Ufanet'.
- Special thanks to JSC 'Optima-Telecom' for a long-time support and production testing.

CONTENTS