# Address Family Transition Router Manual

Internet System Consortium
950 Charter Street
Redwood City, California
USA
http://www.isc.org/

# Contents

# Chapter 1

# INTRODUCTION

ISC AFTR implements a Dual-Stack Lite (DS-Lite) Address Family Transition Router (AFTR), as described in `draft-ietf-softwire-dual-stack-lite-06.txt`. This technology allows end-users with IPv4-only hosts or IPv4-only applications to communicate with IPv4 peers over an IPv6-only network.

A DS-Lite deployment includes at least one AFTR in the ISP's network core, and one Basic Bridging BroadBand element (B4) at each customer premises.

## 1.1 Work in Progress

DS-Lite is in the process of being standardized by the Softwire working group of the IETF. ISC AFTR actively tracks the current specification, but users should be aware that there may be changes to the specification before it is finalized as an RFC. As such, this should be considered a work in progress, for testing and experimentation only.

# Chapter 2

# GETTING STARTED

This section provides a "quick start" to using ISC AFTR in the simplest configuration. For full build and configuration details, see chapter 3.

## 2.1 Setting Up AFTR

### 2.1.1 System Requirements for AFTR

- OS: Linux or FreeBSD. Kernel must be built with IPv6 and tun(4).

  Linux kernel version must be greater than 2.6.26, to correct a small-packet-drop problem in `tunnel46_rcv()`.
- CPU: No special requirement. Note that performance is bound to the kernel/user context switch latency, so processor speed is the single biggest determiner of performance.
- Memory: No special requirement.

### 2.1.2 Building AFTR

Unpack the distribution.

```
tar zxvpf aftr-1.1.tar.gz
```

This creates a a directory named `aftr-1.1`, which we refer to as `$src_path` hereafter.

```
cd $src_path
./configure
make
```

This creates an executable file `aftr`, which is the AFTR daemon program. It is expected to be run in the same directory as the configuration file `aftr.conf` and the script file `aftr-script` (there is no **`make install`** step).

### 2.1.3 Minimal AFTR Configuration

An example `aftr.conf` file is located in the `conf` directory.

The only required parameters are those in section 1 of the config file, and are briefly described here. The full set of configuration commands is described in section 3.3.

**`address endpoint 2001::1`**

This is the IPv6 address of the AFTR. Specifically, it is the endpoint of the IPv6 tunnel between the B4 and the AFTR.

This address is associated with the tunnel interface, but is not explicitly assigned to it; instead, the `aftr-start()` portion of the script file creates a covering route to the tunnel interface.

**`address icmp 192.0.0.1`**

This is a global IPv4 address used as the source for ICMP errors sent from the AFTR to the Internet. The actual address doesn't matter, as no one should try to respond to an ICMP packet.

**`acl6 2001::/48`**

This is an Access Control List for IPv6 traffic from the B4 to the AFTR. It is the IPv6 prefix that encloses the portion of the provider's network that is served by this AFTR. If the AFTR serves customers served by multiple disjoint IPv6 prefixes, there can be multiple **acl6** commands.

**`pool 198.18.200.1`**

This is a global IPv4 address in the service provider's network, which will be used as the NATted address for packets sent to the Internet.

Multiple pool addresses may be defined, and restrictions may be placed on port ranges to use for NAT bindings.

#### 2.1.4 AFTR Script

The **aftr** executable requires a startup/shutdown script. When **aftr** starts up, it calls the script's `start` function to bring up the tunnel interface, and set up routes to the interface.

When **aftr** shuts down, it calls the script's `stop` function to bring down the tunnel interface, and remove all routes to the interface.

By default, the script file is named `aftr-script`, and resides in the same directory as the `aftr` executable file.

The `conf` directory contains example `aftr-script` files for Linux and FreeBSD.

#### 2.1.5 Running AFTR

First, make sure your interfaces are configured correctly, and IPv4 and IPv6 forwarding are enabled. On Linux, you may also want to disable **netfilter** for performance reasons.

Note: Linux netfilter tables need to be flushed ('**`iptables -F`**' and '**`ip6tables -F`**') explicitly.

To start AFTR, run the `aftr` executable in the same directory as the `aftr.conf` configuration file and the `aftr-script` script file. For other startup options, see section 3.2.

AFTR normally starts as a daemon process. To access the control interface, **telnet** to localhost port 1015. AFTR can also be started in foreground mode, which gives immediate access to the control interface. For control commands, see section 3.4.

## 2.2 Setting Up B4

The B4 is the customer-side DS-Lite tunnel initiator. In the most common use case, it is a home gateway, also referred to as a CPE (Customer Premises Equipment).

For testing and demonstration purposes, we have used home gateways (e.g., Linksys WRT54GL) running OpenWrt, a Linux distribution tailored for home gateways and related devices. B4 functionality can also be built into general-purpose computers, and has been demonstrated in FreeBSD and Ubuntu Linux. In all of these cases, the important thing to keep in mind is that the B4 device is the IPv4 default router for all hosts behind it.

### 2.2.1 System Requirements for B4

There are no special requirements above what one might expect for a home gateway. The only added functionality that makes it a B4 is to set up an IPv4-in-IPv6 tunnel, and to encapsulate/decapsulate all IPv4 traffic to/from that tunnel; and all home gateways that can run OpenWrt can do that.

### 2.2.2 Building B4

Note: More information, and prebuilt images for WRT54G devices, can be found at `http://www.kangaroo.comcast.net`

The following instructions are for building an OpenWrt image from sources.

1. Get OpenWrt sources from `https://dev.openwrt.org`
2. Copy the contents of the `conf/b4-openwrt` directory to a new directory named openwrt/*version*/package/dhcp4
3. Go to the openwrt/*version* directory, run **make menuconfig**, and make the following selections:
   - Select 2.6 kernel: Target System > Broadcom BCM947xx/953xx (or your target architecture)
   - Deselect busybox udhcp client: Base system > busybox > Configuration > Networking Utilities
   - Select dhcp4-client and dhcpv6: Network > isc-dhcp
   - Select the non-busybox version of **ip**: Network > ip
   - Select ip6-tunnel: Kernel modules > Network Support > kmod-ip6-tunnel

## 2.3 Other Setup

### 2.3.1 DHCPv6 Configuration

`draft-ietf-softwire-dual-stack-lite-06.txt` says:

> In order to configure the IPv4-in-IPv6 tunnel, the B4 element needs the IPv6 address of the AFTR element. This IPv6 address can be configured using a variety of methods, ranging from an out-of-band mechanism, manual configuration or a variety of DHCPv6 options.
>
> In order to guarantee interoperability, a B4 element SHOULD implement the DHCPv6 option defined in [I-D.ietf-softwire-ds-lite-tunnel-option].

The DHCP server does not have to run on the same computer as the AFTR, but it must be reachable via normal DHCP request channels from the B4, and it must be configured with the AFTR address.

- On the server, add something like the following to `dhcpd6.conf`:

  ```
  option dhcp6.dslite code 54 = ip6-address;
  option dhcp6.dslite 2001::1;
  ```

  NOTE: For testing, we use an unassigned DHCPv6 option code. DO NOT use this option code in production, as it is likely to change when the draft reaches RFC status.

  The IPv6 address must be the same as the **`address endpoint`** option in `aftr.conf`.

- On the B4, `dhclient6.conf` contains the following to request the option:

  ```
  option dhcp6.dslite code 54 = ip6-address;
  also request dhcp6.dslite;
  ```

  If you are using the supplied OpenWrt dhcp4 package, `dhclient6.conf` already contains these lines, and `dhclient-script` contains a few extra lines to set up the tunnel and create a default IPv4 route to it.

### 2.3.2 DNS Configuration

`draft-ietf-softwire-dual-stack-lite-06.txt` recommends configuring the B4 with a DNS proxy resolver, which will forward queries to an external recursive server over IPv6 (this server may be co-located on the AFTR box).

To configure the B4 with an upstream resolver address, add something like the following to dhcpd6.conf:

```
option dhcp6.name-servers 3ffe:501:ffff:100:200:ff:fe00:3f3e;
```

Note: if the B4 uses **dnsmasq** as a DNS proxy, then the used version should be checked. Only recent versions are RFC 5625 compliant; in particular the EDNS0 UDP size can be limited to 1280 bytes instead of the recommended 4096 bytes.

It is easy to fix this last point by configuration:

- Add in `/etc/config/dhcp` in the dnsmasq section the line:

  ```
  option ednspacket_max 4096
  ```

- If **dnsmasq** is launched directly, add these arguments:

  ```
  -P 4096
  ```

# Chapter 3

# AFTR MANUAL

## 3.1 Compile Flags

Here is the list of configuration flags (i.e., `CFLAGS`):

- `AFTRCONFIG`: config file path (default `aftr.conf`)
- `AFTRSCRIPT`: script file path (default `./aftr-script`)
- `AFTRDEVICE`: name of the interface/device (default `tun0`)
- `AFTRPORT`: port for TCP control channels (default 1015)
- `AFTRFACILITY`: syslog facility (default `LOG_LOCAL5`)
- `AFTRLOGOPTION`: openlog option (default `LOG_NDELAY`)
- `TRACE_NAT`: enable tracing of NAT entry creation/deletion (default is `undef`, i.e., only tunnels and buckets are traced)
- `NOPRIVACY`: trace all addresses and ports in NAT entry tracing (default is `undef`)
- `SIGNSHDR`: define it to add a signature header in structures (default is `undef`)
- `SIZES`: define it to print sizes of principal data structures (default is `undef`)
- `USE_TUN_PI`: use the tun_pi struct in tun interface/device I/O (required on some platforms, including RedHat and CentOS v5, for IPv6 support)
- `notyet`: some unfinished and arguable features (`undef` of course)

## 3.2 Command Line Options

Included inline aftr (8)

### 3.2.1 aftr

aftr — Address Family Transition Router

#### Synopsis

`aftr [-g] [-t] [-c config-file] [-d device-name] [-p port-number] [-s script-file] [-u socket-name]`

#### OPTIONS

**-g**
By default the aftr process becomes a daemon, `-g` keeps it in foreground with logging to stderr.

**-t**
`-t` can be used to check a configuration file.

**-c *config-file***
The **aftr** daemon requires a configuration file. By default it is named `aftr.conf`, and is located in `$src_path`. The `AFTRCONFIG` environment variable and the `-c` argument give an alternate path. A sample configuration file is provided in `$src_path/conf/aftr.conf` (OS independent).

**-d *device-name***
Linux: The aftr process opens `/dev/net/tun` and set the name of the interface to the `AFTRDEVICE` environment variable or the `-d` command line argument value or by default 'tun0'.

FreeBSD: The aftr process opens `/dev/tunXXX` from the `AFTRDEVICE` environment variable or the `-d` command or by default `/dev/tun0`. The 'auto' value uses the first free `/dev/tunXXX` device.

The tunnel interface/device specification can be a full path (`/dev/...`), a relative name or a number.

**-p *port-number***
Use the *port-number* for TCP control channels. Default is 1015.

**-s *script-file***
The **aftr** daemon executes a shell script file with `start` on invocation. This is named by default `aftr-script` and located in `$src_path`. The `AFTRSCRIPT` environment variable and the `-s` argument give an alternate path. This file could be even empty, but must exist.

The **aftr** daemon will eventually execute the shell script file with the `stop` argument before it exits.

The `conf` directory provides examples (including the configuration files used in our testbeds). `aftr-script.freebsd` variant is for a FreeBSD based AFTR.

**-u *socket-name***
As an alternative to TCP over IPv4 and IPv6 with localhost control channels, the **aftr** process can accept PF_UNIX stream socket control channel on the *socket-name*.

#### SEE ALSO

aftr.conf(5), aftr.commands(5)

#### AUTHOR

Internet Systems Consortium

## 3.3 Configuration File

Included inline aftr-conf (5)

### 3.3.1 `aftr.conf`

`aftr.conf` — configuration file for aftr

#### Synopsis

`aftr.conf`

#### DESCRIPTION

The **aftr** daemon requires a configuration file. By default it is named `aftr.conf`, and is located in `$src_path`. The `AFTRCONFIG` environment variable and the `-c` argument give an alternate path. Sample configuration files are provided in `$src_path/conf/aftr.conf` (OS independent).

The configuration file consists of a set of one-line configuration commands. Commands are not case sensitive. Any line beginning with '#' or whitespace is ignored as a comment.

Configuration and interactive commands belong to sections:

- section zero is for global parameters which must be defined before anything else when they are not kept to their default values, for instance **defmtu**.
- section one is for required parameters, for instance **acl6**.
- section two is for reloadable parameters, for instance **nat**.
- interactive only commands are in the section three.

Only the section one commands are required; reasonable defaults are provided for all other configuration parameters. See `conf/aftr.conf` for an example of a minimal configuration file.

#### GLOBAL CONFIGURATION COMMANDS

**autotunnel on|off**
Alias of **`default tunnel auto on|off`**.

**bucket tcp|udp|icmp size** ***`size`***
Specifies the bucket size. Compile time options are `[TCP|UDP|ICMP]BUCKSZ`, default values are: `TCPBUCKSZ` 10, `UDPBUCKSZ` 8, `ICMPBUCKSZ` 3. Minimum is 0 (excluded) and maximum 255.

**decay 1|5|15** ***`decay`***
Specifies decay values for 1, 5 and 15 mn rates. Compile time options are `DECAY{1,5,15}`, default values are: `DECAY1` exp(-1/60), `DECAY5` exp(-1/300), `DECAY15` exp(-1/900). Minimum is 0.0 and maximum 1.0.

**default fragment equal on|off**
Enables or disables equalizing the length of IPv6 fragments. Default is off.

**default fragment lifetime** ***lifetime***
Specifies the lifetime of fragments in reassembly queues. Compile time option is `FRAG_LIFETIME`, default value is 30 seconds. Minimum is 0 (excluded) and maximum 1200.

**default fragment ipv6|in|out maxcount** ***maxcount***
Maximum number of entries in reassembly queues ('in' is IPv4 from clients to the Internet, 'out' is IPv4 from the Internet to clients). Compile time options are `FRAG{6,IN,OUT}_MAXCNT`, default values are 1024. Minimum is 0 (included so it is possible to disable reassembly), maximum is 16535.

**default hold lifetime** ***lifetime***
Specifies the lifetime of expired NAT entries in the hold queue. Compile time option is `HOLD_LIFETIME`, default value is 120 seconds. Minimum is 0 (included), maximum is 600.

**default nat lifetime tcp|closed|udp|icmp|retrans** ***lifetime***
Specifies the lifetime of dynamic NAT entries ('closed' is for closed TCP sessions, 'retrans' is used for response not yet received). Compile time options are `[TCP|CLOSED_TCP|UDP|ICMP|RETRANS]_LIFETIME`, default values are TCP (600), closed TCP (120, aka 2*MSL), UDP (300), ICMP (30), retrans (10). Minimum is 0 (excluded), maximum 36000 (10 hours).

**default pool tcp|udp|echo** ***min-max***
Specifies the default port (or id for icmp echo) ranges for pools. Compile time options are `[TCP|UDP]_[MIN|MAX]PORT`, `ICMP_[MIN|MAX]ID`, default values are `TCP_MINPORT` 2048, `UDP_MINPORT` 512, `ICMP_MINID` 0, `TCP_MAXPORT` 65535, `UDP_MAXPORT` 65535, `ICMP_MAXID` 65535. Minimum is 1 (0 for ICMP), maximum 63535.

**default private** ***IPv4_prefix/prefix_length***
Add a private prefix to IPv4 ACLs. The default is RFC 1918 prefixes and the 192.0.0.0/29 from the ds-lite draft.

**default tunnel auto on|off**
Enables or disables on-the-fly tunnel creation. Default is on.

**default tunnel mss on|off**
This enables or disables TCP MSS patching on packets going from and to tunnels. Can be overridden by per-tunnel configuration. If any tunnels are explicitly configured, this must be specified before them. Default is off.

**default tunnel mtu** ***mtu***
Specifies *mtu* as the default IPv6 MTU of tunnels. Can be overridden by per-tunnel configuration.

**default tunnel toobig on|off|strict**
This specifies the policy for packets from the Internet which are too big (i.e., they don't fit in one IPv6 encapsulating packet) and are marked as 'don't fragment'. 'On' means a ICMPv4 packet too big error is returned to the source, 'off' the packet just go through, and 'strict' the packet is dropped with a ICMPv4 error. Default is on (i.e., the packet is encapsulated into some IPv6 fragments and a ICMP error is returned for path MTU determination).

**default tunnel fragment ipv6|ipv4 maxcount** ***maxcount***
Specifies the maximum number of reassembly queue entries per tunnel. Compile time options are `FRAGTN[46]_MAXCNT`, default values are `FRAGTN6_MAXCNT` 16, `FRAGTN4_MAXCNT` 64. Mininum is 0 (included for reassembly disable), maximum is 255.

**default tunnel nat tcp|udp|icmp maxcount** ***maxcount***
Specifies the maximum number of NAT entries per tunnel. Compile time options are `[TCP|UDP|ICMP]_MAXTNATCNT`, default values are `TCP_MAXNATCNT` 2000, `UDP_MAXNATCNT` 200, `ICMP_MAXNATCNT` 50. Minimum is 0 (included), maximum is 65535.

**default tunnel nat tcp|udp|icmp rate** ***limit***
Specifies the maximum rate of dynamic NAT creation per second. Compile time options are `[TCP|UDP|ICMP]_MAXTNATRT`, default values are `TCP_MAXNATRT` 50, `UDP_MAXNATRT` 20, `ICMP_MAXNATRT` 5. Minimum is 0 (included), maximum 255.

**delete private***IPv4_address*
This removes the IPv4 private prefix with the IPv4 address. It is an error to have no private prefixes.

**defmss on|off**
Alias of **`default tunnel mss on|off`**.

**defmtu** ***mtu***
Alias of **`default tunnel mtu`** ***mtu***.

**deftoobig on|off|strict**
Alias of **`default tunnel toobig on|off|strict`**.

**eqfrag on|off**
Alias of **`default fragment equal on|off`**.

**quantum** ***quantum***
Specifies the number of packets dealt with in one main loop round (i.e., the size of a slice of work). Compile time option is `QUANTUM`, default value is 20. Minimum is 2 (included), maximum is 255.

#### REQUIRED CONFIGURATION COMMANDS

**address endpoint** ***IPv6_address***
*`IPv6_address`* is the AFTR endpoint address of the Softwire tunnels. If the DHCPv6 ds-lite option is used, this address must match the advertised address.

It is a required command: it absolutely must be present in the `aftr.conf` file; the **aftr** daemon will not start without it.

**address icmp** ***IPv4_address***
*`IPv4_address`* is a global IPv4 address used as the source for ICMP errors sent back to the Internet (i.e., the ICMPv4 errors will look like returned from an intermediate router that has this address). It is a required command.

**pool** ***IPv4_address*** **[tcp|udp|echo** ***min-max*****]**
This specifies a global IPv4 address that will be used as the source address of NAT'ed packets sent to the Internet. Multiple global addresses can be specified, at least one is required.

The optional part limits the port (or id) range used for the protocol with the global IPv4 address in dynamical bindings (i.e., not static or A+P bindings which can use the reserved ports outside the range).

**acl6** ***IPv6_prefix/prefix_length***
This adds an (accept) entry in the IPv6 ACL. Note for a regular IPv6 packet the ACL is checked only when no tunnel was found, and the default is 'deny all', so at least one acl6 entry in the configuration file is required.

#### RELOADABLE CONFIGURATION COMMANDS

**tunnel** ***IPv6_remote*** **[*IPv4_src*]**
This specifies an IPv4-in-IPv6 tunnel configuration. *IPv6_remote* is the remote (ds-lite client) IPv6 address of the tunnel. Either the tunnel is associated with a source address in a round robin way or it is associated to the specified *IPv4_src*.

**nat** ***IPv6_remote*** **tcp|udp** ***IPv4_src port_src IPv4_new port_new***
This defines a static binding/NAT entry for the client behind the tunnel at *IPv6_remote*. **_src* are the source IPv4 address and port at the tunnel side of the NAT, **_new* are the source IPv4 address and port at the Internet side of the NAT. *IPv4_new* should be a reserved source NAT address, *port_new* must not be inside a dynamic port range.

**prr** ***IPv6_remote*** **tcp|udp** ***IPv4 port***
This defines a Port-Range Router/A+P null NAT entry for the client behind the tunnel at *IPv6_remote*. *IPv4* and *port* are the source IPv4 address and port at the tunnel side of the NAT. They stay unchanged both ways: this entry is used to check authorization and perform port routing.

**nonat** ***IPv6_remote IPv4/prefix_length***
This defines a No-NAT tunnel for the client behind the tunnel at *IPv6_remote* and the prefix *IPv4/prefix_length*. No translation is performed for matching packets.

**mss** ***IPv6_remote*** **on|off**
This enables or disables TCP MSS patching on packets going from and to the tunnel of *IPv6_remote*. Default is off.

**mtu** ***IPv6_remote mtu***
This changes the IPv6 MTU of the tunnel of *IPv6_remote* to *mtu*.

**toobig** ***IPv6_remote*** **on|off|strict**
Per-tunnel configuration of the too big policy.

**debug set [*level*]**
Specifies the debug level. Default is 0. If set to non 0, verbose log messages will be dumped to stderr. The higher the level is, the noiser the logs are. At present, the meaningful levels are 1 (log tunnel creation), 3 (log packet reads and writes), and 10 (function entry tracing). If the level is omitted, it is set to 1.

**try tunnel** ***IPv6_remote*** **[*IPv4_src*]**
Create when it doesn't already exist an IPv4-in-IPv6 tunnel, returns in all cases the description of the tunnel entry. This command should be used by tools managing temporary port forwarding. *IPv6_remote* must be acceptable for IPv6 ACLs.

**try nat** ***IPv6_remote*** **tcp|udp** ***IPv4_src port_src IPv4_new port_new***
Create when it doesn't already exist a static binding/NAT entry. This command should be used by tools managing temporary port forwarding. The tunnel must exist.

#### SEE ALSO

aftr(8), aftr.commands(5)

#### AUTHOR

Internet Systems Consortium

## 3.4 Interactive Commands

Included inline aftr-commands (5)

### 3.4.1 `aftr.commands`

`aftr.command` — interactive commands for aftr

#### Synopsis

`aftr.commands`

#### DESCRIPTION

The **aftr** daemon runs in the background. After it starts, it can be controlled interactively from a control channel (aka. a session).

All of the reloadable configuration commands can be allowed to run from the command line, to add or change configuration. In addition, the following commands can be run interactively.

#### INTERACTIVE COMMANDS

**abort**
Call abort(3) to create a core file. Please try to use it only on forked processes.

**echo** ***xxx***
Echo the command. This can be used for an external tool to synchronize with the AFTR daemon.

**fork**
Fork the **aftr** process. In the parent the current session is closed (so after this command you'll talk only to the child) and other activities, including packet forwarding, are continued. In the child all file descriptors at the exception of the current session are closed.

This command should be used before to execution an expensive and atomic operation like list commands or some debug commands, and of course the abort command.

**help [all]**
List available or all commands.

**kill**
Orderly kill the **aftr** process.

**load** ***file***
Redirect the input of the current session from the content of the file. This is done in an atomic way (i.e., there is no other activity during the operation) but exists if a command fails.

**quit**
Obsolete, use **session close** (for closing the current session) or **kill** (for killing the process).

**reboot**
Reboot the whole process.

**reload**
Reload the section two part of the config file. This is sliced with the packet forwarding, but not with session reading (so you can't execute a command until reload is finished).

The reload process uses a generation system: static NAT, PRR/A+P and no-NAT entries in the reloaded file are put in the next generation. If the reload succeeds, global entries in older generations are garbaged collected, if it fails new generation entries are backtracked to the previous generation. Garbage collection and backtracking are sliced with the packet forwarding, another reload command is forbidden until they finish so a reload flushes the input buffer of the current session.

**show dropped|stat**
Aliases of **debug dropped** and **debug stat**, display dropped packet and general statistics.

#### DEBUG COMMANDS

**noop**
Returns `LOG: alive`.

**debug check [nat|nonat|pool|session|tunnel]**
Performs some sanity checks on structures. Reserved to expert usage on a forked process (or better core file debugged with gdb). Note it uses recusive deep structure walking so can eat a lot of stack.

**debug disable [clear]**
Disable per-tunnel debug counters. Optionally clear them.

**debug dropped**
This displays the dropped packet statistics with reasons.

**debug enable *addr***
Enable per-tunnel debug counters for the tunnel with *addr* remote IPv6 address. Note the counters can be incremented only when the involved tunnel is known, for instance, only after reassembly.

**debug fragment IPv6|in|out**
This displays the list of IPv4 or IPv6 fragments awaiting reassembly.

**debug fragment *addr***
This displays information about a single fragment or fragment chain. *add*> is the memory address of the fragment structure (from a previous **debug fragment** command).

**debug hash**
This displays some statistics about the various hash tables (fragment, nat, and tunnel).

**debug nat**
This displays some information about the nat hash table and entry table.

**debug nat *addr***
This displays detailed information about a single nat binding. *addr* is the memory address of the nat structure (from a previous **debug nat** command).

**debug nonat**
This displays the list of no-nat tunnel entries.

**debug pool**
This displays the global IPv4 addresses that will be used for NAT mapping.

**debug session**
This displays the control channel session types with the number of active sessions.

**debug stat**
This displays some general statistics about packets in and out. If per-tunnel debug counters are enable, displays them.

**debug tunnel**
This displays some information about the tunnel table.

**debug tunnel *IPv6_remote***
This displays some information about a single tunnel.

#### DELETE COMMANDS

**delete acl6 *IPv6_address***
This removes the IPv6 ACL entry with the IPv6 address.

**delete nat *IPv6_remote* tcp|udp *IPv4 port***
This removes a static or dynamic NAT binding.

**delete nonat *IPv6_remote***
This removes a no-nat tunnel entry.

**delete private *IPv4_address***
Look at zone zero configuration commands.

**delete prr *IPv6_remote* tcp|udp *IPv4 port***
This removes a Port-Range Router/A+P null NAT binding.

**delete tunnel *IPv6_remote***
This removes a tunnel and all NAT bindings associated with it.

#### LIST COMMANDS

**list acl6**
List IPv6 ACLs.

**list default**
List all the default values which can be set by a 'default'/'global' command.

**list nat [conf|static|prr|dynamic|all|global]**
List the NAT entries in the configuration file format. Default is to list only the configured ('conf') NAT entries. 'global' lists the the configured global (i.e., not by a session) active (i.e., not to be garbaged collected after a reload) NAT entries.

**list nonat**
List all the No-NAT tunnel entries in the configuration file format.

**list pool**
List the NATted source addresses with current port ranges in the configuration file format.

**list session [*name*|*generation*]**
List the static NAT, PRR/A+P and no-NAT entries created by the current session or the session with *name* or with *generation* (note these entries will be flushed when the session will be closed so this command can be used to get them in order to include them in the config).

**list tunnel**
List the tunnel entries in the configuration file format, including specific MTU (if different from the default MTU).

#### SESSION COMMANDS

These commands deal directly with sessions (aka. control channels).

**session close [*name*|*generation*]**
Close the current or designed session. Delete all the static NAT, PRR/A+P and no-NAT entries created by the current session and which were not promoted to global/permanent entries by a reload.

**session config on|off**
Enable/disable the section two configuration commands. By default configuration commands must go to the config file.

**session log on|off**
Log errors or don't for the current session. Default is on.

**session name [*name*]**
Display or set the name of the current session. The stdio initial session is statically named 'tty'.

**session notify on|off**
Log tunnel removal or don't to the current session. Default is off.

#### SEE ALSO

aftr(8), aftr.conf(5)

#### AUTHOR

Internet Systems Consortium

## 3.5 Command Summary

| Name | Section | Syntax |
|---|---|---|
| abort | interactive | |
| acl6 | one or two | *IPv6*/*prefix_length* |
| address | one | endpoint *IPv6*\|icmp *IPv4* |
| autotunnel | zero | on\|off |
| bucket | zero | tcp\|udp\|icmp size *size* |
| debug | >= two | set\|enable\|...\|tunnel |
| decay | zero | 1\|5\|15 *decay* |
| default | zero | fragment\|hold\|...\|tunnel |
| defmss | zero | on\|off |
| defmtu | zero | *mtu* |
| deftoobig | zero | on\|off\|strict |
| delete | == add | acl6\|nat\|nonat\|private\|prr\|tunnel |
| echo | interactive | *xxx* |
| eqfrag | zero | on\|off |
| fork | interactive | |
| help | interactive | [all] |
| kill | interactive | |
| list | interactive | nat\|nonat\|pool\|tunnel |
| load | interactive | *file* |
| mss | >= two | *IPv6* on\|off |
| mtu | >= two | *IPv6 mtu* |
| nat | two | *IPv6* tcp\|udp *IPv4_src* ... |
| nonat | two | *IPv6 IPv4*/*prefix_length* |
| noop | interactive | |
| pool | one | *IPv4* [tcp\|udp\|echo *min-max*] |
| private | zero | *IPv4*/*prefix_length* |
| prr | two | *IPv6* tcp\|udp *IPv4 port* |
| quantum | zero | *quantum* |
| reboot | interactive | |
| reload | interactive | |
| session | interactive | close\|config\|log\|name\|notify |
| show | interactive | dropped\|stat |
| toobig | >= two | *IPv6* on\|off\|strict |
| try | two | tunnel ... \| nat *IPv6* tcp\|udp *IPv4_src* ... |
| tunnel | two | *IPv6* [*IPv4*] |

# Chapter 4

# Managing AFTR

## 4.1 Syslog

Errors, debug messages, traces, etc, are logged through syslog with aftr as the program name.

The default facility is `LOG_LOCAL5` (can be changed at compile time by setting `AFTRFACILITY`), the default `openlog()` option is `LOG_NDELAY` (can be changed at compile time by setting `AFTRLOGOPTION`, for instance to add `LOG_PID`). Levels are:

- critical errors (i.e., the process must be rebooted) to `LOG_CRIT`
- error conditions (i.e., bad packets, not critical memory allocation failures, bad commands, etc) to `LOG_ERR`
- warnings to `LOG_WARNING`
- informational messages (including I/O logs) to `LOG_INFO`
- debug messages (cf. debug set xxx) to `LOG_DEBUG`
- trace messages (see next section) to `LOG_NOTICE`

### 4.1.1 Trace Logging

Trace messages are:

- tunnel add|del *client_IPv6*
- *seconds* bucket *client_IPv6 natted_IPv4* tcp|udp [#*port*]+

If TRACE_NAT was defined at compile time (default is undefined):
if NOPRIVACY is kept undefined:

- *seconds* nat add|del *client_IPv6* tcp|udp *natted_IPv4 port*

if NOPRIVACY is defined:

- *seconds* nat add|del *client_IPv6* tcp|udp *client_IPv4 client_port natted_IPv4 natted_port destination_IPv4 destination_port*

## 4.2 Security

The **aftr** process needs the root privilege to open the tunnel interface/device. The TCP over IPv4/IPv6 control channels are bound to localhost so are limited to the local node. There are many tools which provide a secure connection forwarding, for instance **ssh -L**. The PF_UNIX control channel relies on standard file system permissions (cf. **umask**), it should be used for finer control than node access.

The source address of encapsulated IPv4 in IPv6 packets must be a private address. The list of IPv4 private prefixes is initialized to RFC 1918 prefixes and the unpublished I-D, it is manageable by zone zero commands. IPv6 ACLs filter incoming IPv6 packets.

The **try** commands are protected against not authorized tunnel creation, i.e., both IPv6 and IPv4 ACLs are applied to try command arguments.

## 4.3 Debug primer

Unlimit the core dump size if you'd like to get core file on crashes or with the abort command. On Linux twist the core naming to something better than `core` (cf. core(5)). Please keep the binary associated to core files. As the **fork** command is fun but eats memory put enough memory in the aftr box...

When the **aftr** process is not (yet) crashed but seems no longer to forward packets:

- go to an open session (try to keep on in case the alternative fails) or if none open a new one
- check if it is responsive using the **noop** (answer `LOG: alive`), if not try to get a core file (attach in gdb and use **gcore**), kill it (another way to get a core file with **^\ / kill**) and relaunch it
- if not in a hurry try to understand the issue with **show stat** and **show dropped**
- open a second session, send **fork** to get a child process where you can use extensive debug, including gdb, on it. If you don't know or you can't understand, **abort** the child process to get a core file.
- update the config file if needed, reboot the parent/main process (it will lose all the state and restart from the beginning)

Summary for the busy operator:

- **noop** -> nothing: go to the shell to kill and relaunch it
- **noop** -> expected message: open another session, send **fork**, wait for the child pid message, send **abort** on this new session. On the previous session (where you sent **noop**), send **reboot**

# Chapter 5

# XML Interface

The "XML interface" is a way for service providers to programmatically manage static port mappings on behalf of their customers.

For instance, the service provider might have a web portal, tied into the provisioning system, through which a customer could request a small number of static port mappings. The provisioning system would send an XML-encoded request to the AFTR that is serving that customer, and the AFTR would send back an XML-encoded reply.

The `xml` subdirectory contains a specification for the remote configuration protocol, together with a server that runs on the AFTR box, and an example client that runs on the provisioning system.

## 5.1 Transport

The server and client communicate over either HTTP or a plain TCP socket. By default, they use HTTP. To change to TCP socket, you must edit both `xmlconf.py` and `xmlclient.py`, commenting out the line **`TRANSPORT = 'http'`**, and uncommenting the line **`TRANSPORT = 'socket'`**.

## 5.2 Remote Configuration Daemon

Included inline xmlconf (8)

### 5.2.1 xmlconf.py

xmlconf.py — remote configuration daemon for aftr

#### Synopsis

`xmlconf.py` [`-l` *`listening-addr`*] [`-p` *`listening-port`*] [`-r` *`remote-addr`*] [`-c` *`config-file`*] [`-v`]

#### OPTIONS

**-l** ***`listening-addr`***

This specifies a local address on which to listen. If not specified, it listens on all local addresses.

**-p** ***listening-port***
: This specifies a local port on which to listen. Defaults to port 4148 for HTTP transport, or port 4146 for socket transport.

**-r** ***remote-addr***
: This specifies a single address that the server will listen to. The server should only get configuration requests from the provisioning system, at a known address, so this is a simple form of access control. Use of this option is not required, but it is recommended.

**-c** ***config-file***
: This specifies the name and location of the **aftr** configuration file. For obvious reasons, this option MUST specify the same file that is used to configure the running **aftr** daemon. Default is `./aftr.conf`.

**-v**
: This enables run-time debugging messages. It should not be used in production, but it can help to debug or monitor interactions between **xmlconf.py** and the **aftr** daemon.

#### SEE ALSO

aftr(8), aftr.conf(5), xmlclient(8)

#### AUTHOR

Internet Systems Consortium

## 5.3 Remote Configuration Client

Included inline xmlclient (8)

### 5.3.1 xmlclient.py

xmlclient.py — remote configuration client for aftr

#### Synopsis

`xmlclient.py` *`aftr-addr`* [*`command`*]

#### OPTIONS

**aftr-addr**
: This is the address (IPv4 or IPv6) the the target AFTR.

#### COMMANDS

**create** ***user-ipv6 protocol src-ipv4 src-port nat-ipv4 nat-port***
This requests the aftr to create a port mapping.

**create** ***user-ipv6 nat-ipv4***
This requests the aftr to create a tunnel entry, using *nat-ipv4* as the natted IPv4 address for all future port mappings on this tunnel (dynamic as well as static).

**delete** ***user-ipv6 protocol src-ipv4 src-port nat-ipv4 nat-port***

**delete** ***user-ipv6 protocol src-ipv4 src-port***

**delete** ***protocol nat-ipv4 nat-port***
These three forms of the **delete** command all request the aftr to delete a port mapping. The mapping can be fully specified (first form), but a mapping can also be uniquely identifed by either internal parameters (second form) or external parameters (third form).

**delete** ***user-ipv6***
This requests the aftr to delete all port mappings (dynamic as well as static) and other state associated with the given tunnel address. This is often done prior to moving the customer to a new natted IPv4 address.

**flush**
This requests the aftr to remove all static port mappings and configured tunnel entries. Note that this is a very drastic action, and should only be undertaken if (for example) the aftr configuration is seriously out of sync with the provisioning system.

**get** ***user-ipv6***
This requests the aftr to report all static port mappings associated with the given tunnel address.

**get**
This requests the aftr to report all static port mappings, and all configured tunnels without static port mappings.

#### SCRIPTING

If no commands are given on the command line, **xmlclient.py** will read commands from stdin. This allows the provisioning system to accumulate changes for a given AFTR, and send them all at once.

In general, it is probably easier for the provisioning system to send requests immediately, and get replies immediately. However, some operators may prefer to batch up requests, and this method sends multiple requests over an open connection, without having to establish a connection for each request.

Example:
**xmlclient.py 2001::500 <script**

where `script` contains:

```
create 2001::525a:8c5a:30d4:e36e tcp 192.168.0.88 6265 198.18.200.174
5005
create 2001::835c:1eff:8d66:22fc tcp 192.168.1.138 3877 198.18.200.121
5572
create 2001::e3:9a2f:8abf:40de:2d87 udp 192.168.0.92 7356
198.18.200.149 5547
```

**SEE ALSO**

xmlconf(8)

**AUTHOR**

Internet Systems Consortium

# Chapter 6

# Advanced Topics

## 6.1 No-Nat/Pass-Through

In a world where IPv4 address sharing is the norm, one might imagine that some customers would be willing to pay a little more for a full, non-shared global IPv4 prefix or address (i.e., a /32 prefix).

In this case, the customer would perform the NAT function in his own CPE, but he would still have to tunnel IPv4 traffic through the provider's IPv6-only network to the AFTR.

As yet, there is no defined signalling for the client to request a non-shared IPv4 prefix, or for the server to establish a non-natted tunnel. All configuration must be done manually, on both the B4 and the AFTR.

- On the B4, the tunnel will still have to be set up, and all IPv4 traffic will have to be routed to the tunnel, but only after the local NAT function is performed.
- On the AFTR, a **nonat** command will have to be entered in `aftr.conf`, and a route to the customer's IPv4 prefix will have to be created in `aftr-script`.

## 6.2 A+P/Port-Range Routing

A+P is a different approach to IPv4 address sharing, described in `draft-ymbk-aplusp-05.txt`. In this scenario, each customer is provisioned with a global IPv4 address, but only a restricted range of ports he can use within that address.

Similar to the no-nat case above, the customer would perform the NAT function within his own CPE (ensuring that all port mappings are within the provisioned range), but he would still have to tunnel IPv4 traffic through the provider's IPv6-only network to the AFTR.

As yet, there is no defined signalling for the client to request an A+P assignment, or for the server to establish a non-natted tunnel. All configuration must be done manually, on both the B4 and the AFTR.

- On the B4, the tunnel will still have to be set up, and all IPv4 traffic will have to be routed to the tunnel, but only after the local NAT function is performed.
- On the AFTR, a **prr** command will have to be entered in `aftr.conf`, and a route to the customer's IPv4 address will have to be created in `aftr-script`.

## 6.3 Sharing a Single Address

In production use, an AFTR will have a pool of global IPv4 addresses that are used exclusively for natted port mappings.

However, for testing or demonstration purposes, you may want to deploy AFTR on a box with only one IPv4 address:

- This address is used for standard services of the box.
- This address is used by application proxies etc., in particular the DNS caching server.
- This address is used to NAT traffic, i.e., for the AFTR function.

The address can be dynamic (provisioned by DHCP), but must not change during an AFTR process run.

The AFTR box is configured to use the eth0 interface on the WAN side, the eth1 on the LAN side. The AFTR process itself is configured as usual, but it uses a pseudo-public address (i.e., an address which is not recognized as private but in fact is a reserved public address, the first to avoid confusion, the second to avoid a collision with a real public address).

Netfilter/iptables is used to map the pseudo-public address to the real public address. Port forwarding is a bit more complex, as the port range used for port forwarding must be port-forwarded (destination natted in netfilter/iptables terms) as-is (i.e., not changing ports) to the pseudo-public address. Of course there is nothing which can be done for no-NATs or for A+P/PRR as the first router/NAT of the Internet connection has no reason to support it.

Note that this creates a double-NAT situation within the AFTR box: customer traffic is natted once in the AFTR itself, to the pseudo-public address, then a second time in netfilter, to the real public address. This is obviously not ideal from a performance perspective, but this scenario is only for testing and demonstration purposes.

For the AFTR box itself configuration should be:

- Use the standard setup for eth0 (i.e., plain DHCP).
- Use the standard setup for the AFTR function, only the script needs to be special.
- IPv4 forwarding must be enabled.
- Don't forget to flush iptables and ip6tables.

Use or adapt the example `aftr.conf` and `aftr-script` files from the `conf/shareone` directory. (They use 198.18.200.111 as the pseudo-public address and 5000-59999 TCP and UDP port ranges for dynamic NAT bindings.)

If the kernel supports it (see **iptables** SNAT section) it can be useful to add **`--random`** to the SNAT rule in order to get back port randomization.

### 6.3.1 For netfilter/iptables Wizards

- **`$PUBLIC`** is the shared real public address, it is taken from the eth0 configuration.
- The flush in `stop` and at the beginning of `start` is for cleaning NAT rules.

- The SNAT rule just creates a new conntrack NAT entry for the first packet of a flow to the Internet coming from the AFTR. It adds no constraint on the protocol or the natted source port (but the AFTR has itself such constraints, protocols are tcp/udp/icmp-echo and the source port will be in the range declared in the pool so the natted port should be in one of the ranges decribed in **iptables** SNAT section).
- The first DNAT rule remaps traffic to a matching port to the pseudo public address without changing the destination port. It is used for port forwarding.
- Destination ports are protocol specific so the rules have to be duplicated from TCP to UDP.
- Locally generated traffic doesn't go through PREROUTING, so the rules have to be duplicated from PREROUTING to OUTPUT.

Don't forget that with conntrack a NAT entry matches the both ways, so what matters is the processing of the first packet of a flow. Further packets are recognized by conntrack to belong to the same flow, including in the "reverse" way, and the NAT rule is applied (the symmetrical rule for reverse way packets). And conntrack is also used to recognize local traffic.

# Appendix A

# Mailing Lists

Bug reports should be sent to: aftr-bugs@isc.org

General questions or feedback (e.g., about configuration, operation, or use cases) should be sent to: aftr-users@isc.org