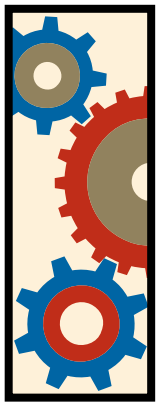
Address Family Transition Router Manual



Internet Systems Consortium

Contents

1 INTRODUCTION **5**
1.1 System Requirements 5

2 BUILD **7**
2.1 Configuration flags 7

3 USAGE **9**
3.1 System setup 9
3.2 Options 9
3.3 aftr 9
3.4 Configuration File 10
3.5 `aftr.conf` 10
3.6 Interactive commands 14
3.7 `aftr.commands` 14
3.8 Command Summary 17

4 SYSLOG **19**
4.1 Trace 19

A Appendices **21**
A.1 Security 21
A.2 Debug primer 21
A.3 BUG REPORTS 21

Chapter 1

INTRODUCTION

This is a stand-alone, user mode application that implements a dual-stack lite (DS-lite) Address Family Transition Router (AFTR, aka. carrier-grade NAT) as described in draft-ietf-softwire-dual-stack-lite-02.txt. It is expected to work on operating systems that support the tun (4) device, and has been tested on Linux and FreeBSD.

The primary purpose of this implementation is to provide a proof of concept of the specification. While it is implemented to be reasonably scalable with regard to the number NAT connections and the number of IPv4/IPv6 tunnels, it is generally not expected to be used in a production environment. Likewise, configuration and management flexibility are limited.

1.1 System Requirements

- OS: Linux or FreeBSD. Linux kernel version must be greater than 2.6.26, to correct a small-packet-drop problem in `tunnel46_rcv()`.
- CPU: Shouldn't matter, but we have only tested it with the Intel architecture. Either 32 or 64 bit CPU should be okay. Note that a 32 bit usermode app can run on a 64 bit kernel.
- Memory: No special requirement for the purpose of proof of concept with a small number of client hosts. Note the connection tracking of netfilter is known to be memory greedy (i.e., more than 16GB system is recommended for production).
- Processor speed: No special requirement for the purpose of proof of concept with a few number of client hosts. Note the performance is bound to the kernel/user context switch latency, so a benchmarking program (`bench.c`, `start` and `stop`) is provided; this sends 1M pings to 192.168.0.4 from 10.0.0.1 through tun0 (*please* change these addresses in the code).

Chapter 2

BUILD

`tar zxvpf aftr-usermode-snapshot-YYYYMMDD.tgz`

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

`cd $src_path`

`./configure`

`make`

An executable file `aftr` will be created, the executable binary of the AFTR daemon program. This is expected to be run on `$src_path` (there is no **`make install`** step) and when needed under gdb.

2.1 Configuration 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 for IPv6 support)
- `notyet`: some unfinished and arguable features (`undef` of course)

Chapter 3

USAGE

3.1 System setup

Linux: Add interface configuration, disable netfilter, enable IPv4 forwarding (look at confs directory). Note on the testbed interface configuration needs some help (**`service network restart`**) and netfilter tables to be flushed (**`iptables -F`** and **`ip6tables -F`**) explicitly.

FreeBSD: Add interface configuration, enable IPv4 forwarding (look at `confs/freebsd-aftr-rc.conf`).

Note: even they don't seem to bother, it is fine to disable ICMP redirects (Linux `net.ipv4.conf.all.send_redire` FreeBSD `net.inet.ip.redirect`).

Clients: look at `confs` directory for configurations used in the testbed.

3.2 Options

Included inline aftr (8)

3.3 aftr

Name

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/confs/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 confs directory provides examples (in fact the script used in our testbed). `freebsd-aftr-*` variant are 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

3.4 Configuration File

Included inline aftr-conf (5)

3.5 `aftr.conf`

Name

`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 `$src_path`. The `AFTRCONFIG` environment variable and the `-c` argument give an alternate path. A sample configuration file is provided in `$src_path/confs/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.

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 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.

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* 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

3.6 Interactive commands

Included inline aftr-commands (5)

3.7 `aftr.commands`

Name

`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 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) 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

3.8 Command Summary

Table 3.1:

Name	Section	Syntax
abort	interactive	
acl6	one or two	*IPv6/prefix_length*
address	one	endpoint *IPv6*|icmp *IPv4*
autotunnel	zero	on|off
debug	>= two	set|enable|...|tunnel

Table 3.1: *(continued)*

Name	Section	Syntax
defmss	zero	on|off
defmtu	zero	*mtu*
deftoobig	zero	on|off|strict
delete	== add	acl6|nat|nonat|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	*IPv6mtu*
nat	two	*IPv6* tcp|udp *IPv4_src* ...
nonat	two	*IPv6IPv4/prefix_length*
noop	interactive	
pool	one	*IPv4* [tcp|udp|echo *min-max*]
prr	two	*IPv6* tcp|udp *IPv4port*
reboot	interactive	
reload	interactive	
session	interactive	close|config|log|name|notify
show	interactive	dropped|stat
toobig	>= two	*IPv6* on|off|strict
try	two	tunnel *IPv6* | nat *IPv6* tcp|udp *IPv4_src* ...
tunnel	two	*IPv6* [*IPv4*]

Chapter 4

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 Trace

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*

Appendix A

Appendices

A.1 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 **try** command is protected against not authorized tunnel creation.

A.2 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**

A.3 BUG REPORTS

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