NAME

pf - packet filter

SYNOPSIS

device pf
options PF_DEFAULT_TO_DROP

DESCRIPTION

Packet filtering takes place in the kernel. A pseudo-device, */dev/pf*, allows userland processes to control the behavior of the packet filter through an ioctl(2) interface. There are commands to enable and disable the filter, load rulesets, add and remove individual rules or state table entries, and retrieve statistics. The most commonly used functions are covered by pfctl(8).

Manipulations like loading a ruleset that involve more than a single ioctl(2) call require a so-called *ticket*, which prevents the occurrence of multiple concurrent manipulations.

Fields of ioctl(2) parameter structures that refer to packet data (like addresses and ports) are generally expected in network byte-order.

Rules and address tables are contained in so-called *anchors*. When servicing an ioctl(2) request, if the anchor field of the argument structure is empty, the kernel will use the default anchor (i.e., the main ruleset) in operations. Anchors are specified by name and may be nested, with components separated by '/' characters, similar to how file system hierarchies are laid out. The final component of the anchor path is the anchor under which operations will be performed.

SYSCTL VARIABLES AND LOADER TUNABLES

The following loader(8) tunables are available.

net.pf.states_hashsize
Size of hash tables that store states. Should be power of 2. Default value is 131072.

net.pf.source_nodes_hashsize
Size of hash table that store source nodes. Should be power of 2. Default value is 32768.

Read only sysctl(8) variables with matching names are provided to obtain current values at runtime.

KERNEL OPTIONS

The following options in the kernel configuration file are related to **pf** operation:

PF_DEFAULT_TO_DROP Change default policy to drop by default

IOCTL INTERFACE

pf supports the following ioctl(2) commands, available through <*net/pfvar.h*>:

DIOCSTART
: Start the packet filter.

DIOCSTOP
: Stop the packet filter.

DIOCSTARTALTQ
: Start the ALTQ bandwidth control system (see altq(9)).

DIOCSTOPALTQ
: Stop the ALTQ bandwidth control system.

DIOCBEGINADDRS *struct pfioc_pooladdr *pp*

```
struct pfioc_pooladdr {
        u_int32_t          action;
        u_int32_t          ticket;
        u_int32_t          nr;
        u_int32_t          r_num;
        u_int8_t           r_action;
        u_int8_t           r_last;
        u_int8_t           af;
        char                         anchor[MAXPATHLEN];
        struct pf_pooladdr addr;
};
```

Clear the buffer address pool and get a *ticket* for subsequent DIOCADDADDR, DIOCADDRULE, and DIOCCHANGERULE calls.

DIOCADDADDR *struct pfioc_pooladdr *pp*

Add the pool address *addr* to the buffer address pool to be used in the following DIOCADDRULE or DIOCCHANGERULE call. All other members of the structure are ignored.

DIOCADDRULE *struct pfioc_rule *pr*

```
struct pfioc_rule {
        u_int32_t action;
        u_int32_t ticket;
        u_int32_t pool_ticket;
        u_int32_t nr;
        char            anchor[MAXPATHLEN];
        char            anchor_call[MAXPATHLEN];
        struct pf_rule  rule;
};
```

Add *rule* at the end of the inactive ruleset. This call requires a *ticket* obtained through a preceding DIOCXBEGIN call and a *pool_ticket* obtained through a DIOCBEGINADDRS call. DIOCADDADDR must also be called if any pool addresses are required. The optional *anchor* name indicates the anchor in which to append the rule. *nr* and *action* are ignored.

DIOCADDALTQ *struct pfioc_altq *pa*

Add an ALTQ discipline or queue.

```
struct pfioc_altq {
        u_int32_t action;
        u_int32_t ticket;
        u_int32_t nr;
        struct pf_altq  altq;
};
```

DIOCGETRULES *struct pfioc_rule *pr*

Get a *ticket* for subsequent DIOCGETRULE calls and the number *nr* of rules in the active ruleset.

DIOCGETRULE *struct pfioc_rule *pr*

Get a *rule* by its number *nr* using the *ticket* obtained through a preceding DIOCGETRULES call. If *action* is set to PF_GET_CLR_CNTR, the per-rule statistics on the requested rule are cleared.

DIOCGETADDRS *struct pfioc_pooladdr *pp*

Get a *ticket* for subsequent DIOCGETADDR calls and the number *nr* of pool addresses in the rule specified with *r_action*, *r_num*, and *anchor*.

DIOCGETADDR *struct pfioc_pooladdr *pp*

Get the pool address *addr* by its number *nr* from the rule specified with *r_action*, *r_num*, and *anchor* using the *ticket* obtained through a preceding DIOCGETADDRS call.

DIOCGETALTQS *struct pfioc_altq *pa*
Get a *ticket* for subsequent DIOCGETALTQ calls and the number *nr* of queues in the active list.

DIOCGETALTQ *struct pfioc_altq *pa*
Get the queueing discipline *altq* by its number *nr* using the *ticket* obtained through a preceding DIOCGETALTQS call.

DIOCGETQSTATS *struct pfioc_qstats *pq*
Get the statistics on a queue.

```
struct pfioc_qstats {
        u_int32_t  ticket;
        u_int32_t  nr;
        void            *buf;
        int              nbytes;
        u_int8_t   scheduler;
};
```

This call fills in a pointer to the buffer of statistics *buf*, of length *nbytes*, for the queue specified by *nr*.

DIOCGETRULESETS *struct pfioc_ruleset *pr*

```
struct pfioc_ruleset {
        u_int32_t  nr;
        char             path[MAXPATHLEN];
        char             name[PF_ANCHOR_NAME_SIZE];
};
```

Get the number *nr* of rulesets (i.e., anchors) directly attached to the anchor named by *path* for use in subsequent DIOCGETRULESET calls. Nested anchors, since they are not directly attached to the given anchor, will not be included. This ioctl returns ENOENT if the parent anchor given at *path* does not exist.

DIOCGETRULESET *struct pfioc_ruleset *pr*
Get a ruleset (i.e., an anchor) *name* by its number *nr* from the given anchor *path*, the maximum number of which can be obtained from a preceding DIOCGETRULESETS call. This ioctl

returns ENOENT if the parent anchor given by *path* does not exist or EBUSY if the index passed in by *nr* is greater than the number of anchors.

DIOCADDSTATE *struct pfioc_state *ps*

Add a state entry.

```
struct pfioc_state {
        struct pfsync_state  state;
};
```

DIOCGETSTATENV *struct pfioc_nv *nv*

Extract the entry identified by the *id* and *creatorid* fields of the *state* nvlist from the state table.

DIOCKILLSTATES *struct pfioc_state_kill *psk*

Remove matching entries from the state table. This ioctl returns the number of killed states in *psk_killed*.

```
struct pfioc_state_kill {
        struct pf_state_cmp psk_pfcmp;
        sa_family_t             psk_af;
        int                     psk_proto;
        struct pf_rule_addr psk_src;
        struct pf_rule_addr psk_dst;
        char                    psk_ifname[IFNAMSIZ];
        char                    psk_label[PF_RULE_LABEL_SIZE];
        u_int                   psk_killed;
};
```

DIOCCLRSTATES *struct pfioc_state_kill *psk*

Clear all states. It works like DIOCKILLSTATES, but ignores the *psk_af*, *psk_proto*, *psk_src*, and *psk_dst* fields of the *pfioc_state_kill* structure.

DIOCSETSTATUSIF *struct pfioc_if *pi*

Specify the interface for which statistics are accumulated.

```
struct pfioc_if {
        char            ifname[IFNAMSIZ];
};
```

DIOCGETSTATUS *struct pf_status *s*

Get the internal packet filter statistics.

```
struct pf_status {
        u_int64_t counters[PFRES_MAX];
        u_int64_t lcounters[LCNT_MAX];
        u_int64_t fcounters[FCNT_MAX];
        u_int64_t scounters[SCNT_MAX];
        u_int64_t pcounters[2][2][3];
        u_int64_t bcounters[2][2];
        u_int32_t running;
        u_int32_t states;
        u_int32_t src_nodes;
        u_int32_t since;
        u_int32_t debug;
        u_int32_t hostid;
        char            ifname[IFNAMSIZ];
        u_int8_t  pf_chksum[MD5_DIGEST_LENGTH];
};
```

DIOCCLRSTATUS
Clear the internal packet filter statistics.

DIOCNATLOOK *struct pfioc_natlook *pnl*
Look up a state table entry by source and destination addresses and ports.

```
struct pfioc_natlook {
        struct pf_addr    saddr;
        struct pf_addr    daddr;
        struct pf_addr    rsaddr;
        struct pf_addr    rdaddr;
        u_int16_t  sport;
        u_int16_t  dport;
        u_int16_t  rsport;
        u_int16_t  rdport;
        sa_family_t       af;
        u_int8_t   proto;
        u_int8_t   direction;
};
```

DIOCSETDEBUG *u_int32_t *level*

Set the debug level.

```
enum    { PF_DEBUG_NONE, PF_DEBUG_URGENT, PF_DEBUG_MISC,
          PF_DEBUG_NOISY };
```

DIOCGETSTATESV2 *struct pfioc_states_v2 *ps*

Get state table entries.

```
struct pfioc_states_v2 {
        int             ps_len;
        uint64_t        ps_req_version;
        union {
                void                    *ps_buf;
                struct pf_state_export  *ps_states;
        };
};

struct pf_state_export {
        uint64_t         version;
        uint64_t         id;
        char             ifname[IFNAMSIZ];
        char             orig_ifname[IFNAMSIZ];
        struct pf_state_key_export      key[2];
        struct pf_state_peer_export     src;
        struct pf_state_peer_export     dst;
        struct pf_addr   rt_addr;
        uint32_t         rule;
        uint32_t         anchor;
        uint32_t         nat_rule;
        uint32_t         creation;
        uint32_t         expire;
        uint32_t         spare0;
        uint64_t         packets[2];
        uint64_t         bytes[2];
        uint32_t         creatorid;
        uint32_t         spare1;
        sa_family_t      af;
        uint8_t          proto;
        uint8_t          direction;
        uint8_t          log;
        uint8_t         state_flags_compat;
        uint8_t         timeout;
        uint8_t         sync_flags;
        uint8_t         updates;
        uint16_t        state_flags;
        uint16_t        qid;
        uint16_t        pqid;
        uint16_t        dnpipe;
        uint16_t        dnrpipe;
        int32_t         rtableid;
        uint8_t         min_ttl;
        uint8_t         set_tos;
        uint16_t        max_mss;
        uint8_t         set_prio[2];
        uint8_t         rt;
        char            rt_ifname[IFNAMSIZ];
        uint8_t         spare[72];
};
```

DIOCCHANGERULE *struct pfioc_rule *pcr*
Add or remove the *rule* in the ruleset specified by *rule.action*.

The type of operation to be performed is indicated by *action*, which can be any of the following:

```
enum    { PF_CHANGE_NONE, PF_CHANGE_ADD_HEAD, PF_CHANGE_ADD_TAIL,
          PF_CHANGE_ADD_BEFORE, PF_CHANGE_ADD_AFTER,
          PF_CHANGE_REMOVE, PF_CHANGE_GET_TICKET };
```

ticket must be set to the value obtained with PF_CHANGE_GET_TICKET for all actions except PF_CHANGE_GET_TICKET. *pool_ticket* must be set to the value obtained with the DIOCBEGINADDRS call for all actions except PF_CHANGE_REMOVE and PF_CHANGE_GET_TICKET. *anchor* indicates to which anchor the operation applies. *nr* indicates the rule number against which PF_CHANGE_ADD_BEFORE, PF_CHANGE_ADD_AFTER, or PF_CHANGE_REMOVE actions are applied.

DIOCCHANGEADDR *struct pfioc_pooladdr *pca*
Add or remove the pool address *addr* from the rule specified by *r_action*, *r_num*, and *anchor*.

DIOCSETTIMEOUT *struct pfioc_tm *pt*

```
struct pfioc_tm {
        int             timeout;
        int             seconds;
};
```

Set the state timeout of *timeout* to *seconds*. The old value will be placed into *seconds*. For possible values of *timeout*, consult the PFTM_* values in <*net/pfvar.h*>.

DIOCGETTIMEOUT *struct pfioc_tm *pt*

Get the state timeout of *timeout*. The value will be placed into the *seconds* field.

DIOCCLRRULECTRS

Clear per-rule statistics.

DIOCSETLIMIT *struct pfioc_limit *pl*

Set the hard limits on the memory pools used by the packet filter.

```
struct pfioc_limit {
        int             index;
        unsigned  limit;
};

enum    { PF_LIMIT_STATES, PF_LIMIT_SRC_NODES, PF_LIMIT_FRAGS,
          PF_LIMIT_TABLE_ENTRIES, PF_LIMIT_MAX };
```

DIOCGETLIMIT *struct pfioc_limit *pl*

Get the hard *limit* for the memory pool indicated by *index*.

DIOCRCLRTABLES *struct pfioc_table *io*

Clear all tables. All the ioctls that manipulate radix tables use the same structure described below. For DIOCRCLRTABLES, *pfrio_ndel* contains on exit the number of tables deleted.

```
struct pfioc_table {
        struct pfr_table        pfrio_table;
        void                    *pfrio_buffer;
        int                      pfrio_esize;
        int                      pfrio_size;
        int                      pfrio_size2;
        int                      pfrio_nadd;
        int                      pfrio_ndel;
        int                 pfrio_nchange;
        int                 pfrio_flags;
        u_int32_t        pfrio_ticket;
};
#define pfrio_exists    pfrio_nadd
#define pfrio_nzero     pfrio_nadd
#define pfrio_nmatch    pfrio_nadd
#define pfrio_naddr     pfrio_size2
#define pfrio_setflag   pfrio_size2
#define pfrio_clrflag   pfrio_nadd
```

DIOCRADDTABLES *struct pfioc_table *io*

Create one or more tables. On entry, *pfrio_buffer* must point to an array of *struct pfr_table* containing at least *pfrio_size* elements. *pfrio_esize* must be the size of *struct pfr_table*. On exit, *pfrio_nadd* contains the number of tables effectively created.

```
struct pfr_table {
        char            pfrt_anchor[MAXPATHLEN];
        char            pfrt_name[PF_TABLE_NAME_SIZE];
        u_int32_t pfrt_flags;
        u_int8_t   pfrt_fback;
};
```

DIOCRDELTABLES *struct pfioc_table *io*

Delete one or more tables. On entry, *pfrio_buffer* must point to an array of *struct pfr_table* containing at least *pfrio_size* elements. *pfrio_esize* must be the size of *struct pfr_table*. On exit, *pfrio_ndel* contains the number of tables effectively deleted.

DIOCRGETTABLES *struct pfioc_table *io*

Get the list of all tables. On entry, *pfrio_buffer[pfrio_size]* contains a valid writeable buffer for *pfr_table* structures. On exit, *pfrio_size* contains the number of tables written into the buffer. If the buffer is too small, the kernel does not store anything but just returns the required buffer size, without error.

DIOCRGETTSTATS *struct pfioc_table *io*

This call is like DIOCRGETTABLES but is used to get an array of *pfr_tstats* structures.

```
struct pfr_tstats {
        struct pfr_table pfrts_t;
        u_int64_t  pfrts_packets
```

```
                                        [PFR_DIR_MAX][PFR_OP_TABLE_MAX];
                u_int64_t  pfrts_bytes
                                        [PFR_DIR_MAX][PFR_OP_TABLE_MAX];
                u_int64_t  pfrts_match;
                u_int64_t  pfrts_nomatch;
                long                pfrts_tzero;
                int                 pfrts_cnt;
                int                 pfrts_refcnt[PFR_REFCNT_MAX];
        };
        #define pfrts_name  pfrts_t.pfrt_name
        #define pfrts_flags   pfrts_t.pfrt_flags
```

DIOCRCLRTSTATS *struct pfioc_table *io*

Clear the statistics of one or more tables. On entry, *pfrio_buffer* must point to an array of *struct pfr_table* containing at least *pfrio_size* elements. *pfrio_esize* must be the size of *struct pfr_table*. On exit, *pfrio_nzero* contains the number of tables effectively cleared.

DIOCRCLRADDRS *struct pfioc_table *io*

Clear all addresses in a table. On entry, *pfrio_table* contains the table to clear. On exit, *pfrio_ndel* contains the number of addresses removed.

DIOCRADDADDRS *struct pfioc_table *io*

Add one or more addresses to a table. On entry, *pfrio_table* contains the table ID and *pfrio_buffer* must point to an array of *struct pfr_addr* containing at least *pfrio_size* elements to add to the table. *pfrio_esize* must be the size of *struct pfr_addr*. On exit, *pfrio_nadd* contains the number of addresses effectively added.

```
struct pfr_addr {
        union {
                struct in_addr          _pfra_ip4addr;
                struct in6_addr         _pfra_ip6addr;
        }                       pfra_u;
        u_int8_t    pfra_af;
        u_int8_t    pfra_net;
        u_int8_t    pfra_not;
        u_int8_t    pfra_fback;
};
#define pfra_ip4addr    pfra_u._pfra_ip4addr
#define pfra_ip6addr    pfra_u._pfra_ip6addr
```

DIOCRDELADDRS *struct pfioc_table *io*

Delete one or more addresses from a table. On entry, *pfrio_table* contains the table ID and *pfrio_buffer* must point to an array of *struct pfr_addr* containing at least *pfrio_size* elements to delete from the table. *pfrio_esize* must be the size of *struct pfr_addr*. On exit, *pfrio_ndel* contains the number of addresses effectively deleted.

DIOCRSETADDRS *struct pfioc_table *io*

Replace the content of a table by a new address list. This is the most complicated command, which uses all the structure members.

On entry, *pfrio_table* contains the table ID and *pfrio_buffer* must point to an array of *struct pfr_addr* containing at least *pfrio_size* elements which become the new contents of the table. *pfrio_esize* must be the size of *struct pfr_addr*. Additionally, if *pfrio_size2* is non-zero, *pfrio_buffer[pfrio_size..pfrio_size2]* must be a writeable buffer, into which the kernel can copy the addresses that have been deleted during the replace operation. On exit, *pfrio_ndel*, *pfrio_nadd*, and *pfrio_nchange* contain the number of addresses deleted, added, and changed by the kernel. If *pfrio_size2* was set on entry, *pfrio_size2* will point to the size of the buffer used, exactly like DIOCRGETADDRS.

DIOCRGETADDRS *struct pfioc_table *io*

Get all the addresses of a table. On entry, *pfrio_table* contains the table ID and *pfrio_buffer[pfrio_size]* contains a valid writeable buffer for *pfr_addr* structures. On exit, *pfrio_size* contains the number of addresses written into the buffer. If the buffer was too small, the kernel does not store anything but just returns the required buffer size, without returning an error.

DIOCRGETASTATS *struct pfioc_table *io*

This call is like DIOCRGETADDRS but is used to get an array of *pfr_astats* structures.

```
struct pfr_astats {
        struct pfr_addr      pfras_a;
        u_int64_t  pfras_packets
                            [PFR_DIR_MAX][PFR_OP_ADDR_MAX];
        u_int64_t  pfras_bytes
                            [PFR_DIR_MAX][PFR_OP_ADDR_MAX];
        long                 pfras_tzero;
};
```

DIOCRCLRASTATS *struct pfioc_table *io*

Clear the statistics of one or more addresses. On entry, *pfrio_table* contains the table ID and *pfrio_buffer* must point to an array of *struct pfr_addr* containing at least *pfrio_size* elements to be cleared from the table. *pfrio_esize* must be the size of *struct pfr_addr*. On exit, *pfrio_nzero* contains the number of addresses effectively cleared.

DIOCRTSTADDRS *struct pfioc_table *io*

Test if the given addresses match a table. On entry, *pfrio_table* contains the table ID and *pfrio_buffer* must point to an array of *struct pfr_addr* containing at least *pfrio_size* elements, each of which will be tested for a match in the table. *pfrio_esize* must be the size of *struct pfr_addr*. On exit, the kernel updates the *pfr_addr* array by setting the *pfra_fback* member appropriately.

DIOCRSETTFLAGS *struct pfioc_table *io*

Change the PFR_TFLAG_CONST or PFR_TFLAG_PERSIST flags of a table. On entry, *pfrio_buffer* must point to an array of *struct pfr_table* containing at least *pfrio_size* elements. *pfrio_esize* must be the size of *struct pfr_table*. *pfrio_setflag* must contain the flags to add, while *pfrio_clrflag* must contain the flags to remove. On exit, *pfrio_nchange* and *pfrio_ndel* contain the number of tables altered or deleted by the kernel. Yes, tables can be deleted if one removes the PFR_TFLAG_PERSIST flag of an unreferenced table.

DIOCRINADEFINE *struct pfioc_table *io*

Defines a table in the inactive set. On entry, *pfrio_table* contains the table ID and *pfrio_buffer[pfrio_size]* contains an array of *pfr_addr* structures to put in the table. A valid ticket must also be supplied to *pfrio_ticket*. On exit, *pfrio_nadd* contains 0 if the table was already defined in the inactive list or 1 if a new table has been created. *pfrio_naddr* contains the number of addresses effectively put in the table.

DIOCXBEGIN *struct pfioc_trans *io*

```
struct pfioc_trans {
        int                  size;     /* number of elements */
        int                  esize;    /* size of each element in bytes */
        struct pfioc_trans_e {
                int                    rs_num;
                char                   anchor[MAXPATHLEN];
                u_int32_t ticket;
        }                   *array;
};
```

Clear all the inactive rulesets specified in the *pfioc_trans_e* array. For each ruleset, a ticket is returned for subsequent "add rule" ioctls, as well as for the DIOCXCOMMIT and

DIOCXROLLBACK calls.

Ruleset types, identified by *rs_num*, include the following:

PF_RULESET_SCRUB	Scrub (packet normalization) rules.
PF_RULESET_FILTER	Filter rules.
PF_RULESET_NAT	NAT (Network Address Translation) rules.
PF_RULESET_BINAT	Bidirectional NAT rules.
PF_RULESET_RDR	Redirect rules.
PF_RULESET_ALTQ	ALTQ disciplines.
PF_RULESET_TABLE	Address tables.

DIOCXCOMMIT *struct pfioc_trans *io*
Atomically switch a vector of inactive rulesets to the active rulesets. This call is implemented as a standard two-phase commit, which will either fail for all rulesets or completely succeed. All tickets need to be valid. This ioctl returns EBUSY if another process is concurrently updating some of the same rulesets.

DIOCXROLLBACK *struct pfioc_trans *io*
Clean up the kernel by undoing all changes that have taken place on the inactive rulesets since the last DIOCXBEGIN. DIOCXROLLBACK will silently ignore rulesets for which the ticket is invalid.

DIOCSETHOSTID *u_int32_t *hostid*
Set the host ID, which is used by pfsync(4) to identify which host created state table entries.

DIOCOSFPFLUSH
Flush the passive OS fingerprint table.

DIOCOSFPADD *struct pf_osfp_ioctl *io*

```
struct pf_osfp_ioctl {
        struct pf_osfp_entry {
                SLIST_ENTRY(pf_osfp_entry) fp_entry;
                pf_osfp_t       fp_os;
                char                    fp_class_nm[PF_OSFP_LEN];
                char                    fp_version_nm[PF_OSFP_LEN];
                char                    fp_subtype_nm[PF_OSFP_LEN];
        }                       fp_os;
        pf_tcpopts_t            fp_tcpopts;
        u_int16_t       fp_wsize;
        u_int16_t       fp_psize;
        u_int16_t       fp_mss;
        u_int16_t       fp_flags;
        u_int8_t        fp_optcnt;
        u_int8_t        fp_wscale;
        u_int8_t        fp_ttl;
        int                     fp_getnum;
};
```

Add a passive OS fingerprint to the table. Set *fp_os.fp_os* to the packed fingerprint, *fp_os.fp_class_nm* to the name of the class (Linux, Windows, etc), *fp_os.fp_version_nm* to the name of the version (NT, 95, 98), and *fp_os.fp_subtype_nm* to the name of the subtype or patchlevel. The members *fp_mss*, *fp_wsize*, *fp_psize*, *fp_ttl*, *fp_optcnt*, and *fp_wscale* are set to the TCP MSS, the TCP window size, the IP length, the IP TTL, the number of TCP options, and the TCP window scaling constant of the TCP SYN packet, respectively.

The *fp_flags* member is filled according to the <*net/pfvar.h*> include file PF_OSFP_* defines. The *fp_tcpopts* member contains packed TCP options. Each option uses PF_OSFP_TCPOPT_BITS bits in the packed value. Options include any of PF_OSFP_TCPOPT_NOP, PF_OSFP_TCPOPT_SACK, PF_OSFP_TCPOPT_WSCALE, PF_OSFP_TCPOPT_MSS, or PF_OSFP_TCPOPT_TS.

The *fp_getnum* member is not used with this ioctl.

The structure's slack space must be zeroed for correct operation; memset(3) the whole structure to zero before filling and sending to the kernel.

DIOCOSFPGET *struct pf_osfp_ioctl *io*
Get the passive OS fingerprint number *fp_getnum* from the kernel's fingerprint list. The rest of the structure members will come back filled. Get the whole list by repeatedly incrementing the *fp_getnum* number until the ioctl returns EBUSY.

DIOCGETSRCNODES *struct pfioc_src_nodes *psn*

```
struct pfioc_src_nodes {
        int     psn_len;
        union {
                caddr_t         psu_buf;
                struct pf_src_node  *psu_src_nodes;
        } psn_u;
#define psn_buf             psn_u.psu_buf
#define psn_src_nodes       psn_u.psu_src_nodes
};
```

Get the list of source nodes kept by sticky addresses and source tracking. The ioctl must be called once with *psn_len* set to 0. If the ioctl returns without error, *psn_len* will be set to the size of the buffer required to hold all the *pf_src_node* structures held in the table. A buffer of this size should then be allocated, and a pointer to this buffer placed in *psn_buf*. The ioctl must then be called again to fill this buffer with the actual source node data. After that call, *psn_len* will be set to the length of the buffer actually used.

DIOCCLRSRCNODES
Clear the tree of source tracking nodes.

DIOCIGETIFACES *struct pfioc_iface *io*
Get the list of interfaces and interface drivers known to **pf**. All the ioctls that manipulate interfaces use the same structure described below:

```
struct pfioc_iface {
        char                     pfiio_name[IFNAMSIZ];
        void                    *pfiio_buffer;
        int                      pfiio_esize;
        int                      pfiio_size;
        int                      pfiio_nzero;
        int                      pfiio_flags;
};
```

If not empty, *pfiio_name* can be used to restrict the search to a specific interface or driver. *pfiio_buffer[pfiio_size]* is the user-supplied buffer for returning the data. On entry, *pfiio_size* contains the number of *pfi_kif* entries that can fit into the buffer. The kernel will replace this value by the real number of entries it wants to return. *pfiio_esize* should be set to sizeof(struct pfi_kif).

The data is returned in the *pfi_kif* structure described below:

```
struct pfi_kif {
        char                             pfik_name[IFNAMSIZ];
        union {
                RB_ENTRY(pfi_kif)        pfik_tree;
                LIST_ENTRY(pfi_kif)      pfik_list;
        };
        u_int64_t                pfik_packets[2][2][2];
        u_int64_t                pfik_bytes[2][2][2];
        u_int32_t                pfik_tzero;
        u_int                    pfik_flags;
        struct ifnet            *pfik_ifp;
        struct ifg_group        *pfik_group;
        u_int                    pfik_rulerefs;
        TAILQ_HEAD(, pfi_dynaddr)        pfik_dynaddrs;
};
```

DIOCSETIFFLAG *struct pfioc_iface *io*

Set the user settable flags (described above) of the **pf** internal interface description. The filtering process is the same as for DIOCIGETIFACES.

```
#define PFI_IFLAG_SKIP    0x0100    /* skip filtering on interface */
```

DIOCCLRIFFLAG *struct pfioc_iface *io*

Works as DIOCSETIFFLAG above but clears the flags.

DIOCKILLSRCNODES *struct pfioc_iface *io*

Explicitly remove source tracking nodes.

FILES

/dev/pf packet filtering device.

EXAMPLES

The following example demonstrates how to use the DIOCNATLOOK command to find the internal host/port of a NATed connection:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/fcntl.h>
#include <net/if.h>
#include <netinet/in.h>
#include <net/pfvar.h>
#include <err.h>
#include <stdio.h>
```

```
#include <stdlib.h>

u_int32_t
read_address(const char *s)
{
        int a, b, c, d;

        sscanf(s, "%i.%i.%i.%i", &a, &b, &c, &d);
        return htonl(a << 24 | b << 16 | c << 8 | d);
}

void
print_address(u_int32_t a)
{
        a = ntohl(a);
        printf("%d.%d.%d.%d", a >> 24 & 255, a >> 16 & 255,
            a >> 8 & 255, a & 255);
}

int
main(int argc, char *argv[])
{
        struct pfioc_natlook nl;
        int dev;

        if (argc != 5) {
                printf("%s <gwy addr> <gwy port> <ext addr> <ext port>\n",
                    argv[0]);
                return 1;
        }

        dev = open("/dev/pf", O_RDWR);
        if (dev == -1)
                err(1, "open(\"/dev/pf\") failed");

        memset(&nl, 0, sizeof(struct pfioc_natlook));
        nl.saddr.v4.s_addr  = read_address(argv[1]);
        nl.sport            = htons(atoi(argv[2]));
        nl.daddr.v4.s_addr  = read_address(argv[3]);
        nl.dport            = htons(atoi(argv[4]));
```

```
            nl.af            = AF_INET;
            nl.proto         = IPPROTO_TCP;
            nl.direction     = PF_IN;

            if (ioctl(dev, DIOCNATLOOK, &nl))
                    err(1, "DIOCNATLOOK");

            printf("internal host ");
            print_address(nl.rsaddr.v4.s_addr);
            printf(":%u\n", ntohs(nl.rsport));
            return 0;
    }
```

SEE ALSO

ioctl(2), altq(4), if_bridge(4), pflog(4), pfsync(4), pfctl(8), altq(9)

HISTORY

The **pf** packet filtering mechanism first appeared in OpenBSD 3.0 and then FreeBSD 5.2.

This implementation is derived from OpenBSD 4.5. A number of individual features, improvements, bug fixes and security fixes have been ported from later versions of OpenBSD. It has been heavily modified to be capable of running in multithreaded FreeBSD kernel and scale its performance on multiple CPUs.