# pf

*Packet Filter pseudo-device*

## Name:

`/dev/pf`

## Description:

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

---

Although the NetBSD documenation talks about *ioctl()*, you should use *ioctl_socket()* instead in your packet-filtering code. With the microkernel message-passing architecture, *ioctl()* calls that have pointers embedded in them need to be handled specially. The *ioctl_socket()* function will default to *ioctl()* for functionality that doesn't require special handling.

---

Manipulations such as loading a rule set that involve more than a single *ioctl_socket()* call require a*ticket*, which prevents the occurrence of multiple concurrent manipulations.

Fields of *ioctl_socket()* parameter structures that refer to packet data (such as addresses and ports) are generally expected in network-byte order.

Rules and address tables are contained in *anchors*. When servicing an *ioctl_socket()* request, if the anchor field of the argument structure is empty, io-pkt uses the default anchor (i.e. the main rule set) in operations. Anchors are specified by name and may be nested, with components separated by slashes, similar to the way that filesystem hierarchies are laid out. The final component of the anchor path is the anchor under which operations will be performed.

### *ioctl_socket()* interface

The pf pseudo-device supports the following *ioctl_socket()* 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 in the NetBSD documentation).

**DIOCSTOPALTQ**

Stop the ALTQ bandwidth control system.

**DIOCBEGINADDRS struct pfioc_pooladdr *pp**

Clear the buffer address pool and get a ticket for subsequent DIOCADDADDR, DIOCADDRULE, andDIOCCHANGERULE calls. The pfioc_pooladdr structure is defined as follows:

```
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;

};
```

**DIOCADDADDR struct pfioc_pooladdr *pp**

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

**DIOCADDRULE struct pfioc_rule *pr**

Add the given rule at the end of the inactive rule set. The pfioc_rule structure is defined as follows:

```
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;

};
```

This call requires a *ticket* obtained through a preceding DIOCXBEGIN call and a *pool_ticket* obtained through a DIOCBEGINADDRS call. You must also call DIOCADDADDR if any pool addresses are required.
The optional anchor name indicates the anchor in which to append the rule. The *nr* and *action*members. are ignored.

**DIOCADDALTQ struct pfioc_altq *pa**

Add an ALTQ discipline or queue. The pfioc_altq structure is defined as follows:

```
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 rule set.

**DIOCGETRULE struct pfioc_rule *pr**

Get a rule by its number *nr*, using the ticket obtained through a preceding DIOCGETRULES call.

**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 precedingDIOCGETALTQS call.

**DIOCGETQSTATS struct pfioc_qstats *pq**

Get the statistics for a queue. The pfioc_qstats structure is defined as follows:

```
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**

Get the number *nr* of rule sets (i.e., anchors) directly attached to the anchor named by *path* for use in subsequent DIOCGETRULESET calls. The pfioc_ruleset structure is defined as follows:

```
struct pfioc_ruleset {
```

```
        u_int32_t        nr;

        char             path[MAXPATHLEN];

        char             name[PF_ANCHOR_NAME_SIZE];

};
```

Nested anchors, since they aren't directly attached to the given anchor, aren't included. This*ioctl_socket()* command returns EINVAL if the given anchor doesn't exist.

**DIOCGETRULESET struct pfioc_ruleset *pr**

Get a rule set (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_socket()*command returns EINVAL if the given anchor doesn't exist, or EBUSY if another process is concurrently updating a rule set.

**DIOCADDSTATE struct pfioc_state *ps**

Add a state entry. The pfioc_state structure is defined as follows:

```
struct pfioc_state {

        u_int32_t        nr;

        struct pf_state  state;

};
```

**DIOCGETSTATE struct pfioc_state *ps**

Extract the entry with the specified number *nr* from the state table.

**DIOCKILLSTATES struct pfioc_state_kill *psk**

Remove matching entries from the state table. This *ioctl_socket()* command returns the number of killed states in *psk_af*. The pfioc_state_kill structure is defined as follows:

```
struct pfioc_state_kill {

        sa_family_t             psk_af;

        int                     psk_proto;

        struct pf_rule_addr     psk_src;

        struct pf_rule_addr     psk_dst;

        char                    psk_ifname[IFNAMSIZ];

};
```

```

```

**DIOCCLRSTATES struct pfioc_state_kill *psk**

Clear all states. This command 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 to gather statistics. The pfioc_if structure is defined as follows:

```
struct pfioc_if {

         char             ifname[IFNAMSIZ];

};
```

**DIOCGETSTATUS struct pf_status *s**

Get the internal packet filter statistics. The pf_status structure is defined as follows:

```
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_int64_t       stateid;

         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];
```

```
};
```

**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. The pfioc_natlookstructure is defined as follows:

```
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 to one of PF_DEBUG_NONE, PF_DEBUG_URGENT, PF_DEBUG_MISC, orPF_DEBUG_NOISY.

**DIOCGETSTATES struct pfioc_states *ps**

Get state table entries. The pfioc_states structure is defined as follows:

```
struct pfioc_states {

         int     ps_len;

         union {
```

```
                caddr_t          psu_buf;

                struct pf_state *psu_states;

        } ps_u;

#define ps_buf          ps_u.psu_buf

#define ps_states       ps_u.psu_states

};
```

If *ps_len* is zero, all states are gathered into *pf_states*, and *ps_len* is set to the size they take in memory (i.e. sizeof(struct pf_state) * nr). If *ps_len* is nonzero, as many states that can fit into*ps_len* as possible are gathered, and *ps_len* is updated to the size those rules take in memory.

**DIOCCHANGERULE struct pfioc_rule *pcr**

Add or remove the rule in the rule set specified by *rule.action*. The type of operation to be performed is indicated by *action*, which can be any of the following:

- 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

You must set *ticket* to the value obtained with PF_CHANGE_GET_TICKET for all actions exceptPF_CHANGE_GET_TICKET. You must set *pool_ticket* to the value obtained with theDIOCBEGINADDRS call for all actions except PF_CHANGE_REMOVE and PF_CHANGE_GET_TICKET. The *anchor* indicates which anchor the operation applies to. The *nr* member indicates the rule number against which to apply PF_CHANGE_ADD_BEFORE, PF_CHANGE_ADD_AFTER, orPF_CHANGE_REMOVE actions.

**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**

Set the state timeout of *timeout* to *seconds*. The pfioc_tm structure is defined as follows:

```
struct pfioc_tm {

        int timeout;

        int seconds;

};
```

The old value is placed into *seconds*. For the possible values of *timeout*, see the PFTM_* values in<net/pfvar.h>.

**DIOCGETTIMEOUT struct pfioc_tm *pt**

Get the state timeout of *timeout*. The value is 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. The pfioc_limit structure is defined as follows:

```
struct pfioc_limit {

         int        index;

         unsigned   limit;

};



enum { PF_LIMIT_STATES, PF_LIMIT_SRC_NODES,

        PF_LIMIT_FRAGS };
```

**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 *ioctl_socket()* commands that manipulate radix tables use the same structure described below. On exit from the DIOCRCLRTABLES command, *pfrio_ndel* contains the number of tables deleted.
The pfioc_table structure is defined as follows:

```
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[pfrio_size]* contains a table of pfr_table structures. On exit, *pfrio_nadd* contains the number of tables effectively created. The pfr_table structure is defined as follows:

```
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[pfrio_size]* contains a table of pfr_table structures. On exit, *pfrio_nadd* contains the number of tables effectively deleted.

**DIOCRGETTABLES struct pfioc_table *io**

Get a list of all tables. On entry, *pfrio_buffer[pfrio_size]* contains a valid writeable buffer for pfr_tablestructures. On exit, *pfrio_size* contains the number of tables written into the buffer. If the buffer is too small, io-pkt doesn't 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. Thepfr_tstats structure is defined as follows:

```
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[pfrio_size]* contains a table ofpfr_table structures. 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[pfrio_size]* contains the list of pfr_addr structures to add. On exit, *pfrio_nadd* contains the number of addresses effectively added.

The pfr_addr structure is defined as follows:

```
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[pfrio_size]* contains the list of pfr_addr structures to delete. 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[pfrio_size]* contains the new list ofpfr_addr structures. Additionally, if *pfrio_size2* is nonzero, *pfrio_buffer[pfrio_size..pfrio_size2]* must be a writeable buffer, into which io-pkt 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 io-pkt. If *pfrio_size2* was set on entry, *pfrio_size2* points to the size of the buffer used, exactly as for 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 is too small, io-pkt doesn't 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[pfrio_size]* contains a table of pfr_addr structures to clear. 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[pfrio_size]* contains a table of pfr_addr structures to test. On exit, io-pkt updates the*pfr_addr* table 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[pfrio_size]* contains a table of pfr_table structures, and *pfrio_setflag* contains the flags to add, while *pfrio_clrflag* contains the flags to remove. On exit, *pfrio_nchange* and *pfrio_ndel* contain the number of tables altered or deleted by io-pkt.

You can delete tables if you remove the PFR_TFLAG_PERSIST flag of an unreferenced table.

**DIOCRINADEFINE struct pfioc_table *io**

Define a table in the inactive set. On entry, *pfrio_table* contains the table ID, and*pfrio_buffer[pfrio_size]* contains the list 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. The *pfrio_naddr* member contains the number of addresses effectively put in the table.

**DIOCXBEGIN struct pfioc_trans *io**

Clear all the inactive rule sets specified in the *pfioc_trans_e* array. The pfioc_trans structure is defined as follows:

```
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;

};
```

```

```

For each rule set, a ticket is returned for subsequent “add rule” *ioctl_socket()* commands, as well as for the DIOCXCOMMIT and DIOCXROLLBACK calls.
Rule set 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 rule sets to the active rule sets. This call is implemented as a standard two-phase commit, which either fails for all rule sets, or completely succeeds. All tickets need to be valid.
This *ioctl_socket()* command returns EBUSY if another process is concurrently updating some of the same rule sets.

**DIOCXROLLBACK struct pfioc_trans *io**

Clean up io-pkt by undoing all changes that have taken place on the inactive rule sets since the lastDIOCXBEGIN. DIOCXROLLBACK silently ignores rule sets for which the ticket is invalid.

**DIOCSETHOSTID u_int32_t *hostid**

Set the host ID, which is used by pfsync to identify which host created state table entries.

**DIOCOSFPFLUSH**

Flush the passive OS fingerprint table.

**DIOCOSFPADD struct pf_osfp_ioctl *io**

Add a passive OS fingerprint to the table. The pf_osfp_ioctl structure is defined as follows:

```
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;

};
```

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 patch level. 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 PF_OSFP_* definition in <net/pfvar.h>. 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.
This *ioctl_socket()* command doesn't use the *fp_getnum* member.

You must zero the structure's slack space for correct operation; *memset()* the whole structure to zero before filling and sending it to `io-pkt`.

**DIOCOSFPGET struct pf_osfp_ioctl *io**

Get the passive OS fingerprint number *fp_getnum* from io-pkt's fingerprint list. The rest of the structure members comes back filled. Get the whole list by repeatedly incrementing the *fp_getnum*number until *ioctl_socket()* gives an error of EBUSY.

**DIOCGETSRCNODES struct pfioc_src_nodes *psn**

Get the list of source nodes kept by sticky addresses and source tracking. The pfioc_src_nodesstructure is defined as follows:

```
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

};
```

You must call *ioctl_socket()* once with *psn_len* set to 0. If *ioctl_socket()* returns without error, *psn_len*is set to the size of the buffer required to hold all the pf_src_node structures held in the table. You should then allocate a buffer of this size and place a pointer to this buffer in *psn_buf*. You must then call *ioctl_socket()* again to fill this buffer with the actual source node data. After that call, *psn_len* is set to the length of the buffer actually used.

**DIOCCLRSRCNODES**

Clear the tree of source-tracking nodes.

**DIOCIGETIFACES struct pfioc_iface *io**

Get a list of interfaces and interface drivers known to pf. All the *ioctl_socket()* commands that manipulate interfaces use the 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;

};



#define PFI_FLAG_GROUP     0x0001  /* gets groups of interfaces */

#define PFI_FLAG_INSTANCE  0x0002  /* gets single interfaces */

#define PFI_FLAG_ALLMASK   0x0003
```

If it isn't empty, you can use *pfiio_name* to restrict the search to a specific interface or driver.
The*pfiio_buffer[pfiio_size]* member is the user-supplied buffer for returning the data. On entry, *pfiio_size*represents

the number of pfi_if entries that can fit into the buffer. The io-pkt manager replaces this value with the real number of entries it wants to return.
You should set *pfiio_esize* to sizeof(struct pfi_if). You should set *pfiio_flags* toPFI_FLAG_GROUP, PFI_FLAG_INSTANCE, or both, to tell io-pkt to return a group of interfaces (drivers, such as fxp), real interface instances (e.g. fxp1), or both. The data is returned in the pfi_ifstructure described below:

```
struct pfi_if {

         char        pfif_name[IFNAMSIZ];

         u_int64_t   pfif_packets[2][2][2];

         u_int64_t   pfif_bytes[2][2][2];

         u_int64_t   pfif_addcnt;

         u_int64_t   pfif_delcnt;

         long        pfif_tzero;

         int         pfif_states;

         int         pfif_rules;

         int         pfif_flags;

};



#define PFI_IFLAG_GROUP    0x0001  /* group of interfaces */

#define PFI_IFLAG_INSTANCE 0x0002  /* single instance */

#define PFI_IFLAG_CLONABLE 0x0010  /* clonable group */

#define PFI_IFLAG_DYNAMIC  0x0020  /* dynamic group */

#define PFI_IFLAG_ATTACHED 0x0040  /* interface attached */
```

**DIOCICLRISTATS struct pfioc_iface *io**

Clear the statistics counters of one or more interfaces. You can use *pfiio_name* and *pfiio_flags* to select which interfaces need to be cleared. The filtering process is the same as for DIOCIGETIFACES. The *pfiio_nzero* member is set by io-pkt to the number of interfaces and drivers that have been cleared.

**DIOCSETIFFLAG struct pfioc_iface *io**

Set the user-setable flags of the pf internal interface description:

- PFI_IFLAG_SKIP — skip the interface.

The filtering process is the same as for DIOCIGETIFACES.

**DIOCCLRIFFLAG struct pfioc_iface *io**

Similar to DIOCSETIFFLAG, but clear the flags.

## 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_socket(dev, DIOCNATLOOK, &nl))
                err(1, "DIOCNATLOOK");

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

## Caveats:

The following functionality is missing from pf in this version of NetBSD:

- The pfsync protocol isn't supported.
- The group keyword isn't supported.

## See also:

pfctl
*ioctl()* in the Neutrino Library Reference