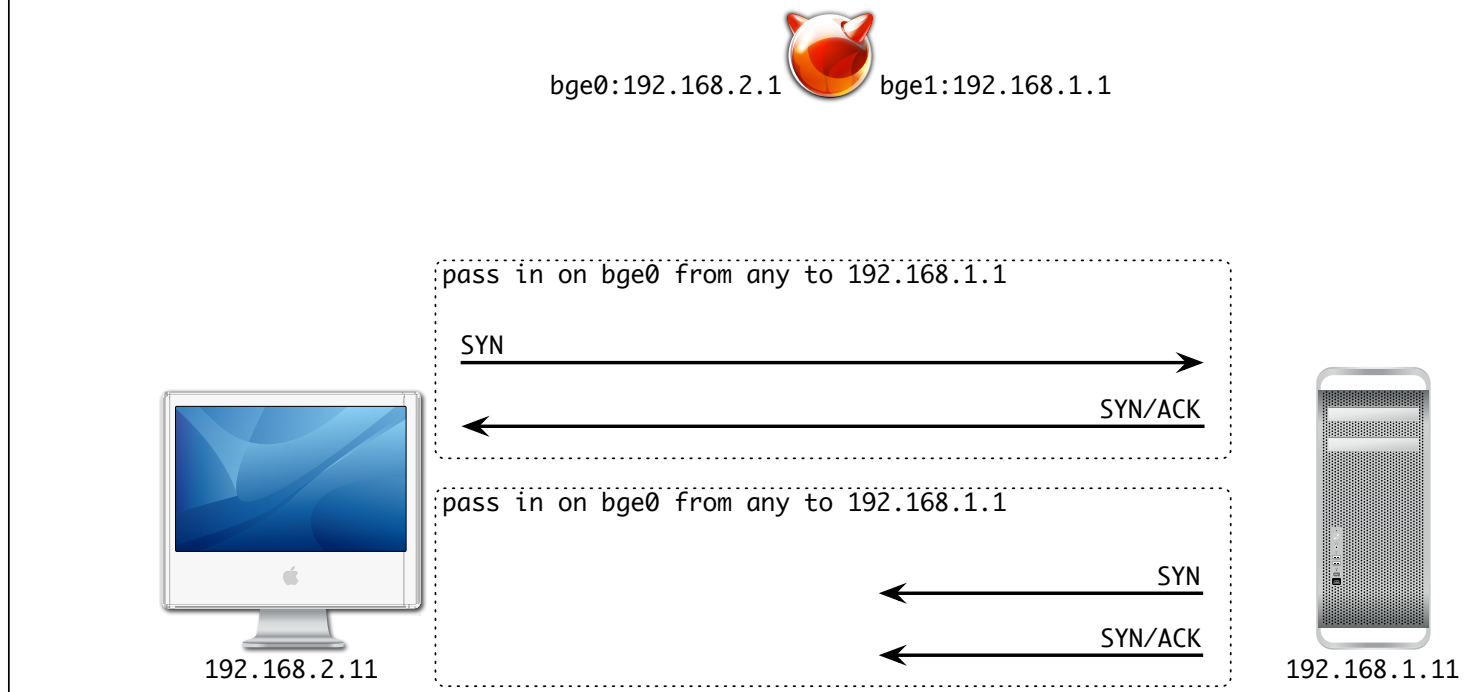
THE DESIGN AND IMPLEMENTATION OF THE PF

有限会社銀座堂 浅間正和

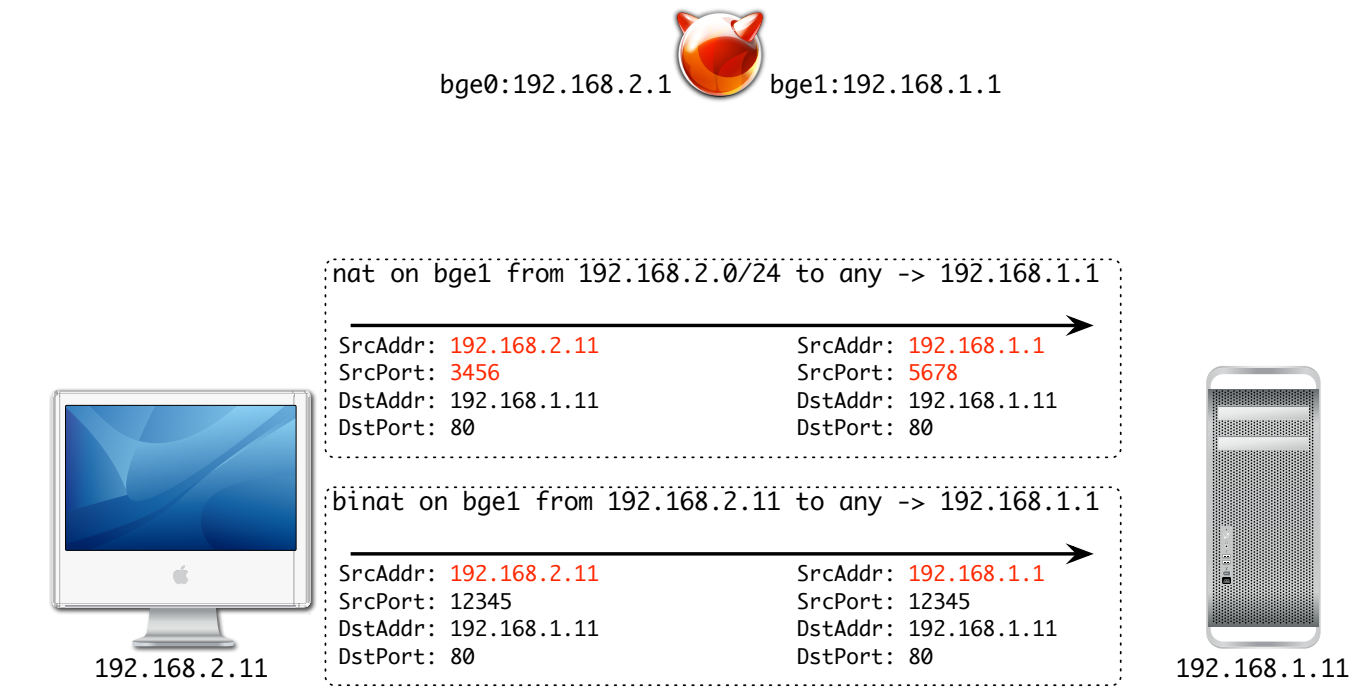
WHAT'S PF

- packet filter, 略して pf. OpenBSD 3.0 で初めて現れ, 現在は多くの *BSD に移植されている. Darren Reed によって開発された IP Filter のライセンス上の問題を解決するため, Daniel Hartmeier によって開発される.
- Stateful Inspection: {出入り|通過}するパケットの状態を逐一把握し, 戻りパケットを暗黙で許可したり, 整合性のとれないパケットを破棄したりする.
- Address Translation: NAPT/1:1 mapping NAT.
- Redirection: Port Forwarding.
- 帯域制御, Scrub(パケット正規化), OS 判定, ...

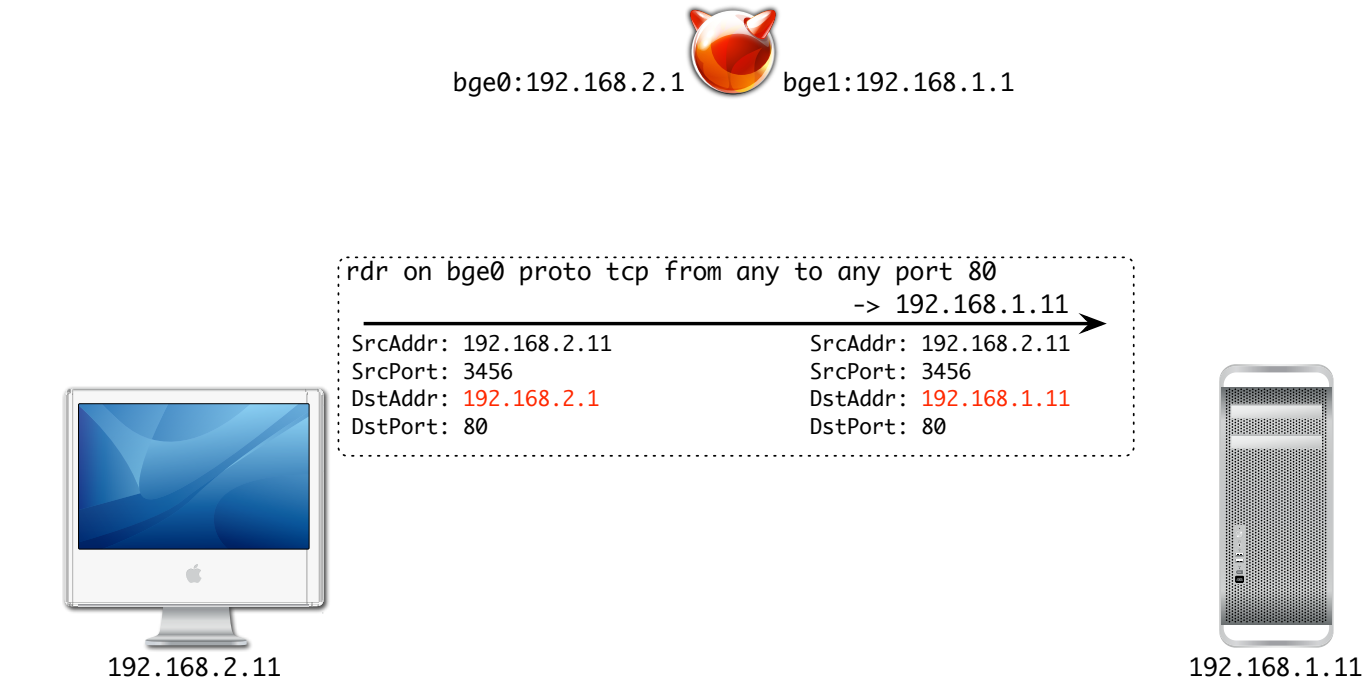
STATEFUL INSPECTION



ADDRESS TRANSLATION



REDIRECTION



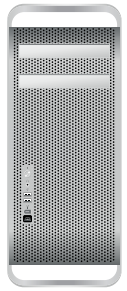
ACTIVATION

- Step 1. `/etc/pf.conf` を作成
  - 後述
- Step 2. `/etc/rc.conf` を編集
  - FreeBSD の場合: `pf_enable="YES"` を追記
  - NetBSD/OpenBSD の場合: `pf=YES` を追記
- Step 3. `reboot`

CONFIGURATION

192.168.2.11

bge0:192.168.2.1 bge1:192.168.1.1



192.168.1.11

```
# Options: Various options to control how PF works.
set block-policy return
set debug misc
set skip on lo0

# Scrub: Reprocessing packets to normalize and defragment them.

# Queueing: Provides bandwidth control and packet prioritization.

# Translation: Controls Network Address Translation and packet redirection.
nat on bge1 from 192.168.2.0/24 to any -> 192.168.1.1
rdr on bge1 proto tcp from any to 192.168.1.1 port 22 -> 192.168.2.11

# Filter Rules: Allows the selective filtering or blocking of packets
# as they pass through any of the interfaces.
block in on bge1 all
pass out on bge1 from 192.168.2.0/24 to any
pass in on bge1 proto tcp from any to 192.168.2.11 port 22
```

FIRST MATCH V.S. LAST MATCH

- pf ではルールセットをすべてチェックし最後にマッチしたルールが適用される(Last Match).
- iptables や cisco ACL のように, マッチした時点でルールが適用されないので注意(First Match).

```
/etc/pf.conf:
block in all
pass in proto tcp from any to any port 22

/etc/sysconfig/iptables:
-A RH-Firewall-1-INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A RH-Firewall-1-INPUT -j REJECT --reject-with icmp-host-prohibited
```

- ただし pf でも `quick` キーワードを使えば First Match でルールを記述することが可能.

```
/etc/pf.conf:
pass in quick proto tcp from any to any port 22
block in all
```

MACROS/LISTS/TABLES

- Macros: IPアドレス, ポート番号, インターフェース名等を保持するユーザ定義の変数.

```
ext_if = "fxp0"
block in on $ext_if from any to any
```

- Lists: ルールの中に判断基準を複数記述するための仕組み.

```
block out on fxp0 proto { tcp udp } \
from { 192.168.0.1, 10.5.32.6 } to any port { ssh telnet }
```

- Tables: IPv4/IPv6アドレスのグループを保持するための仕組み.

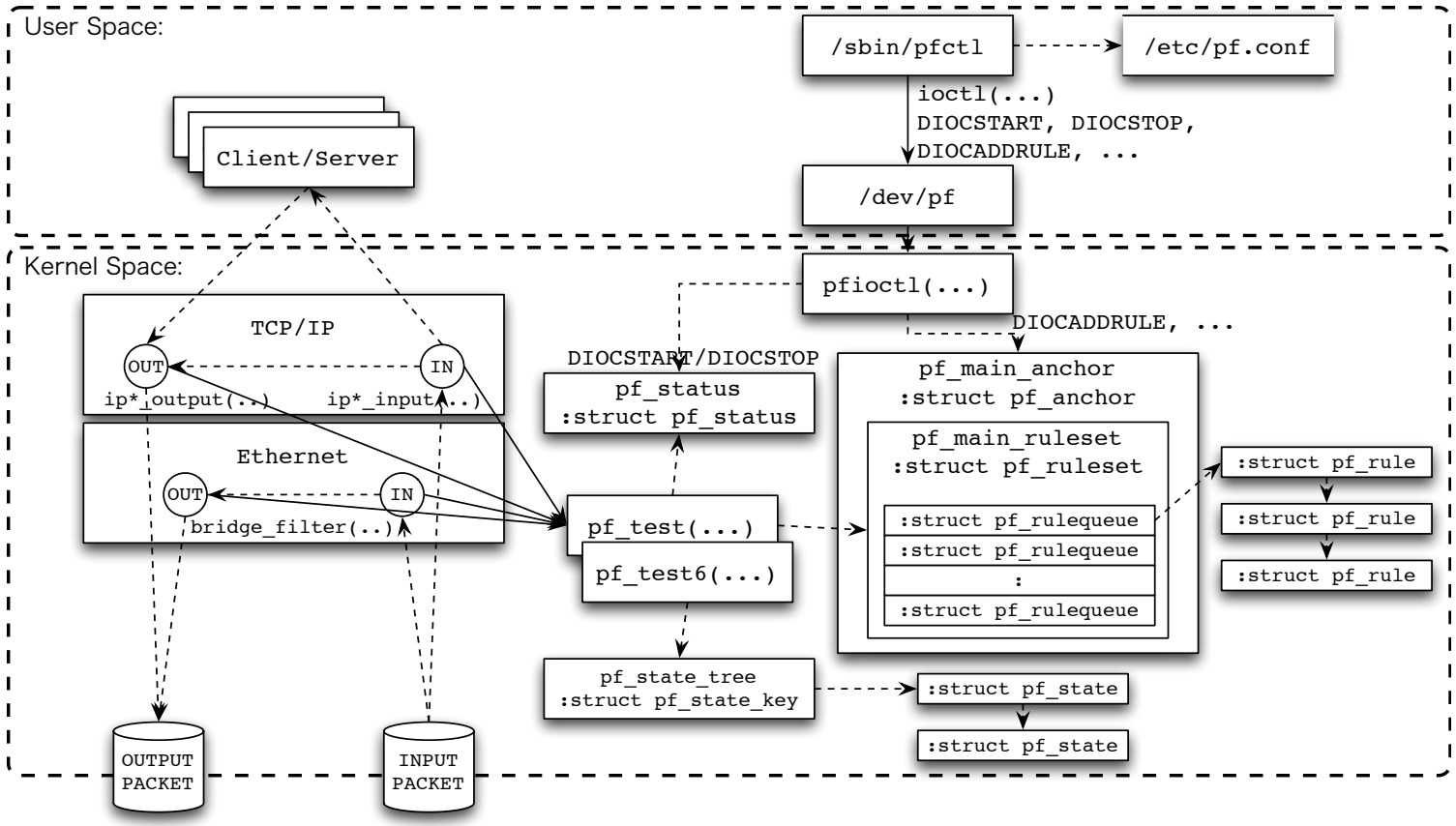
```
table <goodguys> { 192.0.2.0/24 }
table <spammers> persist file "/etc/spammers"
pass  in on fxp0 from <goodguys> to any
block in on fxp0 from <spammers> to any
```

ANCHORS

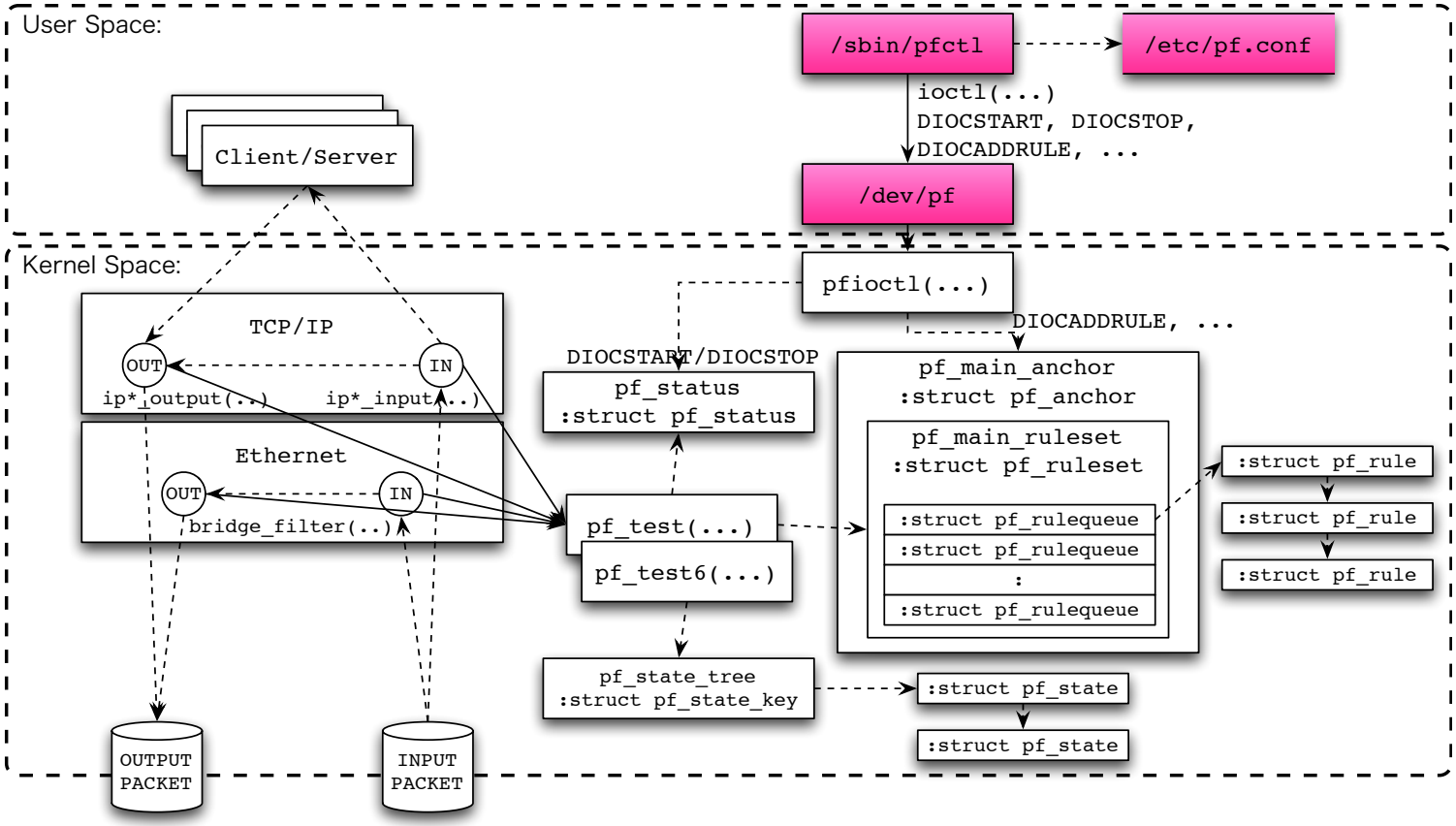
- ルールの固まりをサブルールセットとしてまとめて管理することができる.
- サブルールセットを取り付けるための印をアンカーと呼ぶ.

```
# cat pf.conf
block all
pass out all
anchor a1 {
    block out from 192.168.1.11 to any
    anchor a2 {
        pass out from 192.168.1.11 to 192.168.2.11
    }
}
block out from 192.168.2.11 to any

# pfctl -f pf.conf

# pfctl -s rules -a '*'
block drop all
pass out all flags S/SA keep state
anchor "a1" all {
  block drop out inet from 192.168.1.11 to any
  anchor "a2" all {
    pass out inet from 192.168.1.11 to 192.168.2.11 flags S/SA keep state
  }
}
block drop out inet from 192.168.2.11 to any

# pfctl -F rules -a a1/a2
rules cleared

# pfctl -s rules -a '*'
block drop all
pass out all flags S/SA keep state
anchor "a1" all {
  block drop out inet from 192.168.1.11 to any
  anchor "a2" all {
  }
}
block drop out inet from 192.168.2.11 to any
```

OVERVIEW


User Space:
/sbin/pfctl
/etc/pf.conf
ioctl(...)
DIOCSTART, DIOCSTOP,
DIOCADDRULE, ...
Client/Server
/dev/pf
Kernel Space:
pfioctl(...)
DIOCADDRULE, ...
TCP/IP
OUT
IN
ip*_output(..)
ip*_input(..)
DIOCSTART/DIOCSTOP
pf_status
:struct pf_status
pf_main_anchor
:struct pf_anchor
pf_main_ruleset
:struct pf_ruleset
:struct pf_rule
:struct pf_rule
:struct pf_rule
Ethernet
OUT
IN
bridge_filter(..)
pf_test(...)
pf_test6(...)
:struct pf_rulequeue
:struct pf_rulequeue
:
:struct pf_rulequeue
pf_state_tree
:struct pf_state_key
:struct pf_state
:struct pf_state
OUTPUT
PACKET
INPUT
PACKET

OVERVIEW


User Space:
/sbin/pfctl
/etc/pf.conf
ioctl(...)
DIOCSTART, DIOCSTOP,
DIOCADDRULE, ...
/dev/pf
Client/Server
Kernel Space:
pfioctl(...)
DIOCADDRULE, ...
DIOCSTART/DIOCSTOP
TCP/IP
OUT
IN
ip*_output(..)
ip*_input(..)
Ethernet
OUT
IN
bridge_filter(..)
pf_status
:struct pf_status
pf_main_anchor
:struct pf_anchor
pf_main_ruleset
:struct pf_ruleset
:struct pf_rulequeue
:struct pf_rulequeue
:
:struct pf_rulequeue
:struct pf_rule
:struct pf_rule
:struct pf_rule
pf_test(...)
pf_test6(...)
pf_state_tree
:struct pf_state_key
:struct pf_state
:struct pf_state
OUTPUT
PACKET
INPUT
PACKET

PFCTL

- pf の有効化/無効化からルールの定義, 現在のステータスの確認, ステータスのクリアなど, pf に関するすべての作業を行うためのコマンド.
- pf pseudo-device (通常は `/dev/pf`) に対して ioctl システムコールを発行し pf を制御する.
- 設定ファイル (通常は `/etc/pf.conf`) を解析するためのパーサは yacc で定義されている.

PFCTL

- 処理の流れ:
  - コマンドラインオプションの解析
  - pf pseudo-device の open
  - pf の無効化(コマンドラインオプションで -d が指定されていた場合)

```
# pfctl -d                                  ※ pf の無効化
```

  - 各種表示コマンドの実行(コマンドラインオプションで -s が指定されていた場合)

```
# pfctl -s rules                            ※ ルールセットの表示
```

  - 各種クリアコマンドの実行(コマンドラインオプションで -F が指定されていた場合)

```
# pfctl -F rules                            ※ ルールセットのクリア
```

  - 各種状態消去コマンドの実行(コマンドラインオプションで -k が指定されていた場合)

```
# pfctl -k 192.168.1.11                     ※ 192.168.1.11 に関する状態を消去
```

  - テーブルコマンドの実行(コマンドラインオプションで -T が指定されていた場合)

```
# pfctl -t spammers -T add 192.168.2.11     ※ spammers テーブルに 192.168.2.11 を追加
```

  - ルールセットの読み込み(コマンドラインオプションで -f が指定されていた場合)

```
# pfctl -f /etc/pf.conf                     ※ /etc/pf.conf を読み込み
```

  - pf の有効化(コマンドラインオプションで -e が指定されていた場合)

```
# pfctl -e                                  ※ pf の有効化
```

  - デバッグレベルの設定(コマンドラインオプションで -x が指定されていた場合)

```
# pfctl -x loud                             ※ デバッグレベルを loud に設定
```

PFCTL_ENABLE/PFCTL_DISABLE

```
int
pfctl_enable(int dev, int opts)
{
     if (ioctl(dev, DIOCSTART)) {
          if (errno == EEXIST)
               errx(1, "pf already enabled");
          else
               err(1, "DIOCSTART");
     }
     if ((opts & PF_OPT_QUIET) == 0)
          fprintf(stderr, "pf enabled\n");

     if (altqsupport && ioctl(dev, DIOCSTARTALTQ))
          if (errno != EEXIST)
               err(1, "DIOCSTARTALTQ");

     return (0);
}
```

```
int
pfctl_disable(int dev, int opts)
{
     if (ioctl(dev, DIOCSTOP)) {
          if (errno == ENOENT)
               errx(1, "pf not enabled");
          else
               err(1, "DIOCSTOP");
     }
     if ((opts & PF_OPT_QUIET) == 0)
          fprintf(stderr, "pf disabled\n");

     if (altqsupport && ioctl(dev, DIOCSTOPALTQ))
               if (errno != ENOENT)
                    err(1, "DIOCSTOPALTQ");

     return (0);
}
```

PFCTL_SHOW_RULES(1)

```
int
pfctl_show_rules(int dev, char *path, int opts, enum pfctl_show format,
    char *anchorname, int depth)
{
      struct pfioc_rule pr;
      u_int32_t nr, mnr, header = 0;
      int rule_numbers = opts & (PF_OPT_VERBOSE2 | PF_OPT_DEBUG);
      int len = strlen(path);
      int brace;
      char *p;

      if (path[0])
            snprintf(&path[len], MAXPATHLEN - len, "/%s", anchorname);
      else
            snprintf(&path[len], MAXPATHLEN - len, "%s", anchorname);

      memset(&pr, 0, sizeof(pr));
      memcpy(pr.anchor, path, sizeof(pr.anchor));
...
      pr.rule.action = PF_PASS;
      if (ioctl(dev, DIOCGETRULES, &pr)) {
            warn("DIOCGETRULES");
            goto error;
      }
      mnr = pr.nr;
      for (nr = 0; nr < mnr; ++nr) {
            pr.nr = nr;
            if (ioctl(dev, DIOCGETRULE, &pr)) {
                  warn("DIOCGETRULE");
                  goto error;
            }
...
            switch (format) {
            case PFCTL_SHOW_LABELS:
...
                  break;
```

PFCTL_SHOW_RULES(2)

```
            case PFCTL_SHOW_RULES:
                  brace = 0;
                  if (pr.rule.label[0] && (opts & PF_OPT_SHOWALL))
                        labels = 1;
                  INDENT(depth, !(opts & PF_OPT_VERBOSE));
                  if (pr.anchor_call[0] &&
                     ((((p = strrchr(pr.anchor_call, '_')) != NULL) &&
                     ((void *)p == (void *)pr.anchor_call ||
                     *(--p) == '/')) || (opts & PF_OPT_RECURSE))) {
                        brace++;
                        if ((p = strrchr(pr.anchor_call, '/')) != NULL)
                              p++;
                        else
                              p = &pr.anchor_call[0];
                  } else
                        p = &pr.anchor_call[0];
                  print_rule(&pr.rule, p, rule_numbers);
                  if (brace)
                        printf(" {\n");
                  else
                        printf("\n");
                  pfctl_print_rule_counters(&pr.rule, opts);
                  if (brace) {
                        pfctl_show_rules(dev, path, opts, format,
                            p, depth + 1);
                        INDENT(depth, !(opts & PF_OPT_VERBOSE));
                        printf("}\n");
                  }
                  break;
            case PFCTL_SHOW_NOTHING:
                  break;
            }
...
      }
      path[len] = '\0';
      return (0);
...
}
```

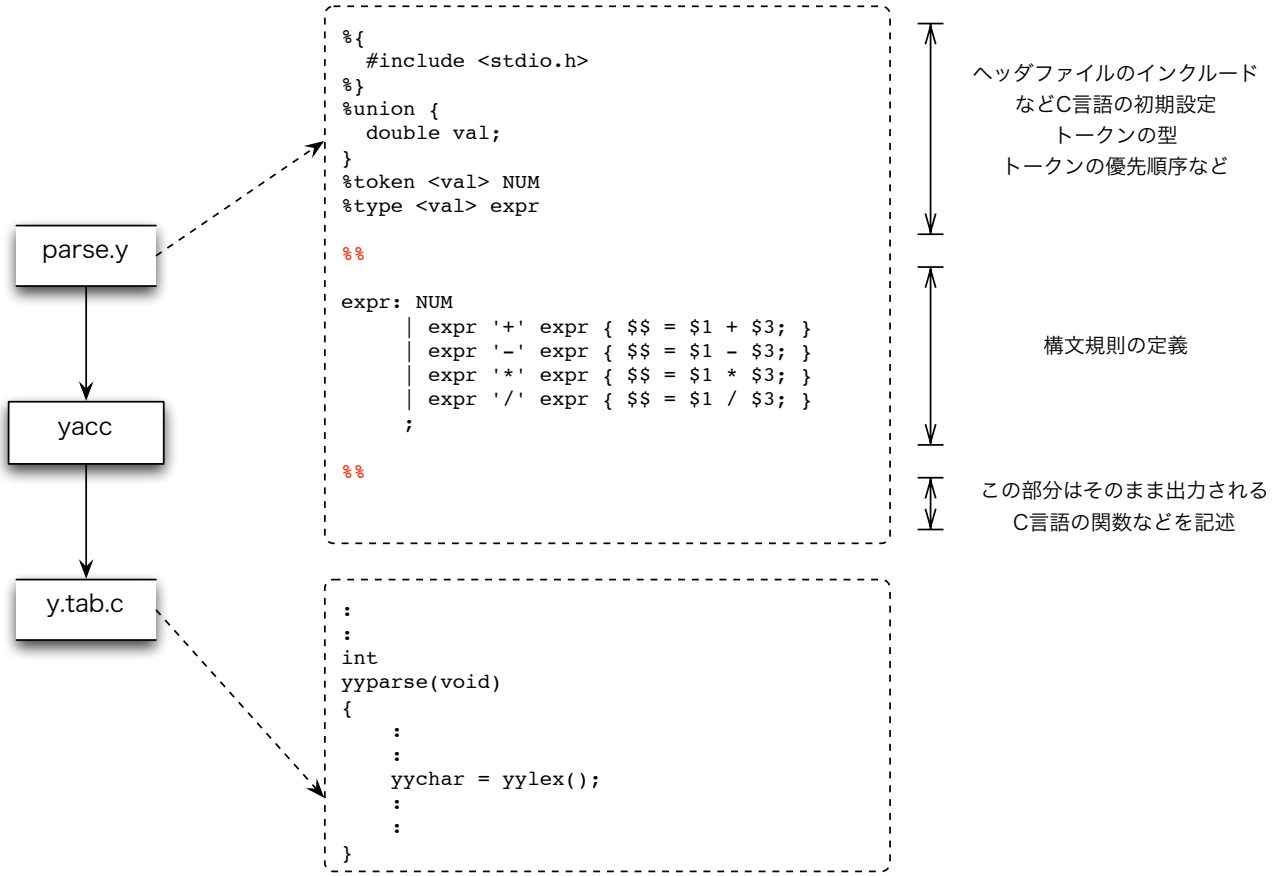
PFCTL_RULES

- でもそのまえに...

WHAT'S YACC

- 構文解析器生成系(compiler-compiler 或は parser generator) のひとつ.
- 1970 年代, AT&T で Stephen C. Johnson により開発される.
- BNF に似た構文定義ファイルから構文解析をする関数を生成する.
- 字句解析器は用意されていないため, 自分で用意する必要がある.

WHAT'S YACC


parse.y
yacc
y.tab.c
%{
#include <stdio.h>
%}
%union {
double val;
}
%token <val> NUM
%type <val> expr
%%
expr: NUM
| expr '+' expr { $$ = $1 + $3; }
| expr '-' expr { $$ = $1 - $3; }
| expr '*' expr { $$ = $1 * $3; }
| expr '/' expr { $$ = $1 / $3; }
;
%%
:
:
int
yyparse(void)
{
:
:
yychar = yylex();
:
:
}
ヘッダファイルのインクルード
などC言語の初期設定
トークンの型
トークンの優先順序など
構文規則の定義
この部分はそのまま出力される
C言語の関数などを記述

```
%{
  #include <stdio.h>
%}
%union {
  double val;
}
%token <val> NUM
%type <val> expr

%%

expr: NUM
     | expr '+' expr { $$ = $1 + $3; }
     | expr '-' expr { $$ = $1 - $3; }
     | expr '*' expr { $$ = $1 * $3; }
     | expr '/' expr { $$ = $1 / $3; }
     ;

%%
```

Example) "4 * 2 * 3.14"

Step 1) "4" ← yylex()

↓ reduce

Step 2) expr

Step 3) expr, "*" ← yylex()

Step 4) expr, "*", "2" ← yylex()

↓ reduce

Step 5) expr, "*", expr

↓ reduce

Step 6) expr

Step 7) expr, "*" ← yylex()

Step 8) expr, "*", "3.14" ← yylex()

↓ reduce

Step 9) expr, "*", expr

↓ reduce

Step 10) expr

YYSTYPE

```
typedef struct {
     union {
          int64_t              number;
          double               probability;
          int             i;
          char           *string;
          u_int                rtableid;
          struct {
               u_int8_t   b1;
               u_int8_t   b2;
               u_int16_t  w;
               u_int16_t  w2;
          }               b;
          struct range {
               int        a;
               int        b;
               int        t;
          }               range;
          struct node_if      *interface;
          struct node_proto   *proto;
          struct node_icmp    *icmp;
          struct node_host    *host;
          struct node_os      *os;
          struct node_port    *port;
          struct node_uid          *uid;
          struct node_gid          *gid;
          struct node_state_opt    *state_opt;
          struct peer          peer;
          struct {
               struct peer     src, dst;
               struct node_os *src_os;
          }               fromto;
          struct {
               struct node_host    *host;
               u_int8_t        rt;
               u_int8_t        pool_opts;
               :
               sa_family_t          af;
               struct pf_poolhashkey    *key;
          }               route;
          struct redirection {
               struct node_host    *host;
               struct range         rport;
          }              *redirection;
          struct {
               int             action;
               struct node_state_opt    *options;
          }               keep_state;
          struct {
               u_int8_t   log;
               u_int8_t   logif;
               u_int8_t   quick;
          }               logquick;
          struct {
               int        neg;
               char      *name;
          }               tagged;
          struct pf_poolhashkey    *hashkey;
          struct node_queue   *queue;
          struct node_queue_opt     queue_options;
          struct node_queue_bw      queue_bwspec;
          struct node_qassign  qassign;
          struct filter_opts   filter_opts;
          struct antispoof_opts     antispoof_opts;
          struct queue_opts    queue_opts;
          struct scrub_opts    scrub_opts;
          struct table_opts    table_opts;
          struct pool_opts     pool_opts;
          struct node_hfsc_opts     hfsc_opts;
     } v;
     int lineno;
} YYSTYPE;
```

TOKEN

```
%token    PASS BLOCK SCRUB RETURN IN OS OUT LOG QUICK ON FROM TO FLAGS
%token    RETURNRST RETURNICMP RETURNICMP6 PROTO INET INET6 ALL ANY ICMPTYPE
%token    ICMP6TYPE CODE KEEP MODULATE STATE PORT RDR NAT BINAT ARROW NODF
%token    MINTTL ERROR ALLOWOPTS FASTROUTE FILENAME ROUTETO DUPTO REPLYTO NO LABEL
%token    NOROUTE URPFFAILED FRAGMENT USER GROUP MAXMSS MAXIMUM TTL TOS DROP TABLE
%token    REASSEMBLE FRAGDROP FRAGCROP ANCHOR NATANCHOR RDRANCHOR BINATANCHOR
%token    SET OPTIMIZATION TIMEOUT LIMIT LOGINTERFACE BLOCKPOLICY RANDOMID
%token    REQUIREORDER SYNPROXY FINGERPRINTS NOSYNC DEBUG SKIP HOSTID
%token    ANTISPOOF FOR INCLUDE
%token    BITMASK RANDOM SOURCEHASH ROUNDROBIN STATICPORT PROBABILITY
%token    ALTQ CBQ PRIQ HFSC BANDWIDTH TBRSIZE LINKSHARE REALTIME UPPERLIMIT
%token    QUEUE PRIORITY QLIMIT RTABLE
%token    LOAD RULESET_OPTIMIZATION
%token    STICKYADDRESS MAXSRCSTATES MAXSRCNODES SOURCETRACK GLOBAL RULE
%token    MAXSRCCONN MAXSRCCONNRATE OVERLOAD FLUSH SLOPPY PFLOW
%token    TAGGED TAG IFBOUND FLOATING STATEPOLICY STATEDEFAULTS ROUTE SETTOS
%token    DIVERTTO DIVERTREPLY
%token    <v.string>          STRING
%token    <v.number>          NUMBER
%token    <v.i>               PORTBINARY
```

TYPE

```
%type     <v.interface>       interface if_list if_item_not if_item
%type     <v.number>          number icmptype icmp6type uid gid tos not yesno
%type     <v.probability>     probability
%type     <v.i>               no dir af fragcache optimizer sourcetrack flush unaryop statelock
%type     <v.b>               action nataction natpasslog scrubaction flags flag blockspec
%type     <v.range>           portplain portstar portrange
%type     <v.hashkey>         hashkey
%type     <v.proto>           proto proto_list proto_item
%type     <v.number>          protoval
%type     <v.icmp>            icmpspec icmp_list icmp_item icmp6_list icmp6_item
%type     <v.number>          reticmpspec reticmp6spec
%type     <v.fromto>          fromto
%type     <v.peer>            ipportspec from to
%type     <v.host>            ipspec toipspec xhost host dynaddr host_list redir_host_list
%type     <v.host>            redirspec route_host route_host_list routespec
%type     <v.os>              os xos os_list
%type     <v.port>            portspec port_list port_item
%type     <v.uid>             uids uid_list uid_item
%type     <v.gid>             gids gid_list gid_item
%type     <v.route>           route
%type     <v.redirection>     redirection redirpool
%type     <v.string>          label stringall tag anchorname string varstring numberstring
%type     <v.keep_state>      keep
%type     <v.state_opt>       state_opt_spec state_opt_list state_opt_item
%type     <v.logquick>        logquick quick log logopts logopt
%type     <v.interface>       antispoof_ifspc antispoof_iflst antispoof_if
%type     <v.qassign>         qname
%type     <v.queue>           qassign qassign_list qassign_item
%type     <v.queue_options>   scheduler
%type     <v.number>          cbqflags_list cbqflags_item priqflags_list priqflags_item
%type     <v.hfsc_opts>       hfscopts_list hfscopts_item hfsc_opts
%type     <v.queue_bwspec>    bandwidth
%type     <v.filter_opts>     filter_opts filter_opt filter_opts_l
%type     <v.antispoof_opts>  antispoof_opts antispoof_opt antispoof_opts_l
%type     <v.queue_opts>      queue_opts queue_opt queue_opts_l
%type     <v.scrub_opts>      scrub_opts scrub_opt scrub_opts_l
%type     <v.table_opts>      table_opts table_opt table_opts_l
%type     <v.pool_opts>       pool_opts pool_opt pool_opts_l
%type     <v.tagged>          tagged
%type     <v.rtableid>        rtable
```

PFRULE

```
pfrule       : action dir logquick interface route af proto fromto
               filter_opts
        {
                struct pf_rule        r;
                struct node_state_opt *o;
                struct node_proto*proto;
                int           srctrack = 0;
                int           statelock = 0;
                int           adaptive = 0;
                int           defaults = 0;

                if (check_rulestate(PFCTL_STATE_FILTER))
                    YYERROR;

                memset(&r, 0, sizeof(r));

                : /*
                :  * この部分で action, dir, ... の値を r(pf_rule) に格納したり
                :  * 構文的には正しくても意味的に誤りな箇所がないか確認する.
                :  */

                expand_rule(&r, $4, $5.host, $7, $8.src_os,
                    $8.src.host, $8.src.port, $8.dst.host, $8.dst.port,
                    $9.uid, $9.gid, $9.icmpspec, "");
        }
        ;
```

Example) pass in on bge0 proto tcp from any to 192.168.1.11 port 80

Step 1) "pass"
Step 2) PASS
Step 3) action
Step 3) action, "in"
Step 4) action, IN
Step 5) action, dir
Step 6) action, dir, "on"
Step 7) action, dir, ON
Step 8) action, dir, ON, "bge0"
Step 9) action, dir, ON, if_item
Step 10) action, dir, ON, if_item_not
Step 11) action, dir, interface
Step 12) action, dir, interface, "proto"
Step 13) action, dir, interface, PROTO
Step 14) action, dir, interface, PROTO, "tcp"
Step 15) action, dir, interface, PROTO, protoval
Step 16) action, dir, interface, PROTO, proto_item
Step 17) action, dir, interface, proto
Step 18) action, dir, interface, proto, "from"
Step 19) action, dir, interface, proto, FROM
Step 20) action, dir, interface, proto, FROM, "any"
Step 21) action, dir, interface, proto, FROM, ANY
Step 22) action, dir, interface, proto, FROM, ipspec
Step 23) action, dir, interface, proto, FROM, ipportspec
Step 24) action, dir, interface, proto, from
Step 25) action, dir, interface, proto, from, "to"
Step 26) action, dir, interface, proto, from, TO
Step 27) action, dir, interface, proto, from, TO, "192.168.1.11"
Step 28) action, dir, interface, proto, from, TO, host
Step 29) action, dir, interface, proto, from, TO, xhost
Step 30) action, dir, interface, proto, from, TO, ipspec
Step 31) action, dir, interface, proto, from, TO, ipspec, "port"
Step 32) action, dir, interface, proto, from, TO, ipspec, PORT
Step 33) action, dir, interface, proto, from, TO, ipspec, PORT, "80"
Step 34) action, dir, interface, proto, from, TO, ipspec, PORT, numberstring
Step 35) action, dir, interface, proto, from, TO, ipspec, PORT, portrange
Step 36) action, dir, interface, proto, from, TO, ipspec, PORT, port_item
Step 37) action, dir, interface, proto, from, TO, ipspec, PORT, portspec
Step 38) action, dir, interface, proto, from, TO, ipportspec
Step 39) action, dir, interface, proto, from, to
Step 40) action, dir, interface, proto, fromto
Step 41) pfrule

```
ipportspec : ipspec             {
                $$.host = $1;
                $$.port = NULL;
           }
           | ipspec PORT portspec          {
                $$.host = $1;
                $$.port = $3;
           }
           | PORT portspec             {
                $$.host = NULL;
                $$.port = $2;
           }
           ;
```

```
to         : /* empty */              {
                ... 省略 ...
           }
           | TO ipportspec       {
                if (disallow_urpf_failed($2.host,
                    "\"urpf-failed\" is "
                    "not permitted in a destination address"))
                    YYERROR;
                $$ = $2;
           }
           ;
```

```
fromto          : ALL                   {
                  ... 省略 ...
           }
           | from os to                 {
                $$.src = $1;
                $$.src_os = $2;
                $$.dst = $3;
           }
           ;
```

EXPAND_RULE

```
void
expand_rule(struct pf_rule *r, struct node_if *interfaces, struct node_host *rpool_hosts, struct node_proto *protos,
    struct node_os *src_oses, struct node_host *src_hosts, struct node_port *src_ports, struct node_host *dst_hosts,
    struct node_port *dst_ports, struct node_uid *uids, struct node_gid *gids, struct node_icmp *icmp_types,
    const char *anchor_call)
{
          :
     LOOP_THROUGH(struct node_if, interface, interfaces,
          : /* interfaces, protos, icmp_types, ..., gids のリストを展開 */
     LOOP_THROUGH(struct node_gid, gid, gids,
               :
          if (*interface->ifname)
               strlcpy(r->ifname, interface->ifname, sizeof(r->ifname));
          else if (if_indextoname(src_host->ifindex, ifname))
               strlcpy(r->ifname, ifname, sizeof(r->ifname));
          else if (if_indextoname(dst_host->ifindex, ifname))
               strlcpy(r->ifname, ifname, sizeof(r->ifname));
          else
               memset(r->ifname, '\0', sizeof(r->ifname));
               :
          if (rule_consistent(r, anchor_call[0]) < 0 || error)
               yyerror("skipping rule due to errors");
          else {
               r->nr = pf->astack[pf->asd]->match++;
               pfctl_add_rule(pf, r, anchor_call);
               added++;
          }
     ))))))))));
     FREE_LIST(struct node_if, interfaces);
          : /* interfaces, protos, icmp_types, ..., gids のリストを解放 */
     FREE_LIST(struct node_host, rpool_hosts);
          :
}
```

PFCTL_ADD_RULE

```
int
pfctl_add_rule(struct pfctl *pf, struct pf_rule *r, const char *anchor_call)
{
     u_int8_t        rs_num;
     struct pf_rule      *rule;
     struct pf_ruleset   *rs;
     char                *p;

     rs_num = pf_get_ruleset_number(r->action);
     if (rs_num == PF_RULESET_MAX)
          errx(1, "Invalid rule type %d", r->action);

     rs = &pf->anchor->ruleset;

     if (anchor_call[0] && r->anchor == NULL) {
          : /*
          :  * アンカー定義だったときの処理...
          :  */
     }

     if ((rule = calloc(1, sizeof(*rule))) == NULL)
          err(1, "calloc");
     bcopy(r, rule, sizeof(*rule));
     TAILQ_INIT(&rule->rpool.list);
     pfctl_move_pool(&r->rpool, &rule->rpool);

     TAILQ_INSERT_TAIL(rs->rules[rs_num].active.ptr, rule, entries);
     return (0);
}
```

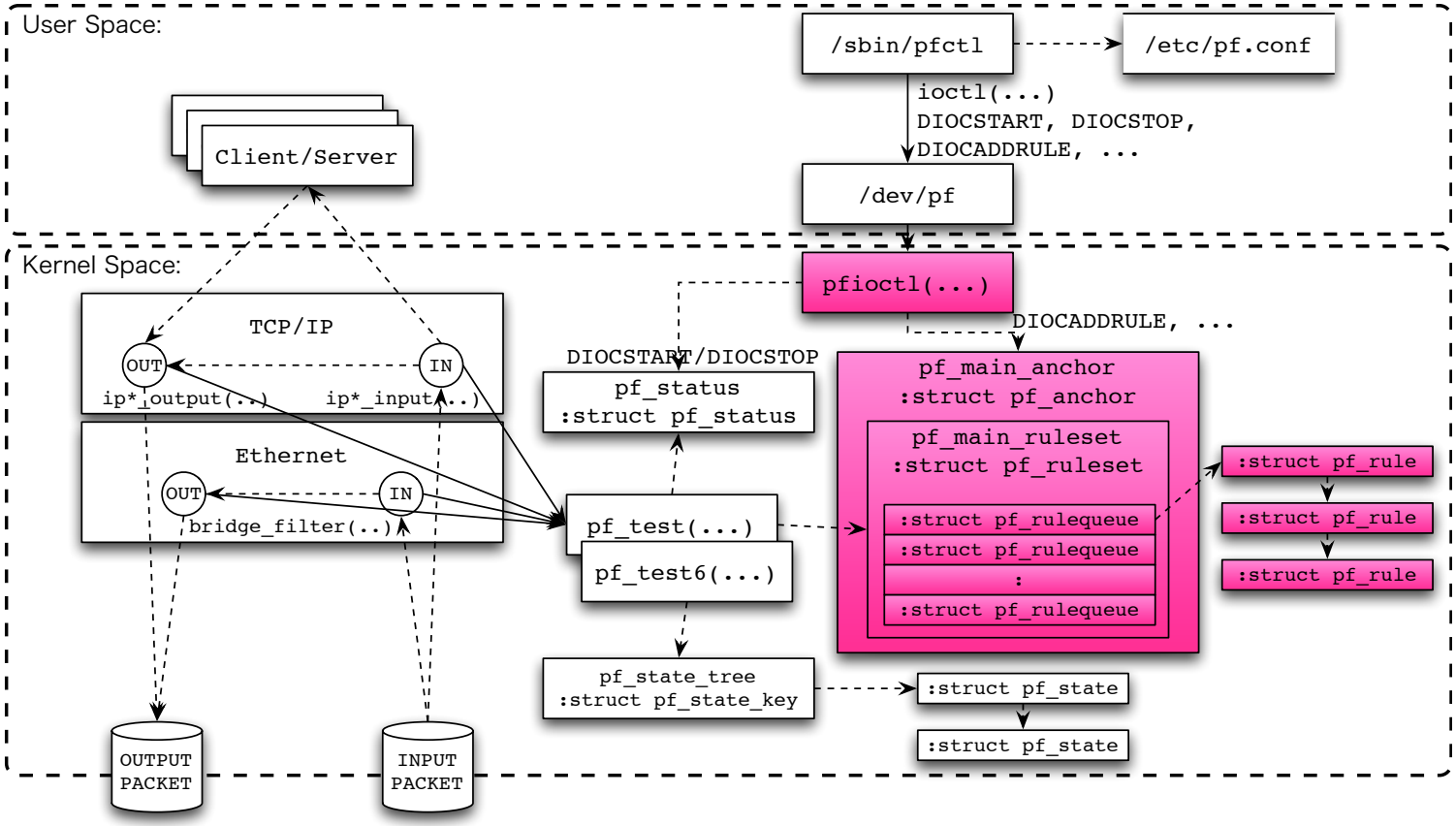
PFCTL_LOAD_RULESET

```
int
pfctl_load_ruleset(struct pfctl *pf, char *path, struct pf_ruleset *rs, int rs_num, int depth)
{
     struct pf_rule *r;
     int       error, len = strlen(path);
     int       brace = 0;

          : /* path の生成など... */

     while ((r = TAILQ_FIRST(rs->rules[rs_num].active.ptr)) != NULL) {
          TAILQ_REMOVE(rs->rules[rs_num].active.ptr, r, entries);
          if ((error = pfctl_load_rule(pf, path, r, depth)))
               goto error;
          if (r->anchor) {
               if ((error = pfctl_load_ruleset(pf, path, &r->anchor->ruleset, rs_num, depth + 1)))
                    goto error;
          } else if (pf->opts & PF_OPT_VERBOSE)
               printf("\n");
          free(r);
     }
     if (brace && pf->opts & PF_OPT_VERBOSE) {
          INDENT(depth - 1, (pf->opts & PF_OPT_VERBOSE));
          printf("}\n");
     }
     path[len] = '\0';
     return (0);

 error:
     path[len] = '\0';
     return (error);

}
```

PFCTL_LOAD_RULE

```
int
pfctl_load_rule(struct pfctl *pf, char *path, struct pf_rule *r, int depth)
{
     u_int8_t       rs_num = pf_get_ruleset_number(r->action);
     char           *name;
     struct pfioc_rule   pr;
     int            len = strlen(path);
     bzero(&pr, sizeof(pr));
          : /* チケットの準備, path の準備など */
     if ((pf->opts & PF_OPT_NOACTION) == 0) {
          if (pfctl_add_pool(pf, &r->rpool, r->af))
               return (1);
          pr.pool_ticket = pf->paddr.ticket;
          memcpy(&pr.rule, r, sizeof(pr.rule));
          if (r->anchor && strlcpy(pr.anchor_call, name,
              sizeof(pr.anchor_call)) >= sizeof(pr.anchor_call))
               errx(1, "pfctl_load_rule: strlcpy");
          if (ioctl(pf->dev, DIOCADDRULE, &pr))
               err(1, "DIOCADDRULE");
     }

     if (pf->opts & PF_OPT_VERBOSE) {
          INDENT(depth, !(pf->opts & PF_OPT_VERBOSE2));
          print_rule(r, r->anchor ? r->anchor->name : "",
              pf->opts & PF_OPT_VERBOSE2);
     }
     path[len] = '\0';
     pfctl_clear_pool(&r->rpool);
     return (0);
}
```

OVERVIEW


User Space:
/sbin/pfctl
/etc/pf.conf
ioctl(...)
DIOCSTART, DIOCSTOP,
DIOCADDRULE, ...
Client/Server
/dev/pf
Kernel Space:
pfioctl(...)
DIOCADDRULE, ...
TCP/IP
OUT
IN
ip*_output(..)
ip*_input(..)
DIOCSTART/DIOCSTOP
pf_status
:struct pf_status
pf_main_anchor
:struct pf_anchor
pf_main_ruleset
:struct pf_ruleset
:struct pf_rule
:struct pf_rule
:struct pf_rule
Ethernet
OUT
IN
bridge_filter(..)
pf_test(...)
pf_test6(...)
:struct pf_rulequeue
:struct pf_rulequeue
:
:struct pf_rulequeue
pf_state_tree
:struct pf_state_key
:struct pf_state
:struct pf_state
OUTPUT
PACKET
INPUT
PACKET

PFIOCTL

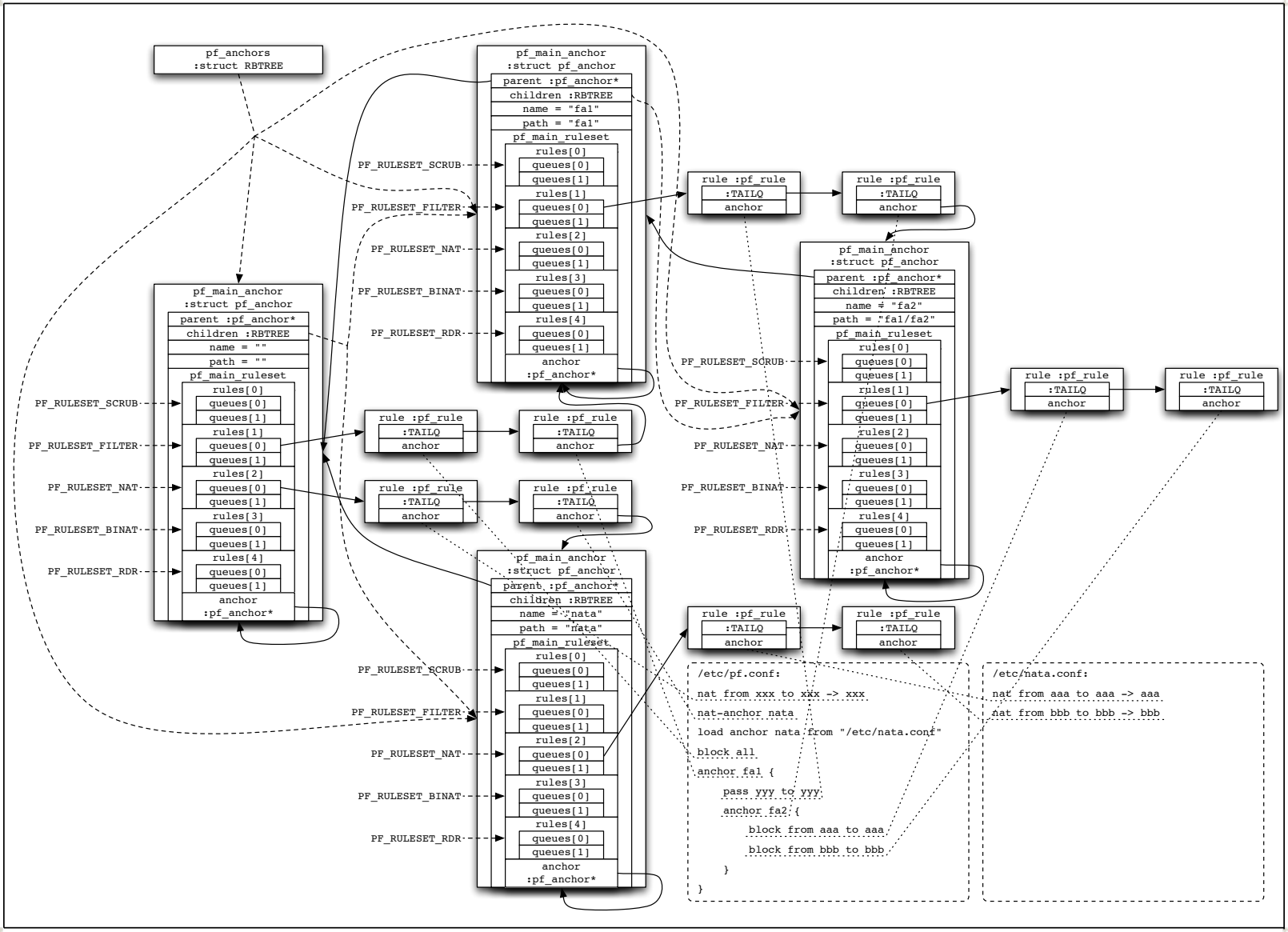
- pf pseudo-device(通常は `/dev/pf`) に対して ioctl を実行した際に実行される関数.
- pf の有効化/無効化(`DIOCSTART/DIOCSTOP`), ルールの追加(`DIOCADDRULE`), ルールの参照(`DIOCGETRULES/DIOCGETRULE`) など, 63 個のコマンドが用意されている.
- コマンドによっては `pfioc_xxx` という名前の構造体を引数に取る. 例えば `DIOCADDRULE` コマンドは `pfioc_rule` という構造体を引数として受け取り, その内容をルールに追加する. また `DIOCGETRULES/DIOCGETRULE` コマンドも `pfioc_rule` の構造体を引数として受け取り, その中にルールを内容を埋め込むことでプロセスにルールを渡す.

PFIOCTL

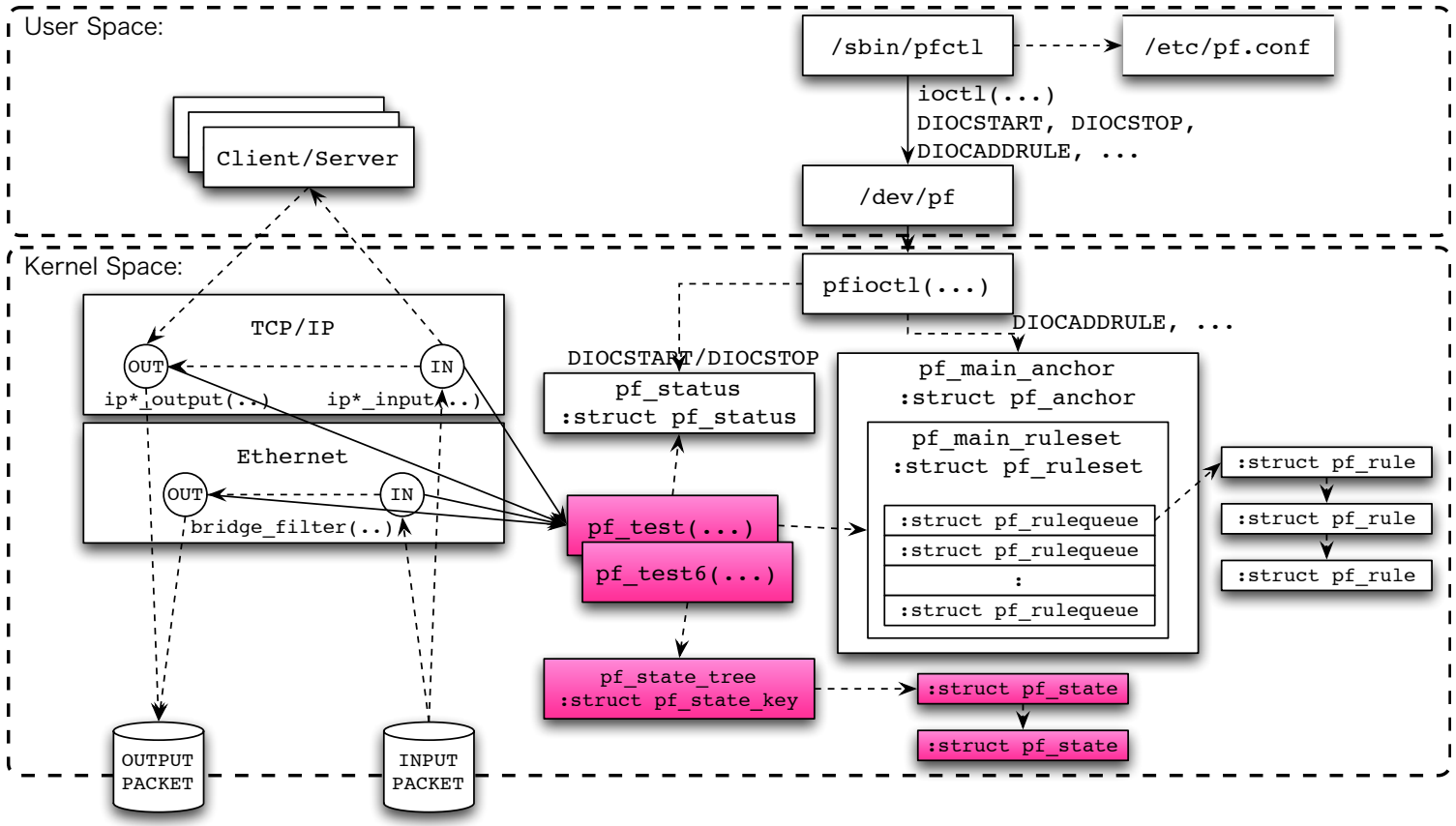
```
int
pfioctl(dev_t dev, u_long cmd, caddr_t addr, int flags, struct proc *p)
{
     /* 省略 */
     if (flags & FWRITE)
          rw_enter_write(&pf_consistency_lock);
     else
          rw_enter_read(&pf_consistency_lock);
     s = splsoftnet();
     switch (cmd) {
     case DIOCSTART:
          if (pf_status.running)
               error = EEXIST;
          else {
               pf_status.running = 1;
               /* 省略 */

          }
          break;
     case DIOCSTOP:
          /* 省略 */
     /* コマンドの数だけ case 節がある... */
     default:
          error = ENODEV;
          break;
     }
fail:
     splx(s);
     if (flags & FWRITE)
          rw_exit_write(&pf_consistency_lock);
     else
          rw_exit_read(&pf_consistency_lock);
     return (error);
}
```

TRANSACTION

- 複数の同時 pfctl 実行による不整合を防ぐため, チケットによるトランザクション管理がなされている.
- ルールの更新途中の中途半端な状態でパケットの評価が行われないよう, active と inactive の 2 つのキューを用意したうえで更新は inactive キューで行い, 最後に active と inactive を切り替えるという処理を行っている.
- ルール更新の処理の流れ:
  - DIOCXBEGIN コマンドを実行: すでに登録されているルールをすべて削除し, チケットを発行. inactive キューの変数 open に 1 を代入.
  - DIOCADDRULE コマンドを実行(ルールの数だけ実行): チケットを確認し, 問題があればエラーとする. 問題なければ inactive キューの最後にルールを追加する.
  - DIOCXCOMMIT/DIOCXROLLBACK コマンドを実行: チケットを確認し, 問題があればエラーとする. 問題がなければ inactive と active を切り替える. 途中でなにかしらのエラーが発生した場合は DIOCXROLLBACK により変更が破棄される.


pf_anchors
:struct RBTREE
pf_main_anchor
:struct pf_anchor
parent :pf_anchor*
children :RBTREE
name = "fa1"
path = "fa1"
pf_main_ruleset
rules[0]
queues[0]
queues[1]
rules[1]
queues[0]
queues[1]
rules[2]
queues[0]
queues[1]
rules[3]
queues[0]
queues[1]
rules[4]
queues[0]
queues[1]
anchor
:pf_anchor*
PF_RULESET_SCRUB
PF_RULESET_FILTER
PF_RULESET_NAT
PF_RULESET_BINAT
PF_RULESET_RDR
rule :pf_rule
:TAILQ
anchor
pf_main_anchor
:struct pf_anchor
parent :pf_anchor*
children :RBTREE
name = "fa2"
path = "fa1/fa2"
pf_main_ruleset
pf_main_anchor
:struct pf_anchor
parent :pf_anchor*
children :RBTREE
name = "nata"
path = "nata"
pf_main_ruleset
/etc/pf.conf:
nat from xxx to xxx -> xxx
nat-anchor nata
load anchor nata from "/etc/nata.conf"
block all
anchor fa1 {
pass yyy to yyy
anchor fa2 {
block from aaa to aaa
block from bbb to bbb
}
}
/etc/nata.conf:
nat from aaa to aaa -> aaa
nat from bbb to bbb -> bbb

OVERVIEW


User Space:
/sbin/pfctl
/etc/pf.conf
ioctl(...)
DIOCSTART, DIOCSTOP,
DIOCADDRULE, ...
Client/Server
/dev/pf
Kernel Space:
pfioctl(...)
DIOCADDRULE, ...
TCP/IP
OUT
IN
ip*_output(..)
ip*_input(..)
DIOCSTART/DIOCSTOP
pf_status
:struct pf_status
pf_main_anchor
:struct pf_anchor
pf_main_ruleset
:struct pf_ruleset
:struct pf_rule
:struct pf_rule
:struct pf_rule
Ethernet
OUT
IN
bridge_filter(..)
pf_test(...)
pf_test6(...)
:struct pf_rulequeue
:struct pf_rulequeue
:
:struct pf_rulequeue
pf_state_tree
:struct pf_state_key
:struct pf_state
:struct pf_state
OUTPUT
PACKET
INPUT
PACKET

PF_TEST/PF_TEST6

- パケットがインターフェースを通過しようとする際に呼び出され, 通過を許可するか判断する.
- まず状態データベースを確認し, 有効な状態が見つかればルールを確認せず許可する. 状態データベースに有効な状態が見つからなければ, ルールを確認し, 許可するか判断する. 許可するときは状態データベースに状態を登録する.
- `nat/binat/rdr` のアドレス/ポート書き換えも `pf_test()/pf_test6()` 内で実行される. アドレス/ポート書き換えが行われる場合, 書き換えられた後のアドレス/ポートに対してルールの確認が行われる(但し, `nat/binat/rdr` の直後に `pass` オプションが指定された場合, フィルタはチェックされない).

PF_TEST(1)

- 処理の流れ:
  - IP パケットの正規化(`pf_normalize_ip()`).
  - TCP の場合:
    - TCP パケットの正規化(`pf_normalize_tcp()`).
    - 状態データベースの評価(`pf_test_state_tcp()`).
    - 有効な状態がデータベースになければルールの評価(`pf_test_rule()`).
  - UDP の場合:
    - 状態データベースの評価(`pf_test_state_udp()`).
    - 有効な状態がデータベースになければルールの評価(`pf_test_rule()`).

PF_TEST(2)

- 処理の流れ(続き):
  - ICMP の場合:
    - 状態データベースの評価(`pf_test_state_icmp()`).
    - 有効な状態がデータベースになければルールの評価(`pf_test_rule()`).
  - その他の場合:
    - 状態データベースの評価(`pf_test_state_other()`).
    - 有効な状態がデータベースになければルールの評価(`pf_test_rule()`).
  - パケットのタグ付け処理(`pf_tag_packet()`).
  - 統計情報の更新.
  - ルーティング処理(`pf_route()`).

PF_TEST_STATE_TCP

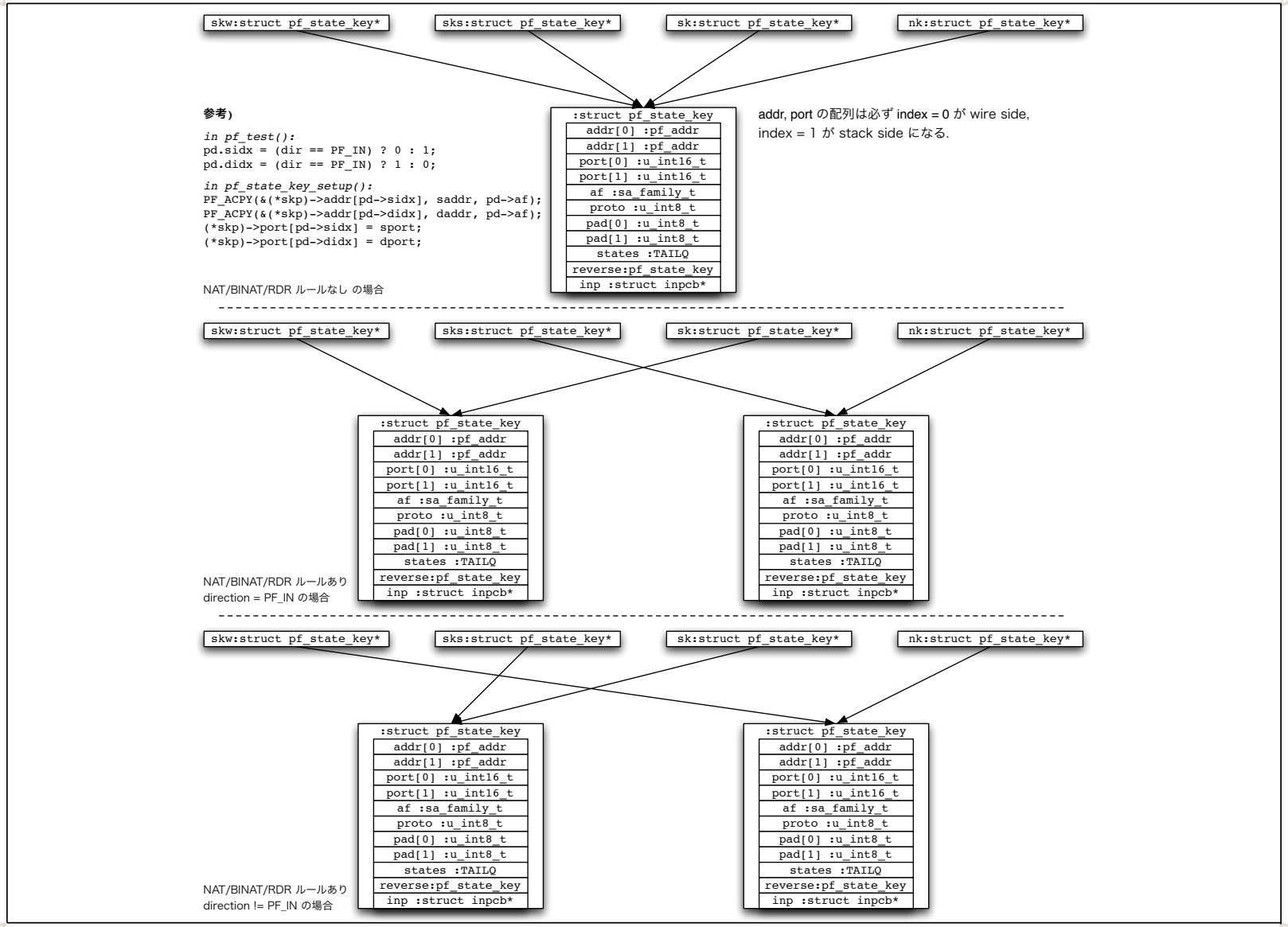
- まず RBTREE `pf_state_tree` から src addr, dst addr, src port, dst port, address family, ... をキーに `pf_state_key` を検索.
- 見つかった `pf_state_key` の TAILQ `states` をたどり `direction`, `kif` が一致する `pf_state` を得る.
- アドレス変換が必要な状態のときは `pf_state` の情報をもとにアドレス変換を行う.

PF_TEST_RULE

- アドレス変換ルールを取得(pf_get_translation()).
  - (アドレス変換ルールが存在した場合) アドレスとポートを必要に応じて書き換える.
- pf_main_ruleset の rules[PF_RULESET_FILTER] のアクティブな方のキューを先頭から評価していく.
  - 但し, pf_rule の skip 配列を用いて無駄な評価はスキップする.
  - もしルールの anchor が NULL でない(pf_anchor が存在する)場合は pf_anchor に入る(pf_step_into_anchor()).
- 必要に応じて状態データベースに新しい状態を登録する (pf_create_state()).

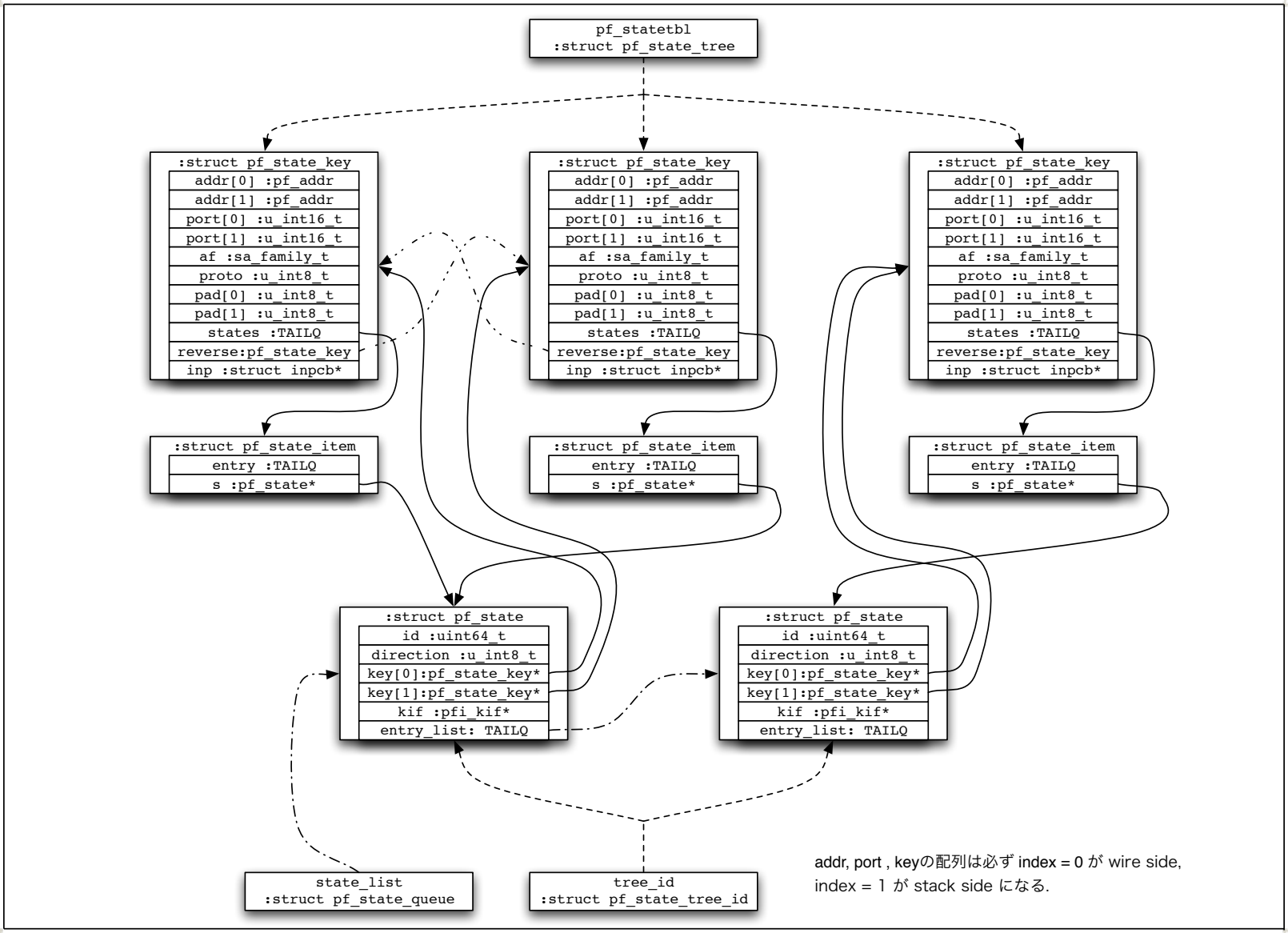
PF_GET_TRANSLATION

- NAT/BINAT/RDR の各ルールセットの中から条件に一致するルールを検索:
  - direction が `PF_OUT` の場合は BINAT 優先, BINAT でマッチするルールがなければ NAT から探す.
  - direction が `PF_OUT` 以外の場合は RDR 優先, RDR でマッチするルールがなければ BINAT から探す.
- `pf_state_key`(`skw`, `sks`, `sk`, `nk`) を準備(`pf_state_key_setup()`)
- ルールの `action`(`PF_NAT/PF_BINAT/PF_RDR`) に応じて `nk` の stack side 側のアドレス, ポートを変更する.

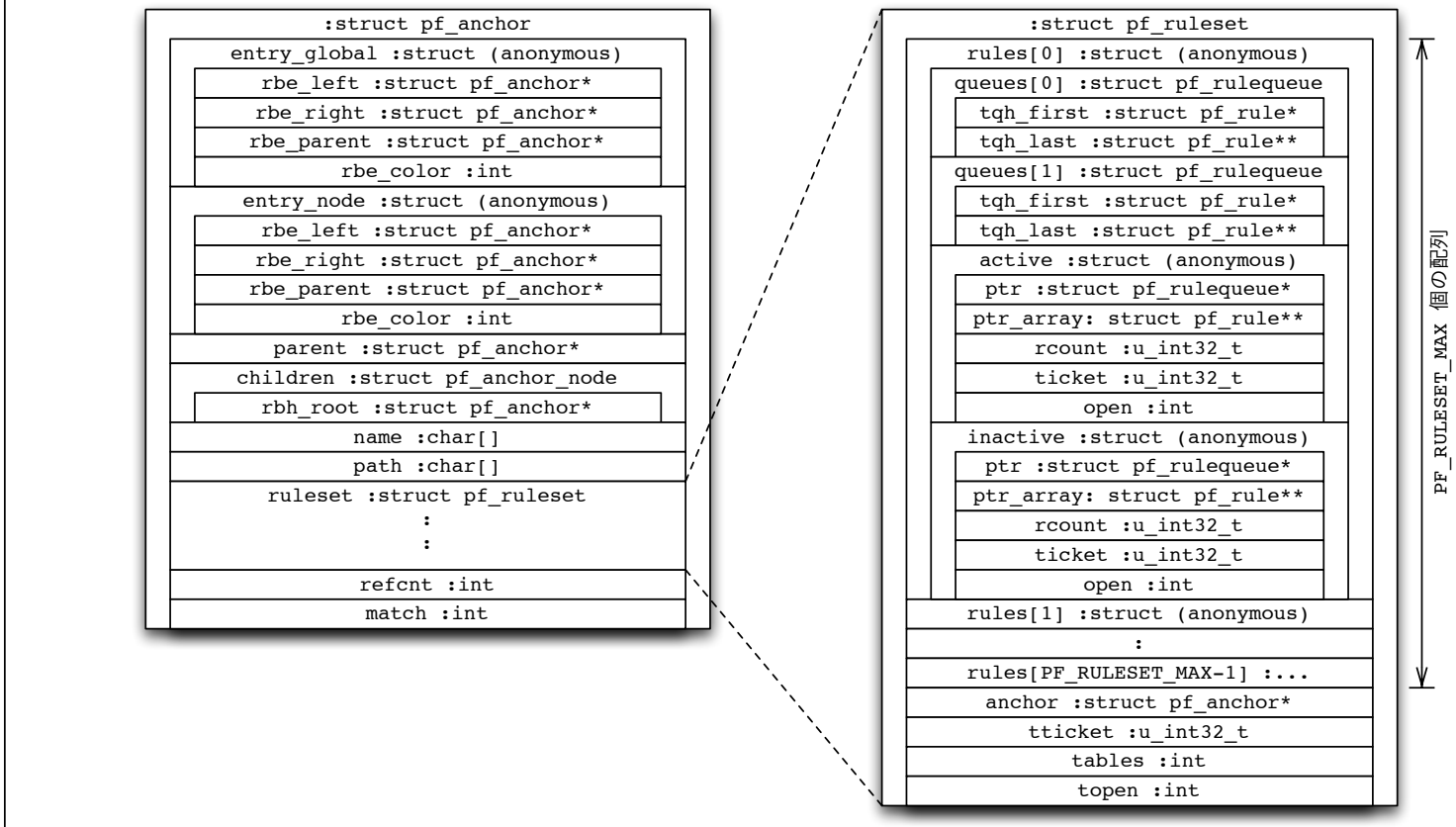

skw:struct pf_state_key*
sks:struct pf_state_key*
sk:struct pf_state_key*
nk:struct pf_state_key*
参考）
in pf_test():
pd.sidx = (dir == PF_IN) ? 0 : 1;
pd.didx = (dir == PF_IN) ? 1 : 0;
in pf_state_key_setup():
PF_ACPY(&(*skp)->addr[pd->sidx], saddr, pd->af);
PF_ACPY(&(*skp)->addr[pd->didx], daddr, pd->af);
(*skp)->port[pd->sidx] = sport;
(*skp)->port[pd->didx] = dport;
:struct pf_state_key
addr[0] :pf_addr
addr[1] :pf_addr
port[0] :u_int16_t
port[1] :u_int16_t
af :sa_family_t
proto :u_int8_t
pad[0] :u_int8_t
pad[1] :u_int8_t
states :TAILQ
reverse:pf_state_key
inp :struct inpcb*
addr, port の配列は必ず index = 0 が wire side,
index = 1 が stack side になる.
NAT/BINAT/RDR ルールなし の場合
NAT/BINAT/RDR ルールあり
direction = PF_IN の場合
NAT/BINAT/RDR ルールあり
direction != PF_IN の場合

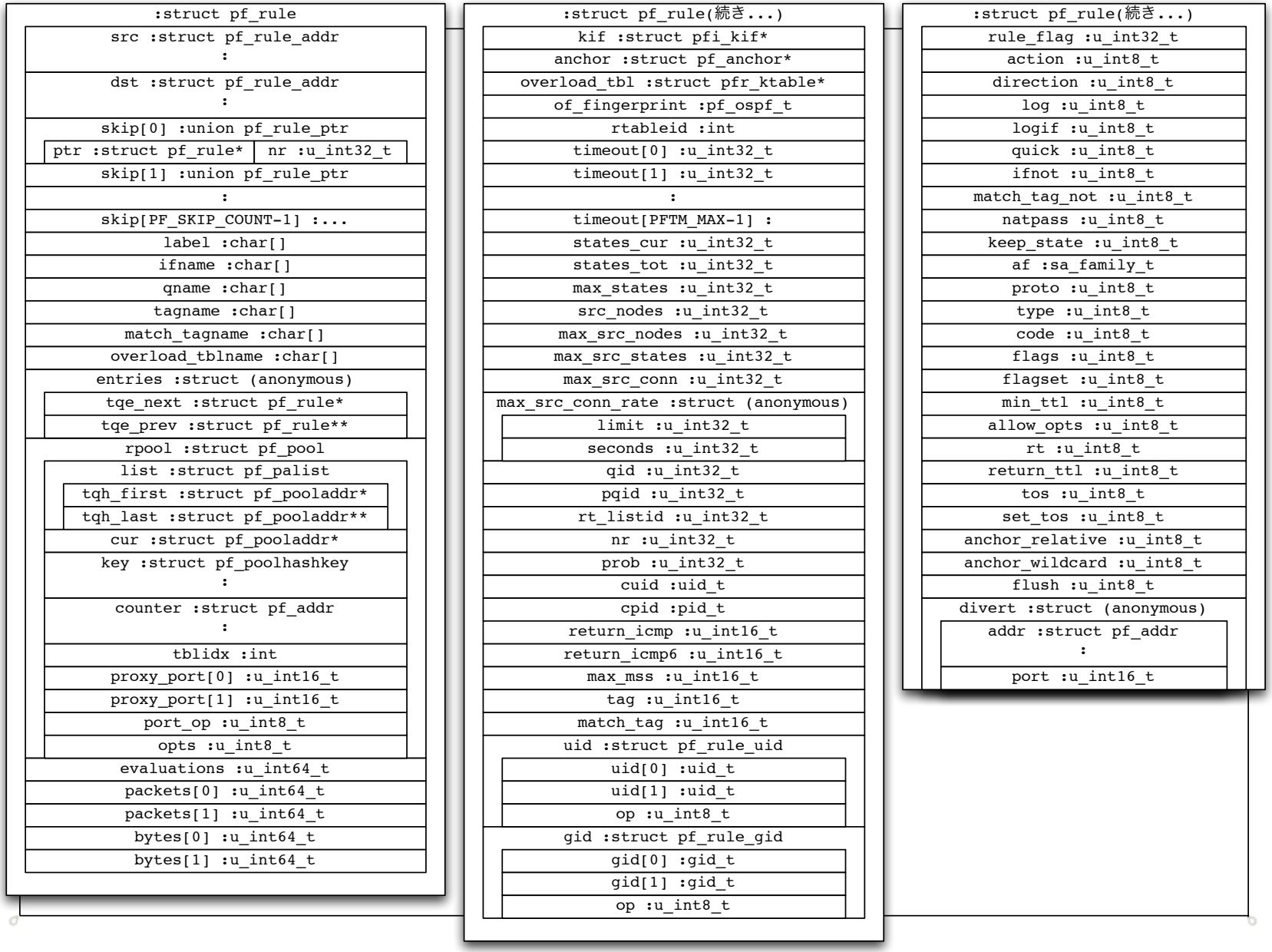
PF_CREATE_STATE

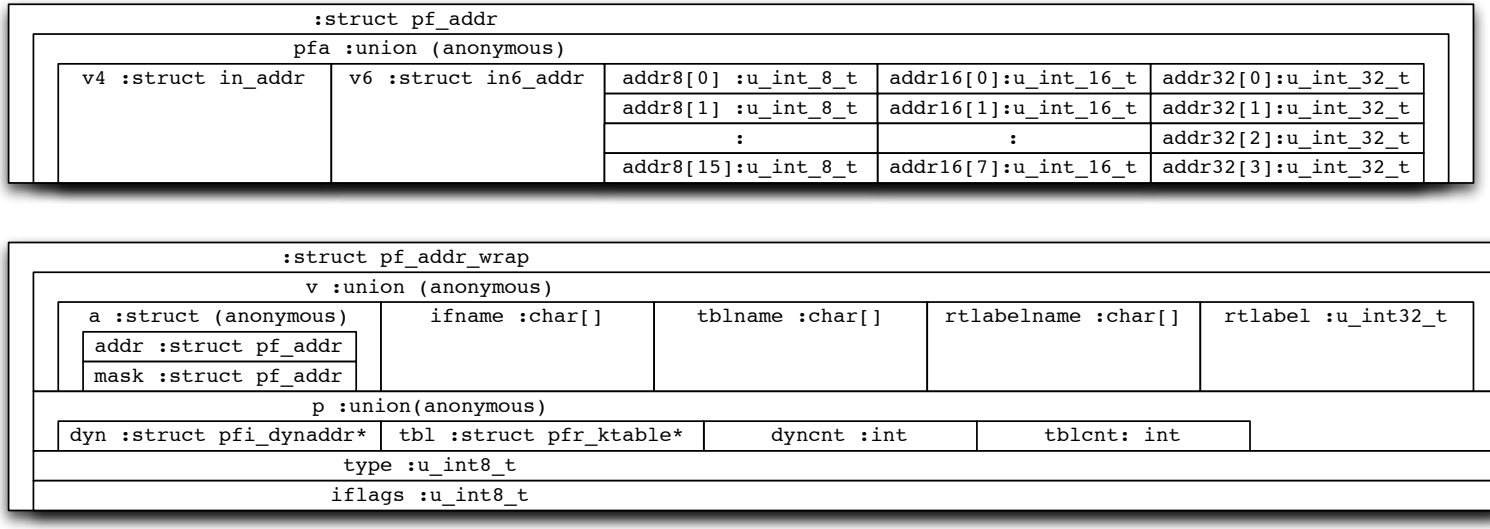
- `pf_state_pl`(アドレスプール) から `pf_state`, `s` を取得.
- `s` に情報(マッチしたルールへのポインタ, src, dst に関する状態情報, タイムアウトに関する設定, 状態作成時刻と失効時刻, など)を格納.
- NAT/BINAT/RDR にマッチしていない(`nk == sk`)場合:
  - `s` の `key[PF_SK_WIRE]`, `key[PF_SK_STACK]` にひとつの `pf_state_key` へのアドレスを代入(もともと `pf_state_key` は一つしか allocate されていない).
  - `pf_state_key` を RBTREE `pf_statetbl` に入れる.
- NAT/BINAT/RDR にマッチしている(`nk != sk`)場合:
  - `s` の `key[PF_SK_WIRE]` には `skw` へのアドレスを代入し, `key[PF_SK_STACK]` には `sks` へのアドレスを代入.
  - `skw`, `sks` を RBTREE `pf_statetbl` に入れる.
- RBTREE `tree_id` と TAILQ `entry_list` に `s` を入れる.

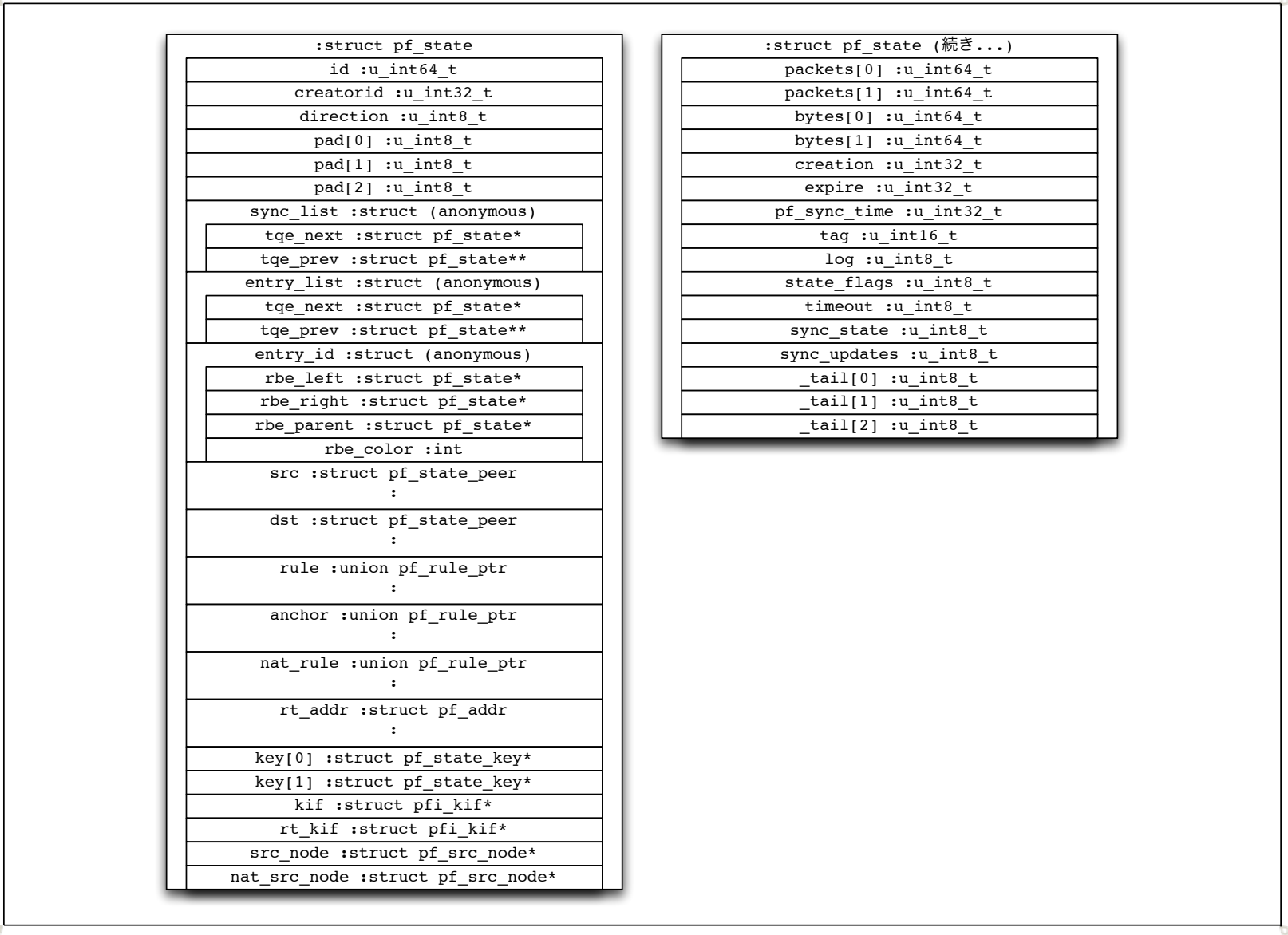

pf_statetbl
:struct pf_state_tree
:struct pf_state_key
addr[0] :pf_addr
addr[1] :pf_addr
port[0] :u_int16_t
port[1] :u_int16_t
af :sa_family_t
proto :u_int8_t
pad[0] :u_int8_t
pad[1] :u_int8_t
states :TAILQ
reverse:pf_state_key
inp :struct inpcb*
:struct pf_state_item
entry :TAILQ
s :pf_state*
:struct pf_state
id :uint64_t
direction :u_int8_t
key[0]:pf_state_key*
key[1]:pf_state_key*
kif :pfi_kif*
entry_list: TAILQ
state_list
:struct pf_state_queue
tree_id
:struct pf_state_tree_id
addr, port , keyの配列は必ず index = 0 が wire side,
index = 1 が stack side になる.

参考情報


:struct pf_anchor
entry_global :struct (anonymous)
rbe_left :struct pf_anchor*
rbe_right :struct pf_anchor*
rbe_parent :struct pf_anchor*
rbe_color :int
entry_node :struct (anonymous)
rbe_left :struct pf_anchor*
rbe_right :struct pf_anchor*
rbe_parent :struct pf_anchor*
rbe_color :int
parent :struct pf_anchor*
children :struct pf_anchor_node
rbh_root :struct pf_anchor*
name :char[]
path :char[]
ruleset :struct pf_ruleset
:
:
refcnt :int
match :int
:struct pf_ruleset
rules[0] :struct (anonymous)
queues[0] :struct pf_rulequeue
tqh_first :struct pf_rule*
tqh_last :struct pf_rule**
queues[1] :struct pf_rulequeue
tqh_first :struct pf_rule*
tqh_last :struct pf_rule**
active :struct (anonymous)
ptr :struct pf_rulequeue*
ptr_array: struct pf_rule**
rcount :u_int32_t
ticket :u_int32_t
open :int
inactive :struct (anonymous)
ptr :struct pf_rulequeue*
ptr_array: struct pf_rule**
rcount :u_int32_t
ticket :u_int32_t
open :int
rules[1] :struct (anonymous)
:
rules[PF_RULESET_MAX-1] :...
anchor :struct pf_anchor*
tticket :u_int32_t
tables :int
topen :int
PF_RULESET_MAX 個の配列


:struct pf_rule
src :struct pf_rule_addr
:
dst :struct pf_rule_addr
:
skip[0] :union pf_rule_ptr
ptr :struct pf_rule*
nr :u_int32_t
skip[1] :union pf_rule_ptr
:
skip[PF_SKIP_COUNT-1] :...
label :char[]
ifname :char[]
qname :char[]
tagname :char[]
match_tagname :char[]
overload_tblname :char[]
entries :struct (anonymous)
tqe_next :struct pf_rule*
tqe_prev :struct pf_rule**
rpool :struct pf_pool
list :struct pf_palist
tqh_first :struct pf_pooladdr*
tqh_last :struct pf_pooladdr**
cur :struct pf_pooladdr*
key :struct pf_poolhashkey
:
counter :struct pf_addr
:
tblidx :int
proxy_port[0] :u_int16_t
proxy_port[1] :u_int16_t
port_op :u_int8_t
opts :u_int8_t
evaluations :u_int64_t
packets[0] :u_int64_t
packets[1] :u_int64_t
bytes[0] :u_int64_t
bytes[1] :u_int64_t
:struct pf_rule(続き...)
kif :struct pfi_kif*
anchor :struct pf_anchor*
overload_tbl :struct pfr_ktable*
of_fingerprint :pf_ospf_t
rtableid :int
timeout[0] :u_int32_t
timeout[1] :u_int32_t
:
timeout[PFTM_MAX-1] :
states_cur :u_int32_t
states_tot :u_int32_t
max_states :u_int32_t
src_nodes :u_int32_t
max_src_nodes :u_int32_t
max_src_states :u_int32_t
max_src_conn :u_int32_t
max_src_conn_rate :struct (anonymous)
limit :u_int32_t
seconds :u_int32_t
qid :u_int32_t
pqid :u_int32_t
rt_listid :u_int32_t
nr :u_int32_t
prob :u_int32_t
cuid :uid_t
cpid :pid_t
return_icmp :u_int16_t
return_icmp6 :u_int16_t
max_mss :u_int16_t
tag :u_int16_t
match_tag :u_int16_t
uid :struct pf_rule_uid
uid[0] :uid_t
uid[1] :uid_t
op :u_int8_t
gid :struct pf_rule_gid
gid[0] :gid_t
gid[1] :gid_t
op :u_int8_t
:struct pf_rule(続き...)
rule_flag :u_int32_t
action :u_int8_t
direction :u_int8_t
log :u_int8_t
logif :u_int8_t
quick :u_int8_t
ifnot :u_int8_t
match_tag_not :u_int8_t
natpass :u_int8_t
keep_state :u_int8_t
af :sa_family_t
proto :u_int8_t
type :u_int8_t
code :u_int8_t
flags :u_int8_t
flagset :u_int8_t
min_ttl :u_int8_t
allow_opts :u_int8_t
rt :u_int8_t
return_ttl :u_int8_t
tos :u_int8_t
set_tos :u_int8_t
anchor_relative :u_int8_t
anchor_wildcard :u_int8_t
flush :u_int8_t
divert :struct (anonymous)
addr :struct pf_addr
:
port :u_int16_t


:struct pf_addr
pfa :union (anonymous)
v4 :struct in_addr
v6 :struct in6_addr
addr8[0] :u_int_8_t
addr16[0]:u_int_16_t
addr32[0]:u_int_32_t
addr8[1] :u_int_8_t
addr16[1]:u_int_16_t
addr32[1]:u_int_32_t
:
:
addr32[2]:u_int_32_t
addr8[15]:u_int_8_t
addr16[7]:u_int_16_t
addr32[3]:u_int_32_t
:struct pf_addr_wrap
v :union (anonymous)
a :struct (anonymous)
addr :struct pf_addr
mask :struct pf_addr
ifname :char[]
tblname :char[]
rtlabelname :char[]
rtlabel :u_int32_t
p :union(anonymous)
dyn :struct pfi_dynaddr*
tbl :struct pfr_ktable*
dyncnt :int
tblcnt: int
type :u_int8_t
iflags :u_int8_t


:struct pf_state
id :u_int64_t
creatorid :u_int32_t
direction :u_int8_t
pad[0] :u_int8_t
pad[1] :u_int8_t
pad[2] :u_int8_t
sync_list :struct (anonymous)
tqe_next :struct pf_state*
tqe_prev :struct pf_state**
entry_list :struct (anonymous)
tqe_next :struct pf_state*
tqe_prev :struct pf_state**
entry_id :struct (anonymous)
rbe_left :struct pf_state*
rbe_right :struct pf_state*
rbe_parent :struct pf_state*
rbe_color :int
src :struct pf_state_peer
:
dst :struct pf_state_peer
:
rule :union pf_rule_ptr
:
anchor :union pf_rule_ptr
:
nat_rule :union pf_rule_ptr
:
rt_addr :struct pf_addr
:
key[0] :struct pf_state_key*
key[1] :struct pf_state_key*
kif :struct pfi_kif*
rt_kif :struct pfi_kif*
src_node :struct pf_src_node*
nat_src_node :struct pf_src_node*
:struct pf_state (続き...)
packets[0] :u_int64_t
packets[1] :u_int64_t
bytes[0] :u_int64_t
bytes[1] :u_int64_t
creation :u_int32_t
expire :u_int32_t
pf_sync_time :u_int32_t
tag :u_int16_t
log :u_int8_t
state_flags :u_int8_t
timeout :u_int8_t
sync_state :u_int8_t
sync_updates :u_int8_t
_tail[0] :u_int8_t
_tail[1] :u_int8_t
_tail[2] :u_int8_t

QUIZ

- 次のルールを読み込ませるとどうなる?

Q.1)

/etc/pf.conf:

```
pass from { 192.168.1.1, 2001:db8::1:1 } to { 192.168.2.1, 2001:db8::2:1 }
```

Q.2)

/etc/pf.conf:

```
nat-anchor na
load anchor na from “/etc/na.conf”
block all
anchor fa
load anchor fa from “/etc/fa.conf”
```

/etc/na.conf:

```
nat from any to any -> 192.168.1.1
pass from any to 10.10.10.1
```

/etc/fa.conf:

```
nat from any to any -> 192.168.2.1
pass from any to 10.10.10.2
```