What’s wrong with PF

```
$ grep XXX pf.c pfvar.h pf_*.[ch] if_pf*.[c,h] \
    ../../sbin/pfctl/*.[chy] | wc -l
      68
$
```

Ryan McBride <mcbride@openbsd.org>

What is PF?

- The standard BSD Packet filter
- Started in 2001 after the removal of ipf from OpenBSD.
- Design goals:
  - Free software
  - Secure, robust packet filtering
  - Correct, readable code
  - Flexible but simple to use
  - Good performance
- Now about 37,000 lines of code

Some caveats for this talk:

This will not be an exhaustive list.

The scope is Architectural and general code quality issues, not bugs.

Talking about PF in OpenBSD

- There have been lots of improvements since I spoke about PF at AsiaBSDCon 2007.
- (Coincidentally, FreeBSD’s PF is from early 2007 - with a couple of bugfixes ported)

About Bugs

- Bugs tend to accumulate in code where actual usage is a subset of possible functionality:
  - anchors
  - ioctl interfaces
  - ipv6
- We need to keep this in mind when making design decisions.

- All the bugs are in code that was not written by Mike Frantzen.

How does development operate?

- Evolution rather than revolution
- Less invested in individual changes
- System always builds
- Rolling forward to new versions is easier
- Other subsystems remain integrated

ISO Model (ISO 9126^H^H2500:2005)

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

What about...

- Aesthetic Beauty
- Software License
- Having fun

Is someone who produces such ugly slides qualified to discuss aesthetics?

Maintainability & Aesthetic Beauty

Neverending code cleanup

- ongoing style(9) cleanup
- Still some minor things to be found with static analyzers (i.e. clang)

Some small things (noticed at n2k10 in Melbourne):

- inconsistent use of "pd" (struct pf_desc)
- some type inconsistencies (int,u_int8_t vs. sa_family_t)
- data structures cleanup (particularly struct pf_state, pf_rule)

Complicated Internals

- Tables code uses the kernel routing table patricia tree code.
- pfr_buffer code in general

  Requires passing arrays of identical objects
- pfctl’s handling of anchors is a nightmare

pf.conf Syntax

- Nominally LALR
- Initially based on ipf syntax
- Organic, mostly unplanned growth as PF gained functionality
- Now very challenging to maintain and extend
- parse.y has become increasingly confused whether or not it is a line oriented parser

```
anchor in on $fxp0 {
    block
    pass in proto tcp from any to $webserver port { 80, 443 }
    pass in proto { udp, tcp } from any to $dnsserver port 53
    pass in proto tcp from any to { $webserver $dnsserver } port 22
}
```

pf.conf Syntax

- Theo is trying to relax some of the rules of the syntax:
  - Ordering of keywords
  - braces "{ }" in lists of hosts: The macro expansion nightmare

```
windows_hosts = "{" $host1 $host2 "}"
broken_hosts = "{" $host3 $host4 "}"
block in quick from any to $windows_hosts $broken_hosts
```

- 2.5 hackathons spent failing to fix this
  - Hostnames are converted to IP addresses at the wrong point in the parser stack
  - IPv6 makes this about 6 times as hard

pf.conf Syntax

- Some improvements can also be obtained by removing features or replacing them with better designed ones ones.
- This can backfire: e.g. route-to and friends were slated for removal. Now we have:
  - route-to & friends
  - alternate routing tables
  - routing domains

Performance

"best-case" performance has improved A LOT in the past 3-4 years

- See henning's EuroBSDCon 2009 talk
- (upcoming data structure diagrams based on this)

"worst-case" performance is still an issue

- The cost of ruleset evaluation is very high
- Two cases:
    - CPU attack: packet traverses the ruleset, gets blocked
    - CPU+RAM attack: packet traverses the ruleset, creates state

- In theory we can fix the first with performance improvements in ruleset evaluation (easy to say, hard to do).
- The second one is much harder to deal with.

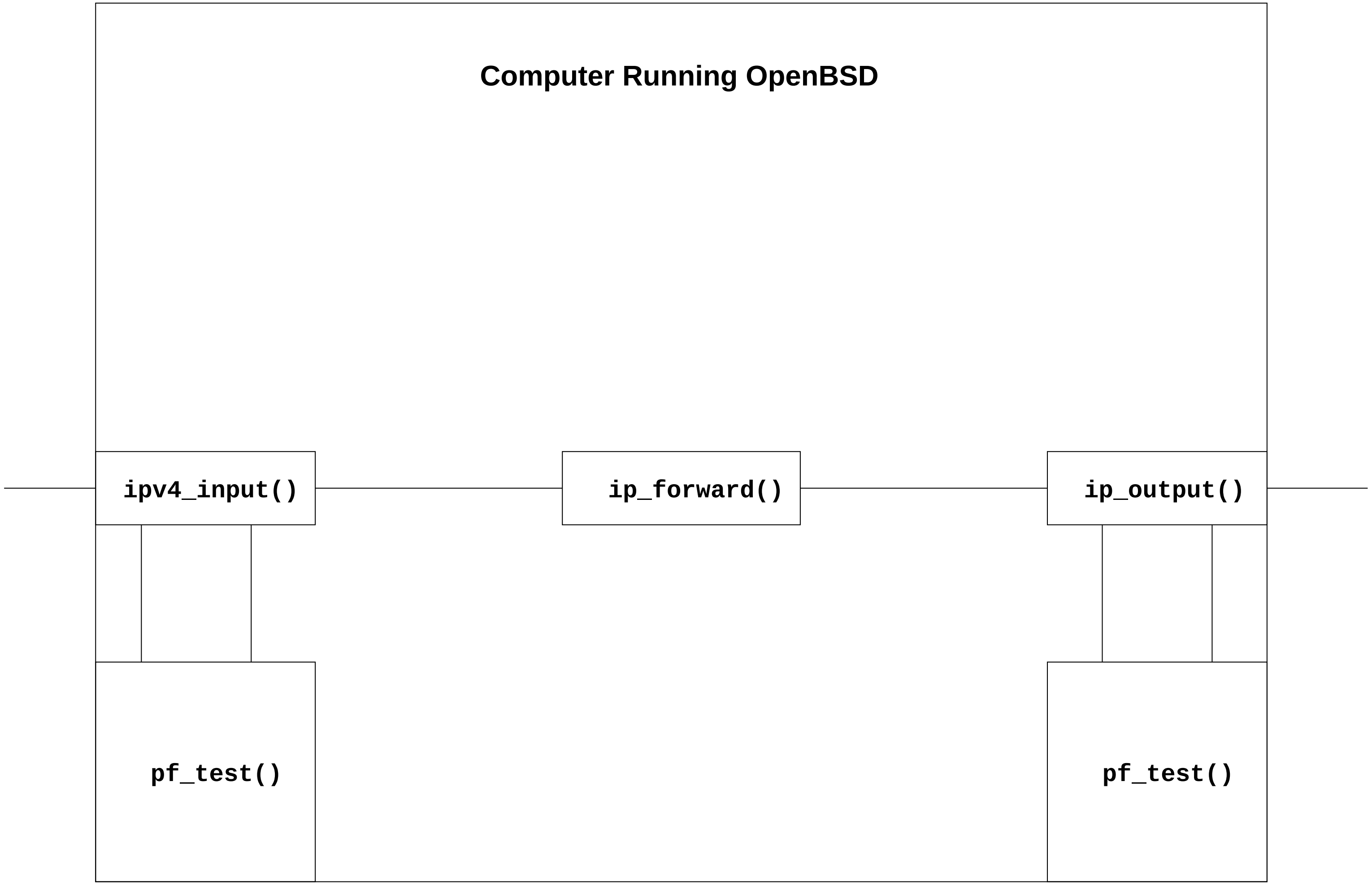
Portability

- Portability within OpenBSD is very good :-)
- Portability to other OSs... Pretty good, but getting harder
  - At least some version of PF runs on all major BSDs
  - Ported to Windows (CoreForce)

  - The project’s policy here is the same as for OpenSSH: we will not complicate the base code with portability goo.
  - Newer performance improvements rely on PF’s tentacles getting into other subsystems.

Tentacles

Where PF fits on the stack



Passing data to from input to output path

The struct pkthdr_pf appears directly in struct mbuf_hdr:

```
struct pkthdr_pf {
        void            *hdr;           /* saved hdr pos in mbuf for ECN */
        u_int            rtableid;      /* alternate routing table id */
        u_int32_t        qid;           /* queue id */
        u_int16_t        tag;           /* tag id */
        u_int8_t         flags;
        u_int8_t         routed;
};
```

- Small amount of data, huge performance improvement vs using mbuf tags.
- In the reald world, packets come on mbuf clusters, so this space in the header is usually unused anyways.

Case study: PF State Table Reorganization

- MAJOR change conducted over a period of years
- Implemented as many individual changes
- Other PF development & improvement efforts continued without being held back by this rearchitecture project.

About the PF state table

State entries contain

- Connection identifier (af, src ip, dst ip, src port, dst port)
- Connection Tracking
- Actions
- Links to other internal structures

Indexed in red-black trees

- Used to be more like a forest:
  - A RB tree for interface, interface group, and "floating" states.
  - "floating" is the default, but searching needs to happen from most specific to least specific.
  - So basically 3 tree searches per state lookup

Evolution by design

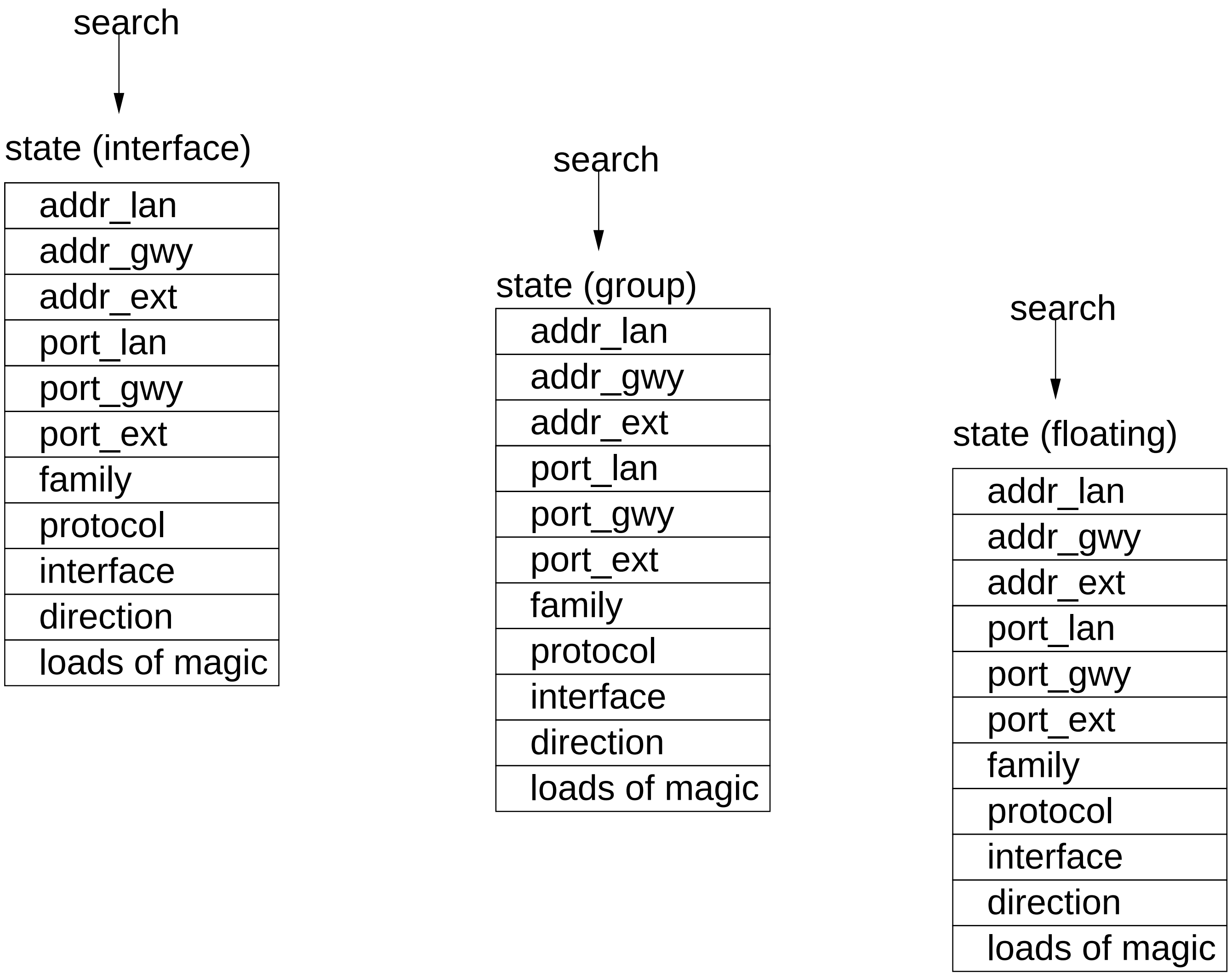
Initial goal: end-to-end connection tracking

- PF states, routing, ipsec, tcp/udp all do similar lookups
- 2 PF state lookups done on a forwarded packet
- We can combine these into a single lookup

A number of other improvements were obtained along the way

- Single ’pf_test_rules’ rather than protocol-specific almost copies
- Improved state creation code
- Fix handling ’if-bound’ states
- Deprecation of ’scrub’ rules
- Deprecation of separate translation ruleset
- ’match’ rules

struct pf_state in the dark ages



search

state (interface)

- addr_lan
- addr_gwy
- addr_ext
- port_lan
- port_gwy
- port_ext
- family
- protocol
- interface
- direction
- loads of magic

search

state (group)

- addr_lan
- addr_gwy
- addr_ext
- port_lan
- port_gwy
- port_ext
- family
- protocol
- interface
- direction
- loads of magic

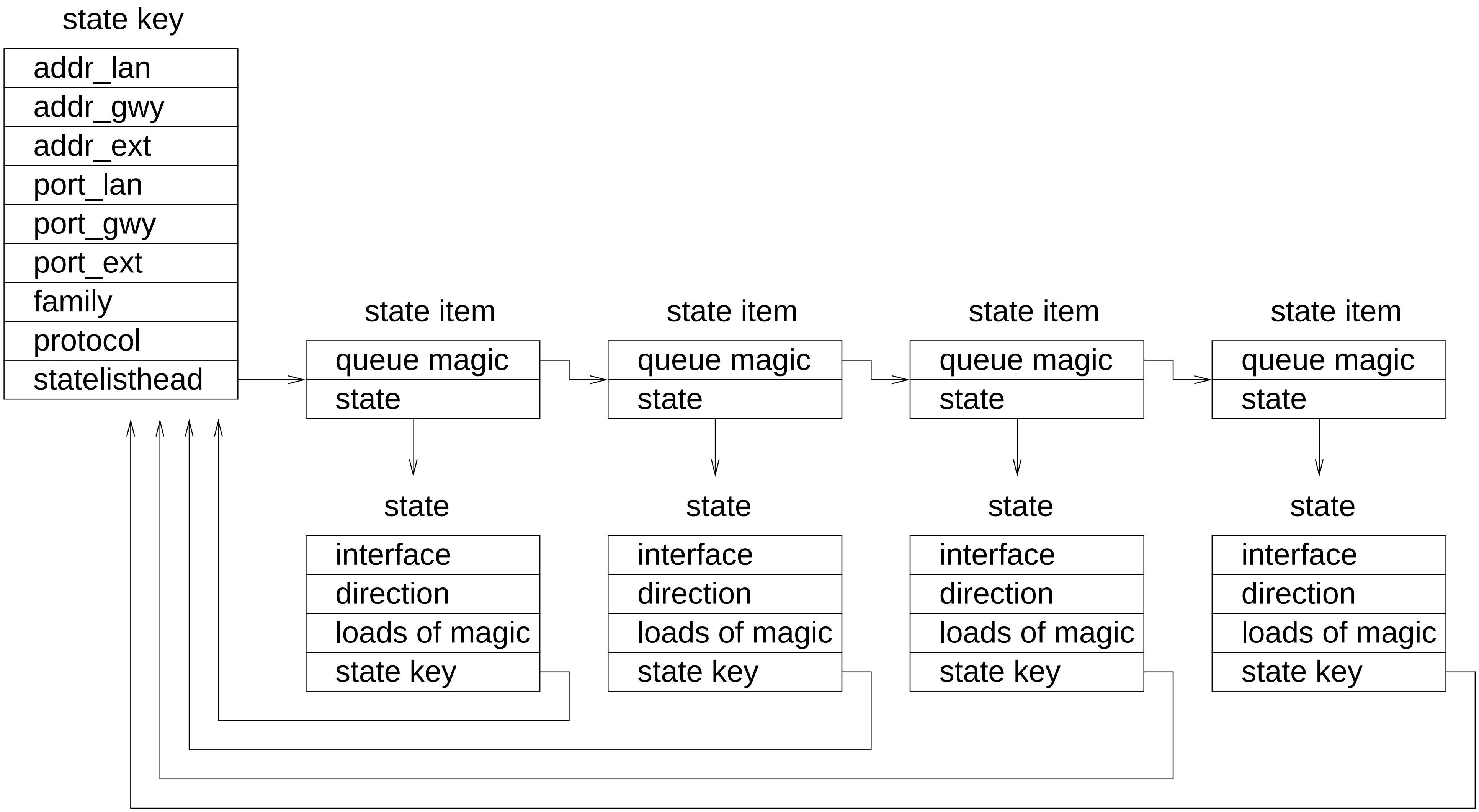
search

state (floating)

- addr_lan
- addr_gwy
- addr_ext
- port_lan
- port_gwy
- port_ext
- family
- protocol
- interface
- direction
- loads of magic

State table reorganization

pf_state / pf_state_key split, single state table


state key

addr_lan
addr_gwy
addr_ext
port_lan
port_gwy
port_ext
family
protocol
statelisthead

state item

queue magic
state

state

interface
direction
loads of magic
state key

state item

queue magic
state

state

interface
direction
loads of magic
state key

state item

queue magic
state

state

interface
direction
loads of magic
state key

state item

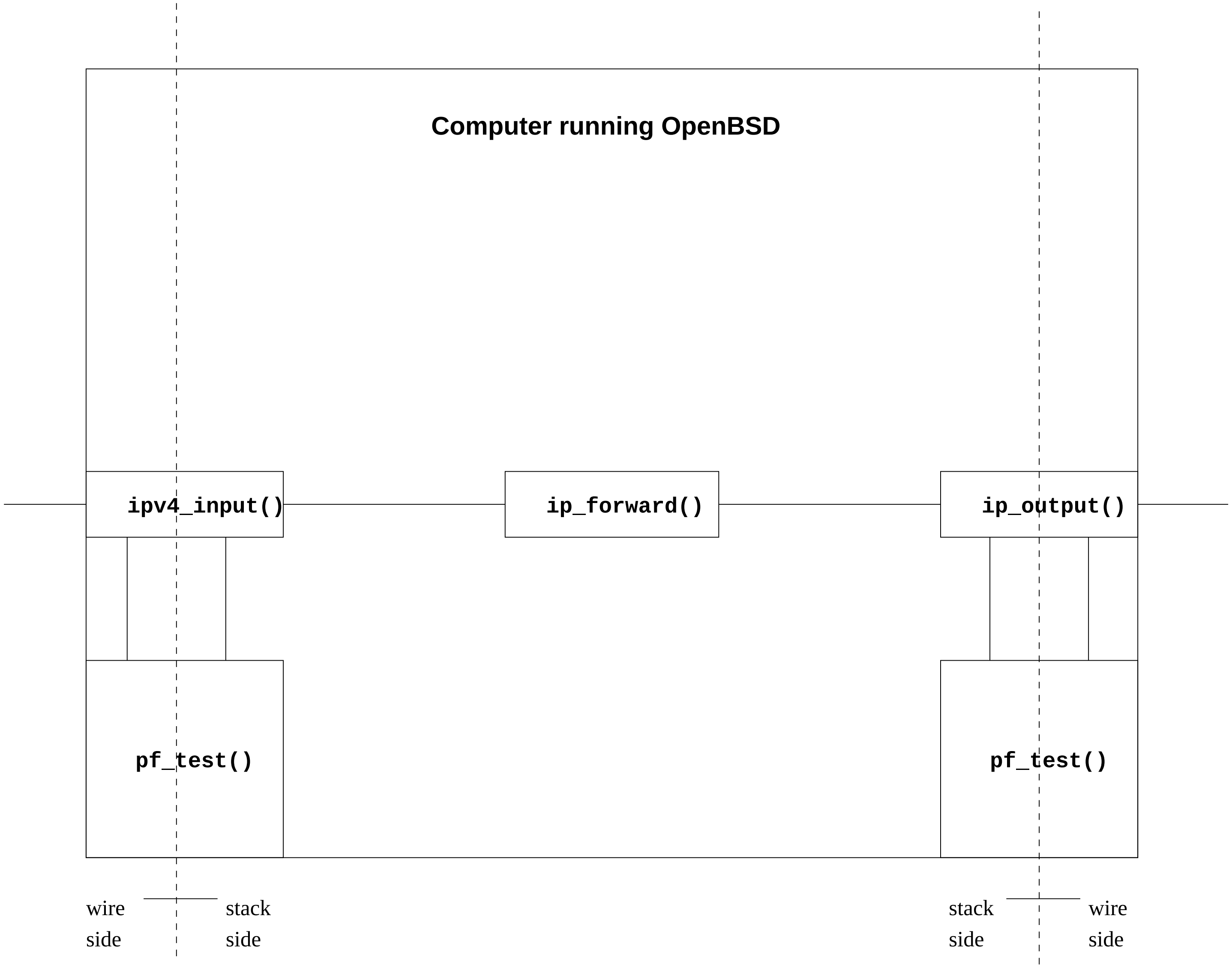
queue magic
state

state

interface
direction
loads of magic
state key

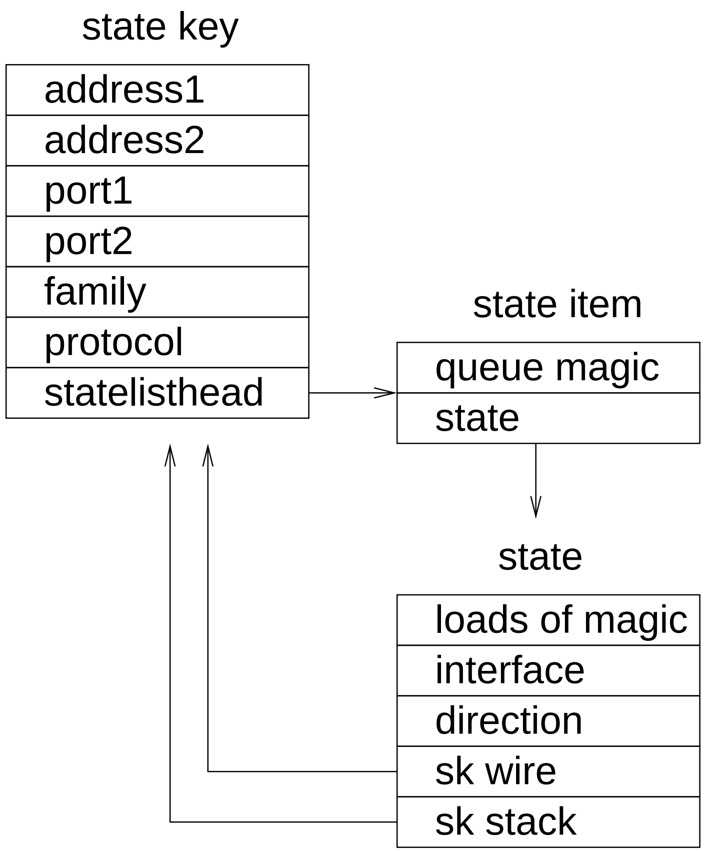
State table reorganization

Stack/Wire distinction



State table reorganization

Stack/Wire distinction: without NAT



state key

address1

address2

port1

port2

family

protocol

statelisthead

state item

queue magic

state

state

loads of magic

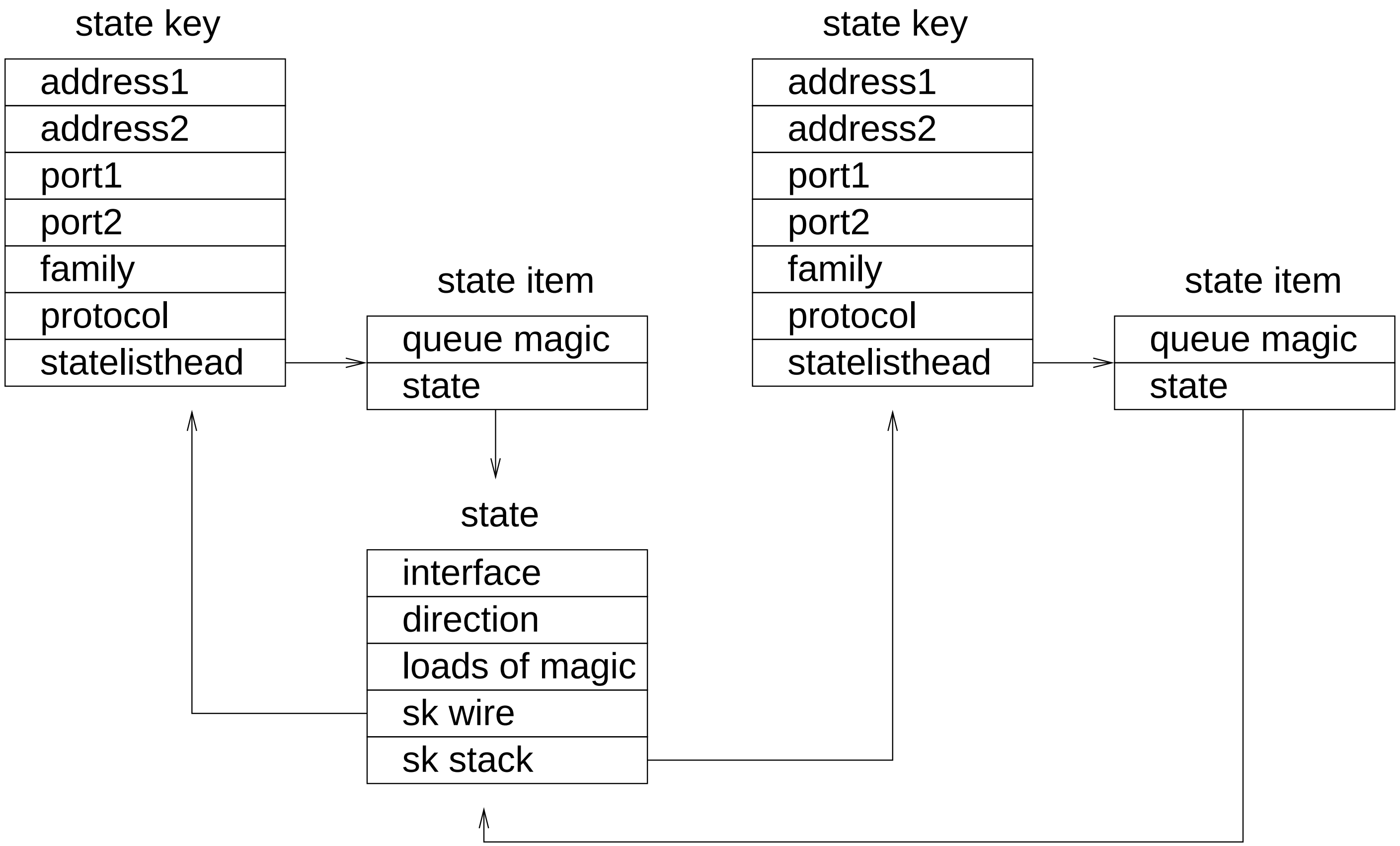
interface

direction

sk wire

sk stack

Stack/Wire distinction: with NAT



state key

- address1
- address2
- port1
- port2
- family
- protocol
- statelisthead

state item

- queue magic
- state

state

- interface
- direction
- loads of magic
- sk wire
- sk stack

state key

- address1
- address2
- port1
- port2
- family
- protocol
- statelisthead

state item

- queue magic
- state

- Determining whether NAT is taking place is just a pointer comparison now.
- There is nothing that says the address family has to be the same...

More tentacles!

Saving a pointer to the state

```
struct pkthdr_pf {
    void            *hdr;       /* saved hdr pos in mbuf for ECN */
    void            *statekey;  /* pf stackside statekey */
    u_int            rtableid;  /* alternate routing table id */
    u_int32_t        qid;       /* queue id */
    u_int16_t        tag;       /* tag id */
    u_int8_t         flags;
    u_int8_t         routed;
};
```

State linking

- Inbound: we store a pointer to the stackside state key in the pkthdr

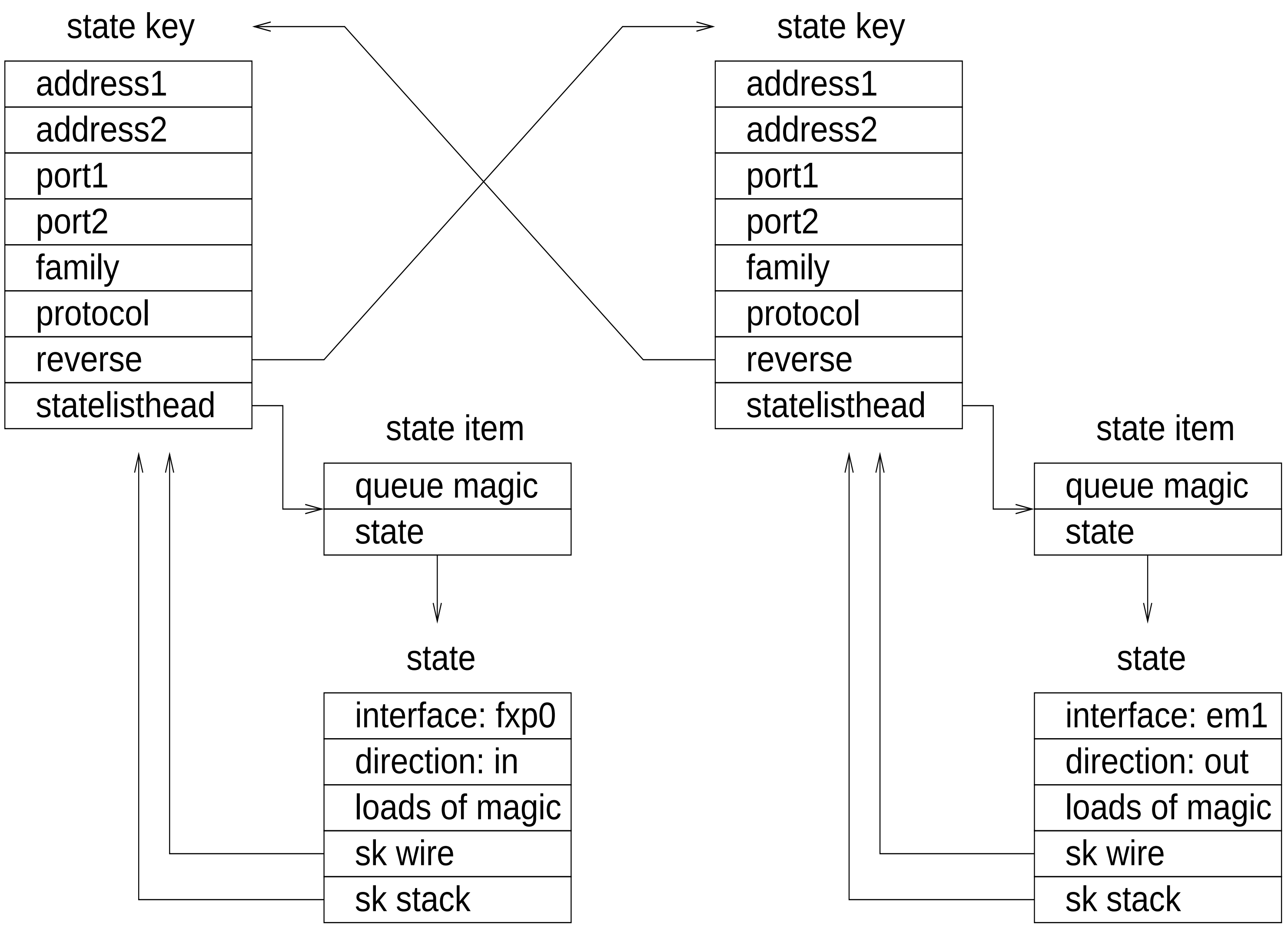
- Outbound: finding the state key is as simple as:

```
if (dir == PF_OUT && m->m_pkthdr.pf.statekey &&
  ((struct pf_state_key *)m->m_pkthdr.pf.statekey)->reverse)
    sk = ((struct pf_state_key *)m->m_pkthdr.pf.statekey)->reverse;
```

- no more redundant state table searches!

State table reorganization

State linking in the forwarding case



Sticking more in the state

- Now that relevant PF states are directly to the packet, we can use the state to cache other things:
  - TCP/UDP PCBs (for locally terminated connections)
  - route lookups
  - IPsec SAs
  - Other tunnel/connection contexts (npppd?)

State table reorganization

Timeline

- Initially concieved ~ May 2004 (after pf2k4)
- Interface abstraction cleanup 2005-05-21 1.489
- Alternate Routing Tables 2006-07-06 1.513
- Basic split of state struct 2007-05-29 1.534
- Fix interface-bound states 2007-06-21 1.546
- Core state table change 2008-05-29 1.576/1.577
- Link inbound/outbound states 2008-06-11 1.590
- Link states to PCBs 2008-07-03 1.604
- Remove scrub rules, add match 2009-04-06 1.640
- Remove NAT/RDR/BINAT rules 2009-09-01 1.658

Some observations

- Except for the alternate routing tables (which was needed for other reasons anyways), no major backouts were required.
- Some small but scary changes were temporarily disabled when problems were encountered.

Unfortunate side effects:

- New model is more challenging to understand and work with.

Fortunate side effects

- We get the ability to do nat/rdr on both inbound & outbound
  - Particularly helpful when you need to rewrite both addresses
  - Currently disabled in the parser (lots of documentation and possibly a little code needed to handle the routing challenges).
- New model makes it easier to implement things we didn’t plan for (like NAT64).

What’s Next

- A couple more performance optimizations
  - Linking of route-table entries to states
  - Work on Congestion handling has moved lower in the network stack (drop packets earlier)

- Hopefully, a period of stabilization and polishing
  - 3000 line diffs are not fun for anyone

- Documentation of PF internals
  - It is now impossible to plan core PF changes without some version of the state-linking diagram

What can YOU do to help?

Developers

- GOOD code, especially:
  - bug fixes
  - simplification / cleanup

Users

- Good bug reports
- Buy CD’s, TShirts
- Donate
- Encourage companies to donate
- Documentation

Any questions?