# OpenBSD's pf: Design, Imlementation, Future

Henning Brauer <henning@openbsd.org>

# The Dark Age

- no OpenBSD, no pf available
- doors wide open
- security poor

# 1993: The Computer is invented

- in Soviet-Unterzoegersdorf by Nikita Perostek Chrusov
- coal shortage lead to quick shutdown

# 1995: OpenBSD born

- Theo comes up with OpenBSD
- 2 years later, Georg Paul Thomann forks NetBSD
- OpenBSD gained reputation, especially as secure firewall OS

# ipf

- OpenBSD shipped with ipf
- written by Darren Reed from Australia
- some limitations, shortcomings, technical issues

# 2001: a license game

- Darren Reed "clarifies" his license
  - no shipping of modified versions
- one word missing from the license
  - nobody had noticed that so far
- he insisted on being in control
  - that's not what free source code is about
- on May 30, we delete that shit

# The birth of pf

- OpenBSD without packet filter - many people had lots to say
- Daniel Hartmeier sat down and coded
- had an early prototype quickly
  - comitted June 24
- many OpenBSD devs jumped in
  - and we had a working firewall package again

# OpenBSD 3.0: Dec 1, 2001

- first release with pf
- mostly an ipf clone
- most design choices inherited
- many things we'd design differently now

# design basics

- last match wins
- sane defaults, brevity, flexible language
- stateful
  - by default since OpenBSD 4.1
  - explicit "keep state" needed before

# design basics

- NAT integrated
  - no seperate nat.conf like ipf
- more integration where it makes sense
  - queueing, some ipsec, ...
- well integrated into the network stack
- entry points: ip_input, ip_output, bridge

# rulesets

- yacc parser, handwritten lexer
- ruleset: TAILQ of rules
  - two actually, active and inactive
    - for atomic reloads
    - load inactive, swap, clear now inactive

# anchors

- named subrulesets
  - can be nested, filesystem-style semantics
- heavily used by proxies

```
anchor bar from 10/8 {
  pass to 192.168.42.23
}
anchor "ftp-proxy/*"
```

- subrulesets inline or loaded later

```
pfctl -a bar -f /etc/pf.bar.conf
```

# anchors

- step in
  - save parser state
- parse anchor ruleset
- step out
  - restore parser state

# before ruleset evaluation

- set up a pf_pdesc
- describes the packet
  - pointers to headers, offsets to headers/fields
  - remembers addresses + ports before and after rewriting
  - various flags
- pf_setup_pdesc()
  - does some sanity checks on the packet before

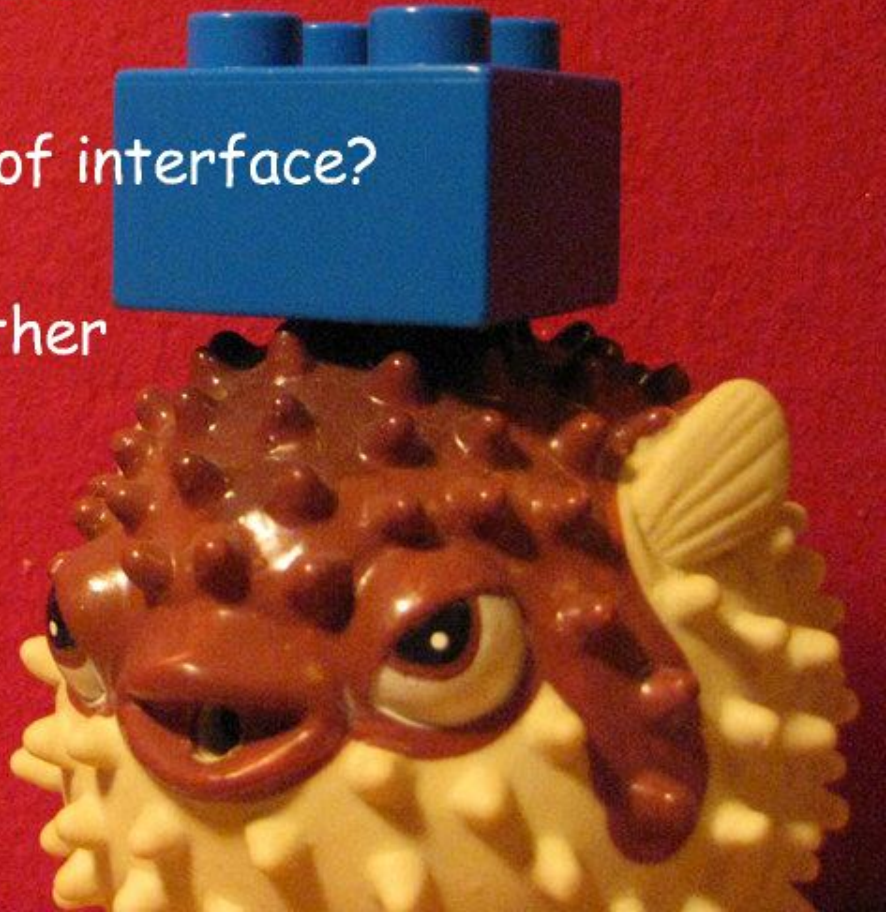
# rule matching

- walk the tailq
  - pf_test_rule()
- original: until last match, that rule decides everything
- now: "match" rules, intermediate matches, with actions
  - remember which rules matched
  - remember actions
    - pf_rule_to_actions()
  - remember rewriting address(es)
  - apply tags
  - log

# skip steps

```
pass in on em0 from 1.2.3.4
pass in on em0 proto tcp to 9.8.7.6 port 80
pass in on em0 proto tcp to 9.8.7.1 port 25
pass in on em0 to 5.6.7.8 rdr-to 9.8.7.6
pass in on em1
```

- rule 0 not matching because of interface?
- then 1, 2, 3 cannot match, either
- 4 is the next possible match



# skip steps

- calculated at load time

```
@0 pass in on em0 inet proto tcp from any to 9.8.7.6 port = 80 flags S/SA
  [ Skip steps: i=4 d=end r=end f=4 p=2 sa=2 sp=end ]
@1 pass in on em0 inet proto tcp from any to 9.8.7.1 port = 25 flags S/SA
  [ Skip steps: i=4 d=end r=end f=4 sp=end ]
@2 pass in on em0 inet from 1.2.3.4 to any flags S/SA
  [ Skip steps: i=4 d=end r=end f=4 p=end sp=end dp=end ]
@3 pass in on em0 inet from any to 5.6.7.8 flags S/SA rdr-to 9.8.7.6
  [ Skip steps: d=end r=end p=end sa=end sp=end dp=end ]
@4 pass in on em1 all flags S/SA
  [ Skip steps: i=end d=end r=end f=end p=end sa=end da=end sp=end dp=end ]
```

# ruleset optimizer

- combines and reorders rules
  - without changing meaning, of course
- glory details in pfctl_optimize.c

# optimizer in action

- input ruleset

```
pass in from 10/8
pass in from 10.0.0.1
pass in from 10.0.0.2
pass in to { 10.1.0.1, 10.1.0.2, 10.1.0.3 10.1.0.4 }
pass in to { 10.1.0.5, 10.1.0.6 }
```

- optimized

```
pass in inet from 10.0.0.0/8 to any flags S/SA
pass in inet from any to <__automatic_edccf0fc_0> \
    flags S/SA
```

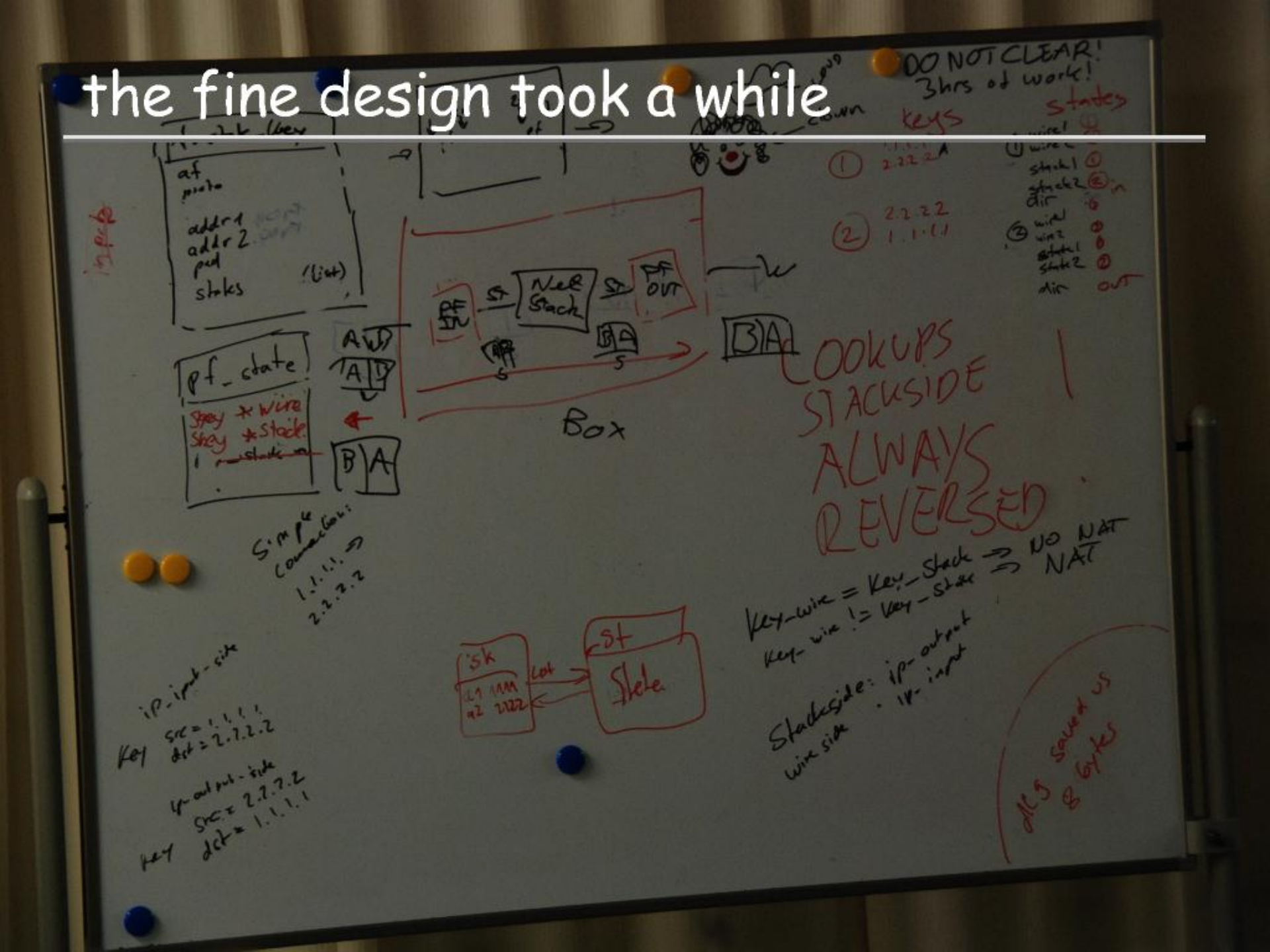
# tables

- lists of addresses
- relatively compact (memory), relatively fast lookup
- radix trees, thus longest match
  - think routing table
- the feature is great
  - the code, less so

# state table

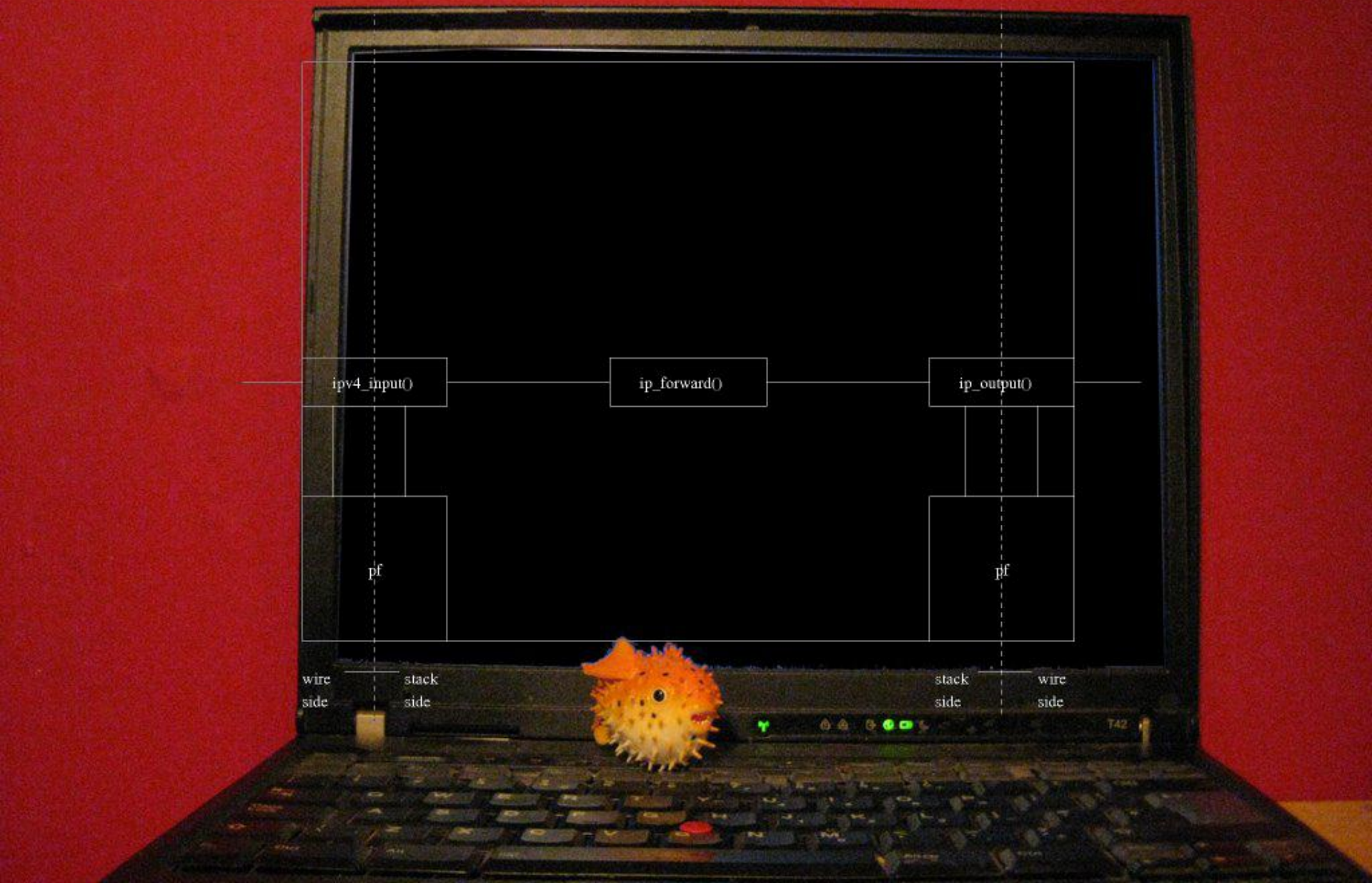
- re-designed and re-implemented in japan, 2008
- split state keys and states

# the fine design took a while

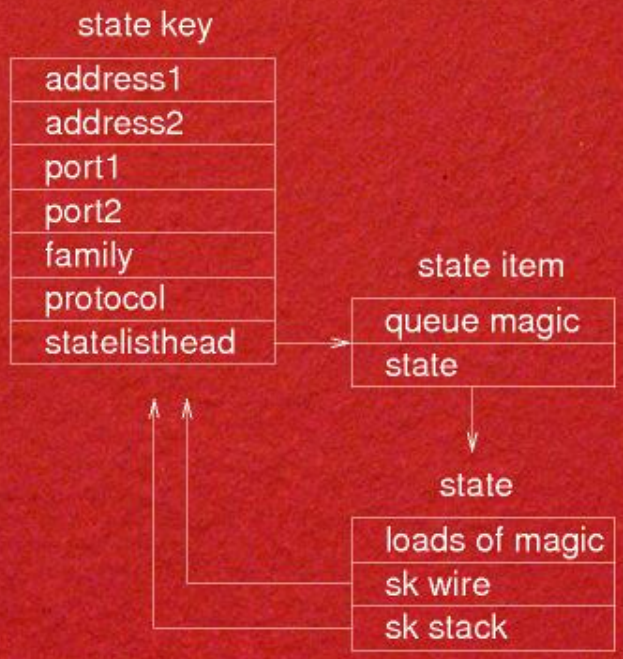


# Xfig pr0n: initial

# stack and wire side

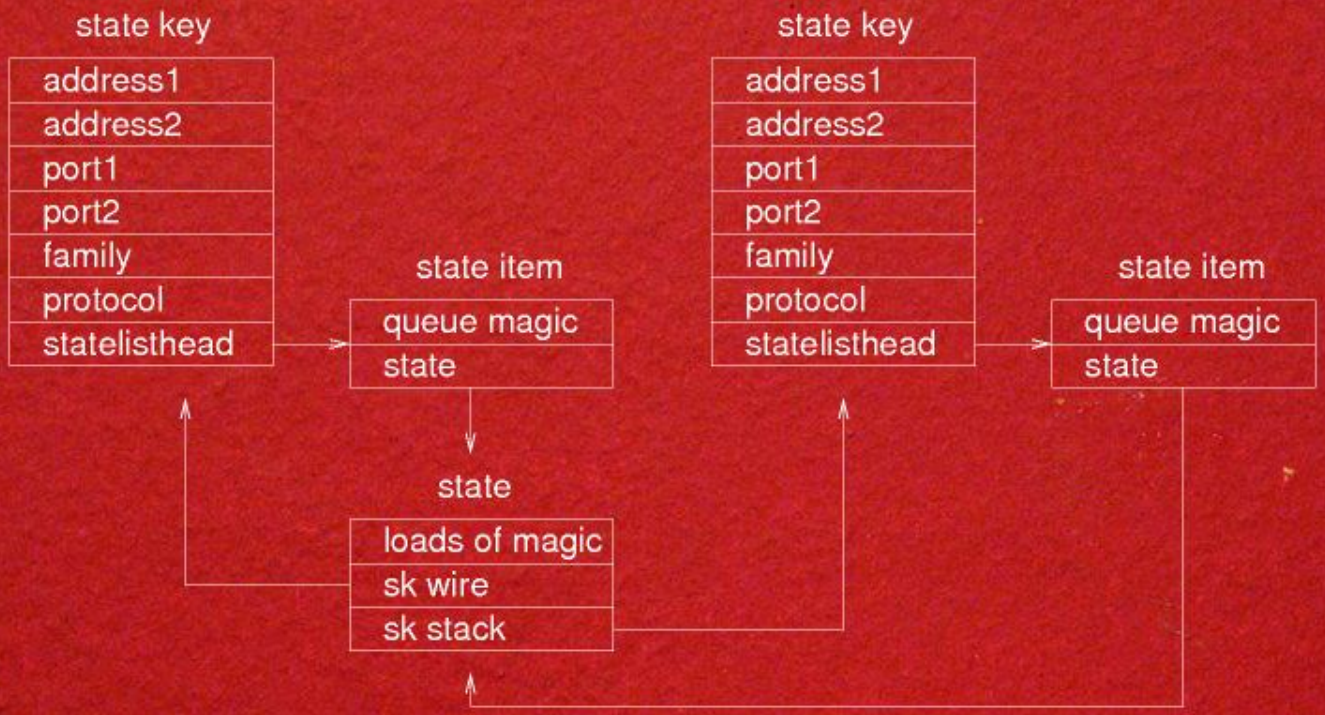


# Xfig prOn

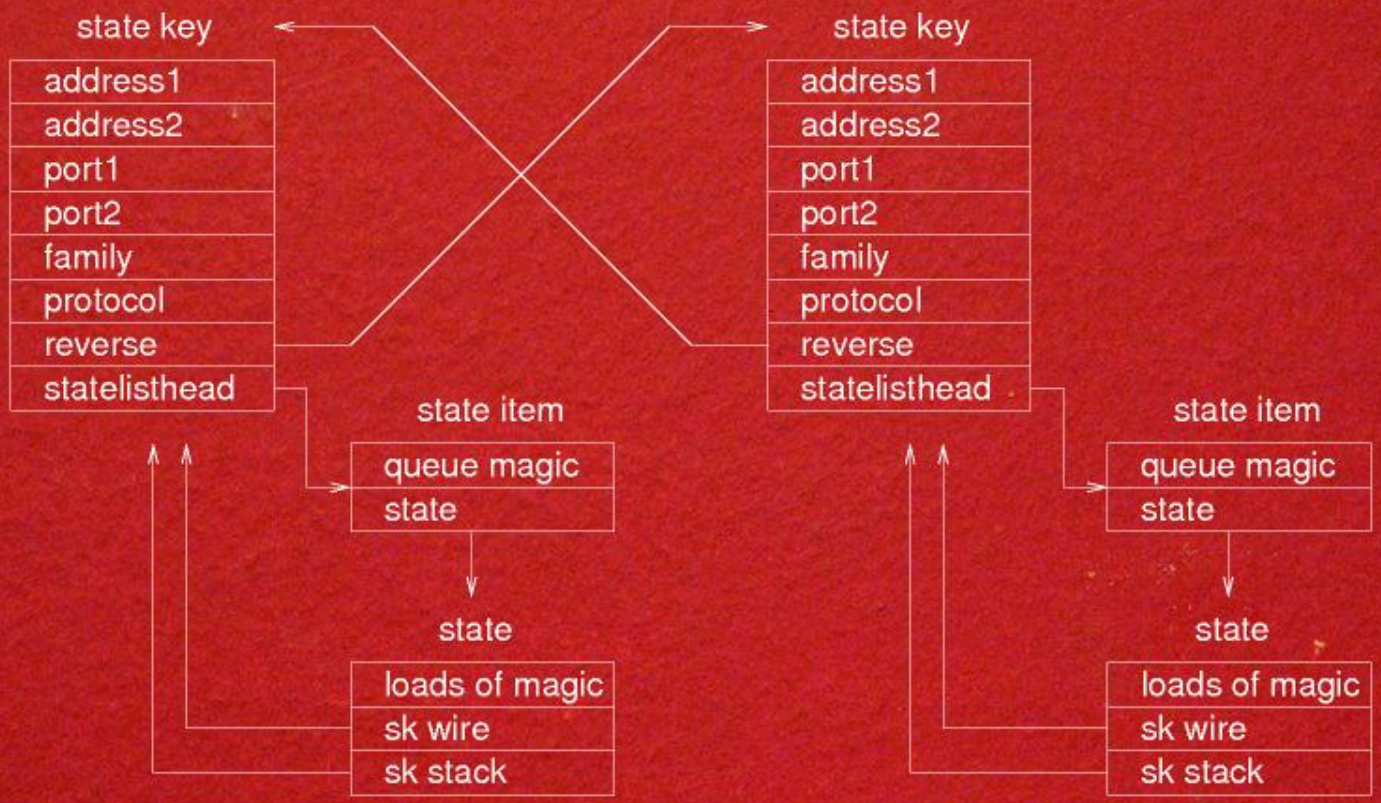

state key
address1
address2
port1
port2
family
protocol
statelisthead
state item
queue magic
state
state
loads of magic
sk wire
sk stack

# Xfig pr0n with NAT

- with NAT, state key pointers are different

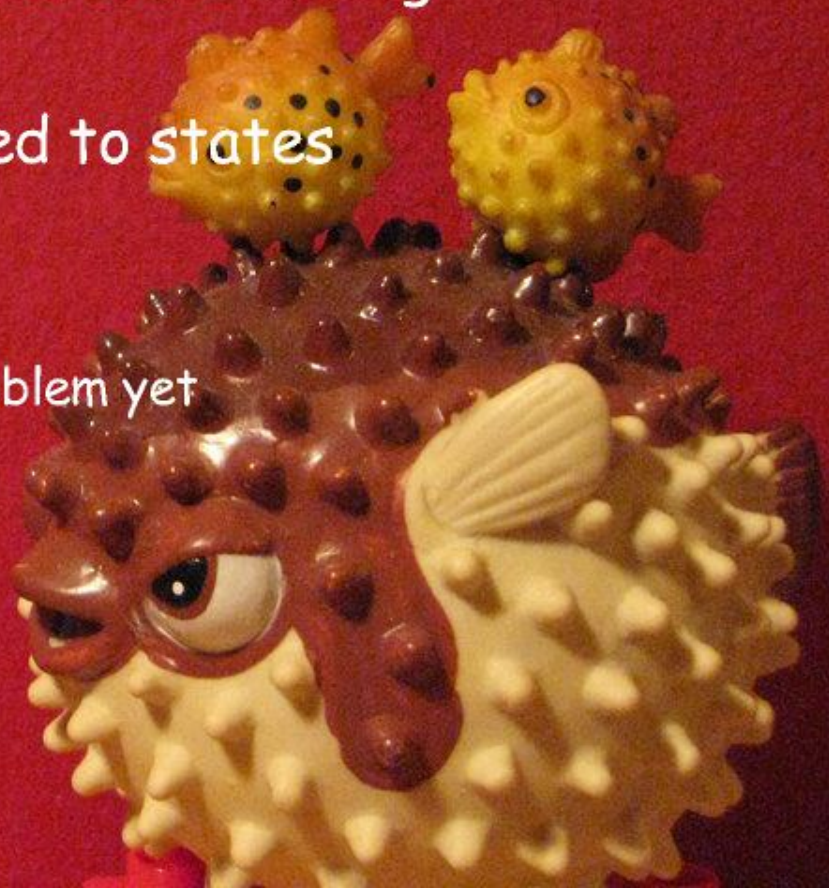


# states can be linked, saving lookups

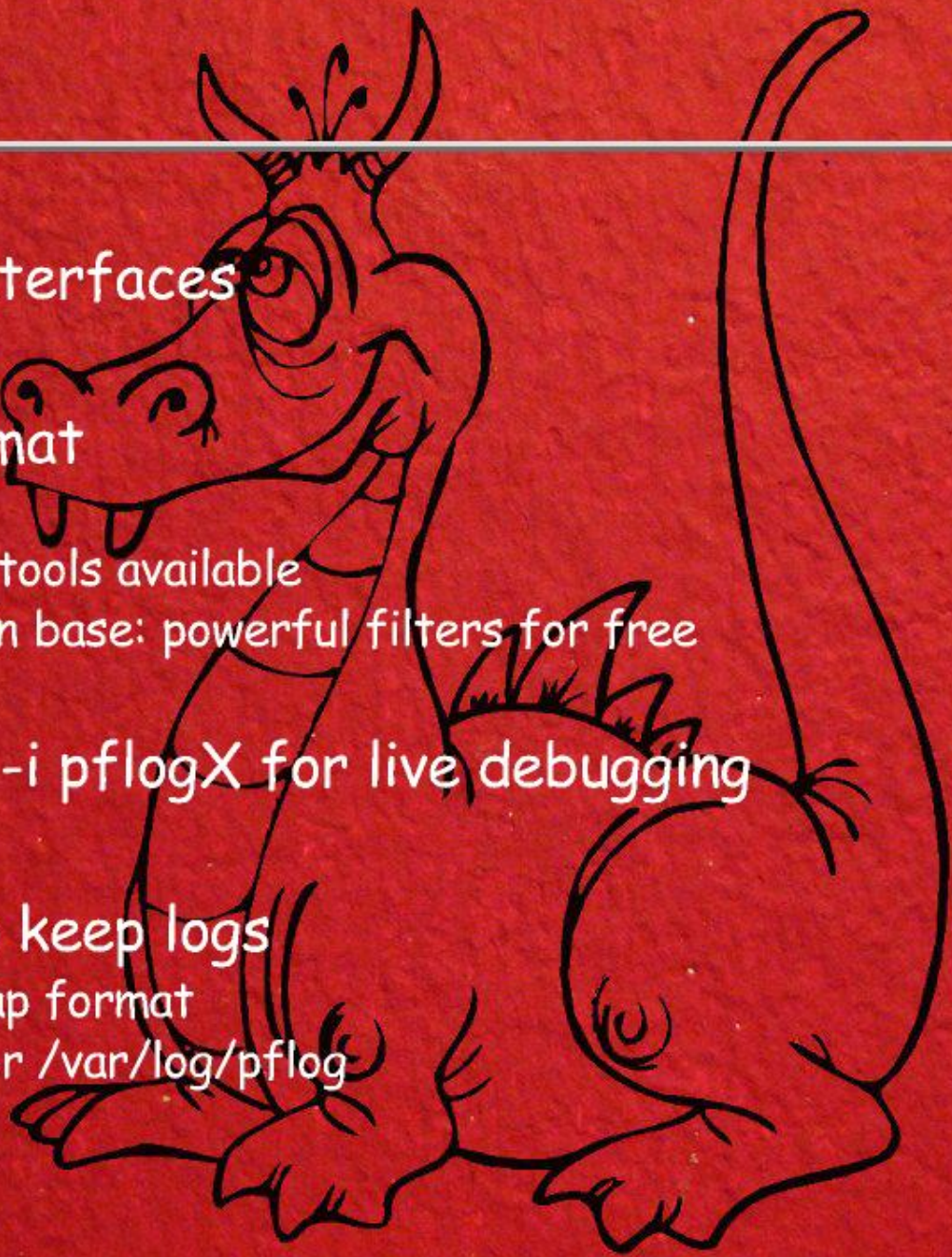


# linking

- matching state is recorded in the pkthdr mbuf
- outbound state can be found via state linking
- udp and tcp PCBs can get linked to states
- routes, potentially
  - nobody solved the more specific problem yet



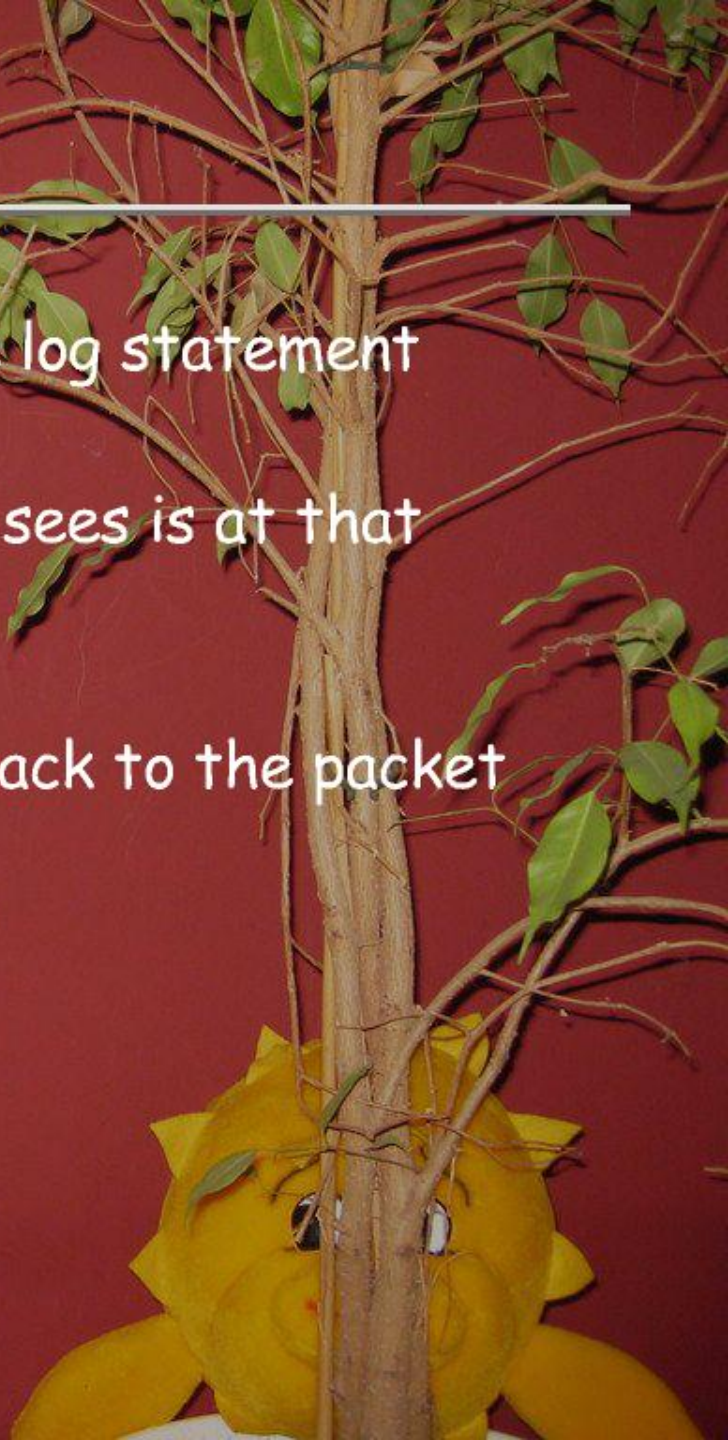
# pflog



- pflogX interfaces
- pcap format
  - compact
  - plenty of tools available
  - tcpdump in base: powerful filters for free
- tcpdump -i pflogX for live debugging
- pflogd to keep logs
  - still in pcap format
  - tcpdump -r /var/log/pflog

# pflog

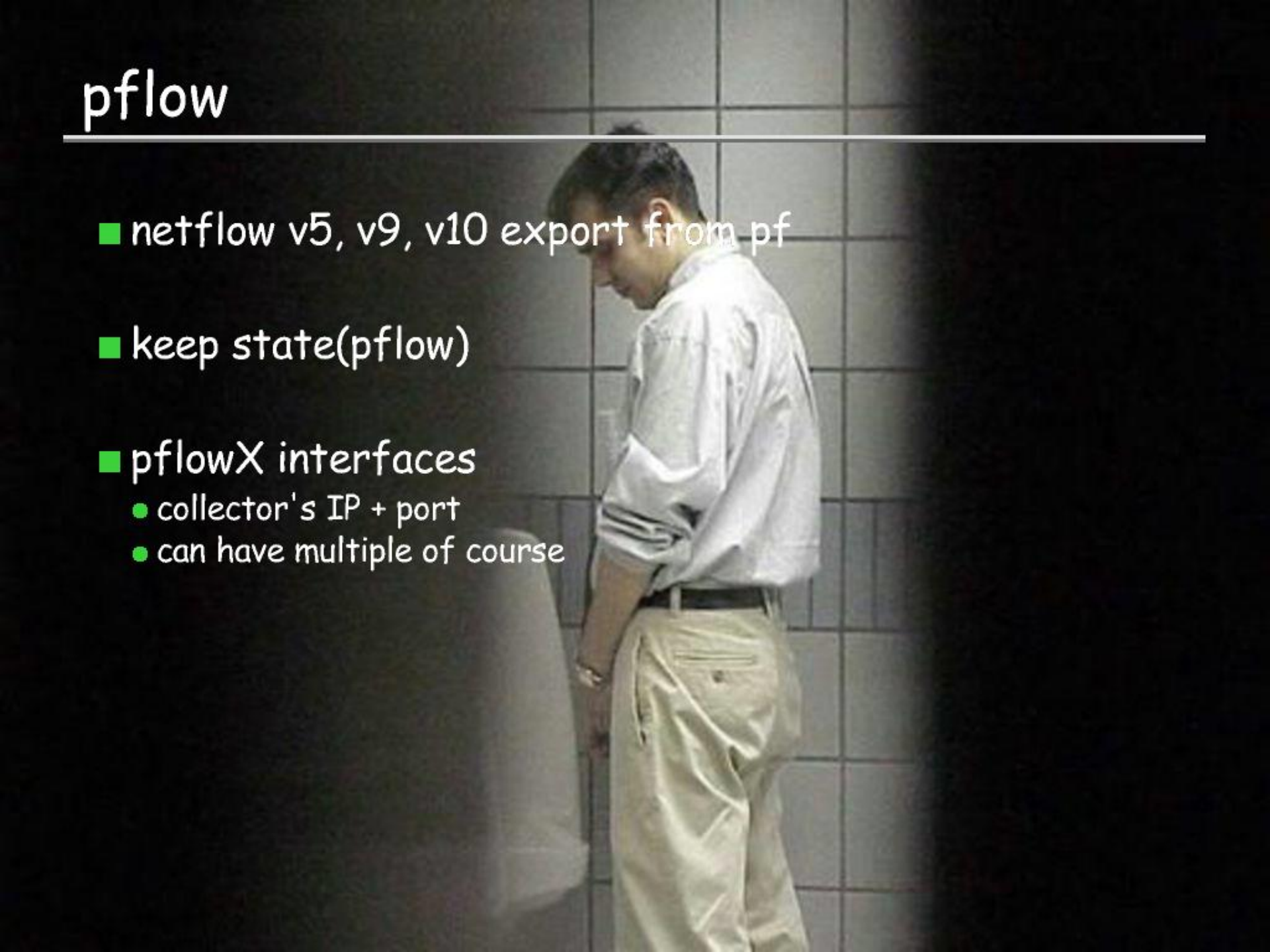
- want to log the moment we hit the log statement
- want to log the packet the way pf sees is at that moment
- but changes not actually written back to the packet yet
- do not want to copy twice



# pflog

- solution: custom bpf_mtap_pflog, pflog_bpfcopy
- pflog_bpfcopy copies the packet data into the bpf ringbuffer
- makes up an mbuf pointing to the bpf buffer
- calls pf_setup_pdesc + pf_translate on it
- stores original src and dst addresses/ports in the pflog header

# pflow



- netflow v5, v9, v10 export from pf
- keep state(pflow)
- pflowX interfaces
  - collector's IP + port
  - can have multiple of course

# 5 year plans

- the big redesign took 6 years
  - so it was a proper 5 year plan
  - ryan and I thought it would take us 2-3 years...
- next 5 year plan: queueing rewrite
  - took "only" 3
  - some cleanup after transition is complete

# next 5 year plan

- I think we're feature complete
  - maybe small bits here and there
- table code needs a rewrite
- SMP!
  - held up by the rest of the kernel not being there yet
  - rest of the network stack is neither
- lots of work to do "near" pf
  - foremost: routing table

# Questions?

# c'mon!