Pflua

Filtering packets with LuaJIT

FOSDEM 2015

Andy Wingo

wingo@igalia.com

https://github.com/Igalia/pflua

Agenda

Story time

High-performance packet filtering in software

Pflua

Forward-looking statements

Once upon a time

People had to buy operating systems pre-made

Problems you can solve == problems thought of by OS vendor

No fun :(

Once upon a time

People had to buy networking appliances

Problems you can solve == problems thought of by appliance vendor

No fun :(

Commodity hardware, commodity software

Rise of cheap x86 systems hand in hand with rise of free software

More users, more problems, more tinkerers, more solutions

Commodity networking

Q: Why are high-end networking appliances still sold as special-purpose rackable boxes?

Commodity networking

Multiple 10Gbps NICs, 1-2 Xeon sockets, 12-18 cores per socket, 2GHz cores

10 gigabits/s at 64 bytes/packet == 20 million packets/s (MPPS)

1 second / 20e7 PPS == 50 nanoseconds per packet

100 cycles

200 instructions, optimistically (gulp)

Commodity networking

Q: Why are high-end networking appliances still sold as special-purpose rackable boxes?

A: Although commodity hardware is ready for it, commodity software is not.

Linux TCP stack: 1 MPPS or so :(

Snabb: A new architecture

User-space networking stack

- Boot and drive NIC from user-space
- Affinity: one dedicated core per NIC

Nimble

- 10,000 SLOC
- Takes < 1 minute to build

Embracing constraints

Secret weapons

Lua: Tiny but expressive language

LuaJIT: Tiny but advanced Lua implementation

- Just-in-time compilation
- World-class performance
- Extensions: FFI, bit operations

Apps on a budget

200 instructions per packet:

- ~100 instructions of overhead
- ~100 instructions for the “app”

So what about packet filtering?

Packet filtering: a background

All about language

Filtering appliance implements a language

User writes in language

- `iptables`
- `tcpdump`
- Haka

tcpdump and libpcap

Well-loved (?) standard: `tcpdump`, which uses `libpcap`

User-facing “pflang”:

- `tcp port 80`
- `ip6 and udp src port 20`
- `tcp[9:4] = 0xdeadbeef`

Compiles to Berkeley Packet Filter (BPF) bytecode

BPF bytecode

Interpreter in `libpcap`

Interpreter in Linux, BSD kernels

JIT in Linux kernel (two versions)

JIT in BSD kernels

libpcap

“Venerable”

Good: Well-deployed, well-tested, users like the language

Bad: Pile of 90s C code; slow in user-space

Luke: Anyone want to implement a JIT for BPF using LuaJIT's DynASM?

Luke: Anyone want to implement a JIT for BPF using LuaJIT's DynASM?

Me: That's silly, you should just compile BPF to Lua and let LuaJIT handle it

Luke: Anyone want to implement a JIT for BPF using LuaJIT's DynASM?

Me: That's silly, you should just compile BPF to Lua and let LuaJIT handle it

Me: Hey let's do this

```
function tcp_port_80(P, length)
   local A, X, T = 0, 0, 0

   -- 000: A = P[12:2]
   if 14 > length then return 0 end
   A = bit.bor(bit.lshift(P[12], 8), P[12+1])

   -- 001: if (A == 34525) goto 2 else goto 8
   if not (A==34525) then goto L7 end

   -- 002: A = P[20:1]
   if 21 > length then return 0 end
   A = P[20]

   -- 003: if (A == 6) goto 4 else goto 19
   if not (A==6) then goto L18 end

   -- 004: A = P[54:2]
   if 56 > length then return 0 end
   A = bit.bor(bit.lshift(P[54], 8), P[54+1])

   -- 005: if (A == 80) goto 18 else goto 6
   if (A==80) then goto L17 end
```

`tcp port 80`, continued

```
   -- 006: A = P[56:2]
   if 58 > length then return 0 end
   A = bit.bor(bit.lshift(P[56], 8), P[56+1])

   -- 007: if (A == 80) goto 18 else goto 19
   if (A==80) then goto L17 end
   goto L18

   -- 008: if (A == 2048) goto 9 else goto 19
   ::L7::
   if not (A==2048) then goto L18 end

   ...
end
```

`tcp port 80`, continued

```
-- 009: A = P[23:1]
-- 010: if (A == 6) goto 11 else goto 19
-- 011: A = P[20:2]
-- 012: if (A & 8191 != 0) goto 19 else goto 13
-- 013: X = (P[14:1] & 0xF) << 2
-- 014: A = P[X+14:2]
-- 015: if (A == 80) goto 18 else goto 16
-- 016: A = P[X+16:2]
-- 017: if (A == 80) goto 18 else goto 19
-- 018: return 65535
-- 019: return 0
```

Result?

Straightforward, easy to get right

- bitops, `goto` make it easy

Good perf! (More later)

LuaJIT does heavy lifting

Irritations

Pflang numbers are 32-bit unsigned integers

Lua numbers are 64-bit floating-point numbers (doubles)

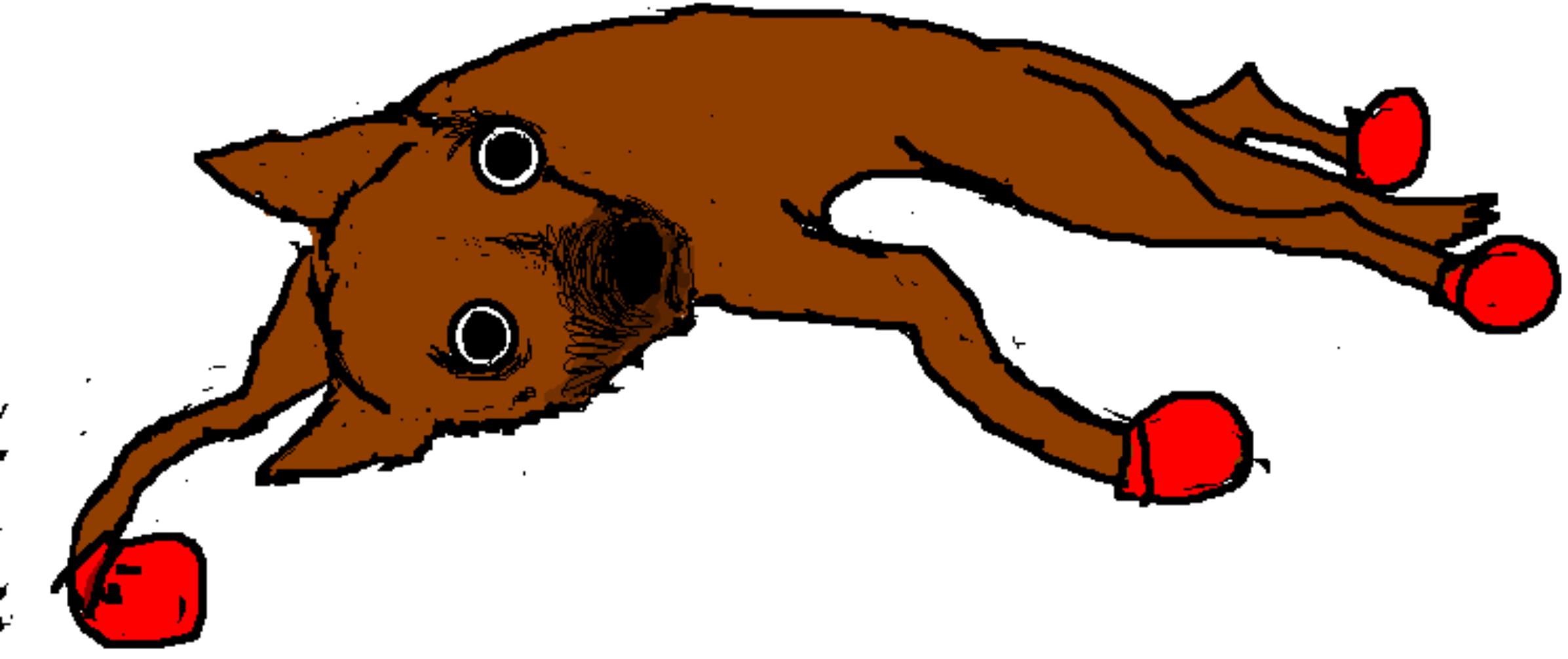
Irritations

Pflang numbers are 32-bit unsigned integers

Lua numbers are 64-bit floating-point numbers (doubles)

Bitops module returns *signed* 32-bit integers :-((((

why would you do this to me?



Irritations

Pflang numbers are 32-bit unsigned integers

Lua numbers are 64-bit floating-point numbers (doubles)

Bitops module returns *signed* 32-bit integers :-((((

No visibility for optimizations

Still have 90s-flashback `libpcap` around

We can do better!

Native pflang pipeline

Solution: Implement pflang compiler from scratch, avoiding `libpcap`

Parse → Lower → Optimize → Generate

```
function tcp_port_80(P,length)
   if length < 34 then return false end
   local var1 = cast("uint16_t*", P+12)[0]
   if var1 == 8 then
      if P[23] ~= 6 then return false end
      if band(cast("uint16_t*", P+20)[0],65311) ~= 0 then
         return false
      end
      local var7 = lshift(band(P[14],15),2)
      local var8 = (var7 + 16)
      if var8 > length then return false end
      if cast("uint16_t*", P+(var7 + 14))[0] == 20480 then
         return true
      end
      if (var7 + 18) > length then return false end
      return cast("uint16_t*", P+var8)[0] == 20480
   else
      if length < 56 then return false end
      if var1 ~= 56710 then return false end
      local var24 = P[20]
      if var24 == 6 then goto L22 end
      do
         if var24 ~= 44 then return false end
         if P[54] == 6 then goto L22 end
```

Optimization opportunities

Algebraic simplifications

Range inference

Length-check hoisting

Constant folding

Common subexpression elimination

Optimizations necessary, given duplication exposed by the lowering pflang to a minimal intermediate language

```
-- No packet smaller than 34 bytes will pass this filter.
if length < 34 then return false end

-- Access ethernet protocol number in native endianness.
local var1 = cast("uint16_t*", P+12)[0]

-- Compare ethernet protocol number to ntohs(2048).
if var1 == 8 then
   -- So it's IPv4.  If it's not TCP, fail.
   if P[23] ~= 6 then return false end

   -- Access flags, no need to byte-swap.
   if band(cast("uint16_t*", P+20)[0],65311) ~= 0 then
      return false
   end

   -- Compute offset of first byte of IP payload.
   local var7 = lshift(band(P[14],15),2)

   -- If the port number in network order is 80, pass.
   local var8 = (var7 + 16)
   if var8 > length then return false end
   if cast("uint16_t*", P+(var7 + 14))[0] == 20480 then
      return true
   end
end
```

LuaJIT still kicks in

Tracing JIT: Shape of machine code is shape of network traffic

Register allocation

Work around dynamic nature of Lua

- Allocation sinking
- Integer specialization
- Hoisting of checked loads (is `math.floor` actually `floor`?)

Project status

Pipelines

Perf

Compatibility

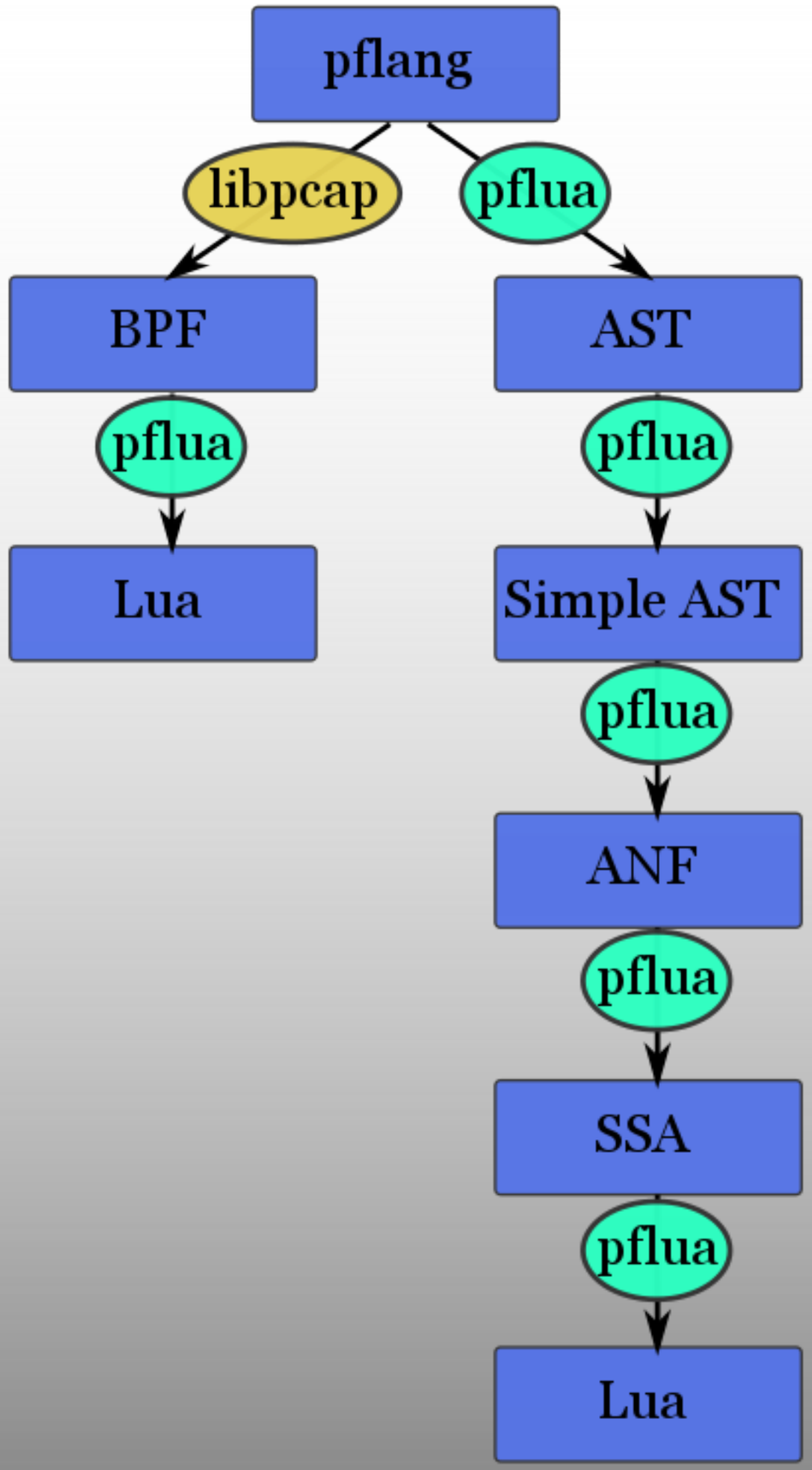
Adoption

Future?

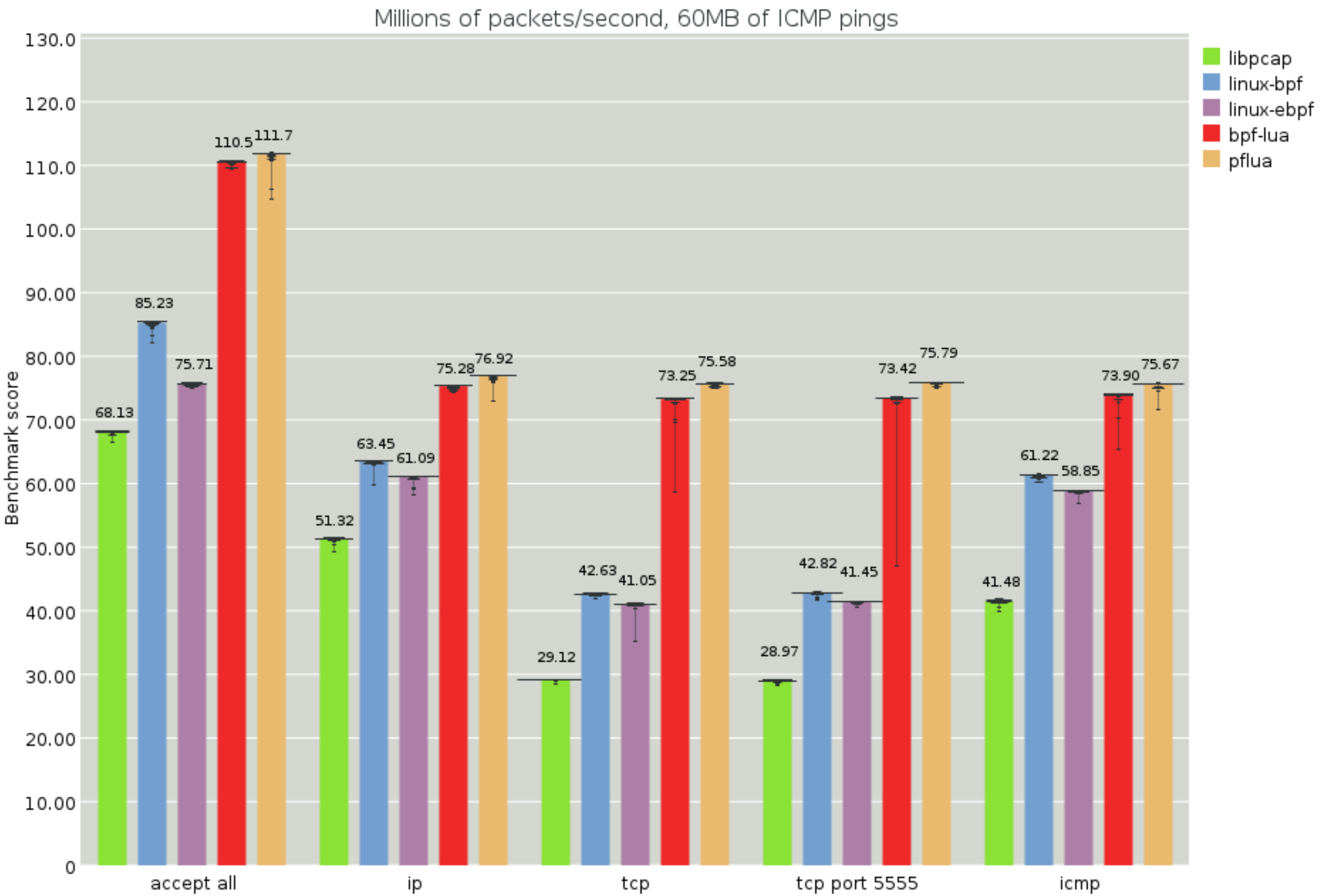
Two pipelines

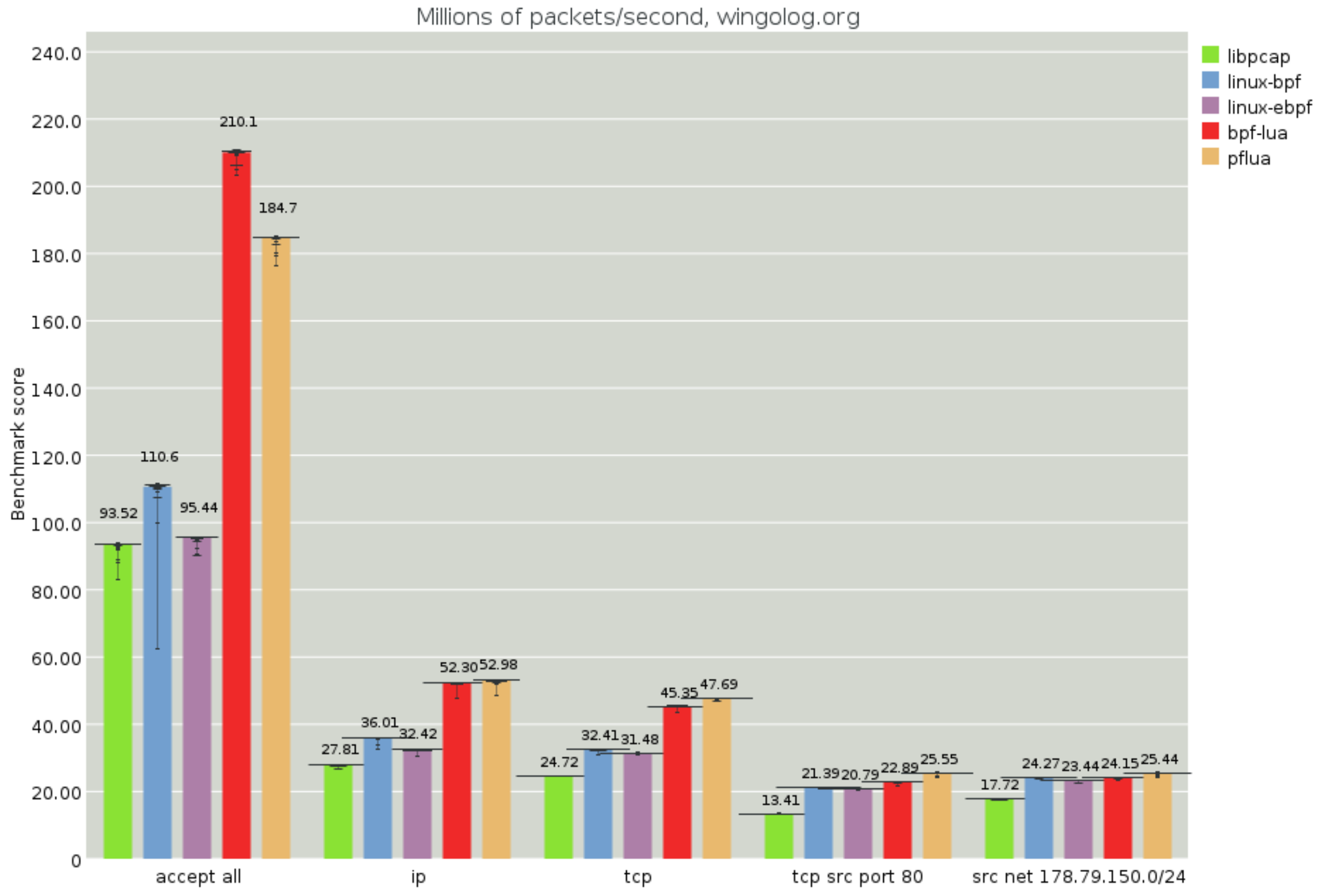
User chooses

Default: “native”

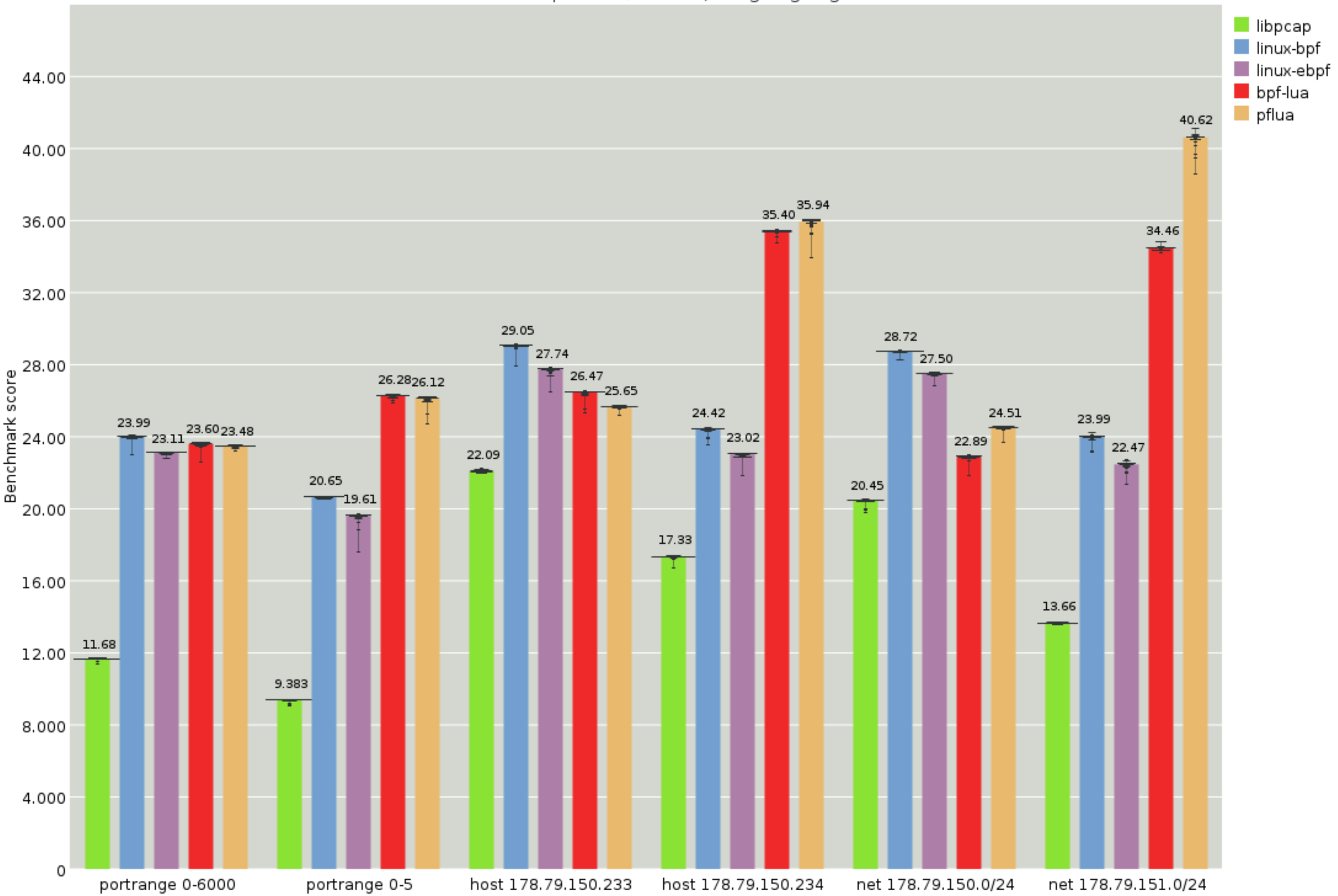


Performance


Millions of packets/second, 60MB of ICMP pings
Benchmark score
130.0
120.0
110.0
100.0
90.00
80.00
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0
libpcap
linux-bpf
linux-ebpf
bpf-lua
pflua
68.13
85.23
75.71
110.5
111.7
51.32
63.45
61.09
75.28
76.92
29.12
42.63
41.05
73.25
75.58
28.97
42.82
41.45
73.42
75.79
41.48
61.22
58.85
73.90
75.67
accept all
ip
tcp
tcp port 5555
icmp


Millions of packets/second, wingolog.org
Benchmark score
240.0
220.0
200.0
180.0
160.0
140.0
120.0
100.0
80.00
60.00
40.00
20.00
0
libpcap
linux-bpf
linux-ebpf
bpf-lua
pflua
93.52
110.6
95.44
210.1
184.7
27.81
36.01
32.42
52.30
52.98
24.72
32.41
31.48
45.35
47.69
13.41
21.39
20.79
22.89
25.55
17.72
24.27
23.44
24.15
25.44
accept all
ip
tcp
tcp src port 80
src net 178.79.150.0/24

Millions of packets/second, wingolog.org



Performance challenges

Consistency – all very dependent on caches

Int/float conversion and dynamic checks taking time, blowing icache?

Trace topology random by nature, but perf impacts are not constant

Solution: either improve LuaJIT or write our own code generator

Compatibility with libpcap

Completeness

- All ethernet-encapsulated operators implemented, except vlan and protochain
- Hostname resolution not implemented
- Keyword elision not implemented

Compatibility with libpcap

Correctness

- Parser bugs?
- Optimizer bugs?
- Semantics bugs?
- Solution: Extensive randomized checking. Catch Katerina Barone-Adesi on Sunday at 14h35 in the testing devroom!

Adoption

Snabb branch to be merged soon (depends on other snabb things)

Your tool?

Beyond pflang

Pflang could be better

- HTTP and other protocol support
- Call-outs to user-defined functions?
- Pattern matching

```
match {
  tcp src port $a => $a % 2 = 0;
  udp => true;
}
```

To the moon!

Check it out!

`https://github.com/Igalia/pflua`,

`https://github.com/SnabbCo/snabbswitch`

`wingo@igalia.com`

Partner with us to build high-performance networking apps with LuaJIT! ☐

Questions?