Production
Snabb

Simple, fast software networking with Snabb

20 January 2017 – linux.conf.au

Andy Wingo wingo@igalia.com

@andywingo
hey hacker

User-space networking is for us!
Snabb is a great way to do it!
Make a thing with Snabb!
You are an ISP
The distant past: the year 2000
To set up: you lease DSL exchanges, bandwidth, core routers
Mission accomplished!
(hi)story

The distant past: the year 2005

You still pay for DSL, bandwidth, routers

Also you have some boxes doing VoIP (more cash)
(hi)story

The distant past: the year 2010
You still pay for DSL, bandwidth, routers, VoIP
OMG TV!!!
Also we are running out of IPv4!!!
Also the subscriber fee is still the same!!!!!!!
(hi)story

Trend: ISPs have to do more (VoIP, TV, VOD, cloud, carrier NAT)

“Doing more”: more expensive boxes in the rack ($70k/port?)

Same story with many other users

Isn’t there a better way?
material conditions

In the meantime, commodity hardware caught up

- Xeon dual-socket, >12 core/socket
- Many 10Gbps PCIe network cards (NICs)
- 100-200 Gbps/server
- 10-15 million packets per second (MPPS) per core+NIC pair
- 70 ns/packet

Let’s do it!
alternate
(hi)story

The teleology of open source: “one day this will all run Linux”

Conventional wisdom: if I walk the racks of a big ISP, it’s probably all Linux
The teleology of open source: “one day this will all run Linux”

Conventional wisdom: if I walk the racks of a big ISP, it’s probably all Linux

Q: The hardware is ready for 10 MPPS on a core. Is Linux?
The teleology of open source: “one day this will all run Linux”

Conventional wisdom: if I walk the racks of a big ISP, it’s probably all Linux

Q: The hardware is ready for 10 MPPS on a core. Is Linux?
A: Nope
<table>
<thead>
<tr>
<th>why not linux</th>
<th>Heavyweight networking stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>System/user barrier splits your single network function into two programs</td>
</tr>
<tr>
<td></td>
<td>Associated communication costs</td>
</tr>
</tbody>
</table>
user-space networking

Cut Linux-the-kernel out of the picture; bring up card from user space

- tell Linux to forget about this PCI device
- mmap device’s PCI registers into address space
- poke registers as needed
- set up a ring buffer for receive/transmit
- profit!
The distant past: the year 2017

Multiple open source user-space networking projects having success


Deutsche Telekom’s TeraStream: Vendors provide network functions as software, not physical machines

How do software network functions work?
aside

Snabb aims to be rewritable software

The hard part: searching program-space for elegant hacks

“Is that all? I could rewrite that in a weekend.”
nutshell

A snabb program consists of a graph of *apps*

Apps are connected by directional *links*

A snabb program processes packets in units of *breaths*
local Intel82599 =
    require("apps.intel.intel_app").Intel82599
local PcapFilter =
    require("apps.packet_filter.pcap_filter").PcapFilter

local c = config.new()
config.app(c, "nic", Intel82599, {pciaddr="82:00.0"})
config.app(c, "filter", PcapFilter, {filter="tcp port 80"})

config.link(c, "nic.tx -> filter.input")
config.link(c, "filter.output -> nic.rx")

engine.configure(c)

while true do engine.breathe() end
Each breath has two phases:

- **inhale** a batch of packets into the network
- **process** those packets

To inhale, run `pull` functions on apps that have them

To process, run `push` functions on apps that have them
function Intel82599:pull ()
    for i = 1, engine.pull_npackets do
        if not self.dev:can_receive() then
            break
        end
        local pkt = self.dev:receive()
        link.transmit(self.output.tx, pkt)
    end
end
function PcapFilter:push ()
    while not link.empty(self.input.rx) do
        local p = link.receive(self.input.rx)
        if self.accept_fn(p.data, p.length) then
            link.transmit(self.output.tx, p)
        else
            packet.free(p)
        end
    end
end
end
struct packet {
    uint16_t length;
    unsigned char data[10*1024];
};
struct link {
struct packet *packets[1024];
// the next element to be read
int read;
// the next element to be written
int write;
};
// (Some statistics counters elided)
voilà

At this point, you can rewrite Snabb
(Please do!)
But you might want to use it as-is...
Snabby design principles

- Simple > Complex
- Small > Large
- Commodity > Proprietary
simple

Compose network functions from simple parts

intel10g | reassemble | filter | fragment | intel10g

Apps independently developed
Linked together at run-time
Communicating over simple interfaces (packets and links)
small Early code budget: 10000 lines
Build in a minute
Constraints driving creativity
Secret weapon: Lua via LuaJIT
High performance with minimal fuss
small

Minimize dependencies

1 minute make budget includes Snabb and all deps (luajit, pflua, ljsyscall, dynasm)

Deliverable is single binary

./snabb --help
./snabb top
./snabb lwaftr run ...
Writing our own drivers, in Lua
User-space networking

- The data plane is our domain, not the kernel’s
- Not DPDK’s either!
- Fits in 10000-line budget
commodity

What’s special about a Snabb network function?

Not the platform (assume recent Xeon)

Not the NIC (just need a driver to inhale some packets)

Not Snabb itself (it’s Apache 2.0)
commodity

Open data sheets

Intel 82599 10Gb

Mellanox ConnectX-4 (10, 25, 40, 100Gb)

Also Linux tap interfaces, virtio host and guest
commodity

Prefer CPU over NIC where possible
Commoditize NICs – no offload
Double down on 64-bit x86 servers
status

Going on 5 years old

27 patch authors last year, 1400 non-merge commits

Deployed in a dozen sites or so

Biggest programs: NFV virtual switch, lwAFTR IPv6 transition core router, SWITCH.ch VPN

New in 2016: multi-process, guest support, 100G, control plane integration
Igalia developed “lwAFTR” (lightweight address family translation router)

Central router component of “lightweight 4-over-6” deployment

lw4o6: IPv4-as-a-service over pure IPv6 network

Think of it like a big carrier-grade NAT

20Gbps, 4MPPS per core
challenges

(1) Make it fast
(2) Make it not lose any packets
(3) Make it integrate
(4) Make it scale up and out
fast

LuaJIT does most of the work

App graph plays to LuaJIT’s strengths: lots of little loops

❖ Loop-invariant code motion boils away Lua dynamism

❖ Trace compilation punches through procedural and data abstractions

❖ Scalar replacement eliminates all intermediate allocations
Speed tips could fill a talk
Prefer FFI data structures (Lua arrays can be fine too)
Avoid data dependency chains
4MPPS: 250 ns/packet
One memory reference: 80ns
Example: hash table lookups
lossless

Max average latency for 100 packets at 4MPPS: 25 us

Max latency (512-packet receive ring buffer): 128 us

Avoid allocation

Avoid syscalls

Avoid preemption – reserved CPU cores, no hyperthreads

Avoid faults – NUMA / TLB / hugepages

Lots of tuning
Operators have monitoring and control infrastructure – command line necessary but not sufficient

Snabb now does enough YANG to integrate with an external NETCONF agents

Runtime configuration and state query, update

Avoid packet loss via multi-process protocol
2017 is the year of 100G in production Snabb; multiple coordinated data-plane processes

Also horizontal scaling via BGP/ECMP: terabit lw4o6 deployments

Work in progress!
more

Pflua: tcpdump / BPF compiler (now with native codegen!)

NFV: fast virtual switch

Perf tuning: “x-ray diffraction” of internal CPU structure via PMU registers and timelines

DynASM: generating machine code at run-time optimized for particular data structures

Automated benchmarking via Nix, Hydra, and RMarkdown!

[Your cool hack here!]
thanks!

Make a thing with Snabb!

git clone https://github.com/SnabbCo/snabb

cd snabb
make

wingo@igalia.com

@andywingo
oh no here comes the hidden track!
Storytime! Modern x86: who’s winning?
Clock speed same since years ago
Main memory just as far away
“We need to do work on data... but there’s just so much of it and it’s really far away.”

Three primary improvements:

- CPU can work on more data per cycle, once data in registers
- CPU can load more data per cycle, once it’s in cache
- CPU can make more parallel fetches to L3 and RAM at once
Networking folks can win too. Instead of chasing zero-copy, tying yourself to ever-more-proprietary features of your NIC, just take the hit once: **DDIO into L3**. Copy if you need to – copies with L3 not expensive. Software will eat the world!
Networking folks can win too.

Once in L3, you have:

- wide loads and stores via AVX2 and soon AVX-512 (64 bytes!)
- pretty good instruction-level parallelism: up to 16 concurrent L2 misses per core on haswell
- wide SIMD: checksum in software!
- software, not firmware