How to Write Your Own NIC Driver (and why)

Our experience writing 10G/100G drivers for Snabb in Lua (without NDAs)

Luke Gorrie, Asumu Takikawa

FOSDEM, 3 Feb 2018

snabb

igalia

Why write device drivers?

- Fear of eternal damnation
- Pursuit of destiny
- Lust for power

Driver Heaven & Hell

In "Driver Heaven":

- Spec is 20 pages
- Driver is 500 lines
- Lots of drivers on Github

In "Driver hell":

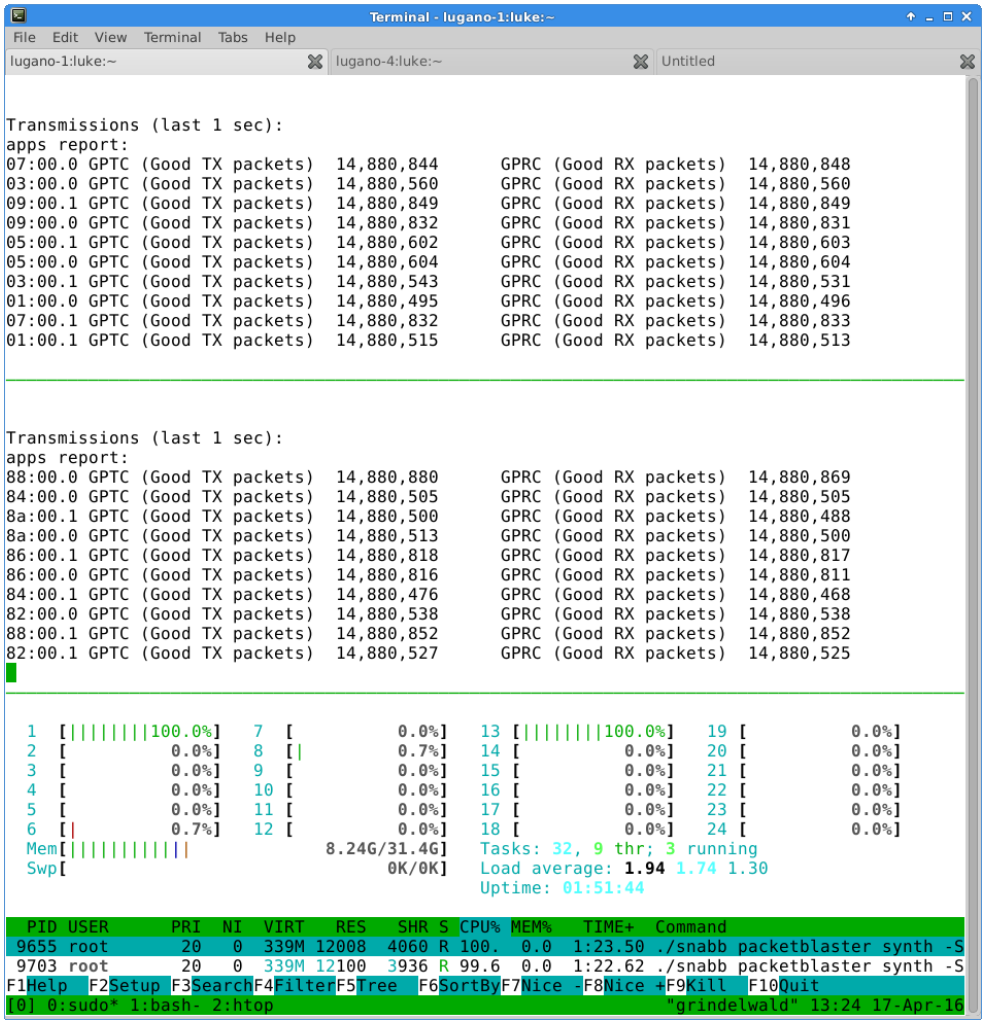
- Spec is 1k pages and secret
- Driver is 50KLOC + 500KLOC deps
- Nobody understands but vendors
- Feature creep, binary blobs...

Road to hell:

- Use HW without public docs
- Use drivers you don't understand
- Depend on vendors for everything

Road to heaven:

- Insist on hardware docs
- Read and understand
- Write drivers together
- Engage vendors
- Seek out kindred spirits



Lust for power

- Packetblaster
- Firehose
- Sidespy

Part 2: How

Luke gave you the ***why***

(why **driver heaven**)

This part is about ***how***

(how Snabb's drivers work)

This part just gives a **flavor** of implementation

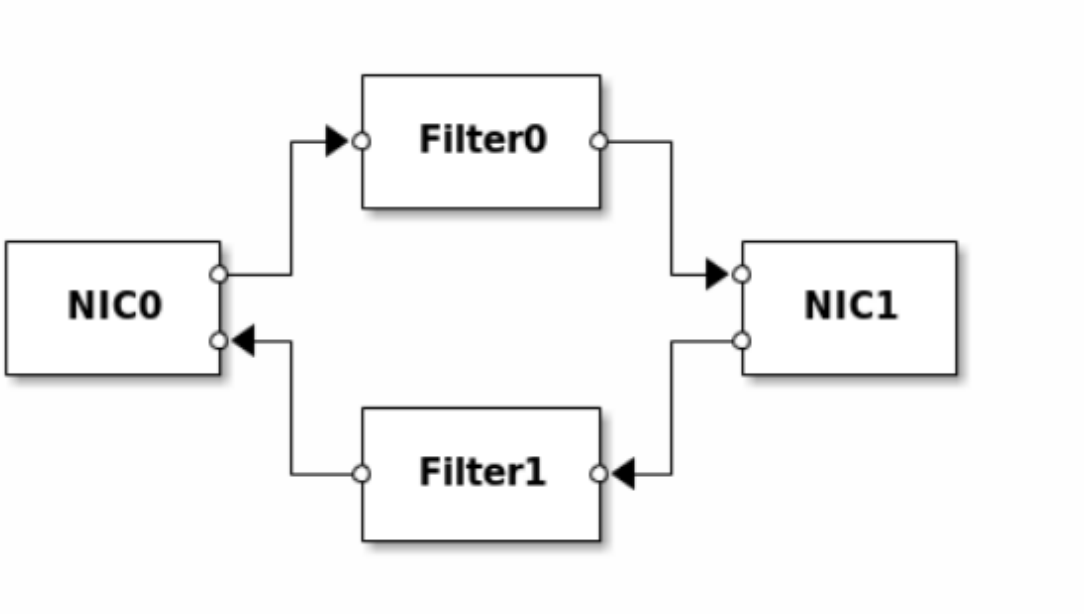
Big picture: **1,485 LOC** of Lua code is pretty high-level

LuaJIT - easy to understand + abstractions with low cost

Receiving packets

Snabb driver is an **app**

(like everything else)


Filter0
NIC0
NIC1
Filter1

App = obj with some methods

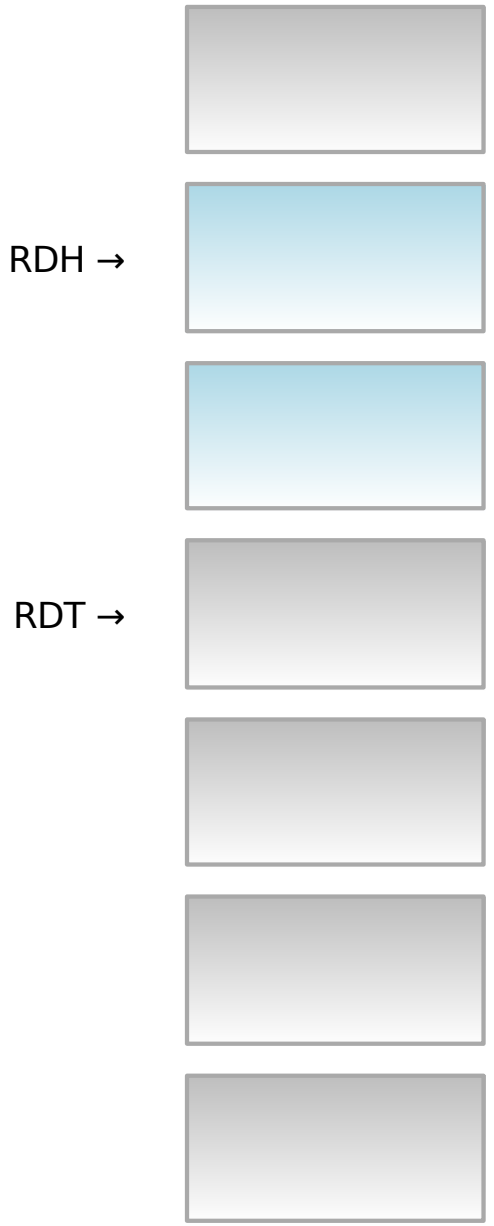
`new`, `push`, `pull`

Let's consider **`pull`** (Rx)

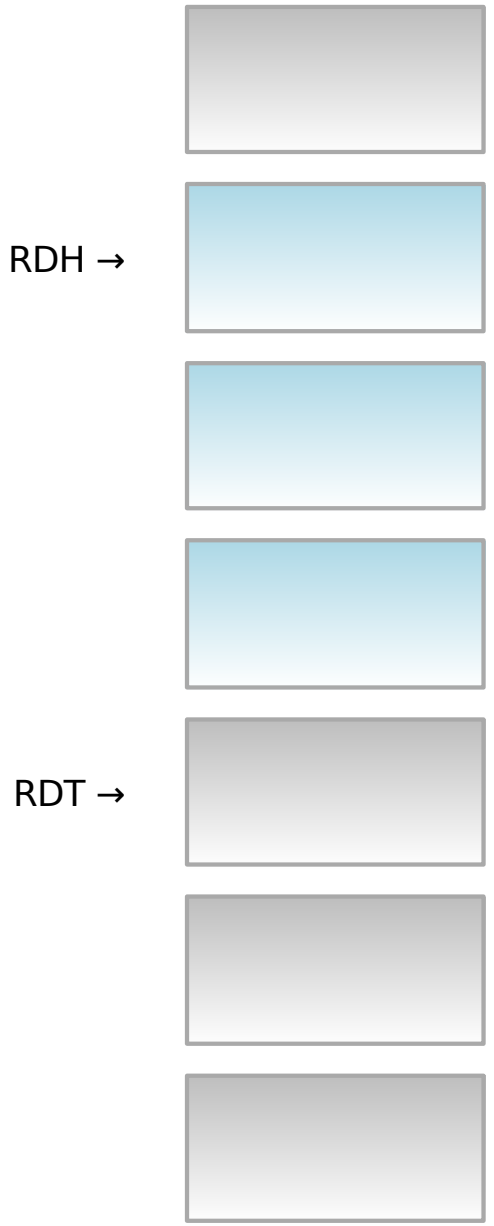
Driver maintains **ring buffer**

NIC **DMA**s pkts to ring

Driver sets base addr register
Maintains tail reg (**RDT**)



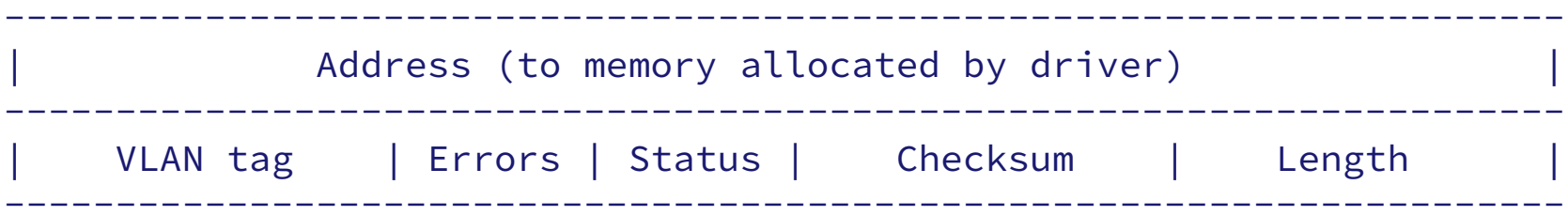
Driver sets base addr register
Maintains tail reg (**RDT**)



How to access registers

```
-- self        => driver object
-- self.r      => registers object
-- self.r.RDT => tail register
self.r.RDT()

-- increment (or loop) tail reg after reading pkt
self.r.RDT(band(self.r.RDT() + 1, ring_size - 1))
```

Ring buffer representation



```
rxdesc_t = ffi.typeof([[
   struct {
      uint64_t address;
      uint16_t length, cksum;
      uint8_t status, errors;
      uint16_t vlan;
   } __attribute__((packed))
]])

-- allocate driver's ring buffer
local buffer_size = sizeof(rxdesc_t) * ring_size
self.rxdesc = memory.dma_alloc(buffer_size)
```

```
function Driver:pull ()
   self:sync_receive() -- sync driver & HW pointers

   for i = 1, engine.pull_npackets do
      -- check ptrs for pkt availability
      if not self:can_receive() then break end

      -- move pkt from rx ring to Snabb
      local pkt = self:receive()
      link.transmit(self.output.tx, pkt)
   end

   -- alloc new buffers for ring
   self:add_receive_buffers()
end
```

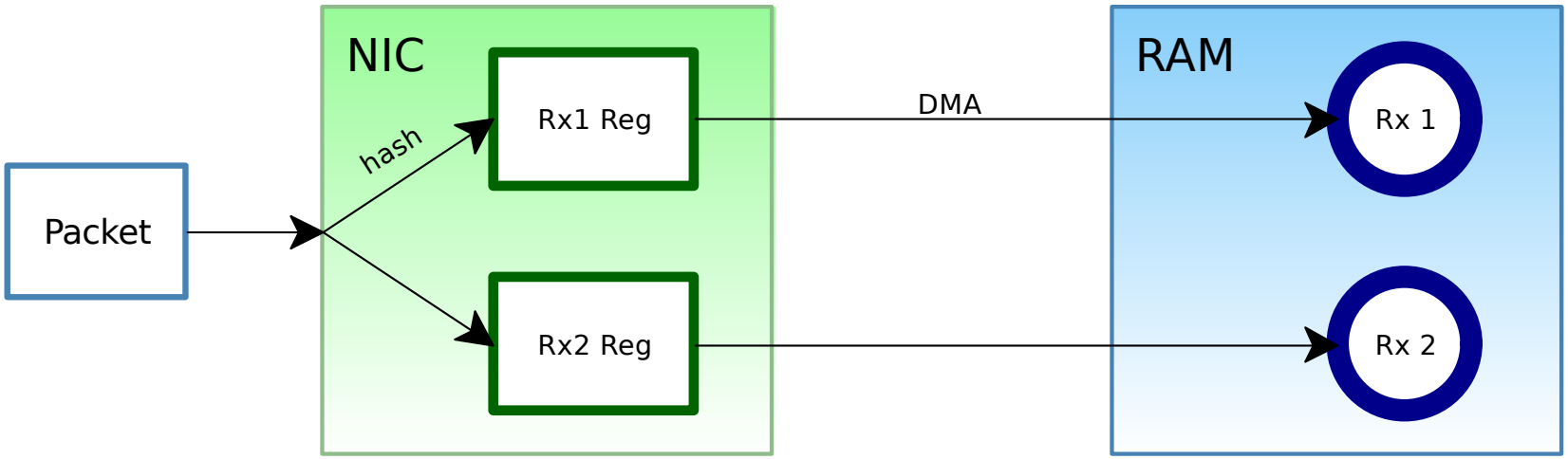
```
-- method that fetches next packet to read
function Driver:receive()
   -- copy of tail register
   local tail = self.r.RDT()

   -- get length from ring buffer
   -- get actual packet from array of packets
   local len = self.rxdesc[tail]
   local p   = self.rxpackets[tail]
   p.length = len

   -- delete packet from array, increment tail
   self.rxpackets[tail] = nil
   self.r.RDT(band(tail + 1, ring_size - 1))

   return p
end
```

Recent & future work

RSS + VMDq support

Scales to multiple cores

Hash flows to distribute to separate queues


NIC
RAM
Packet
hash
Rx1 Reg
Rx2 Reg
DMA
Rx 1
Rx 2

Future: XL710 (40G) support?

Thanks!

https://github.com/snabbco/snabb

snabb

igalia