Lua for Low-Level Programming

some notes from the trenches

Lua Workshop Moscow 2014

Javier Guerra G.
javier@guerrag.com

What's this about

- We've been writing SnabbSwitch fully in LuaJIT for a while.
- We want to share a little about things we've found.
- It's not a general LuaJIT optimization guide!
- Please go to the LuaJIT wiki for extra enlightenment.

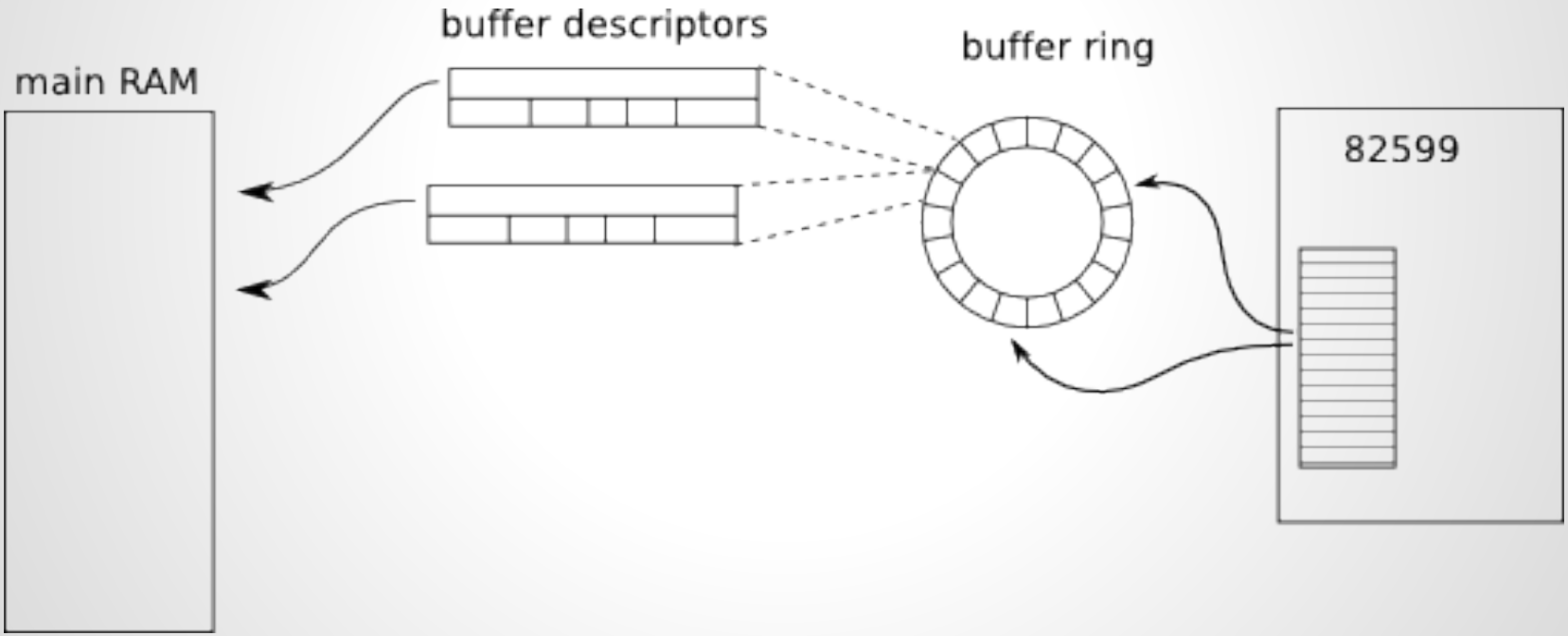
What is Snabb Switch?

- Luke Gorrie's brainchild.
- It's a *high performance* software switch.
- It has its own 10Gb Ethernet driver.
- It does zero-copy from inside a VM to the controller.
- It's written in Lua!
- Very extensible, in Lua
- Totally OSS, developed in the open. https://github.com/SnabbCo/snabbswitch, http://groups.google.com/group/snabb-devel

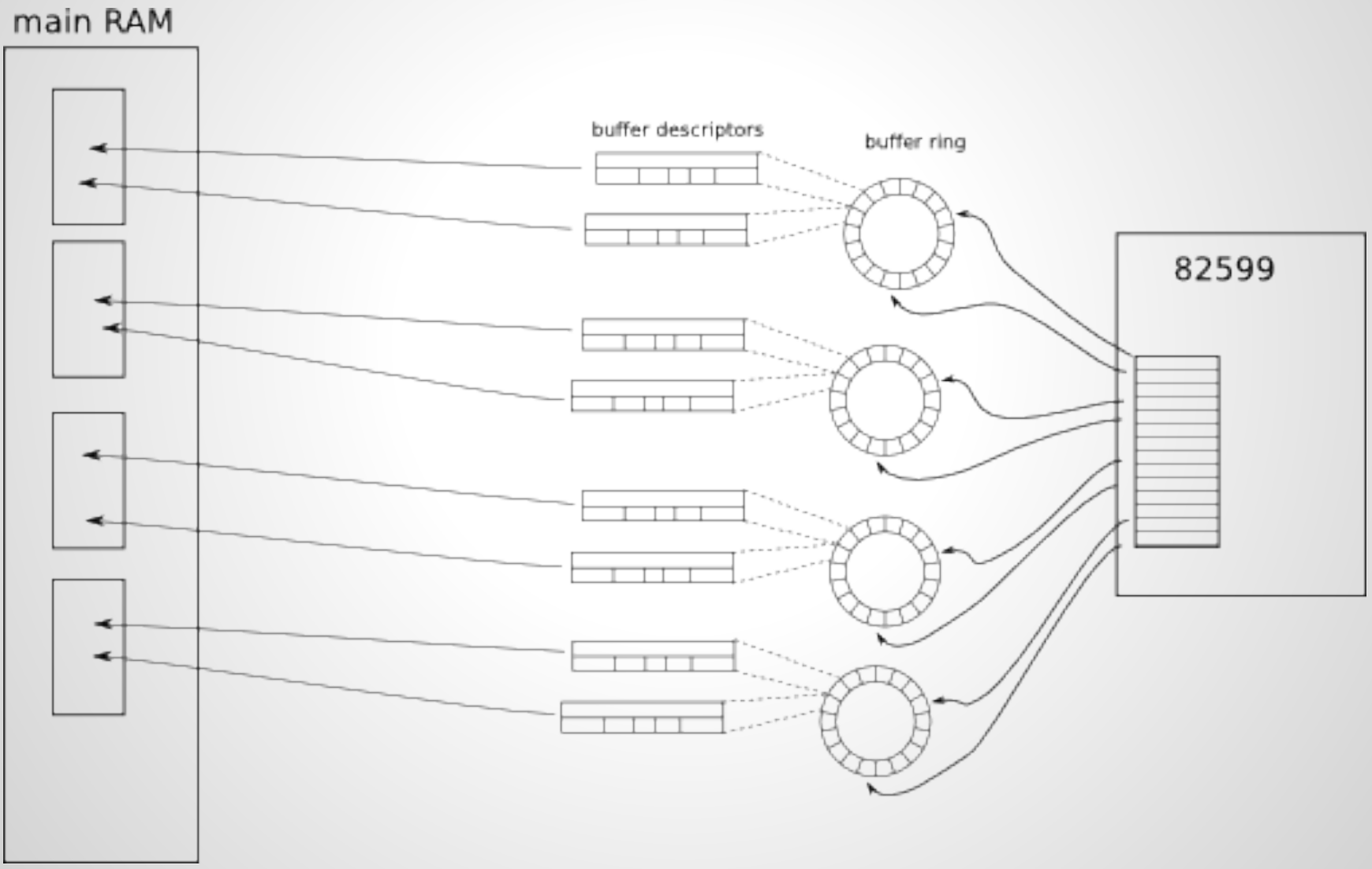
Why Lua?

- It’s fast, especially when compiled by LuaJIT.
- The FFI makes it easy to talk to other layers and to hardware.
- Has to be deeply customized by non-hackers, network engineers know what they want.

Snabb Switch NIC driver



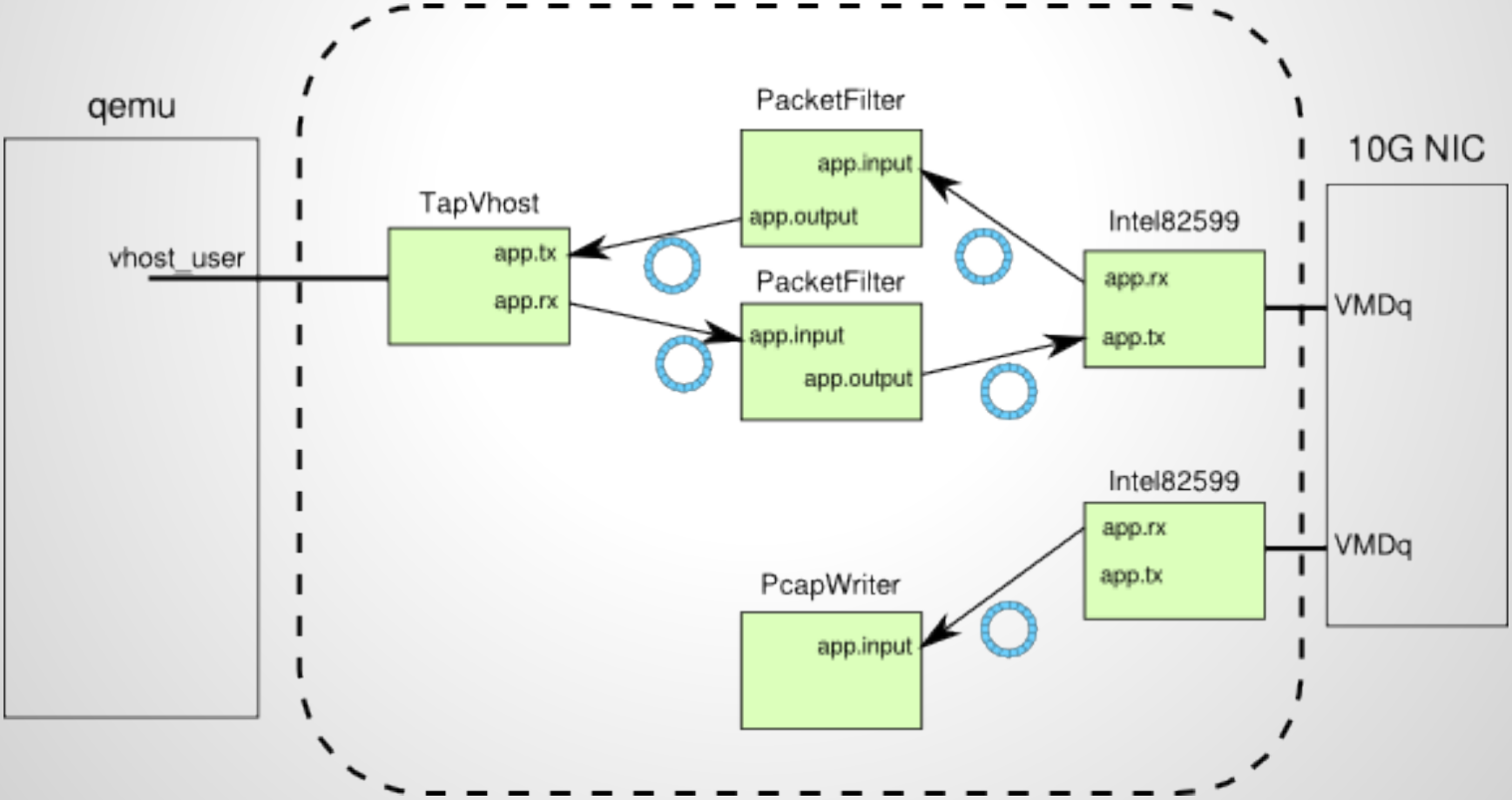
Drive into VMs at wire speed



App structure

- Each app instance receives a name and a config string, usually parsed as a Lua table.
- Tied into a graph by strings like '`eth0.rx -> filterA.tx`'
- Links between apps are ring buffers: an FFI struct with Lua functions.
- "breath" cycle, with "inhale" and "exhale" steps, where apps pull and push packets from/to links.

App structure



LuaJIT lessons

- It is mindblowingly fast.
- … but it can slow down if you don't watch it carefully.
- `table.pairs()` is bad.
- Adaptor functions and shallow OOP are almost free.
- Aim for 100% compiled inner loops.
- Inner loops are wider than they seem.
- Profile everything!

FFI for interaction with hardware/libs

```
rxdesc_t = ffi.typeof [[
   union {
      struct {
         uint64_t address;
         uint64_t dd;
      } __attribute__((packed)) data;
      struct {
         uint16_t rsstype_packet_type;
         uint16_t rsccnt_hdrlen_sph;
         uint32_t xargs;
         uint32_t xstatus_xerror;
         uint16_t pkt_len;
         uint16_t vlan;
      } __attribute__((packed)) wb;
   }
]]
```

FFI for interaction with hardware/libs

```
function M_sf:receive ()
   assert(self.rdh ~= self.rxnext)
   local p = packet.allocate()
   repeat
      local wb = self.rxdesc[self.rxnext].wb
      if band(wb.xstatus_xerror, 1) == 1  then -- Descriptor Done
         local b = self.rxbuffers[self.rxnext]
         packet.add_iovec(p, b, wb.pkt_len)
         self.rxnext = band(self.rxnext + 1, num_descriptors - 1)
      end
   until band(wb.xstatus_xerror, 2) == 2   -- End Of Packet
   return p
end
```

FFI structs for speed…

```
-- freelist.lua

function new (type, size)
   return { nfree = 0,
            max = size,
            list = ffi.new(type.."[?]", size) }
end

function add (freelist, element)
   freelist.list[freelist.nfree] = element
   freelist.nfree = freelist.nfree + 1
end

function remove (freelist)
   if freelist.nfree == 0 then return nil end
   freelist.nfree = freelist.nfree - 1
   return freelist.list[freelist.nfree]
end

function nfree (freelist)
   return freelist.nfree
end
```

FFI structs for speed… maybe not

```
-- freelist.lua

function new (type, size)
   local typ = ffi.typeof([[
      struct {
         int nfree, max;
         $ list[?];
      }
   ]], ffi.typeof(type))
   return typ(size, {nfree=0, max=size})
end

function add (freelist, element)
   freelist.list[freelist.nfree] = element
   freelist.nfree = freelist.nfree + 1
end
function remove (freelist)
   if freelist.nfree == 0 then return nil end
   freelist.nfree = freelist.nfree - 1
   return freelist.list[freelist.nfree]
end
function nfree (freelist)
   return freelist.nfree
end
```

Best things in life are free

Part of link API:

```
function empty (r)
   return r.read == r.write
end

function full (r)
   return band(r.write + 1, size - 1) == r.read
end

function nreadable (r)
   if r.read > r.write  then
      return r.write + size - r.read
   else
      return r.write - r.read
   end
end

function nwritable (r)
   return max - nreadable(r)
end
```

Best things in life are free

which one is faster?

the readable version...

```
while not link.empty(inport) and not link.full(outport) do
   transmit(outport, receive(inport))
end
```

… or the numerical one?

```
for _ = 1, math.min(link.nreadable(inport), link.nwritable(outport)) do
   transmit(outport, receive(inport))
end
```

variable scoping

```
-- Transmit packets from the app input queue to the VM.
function VirtioNetDevice:transmit_packets_to_vm ()
   local l = self.owner.input.rx
   if not l then return end
   local should_continue = not self.not_enough_vm_bufers

   while (not link.empty(l)) and should_continue do
      local p = link.receive(l)

      -- ensure all data is in a single buffer
      if p.niovecs > 1 then
         packet.coalesce(p)
      end

      local iovec = p.iovecs[0]
      should_continue, b = self:vm_buffer(iovec)

      -- fill in the virtio header
      local virtio_hdr = b.origin.info.virtio.header_pointer
      ffi.copy(virtio_hdr, p.info, packet_info_size)

      local used = self.txring.used.ring[self.txused%self.tx_vring_num]
      local v = b.origin.info.virtio
      --assert(v.header_id ~= invalid_header_id)
      used.id = v.header_id
      used.len = virtio_net_hdr_size + iovec.length
      self.txused = (self.txused + 1) % 65536

      packet.deref(p)
   end

   if not should_continue then
      -- not enough buffers detected, verify once again
      self.not_enough_vm_bufers = not self:more_vm_buffers()
   end

   if self.txring.used.idx ~= self.txused then
      self.txring.used.idx = self.txused
      if bit.band(self.txring.avail.flags, C.VRING_F_NO_INTERRUPT) == 0 then
         C.write(self.callfd[0], eventfd_one, 8)
      end
   end
end
```

variable scoping

```
   -- fill in the virtio header
   local virtio_hdr = b.origin.info.virtio.header_pointer
   ffi.copy(virtio_hdr, p.info, packet_info_size)

   local used = self.txring.used.ring[self.txused%self.tx_vring_num]
   local v = b.origin.info.virtio
   --assert(v.header_id ~= invalid_header_id)
   used.id = v.header_id
   used.len = virtio_net_hdr_size + iovec.length
   self.txused = (self.txused + 1) % 65536

   packet.deref(p)
end
```

variable scoping

```
   -- fill in the virtio header
   do local virtio_hdr = b.origin.info.virtio.header_pointer
      ffi.copy(virtio_hdr, p.info, packet_info_size)
   end

   do local used = self.txring.used.ring[
                     band(self.txused, self.tx_vring_num-1)]
      local v = b.origin.info.virtio
      --assert(v.header_id ~= invalid_header_id)
      used.id = v.header_id
      used.len = virtio_net_hdr_size + iovec.length
   end

   packet.deref(p)

   self.txused = band(self.txused + 1, 65535)
end
```

credits: Antonio Nikishaev

flow patterns - #1: finish fast

Good:

```
function M_sf:can_receive ()

   local nxt = self.rxnext
   while nxt ~= self.rdh do
    local flags = bit.band(self.rxdesc[nxt].wb.xstatus_xerror, 3)
    if flags == 3 then return true end
    if flags == 0 then return false end
    nxt = (nxt + 1) % num_descriptors
   end
   return false
end
```

flow patterns - #1: finish fast

Better:

```
function M_sf:can_receive ()
   local result = false
   local nxt = self.rxnext
   while nxt ~= self.rdh do
    local flags = bit.band(self.rxdesc[nxt].wb.xstatus_xerror, 3)
    if flags == 3 then result = true; break end
    if flags == 0 then break end
    nxt = (nxt + 1) % num_descriptors
   end
   return result
end
```

credits: Alexander Gall

flow patterns - #1: finish fast

Best:

```
function M_sf:can_receive ()
   return self.rxnext ~= self.rdh  and
      band(self.rxdesc[self.rxnext].wb.xstatus_xerror, 1) == 1
end
```

credits: Alexander Gall

flow patterns - #2: avoid little loops

Initially, a small loop

```
function M_sf:receive ()
   assert(self.rdh ~= self.rxnext)
   local p = packet.allocate()
   repeat
      local wb = self.rxdesc[self.rxnext].wb
      if band(wb.xstatus_xerror, 1) == 1  then -- Descriptor Done
         local b = self.rxbuffers[self.rxnext]
         packet.add_iovec(p, b, wb.pkt_len)
         self.rxnext = band(self.rxnext + 1, num_descriptors - 1)
      end
   until band(wb.xstatus_xerror, 2) == 2   -- End Of Packet
   return p
end
```

flow patterns - #2: avoid little loops

Obvious next step: an ugly unroll

```
function M_sf:receive ()
   assert(self.rdh ~= self.rxnext)
   local p = packet.allocate()
   do
      local wb = self.rxdesc[self.rxnext].wb
      if band(wb.xstatus_xerror, 1) == 1 then -- Descriptor Done
         local b = self.rxbuffers[self.rxnext]
         packet.add_iovec(p, b, wb.pkt_len)
         self.rxnext = band(self.rxnext + 1, num_descriptors - 1)
      end
   end
   if not band(wb.xstatus_xerror, 2) == 2 then  -- End Of Packet
      local wb = self.rxdesc[self.rxnext].wb
      if band(wb.xstatus_xerror, 1) == 1 then -- Descriptor Done
         local b = self.rxbuffers[self.rxnext]
         packet.add_iovec(p, b, wb.pkt_len)
         self.rxnext = band(self.rxnext + 1, num_descriptors - 1)
      end
      if not band(wb.xstatus_xerror, 2) == 2 then  -- End Of Packet
         repeat
            local wb = self.rxdesc[self.rxnext].wb
            if band(wb.xstatus_xerror, 1) == 1 then -- Descriptor Done
               local b = self.rxbuffers[self.rxnext]
               packet.add_iovec(p, b, wb.pkt_len)
               self.rxnext = band(self.rxnext + 1, num_descriptors - 1)
            end
         until band(wb.xstatus_xerror, 2) == 2 -- End Of Packet
      end
   end
   return p
end
```

flow patterns - #2: avoid little loops

Trust the Inliner:

```
local function receive_aux(self, p)
   local wb = self.rxdesc[self.rxnext].wb
   if band(wb.xstatus_xerror, 1) == 1  then -- Descriptor Done
      local b = self.rxbuffers[self.rxnext]
      packet.add_iovec(p, b, wb.pkt_len)
      self.rxnext = band(self.rxnext + 1, num_descriptors - 1)
   end
   return band(wb.xstatus_xerror, 2) == 2   -- End Of Packet
end

function M_sf:receive ()
   assert(self.rdh ~= self.rxnext)
   local p = packet.allocate()
   if not receive_aux(self, p)  then
      if not receive_aux(self, p)  then
         repeat
         until receive_aux(self, p)
      end
   end
   return p
end
```

credits: Alexander Gall

conclusions

not too hard, not even too weird, but very unforgiving.

Don’t assume, profile!

Thanks!