Rapid prototyping of DPDK applications with libmoon

Paul Emmerich
emmericp@net.in.tum.de
Technical University of Munich
Chair of Network Architectures and Services
DPDK Summit, 27.9.2017
About me

- PhD student at Technical University of Munich
- Started in 2014, DPDK user since 2013
- PhD thesis about testing network devices
- Built the MoonGen packet generator for this
  - Talked about MoonGen here last year
  - Often used in academia nowadays :)
Using DPDK in academia

- Lots of one-off prototypes or implementations deployed only once
- Proof-of-concepts, benchmarks, traffic analysis
- Work often “outsourced“ to student theses
  - Advisor for > 10 Bachelor’s and Master’s Theses using DPDK

Teaching DPDK
- Exercise for our Advanced Computer Networks lecture: build a router
- Simple DPDK-based router, 4 VMs for each student for testing
- ~35 students participated
- Provided boilerplate code for device and memory initialization
- Most common mistake: handling mbufs
Problems with DPDK for prototypes

• Lots of boilerplate code required for initialization
• Things that should be simple often require lots of code
• Build system can be… annoying
• Hard to get students started

• Typical time frame for a student project: 4-6 months part-time
  – Need to understand and research the general topic
  – Design and implement a prototype
  – Often: brush up on C skills before
  – Analyze results, write thesis

• Hard to really get into DPDK in this scope for most students
The libmoon library

- libmoon is a Lua wrapper for DPDK
- Originally written for the MoonGen packet generator

Why Lua?
- Scripting language
- Can call existing C/C++ functions without overhead
- As fast as C/C++

Comes with all the utilities you need for prototypes
- Simplified device initialization with reasonable defaults
- Command line parsing library
- Predefined helper threads for statistics, ARP, ICMP, LACP, …
- MoonGen packet library for structured access to packet data
Example: l2-forward.lua

• libmoon example script, similar to DPDK’s l2fwd
  − Multi-threaded
  − Multi-queue with RSS
  − Prints statistics
• 40 lines of code (without comments)

• DPDK l2fwd: ~650 lines of code (without comments)
  − Used to be more in older versions, so it improved!

• We based our prototypes on the DPDK examples before libmoon
• Huge mess of copied & pasted code just to get basic functionality, e.g.:
  − IO statistics
  − Device configuration
Example: reflector.lua

- Reflects packets on multiple links, multi-thread/queue with RSS
- Worker thread, started once per queue pair

```lua
function reflector(rxQ, txQ)
    local bufs = memory.bufArray()
    while lm.running() do
        local rx = rxQ:tryRecv(bufs, 1000)
        for i = 1, rx do
            local pkt = bufs[i]:getEthernetPacket()
            local tmp = pkt.eth:getDst()
            pkt.eth:setDst(pkt.eth:getSrc())
            pkt.eth:setSrc(tmp)
            local vlan = bufs[i]:getVlan()
            if vlan then
                bufs[i]:setVlan(vlan)
            end
        end
        txQ:sendN(bufs, rx)
    end
end
```

Example: reflector.lua
Small things that can help a lot
Check out libmoon on GitHub

libmoon comes with a lot of examples to get started

https://github.com/libmoon/libmoon
Questions?
```lua
function master(args)
    local lacpQueues = {}
    for i, dev in ipairs(args.dev) do
        local dev = device.config{
            port = dev,
            rxQueues = args.threads + (args.lacp and 1 or 0),
            txQueues = args.threads + (args.lacp and 1 or 0),
            rssQueues = args.threads
        }
        -- last queue for lacp
        if args.lacp then
            table.insert(lacpQueues,
                {rxQueue = dev:getRxQueue(args.threads), txQueue = dev:getTxQueue(args.threads)})
        end
        args.dev[i] = dev
    end
    device.waitForLinks()

    -- setup lacp if requested
    if args.lacp then
        lacp.startLacpTask("bond0", lacpQueues)
        lacp.waitForLink("bond0")
    end

    -- print statistics
    stats.startStatsTask{devices = args.dev}
    for i, dev in ipairs(args.dev) do
        for i = 1, args.threads do
            lm.startTask("reflector", dev:getRxQueue(i - 1), dev:getTxQueue(i - 1))
        end
    end
    lm.waitForTasks()
end
```
reflector.lua boilerplate and CLI

```lua
local lm = require "libmoon"
local memory = require "memory"
local device = require "device"
local stats = require "stats"
local lACP = require „proto.lacp"

function configure(parser)
    parser:argument("dev", "Devices to use."):args("+"):convert(tonumber)
    parser:option("-t --threads",
        "Number of threads per device."):args(1):convert(tonumber):default(1)
    parser:flag("-l --lacp", "Try to setup an LACP channel.")
end
```