DPDK

DATA PLANE DEVELOPMENT KIT

Fast Prototyping DPDK Apps in Containernet

ANDREW WANG, PRINCIPAL SOFTWARE ENGINEER

COMCAST

Who Am I

DPDK
DATA PLANE DEVELOPMENT KIT

- Principal Software Engineer at Comcast
- Past projects
  - GDB
  - Content based routing in ad hoc networks
  - Key value stores
  - Load balancing services

Agenda

- Overview
  - What happened when we started writing network functions
- Containernet
  - What it is, what it is capable of and why should I care
- DPDK in Containernet
  - Setup
  - Running
- Short Demo

So we wanted to write fast network functions using DPDK

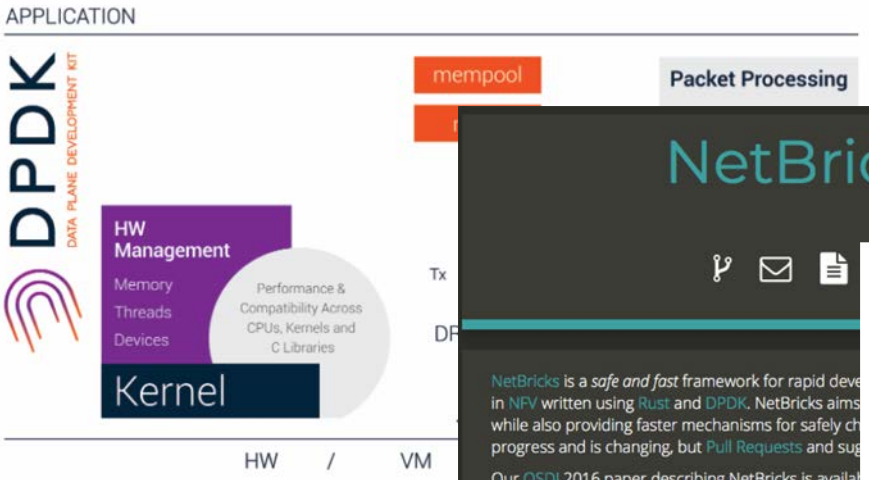
DATA PLANE DEVELOPMENT KIT

Photo by Mathew Schwartz on Unsplash

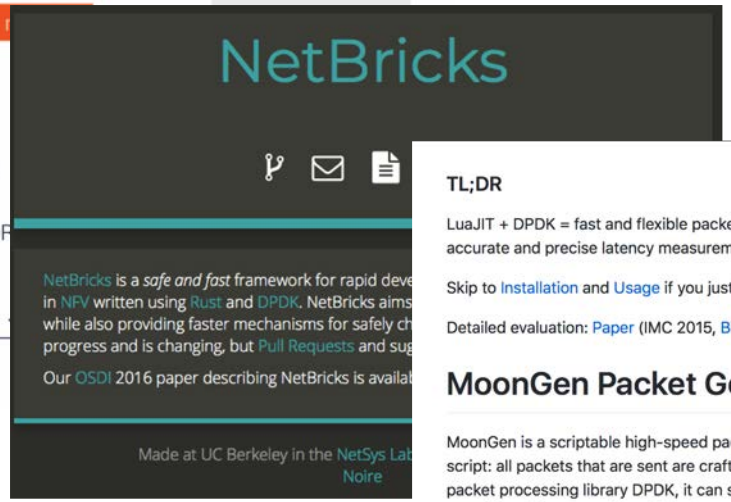
And everyone was excited, including us!

Photo by Val Vesa on Unsplash

We set up everything we needed…

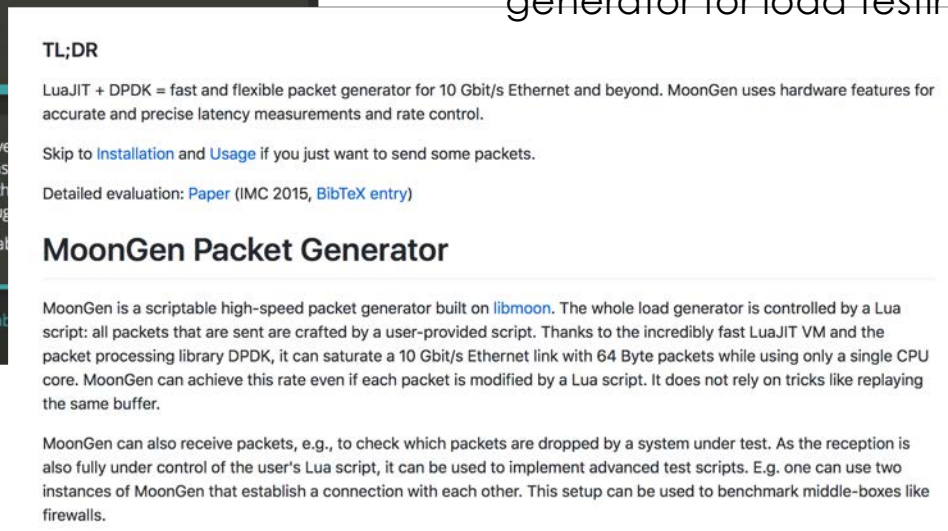


For fast user layer packet processing



Framework to write network functions in Rust & DPDK

High speed packet generator for load testing



TL;DR

LuaJIT + DPDK = fast and flexible packet generator for 10 Gbit/s Ethernet and beyond. MoonGen uses hardware features for accurate and precise latency measurements and rate control.

Skip to Installation and Usage if you just want to send some packets.

Detailed evaluation: Paper (IMC 2015, BibTeX entry)

MoonGen Packet Generator

MoonGen is a scriptable high-speed packet generator built on libmoon. The whole load generator is controlled by a Lua script: all packets that are sent are crafted by a user-provided script. Thanks to the incredibly fast LuaJIT VM and the packet processing library DPDK, it can saturate a 10 Gbit/s Ethernet link with 64 Byte packets while using only a single CPU core. MoonGen can achieve this rate even if each packet is modified by a Lua script. It does not rely on tricks like replaying the same buffer.

MoonGen can also receive packets, e.g., to check which packets are dropped by a system under test. As the reception is also fully under control of the user's Lua script, it can be used to implement advanced test scripts. E.g. one can use two instances of MoonGen that establish a connection with each other. This setup can be used to benchmark middle-boxes like firewalls.

But then we stopped and thought

DPDK
DATA PLANE DEVELOPMENT KIT

- Where can we *run* this?
- How can we *verify* correct behavior?
- How easy is it to *debug* any issues found?

VMs *How about VMs in a single host?*

DPDK
DATA PLANE DEVELOPMENT KIT

- Each host in the network is a VM
- Use host-only networking to connect multiple VMs
- Use tools like Vagrant and Virtualbox

VMs *The problem with VMs*

DPDK
DATA PLANE DEVELOPMENT KIT

- High resource utilization
  - CPU
  - RAM
- Virtualization and Anycast
  - Same IP address bound to multiple interfaces creates problems

Lab

DPDK
DATA PLANE DEVELOPMENT KIT

- Dedicated space
- Duplicate approximate network conditions where we want to deploy our functions

Lab *The problem with Lab*

DPDK
DATA PLANE DEVELOPMENT KIT

- Not under our control
  - Hard to make any changes
  - Resources can be pre-empted for other higher priority projects
- Hard to debug
- Scheduling experiments

Production networks?

DPDK
DATA PLANE DEVELOPMENT KIT

SSSUUUUUREEEE!

Yeah, right!

What is Containernet?

DPDK
DATA PLANE DEVELOPMENT KIT

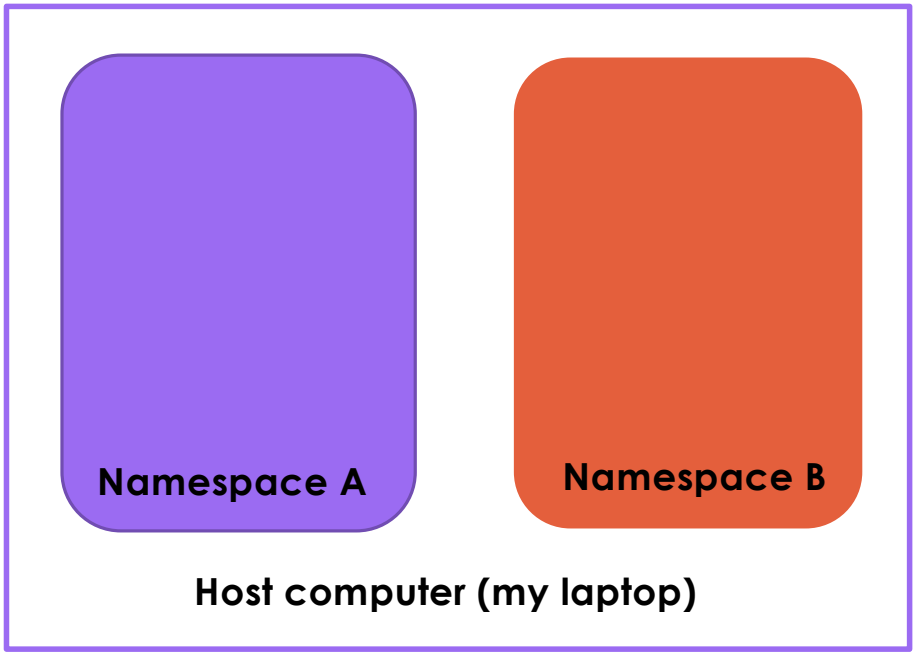
- Fork of Mininet that supports using Docker containers as Mininet hosts
  - https://containernet.github.io
  - M. Peuster, H. Karl, and S. v. Rossem: MeDICINE: Rapid Prototyping of Production-Ready Network Services in Multi-PoP Environments. IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, pp. 148-153. doi: 10.1109/NFV-SDN.2016.7919490. (2016)
- What is Mininet?
  - From http://mininet.org/
    - "An Instant Virtual Network on your Laptop (or other PC)"
    - "Mininet is a *network emulator*, or perhaps more precisely a *network emulation orchestration system*. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code."

How does it work? *Nodes*

DPDK
DATA PLANE DEVELOPMENT KIT

Network nodes are *network namespaces*

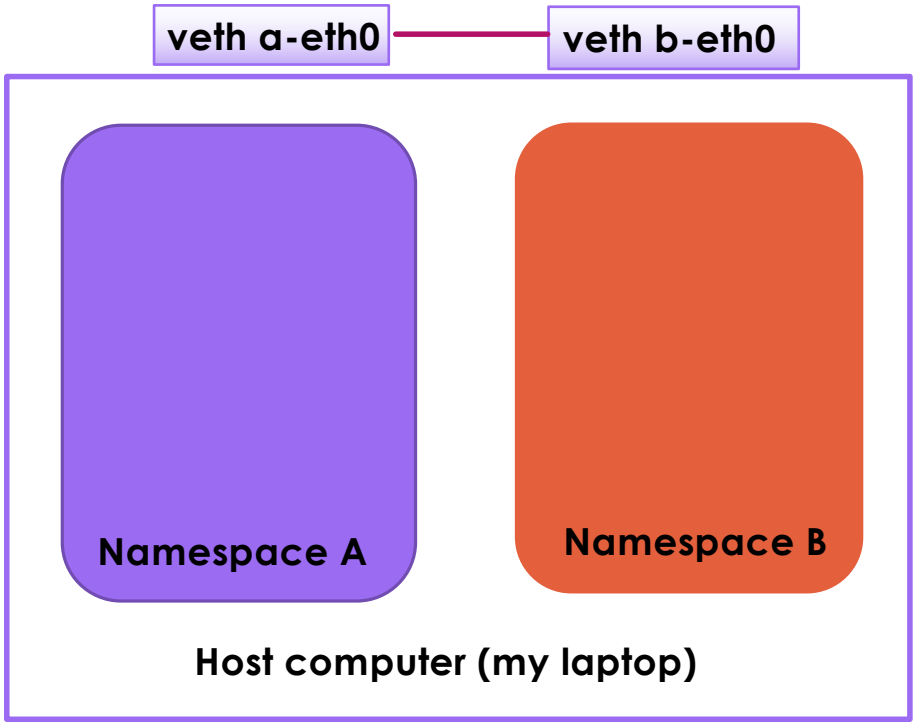
- Processes in a network namespace do not have access to the host's network interfaces
  - Think how `chroot` removes access to the host's root filesystem
  - In addition to network interfaces, the routing tables and iptables rules are different
  - Processes started in a network namespace inherit the view of the namespace
  - `ip netns add <name>`



How does it work? *Links*

Network links are established via virtual Ethernet (`veth`) interface pairs

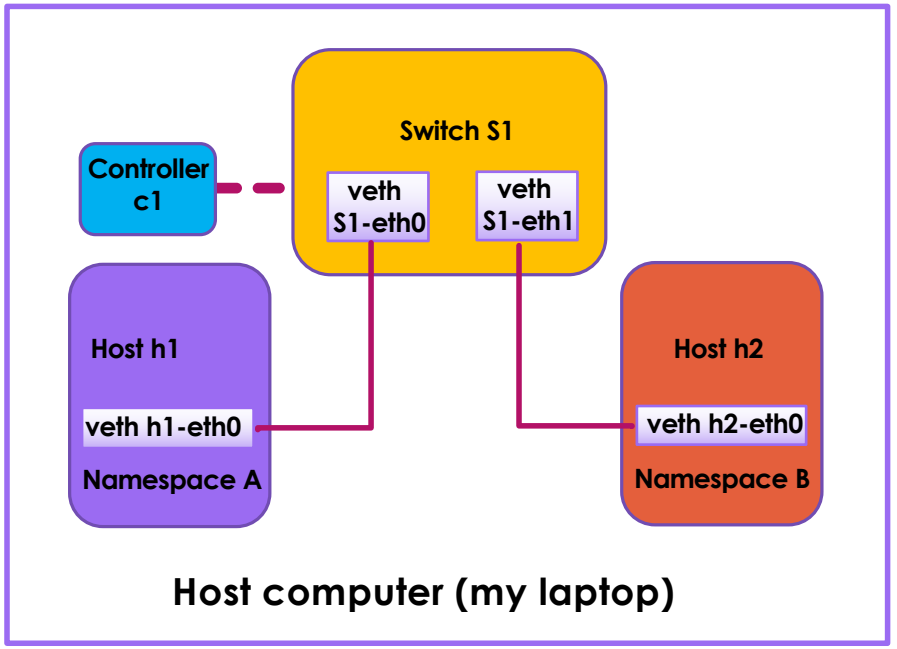
- "Act as tunnels between network namespaces to create a bridge to a physical network device in another namespace"
- "Created in interconnected pairs […] packets transmitted on one device in the pair are immediately received on the other device"
  - `ip link add <interface name> type veth peer name <interface pair name>`
  - `ip link set <interface name> netns <namespace>`



How does it work? *Switches*

The default switch is Open vSwitch running in kernel space, connected to the OpenFlow reference controller.

- Default behavior of the switch is the same as an Ethernet learning switch.
- Connect multiple hosts
- By default switches live in the host name space, but moving them to their own namespaces is supported



How does it work? *Setup*

DPDK
DATA PLANE DEVELOPMENT KIT

Python API

```
#!/usr/bin/python
"""
This is the most simple example to showcase Containernet.
"""
from mininet.net import Containernet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.log import info, setLogLevel
setLogLevel('info')

net = Containernet(controller=Controller)
info('*** Adding controller\n')
net.addController('c0')
info('*** Adding docker containers\n')
d1 = net.addDocker('d1', ip='10.0.0.251', dimage="ubuntu:trusty")
d2 = net.addDocker('d2', ip='10.0.0.252', dimage="ubuntu:trusty")
info('*** Adding switches\n')
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
info('*** Creating links\n')
net.addLink(d1, s1)
net.addLink(s1, s2, cls=TCLink, delay='100ms', bw=1)
net.addLink(s2, d2)
info('*** Starting network\n')
net.start()
info('*** Testing connectivity\n')
net.ping([d1, d2])
info('*** Running CLI\n')
CLI(net)
info('*** Stopping network')
net.stop()
```

https://github.com/containernet/containernet/blob/master/examples/containernet_example.py

How does it work? *Python API*

DPDK
DATA PLANE DEVELOPMENT KIT

- Python wraps around commands to create switches, veth, namespaces
- Can execute commands in each “host”

How does it work? *Running*

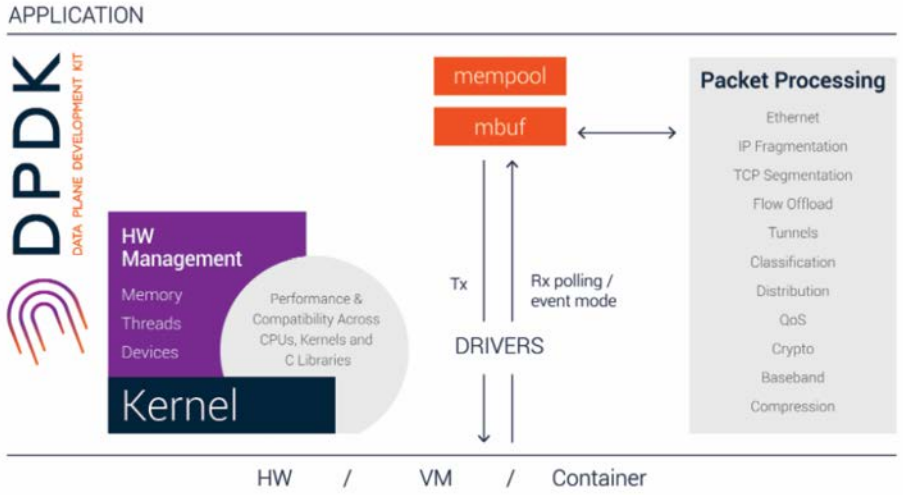
DPDK
DATA PLANE DEVELOPMENT KIT

Containernet (Mininet) performs **emulation**

- Setup the topology desired, and start traffic sources, routing software, etc
- Software runs as-is, interacting with real network stack, at wall clock speed
  - When TCP BBR was released as a linux module, it could be run in Mininet
  - Shared resources limited by hardware speed (cannot emulate link speed faster than supported by underlying hardware)
- Linux only
- Ethernet links

DPDK

- Setup DPDK to run in containernet
- Aim to verify correct behavior, not performance
  - Environmental Abstraction Layer
  - Poll Mode Driver: AF_PACKET



DPDK in Containernet

- AF_PACKET PMD
  - AF_PACKET: “raw socket”, bypasses normal in-kernel TCP/IP processing
  - In MoonGen, we pass the following config parameters

```
"--file-prefix", "vdev",
"--no-pci",
"--vdev=eth_af_packet,iface=<interface name>"
```

  - In Netbricks, we pass the following config line for `[[dpdk.port]]`

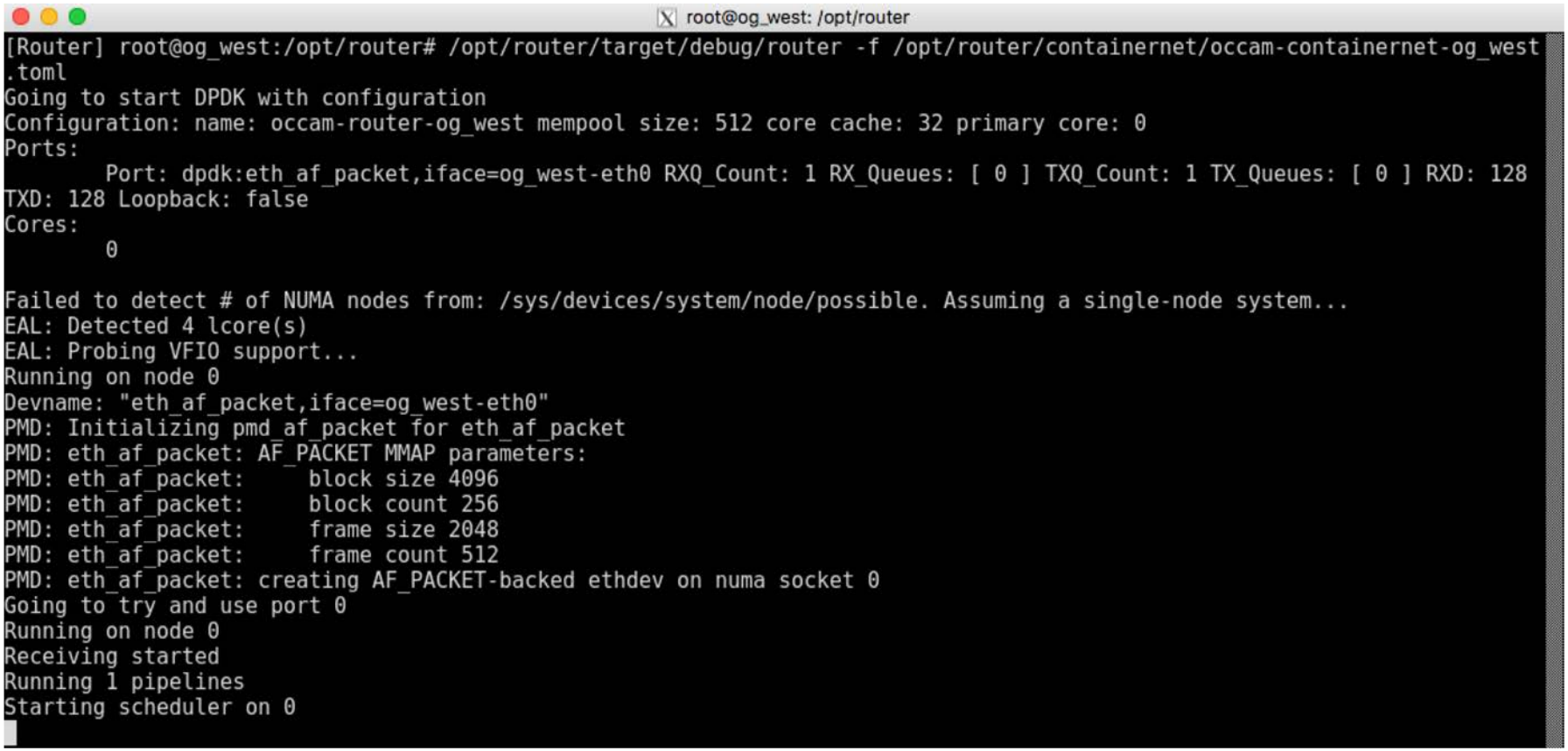
```
"dpdk:eth_af_packet,iface=<interface name>"
```

- Application name
  - Apps running on different containernet nodes must have different names

DPDK in Containernet

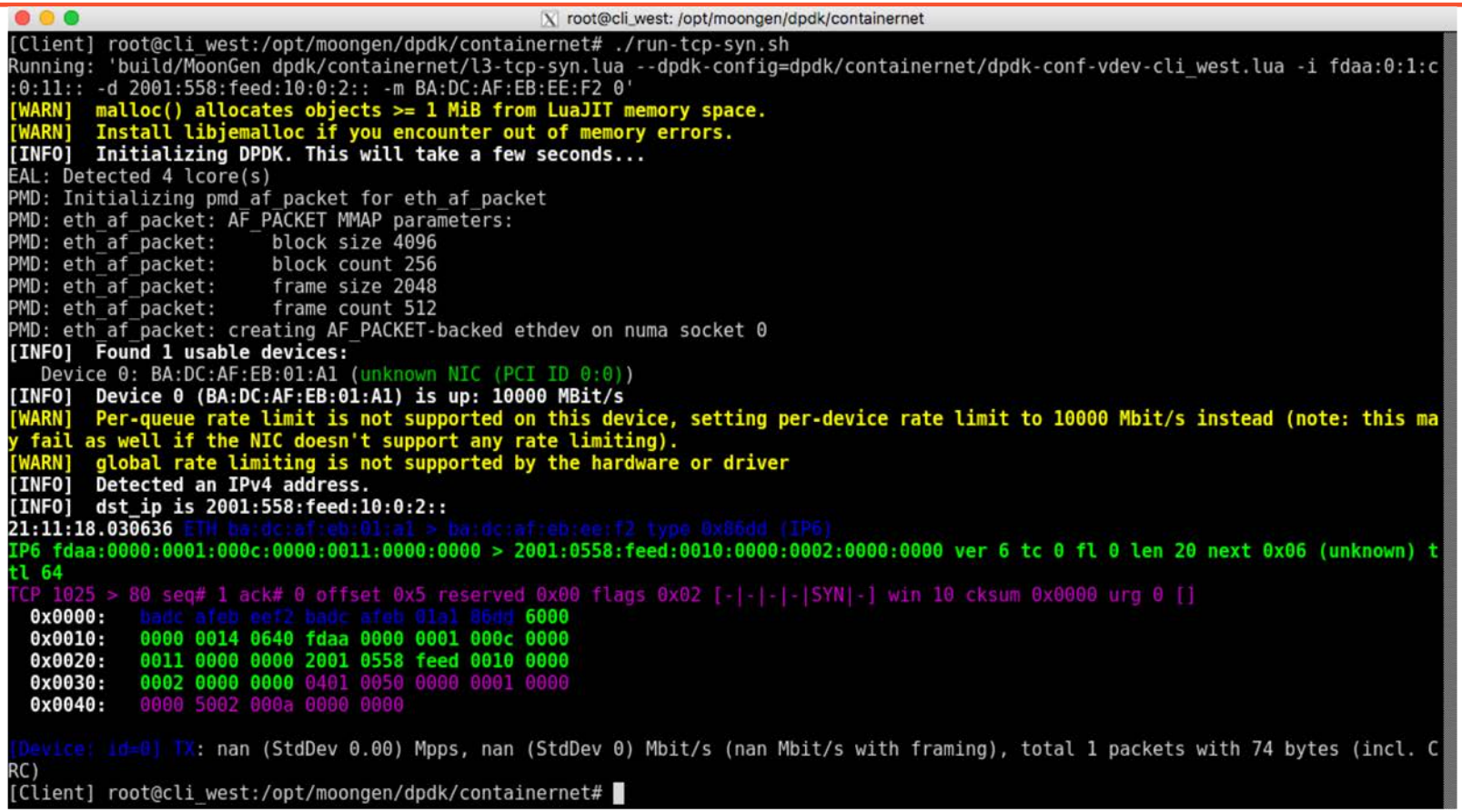
- Shared filesystem
  - Make sure configs have different paths
- TCP Checksum/segmentation offloading
  - Usually performed by hardware NIC
  - *Must* be disabled (in other nodes) for correct checksums
- Access to all hosts and links in the network, most tools
  - Can run wireshark (in the switches), tcpdump, dropwatch

Voilá!


root@og_west: /opt/router
[Router] root@og_west:/opt/router# /opt/router/target/debug/router -f /opt/router/containernet/occam-containernet-og_west
.toml
Going to start DPDK with configuration
Configuration: name: occam-router-og_west mempool size: 512 core cache: 32 primary core: 0
Ports:
Port: dpdk:eth_af_packet,iface=og_west-eth0 RXQ_Count: 1 RX_Queues: [0] TXQ_Count: 1 TX_Queues: [0] RXD: 128
TXD: 128 Loopback: false
Cores:
0
Failed to detect # of NUMA nodes from: /sys/devices/system/node/possible. Assuming a single-node system...
EAL: Detected 4 lcore(s)
EAL: Probing VFIO support...
Running on node 0
Devname: "eth_af_packet,iface=og_west-eth0"
PMD: Initializing pmd_af_packet for eth_af_packet
PMD: eth_af_packet: AF_PACKET MMAP parameters:
PMD: eth_af_packet: block size 4096
PMD: eth_af_packet: block count 256
PMD: eth_af_packet: frame size 2048
PMD: eth_af_packet: frame count 512
PMD: eth_af_packet: creating AF_PACKET-backed ethdev on numa socket 0
Going to try and use port 0
Running on node 0
Receiving started
Running 1 pipelines
Starting scheduler on 0

Voilá!

DPDK
DATA PLANE DEVELOPMENT KIT



Summary

- We wanted to write network functions
  - DPDK, Netbricks, MoonGen
- Environment to develop
  - VMs, Labs, Production networks
- Containernet
  - Scalable
  - Configurable
  - Emulation: run real software stack
  - Access to all hosts in the network

Thank you

https://www.linkedin.com/in/andrewkewang