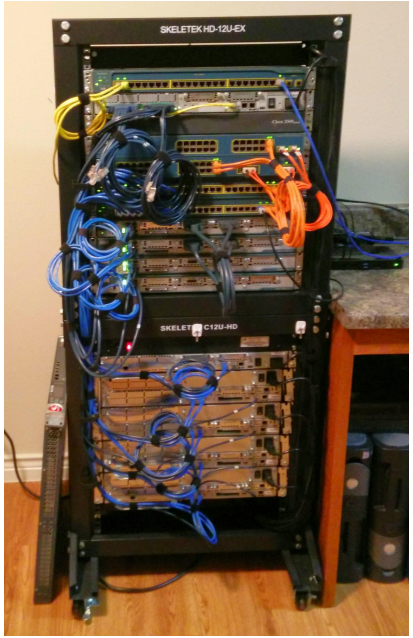
Physical and Virtual Network Lab Integration

The Irony of using SDN and NFV to study legacy network technologies

The Physical Lab

- 4 x lab switches
- 9 x lab routers
- 2 x console routers
- 1 x mgmt switch



The Physical Lab - The Benefits

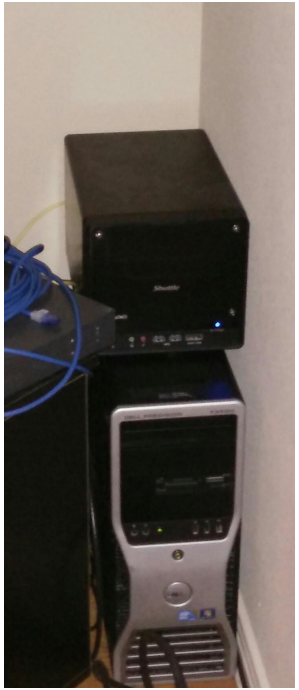
- It’s stable
- It’s relatively cheap
  - Good deals are to be found on Ebay
  - I generally spend <= $50 per device

The Physical Lab - The Drawbacks

- It takes up space
- It generates heat
  - Good in the winter
  - Bad in the summer
- It draws a lot of power
  - Power == money spent
  - I have to take into account what I plug in on the same power circuit. I've tripped 15 amp breakers
- It's noisy
  - I live in a small apartment. I don't really have space to put it in it's own secluded room to help drown out the noise.

The Virtual Lab

- 1 x Intel i5 Quad Core, 16GB/Ram, 128GB/SSD
  - Hypervisor
- 1 x Intel Xeon 8 Core, 12GB/RAM, Raid 1 - 80GB Platter drives
  - Hypervisor
- 1 x Intel Atom Dual Core, 4GB/Ram, 64GB/SSD
  - SDN Controller



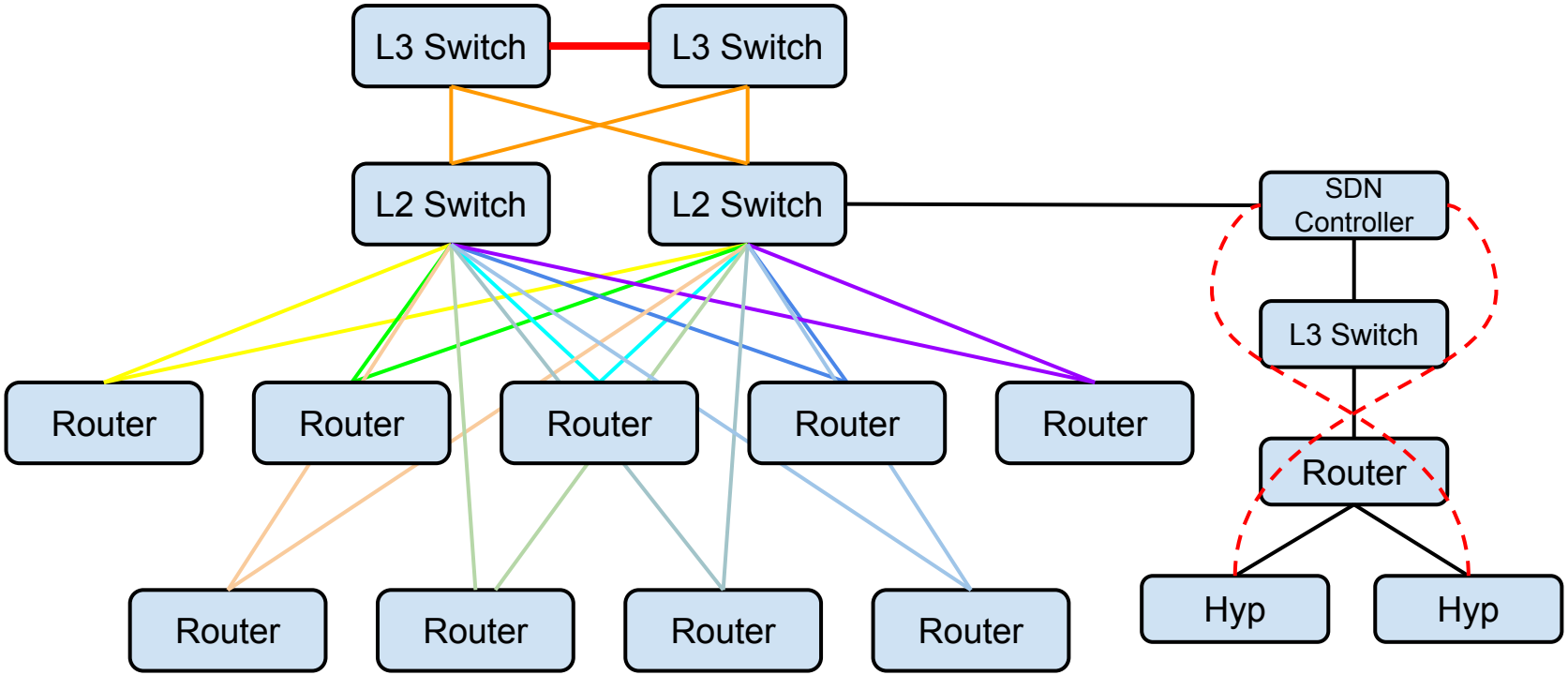
The Virtual Lab - The Benefits

- It’s quiet
- It consumes less power
- It’s multi-functional
  - I can use it to write code, create and deploy docker containers, spin up other virtual machines, play with software
- It can scale up or down based on need
- I’m learning SDN in the process

The Virtual Lab - The Drawbacks

- There is a lot of trial and error
  - I get a lot of unexpected results, thus spend a lot of time troubleshooting and tinkering. Especially when I am attempting to spin up a lab that over-subscribes my hypervisors. It ends up being time taken away from studying, but at that same time it’s a good learning experience.

The Physical Layout of the Lab



OpenVswitch - Setup

On the Hypervisor and SDN Controller - Create OVS Bridge and L2 Tunnels

- Create a bridge
  - *ovs-vsctl add-br ovs-br0*
- Enable STP
  - *ovs-vsctl set Bridge ovs-br0 stp_enable=true*
- Create vxlan tunnel between hypervisor and sdn controller
  - On the hypervisor:
    - *ovs-vsctl add-port ovs-br0 vxlan1*
    - *ovs-vsctl set Interface vxlan1 type=vxlan options:remote_ip=<ip_of_sdn_ctlr>*
  - On the sdn controller:
    - *ovs-vsctl add-port ovs-br0 vxlan1*
    - *ovs-vsctl set Interface vxlan1 type=vxlan options:remote_ip=<ip_addr_of_hyp>*

OpenVswitch - Setup

On the hypervisor and sdn controller

- Enable Openflow Support
  - *ovs-vsctl set-controller ovs-br0 tcp:<ip_of_sdn_controller>:6633*
- Optional - Make Openflow the only control-plane
  - *ovs-vsctl set-fail-mode ovs-br0 secure*

OpenVswitch - Setup

On the SDN Controller

- Associate an unused NIC with the OVS bridge
  - *ovs-vsctl add-port ovs-br0 eth1*
- This NIC will be plugged into a lab switch in the physical lab. The interface on the switch is configured as a trunk

Enable OpenFlow

- I’m using the POX controller
  - It’s quick and easy to set up - Requires Python 2.6+
  - http://www.noxrepo.org/pox/about-pox/
    - *./pox.py forwarding.l2_learning &*

Spin up a virtual router

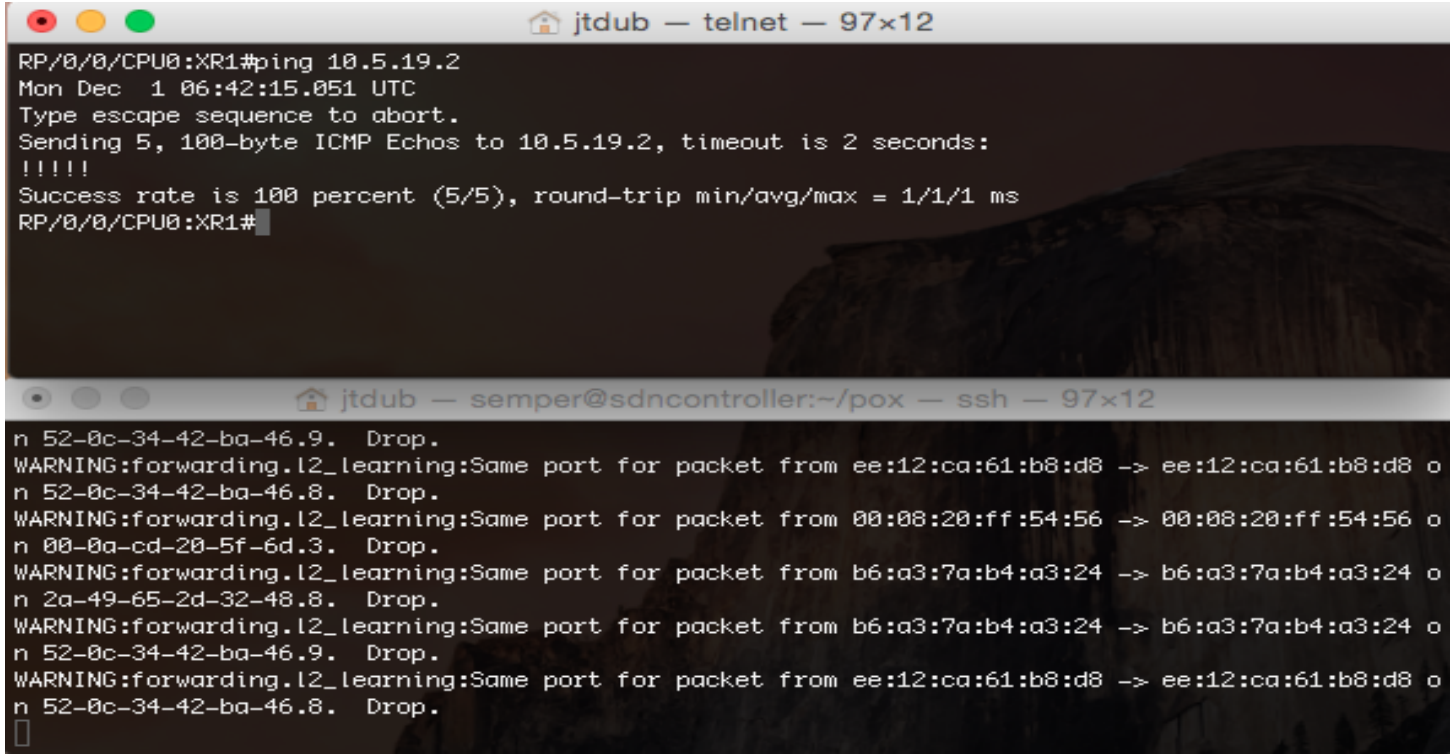
- Spinning up routers in KVM is tedious
  - I wrote a tool to help with this - It's buggy and limited to v2v only, AKA incomplete
  - http://tools.packetgeek.net/vnlcg/
- The future is virtual routers in Docker containers
  - Rackspace currently does this in lab and testing environments
  - Currently Docker networking isn't great, but it will be getting better thanks to a new startup.
    - http://socketplane.io/
    - https://github.com/docker/docker/issues/8951

```
qemu-kvm -cpu kvm64 -nographic -m 3072 -hda /var/lib/libvirt/images/iosxrv-k9-5.1.1-1.img \
-serial telnet::8003,server,nowait \
-net nic,model=virtio,macaddr=4E:33:90:68:3B:C3 \
-net tap,ifname=3xrm0,script=/usr/local/sbin/ovs0-ifup \
-net nic,model=virtio,macaddr=2A:A4:39:07:B3:47 \
-net tap,ifname=xr3g0,script=/usr/local/sbin/ovs1-ifup &
sleep 2
```

Configure a virtual and physical router

- Use subinterfaces with vlan support
  - R5
    - interface fa0/0
    - no shut
    - interface fa0/0.519
    - description R5 x XR1
    - encapsulation dot1q 519
    - ip address 10.5.19.1 255.255.255.0
  - XR1
    - interface gig0/0/0/0
    - no shut
    - interface gig0/0/0/0.519
    - description XR1 x R5
    - encapsulation dot1q 519
    - ipv4 address 10.5.19.2/24
    - commit

Test Connectivity



Questions, Comments, Gripes, Complaints?