CMP3214
Computer
Communication Networks

Lecture 12

Software Defined Networks (SDN)

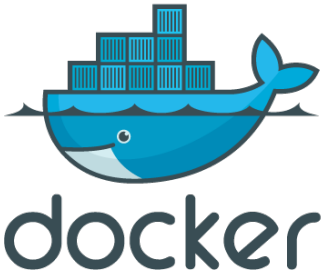
Diarmuid Ó Briain
CEng, FIEI, FIET, CISSP

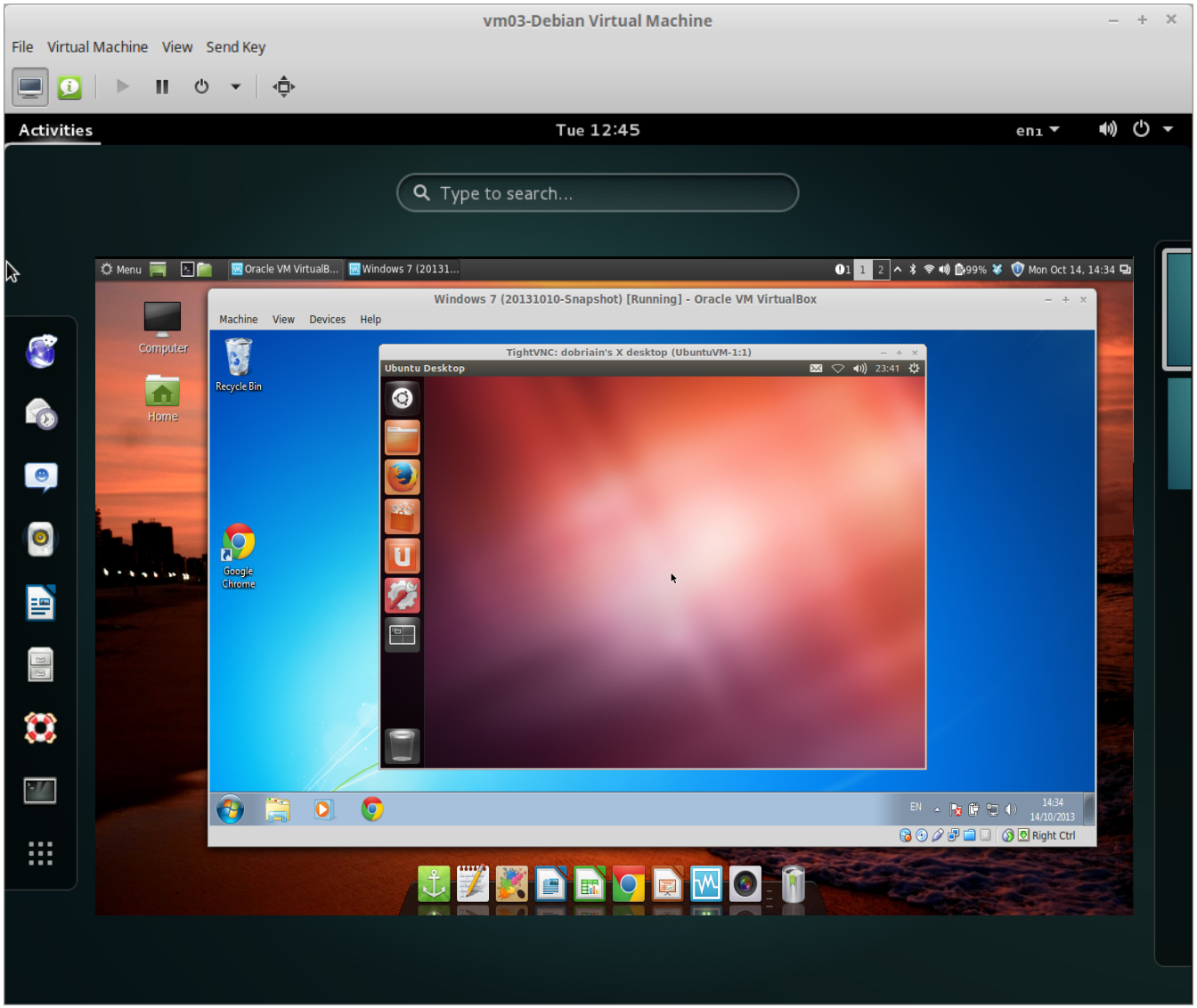
diarmuid@obriain.com

Why the need for change

- Change in today's traffic patterns.
- The consumerism of ICT.
- The rise of cloud services.
- The rise of the Internet of Things (IoT).
- The bandwidth requirements of big data.
- Network complexity discourages change.
- Inability to scale.
- Vendor dependence.

Virtualisation and Containers



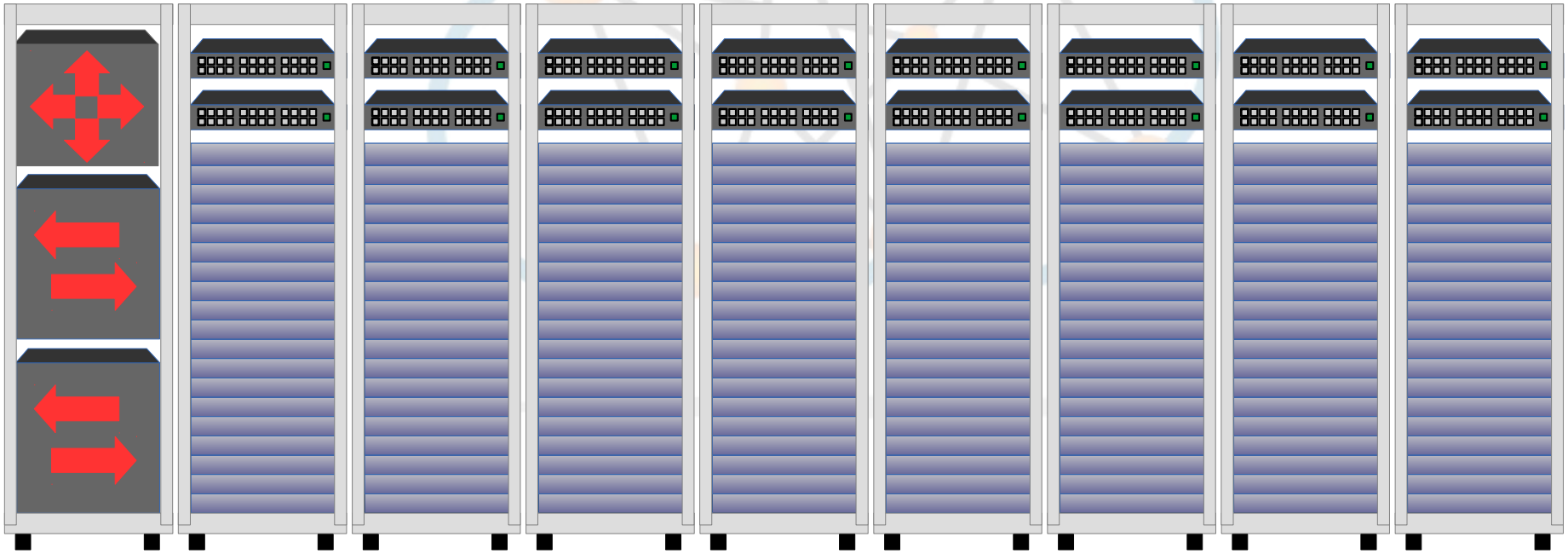


Cloud computing

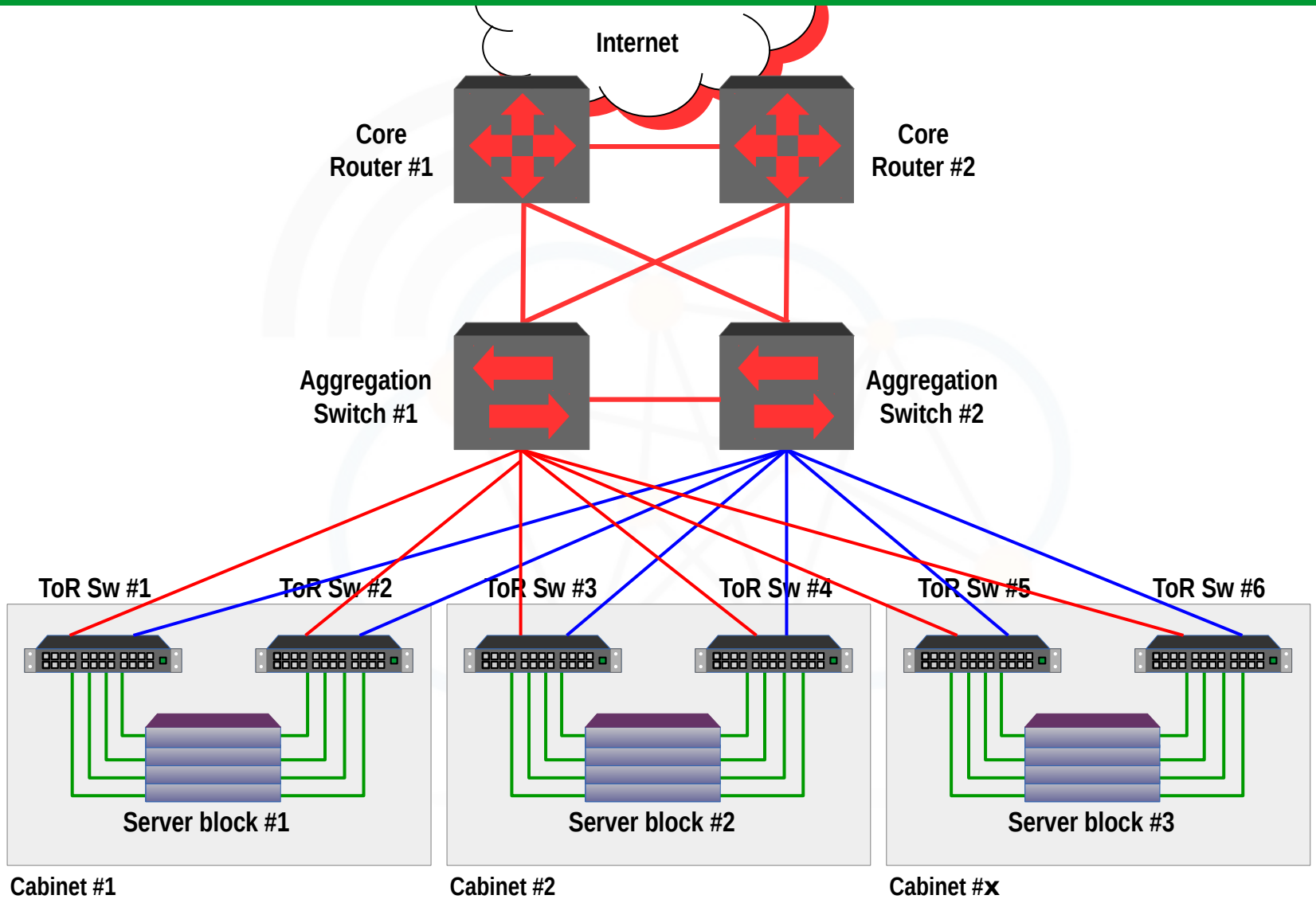
- Elastic Compute
- Elastic MapReduce

Traditional Data Centre layout

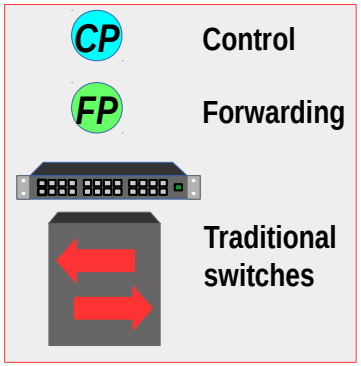
- Racks of servers.
- Two Top of Rack (ToR) switches.
- Aggregation switches.
- Core Routers.



Traditional Data Centre interconnections



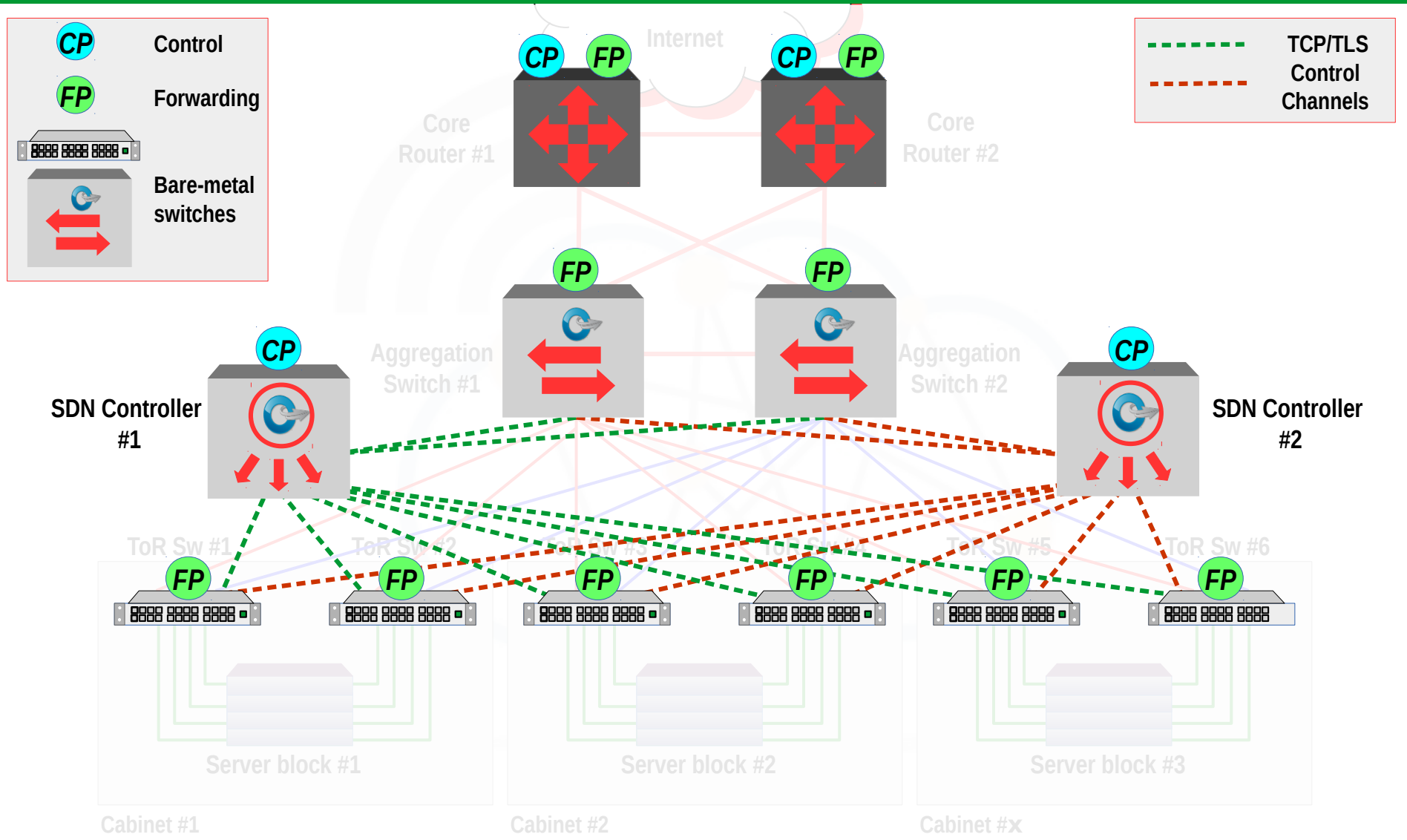
Traditional Control and Forwarding Planes



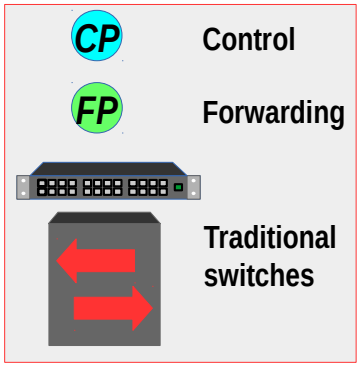
What is Software Defined Networking (SDN)

- SDN is a network architecture that is:
  - Dynamic
  - Manageable
  - Cost-effective
  - Adaptable.
- SDN decouples:
  - Network control
  - Forwarding functions.
- Network control becomes more:
  - Centralised
  - Programmable.

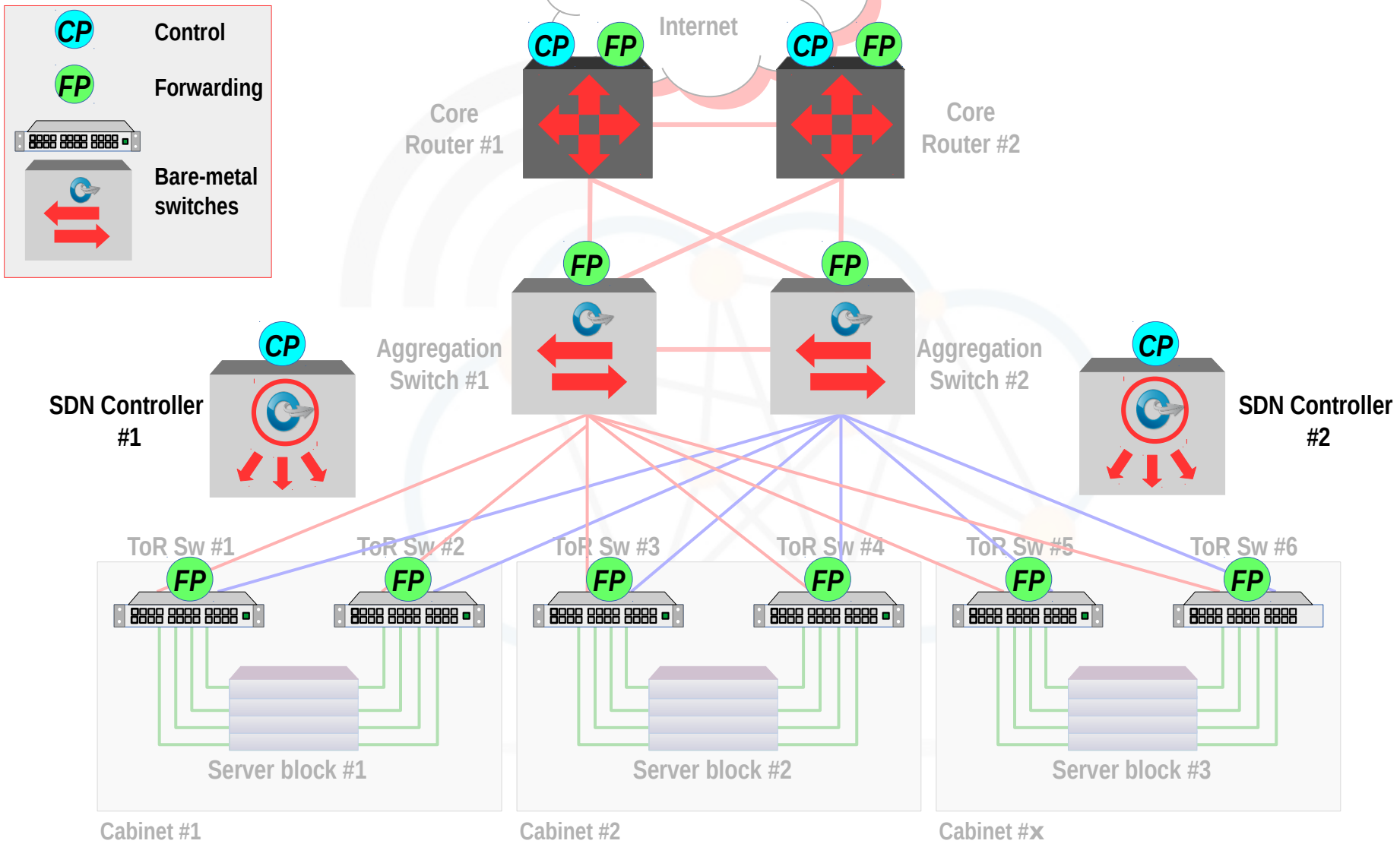
SDN Controller – Device communication



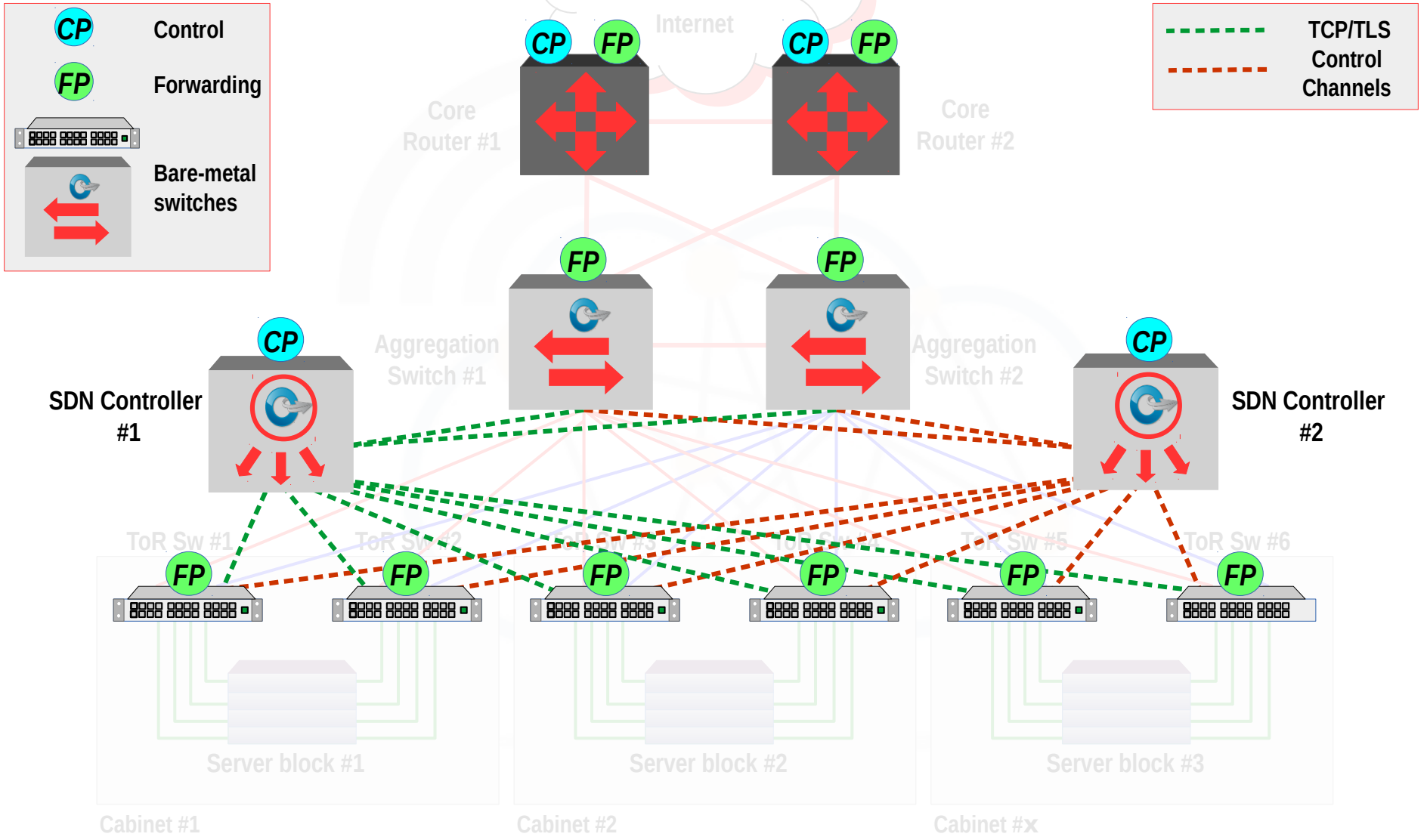
Traditional Control and Forwarding Planes



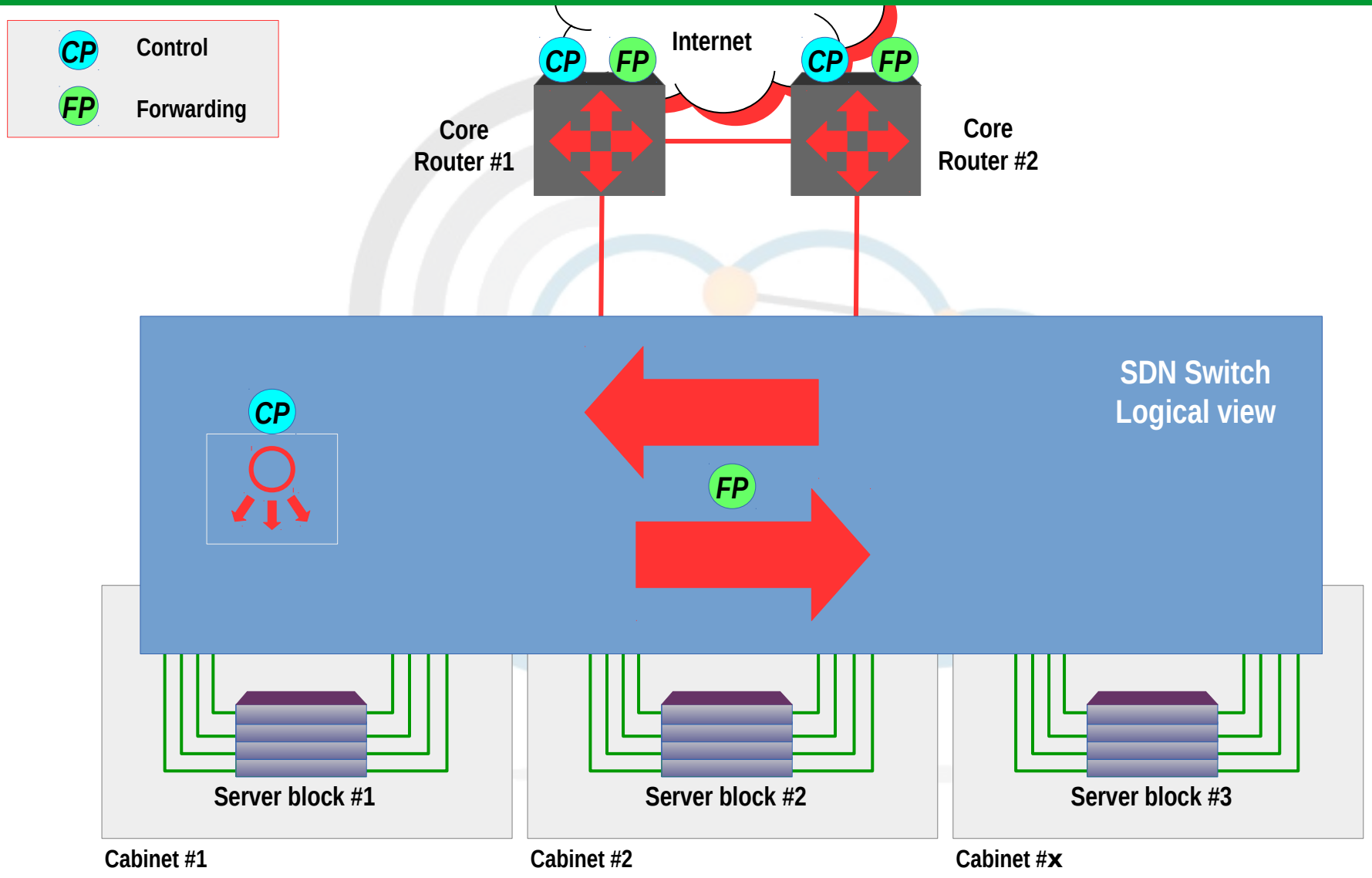
How SDN changes the Data Centre



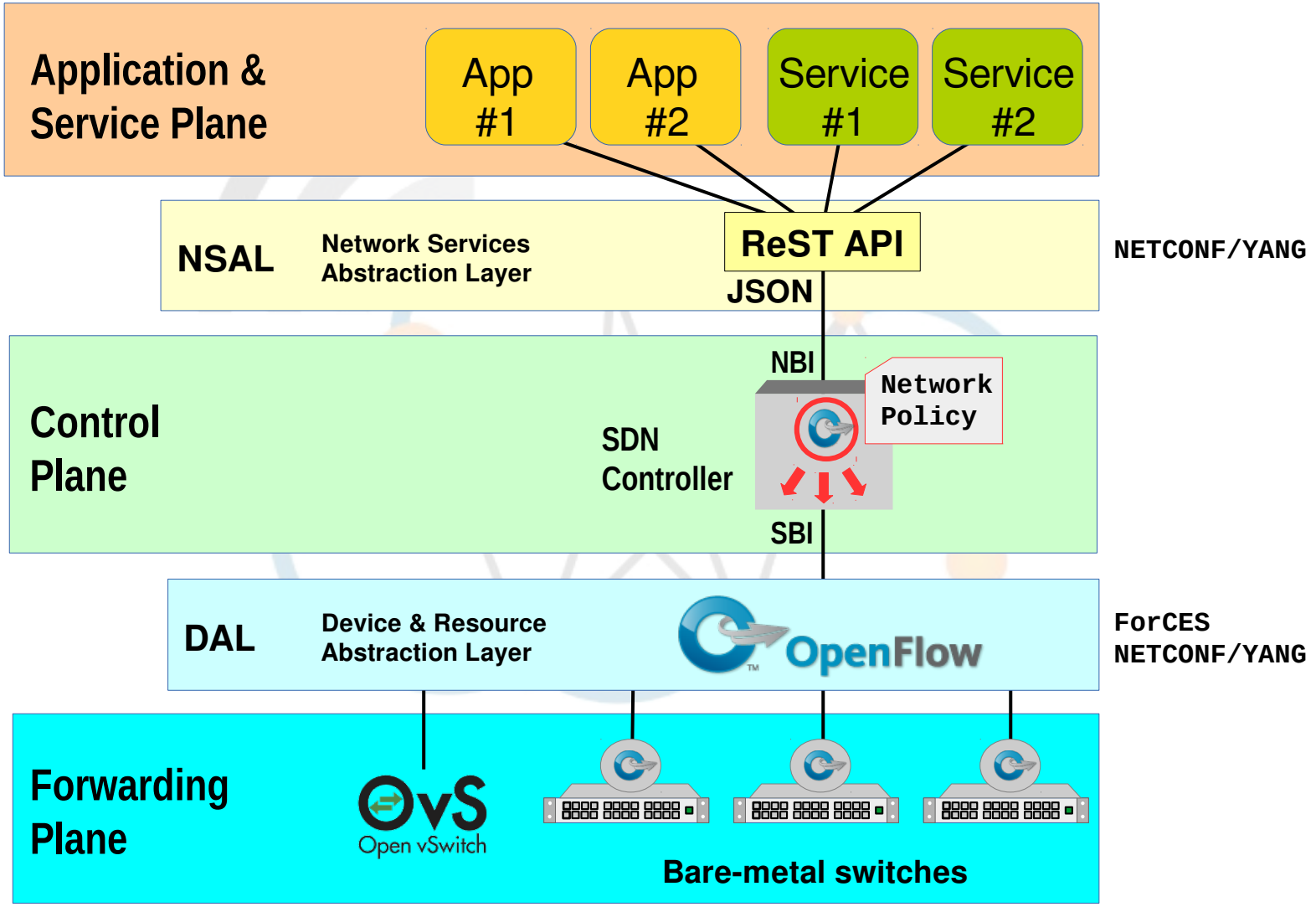
SDN Controller – Device communication



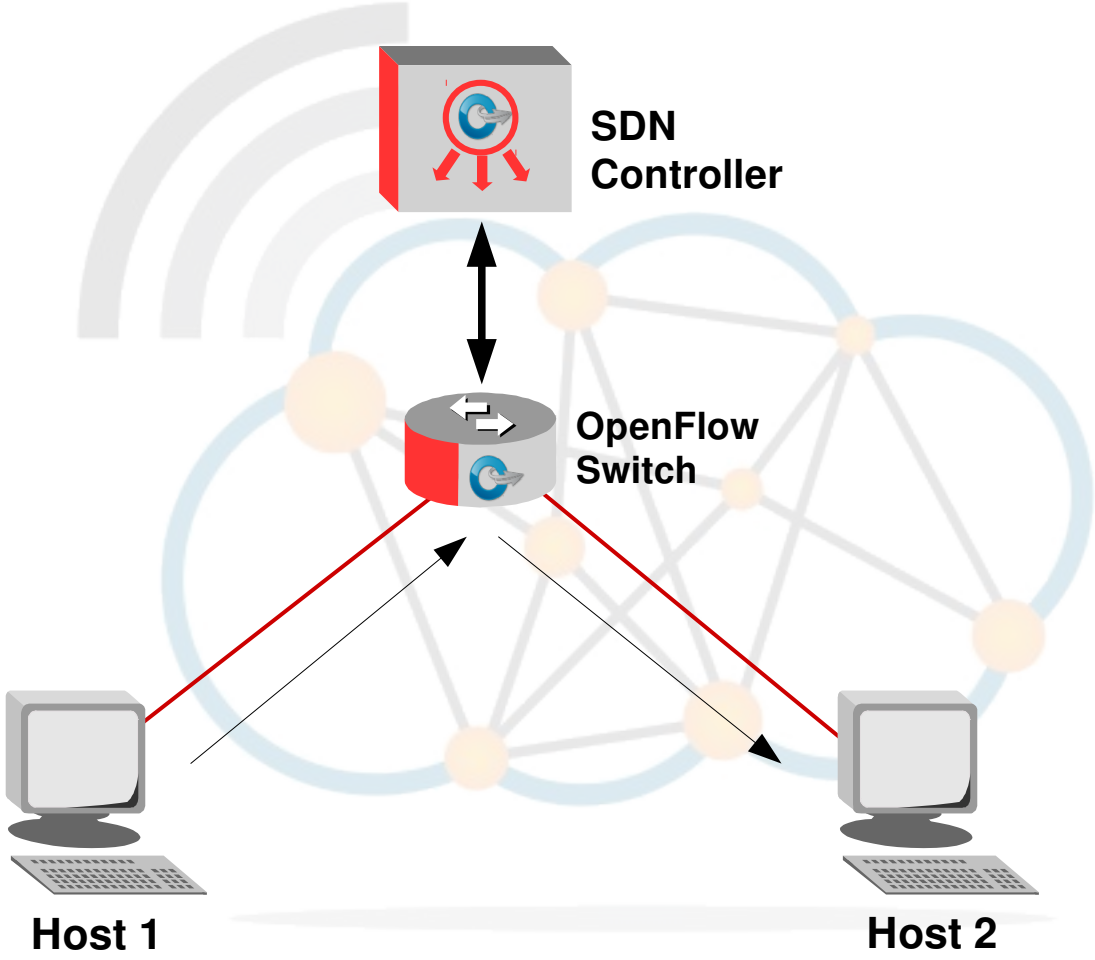
SDN logical view



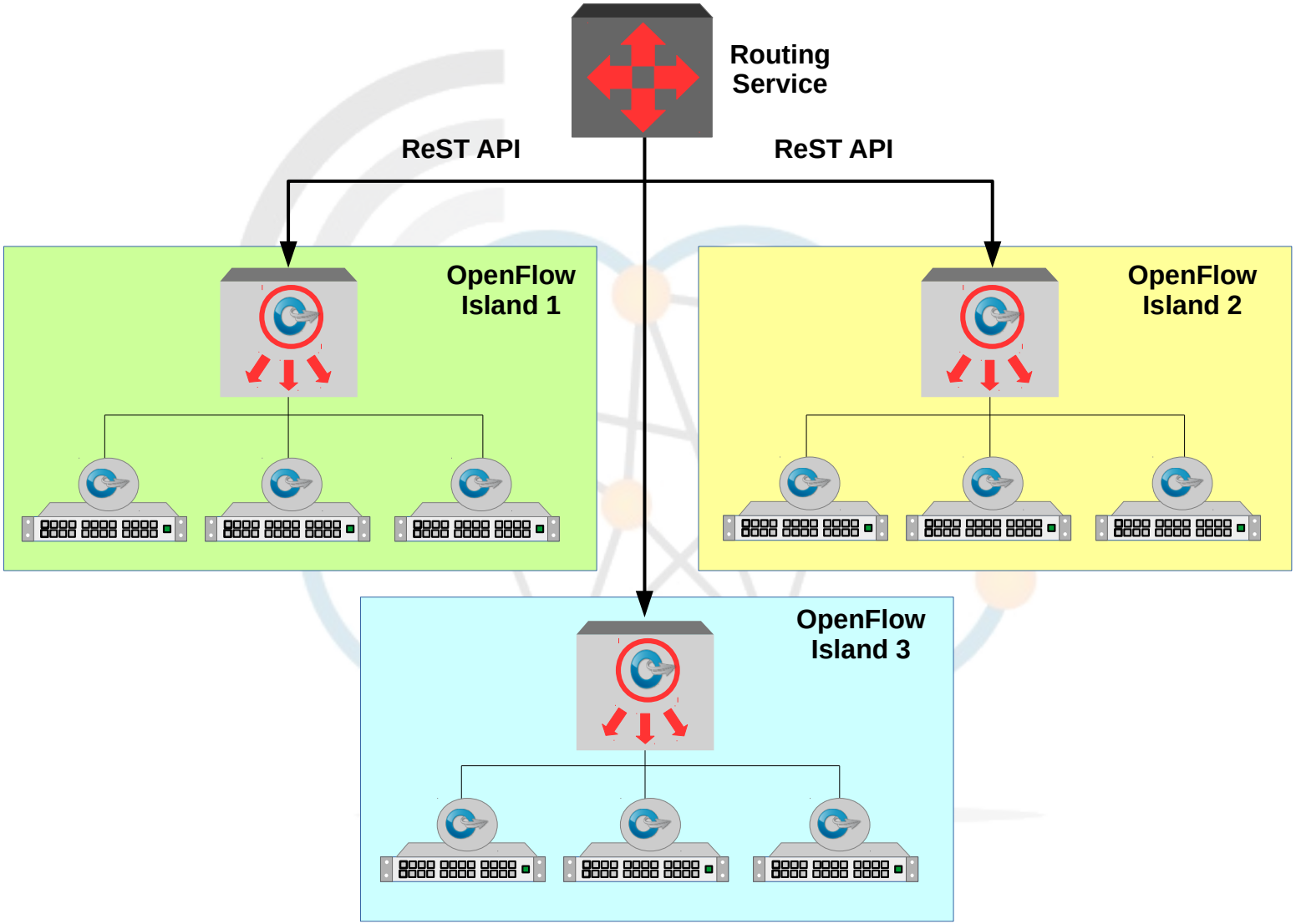
SDN Architecture



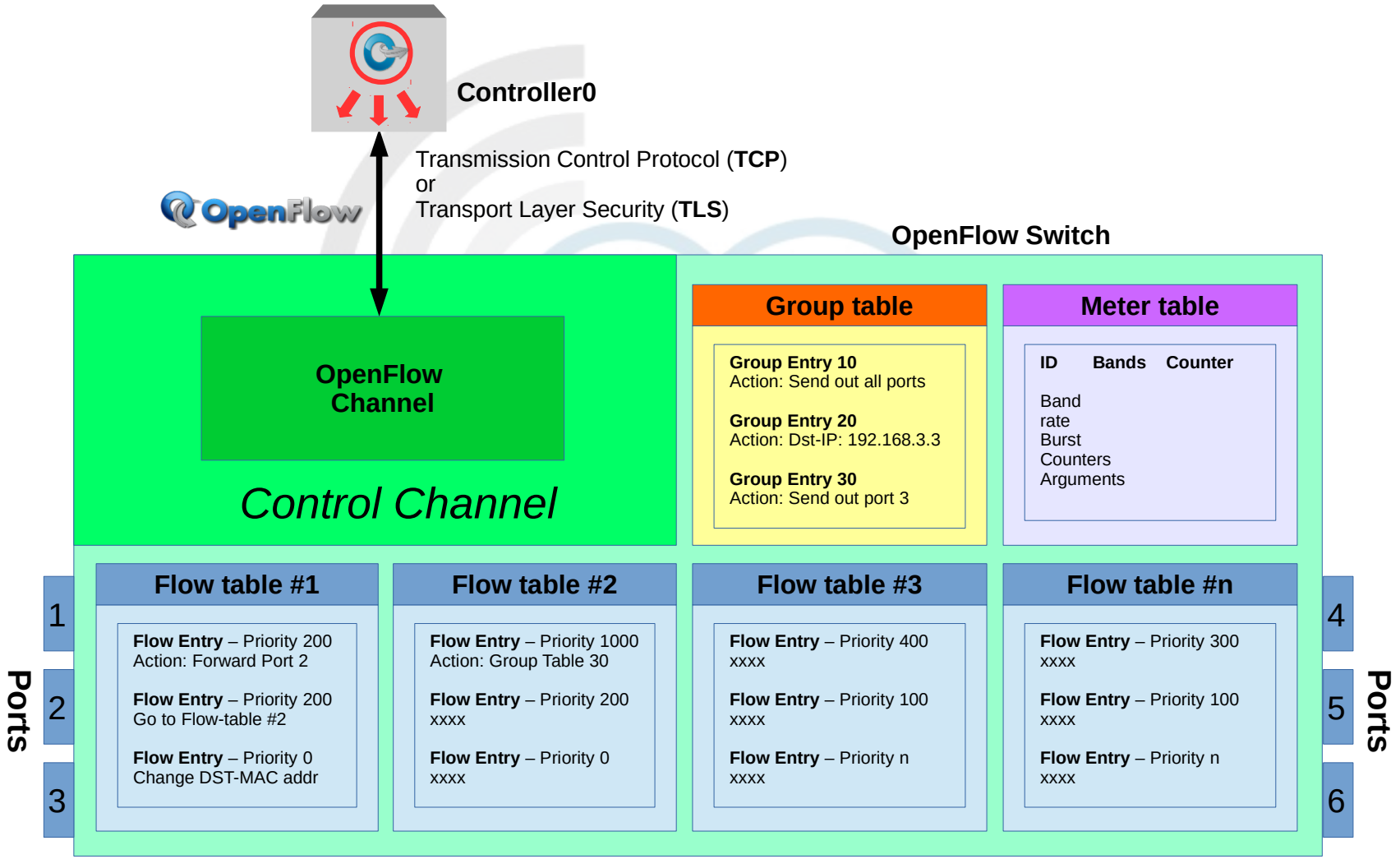
SDN Operation



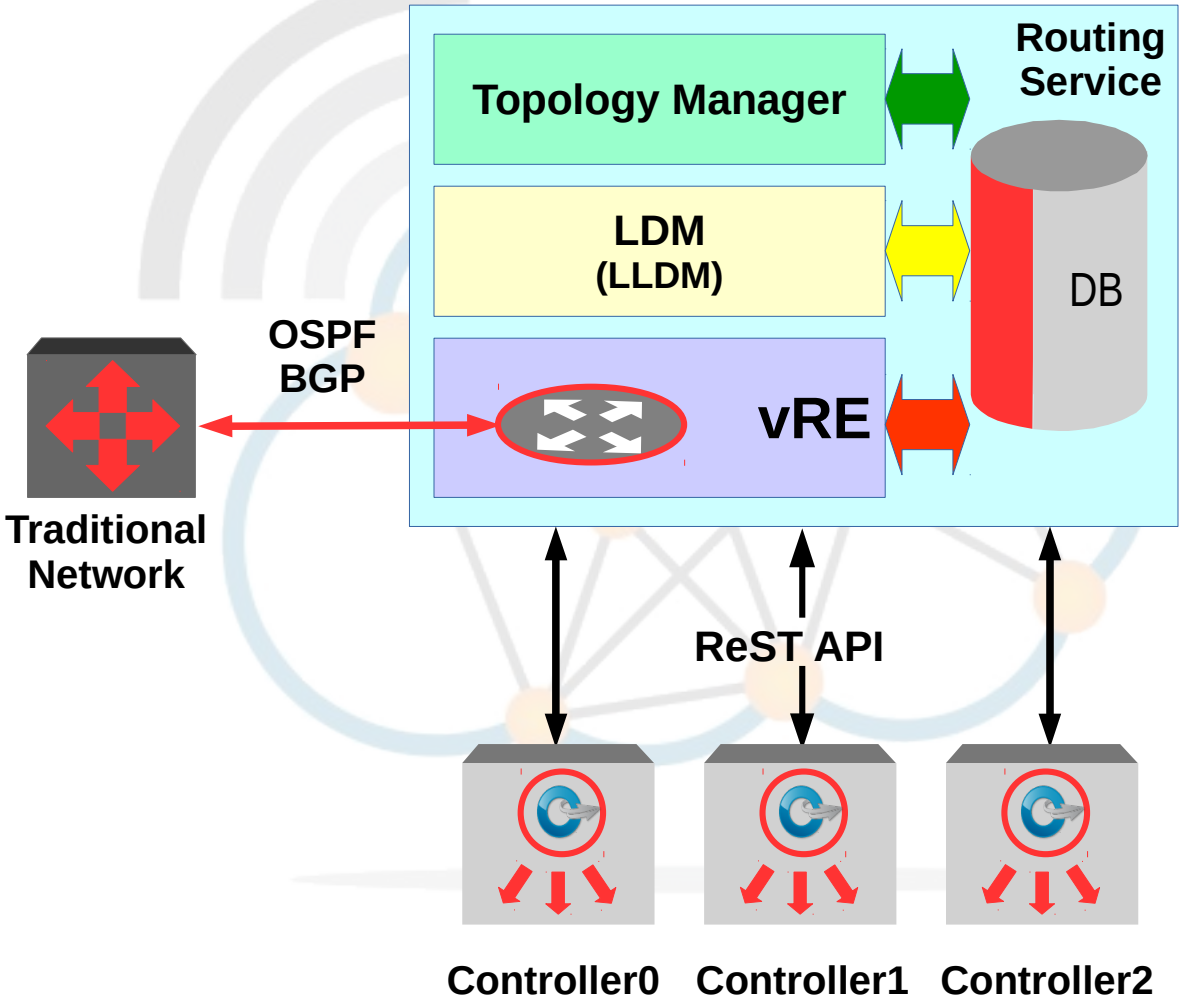
SDN Routing Islands



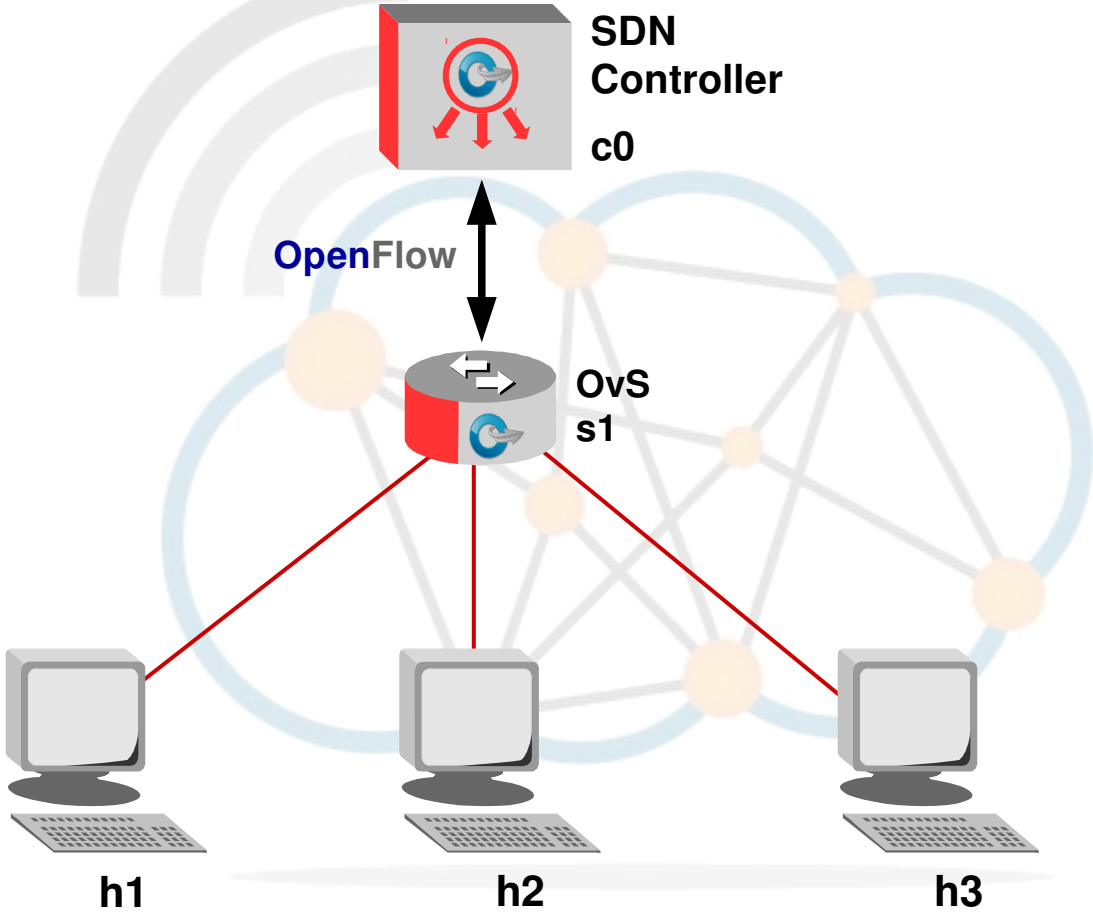
OpenFlow Switch Tables



SDN Routing Service



SDN Operation



Mininet options

- Switch
  - ivs - Indigo Virtual Switch
  - lxbr - Linux Bridge
  - ovs - Open vSwitch
  - ovsbr - Open vSwitch in standalone/bridge mode
  - ovsk - OpenFlow 1.3 switch
  - ovsl - Open vSwitch legacy kernel-space switch using ovs-openflowd
- Controller
  - nox - Nicira Networks OpenFlow controller
  - ovsc - Open vSwitch controller
  - ref - OpenFlow reference controller
  - remote - Controller running outside of mininet
  - ryu - RYU Network Operating System

Mininet options

- topo
  - linear - Linear topology of k switches, with n hosts per switch
  - minimal - Single switch and two hosts
  - reversed - Single switch connected to k hosts, with reversed ports, the lowest-numbered host is connected to the highest-numbered port
  - single - Single switch connected to k hosts
  - torus - 2-D Torus mesh interconnect topology
  - tree - a tree network with a given depth and fanout
- mac - automatically set host MACs

Create mininet network

```
$ sudo mn --topo tree,depth=1,fanout=3 --switch ovsk --controller ref --mac
  *** Creating network
  *** Adding controller
  *** Adding hosts:
  h1 h2 h3
  *** Adding switches:
  s1
  *** Adding links:
  (s1, h1) (s1, h2) (s1, h3)
  *** Configuring hosts
  h1 h2 h3
  *** Starting controller
  c0
  *** Starting 1 switches
  s1
  *** Starting CLI:
```

Review network

```
mininet> hosts
  *** Unknown command: hosts
  mininet> nodes
  available nodes are:
  c0 h1 h2 h3 s1

  mininet> links
  s1-eth1<->h1-eth0 (OK OK)
  s1-eth2<->h2-eth0 (OK OK)
  s1-eth3<->h3-eth0 (OK OK)

  mininet> dump
  <Host h1: h1-eth0:10.0.0.1 pid=7960>
  <Host h2: h2-eth0:10.0.0.2 pid=7963>
  <Host h3: h3-eth0:10.0.0.3 pid=7965>
  <OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None
pid=7970>
  <Controller c0: 127.0.0.1:6633 pid=7952>
```

Review network

```
mininet> h2 ip addr show | grep eth0
  159: h2-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
  group default qlen 1000
      inet 10.0.0.2/8 brd 10.255.255.255 scope global h2-eth0


  mininet> h2 ip route
  10.0.0.0/8 dev h2-eth0  proto kernel  scope link  src 10.0.0.2


  mininet> h1 ping -c1 h3
  PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
  64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=4.66 ms

  --- 10.0.0.3 ping statistics ---
  1 packets transmitted, 1 received, 0% packet loss, time 0ms
  rtt min/avg/max/mdev = 4.664/4.664/4.664/0.000 ms
```

Mininet test options

- build
- pingall - Ping between all hosts
- pingallfull - Ping between all hosts, including times
- pingpair - Ping between first two hosts
- iperf <host1> <host2> - Run TCP iperf between two hosts
- iperfudp <bw i.e. 100M> <host1> <host2> - UDP iperf

Review network

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
c0
```

```
mininet> pingall
*** Ping: testing ping reachability
h1 → h2 h3
h2 → h1 h3
h3 → h1 h2
*** Results: 0% dropped (6/6 received)
```

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['18.0 Gbits/sec', '18.0 Gbits/sec']
```

Mininet individual shells

Exiting mininet

```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 1 switches
s1 ...
*** Stopping 3 links

*** Stopping 3 hosts
h1 h2 h3
*** Done
completed in 3.536 seconds
```

Cleanup mininet

```
$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd
ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core
ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapathsovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
```

Host configuration

- host
  - cfs - Completely Fair Scheduler (CFS)
  - proc
  - rt, - Real Time (RT)
    - cpu=<positive fraction, or -1> - CPU bandwidth limit
- RT_GROUP_SCHED must be enabled in the kernel to change the rt,cpu.

```
$ sudo mn --topo tree,depth=1,fanout=3 --switch ovsk
  --controller ref --mac --host rt,cpu=0.25
```

Link configuration

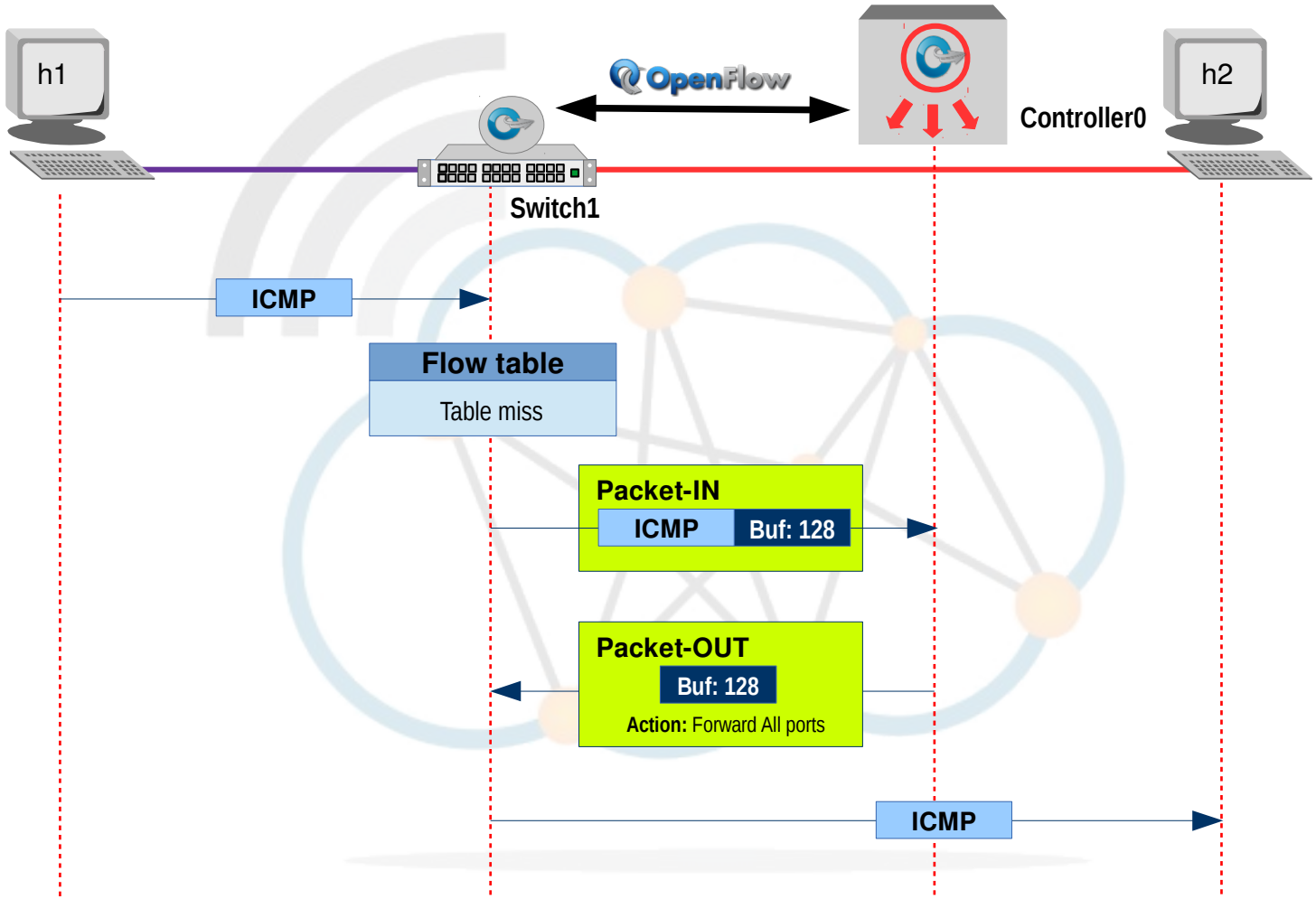
- link tc
  - bw=<value> - Value in Mb/s
  - delay=<value> - Time unit expressed as '5ms', '50us' or '1s'
  - max_queue_size=<x> - Queue size in packets
  - loss=<0 - 100> - Percentage loss
  - use_htb=<True | False> - Hierarchical Token Bucket (HTB) rate limiter

```
$ sudo mn --topo tree,depth=1,fanout=3 --switch ovsk
  --controller ref --mac --link tc,bw=100,delay=20ms
```

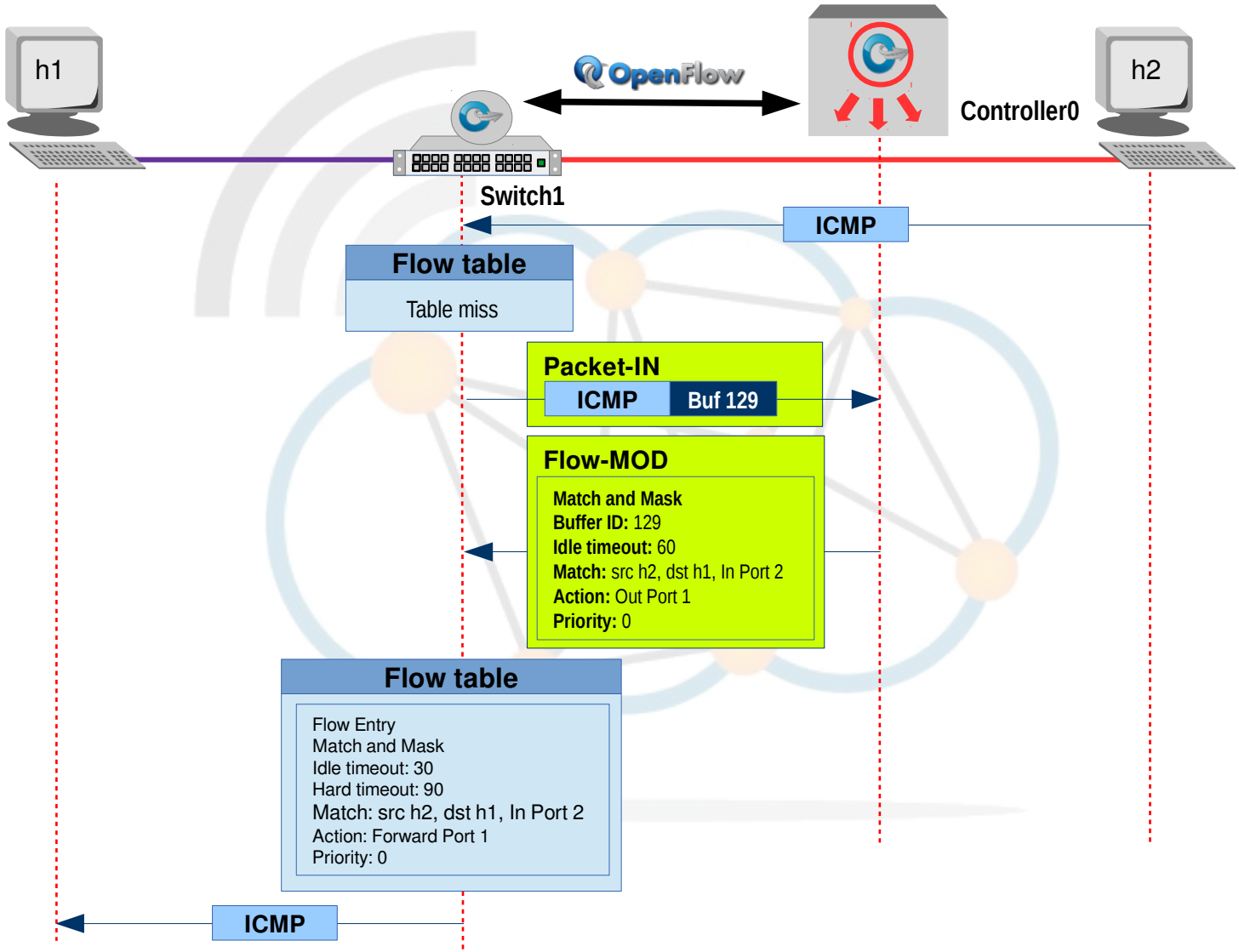
Compare with earlier test

```
*** Results: ['18.0 Gbits/sec', '18.0 Gbits/sec'].


mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['85.7 Mbits/sec', '99.3 Mbits/sec']
```

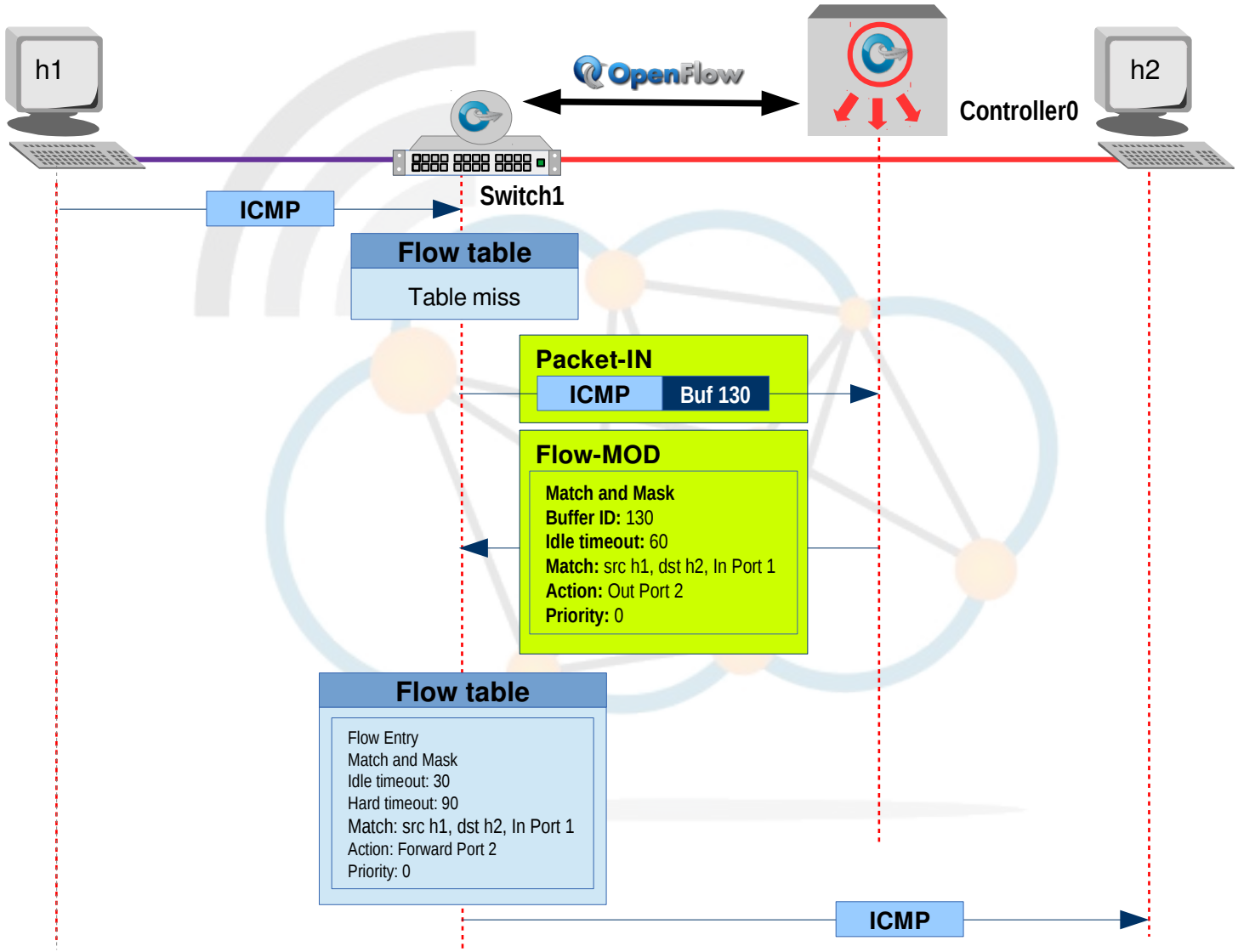
SDN Action


h1
OpenFlow
Controller0
h2
Switch1
ICMP
Flow table
Table miss
Packet-IN
ICMP
Buf: 128
Packet-OUT
Buf: 128
Action: Forward All ports
ICMP

SDN Flow-MOD


h1
OpenFlow
Controller0
h2
Switch1
ICMP
Flow table
Table miss
Packet-IN
ICMP
Buf 129
Flow-MOD
Match and Mask
Buffer ID: 129
Idle timeout: 60
Match: src h2, dst h1, In Port 2
Action: Out Port 1
Priority: 0
Flow table
Flow Entry
Match and Mask
Idle timeout: 30
Hard timeout: 90
Match: src h2, dst h1, In Port 2
Action: Forward Port 1
Priority: 0
ICMP

SDN Flow-MOD


h1
OpenFlow
Controller0
h2
Switch1
ICMP
Flow table
Table miss
Packet-IN
ICMP
Buf 130
Flow-MOD
Match and Mask
Buffer ID: 130
Idle timeout: 60
Match: src h1, dst h2, In Port 1
Action: Out Port 2
Priority: 0
Flow table
Flow Entry
Match and Mask
Idle timeout: 30
Hard timeout: 90
Match: src h1, dst h2, In Port 1
Action: Forward Port 2
Priority: 0
ICMP

Webserver test

```
mininet> h1 ip addr | grep "inet.*eth0"
    inet 10.0.0.1/8 brd 10.255.255.255 scope global h1-eth0

mininet> h3 ip addr | grep "inet.*eth0"
    inet 10.0.0.3/8 brd 10.255.255.255 scope global h3-eth0

mininet> h1 ping -c1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=4.46 ms

mininet> xterm h1

mininet> xtern h3
```

Webserver test

h1

```
root@mininet-vm:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 …
```

h3

```
root@mininet-vm:~# lynx 10.0.0.1
```

```
                                   Directory listing for / (p1 of 2)
Directory listing for /
    _____________________________________________________________________


     * .bash_history
     * .bash_logout
     * .bashrc
     * .cache/
     * .gitconfig
     * .mininet_history
     * .profile
     * .rnd
     * .ssh/
     * .wireshark/
     * .Xauthority
     * install-mininet-vm.sh
     * loxigen/
     * mininet/
     * oflops/
     * oftest/
     * openflow/
-- press space for next page --
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

Webserver test

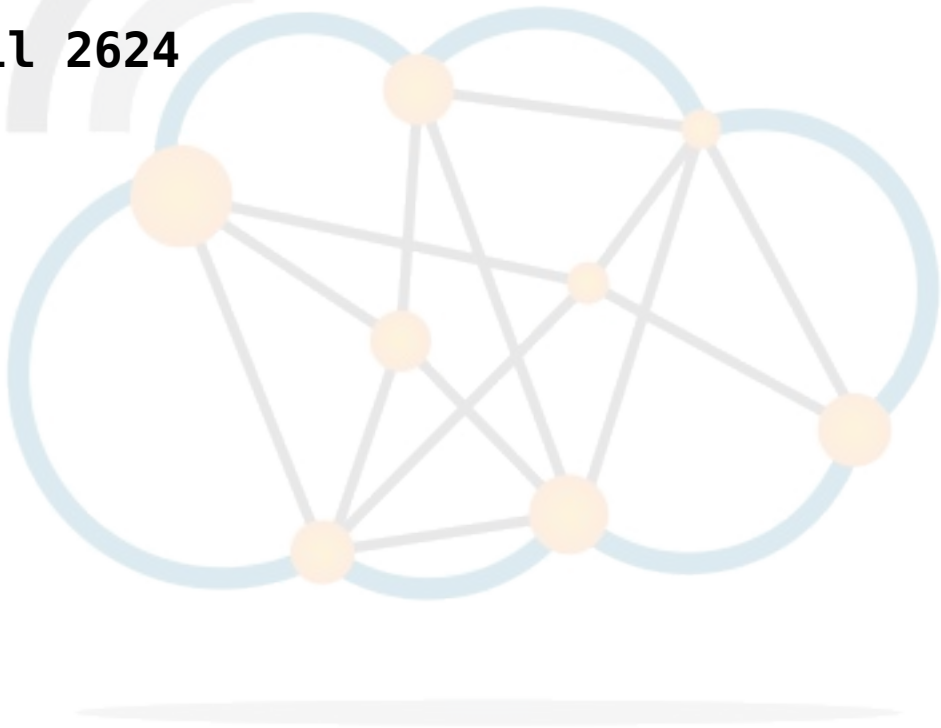
h1

```
10.0.0.3 - - [18/Mar/2015 01:11:34] "GET / HTTP/1.0" 200 -…
```

Kill webserver on h1

```
mininet> h1 ps -ef | grep SimpleHTTPServer
root      2624  1406  0 01:17 pts/4    00:00:00 python -m SimpleHTTPServer 80
root      2669  1406  0 01:19 pts/4    00:00:00 grep SimpleHTTPServer
```

```
mininet> h1 kill 2624
```



What happened ?

From → to	Ver	Len	Type	BufID	Reason/Action
S1 → C0	OF 1.0	158	OFPT_PACKET_IN	342	Reason: OFPR_NO_MATCH 10.0.0.3 → 10.0.0.1 TCP SYN 39109 → 80 In pt: 3
C0 → S1	OF 1.0	90	OFPT_PACKET_OUT	342	Action: OFPAT_OUTPUT In pt: 3 Out pt: 65531
S1 → C0	OF 1.0	158	OFPT_PACKET_IN	343	Reason: OFPR_NO_MATCH 10.0.0.1 → 10.0.0.3 TCP SYN, ACK 80 → 39109 In pt: 1
C0 → S1	OF 1.0	146	OFPT_FLOW_MOD	343	Action: OFPR_FLOW_MOD 10.0.0.1 → 10.0.0.3 TCP SRC: 80 TCP DST: 39109 In pt: 1 Out pt:3
S1 → C0	OF 1.0	150	OFPT_PACKET_IN	344	Reason: OFPR_NO_MATCH 10.0.0.3 → 10.0.0.1 TCP ACK 39109 → 80 In pt: 3
C0 → S1	OF 1.0	146	OFPT_FLOW_MOD	344	Action: OFPR_FLOW_MOD 10.0.0.3 → 10.0.0.1 TCP SRC: 39109 TCP DST: 80 In pt: 3 Out pt:1

Custom topologies

/home/mininet/mininet/examples

/home/mininet/mininet/custom

```
$ cd /home/mininet/mininet/custom
/home/mininet/mininet/custom$ ls
README  topo-2sw-2host.py

$ cat README
This directory should hold configuration files for custom mininets.

See custom_example.py, which loads the default minimal topology. The advantage
of defining a mininet in a separate file is that you then use the --custom
option in mn to run the CLI or specific tests with it.

To start up a mininet with the provided custom topology, do:
  sudo mn --custom custom_example.py --topo mytopo
```

Example custom topologies

/home/mininet/mininet/custom$ **cat topo-2sw-2host.py**

```
from mininet.topo import Topo

class MyTopo( Topo ):

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch, bw=50, delay='3ms', loss=10 )
        self.addLink( leftSwitch, rightSwitch, bw=100, delay='1ms' )
        self.addLink( rightSwitch, rightHost, bw=50, delay='3ms', loss=10 )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Run custom topologies

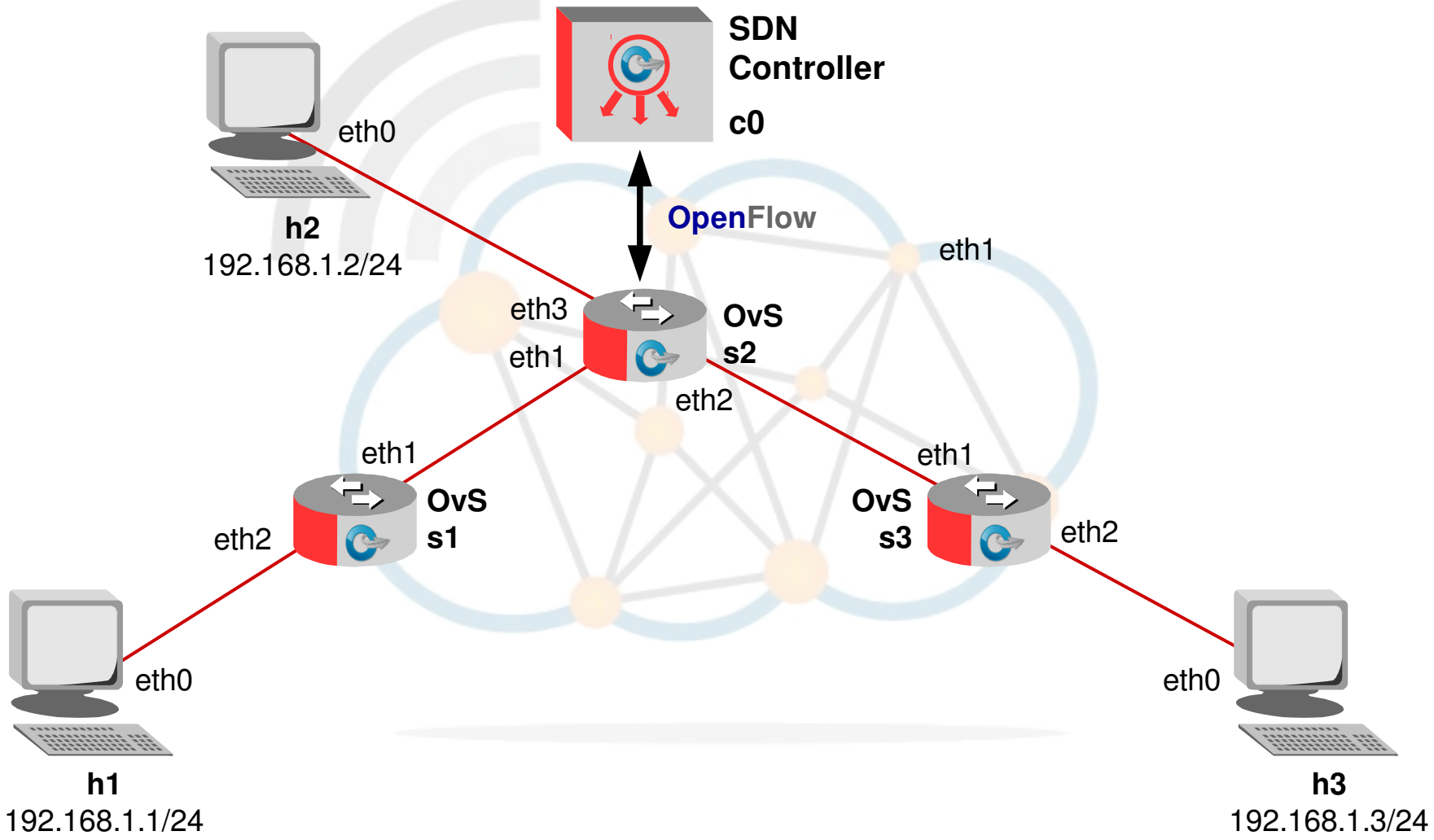
```
$ sudo mn --custom topo-2sw-2host.py --topo mytopo

*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3 s4
*** Adding links:
(30.00Mbit 3ms delay 10% loss) (30.00Mbit 3ms delay 10% loss) (h1, s3)
(100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (s3, s4) (50.00Mbit 3ms
delay 10% loss) (50.00Mbit 3ms delay 10% loss) (s4, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s3 s4
*** Starting CLI:
```

Run custom topologies

```
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s4-eth2
s3 lo:  s3-eth1:h1-eth0 s3-eth2:s4-eth1
s4 lo:  s4-eth1:s3-eth2 s4-eth2:h2-eth0
C0
```

```
mininet> iperfudp 100M h1 h2
*** Iperf: testing UDP bandwidth between h1 and h2
*** Results: ['100M', '26.2 Mbits/sec', '26.2 Mbits/sec']
```

Custom topology example



Review custom topology code

```
----------------Custom -OvS.py---------------------

#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink
from mininet.topo import Topo

def customNet():

    "Create a customNet and add devices to it."

    net = Mininet( controller=Controller )

    # Add controller
    info( 'Adding controller\n' )
    net.addController ('c0')
```

Review custom topology code

```
# Add hosts
info( 'Adding hosts\n' )
h1 = net.addHost( 'h1' )
h2 = net.addHost( 'h2' )
h3 = net.addHost( 'h3' )

# Add switches
info( 'Adding switches\n' )
s1 = net.addSwitch( 's1' )
s2 = net.addSwitch( 's2' )
s3 = net.addSwitch( 's3' )

# Add links
info( 'Adding switch links\n' )
net.addLink( s1, s2, bw=1000, delay='1ms' )
net.addLink( s2, s3, bw=1000, delay='1ms' )
```

Review custom topology code

```
    info( 'Adding host links\n' )
    net.addLink( h1, s1, bw=50, delay='3ms' )
    net.addLink( h2, s2, bw=50, delay='2ms' )
    net.addLink( h3, s3, bw=50, delay='2ms', loss=15 )

    info( '*** Starting network\n')
    net.start()

    info( '*** Running CLI\n' )
    CLI( net )

    info( '*** Stopping network' )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    customNet()
```

Start custom topology

```
/home/mininet/mininet/custom$ sudo ./custom-OvS.py
  Adding controller
  Adding hosts
  Adding switches
  Adding switch links
  (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit
  1ms delay) (1000.00Mbit 1ms delay) Adding host links
  (50.00Mbit 3ms delay) (50.00Mbit 3ms delay) (50.00Mbit 2ms
  delay) (50.00Mbit 2ms delay) (50.00Mbit 2ms delay 15% loss)
  (50.00Mbit 2ms delay 15% loss)
  Adding host links
  *** Starting network
  *** Configuring hosts
  h1 h2 h3
```

Start custom topology

```
*** Starting controller
c0
*** Starting 3 switches
s1 (1000.00Mbit 1ms delay) (50.00Mbit 3ms delay) s2
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (50.00Mbit
2ms delay) s3 (1000.00Mbit 1ms delay) (50.00Mbit 2ms delay
15% loss) ...(1000.00Mbit 1ms delay) (50.00Mbit 3ms delay)
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (50.00Mbit
2ms delay) (1000.00Mbit 1ms delay) (50.00Mbit 2ms delay 15%
loss)
*** Running CLI
*** Starting CLI:
mininet>
```

Review custom topology

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=10517>
<Host h2: h2-eth0:10.0.0.2 pid=10519>
<Host h3: h3-eth0:10.0.0.3 pid=10521>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=10526>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=10529>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=10532>
<Controller c0: 127.0.0.1:6653 pid=10510>

mininet> net
h1 h1-eth0:s1-eth2
h2 h2-eth0:s2-eth3
h3 h3-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth1 s1-eth2:h1-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:h2-eth0
s3 lo:  s3-eth1:s2-eth2 s3-eth2:h3-eth0
c0
```

Review custom topology

```
mininet> pingall
  *** Ping: testing ping reachability
  h1 -> h2 h3
  h2 -> h1 h3
  h3 -> h1 h2
  *** Results: 0% dropped (6/6 received)

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> X h2
*** Results: 16% dropped (5/6 received)
```

Review custom topology

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['43.3 Mbits/sec', '47.7 Mbits/sec']


mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['179 Kbits/sec', '193 Kbits/sec']


mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['318 Kbits/sec', '320 Kbits/sec']
```

Review custom topology flows before ping

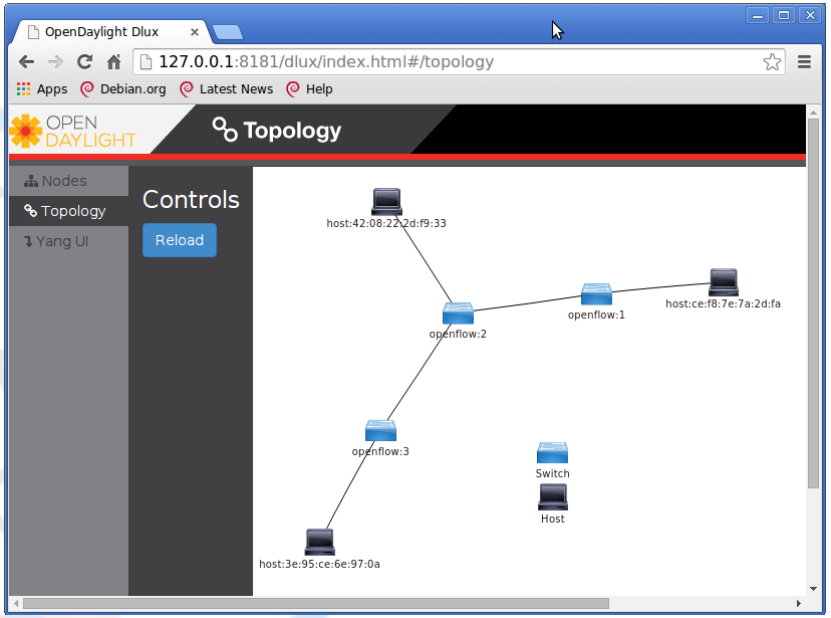
```
mininet> dpctl dump-flows
*** s1
------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s2
------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s3
------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):


mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.81 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2.82 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.595 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.595/2.078/2.826/1.048 ms
```

Review custom topology flows after ping

```
mininet> dpctl dump-flows
*** s1
------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=3.854s, table=0, n_packets=3, n_bytes=294,
idle_timeout=60,idle_age=1, priority=65535,icmp,in_port=1,
vlan_tci=0x0000,dl_src=b2:a6:82:06:f5:0b,dl_dst=da:f0:ca:b8:ca:79,
nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0
actions=output:2 cookie=0x0, duration=2.856s, table=0, n_packets=2,
n_bytes=196, idle_timeout=60, idle_age=1,  priority=65535,icmp,
in_port=2,vlan_tci=0x0000,dl_src=da:f0:ca:b8:ca:79,
dl_dst=b2:a6:82:06:f5:0b,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,
icmp_type=8,icmp_code=0 actions=output:1

*** s2   ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=2.859s, table=0, n_packets=2, n_bytes=196,
idle_timeout=60, idle_age=1, priority=65535,icmp,in_port=1,
vlan_tci=0x0000,dl_src=da:f0:ca:b8:ca:79,dl_dst=b2:a6:82:06:f5:0b,
nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,icmp_code=0
actions=output:3 cookie=0x0, duration=3.86s, table=0, n_packets=3,
n_bytes=294, idle_timeout=60, idle_age=1, priority=65535,icmp,
in_port=3,vlan_tci=0x0000,dl_src=b2:a6:82:06:f5:0b,
dl_dst=da:f0:ca:b8:ca:79,nw_src=10.0.0.2,nw_dst=10.0.0.1,
nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
*** s3 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------
```

Opensource SDN Controllers

- Network Operating System (NOX)
- POX
  - Python version of NOX
- Big Switch networks
  - Project Floodlight
- Linux Foundation
  - Open Daylight (ODL)



POX

Floodlight

OPENDAYLIGHT

POX

```
$ ~/pox/pox.py forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.

INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-03 2] connected
INFO:openflow.of_01:[00-00-00-00-00-01 5] connected
INFO:openflow.of_01:[00-00-00-00-00-04 3] connected
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
```

Testing POX

```
$ sudo mn --topo tree,depth=2,fanout=3 --switch ovsk
--controller=remote,ip=127.0.0.1,port=6633 --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s2, h1) (s2, h2) (s2, h3) (s3, h4) (s3, h5) (s3, h6)
(s4, h7) (s4, h8) (s4, h9)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9
h2 -> h1 h3 h4 h5 h6 h7 h8 h9
h3 -> h1 h2 h4 h5 h6 h7 h8 h9
h4 -> h1 h2 h3 h5 h6 h7 h8 h9
h5 -> h1 h2 h3 h4 h6 h7 h8 h9
h6 -> h1 h2 h3 h4 h5 h7 h8 h9
h7 -> h1 h2 h3 h4 h5 h6 h8 h9
h8 -> h1 h2 h3 h4 h5 h6 h7 h9
h9 -> h1 h2 h3 h4 h5 h6 h7 h8
*** Results: 0% dropped (72/72 received)
mininet>
```

Project Floodlight

```
$ cd floodlight/
~/floodlight$ java -jar ./target/floodlight.jar
05:42:25.822 INFO [n.f.c.m.FloodlightModuleLoader:main] Loading modules from
src/main/resources/floodlightdefault.properties
05:42:26.550 WARN [n.f.r.RestApiServer:main] HTTPS disabled; HTTPS will not be
used to connect to the REST API.
05:42:26.550 WARN [n.f.r.RestApiServer:main] HTTP enabled; Allowing unsecure
access to REST API on port 8080.
05:42:29.684 WARN [n.f.c.i.OFSwitchManager:main] SSL disabled. Using unsecure
connections between Floodlight and switches.
05:42:29.684 INFO [n.f.c.i.OFSwitchManager:main] Clear switch flow tables on
initial handshake as master: TRUE
05:42:29.685 INFO [n.f.c.i.OFSwitchManager:main] Clear switch flow tables on
each transition to master: TRUE
05:42:29.685 INFO [n.f.c.i.OFSwitchManager:main] Setting 0x1 as the default max
tables to receive table-miss flow
05:42:29.700 INFO [n.f.c.i.OFSwitchManager:main] Setting max tables to receive
table-miss flow to 0x1 for DPID 00:00:00:00:00:00:00:01
05:42:29.701 INFO [n.f.c.i.OFSwitchManager:main] Setting max tables to receive
table-miss flow to 0x1 for DPID 00:00:00:00:00:00:00:02
```

Project Floodlight

```
$ sudo mn --topo tree,depth=2,fanout=3 --switch ovsk
--controller=remote,ip=127.0.0.1,port=6653 --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s2, h1) (s2, h2) (s2, h3) (s3, h4) (s3, h5) (s3, h6)
(s4, h7) (s4, h8) (s4, h9)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9
h2 -> h1 h3 h4 h5 h6 h7 h8 h9
h3 -> h1 h2 h4 h5 h6 h7 h8 h9
h4 -> h1 h2 h3 h5 h6 h7 h8 h9
h5 -> h1 h2 h3 h4 h6 h7 h8 h9
h6 -> h1 h2 h3 h4 h5 h7 h8 h9
h7 -> h1 h2 h3 h4 h5 h6 h8 h9
h8 -> h1 h2 h3 h4 h5 h6 h7 h9
h9 -> h1 h2 h3 h4 h5 h6 h7 h8
*** Results: 0% dropped (72/72 received)
mininet>
```

```
$ ~/odl/bin/karaf

    ________                       ________                .__  .__       .__     __
    \_____  \ ______   ____   ____ \______ \ _____  ___.__.|  | |__| ____ |  |___/  |_
     /   |   \\____ \_/ __ \ /    \ |    |  \\__  \<   |  ||  | |  |/ ___\|  |  \   __\
    /    |    \  |_> >  ___/|   |  \|    `   \/ __ \\___  ||  |_|  / /_/  >   Y  \  |
    \_______  /   __/ \___  >___|  /_______  (____  / ____||____/__\___  /|___|  /__|
            \/|__|        \/     \/        \/     \/\/            /_____/      \/

  Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

OPENDAYLIGHT User Experience (DLUX)

- On the karaf shell install DLUX.
  - odl-restconf -
  - odl-l2switch-switch -
  - odl-mdsal-apidocs -
  - odl-dlux-core -

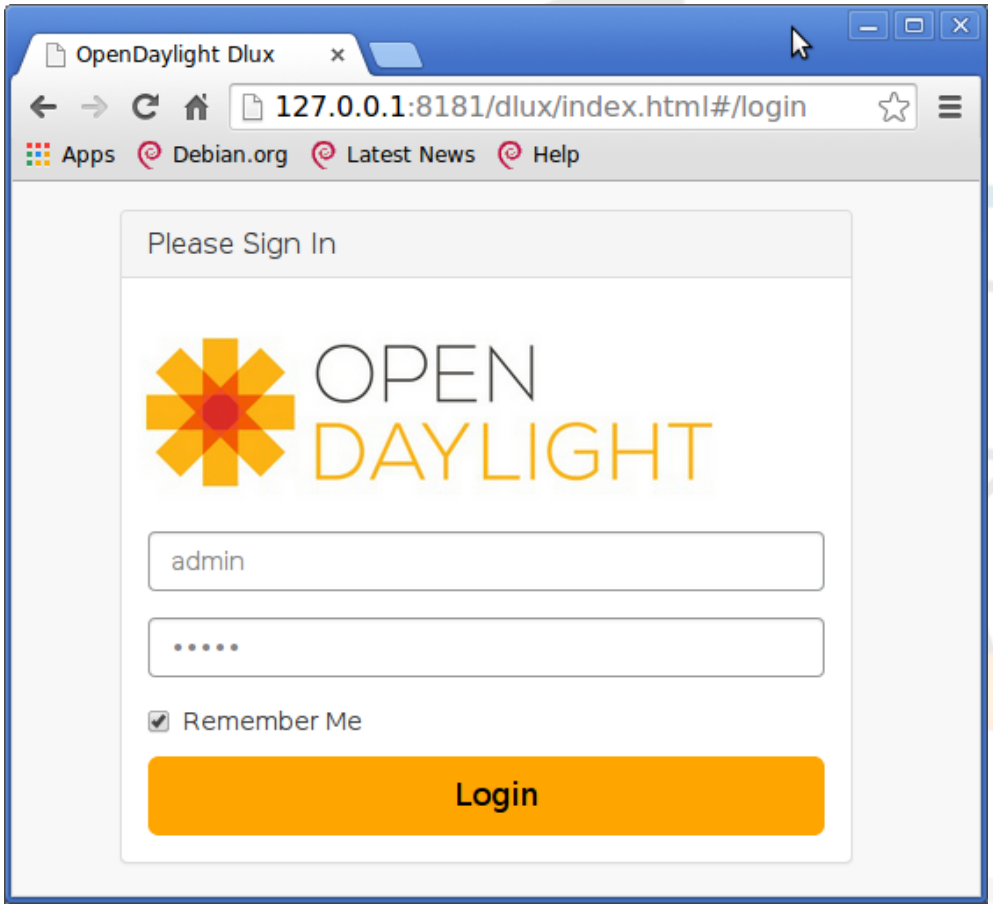
```
opendaylight-user@root> feature:install odl-restconf odl-l2switch-switch
odl-mdsal-apidocs odl-dlux-core
```

```
$ ssh -X sdn@192.168.25.111
$ sudo apt-get install chromium
```

OPENDAYLIGHT

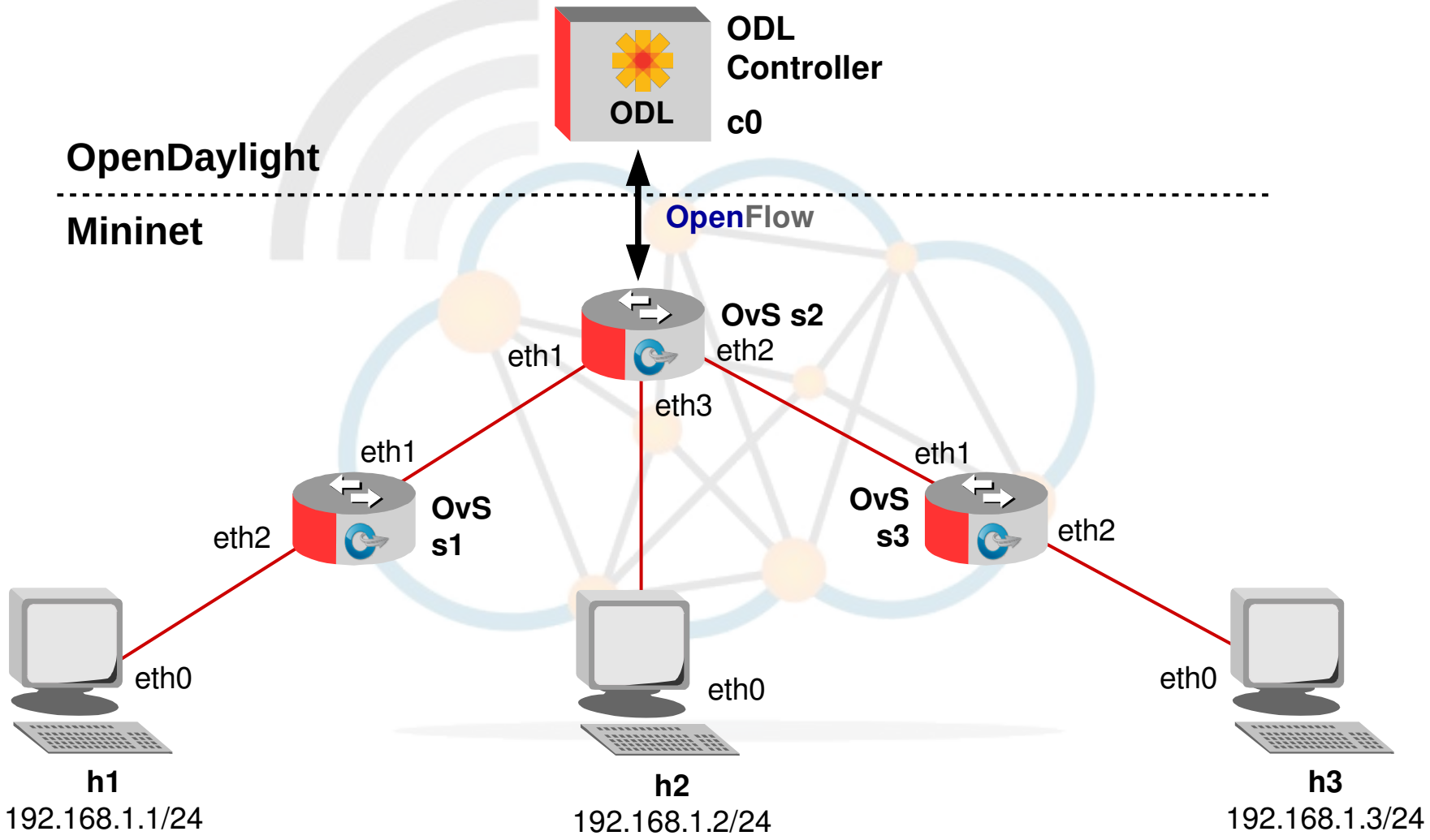
OPENDAYLIGHT User Experience (DLUX)

http://127.0.0.1:8181/index.html



OPENDAYLIGHT

ODL in the custom network



Custom topology with Remote ODL

```
--------------------Custom-RemoteODL.py----------------------

  from mininet.net import Mininet
  from mininet.node import Controller, RemoteController
  from mininet.cli import CLI
  from mininet.log import setLogLevel, info
  #from mininet.topo import Topo

  # OpenDayLight controller
  ODL_CONTROLLER_IP='192.168.25.111'
  ODL_CONTROLLER_PORT=6633

  # Define remote OpenDaylight Controller

  print 'OpenDaylight IP Addr:', ODL_CONTROLLER_IP
  print 'OpenDaylight Port:', ODL_CONTROLLER_PORT
```

Custom topology with Remote ODL

```
def customNet():

      "Create a customNet and add devices to it."

      net = Mininet( topo=None, build=False )


     # Add controller
      info( 'Adding controller\n' )
      net.addController( 'c0',
                         controller=RemoteController,
                         ip=ODL_CONTROLLER_IP,
                         port=ODL_CONTROLLER_PORT
                       )

      # Add hosts
      info( 'Adding hosts\n' )
      h1 = net.addHost( 'h1' )
      h2 = net.addHost( 'h2' )
      h3 = net.addHost( 'h3' )
```

Custom topology with Remote ODL

```
# Add switches
info( 'Adding switches\n' )
s1 = net.addSwitch( 's1' )
s2 = net.addSwitch( 's2' )
s3 = net.addSwitch( 's3' )

# Add links
info( 'Adding switch links\n' )
net.addLink( s1, s2 )
net.addLink( s2, s3 )

info( 'Adding host links\n' )
net.addLink( h1, s1 )
net.addLink( h2, s2 )
net.addLink( h3, s3 )
```

Custom topology with Remote ODL

```
        info( '*** Starting network ***\n')
        net.start()

        info( '*** Running CLI ***\n' )
        CLI( net )

        info( '*** Stopping network ***' )
        net.stop()

    if __name__ == '__main__':
        setLogLevel( 'info' )
        CustomNet()

-----------------------------------------
```

Run script

```
$ sudo ./Custom-RemoteODL.py
  OpenDaylight IP Addr: 192.168.25.111
  OpenDaylight Port: 6633
  Adding controller
  Adding hosts
  Adding switches
  Adding switch links
  Adding host links
  *** Starting network ***
  *** Configuring hosts
  h1 h2 h3
  *** Starting controller
  c0
  *** Starting 3 switches
  s1 s2 s3
  *** Running CLI ***
  *** Starting CLI:
```

Review network

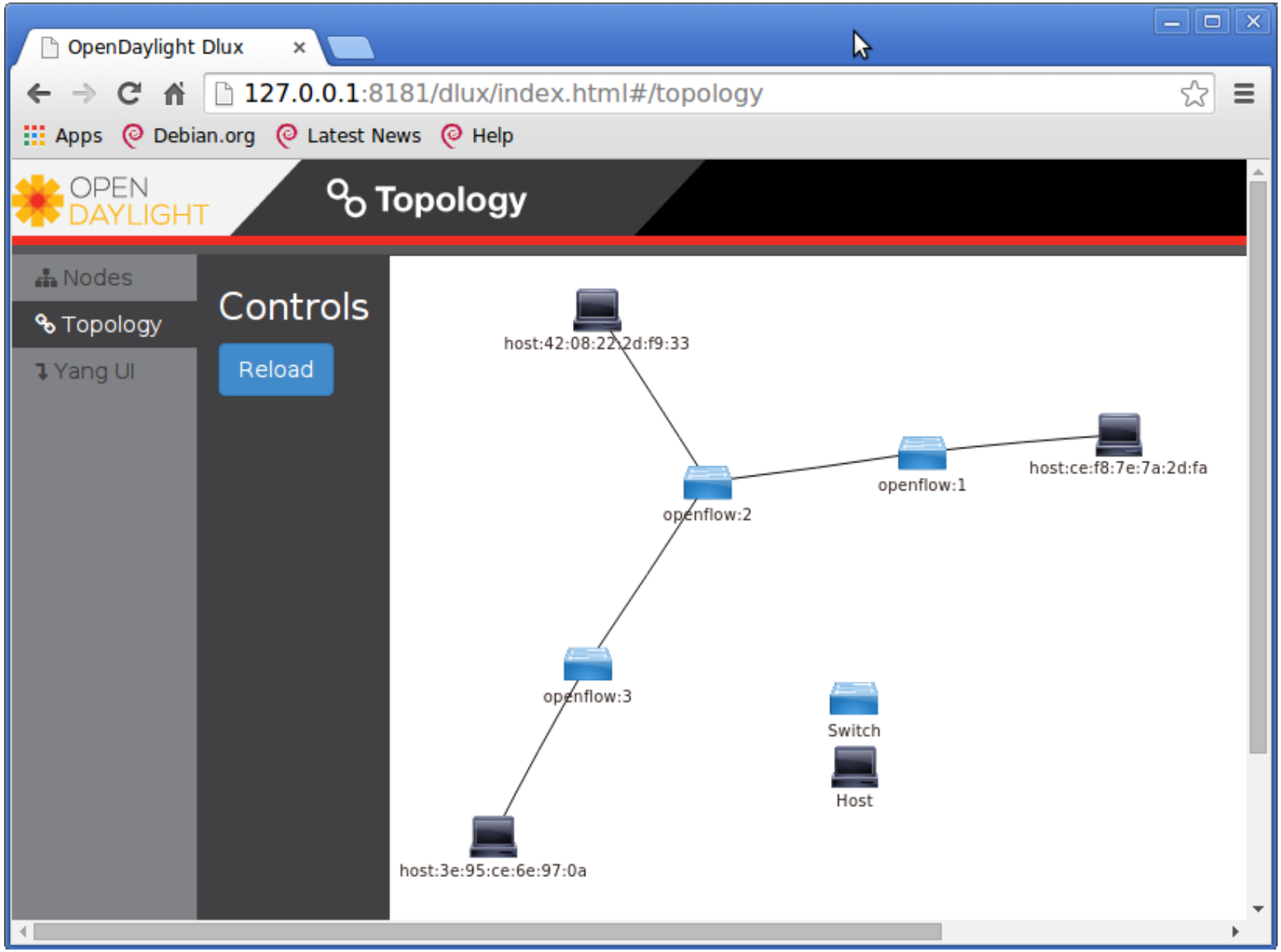
```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=10598>
<Host h2: h2-eth0:10.0.0.2 pid=10601>
<Host h3: h3-eth0:10.0.0.3 pid=10603>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=10608>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=10611>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=10614>
<RemoteController c0: 192.168.25.111:6633 pid=10591>

mininet> net
h1 h1-eth0:s1-eth2
h2 h2-eth0:s2-eth3
h3 h3-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth1 s1-eth2:h1-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:h2-eth0
s3 lo:  s3-eth1:s2-eth2 s3-eth2:h3-eth0
c0
```

Test the topology

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

```
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
Waiting for iperf to start up...*** Results: ['227 Mbits/sec', '234 Mbits/sec']
```

DLUX Topology dashboard



North Bound Interface (NBI)

- Representational State Transfer (ReST) API
  - CORBA, RPC or SOAP to complex
  - ReST uses HTTP for CRUD operations
  - Simple but fully featured
  - HTTP style verbs
    - GET
    - POST
    - PUT
    - DELETE.

- Family of network control languages.
- High-level abstraction
  - Control.
- Modular constructs
  - Compositional reasoning
    - Sequential (>>)
    - Parallel (|).
- Portability
  - Operate on many devices.
- Rigorous semantic foundations
  - Mechanical program analysis tools.

Pyretic

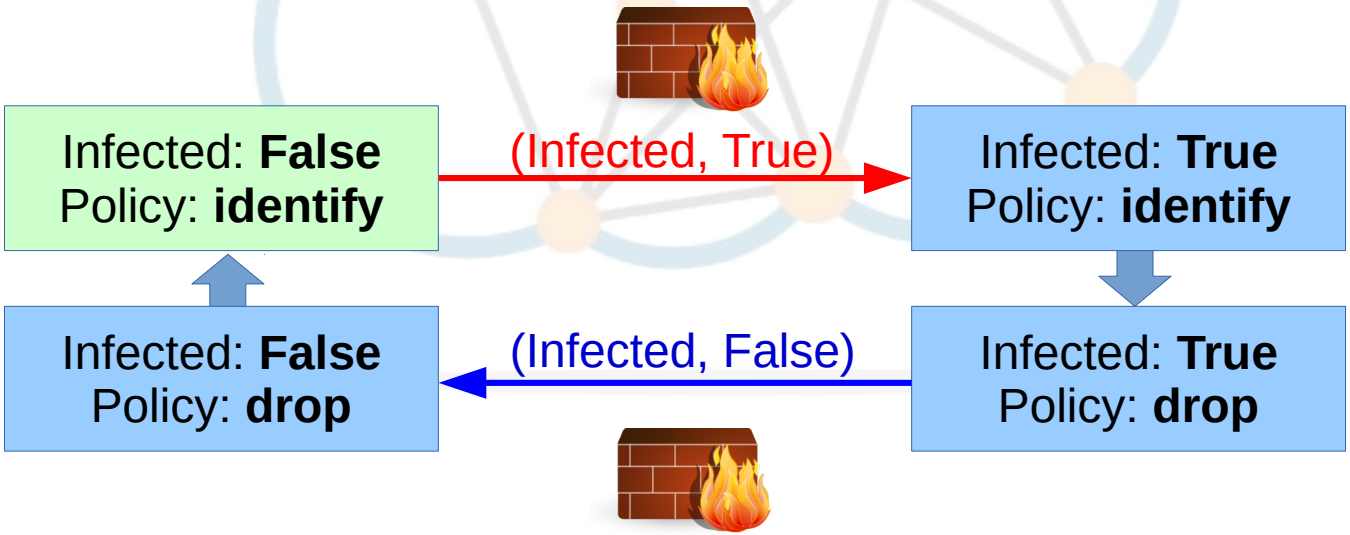
- Python + Frenetic = Pyretic
  - pyretic.lib

```
from pyretic.lib.corelib import *

def main():
    return flood()
```

Kinetic

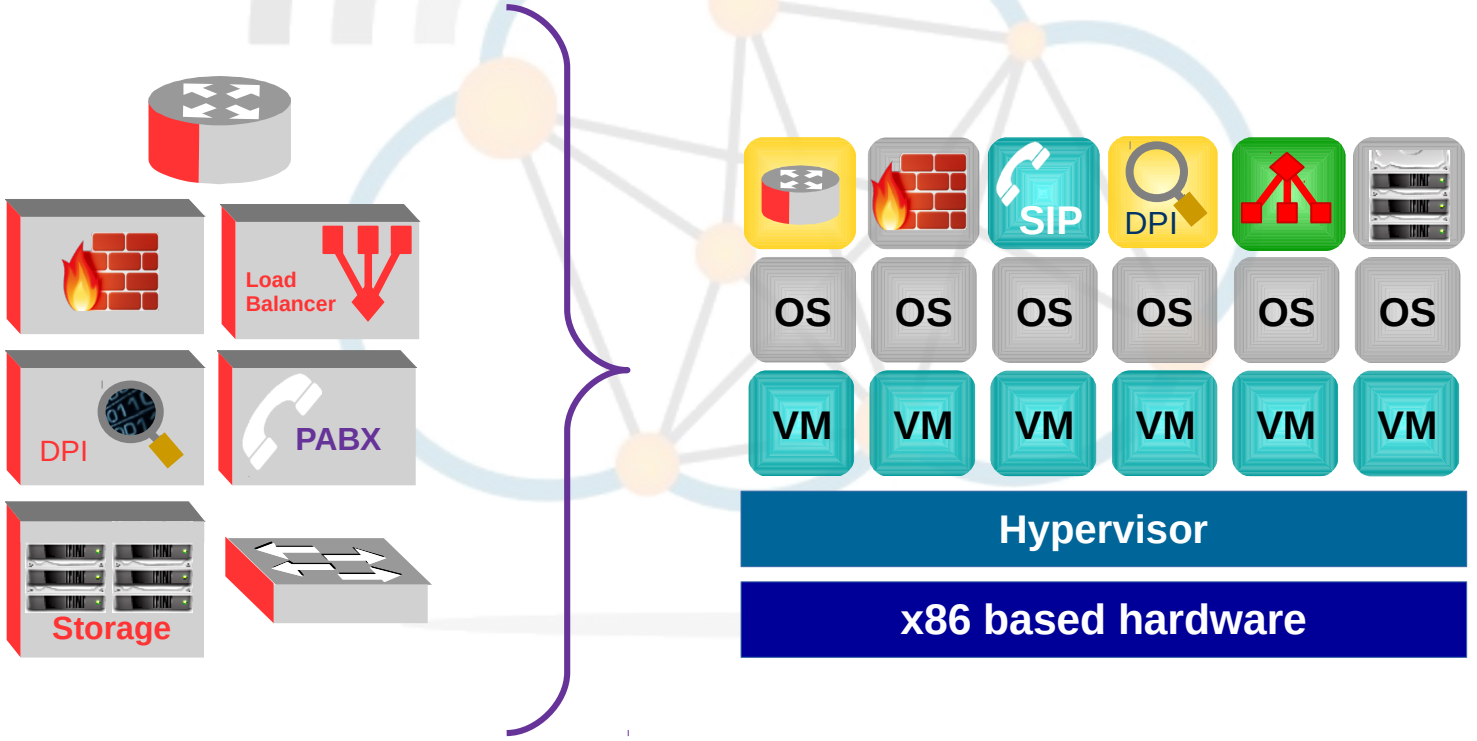
- Kenetic = Pyretic module
  - Network policy as a Finite State Machine (FSM).

```
from pyretic.kinetic.fsm_policy import *
from pyretic.kinetic.smv.model_checker import *

def policy(self):
            self.case(is_true(V('infected')),C(drop))
            self.default(C(identity))
```

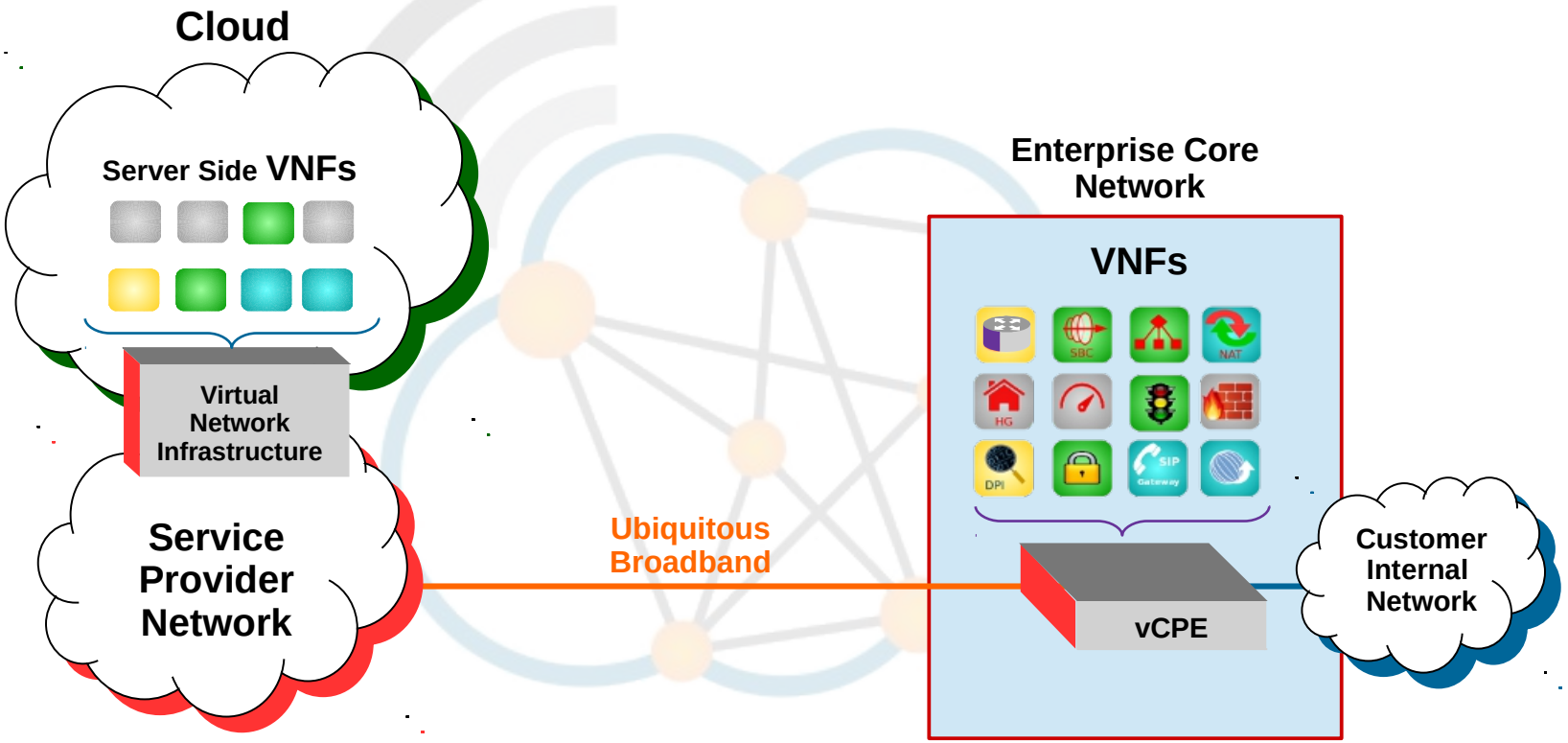


Network Function Virtualisation (NFV)

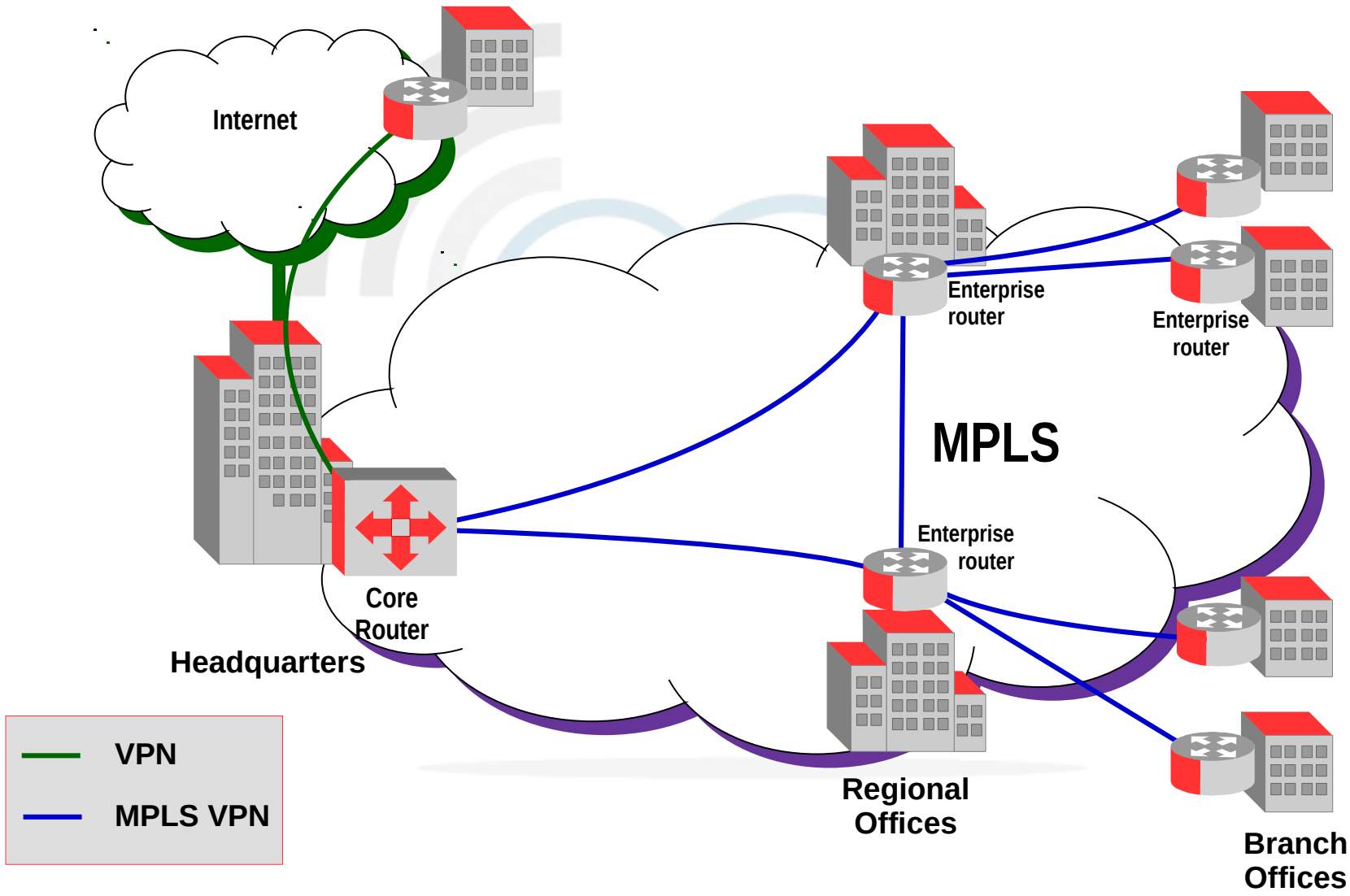
- NFV launched by Tier 1 group
  - SDN & OpenFlow World Congress in Darmstadt, Germany in October 2012



Virtual Customer Premise Equipment (vCPE)

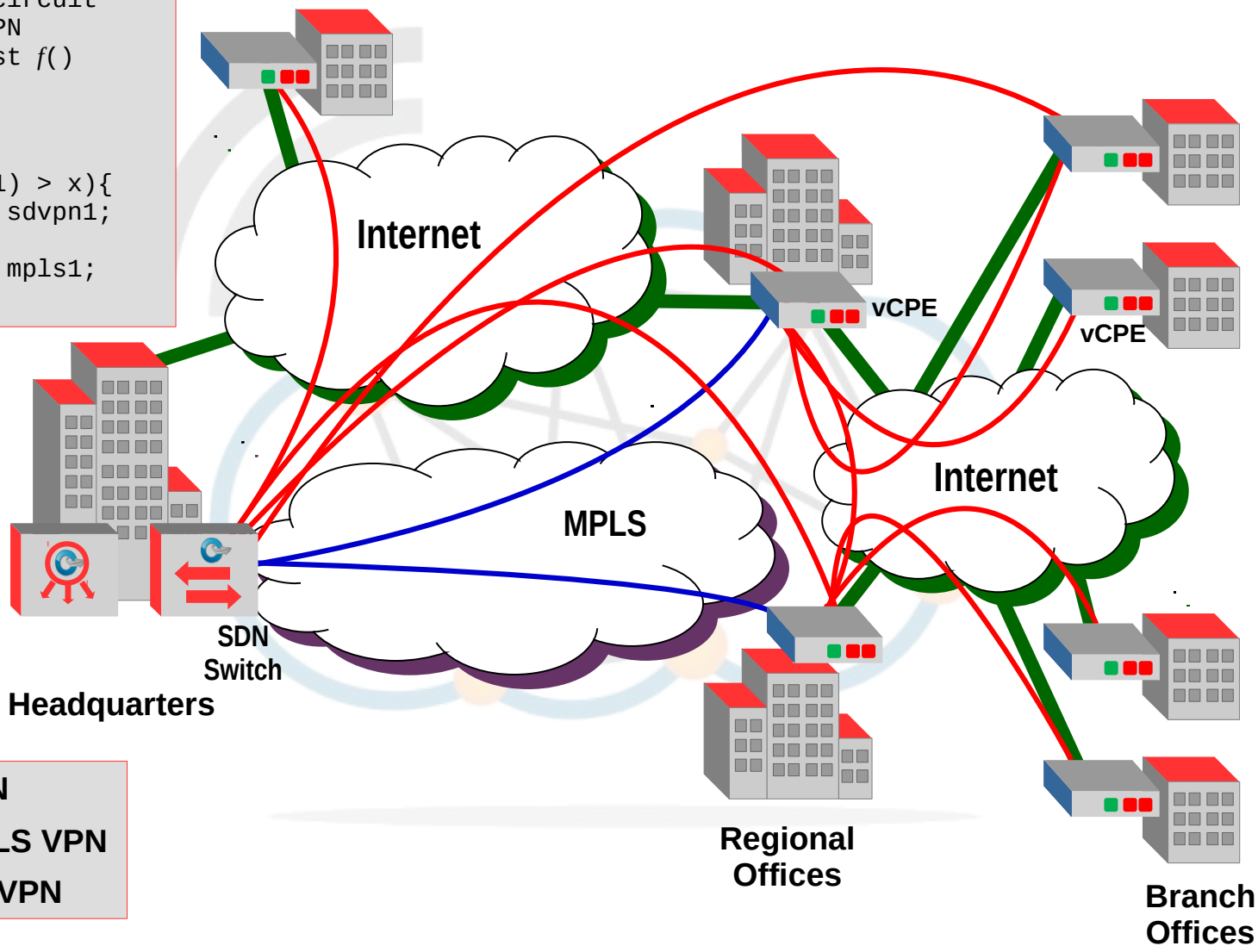


Traditional Wide Area Network (WAN)



Software Defined-WAN (SD-WAN)

```
# mpls1 - MPLS circuit
# sdvpn1 - SD-VPN
# bwt() - BW test f()

x = 100

forward:
  if (bwt(sdvpn1) > x){
     forward => sdvpn1;
  }else{
     forward => mpls1;
  }
```



Software Defined-WAN (SD-WAN)

- **Lower cost**
  - Software-defined WAN, an enterprise can rely more on broadband and less on private links.
  - Broadband has no quality guarantees, so the SD-WAN manages the situation.
- **Reduced complexity**
  - SD-WAN automates, routing and rerouting traffic dynamically based on the current state of the network.
- **Increased flexibility**
  - SD-WAN technology enables the hybrid WAN to react to changing network conditions automatically.

Thank You

Diarmuid Ó Briain
CEng, FIEI, FIET, CISSP

diarmuid@obriain.com