Software Defined Networks

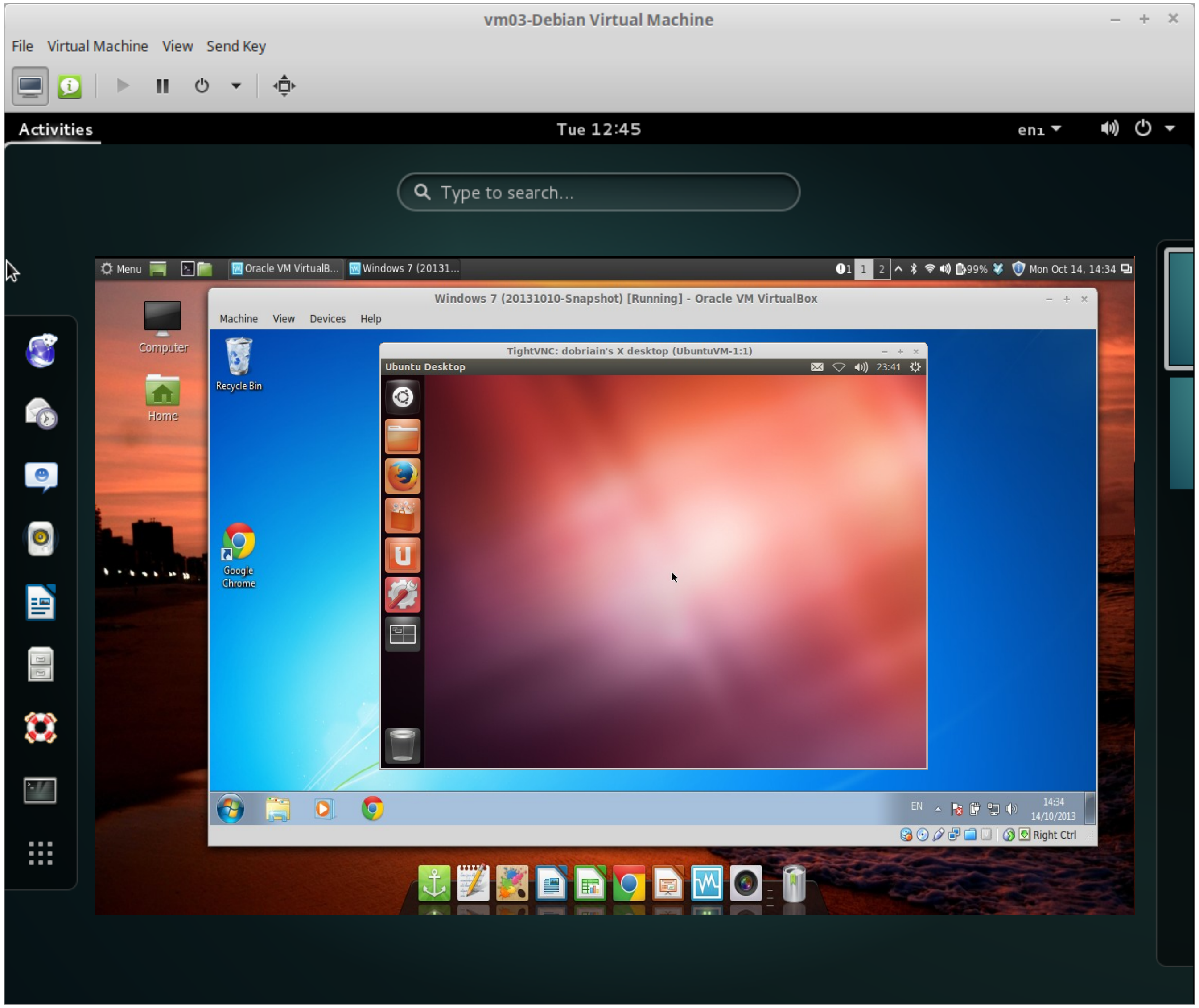
Dr Diarmuid Ó Briain

Learning objectives

- Introduction to Software Defined Networking (SDN)
- SDN Architecture
- Build a Ryu SDN testbed
- Build a Mininet test network
- The Open vSwitch
- OpenFlow communications
- RESTful API
- Building a simple test network
- Ryu Framework
- Custom Topologies
- Custom script to Ryu remote controller
- Developing Ryu applications
- Flow parameters
- OpenFlow pipeline processing
- Splitting domains
- Building a simple L3 and L4 switches

Introduction to
Software Defined Networking

Virtualisation and Containers

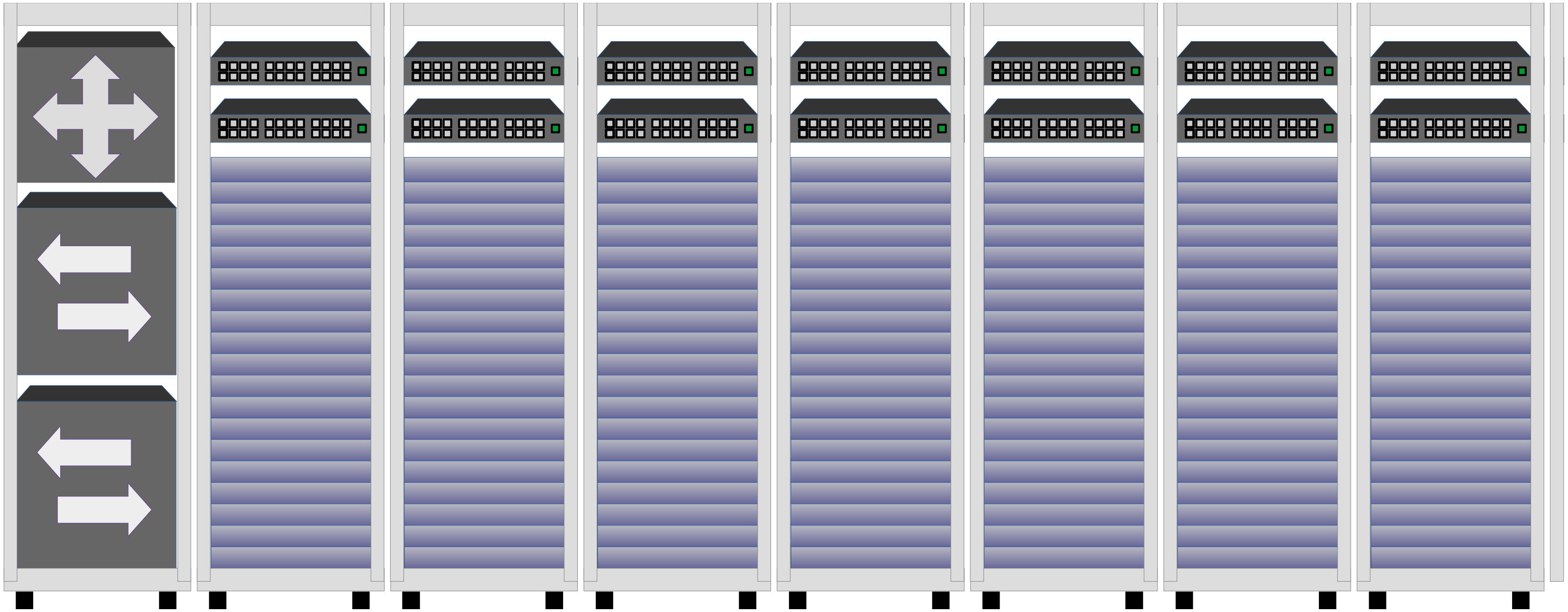


Cloud Computing

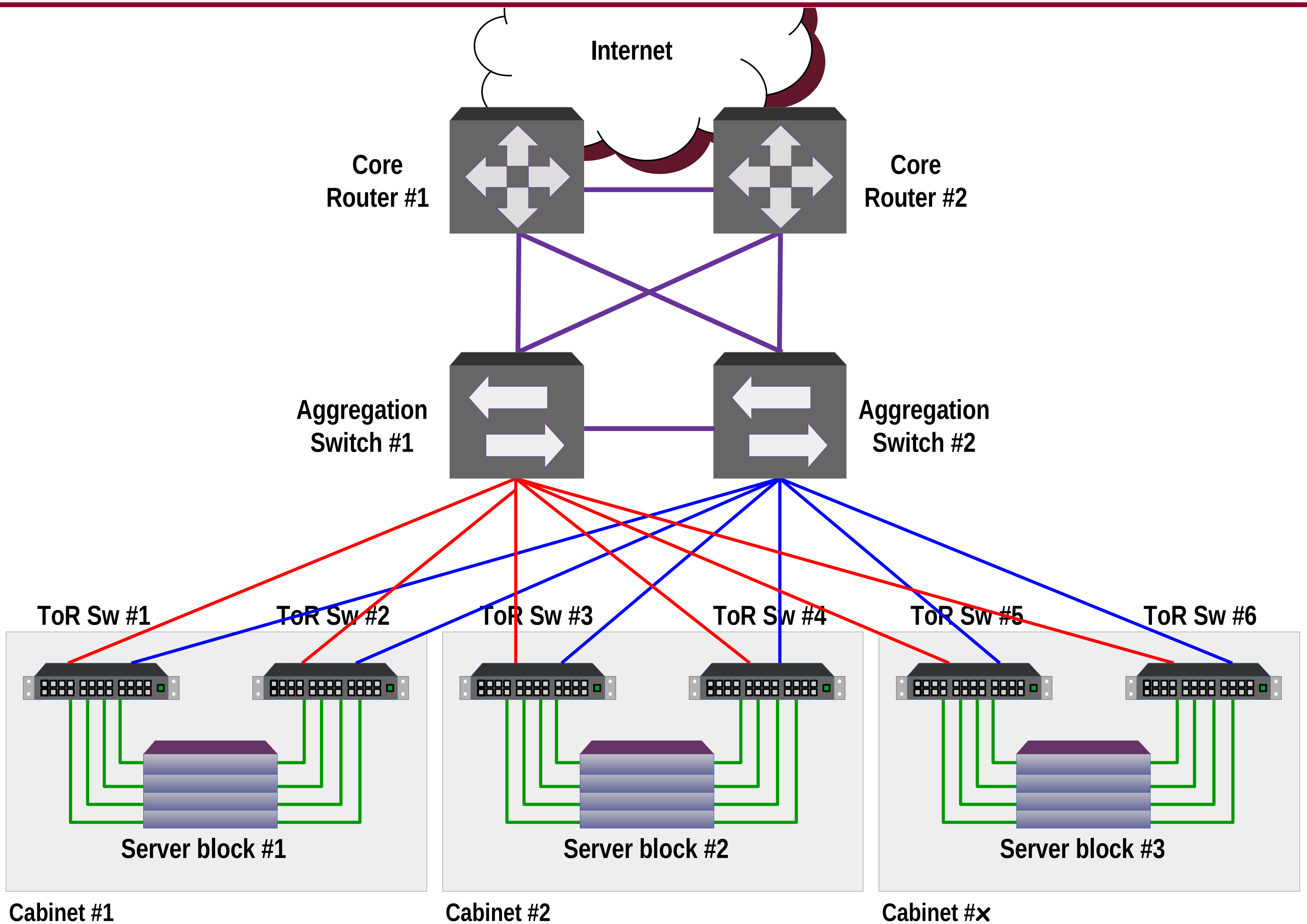
- Elastic Compute
- Elastic MapReduce

Traditional Data Centre layout

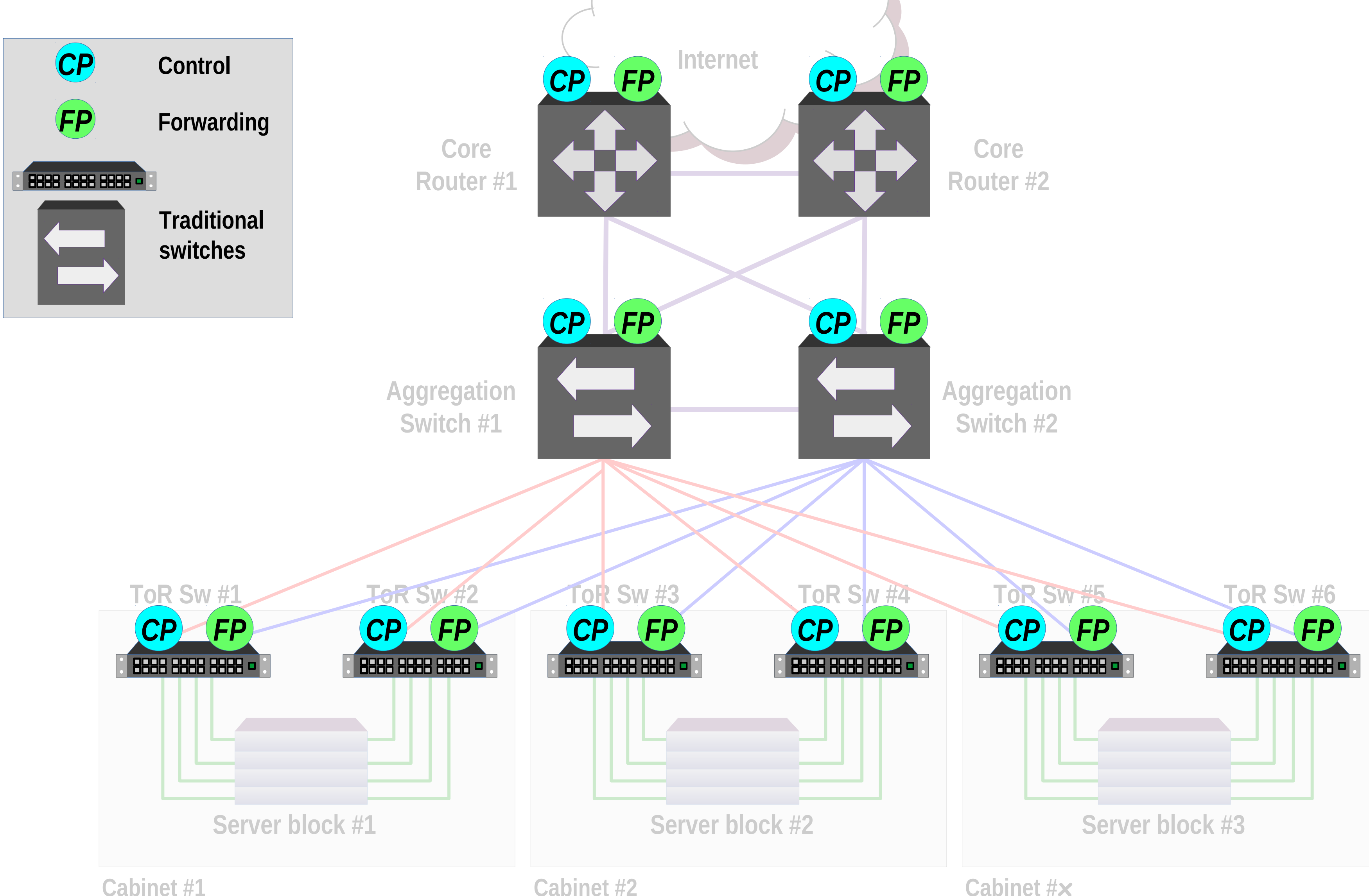
- Racks of servers
- Two Top of Rack (ToR) switches
- Aggregation switches
- Core Routers



Traditional Data Centre layout



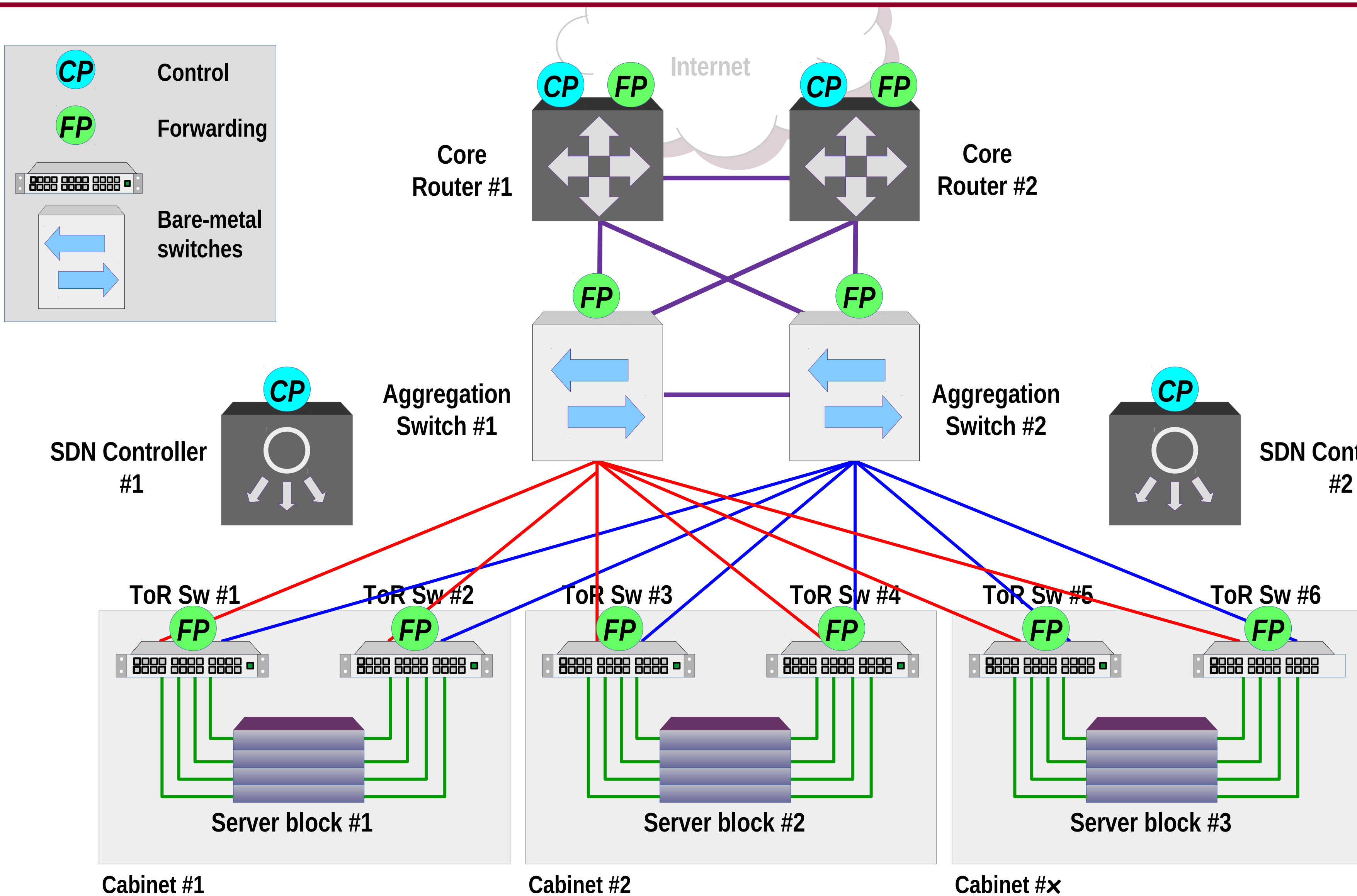
Traditional Control & Forwarding Planes



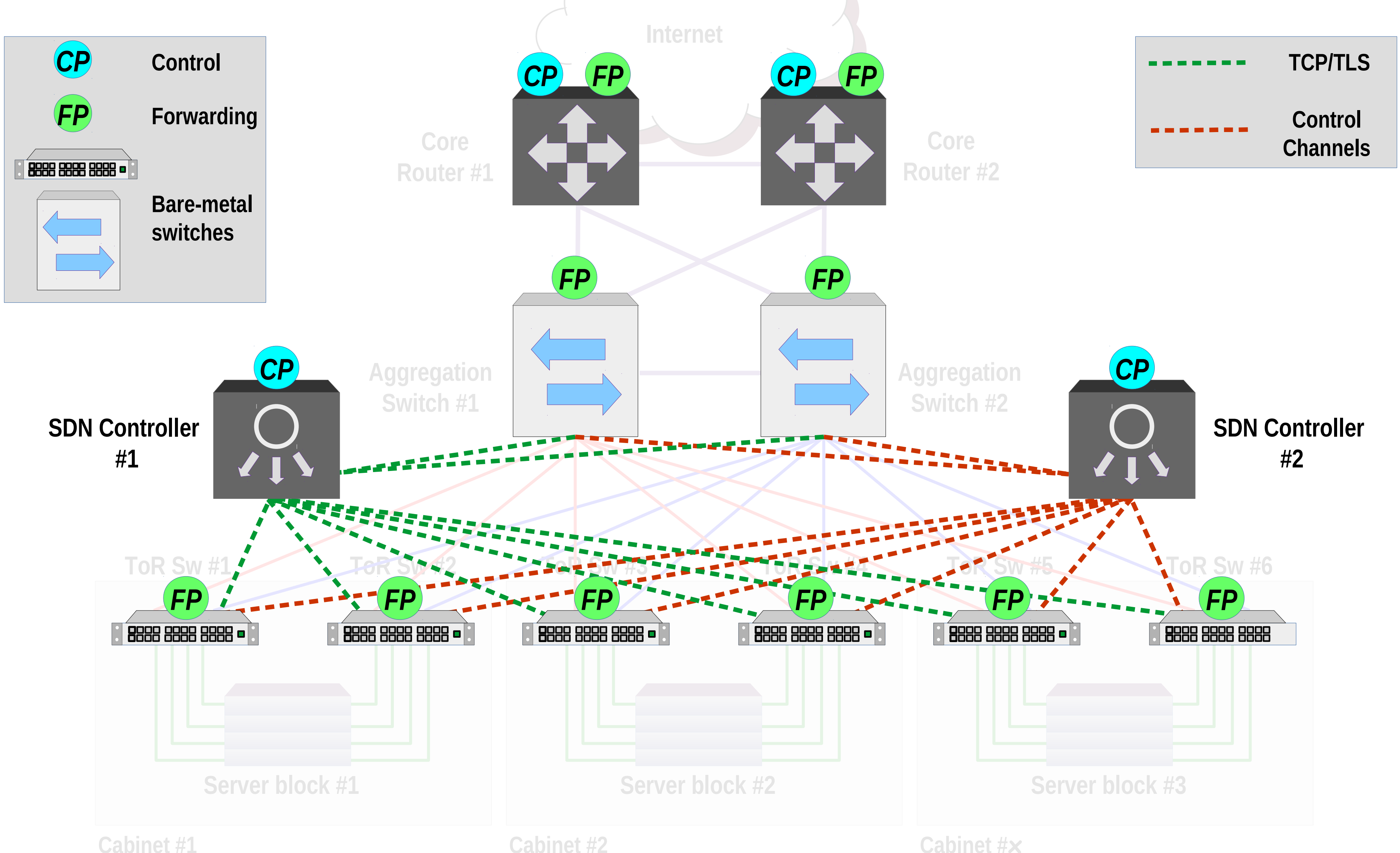
What is Software Defined Networking

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

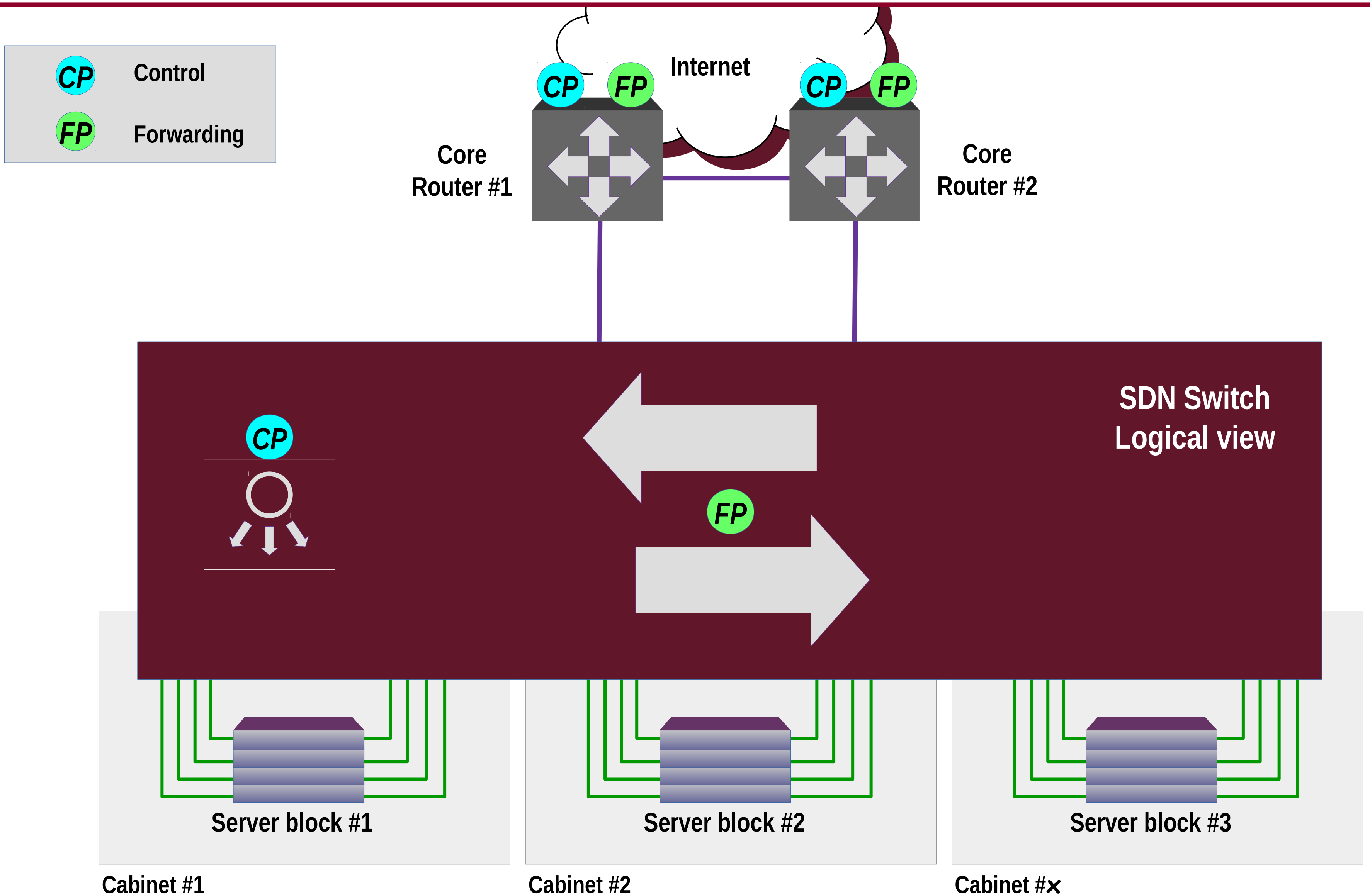
How SDN has changed the Data Centre



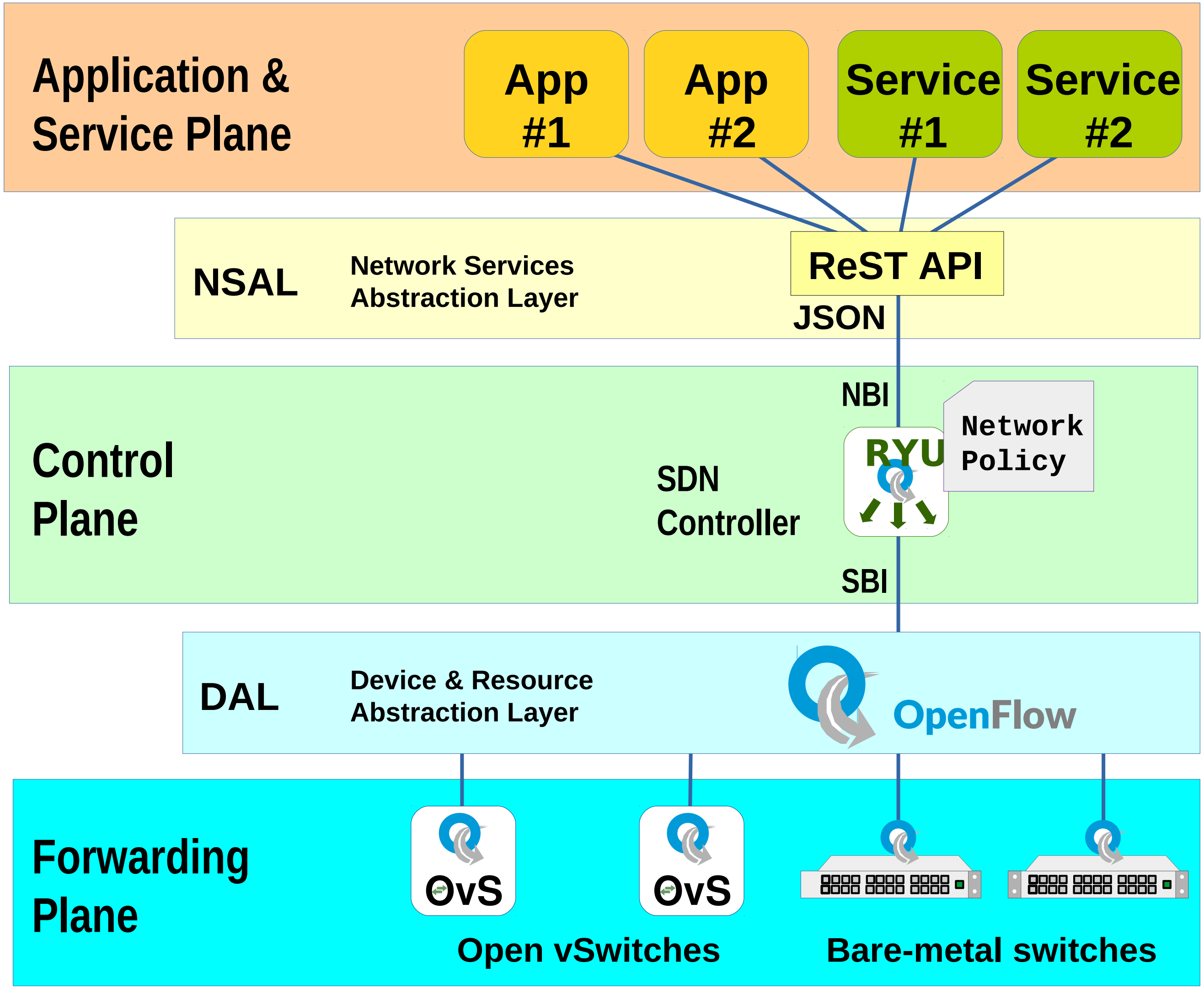
SDN Controller – Device communication



SDN logical view



SDN Architecture



- Southbound SDN protocol to facilitate modification of the flow table in a supporting switch
- Secure channel, TLS, on either TCP port 6633 or 6653
- Managed by the ONF founded in 2011
- OpenFlow has evolved to version 1.5.1; however, hardware typically supports up to v1.3

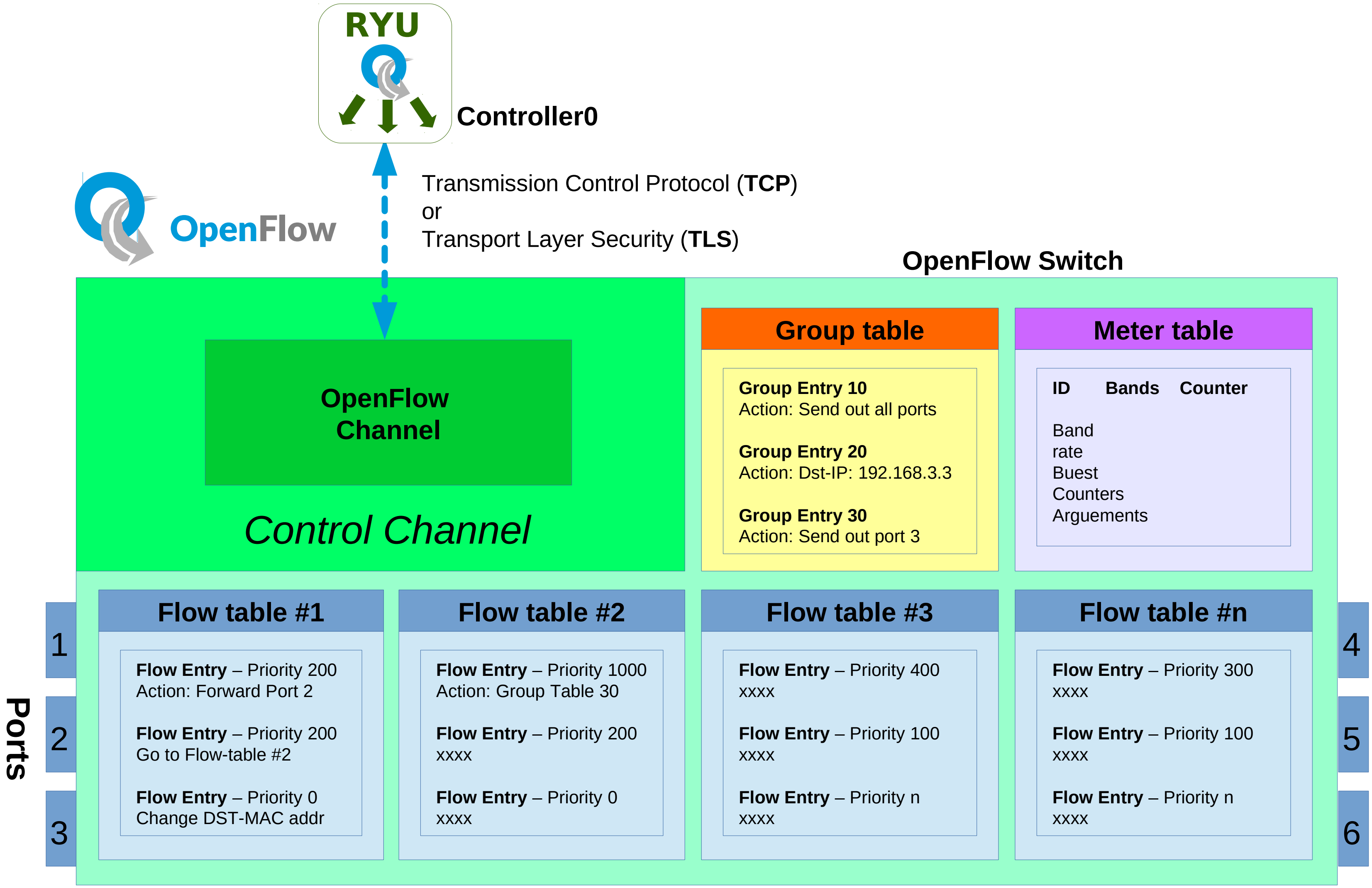
OvS

- The OpenFlow spec included a virtual Switch daemon (vswitchd)
- OvS a softswitch solution that operates over OpenFlow and can be used in virtualised situations where a physical switch is unnecessary

Whitebox switches

- Reduced dependency on vendors like Cisco, Juniper and HP whose switches were very expensive and over featured
- Various hardware vendors produced whitebox switches without software which were ideal for building custom OpenFlow implementations
- Other vendors such as Netgear have incorporated OpenFlow into their enterprise stacked switch models

OpenFlow internals



OpenFlow fields

OpenFlow Flow

Cookie	Timeouts	Priority	Match fields	Instructions	Counters
0x0	idle_timeout=20 hard_timeout=50	1	in_port=1 dl_src=00:00:00:00:00:01 dl_dst=00:00:00:00:00:02	actions=output:2	duration=764.169s n_packets=242538 n_bytes=13851836796

Ryu SDN Controller

- Ryū (竜) the Japanese word for dragon
- SDN Framework developed at NTT Laboratories, Japan
- Released in 2012
- Apache 2.0 open-source license
- Tools and libraries for the assembly of SDN networks
- Compatible with OpenFlow 1.0, 1.2, 1.3, 1.4, 1,5 and Nicira
- A component-based, Python networking framework
- Supports multiple southbound protocols
  - OpenFlow
  - NETCONF
  - OF-Config

Mininet

- Network emulator
- Creates a network of virtual hosts, switches, controllers, and links
- Runs on standard Linux OS
- Switches support OpenFlow for highly flexible custom routing and SDN

Laboratory

Setup Testbed Enviornment

Laboratory

- Follow the instructions in the lab manual or download the VM and install in VirtualBox, KVM or other Virtualisation platform.

http://www.obriain.com/training/sdn

Virtual Machine

Example scripts

Test the laboratory

Test the laboratory

- Run up three terminal windows
  - 1: The Ryu Controller
  - 2: The mininet network
  - 3: The Open vSwitch

Ryu Controller

- Simple Switch OpenFlow v1.3

Window: 1 - Ryu Controller

```
sdn@sdn-mn:~$ ryu-manager ryu.app.simple_switch_13
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Mininet

Window: 2 - Mininet

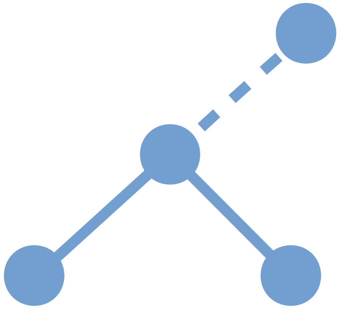
```
sdn@sdn-mn:~$ sudo mn --controller remote,ip=127.0.0.1  --
switch ovsk,protocols=OpenFlow13 --mac --ipbase=10.1.1.0/24 --
topo single,4
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Mininet

Window: 2 - Mininet

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

> sudo mn



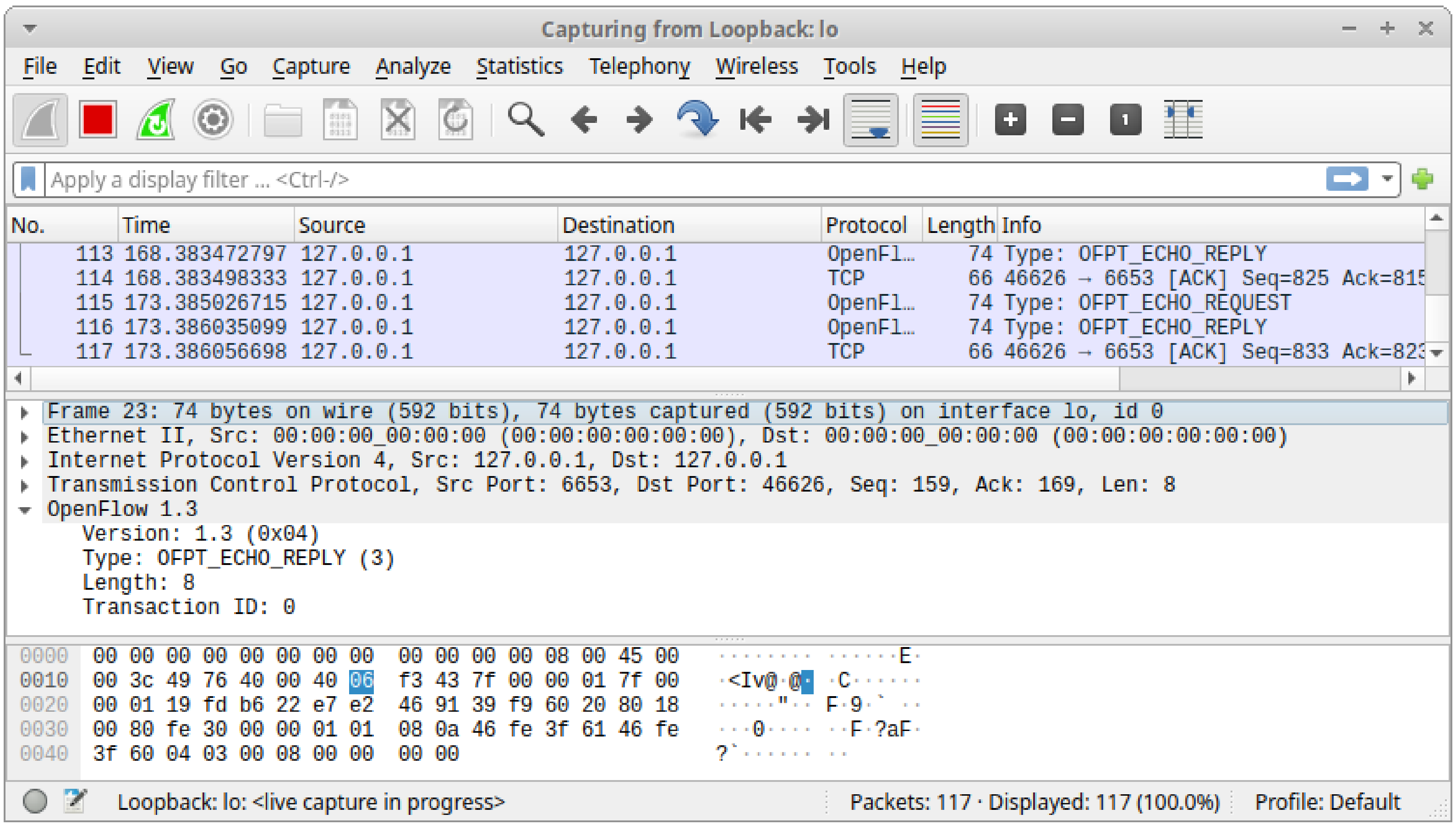
Open vSwitch

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-vsctl show

sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
```

Wireshark

- Interface: **loopback lo**
- Filter: **openflow_v4** (OpenFlow 1.3)



Mininet, quit

Window: 2 - Mininet

```
mininet> quit
*** Stopping 1 controllers
c0
*** Stopping 6 terms
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 6.658 seconds
```

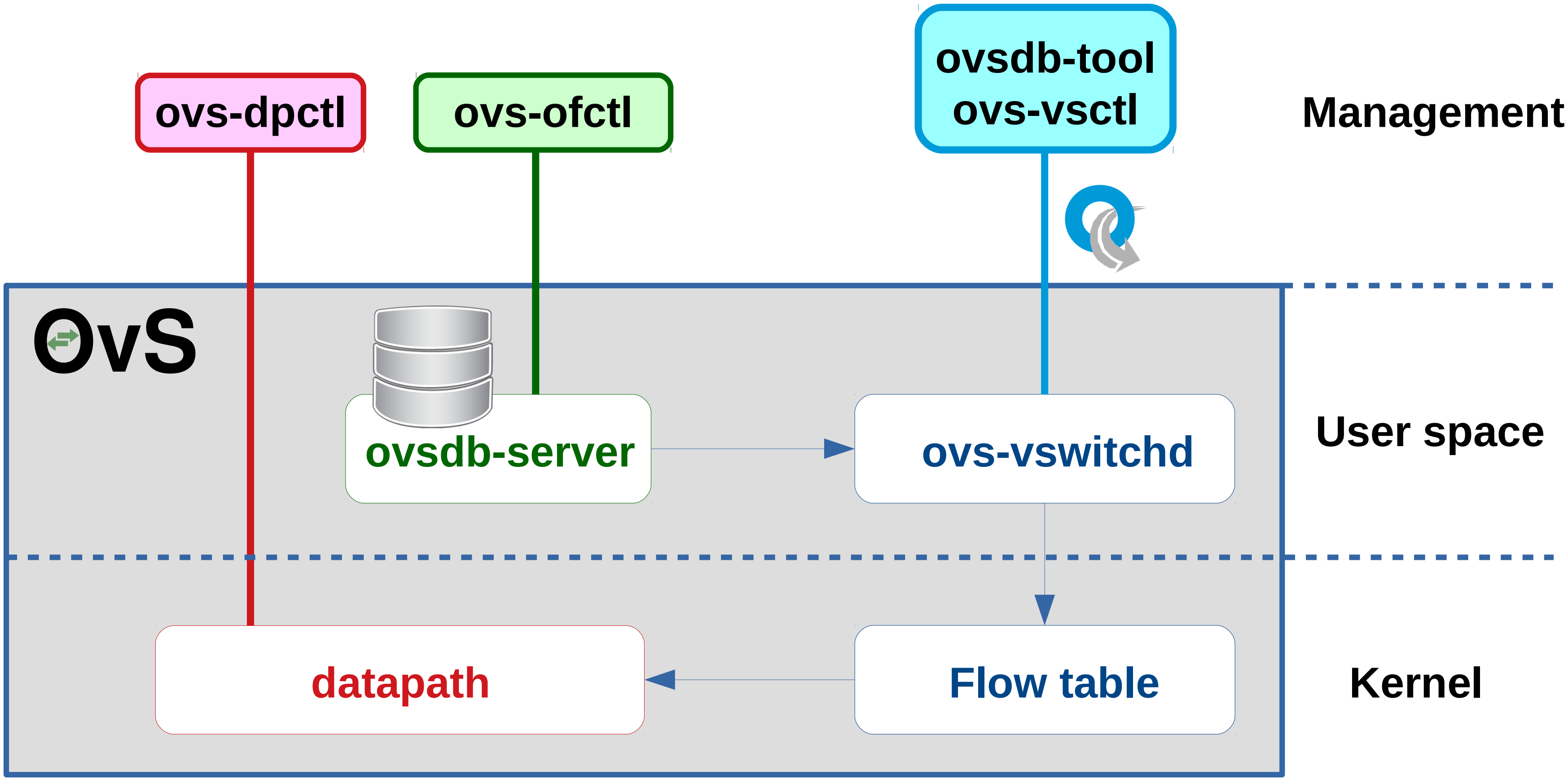
Mininet - cleanup

Window: 2 - Mininet

```
sdn@sdn-mn:~$ sudo mn --clean
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-
openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager
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 datapaths
ovs-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:]]+)'
ip link show
*** 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.
```

A closer view of
Open virtual Switch (OvS)

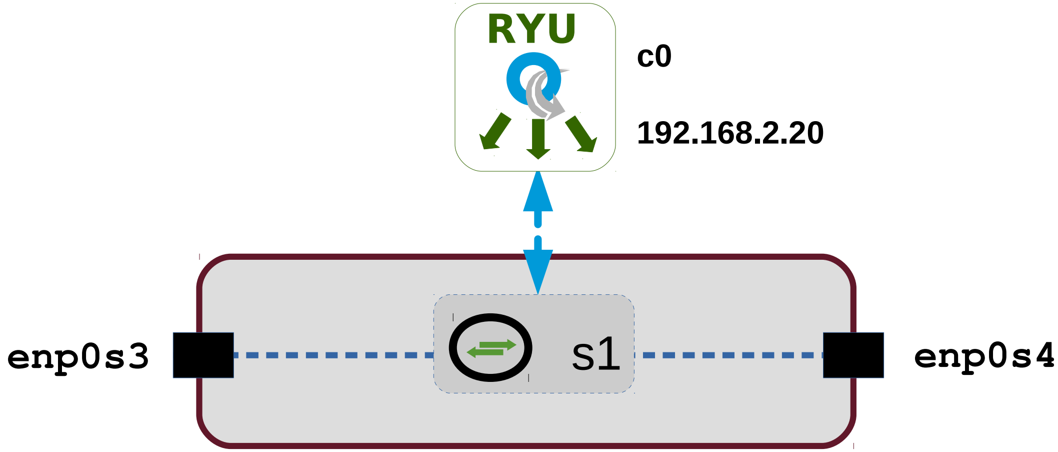
Open vSwitch



Open vSwitch

- **ovsdb-server**:
  - RPC interfaces to one or more OVSDB
  - Switch table database and list of external clients can talk to ovsdb-server
  - ovsdb clients can manipulate using the ovsdb management protocol
- **ovs-vswitchd**:
  - Main OvS userspace program, a daemon to manage any number of OvS switches on the local machine
  - Retrieves OvS configuration from ovsdb-server using an IPC channel
  - Passes status and statistical information to the database
- **ovs-vsctl**:
  - Utility for querying and updating the configuration of ovs-vswitchd (with the help of ovsdb-server)

Open vSwitch



Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-vsctl show
8ba60966-6a3b-4696-884d-745a1ab733b4
    ovs_version: "2.13.0"

sdn@sdn-mn:~$ ls /sys/class/net
enp0s3  enp0s4  lo

sdn@sdn-mn:~$ sudo ovs-vsctl add-br s1
sdn@sdn-mn:~$ sudo ovs-vsctl add-port s1 enp0s3
sdn@sdn-mn:~$ sudo ovs-vsctl add-port s1 enp0s4
sdn@sdn-mn:~$ sudo ovs-vsctl set-controller s1 tcp:192.168.2.20:6633
```

Open vSwitch

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-vsctl show
8ba60966-6a3b-4696-884d-745a1ab733b4
    Bridge s1
        Controller "tcp:192.168.2.20:6633"
            is_connected: true
        Port s1
            Interface s1
                type: internal
        Port enp0s3
            Interface enp0s3
        Port enp0s4
            Interface enp0s4
    ovs_version: "2.13.0"

sdn@sdn-mn:~$ sudo ovs-ofctl add-flow s1 actions=NORMAL
sdn@sdn-mn:~$ sudo ovs-ofctl del-flows s1
sdn@sdn-mn:~$ sudo ovs-ofctl add-flow s1 in_port=1,actions=output:2
sdn@sdn-mn:~$ sudo ovs-ofctl add-flow s1 in_port=2,actions=output:1

sdn@sdn-mn:~$ sudo ovs-vsctl del-br s1
sdn@sdn-mn:~$ sudo ovs-vsctl show
8ba60966-6a3b-4696-884d-745a1ab733b4
    ovs_version: "2.13.0"
```

Open vSwitch

- **ovs-ofctl**:
  - Monitor and administer OpenFlow switches
  - List implemented flows in the OVS kernel module
- **ovs-dpctl**:
  - create, modify, and delete OvS datapaths
- **ovsdb-tool**:
  - Manage OVSDB files
  - It does not interact directly with running OvSDB servers

Open vSwitch - Description

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-desc s1
OFPST_DESC reply (OF1.3) (xid=0x2):
Manufacturer: Nicira, Inc.
Hardware: Open vSwitch
Software: 2.13.0
Serial Num: None
DP Description: s1
```

Open vSwitch - Details

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 show s1
OFPT_FEATURES_REPLY (OF1.3) (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS GROUP_STATS QUEUE_STATS
OFPST_PORT_DESC reply (OF1.3) (xid=0x3):
 1(s1-eth1): addr:22:c8:cd:f9:8f:6e
     config:     0
     state:      LIVE
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:4a:d3:34:85:d8:b3
     config:     0
     state:      LIVE
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:8e:12:3f:bd:a3:a4
     config:     0
     state:      LIVE
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:c6:b6:be:aa:e0:4f
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (OF1.3) (xid=0x9): frags=normal miss_send_len=0
```

Open vSwitch – Port statistics

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-ports s1
OFPST_PORT reply (OF1.3) (xid=0x2): 4 ports
  port LOCAL: rx pkts=0, bytes=0, drop=30, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=117.904s
  port  "s1-eth1": rx pkts=19, bytes=1486, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=53, bytes=5125, drop=0, errs=0, coll=0
           duration=117.922s
  port  "s1-eth2": rx pkts=19, bytes=1486, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=53, bytes=5125, drop=0, errs=0, coll=0
           duration=117.923s
  port  "s1-eth3": rx pkts=19, bytes=1486, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=53, bytes=5125, drop=0, errs=0, coll=0
           duration=117.922s
```

Open vSwitch – Activity snoop

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-ofctl --verbose snoop s1
2020-05-14T10:44:57Z|00001|stream_unix|DBG|/var/run/openvswitch/s1:
connection failed (No such file or directory)
2020-05-14T10:44:57Z|00002|ofctl|DBG|connecting to
unix:/var/run/openvswitch/s1.snoop
2020-05-14T10:44:57Z|00003|hmap|DBG|../lib/ofp-msgs.c:1381: 1
bucket with 6+ nodes, including 1 bucket with 6 nodes (128 nodes
total across 128 buckets)
2020-05-14T10:44:57Z|00004|hmap|DBG|../lib/ofp-msgs.c:1381: 1
bucket with 6+ nodes, including 1 bucket with 6 nodes (256 nodes
total across 256 buckets)
2020-05-14T10:44:57Z|00005|hmap|DBG|../lib/ofp-msgs.c:1381: 4
buckets with 6+ nodes, including 1 bucket with 8 nodes (512 nodes
total across 512 buckets)
2020-05-14T10:44:57Z|00006|hmap|DBG|../lib/ofp-msgs.c:1381: 8
buckets with 6+ nodes, including 2 buckets with 7 nodes (1024 nodes
total across 1024 buckets)
```

Open vSwitch – Log

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo head /var/log/openvswitch/ovs-vswitchd.log
2020-05-13T16:41:27.974Z|00001|vlog|INFO|opened log file
/var/log/openvswitch/ovs-vswitchd.log
2020-05-13T16:41:28.003Z|00002|ovs_numa|INFO|Discovered 1 CPU cores
on NUMA node 0
2020-05-13T16:41:28.003Z|00003|ovs_numa|INFO|Discovered 1 NUMA
nodes and 1 CPU cores
2020-05-13T16:41:28.003Z|00004|reconnect|INFO|unix:/var/run/
openvswitch/db.sock: connecting...
2020-05-13T16:41:28.003Z|00005|reconnect|INFO|unix:/var/run/
openvswitch/db.sock: connected
2020-05-13T16:41:28.016Z|00006|bridge|INFO|ovs-vswitchd (Open
vSwitch) 2.13.0
2020-05-13T16:45:32.370Z|00007|memory|INFO|13192 kB peak resident
set size after 249.9 seconds
2020-05-13T17:39:18.952Z|00008|ofproto_dpif|INFO|system@ovs-system:
Datapath supports recirculation
2020-05-13T17:39:18.952Z|00009|ofproto_dpif|INFO|system@ovs-system:
VLAN header stack length probed as 2
2020-05-13T17:39:18.952Z|00010|ofproto_dpif|INFO|system@ovs-system:
MPLS label stack length probed as 1
```

Open vSwitch – MAC address table

Window: 3 - Open vSwitch

```
~$ sudo ovs-appctl fdb/show s1
 port  VLAN  MAC                  Age
```

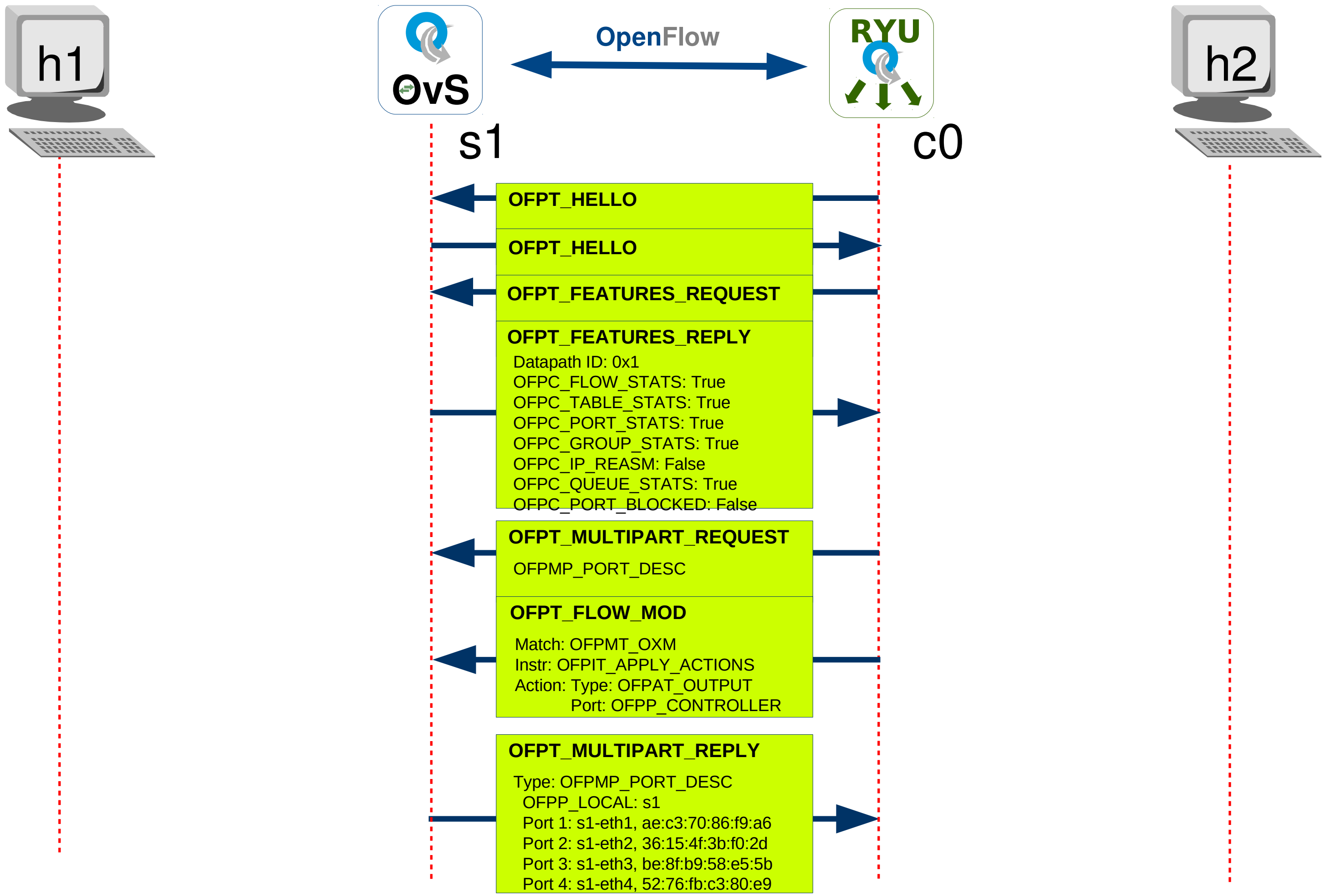
Open vSwitch – Flows

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
 cookie=0x0, duration=194.361s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=194.351s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=194.337s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=194.328s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
 cookie=0x0, duration=194.317s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=194.315s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
 cookie=0x0, duration=217.077s, table=0, n_packets=39, n_bytes=2934, priority=0
actions=CONTROLLER:65535
```

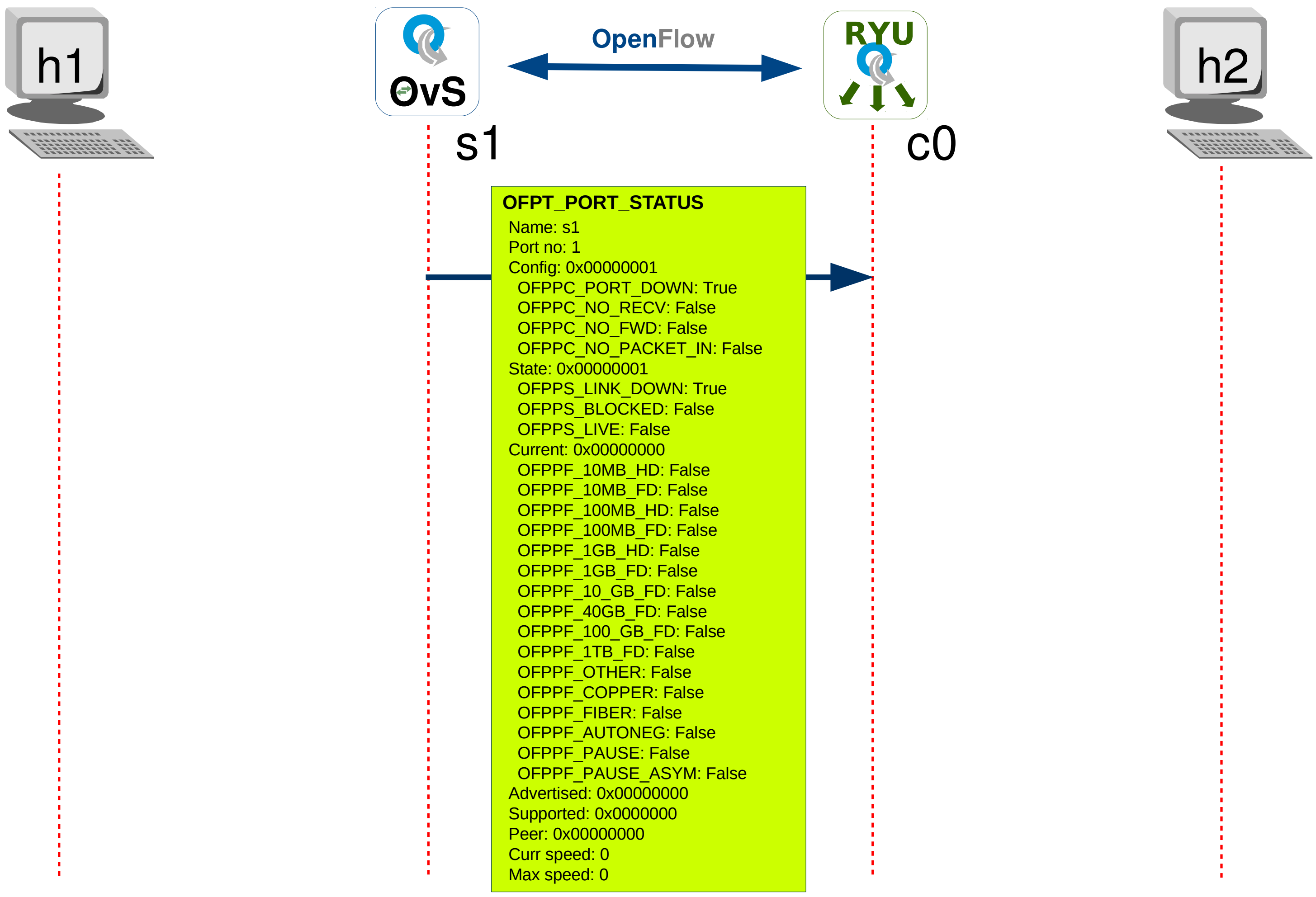
OpenFlow Communications

OpenFlow Handshake

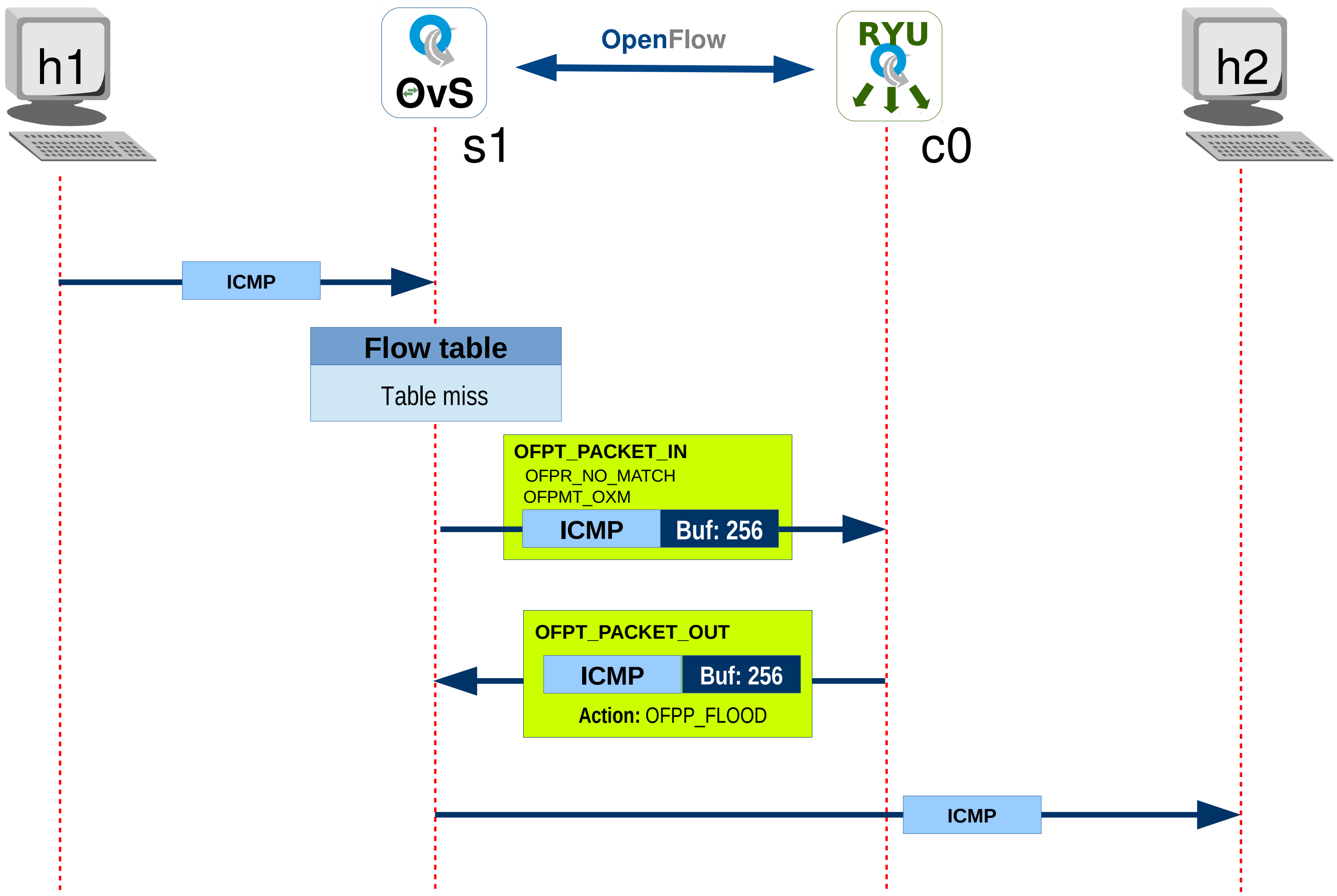



```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=87.447s, table=0, n_packets=27, n_bytes=2138,
priority=0 actions=CONTROLLER:65535
```

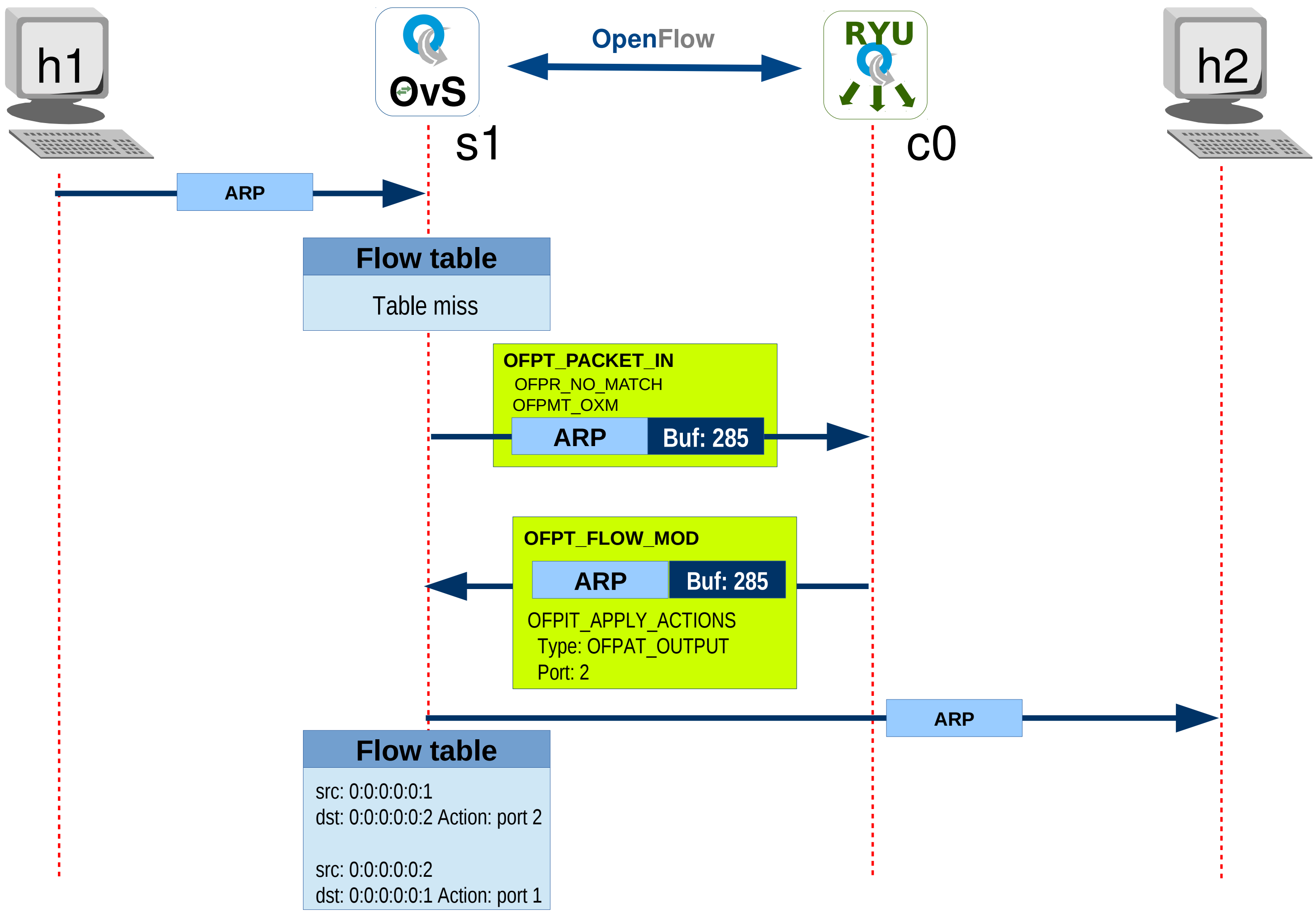
OpenFlow Port status



OpenFlow Packet in, Packet out



OpenFlow Flow modification



Build a Mininet test Network

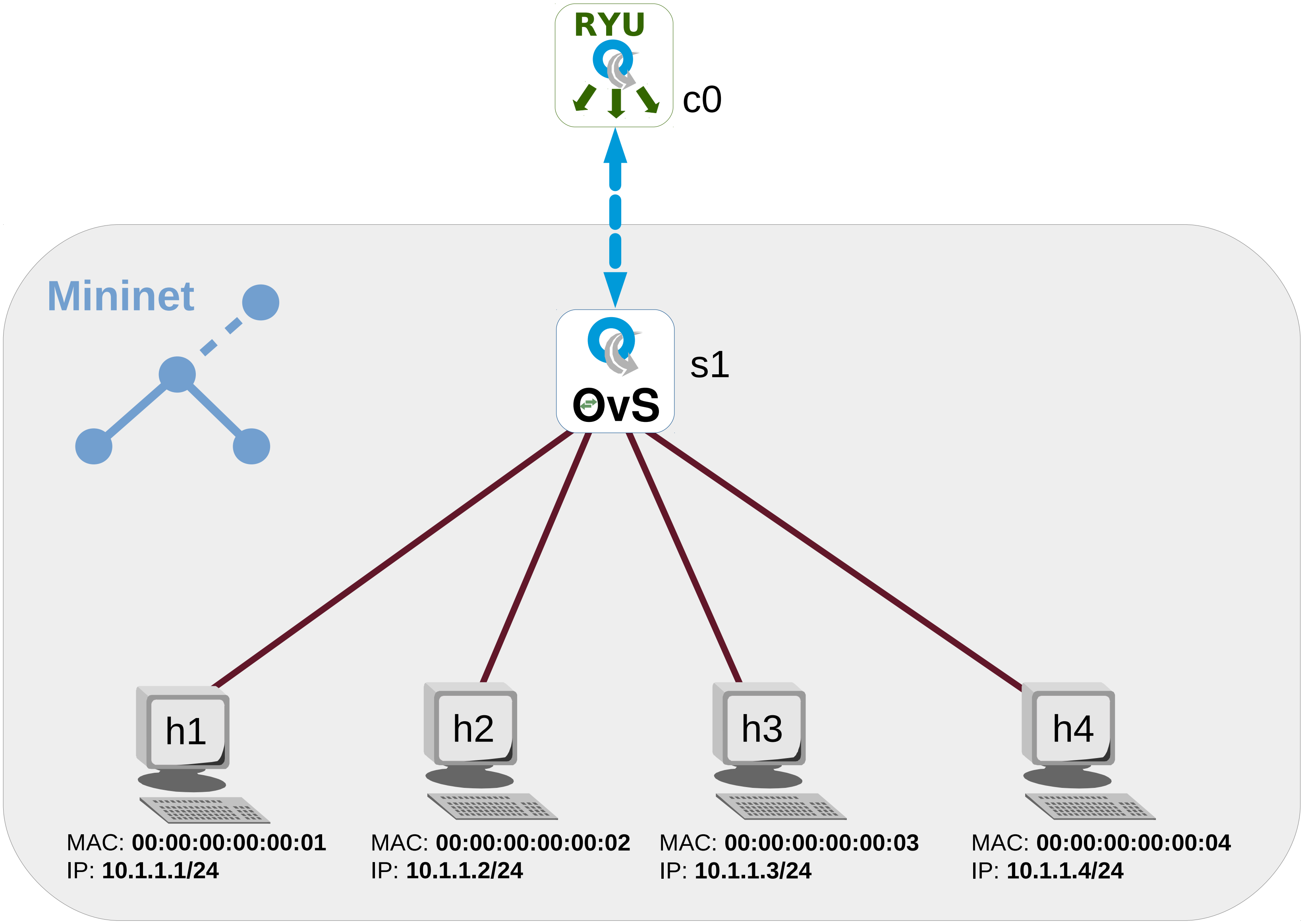
```
> sudo mn
```

Mininet, help

Window: 2 - Mininet

```
sdn@sdn-mn:~$ sudo mn --help
Usage: mn [options]
(type mn -h for details)
The mn utility creates Mininet network from the command line. It can
create parametrized topologies, invoke the Mininet CLI, and run tests.

Options:
  -h, --help                         -v VERBOSITY, --verbosity=VERBOSITY
  --switch=SWITCH                      --innamespace
  --host=HOST                          --listenport=LISTENPORT
  --controller=CONTROLLER              --nolistenport
                                       --pre=PRE
  --link=LINK                          --post=POST
  --topo=TOPO                          --pin
  -c, --clean                          --nat
  --custom=CUSTOM                      --version
  --test=TEST                          --cluster=server1,server2...
  -x, --xterms                         --placement
  -i IPBASE, --ipbase=IPBASE
  --mac
  --arp
```

Build a simple test network



Establish Ryu controller

Window: 1 - Ryu Controller

```
sdn@sdn-mn:~$ ryu-manager ryu.app.simple_switch_13
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

- **ryu-manager**
  - Loads Ryu applications and runs them
- **ryu.app.**
  - Path to Ryu application files
  - ~/.local/lib/python3.8/site-packages/ryu/app
- **simple_switch_13.py**
  - Class: SimpleSwitch13
  - Layer 2 learning switch controller

Mininet network

Window: 2 - Mininet

```
sdn@sdn-mn:~$ sudo mn --controller remote,ip=127.0.0.1  --
switch ovsk,protocols=OpenFlow13 --mac --ipbase=10.1.1.0/24
--topo tree,depth=1,fanout=4

*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Mininet network

- **--controller** remote,ip=127.0.0.1,port=6653
  - SDN Controller IP address and port
- **--switch** ovsk,protocols=OpenFlow13
  - OvS with OpenFlowc v1.3
- **--mac**
  - Automatically set host MAC addresses
- **--ipbase**=10.1.1.0/24
  - IP subnet for hosts
- **--topo** tree,depth=1,fanout=4
  - Network topology

OvS Dump Flows

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=34.792s, table=0, n_packets=75237,
n_bytes=4965646, priority=1,in_port="s1-
eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=34.783s, table=0, n_packets=242277,
n_bytes=12935495194, priority=1,in_port="s1-
eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=230.554s, table=0, n_packets=33,
n_bytes=2506, priority=0 actions=CONTROLLER:65535
```

OvS Add Flows

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 add-flow s1
dl_type=0x0800,nw_dst=10.1.1.3,actions=3

sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 add-flow s1
dl_type=0x0800,nw_dst=10.1.1.4,actions=4
```

OvS Dump Flows

Window: 3 - Open vSwitch

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows
s1

 cookie=0x0, duration=21.914s, table=0, n_packets=0, n_bytes=0,
ip,nw_dst=10.1.1.3 actions=output:"s1-eth3"
 cookie=0x0, duration=14.321s, table=0, n_packets=0, n_bytes=0,
ip,nw_dst=10.1.1.4 actions=output:4
 cookie=0x0, duration=86.348s, table=0, n_packets=75237,
n_bytes=4965646, priority=1,in_port="s1-
eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=86.339s, table=0, n_packets=242277,
n_bytes=12935495194, priority=1,in_port="s1-
eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=282.110s, table=0, n_packets=36,
n_bytes=2716, priority=0 actions=CONTROLLER:65535
```

Ping test

Window: 3 - Open vSwitch

```
mininet> h3 fping h1
10.1.1.1 is alive


packet in 1 00:00:00:00:00:03 ff:ff:ff:ff:ff:ff 3
packet in 1 00:00:00:00:00:01 00:00:00:00:00:03 1
packet in 1 00:00:00:00:00:03 00:00:00:00:00:01 3


mininet> h4 fping h1
10.1.1.1 is alive

packet in 1 00:00:00:00:00:04 ff:ff:ff:ff:ff:ff 4
packet in 1 00:00:00:00:00:01 00:00:00:00:00:04 1
packet in 1 00:00:00:00:00:04 00:00:00:00:00:01 4


mininet> h3 fping h4
10.1.1.4 is alive
```

Connectivity test

Window: 3 - Open vSwitch

```
mininet> h1 ip addr | grep "inet.*eth0"
    inet 10.1.1.1/24 brd 10.1.1.255 scope global h1-eth0

mininet> h3 ip addr | grep "inet.*eth0"
    inet 10.1.1.3/24 brd 10.1.1.255 scope global h3-eth0

mininet> h1 fping 10.1.1.3
10.1.1.3 is alive

mininet> h1 ping -c1 10.1.1.3
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.
64 bytes from 10.1.1.3: icmp_seq=1 ttl=64 time=0.089 ms

--- 10.1.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.089/0.089/0.089/0.000 ms
```

Webserver test

Window: 2 - Mininet

```
mininet> xterm h1

mininet> hterm h3
```

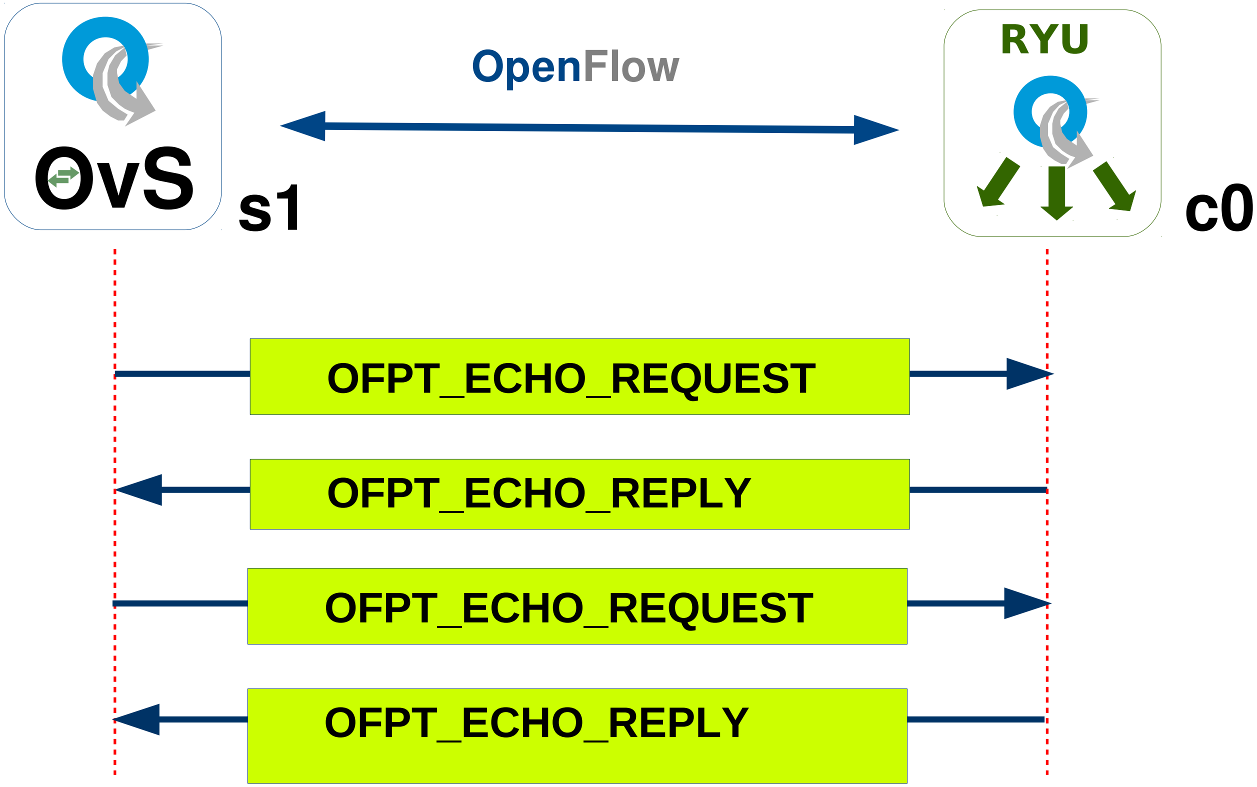
Node: h1

```
root@ryu-mn:~# python2 -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 …
10.1.1.3 - - [14/May/2020 12:30:34] "GET / HTTP/1.0" 200 -
```

Node: h3

```
root@ryu-mn:~# lynx 10.1.1.1
Directory listing for /
____________________________________________

     * .bash_history
     * .bash_logout
     * .bashrc
     * .cache/
```

OpenFlow Echo Request/Reply Message



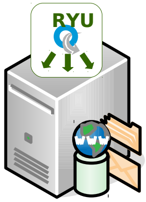
RESTful API

REST or a RESTful API

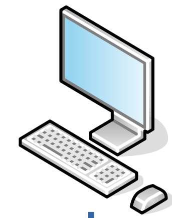
- Architectural style that defines a set of constraints to be used for creating web services
- WSGI provide RESTful web services as a NBI to SDN Controllers
- Data is often returned in JSON format
- All communication between the client carried out via REST API used only HTTP request

RESTful API

Server



Client



Send Request ←

```
Ethernet II
IP
TCP, Src Port: 8080, Dst Port: 39106, Seq: 1, Ack: 117, Len: 292
HTTP
GET http://127.0.0.1:8080/simpleswitch/mactable/1 HTTP/1.1\r\n
```

Send Response →

```
Ethernet II
IP
TCP, Src Port: 8080, Dst Port: 39106, Seq: 1, Ack: 117, Len: 292
HTTP
JavaScript Object Notation: application/json
{
    "00:00:00:00:00:03": 2,
    "d2:ff:c1:92:32:93": 2,
    "00:00:00:00:00:01": 1,
    "00:00:00:00:00:02": 2,
    "00:00:00:00:00:04": 2,
    "76:f6:5c:7b:8c:00": 2,
    "b6:62:01:2e:64:b6": 2
}
```

REST based Architecture

- POST:
  - used to create new resources
- GET:
  - used to read from a resource
- PUT:
  - used to update capabilities
- DELETE:
  - used to delete resources

REST example – Ryu Controller

Window: 1 - Ryu Controller

```
sdn@sdn-mn:~$ ryu-manager ryu.app.simple_switch_rest_13
loading app ryu.app.simple_switch_rest_13
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app ryu.app.simple_switch_rest_13 of
SimpleSwitchRest13
instantiating app ryu.controller.ofp_handler of OFPHandler
(3728) wsgi starting up on http://0.0.0.0:8080
```

REST example - Mininet

Window: 2 - Mininet

```
sdn@sdn-mn:~$ sudo mn --topo tree,depth=1,fanout=3 --switch ovsk
--controller remote,ip=127.0.0.1 --mac --ipbase=10.1.1.0/24

*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** 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:
mininet>
```

REST example - Mininet

Window: 2 - Mininet

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

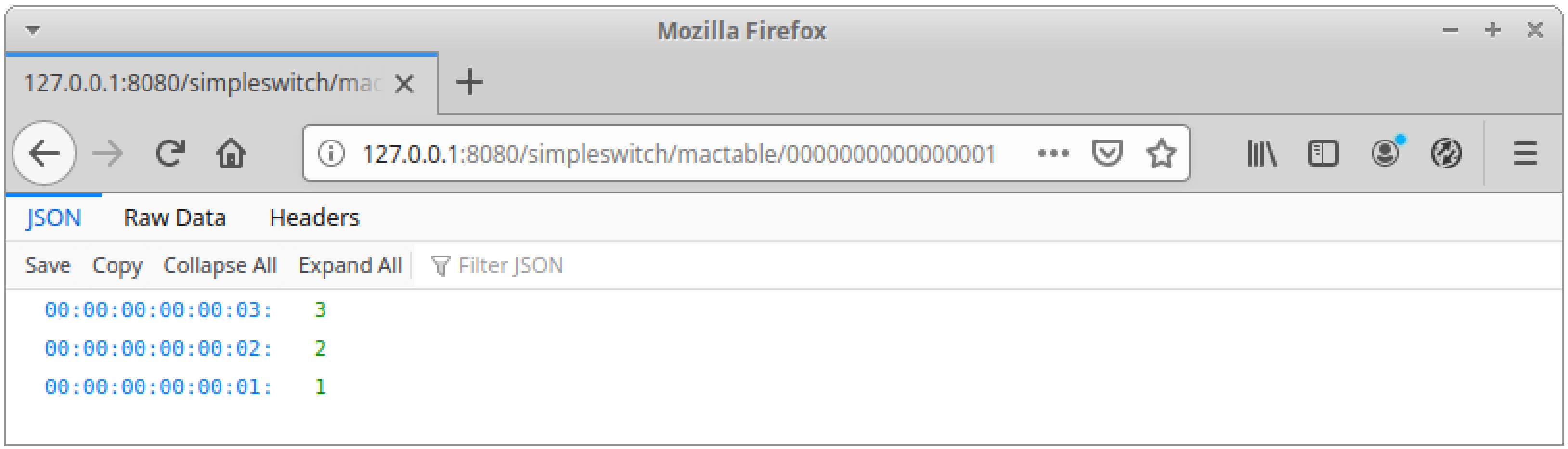
REST example - cURL

Window: 3 - cURL

```
sdn@sdn-mn:~$ curl -X GET
http://127.0.0.1:8080/simpleswitch/mactable/0000000000000001;echo
{"00:00:00:00:00:03": 3, "00:00:00:00:00:02": 2, "00:00:00:00:00:01": 1}
```

REST example – Web browser

- JSON output to web browser.



REST example – Python

Window: 4 - Python

```
sdn@sdn-mn:~$ cd example_scripts/
sdn@sdn-mn:~/example_scripts$ ./ryu_rest_client.py

Equivalent cURL command to interact with server from server
-----------------------------------------------------------

  $ curl http://127.0.0.1:8080/simpleswitch/mactable/0000000000000001

Naked JSON list received from server
------------------------------------

{'00:00:00:00:00:03': 3, '00:00:00:00:00:02': 2, '00:00:00:00:00:01': 1}

Breakdown list into individual element pairs
--------------------------------------------

MAC address          Port
-----------          ----

00:00:00:00:00:01      1
00:00:00:00:00:02      2
00:00:00:00:00:03      3
```

Laboratory

Python RESTful client

Exercise:

- Open the **ryu_rest_client.py** program in the example scripts and consider how it works

```
impor...
import js...

ip_address = '12...
port = '8080'
rest_path = '/simpleswi...
switch = '0000000000000001'
pad = 20

# Get the JSON output from the RESTfu...
url = 'http://{}:{}{}/{}'.format(ip_addr...

response = requests.get(url)
response_bytes = response.content
response_str = response_bytes.decode('utf-8')

if (response.ok):
    json_dict = json.loads(response_str)
else:
    print('There has been a problem connecting to REST API')
    exit(1)


# Print equivalent 'curl' command
heading = 'Equivalent cURL command to interact with server from server'
print('\n{}'.format(heading))
print('-' * len(heading), '\n')

print('  $ curl {}'.format(url))

# Print out the naked list received
heading = 'Naked JSON list received from server'
print('\n{}'.format(heading))
print('-' * len(heading), '\n')

print(json_dict)

# Breakdown list into individual element pairs
heading = 'Breakdown list into individual element pairs'
print('\n{}'.format(heading))
print('-' * len(heading), '\n')

print ('MAC address {} Port'.format(' ' * 7))
print ('{} {} {}'.format('-' * 11, ' ' * 7, '-' * 4))
...t()
...orted(json_dict.keys()):
... = pad - len(mac)
...{}'.format(mac, ' ' * pad_digits, json_dict[mac]))
```

REST example – Ryu Controller

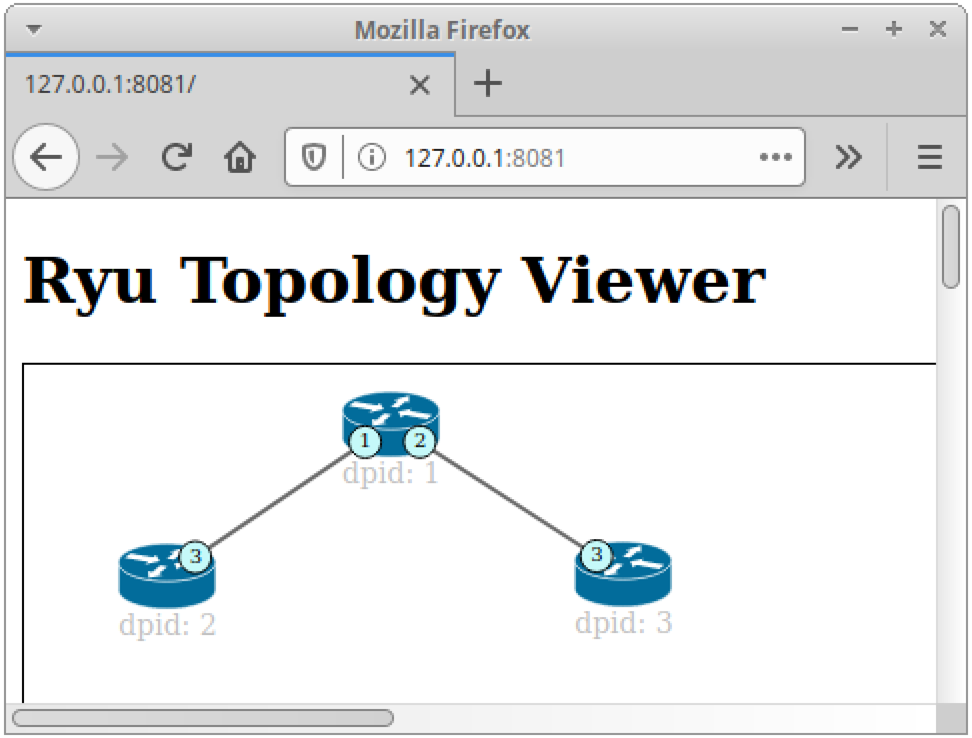
Window: 1 - Ryu Controller

```
sdn@sdn-mn:~$ ryu-manager ryu.app.simple_switch_rest_13
loading app ryu.app.simple_switch_rest_13
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app ryu.app.simple_switch_rest_13 of
SimpleSwitchRest13
instantiating app ryu.controller.ofp_handler of OFPHandler
(3728) wsgi starting up on http://0.0.0.0:8080
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3
(9873) accepted ('127.0.0.1', 49302)
127.0.0.1 - - [14/May/2020 12:41:10] "GET
/simpleswitch/mactable/0000000000000001 HTTP/1.1" 200 180
0.001291
```

Topology viewer

Window: 1 - Ryu Controller

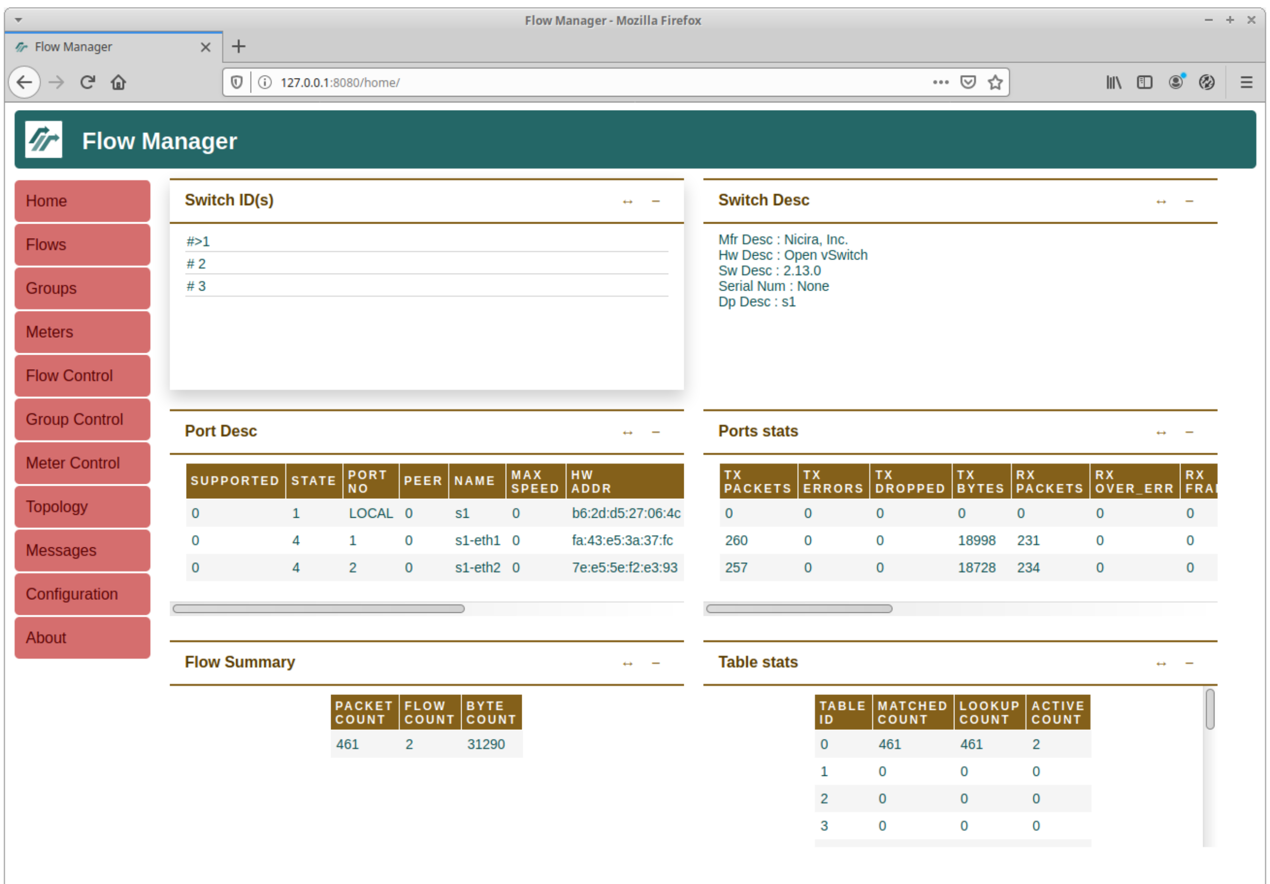
```
sdn@sdn-mn:~$ ryu-manager --ofp-tcp-listen-port 6653
--wsapi-port 8081 --observe-links --app-lists
ryu.app.simple_switch_13 ryu.app.ofctl_rest
ryu.app.gui_topology.gui_topology


(14947) wsgi starting up on http://0.0.0.0:8081
```



Flowmanager

Window: 1 - Ryu Controller

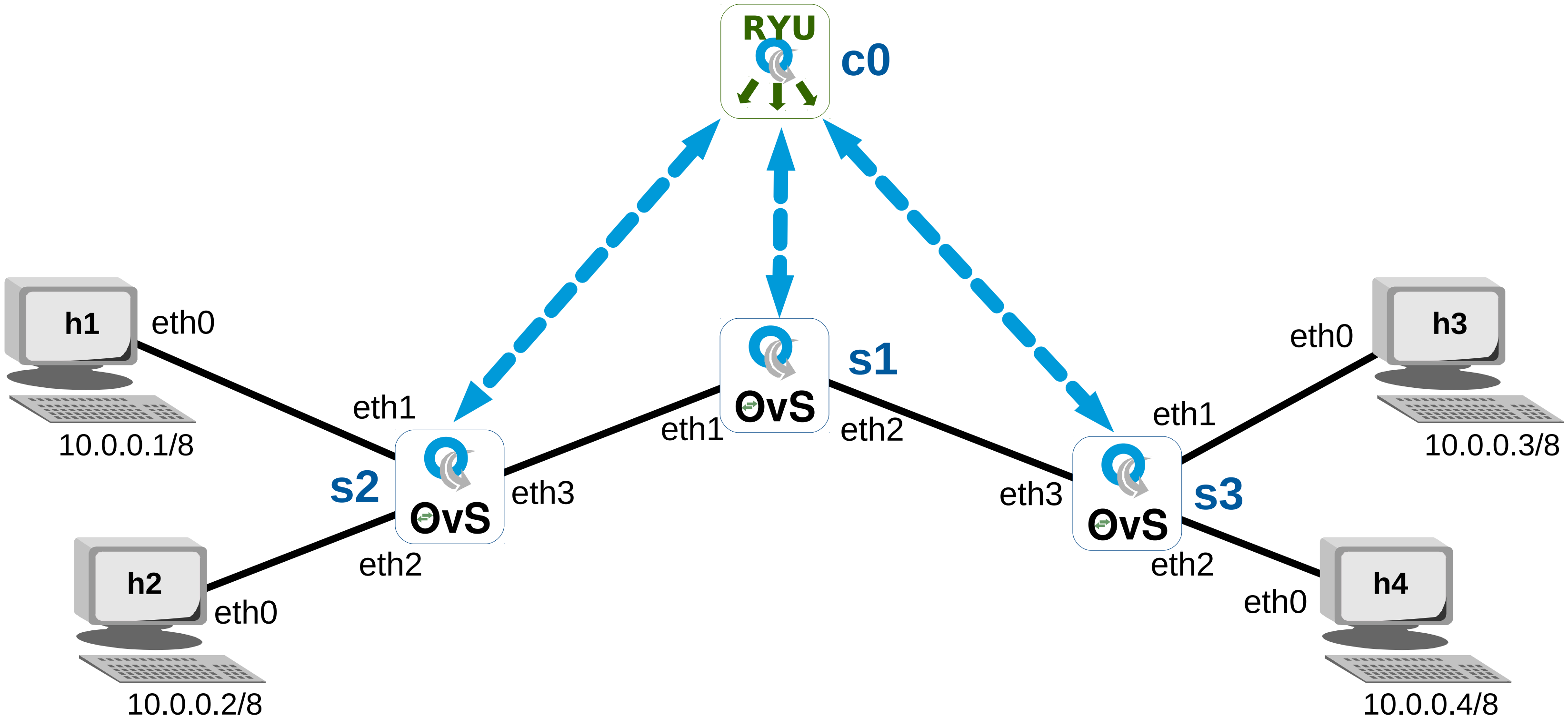
```
sdn@sdn-mn:~$ ryu-manager --observe-links --app-lists
~/flowmanager/flowmanager.py ryu.app.simple_switch_13
instantiating app /home/sdn/flowmanager/flowmanager.py of FlowManager
(15142) wsgi starting up on http://0.0.0.0:8080
```



Laboratory

Build test network

Exercise: Build the following test network



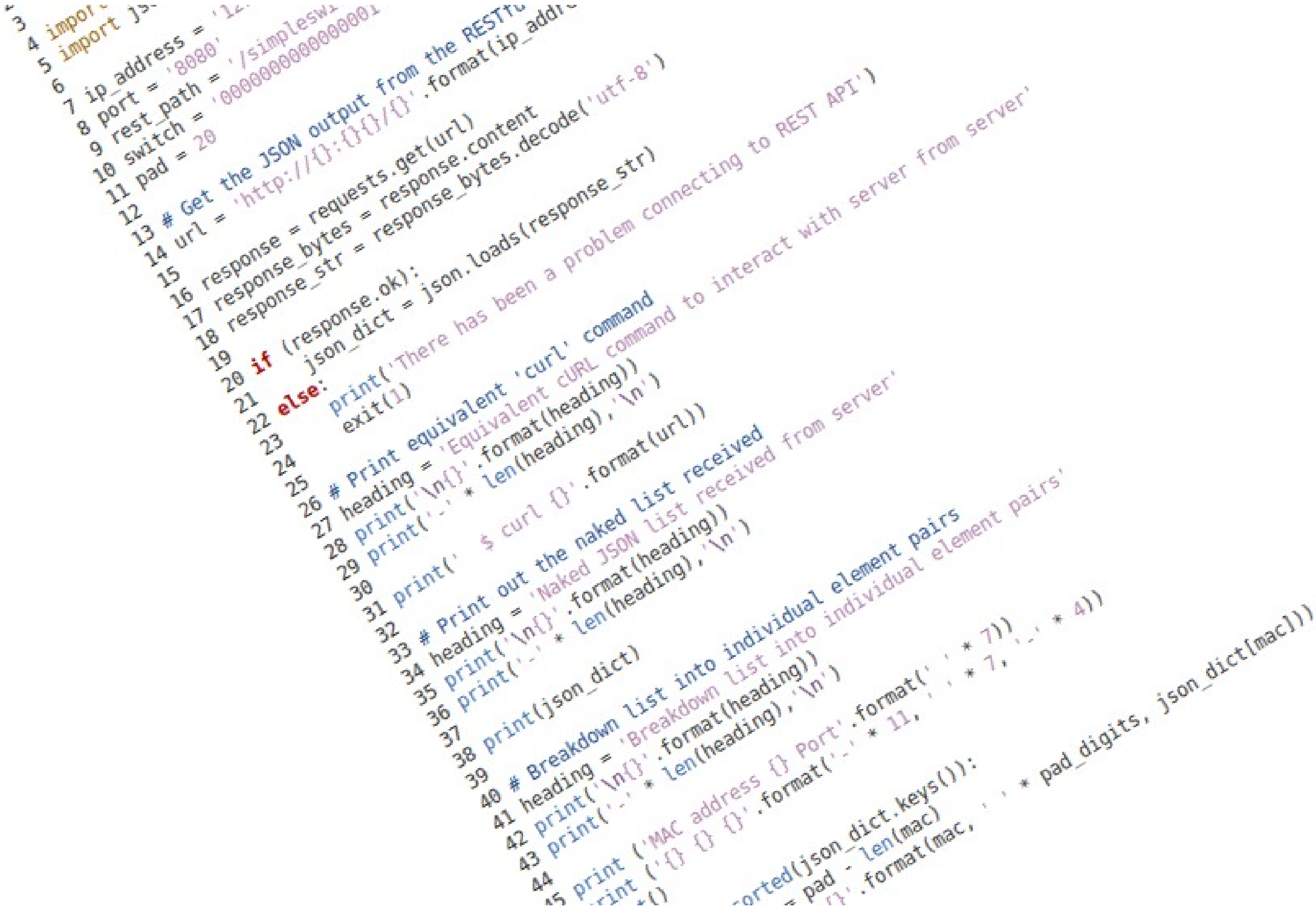
- Build the network shown from Chapter 8 of the notes
- Follow the instructions in Chapter 9

Laboratory

Python RESTful client #2

Exercise:

- Open the **ryu_curl_post.py** program in the example scripts and consider how it works



Custom mininet topologies

```
> sudo mn
```

Mininet examples

Window: 2 - Mininet

```
sdn@sdn-mn:~$ ls ~/mininet/examples
baresshd.py       controlnet.py        mobility.py         README.md
bind.py           cpu.py               multilink.py        scratchnet.py
clustercli.py     emptynet.py          multiping.py        scratchnetuser.py
clusterdemo.py    hwintf.py            multipoll.py        simpleperf.py
clusterperf.py    __init__.py          multitest.py        sshd.py
cluster.py        intfoptions.py       natnet.py           test
clusterSanity.py  limit.py             nat.py              tree1024.py
consoles.py       linearbandwidth.py   numberedports.py    treeping64.py
controllers2.py   linuxrouter.py       popenpoll.py        vlanhost.py
controllers.py    miniedit.py          popen.py
```

Mininet custom repository

Window: 2 - Mininet

```
sdn@sdn-mn:~$ ls ~/mininet/custom
README  topo-2sw-2host.py

sdn@sdn-mn:~$ cat ~/mininet/custom/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
```

Mininet custom repository

Editor

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    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 )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )


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

Mininet custom repository

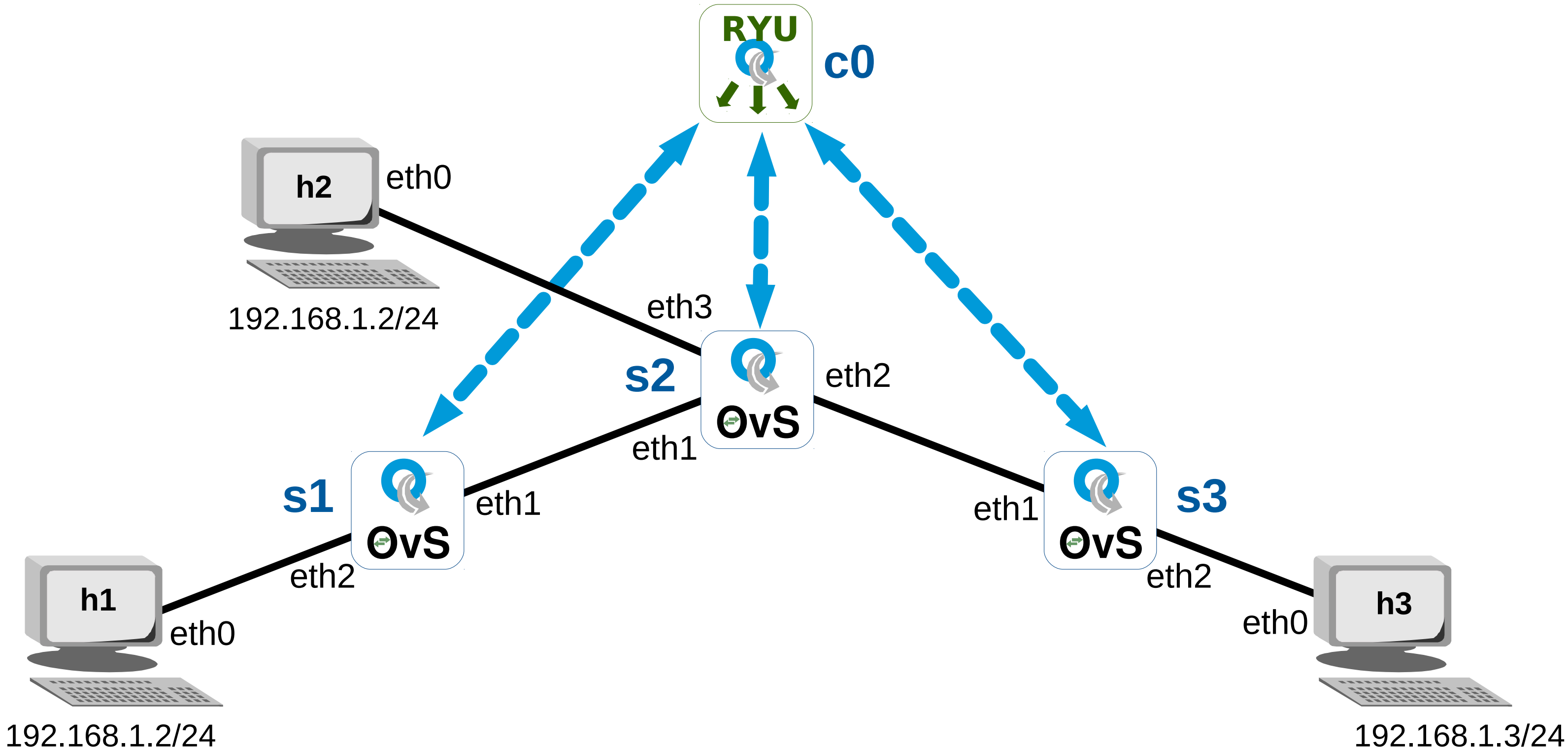
Window: 2 - Mininet

```
sdn@sdn-mn:~$ sudo mn --custom ~/example_scripts/topo-2sw-2host.py --topo mytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3 s4
*** Adding links:
(h1, s3) (s3, s4) (s4, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s3 s4 ...
*** Starting CLI:



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
```

Laboratory

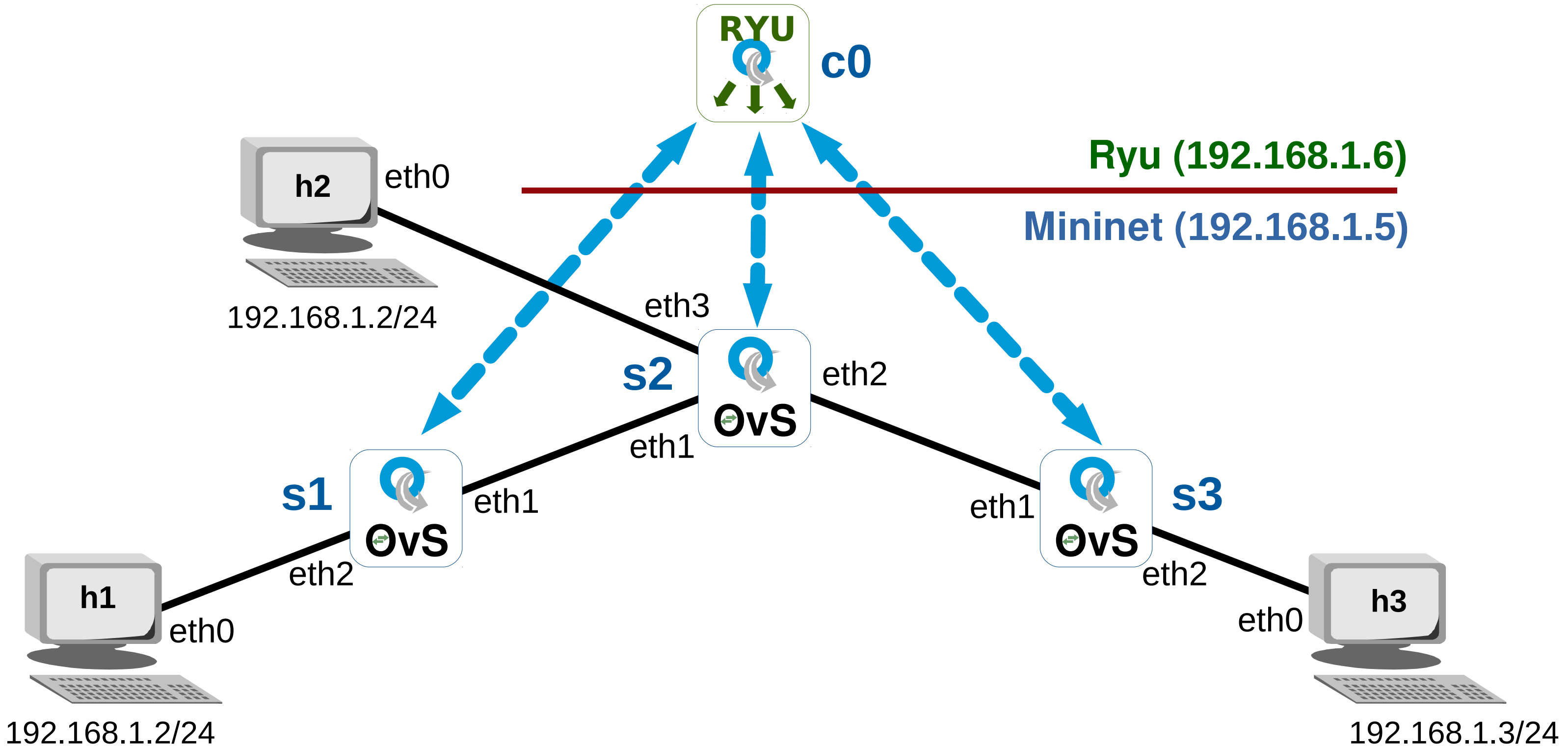
Custom Topology

Exercise: Build the custom topology



- Build the network shown from Chapter 10 of the notes

Exercise: Separate functions



- Instantiate an additional VM and run the Ryu Controller in one and Mininet in the other as per Chapter 11 of the notes

Developing
Ryu SDN Controller
applications

Base classes and library

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
```

Application Manager

- app_manager
  - Main entry point for the application
  - The central management of Ryu applications and includes the following classes:
    - **RyuApp**: the base class for Ryu applications
    - **AppManager**: the management class

OpenFlow event definition

- OpenFlow event definitions are handled by the ‘**ofp_event**’ dispatcher
- Includes the base class of the OpenFlow event class for events with the following:

Attribute	**Description**
msg	An object which describes the corresponding OpenFlow message.
msg.datapath	A datapath instance description of the OpenFlow switch the packet came from.
timestamp	Timestamp of when this event was generated.

Handler

- Python decorator, **set_ev_cls**

Negotiation phase	Description
HANDSHAKE_DISPATCHER	Sending and waiting for hello message
CONFIG_DISPATCHER	Version negotiated and sent features-request message
MAIN_DISPATCHER	Switch-features message received and message
DEAD_DISPATCHER	Disconnect from the peer. Or disconnecting due to some unrecoverable errors.

Frame and packet processing

- **packet, ethernet, ether_types**:
  - packet processing library that includes:
    - Packet decoder/encoder class
    - Ethernet header encoder/decoder class
    - Alookup list of Ethernet type values like
      - ETH_TYPE_IP = 0x0800
      - ETH_TYPE_ARP = 0x0806
      - etc..

Application class

```
class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
```

- Derived from the app_manager **RyuApp** class
- The OpenFlow version is defined and a constructor method that is called when an object is instantiated using the definitions found within the class

Event methods

- Event handler methods
  - **switch_features_handler**
  - **add_flow**
  - **packet_in_handler**

switch_features_handler

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
```

- **set_ev_cls** decorator links the
  - **switch_features_handler** method
- with the **EventOFPSwitchFeatures** event
- This method installs table-miss flow entry in each switch

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=62.808s, table=0, n_packets=66,
n_bytes=8320, priority=0 actions=CONTROLLER:65535
```

add_flow method

- The **add_flow** method accepts requests from other methods to add flows in switches
- To do that it expects to be supplied with:
  - Datapath ID
  - Priority
  - Match details
  - Actions
  - Buffer_id

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=23.971s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
```

packet_in_handler

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

- The **_packet_in_handler** method is associated with the decorator **set_ev_cls** which calls the method for packet in **EventOFPPacketIn** events
- It:
  - Inspects incoming packets from switches
  - Updates the MAC table
    - MAC to port dictionary pairing source MAC with the incoming port
    - It checks if destination MAC is present in the dictionary
      - If so; installs a flow
      - Else; it instructs the switch to flood to all its remaining ports

Test the laboratory

Window: 1 - Ryu Controller

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/L2_simple_switch_13.py
loading app ./example_scripts/L2_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/L2_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Window: 2 - Mininet

```
sdn@sdn-mn:~$ sudo mn --controller remote,ip=127.0.0.1 -mac
 --switch ovsk,protocols=OpenFlow13 --ipbase=10.1.1.0/24 --topo single,2
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

Ping and check flows

Window: 2 - Mininet

```
mininet> pingall

*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Window: 2 - Mininet

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=73.590s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:1

 cookie=0x0, duration=73.584s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:2

 cookie=0x0, duration=82.628s, table=0, n_packets=17, n_bytes=1298, priority=0
actions=CONTROLLER:65535
```

Output

- Note the matches for the two flow entries
  - **Flow 1**
    - in_port=2
    - dl_src=00:00:00:00:00:02
    - dl_dst=00:00:00:00:00:01
  - **Flow 2**
    - in_port=1
    - dl_src=00:00:00:00:00:01
    - dl_dst=00:00:00:00:00:02

Laboratory

Flow parameters

Exercise: Flow parameters

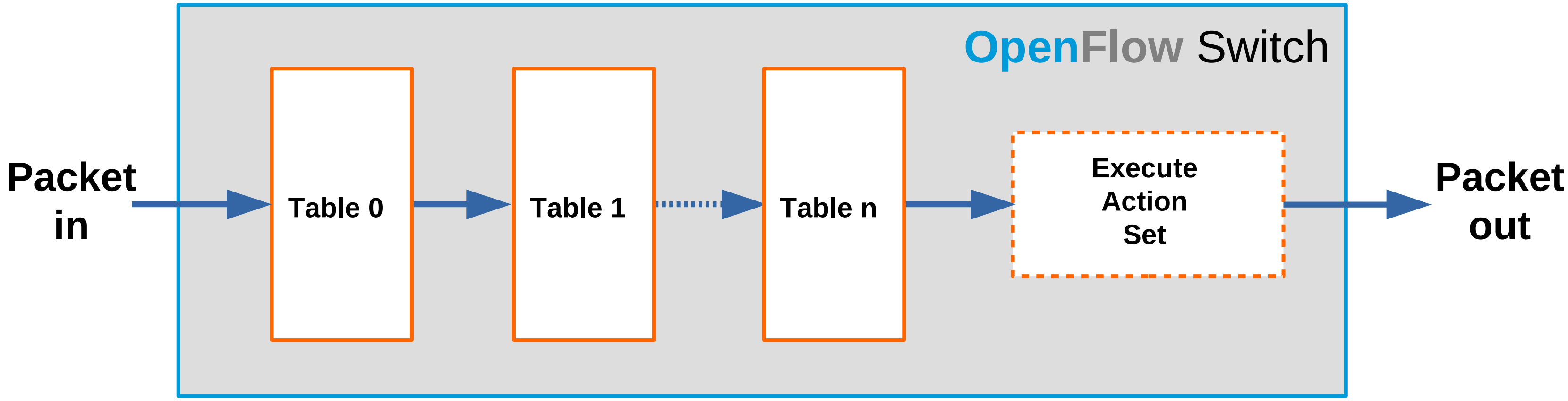
- Work through Chapter 13 of the notes
- Adjust
  - **simple_switch_13.py**
- to create a new application
  - **priority_simple_switch_13.py**
- Which prioritises particular types of traffic over other types of traffic

OpenFlow
Pipeline processing

```
> sudo mn
```

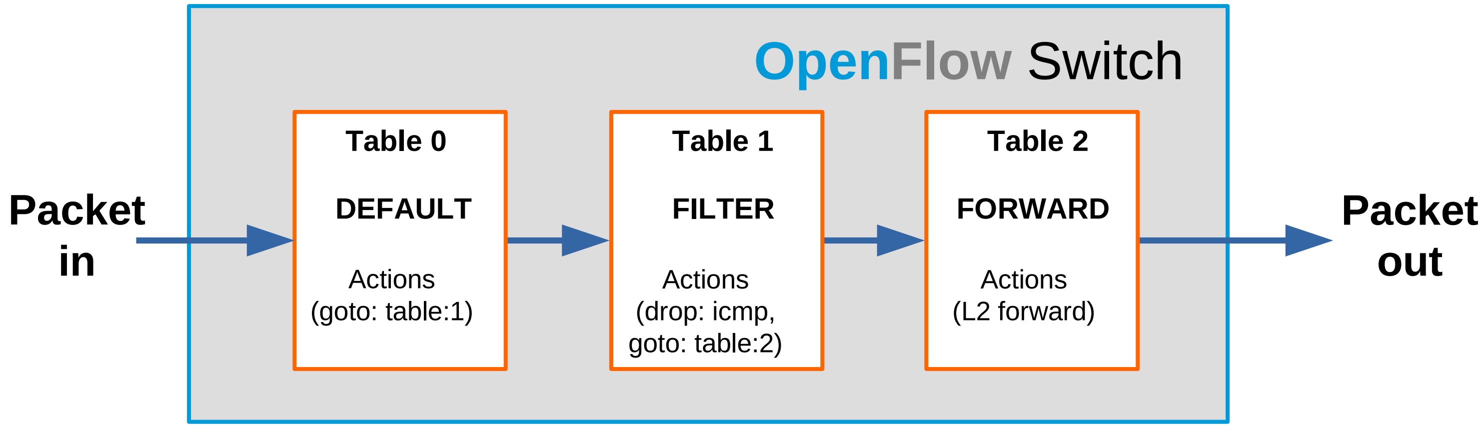
OpenFlow pipeline

- The OpenFlow pipeline in a switch contains multiple flow tables, each flow table containing multiple flow entries



- An OpenFlow switch is required to have at least one flow table, and can optionally have more flow tables
- An OpenFlow switch with only a single flow table is a case of a simplified pipeline process

Example OpenFlow pipeline



- Ingress packets arriving are always checked against, the DEFAULT, table=0
- All packets are then forwarded to be matched by the flow rules in, the FILTER, table=1
  - Rule to drop ICMP packets and forward all other packets to, the FORWARD, table=2
- Surviving packets are L2 forwarded

Laboratory

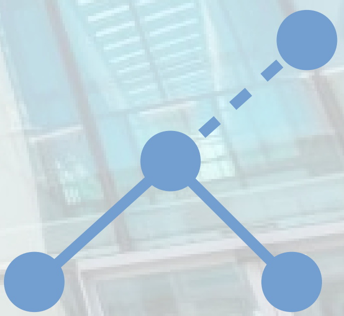
Pipeline processing

Exercise: Pipeline Processing

- Work through chapter 14.1 of the notes
- Adjust
  - **simple_switch_13.py**
- to create a new application
  - **pp_simple_switch_13.py**
- Which generates pipeline processing tables

OpenFlow
Group Tables

```
> sudo mn
```



Group tables

OpenFlow Group table

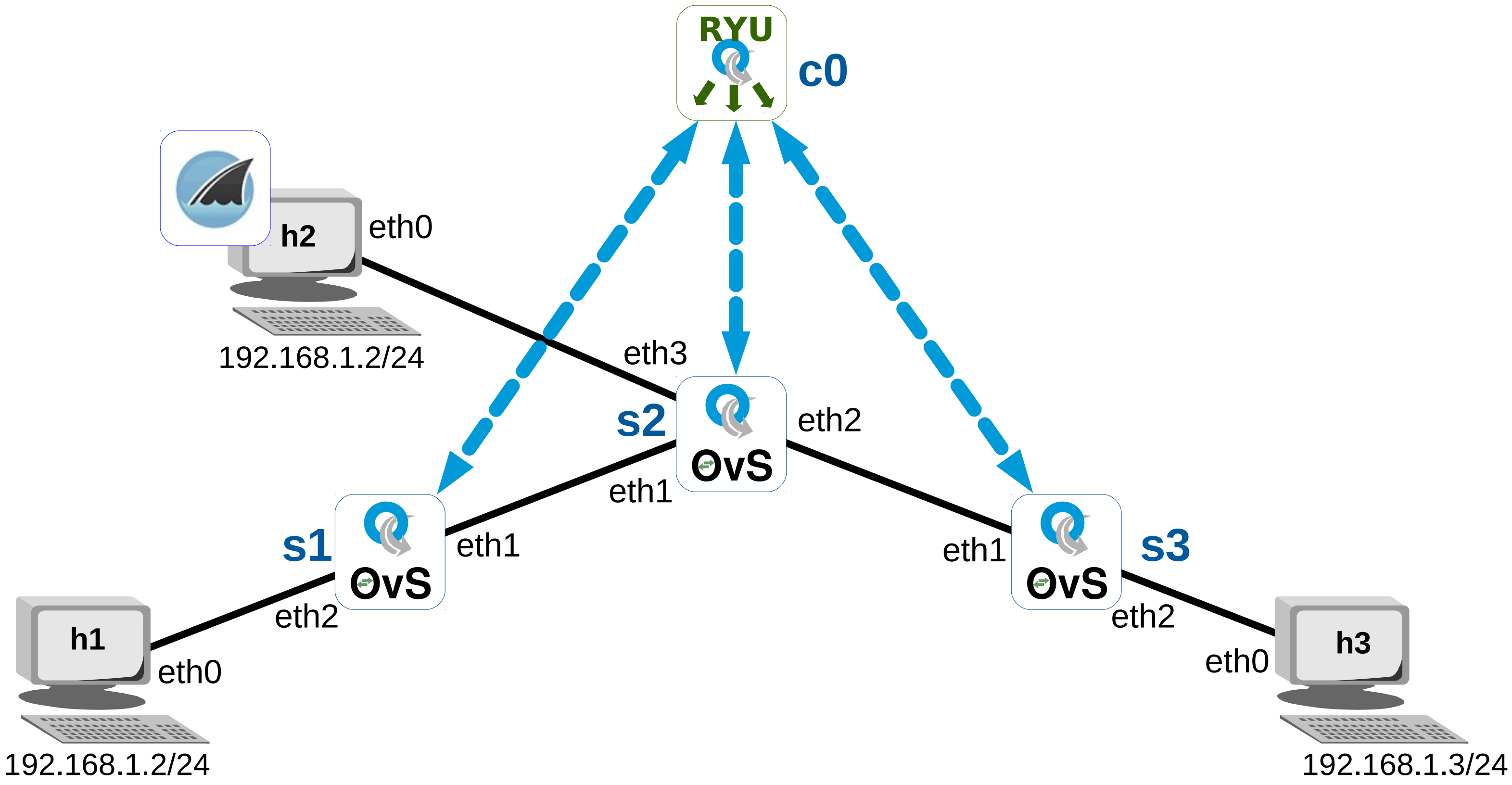
Group ID	Group type	Counters	Action buckets
0	all select indirect fast-failover	duration=764.169s n_packets=242538 n_bytes=13851836796	in_port=1 dl_src=00:00:00:00:00:01 dl_dst=00:00:00:00:00:02

- Group tables consist of group entries. A flow entry can point to a group as well as tables and this enables OpenFlow to represent additional methods of forwarding like **select** and **all**
- A group can include many buckets, and in turn, a bucket can have a set of actions (set, pop, or output)
- Sample use cases include the ability to copy a packet to ALL buckets and process it. This has obvious application for traffic mirroring for monitoring or a load balancer function can be achieved by forwarding packets to 1 bucket from many buckets and processing it

Laboratory

Ryu Packet Sniffer

Exercise: Packet Sniffer



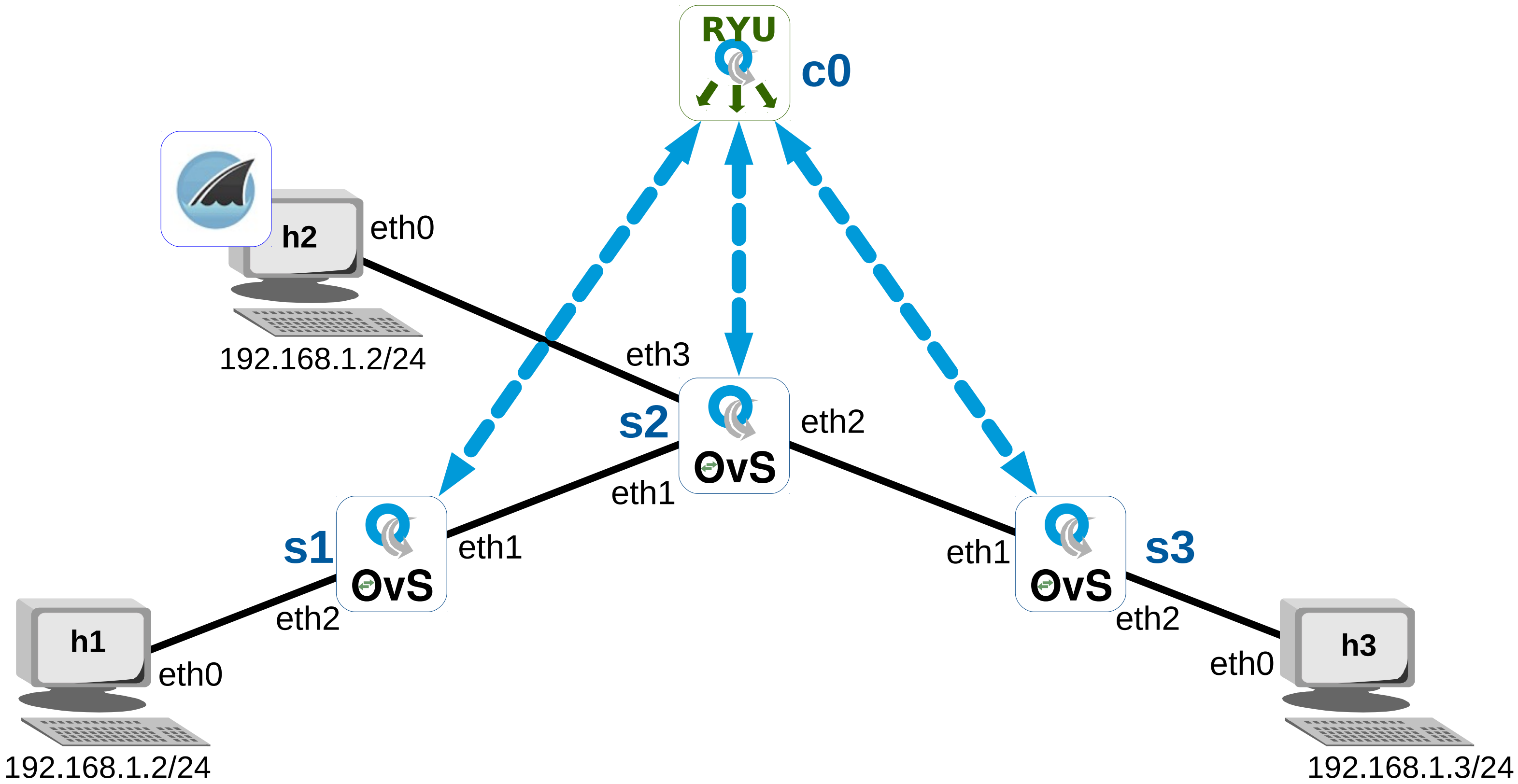
Exercise: Packet Sniffer

- Work through chapter 14.2 of the notes
- Adjust
  - **simple_switch_13.py**
- to create a new application
  - **gpt_simple_switch_13.py**
- Which sniffs all traffic to host 2 where it can be analysed by Wireshark

Laboratory

Ryu Proxy ARP

Exercise: Proxy ARP



Exercise: Proxy ARP

- Work through chapter 14.3 of the notes
- Adjust
  - **simple_switch_13.py**
- to create a new application
  - **pa_simple_switch_13.py**
- Which allows the Ryu controller to answer for ARP queries directly, from a table, instead of forwarding to all ports
- This reduces ARP traffic on the network

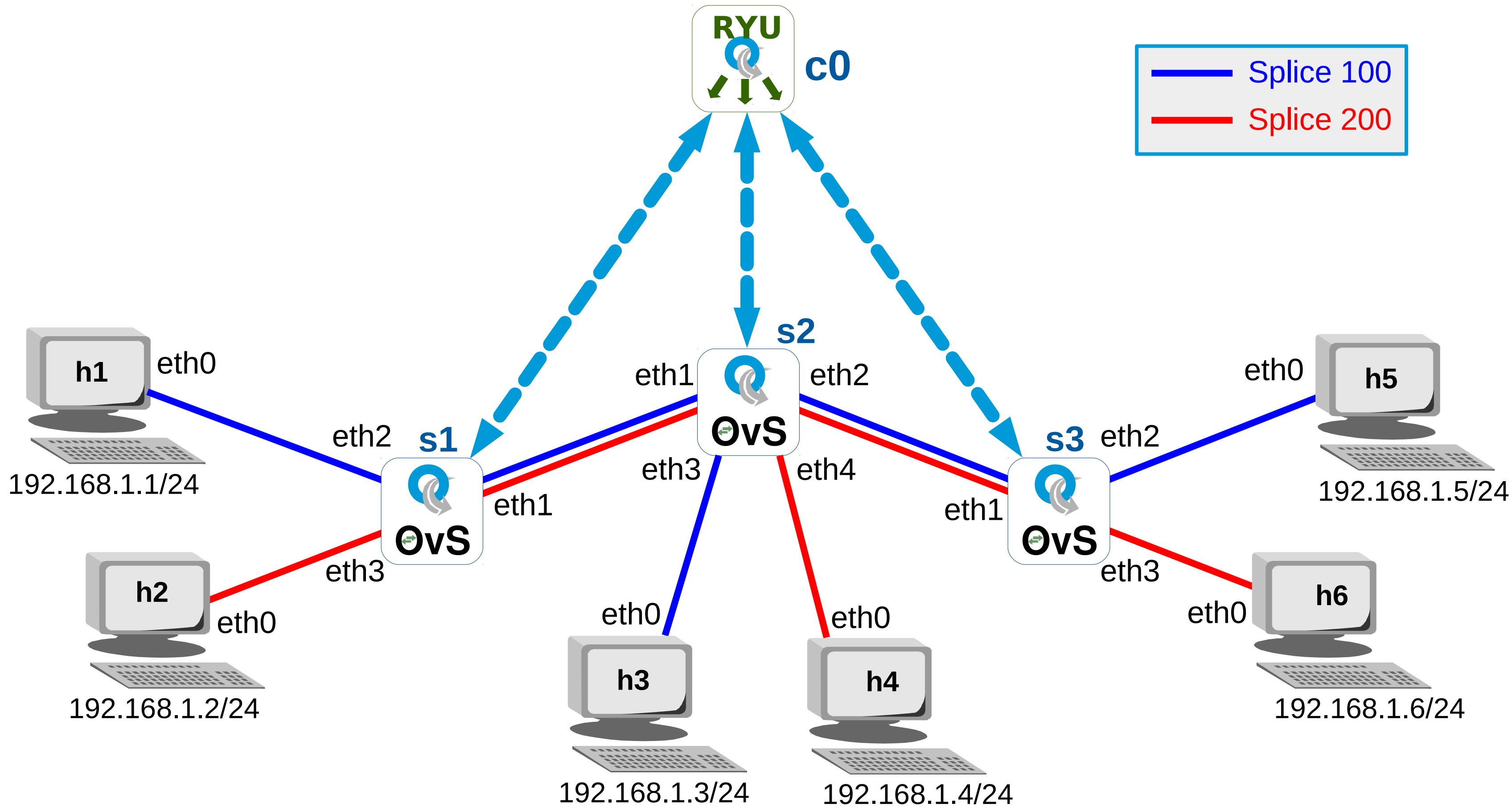
Splitting Domains

- Flowspace splicing
- VLANs

Splitting domains, flow splicing or VLAN

- **Flow splicing**
  - SDN Controller has control of all switches so the interfaces on the switches can be segregated into flow splices
  - There are no indicators in the packets to identify the flow splice the packet belongs to
- VLAN
  - Each packet is assigned a VLAN tag to identify which network it belongs to

Flowspace splicing



Laboratory

Flow splicing

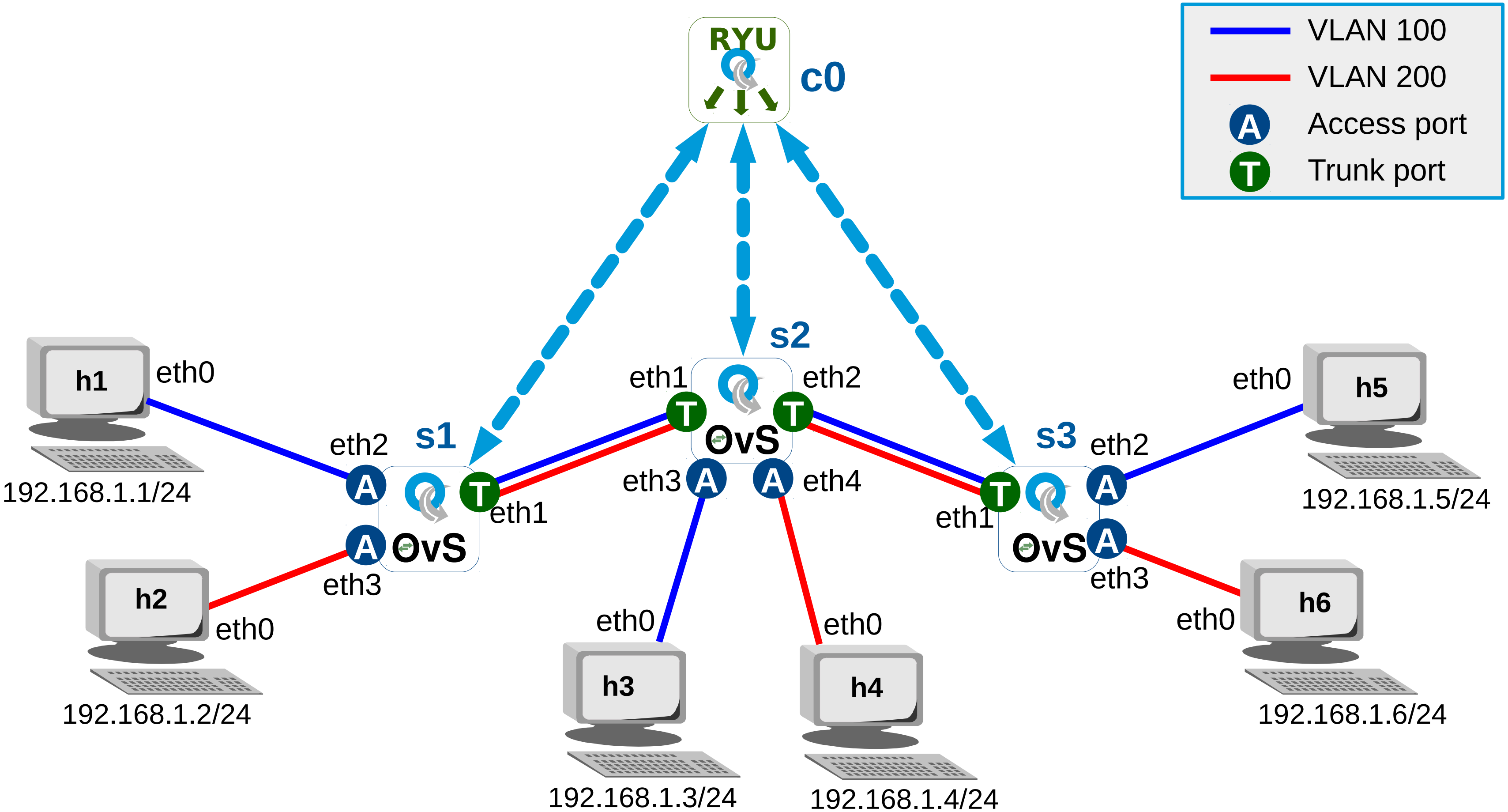
Exercise: Network splicing

- Work through the chapter 15.1 of the notes
- Adjust
  - **simple_switch_13.py**
- to create a new application
  - **splice_simple_switch_13.py**
- Which allows the Ryu controller separate traffic into two different splices within the domain

Laboratory

VLANs

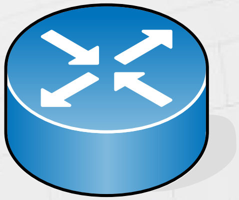
VLANs


RYU
c0
s1
s2
s3
OvS
OvS
OvS
h1
eth0
192.168.1.1/24
h2
eth0
192.168.1.2/24
h3
eth0
192.168.1.3/24
h4
eth0
192.168.1.4/24
h5
eth0
192.168.1.5/24
h6
eth0
192.168.1.6/24
eth2
eth3
eth1
eth1
eth2
eth3
eth4
eth1
eth2
eth3
VLAN 100
VLAN 200
Access port
Trunk port

Exercise: VLANs

- Work through the chapter 15.2 of the notes
- Adjust
  - **simple_switch_13.py**
- to create a new application
  - **VLAN_simple_switch_13.py**
- Which allows the Ryu controller to:
  - insert VLAN tags at ingress ports
  - remove them at egress ports
  - obey the tags on internal (trunk) ports

Simple Layer 3 and 4 switches



L3 and L4 switching

- To this point, matches and flows have been based on L2 MAC addressing and in ports
- OpenFlow is not limited to L2 fields and can switch based on a L3 or L4 data
- The final two sections demonstrates how the L2 simple switch can be modified to switch based on IP addressing at L3, the network layer and segment protocols at L4, the transport layer

Laboratory

Layer 3 switching

Exercise: Layer 3 switching

- Work through the chapter 16.1 of the notes
- Adjust
  - **simple_switch_13.py**
- to create a new application
  - **L3_simple_switch_13.py**
- Which allows the Ryu controller to switch based on IP addressing information

Laboratory

Layer 4 switching

Exercise: Layer 4 switching

- Work through the chapter 16.2 of the notes
- Adjust
  - **simple_switch_13.py**
- to create a new application
  - **L4_simple_switch_13.py**
- Which allows the Ryu controller to switch based on UDP and TCP port number information

Bóthar Chill Chainnigh, Ceatharlach

Suiomh Gréasáin: www.itcarlow.ie

Guthán: (059) 917 5000

R-phost: diarmuid.obriain@itcarlow.ie

INSTITUTE of TECHNOLOGY CARLOW

Institiúid Teicneolaíochta Cheatharlach

Dr Diarmuid Ó Briain

Innealtóir Cairte

Léachtóir

CELEBRATING 50 YEARS 1970-2020

Athena SWAN Bronze Award

GENDER CHARTER

engcore

advancing technology

References

- Ryu website:
  - https://osrg.github.io/ryu/
  - https://github.com/osrg/ryu/tree/master/ryu
- Ryu example apps: https://github.com/osrg/ryu/tree/master/ryu/app
- Documentation: https://ryu.readthedocs.io/en/latest/index.html
- Development mail-list:
  - https://sourceforge.net/projects/ryu
  - To post to this list, send message to: ryu-devel@lists.sourceforge.net
  - General information about the mailing list is at: https://lists.sourceforge.net/
- Mininet website: http://mininet.org
- Flowmanager: https://martimy.github.io/flowmanager/
- Udemy: https://github.com/knetsolutions/learn-sdn-with-ryu

Software versions

• Ubuntu/Xubuntu	20.04
• Ryu	4.32
• Mininet	2.3.0d6
• Open vSwitch	2.13.0
• iperf:	2.0.13
• mtr	0.93
• Wireshark	3.2.3
• Mozilla Firefox	76.0.1
• RESTer	4.1.1

Learning outcomes

- ✔Introduction to SDN
- ✔SDN Architecture
- ✔Build a Ryu SDN testbed
- ✔Build a Mininet test network
- ✔The Open vSwitch
- ✔OpenFlow communications
- ✔RESTful API
- ✔Building a simple test network
- ✔Ryu Framework
- ✔Custom Topologies
- ✔Custom script to Ryu remote controller
- ✔Developing Ryu applications
- ✔Flow parameters
- ✔OpenFlow pipeline processing
- ✔Splitting domains
- ✔Building a simple L3 and L4 switches