Virtual Switch-Openflow-OVS-Mininet

Guest Lecture – Pertamina University

Session 2

Ardimas Purwita
ardimas.andi@binus.ac.id

Objectives

- Understand about the context behind SDN: building blocks
- Quick introduction to SDN

Virtual Switch: Linux Bridge

- A Linux bridge behaves like a network switch.
- It forwards packets between interfaces that are connected to it.
- It's usually used for forwarding packets on routers, on gateways, or between VMs and network namespaces on a host.
- It also supports STP, VLAN filter, and multicast snooping.

- To sum up, other than L2-switch capability (i.e., L3-switch), we pretty much needs a third-party tool to add additional functionality to the Linux bridge

Ref: https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking#bridge

Important Commands

- To create virtual bridge

  # ip link add <virtual bridge name> type bridge

  ip link add v-bridge type bridge

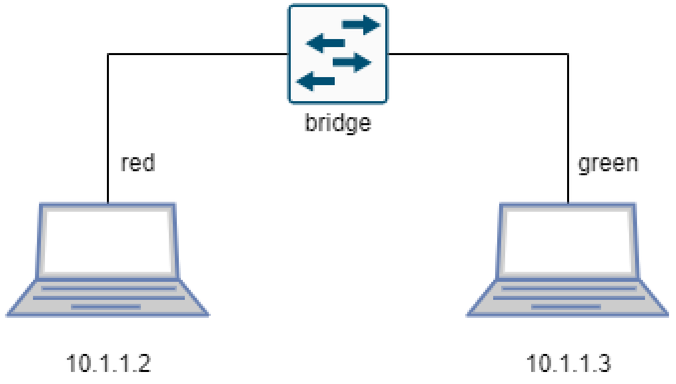
- To attach a veth to virtual bridge

  # ip link set <veth name> master <virtual bridge name>

  ip link set veth-br master v-bridge

Examples of Linux Bridge acting as a normal L2 switch

```
#!/bin/sh

# create netns red and green
ip netns add red
ip netns add green

# create veth
ip link add veth-r type veth peer name veth-br-r
ip link add veth-g type veth peer name veth-br-g

# create v-bridge
ip link add v-bridge type bridge

# attach veth
ip link set veth-r netns red
ip link set veth-g netns green
ip link set veth-br-r master v-bridge
ip link set veth-br-g master v-bridge

# activate veth
ip netns exec red ip link set dev veth-r up
ip netns exec green ip link set dev veth-g up
ip link set dev veth-br-r up
ip link set dev veth-br-g up
ip link set dev v-bridge up

# assign ip address
ip netns exec red ip a add 10.1.1.2/24 dev veth-r
ip netns exec green ip a add 10.1.1.3/24 dev veth-g

# test
ip netns exec red ping 10.1.1.3 -c 3
```

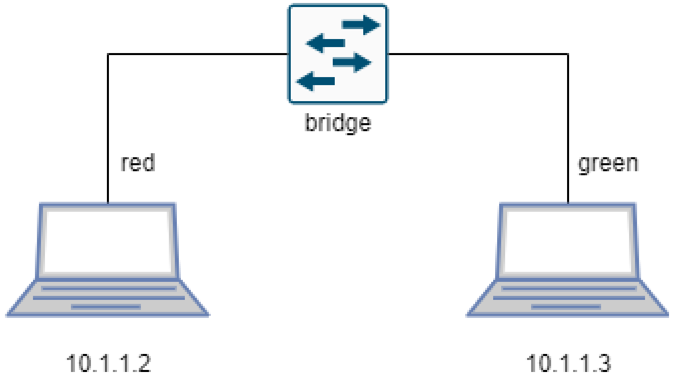


Note that I don’t assign any IP to the bridge

Examples of Linux Bridge acting as a normal L2 switch

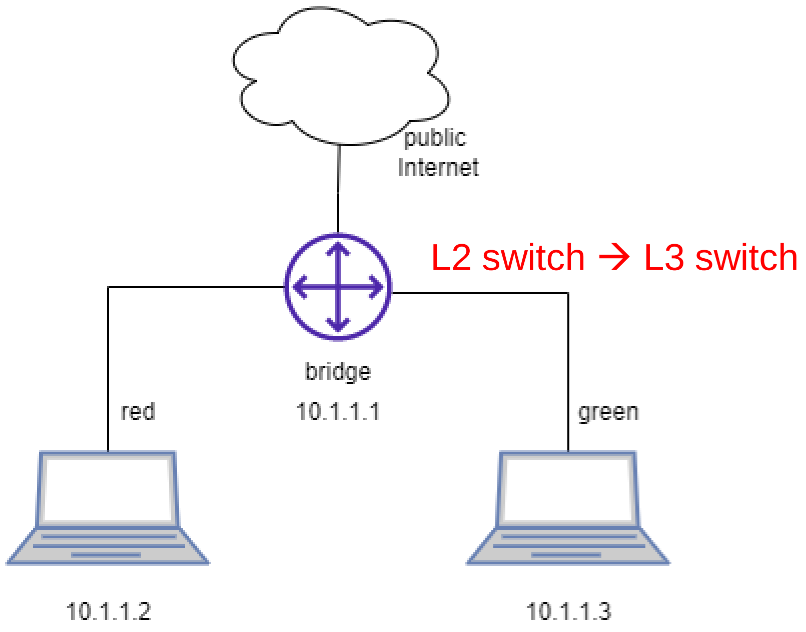
```
#!/bin/sh

# create netns red and green
ip netns add red
ip netns add green

# create veth
ip link add veth-r type veth peer name veth-br-r
ip link add veth-g type veth peer name veth-br-g

# create v-bridge
ip link add v-bridge type bridge

# attach veth
ip link set veth-r netns red
ip link set veth-g netns green
ip link set veth-br-r master v-bridge
ip link set veth-br-g master v-bridge

# activate veth
ip netns exec red ip link set dev veth-r up
ip netns exec green ip link set dev veth-g up
ip link set dev veth-br-r up
ip link set dev veth-br-g up
ip link set dev v-bridge up

# assign ip address
ip netns exec red ip a add 10.1.1.2/24 dev veth-r
ip netns exec green ip a add 10.1.1.3/24 dev veth-g

# test
ip netns exec red ping 10.1.1.3 -c 3
```

Note that I don’t assign any IP to the bridge



Examples of Linux Bridge acting as a normal L3 switch

```
#!/bin/sh

# create netns red and green
ip netns add red
ip netns add green

# create veth
ip link add veth-r type veth peer name veth-br-r
ip link add veth-g type veth peer name veth-br-g

# create v-bridge
ip link add v-bridge type bridge

# attach veth
ip link set veth-r netns red
ip link set veth-g netns green
ip link set veth-br-r master v-bridge
ip link set veth-br-g master v-bridge

# activate veth
ip netns exec red ip link set dev veth-r up
ip netns exec green ip link set dev veth-g up
ip link set dev veth-br-r up
ip link set dev veth-br-g up
ip link set dev v-bridge up

# assign ip address
ip netns exec red ip a add 10.1.1.2/24 dev veth-r
ip netns exec green ip a add 10.1.1.3/24 dev veth-g

# test
ip netns exec red ping 10.1.1.3 -c 3

# enable ipv4 forwarding
sysctl -w net.ipv4.ip_forward=1

# assign ip address to the v-bridge
ip a add 10.1.1.1/24 dev v-bridge

# set routing table in the red and green
ip netns exec red ip route add default via 10.1.1.1
ip netns exec green ip route add default via 10.1.1.1

# set NAT
iptables -t nat -A POSTROUTING -s 10.1.1.0/24 -j
MASQUERADE

# test
Ip netns exec red ping 1.1.1.1 –c 3
```



Note that we need a third-party tool to program (e.g., iptables) the virtual bridge

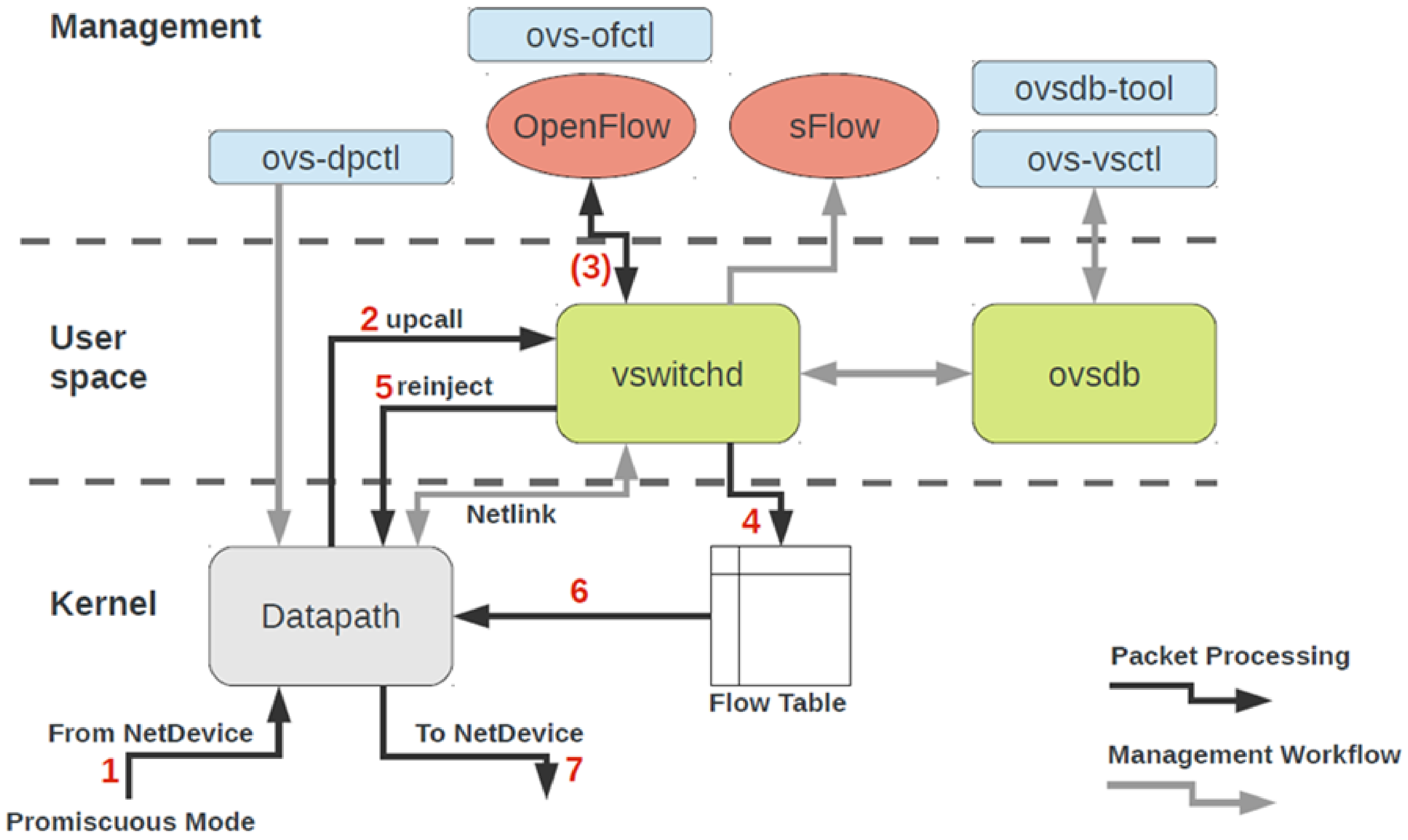
There is a way to not use a third-party tool, but to use a ‘language’ called Openflow

First things first, we need to familiarize ourselves with a device that can talk Openflow.

One of the devices is **Open Vswitch**

Open Vswitch

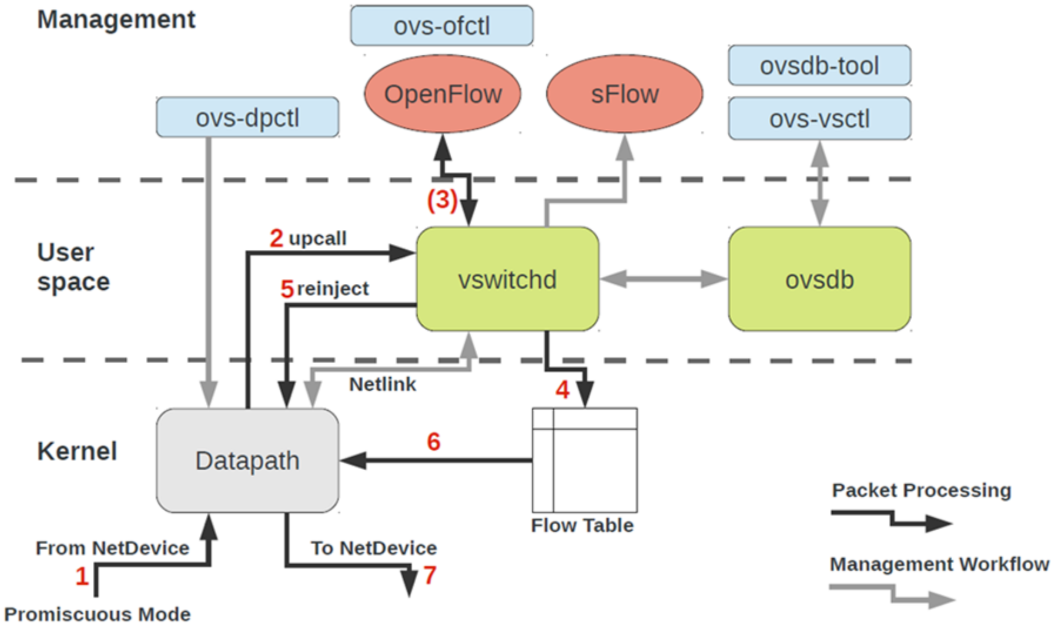
- Open vSwitch is a **production quality**, **multilayer** virtual switch licensed under the open source Apache 2.0 license.
- https://www.openvswitch.org/



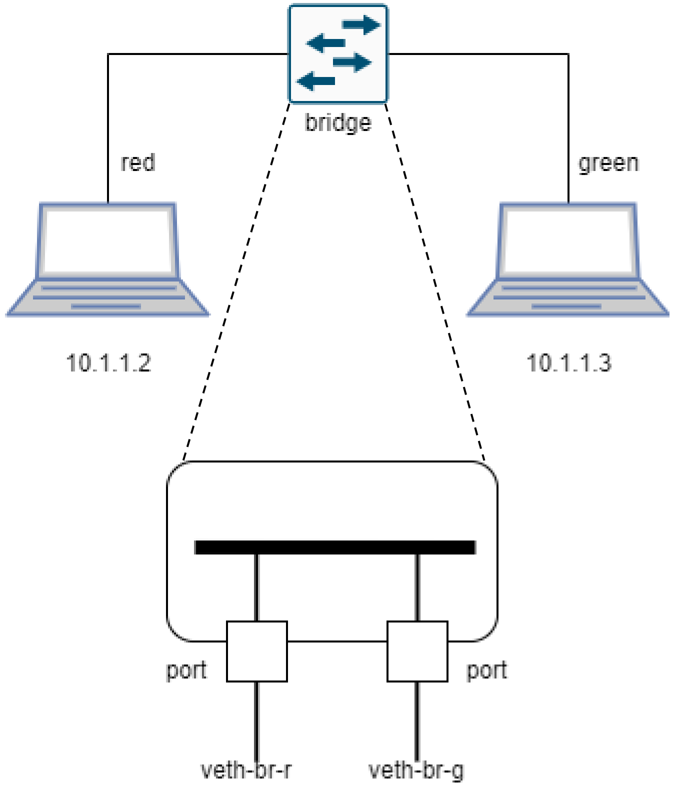
Let’s redo this, but using Open Vswitch

```
#!/bin/sh

# create netns red and green
ip netns add red
ip netns add green

# create veth
ip link add veth-r type veth peer name veth-br-r
ip link add veth-g type veth peer name veth-br-g

# create v-bridge
ip link add v-bridge type bridge
ovs-vsctl add-br v-bridge

# attach veth
ip link set veth-r netns red
ip link set veth-g netns green
ip link set veth-br-r master v-bridge
ip link set veth-br-g master v-bridge
ovs-vsctl add-port v-bridge veth-br-r
ovs-vsctl add-port v-bridge veth-br-g

# activate veth
ip netns exec red ip link set dev veth-r up
ip netns exec green ip link set dev veth-g up
ip link set dev veth-br-r up
ip link set dev veth-br-g up
ip link set dev v-bridge up
```

```
# assign ip address
ip netns exec red ip a add 10.1.1.2/24 dev veth-r
ip netns exec green ip a add 10.1.1.3/24 dev veth-g

# test
ip netns exec red ping 10.1.1.3 -c 3
```

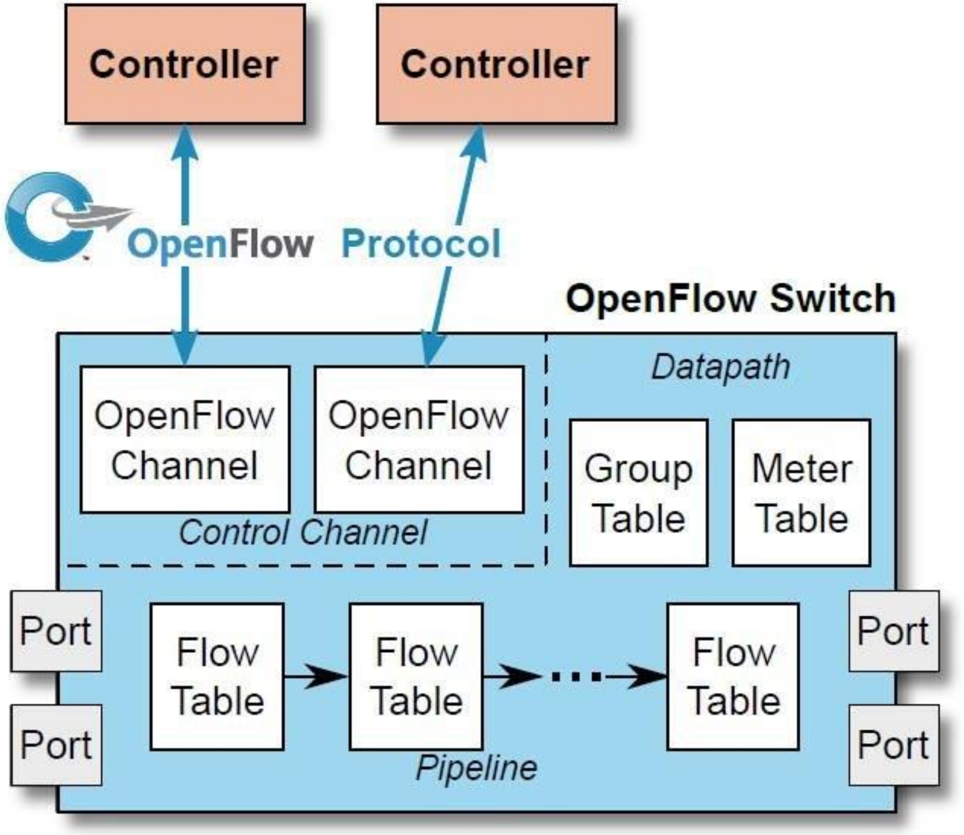


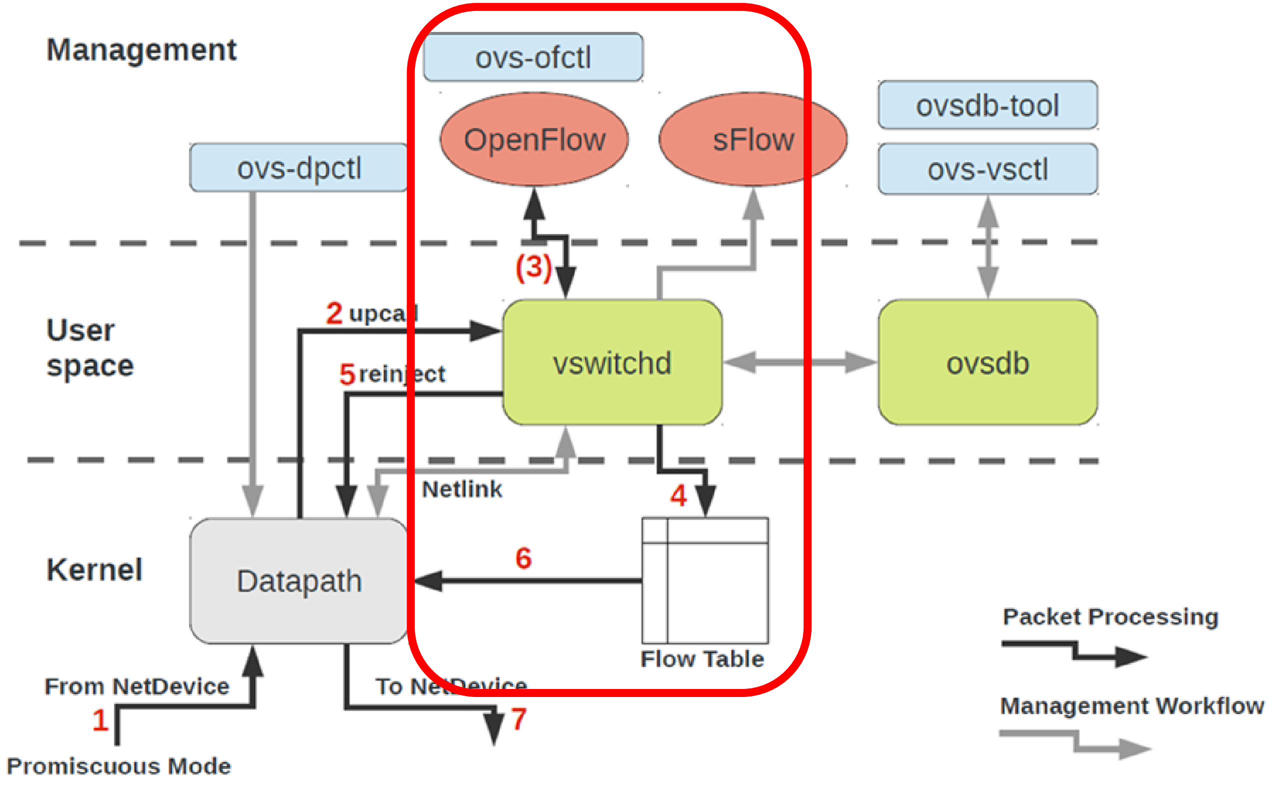
This is now using OVS



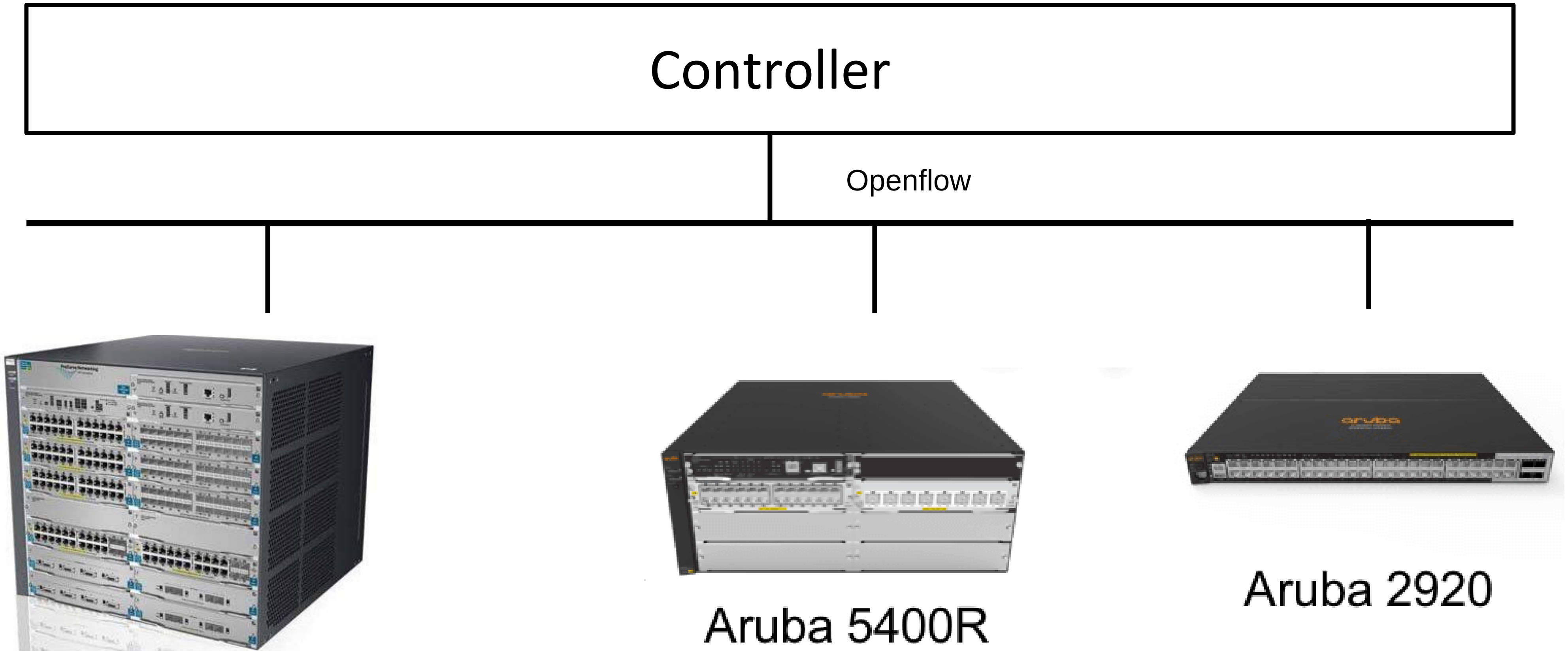
Openflow

- Openflow is a standardized protocol to configure forwarding planes of network devices
- https://opennetworking.org/sdn-resources/customer-case-studies/openflow/





Other Examples



Openflow: How it works

Ref: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Table 1: Main components of a flow entry in a flow table.

Each flow table entry (see Table 1) contains :

- **match fields**: to match against packets. These consist of the ingress port and packet headers, and optionally other pipeline fields such as metadata specified by a previous table.
- **priority**: matching precedence of the flow entry.
- **counters**: updated when packets are matched.
- **instructions**: to modify the action set or pipeline processing.
- **timeouts**: maximum amount of time or idle time before flow is expired by the switch.
- **cookie**: opaque data value chosen by the controller. May be used by the controller to filter flow entries affected by flow statistics, flow modification and flow deletion requests. Not used when processing packets.
- **flags**: flags alter the way flow entries are managed, for example the flag `OFPFF_SEND_FLOW_REM` triggers flow removed messages for that flow entry.

Openflow: How it works

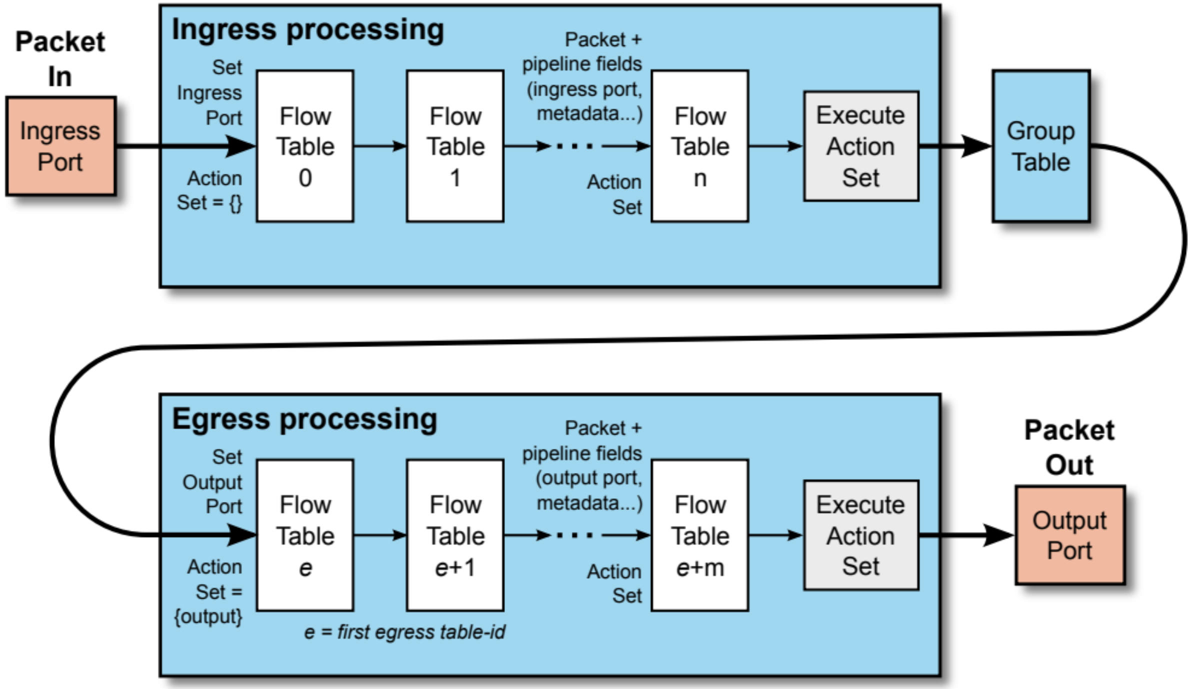


Figure 2: Packet flow through the processing pipeline.

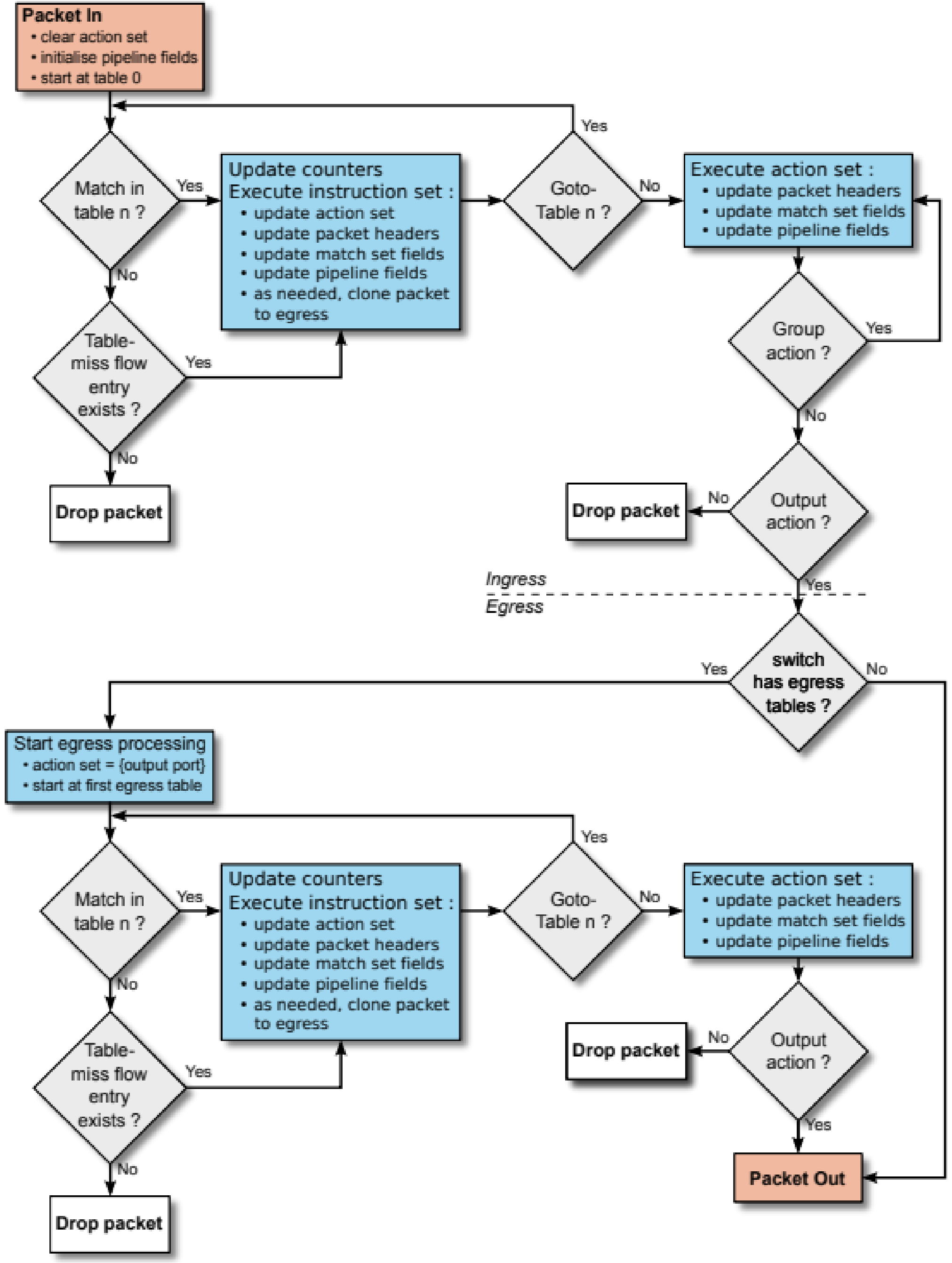
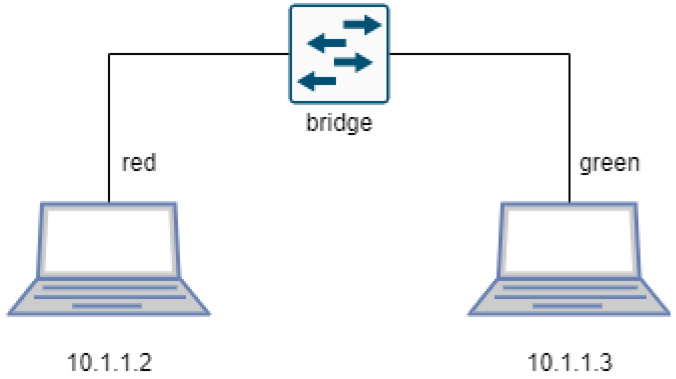


Figure 3: Simplified flowchart detailing packet flow through an OpenFlow switch.

Let’s see it in action: Start from the trivial one

1. Run the setup

```
root@ubuntu2004:/home/ubuntu/Documents# bash ./create-ovs-l2switch.sh
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=0.352 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=64 time=0.026 ms
64 bytes from 10.1.1.3: icmp_seq=3 ttl=64 time=0.046 ms

--- 10.1.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2048ms
rtt min/avg/max/mdev = 0.026/0.141/0.352/0.149 ms
```



2. Dump the flow table

```
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl dump-flows v-bridge
 cookie=0x0, duration=675.543s, table=0, n_packets=67, n_bytes=6304, priority=0 actions=NORMAL
```

3. Delete the flow table

```
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl del-flows v-bridge
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl dump-flows v-bridge
root@ubuntu2004:/home/ubuntu/Documents#
```

```
root@ubuntu2004:/home/ubuntu/Documents# ip netns exec red ping 10.1.1.3
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.
```

4. Set the flow table such that the OVS acts as a typical L2-switch

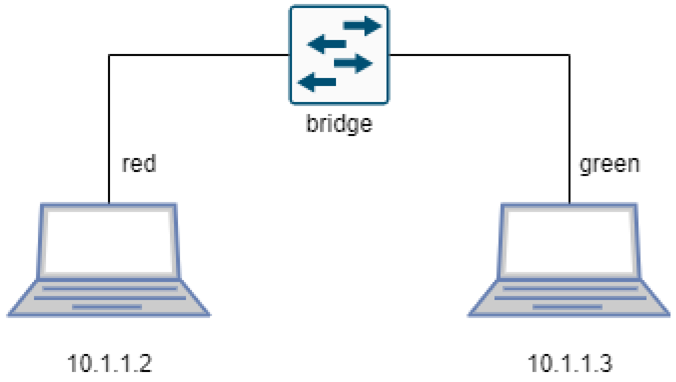
```
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl add-flow v-bridge actions=NORMAL
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl dump-flows v-bridge
 cookie=0x0, duration=431.524s, table=0, n_packets=14, n_bytes=1148, actions=NORMAL
root@ubuntu2004:/home/ubuntu/Documents#
```

```
root@ubuntu2004:/home/ubuntu/Documents# ip netns exec red ping 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=172 ttl=64 time=1024 ms
64 bytes from 10.1.1.3: icmp_seq=173 ttl=64 time=0.258 ms
64 bytes from 10.1.1.3: icmp_seq=174 ttl=64 time=0.045 ms
64 bytes from 10.1.1.3: icmp_seq=175 ttl=64 time=0.043 ms
64 bytes from 10.1.1.3: icmp_seq=176 ttl=64 time=0.043 ms
```

Manually using Openflow to enable L2-switch

1. Run the setup and delete the flow table

```
root@ubuntu2004:/home/ubuntu/Documents# bash ./create-ovs-l2switch.sh
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=0.377 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=64 time=0.054 ms
64 bytes from 10.1.1.3: icmp_seq=3 ttl=64 time=0.043 ms

--- 10.1.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2032ms
rtt min/avg/max/mdev = 0.043/0.158/0.377/0.154 ms
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl del-flows v-bridge
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl dump-flows
ovs-ofctl: 'dump-flows' command requires at least 1 arguments
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl dump-flows v-bridge
root@ubuntu2004:/home/ubuntu/Documents#
```



2. Check the port number

```
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl show v-bridge
OFPT_FEATURES_REPLY (xid=0x2): dpid:000002086d42a444
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MA
H_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_d
src mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_
t
 1(veth-br-r): addr:42:58:9c:63:a0:fe
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(veth-br-g): addr:d2:ad:11:b8:28:d3
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(v-bridge): addr:02:08:6d:42:a4:44
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
root@ubuntu2004:/home/ubuntu/Documents#
```

3. Add a flow entry

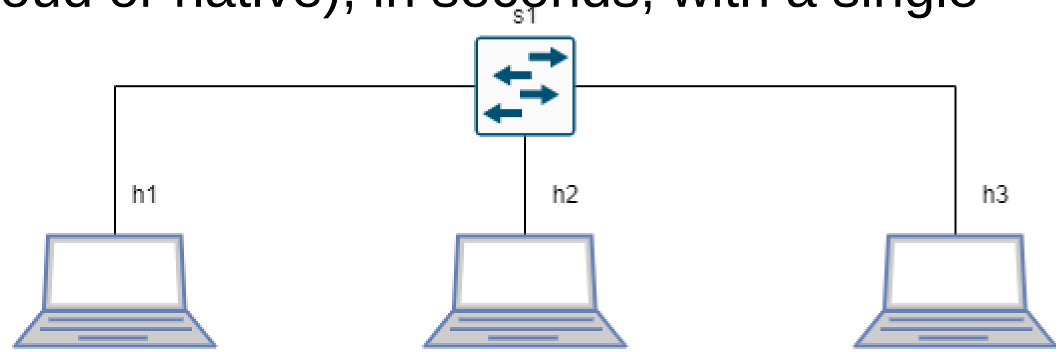
```
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl add-flow v-bridge in_port=1,actions=output:2
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl add-flow v-bridge in_port=2,actions=output:1
root@ubuntu2004:/home/ubuntu/Documents# ovs-ofctl dump-flows v-bridge
 cookie=0x0, duration=11.708s, table=0, n_packets=0, n_bytes=0, in_port="veth-br-r" actions=output:"veth-br-g"
 cookie=0x0, duration=3.384s, table=0, n_packets=0, n_bytes=0, in_port="veth-br-g" actions=output:"veth-br-r"
root@ubuntu2004:/home/ubuntu/Documents# ip netns exec red ping 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=0.275 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=64 time=0.058 ms
```

Are you not tired to create the setup manually?

Here comes mininet

Mininet and OVS

- Mininet creates a **realistic virtual network**, running **real kernel, switch and application code**, on a single machine (VM, cloud or native), in seconds, with a single command.
- http://mininet.org/



1. A single command to establish the topology

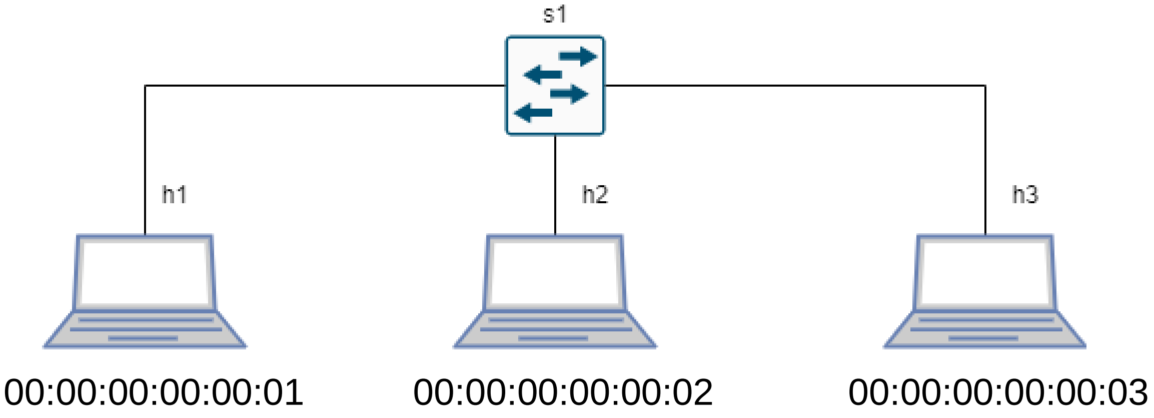
```
root@ubuntu2004:/home/ubuntu/Documents# mn --topo=single,3 --controller=none --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

2. Add a flow entry and check reachability

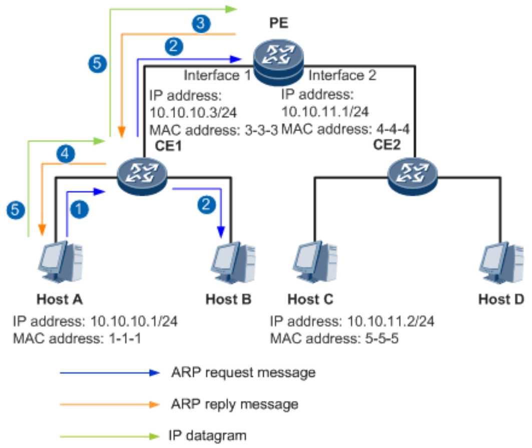
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet> sh ovs-ofctl add-flow s1 actions=normal
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Openflow L2 matching

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions=output:2
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions=output:1
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
```



What's wrong with this?




```
  1 0.000000000   00:00:00_00:00:01    ARP   44 Who has 10.0.0.2? Tell 10.0.0.1
  2 0.000140947   00:00:00_00:00:01    ARP   44 Who has 10.0.0.2? Tell 10.0.0.1
  3 0.000143241   00:00:00_00:00:01    ARP   44 Who has 10.0.0.2? Tell 10.0.0.1
  4 0.000151906   00:00:00_00:00:02    ARP   44 10.0.0.2 is at 00:00:00:00:00:02
  5 0.000187644   00:00:00_00:00:02    ARP   44 10.0.0.2 is at 00:00:00:00:00:02
 10 0.001648989   00:00:00_00:00:01    ARP   44 Who has 10.0.0.3? Tell 10.0.0.1
 11 0.001652469   00:00:00_00:00:01    ARP   44 Who has 10.0.0.3? Tell 10.0.0.1
 12 0.001653221   00:00:00_00:00:01    ARP   44 Who has 10.0.0.3? Tell 10.0.0.1
 13 0.001657191   00:00:00_00:00:03    ARP   44 10.0.0.3 is at 00:00:00:00:00:03
 14 1.005638802   00:00:00_00:00:01    ARP   44 Who has 10.0.0.3? Tell 10.0.0.1
 15 1.005679279   00:00:00_00:00:01    ARP   44 Who has 10.0.0.3? Tell 10.0.0.1
 16 1.005680791   00:00:00_00:00:01    ARP   44 Who has 10.0.0.3? Tell 10.0.0.1
 17 1.005690040   00:00:00_00:00:03    ARP   44 10.0.0.3 is at 00:00:00:00:00:03

Frame 1: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface any, id 0
Linux cooked capture
   Packet type: Broadcast (1)
   Link-layer address type: 1
   Link-layer address length: 6
   Source: 00:00:00_00:00:01 (00:00:00:00:00:01)
   Unused: 0000
   Protocol: ARP (0x0806)
Address Resolution Protocol (request)
   Hardware type: Ethernet (1)
   Protocol type: IPv4 (0x0800)
   Hardware size: 6
   Protocol size: 4
   Opcode: request (1)
   Sender MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)

0000  00 01 00 01 00 06 00 00  00 00 00 01 00 00 08 06   ........ ........
0010  00 01 08 00 06 04 00 01  00 00 00 00 00 01 0a 00   ........ ........
0020  00 01 00 00 00 00 00 00  0a 00 00 02               ........ ....
```

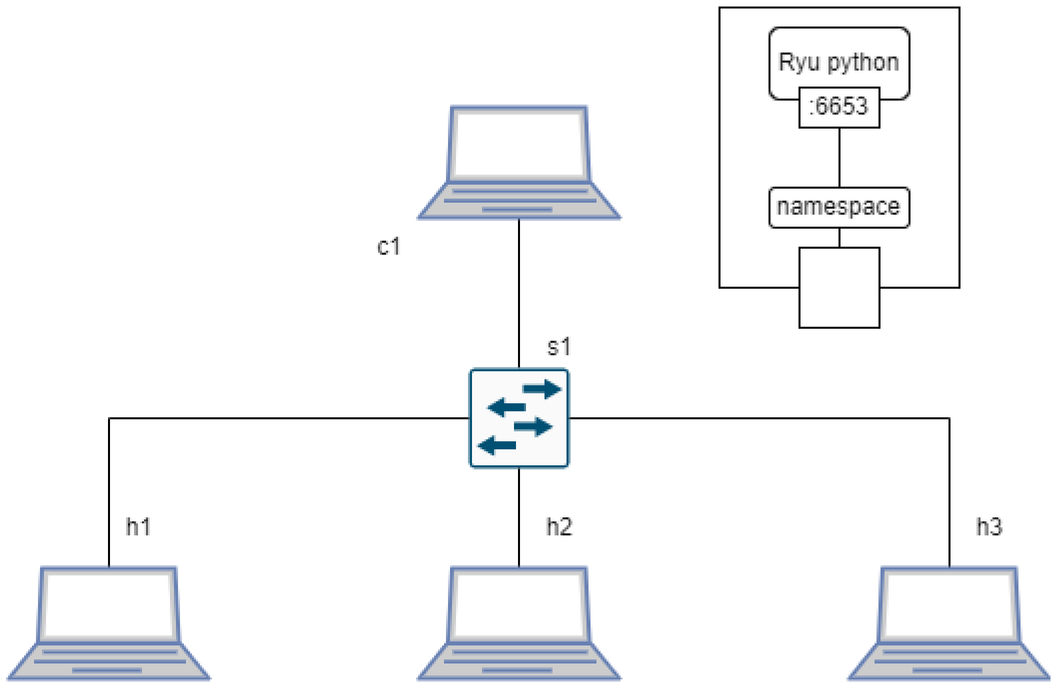
```
mininet> sh ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,actions=flood
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (2/6 received)
```

Are you not tired to type ovs-ofctl all the time and fancy bit of python?

Here comes Ryu (be patient, we're getting to the SDN real soon)

Ryu-OVS-mininet

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_0

class L2Switch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath
        ofp = dp.ofproto
        ofp_parser = dp.ofproto_parser

        actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]

        data = None
        if msg.buffer_id == ofp.OFP_NO_BUFFER:
             data = msg.data

        out = ofp_parser.OFPPacketOut(
            datapath=dp, buffer_id=msg.buffer_id, in_port=msg.in_port,
            actions=actions, data = data)
        dp.send_msg(out)
```



1. Run your Ryu

   ryu-manager ./l2-ryu.py

2. Run your mininet topology

   mn --topo=single,3 --controller=remote --mac

Ref: https://ryu.readthedocs.io/en/latest/writing_ryu_app.html

Software Defined Networking

Definition

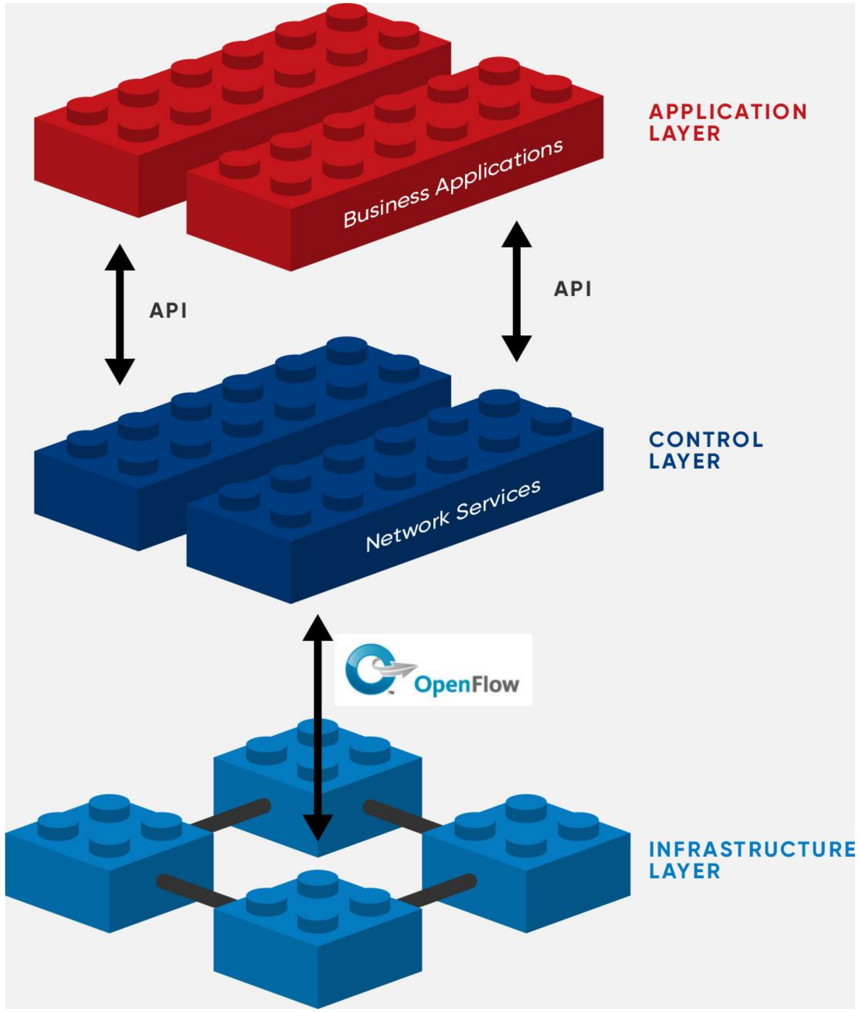
- The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices.
- Features:
  - Directly programmable

    Network control is directly programmable because it is decoupled from forwarding functions.
  - Agile

    Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.
  - Centrally managed

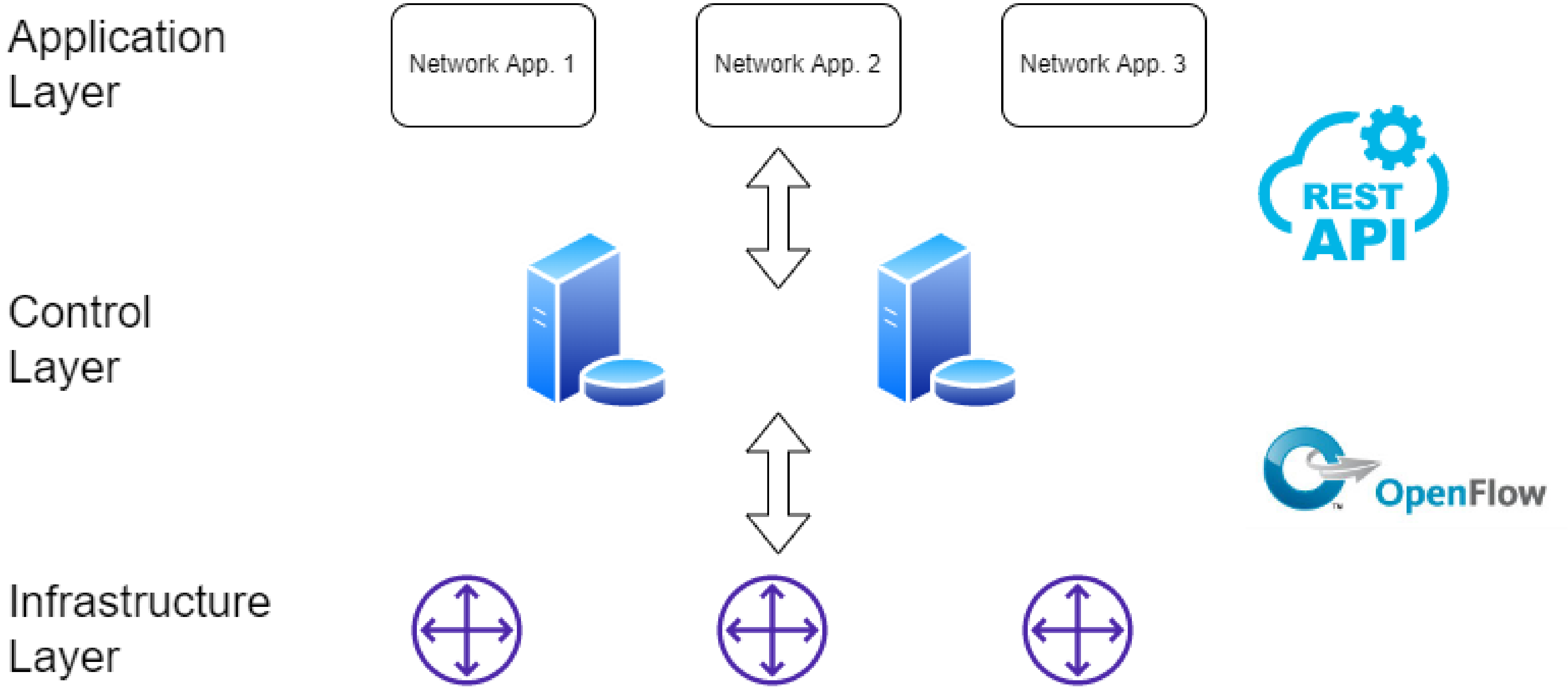
    Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
  - Open standards-based and vendor-neutral

    When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.



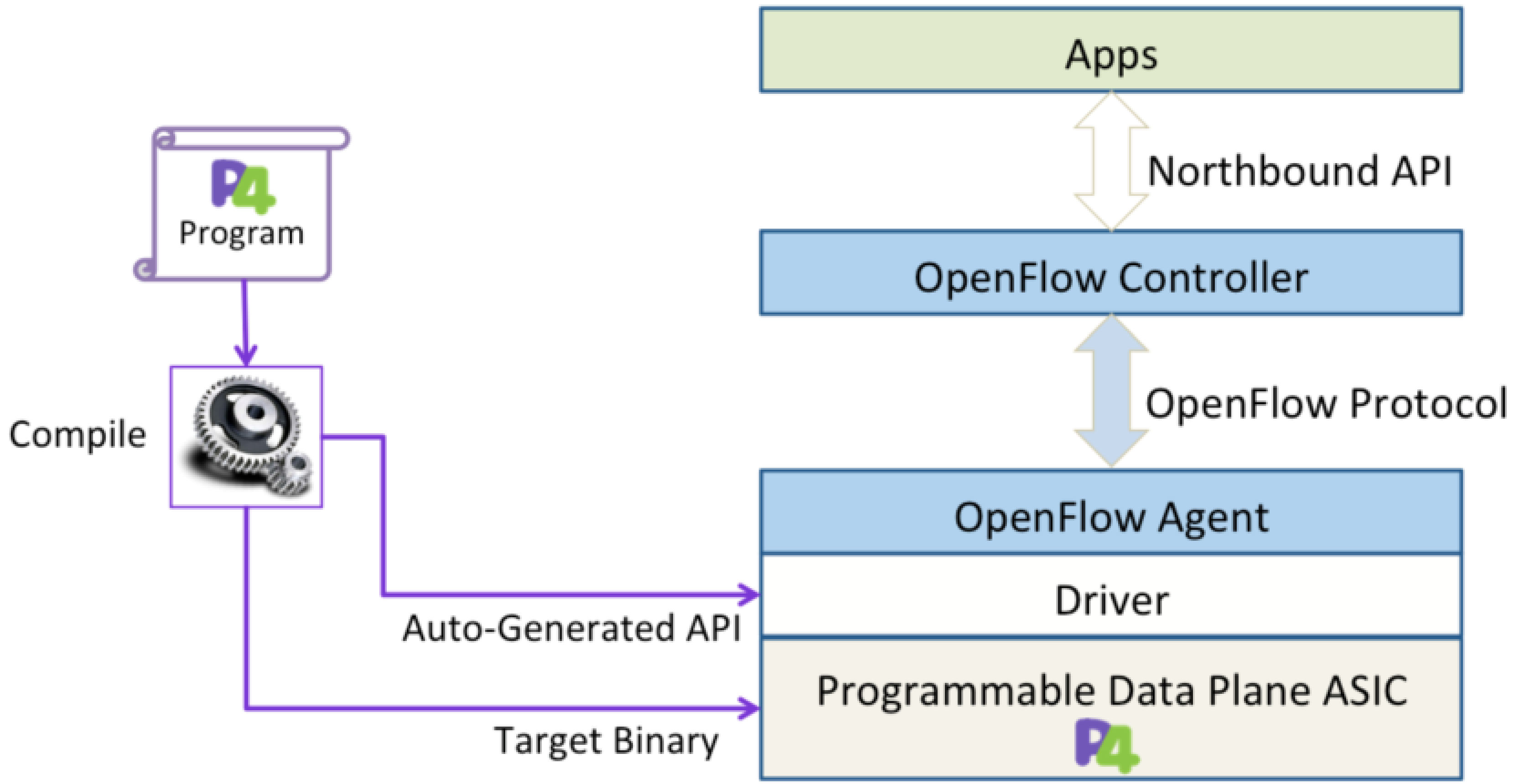
https://opennetworking.org/sdn-definition/

Realization

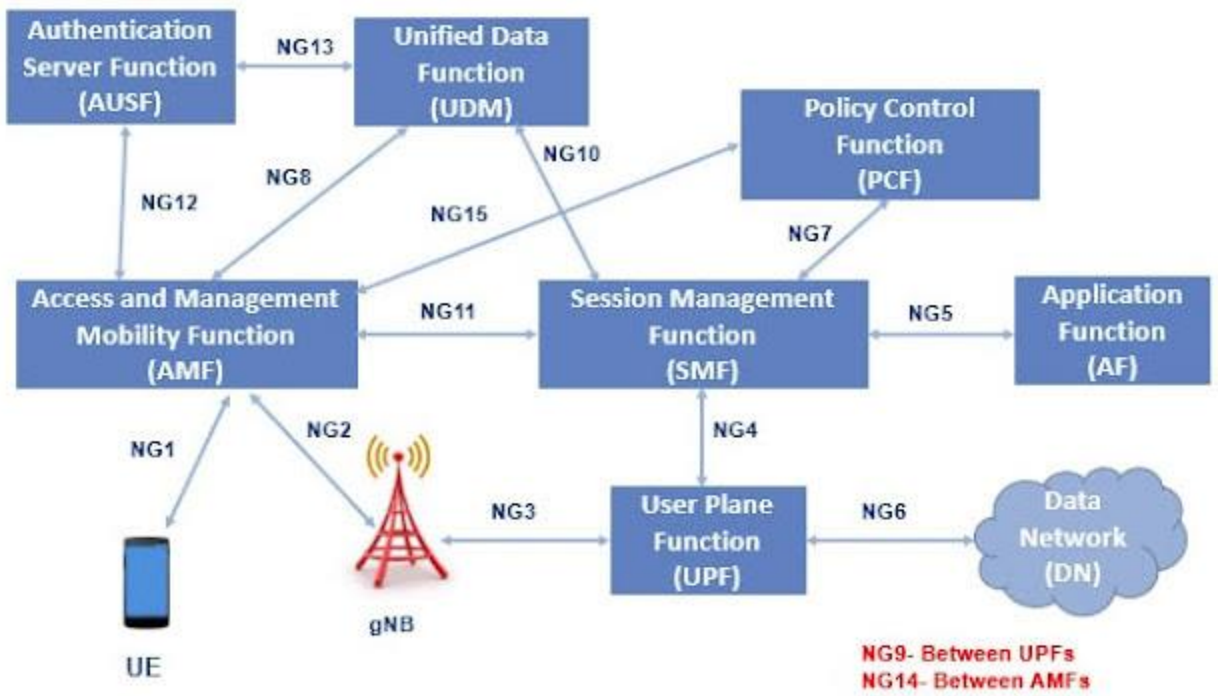


The Future

P4 & OpenFlow



The Future


Authentication Server Function (AUSF)
NG13
Unified Data Function (UDM)
Policy Control Function (PCF)
NG12
NG8
NG10
NG15
NG7
Access and Management Mobility Function (AMF)
NG11
Session Management Function (SMF)
NG5
Application Function (AF)
NG1
NG2
NG4
NG3
User Plane Function (UPF)
NG6
Data Network (DN)
UE
gNB
NG9- Between UPFs
NG14- Between AMFs

Conclusions

- Key enablers:
    - virtualization technologies,
    - programmable network functions,
    - separation of concerns,
    - open interface and open standards.
- SDN is cool!!