Software Defined Networking Tutorial

Shihabur R. Chowdhury
CS856 - Fall 2015
University of Waterloo

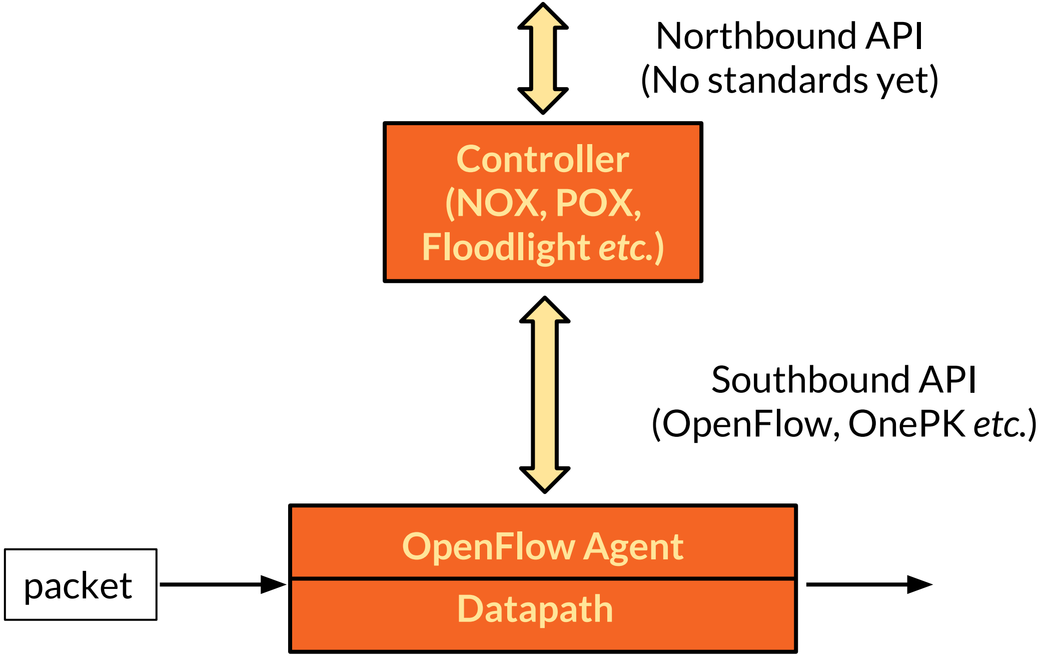
SDN Quick Recap

- Traditional networks run distributed protocols to take forwarding decisions
- SDN has a centralized control plane that makes forwarding decisions and asks the switches to act according to that

VM Credentials

- Username: sdn
- Password: sdnpass

SDN and OpenFlow



Northbound API
(No standards yet)

Controller
(NOX, POX,
Floodlight *etc.*)

Southbound API
(OpenFlow, OnePK *etc.*)

packet

OpenFlow Agent

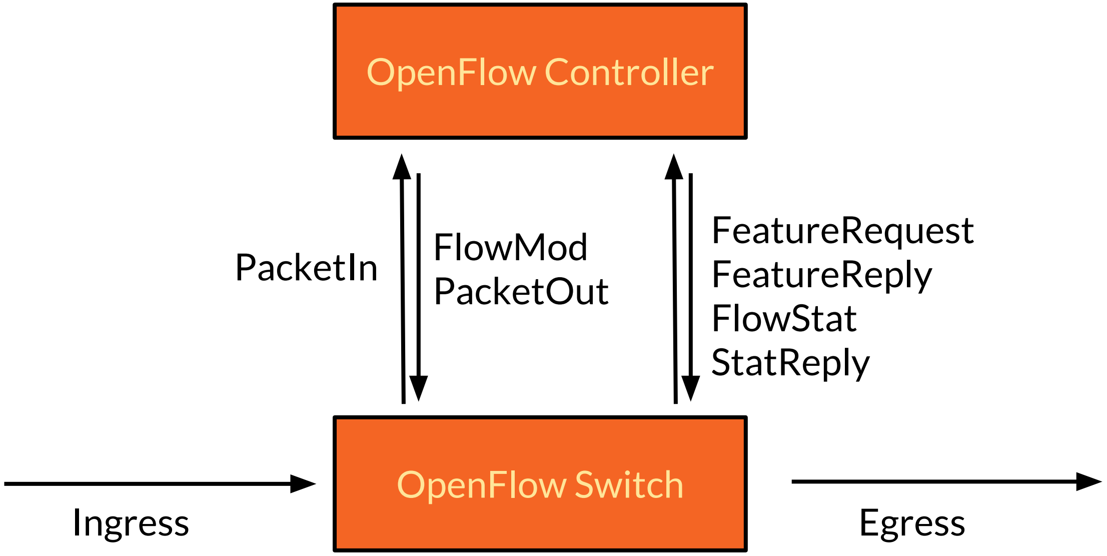
Datapath

SDN switch (pica8, indigo, Open vSwitch *etc.*)

OpenFlow

- A **switch specification** and a **switch to controller communication protocol**
- Switches have forwarding tables
    - header → (action,counter)
        - header:
            - source/destination IP
            - MAC
            - VLAN
            - TCP/UDP port *etc.*
    - header can have exact fields or wildcard fields

OpenFlow in Action



Open vSwitch

- An OpenFlow enabled virtual switch that can run on commodity Linux machines
  - kernel module forwards the packet (data plane)
  - userspace module talks to the controller
  - A remote controller can control an OVS instance (control plane)
- **`ovs-vsctl`** → create/manage bridges
- **`ovs-ofctl`** → create/manage forwarding rules
- But we need a network first !!

Mininet

- *De facto* emulator for SDN
- Uses **Open vSwitch (ovs)** to create SDN switches
- Uses network namespaces to create hosts in their own network namespace
- Can emulate a whole network in one single machine (even on a Raspberry pi)

Mininet Installation

- Install mininet
  - `sudo apt-get install mininet`
  - Already installed in the VM
- Show mininet options
  - `mn -h`

Start Mininet

- Starting without any parameter creates a single switch topology with two hosts connected with it and opens mininet console
  - **`sudo mn`**
- To view information about hosts and network use the following commands
  - **`nodes, net, dump`**

Mininet Hosts

- Hosts are processes running in their own network namespace, *i.e.*, hosts are processes with their own network configuration
- Run a command inside some host
    - h# command
        - **`h1 ifconfig`**
        - **`h1 ping -c 2 h2`**

Mininet

- Open terminal to a host
  - xterm h#
    - e.g., xterm h1
- Test network connectivity
  - pingall
- Run an iperf between random pair of hosts
  - iperf
- Set link bandwidth and delays
  - **`sudo mn --topo=single --link=tc,bw=10,delay=5ms`**

More Mininet

- Python interpreter from Mininet terminal
  - `py …`
- Show the list of available methods in a host object
  - `py dir(h1)`
- Show the IP address of a host
  - `py h1.IP()`
- Set cpu usage limit for the hosts
  - `sudo mn --topo=linear,3 --host=cfs,cpu=0.1`

Mininet Built-in Topologies

- Linear topology with 3 switches
  - `sudo mn --topo=linear,3 --switch ovsk`
- Tree topology with depth 2
  - `sudo mn --topo=tree,depth=2,fanout=2 --switch ovsk`
- Topology with a single switch
  - `sudo mn --topo=single --switch ovsk`

Working with OVS

- Show details of switch s1
  - **`ovs-ofctl show s1`**
- Show the flow rules in switch s1
  - **`ovs-ofctl dump-flows s1`**
- Show port statistics in switch s1
  - **`ovs-ofctl dump-ports s1`**
- Add a flow forwarding rule in switch s1
  - **`ovs-ofctl add-flow s1 <flow_spec>`**

Quick Exercise

- Create a linear topology with 2 nodes
- Open another terminal and dump flows in **`s1`**
- Run **`iperf`** from mininet console
- Dump the flows of **`s1`** again
- Dump the port statistics of **`s2`**

Mininet with Remote Controller

- **`sudo mn --topo=single --controller=remote,ip=127.0.0.1,port=6653`**
- Try to ping h2 from h1
  - **`h1 ping h2`**

Manually Adding Flow Rules

- There is currently no controller, therefore, no paths
- Manually add a flow rule using ovs-ofctl
  - `ovs-ofctl add-flow s1 in_port=1,action:output=2`
  - `ovs-ofctl add-flow s2 in_port=2,action:output=2`

Mininet Python API

- Mininet has a rich set of API in Python for creating your own experiment
- Create custom topologies, traffic patterns
- Run applications inside hosts, etc.
- Examples:
    - https://github.com/mininet/mininet/tree/master/examples
    - https://reproducingnetworkresearch.wordpress.com/

Mininet Python API Example

```
# Import mininet related packages
from mininet.net import Mininet
from mininet.node import Node, RemoteController
from mininet.log import setLogLevel, info
from mininet.node import CPULimitedHost
from mininet.link import TCLink

def run():
  # Construct the network with cpu limited hosts and shaped links
  net = Mininet(host = CPULimitedHost, link=TCLink)
  # Create the network switches
  s1, s2, s3 = [net.addSwitch(s) for s in 's1', 's2', 's3']
  # Create the network hosts, each having 10% of the system’s CPU
  h1, h2, h3 = [net.addHost(h, cpu=0.1) for h in 'h1', 'h2', 'h3']
  # Tell mininet to use a remote controller located at 127.0.0.1:6653
  c1 = RemoteController('c1', ip='127.0.0.1', port=6653)
  net.addController(c1)
  # Add link between switches. Each link has a delay of 5ms and 10Mbps bandwidth
  net.addLink(s1, s2, bw=10, delay='5ms')
  net.addLink(s2, s3, bw=10, delay='5ms')
  net.addLink(s3, s1, bw=10, delay='5ms')
```

Mininet Python API Example

```
  # Add link between a host and a switch
  for (h, s) in [(h1, s1), (h2, s2), (h3, s3)]:
    net.addLink(h, s, bw=10, delay='10ms')
  # Start each switch and assign it to the remote controller
  for s in [s1, s2, s3]:
    s.start([c1])
  net.start()
  # Start iperf server in h1
  h1.cmd('iperf -s &')
  # Run a iperf client on h2 and print the throughput
  result = h2.cmd('iperf  -yc -c ' + h1.IP() + ' -t 2').split(",")[-1]
  print "Throughput between h1<-->h2: " + str(float(output)/1000000.0) + "Mbps"
  net.stop()

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

FlowVisor

- A special OpenFlow controller that can **slice the network**
- Allows multiple tenants to use the same physical network

FlowVisor Installation

- Download flowvisor
  - `git clone git://github.com/OPENNETWORKINGLAB/flowvisor.git`
  - `sudo apt-get install ant default-jdk build-essential`
- Build
  - `cd flowvisor && make`
- Install
  - `sudo make install`

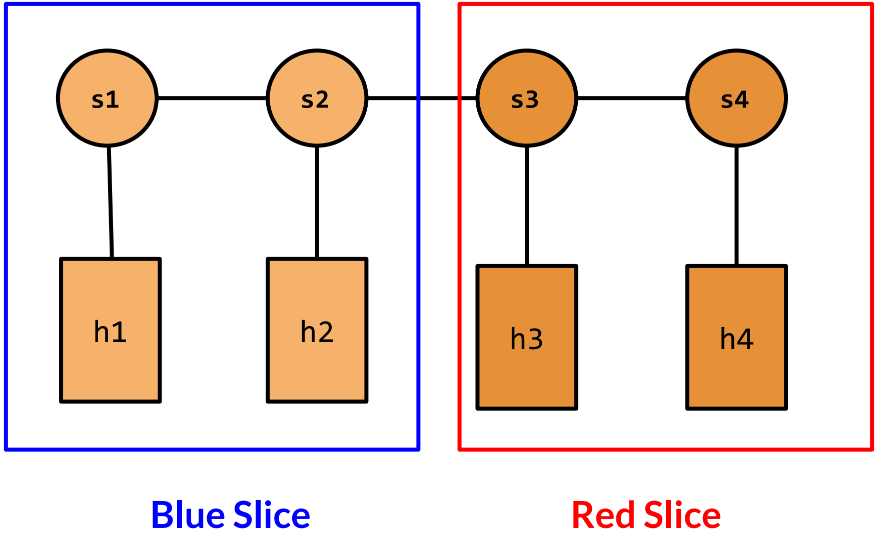
FlowVisor Installation

- Change directory ownership and permissions
  - `sudo chown sdn:sdn -R /usr/local/share/db`
  - `sudo chmod -R 777 /usr/local/share/db`

FlowVisor Configuration

- Load the configuration file
  - **`sudo fvconfig load /etc/flowvisor/config.json`**
- Stop any running OpenFlow controller
- Start flowvisor
  - **`sudo /etc/init.d/flowvisor start`**
- Enable topology controller
  - **`fvctl set-config --enable-topo-ctrl`**
- Check configuration
  - **fvctl get-config**

Create Topology and Slices



Create topology

- Create a mininet topology
  - `sudo mn --topo=linear,4 --arp --mac --controller=remote`
- Check the nodes and links from flowvisor
  - `fvctl list-datapaths`
  - `fvctl list-links`

Click network slices

- Create two slices

  -
    ```
    fvctl add-slice blue tcp:127.0.0.1:7000 admin@blue
    ```

  -
    ```
    fvctl add-slice right tcp:127.0.0.1:8000 admin@red
    ```
- List the slices

  -
    ```
    fvctl list-slices
    ```

Create flowspaces

- Create flowspace partitions
    - `fvctl add-flowspace dpid1 1 1 any blue=7`
    - `fvctl add-flowspace dpid2-p1 2 1 in_port=1 blue=7`
    - `fvctl add-flowspace dpid2-p2 2 1 in_port=2 blue=7`
    - `fvctl add-flowspace dpid4 4 1 any red=7`
    - `fvctl add-flowspace dpid3-p1 3 1 in_port=1 red=7`
    - `fvctl add-flowspace dpid3-p3 3 1 in_port=3 red=7`

Run Controllers

- Open two terminals
- In terminal 1
  - `sudo ovs-controller ptcp:7000`
- In terminal 2
  - `sudo ovs-controller ptcp:8000`

Test

- In mininet console
  - `h1 ping h2`
  - `h3 ping h4`
  - `h1 ping h3`