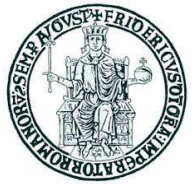
Cloud and Datacenter Networking

Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione DIETI
Laurea Magistrale in Ingegneria Informatica

Prof. Roberto Canonico

OpenFlow Tutorial:
Mininet and the Floodlight Controller



Mininet

- Mininet is a tool that allows to create machine-local virtual networks with arbitrary topologies
- More precisely, Mininet is a lightweight virtualization/container based emulator
  - modest hardware requirements, fast startup, hundreds of nodes
  - command line tool, CLI, simple Python API
- Supports parameterized topologies
- Python scripts can be used to orchestrate an experiment
  - Network topology definition
  - Events to be triggered in network nodes (e.g. execution of a program)
- Mininet is frequently used to test OpenFlow controllers
- Abstraction
  - Host: emulated as an OS level process
  - Switch: emulated by using software-based switch
    - E.g., Open vSwitch, SoftSwitch
- Mininet VM installation: the easiest way of installing Mininet
  - Download the Mininet pre-installed VM image
  - Download and install one of the hypervisors (e.g., VirtualBox, VMware Workstation, or KVM)
  - Import VM image into selected hypervisor

Mininet Command Line Interface Usage

- Start a minimal topology

```
$ sudo mn
```

- Start a minimal topology using a remote controller

```
$ sudo mn --controller=remote,ip=[IP_ADDDR],port=[listening port]
```

- Start a custom topology

```
$ sudo mn --custom [topo_script_path] --topo=[topo_name]
```

- Display nodes

```
mininet> nodes
```

- Display links

```
mininet> net
```

- Dump information about all nodes

```
mininet> dump
```

Mininet: interact with hosts and switches

- Check the IP address of a certain node

```
mininet> h1 ifconfig -a
```

- Print the process list from a host process

```
mininet> h1 ps -a
```

- Test connectivity between hosts
- Verify the connectivity by pinging from host h1 to host h2

```
mininet> h1 ping -c 1 h2
```

- Verify the connectivity between all hosts

```
mininet> pingall
```

Mininet: node and link classes

- mininet.node.Node
  - A virtual network node, which is a simply in a network namespace
- mininet.link.Link
  - A basic link, which is represented as a pair of nodes

Class	Method	Description
Node	MAC/setMAC	Return/Assign MAC address of a node or specific interface
	IP/setIP	Return/Assign IP address of a node or specific interface
	cmd	Send a command, wait for output, and return it
	terminate	Send kill signal to Node and clean up after it
Link	Link	Create a link to another node, make two new interfaces

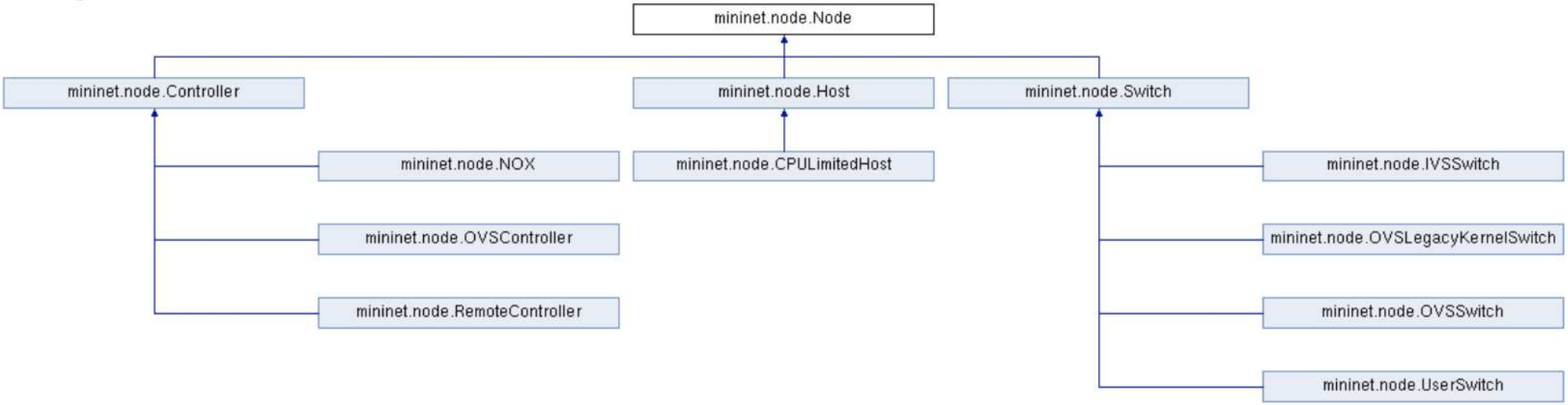
```
h1 = Host( 'h1' )
h2 = Host( 'h2' )
s1 = OVSSwitch( 's1', inNamespace=False )
c0 = Controller( 'c0', inNamespace=False )
Link( h1, s1 )
Link( h2, s1 )
h1.setIP( '10.1/8' )
h2.setIP( '10.2/8' )
```

```
c0.start()
s1.start( [ c0 ] )
print h1.cmd( 'ping -c1', h2.IP() )
s1.stop()
c0.stop()
```

Mininet Node class and subclasses

- Node generic class
- 3 subclasses: Controller, Host, Switch

Inheritance diagram for mininet.node.Node:



Mininet net class

- mininet.net.Mininet
  - Network emulation with hosts spawned in network namespaces

Class	Method	Description
Net	addHost	Add a host to network
	addSwitch	Add a switch to network
	addLink	Link two nodes into together
	addController	Add a controller to network
	getNodeByName	Return node(s) with given name(s)
	start	Start controller and switches
	stop	Stop the controller, switches and hosts
	ping	Ping between all specified hosts and return all data

```
net = Mininet()
h1 = net.addHost( 'h1' )
h2 = net.addHost( 'h2' )
s1 = net.addSwitch( 's1' )
c0 = net.addController( 'c0' )
net.addLink( h1, s1 )
net.addLink( h2, s1 )
```

```
net.start()
print h1.cmd( 'ping -c1', h2.IP() )
CLI( net )
net.stop()
```

Mininet topo class

- mininet.topo.Topo
  - Data center network representation for structured multi-trees

Class	Method	Description
Topo	Methods similar to net	E.g., addHost, addSwitch, addLink,
	addNode	Add node to graph
	addPort	Generate port mapping for new edge
	switches	Return all switches
	Hosts/nodes/switches/links	Return all hosts
	isSwitch	Return true if node is a switch, return false otherwise

```
class SingleSwitchTopo( Topo ):
    "Single Switch Topology"
    def build( self, count=1):
        hosts = [ self.addHost( 'h%d' % i )
            for i in range( 1, count + 1 ) ]
        s1 = self.addSwitch( 's1' )
        for h in hosts:
            self.addLink( h, s1 )
```

```
net = Mininet( topo=SingleSwitchTopo( 3 ) )
net.start()
CLI( net )
net.stop()
```

First Mininet experiments

- Start Mininet and create a simple topology with 1 switch and 2 hosts
  - `$ sudo mn --topo single,2 --mac --switch ovsk --controller remote,port=6653`
- Hosts are named h1 and h2
- Open xterms for hosts h1 and h2 from Mininet prompt
  - `mininet> xterm h1 h2`
- Open wireshark from xterm on h1
  - `h1# wireshark &`
- Exec a simple web server (listenong on port 8000) from xterm on h2
  - `h2# python –m SimpleHTTPServer &`
- Let h1 ping h2 from Mininet prompt
  - `mininet> h1 ping h2`
- Start Mininet and create a custom topology from Python script custom.py
  - `$ sudo mn --custom custom.py --topo mytopo`

Mininet: script to create a linear topology (1)

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import irange,dumpNodeConnections
from mininet.log import setLogLevel

class LinearTopo(Topo):
   "Linear topology of k switches, with one host per switch."

   def __init__(self, k=2, **opts):
       """Init.
           k: number of switches (and hosts)
           hconf: host configuration options
           lconf: link configuration options"""

       super(LinearTopo, self).__init__(**opts)

       self.k = k
```

Mininet: script to create a linear topology (2)

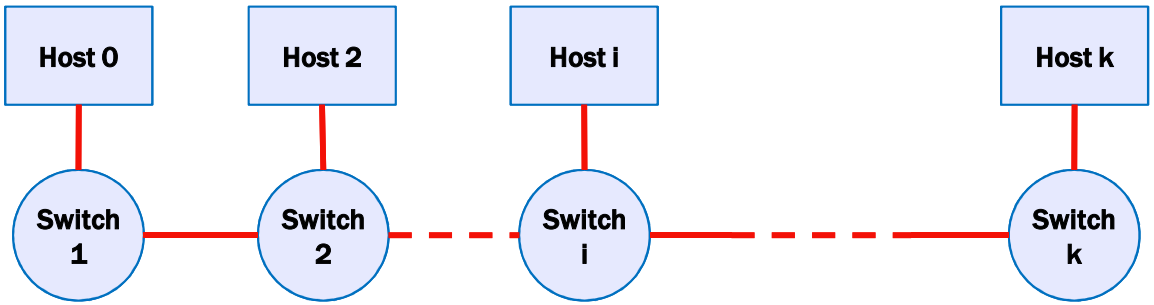
```
        lastSwitch = None
        for i in irange(1, k):
            host = self.addHost('h%s' % i, cpu=.5/k)
            switch = self.addSwitch('s%s' % i)
            # 10 Mbps, 5ms delay, 1% loss, 1000 packet queue
            self.addLink( host, switch, bw=10, delay='5ms', loss=1,
max_queue_size=1000, use_htb=True)
            if lastSwitch:
                self.addLink(switch, lastSwitch, bw=10, delay='5ms', loss=1,
max_queue_size=1000, use_htb=True)
            lastSwitch = switch
```

Set CPU limit (f≤1)

Set link parameters

Host-to-Switch link

Switch-to-Switch link



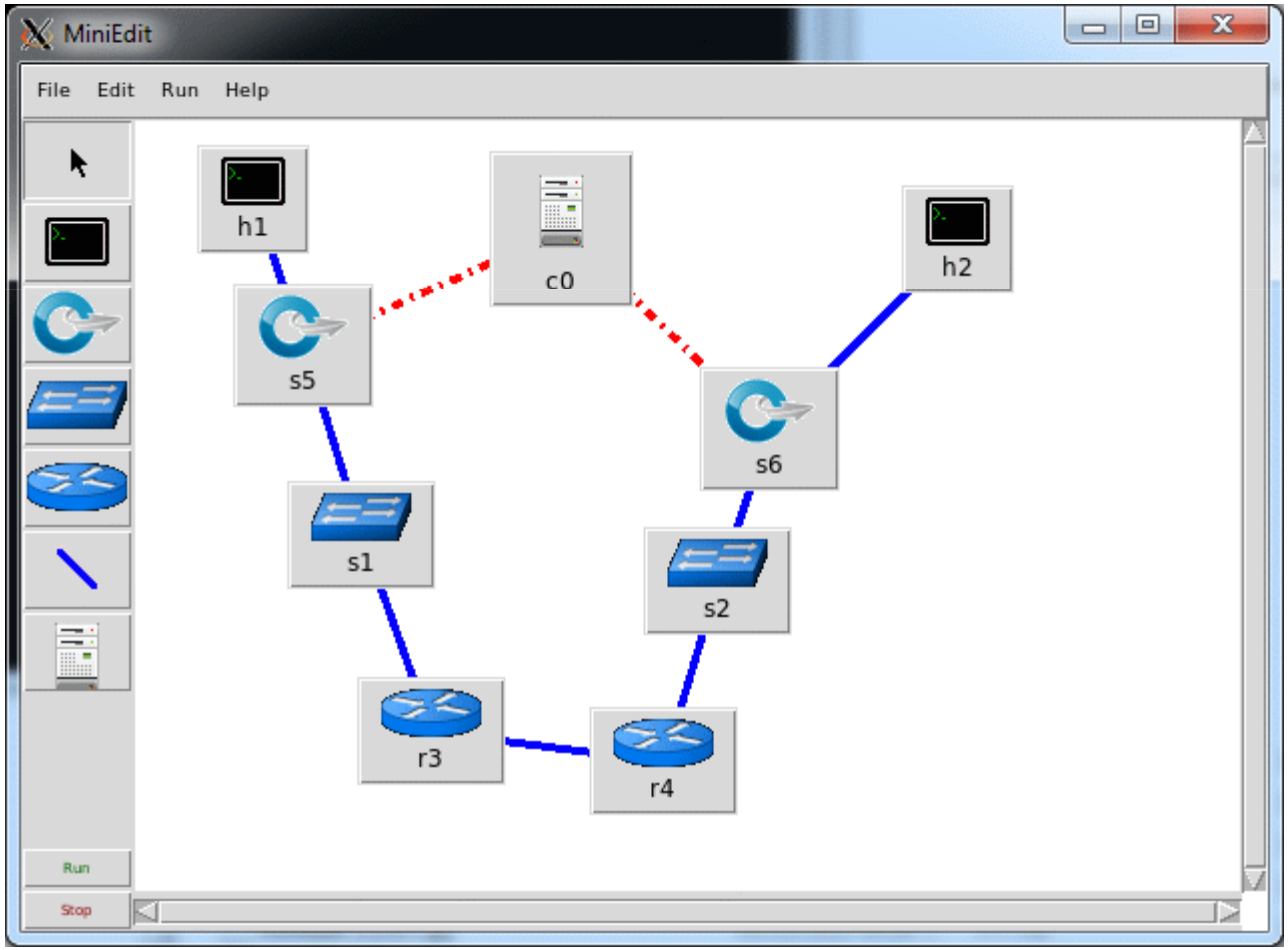
Mininet: script to create a linear topology (3)

```
def perfTest():
   "Create network and run simple performance test"
   topo = LinearTopo(k=4)
   net = Mininet(topo=topo,
                 host=CPULimitedHost, link=TCLink)
   net.start()

   print "Dumping host connections"
   dumpNodeConnections(net.hosts)
   print "Testing network connectivity"
   net.pingAll()
   print "Testing bandwidth between h1 and h4"
   h1, h4 = net.get('h1', 'h4')
   net.iperf((h1, h4))
   net.stop()

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

- Create linear topology with k=4
- Run iperf between hosts h1 and h4

Mininet graphical tools

- MiniEdit
  - A GUI application which eases the Mininet topology generation
  - Either save the topology or export as a Mininet python script

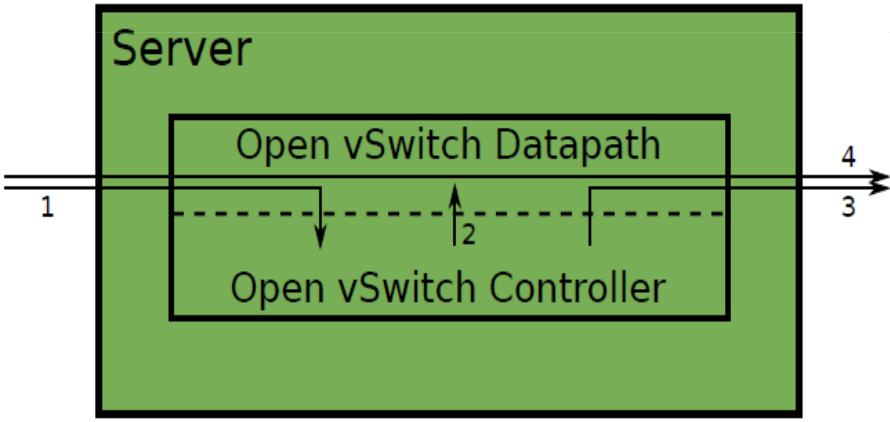


OpenFlow Tutorials

- http://mininet.org/walkthrough/
- https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
- http://www.openflow.org/wk/index.php/OpenFlow_Tutorial
- https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Getting+Started

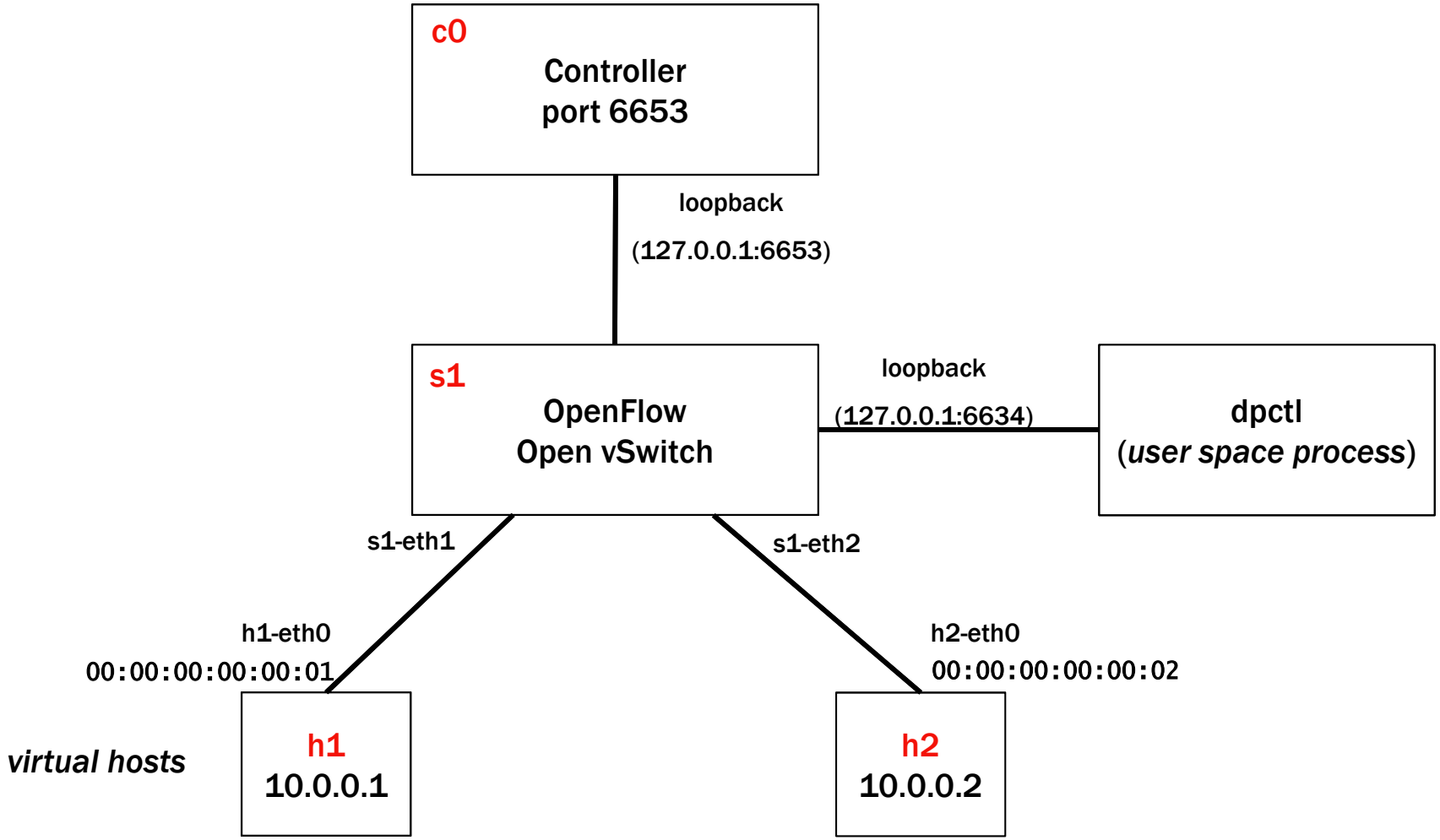
Open vSwitch as an OpenFlow software switch

- Open vSwitch design choices:
  - Flexible Controller computation in User space
  - Fast Datapath packet handling in Kernel space
- The 1st packet of a flow is sent to the controller
- The controller programs the datapath's actions for a flow
  - Usually one, but may be a list
- Actions include:
  - Forward to a port or ports
  - Mirror
  - Encapsulate and forward to controller
  - Drop
- And returns the packet to the datapath
  - Subsequent packets are handled directly by the datapath



OpenFlow Tutorial: network topology

```
$ sudo mn --topo single,2 --mac --switch ovsk --controller remote,port=6653
```



Manual insertion of OpenFlow rules in Open vSwitch

```
$ sudo mn --topo single,2 --mac --switch ovsk --controller remote
```

- Before controller is started, execute the following

```
$ sudo ovs-ofctl show tcp:127.0.0.1:6634
$ sudo ovs-ofctl dump-flows tcp:127.0.0.1:6634
mininet> h1 ping h2
```

All ports of switch shown, but no flows installed. Ping fails because ARP cannot go through

```
$ sudo ovs-ofctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
$ sudo ovs-ofctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1
mininet> h1 ping h2
```

Ping works now!

- Start controller and check OF messages on wireshark (enabling OFP decode)
  - Openflow messages exchanged between switch and controller: openflow/include/openflow/openflow.h

```
/* Header on all OpenFlow packets. */
struct ofp_header {
        uint8_t version;   /* OFP_VERSION. */
        uint8_t type;      /* one of the OFPT_ constants.*/
        uint 16_t length;  /*Length including this ofp_header. */
        uint32_t xid;      /*Transaction id associated with this packet..*/
};
```

Capture OpenFlow packets with wireshark

- If wireshark is not able to decode OF packets, reinstall a newer version

```
sudo apt-get remove wireshark
sudo apt-get -y install libgtk-3-dev libqt4-dev flex bison
wget https://www.wireshark.org/download/src/all-versions/wireshark-1.12.3.tar.bz2
tar xvfj wireshark-1.12.3.tar.bz2
cd wireshark-1.12.3
./configure
make -j4
sudo make install
sudo echo "/usr/local/lib" >> /etc/ld.so.conf
sudo ldconfig
```

- If the controller is running locally, capture packets on `lo` interface (*loopback*) on port TCP/6653 (filter = `tcp port 6653`)

OpenFlow rules set by Floodlight Controller

- For the topology created with

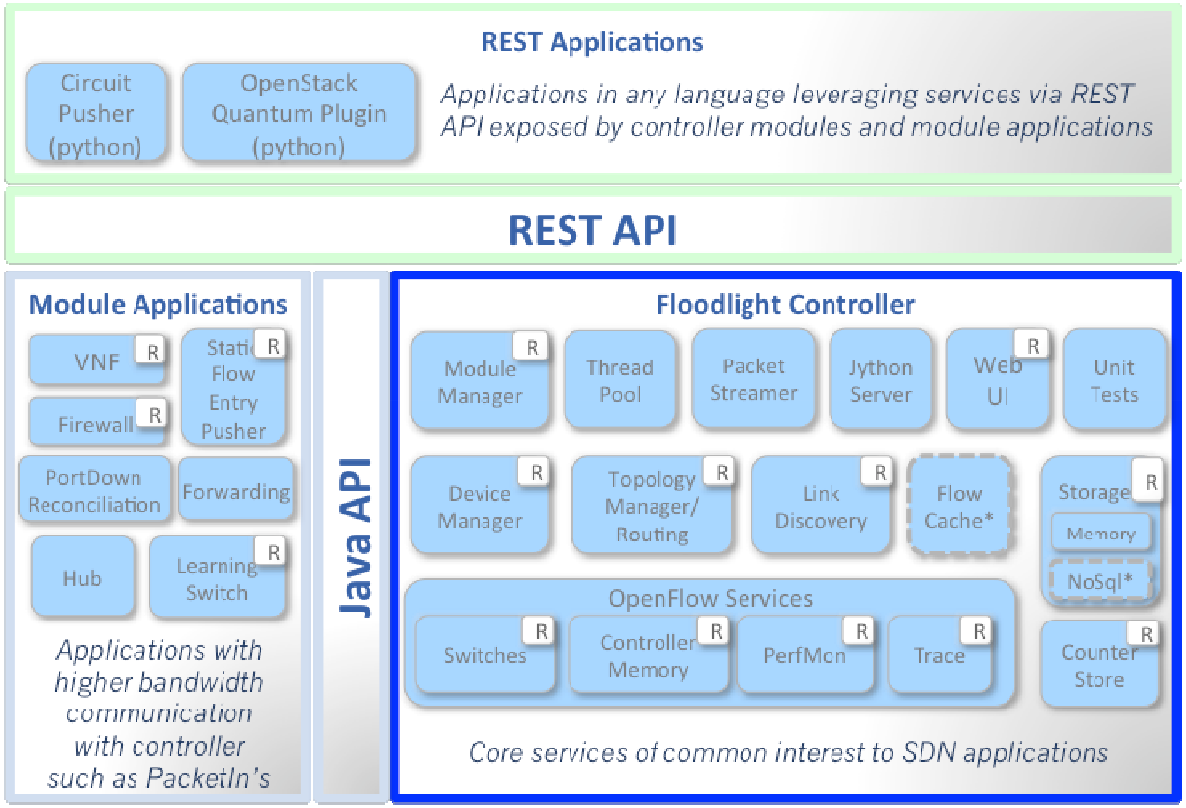
```
$ sudo mn --topo single,2 --mac --switch ovsk --controller remote,port=6653
```

- If the OpenFlow controller is running (with the Forwarding module enabled)
- Command `sudo ovs-ofctl dump-flows tcp:127.0.0.1:6634` produces the following output:

```
cookie=0x20000000000000, duration=1.343s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, idle_age=1,
    priority=1,arp,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:2
cookie=0x20000000000000, duration=6.361s, table=0, n_packets=1, n_bytes=42, idle_timeout=5, idle_age=1,
    priority=1,arp,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:1
cookie=0x20000000000000, duration=6.356s, table=0, n_packets=6, n_bytes=588, idle_timeout=5, idle_age=0,
    priority=1,ip,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2
    actions=output:2
cookie=0x20000000000000, duration=6.354s, table=0, n_packets=6, n_bytes=588, idle_timeout=5, idle_age=0,
    priority=1,ip,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1
    actions=output:1
```

Floodlight OpenFlow controller

- Floodlight is an open-source OpenFlow controller originally developed by BigSwitch Networks
- Provides a rich, extensible REST API to applications
- Applications can be developed either as Floodlight modules or as external applications interacting with Floodlight through the REST API



* Interfaces defined only & not implemented: FlowCache, NoSql

Floodlight modules

- Floodlight is a collection of Java modules
- Some modules (not all) export *services*

• Translates OF messages to Floodlight events • Managing connections to switches via Netty	FloodlightProvider (IFloodlightProviderService)
• Computes shortest path using Dijsktra • Keeps switch to cluster mappings	TopologyManager (ITopologyManagerService)
• Maintains state of links in network • Sends out LLDPs	LinkDiscovery (ILinkDiscoveryService)
• Installs flow mods for end-to-end routing • Handles island routing	Forwarding
• Tracks hosts on the network • MAC -> switch,port, MAC->IP, IP->MAC	DeviceManager (IDeviceService)
• DB style storage (queries, etc) • Modules can access all data and subscribe to changes	StorageSource (IStorageSourceService)
• Implements via Restlets (restlet.org) • Modules export RestletRoutable	RestServer (IRestApiService)
• Supports the insertion and removal of static flows • REST-based API	StaticFlowPusher (IStaticFlowPusherService)
• Create layer 2 domain defined by MAC address • Used for OpenStack / Quantum	VirtualNetworkFilter (IVirtualNetworkFilterService)

Floodlight Programming Model

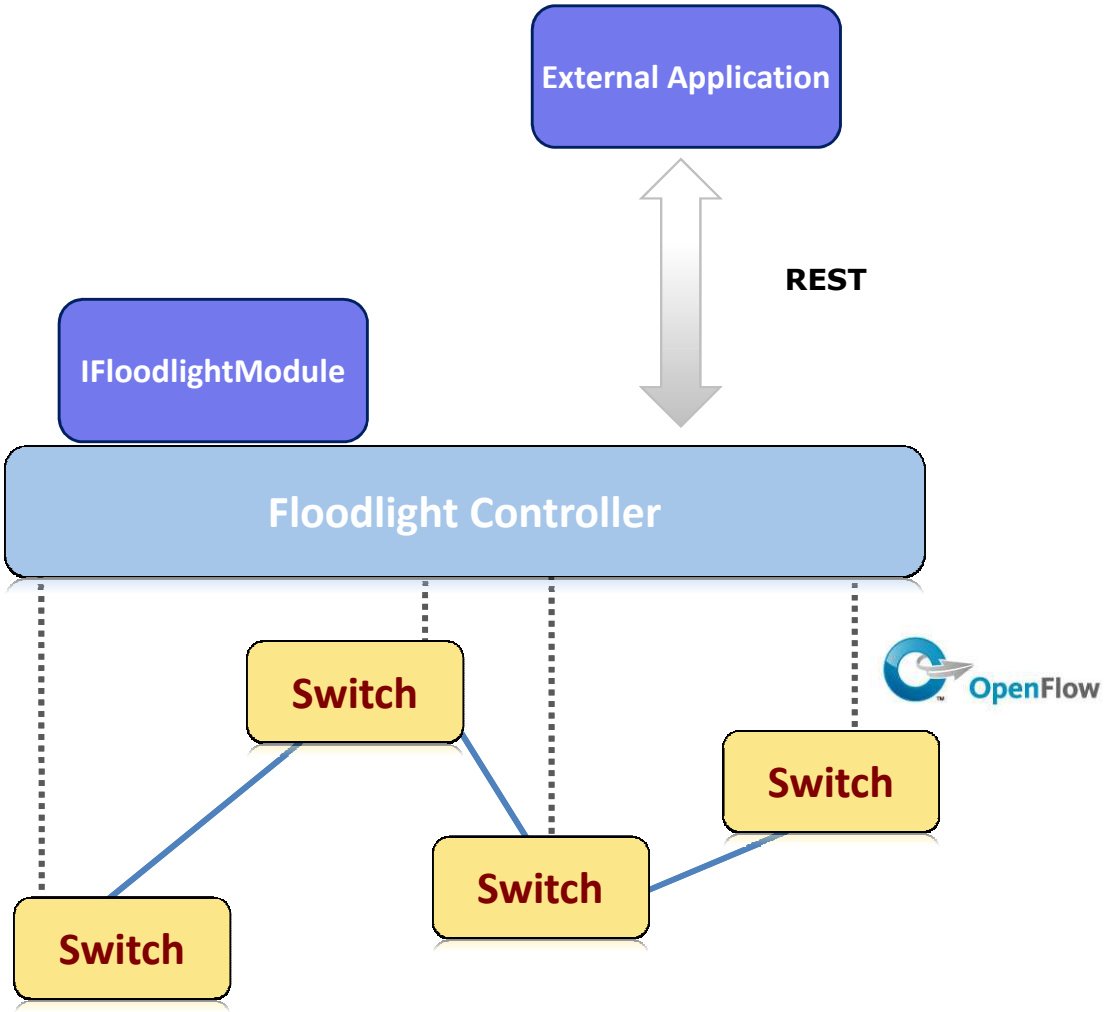
Northbound APIs

IFloodlightModule

- Java module that runs as part of Floodlight
- Consumes services and events exported by other modules
  - OpenFlow (ie. Packet-in)
  - Switch add / remove
  - Device add /remove / move
  - Link discovery

External Application

- Communicates with Floodlight via REST
  - Quantum / Virtual networks
  - Normalized network state
  - Static flows



Programming Floodlight: REST API

- Fine-grained ability to push flows over REST
- Access to normalized topology and device state
- Extensible access to add new APIs

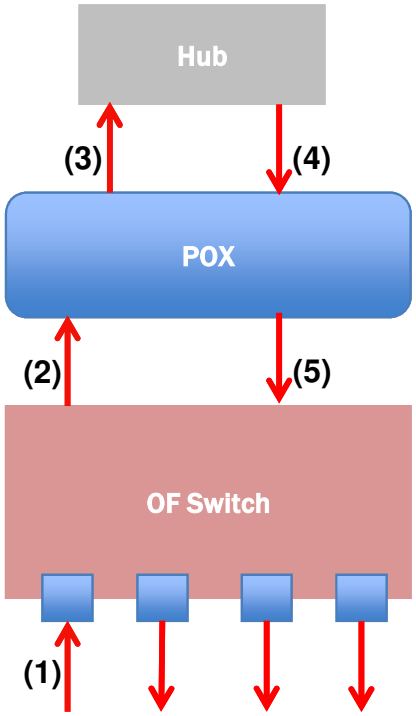
```
import httplib
import json

class StaticFlowPusher(object):

    def __init__(self, server):
        self.server = server

    def set(self, data):
        path = '/wm/staticflowentrypusher/json'
        headers = {
            'Content-type': 'application/json',
            'Accept': 'application/json',
            }
        body = json.dumps(data)
        conn = httplib.HTTPConnection(self.server, 8080)
        conn.request('POST', path, body, headers)
        response = conn.getresponse()
        ret = (response.status, response.reason, response.read())
        print ret
        conn.close()
        return ret

pusher = StaticFlowPusher('<insert_controller_ip')

flow1 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow-mod-1",
    "cookie":"0",
    "priority":"32768",
    "ingress-port":"1",
    "active":"true",
    "actions":"output=flood"
    }

pusher.set(flow1)
```

Programming Floodlight: custom module

- Custom modules implement the IFloodlightModule interface
- Handle OpenFlow messages directly (ie. PacketIn)
- Expose services to other modules
- Add new REST APIs

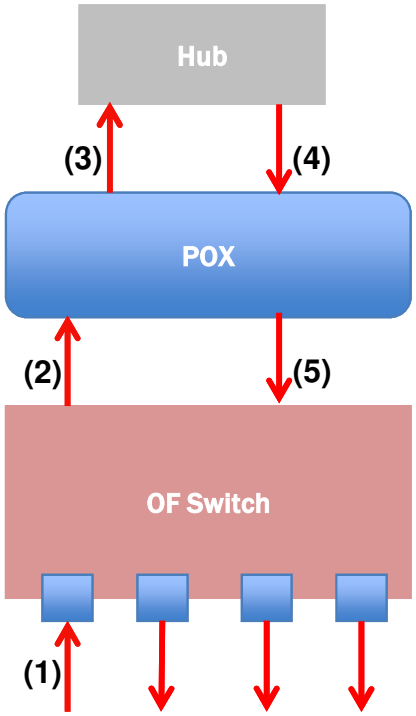
```
public class PktInHistory implements IFloodlightModule {

        @Override
        public Collection<Class<? extends IFloodlightService>>
         getModuleServices() {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public Map<Class<? extends IFloodlightService>,
         IFloodlightService> getServiceImpls() {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public Collection<Class<? extends IFloodlightService>>
         getModuleDependencies() {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public void init(FloodlightModuleContext context)
                        throws FloodlightModuleException {
                // TODO Auto-generated method stub

        }

        @Override
        public void startUp(FloodlightModuleContext context) {
                // TODO Auto-generated method stub

        }

}
```

Test case #1: hub



- App logic:
  - On init, register the appropriate packet_in handlers or interfaces
  - On packet_in,
    - Extract full packet or its buffer id
    - Generate packet_out msg with data or buffer id of the received packet
    - Set action = FLOOD
    - Send packet_out msg to the switch that generated the packet_in

Test case #1: hub (POX controller)




```
from pox.core import core
import pox.openflow.libopenflow_01 as of

# Object spawned for each switch
class L2Hub (object):
    def __init__ (self, connection):
        # Keep track of the connection to the switch so that we can
        # send it messages!
        self.connection = connection

        # This binds all our event listener
        connection.addListeners(self)

    # Handles packet in messages from the switch.
    def _handle_PacketIn (self, event):
        packet = event.parsed # This is the parsed packet data.

        packet_in = event.ofp # The actual ofp_packet_in message.

        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id

        # Add an action to send to the specified port
        action = of.ofp_action_output(port = of.OFPP_FLOOD)
        msg.actions.append(action)

        # Send message to switch
        self.connection.send(msg)


def launch ():
    def start_switch (event):
        L2Hub(event.connection)

    core.openflow.addListenerByName("ConnectionUp", start_switch)
```

Test case #2: learning switch

- App logic:
  - On init, create a dict to store MAC to switch port mapping
    - self.mac_to_port = {}
  - On packet_in,
    - Parse packet to reveal src and dst MAC addr
    - Map src_mac to the incoming port
      - self.mac_to_port[dpid] = {}
      - self.mac_to_port[dpid][src_mac] = in_port
    - Lookup dst_mac in mac_to_port dict to find next hop
    - If found, create flow_mod and send
    - Else, flood like hub