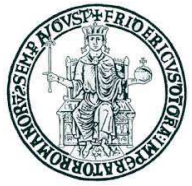
Cloud e Datacenter Networking

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



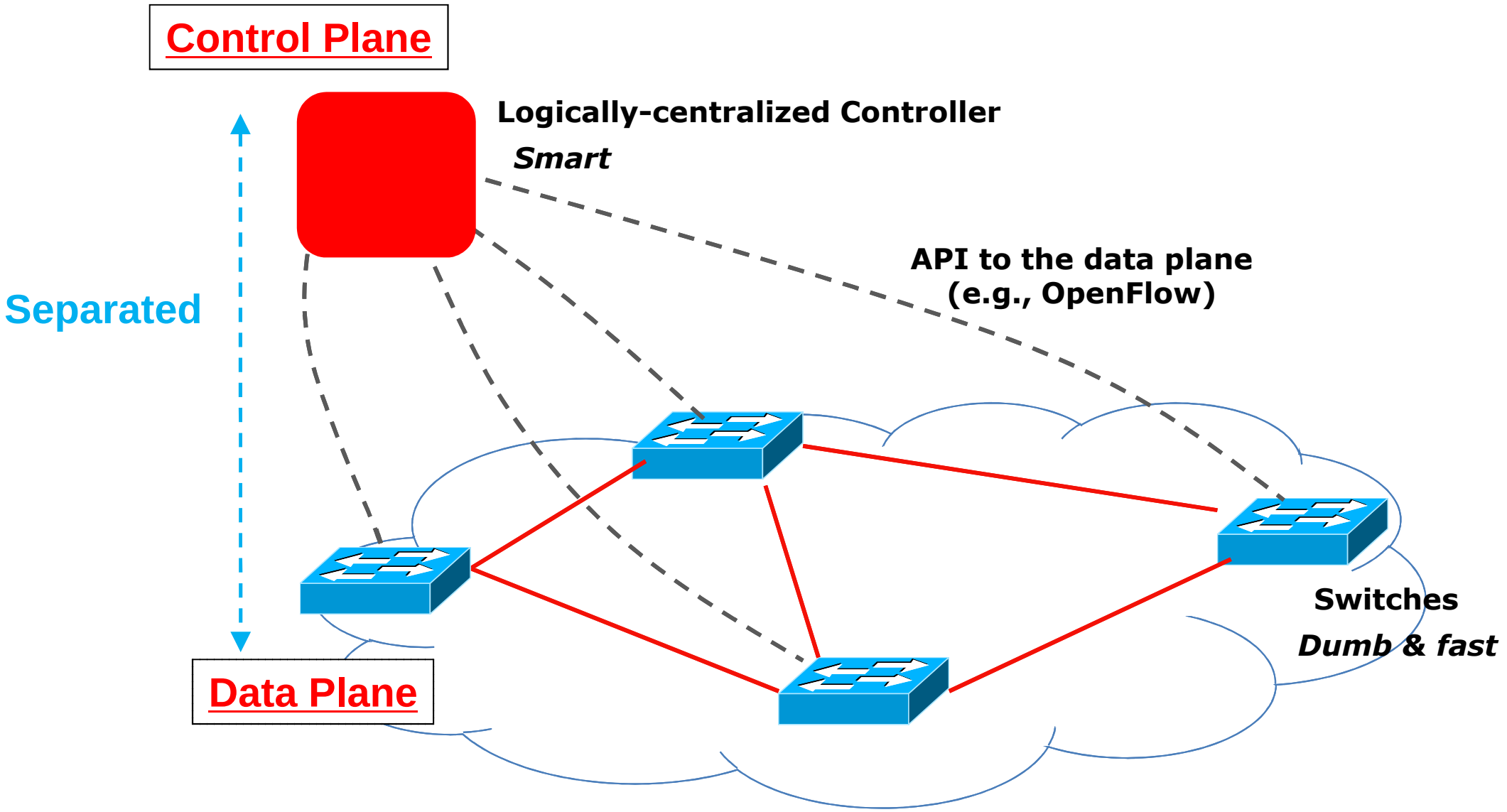
Lesson outline

- OpenFlow
- Credits for the material:
  - Jennifer Rexford
  - Nick McKeown
  - Srini Seetharaman
  - Scott Shenker

SDN Concept

- Separate control plane and data plane entities
  - Network intelligence and state are **logically centralized**
  - The underlying network infrastructure is **abstracted** from the applications
- Remotely control network devices from a central entity
- Execute or run control plane software on general purpose hardware
  - Decouple from specific networking hardware
  - Use commodity servers
- Expected advantages:
  - Ability to innovate through software
    - Overcome the “Internet ossification problem”
  - Cost reductions through increased competition, hardware commoditization and open-source software
- OpenFlow is the most popular implementation of the SDN paradigm

Software Defined Networking (SDN)



- A logically centralized "Controller" uses an open protocol to:
  - Get state information **from** forwarding elements (i.e. switches)
  - Give controls and directives **to** forwarding elements

What is OpenFlow

- OpenFlow is an *open* API that provides a standard interface for programming the data plane of switches
- OpenFlow assumes an SDN network model, i.e. separation of control plane and data plane
  - The datapath of an OpenFlow Switch consists of a **Flow Table**, and an action associated with each flow entry
  - The control path consists of a **controller** which programs the flow entry in the flow table

- But, SDN is not OpenFlow
  - OpenFlow is just one of many possible data plane forwarding abstraction

- Openflow standardization
  - Version 1.0: December 2009
  - Version 1.1: February 2011
  - OpenFlow transferred to ONF in March 2011
  - Version 1.5.0 Dec 19, 2014

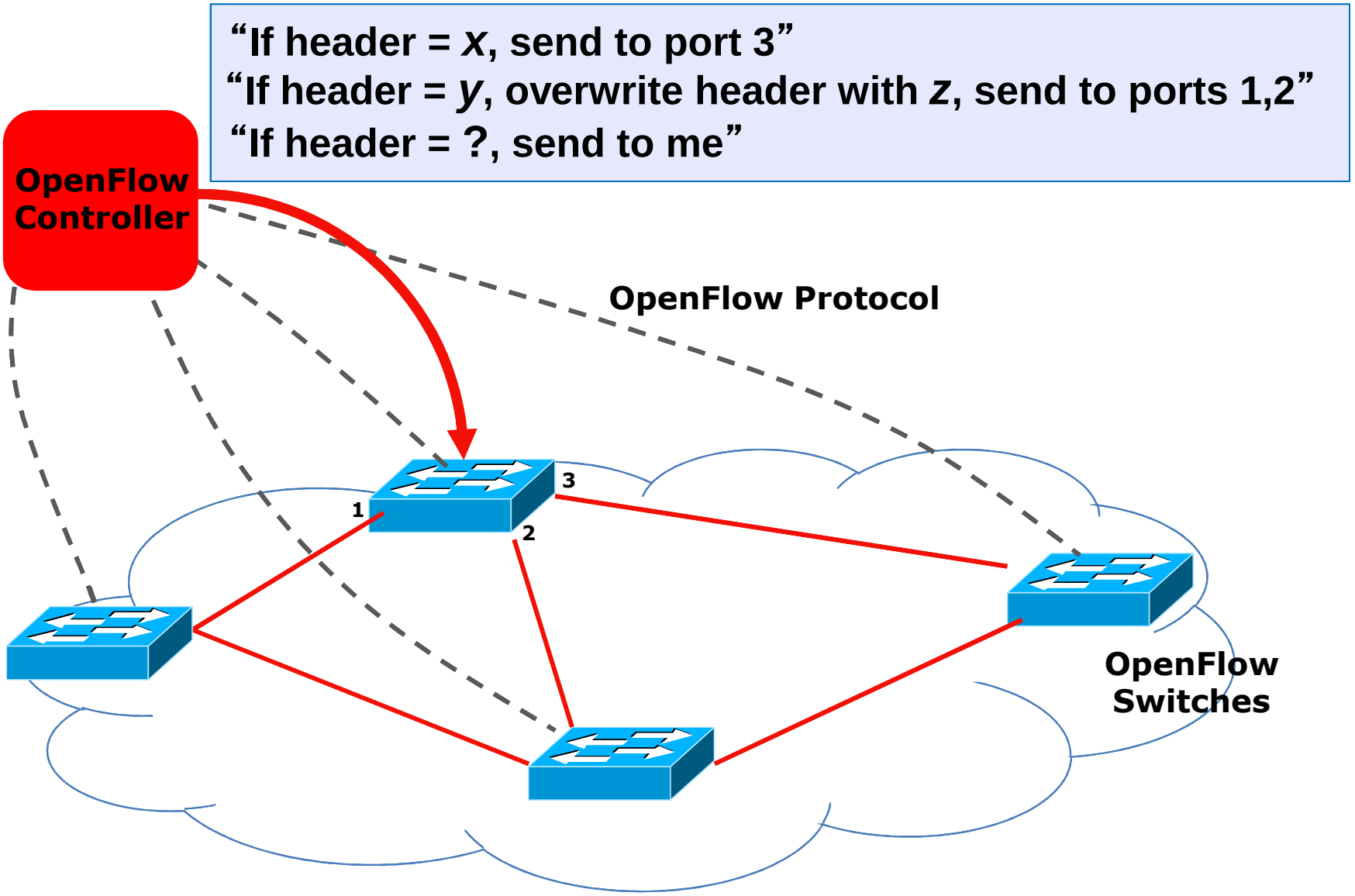
OpenFlow Consortium

http://OpenFlowSwitch.org

- Goal
  - Evangelize OpenFlow to vendors
  - Free membership for all researchers
  - Whitepaper, OpenFlow Switch Specification, Reference Designs
  - Licensing: Free for research and commercial use

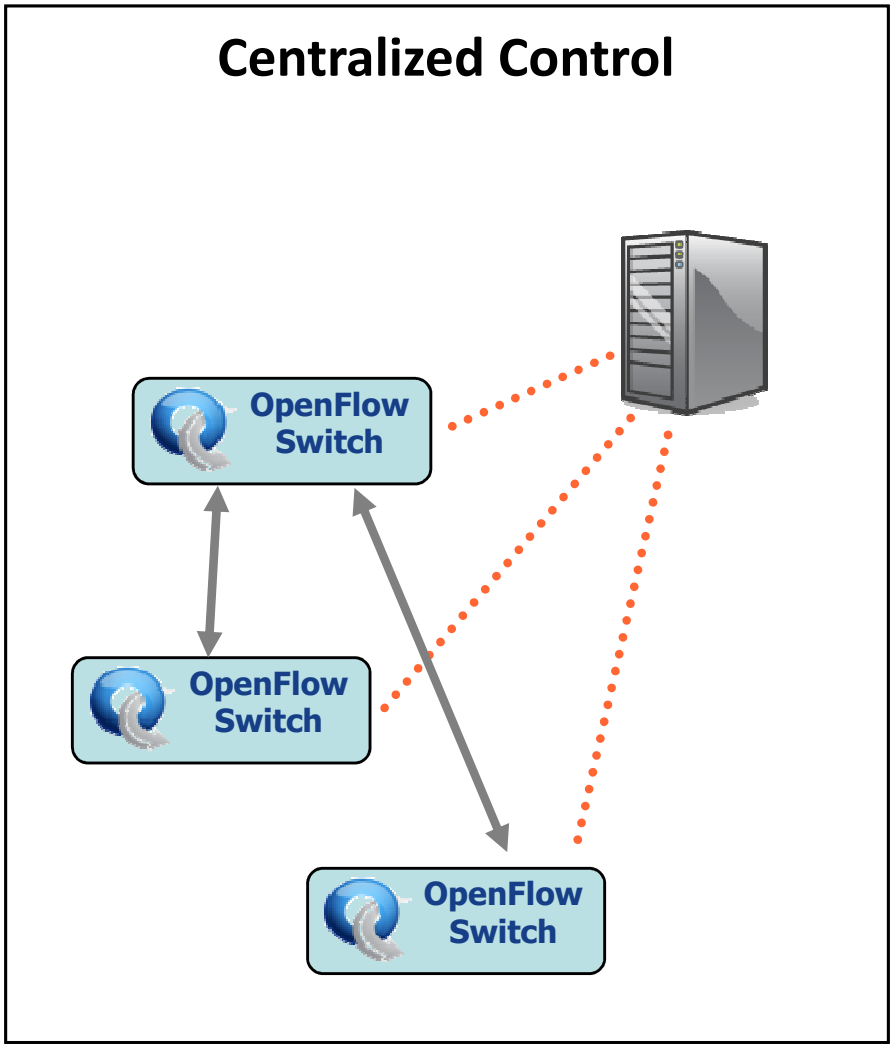
OpenFlow network model

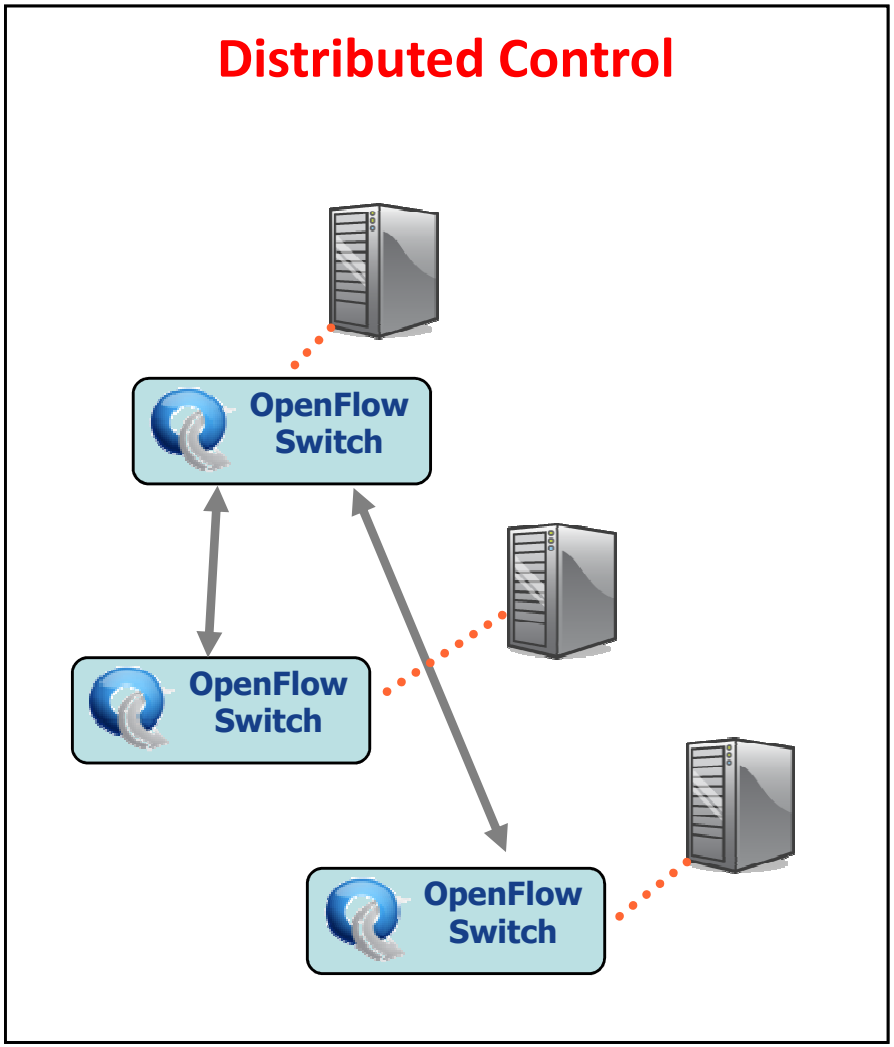
- The OpenFlow controller instructs switches about how they should process packets



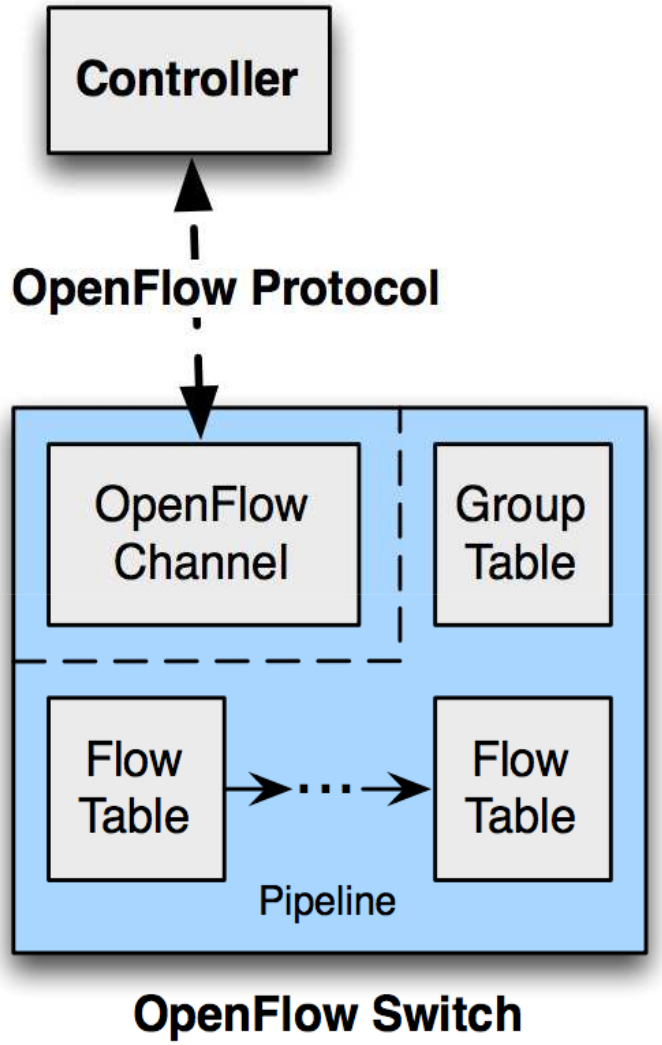
OpenFlow: centralized vs. distributed control

- Both models are possible with OpenFlow
  - Distributed control to reduce switch-controller latency and to avoid performance problems and a single-point-of-failure





OpenFlow switch: components



In current OpenFlow switches, Flow Tables are implemented by leveraging existing hardware components such as TCAMs (ternary content-addressable memory)

OpenFlow datapath

- The OpenFlow specification defines three types of tables in the logical switch architecture
  1. A *Flow Table* matches incoming packets to a particular flow and specifies the functions that are to be performed on the packets
     - There may be multiple flow tables that operate in a pipeline fashion
  2. A flow table may direct a flow to a *Group Table,* which may trigger a variety of actions that affect one or more flows
  3. A *Meter Table* can trigger a variety of performance-related actions on a flow
- An OpenFlow switch process packets by associating them to ***flows***
- In general terms, a flow is a sequence of packets traversing a network that share a set of header field values
  - Curiously, this term is not defined in the OpenFlow specification

OpenFlow: Secure Channel (SC)

- SC is the interface that connects each OpenFlow switch to controller
- A controller configures and manages the switch via this interface
  - Receives events from the switch
  - Send packets out the switch
- SC establishes and terminates the connection between OpenFlow Switch and the controller using the procedures
  - Connection Setup
  - Connection Interrupt
- The SC connection is a TLS connection
  - Switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key

OpenFlow: ports

- OpenFlow switches are connected through OpenFlow ports
  - Network interfaces to exchange packets with the rest of the network
- Types:
  - **Physical Ports**
    - Switch defined ports correspond to a hardware interface (e.g., map one-to-one to the Ethernet interfaces)
  - **Logical Ports**
    - Switch defined ports that do not correspond to a hardware switch interface (e.g. Tunnel-ID)
  - **Reserved Ports**
    - Defined by ONF 1.4.0
    - specify generic forwarding actions such as sending to the controller, flooding and forwarding using non-OpenFlow methods, such as normal switch processing

Ports - Reserved Port Types (Required)

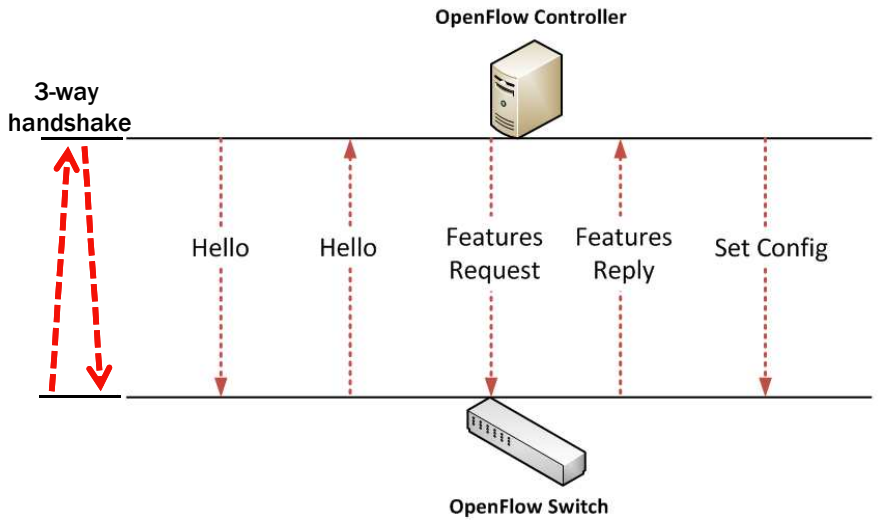
- ALL
  - Represents all ports the switch can use for forwarding a specific packets
  - Can be used only as output interface
- CONTROLLER
  - Represents the control channel with the OpenFlow controller
  - Can be used as an ingress port or as an output port
- TABLE
  - Represents the start of the OpenFlow pipeline
  - Submits the packet to the first flow table
- IN_PORT
  - Represents the packet ingress port
  - Can be used only as an output port
- ANY
  - Special value used in some OpenFlow commands when no port is specified
  - Can neither be used as an ingress port nor as an output port

Ports - Reserved Port Types (Optional)

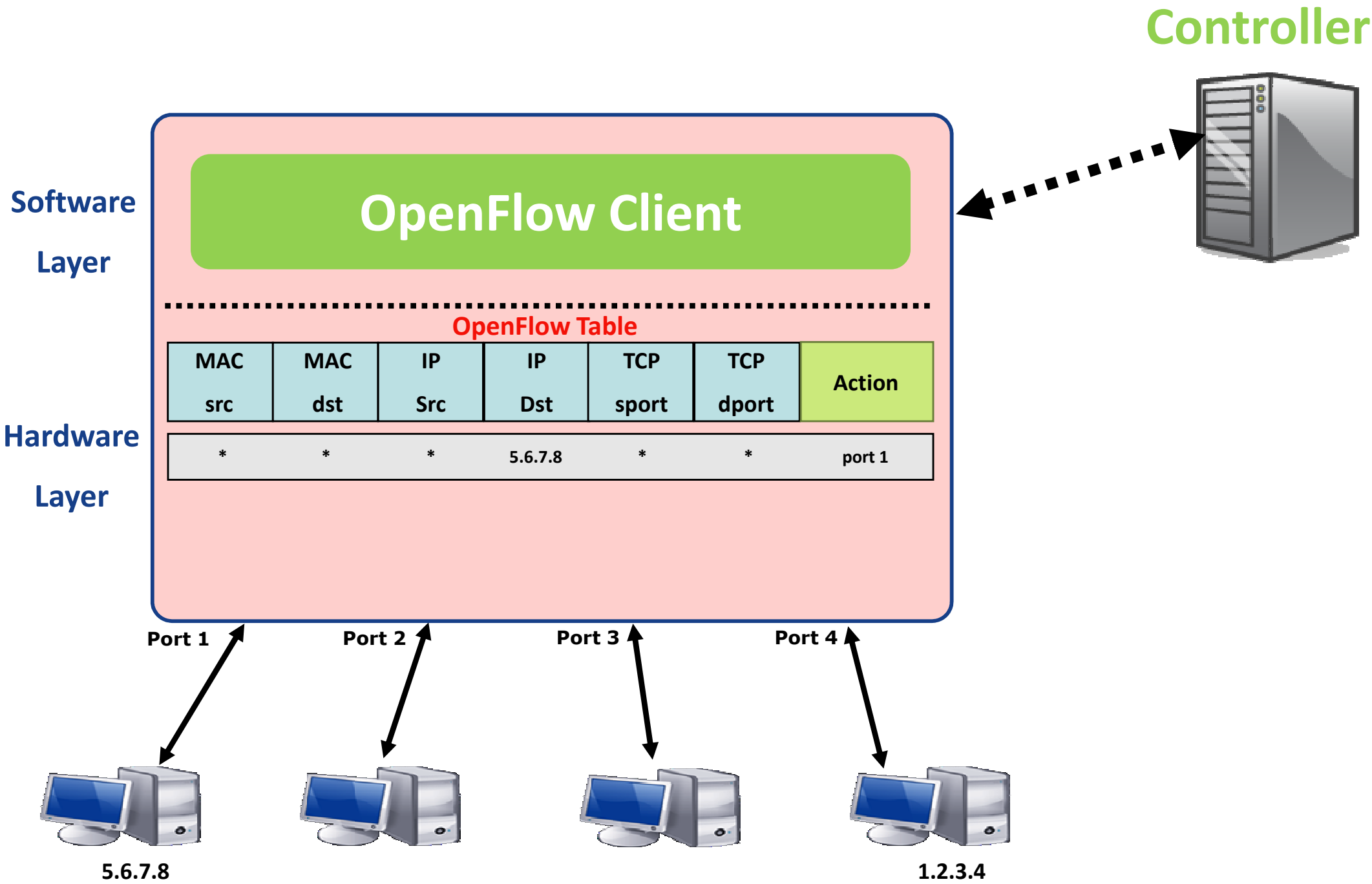
- LOCAL
  - Represents the switch's local networking stack and its management stack
  - Can be used as an ingress port or as an output port
- NORMAL
  - Represents the traditional non-OpenFlow pipeline of the switch
  - Can be used only as an output port and processes the packet using the normal pipeline
- FLOOD
  - Represents flooding using the normal pipeline
  - Can be used only as an output port
  - Send the packet out on all ports except the incoming port and the ports that are in blocked state

OpenFlow switch – Controller interactions

- An OpenFlow switch establishes a TCP connection to its Controller
  - An openFlow Controller by default listen on TCP port 6653 since OpenFlow 1.4.0
  - It used to be TCP port 6633 in previous OF versions
- Then the Controller starts an exchange of messages with the switch



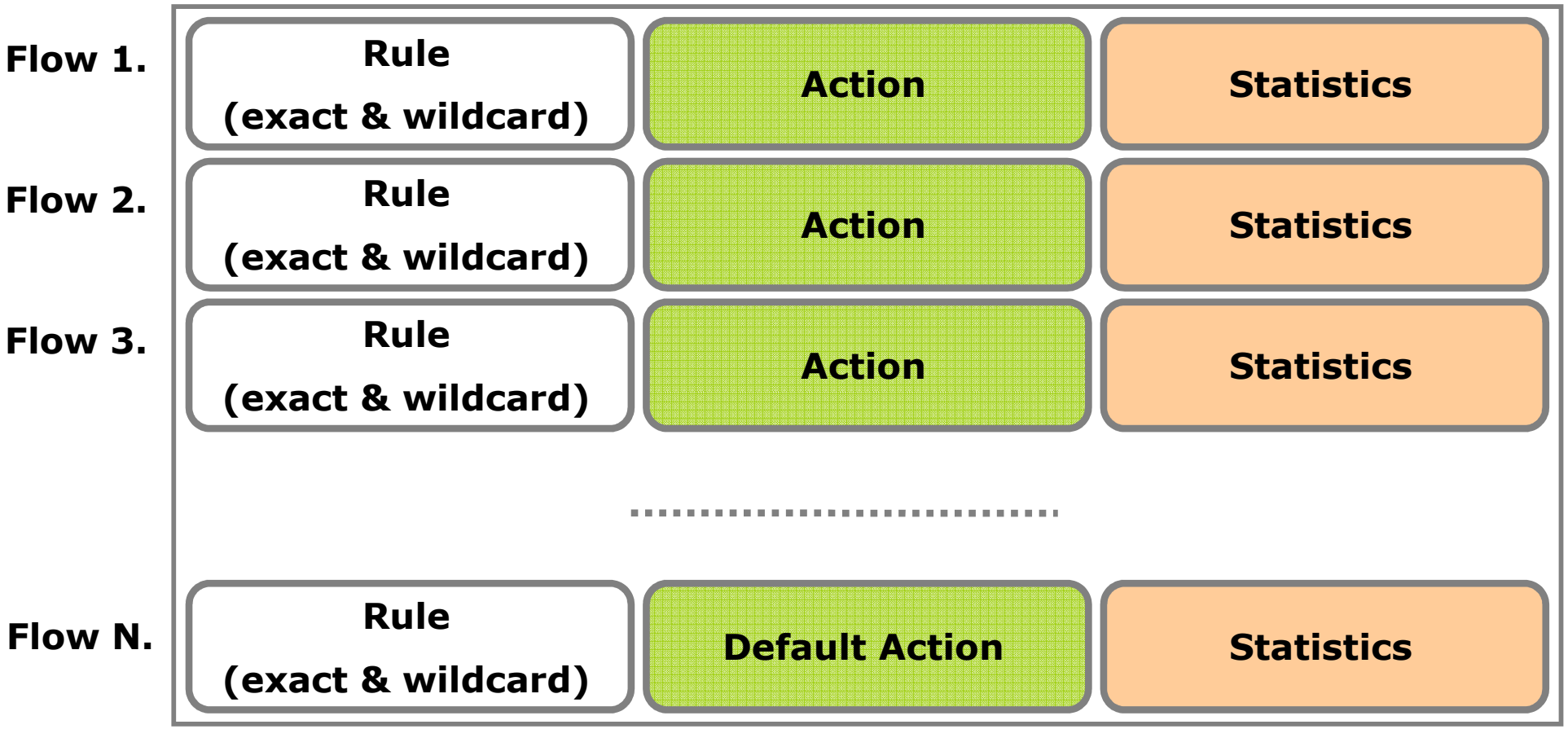
OpenFlow switching



OpenFlow switching

- The datapath of an OpenFlow Switch is governed by a Flow Table
- The control path consists of a Controller which programs the Flow Table
- The Flow Table consists of a number of ***flow entries***
- Each ***Flow Entry*** consists of
  - Match Fields
    - Match against packets
  - Action
    - Modify the action set or pipeline processing
  - Stats
    - Update the matching packets
- A Flow Table may include a **table-miss Flow Entry**, which renders all Match Fields wildcards (every field is a match regardless of value) and has the lowest priority (priority 0)

Flow Table


Flow 1.
Rule
(exact & wildcard)
Action
Statistics
Flow 2.
Rule
(exact & wildcard)
Action
Statistics
Flow 3.
Rule
(exact & wildcard)
Action
Statistics
Flow N.
Rule
(exact & wildcard)
Default Action
Statistics

OpenFlow: packet processing

Match: 1000x01xx0101001x

Match arbitrary bits in headers:

- Match on any header, or new header
- Allows any flow granularity

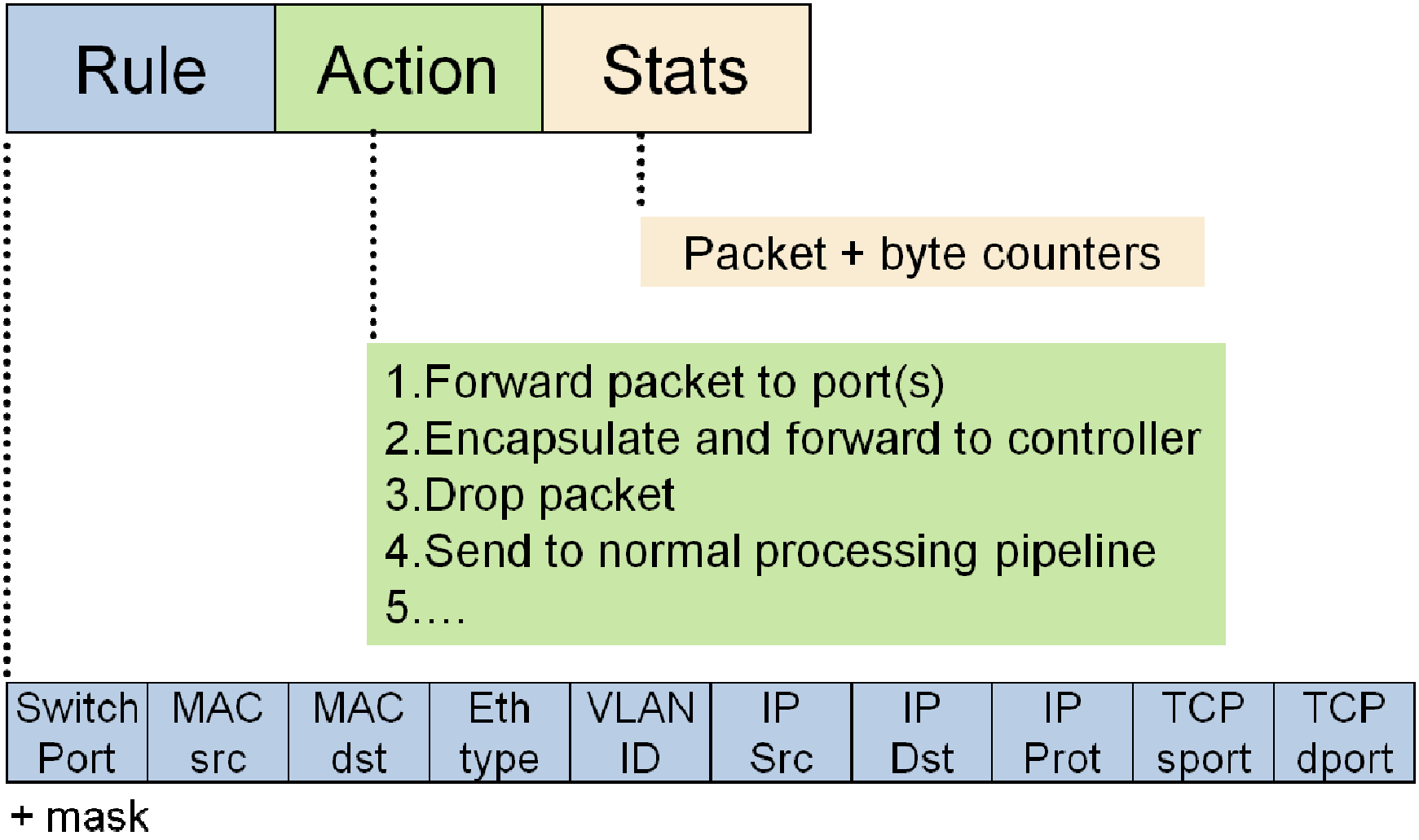
Action

- Forward to port(s), drop, send to controller
- Modify header
- …

OpenFlow actions

- Forward this flow's packets to a given port
  - This action allows packets to be routed
- Encapsulate and forward this flow's packets to a controller
  - This action allows controller to decide whether the flow should be added to the Flow Table
- Drop this flow's packets
  - This action can be used for security reasons, etc.
- Forward this flow's packets through the switch's normal processing pipeline (optional)
  - This action allows experimental traffic to be isolated from production traffic
  - Alternatively, isolation can be achieved through defining separate sets of VLANs
  - We can also treat OpenFlow as generalization of VLAN!
- Actions associated with flow entries may also direct packets to a ***group***
  - Groups represent sets of actions for flooding, as well as more complex forwarding semantics (e.g. multipath, fast reroute, and link aggregation)
  - As a general layer of indirection, groups also enable multiple flow entries to forward to a single identifier (e.g. IP forwarding to a common next hop)
  - This abstraction allows common output actions across flow entries to be changed efficiently

OpenFlow flow entry



Data-Plane: Simple Packet Handling

- Simple packet-handling rules
  - Pattern: match packet header bits
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets

1. IP_src=1.2.*.*, IP_dest=3.4.5.* → drop
2. IP_src = *.*.*.*, IP_dest=3.4.*.* → forward to port 2
3. IP_src=10.1.2.3, IP_dest=*.*.*.* → send to controller

Overlapping rules !

OpenFlow examples

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

Routing

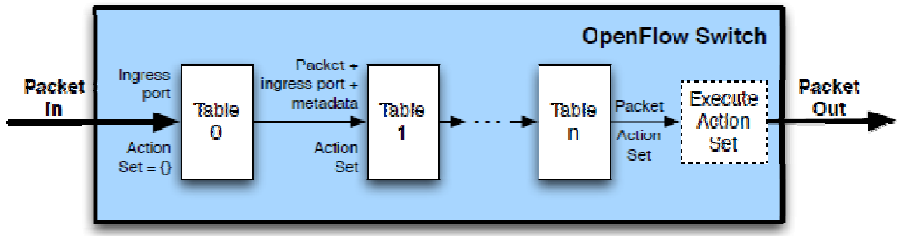
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

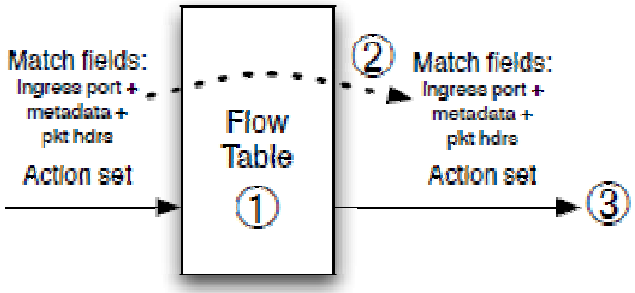
Flow Table pipelining (1)

- A switch includes one or more Flow Tables
- If there is more than one Flow Table, they are organized as a *pipeline*
- When a packet is presented to a Table for matching, the input consists of
  - the packet,
  - the identity of the ingress port,
  - the associated metadata value,
  - and the associated action set



(a) Packets are matched against multiple tables in the pipeline

- For Table 0, metadata value is blank and action set is null
- Each incoming packet is processed according to Flow Table entries
- A Flow Table entry may explicitly direct the packet to another Flow Table (using the Goto Instruction), where the same process is repeated again
- A flow entry can only direct a packet to a Flow Table number which is greater than its own flow table no.
  - Flow entries of the last Table of the pipeline cannot include the Goto instruction
- If the matching flow entry does not direct packets to another Flow Table, processing stops at this table. When pipeline processing stops, packet is processed with its associated action set and usually forwarded



① Find highest-priority matching flow entry

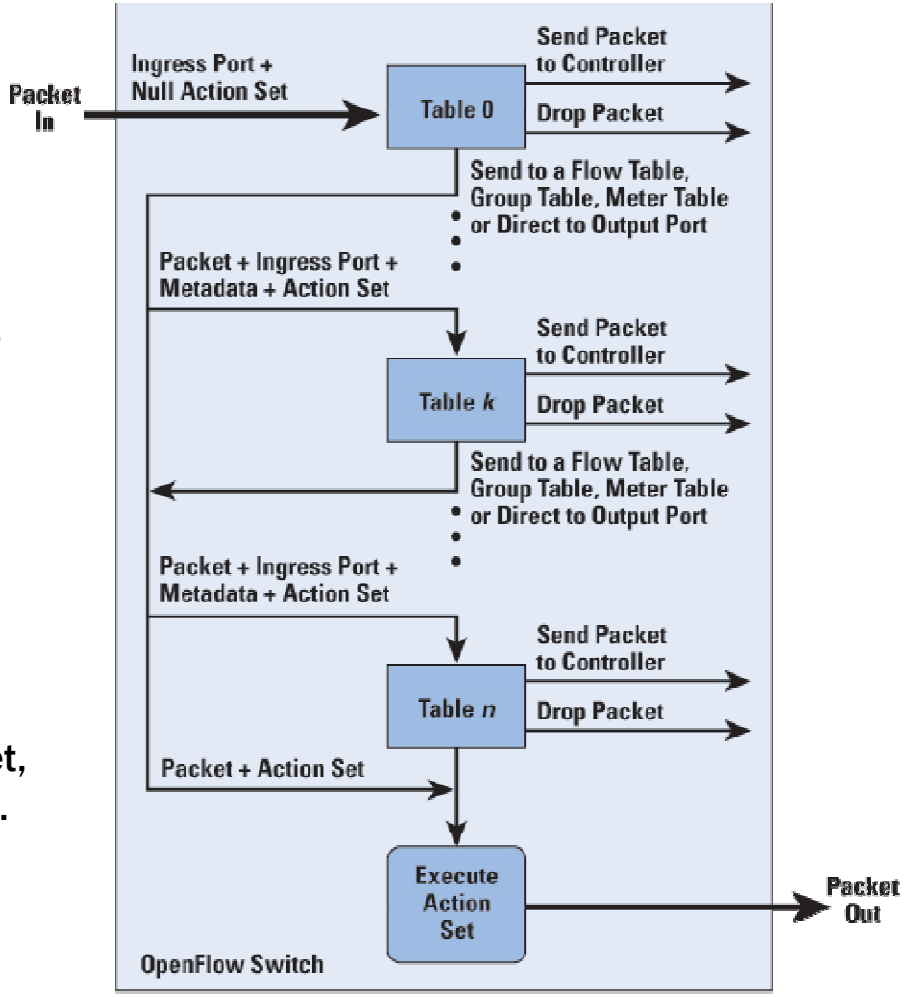
② Apply instructions:
- i. Modify packet & update match fields (apply actions instruction)
- ii. Update action set (clear actions and/or write actions instructions)
- iii. Update metadata

③ Send match data and action set to next table

(b) Per-table packet processing

Flow Table pipelining (2)

- At each table, find the highest-priority matching flow entry
  1. If there is no match on any entry and there is no table-miss entry, then the packet is dropped
  2. If there is a match only on a table-miss entry, then that entry specifies one of three actions:
     - Send packet to controller.
       This action will enable the controller to define a new flow for this and similar packets, or decide to drop the packet
     - Direct packet to another flow table farther down the pipeline
     - Drop the packet
  3. If there is a match on one or more entries other than the table-miss entry, then the match is defined to be with the highest-priority matching entry.
     The following actions may then be performed:
     - Update any counters associated with this entry.
     - Execute any instructions associated with this entry. These instructions may include updating the action set, updating the metadata value, and performing actions.
     - The packet is then forwarded to a flow table further down the pipeline, to the group table, or to the meter table, or it could be directed to an output port.
- If and when a packet is finally directed to an output port, the accumulated action set is executed and then the packet is queued for output



OpenFlow: Flow Routing vs. Aggregation

- Both models are possible with OpenFlow
  - Aggregated rules are necessary to cope with the hardware limit on number of entries imposed by current TCAMs

Flow-Based

- Every flow is individually set up by controller
- Exact-match flow entries
- Flow table contains one entry per flow
- Good for fine grain control, e.g. campus networks

Aggregated

- One flow entry covers large groups of flows
- Wildcard flow entries
- Flow table contains one entry per category of flows
- Good for large number of flows, e.g. backbone

OpenFlow: Reactive vs. Proactive management

- Both models are possible with OpenFlow

Reactive

- First packet of flow triggers controller to insert flow entries
- Efficient use of flow table
- Every flow incurs small additional flow setup time
- If control connection lost, switch has limited utility

Proactive

- Controller pre-populates (*a priori*) flow table in switch
- Zero additional flow setup time
- Loss of control connection does not disrupt traffic
- Essentially requires aggregated (*wildcard*) rules

OpenFlow software switches

- Open vSwitch: Open Source and popular
- Of13softswitch: User-space software switch based on Ericsson TrafficLab 1.1
- Indigo: Open source implementation that runs on Mac OS X
- LINC: Open source implementation that runs on Linux, Solaris, Windows, MacOS, and FreeBSD
- Pantou: Turns a commercial wireless router/access point to an OpenFlow enabled switch. Supports generic Broadcom and some models of LinkSys and TP-Link access points with Broadcom and Atheros chipsets

OpenFlow Controllers: first wave

Name	Lang	Platform(s)	License	Original Author	Notes
OpenFlow Reference	C	Linux	OpenFlow License	Stanford/Nicira	not designed for extensibility
NOX	Python, C++	Linux	GPL	Nicira	
POX	Python	Any	Apache	Murphy McCauley (UC Berkeley)	
Ryu	Python	Linux	Apache	NSRC	Component based design Supports OpenStack integration
Trema	Ruby, C	Linux	GPL	NEC	includes emulator, regression test framework
Floodlight	Java	Any	Apache	BigSwitch Networks	
RouteFlow	?	Linux	Apache	CPqD (Brazil)	Special purpose controller to implement virtual IP routing as a service

OpenFlow controllers: new generation

- OpenDayLight
- ONOS

ONOS: Architecture Tiers

Northbound Abstraction:

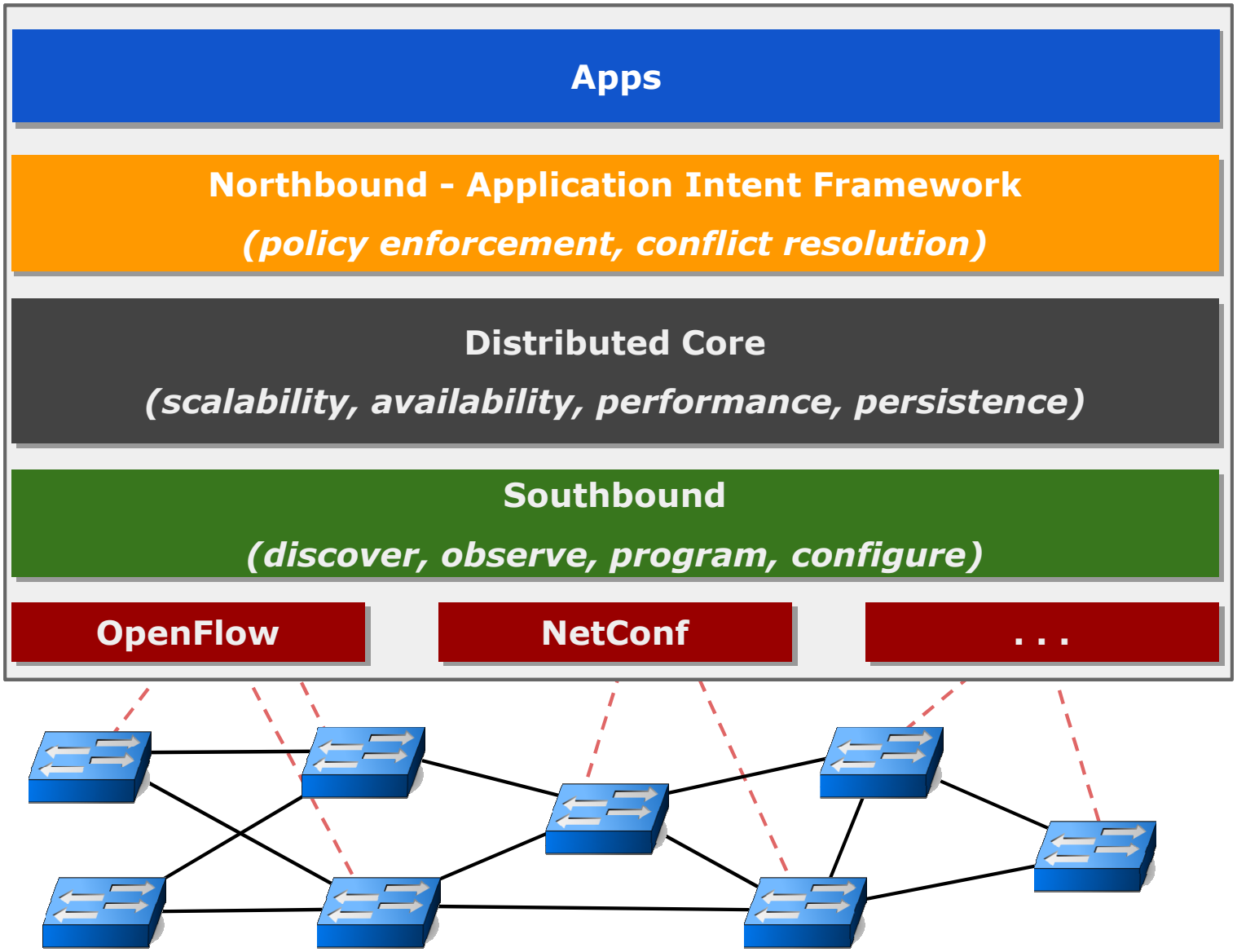
- network graph
- application intents

Core:

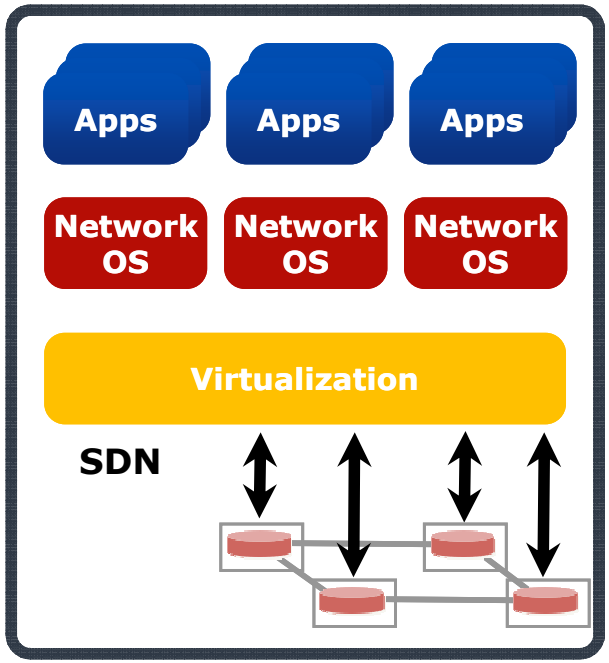
- distributed
- protocol independent

Southbound Abstraction:

- generalized OpenFlow
- pluggable & extensible



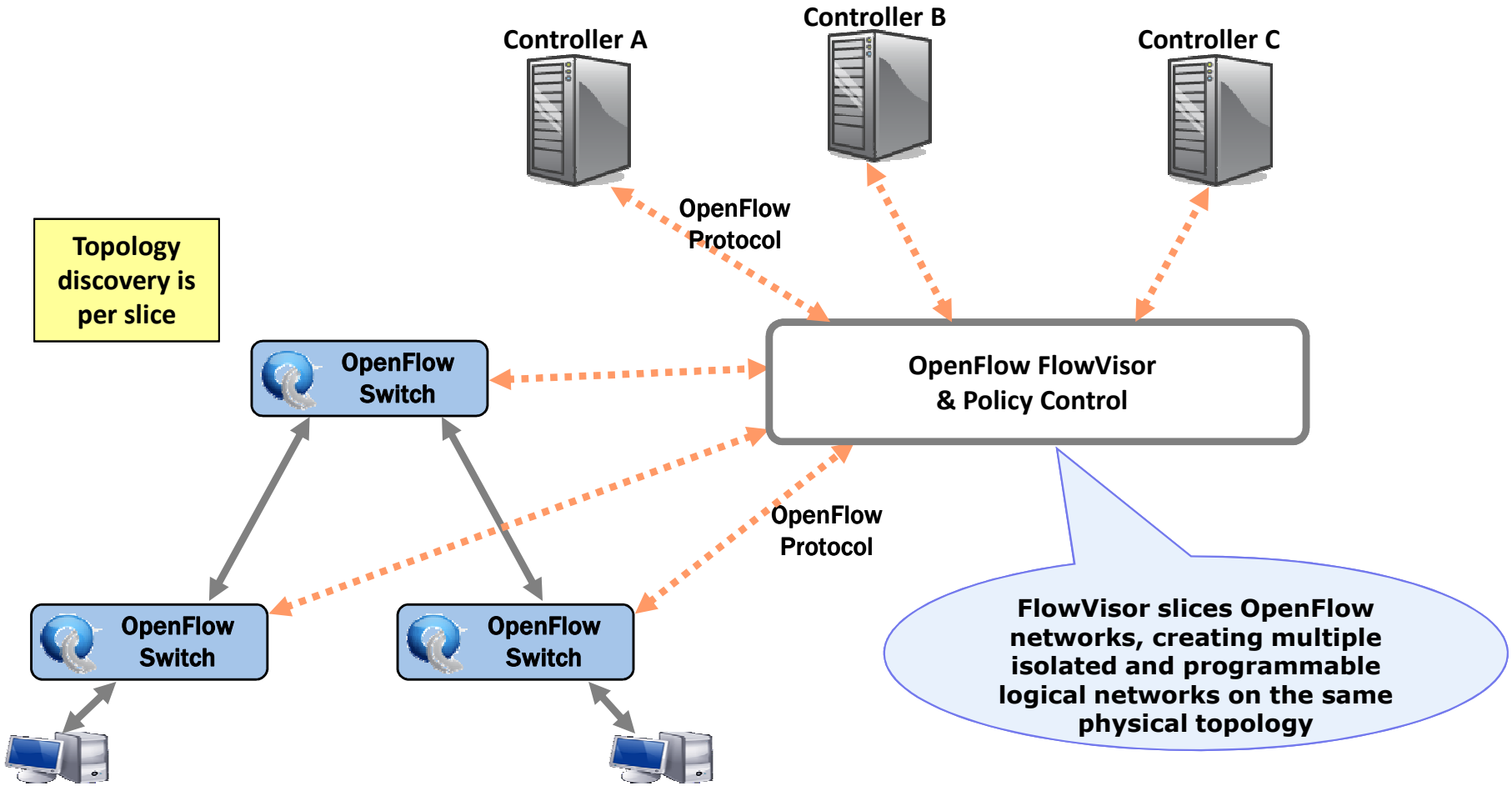
Virtualizing OpenFlow networks



- One of the goals of the SDN approach is to enable *Network Virtualization*, i.e. the possibility of creating and managing separately multiple logically-defined virtual infrastructures on top of a single shared substrate
- *FlowVisor* is a solution developed at Stanford University that allows network virtualization in the context of an OpenFlow network
- Network operators "delegate" control of subsets (*slices*) of network hardware and/or traffic to other network operators or users
- Multiple controllers can talk to the same set of switches
- FlowVisor is a software proxy between the forwarding and control planes of network devices
- FlowVisor intercepts OpenFlow messages from devices
  - FV only sends control plane messages to the Slice controller if the source device is in the Slice topology
  - Rewrites OF feature negotiation messages so the slice controller only sees the ports in its slice
  - Port up/down messages are pruned and only forwarded to affected slices
- Likewise, FlowVisor intercepts OpenFlow messages from controllers to preserve slice isolation

Network virtualization with OpenFlow and FlowVisor

- Slices are defined using a ***slice definition policy***
- The policy language specifies the slice's resource limits, flowspace, and controller's location in terms of IP and TCP port-pair
- FlowVisor enforces transparency and isolation between slices by inspecting, rewriting, and policing OpenFlow messages as they pass

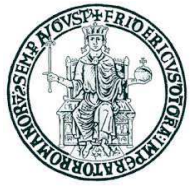


Cloud e Datacenter Networking

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

- 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

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)

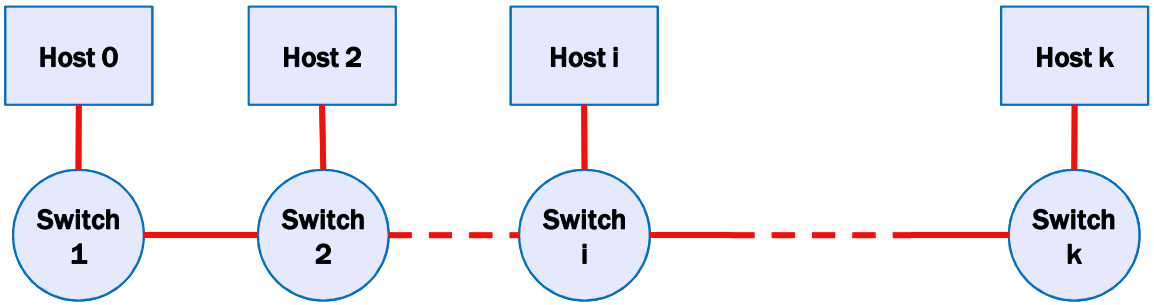
```
        lastSwitch = None
        for i in irange(1, k):
            host = self.addHost('h%s' % i, cpu=.5/k)
            switch = self.addSwitch('s%s' % i)
            # 10 Mbps, 5ms delay, 1% loss, 1000 packet queue
            self.addLink( host, switch, bw=10, delay='5ms', loss=1,
max_queue_size=1000,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

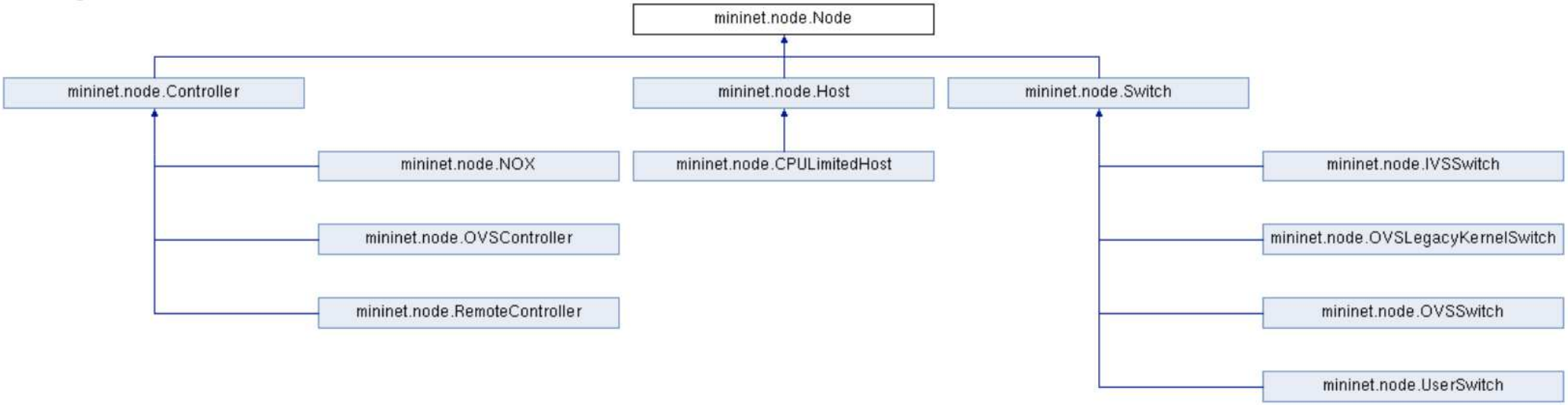
```
def perfTest():
   "Create network and run simple performance test"
   topo = LinearTopo(k=4)
   net = Mininet(topo=topo,
                 host=CPULimitedHost, link=TCLink)
   net.start()

   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 classes

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

Inheritance diagram for mininet.node.Node:

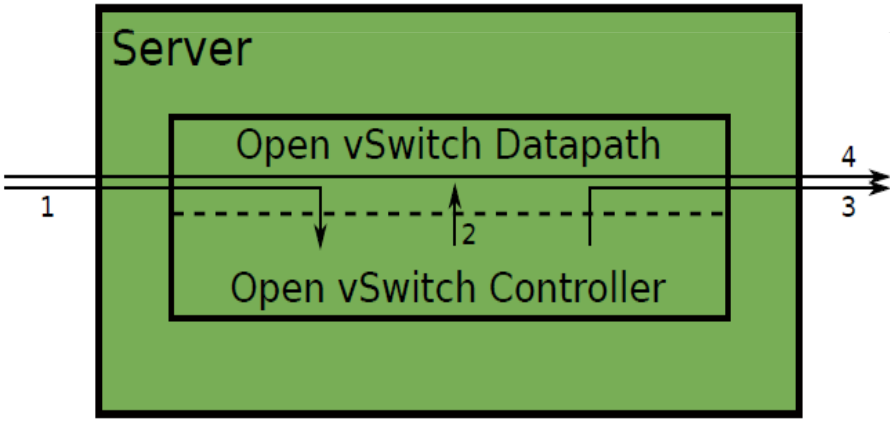


OpenFlow Tutorials

- 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

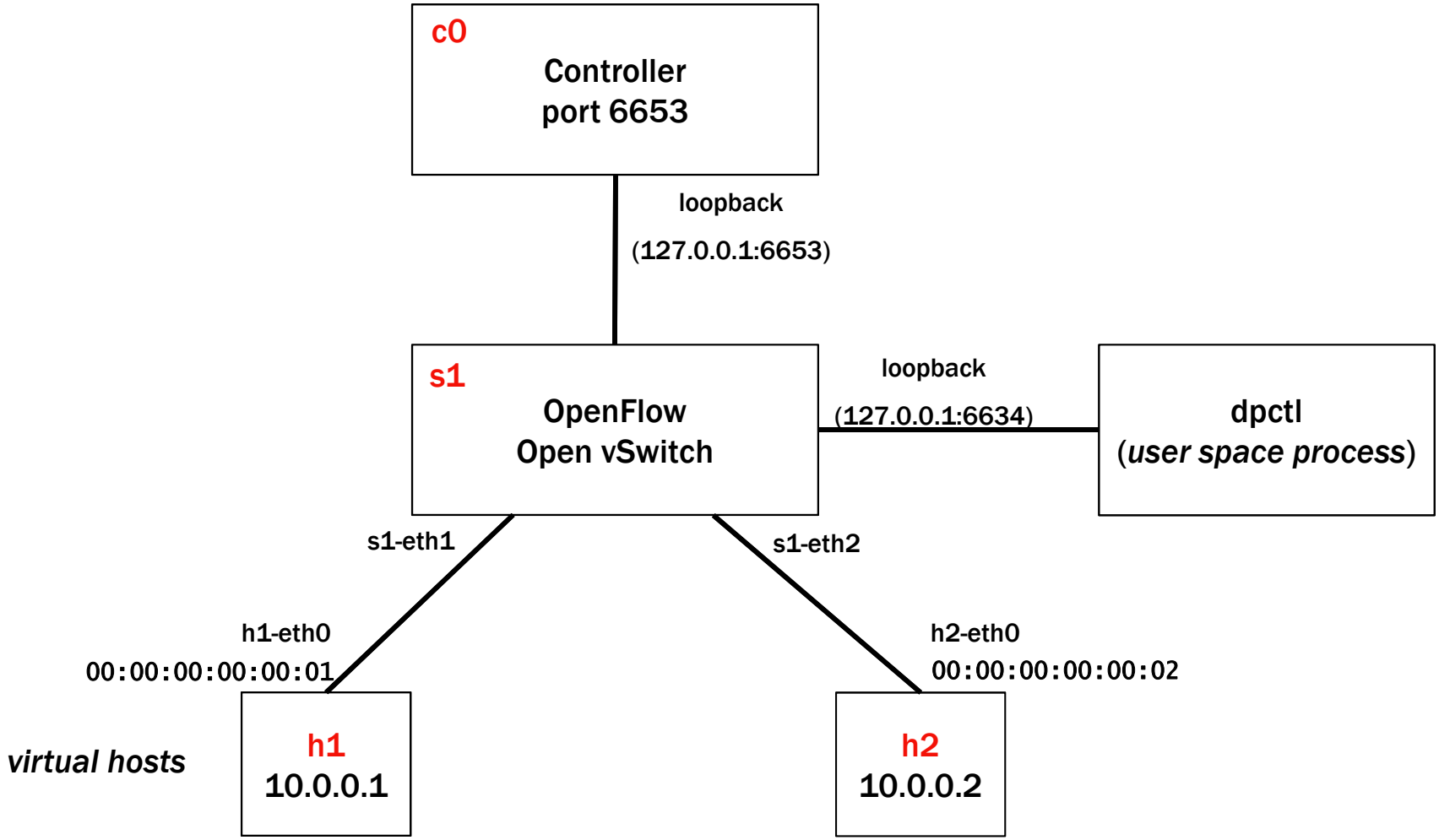
Open vSwitch as an OpenFlow software switch

- 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

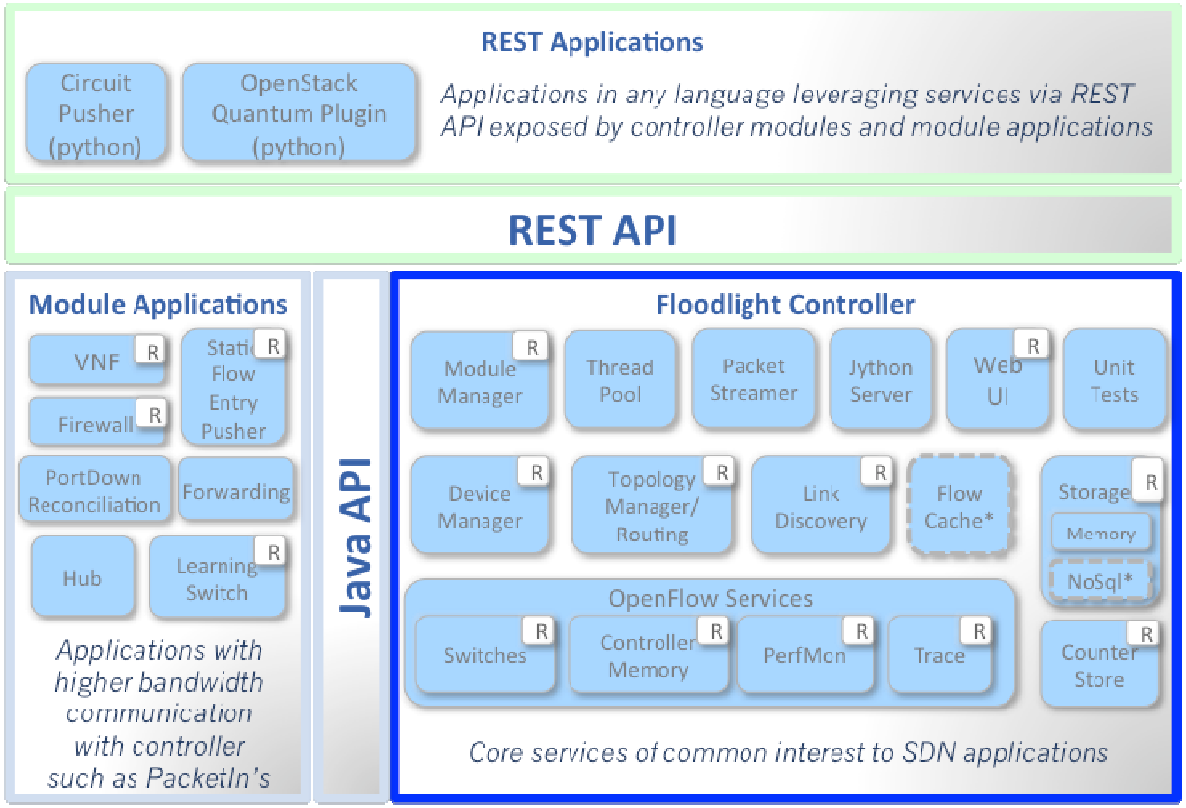
```
$ sudo mn --topo single,2 --mac --switch ovsk --controller remote,port=6653
```

- 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

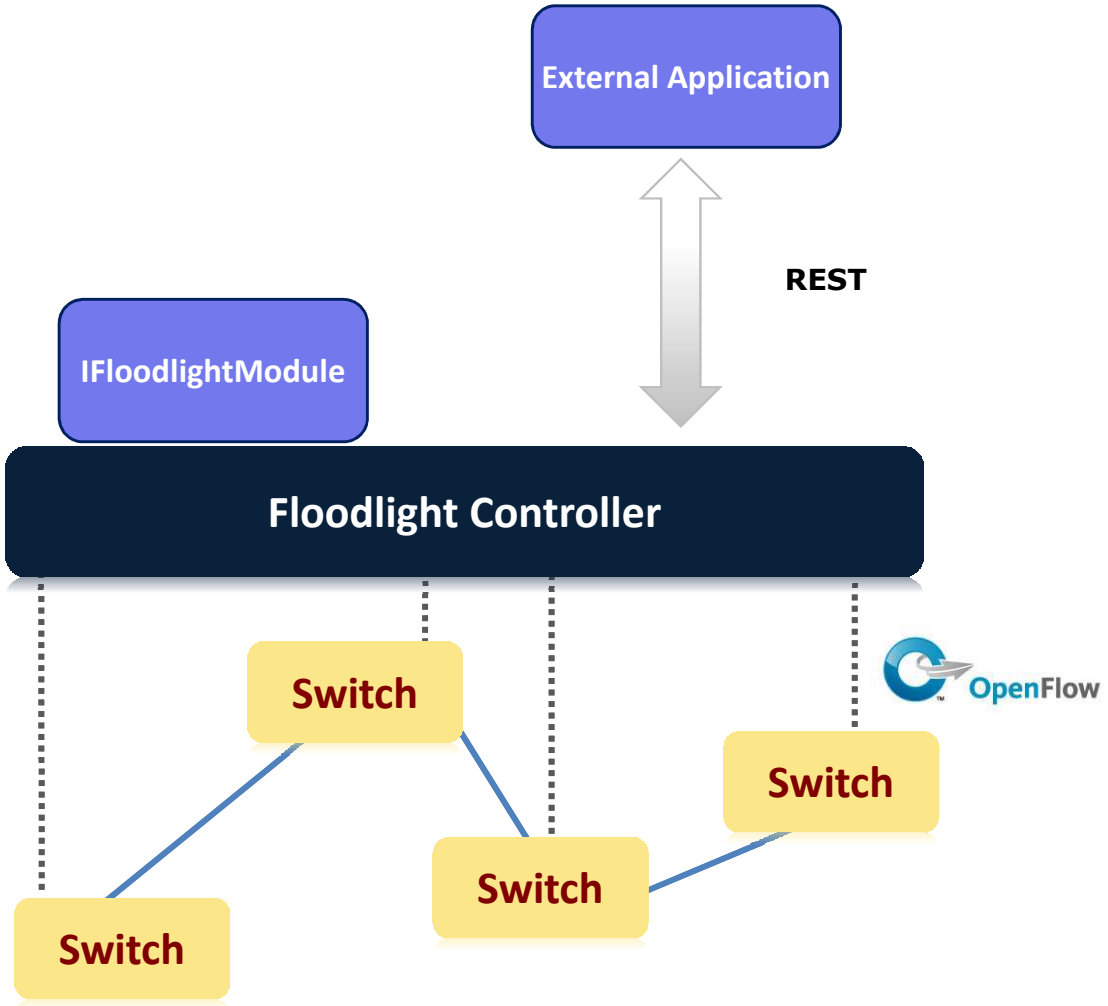
Northbound APIs

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

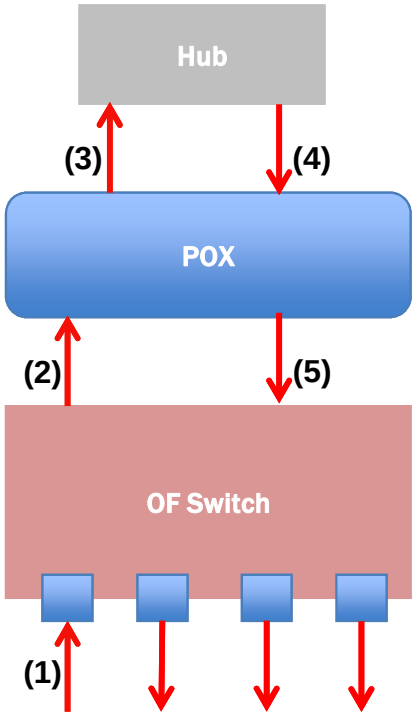
```
import httplib
import json

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

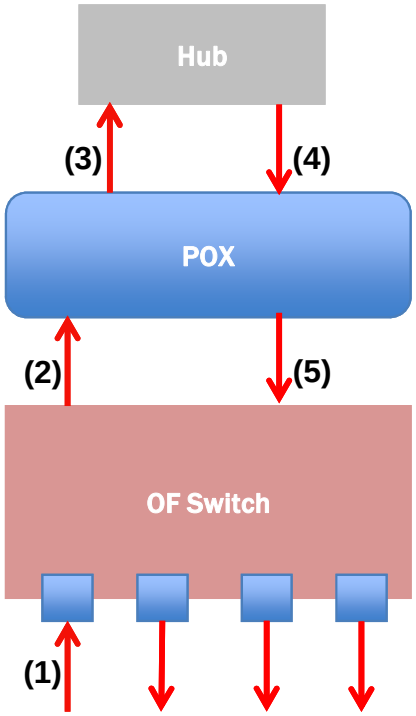
```
public class PktInHistory implements IFloodlightModule {

        @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 #1: 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