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

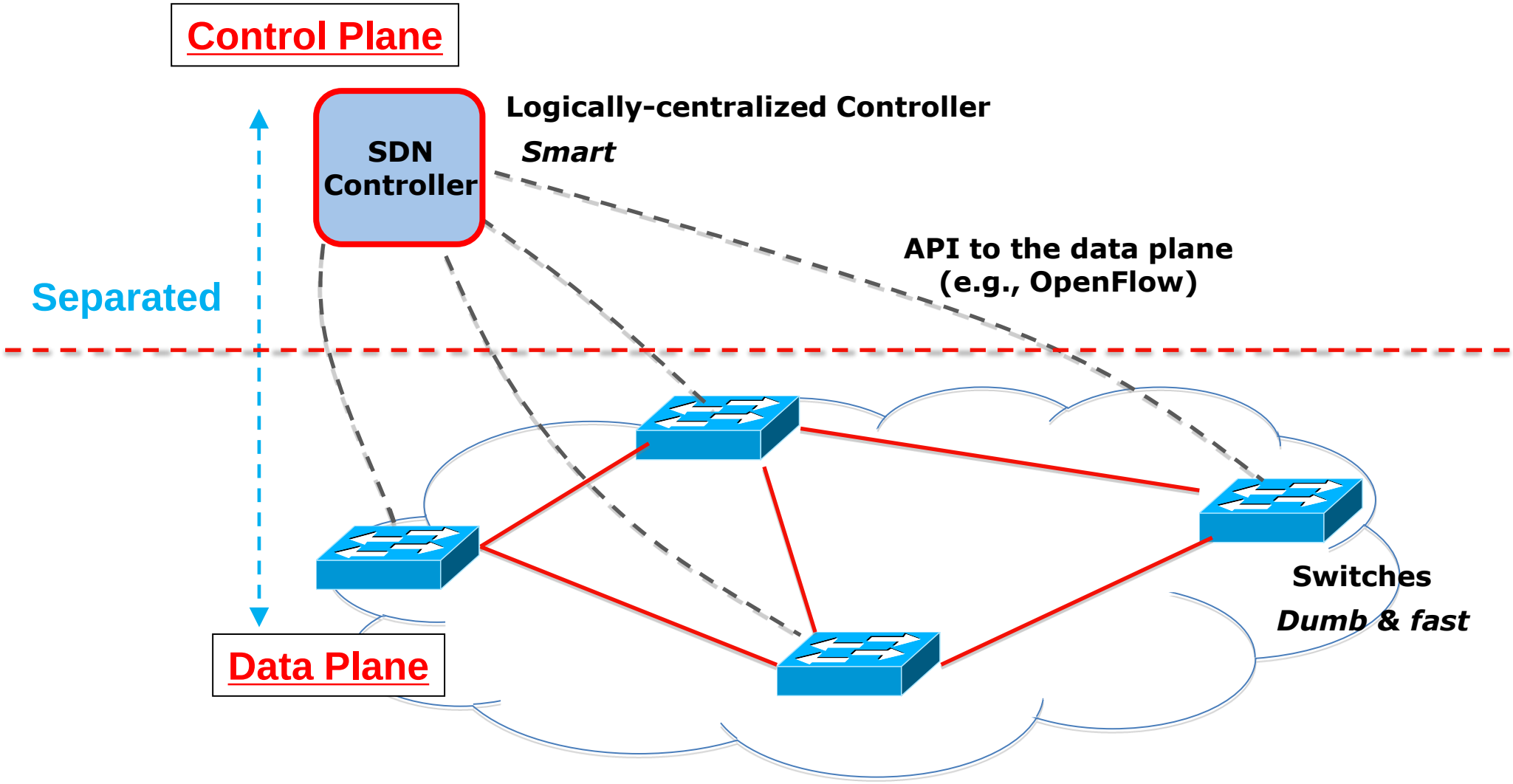
Lesson outline

- 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 2014
  - Version 1.5.1 Apr 2015

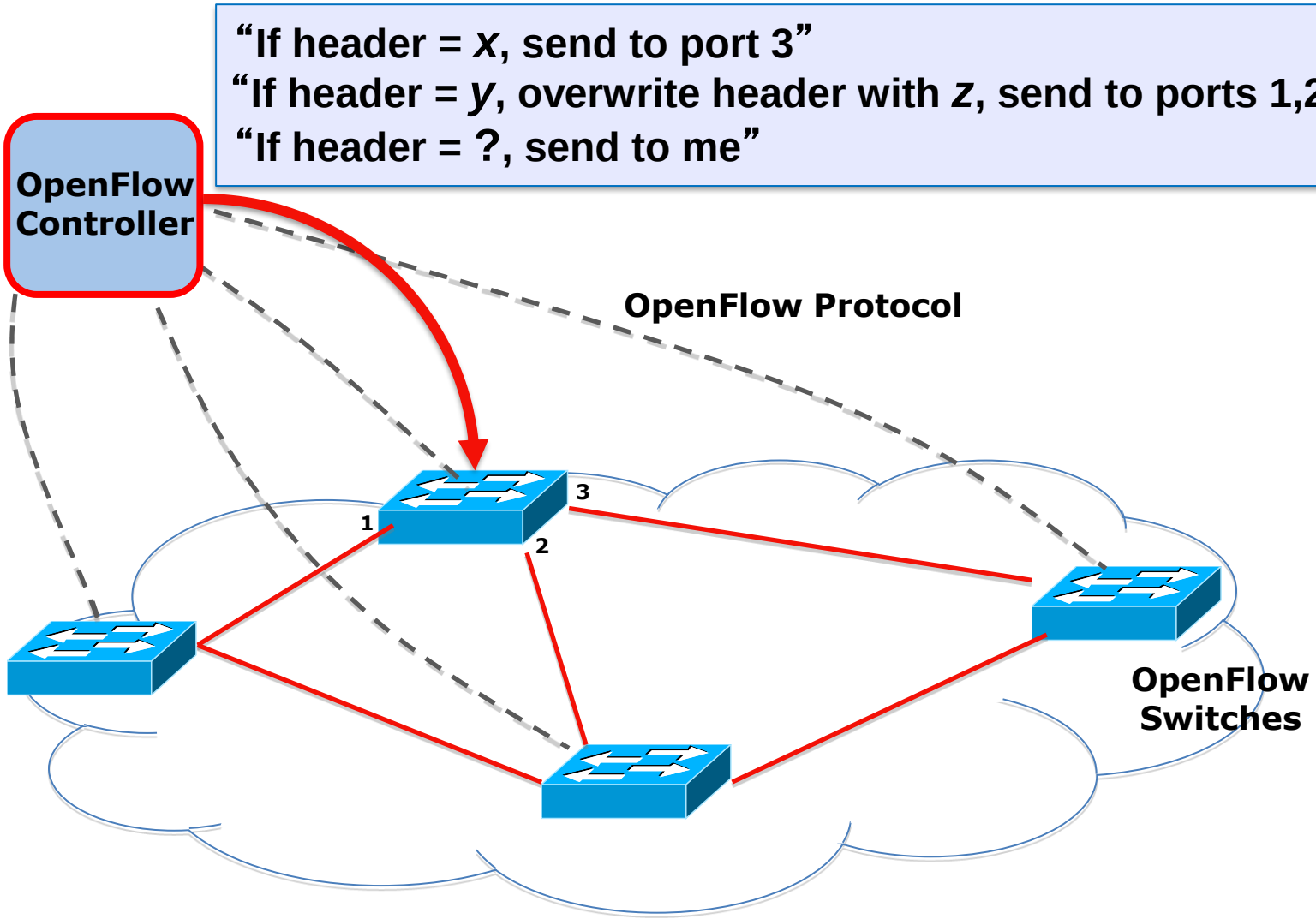
Open Networking Foundation (ONF)

https://www.opennetworking.org/

- The Open Networking Foundation (ONF) is a non-profit operator-led consortium driving transformation of network infrastructure and carrier business models
- Open, collaborative, community of communities
- Produce OpenFlow Switch Specification, Reference Designs and whitepapers
- The ONF serves as the umbrella for a number of projects building solutions by leveraging network disaggregation, white box economics, open source software and software defined standards to revolutionize the carrier industry

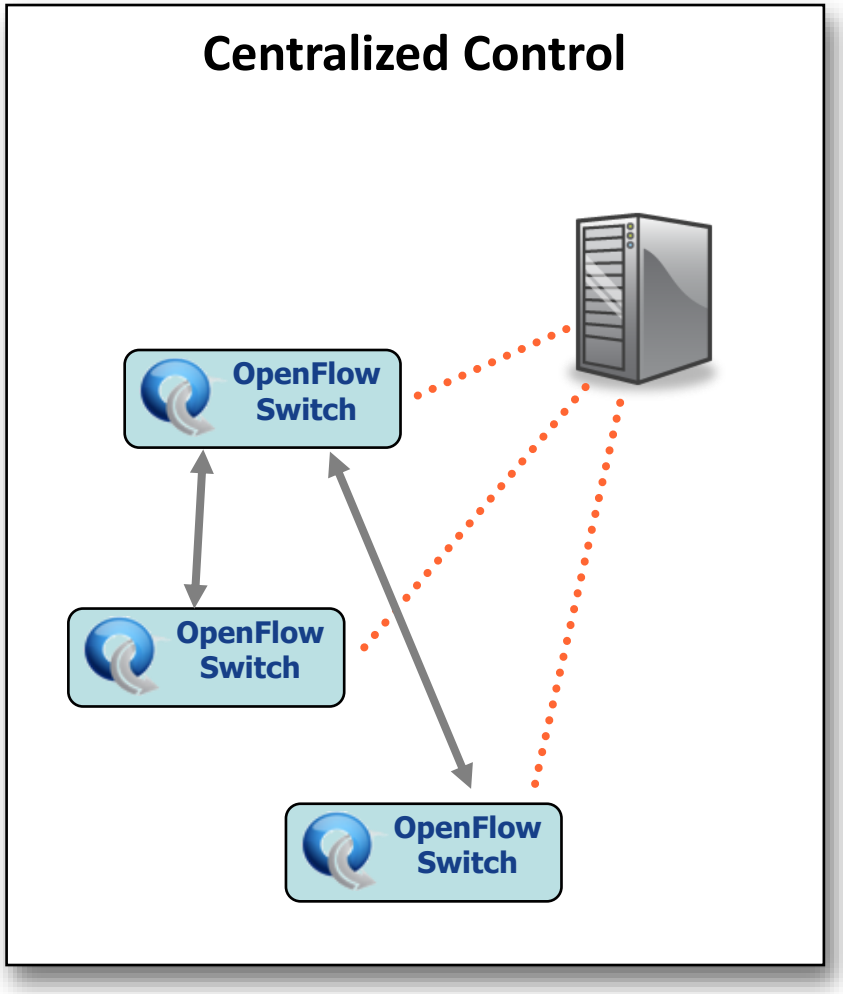
OpenFlow network model

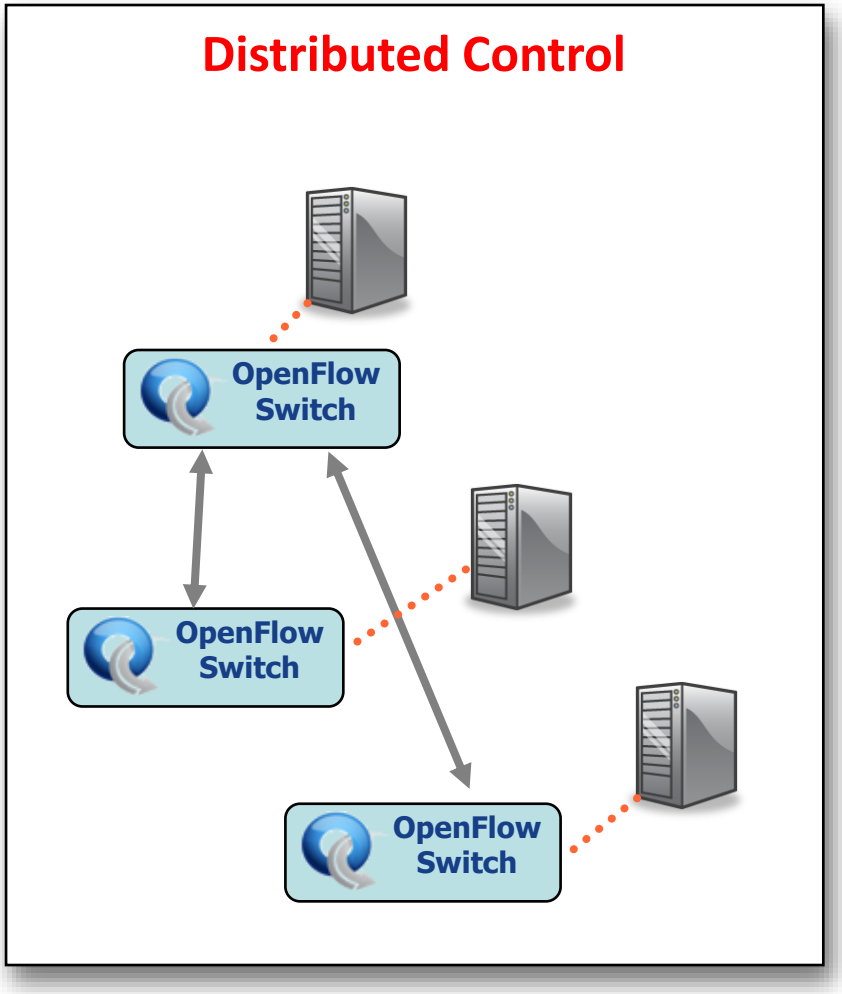
- The OpenFlow controller instructs switches about how they should process packets



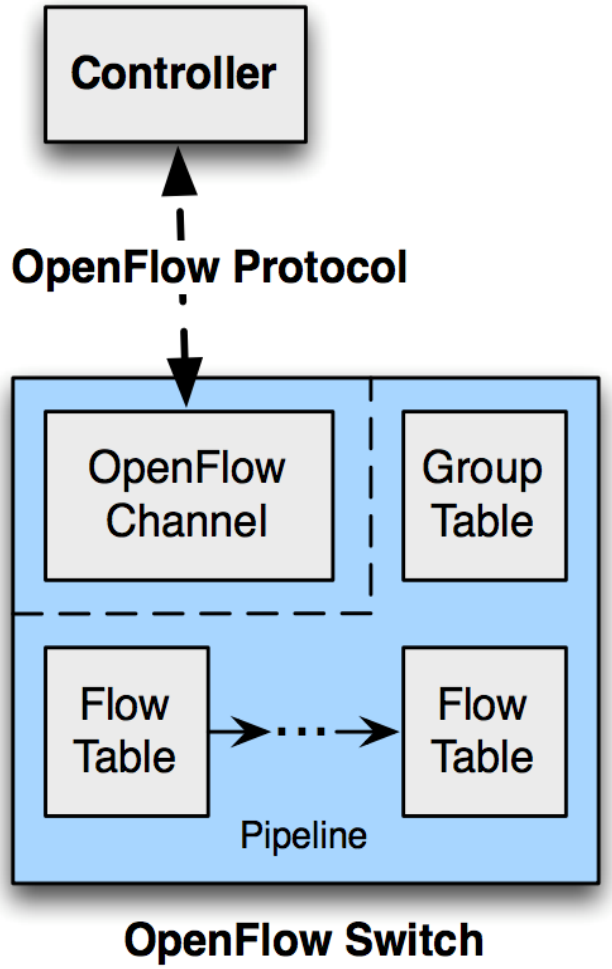
OpenFlow: centralized vs. distributed control

- 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 Switch specifications

- OpenFlow 1.0 (TS-001) – December 2009
  - https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf
- OpenFlow 1.1 (TS-002) – February 2011
  - https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf
- OpenFlow 1.2 (TS-003) – December 2011
  - https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf
- OpenFlow 1.3.0 (TS-006) – June 2012
  - https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf
    - OpenFlow 1.3.1 (TS-007) – September 2012
    - …
    - OpenFlow 1.3.5 (TS-023) – April 2015 [LINK]
- OpenFlow 1.4.0 (TS-012) – October 2013
  - https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf
    - OpenFlow 1.4.1 (TS-024) – April 2015 [LINK]
- OpenFlow 1.5.0 (TS-020) – December 2014
  - https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.pdf
    - OpenFlow 1.5.1 (TS-025) – April 2015 [LINK]

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)

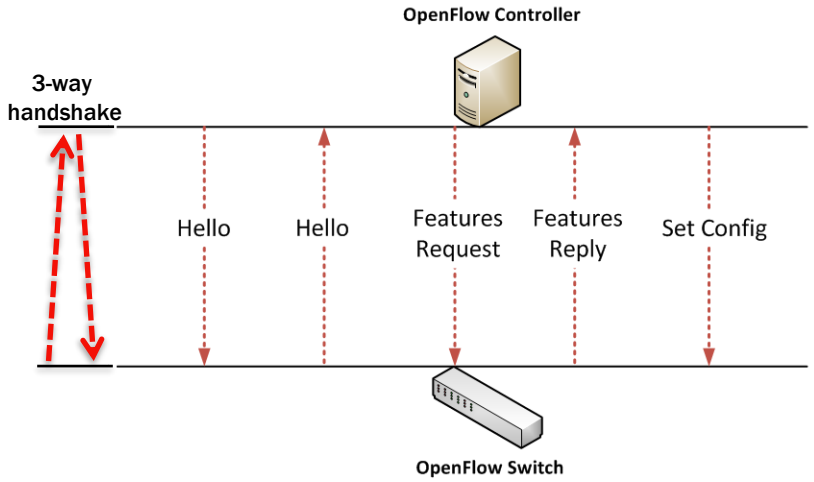
- ALL
  - Represents all ports the switch can use for forwarding a specific packets
  - Can be used only as output interface
- CONTROLLER
  - Represents the control channel with the OpenFlow controller
  - Can be used as an ingress port or as an output port
- TABLE
  - Represents the start of the OpenFlow 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

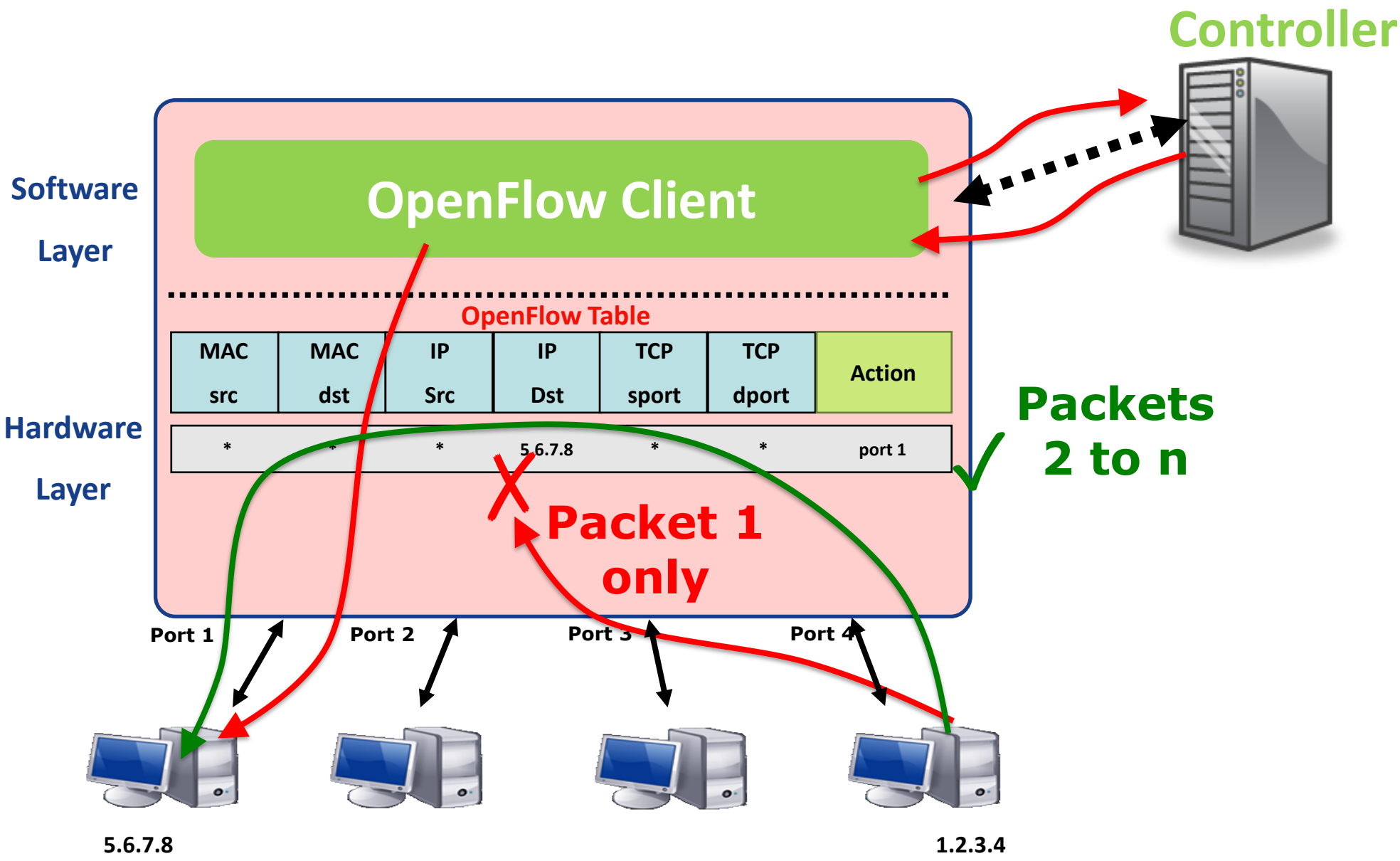
OpenFlow switch – Controller interactions

- 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
- Header of Openflow messages exchanged between switch and controller:

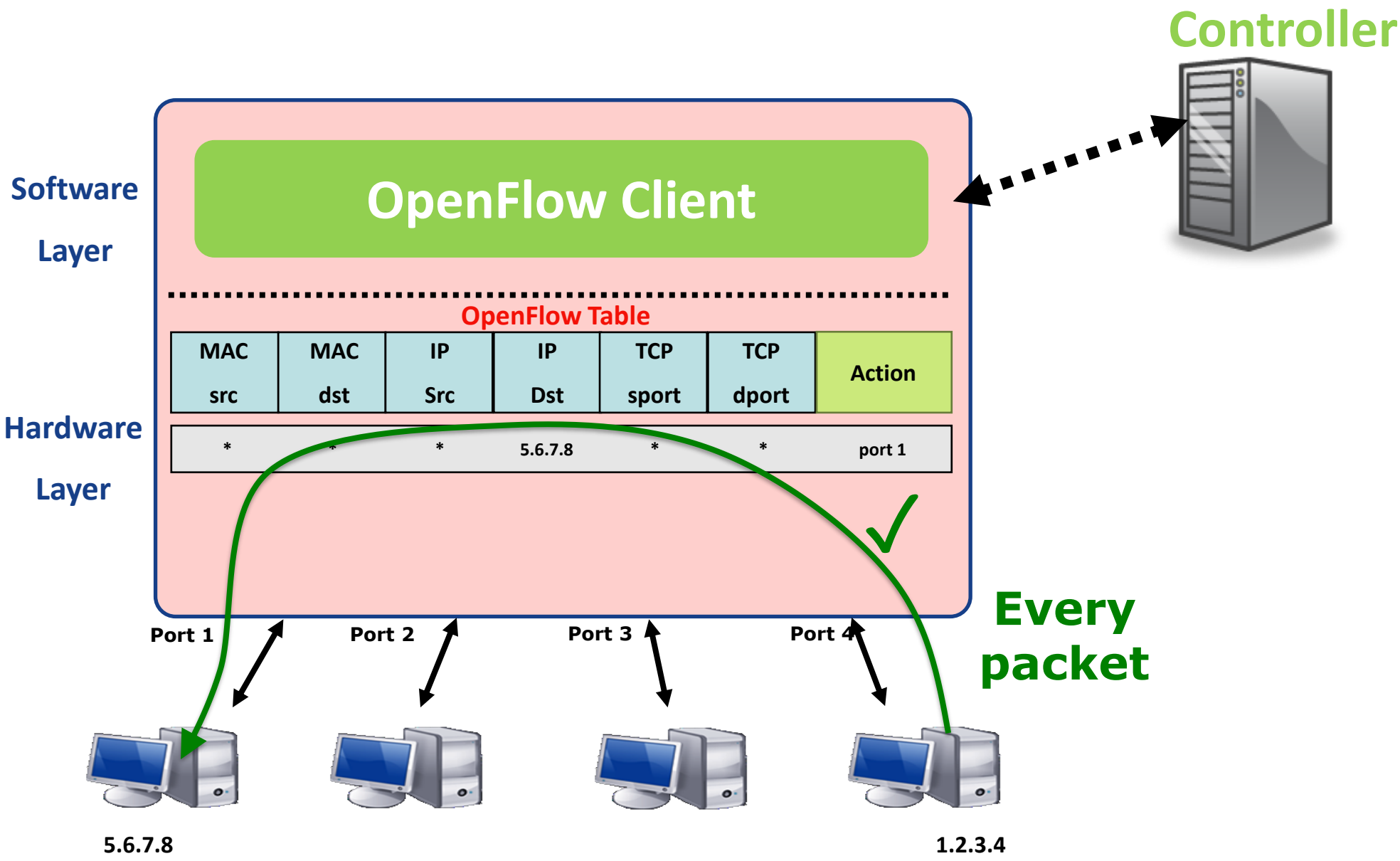



```
/* Header on all OpenFlow packets. */
struct ofp_header {
            uint8_t version;    /* OFP_VERSION. */
            uint8_t type;       /* one of the OFPT_ constants.*/
            uint 16_t length;   /* Length including this ofp_header. */
            uint32_t xid;       /* Transaction id associated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

OpenFlow switching with reactive packet processing



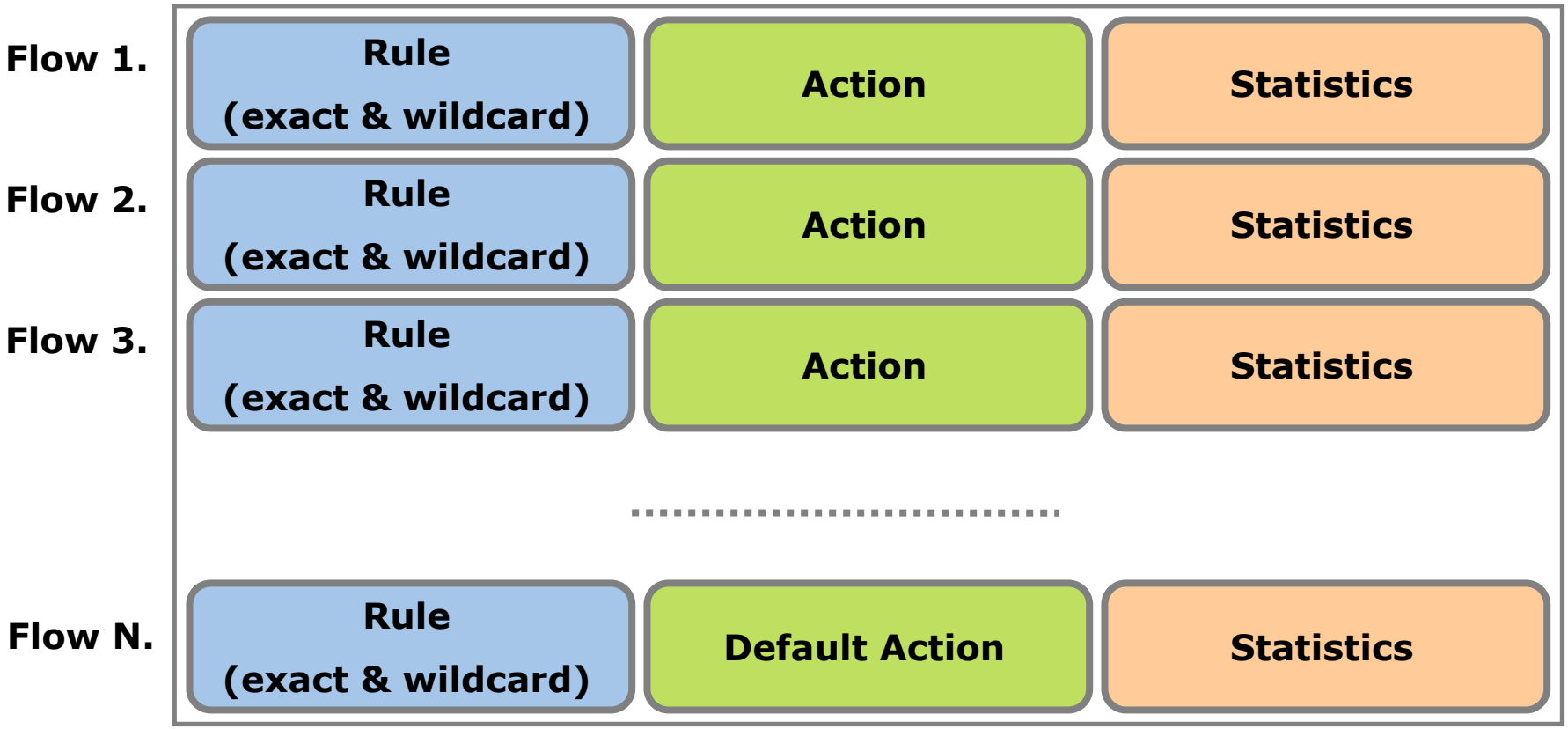
OpenFlow switching with proactive packet processing



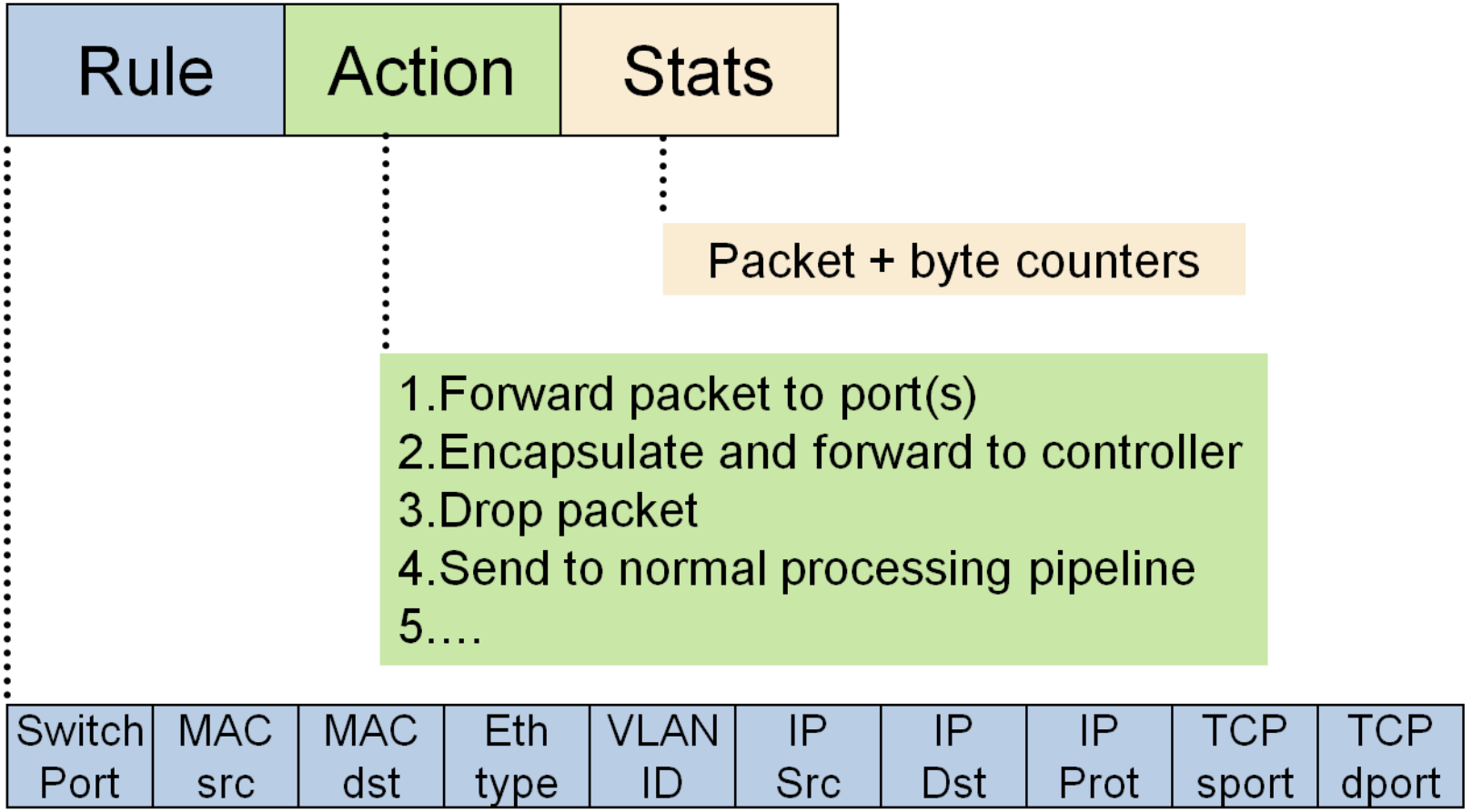
OpenFlow switching

- 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 flow entry



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 (Openflow 1.1+)***
  - 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

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

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 1.0 deals with three things

State:

What can software configure to match packets, and how is it represented?

The state is the flow table plus the port table, which has its own counters

single flow table + port table

match	action	counters
match	action	counters
:	:	:

port	counters
port	counters
:	:

Behavior:

Given a state, how can (and should) the switch forward or modify packets?

The behavior is match-one, with a default send-to-controller entry, plus a concept of flow expiration

match	action	counters
match	action	counters
:	:	:

match-one / send-to-controller, expire soft flows, modify or forward packets

Control Interface:

How do I describe desired changes to the switch state?

The interface is a single message queue.
It is optionally possible to add a barrier, which means "make sure every previous message was processed"

flow-mod
features request
packet-out
packet-in
echo-request
:

Limitations of OpenFlow 1.0

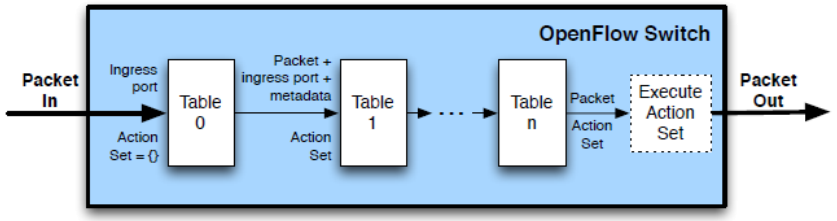
- Small table size
- Flow-space explosion
  - Since multiple independent header fields may affect a packet's forwarding, this may lead to the use of large number of flow entries in a single table
- Limited set of pre-defined fields for matching flows
  - Supported: MAC, VLAN, IP, L4 ports
  - Missing: IPv6, QinQ, MPLS, SCTP, optical circuits, ...
- Limited forwarding options
  - Supported: broadcast, multicast, drop
  - Missing: packet spreading, forwarding to a virtual port, general byte modifications
  - Useful for link aggregation, tunneling, etc.

OpenFlow evolution

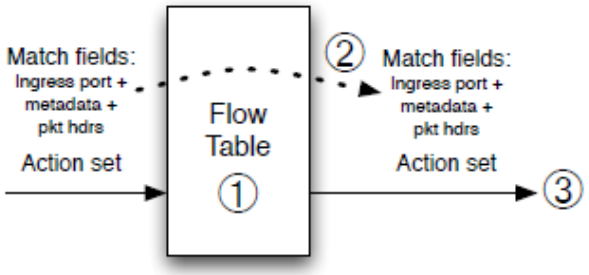
	state	behavior	interface msg
1.0 Q4 '09	flows ports	forward {0, 1, n} match Eth, VLAN, IP, L4	single message queue w/optional barriers
1.1 Q1 '11	+ Group Tables + Multiple Tables/Pipelines:	+ forward 1-in-n (ECMP) + match QinQ, MPLS, SCTP + match virtual ports	
1.2 Q4 '11		+ IPv6 + multiple controllers	+ extensible match + extensible actions
1.3 Q2 '12		+ per-flow metering + tunnel-id	+ multiple channels (auxiliary connections)
1.4 Q4 '13	+ optical ports + synchronized tables		+ bundle messages

OpenFlow 1.1+: 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
- 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



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



① Find highest-priority matching flow entry

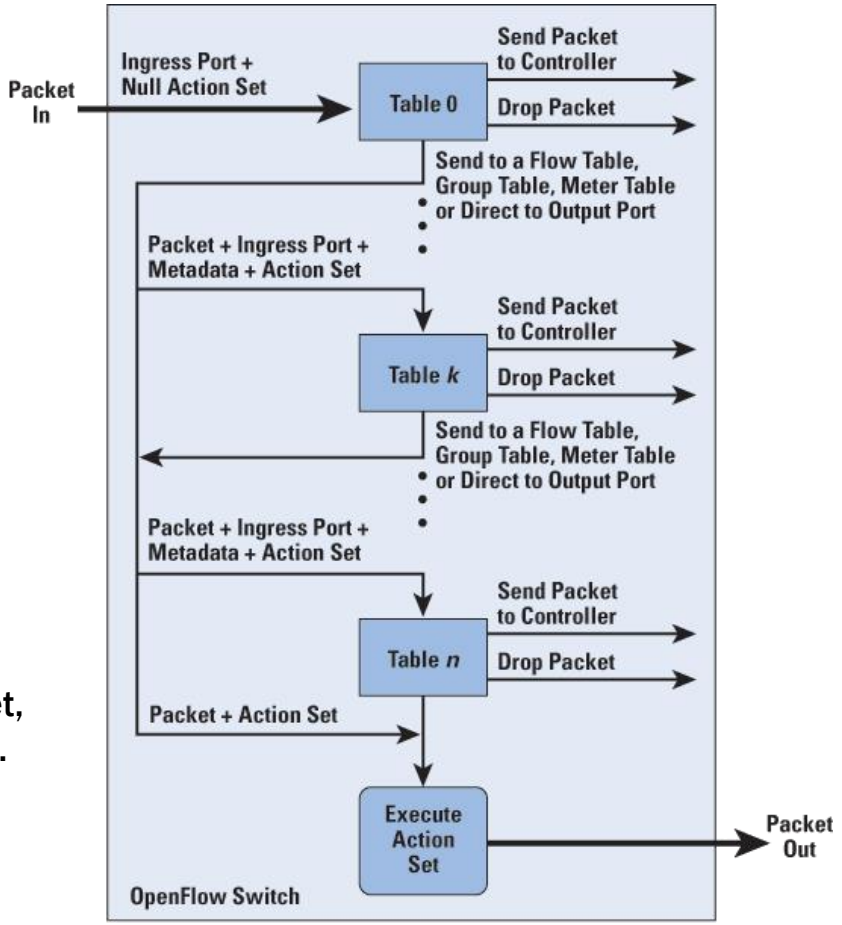
② Apply instructions:
- i. Modify packet & update match fields (apply actions instruction)
- ii. Update action set (clear actions and/or write actions instructions)
- iii. Update metadata

③ Send match data and action set to next table

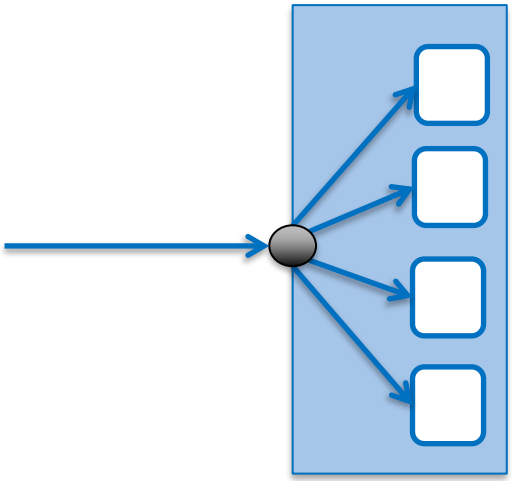
(b) Per-table packet processing

OpenFlow 1.1+: Flow Table pipelining (2)

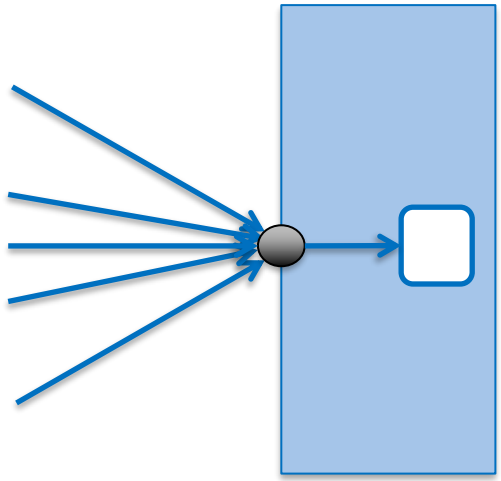
- At each table, find the highest-priority matching flow entry
  1. If there is no match on any entry and there is no table-miss entry, then the packet is dropped
  2. If there is a match only on a table-miss entry, then that entry specifies one of three actions:
     - Send packet to controller.
       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



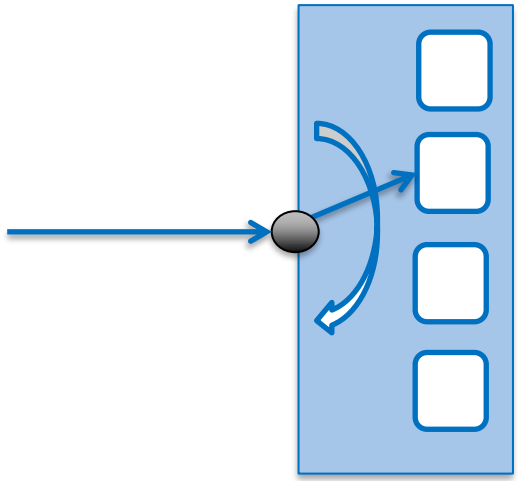
OpenFlow 1.1+: groups



Group *all*



Group *indirect*



Group *select*

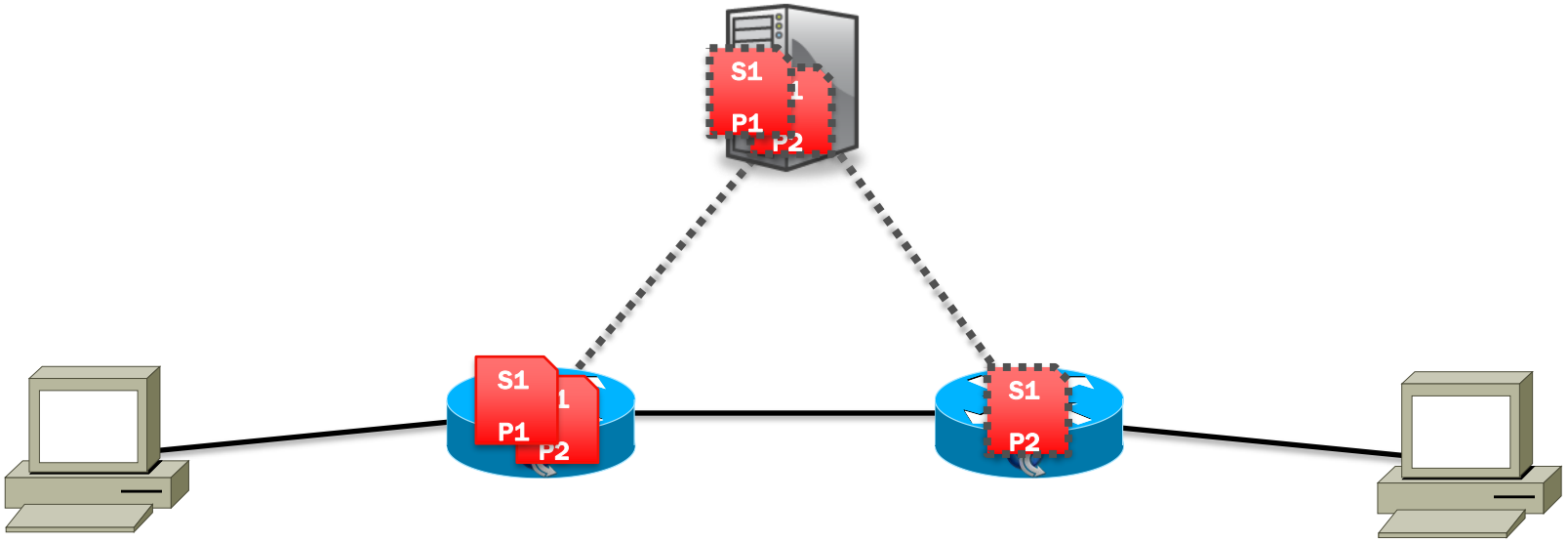
Group *fast-failover*

- Useful to reduce the number of flow entries when the same behavior should apply to multiple ports (e.g. for multicast, broadcast) or when the choice of the output port is directly delegated to the switch hardware (e.g. LAG, ECMP or for fast-failover)

How topology discovery works with LLDP

- Topology discovery in an OpenFlow network relies on periodic exchange of LLDP (*Link Layer Discovery Protocol*) packets between switches originated by the controller

Src Switch	Src Port	Dst Switch	Dst Port
S1	2	S2	1



S1 Flow Table

Eth Src	Eth Dst	Out Port

S2 Flow Table

Eth Src	Eth Dst	Out Port

OpenFlow 1.3 and support for IPv6

- OpenFlow 1.2 already allowed to match on IP protocol number (Ethernet type 0x86dd = IPv6), IPv6 source/destination address, traffic class, flow label, and ICMPv6 types/codes
- OpenFlow 1.3 allows to match on more IPv6 header fields such as source/destination address, protocol number (next header, extension header), hop-limit, traffic class, flow label, and ICMPv6 type/code (e.g. Neighbor Discovery Protocol (NDP))
- OpenFlow 1.3 also added the ability to rewrite packet headers via flexible match support (OXM)
  - Added three *new OpenFlow Extensible Match* (oxm) fields: MPLS BoS, PBB I-SID, TunnelID, and IPv6ExtHdr
  - IPv6ExtHdr indicates whether certain IPv6 header extensions are present: No Next Header, Encrypted ESP, Authentication header, 1 or 2 dest headers, fragment, router, hop-by-hop, unexpected repeats, and unexpected sequencing

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

SDN controllers

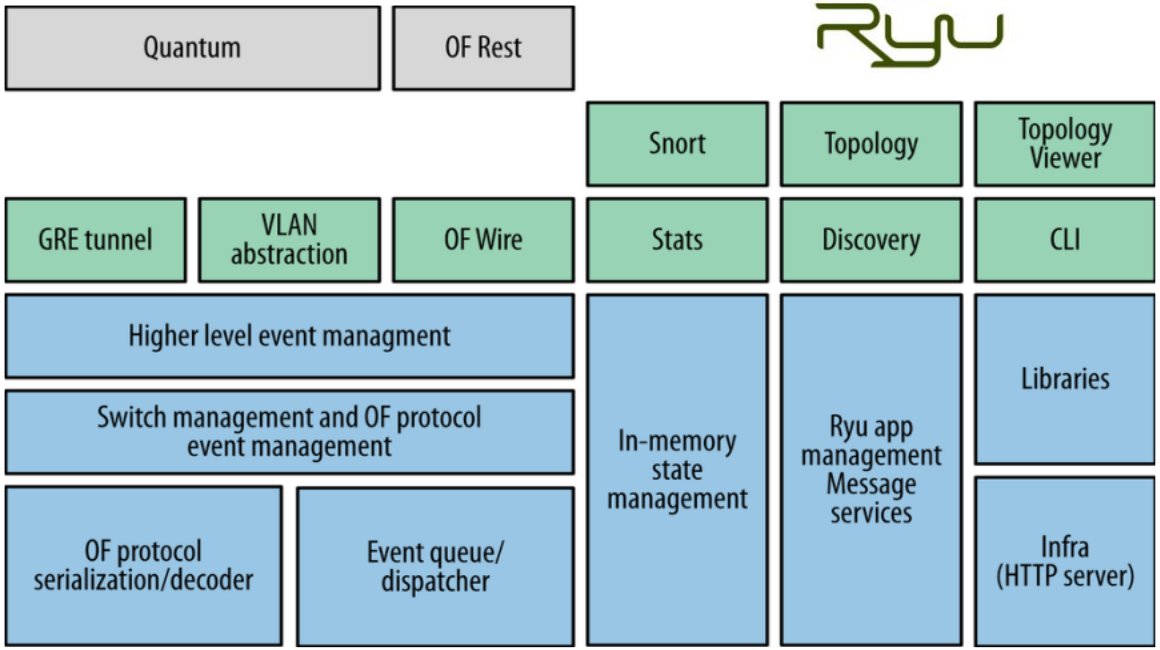
- An SDN controller combines a number of basic functions:
  1. Management of network state:
     - State management may rely on a database to keep information gathered from the controlled network elements
  2. A high-level data model:
     - Captures the relationships between managed resources, policies and services
  3. A north-bound RESTful API:
     - Exposes the controller services to applications
  4. Support for south-bound interfaces (e.g. OpenFlow) to control network elements
  5. Other supporting functions such as:
     - Network topology discovery
     - Terminal discovery
     - Shortest path computation

OpenFlow controllers: first wave (single instance)

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

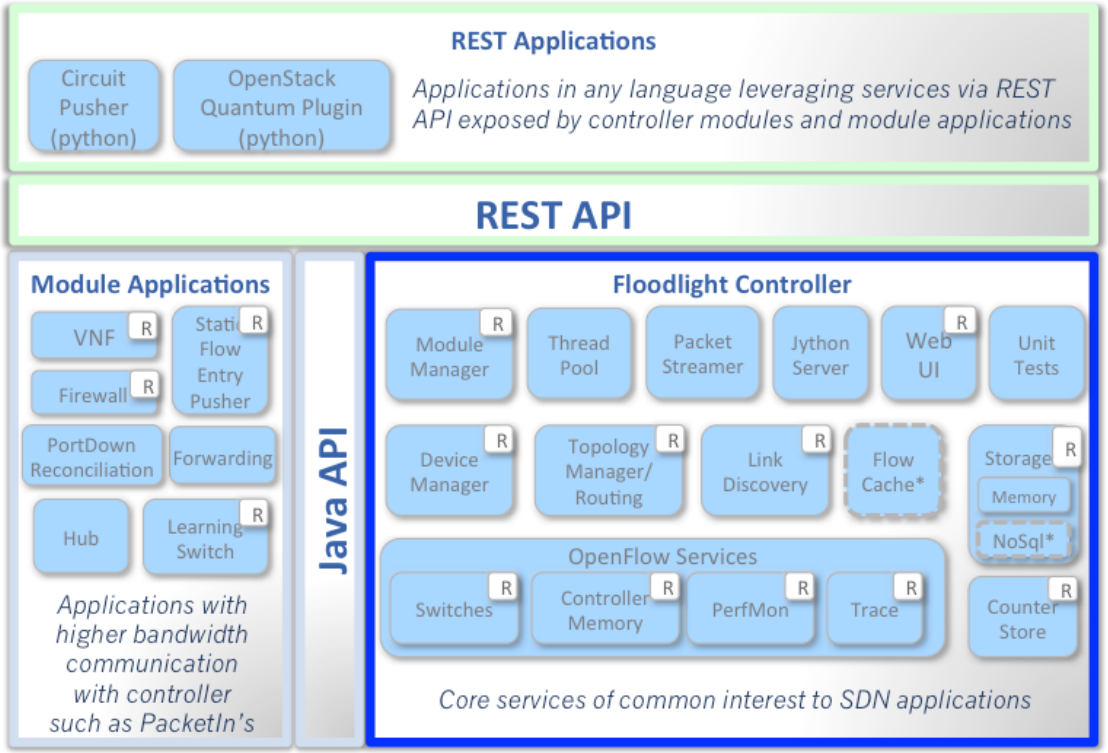
NOX, POX, Ryu controllers

- NOX developed by Nicira and donated to the research community, now open source
  - NOX provides a C++ API to OpenFlow and an asynchronous event-based model
- POX is a newer Python-based version of NOX
  - POX has reusable sample components for path selection, topology discovery, and so on
- Ryu is component-based, open source framework implemented in Python
  - Has an Openstack Quantum plug-in that supports both GRE based overlay and VLAN



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

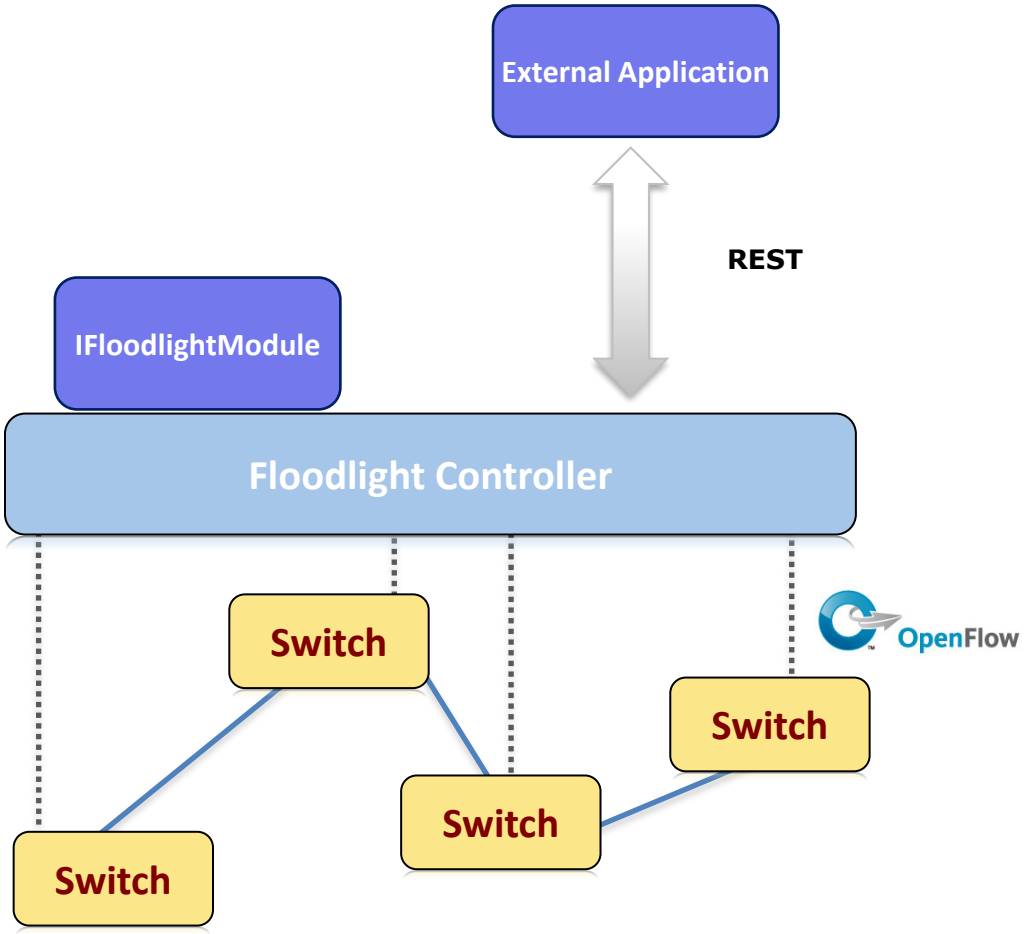
Northbound APIs

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)
```

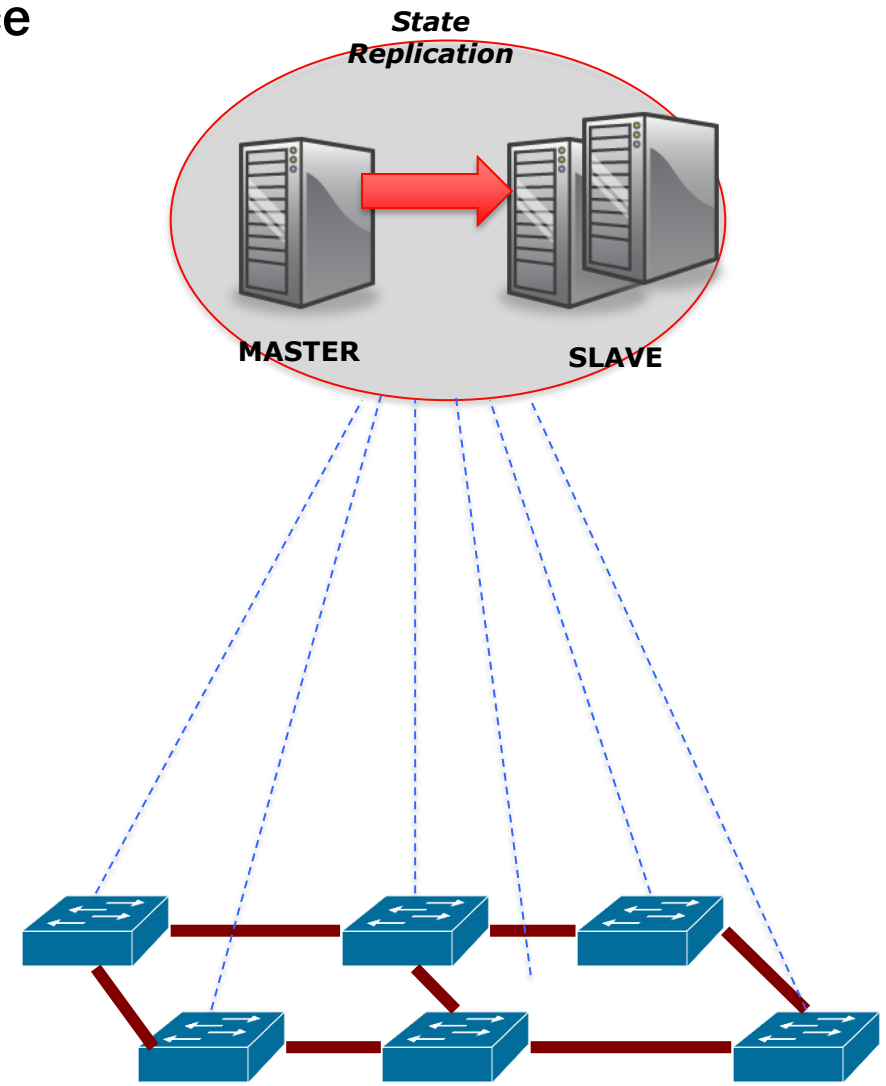
Programming Floodlight: custom module

- 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

        }

}
```

New generation OpenFlow controllers

- Early days controllers were single instance
- New generation SDN controllers support High-Availability configurations with multiple controllers
  - Opendaylight, ONOS
- Leader Election:
  - Two instances – alive & backup
  - More than two – consensus algorithm
- State synchronization
  - Configured state
  - Operational state
- Switches simultaneously connect to multiple controllers



OpenDaylight Controller

- Forked from Beacon by OpenDaylight consortium
- The **OpenDaylight Project** is a collaborative open source project hosted by Linux Foundation
- The software is written in Java
- Data center is the main use case
- Differentiating Features
  - Abstracted Southbound: “Service Abstraction Layer”
  - Wide scope
  - Many contributing sub-projects
    - Virtual Tenant Networks (VL2) (NEC)
    - Distributed Overlay Virtual Ethernet (IBM)

Release Name	Release Date
Hydrogen	February 2014
Helium	October 2014
Lithium	June 2015
Beryllium	February 2016
Boron	November 2016
Carbon	June 2017
Nitrogen	September 2017
Oxygen	March 2018
Fluorine	August 2018
Neon	March 2019
Sodium	September 2019

ONOS Controller

- Developed by Open Networking Labs (ON.Lab) with contributions & use-cases from partners
  - AT&T, NTT, …
- The **ONOS** (**Open Network Operating System**) project is an open source community hosted by the Linux Foundation
- The software is written in Java and relies on the Apache Karaf OSGi container platform
- Differentiating features:
  - Northbound interface: network graph
  - Scale-out operation
  - Telecom-oriented
- Made available Q4 2014

Release Name	Release Date
Avocet	December 5, 2014
Blackbird	February 28, 2015
Cardinal	May 31, 2015
Drake	September 18, 2015
Emu	December 18, 2015
Falcon	March 10, 2016
Goldeneye	June 24, 2016
Hummingbird	September 23, 2016
Ibis	December 9, 2016
Junco	February 28, 2017
Kingfisher	June 5, 2017
Loon	September 8, 2017
Magpie (LTS)	December 11, 2017
Nightingale	May 2, 2018
Owl	September 4, 2018
Peacock (LTS)	November 29, 2018
Quail	January 18, 2019
Raven (in-progress)	April 29, 2019

ONOS: Architecture Tiers

Northbound Abstraction:

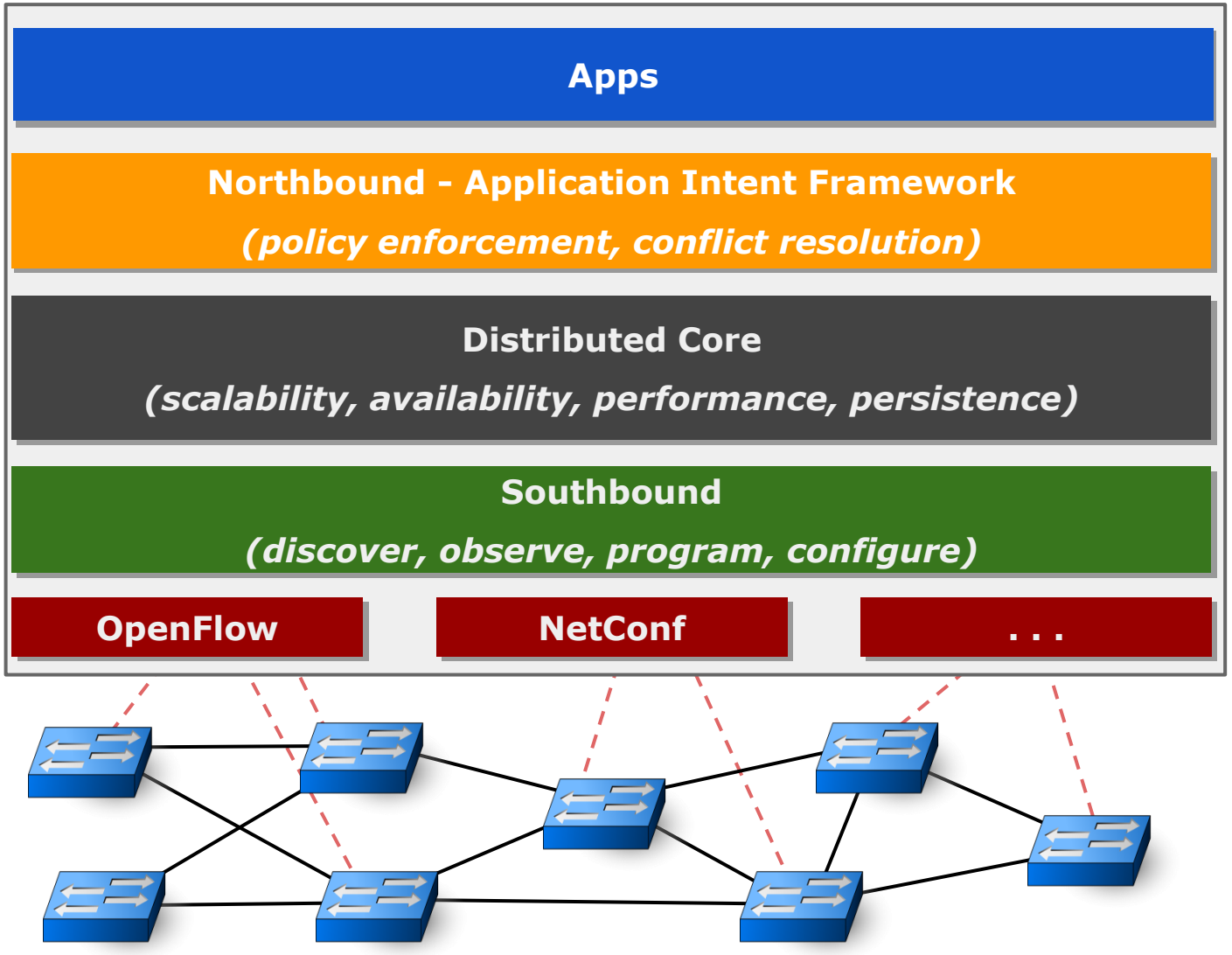
- network graph
- application intents

Core:

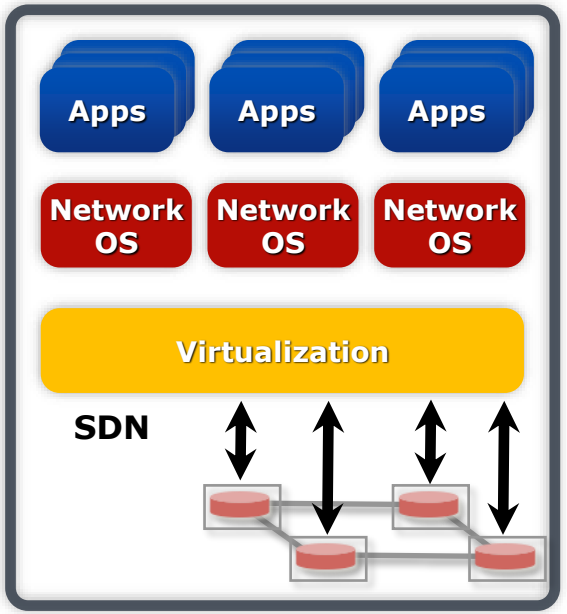
- distributed
- protocol independent

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