Principals of Mobile Cloud Computing

Part 2: Network Virtualization

Cloud-native Software-defined Mobile Networks
Department of Electrical Engineering
Sharif University of Technology
Azad Ravanshid

Outline:

- *Introduction to Network Virtualization*
- *Review on Legacy Networking Principals*
- *Introduction to Software-defined Networking*
  - *Control/Data Management Planes*
- *SDN Architectural Framework*
  - *SDN Architecture*
  - *South/North Bound APIs*
  - *SDN Controller*
- *SDN in Overlay Networks*
  - *VM Connectivity*
  - *Open vSwitch*

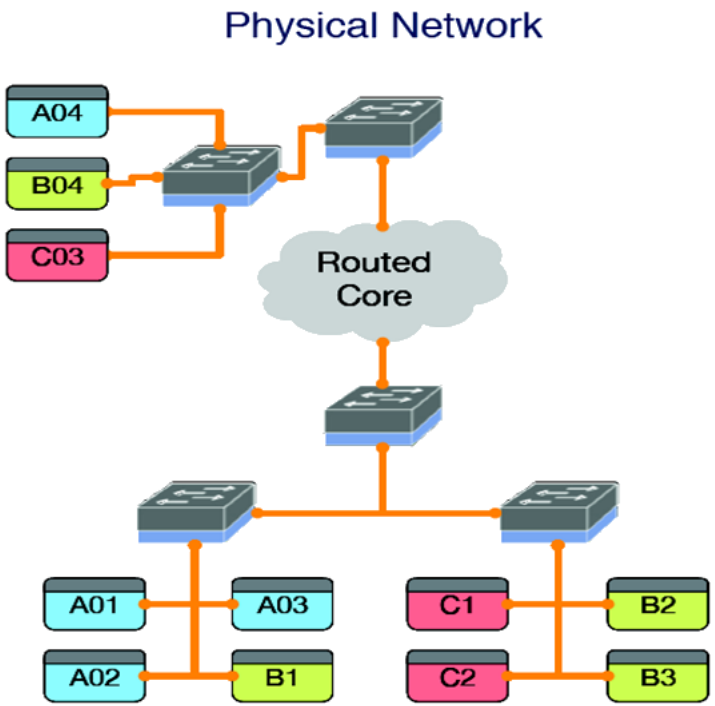
Section 1

INTRODUCTION TO NETWORK VIRTUALIZATION

Network Virtualization Concept

- Network virtualization is an approach in which a single physical network is *logically* split into multiple *logical networks*.
- These networks share the underlying infrastructure, but to the end user this sharing is not visible, and the protocols and technologies used make these networks appear to be completely independent and separate networks running on a dedicated infrastructure.
- Examples
  - **L3VPN** (IP Layer 3 Virtual Private Network)
  - **VLAN** (Virtual Local Area Network)
  - **VXLAN** (Virtual Extended LAN)
  - **NVGRE** (Network Virtualization using Generic Routing Encapsulation)

Multiple Applications/Tenants

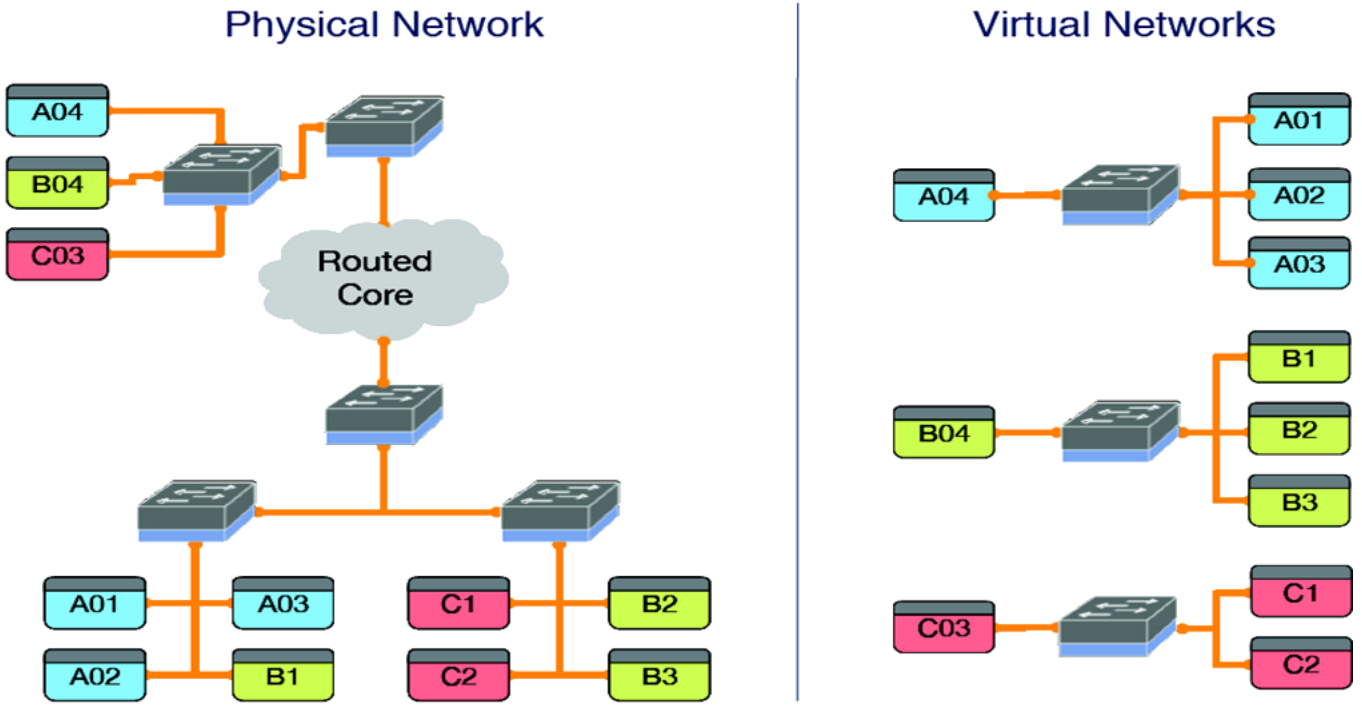


Application 1
Payment Services

Application 2
Production Intranet

Application 3
Test and Development

Multiple Applications/Tenants



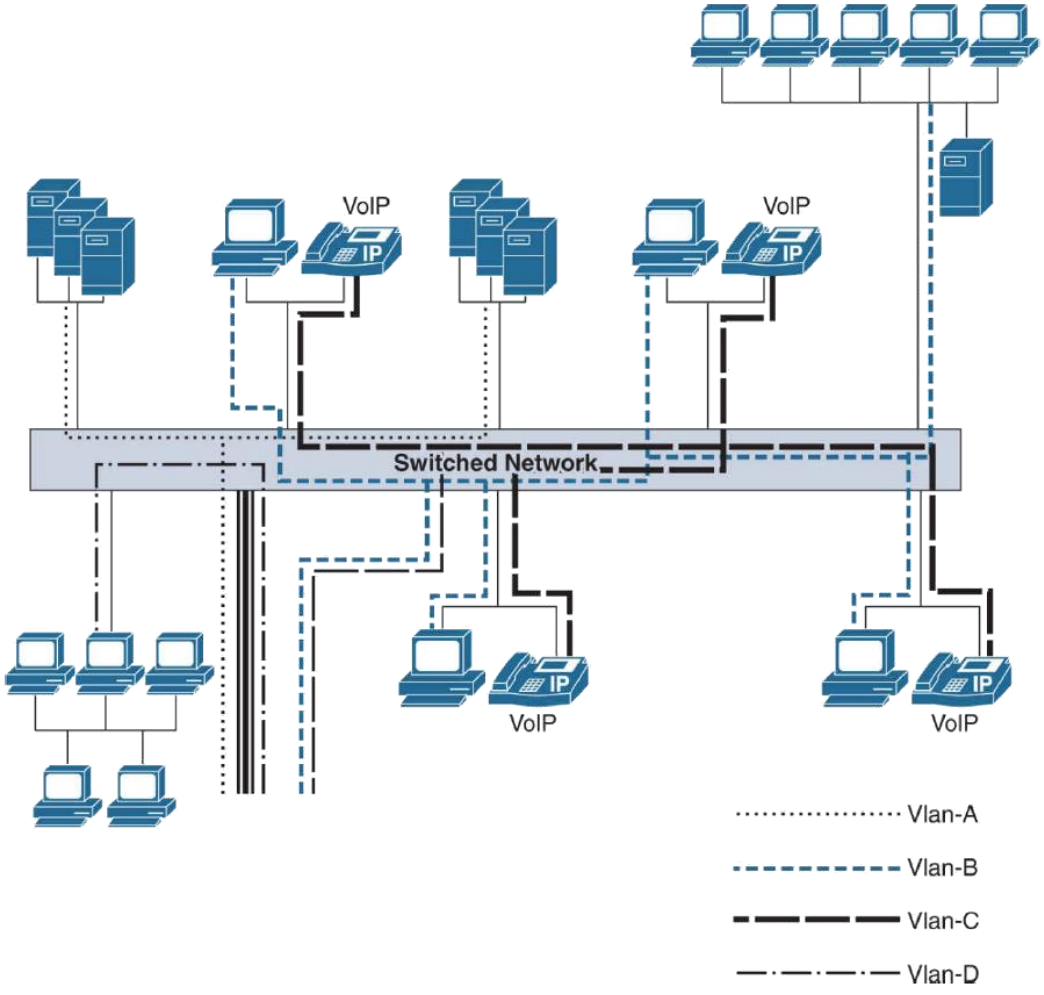
Network Virtualization Goals

- **Give each application/tenant its own virtual network with its own**
  - Topology
  - Bandwidth
  - Broadcast domain
  - …

Virtual Local Area Networks

- **Group related hosts**
  - Same company
  - Same role (e.g., faculty vs. students)
  - All WiFi users
- **Treat them as a single LAN**
  - Single IP address block
  - Single broadcast domain
  - No access control
- **Independent of their location**

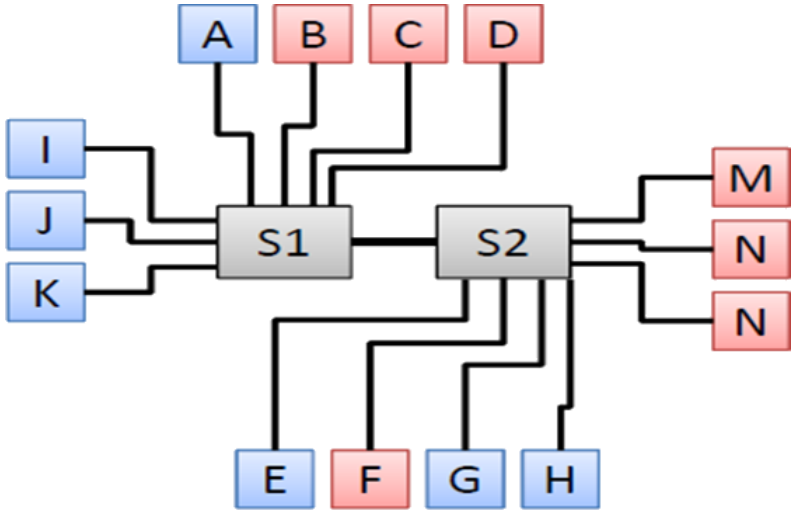
Rewire the network in software!


VoIP
IP
VoIP
IP
Switched Network
IP
VoIP
IP
VoIP
Vlan-A
Vlan-B
Vlan-C
Vlan-D

Making VLANs Work

Creating multiple virtual LANs from a single physical network

- Nodes in different LANs can't communicate, needs a router
- Broadcast isolation (e.g. ARP)

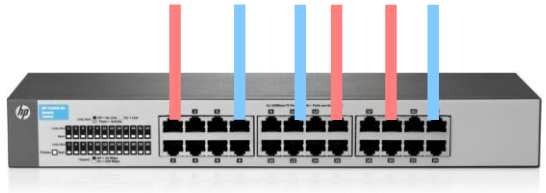


Making VLANs Work

L2 Switching Methods

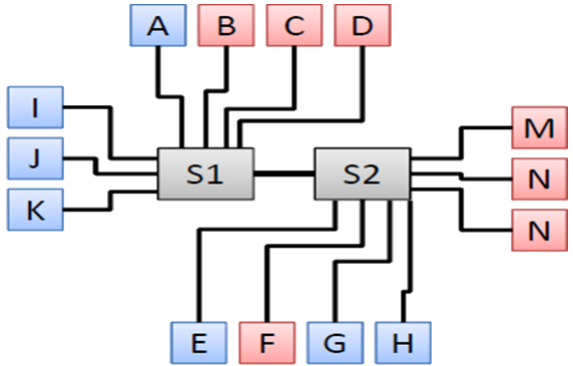
- **Static: Port-based**

Switch port statically defines the VLAN a host is part of



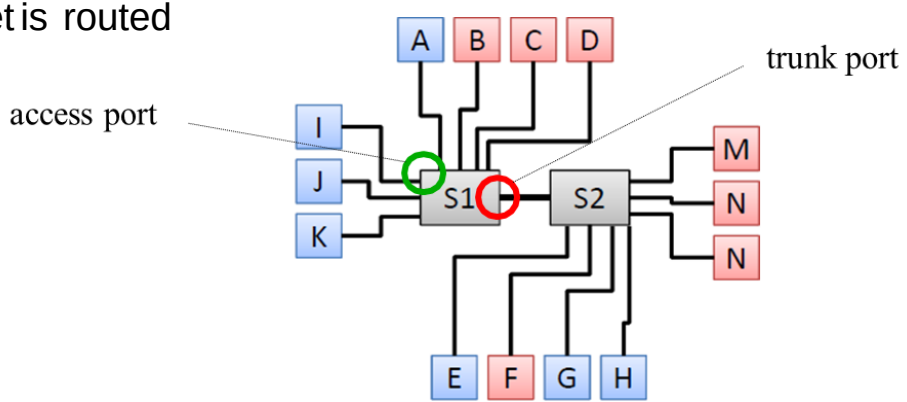
- **Dynamic: MAC-based**

MAC address defines which VLAN a host is part of Typically configured by network administrator.

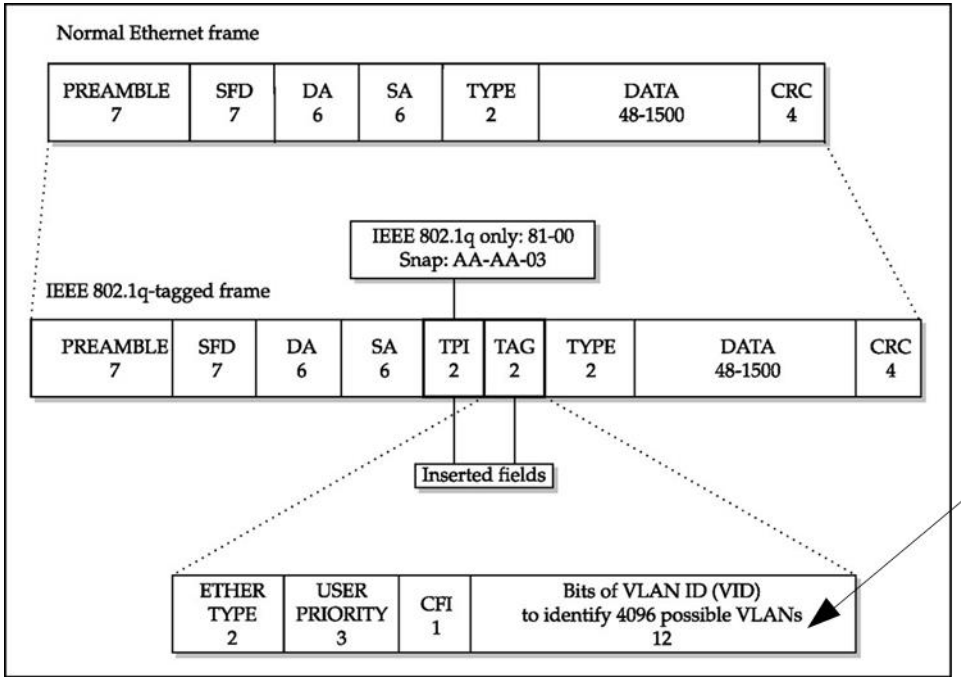


VLAN Tagging

- Access port: port connected to end host, assigned to a single VLAN
- Trunk port: port that sends and receives tagged frames on multiple VLANs
- VLAN tagging at trunk link
  - Tag added to layer-2 frame at ingress switch
  - Tag stripped at egress switch
  - Tag defines on which VLAN the packet is routed

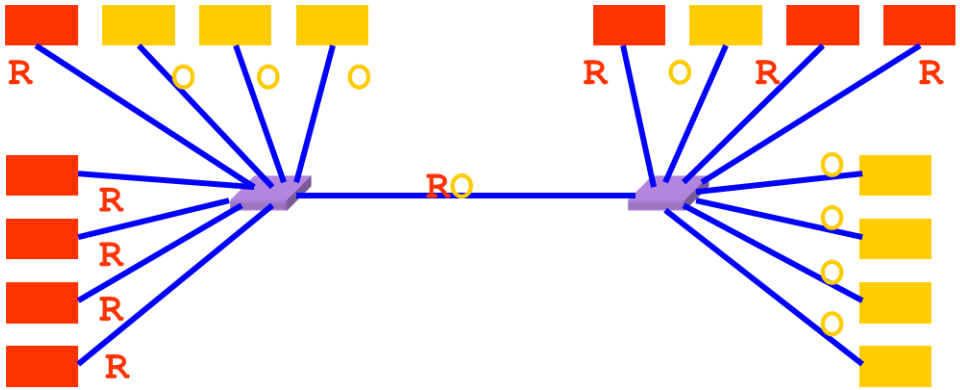


802.1Q tagging



Making VLANs Work

- **Changing the Ethernet header**
  - Adding a field for a VLAN tag
  - Implemented on the bridges/switches
  - … but can interoperate with old Ethernet cards
- **Bridges/switches trunk links**
  - Say which VLANs are accessible via which interfaces
- **Approaches to mapping access links to VLANs**
  - Each interface has a VLAN "color"
  - Each MAC address has a VLAN "color"



Red VLAN and Orange VLAN Switches forward traffic as needed

Problems with VLANs

- Requires separate configuration of every network node
- Static configuration
  - Requires administrator
- Number of VLANs limited (4096)

Challenges Faced by Traditional Networks

Today's Network are defined by the "Box"

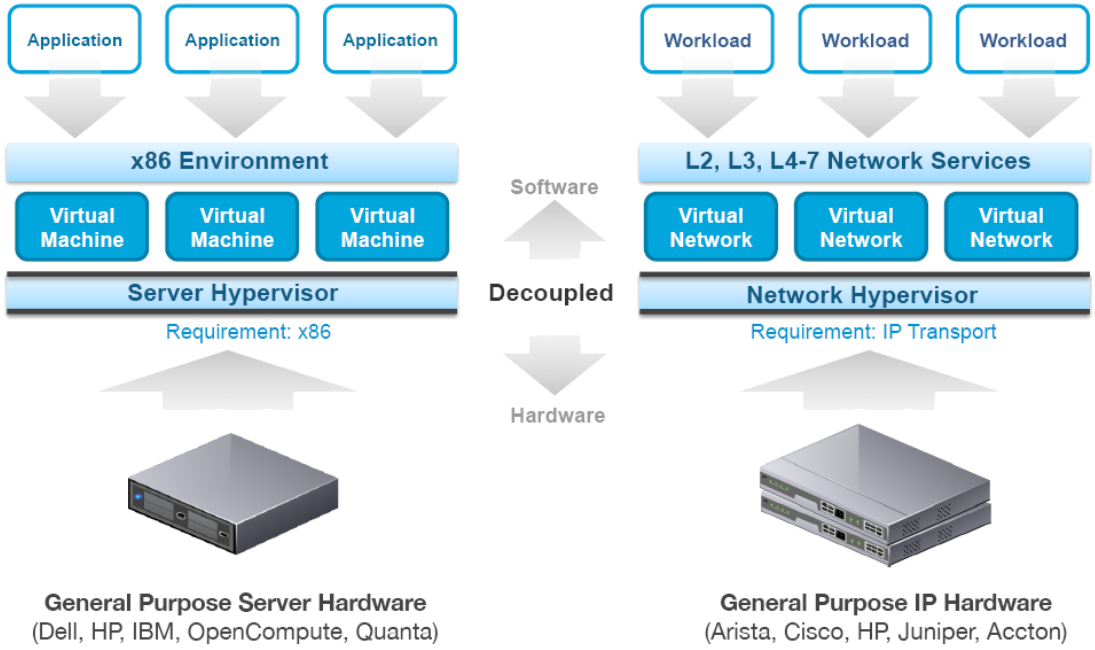
- **Hardware, OS, and App built into a Box**
  Routers, switches, firewalls are basically a hardware that are tuned for a specific tasks.

- **Cannot Mix and Match**
  You can not use the hardware of a switch for a router for example
  Furthermore, not the same task between different vendors

- **Lack of Competition at each layer**
  Competeion at the total solution not different parts of it

- **Barrier to entry**
  Because of the evolved complexity, not any one can do this work, threat of monopoly

Network Virtualization



Why Network Hypervisor?

- The main network requirements can be captured along two dimensions:
  - network topology
  - address space
- Different workloads require different network topologies and services, such as flat L2 or L3 services, or even more complex L4- L7 services for advanced functionality.
- The network infrastructure should be able to support arbitrary network topologies and addressing schemes
- The long standing virtualization primitives such as VLANs (virtualized L2 domain), NAT (Virtualized IP address space), and MPLS (virtualized path) are anchored on a box-by-box basis configuration, i.e. no single abstraction for network (re-) configuration in a global manner

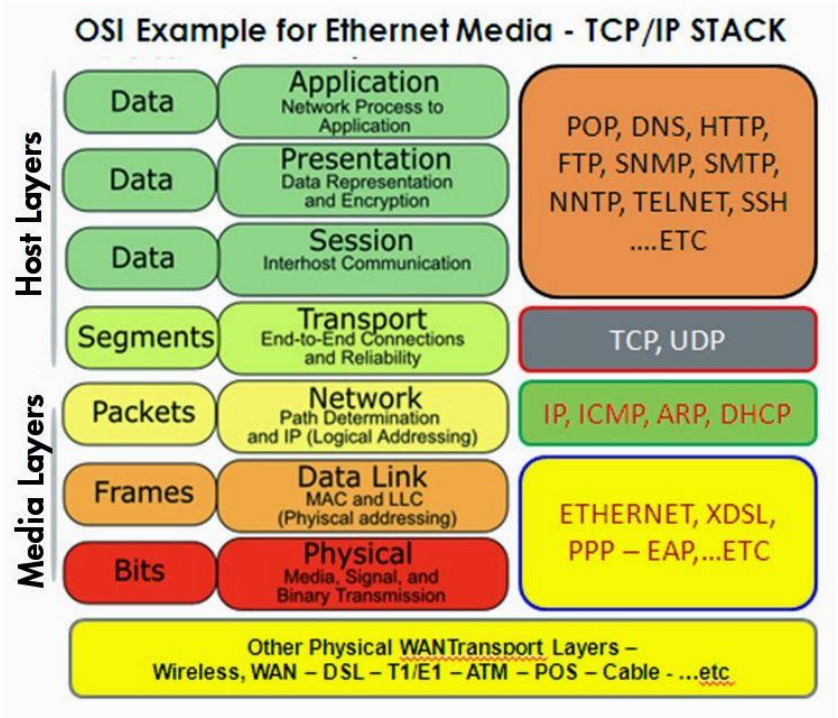
Section 2

REVIEW ON LEGACY NETWORKING PRINCIPALS

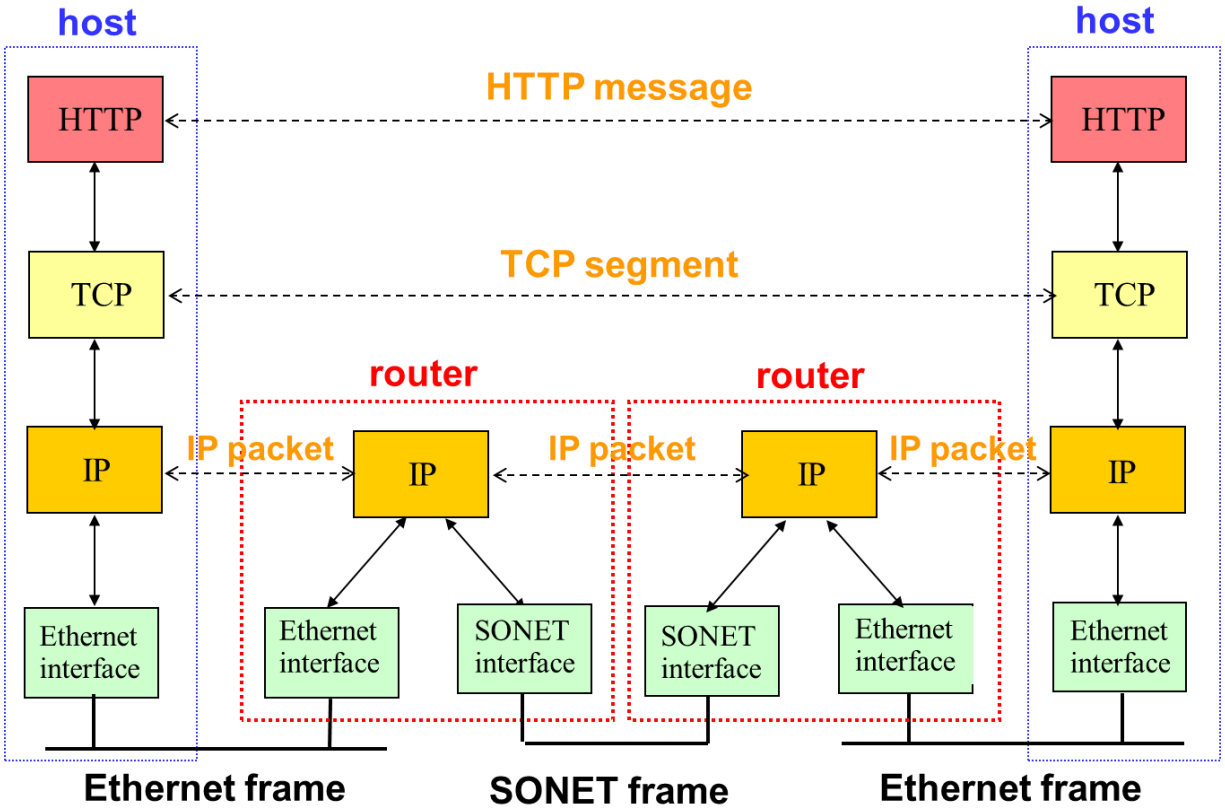
Why A Review?

- SDN interacts with “legacy” networks
  - Unmodified end-host computers
  - Hybrid deployments of SDN
  - Connecting to non-SDN domains
- SDN is a reaction to legacy networks
  - Challenges of managing and changing them
- General lessons on abstractions
  - Practice talking about abstractions
  - Some abstractions should be retained

Layering: Internet Protocol Stack

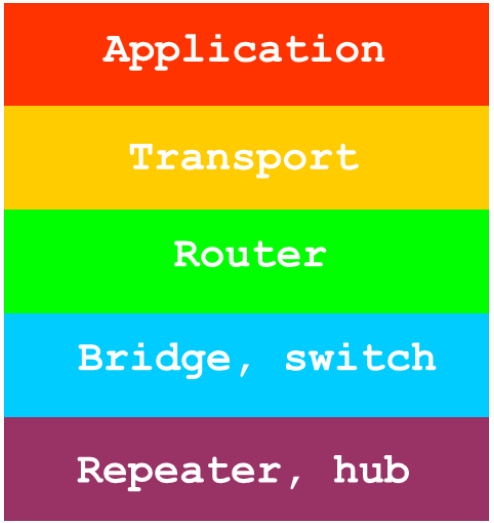


Layering: End-to-End



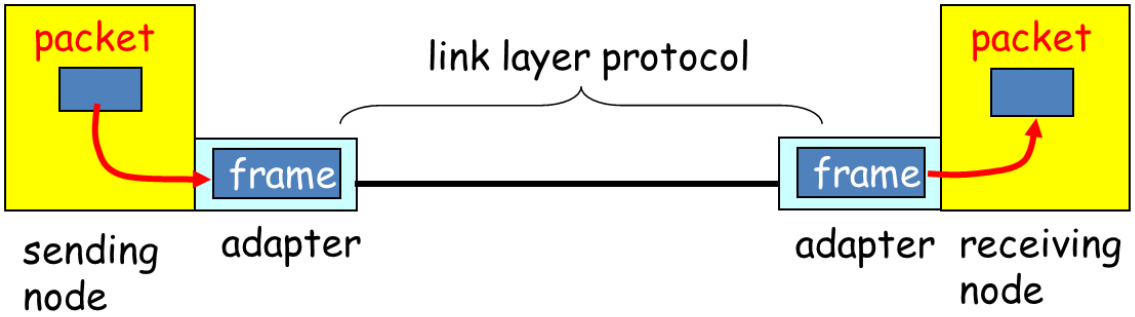
Layering: Packet Encapsulation

- Different devices switch different things
  - Network layer: packets (routers)
  - Link layer: frames (bridges and switches)
  - Physical layer: electrical signals (repeaters and hubs)



Link Layer: Adaptors Communicating

- Sending side
  - Encapsulates packet in a frame
  - Adds error checking bits, flow control, etc.
- Receiving side
  - Looks for errors, flow control, etc.
  - Extracts datagram and passes to receiving node



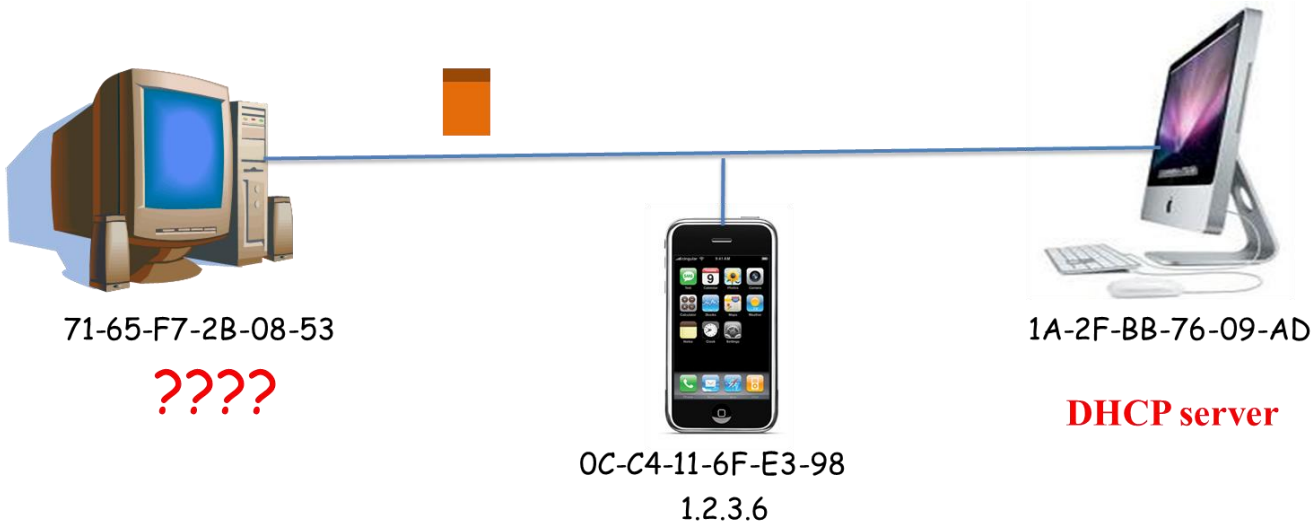
Link Layer: Medium Access Control Address

- MAC address (e.g., 00-15-C5-49-04-A9)
  - Numerical address used within a link
  - Unique, hard-coded in the adapter when it is built
  - Flat name space of 48 bits
- Hierarchical allocation
  - Blocks: assigned to vendors (e.g., Dell) by the IEEE
  - Adapters: assigned by the vendor from its block
- Broadcast address (i.e., FF-FF-FF-FF-FF-FF)
  - Send the frame to *all* adapters

Link Layer: Why Not Just Use IP Addresses?

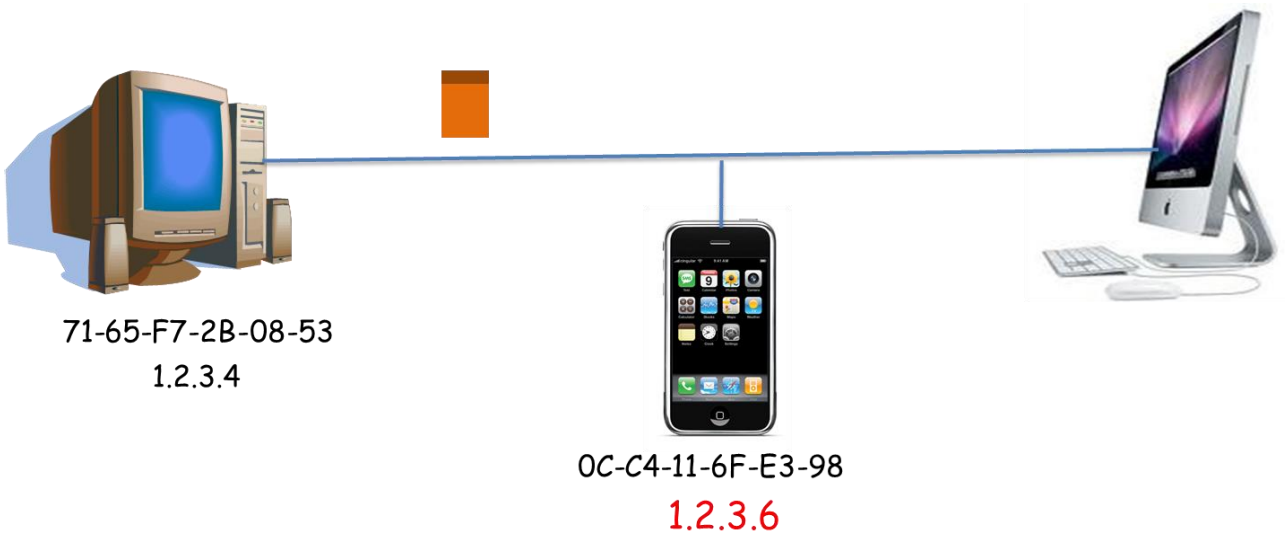
- Links can support *any* network protocol
  - Not just for IP (e.g., IPX, Appletalk, X.25, …)
  - Different addresses on different kinds of links
- An adapter may move to a new location
  - So, cannot simply assign a static IP address
  - Instead, must reconfigure the adapter's IP address
- Must identify the adapter during bootstrap
  - Need to talk to the adapter to assign it an IP address

Link Layer: Who am I?



- Dynamic Host Configuration Protocol (DHCP)
  - Broadcast "I need an IP address, please!"
  - Response "You can have IP address 1.2.3.4."

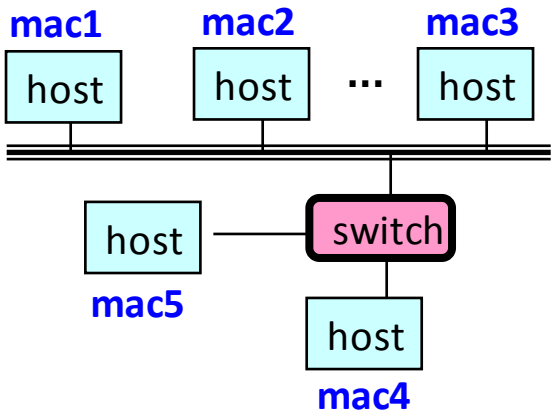
Link Layer: Who are You?



- Address Resolution Protocol (ARP)
  - Broadcast "who has IP address 1.2.3.6?"
  - Response "0C-C4-11-6F-E3-98 has 1.2.3.6!"

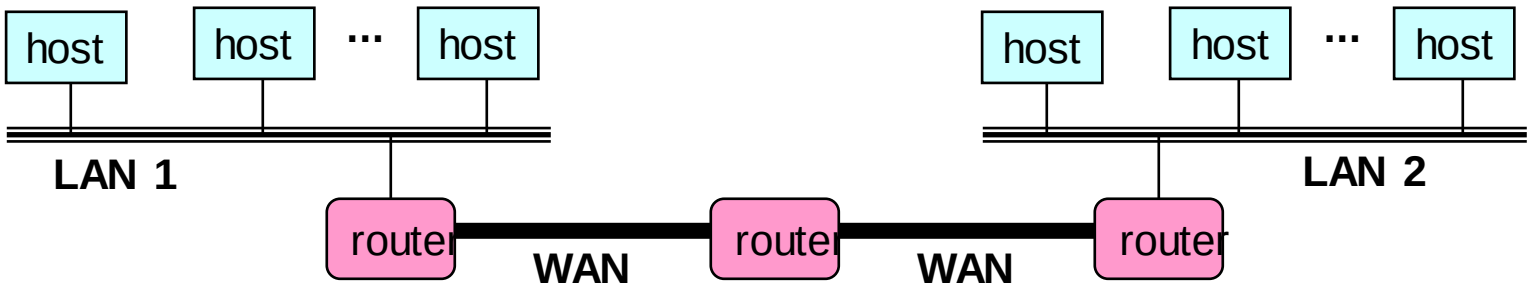
Switch: Match on Destination MAC

- **MAC addresses are location independent**
  - Assigned by the vendor of the interface card
  - Cannot be aggregated across hosts in LAN



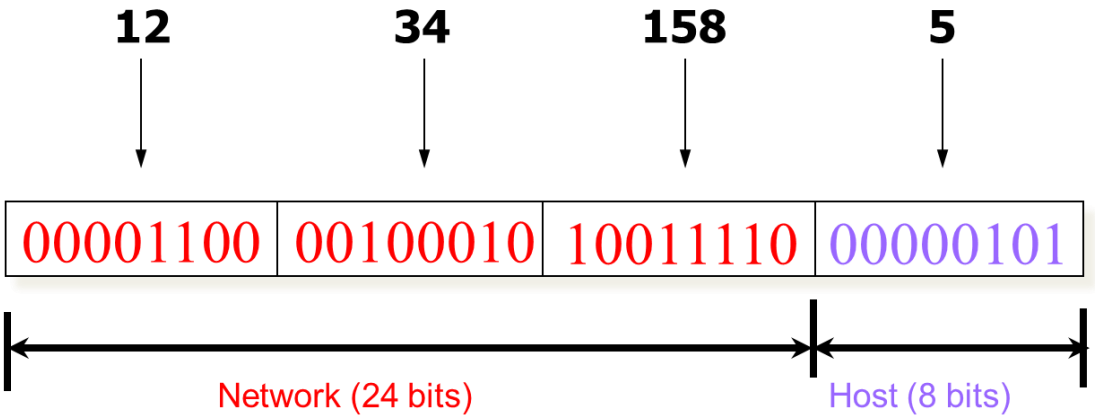
mac1	↑
mac2	↑
mac3	↑
mac4	←
mac5	↓

Network Layer: Connect Local Networks

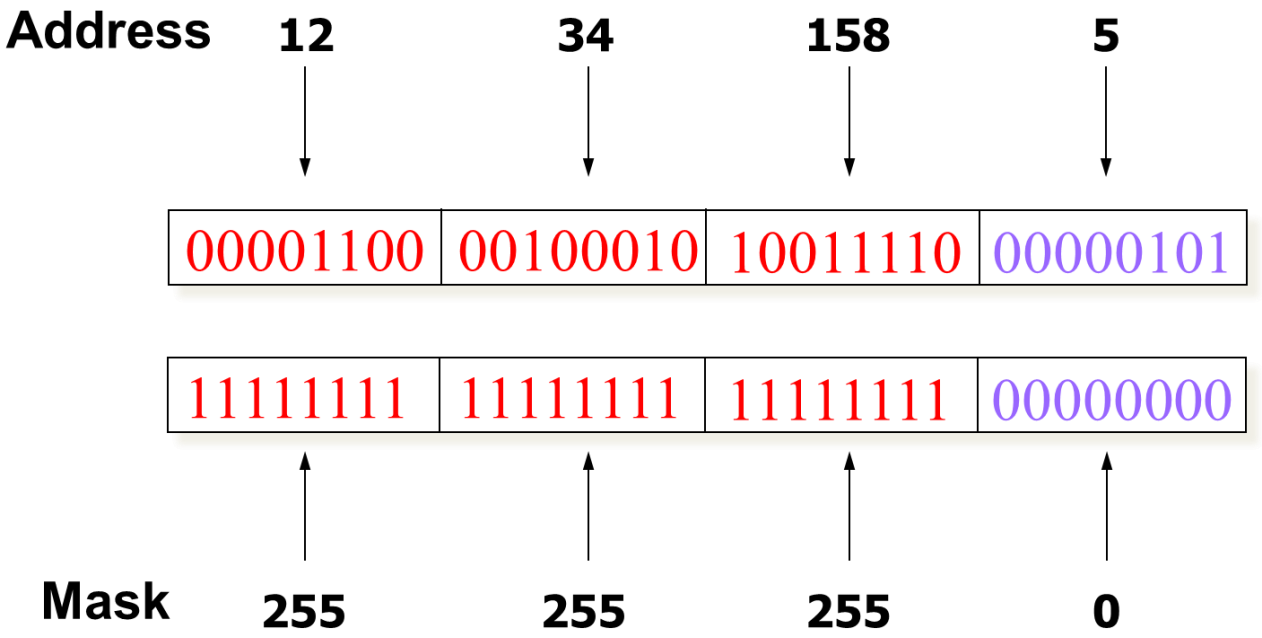


Network Layer: Hierarchical Addressing

- Network and host portions (left and right)
- 12.34.158.0/24 is a 24-bit **prefix** with 2^8 addresses

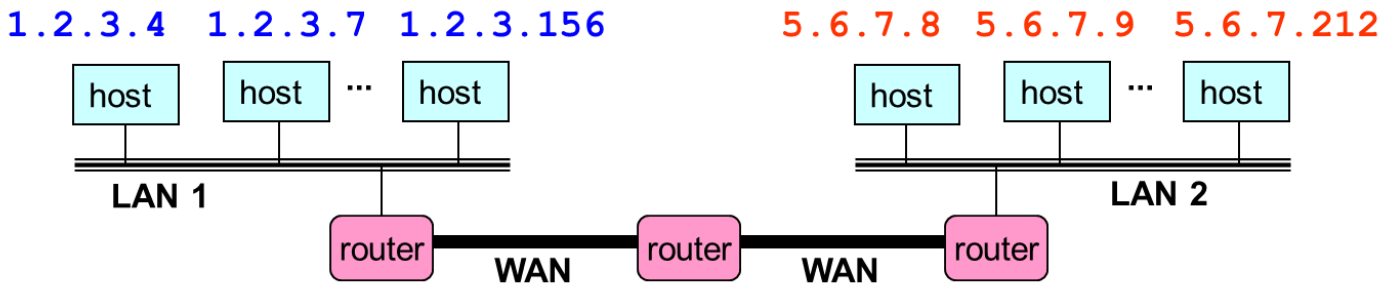


Network Layer: Address and Mask



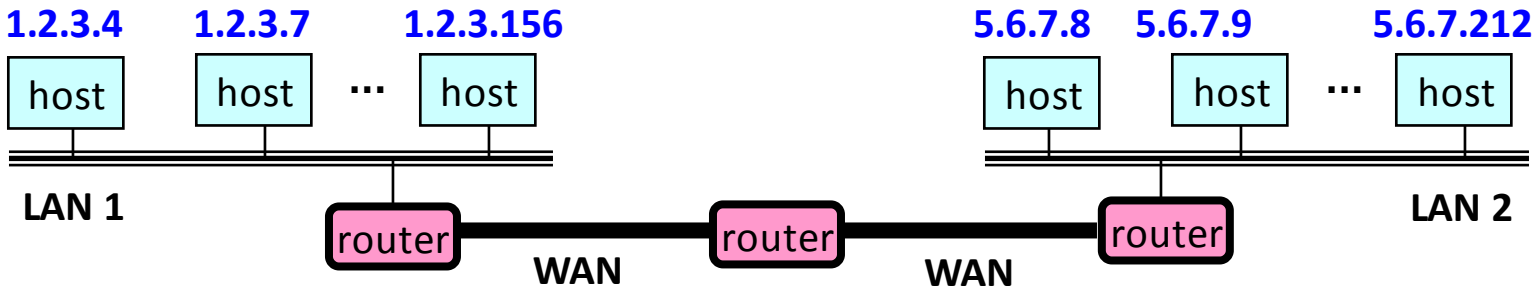
Network Layer: Scalability

- Number related hosts from a common subnet
  - 1.2.3.0/24 on the left LAN
  - 5.6.7.0/24 on the right LAN



Router: Match on IP Prefix

- **IP addresses grouped into common subnets**
  - Allocated by ICANN, regional registries, ISPs, and within individual organizations
  - Variable-length prefix identified by a mask length



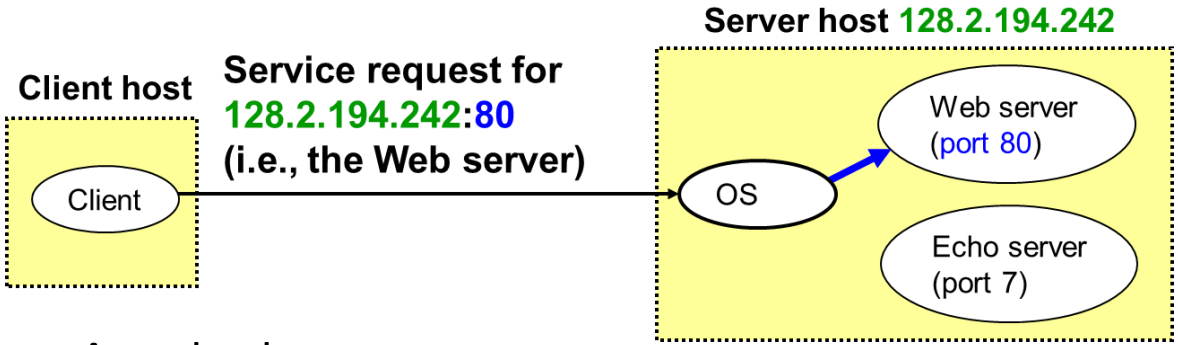
Routers identify the *longest matching* prefix.

1.2.3.0/24	←
5.6.7.0/24	→

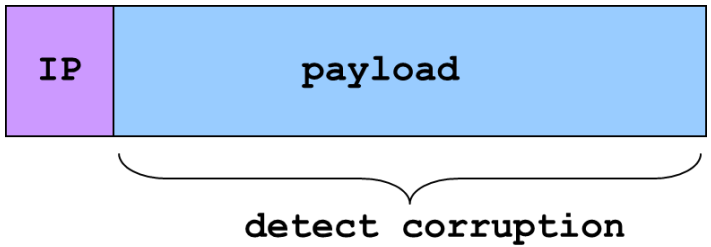
Forwarding Table

Transport Layer: Two Main Ideas

- **De-multiplexing:** port numbers

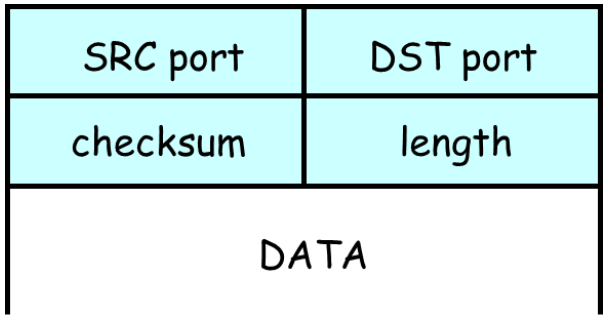


- **Error detection:** checksums



Transport Layer: User Datagram Protocol (UDP)

- Datagram messaging service
  - De-multiplexing: port numbers
  - Detecting corruption: checksum
- Lightweight communication between processes
  - Send and receive messages
  - Avoid overhead of ordered, reliable delivery

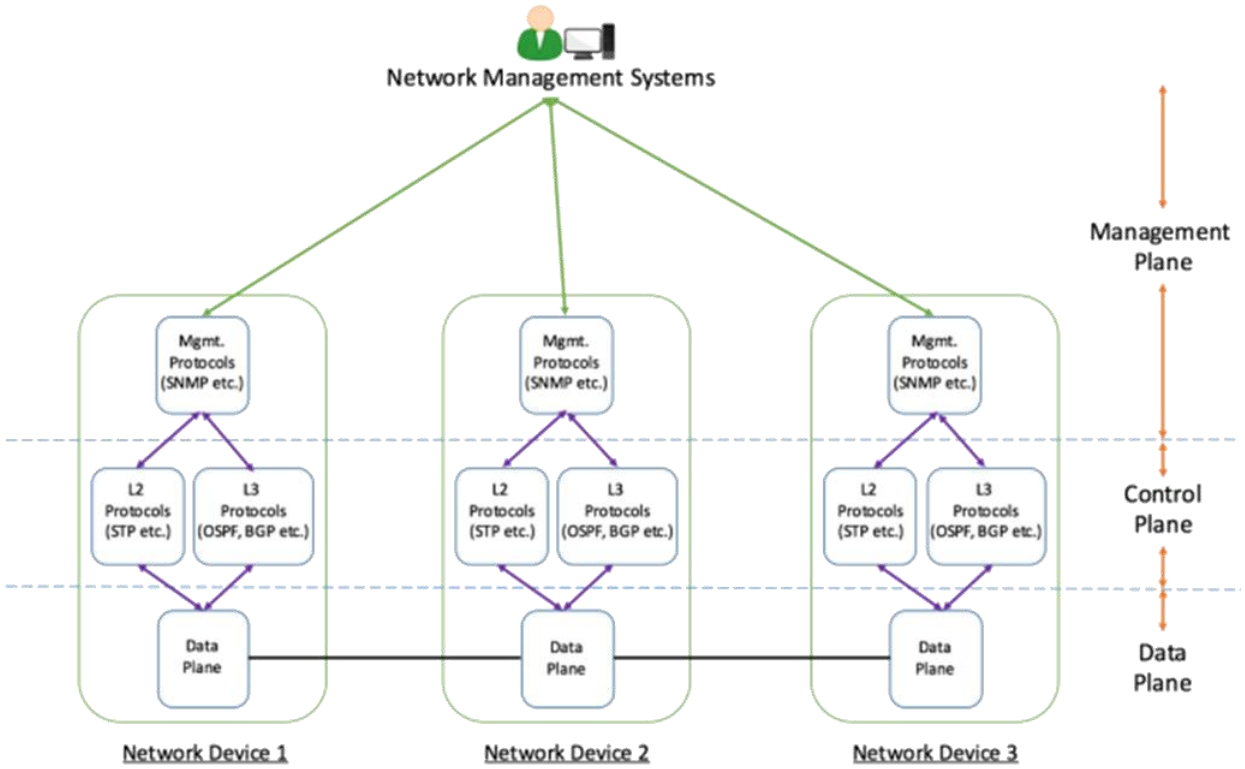


Transport Layer: Transmission Control Protocol (TCP)

- Stream-of-bytes service
  - Sends and receives a stream of bytes
- Reliable, in-order delivery
  - Corruption: checksums
  - Detect loss/reordering: sequence numbers
  - Reliable delivery: acknowledgments and retransmissions
- Connection oriented
  - Explicit set-up and tear-down of TCP connection
- Flow control
  - Prevent overflow of the receiver's buffer space
- Congestion control
  - Adapt to network congestion for the greater good

The Networking “Planes”

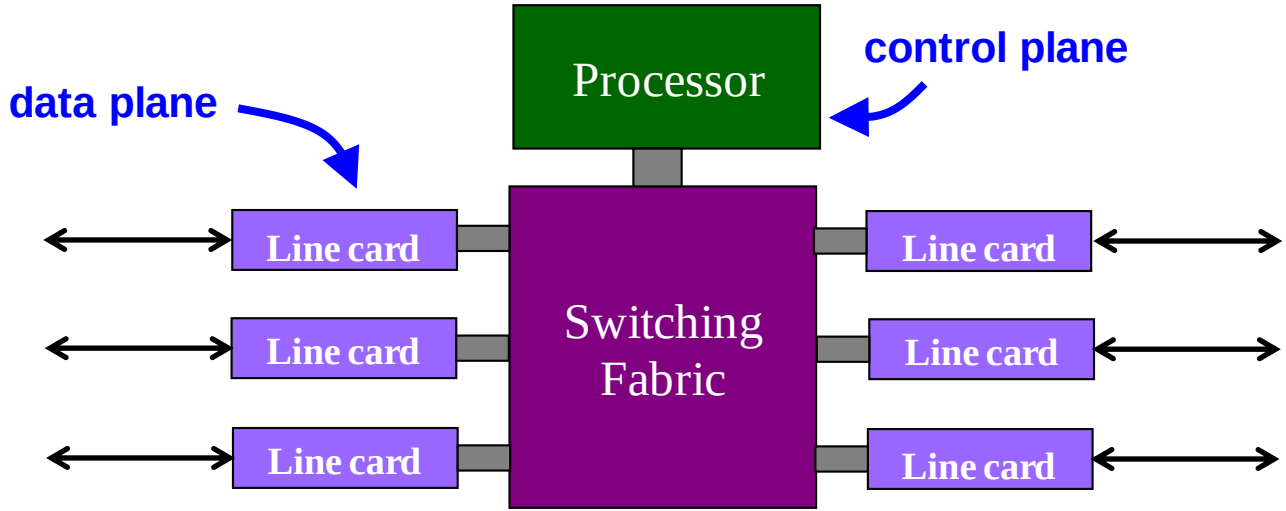
- **Data plane**: **processing and delivery of packets with local forwarding state**
  - Forwarding state + Packet header → forwarding decision completion
  - Filtering, Buffering, Scheduling
- **Control plane: computing the forwarding state in routers**
  - Determines how and where packets are forwarded
  - Routing, Switching, Failure detection/recovery, …
- **Management plane: configuring and tuning the network**
  - Traffic engineering, ACL Configuration, Device provisioning, …


Network Management Systems
Management Plane
Control Plane
Data Plane
Mgmt. Protocols (SNMP etc.)
L2 Protocols (STP etc.)
L3 Protocols (OSPF, BGP etc.)
Data Plane
Network Device 1
Network Device 2
Network Device 3

Timescales

	Data	Control	Management
Time-scale	Packet (nsec)	Event (10 msec to sec)	Human (min to hours)
Tasks	Forwarding, buffering, filtering, scheduling	Routing, circuit set-up	Analysis, configuration
Location	Line-card hardware	Router software	Humans or scripts

Data and Control Planes



Data Plane

- **Streaming algorithms on packets**
  - Matching on some bits
  - Perform some actions
- **Wide range of functionality**
  - Forwarding
  - Access control
  - Mapping header fields
  - Traffic monitoring
  - Buffering and marking
  - Shaping and scheduling
  - Deep packet inspection

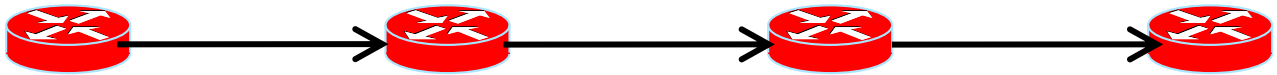
Control Plane

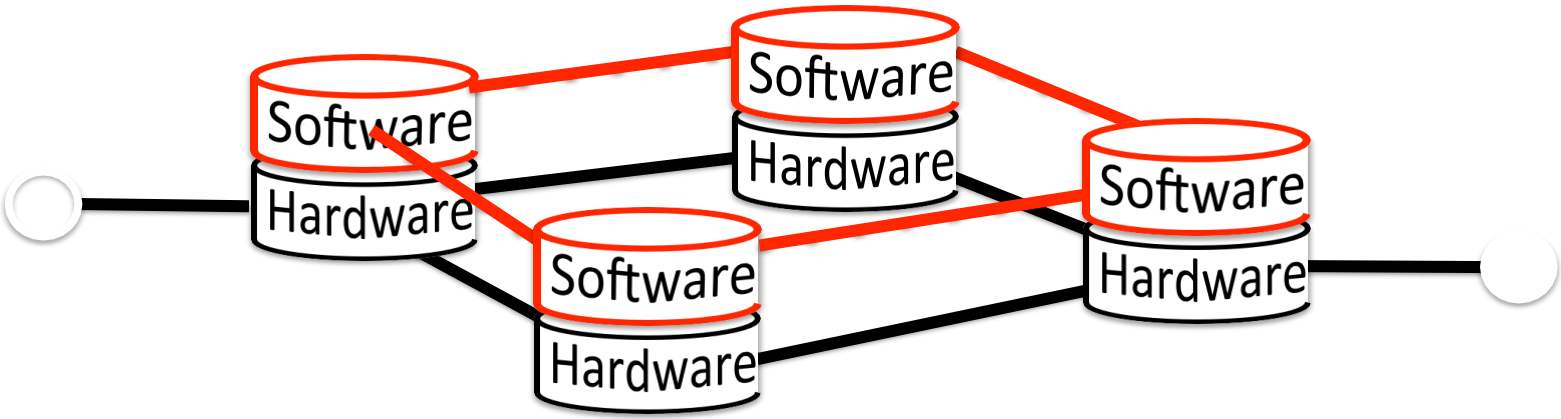
- **Compute paths the packets will follow**
  - Populate forwarding tables
  - Traditionally, a distributed protocol

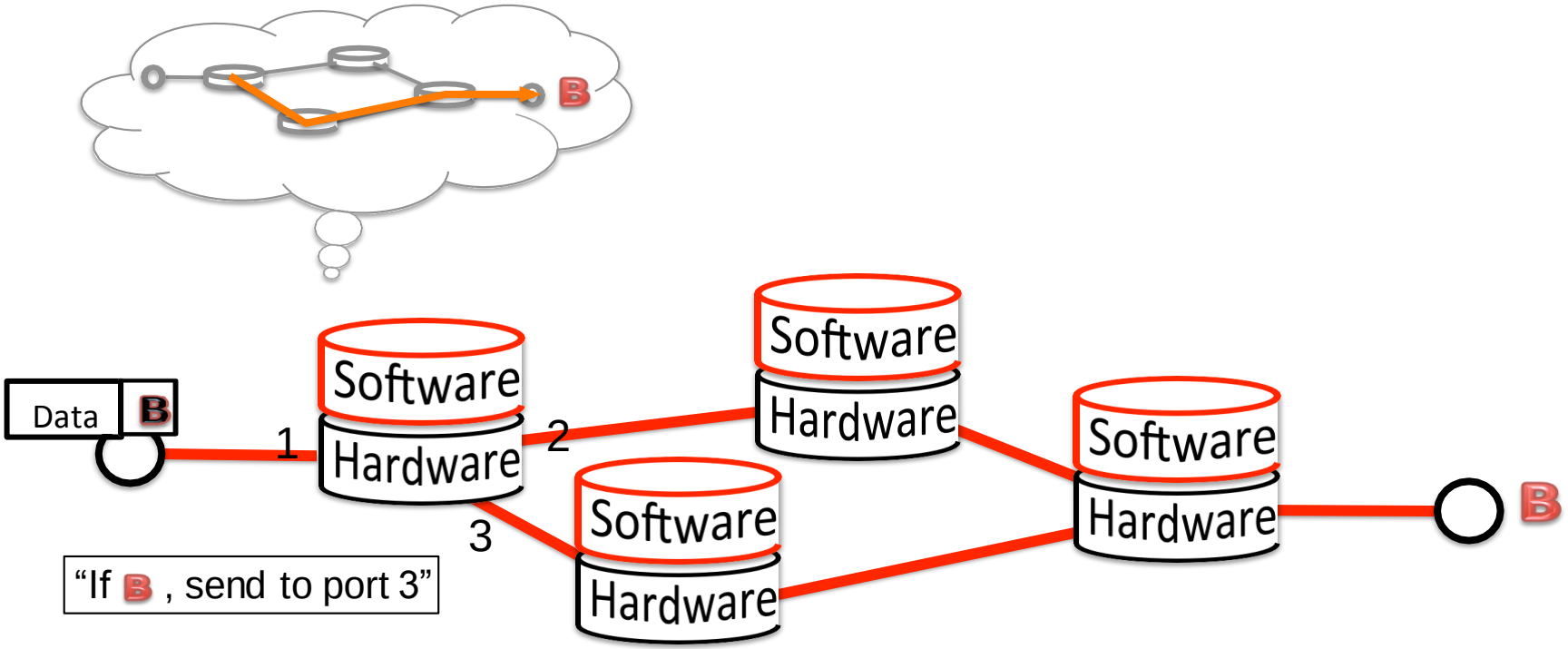
- **Example: Link-state routing (OSPF, IS-IS)**
  - Flood the entire topology to all nodes
  - Each node computes shortest paths
  - Dijkstra's algorithm

Forwarding vs. Routing

- **Forwarding: data plane**
  - Directing a data packet to an outgoing link
  - Individual router *using* a forwarding table
- **Routing: control plane**
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Individual router creating a forwarding table




Software
Hardware
Software
Hardware
Software
Hardware
Software
Hardware



- **Figure out which routers and links are present**
  Run Dijkstra's algorithm to find shortest paths

Distributed Control Plane

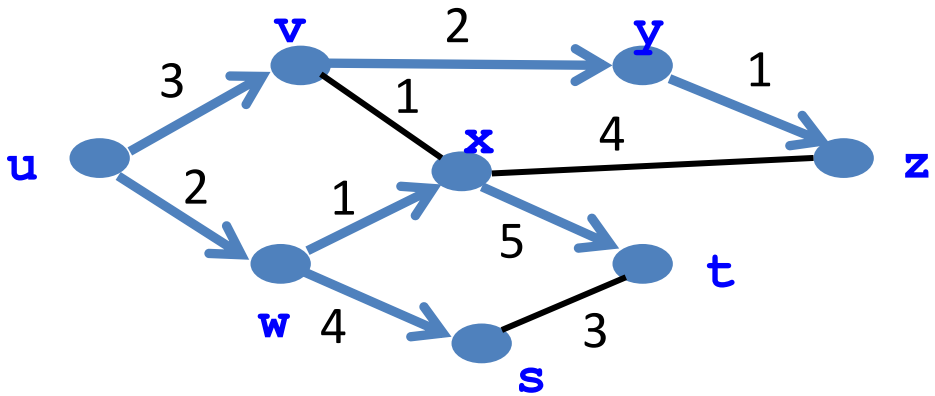
- **Link-state Routing Protocol**

  Link state routing is a technique in which each router shares the knowledge of its neighbourhood with every other router in the internetwork.

  – Flood the entire topology to all nodes

  – Each node computes shortest paths based on **iterative Dijkstra's algorithm**

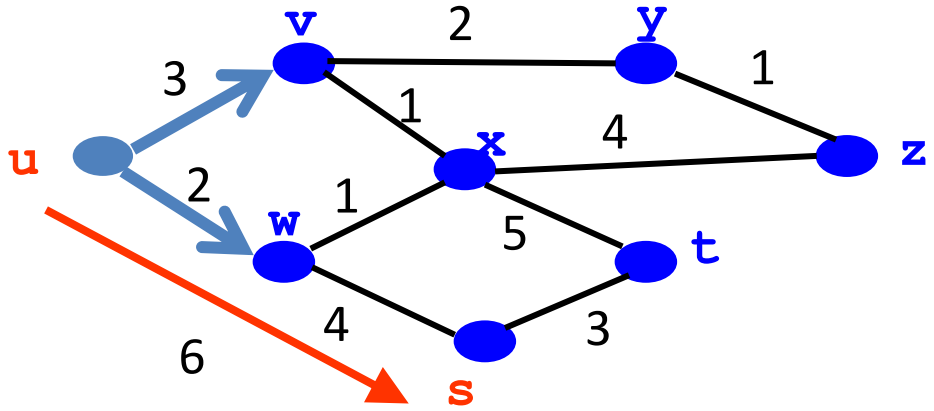
  – Open Shortest-Path First (OSPF), ISIS



Vert.	Link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

Example: Shortest-Path Routing

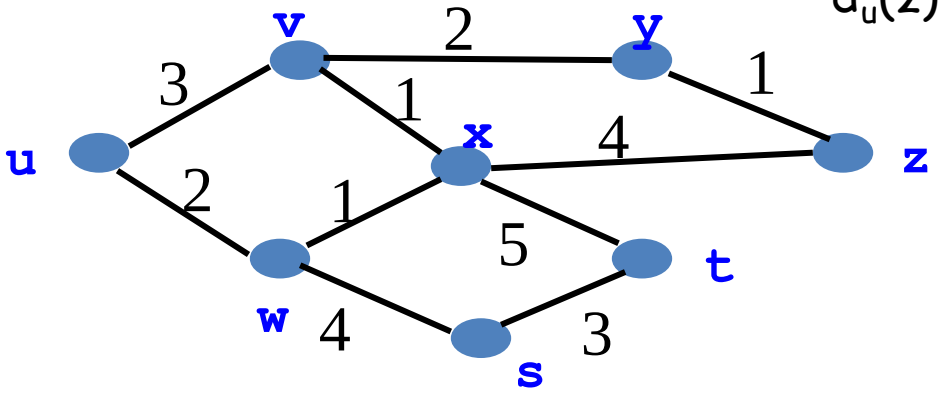
- **Compute: *path costs* to all nodes**
  - From a source **U** to all other nodes
  - Cost of the path (e.g. distance, delay, bandwidth, average que, operational costs and etc.) through each link
  - Next hop along least-cost path to **S**



	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

Distributed Control Plane

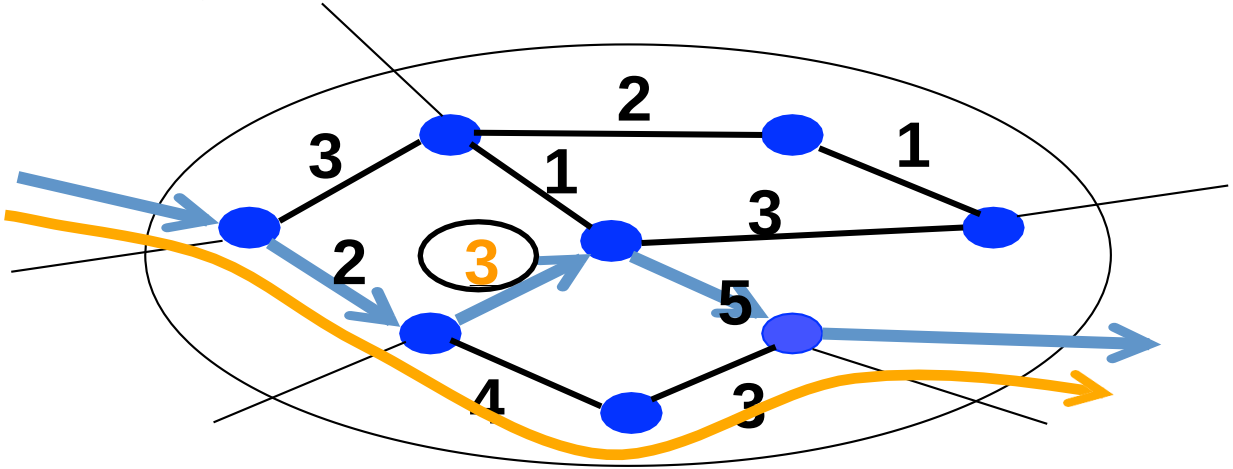
- **Distance-vector Routing Protocol**
  - Measuring the distance in terms of the numbers of routers (hops) a packet has to pass to reach its destination networkbased on each neighbors' path cost
  - Exchanging the information only with the neighbors; no access to network topology
  - Bellman-Ford algorithm



$$d_u(z) = \min\{c(u,v) + d_v(z),\ c(u,w) + d_w(z)\}$$

Management Plane

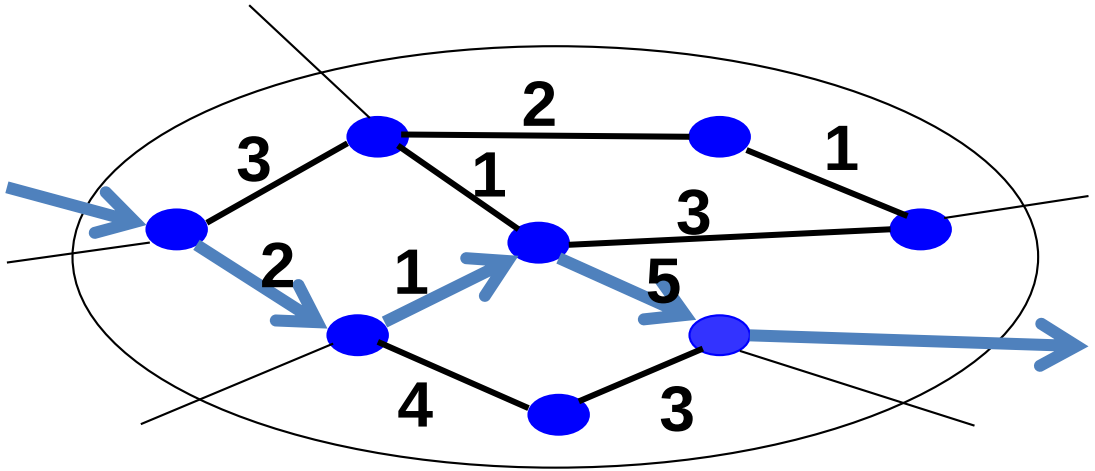
- **Traffic Engineering: setting the weights**
  - Inversely proportional to link capacity?
  - Proportional to propagation delay?
  - Network-wide optimization based on traffic?



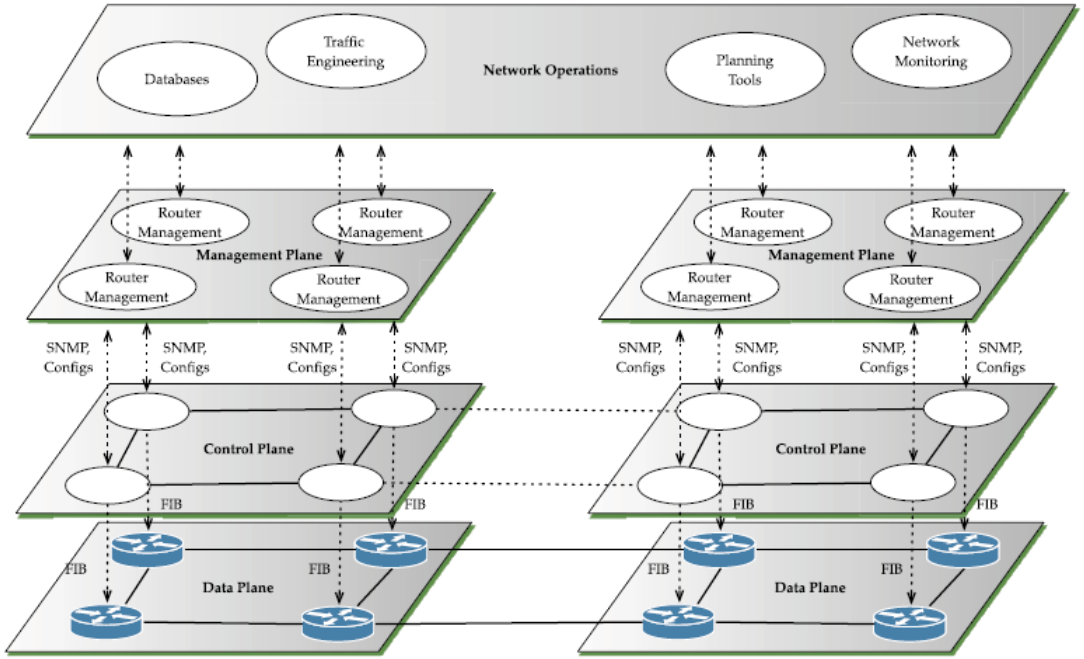
Traffic Engineering

Network Routing Optimization

- Inputs
  - Network topology
  - Link capacities
  - Traffic matrix
- Output
  - Link weights
- Objective
  - Minimize max-utilized link
  - Minimize a sum of link congestion

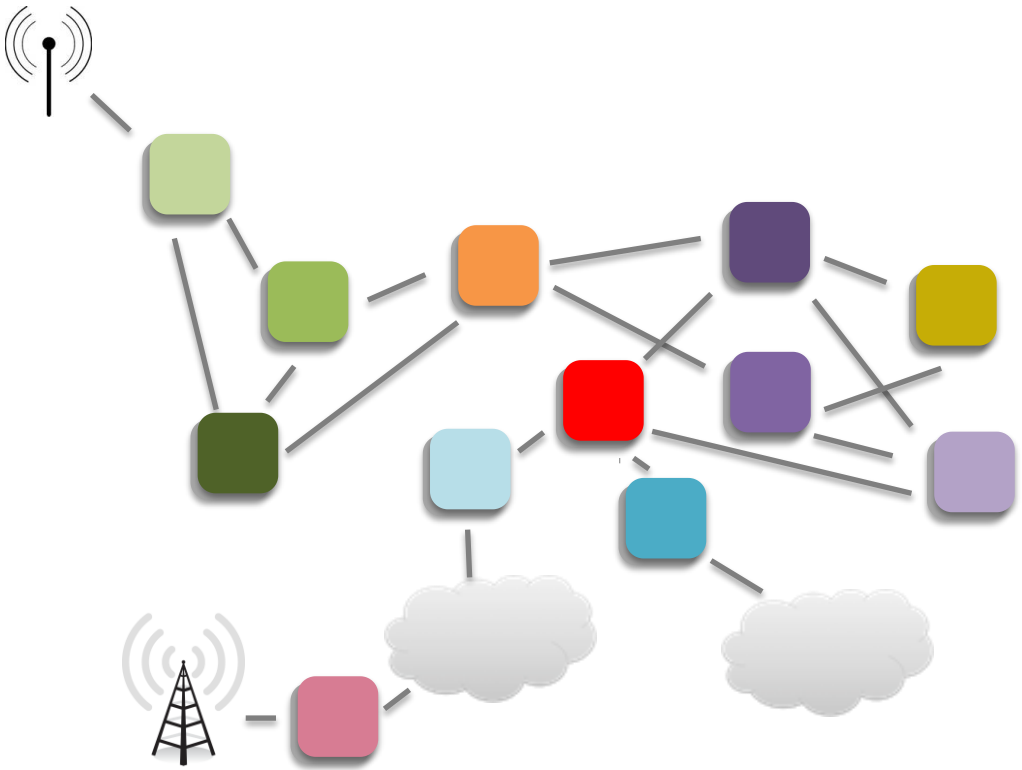


Network Management Architecture

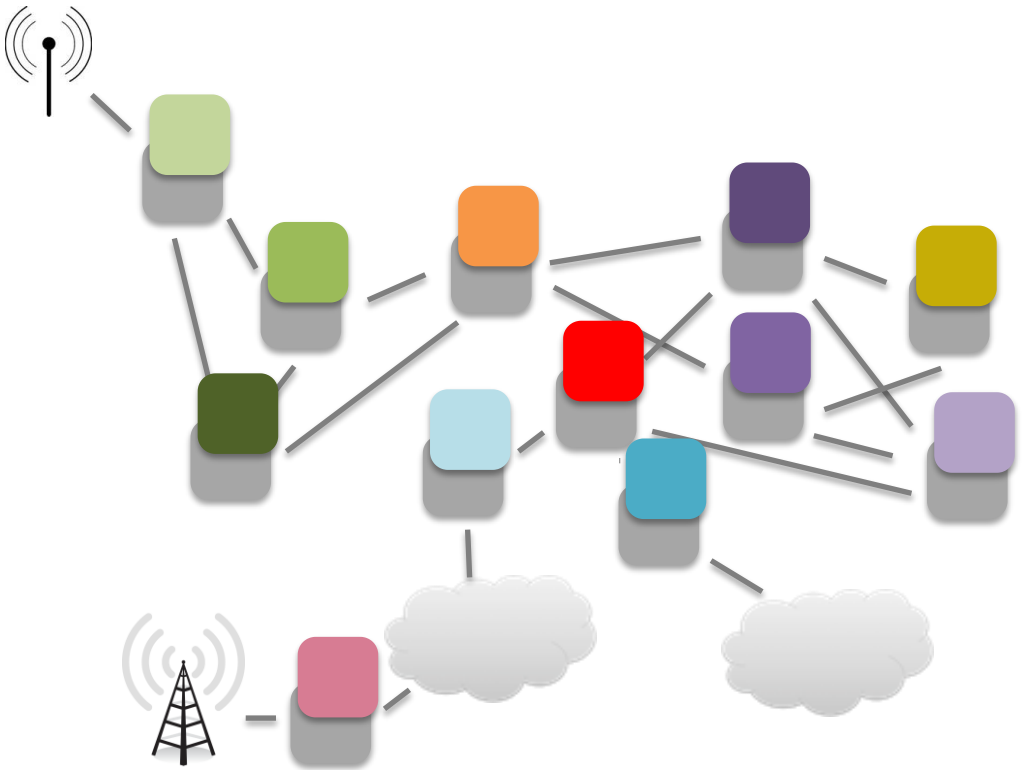


Section 3

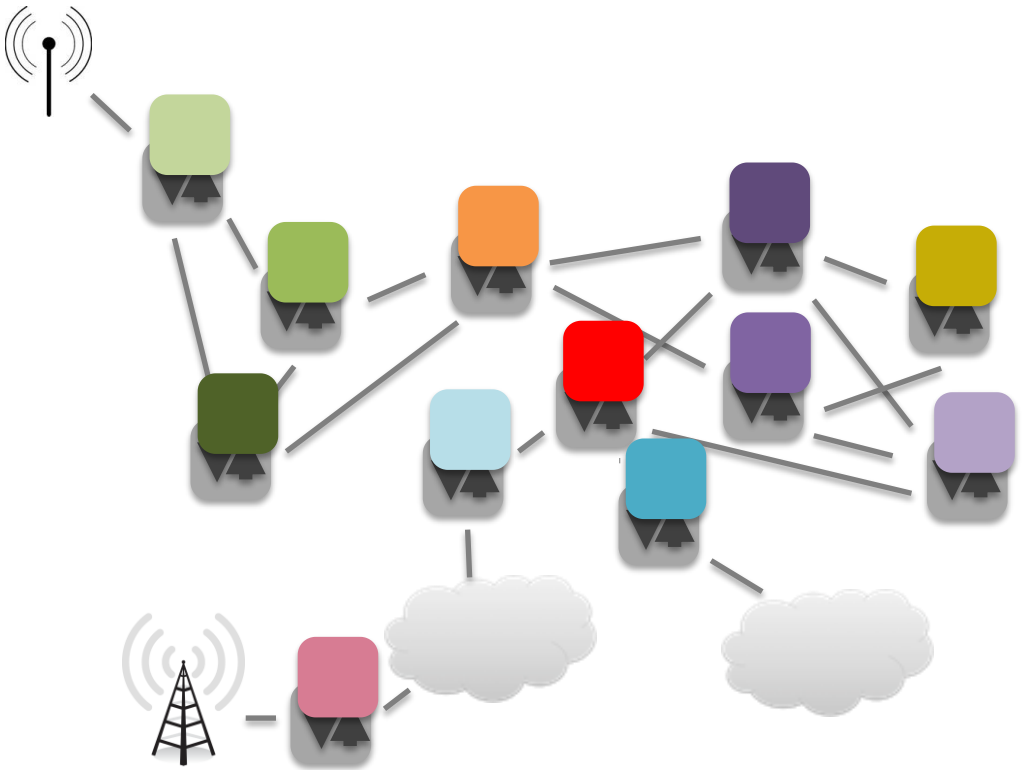
SDN ARCHITECTURAL FRAMEWORK



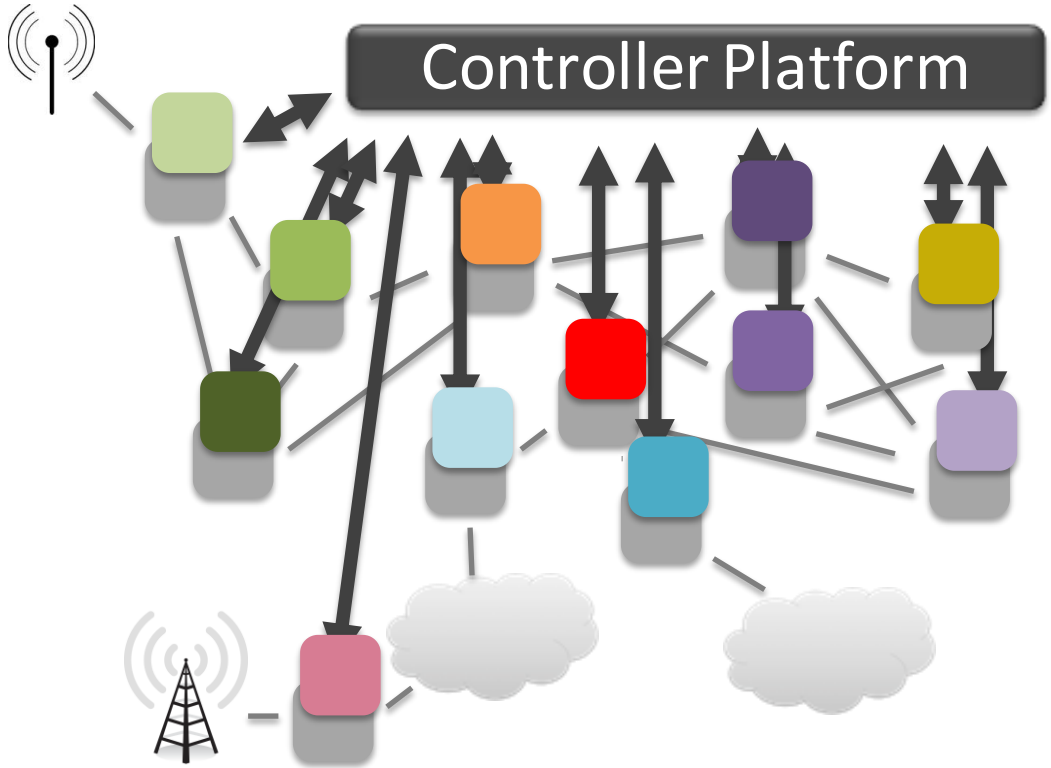
Control plane: distributed algorithms
Data plane: packet processing



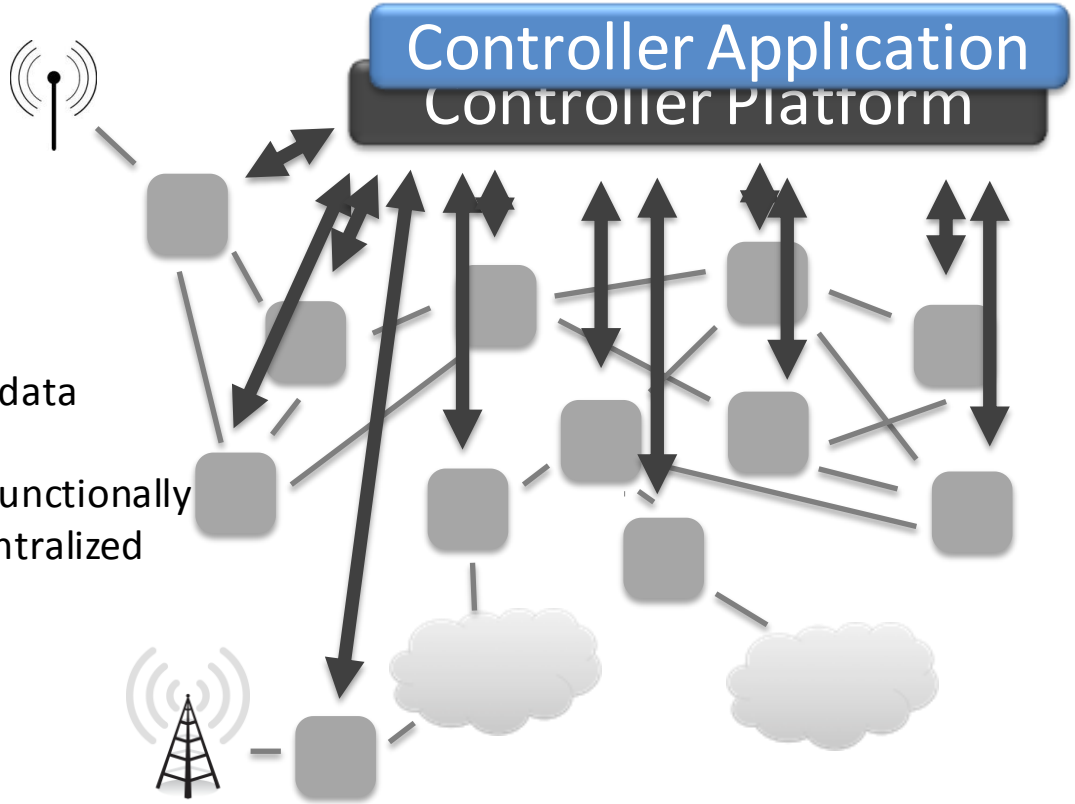
Decoupling control and data planes



Decoupling control and data planes by providing open standard API



Centralized Controller



Idea:

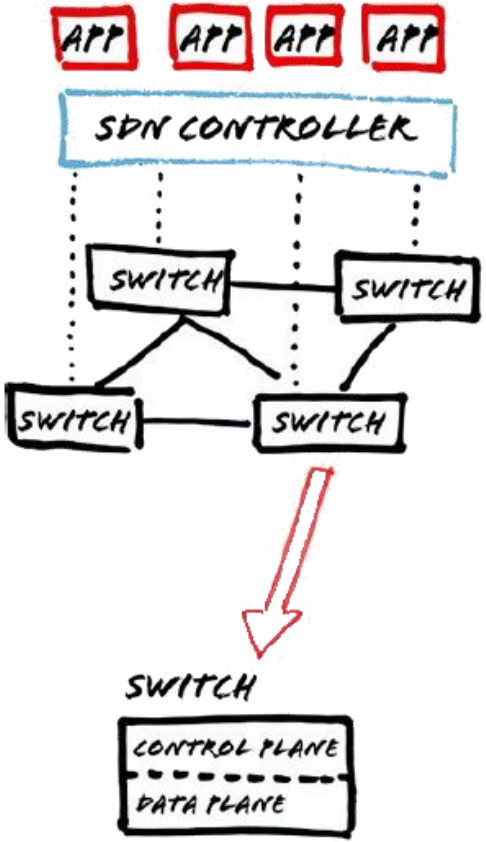
- Separate network intelligence from data path
- Combine control functionally into a logically centralized controller

Protocols ➔ Controller Applications

Network Programmability

SDN realizes the network programmability and adaptability by abstracting the control plane (the intelligence about how packets flow) from data plane.

- SDN is a new approach to networking
  - Not about "architecture": IP, TCP, etc.
  - But about design of network control (routing, TE,…)
- SDN is predicated around two simple concepts
  - ***Separating the control- from the data-plane***
  - ***Providing open API to directly access the data-plane***



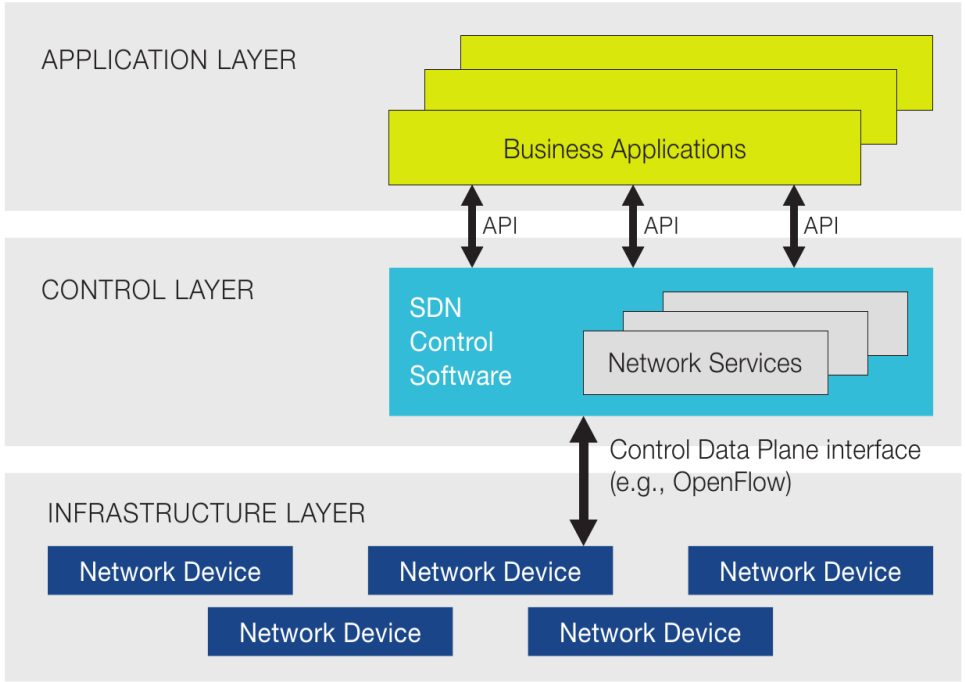
SDN Layered View

- **Infrastructure layer:**

  decouples the hardware from the software, and is responsible for forwarding of the packets.
- **Control layer:**

  contains the controllers providing control data to the data plane (i.e., Infrastructure layer) so that the data traffic gets forwarded as effectively as possible.
- **Application layer:**

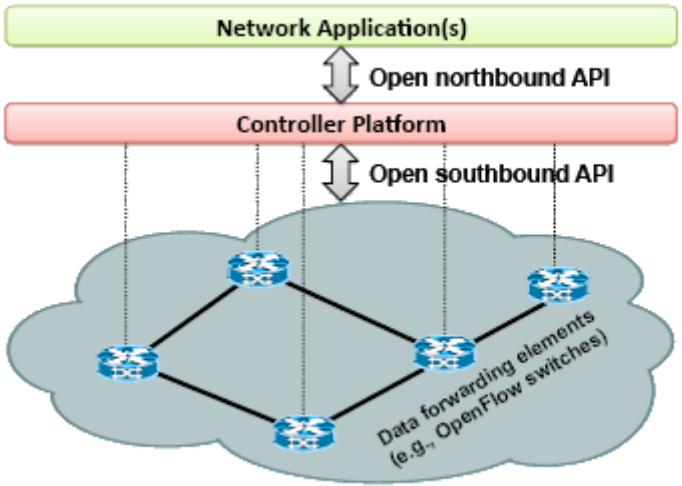
  implementation of the application-specific requirements



SDN Simplified Architecture

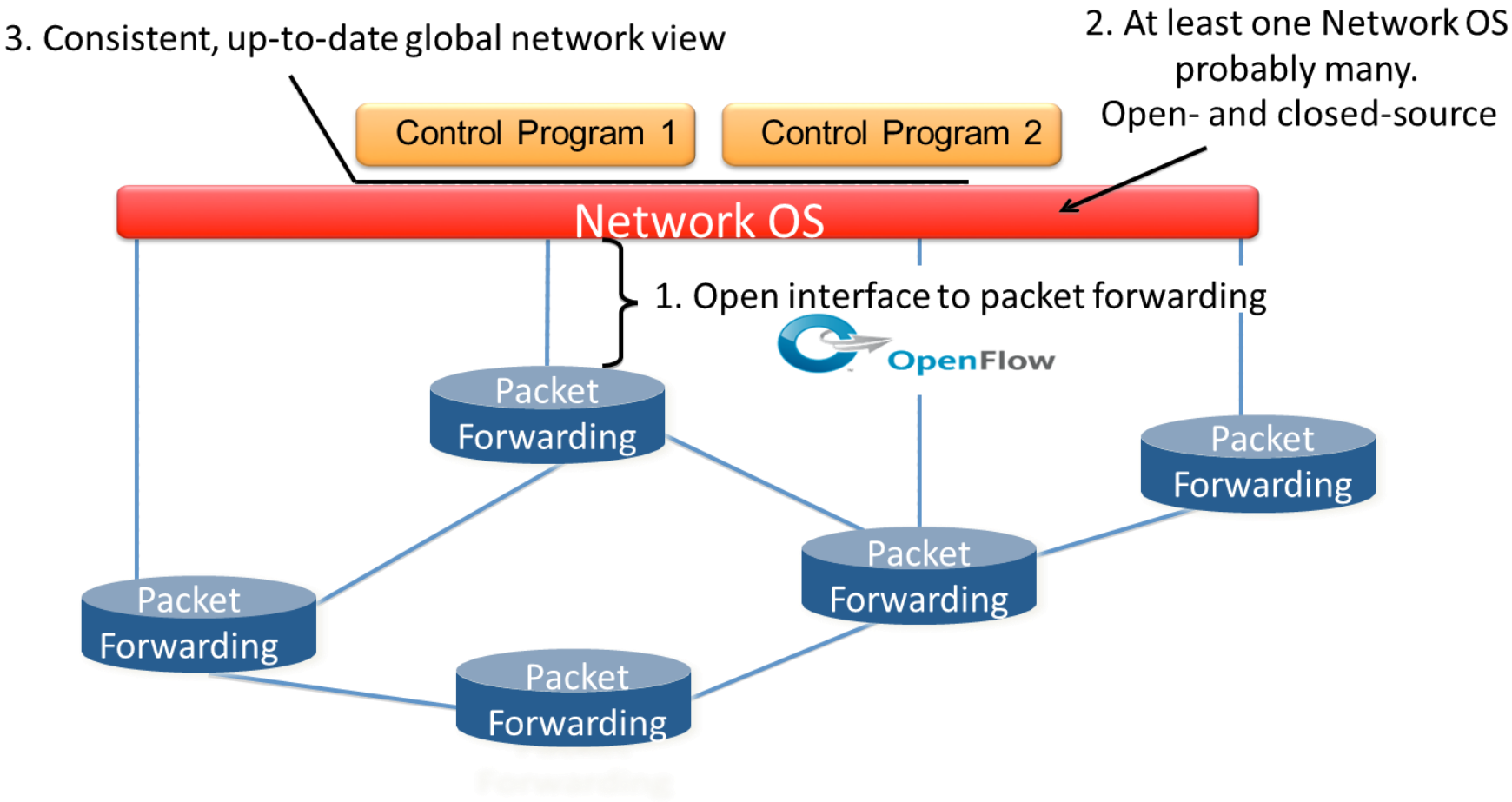
Terminology

- Forwarding Devices (FD)
- Data Plane (DP)
- Southbound Interface (SI)
- Control Plane (CP)
- Northbound Interface (NB)
- Application Plane (AP)



SDN Attributes

1. The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.
2. **Forwarding decisions are flow-based, instead of destination-based**. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions).
3. Control logic is moved to an external entity, the so-called SDN controller or NOS.
4. The network is programmable through software applications running on top of the NOS that interacts with the underlying data plane devices.


3. Consistent, up-to-date global network view
2. At least one Network OS
probably many.
Open- and closed-source
Control Program 1
Control Program 2
Network OS
1. Open interface to packet forwarding
OpenFlow
Packet Forwarding
Packet Forwarding
Packet Forwarding
Packet Forwarding
Packet Forwarding

Fundamental Abstractions

- **Forwarding Abstractions**

  "low-level" instruction sets equivalent to ***device driver***

- **Distribution Abstractions**

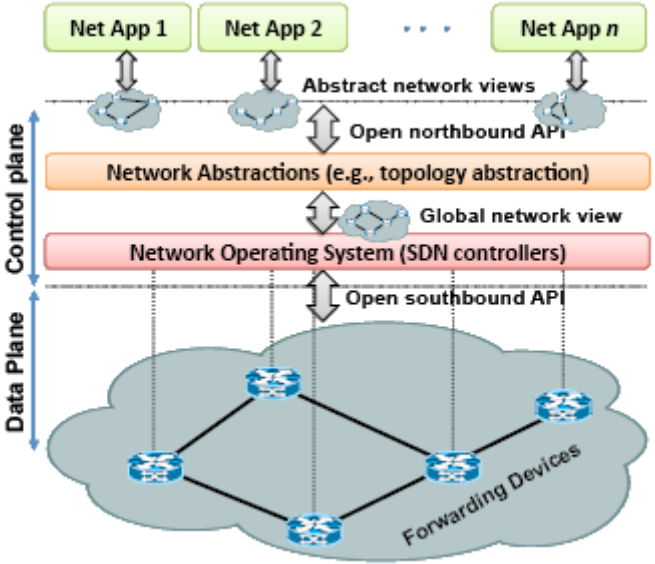
  Realization of a logically centralized control through a common distribution layer equivalent to ***NOS***

  – installing the control commands on the forwarding devices

  – collecting status information about the forwarding layer

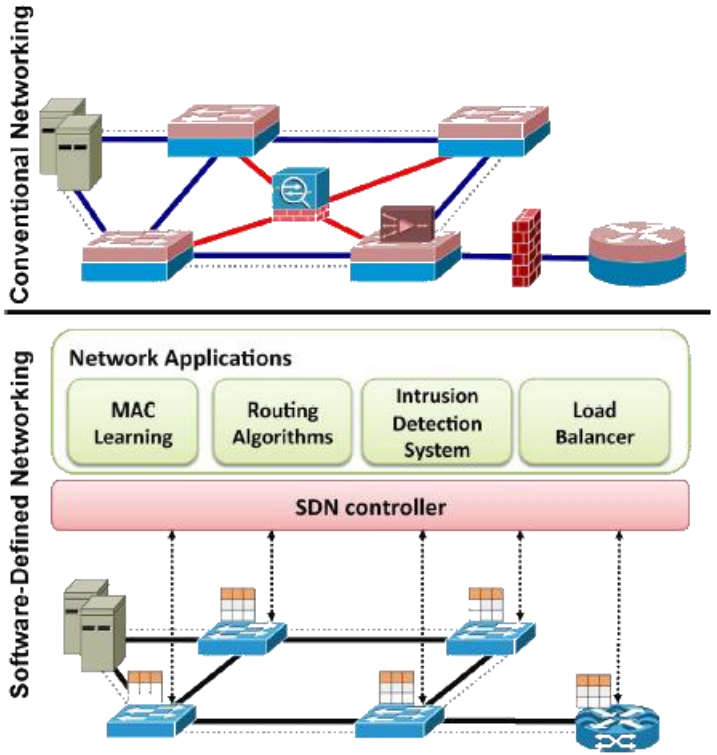
- **Specification Abstractions**

  "high-level" instruction sets

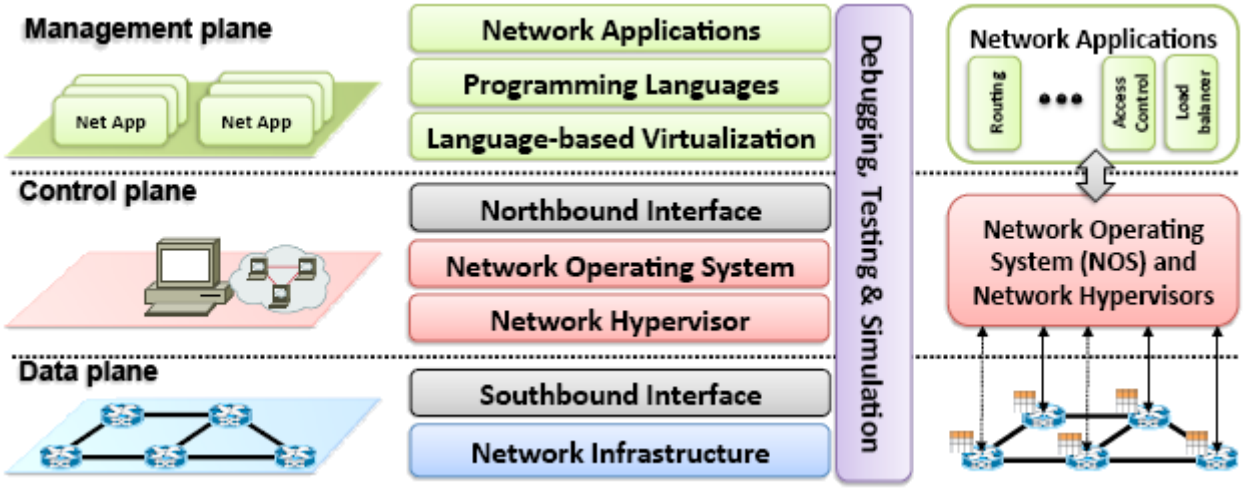


SDN Middle-Box Services

With SDN, management becomes simpler and middle-boxes services can be delivered as SDN controller applications.
It becomes easier to program these applications since the abstractions provided by the control platform and/or the network programming languages can be shared.

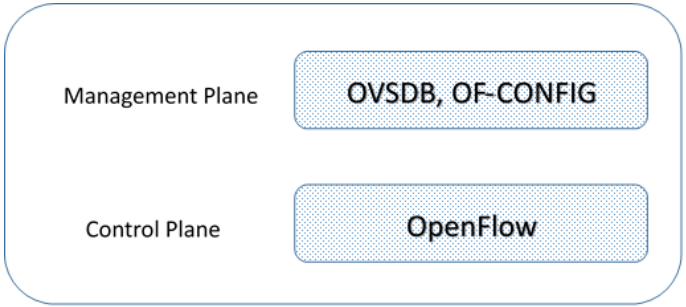


SDN Layered Architecture



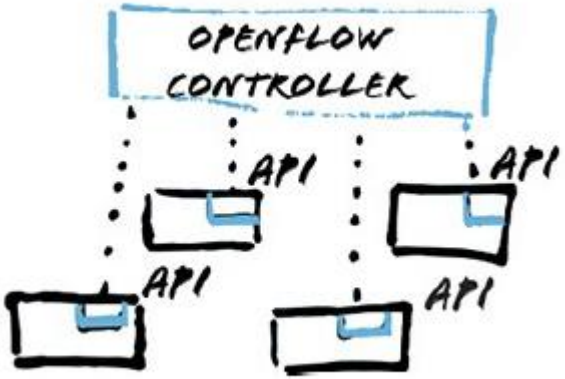
Southbound APIs

- Southbound APIs can be seen as a layer of device drivers.
- They provide a common interface for the upper layers, while
  - and protocol plug-ins to ***manage*** existing or new physical or virtual devices e.g., SNMP, BGP, NetConf and OVSDB (for Open vSwitches)
  - allowing a ***control*** platform to use different southbound APIs e.g., OpenFlow



OpenFlow-enabled SDN

- OpenFlow is an open source control protocol that all vendors can equally support.
- An API can be places on the vendor switches, enabling them to be programmed without exposing their code.

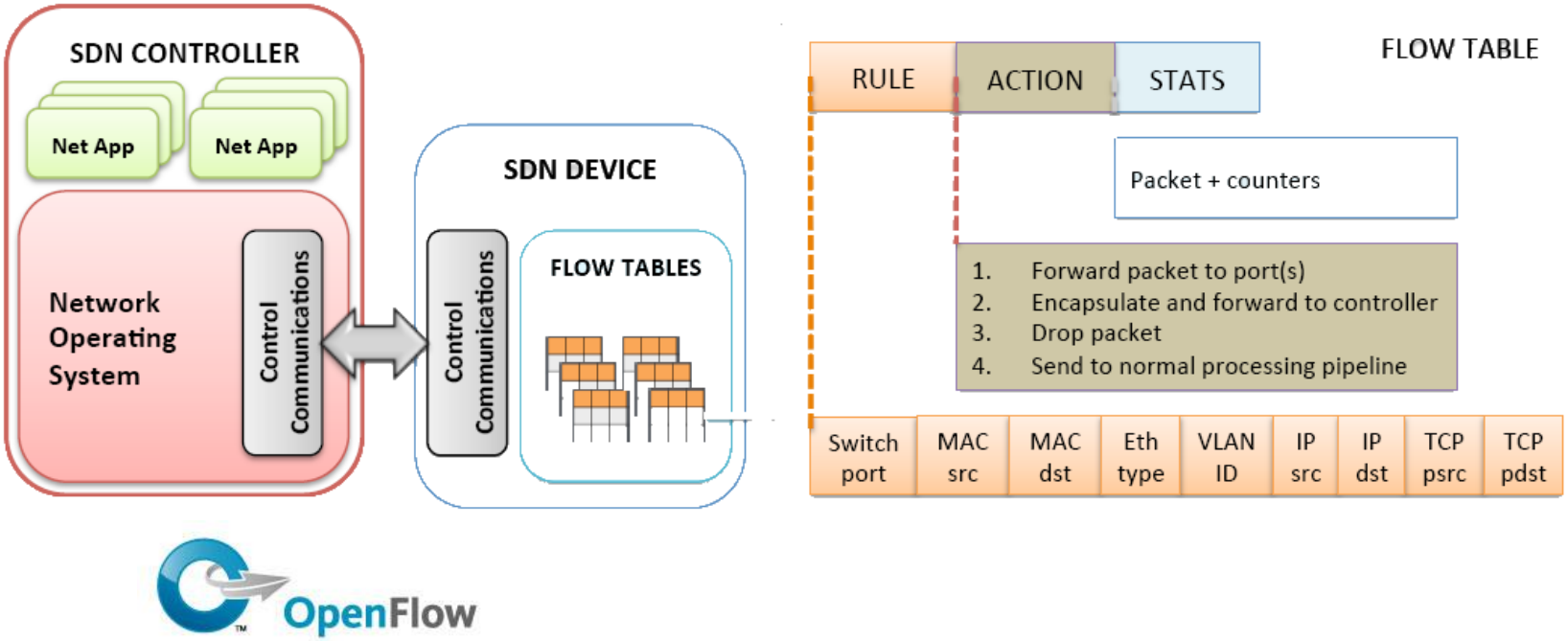


- OpenFlow as the most widely accepted and deployed open southbound standard for SDN promotes **interoperability**, allowing the deployment of vendor-agnostic network devices from different vendors.

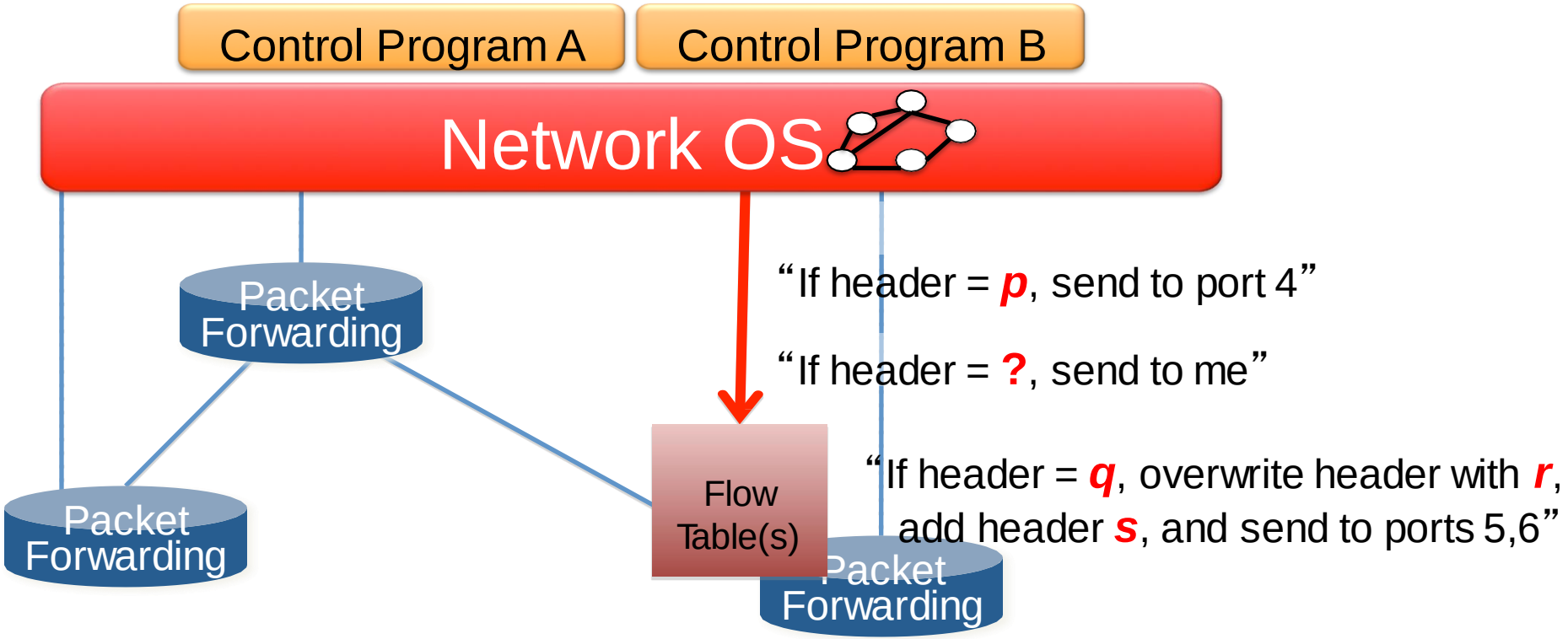
OpenFlow Protocol

- Standardized by the Open Networking Foundation (ONF).
- It describes the interaction of one or more control servers with OpenFlow-compliant switches.
- the OpenFlow protocol provides a “southbound” interface that allows a control software to program switches in the network
- OpenFlow maintains what it calls a flow table on the device, which contains the information of how the data needs to be forwarded.
- The SDN controller can then use OpenFlow to program the forwarding plane of an OpenFlow-enabled switch and changing its forwarding behaviour by altering this flow table.

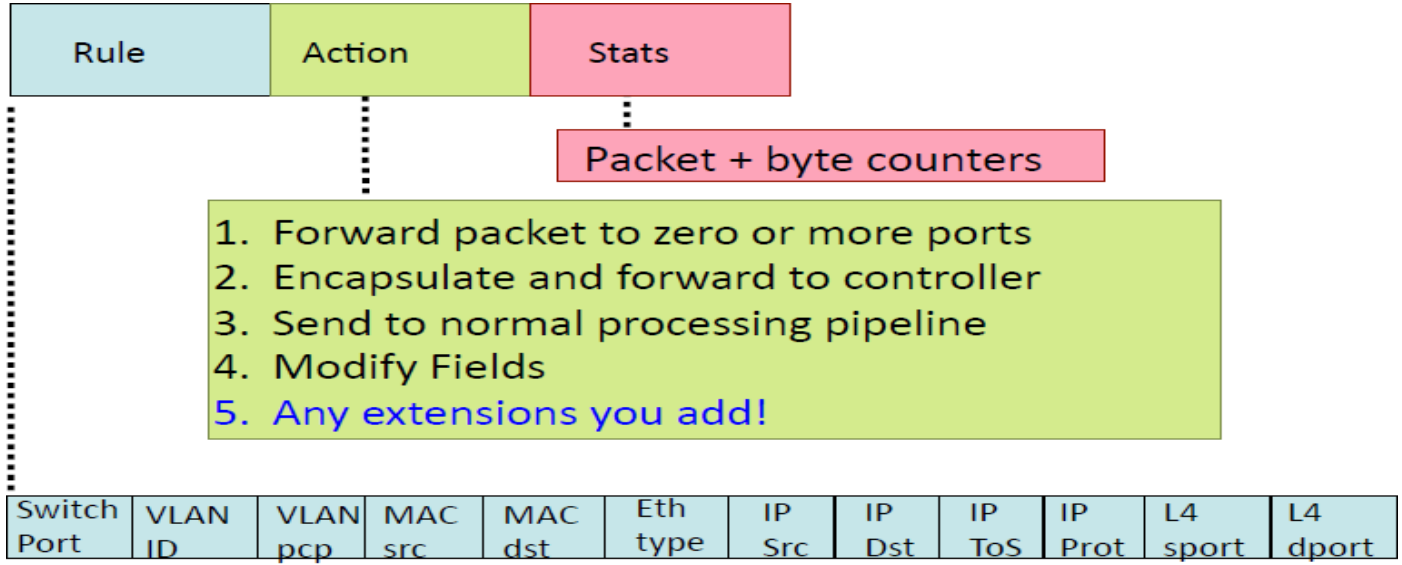
OpenFlow-enabled Switches



OpenFlow Basics



Flow Table Entries



+ mask what fields to match

Flow Table 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

Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

Firewall

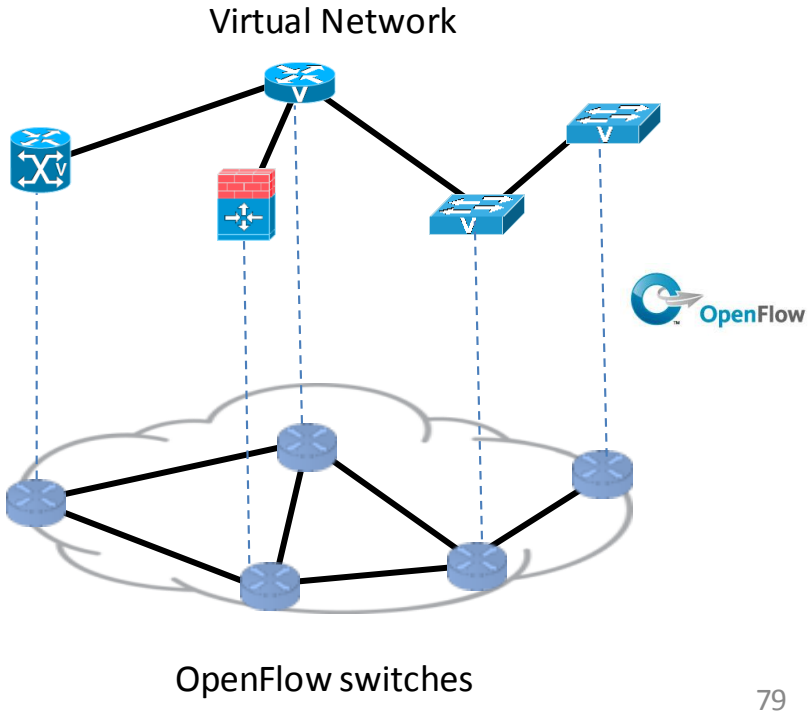
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

OpenFlow Unified "Boxes"

- Router
  - Match: longest destination IP prefix
  - Action: forward out a link
- Switch
  - Match: destination MAC address
  - Action: forward or flood
- Firewall
  - Match: IP addresses and TCP/UDP port numbers
  - Action: permit or deny
- NAT
  - Match: IP address and port
  - Action: rewrite address and port

Network Node Type Abstraction

- **OpenFlow switch** can become any of classical network elements:
  - Router
  - Switch
  - Gateway
  - Firewall
  - Load balancer
- **Freedom** of choosing virtual nodes type and functionality



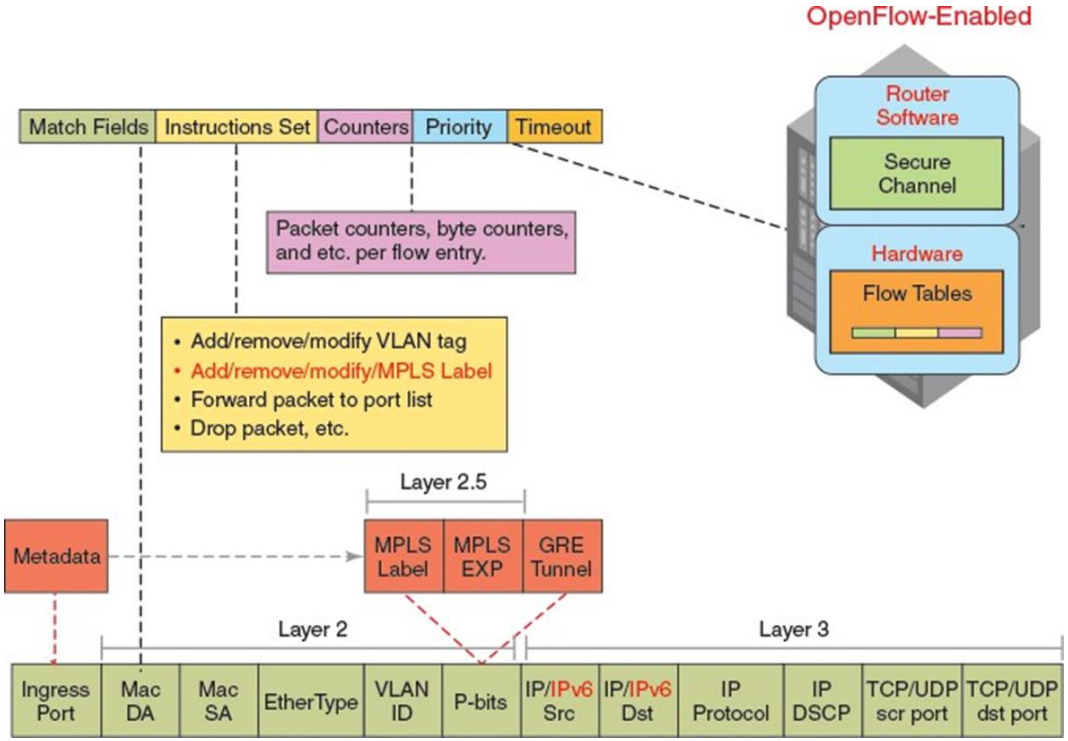
Supported Match Fields

OpenFlow Version	Match fields	Statistics	# Matches		# Instructions		# Actions		# Ports	
			Req	Opt	Req	Opt	Req	Opt	Req	Opt
v 1.0	Ingress Port	Per table statistics	18	2	1	0	2	11	6	2
	Ethernet: src, dst, type, VLAN	Per flow statistics								
	IPv4: src, dst, proto, ToS	Per port statistics								
	TCP/UDP: src port, dst port	Per queue statistics								
v 1.1	Metadata, SCTP, VLAN tagging	Group statistics	23	2	0	0	3	28	5	3
	MPLS: label, traffic class	Action bucket statistics								
v 1.2	OpenFlow Extensible Match (OXM)		14	18	2	3	2	49	5	3
	IPv6: src, dst, flow label, ICMPv6									
v 1.3	PBB, IPv6 Extension Headers	Per-flow meter	14	26	2	4	2	56	5	3
		Per-flow meter band								
v 1.4	—	—	**14**	**27**	**2**	**4**	**2**	**57**	**5**	**3**
		Optical port properties								

L2/3/4 Match Fields Specification

Field	Meaning
Layer 2 fields	
Ingress Port	Switch port over which the packet arrived
Metadata	64-bit field of metadata used in the pipeline
Ether src	48-bit Ethernet source address
Ether dst	48-bit Ethernet destination address
Ether Type	16-bit Ethernet type field
VLAN id	12-bit VLAN tag in the packet
VLAN priority	3-bit VLAN priority number
ARP opcode	8-bit ARP opcode
Layer 3 fields	
MPLS label	20-bit MPLS label
MPLS class	3-bit MPLS traffic class
IPv4 src	32-bit IPv4 source address
IPv4 dst	32-bit IPv4 destination address
IPv6 src	128-bit IPv6 source address
IPv6 dst	128-bit IPv6 destination address
IPv4 Proto	8-bit IPv4 protocol field
IPv6 Next Header	8-bit IPv6 next header field
TOS	8-bit IPv4 or IPv6 Type of Service bits
Layer 4 fields	
TCP/UDP/SCTP src	16-bit TCP/UDP/SCTP source port
TCP/UDP/SCTP dst	16-bit TCP/UDP/SCTP destination port
ICMP type	8-bit ICMP type field
ICMP code	8-bit ICMP code field

Switch Architecture



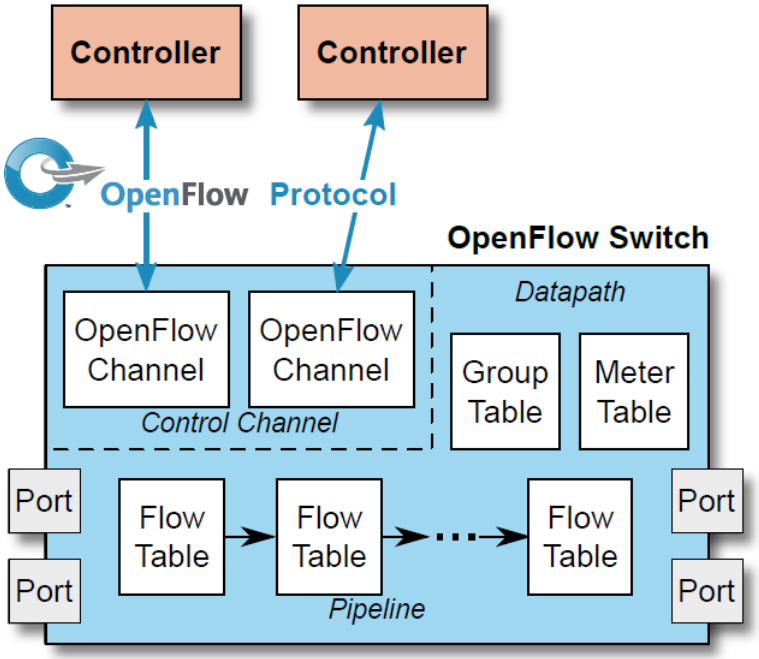
Flow-based- vs. IP-Forwarding

In SDN framework s single classification pattern can check items from multiple layers of the protocol stack at the same time.

- Example: key items that determine whether a packet contains web traffic
  - The (layer 2) frame type specifies IP
  - The (layer 3) IP protocol field specifies TCP
  - The (layer 4) TCP destination protocol port is 80

Because it uses classification, OpenFlow can define new forms of IP forwarding in which the choice of a next hop depends on fields other than the destination IP address; OpenFlow can use the IP source address or fields in the layer 4 header, such as the TCP port numbers.

OF-enabled Switch Architecture



Control Channel

- Each OpenFlow switch connects to the SDN controller over a TCP connection
- OpenFlow protocol defines a set of messages between the switch and the controller that allow:
  - Controller to program flow tables on the switches
  - Controller to query information from switches
  - Switches to send asynchronous notifications to the controller
- Control channel can be
  - In-band: Switches have default rules to forward control packets
  - Out-of-band: Control network can be a non-SDN network

Ports

- Ports are abstractions for switch interfaces where packets can be pushed to/pulled from
- Three types of ports:
  - *Physical*: corresponds to a hardware interface
  - *Logical*: higher level abstractions that might not physically exist, *e.g.*, representing multiple physical ports as a single logical port
  - *Reserved*: restricted-use ports as defined by OpenFlow specification, *e.g.*, port used for communication with controller

Flow Table

- Flow table matches incoming packets against a set of forwarding rules (each rule is called a flow table entry)
- For faster packet matching, the flow table is implemented in hardware using Ternary Content Addressable Memory (TCAM)
  - expensive and power-hungry thus very limited resource
  - can hold limited flow table entries, therefore, old entries are evicted

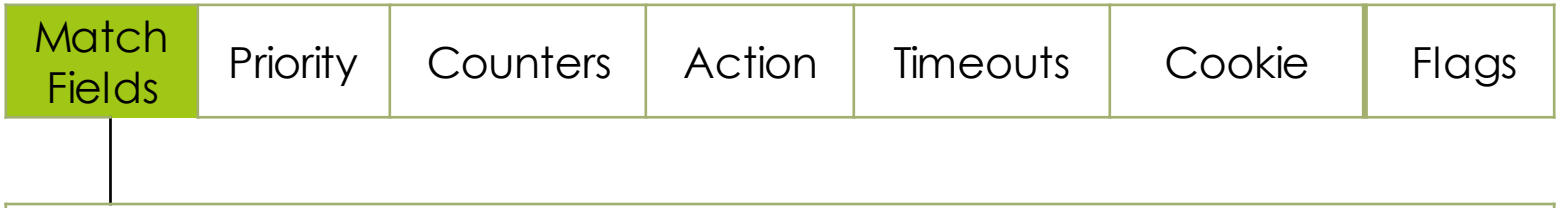
Flow Table Structure

- Flow table consists of rows called Flow table entries. An entry is structured as follows:

Match Fields	Priority	Counters	Action	Timeouts	Cookie	Flags

  - Match fields: used to match packets to flows based on header fields
  - Priority: breaks tie between conflicting matches
  - Counters: per-flow entry statistics (*e.g.*, number of packets/bytes matched)
  - Action: to be performed after a match
  - Timeouts: idle or a hard timeout before flow entry is evicted
  - Cookie: set by the controller to identify a flow entry
  - Flags: determine behaviour wrt. a flow entry (*e.g.*, notify controller when entry is evicted)

Flow Table Structure (cont)

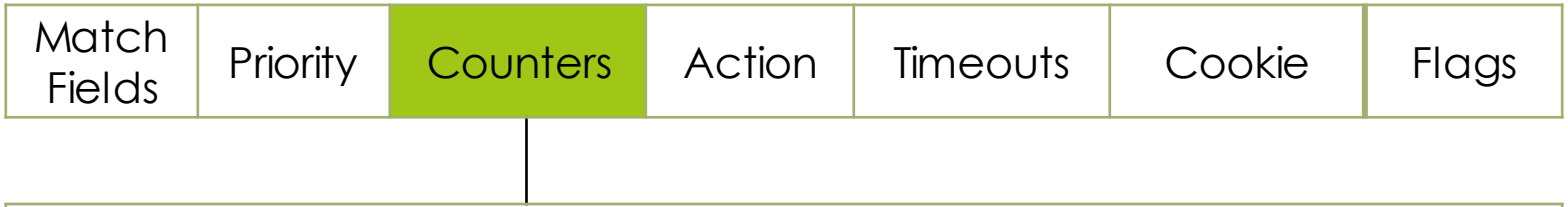


- Ingress Port: The identifier of the port where packet was received
- Layer 2 Headers: Source/Destination MAC addresses, Ethernet type etc.)
- Layer 3 Headers: Source/Destination IP addresses, IP protocol *etc.*
- Layer 4 Headers: TCP/UDP source/destination ports
- VLAN ID: Virtual LAN tag number
- Meta-data: Additional information passed from a previous flow table (when flow tables are pipelined)
- Many more (OpenFlow 1.4 defines 41 match fields)

Flow Table Structure (cont)

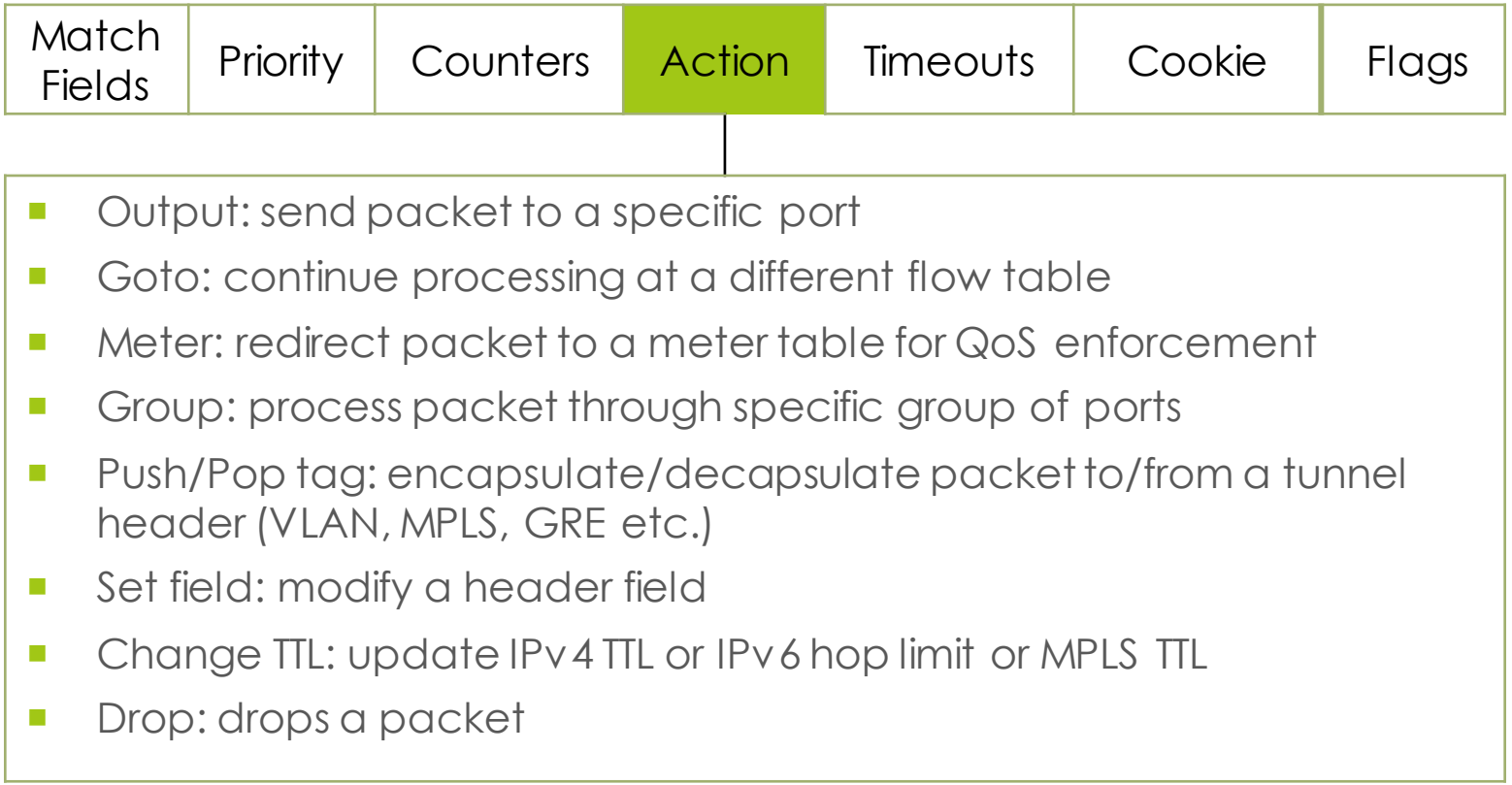
- Part of every single rule
- If several patterns match a packet header :
  - the highest priority is used
  - if similar priorities, the most precise match is used (lowest wildcard)
- The values vary from 0 to 65535, if unspecified, 32768 set by default

Flow Table Structure (cont)



- Counters are associated with flow table entries, the flow table itself, queues, groups, *etc.*
  - reference count: number of active entries in flow table
  - duration: duration a flow table entry is in TCAM
  - received/sent packets: per port counter
  - matched packets/bytes: number of packets/bytes matching a flow table entry
  - many more…

Flow Table Structure (cont)



Flow Table Structure (cont)

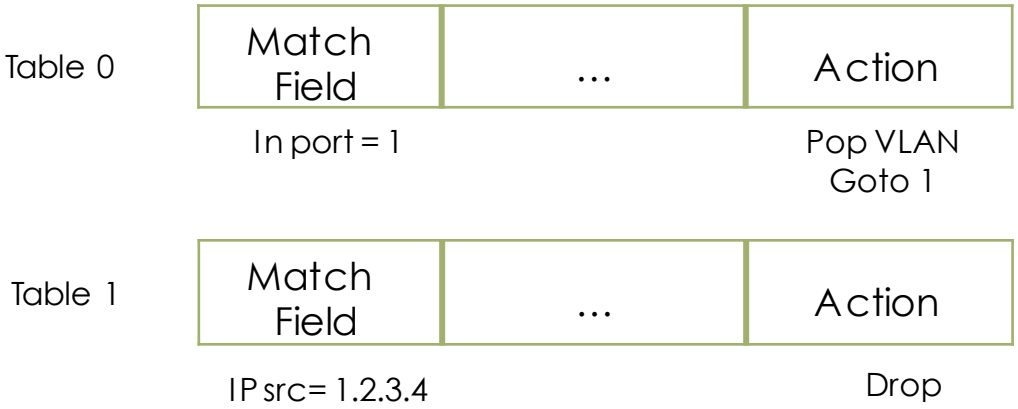
- idle_timeout: A flow table entry is evicted from TCAM if no packets have matched the entry in the past idle_timeout time duration

  Setting a value of 0 disables the timer
- hard_timeout: A flow table entry is evicted from TCAM after hard_timeout period elapses from the time of installing the entry

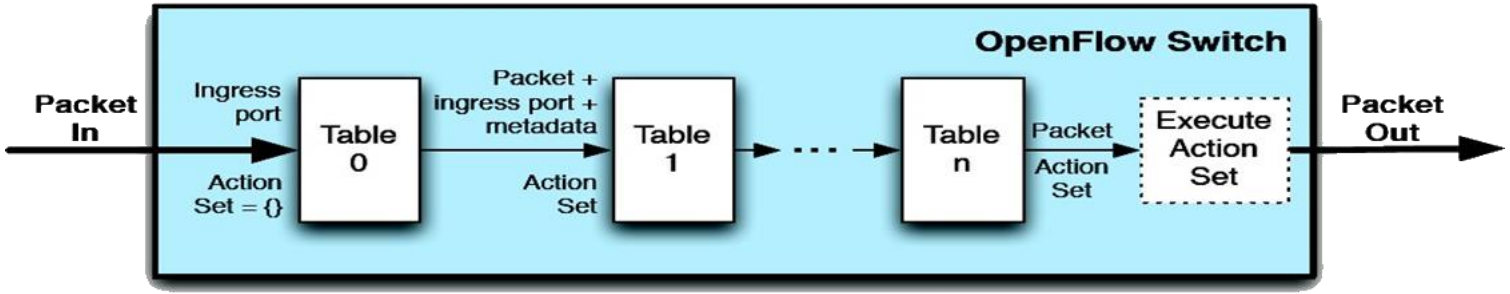
  Setting a 0 value disables this timer

Flow Table Pipelining

- Packets can be processed sequentially in multiple flow tables
- Pipelining modularizes different packet processing actions
  - For example, the first table can perform only VLAN stripping and the second table can perform access control



Pipeline Processing



- Flow tables sequentially numbered
- A flow table entry adds actions to an action set of a packet
- A flow table entry may explicitly direct the packet to another flow table
  - Can only direct a packet to a flow table with number which is greater than its own table's number
- If table entry does NOT direct a packet to another table, the pipeline stops and the associated action set is executed

OpenFlow key messages

- Southbound API supports communication between SDN control and data plane
- TCP is used to exchange OpenFlow messages between a switch and the controller
  - Port 6633 and 6653 have been reserved by IANA
  - TLS encryption is supported (optional)
- three classes of OpenFlow messages:
  - Asynchronous switch-to-controller
  - Controller-to-switch
  - Symmetric messages

OpenFlow key messages (cont)

- Switch-to-controller messages
  - PacketIn: When a packet does not match any flow table entry the switch encapsulates the packet in a PacketIn message and sends it to the controller. This message is a request from the switch to the controller for path setup
  - Flow-based Statistics
    - Flow Removed: When a flow table entry expires a Flow Removed message is sent to the controller
  - Event-based
    - Port Status: A switch uses this message to inform controller about any status change of a port (e.g., port went down)

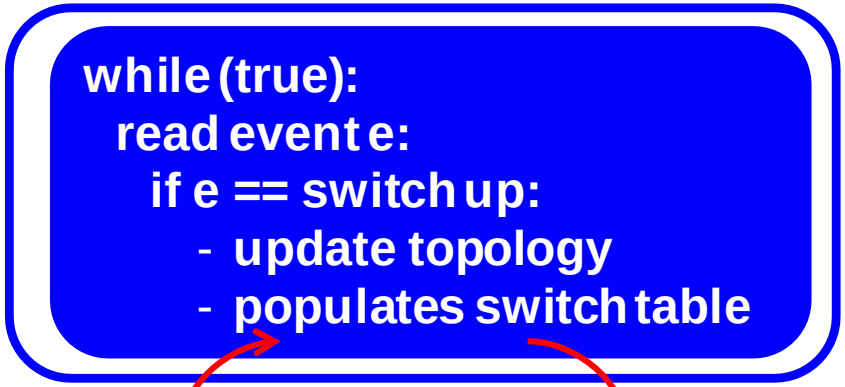
OpenFlow key messages (cont)

- Controller-to-switch messages
  - *SetConfig*: this message is used by a controller to configure switch parameters
  - *FlowMod/GroupMod/PortMod*: controller uses these messages to modify switch state such as flow table entry, group tables, port status *etc.*
  - *PacketOut*: with this message the controller asks a switch to send packet out of a specific switch port

OpenFlow key messages (cont)

- Symmetric Messages
  - *GetConfigReq/GetConfigRes:* these messages allow synchronous communication between switch and controller to exchange current configuration state
  - *FeatureReq/FeatureRes*: these messages are used by a controller to query the features supported by a switch
  - *StatReq/StatRes:* these messages allow controller to read statistics (packet count, byte count, errors *etc.*) from the switch. Statistics can be per port, per table or per flow entry

Controller Programmability



Receive flow-level information from switches

Topology changes, traffic statistics and arriving packets

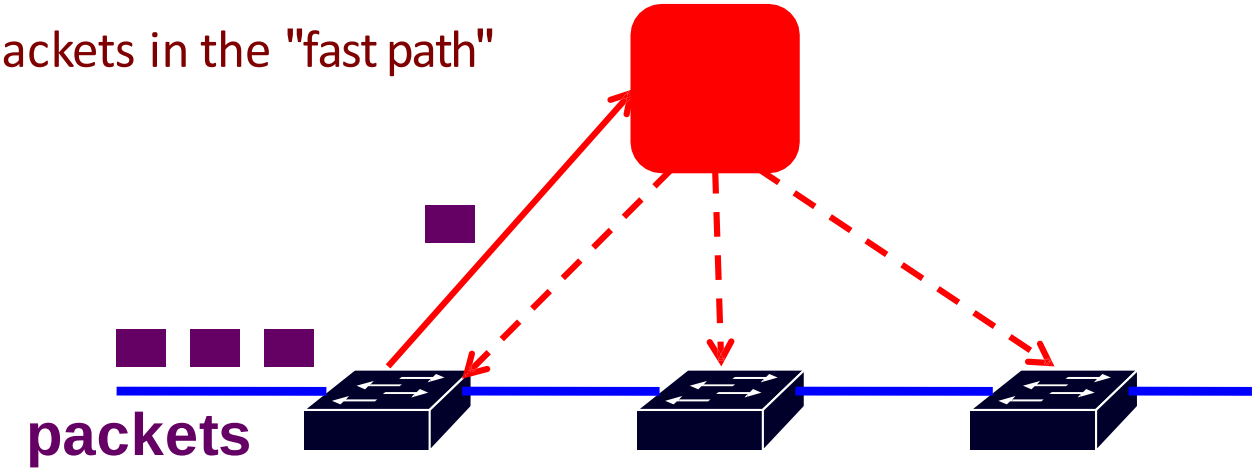
Send commands to switches

(Un-)install rules, query statistics and send packets

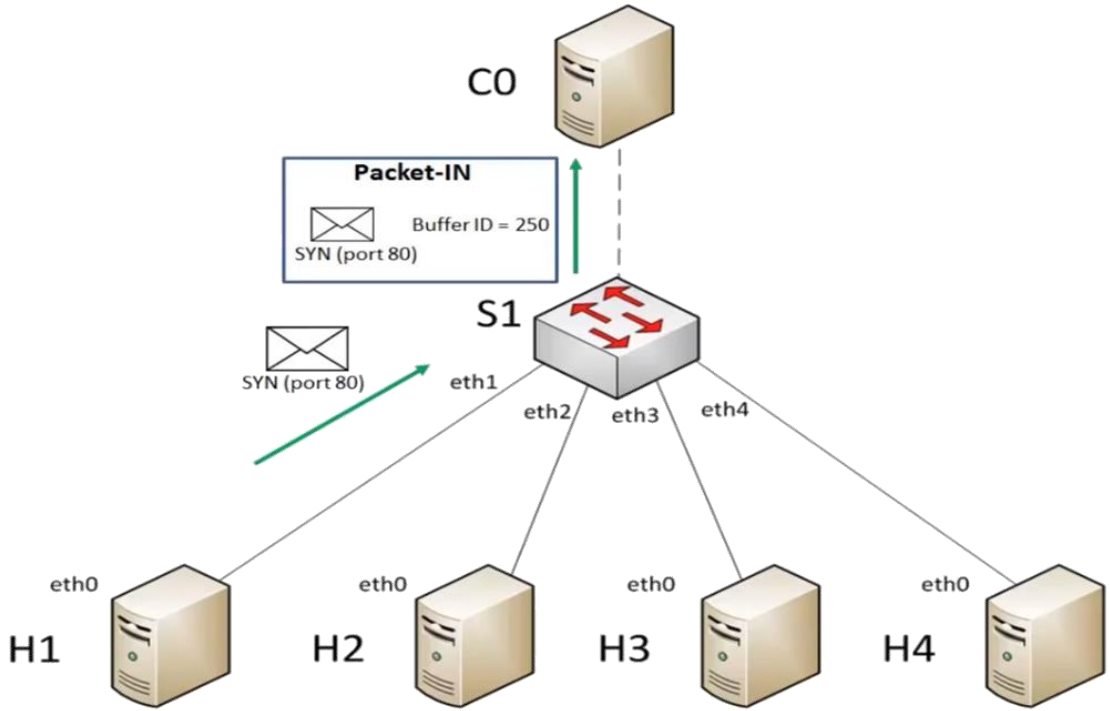
SDN Challenges

Controller Delay and Overhead

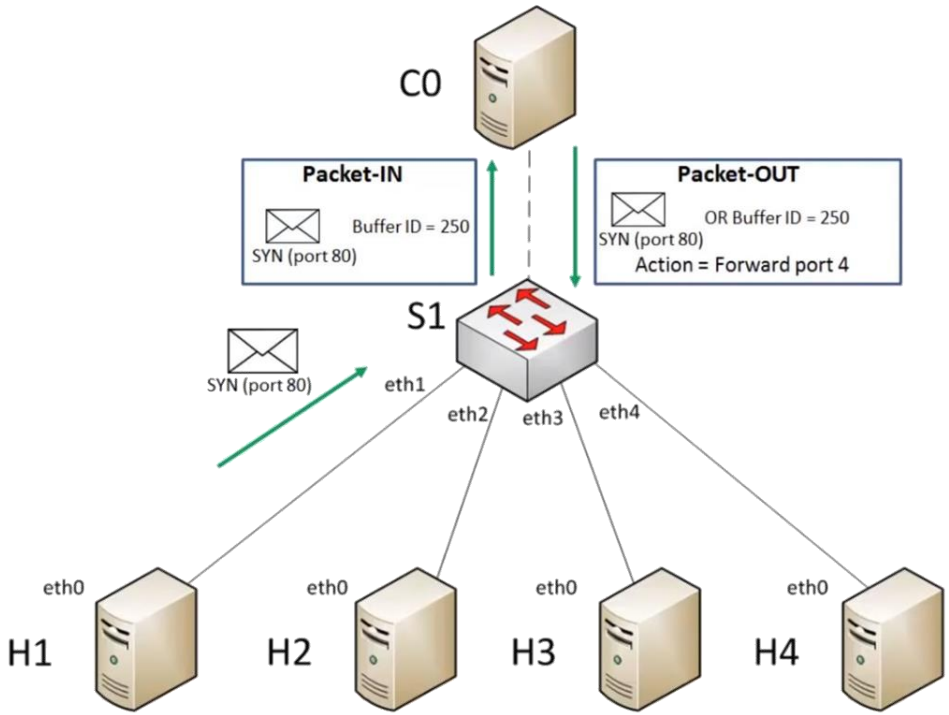
- Controller is much slower than the switch
- Processing packets leads to delay and overhead
- Need to keep most packets in the "fast path"



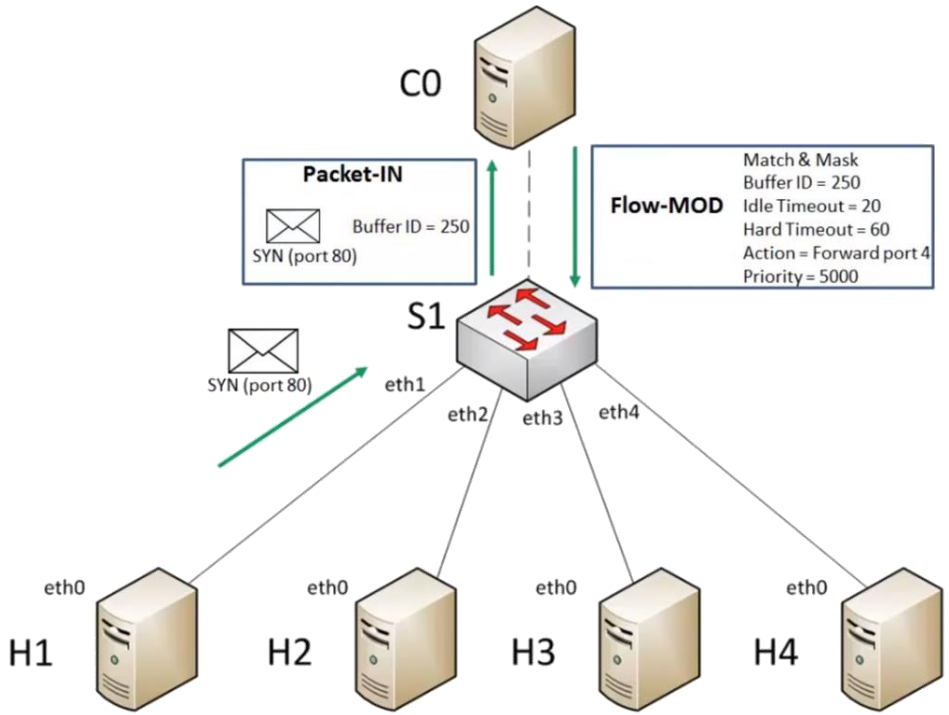
OVS Flow Forwarding Example



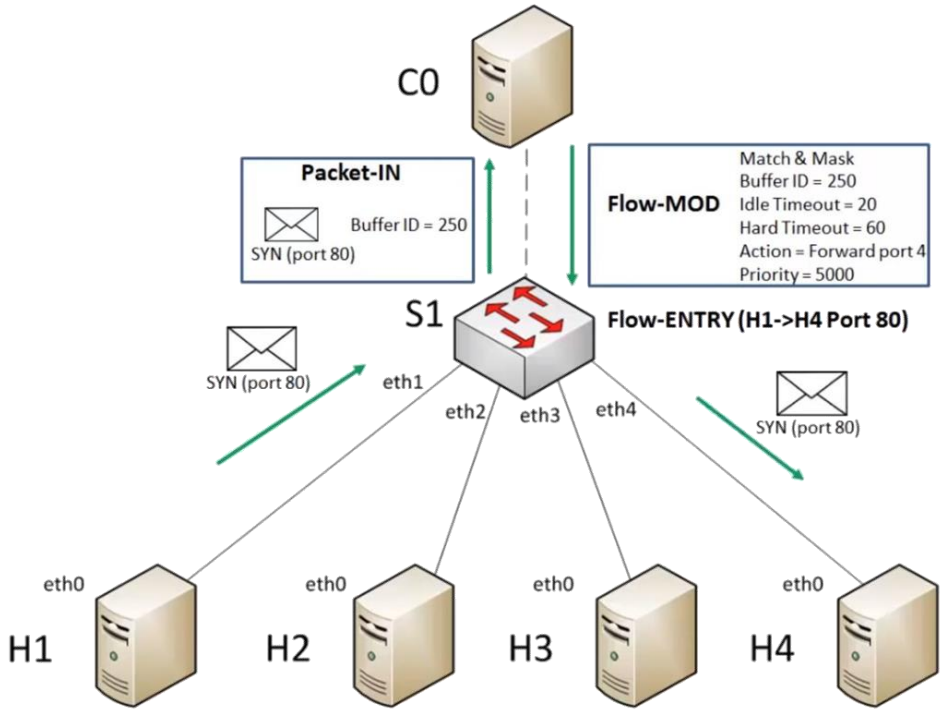
OVS Flow Forwarding Example



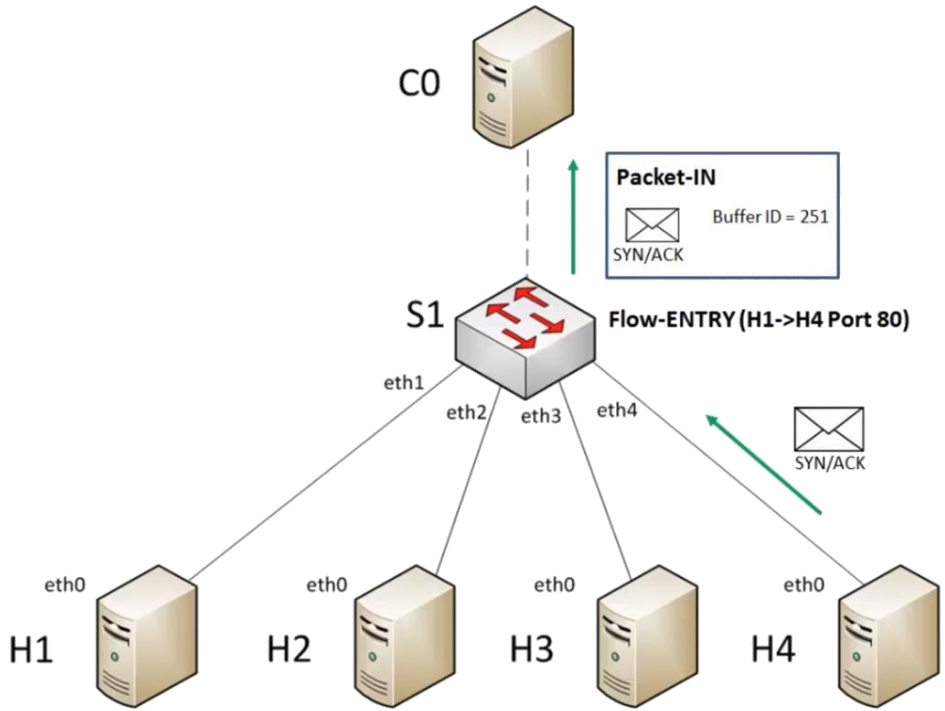
OVS Flow Forwarding Example



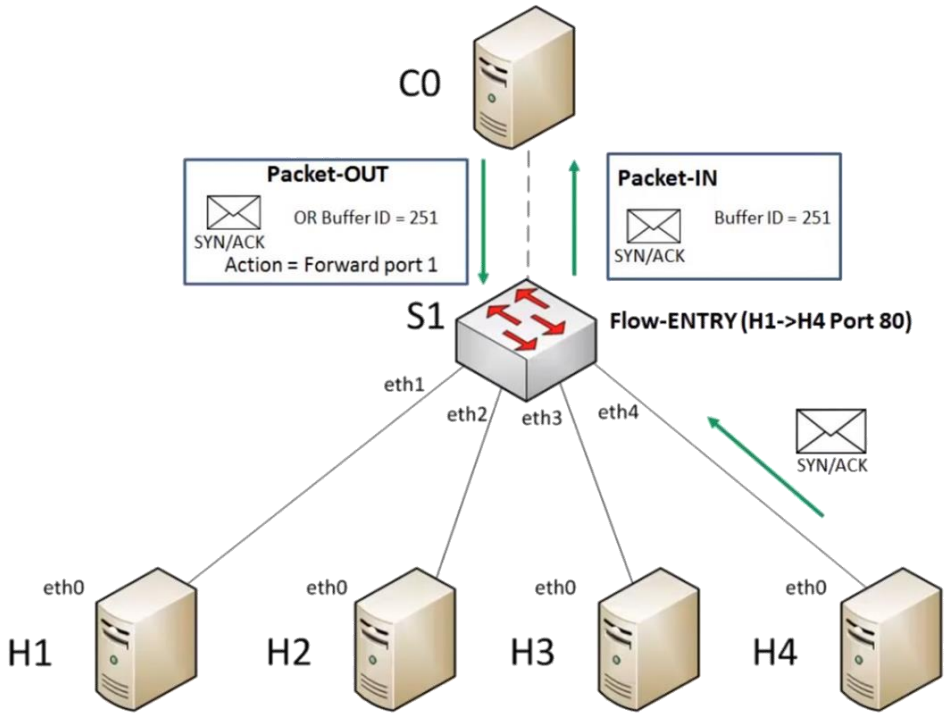
OVS Flow Forwarding Example



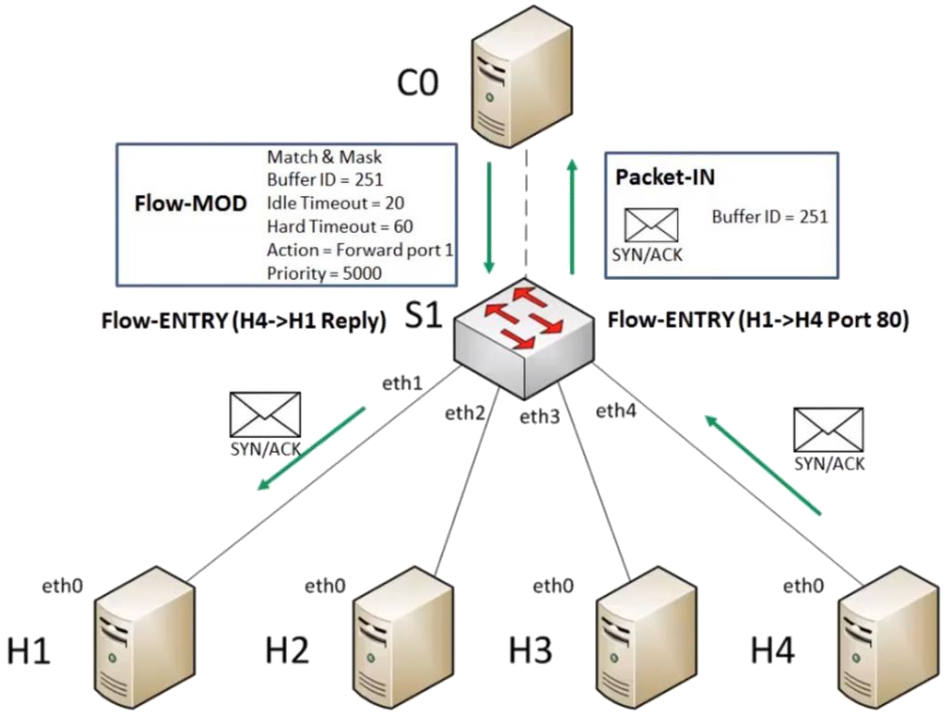
OVS Flow Forwarding Example



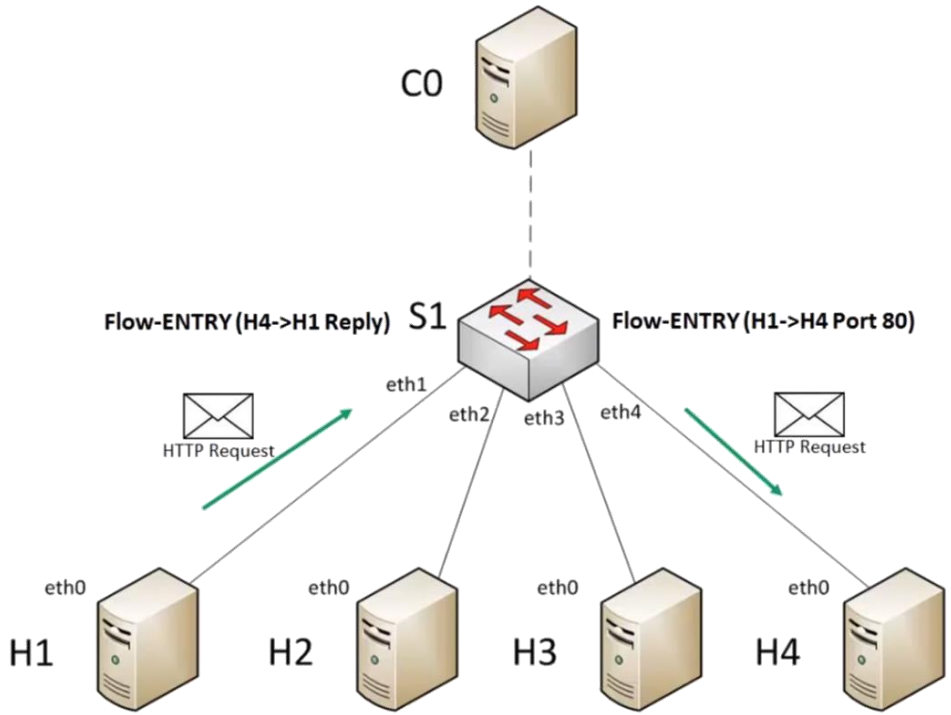
OVS Flow Forwarding Example



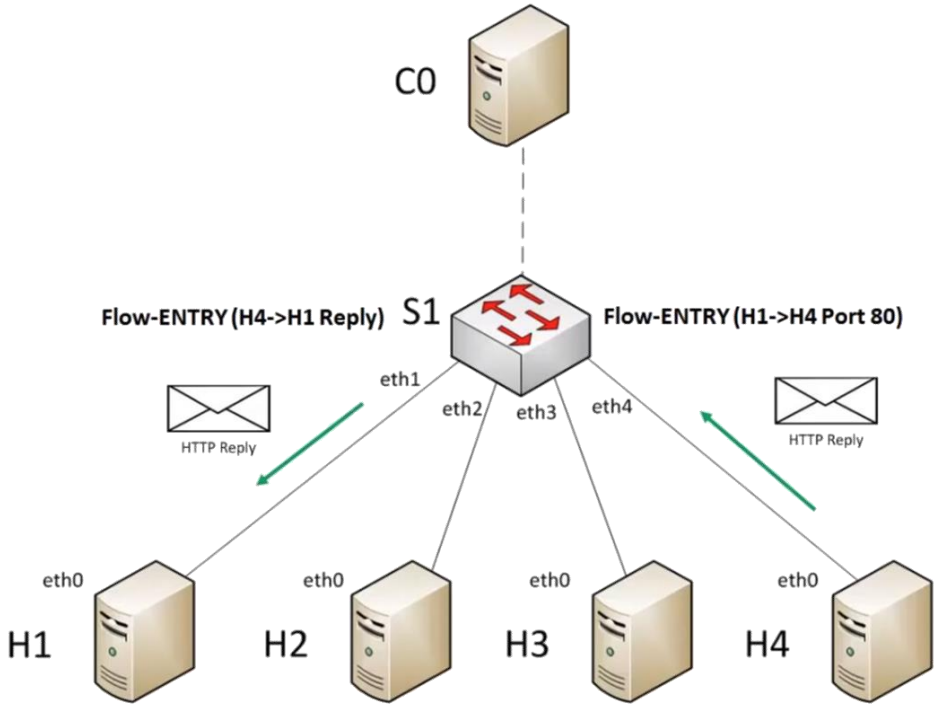
OVS Flow Forwarding Example

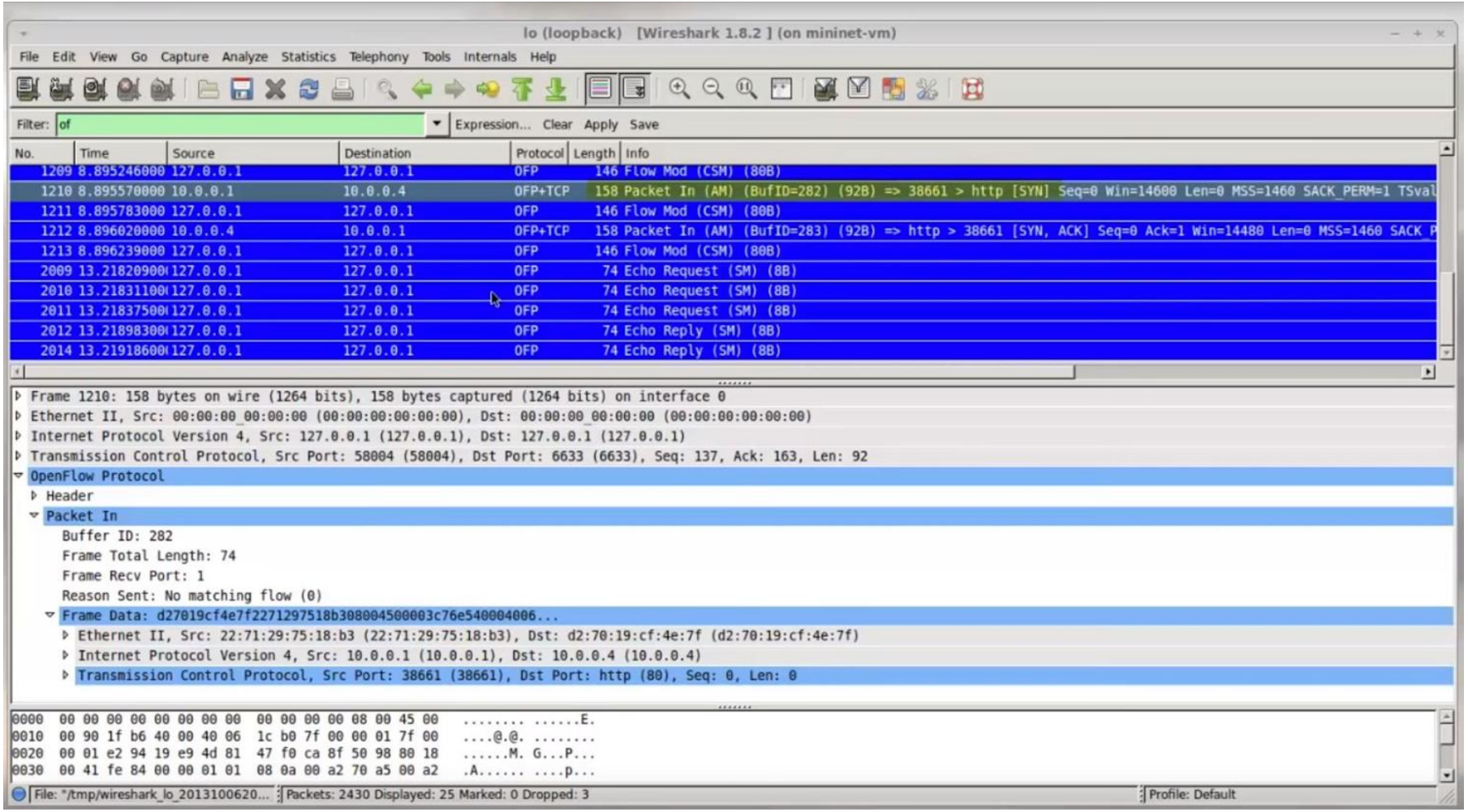


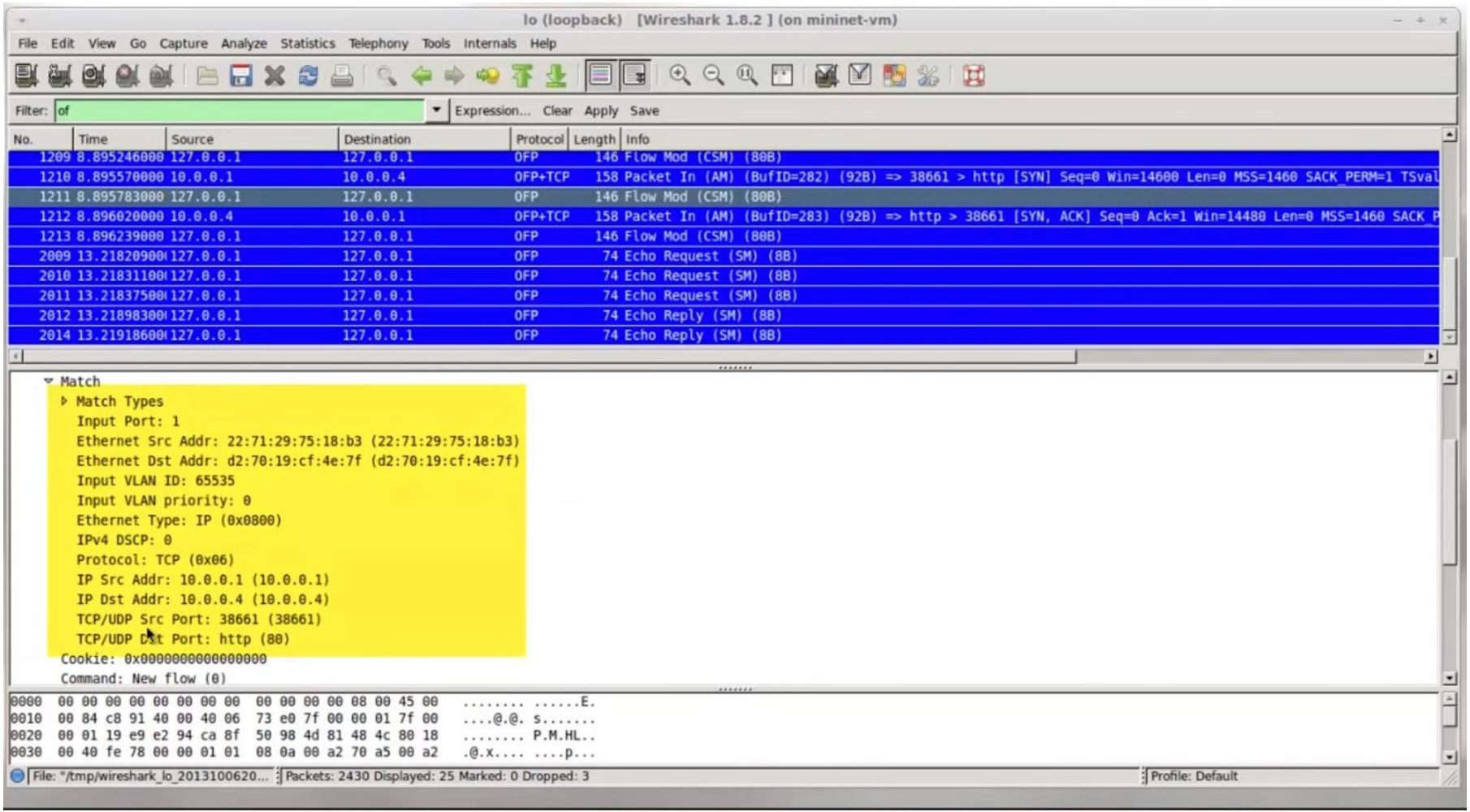
OVS Flow Forwarding Example



OVS Flow Forwarding Example




lo (loopback) [Wireshark 1.8.2] (on mininet-vm)
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: of
Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
1209 8.895246000 127.0.0.1 127.0.0.1 OFP 146 Flow Mod (CSM) (80B)
1210 8.895570000 10.0.0.1 10.0.0.4 OFP+TCP 158 Packet In (AM) (BufID=282) (92B) => 38661 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval
1211 8.895783000 127.0.0.1 127.0.0.1 OFP 146 Flow Mod (CSM) (80B)
1212 8.896020000 10.0.0.4 10.0.0.1 OFP+TCP 158 Packet In (AM) (BufID=283) (92B) => http > 38661 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK P
1213 8.896239000 127.0.0.1 127.0.0.1 OFP 146 Flow Mod (CSM) (80B)
2009 13.21820900 127.0.0.1 127.0.0.1 OFP 74 Echo Request (SM) (8B)
2010 13.21831100 127.0.0.1 127.0.0.1 OFP 74 Echo Request (SM) (8B)
2011 13.21837500 127.0.0.1 127.0.0.1 OFP 74 Echo Request (SM) (8B)
2012 13.21898300 127.0.0.1 127.0.0.1 OFP 74 Echo Reply (SM) (8B)
2014 13.21918600 127.0.0.1 127.0.0.1 OFP 74 Echo Reply (SM) (8B)
Frame 1210: 158 bytes on wire (1264 bits), 158 bytes captured (1264 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 58004 (58004), Dst Port: 6633 (6633), Seq: 137, Ack: 163, Len: 92
OpenFlow Protocol
Header
Packet In
Buffer ID: 282
Frame Total Length: 74
Frame Recv Port: 1
Reason Sent: No matching flow (0)
Frame Data: d27019cf4e7f2271297518b308004500003c76e540004006...
Ethernet II, Src: 22:71:29:75:18:b3 (22:71:29:75:18:b3), Dst: d2:70:19:cf:4e:7f (d2:70:19:cf:4e:7f)
Internet Protocol Version 4, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.4 (10.0.0.4)
Transmission Control Protocol, Src Port: 38661 (38661), Dst Port: http (80), Seq: 0, Len: 0
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 ....E.
0010 00 90 1f b6 40 00 40 06 1c b0 7f 00 00 01 7f 00 ....@.@. .....
0020 00 01 e2 94 19 e9 4d 81 47 f0 ca 8f 50 98 80 18 .....M. G...P...
0030 00 41 fe 84 00 00 01 01 08 0a 00 a2 70 a5 00 a2 .A..... ....p...
File: "/tmp/wireshark_lo_2013100620... Packets: 2430 Displayed: 25 Marked: 0 Dropped: 3
Profile: Default


lo (loopback) [Wireshark 1.8.2] (on mininet-vm)
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: of
Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
1209 8.895246000 127.0.0.1 127.0.0.1 OFP 146 Flow Mod (CSM) (80B)
1210 8.895570000 10.0.0.1 10.0.0.4 OFP+TCP 158 Packet In (AM) (BufID=282) (92B) => 38661 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval
1211 8.895783000 127.0.0.1 127.0.0.1 OFP 146 Flow Mod (CSM) (80B)
1212 8.896020000 10.0.0.4 10.0.0.1 OFP+TCP 158 Packet In (AM) (BufID=283) (92B) => http > 38661 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_P
1213 8.896239000 127.0.0.1 127.0.0.1 OFP 146 Flow Mod (CSM) (80B)
2009 13.21820900 127.0.0.1 127.0.0.1 OFP 74 Echo Request (SM) (8B)
2010 13.21831100 127.0.0.1 127.0.0.1 OFP 74 Echo Request (SM) (8B)
2011 13.21837500 127.0.0.1 127.0.0.1 OFP 74 Echo Request (SM) (8B)
2012 13.21898300 127.0.0.1 127.0.0.1 OFP 74 Echo Reply (SM) (8B)
2014 13.21918600 127.0.0.1 127.0.0.1 OFP 74 Echo Reply (SM) (8B)
Match
Match Types
Input Port: 1
Ethernet Src Addr: 22:71:29:75:18:b3 (22:71:29:75:18:b3)
Ethernet Dst Addr: d2:70:19:cf:4e:7f (d2:70:19:cf:4e:7f)
Input VLAN ID: 65535
Input VLAN priority: 0
Ethernet Type: IP (0x0800)
IPv4 DSCP: 0
Protocol: TCP (0x06)
IP Src Addr: 10.0.0.1 (10.0.0.1)
IP Dst Addr: 10.0.0.4 (10.0.0.4)
TCP/UDP Src Port: 38661 (38661)
TCP/UDP Dst Port: http (80)
Cookie: 0x0000000000000000
Command: New flow (0)
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 ....E.
0010 00 84 c8 91 40 00 40 06 73 e0 7f 00 00 01 7f 00 ....@.@. s.....
0020 00 01 19 e9 e2 94 ca 8f 50 98 4d 81 48 4c 80 18 ..... P.M.HL..
0030 00 40 fe 78 00 00 01 01 08 0a 00 a2 70 a5 00 a2 .@.x.... ....p...
File: "/tmp/wireshark_lo_2013100620... Packets: 2430 Displayed: 25 Marked: 0 Dropped: 3
Profile: Default

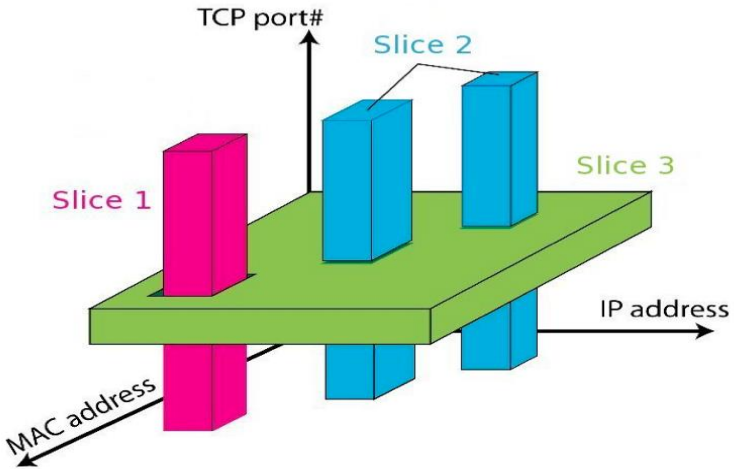
OF-enabled HW/SW Devices

Group	Product	Type	Maker/Developer	Version	Short description
Hardware	8200zl and 5400zl [125]	chassis	Hewlett-Packard	v1.0	Data center class chassis (switch modules).
	Arista 7150 Series [126]	switch	Arista Networks	v1.0	Data centers hybrid Ethernet/OpenFlow switches.
	BlackDiamond X8 [127]	switch	Extreme Networks	v1.0	Cloud-scale hybrid Ethernet/OpenFlow switches.
	CX600 Series [128]	router	Huawei	v1.0	Carrier class MAN routers.
	EX9200 Ethernet [129]	chassis	Juniper	v1.0	Chassis based switches for cloud data centers.
	EZchip NP-4 [130]	chip	EZchip Technologies	v1.1	High performance 100-Gigabit network processors.
	MLX Series [131]	router	Brocade	v1.0	Service providers and enterprise class routers.
	NoviSwitch 1248 [124]	switch	NoviFlow	v1.3	High performance OpenFlow switch.
	NetFPGA [48]	card	NetFPGA	v1.0	1G and 10G OpenFlow implementations.
	RackSwitch G8264 [132]	switch	IBM	v1.0	Data center switches supporting Virtual Fabric and OpenFlow.
	PF5240 and PF5820 [133]	switch	NEC	v1.0	Enterprise class hybrid Ethernet/OpenFlow switches.
	Pica8 3920 [134]	switch	Pica8	v1.0	Hybrid Ethernet/OpenFlow switches.
	Plexxi Switch 1 [135]	switch	Plexxi	v1.0	Optical multiplexing interconnect for data centers.
	V330 Series [136]	switch	Centec Networks	v1.0	Hybrid Ethernet/OpenFlow switches.
	Z-Series [137]	switch	Cyan	v1.0	Family of packet-optical transport platforms.
Software	contrail-vrouter [138]	vrouter	Juniper Networks	v1.0	Data-plane function to interface with a VRF.
	LINC [139], [140]	switch	FlowForwarding	v1.4	Erlang-based soft switch with OF-Config 1.1 support.
	ofsoftswitch13 [141]	switch	Ericsson, CPqD	v1.3	OF 1.3 compatible user-space software switch implementation.
	Open vSwitch [142], [109]	switch	Open Community	v1.0-1.3	Switch platform designed for virtualized server environments.
	OpenFlow Reference [143]	switch	Stanford	v1.0	OF Switching capability to a Linux PC with multiple NICs.
	OpenFlowClick [144]	vrouter	Yogesh Mundada	v1.0	OpenFlow switching element for Click software routers.
	Switch Light [145]	switch	Big Switch	v1.0	Thin switching software platform for physical/virtual switches.
	Pantou/OpenWRT [146]	switch	Stanford	v1.0	Turns a wireless router into an OF-enabled switch.
	XorPlus [46]	switch	Pica8	v1.0	Switching software for high performance ASICs.

Network Slicing

Possible only in OpenFlow networks:

- Defined with notion of **flowspace** (the set of all possible header values defined by the OpenFlow tuple)
- The **slice** (virtual network) is any subset of OpenFlow flowspace:
  - To a slice belongs all frames with specific values of **header fields**
  - Network segmentation on any network protocol or combination of network protocols (we can **emulate VLAN, MPLS, IP segmentation** and any other technique)
  - OpenFlow controller can set **flow entries** within a slice
- Very **flexible** approach for network sharing

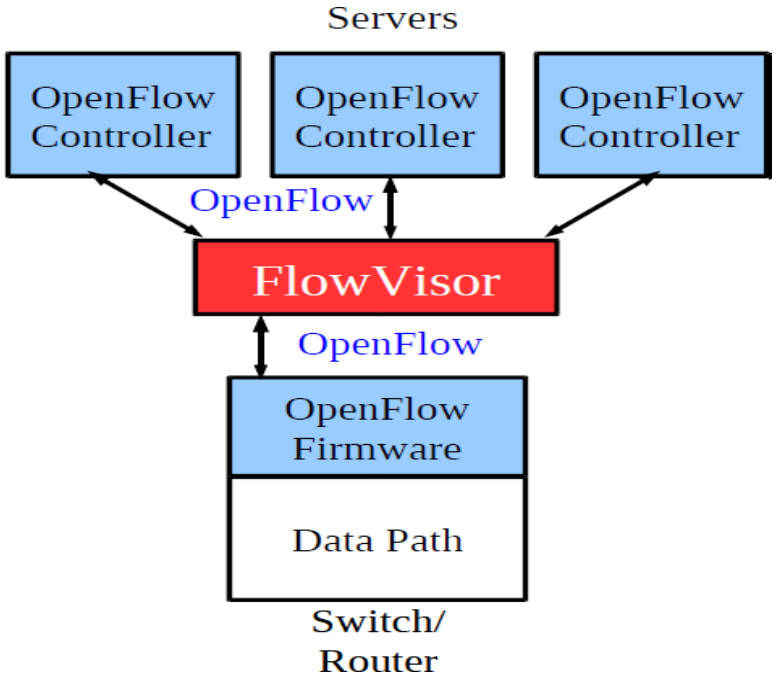


Network Hypervisors

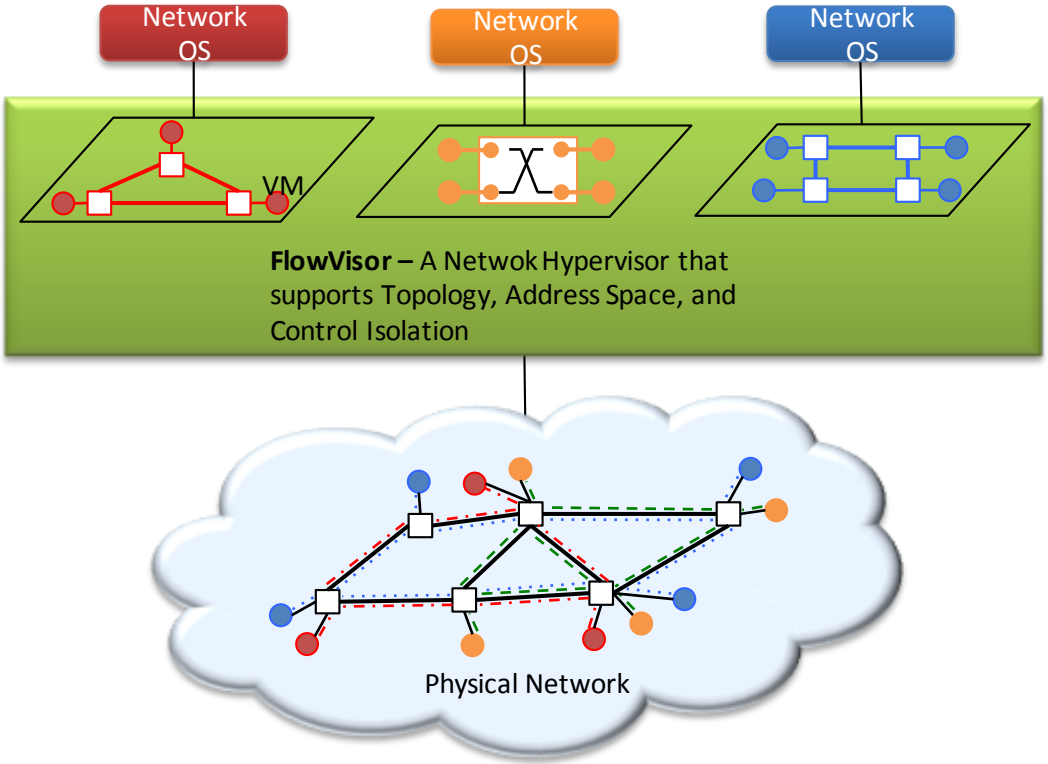
- The basic idea is to allow multiple logical networks share the same **OpenFlow networking infrastructure**.
- The main goal is to provide virtual SDNs through both **topology, address, and control function** virtualization.
- FlowVisor is one of the early technologies to virtualize a SDN providing an abstraction layer that makes it easier to slice a data plane based on OpenFlow-enabled switches, allowing multiple and diverse networks to co-exist.
- In general terms, a slice is defined as a particular set of flows on the data plane.

FlowVisor

- From a system design perspective, FlowVisor is *a* **transparent proxy** that intercepts OpenFlow messages between switches and controllers.
- It partitions the link bandwidth and flow tables of each switch and each guest controller gets its own **virtual flow table** in the switches, i.e. multiple controllers can co-exist on top of the same physical network infrastructure.



Topology Abstraction



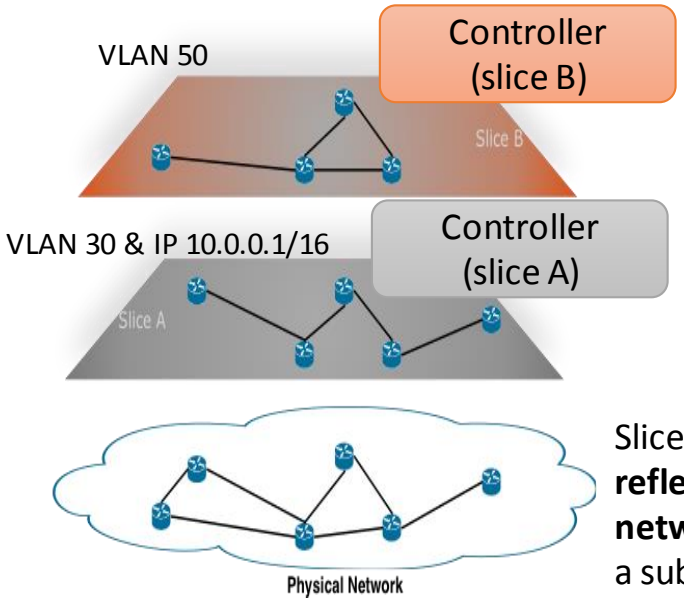
Topology abstraction:

- Virtual network topology can be different than physical topology
  - Controller can see **simplified topology**
  - Collapse **multi-hop path** into one-hop link
- **Hosts** (endpoints) could be part of virtual network or not

FlowVisor-enabled Network Slicing

- Divide the physical network into logical slices
  - Each slice/service controls its own packet
  - Forwarding Give different slices to different application or owners
  - Enforce strong isolation between slices
- A network slice is a collection of sliced switches/routers
- Each slice believes it owns the data path
- Slicing Policy: specifies resource limits for each slice
  - Link Bandwidth
  - Topology
  - Traffic Type
  - Device CPU Usage (not directly exposed by OpenFlow, some workaround needed)

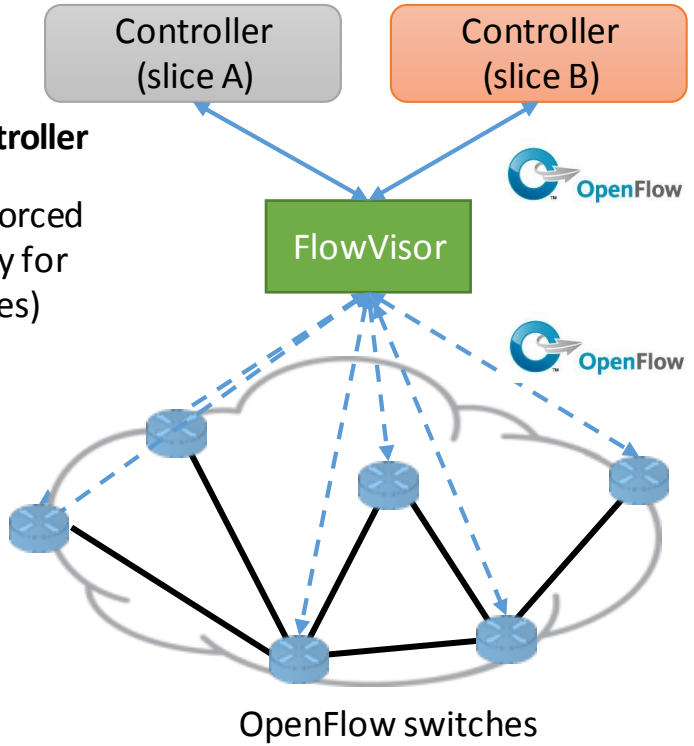
Control Isolation



Each slice **associated to a controller**

Isolation of slices enforced by FlowVisor (a proxy for OpenFlow messages)

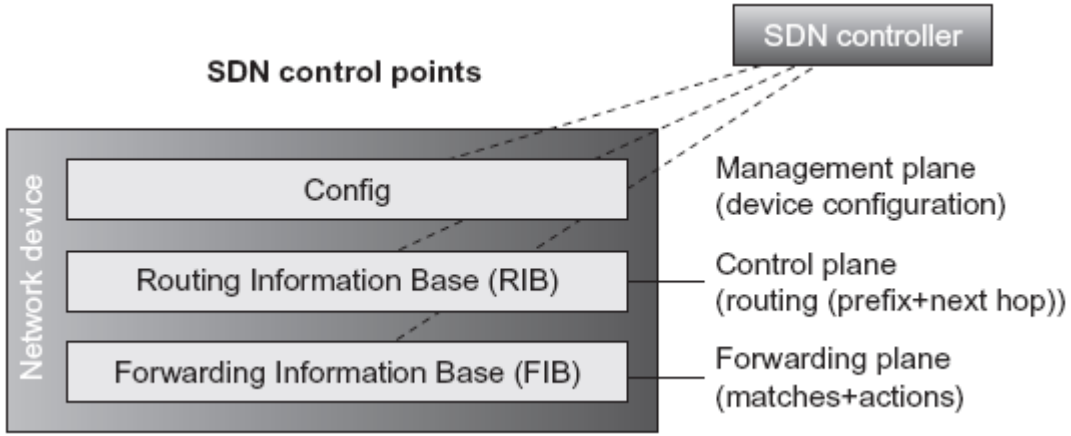
Slice topology directly **reflects the physical network topology** and is a subset of it



SDN Control Points

The control points that can be managed and configured by the SDN application:

- *Config*, where general configuration is done,
- *Routing Information Base* (RIB), where routes(e.g., prefixes and next-hops) are set,
- *Forwarding Information Base* (FIB), which is lower level and can be considered *flows*, where packet headers are matched and actions are taken.



Configuration Management Protocols

- The configuration management protocols such as SNMP lack support for features required for SDN deployments.
- Transactions across multiple network elements, the ability to back up and restore configuration, error recovery, and operational ease are vital for large scale SDN deployments.
- SDN implementations define sets of protocols that support switching configuration management, among others, **NETCONF**, **OF-CONF** and **OVSDB.**
- Protocols are designed for extensibility, interoperability, and reuse of configuration components to fit into SDN deployment requirements.

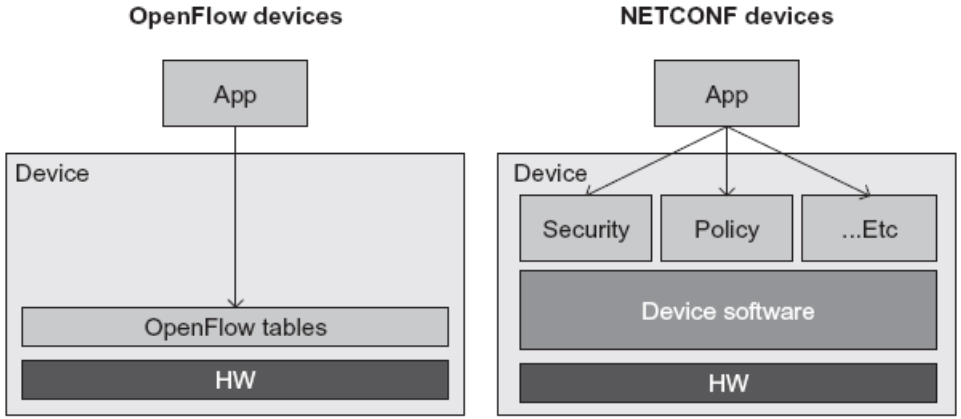
NETCONF

- NETCONF was developed as a successor to the *Simple Network Management Protocol* (SNMP) and attempted to address some of SNMP's shortcomings, e.g. SNMP was mainly being used to monitor networks, not to configure them
- NETCONF is a protocol that was designed with programmability in mind.
- *The purpose of NETCONF is to give applications a simple, standards-based, and robust API (application programming interface) to apply and read configurations.*
- It is a management protocol and as such it has the ability to configure only those capabilities which are exposed by the device.

NETCONF vs. OpenFlow

- OpenFlow configures the lower levels of the networking device, i.e. setting the matches and actions of the FIB.

- NETCONF, on the other hand, performs traditional configuration of the device but via the NETCONF protocol rather than via the CLI or SNMP.



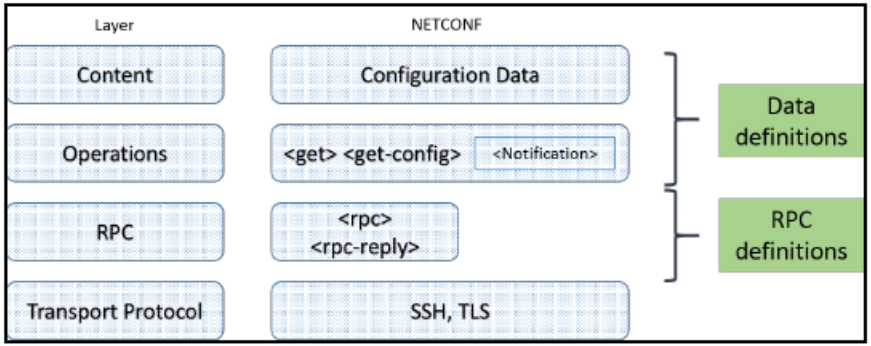
NETCONF Data Model

- NETCONF follows a standard client-server model that uses RPCs (remote procedure calls).
- With NETCONF, server configurations are stored in a NETCONF configuration data store that follows a YANG defined-data format.
- To query or change data, a client sends an XML-based remote procedure call over one of the supported secure transfer methods, and the server replies with XML-encoded data.

```
<?xmlversion="1.0" encoding="UTF-8"?>
<rpc message-id="1"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <target>
      <candidate/>
    </target>
    <default-operation>merge</default-operation>
    <test-option>set</test-option>
    <config>
      <capable-switch
        xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
        nc:operation="create"
        xmlns="urn:onf:of12:config:yang">
        <id>capable-switch-0</id>
        <logical-switches>
          <switch>
            <id>logic-switch-1</id>
            <datapath-id>11:11:11:11:11:11:11:11</datapath-id>
            <enabled>true</enabled>
            <controllers>
              <controller>
                <id>controller-0</id>
                <role>master</role>
                <ip-address>192.168.2.1</ip-address>
                <port>6633</port>
                <protocol>tcp</protocol>
              </controller>
            </controllers>
          </switch>
        </logical-switches>
      </capable-switch>
    </config>
  </edit-config>
</rpc>
```

NETCONF Protocol Layers

- **Content layer**: Device configuration and notification data store.
- **Operations layer**: Set of operations to manipulate configuration data in the data store.
- **RPC Messages layer**: Supports remote procedure calls (RPCs) and notifications.
- **Secure Transport layer**: TCP-based secure transport protocol for reliable transport of messages. Specifies Secure Shell (SSH) as the mandatory option and TLS as an alternative option.



NETCONF Key Attributes

- **Separation of configuration and state (operational) data.** Configuration data is set on the device to cause it to operate in a particular way. State (operational) data is set by the device as a result of dynamic changes on the device due to network events and activities.
- **Support for *Remote Procedure Call* (RPC)-like functionality.** Such functionality was not available in SNMP. With NETCONF, it is possible to invoke an operation on a device, passing parameters and receiving returned results, much like RPC calls in the programming paradigm.
- **Support for Notifications.** This capability is a general event mechanism, whereby the managed device can notify the management station of significant events. Within SNMP this concept is called a trap.
- **Support for transaction-based configurations.** This allows for the configuration of multiple devices to be initiated, but then rolled back in case of a failure at some point in the process.

NETCONF for SDN

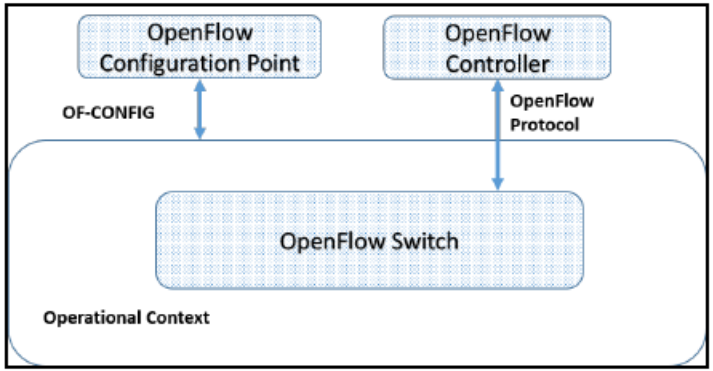
- One of the first operations that takes place between a NETCONF client on the controller and a NETCONF server running on the device is for the device to inform the client which data models are supported.
- This allows the SDN application running on the controller to know which operations are possible on each device.
- YANG provides a standardized way for devices to support and advertise their capabilities.
- The NETCONF programmer can learn which capabilities are exposed from YANG data models supported by the device.

NETCONF for SDN

- The SDN application developer is limited by what the device exposes as configurable, e.g.
  - If the device exposes NETCONF data models that allow for the configuration of *Access Control Lists* (ACLs), then the application can configure those.
  - Similarly, if the device has data models for configuring *Quality of Service* (QoS) or static routes, then those will be possible as well.
- This granularity is key, since different devices will often vary in their capabilities. One of the current drawbacks of NETCONF and YANG is that different vendors often support different YANG models.

OF-CONFIG

- Open Networking Foundation is defines the **OF-CONFIG** protocol to manage the configuration operation of an **OpenFlow Switch**.
- OF-CONFIG leverages the NETCONF protocol as the transport protocol.
- OF-CONFIG defines the XML schema for the content layer of the NETCONF protocol and companion YANG module for the **OF-CONFIG** data model.



OF-CONFIG Functionalities

- OF-CONFIG is focused on functions required to configure an OpenFlow **logical switch**.
- An OpenFlow Logical Switch is an instantiation of an **OpenFlow Switch** with a set of resources (example ports), which can be associated with an OpenFlow controller.
- It supports OpenFlow controller assignments to OpenFlow data planes.
- It also supports
  - Configuration of ports, queues, tunnel types such as IP-in-GRE, NV-GRE, VxLAN
  - Modification of operational status such as ports, capability discovery of the OpenFlow logical switch
  - Instantiation of OpenFlow Logical Switches (OpenFlow datapaths)
  - Configuration certification for secure communication

Example: VLANs in SDN

- Hybrid deployment
  - VLAN for SDN adopters
  - Remaining traffic using legacy protocols
- Switch-controller communication
  - Separate VLAN
  - Using legacy protocols
- Tagging of packets
  - VLAN header as a virtual “tag” on packets

OVSDB

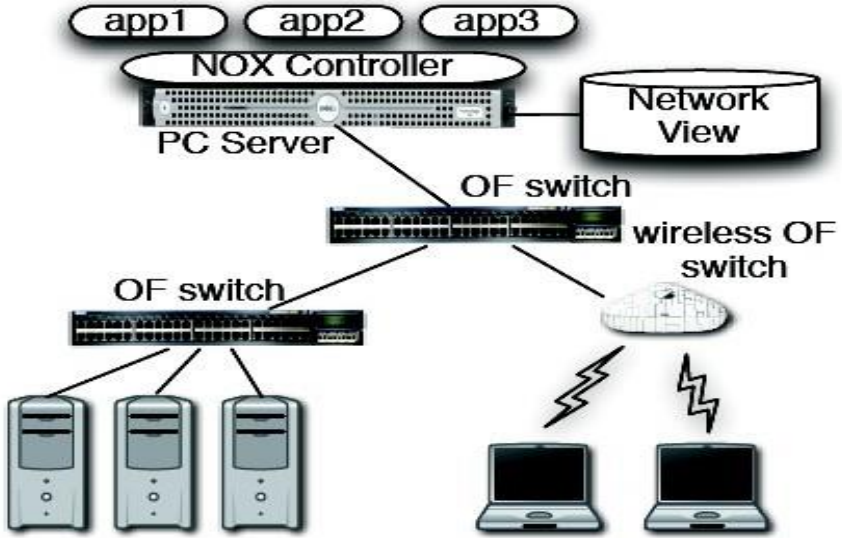
- OVSDB is another type of southbound API, designed to provide advanced management capabilities for Open vSwitches.
- Beyond OpenFlow's capabilities to configure the behavior of flows in a forwarding device, an Open vSwitch offers other networking functions.
- For instance, it allows the control elements to create multiple virtual switch instances, set quality of service (QoS) policies on interfaces, attach interfaces to the switches, configure tunnel interfaces on OpenFlow data paths, manage queues, and collect statistics.
- Therefore, the OVSDB is a complementary protocol to OpenFlow for Open vSwitch.

Network OS

- SDN is promised to facilitate network management and ease the burden of solving networking problems by means of the logically-centralized control offered by a network operating system (NOS).
- Generic functionality as network state and network topology information, device discovery, and distribution of network configuration can be provided as services of the NOS.
- A NOS (or controller) is a critical element in an SDN architecture as it is the key supporting piece for the control logic (applications) to generate the network configuration based on the policies defined by the network operator.

Network OS (NOS)

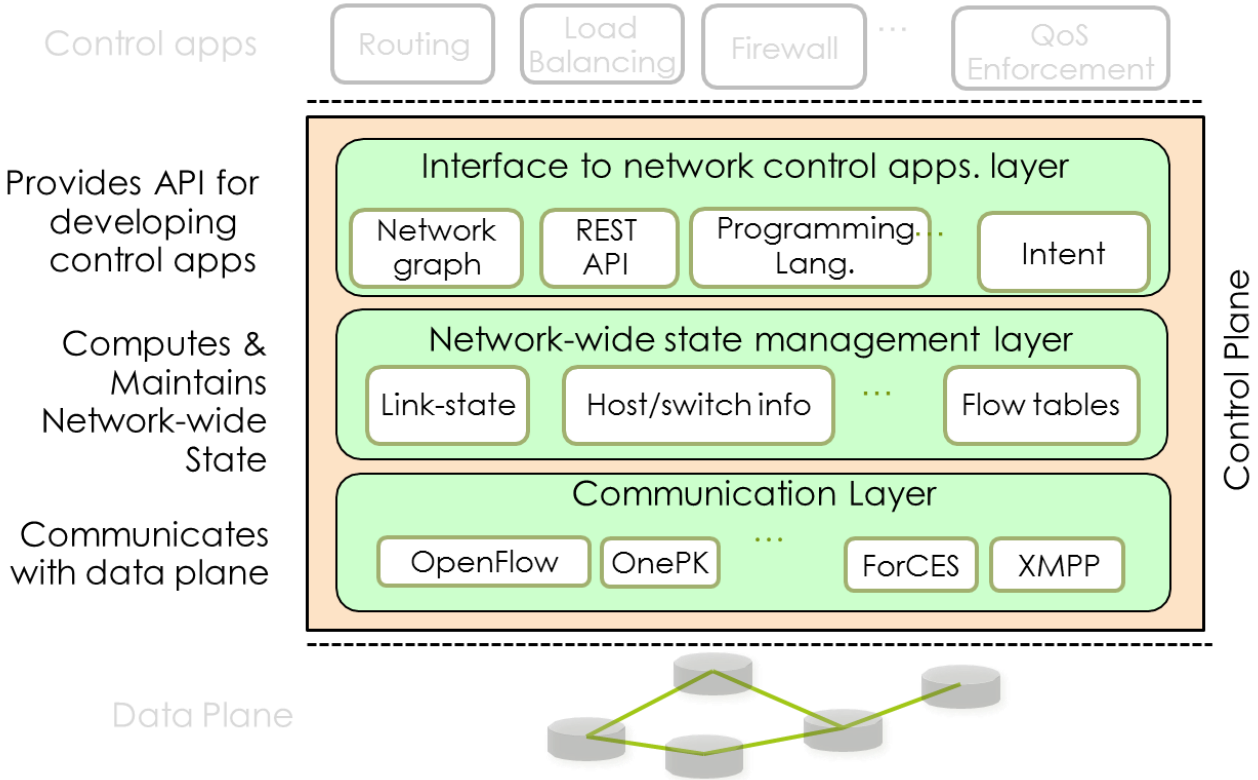
- Network OS: distributed system that creates a consistent, up-to-date network view
- Runs on servers (controllers) in the network
- Uses forwarding abstraction to **get the state to and give control from forwarding elements**
- Examples: NOX, ONIX, Floodlight, Trema, OpenDaylight, HyperFlow, Kandoo, Beehive, Beacon, Maestro, … + more



Control Plane Functions

- Topology Management
  - Runs topology discovery protocol to discover network devices and their interconnections
- Path computation
  - Uses topology information to compute shortest path and setup necessary forwarding rules for routing flows.
- Notification Management
  - Receives, processes and forwards network event information to applications
- Statistics Collection
  - Collects statistics from network devices by reading different counters
- Device Manager
  - Configures network devices

Components of SDN Controller



Communication layer

- Implements protocols (*e.g.*, OpenFlow, OnePK *etc.*) to transfer information between the controller and network devices
- Provides controller a uniform way to communicate with devices from different vendors
- Provides mechanisms for receiving asynchronous event notifications (*e.g.*, link up/down) from network devices.

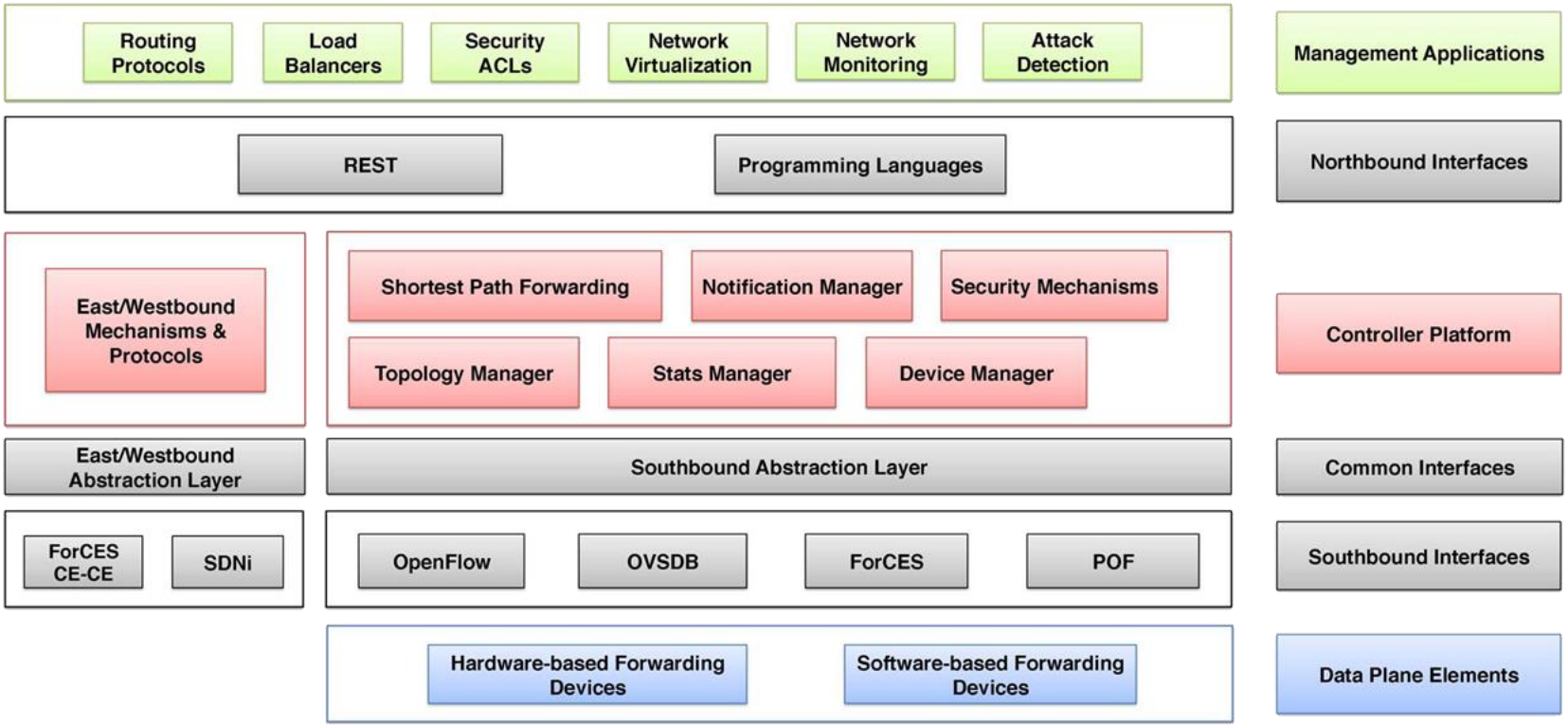
Network-wide state management

- During controller initialization executes appropriate protocol to obtain and build a global network view
- Updates global network view periodically or in response to network events (*e.g.*, new device added, link down, flow table miss *etc.*)
- In case of distributed controller implementation an east-west API is present to communicate with other controllers
- Network-wide state may include
  - Network topology
  - Health status of switches, links, hosts
  - Flow table content of each switch
  - Statistics counters from the switches
  - …

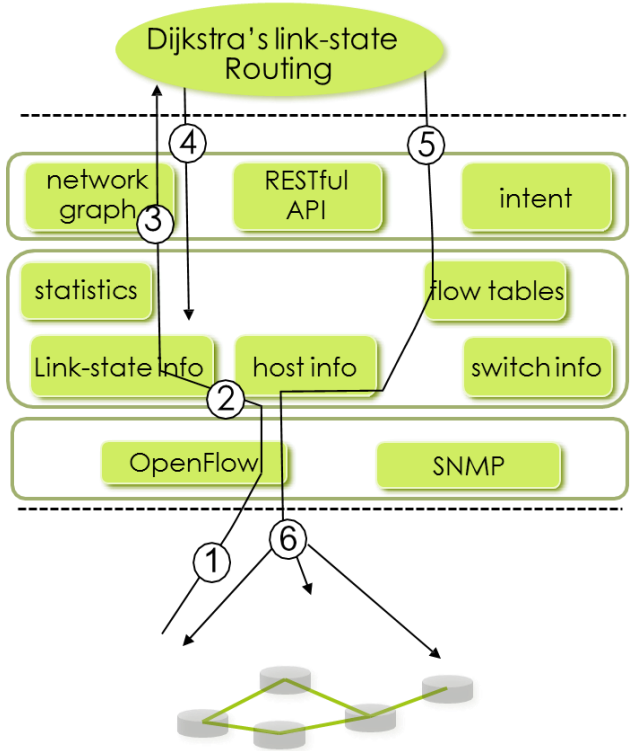
Interface to control apps layer

- Provides an abstract view of the network to control applications (*i.e.*, hides device level details)
- Allows control applications to read/modify network state (e.g., read link utilization, add flow rule etc.)
- No standards yet; Northbound API exists in different forms
  - RESTful API
  - Programming Language
  - Intent Language

SDN Framework Building Blocks



OpenFlow in Action: Flow Setup

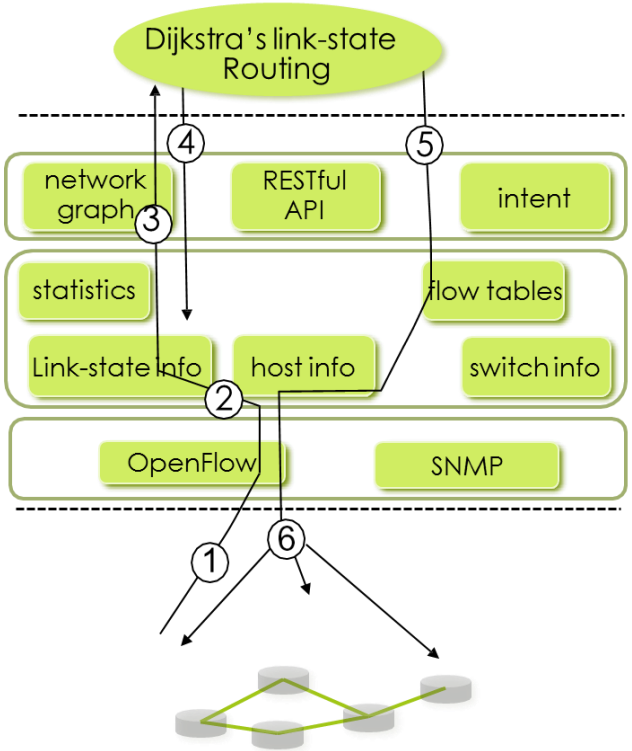


① S1, receives a packet and there is no forwarding rule matching the packet. S1 sends a PacketIn message to controller

② SDN controller receives PacketIn message, identifies source and destination

③ Dijkstra's routing algorithm application has previously registered to be called when PacketIn arrives. It is called.

OpenFlow in Action: Flow Setup

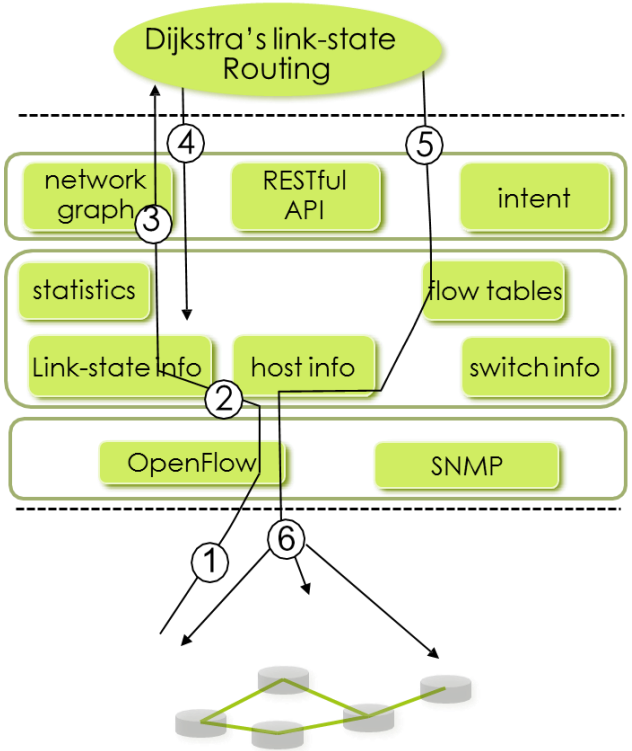


④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed

⑥ Controller uses FlowMod message to install new tables in switches that need updating

OpenFlow in Action: Failure Handling

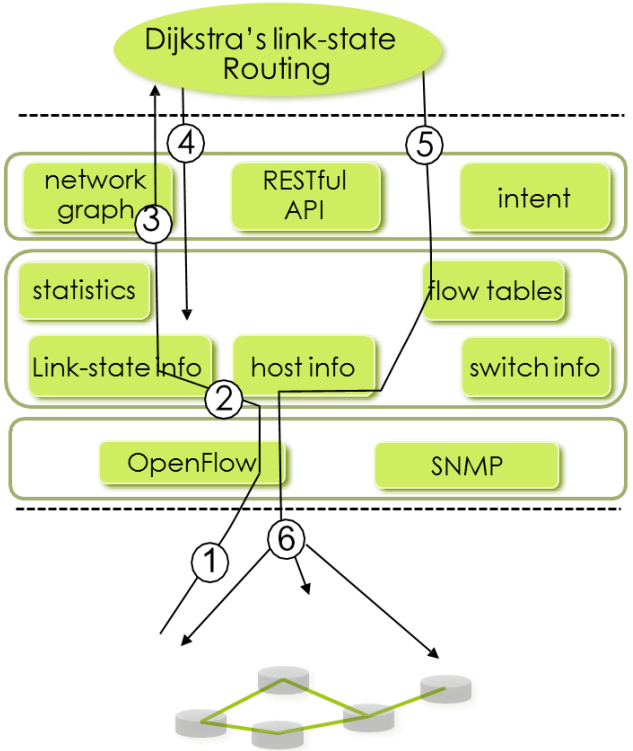


1. S1, experiencing link failure uses OpenFlow PortStatus message to notify controller

2. SDN controller receives OpenFlow message, updates link status info

3. Dijkstra's routing algorithm appl has previously registered to be called whenever a link status changes. It is called.

OpenFlow in Action: Failure Handling



4. Dijkstra's routing algorithm accesses network graph and link state info in controller, computes new routes

5. link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow rules as needed

6. Controller uses FlowMod message to install new rules in switch tables that need updating

Examples for SDN Controllers

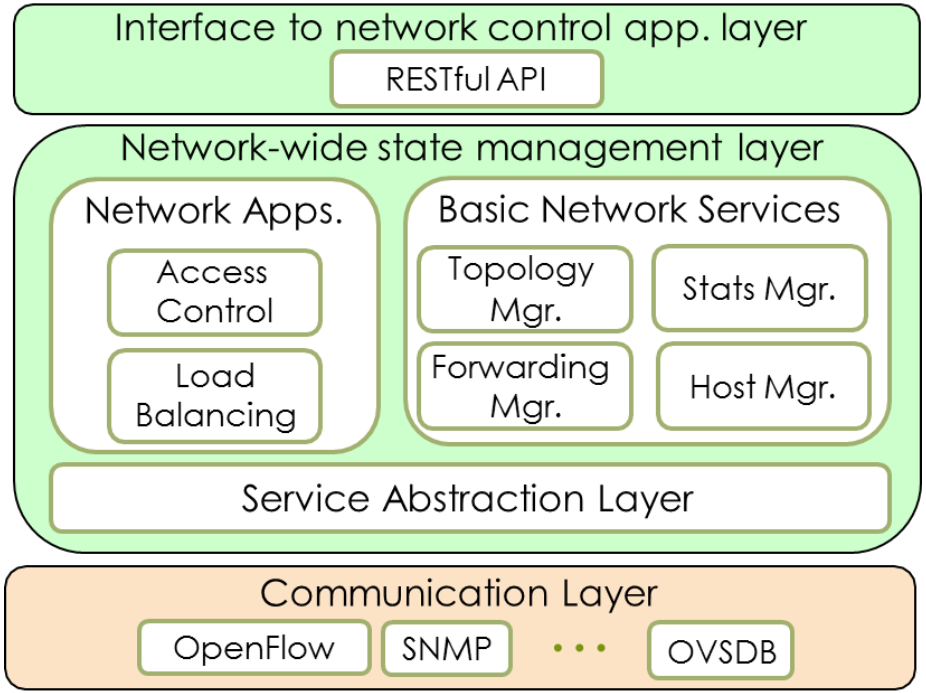
- **OpenDaylight**
  - OpenDaylight can provide centralized control over any vendor equipment. This is the subject of the next chapter.
- **OpenContrail SDN controller**
  - This controller is actually a stem from Juniper ’s commercial offering and can be used as a platform for network virtualization under open source licensing.
- **Floodlight**
  - This is a Java-based controller for SDN that supports a range of OpenFlow virtual or real switches. It is part of the Big Switch open source project and has been successfully used in a number of SDN projects
- **Ryu**
  - Ryu is an open source SDN controller framework that supports several protocols, including OpenFlow, Netconf, and OF-config.
- **Flowvisor**
  - This is a specialist OpenFlow controller that acts as a go-between for OpenFlow switches and multiple OpenFlow Controllers.

OpenDaylight Controller

- An open source Java based SDN controller
- Started as a Linux foundation project in 2013 and the first code release was in 2014
- Founding members included vendors and service providers such as Arista, Big Switch Networks, Cisco, Juniper, Ericsson, IBM, Microsoft, VMware, etc.
  - Currently operators such as AT&T, Comcast, and Telefonica are also contributors to the project
- Currently deployed by major telecom operators and also in enterprise networks

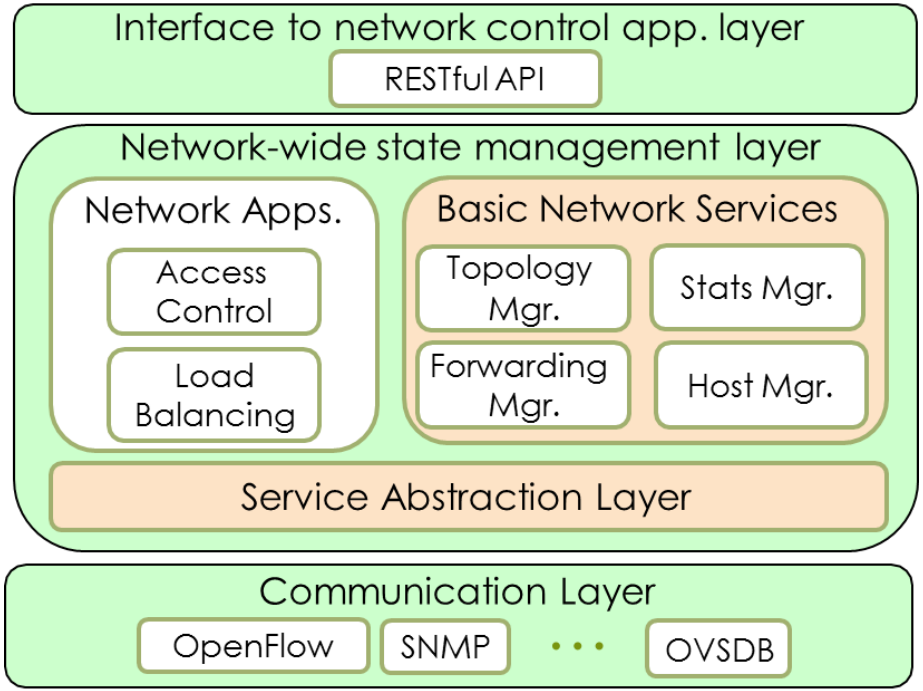
OpenDaylight Controller (cont)

- Designed as a single stand- alone controller
- Also supports cluster mode operation.
- Communication layer supports a wide range of south-bound APIs contributed by different vendors
  - OpenFlow
  - Cisco OnePK
  - XMPP
  - SNMP
  - OVSDB
  - …



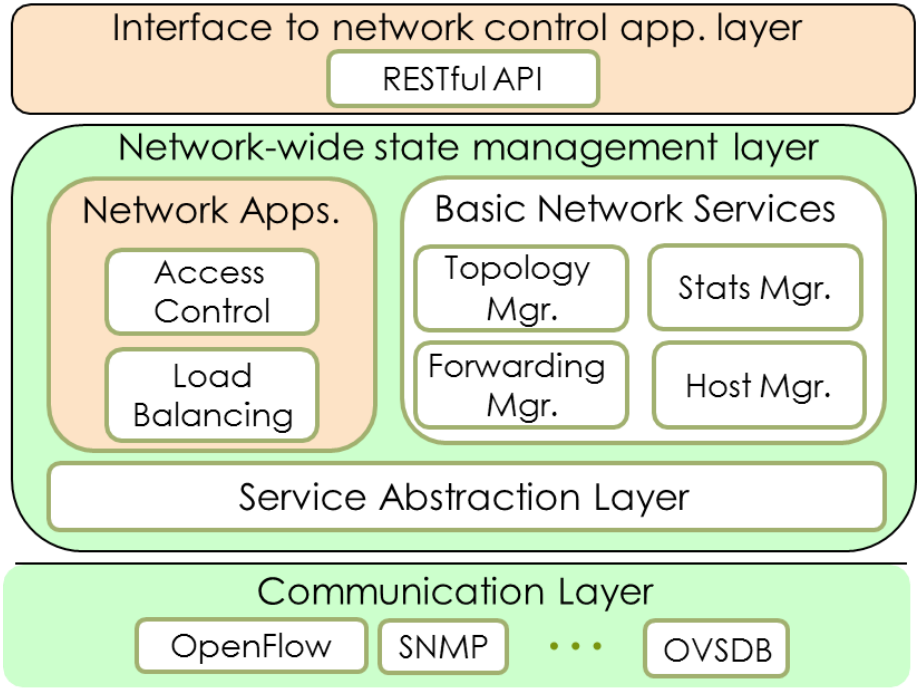
OpenDaylight Controller (cont)

- Service Abstraction Layer
  - provides an uniform way to communicate with the network
  - hides the heterogeneity in communication layer
- A set of basic network services comes shipped with the controller

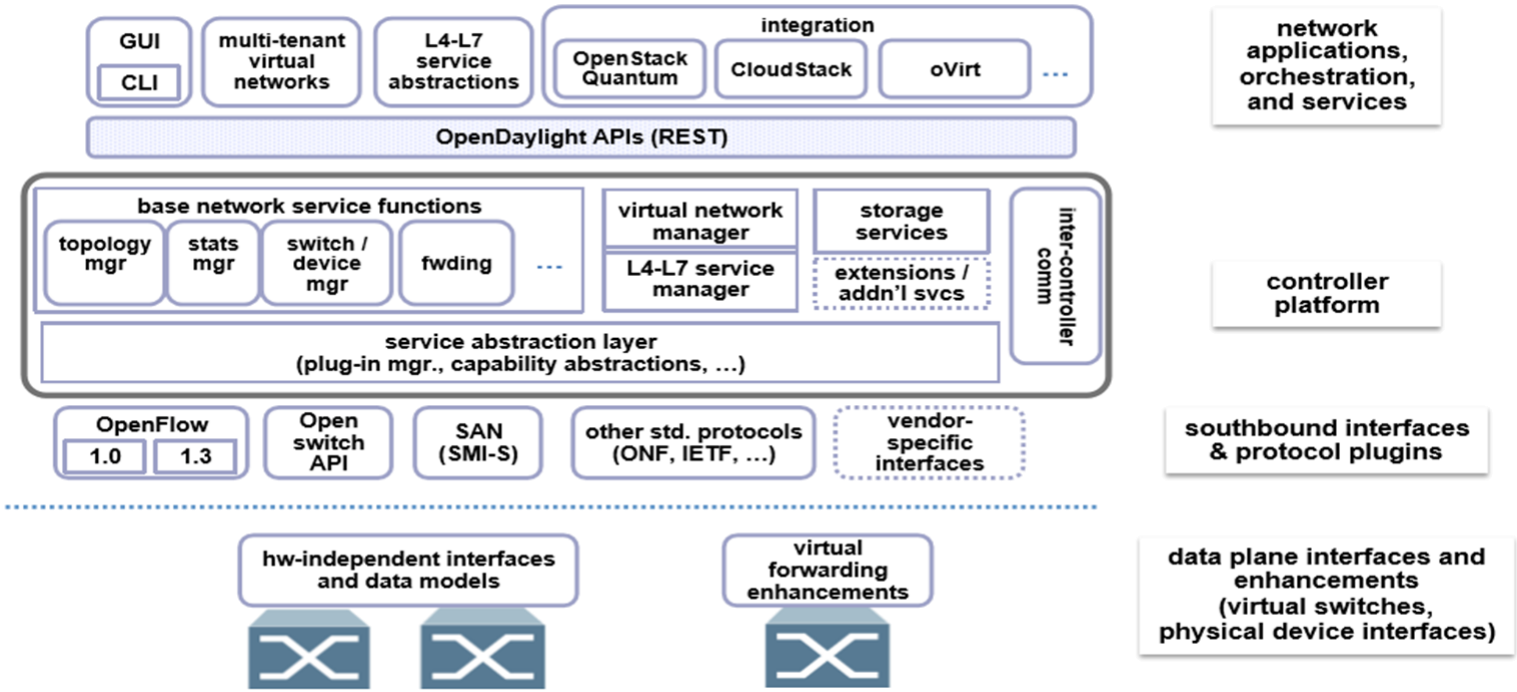


OpenDaylight Controller (cont)

- RESTful API exposed for control apps.
- Network apps may be integrated with, or be external to OpenDaylight
  - External apps use RESTful API
  - Integrated apps use APIs of Service Abstraction Layer and require recompiling the controller



OpenDaylight Platform



Northbound API

- A common northbound interface is still an open issue
- The northbound interface is mostly a *software* ecosystem, not a *hardware* one, as is the case of the southbound APIs.
- Existing controllers such as Floodlight, Trema, NOX, Onix, and OpenDaylight propose and define their own northbound APIs.
- Programming languages also abstract the inner details of the controller functions and data plane behaviour from the application developers.

RESTFULL API

A new technology being successfully used as an SDN API is REST, which uses HTTP or HTTPS. APIs based on this technology are called *RESTful* interfaces. REST technology was introduced a number of years ago and has been used primarily for access to information through a web service. The movement toward REST as one of the means for manipulating SDN networks is based on a number of its perceived advantages:

• **Simplicity**. The web-based REST mechanism utilizes the simple HTTP GET, PUT, POST, and DELETE commands. Access to data involves referencing *resources* on the target device using normal and well-understood URL encoding.

• **Flexibility**. Requesting entities can access the defined configuration components on a device using REST resources which are represented as separate URLs. There is no need for complicated schemas or MIB definitions to make the data accessible.

• **Extensibility**. Support for new resources does not require recompilation of schemas or MIBs, but only requires that the calling application access the appropriate URL.

• **Security**. It is very straightforward to secure REST communications. Simply by running this web-based protocol through HTTPS adequately addresses security concerns. This has the advantage of easily penetrating firewalls, a characteristic not shared by all network security approaches.

RESTCONF

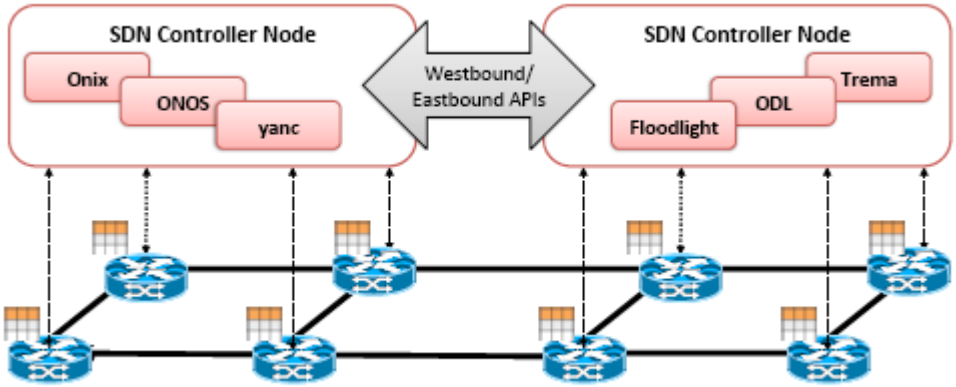
- RESTCONF is a protocol wherein NETCONF requests are bundled in easier-to-use REST messages.
- A related use of REST is in the RESTCONF protocol. While more often used as a northbound API
- to the controller, RESTCONF can also be used as a device API. RESTCONF is an implementation of
- NETCONF/YANG using JSON or XML over HTTP rather than XML RPCs over SSH.
- In the context of SDN via Controller APIs, the application typically makes RESTCONF requests to the controller.
- The controller in turn translates the RESTCONF request into the NETCONF protocol, with the contents of the request remaining fundamentally the same.
- Such
- northbound APIs are used at different levels of the controller and also may exist as high as the
- management layer that oversees a cluster of controllers

Controllers Classification

Name	Architecture	Northbound API	Consistency	Faults	License	Prog. language	Version
Beacon [186]	centralized multi-threaded	ad-hoc API	no	no	GPLv2	Java	v1.0
DISCO [185]	distributed	REST	—	yes	—	Java	v1.1
ElastiCon [228]	distributed	RESTful API	yes	no	—	Java	v1.0
Fleet [199]	distributed	ad-hoc	no	no	—	—	v1.0
Floodlight [189]	centralized multi-threaded	RESTful API	no	no	Apache	Java	v1.1
HP VAN SDN [184]	distributed	RESTful API	weak	yes	—	Java	v1.0
HyperFlow [195]	distributed	—	weak	yes	—	C++	v1.0
Kandoo [229]	hierarchically distributed	—	no	no	—	C, C++, Python	v1.0
Onix [7]	distributed	NVP NBAPI	weak, strong	yes	commercial	Python, C	v1.0
Maestro [188]	centralized multi-threaded	ad-hoc API	no	no	LGPLv2.1	Java	v1.0
Meridian [192]	centralized multi-threaded	extensible API layer	no	no	—	Java	v1.0
MobileFlow [222]	—	SDMN API	—	—	—	—	v1.2
MuL [230]	centralized multi-threaded	multi-level interface	no	no	GPLv2	C	v1.0
NOX [26]	centralized	ad-hoc API	no	no	GPLv3	C++	v1.0
NOX-MT [187]	centralized multi-threaded	ad-hoc API	no	no	GPLv3	C++	v1.0
NVP Controller [112]	distributed	—	—	—	commercial	—	—
OpenContrail [183]	—	REST API	no	no	Apache 2.0	Python, C++, Java	v1.0
OpenDaylight [13]	distributed	REST, RESTCONF	weak	no	EPL v1.0	Java	v1.{0,3}
ONOS [117]	distributed	RESTful API	weak, strong	yes	—	Java	v1.0
PANE [197]	distributed	PANE API	yes	—	—	—	—
POX [231]	centralized	ad-hoc API	no	no	GPLv3	Python	v1.0
ProgrammableFlow [232]	centralized	—	—	—	—	C	v1.3
Rosemary [194]	centralized	ad-hoc	—	—	—	—	v1.0
Ryu NOS [191]	centralized multi-threaded	ad-hoc API	no	no	Apache 2.0	Python	v1.{0,2,3}
SMaRtLight [198]	distributed	RESTful API	yes	yes	—	Java	v1.0
SNAC [233]	centralized	ad-hoc API	no	no	GPL	C++	v1.0
Trema [190]	centralized multi-threaded	ad-hoc API	no	no	GPLv2	C, Ruby	v1.0
Unified Controller [171]	—	REST API	—	—	commercial	—	v1.0
yanc [196]	distributed	file system	—	—	—	—	—

Distributed Controllers

- East/Westbound API: the functions of these interfaces *required by distributed controllers* include import/export data between controllers, algorithms for data consistency models, and monitoring/notification capabilities.
- Multi-domain SDN Controllers
- SDN-to-SDN Communication Protocols



SDN Controllers API Specifications

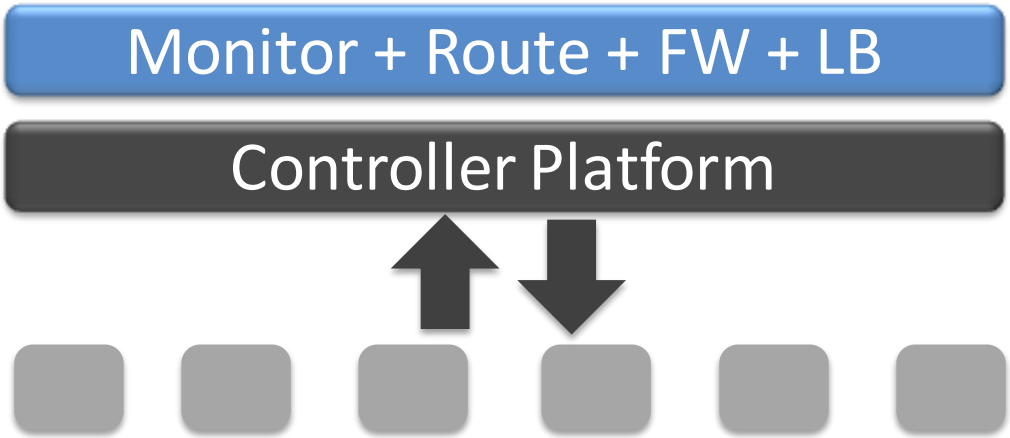
Component	OpenDaylight	OpenContrail	HP VAN SDN	Onix	Beacon
Base network services	Topology/Stats/Switch Manager, Host Tracker, Shortest Path Forwarding	Routing, Tenant Isolation	Audit Log, Alerts, Topology, Discovery	Discovery, Multi-consistency Storage, Read State, Register for updates	Topology, device manager, and routing
East/Westbound APIs	—	Control Node (XMPP-like control channel)	Sync API	Distribution I/O module	*Not present*
Integration Plug-ins	OpenStack Neutron	CloudStack, OpenStack	OpenStack	—	—
Management Interfaces	GUI/CLI, REST API	GUI/CLI	REST API Shell / GUI Shell	—	Web
Northbound APIs	REST, REST-CONF , Java APIs	REST APIs (configu-ration, operational, and analytic)	REST API, GUI Shell	Onix API (general purpose)	API (based on OpenFlow events)
Service abstraction layers	Service Abstraction Layer (SAL)	—	Device Abstraction API	Network Information Base (NIB) Graph with Import/Export Functions	—
Southbound APIs or connectors	OpenFlow, OVSDB, SNMP, PCEP, BGP, NETCONF	—	OpenFlow, L3 Agent, L2 Agent	OpenFlow, OVSDB	OpenFlow

Control Program

- Control program operates on view of network
  - **Input**: global network view (graph/database)
  - **Output**: configuration of each network device
- Control program is not a distributed system
  - Abstraction hides details of distributed state

Combining Many Networking Tasks

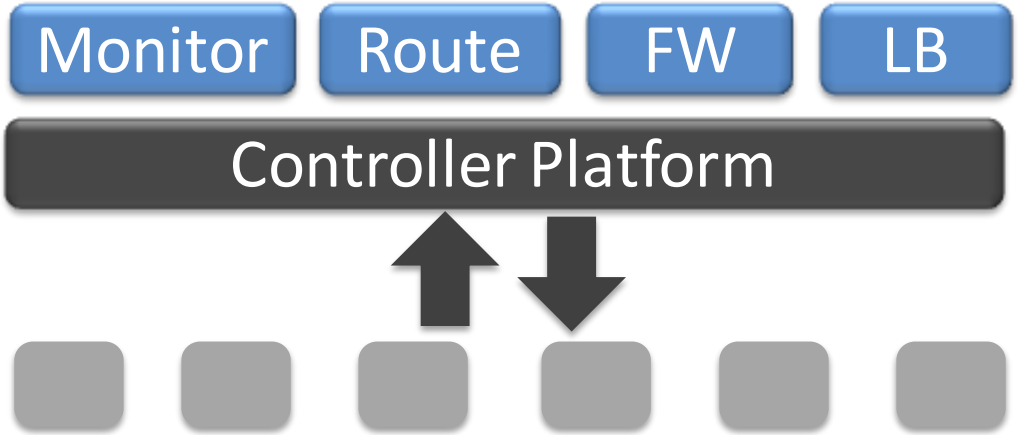
Monolithic application



Hard to program, test, debug, reuse, port, …

Modular Controller Applications

A module for each task

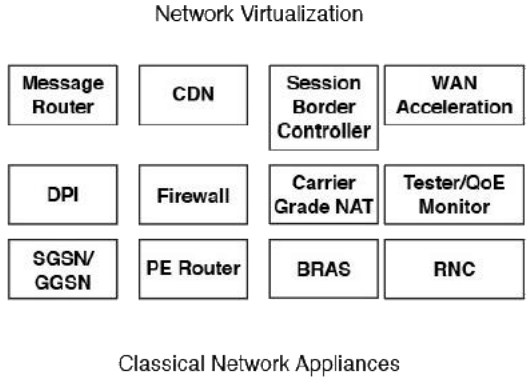


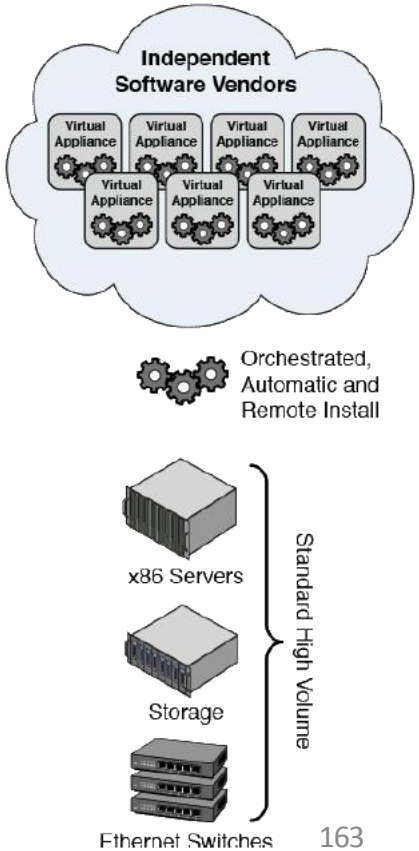
Easier to program, test, and debug

Greater reusability and portability

Network Function Virtualization

- Replacing the need for physical appliances with virtualized network functions running on standard server platforms
- Moving the functions usually embedded in network hardware into software that can run in a VM on the standard servers.
  - Functions: Routers, DPI, EPC, Firewalls.





SDN and NFV

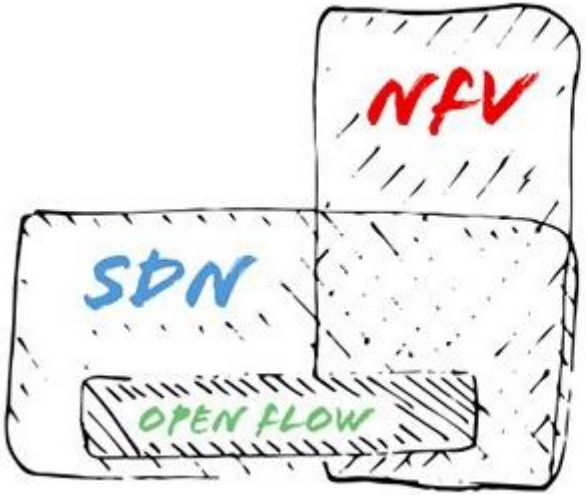
Software Defined Networking

- DP/CP split
- DP/CP interface: e.g. Open Flow, Forces, …
- Gives
  - Centralized algorithms
  - Easy availability of control data for all functions
  - Better control over the network resources

Network Function Virtualization

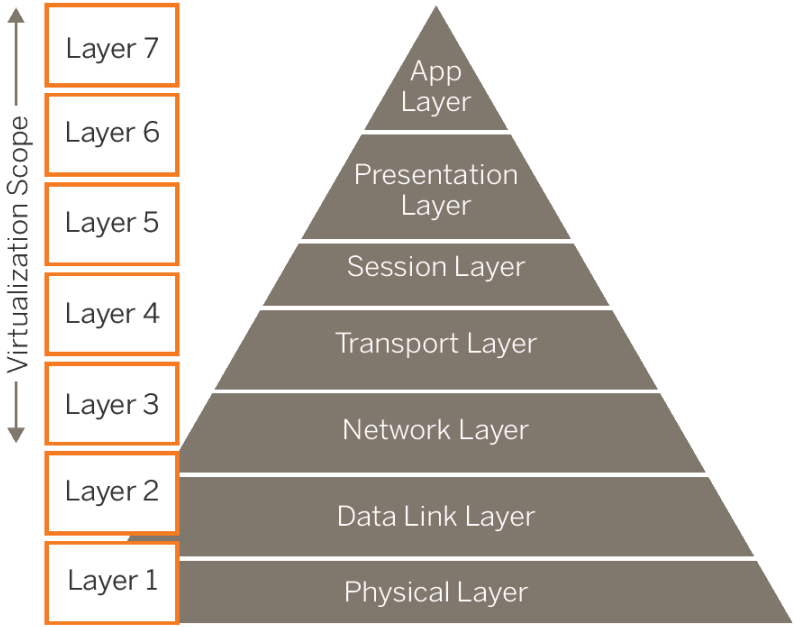
- Implementing network control functions as
  - Virtual machines (e.g. over KVM)
  - In containers of VMs
  - Better scalability of control functions to CPU and memory capacity
  - Flexibility of placement
- Business impact!

SDN and NFV (cont)

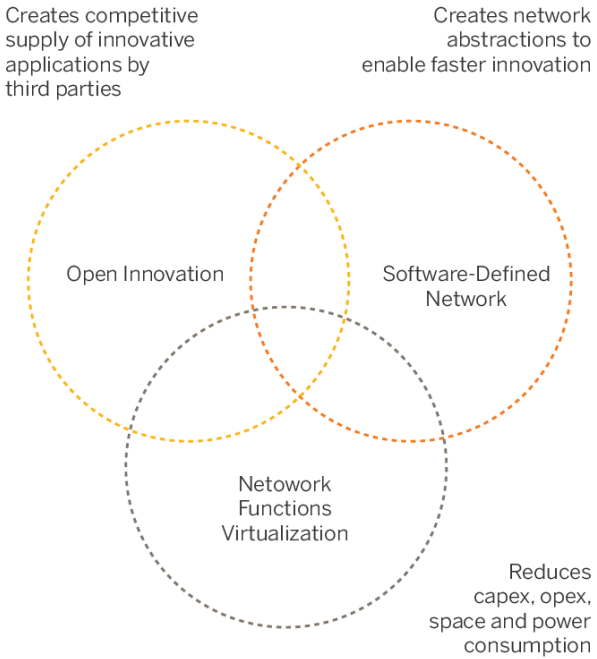


SDN AND NFV ARE RELATED BUT NOT THE SAME

SDN and NFV (cont)



Scope of Virtualization



Venn diagram – interaction of SDN, NFV, Open Innovation