

EEE 4121F-B

Software-Defined Networking

SDN Overview

<https://youtu.be/aa6OrUjdrTA?t=35>

Recap: Why is configuration hard?

1. Difficult to get things configured correctly
2. Too many differences results in unpredictability
 - a. different protocols
 - b. different network manufacturers
 - c. different network operators and policies
3. Operators make mistakes all the time

Recap: Why network operators need SDN

1. Network-wide views
 - a. topology
 - b. traffic
2. Network-network level objectives
 - a. load balance
 - b. security
3. Direct access and control
 - a. configure and control a network from a logically central location

Recap: Routers should

- ✓ Forward Packets
- ✓ Collect measurements
- ✗ Compute routes

Software-Defined
networking ==
“removing routing
from the routers”

Pre SDN

- Closed Systems with no or very minimal abstractions in the network design.
- Hardware centric – usage of custom ASICs with Vendor Specific Software.
- Difficult to perform real world experiments on large scale production networks.
- No standard abstractions towards northbound and southbound interfaces, even though we have standard abstractions in the east/west bound interface with peer routers/switches.

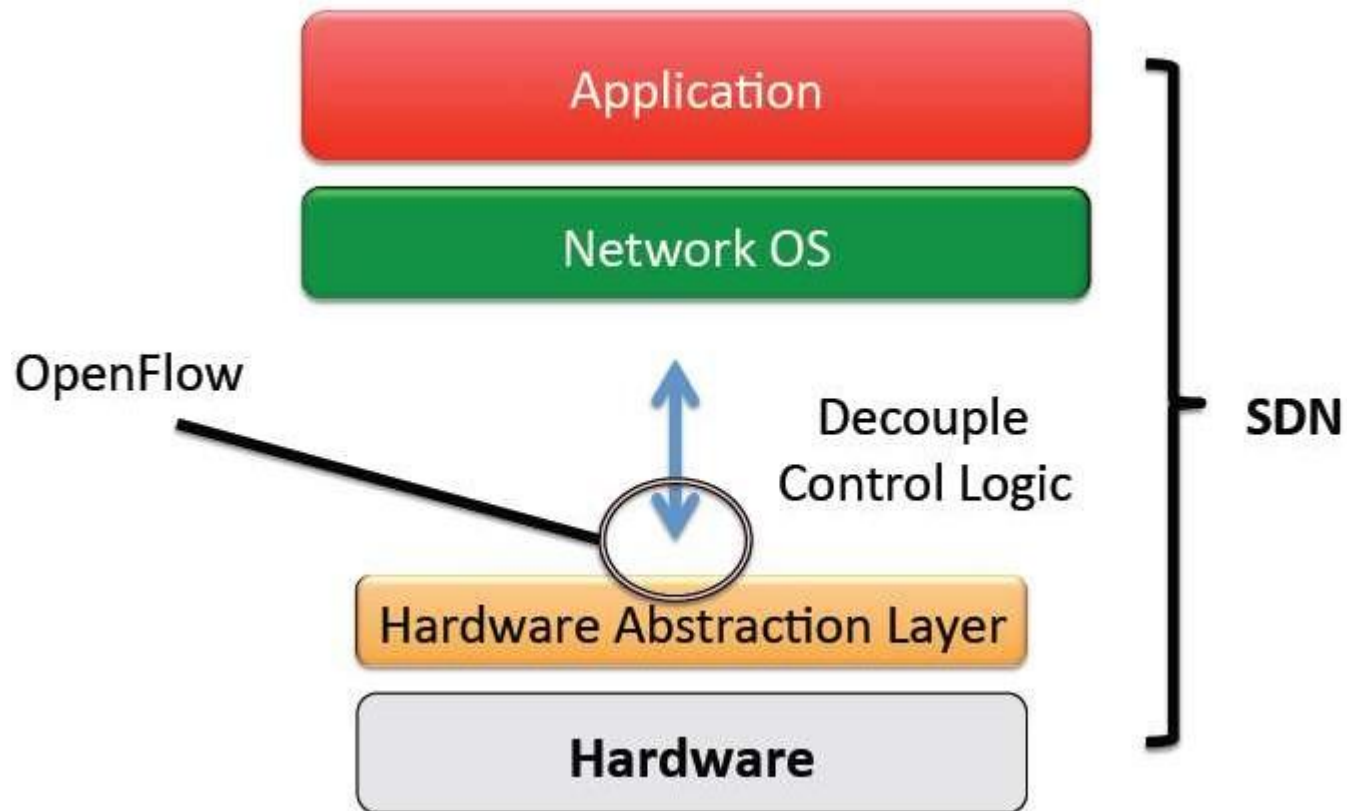
SDN Overview

<https://youtu.be/87pmirhxEcE>

OpenFlow

- OpenFlow is a protocol which enables programmability of the forwarding plane
 - OpenFlow is like **an x86 instruction set** for the network nodes.
- Provides open interface to “black box” networking node (ie. Routers, L2/L3 switch) to enable visibility and openness in network
- 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

SDN and OpenFlow



OpenFlow

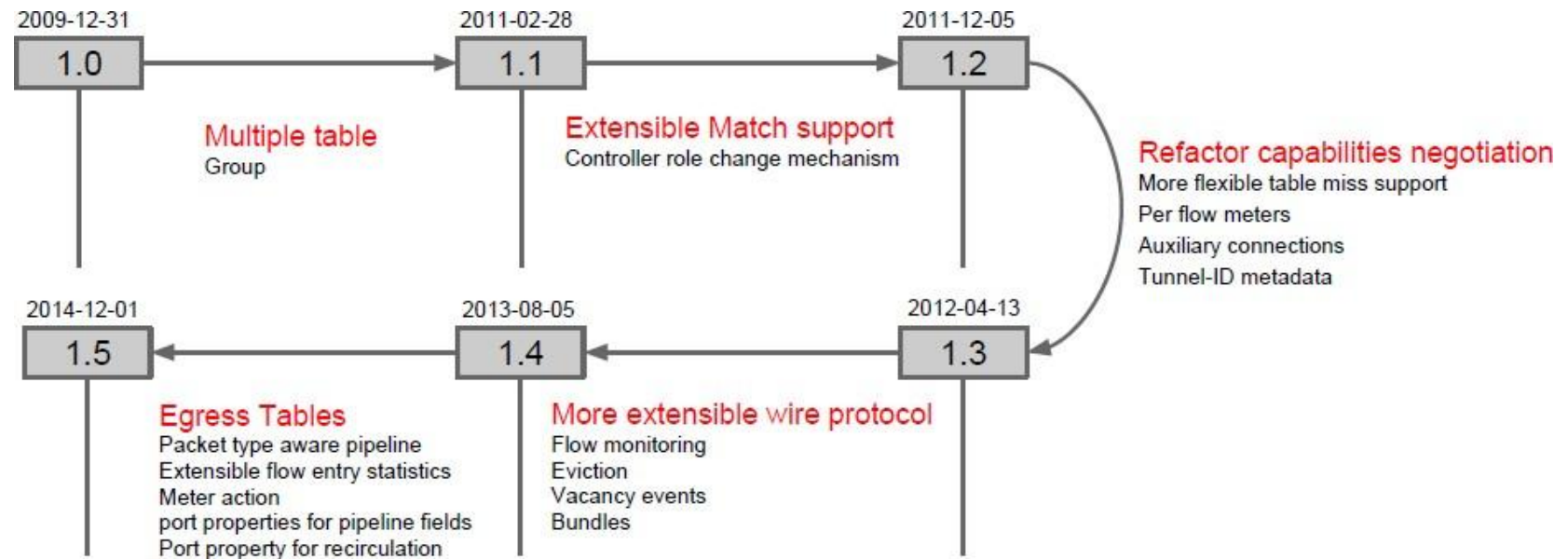
General Myth

SDN is OpenFlow

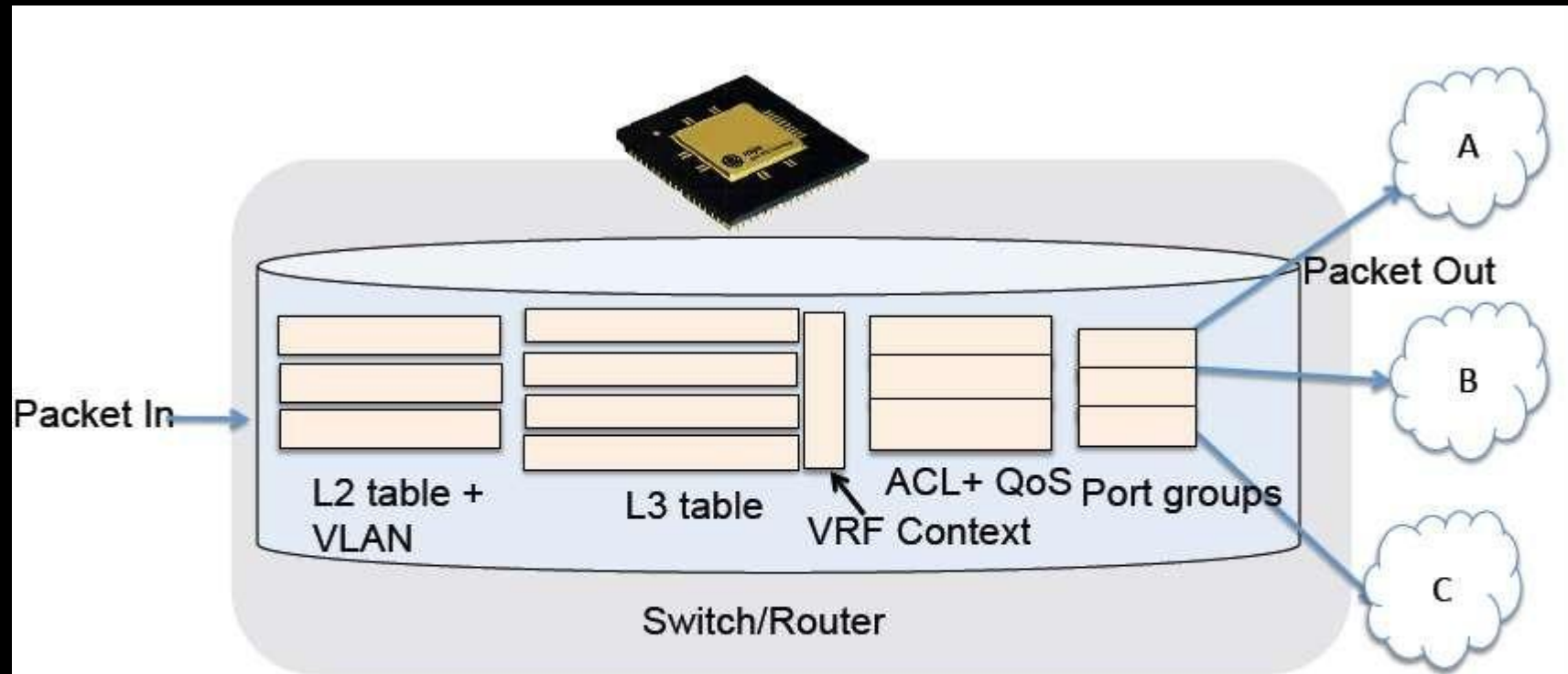
Reality

OpenFlow is an open API that provides a standard interface for programming the data plane switches (i.e OpenFlow is one way to achieve an SDN, but there are many others)

OpenFlow Specification History

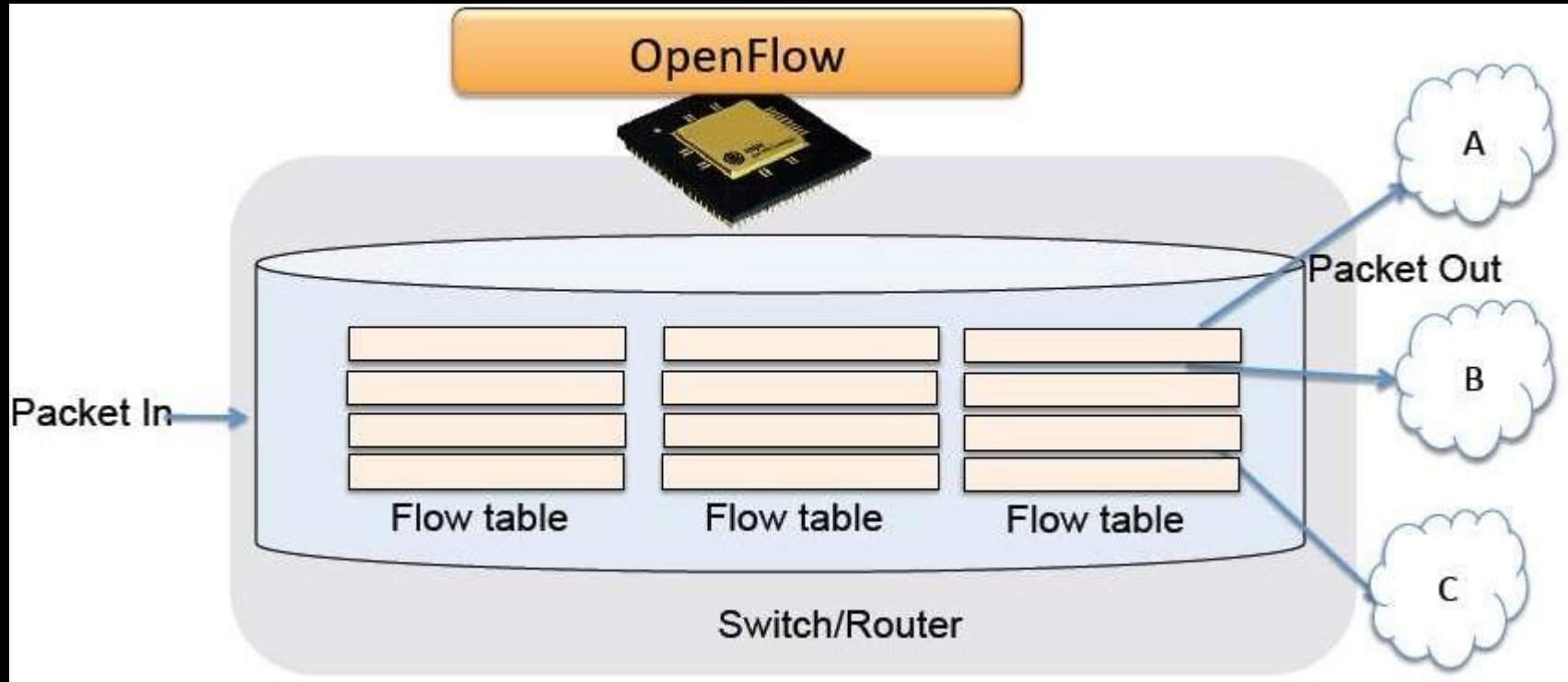


Traditional Switch Forwarding

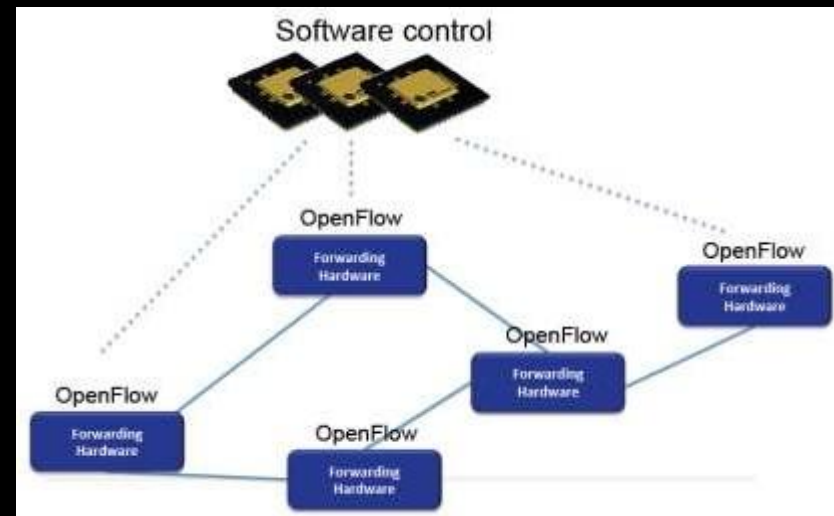
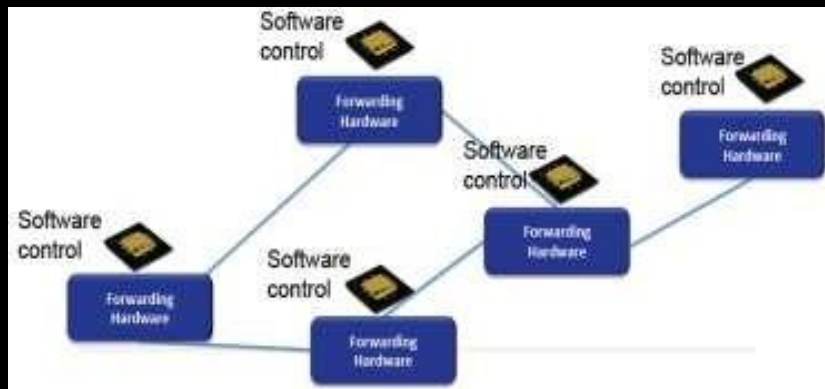


- Fixed function
- Often expose implementation details
- Non-standard/non-existent state management APIs

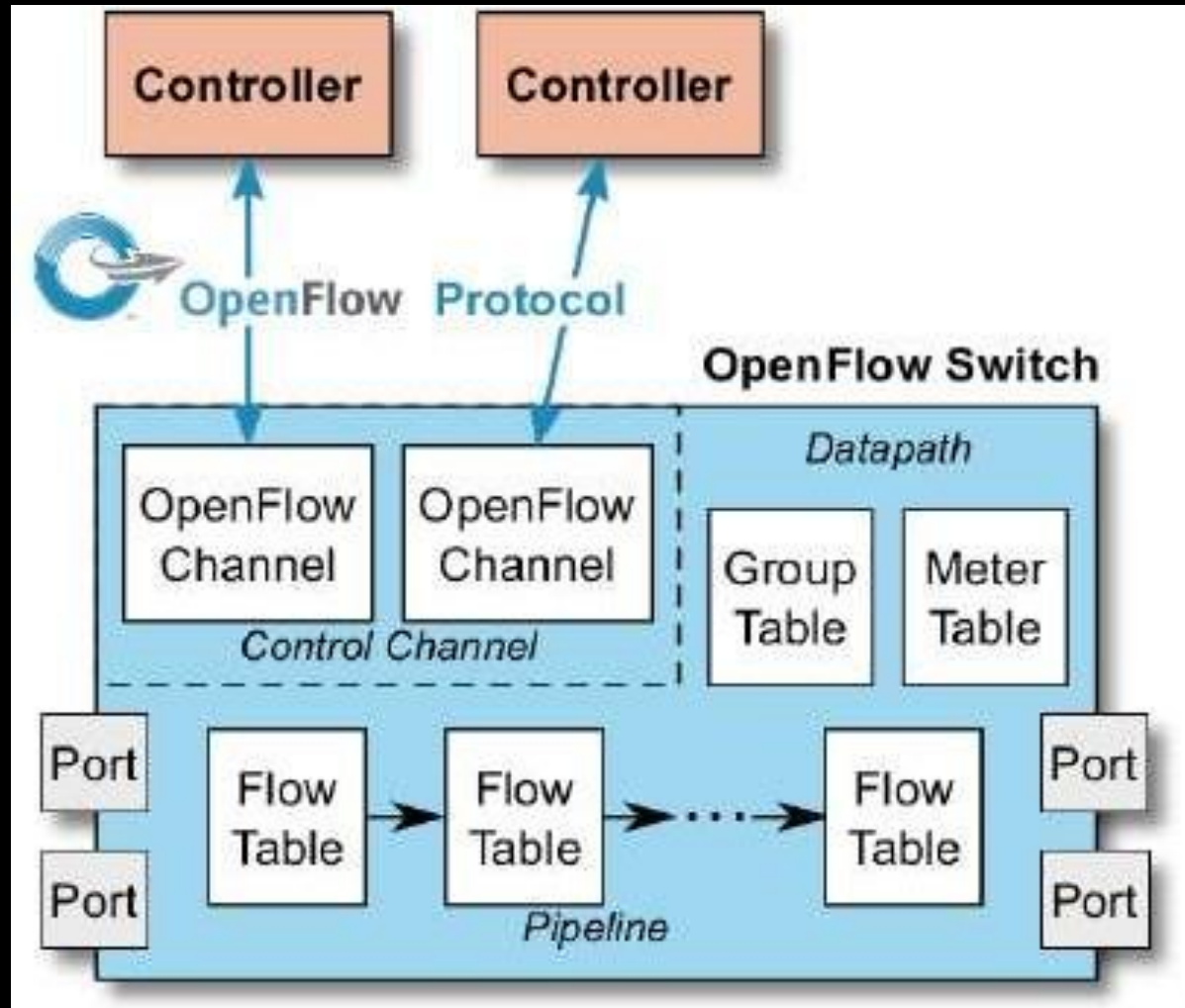
OpenFlow Switch Forwarding



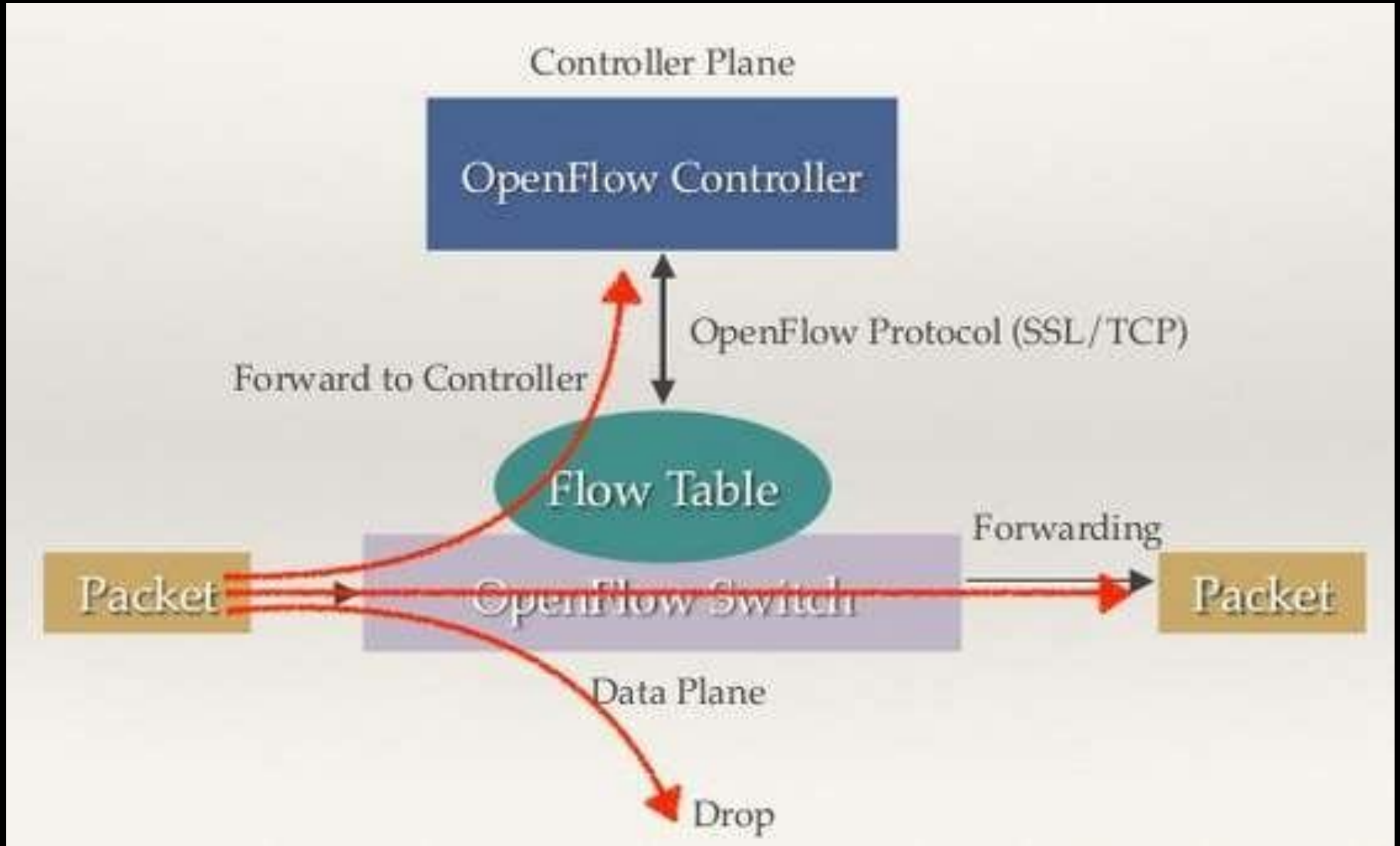
OpenFlow Illustration



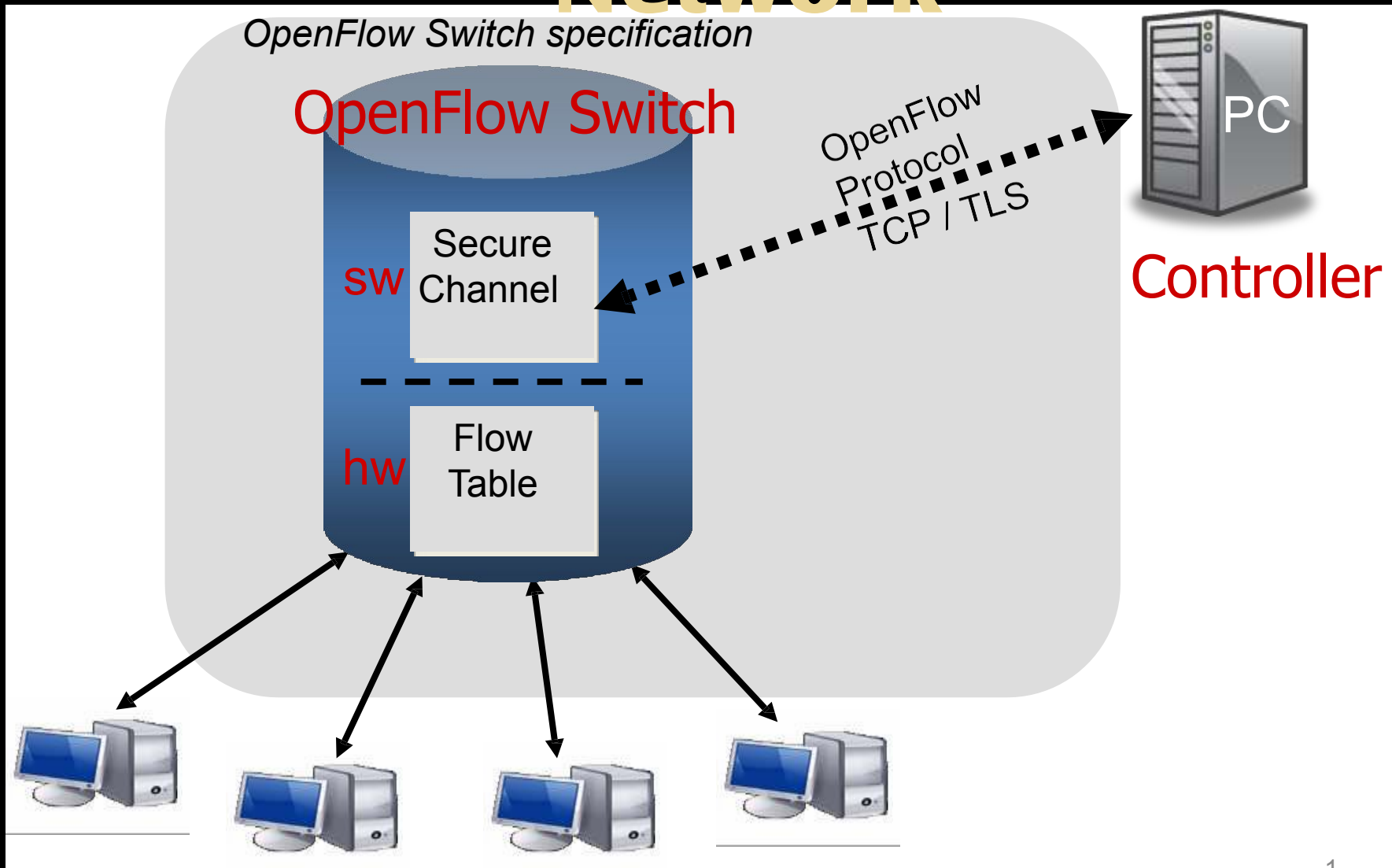
Components of OpenFlow



OpenFlow Packet



Components of OpenFlow Network



OpenFlow Controller

- Manages one or more switch via OpenFlow channels.
- Uses OpenFlow protocol to communicate with a OpenFlow aware switch.
- Acts similar to control plane of traditional switch.
- Provides a network wide abstraction for the applications on northbound.
- Responsible for programming various tables in the OpenFlow Switch.
- **Single switch can be managed by more than one controller for load balancing or redundancy purpose.** In this case the controller can take any one of the following roles.
 - Master.
 - Slave.
 - Equal.

OpenFlow Controllers

- OpenSource
 - OpenDayLight
 - Floodlight
 - RYU
 - NOX/POX
 - ONOS

- Commercial Controllers
 - Cisco APIC
 - VMware NSX Controller
 - HP VAN SDN Controller
 - NEC ProgrammableFlow PF6800 Controller
 - Nuage Networks Virtualized Services Controller (VSC)

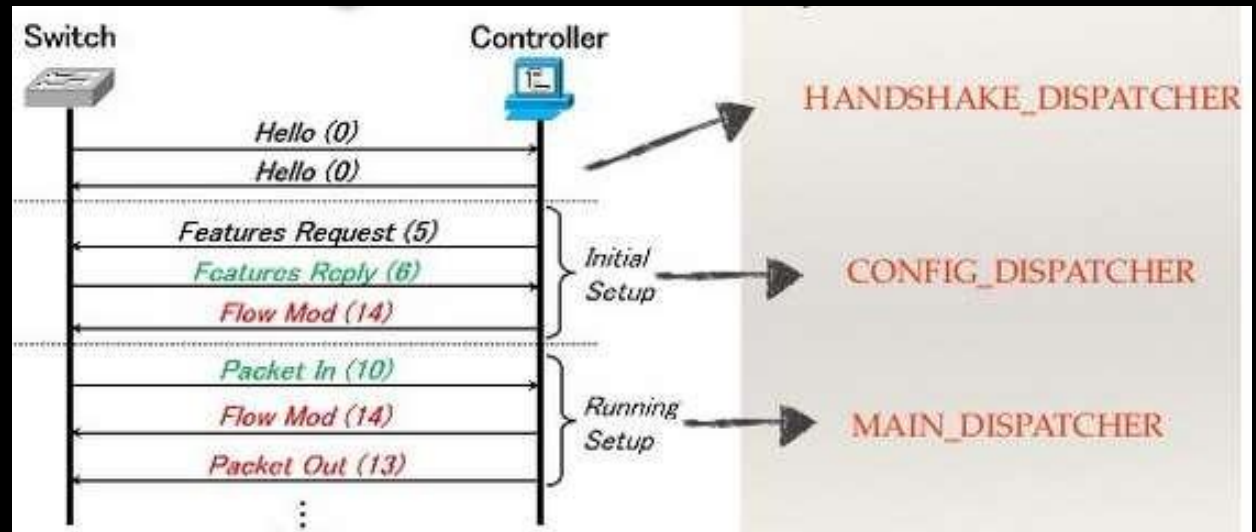
OpenFlow Channel

- Used to exchange OpenFlow message between switch and controller.
- Switch can establish single or multiple connections to same or different controllers (auxiliary connections).
- A controller configures and manages the switch, receives events from the switch, and send packets out the switch via this interface
- The Secure Channel connection is a TLS/TCP connection. Switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key

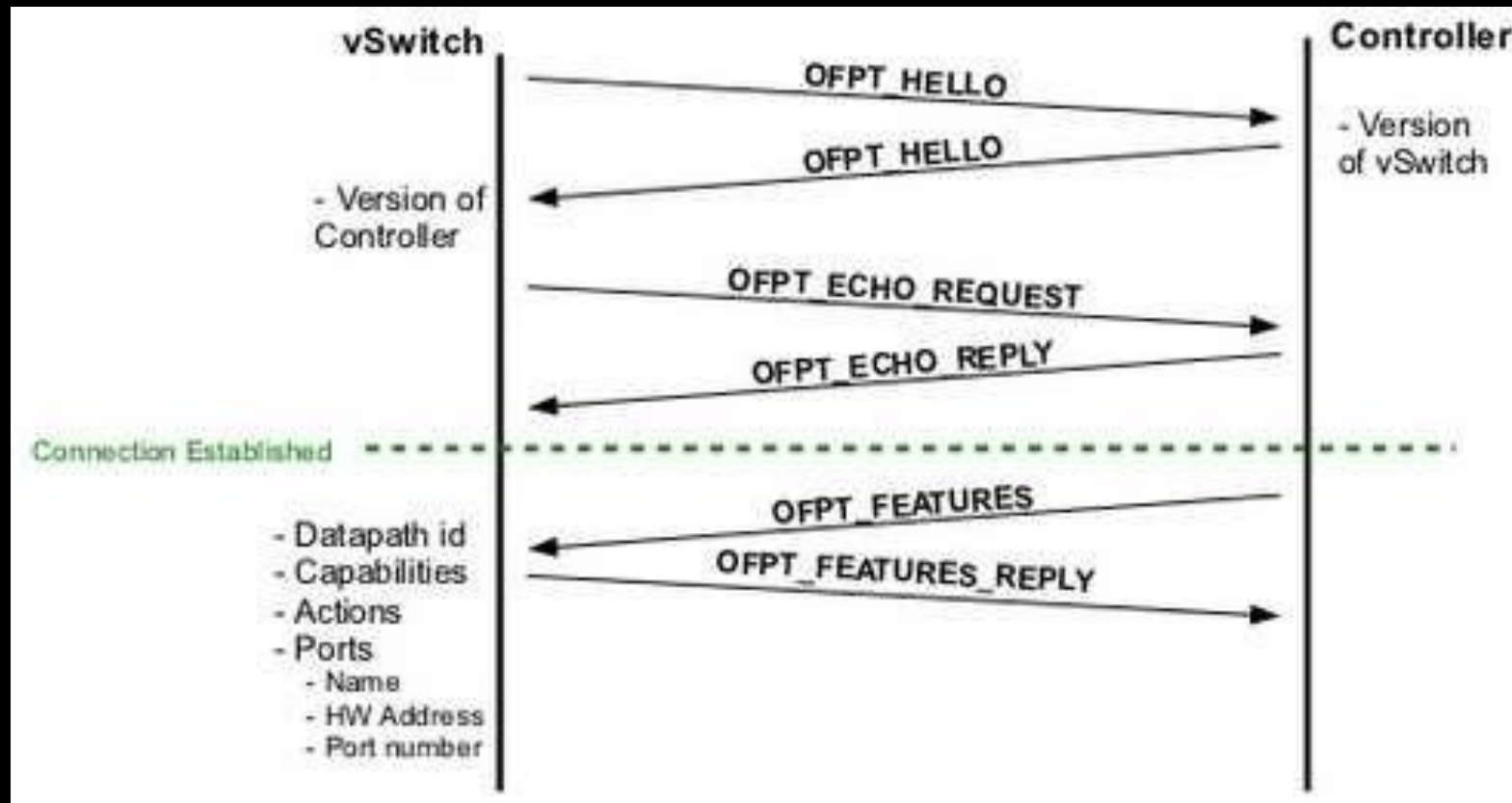
OpenFlow Message

Three types of OpenFlow messages

1. Controller to Switch
 - Feature Request
 - Configuration
 - Modify State (flow)
 - Packet Out
2. Asynchronous (switch to controller)
 - Packet-in
 - Flow Removed
 - Port Status
 - Error
3. Asymmetric
 - Hello
 - Echo
 - Experimenter



Initial Connection Setup



OpenFlow - Hello (From Switch to Controller)

The screenshot shows a Wireshark interface with a packet capture filter set to `tcp.port==6633`. The capture is from a loopback interface. The packet list shows several TCP SYN and ACK packets, followed by two OpenFlow HELLO messages (Type: OFPT_HELLO) from the switch to the controller.

No.	Time	Source	Destination	Protocol	Length	Info
15	32.55746000	10.0.2.15	10.0.2.15	TCP	74	59773->6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=64510 TSecr=0 WS=512
16	32.55748300	10.0.2.15	10.0.2.15	TCP	74	6633->59773 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=64510 TSecr=0
17	32.55750400	10.0.2.15	10.0.2.15	TCP	66	59773->6633 [ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=64510 TSecr=64510
18	32.55906000	10.0.2.15	10.0.2.15	TCP	74	59774->6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=64510 TSecr=0 WS=512
19	32.55908100	10.0.2.15	10.0.2.15	TCP	74	6633->59774 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=64510 TSecr=0
20	32.55910100	10.0.2.15	10.0.2.15	TCP	66	59774->6633 [ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=64510 TSecr=64510
21	32.56135100	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO
22	32.56147400	10.0.2.15	10.0.2.15	TCP	66	6633->59772 [ACK] Seq=1 Ack=17 Win=44032 Len=0 TSval=64511 TSecr=64511
23	32.56160600	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO

Frame 21: 82 bytes on wire (656 bits), 82 bytes captured (656 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: 10.0.2.15 (10.0.2.15), Dst: 10.0.2.15 (10.0.2.15)
- Transmission Control Protocol, Src Port: 59772 (59772), Dst Port: 6633 (6633), Seq: 1, Ack: 1, Len: 16
- OpenFlow 1.3
 - Version: 1.3 (0x04)
 - Type: OFPT_HELLO (0)
 - Length: 16
 - Transaction ID: 9

Frame (frame): 82 bytes | Packets: 1448 - Display... | Profile: Default

OpenFlow - Hello (From Controller to Switch)

The screenshot shows a Wireshark interface capturing traffic on a loopback interface. The filter is set to `tcp.port==6633`. The packet list shows several TCP and OpenFlow packets. The selected packet is an OpenFlow 1.3 Hello message (Type: OFPT_HELLO) from source 10.0.2.15 to destination 10.0.2.15. The packet details show the OpenFlow 1.3 structure, including the Version (1.3), Type (OFPT_HELLO), Length (16), Transaction ID (0), and an Element of Type OFPHET_VERSIONBITMAP (1) with a Length of 8 and Bitmap 00000012. The packet bytes are displayed at the bottom.

No.	Time	Source	Destination	Protocol	Length	Info
19	32.559681000	10.0.2.15	10.0.2.15	TCP	74	6633→59774 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=64510 TS...
20	32.559101000	10.0.2.15	10.0.2.15	TCP	66	59774→6633 [ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=64510 TSecr=64510
21	32.561351000	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO
22	32.561474000	10.0.2.15	10.0.2.15	TCP	66	6633→59772 [ACK] Seq=1 Ack=17 Win=44032 Len=0 TSval=64511 TSecr=64511
23	32.561606000	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO
24	32.561650000	10.0.2.15	10.0.2.15	TCP	66	6633→59773 [ACK] Seq=1 Ack=17 Win=44032 Len=0 TSval=64511 TSecr=64511
25	32.561732000	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO
26	32.561753000	10.0.2.15	10.0.2.15	TCP	66	6633→59774 [ACK] Seq=1 Ack=17 Win=44032 Len=0 TSval=64511 TSecr=64511
27	32.571849000	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO

Packet 27 details:

- Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15
- Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 59772 (59772), Seq: 1, Ack: 17, Len: 16
- OpenFlow 1.3
 - Version: 1.3 (0x04)
 - Type: OFPT_HELLO (0)
 - Length: 16
 - Transaction ID: 0
 - Element
 - Type: OFPHET_VERSIONBITMAP (1)
 - Length: 8
 - Bitmap: 00000012

Packet bytes (hex):

```
0020 02 0f 19 e9 e9 7c 09 1e 1a 05 2a a6 27 b9 00 10
0030 00 50 18 54 00 00 01 01 08 0a 00 00 fc 02 00 00
0040 fb ff 04 00 00 10 00 00 00 0a 00 01 00 00 00 00
0050 00 12
```


OpenFlow - Feature Request (From Controller to Switch)

SDN Hub tutorial VM 64-bit with Docker [Running] - Oracle VM VirtualBox

Machine View Devices Help

Applications Menu OpenDaylight Dlux - Mozill... Capturing from Loopback:... Terminal 13 Nov, 07:20

Capturing from Loopback: lo [Wireshark 1.12.1 (Git Rev Unknown from unknown)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: tcp.port==6633 Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
21	32.561351000	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO
22	32.561474000	10.0.2.15	10.0.2.15	TCP	66	6633->59772 [ACK] Seq=1 Ack=17 Win=44032 Len=0 TSval=64511 TSecr=64511
23	32.561606000	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO
24	32.561650000	10.0.2.15	10.0.2.15	TCP	66	6633->59773 [ACK] Seq=1 Ack=17 Win=44032 Len=0 TSval=64511 TSecr=64511
25	32.561732000	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO
26	32.561753000	10.0.2.15	10.0.2.15	TCP	66	6633->59774 [ACK] Seq=1 Ack=17 Win=44032 Len=0 TSval=64511 TSecr=64511
27	32.571840000	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO
28	32.571899000	10.0.2.15	10.0.2.15	TCP	66	59772->6633 [ACK] Seq=17 Ack=17 Win=44032 Len=0 TSval=64514 TSecr=64514
29	32.572470000	10.0.2.15	10.0.2.15	OpenFlow	74	Type: OFPT_FEATURES_REQUEST

Frame 29: 74 bytes on wire (592 bits), 74 bytes captured (592 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: 10.0.2.15 (10.0.2.15), Dst: 10.0.2.15 (10.0.2.15)

Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 59772 (59772), Seq: 17, Ack: 17, Len: 8

OpenFlow 1.3

Version: 1.3 (0x04)

Type: OFPT_FEATURES_REQUEST (5)

Length: 8

Transaction ID: 11

0000 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.

0010 00 3c e1 7d 40 00 40 06 41 21 0a 00 02 0f 0a 00 .<.)@.@. A!.....

0020 02 0f 19 e9 e9 7c 09 1e 1a 95 2a a6 27 b9 80 18|. *.'....

0030 00 56 18 4c 00 00 01 01 08 0a 00 00 fc 02 00 00 .V.L.....

Frame (frame), 74 bytes Packets: 5318 - Display... Profile: Default

20:50 13-11-2017

OpenFlow - Feature Reply (From Switch to Controller)

The image shows a Wireshark packet capture window titled "Capturing from Loopback: lo [Wireshark 1.12.1 (Git Rev Unknown from unknown)]". The filter is set to "tcp.port==6633". The packet list shows several packets, with the selected packet being an OpenFlow message (No. 31, Time 32.572727000, Source 10.0.2.15, Destination 10.0.2.15, Protocol OpenFlow, Length 98, Info Type: OFPT_FEATURES_REPLY).

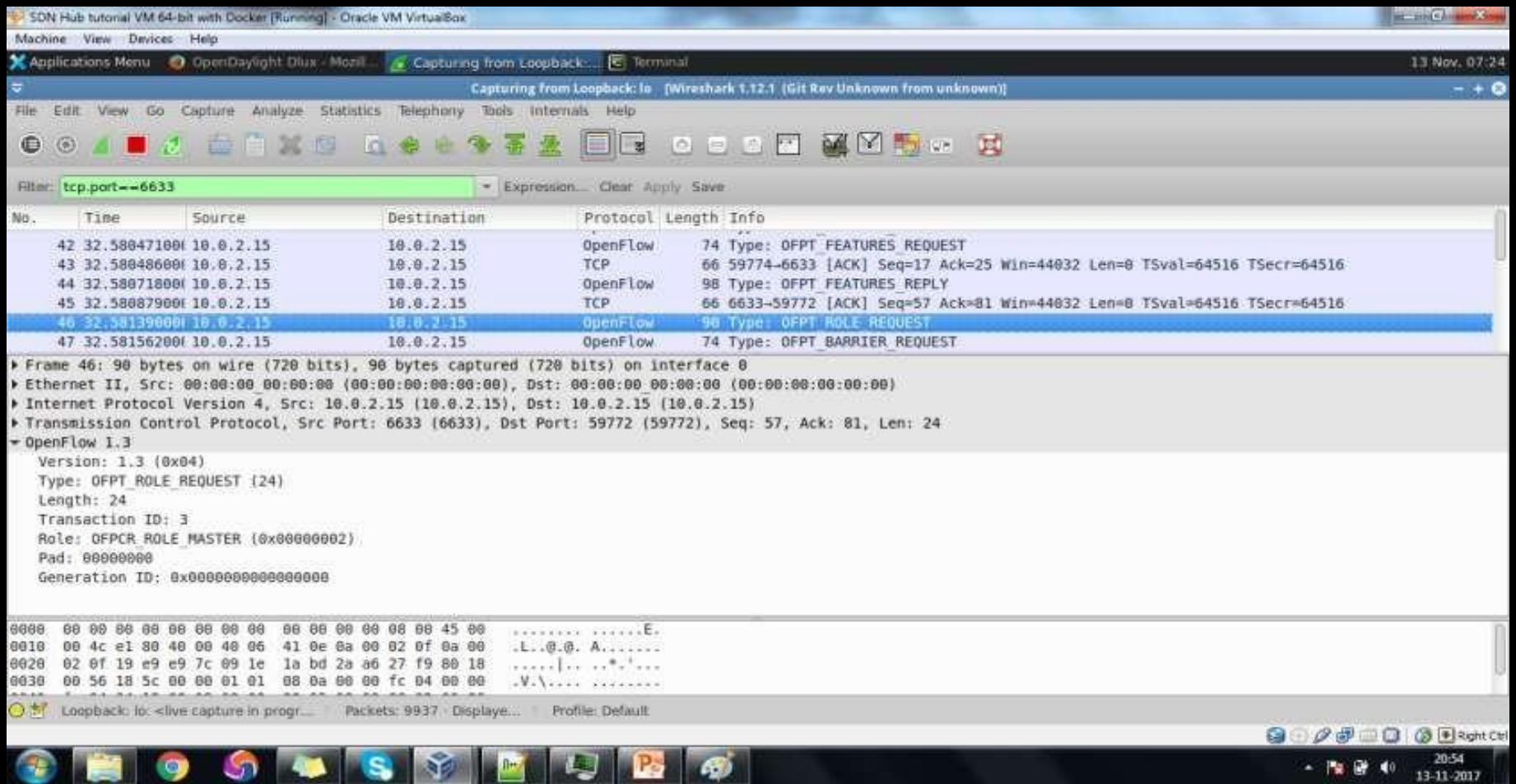
No.	Time	Source	Destination	Protocol	Length	Info
26	32.561753000	10.0.2.15	10.0.2.15	TCP	66	6633->59774 [ACK] Seq=1 Ack=17 Win=44032 Len=0 TSval=64511 TSecr=64511
27	32.571840000	10.0.2.15	10.0.2.15	OpenFlow	82	Type: OFPT_HELLO
28	32.571899000	10.0.2.15	10.0.2.15	TCP	66	59772->6633 [ACK] Seq=17 Ack=17 Win=44032 Len=0 TSval=64514 TSecr=64514
29	32.572470000	10.0.2.15	10.0.2.15	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
30	32.572522000	10.0.2.15	10.0.2.15	TCP	66	59772->6633 [ACK] Seq=17 Ack=25 Win=44032 Len=0 TSval=64514 TSecr=64514
31	32.572727000	10.0.2.15	10.0.2.15	OpenFlow	98	Type: OFPT_FEATURES_REPLY

The packet details pane shows the following information for the selected packet:

- Transmission Control Protocol, Src Port: 59772 (59772), Dst Port: 6633 (6633), Seq: 17, Ack: 25, Len: 32
- OpenFlow 1.3
 - Version: 1.3 (0x04)
 - Type: OFPT_FEATURES_REPLY (6)
 - Length: 32
 - Transaction ID: 11
 - datapath_id: 0x0000000000000001
 - n_buffers: 256
 - n_tables: 254
 - auxiliary_id: 0
 - Pad: 0
 - capabilities: 0x0000004f
 - Reserved: 0x00000000

The packet bytes pane shows the raw data in hexadecimal and ASCII format.

OpenFlow - Role Request (From Controller to Switch)



OpenFlow - Role Reply (From Switch to Controller)

The image shows a Wireshark packet capture window titled "Capturing from Loopback: lo [Wireshark 1.12.1 (Git Rev Unknown from unknown)]". The filter is set to "tcp.port==6633". The packet list shows several packets, with packet 48 selected, which is an OpenFlow message of type OFPT_ROLE_REPLY.

No.	Time	Source	Destination	Protocol	Length	Info
43	32.580486000	10.0.2.15	10.0.2.15	TCP	66	59774→6633 [ACK] Seq=17 Ack=25 Win=44032 Len=0 TSval=64516 TSecr=64516
44	32.580718000	10.0.2.15	10.0.2.15	OpenFlow	98	Type: OFPT_FEATURES_REPLY
45	32.580879000	10.0.2.15	10.0.2.15	TCP	66	6633→59772 [ACK] Seq=57 Ack=81 Win=44032 Len=0 TSval=64516 TSecr=64516
46	32.581390000	10.0.2.15	10.0.2.15	OpenFlow	90	Type: OFPT_ROLE_REQUEST
47	32.581562000	10.0.2.15	10.0.2.15	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
48	32.581711000	10.0.2.15	10.0.2.15	OpenFlow	90	Type: OFPT_ROLE_REPLY

Frame 48: 90 bytes on wire (720 bits), 90 bytes captured (720 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: 10.0.2.15 (10.0.2.15), Dst: 10.0.2.15 (10.0.2.15)
Transmission Control Protocol, Src Port: 59772 (59772), Dst Port: 6633 (6633), Seq: 81, Ack: 89, Len: 24
OpenFlow 1.3
Version: 1.3 (0x04)
Type: OFPT_ROLE_REPLY (25)
Length: 24
Transaction ID: 3
Role: OFPCR_ROLE_MASTER (0x00000002)
Pad: 00000000
Generation ID: 0x0000000000000000

Loopback: lo: <live capture in progr... Packets: 10661 - Display... Profile: Default

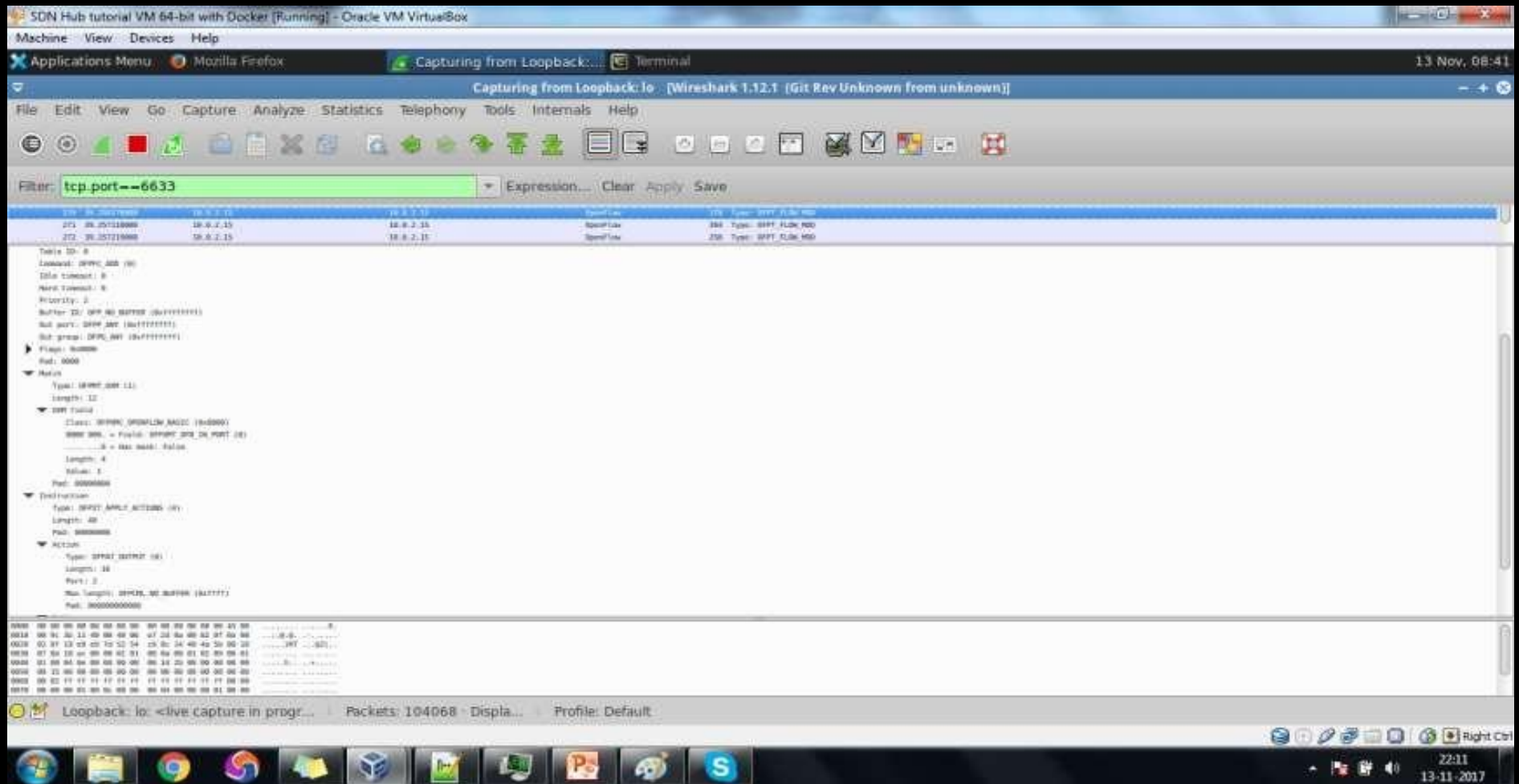
OpenFlow - Packet IN (From Switch to Controller)

The screenshot shows a Wireshark capture window titled "Capturing from Loopback: lo [Wireshark 1.12.1 (Git Rev Unknown from unknown)]". The filter is set to "tcp.port==6633". The capture shows several packets, with packet 189 highlighted. The packet details pane shows the following information:

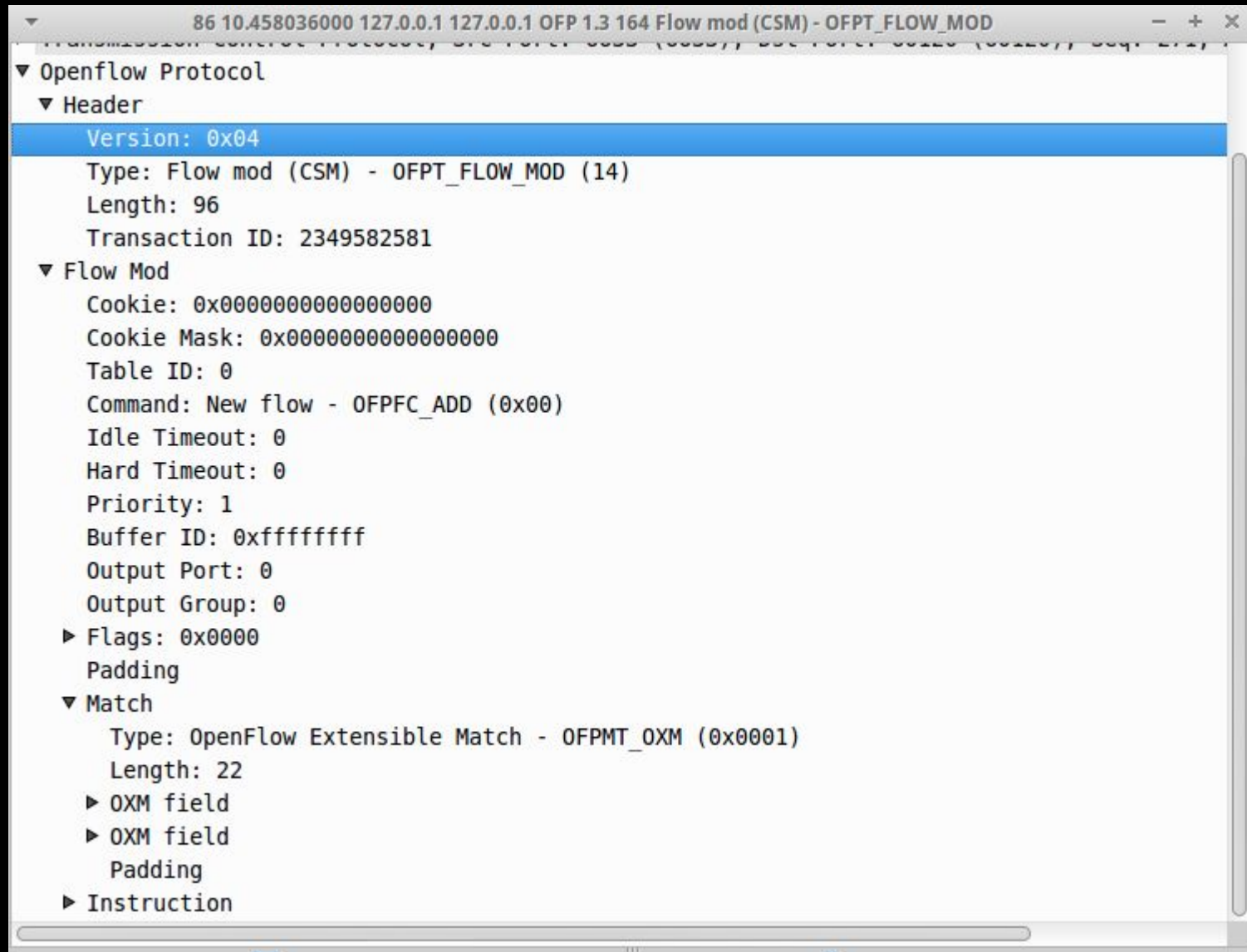
- Frame 189: 178 bytes on wire (1424 bits), 178 bytes captured (1424 bits) on interface 0
- Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
- Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 10.0.2.15 (10.0.2.15)
- Transmission Control Protocol, Src Port: 59773 (59773), Dst Port: 6633 (6633), Seq: 476311, Ack: 1193, Len: 112
- OpenFlow 1.3
 - Version: 1.3 (0x04)
 - Type: OFPT_PACKET_IN (10)
 - Length: 112
 - Transaction ID: 0
 - Buffer ID: OFP_NO_BUFFER (0xffffffff)
 - Total length: 70
 - Reason: OFPR_ACTION (1)

The packet bytes pane shows the raw data of the packet, including the Ethernet II header, IP header, TCP header, and OpenFlow packet body.

OpenFlow - Flow Config (From Switch to Controller)



OpenFlow - Flow Config (From Switch to Controller)



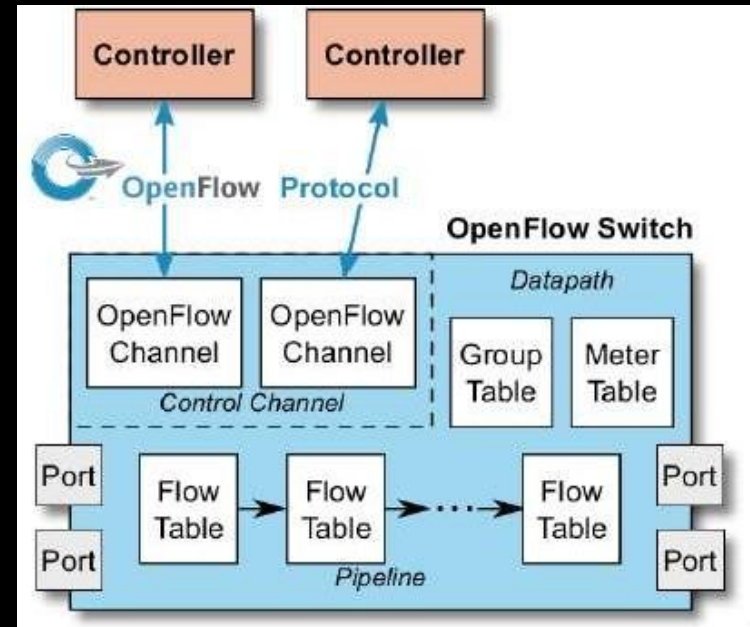
OpenFlow Switches

■ Types of the switches:

- OpenFlow only
- OpenFlow hybrid

■ Major Components

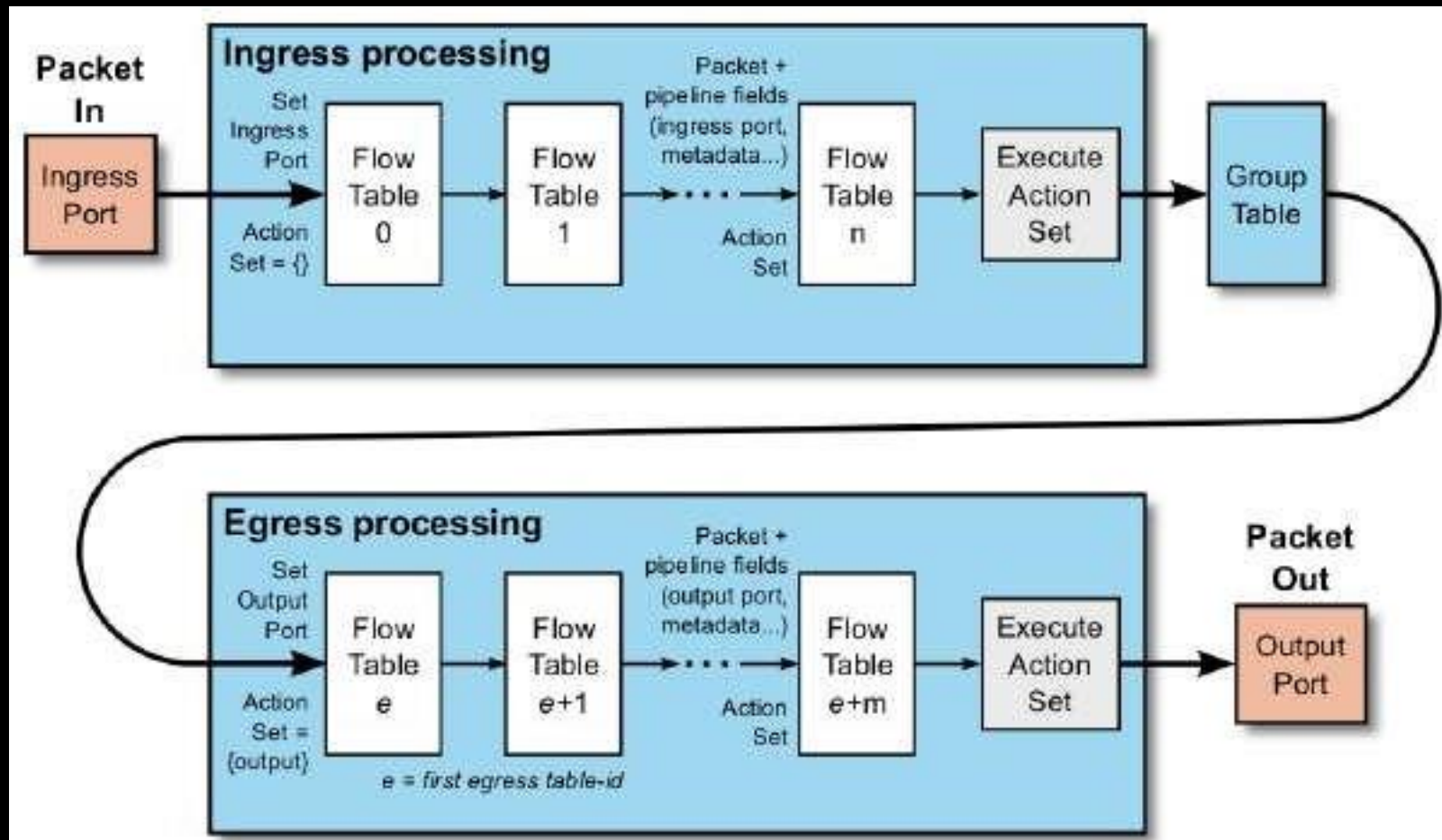
- Ports
- Flow Table
- Group Table
- Meter Table



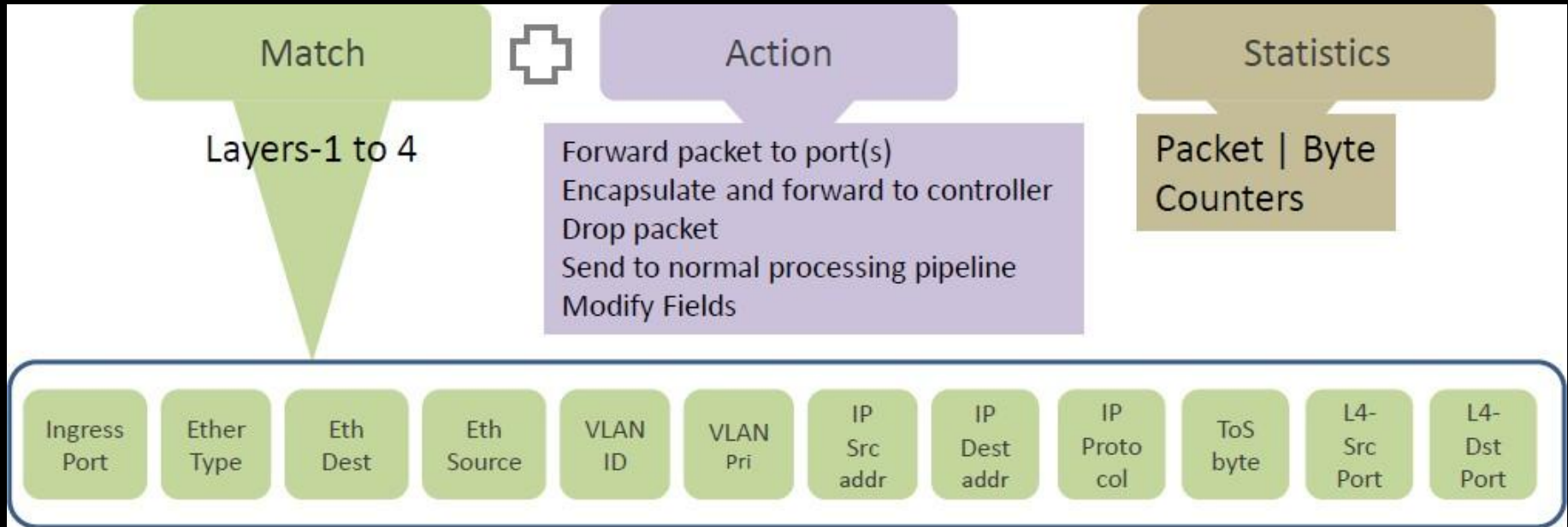
OpenFlow Ports

- Network interfaces for passing packets between Openflow Processing and the rest of the Network.
- Types of Ports:
 - Physical Ports
 - Switch defined ports
 - Eg. Physical ports map one to one Ethernet interfaces
 - Logical Ports
 - Switch defined ports that don't correspond to a hardware interface of switch
 - Logical ports include "Tunnel-ID".
 - Reserved Ports
 - Defined by OpenFlow Spec
 - Specifies generic forwarding actions such as sending to the controller, flooding and forwarding using non-openflow methods

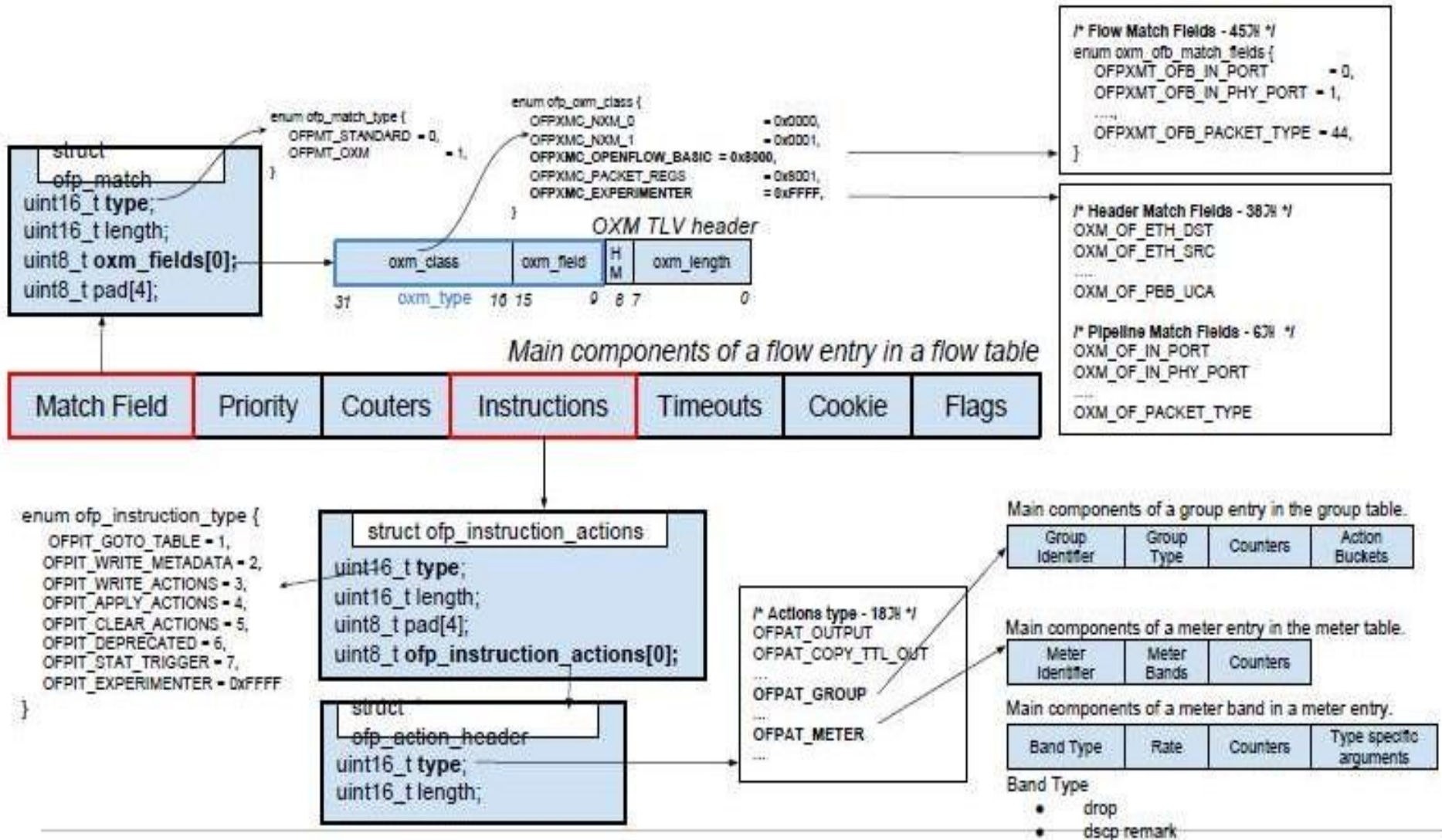
Pipeline Processing



Flow Table Components



Flow Entry



OpenFlow Table

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie
--------------	----------	----------	--------------	----------	--------

■ Instructions are executed when a packet matches entry.

■ Instruction result can

- Change the packet
- Add to the Action set
- Affect the Pipeline processing

■ Sample Instruction Types

- Meter ID - Direct a packet to the meter id.
- Apply-Actions - Apply a specific action immediately
- Clear-Actions - Clear all the actions in the action set
- Write-Actions - Add a new action into the existing action set
- Write-Metadata - Write the masked meta data value
- Goto-Table - Indicate the next table in the processing pipeline

Action Set and Action List

■ Action Set

- Action set is associated with each packet. Flow Entries modify the action set using write-action/ clear-action
- Actions in the action-set will be executed when pipeline is **stopped**
- Action set contains maximum of one action of each type. If multiple actions of the same type need to be added then use “Apply-Actions”

■ Action List

- **Execute an action immediately**
- Actions are executed sequentially in the order they have been specified
- If action list contains an output action, a copy of the packet is forwarded in its current state to the desired port
- Action-set shouldn't be changed because of action-list

Action

Action

■ What to do with the packet when match criteria matches with the packet

■ Some of the Action Type

- **Output** - Fwd a pkt to the specified openflow port (physical/ logical/reserved)
- **Set Queue** - Determines which queue should be used for scheduling and forwarding packet
- **Drop** - Packets which doesn't have output action should be dropped
- **Group** - Process the packet through specified group
- Push-Tag/ Pop-Tag - Insert VLAN, MPLS, PBB tage
- Set-Field - Rewriting a field in the packet header
- Change TTL - Decrement TTL
- Copy TTL inwards – apply copy inward actions to the packet
- Push MPLS – apply MPLS tag push action to the packet
- Push PBB – apply PBB tag push action to the packet

Flow Entry

Flow Entry Timeouts

- Each Flow Entry contains an optional Idle and Hard Timeout field.
- Switch removes entry and sends flow removed message to controller when,
 - No packets have matched within the Idle Timeout
 - The flow was inserted more than the Hard Timeout.
- The timeouts are optional

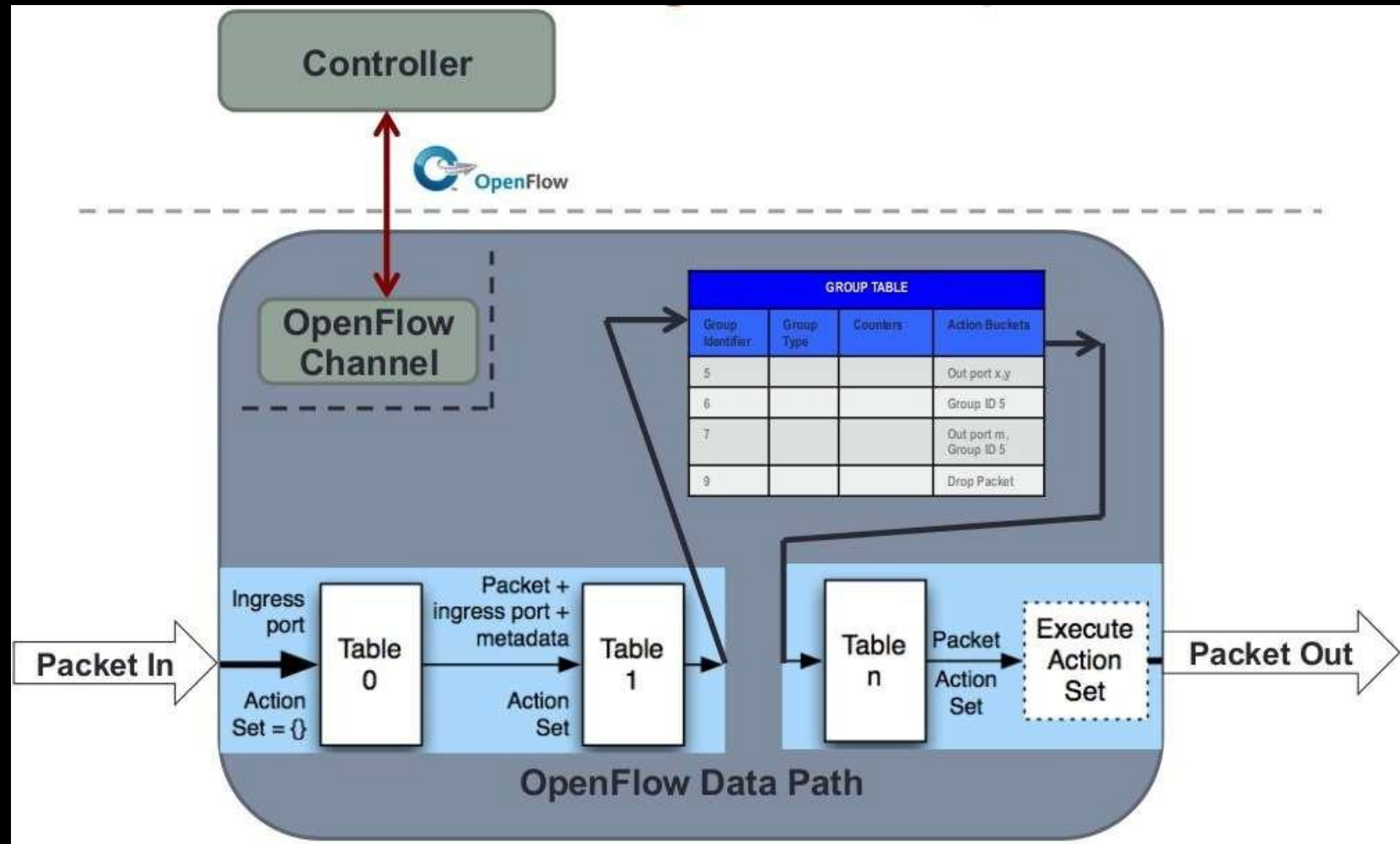
Flow Entry

Reactive Flows	Proactive Flows
First packet of flow is sent to controller for processing	Controller pre-populates flow table in switch.
Controller evaluates packet and sends Flow message to switch insert a flow table entry for the packet	Zero set-up time for new flows
Subsequent packets (in flow) match the entry until idle or hard timeout	Switch in Proactive forwarding 'only' would default to action=drop
Loss of connectivity between switch and controller limit utility of switch	Loss of connection between switch and controller does not disrupt network traffic

Group Table

- Additional method for forwarding to a group of entries.
- Comprises of Group ID, Group Type, Counters, Action buckets (each action bucket contains a set of actions to be executed)
- Group Types:
 - All
 - Execute all buckets in a group
 - Used mainly for multicast and broadcast – fwd a pkt on all the ports
 - Select
 - Execute one bucket in a Group (Eg. ECMP packets)
 - Implemented for load sharing and redundancy
 - Indirect
 - Execute one defined bucket in this Group
 - Supports only a single bucket (Eg. 40K routes are pointing to same next hop)
 - Fast failover
 - Execute the first live bucket
 - Eg. There is a primary path and secondary path – pass the traffic on primary path and if it fails use the secondary one

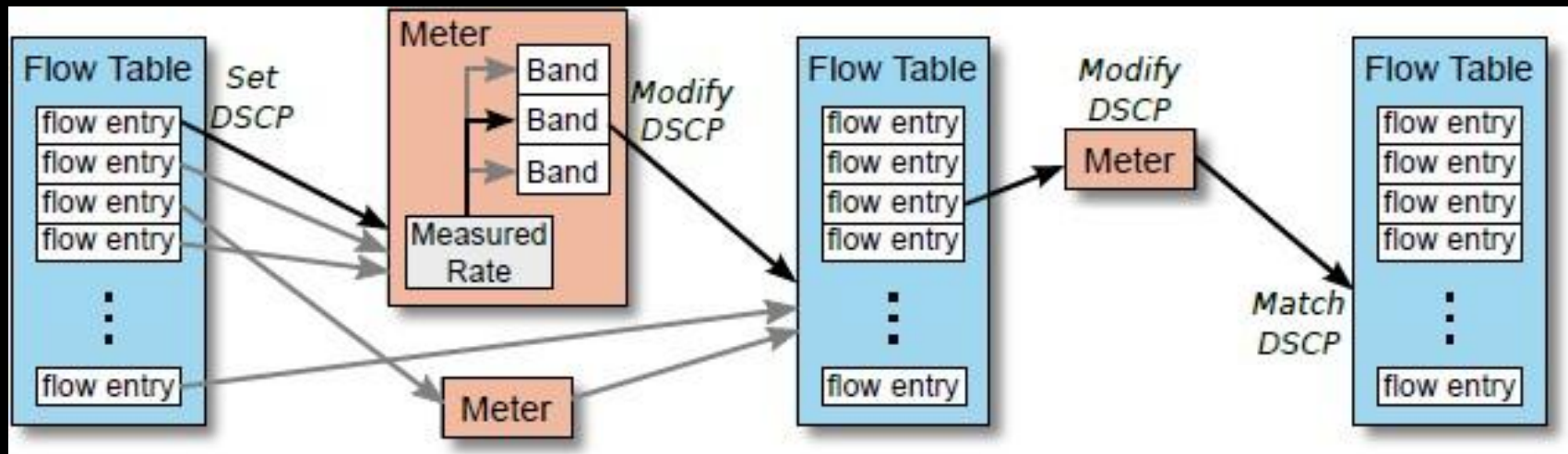
Instruction Execution in Group Table



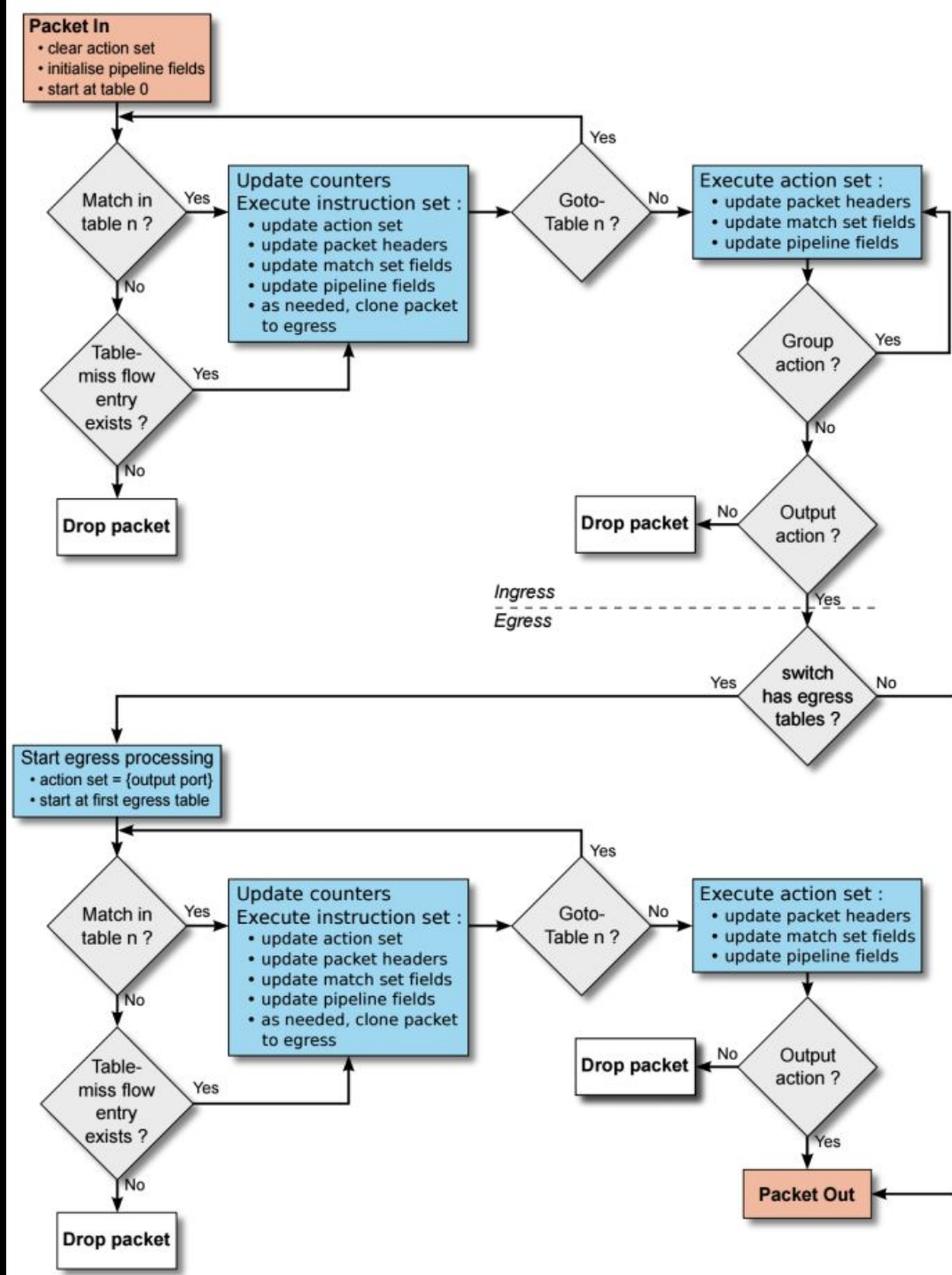
Meter Table

- Consists of meter entries and defining per-flow meters.
 - Per-flow meters enable OF to **implement QoS operations (rate-limiting)**
- Components of Meter table:
 - Meter ID, Meter Band, Counters
- Meters measures the rate of packets assigned to it and enable controlling the rate of those packets
- Meters are attached directly to flow entries
- Meter band: unordered list of meter bands, where each meter band specifies the rate of the band and the way to process packet
- Components of Meter band:
 - Band Type, Rate, Counters, Type specific arguments
 - Band Type : defined how to process a packet (drop/ dscp remark)

Meter Table

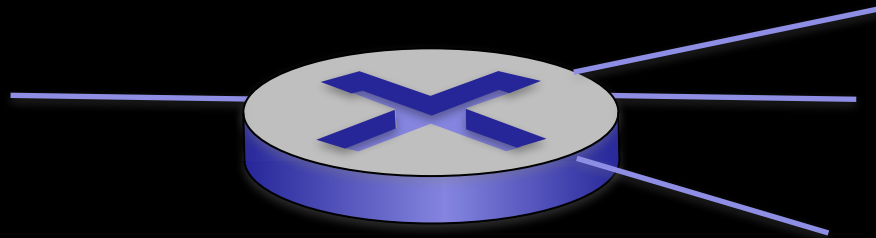


Packet Flow in OpenFlow Switch



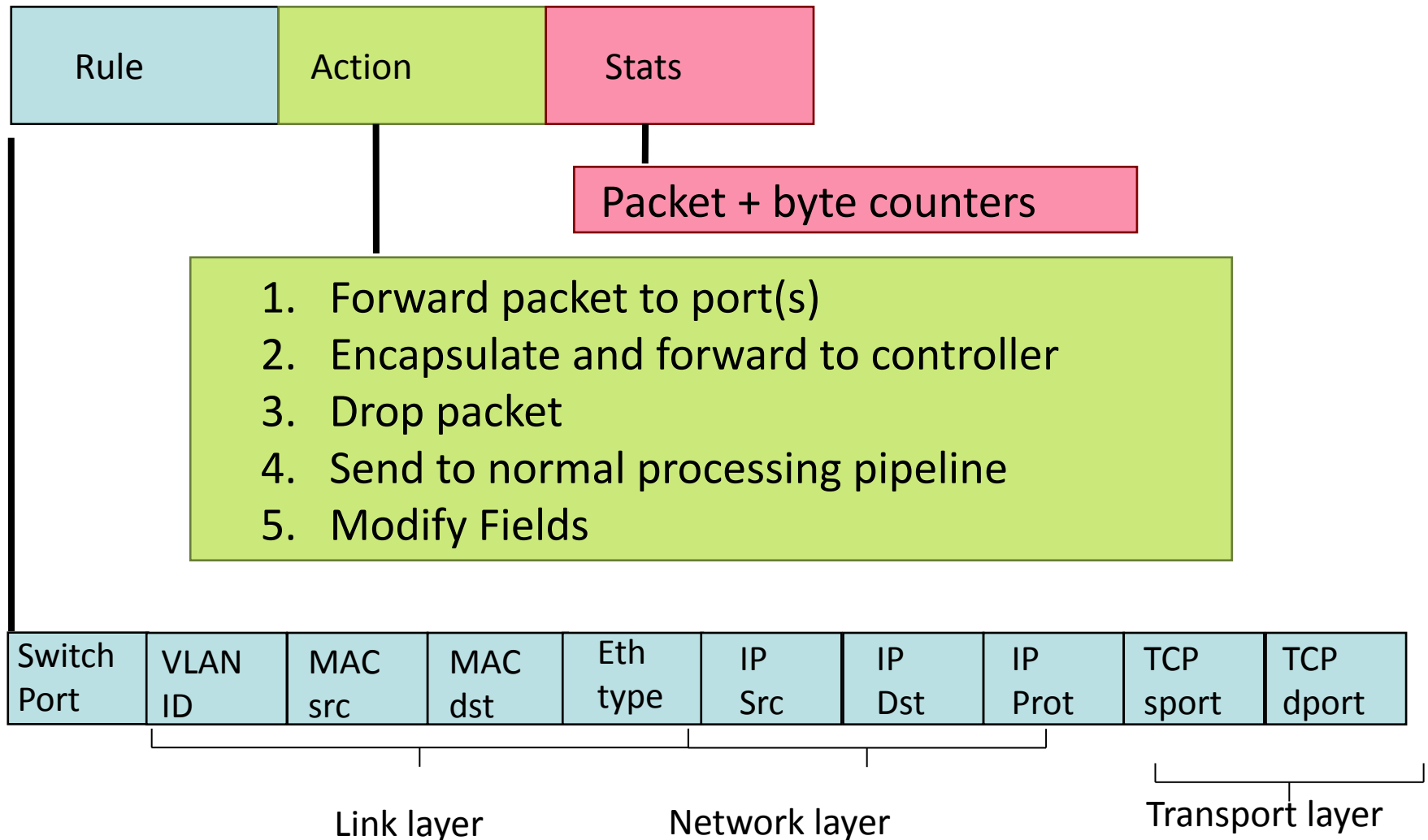
OpenFlow data plane abstraction

- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
 - **Pattern**: match values in packet header fields
 - **Actions**: *for matched packet*: drop, forward, modify, matched packet or send matched packet to controller
 - **Priority**: disambiguate overlapping patterns
 - **Counters**: #bytes and #packets



1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow: Flow Table Entries



Examples

Destination-based forwarding:

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

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

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

do not forward (block) all datagrams destined to TCP port 22

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	128.119.1.1	*	*	*	*	

do not forward (block) all datagrams sent by host 128.119.1.1

Examples

Destination-based layer 2 (switch) forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	port3

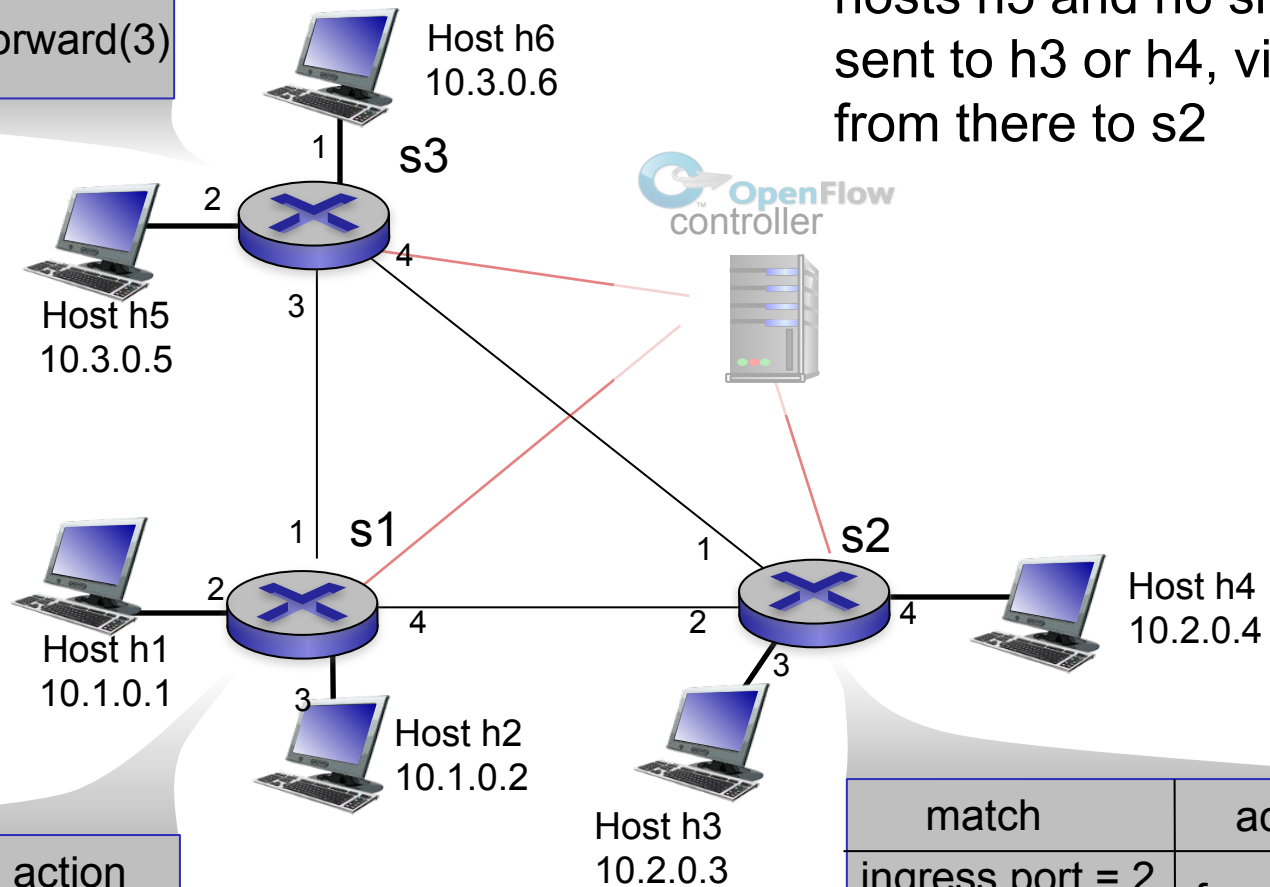
*layer 2 frames from MAC address 22:A7:23:11:E1:02 should
be forwarded to output port 6*

OpenFlow abstraction

- **match+action**: unifies different kinds of devices
- 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

OpenFlow Example

Example: datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2



match	action
IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(3)

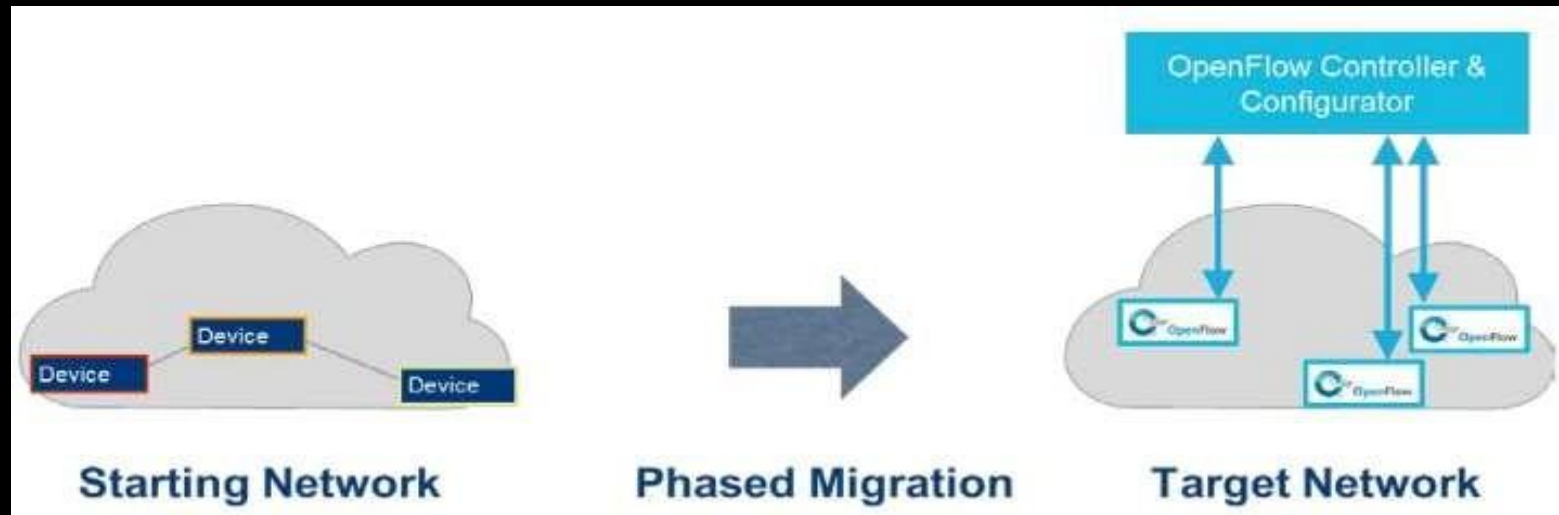
match	action
ingress port = 1 IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(4)

match	action
ingress port = 2 IP Dst = 10.2.0.3	forward(3)
ingress port = 2 IP Dst = 10.2.0.4	forward(4)

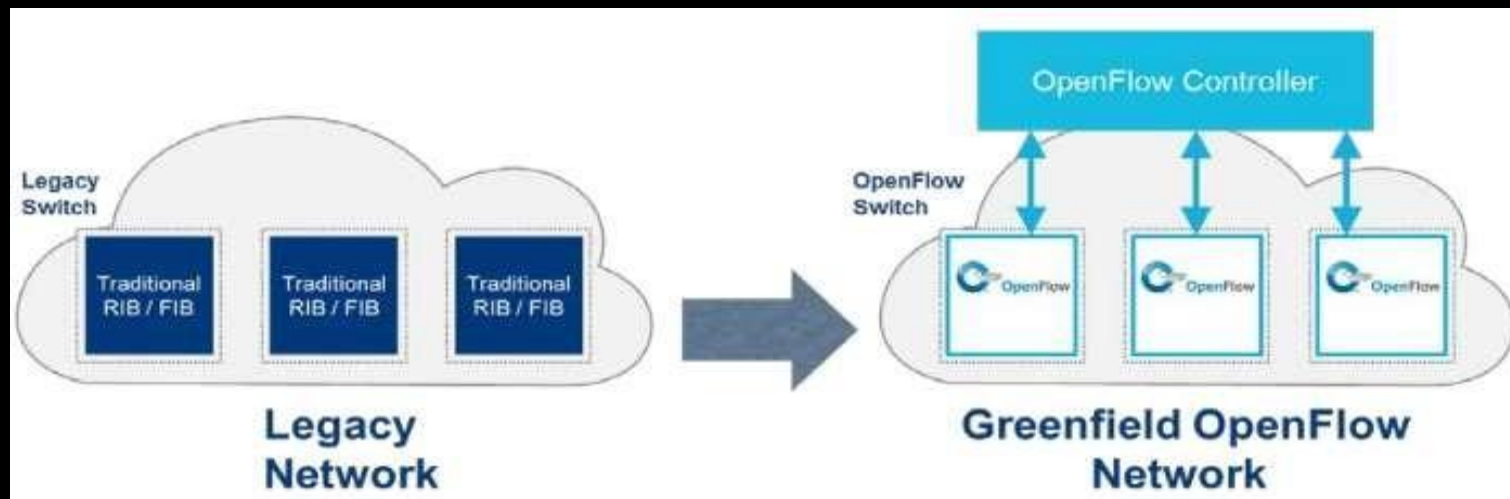
How to Migrate to SDN

Key Steps for Migration

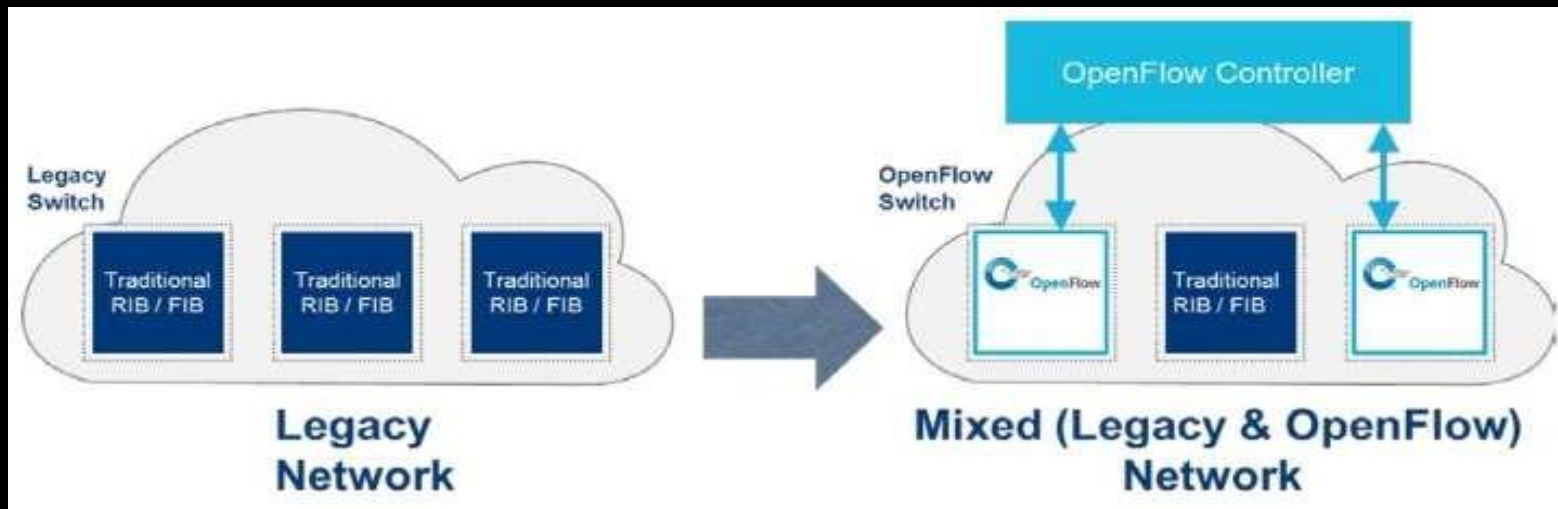
- Identify and prioritize the core requirements of the target network
- Prepare the starting network for migration
- Implement a phased network migration
- Validate the results



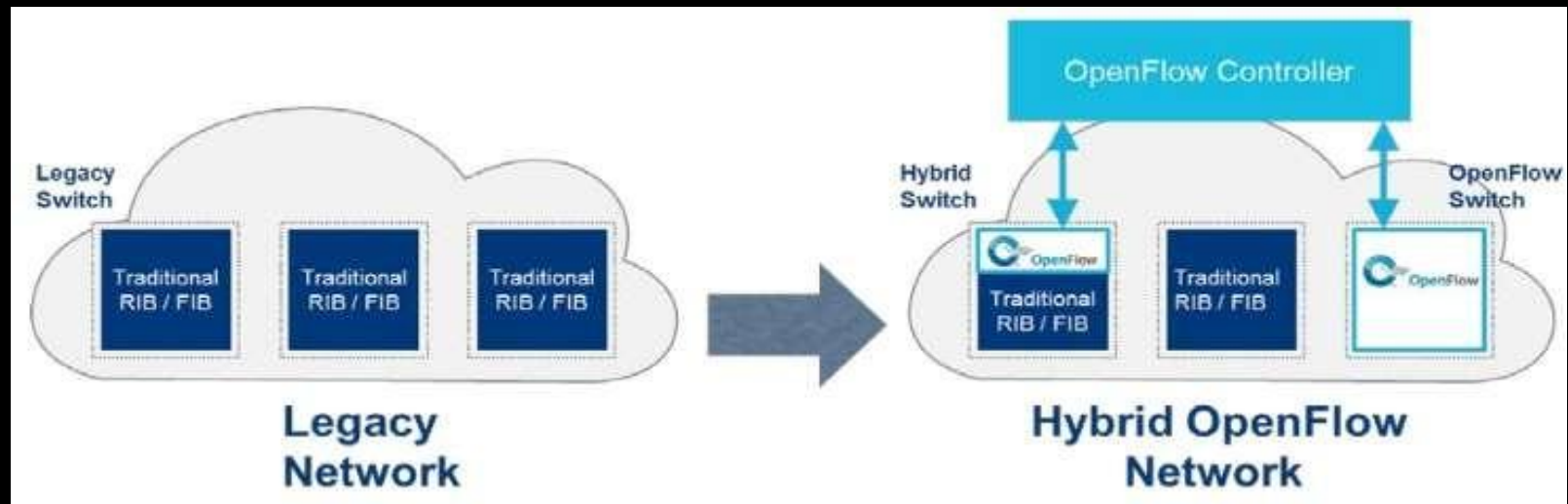
Migration Approaches: Greenfield



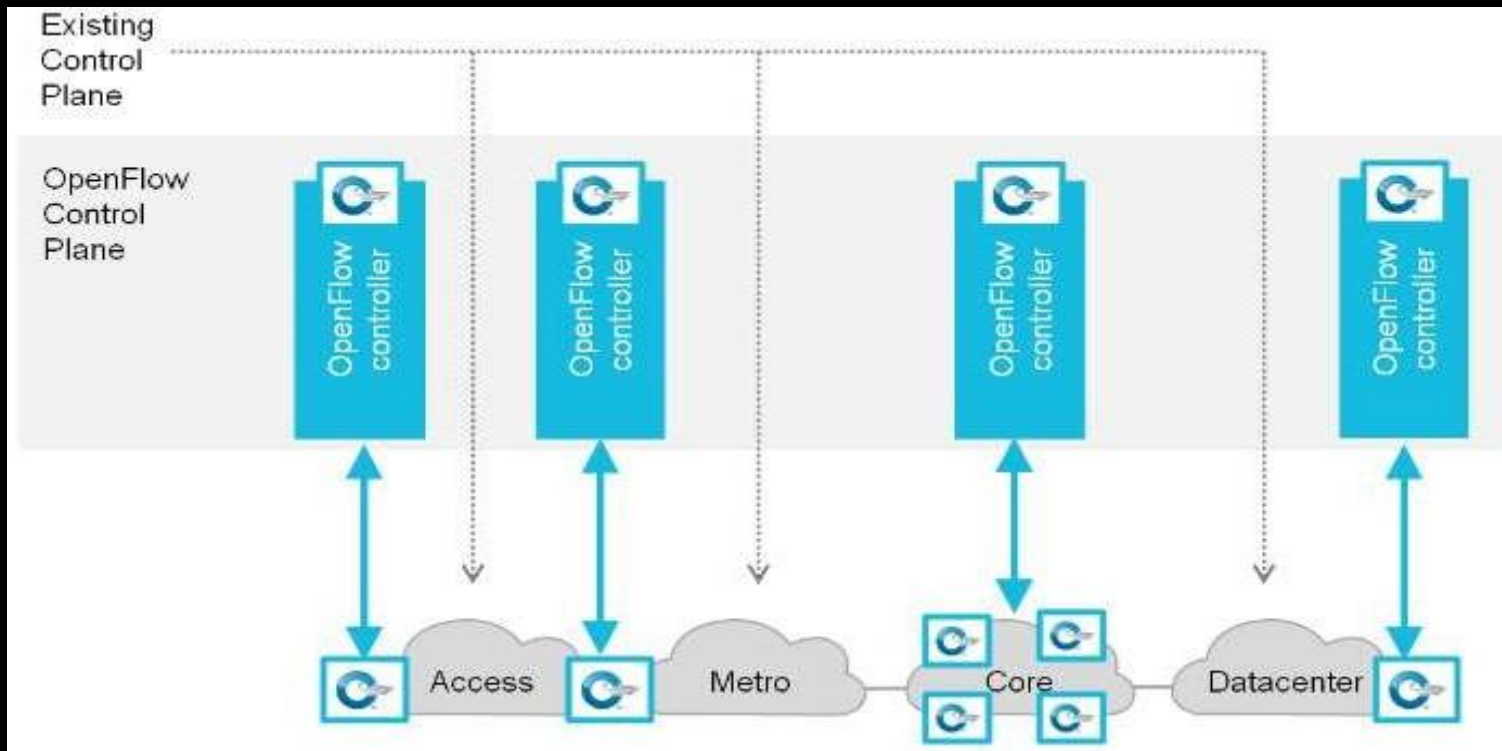
Migration Approaches: Mixed



Migration Approaches: Hybrid



Migration Approaches: Hierarchical



B4: Experience with a Globally-Deployed Software Defined WAN

(Sigcomm 2013)

<http://dl.acm.org/citation.cfm?id=2486019>



OpenFlow @ Google - Urs Hoelzle, Google



Open Networking Summit

✓ Subscribed 5,404


37,269

+ Add to ➔ Share ... More


👍 222 🗨️ 9

<https://youtu.be/JMkvCBOMhno>

Two Kind of Networks

A blue cloud-shaped graphic with a dark blue outline, containing text.

Internet
-facing
user traffic

A green cloud-shaped graphic with a dark green outline, containing text.

Internal
traffic between
Google's global
data
centers

To improve scalability, flexibility, and agility in managing the Internet-facing WAN fabric to enhance Google's user-based services, including Google+, Gmail, YouTube, Google Maps, and others

Problem

What are the problems Google set out to solve?

- Expensive routers
- 30-40% utilization of WANs

Google's situation:

- Two networks: public facing and private facing
- They control applications, servers, network all the way to the edge. *Can control bursts from servers*
- Scale makes them cost sensitive
- Biggest traffic = copies between sites
- Only a “few dozen” locations

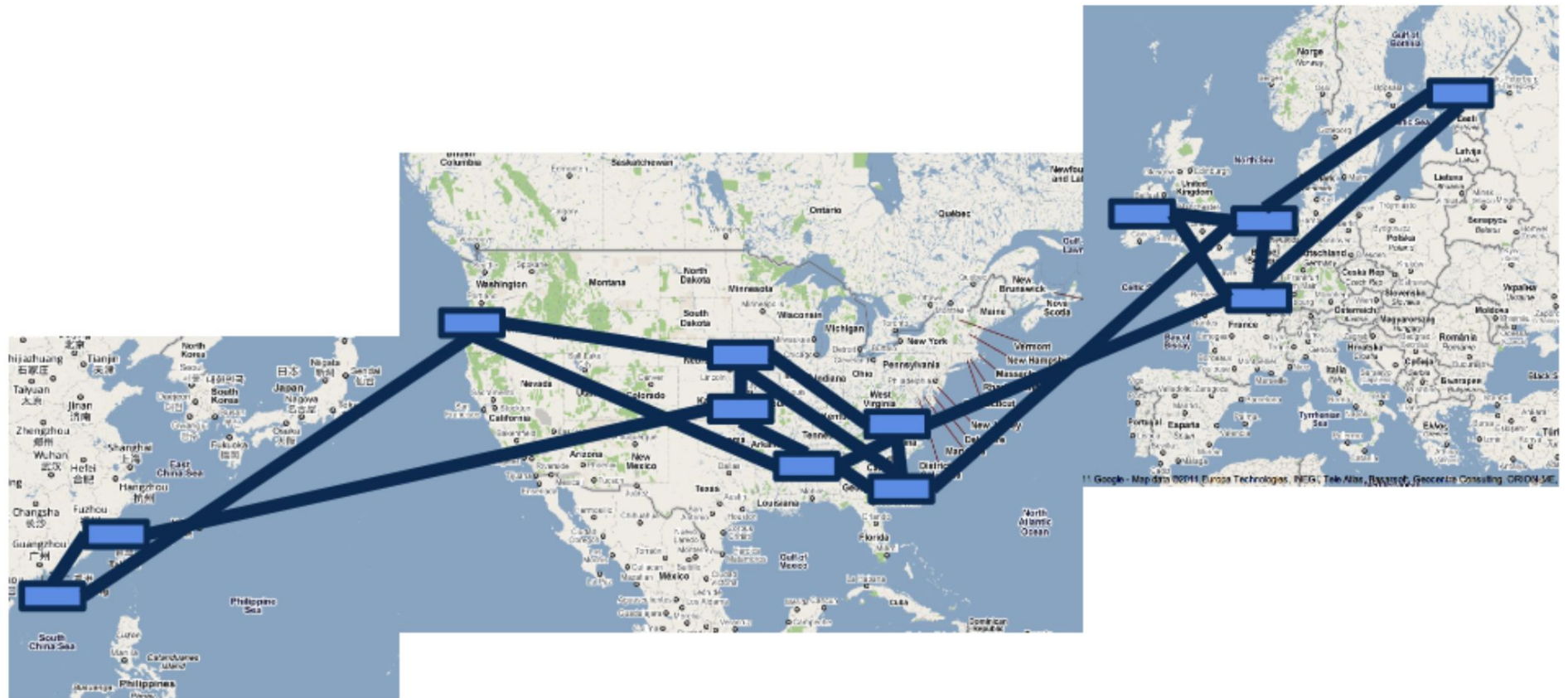


Figure 1: B4 worldwide deployment (2011).

Why SDN for B4?

Design assumptions:

- Failures are inevitable; expose to end applications
- Switch hardware exports a simple interface to populate forwarding entries centrally
- Protocols and control runs on regular servers (of which they have a whole bunch...)

Q: What did they build?

Software Defined Network

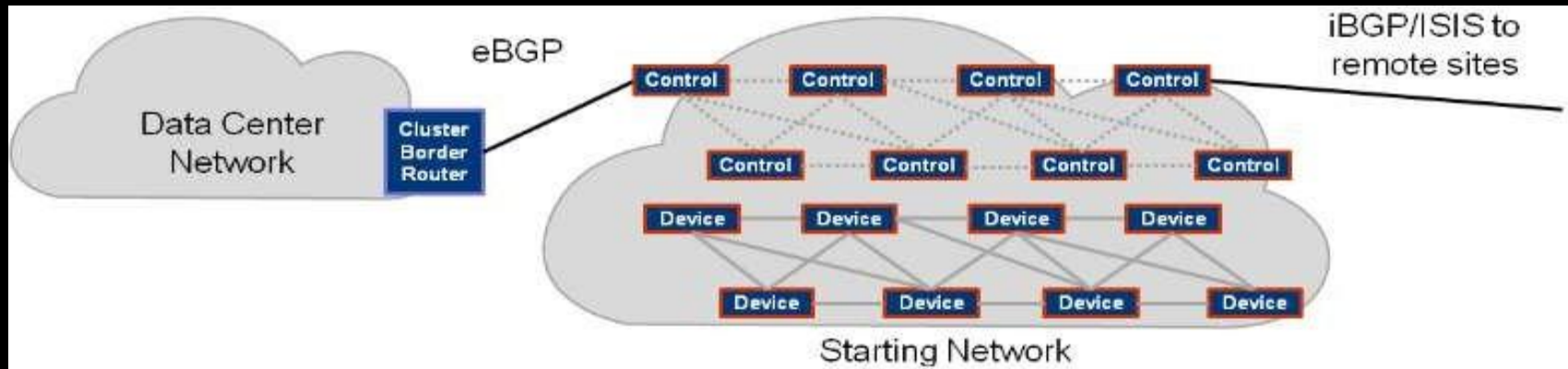
A network in which the control plane is physically separate from the forwarding plane.

and

A single control plane controls several forwarding devices.

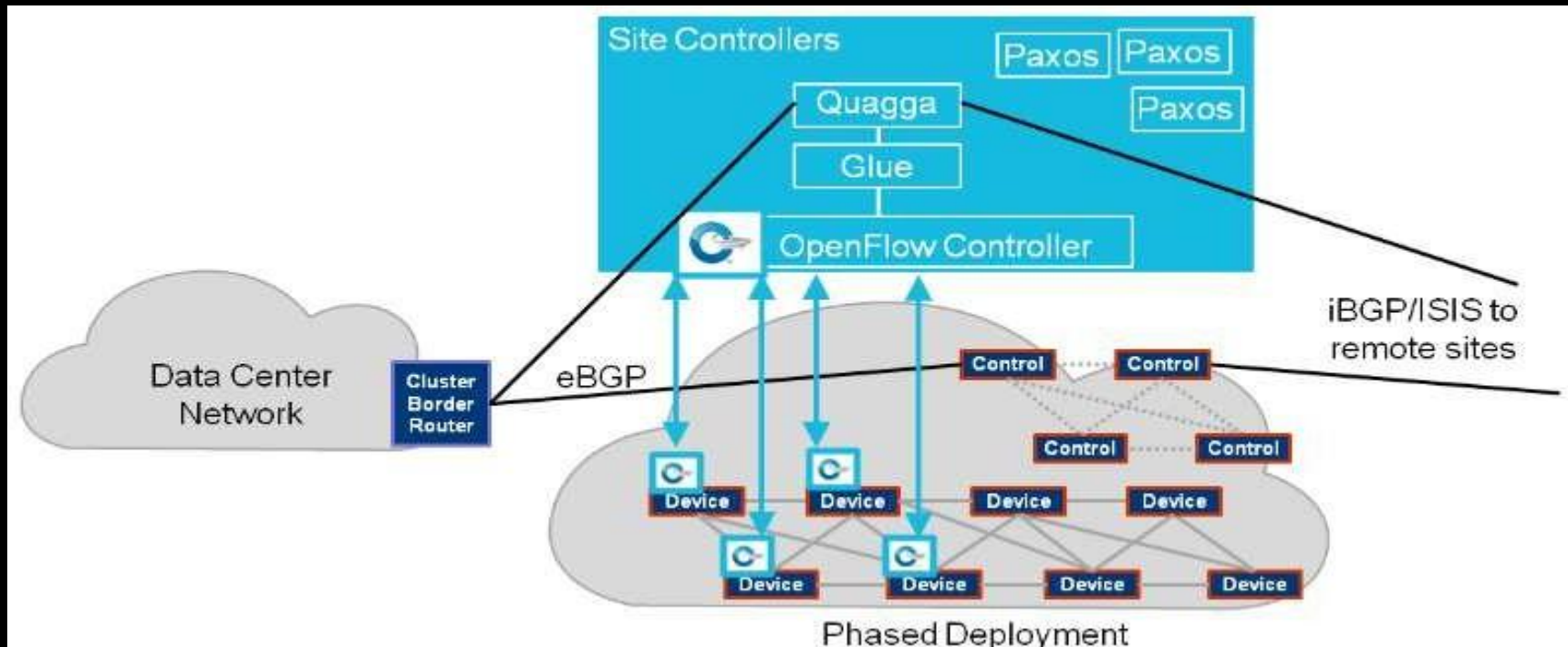
(That's it)

Starting Network



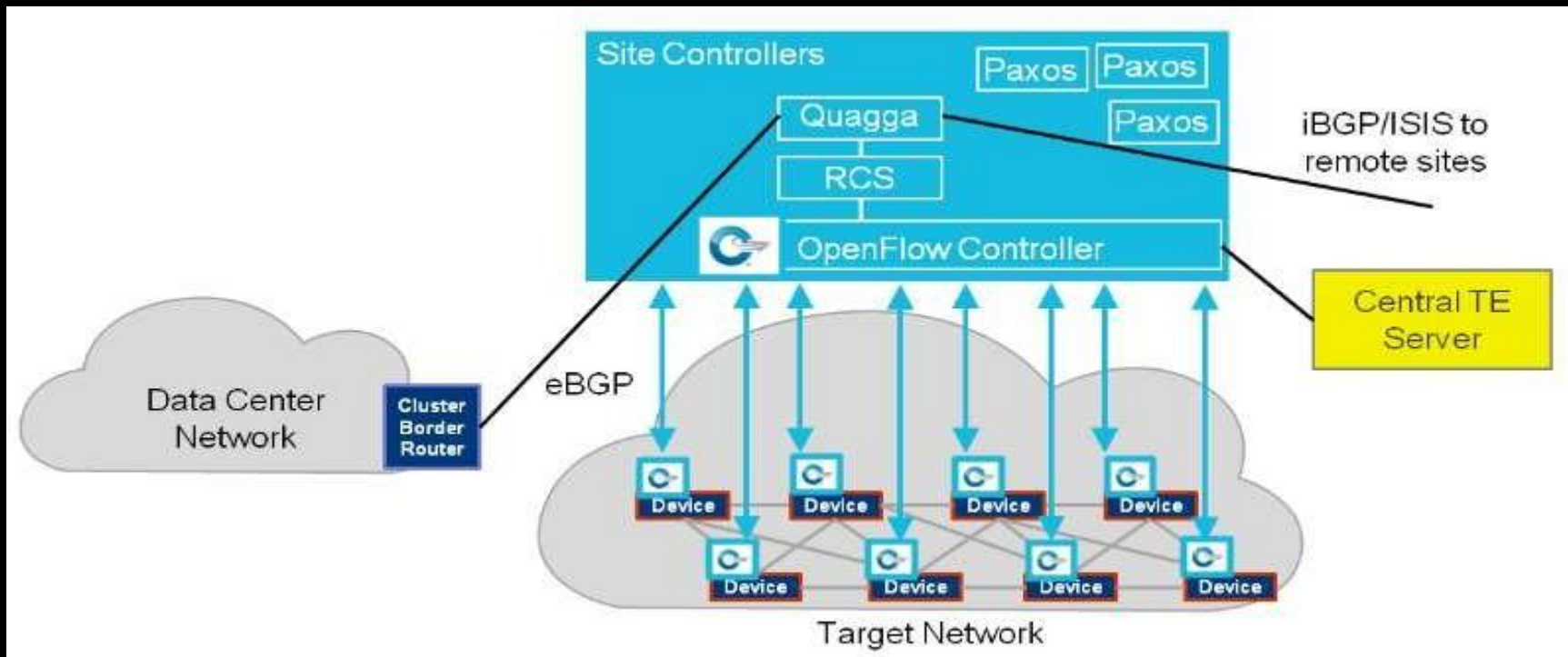
Fully distributed monolithic control and data plane hardware architecture to a physically decentralized (though logically centralized) control plane architecture

Phased deployment



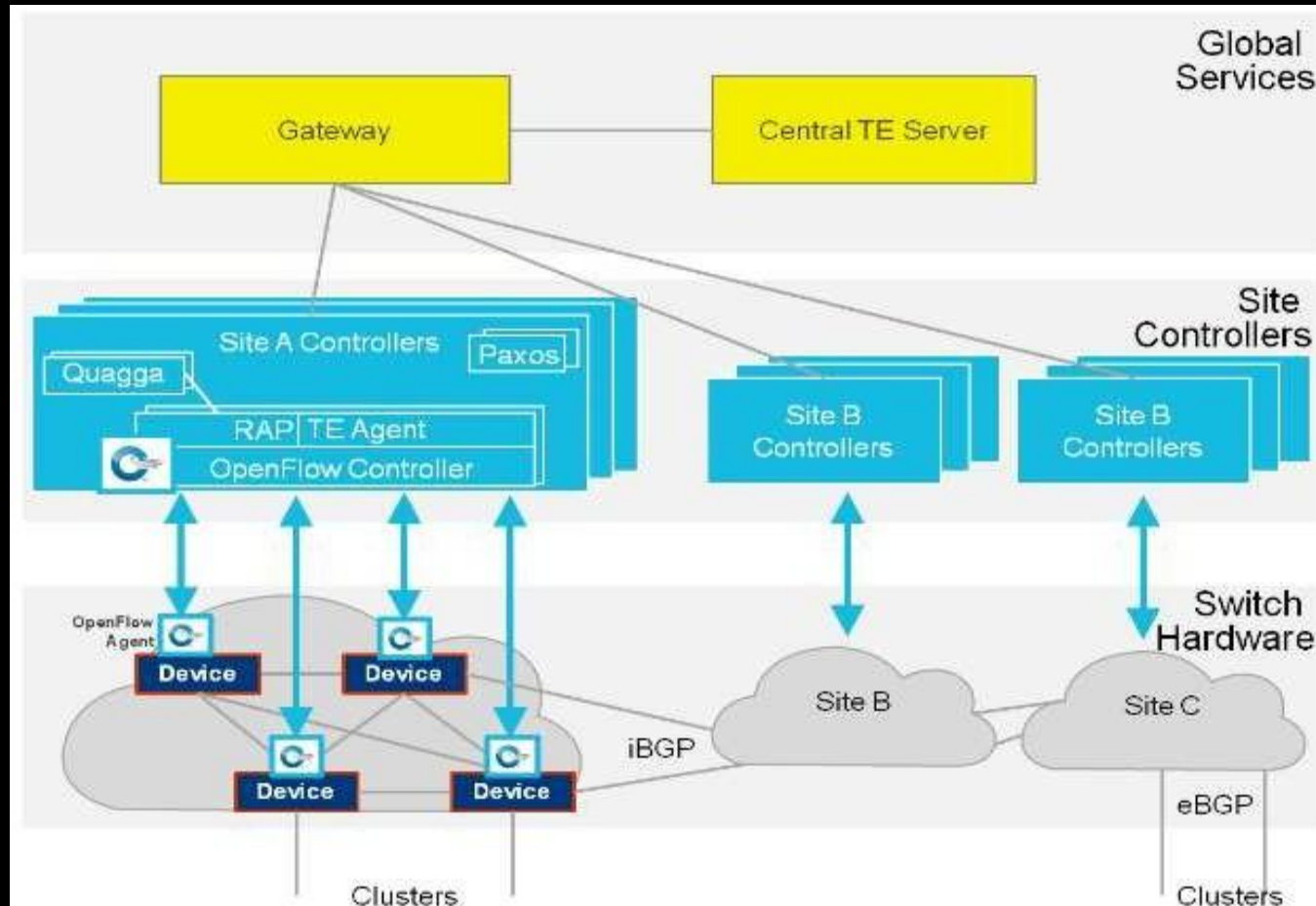
A subset of the nodes in the network were OpenFlow-enabled and controlled by the logically centralized controller utilizing Paxos, an OpenFlow controller, and Quagga open source routing stack that Google adapted to its requirements

Complete OpenFlow



All nodes were OpenFlow-enabled. In the target network, the controller controls the entire network. There is no direct correspondence between the data center and the network. The controller also has a TE server that guides the traffic engineering in the network.

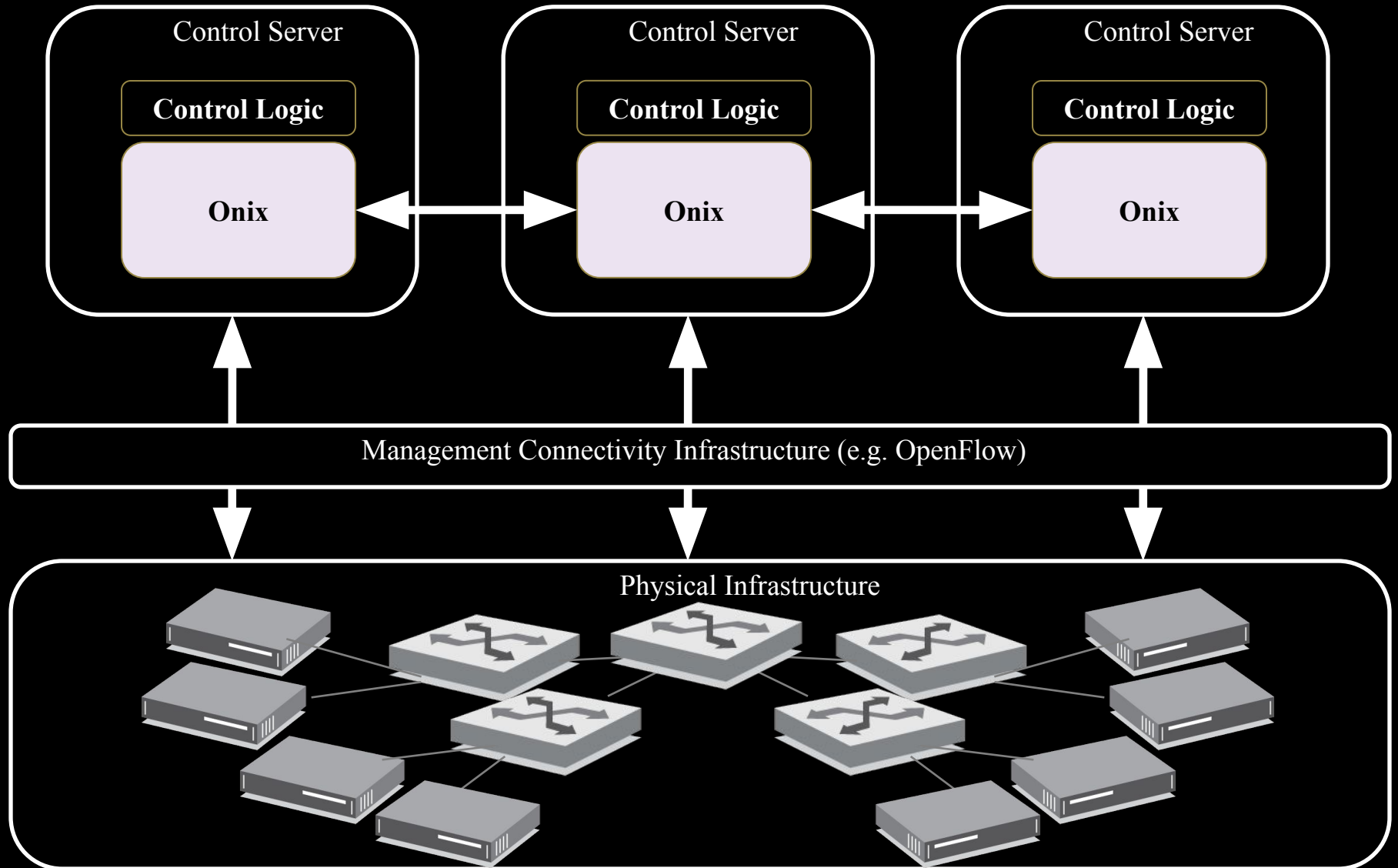
Final Deployment



Why SDN for B4?

- OpenFlow decouples sw and hw evolution
- Dedicated software control plane on commodity servers
- Server CPUs faster than embedded CPUs
- Can reason about global state -> better coordination of planned/unplanned failure

Onix



Onix

- Centralized control logic -> distributed system
- Gives “application” developers useful tools:
 - General API
 - Flexible state distribution mechanisms
 - Flexible scaling and reliability mechanisms

References

1. <http://web.stanford.edu/class/cs244/>
2. B4: Experience with a Globally-Deployed Software Defined WAN
<http://dl.acm.org/citation.cfm?id=2486019>
3. Computer Networking - A top down approach

Conclusions

- Assignment 2 based on Network Management and SDN
- Lab 2 is based on SDN
 - We will be writing an SDN application that acts as a firewall
 - The application should be able to create rules in a network such that certain hosts who are deemed to be “malicious” are not able to send or receive data while all other hosts can send and receive at will
 - There will be a lab session next monday in the BTF lab for those who need help with the lab.

Not examinable

from this slide

Informational

- Different SDN Controllers
- Basic understanding of each controller
 - Concepts
 - Architecture
 - Programming model
- Pros and cons of each controller

Many Different Controllers

- ❑ NOX/POX
- ❑ Ryu
- ❑ Floodlight
- ❑ OpenDaylight
- ❑ Pyretic
- ❑ Frenetic
- ❑ Procera
- ❑ RouteFlow
- ❑ Trema

Considerations

When choosing which controller to implement, some things to consider:

- Programming language
- Learning curve
- User base and community support
- Focus
 - Southbound API
 - Northbound API / “Policy Layer”
 - Support for Cloud (OpenStack)
 - Education, research, production, hybrid?

NOX: Overview

- 1st-gen OpenFlow controller
 - Open source, stable, widely used
- Two flavours
 - NOX Classic: C++/Python. No longer supported
 - NOX (new NOX)
 - C++ only
 - fast clean codebase
 - well maintained and supported

NOX: Characteristics

- Users implement control in C++
- Supports OpenFlow v1.0
 - a fork (CPqD) supports 1.1, 1.2, 1.3
- Programming model
 - controller registers for events
 - programmer writes event handlers

NOX: Why use it?

- Know C++
- Willing to use low-level facilities of OpenFlow
- Need good performance

POX: Overview

- NOX in Python
 - supports only v1.0
- Advantages
 - widely used, maintained, supported
 - relatively easy to read and write code
- Disadvantage
 - Performance not great

POX: Why use it?

- Know Python (or willing to learn)
- Not concerned about performance
- Rapid prototyping and experimentation
 - research, experimentation, demonstrations
 - learning concepts of SDN-OpenFlow

Ryu: Overview

- Open source Python controller
 - support OF 1.0, 1.2, 1.3, 1.4, vendor extensions
 - Works with OpenStack
- Aims to be an “OS” for SDN
- Advantages
 - supporting all versions of OpenFlow
 - support for OpenStack (cloud and datacenter)
- Disadvantage
 - performance

Floodlight: Overview

- Open source Java controller
 - support OF V1.0, 1.3, 1.4
 - extension of Beacon Java OF controllers
 - Maintained by Big Switch Networks
- Advantages
 - very good documentation
 - integration with REST API
 - production level performance (OpenStack, multi tenant clouds)
- Disadvantages
 - steep learning curve

Floodlight: Why use it?

- Know Java
- Require production level performance and support
- Use REST API to interact with the controller

OpenDaylight: Overview

- **Goal: common industry supported platform**
 - robust, extensible open source codebase
 - common abstractions for northbound capabilities
- **Advantages**
 - industry acceptance
 - integration with OpenStack and cloud applications
- **Disadvantages**
 - very complex
 - steep learning curve

OpenDaylight: Why use it?

- ❑ Know Java
- ❑ Require production level performance and support
- ❑ Require support with cloud applications, OpenStack
- ❑ Utilise modular functions
- ❑ Utilise vendor specific apps