


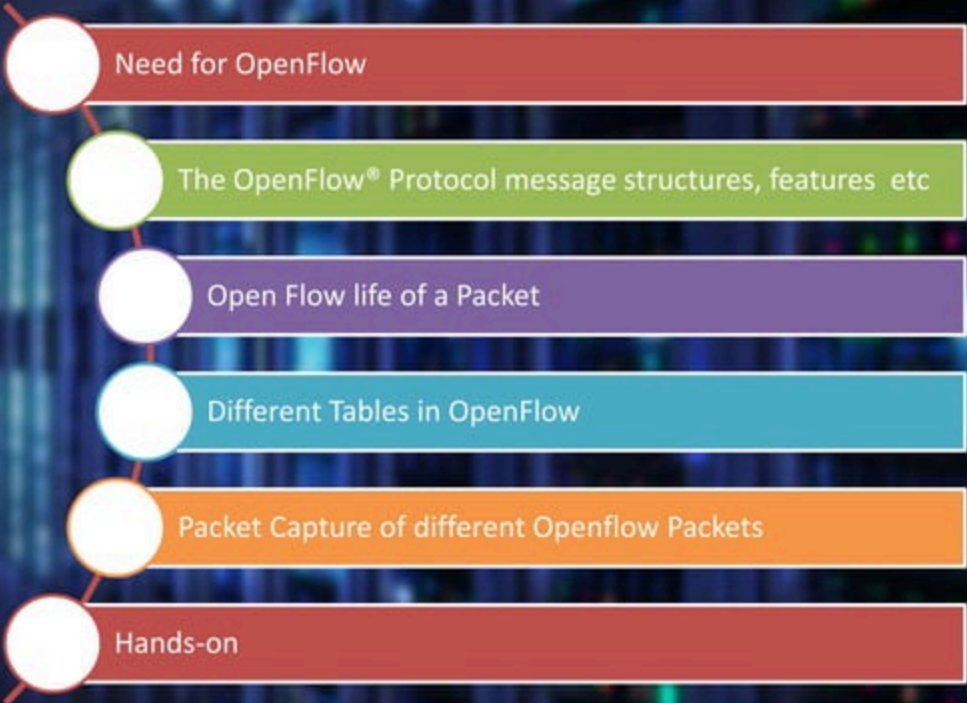
A blurred background image of a server room with rows of server racks and glowing lights.

# OpenFlow

S. Kingston Smiler  
kingstonsmiler@gmail.com

Three horizontal bars in blue, red, and green colors are positioned at the bottom of the slide.

# Course Objective



Need for OpenFlow


The OpenFlow® Protocol message structures, features etc

Open Flow life of a Packet

Different Tables in OpenFlow

Packet Capture of different Openflow Packets

Hands-on



# Problems

---

- 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 north bound and south bound interfaces, even though we have standard abstractions in the east / west bound interface with peer routers / switches.

# Need for OpenFlow

---

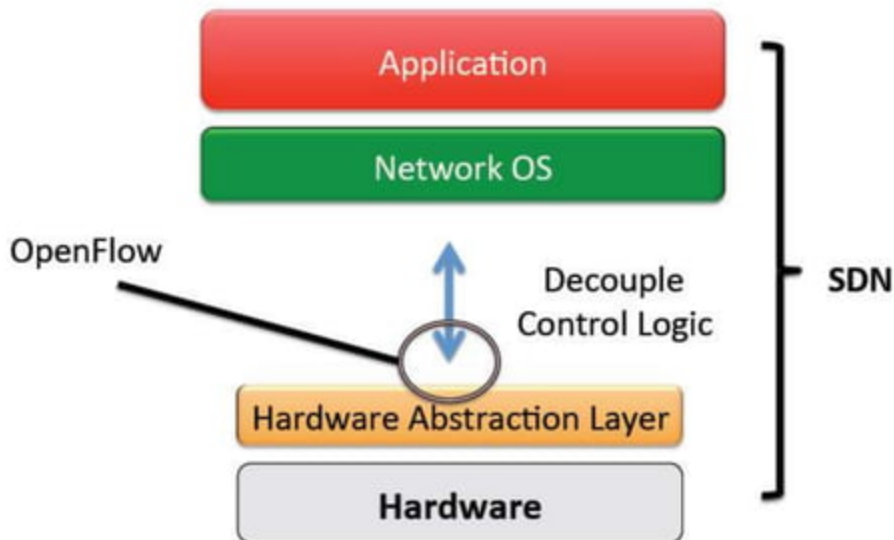
- Facilitate Innovation in Network
- Layered architecture with Standard Open Interfaces
- Independent innovation at each layer
- More accessibility since software can be easily developed by more vendors
- Speed-to-market – no hardware fabrication cycles
- More flexibility with programmability and ease of customization and integration with other software applications
- Fast upgrades
- Program a network vs Configure a network

# 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
-

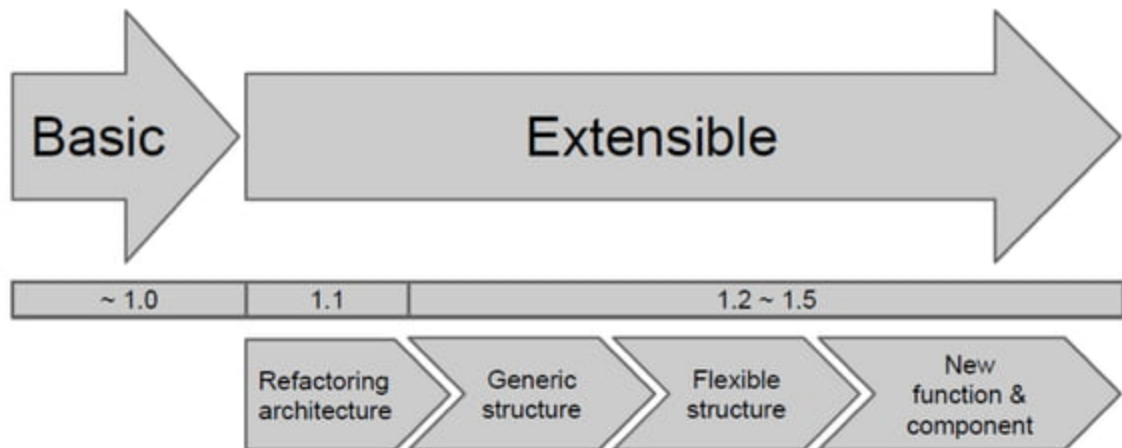
# SDN and Open Flow



- General Myth
  - SDN is Open Flow
- Reality
  - OpenFlow is an open API that provides a standard interface for programming the data plane switches

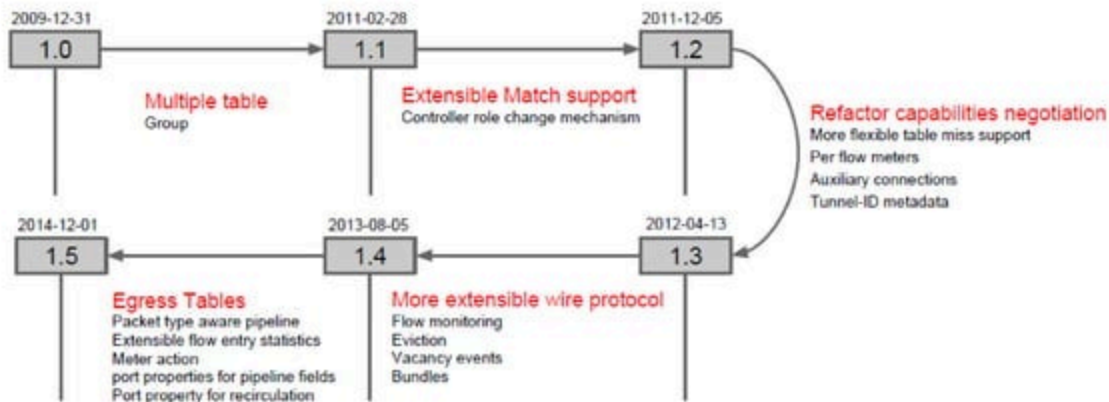
# OpenFlow Specification History

---

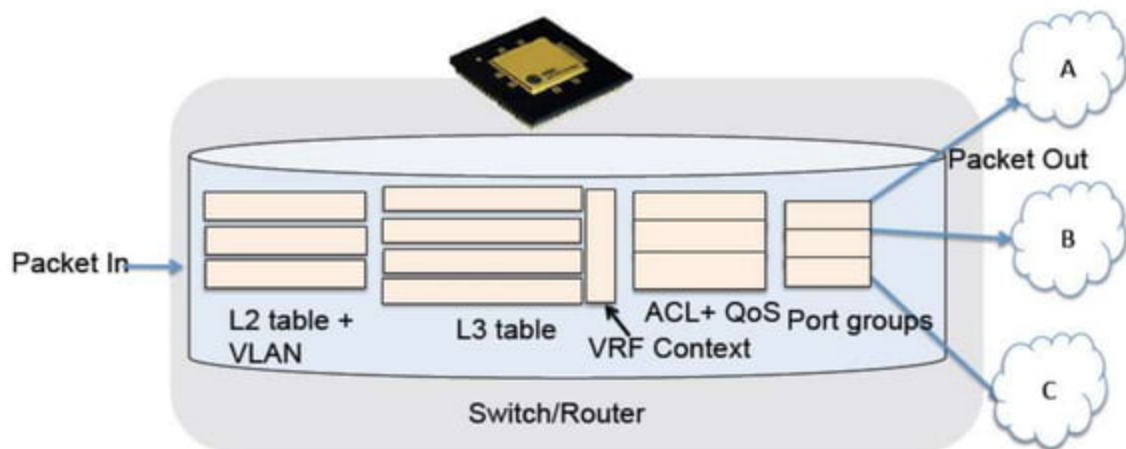




# OpenFlow Specification History

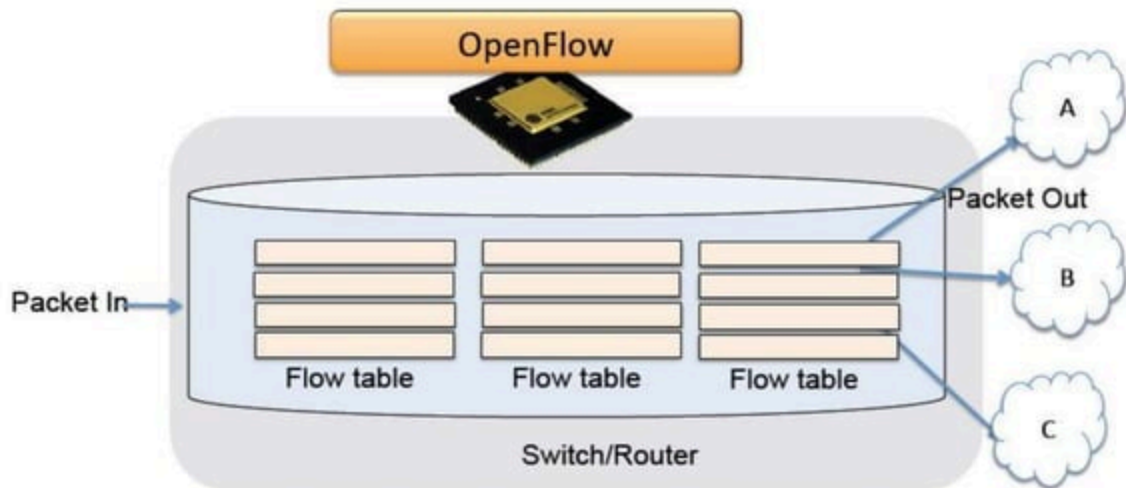


# Traditional Switch Forwarding

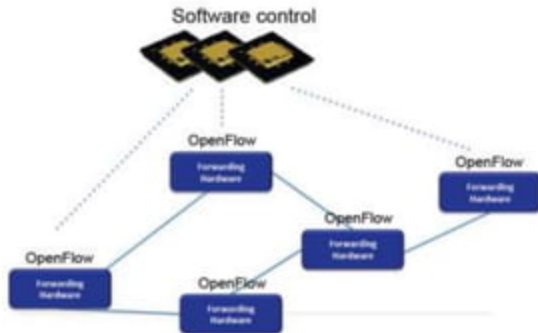


- Fixed function
- Often expose implementation details
- Non-standard/non-existent state management APIs

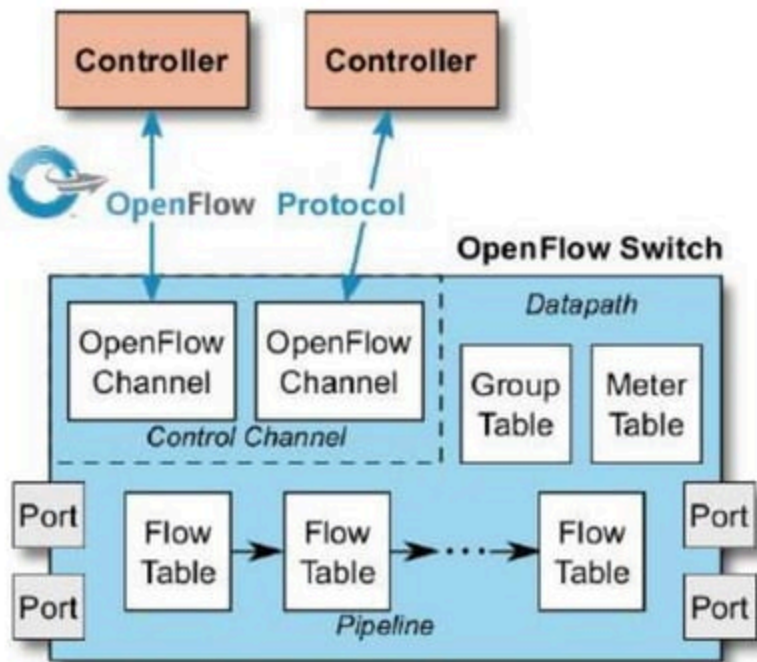
# OpenFlow Switch Forwarding



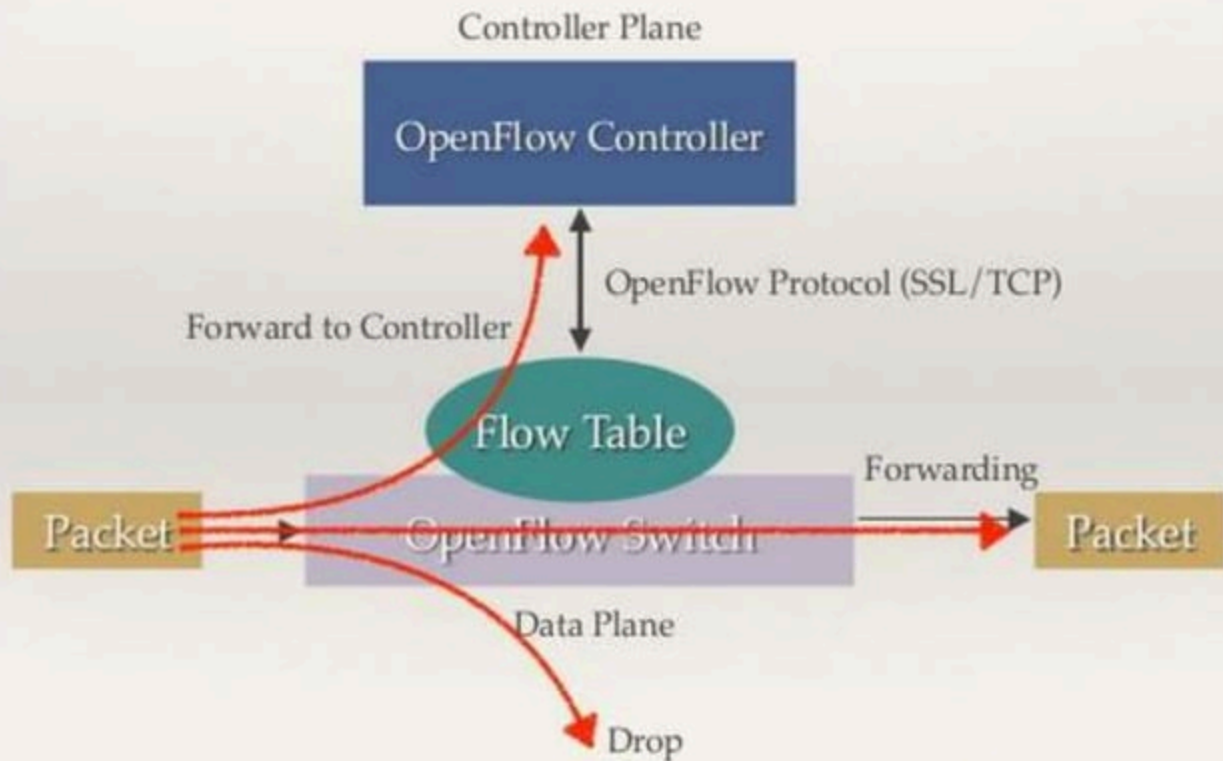
# Open Flow Illustration



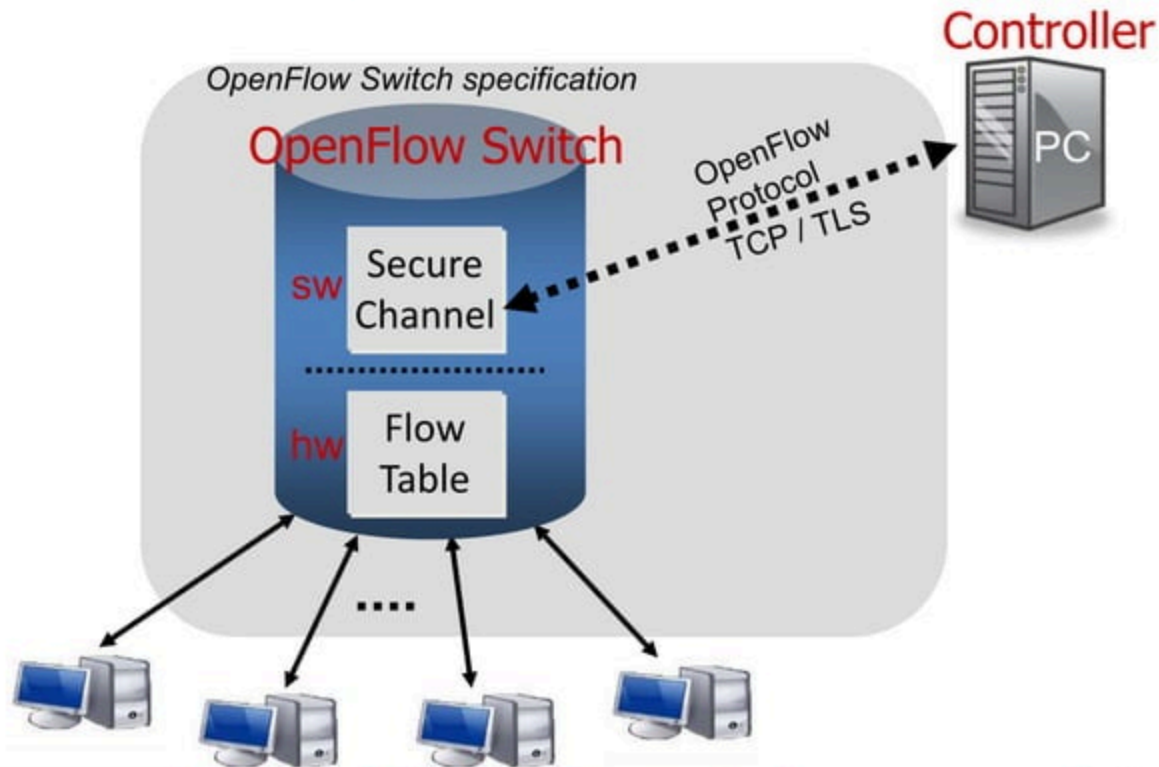
# Components of OpenFlow



# OpenFlow Packet Flow



# Components of OpenFlow Network



\* Figure From OpenFlow Switch Specification

# 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 north bound.
- 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 Controller

- 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 SC connection is a TLS/TCP connection. Switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key

# OpenFlow Messages

- Three Message Types
  - Controller to Switch
  - Asynchronous Message
  - Asymmetric Message

- Controller to Switch

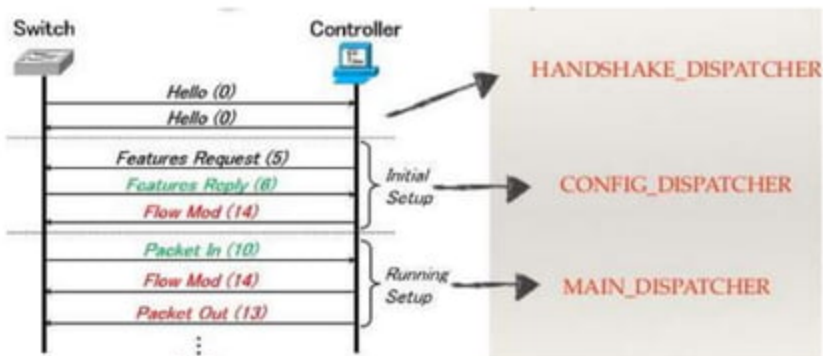
- Feature Request
- Configuration
- Modify State
- Packet Out etc.

- Asynchronous

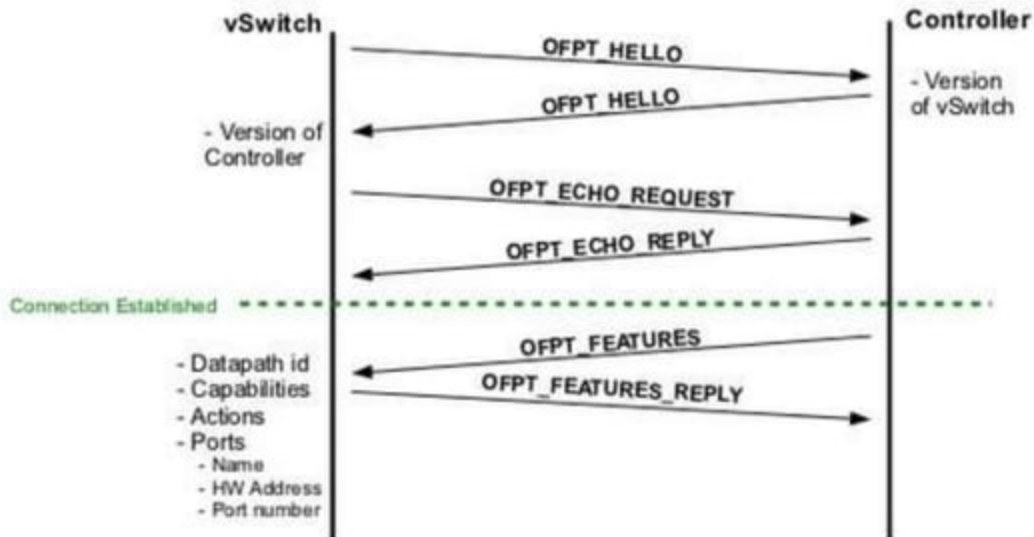
- Packet-in
- Flow Removed
- Port Status
- Error etc.

- ASymmetric

- Hello
- Echo
- Experimenter etc



# Initial Connection Setup



## OpenFlow – Hello ( From Switch to Controller)

The screenshot displays the Wireshark 1.12.0 interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Tools, Internals, and Help. The top toolbar contains various icons for file operations, capture control, and analysis. The main window is divided into three panes:

- Packet List:** Shows a list of captured packets. The selected packet is number 22, a Transmission Control Protocol (TCP) packet from 10.0.2.15 to 10.0.2.15, type OpenFlow, length 82 bytes.
- Packet Details:** Shows the hierarchical structure of the selected packet. It includes Ethernet II (Type: OpenFlow HELLO), Internet Protocol Version 4 (Source: 10.0.2.15, Destination: 10.0.2.15), and Transmission Control Protocol (Source Port: 59772, Destination Port: 6633, Seq: 1, Ack: 1, Len: 16).
- Packet Bytes:** Shows the raw data of the selected packet, displayed in hexadecimal and ASCII. The data is an OpenFlow HELLO message.

The status bar at the bottom indicates the current frame (Frame 22), the number of packets (1448), and the display filter (Default).

# OpenFlow – Hello ( From Controller to Switch )

The image shows a Wireshark packet capture interface. The top bar indicates the capture is from 'Loopback...'. The filter bar shows 'tcp.port==6633'. The packet list shows several TCP and OpenFlow packets. The selected packet (No. 27) is an OpenFlow Hello message from the controller to the switch. The packet details pane shows the following structure:

- Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 39772 (39772), Seq: 1, Ack: 17, Len: 16
- OpenFlow 1.3
  - Version: 1.3 (0x04)
  - Type: OFPT\_HELLO (0)
  - Length: 16
  - Transaction ID: 10
  - Element
    - Type: OFPHET\_VERSIONBITMAP (1)
    - Length: 8
    - Bitmap: 00000012

The packet bytes pane shows the raw data: 02 07 10 00 00 00 00 00 00 00 00 00 00 00 00 00.

# OpenFlow – Feature Request (From Controller to Switch)

The screenshot shows a Wireshark interface with a packet capture filter set to `tcp.port==6633`. The capture is from a loopback interface. The selected packet is number 29, an OpenFlow 1.3 FEATURES\_REQUEST packet. The packet details show it is an OFPT\_HELLO packet (Type: OFPT\_HELLO) with a length of 8 bytes. The transaction ID is 11. The packet bytes are displayed in hexadecimal and ASCII.

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

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

# OpenFlow – Feature Reply (From Switch to Controller)

The image shows a Wireshark packet capture window. The filter is set to `tcp.port==6633`. The packet list shows several packets, with packet 17 selected. The packet details pane shows the following information:

- Transmission Control Protocol, Src Port: 59772 (59772), Dst Port: 6633 (6633), Seq: 17, Ack: 25, Len: 32
- OpenFlow 1.3
  - Version: 1.3 (804)
  - Type: OFPT\_FEATURE\_REPLY (6)
  - Length: 32
  - Transaction ID: 11
  - datapath\_id: 0x0000000000000001
  - n\_buffers: 256
  - n\_tables: 254
  - auxiliary\_id: 0
  - Pad: 0
  - capabilities: 0x00000047
  - Reserved: 0x00000000

The packet bytes pane shows the raw data of the packet, including the OpenFlow header and the capabilities field.



# OpenFlow – Role Request (From Controller to Switch)

The image shows a Wireshark packet capture window. The filter is set to `tcp.port==6633`. The packet list shows several packets, with packet 46 selected. The packet details pane shows the structure of the OpenFlow 1.3 Role Request packet.

No.	Time	Source	Destination	Protocol	Length	Info
42	32.588471000	10.0.2.15	10.0.2.15	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
43	32.588486000	10.0.2.15	10.0.2.15	TCP	66	59774->6633 [ACK] Seq=17 Ack=25 Win=4832 Len=0 TSval=64516 TSecr=64516
44	32.588718000	10.0.2.15	10.0.2.15	OpenFlow	90	Type: OFPT_FEATURES_REPLY
45	32.588879000	10.0.2.15	10.0.2.15	TCP	66	6633->59772 [ACK] Seq=57 Ack=81 Win=4832 Len=0 TSval=64516 TSecr=64516
46	32.581374000	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

Frame 46: 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: 6633 (6633), Dst Port: 59772 (59772), Seq: 57, Ack: 81, Len: 24  
OpenFlow 1.3  
Version: 1.3 (0x04)  
Type: OFPT\_ROLE\_REQUEST (24)  
Length: 24  
Transaction ID: 3  
Role: OFPCR\_ROLE\_MASTER (0x00000002)  
Pad: 00000000  
Generation ID: 8x0000000000000000

0000 00 00 00 00 00 00 00 00 00 00 00 00 00 45 00 .....E  
0010 00 4c e1 00 40 00 00 41 0e 0a 00 02 0f 0a 00 .....L.:@.A.....  
0020 02 0f 19 e9 e9 7c 09 1e 1a bd 2a 00 27 f9 00 18 .....:..\*.....  
0030 00 56 18 5c 00 00 01 01 00 0a 00 00 fc 04 00 00 .....V.....

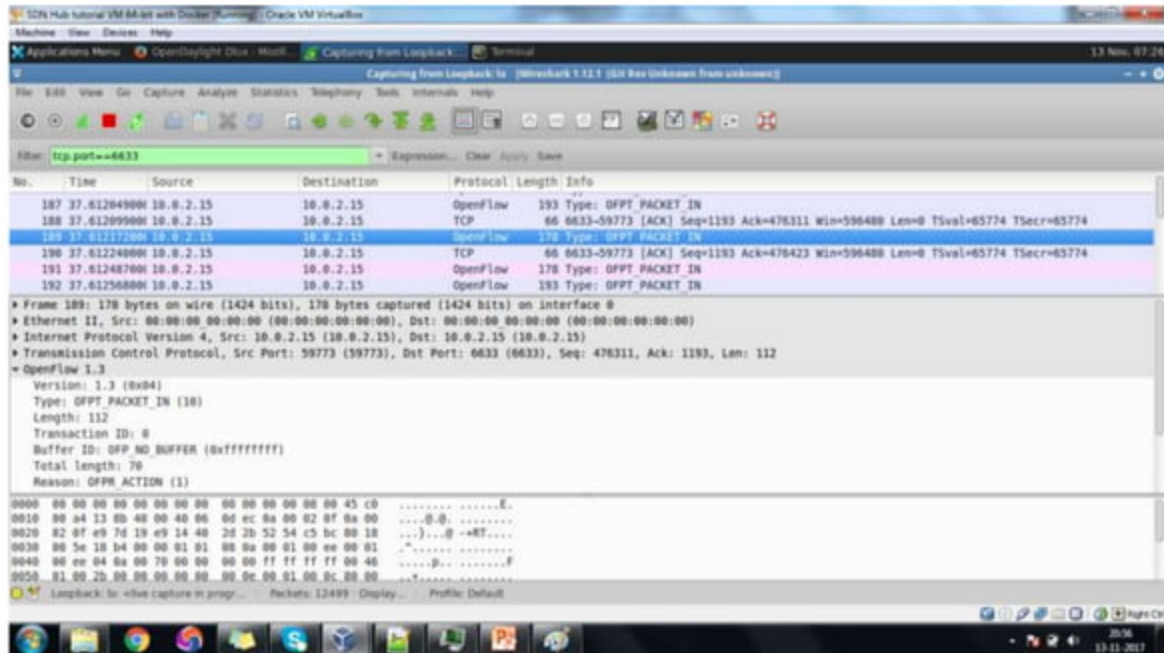
# OpenFlow – Role Reply (From Switch to Controller)

The image shows a Wireshark packet capture window. The filter is set to `tcp.port==6633`. The capture is from a network interface named `Loopback: lo`. The packet list shows several packets, with packet 48 selected. The packet details pane shows the following information:

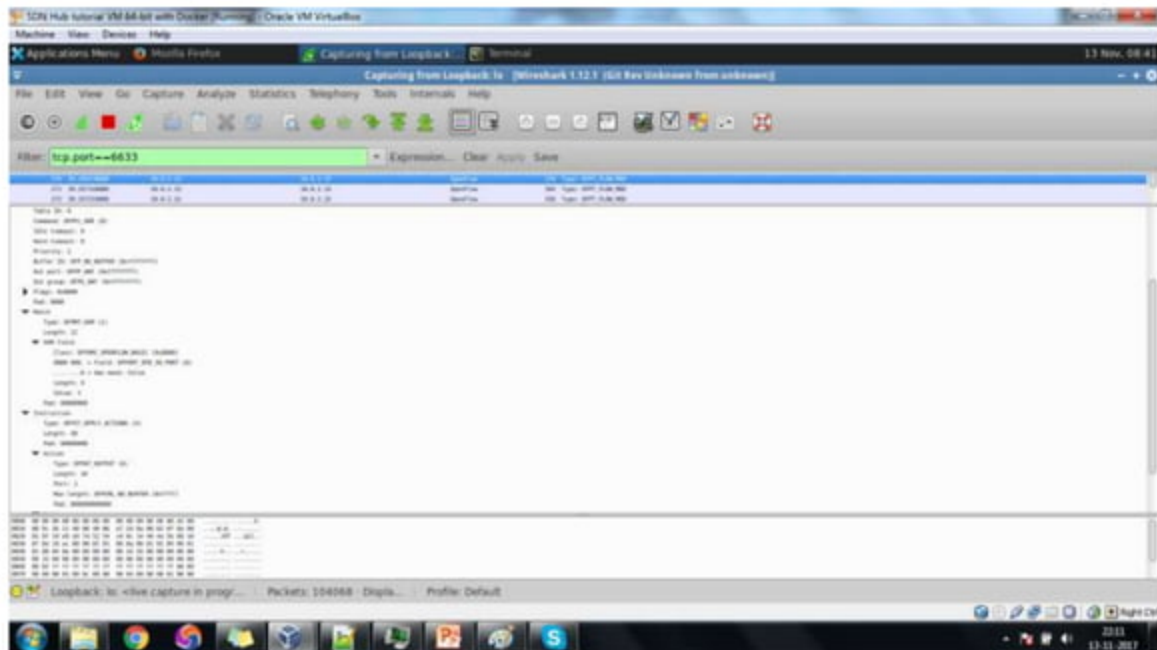
- Frame 48: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
- Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
- Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 10.0.2.15 (10.0.2.15)
- Transmission Control Protocol, 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

The packet bytes pane shows the raw data of the packet, including the Ethernet II header, IP header, TCP header, and the OpenFlow 1.3 Role Reply payload.

## OpenFlow – Packet IN (From Switch to Controller)



# OpenFlow – Flow Config (From Switch to Controller)



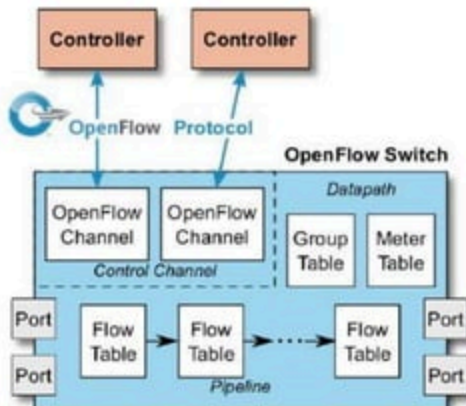
# OpenFlow Switch

- Types of the switches:

- Open-flow only
- Open-flow hybrid

- Major Components

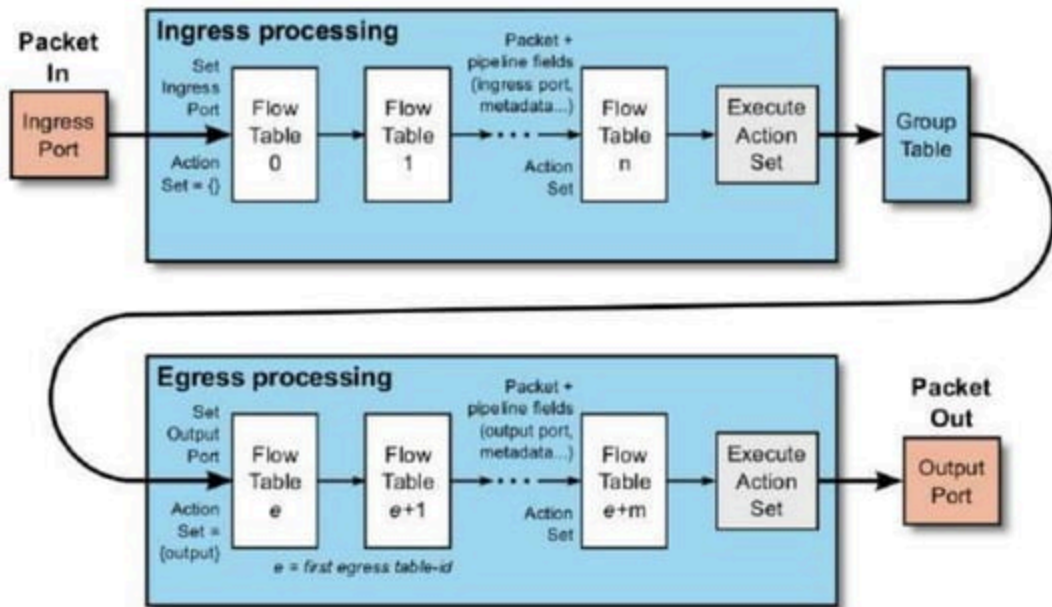
- Ports
- Flow Table
- Group Table
- Meter Table



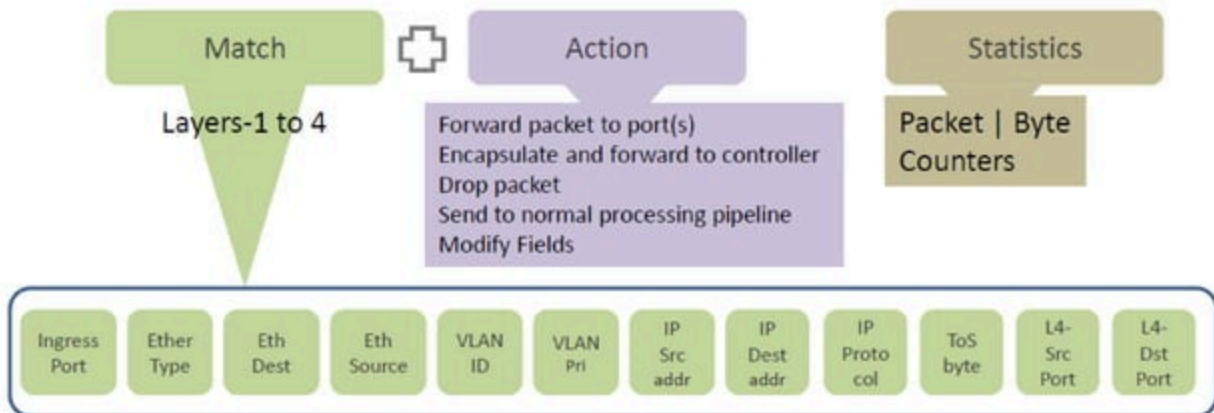
# OpenFlow Ports

- 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

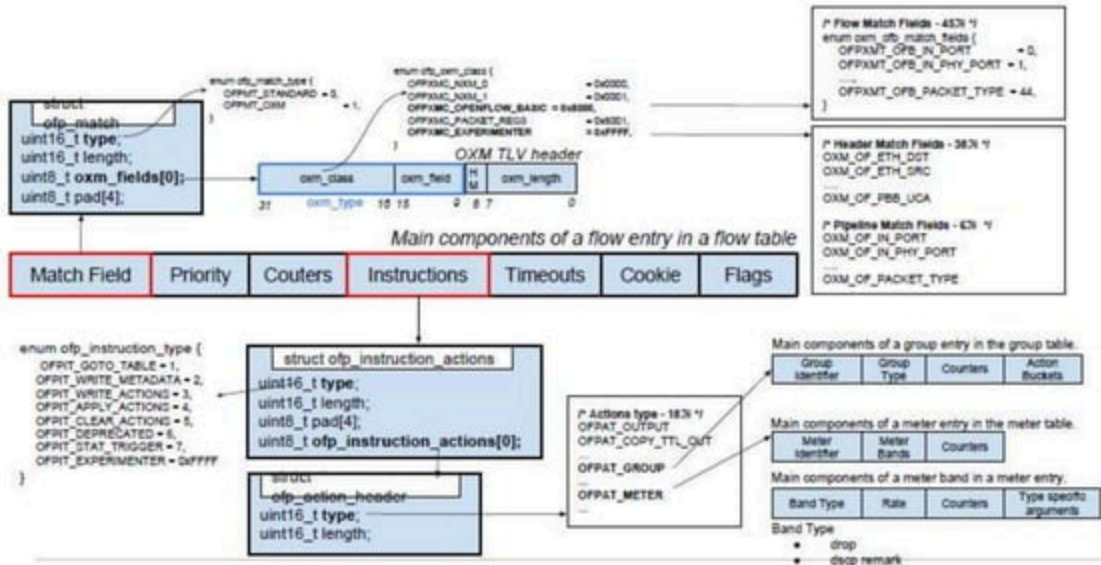
# Pipeline Processing



# Flow Table Components







\* Figure From OpenFlow Switch Specification

# Openflow Table Instructions

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

- Instructions are executed when a packet matches entry.
- Instruction result can
  - Change the packet
  - Action set
  - Pipeline processing etc.
- 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 & Action List

## ■ Action Set

- Action set is associated with each packet. FE 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

- "Apply-action" , "packet-out" messages include 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 open flow 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

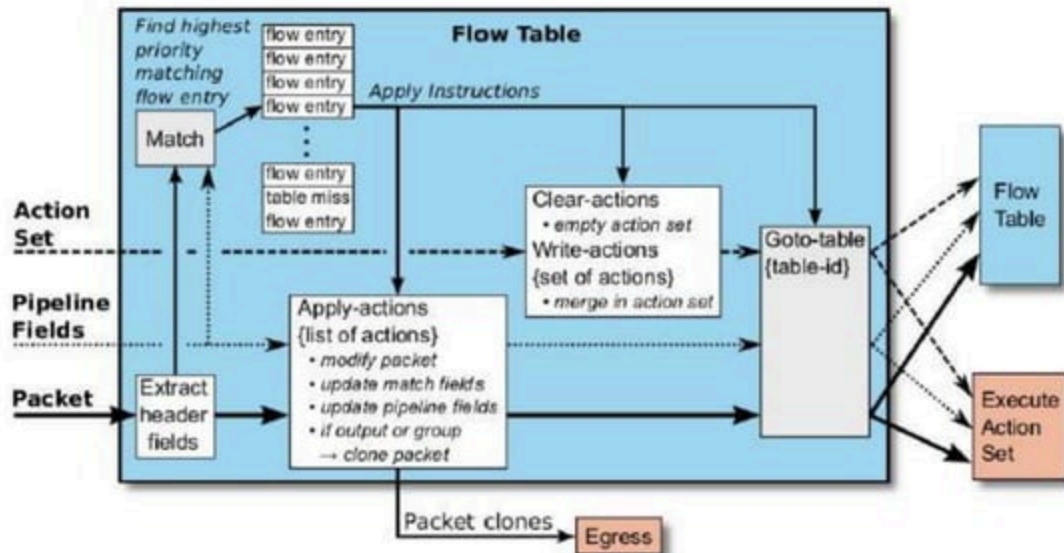
## Flow Entry Timeouts

- 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 (Contd...)

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

# Instruction Execution in Flow Table





# Group Table

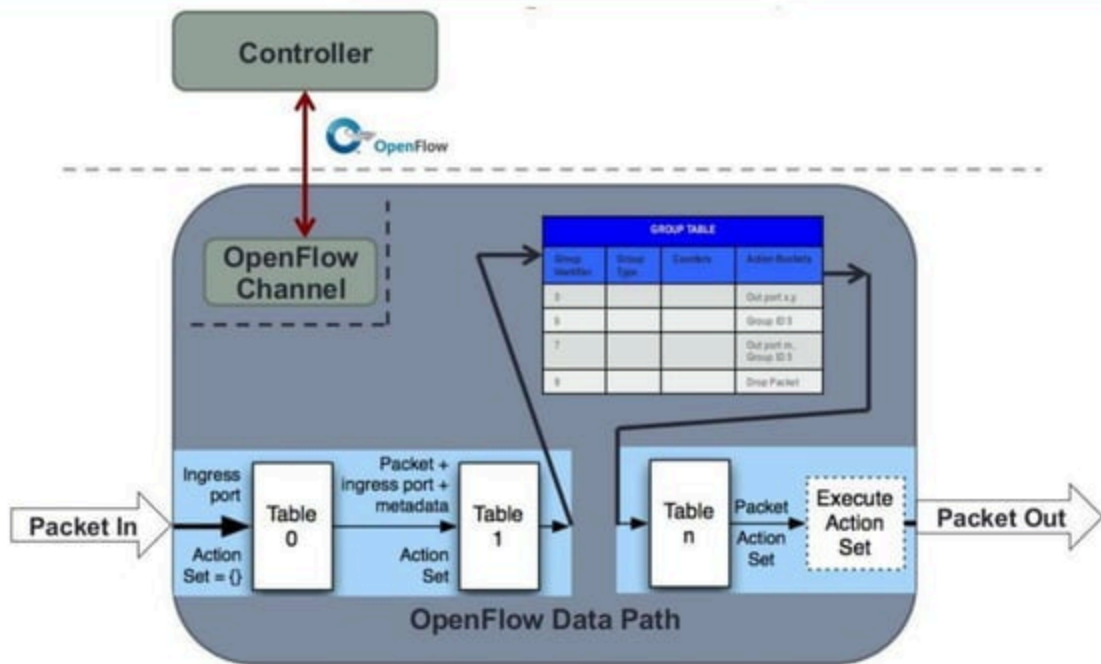
- Additional method for forwarding to a group of entries.
- Comprises of Group ID, Group Type, Counters, Action buckets (each action bucket contains a set of actions to be executed)
- Group Types:
  - All
    - Execute all buckets in a group
    - Used mainly for multicast and broadcast – fwd a pkt on 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



# Group Table

Group Identifier	Group Type	Counters	Action Buckets
5			Out port x,y
15			Out port a
6			Group ID 5
7			Out port m, Group ID 15
9			Drop Packet

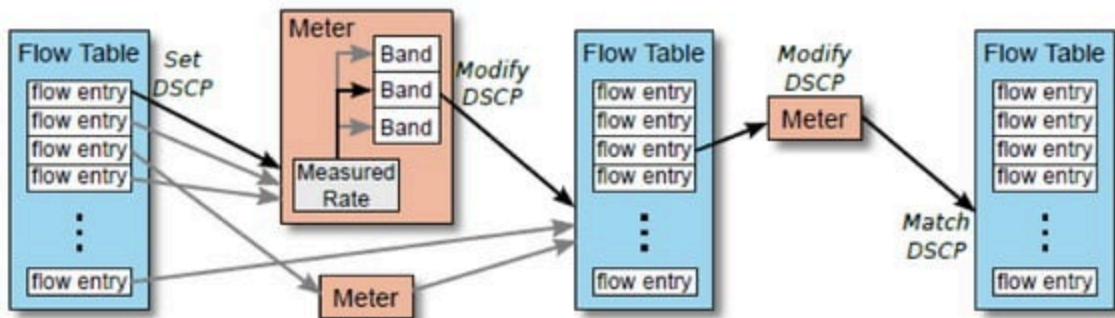
# Instruction Execution in Group Table



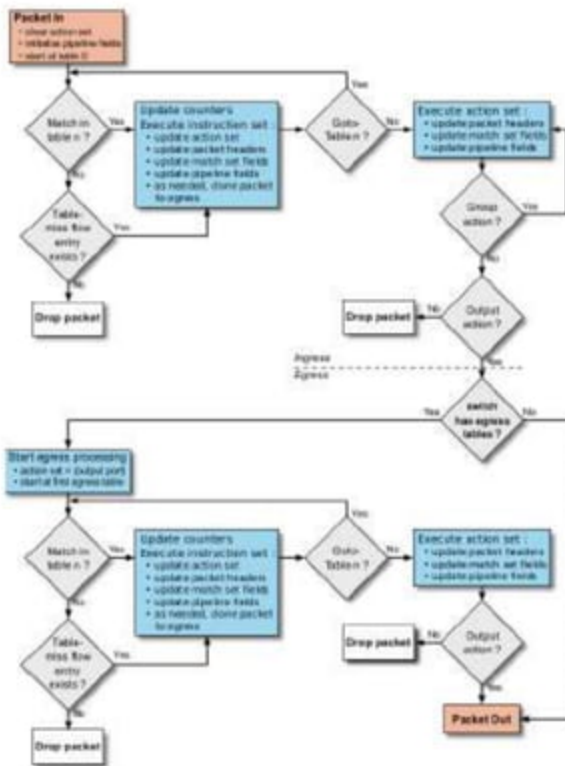
# Meter Table

- 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

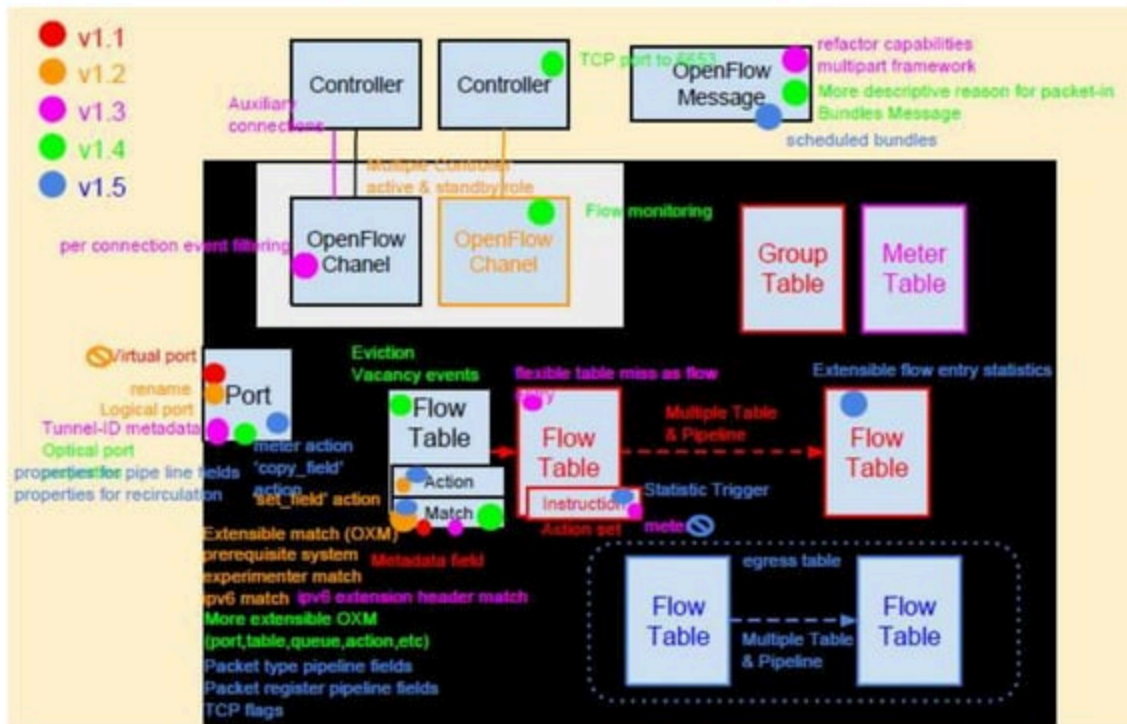


# Packet Flow in OpenFlow Switch



\* Figure From OpenFlow Switch Specification

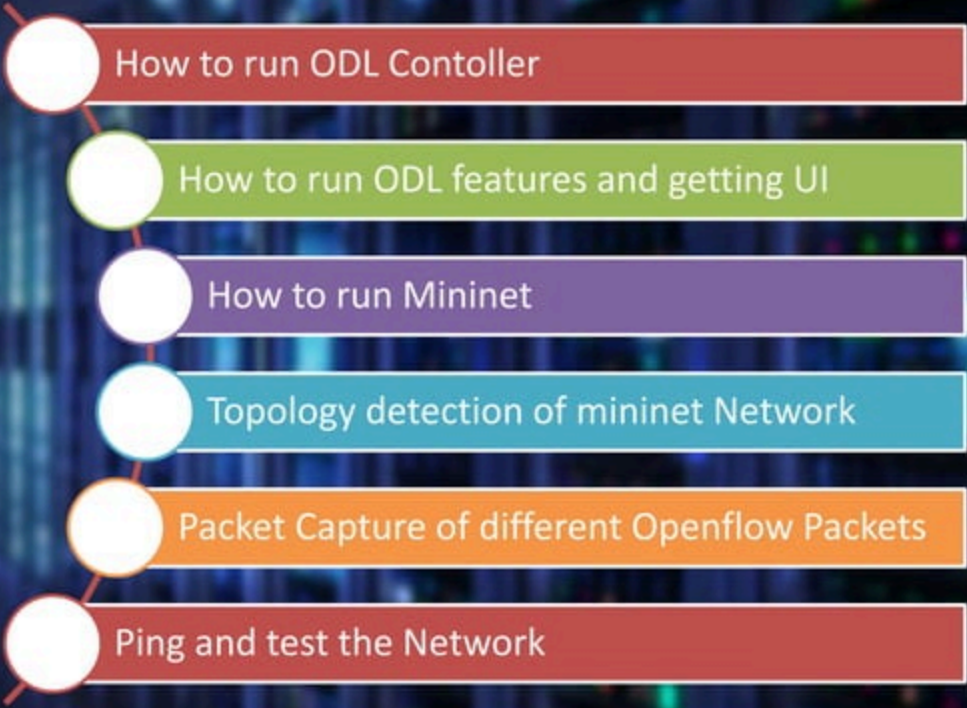
# OpenFlow Features Vs Specification





## LAB Exercise

# Course Objective



How to run ODL Contoller


How to run ODL features and getting UI

How to run Mininet

Topology detection of mininet Network

Packet Capture of different Openflow Packets

Ping and test the Network



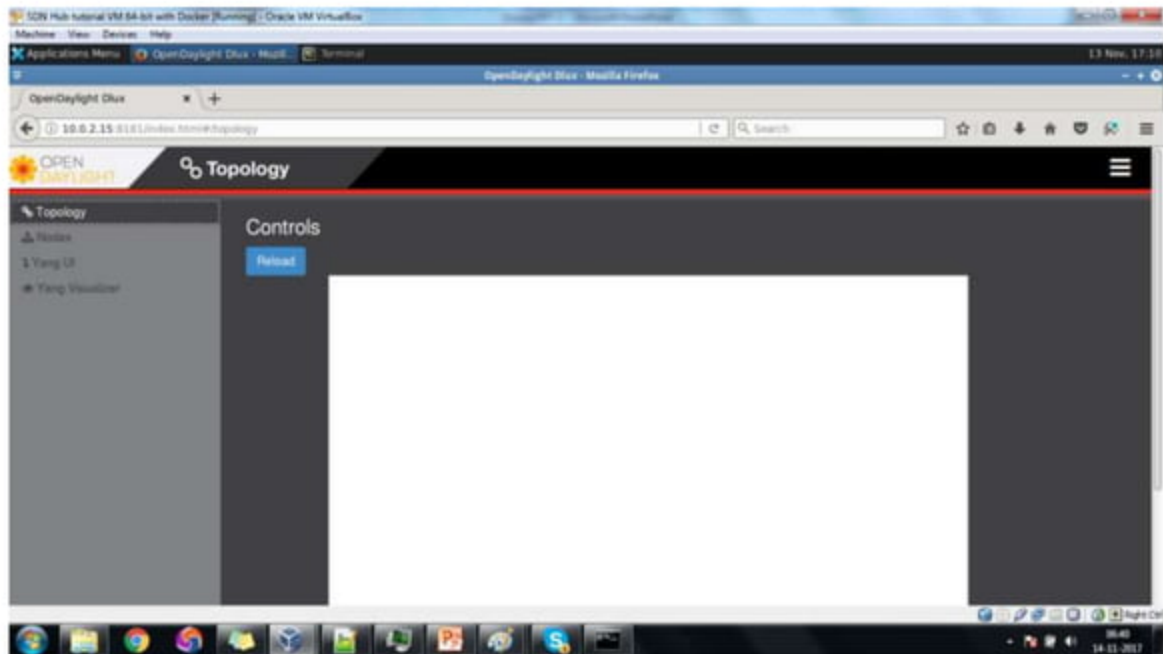




# How to start OpenDayLight Controller

---


- Run the given VM in Virtual Box
  - Goto distribution-karaf-0.4.4-Beryllium-SR4/
    - `cd distribution-karaf-0.4.4-Beryllium-SR4/`
  - Run `./bin/karaf`
  - Install the basic required karaf features/bundles for the lab exercise
    - `feature:install odl-restconf-all odl-l2switch-switch odl-mdsal-apidocs odl-dlux-all`
    - `feature:install odl-restconf-noauth odl-netconf-connector-all`
  - Go to browser and open the page <http://10.0.2.15:8181/index.html>
  - Login with admin, admin
-

# OpenDaylight Controller UI



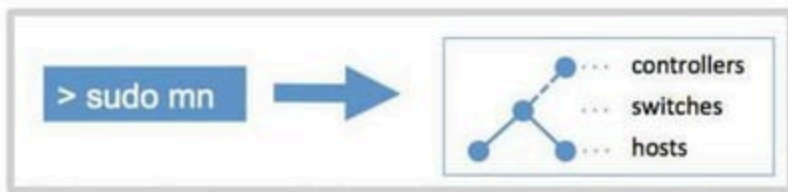


## MININET & OpenVSwitch (OVS)



# Mininet

- Mininet creates a realistic OpenFlow network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single command
- `sudo mn --topo linear,3 --mac --controller=remote,ip=10.0.2.15,port=6633 -  
-switch ovs,protocols=OpenFlow13`
- `sudo ovs-ofctl -O OpenFlow13 dump-flows s2`
- `sudo ovs-vsctl show`

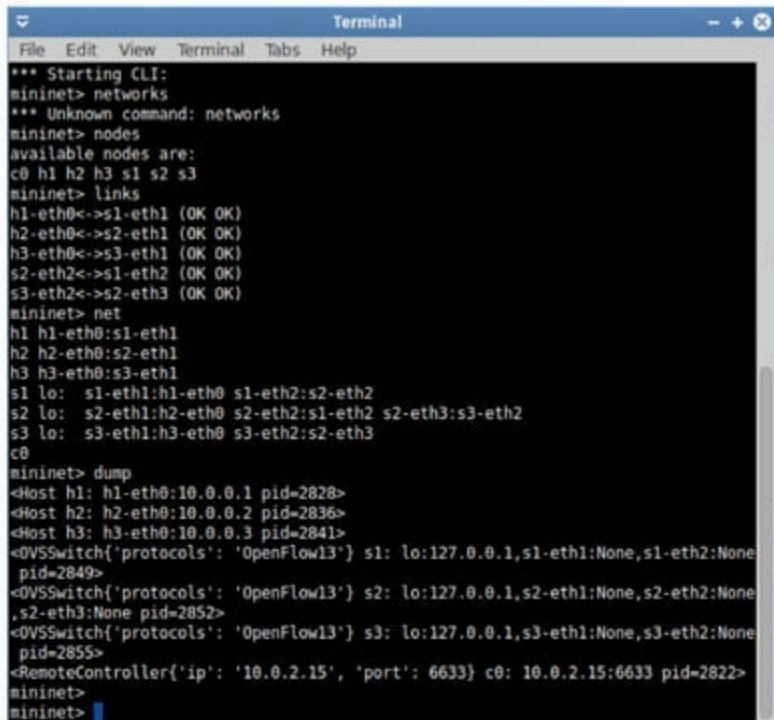


# Mininet - Installation

---

- Option 1: Mininet VM Installation -  
<http://mininet.org/download/#option-1-mininet-vm-installation-easy-recommended>
  - Option 2: Native Installation from Source -  
<http://mininet.org/download/#option-2-native-installation-from-source>
  - Option 3: Installation from Packages -  
<http://mininet.org/download/#option-3-installation-from-packages>
  - Option 4: Upgrading an existing Mininet Installation -  
<http://mininet.org/download/#option-4-upgrading-an-existing-mininet-installation>
-

# Mininet – Some show commands

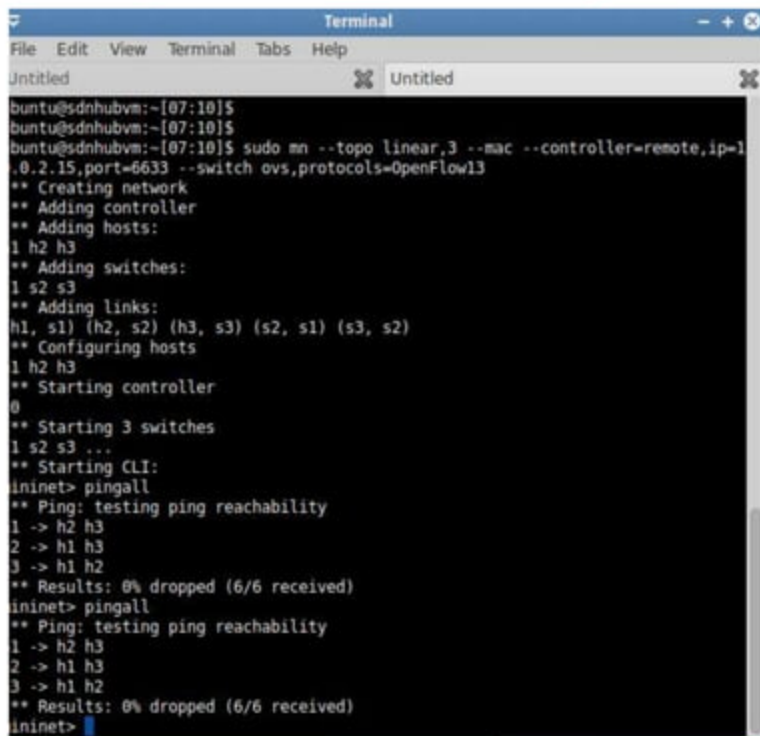


```
*** Starting CLI:
mininet> networks
*** Unknown command: networks
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1 s2 s3
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s2-eth1 (OK OK)
h3-eth0<->s3-eth1 (OK OK)
s2-eth2<->s1-eth2 (OK OK)
s3-eth2<->s2-eth3 (OK OK)
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo: s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo: s3-eth1:h3-eth0 s3-eth2:s2-eth3
c0
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=2828>
<Host h2: h2-eth0:10.0.0.2 pid=2836>
<Host h3: h3-eth0:10.0.0.3 pid=2841>
<OVSSwitch{'protocols': 'OpenFlow13'} s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None
pid=2849>
<OVSSwitch{'protocols': 'OpenFlow13'} s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None
s2-eth3:None pid=2852>
<OVSSwitch{'protocols': 'OpenFlow13'} s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None
pid=2855>
<RemoteController{'ip': '10.0.2.15', 'port': 6633} c0: 10.0.2.15:6633 pid=2822>
mininet>
mininet>
```

## Mininet – Some show commands

```
File Edit View Terminal Tabs Help
RemoteController(ip: '10.0.2.15', 'port': 6033) c0: 10.0.2.15:6033 pid=2822-
mininet>
mininet>
mininet> dpctl show
** s1
-----
017-11-13T14:45:16Z[0000]vconn[MARN|unix:/var/run/openvswitch/s1.mgmt: version
negotiation failed (we support version 0x01, peer supports version 0x04)
vs-ofctl: s1: failed to connect to socket (Broken pipe)
** s2
-----
017-11-13T14:45:16Z[0000]vconn[MARN|unix:/var/run/openvswitch/s2.mgmt: version
negotiation failed (we support version 0x01, peer supports version 0x04)
vs-ofctl: s2: failed to connect to socket (Broken pipe)
** s3
-----
017-11-13T14:45:16Z[0000]vconn[MARN|unix:/var/run/openvswitch/s3.mgmt: version
negotiation failed (we support version 0x01, peer supports version 0x04)
vs-ofctl: s3: failed to connect to socket (Broken pipe)
mininet> ping h1 h2
** Unknown command: ping h1 h2
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
from 10.0.0.1 icmp_seq=1 Destination Host Unreachable
from 10.0.0.1 icmp_seq=2 Destination Host Unreachable
from 10.0.0.1 icmp_seq=3 Destination Host Unreachable
^C
-- 10.0.0.2 ping statistics --
0 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2999ms
mininet>
mininet>
mininet> xterm h1
mininet> xterm s1
mininet>
```

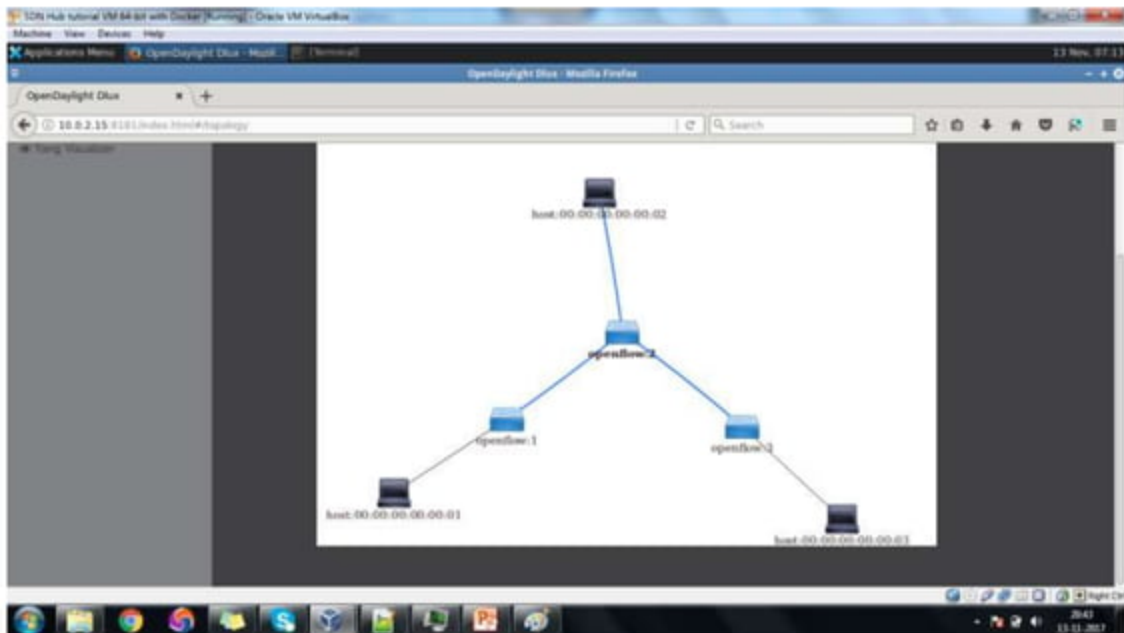
# Mininet – Some show commands



```
Terminal
File Edit View Terminal Tabs Help
Untitled Untitled
buntu@sdnhubvm:~[07:10]$
buntu@sdnhubvm:~[07:10]$
buntu@sdnhubvm:~[07:10]$ sudo mn --topo linear,3 --mac --controller=remote,ip=1
.0.2.15,port=6633 --switch ovs,protocols=OpenFlow13
** Creating network
** Adding controller
** Adding hosts:
1 h2 h3
** Adding switches:
1 s2 s3
** Adding links:
h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
** Configuring hosts
1 h2 h3
** Starting controller
0
** Starting 3 switches
1 s2 s3 ...
** Starting CLI:
mininet> pingall
** Ping: testing ping reachability
1 -> h2 h3
2 -> h1 h3
3 -> h1 h2
** Results: 0% dropped (6/6 received)
mininet> pingall
** Ping: testing ping reachability
1 -> h2 h3
2 -> h1 h3
3 -> h1 h2
** Results: 0% dropped (6/6 received)
mininet>
```



# Mininet – Some show commands





Thank you

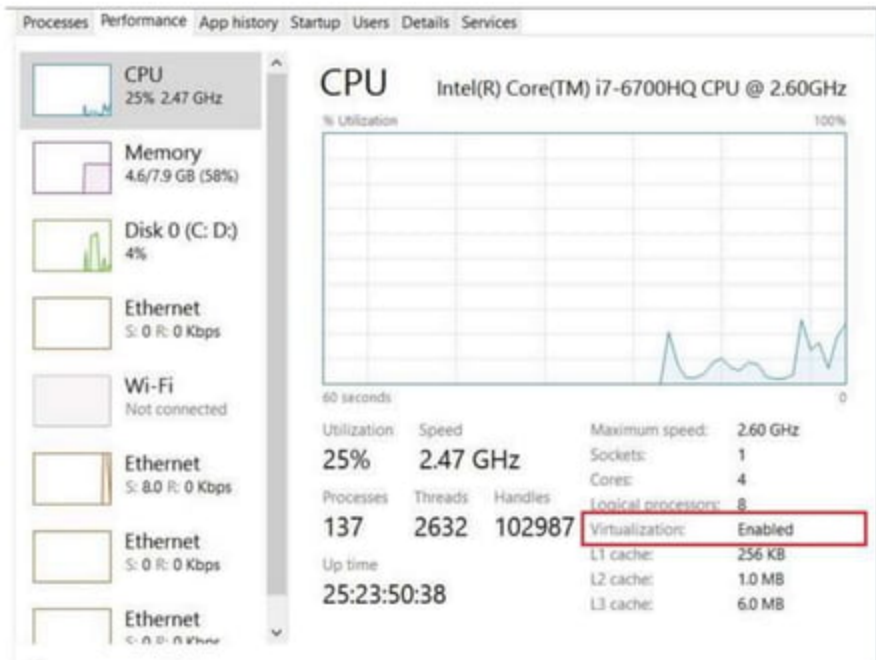




## Backup Slides



# How to check VT-x enabled in Windows 8?



# How to check VT-x enabled in Windows 7?

- Install the Microsoft® Hardware-Assisted Virtualization Detection Tool from <https://www.microsoft.com/en-us/download/details.aspx?id=592>
- Run it and see whether you get this output

