
Oct. 17, 2021
Outline

• SDN Basic
• From Legacy Network to SDN
• How SDN Works
• OpenFlow Overview
  - OpenFlow Switch
  - OpenFlow Controller
  - The Controller-Switch Secure Channel

Ref:
COS 597E - Software Defined Networking, Jennifer Rexford, Princeton University
COMS E6998-10 2014 - Software Defined Networking, Li Erran Li, Columbia University
SDN Basic
## SDN History

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
<th>Products and Initiatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>~2004</td>
<td>Research on new management paradigms</td>
<td>RCP, 4D [Princeton, CMU, …] SANE, Ethane [Stanford/Berkeley]</td>
</tr>
<tr>
<td>2011</td>
<td>Open Networking Foundation</td>
<td>Board: Google, Yahoo, Verizon, Deutsche Telekom, Microsoft, Facebook, NTT Members: Cisco, Juniper, HP, Dell, Broadcom, IBM, …</td>
</tr>
<tr>
<td>Current</td>
<td></td>
<td>Google is using SDN in its private WAN B4 Many commercialized SDN products</td>
</tr>
</tbody>
</table>
SDN Background

• To solve the limitations faced by the traditional physical network environment operation architecture.
• Software Defined Networking (SDN) architecture proposed to significantly improve the flexibility, efficiency and cost reduction of network operations.
• SDN has become the focus of next-generation network technology development.
• Many index companies have been actively involved.
Software Defined Networking

• A new network architecture:
  - Using the OpenFlow protocol, the control plane of the router is separated from the data plane and implemented in software.

• This architecture allows network administrators to re-plan the network in a centrally controlled manner without changing the hardware.

• It provides a new way to control network traffic and provides a good platform for core network and application innovation.

• Three major factors that make SDN important to the enterprise:
  - automation
  - rapid deployment
  - simple network management.
Software Defined Networks

control plane: distributed algorithms

data plane: packet processing
Software Defined Networks

decouple control and data planes
Software Defined Networks

decouple control and data planes by providing open standard API
Simple, Open Data-Plane API

- Prioritized list of rules
  - Pattern: match packet header bits
  - Actions: drop, forward, modify, send 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
(Logically) Centralized Controller
Protocols ➔ Applications
Seamless Mobility

• See host sending traffic at new location
• Modify rules to reroute the traffic
Server Load Balancing

- Pre-install load-balancing policy
- Split traffic based on source IP

src=0*, dst=1.2.3.4

src=1*, dst=1.2.3.4
Definition of SDN

The separate of control plane from data plane

• **Data Plane**: processing and delivery of packets
  - Based on state in routers and endpoints
    ✓ IP
    ✓ TCP
    ✓ Ethernet, etc.
  - Fast timescales
    ✓ per-packet

• **Control Plane**: establishing the state in routers
  - Determines how and where packets are forwarded
    ✓ Routing
    ✓ traffic engineering
    ✓ firewall state, etc.
  - Slow time-scales
    ✓ per control event
Example SDN Applications

• Seamless mobility and migration
• Server load balancing
• Dynamic access control
• Using multiple wireless access points
• Energy-efficient networking
• Adaptive traffic monitoring
• Denial-of-Service attack detection
• Network virtualization

See http://www.openflow.org/videos/
**SDN Concept**

- SDN separates Control and Data plane functions

![SDN Diagram](source "Understanding L3 Switch", Netmanias Talk, 2011/11/09)

资 料 來 源：Korea,Postech,Department of Computer Science and Engineering,James Won-Ki Hong：Software Defined Networking — Introduction to SDN&Openflow
SDN Concept

- **Separates control plane and data plane entities**
  - Network intelligence and state are logically centralized
  - The underlying network infrastructure is abstracted from the applications

- **Execute or run control plane software on general purpose hardware**
  - De-couple from specific networking hardware
  - Use commodity computers

- **Have programmable data planes**
  - Maintain, control and program data plane state from a central entity

- **An architecture to control not only a networking device but an entire network**
  - Similar to existing Network Management System (NMS), but more powerful

**Control Software (SW)**
- Control SW operates on view of network
- Control SW is not a distributed system
  - Abstraction hides details of distributed states
From Legacy Network to SDN
Megatrend in the computer industry

Specialized Applications
Specialized Operating System
Specialized Hardware

Vertically integrated
Closed, proprietary
Slow innovation
Small industry

--- Open Interface ---

Windows (OS) or Linux or Mac OS

--- Open Interface ---

Horizontal
Open interfaces
Rapid innovation
Huge industry

Source: Nick McKeown, Stanford
The same trend in the network industry

Vertically integrated
Closed, proprietary
Slow innovation

Specialized Features
Specialized Control Plane
Specialized Hardware

App or App or App or App

Control Plane or Control Plane or Control Plane

Open Interface

Merchant Switching Chips

Horizontal
Open interfaces
Rapid innovation

Source: Nick McKeown, Stanford
Traditional Network Data Center
The topology of a traditional network data center
The Fat tree topology of Traditional Network
The current network architecture is inadequate

• Today's network architecture is a three-tier architecture built on the Spanning Tree Protocol (STP) that delivers packets over a variety of transport protocols.

• However, with the increasing demand for cloud application services and huge amounts of data, the routing tables of the Internet have become more and more complex, which has caused many problems in the current network architecture and is becoming more and more inadequate.

• In order to implement various network protocols, switches or routers must constantly split and reassemble packets, resulting in poor transmission efficiency and ineffective network bandwidth.
The current network architecture is inadequate

- When network administrators needed, use the command-line interface (CLI) settings for each switch or router.
  - Troublesome, high risk in manually setting one by one, easy to cause network service failure.

- Network management software is difficult to be compatible with each other.
Traditional Network vs SDN
Traditional Network vs SDN

• Ethernet network use Spanning Tree Protocol (STP) to forward frame.
  - IEEE 802.1D
  - Avoid loop and ARP storm
  - Analysis and decide which port can transmit
  - Single path routing

• STP
**Traditional Network vs SDN**

- Load balancing
  - Achieve higher bandwidth utilization
  - Balancing the traffic load
  - Static load balancing
  - Dynamic load balancing

![Traditional load balancing model](image1)

![SDN based load balancing model](image2)
SDN is a new generation network concept and architecture

- The control plane of the network is completely independent of the data plane.
- Centrally manage data traffic behavior across the network through the Controller software.
- The Controller software provides a application programming interface (API) for further integration with other upper-layer devices such as VMs.
- With the API, users can develop a variety of additional services on the Controller, such as Firewall, IDP.
- DPI (Deep Packet Inspection), LB (Load Balance), Schedule ..., etc., can be deployed in a unified manner to provide more diversified service projects for enterprises.
SDN Architecture (1/4)
SDN Architecture (2/4)

Automation and Orchestration

Software Ecosystem:
Load balancing, security, aggregation, monitoring, Performance mgmt. etc.

Controller

Other API
Open Flow

Physical Switches

Virtual Switches

Controller

Open Flow

Other API

1. Network Devices
2. Southbound
3. Controllers
4. Northbound
5. Services
6. Automation
**SDN Architecture (3/4)**

- **Network Devices**: switch, router, virtual switch, or abstract forwarding plane (Forwarding/Data Plane)
  - All forwarding rules are stored in the network device, and the user data packets are processed and forwarded here.
  - The network device receives the command sent by the controller through the southbound interface, and also actively reports the event to the controller through the southbound interface.

- **Southbound Interface**: between the control plane and the data forwarding layer
  - The traditional network exists in the private code of each device vendor and is not standardized.
  - In SDN the southbound interface is standardized, such as the Openflow standard interface.

- **Controller**:
  - The core elements of the SDN network provide up to the application's programming interface and down control of the hardware.
  - Usually run on a separate server, such as an x86 Linux server or Windows server.
SDN Architecture (4/4)

• Northbound Interface:
  - In the traditional network, the northbound interface refers to the interface between the switch control plane and the network management software.
  - In the SDN architecture, it refers to the interface between the controller and the application.

• Service:
  - Control and manage the network in the form of software applications, such as: Load Balancing, Security, Monitoring (including congestion and latency, network performance management and detection), LLDP (topology detection) and other functions.

• Automation: Automation is the packaging and integration of applications.
  - It usually comes with Orchestration, such as including multiple applications and services in a system management framework, and regularly collecting device line load through the controller.
SDN Scheme — Advantages

• **Higher automation** and reduces the misconfiguration of enterprises caused by humans.

• With SDN, customers only need to select the applications and necessary resources they want to run in the cloud, and the control plane intuitively deploys services using the optimal configuration of compute, storage, and network resources.

• **Quickly deploy** and **scale** your application can make a business or ruin a business.

• In addition to making employees easy to access, SDN can quickly respond to changing business and reduce the time it takes for new products to enter the market.

• SDN will greatly change the way the network infrastructure is configured and managed.
  - By separating the control functions from the rest of the network, SDN allows IT teams to manage the network environment in a bird's eye view of the business so that each business do not operate in isolation.
SDN Scheme — benefit

- Developable applications make network data traffic more flexible and bandwidth usage more efficient.
- Equivalent to traffic engineering and know the status of real-time traffic.
- Dynamically change the traffic path based on bandwidth usage to increase network usage.
- Can be added to the schedule for flexible use.
- Reduce the cost of maintenance manpower or equipment.
- Uniform control, easy to operate and manage.
- Improve the speed of obstacle removal.
- Centralized management, single inspection.
- Unlimited equipment brand, unified operation mode.
- The same standard, across the label restrictions.
- Flexible and variable value-added development space.
- It can integrate future FW, IDP (Intrusion Detection and Defense), DPI, VM, LB, Schedule, etc. to provide diversified services.
SDN Scheme — Misunderstanding (1/2)

• Like any new technology, as long as SDN exists, there must be people who argue the toss.
  - For any business, you want to understand the truth behind the biggest misunderstandings before deploying an SDN solution.

• SDN is not suitable for small data centers
  People tend to think that it is only suitable for large data centers (that is, data centers that provide public, private, and hybrid cloud services) when SDN is mentioned. Although these larger providers are early adopters, in fact, SDN is beneficial for all levels of data centers. Not only does it make configuration, management, and monitoring tasks simple, it also greatly reduces the burden on the IT department, which is the perfect choice for small companies with a lean team.

• SDN means that many IT jobs will disappear
  An SDN-enabled environment requires less manual work to maintain normal operation than traditional network environments. This statement is true, but that does not mean that traditional network management positions will disappear. As enterprises transition to SDN mode, networking skills evolve, so the demand for network skills also increased. In fact, the type of skills needed for the new era of IP will continue to change. Business and IT professionals should be aware of this, and accordingly tailor their own training and development programs.
SDN Scheme — Misunderstanding (2/2)

• If the server is already virtualized, you don’t need SDN.
  This is not true. Extending the principles of server virtualization to the
  network by replacing traditional hardware with a more flexible virtualized
  network infrastructure will bring more of the same important benefits.
  SDN can also play a greater role, particularly it allows to extend the
  network to the server is provided and more efficient management of traffic
  between the servers can be visualized.

• To Implement SDN, the entire data center network must be replaced.
  “Dismantling the existing system " is not a necessary condition for
  successful implementation of SDN. The more scientific method is to
  gradually migrate from traditional network infrastructure to SDN. In fact,
  Implementing SDN is very simple: use SDN devices as a default choice for
  network components, as part of an existing hardware update plan; or
  deploy SDN when new projects or expansions need to add new devices.
SDN Commanded by the Controller

- The management authority of the network is transferred to the controller software of the control layer, and the centralized control is adopted.
  - The controller software is like a human brain, and the instructions are given to the network device.
  - The network device is dedicated to the transmission of the packet, just like the human limbs are responsible for performing various actions.
  - This concept allows network administrators to configure network resources more flexibly.
  - In the future, network administrators can set up automation automatically by simply issuing commands to the controller. They do not need to log in to the network device one by one to make individual settings.
An Opportunity to Rethink

• How should future networks be
  - Designed
  - Managed
  - Programmed

• What are the right abstractions
  - Simple
  - Powerful
  - Reusable

• A specific SDN: configuration, distribution and forwarding abstraction
How SDN Works
SDN Software Stack

Application

Network OS

Controller

Switch

Switch

Open vSwitch (OVS) / SDN switch

API: OpenFlow
Ethane: Centralized, reactive, per-flow control
OpenFlow: a pragmatic compromise

• + Speed, scale, fidelity of vendor hardware
• + Flexibility and control of software and simulation
• Vendors don’t need to expose implementation
• Leverages hardware inside most switches today
  - Access control list (ACL) tables
    ✓ rules applied to port numbers or IP addresses
## Areas in ONF

<table>
<thead>
<tr>
<th>Areas</th>
<th>Operator</th>
<th>Services</th>
<th>Specification</th>
<th>Market</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carrier Grade SDN</td>
<td>Architecture</td>
<td>Cross Stratum Optimization</td>
<td>Open Datapath</td>
<td>Liaisons</td>
</tr>
<tr>
<td>Data Center</td>
<td>Information Modeling</td>
<td></td>
<td>OF-Config</td>
<td>Proofs of Concept</td>
</tr>
<tr>
<td>Enterprise</td>
<td>Northbound Interfaces</td>
<td></td>
<td>Open Transport</td>
<td>Publications</td>
</tr>
<tr>
<td>Migration</td>
<td>Security</td>
<td></td>
<td>Protocol Independent</td>
<td>SDN Solutions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Forwarding</td>
<td>Showcase</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Testing &amp; Interop</td>
<td>Skills Certification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mobile Networks</td>
<td>Workshops</td>
</tr>
</tbody>
</table>
Members in ONF (1/2)
Members in ONF (2/2)
Three Layers in SDN

- Application
- Network OS
- Controller

API: OpenFlow

Northbound
Southbound

Switch
Open vSwitch (OVS) / SDN switch
Ethernet Switch

Control Path (Software)

Data Path (Hardware)
OpenFlow: the Southbound Interface

OpenFlow Controller

OpenFlow Protocol (SSL/TCP)

Switch

Control Path (OpenFlow Client)

Data Path (Hardware)
Centralized vs Distributed Control
Both models are possible with OpenFlow
# Flow Routing vs. Aggregation

Both models are possible with OpenFlow

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

## Aggregated
- One flow entry covers large groups of flows
- Wildcard flow entries
- Flow table contains one entry per category of flows
- Good for large number of flows, e.g. backbone
Reactive vs. Proactive (pre-populated)
Both models are possible with OpenFlow

Reactive

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

Proactive

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

- OpenFlow technology is a communication protocol used to establish a transmission channel between the control layer and the data layer.
- OpenFlow technology regards the path of packet transmission as a “Flow”, just like a dedicated transmission path.
  - The network administrator can determine the packet transmission mode by setting various network management functions on the controller software and pre-establishing the logical network according to the enterprise policy or service level agreement (SLA).
Openflow Introduction (cont.)

• Then, a secure transmission channel is established between the control layer and the data layer by using SSL encryption technology, and the controller transmits the set OpenFlow routing table to the network device of the data layer through the transmission channel for packet delivery.
  - Because the transmission path is pre-set, the switch does not need to continuously learn to find the path of the packet transmission, which can greatly improve the transmission efficiency and reduce the delay time.

• In the future, enterprises only need to update their OpenFlow firmware provided by the manufacturer.
  - In other words, no matter which manufacturer purchases the network equipment that supports OpenFlow technology, it will be managed by the controller, and the problem of being bound by a single network communication vendor can be solved.
Standardization of OpenFlow

• The nonprofit Internet organization openflow.org was created in 2008.
  - As a mooring to promote and support OpenFlow.
  - The physical organization was really just a group of people that met informally at Stanford University.
  - The first release, OpenFlow1.0.0, appeared on Dec. 31, 2009.
  - Later, OpenFlow 1.1.0 was released on Feb. 28, 2011.

• On March 21, 2011, the Open Network Foundation (ONF) was created.
  - For the purpose of accelerating the delivery and commercialization of SDN.
Components of OpenFlow Network

- **Controller**
  - OpenFlow protocol messages
  - Controlled channel
  - Processing
    - Pipeline Processing
    - Packet Matching
    - Instructions & Action Set
- **OpenFlow switch**
  - Secure Channel (SC)
  - Flow Table
    - Flow entry
OpenFlow Overview

• OpenFlow defines both the communications protocol
  ✓ between the SDN data plane and the SDN control plane
  ✓ part of the behavior of the data plane

• OpenFlow does not describe the behavior of the controller itself.

• The OpenFlow behavior specifies
  - how the device should react in various situations
  - how it should respond to commands from the controller

• There is always an OpenFlow controller that communicates to one or more OpenFlow switches.
OpenFlow Switch (1/2)

- The packet-matching function tries to
  ✓ match the incoming packet (X) with an entry in flow table
  ✓ and then direct the packet to an action box

- The action box has three fundamental options:
  ✓ (A) Forward the packet out, possibly modifying certain header fields first
  ✓ (B) Drop the packet
  ✓ (C) Pass the packet to the controller through a OpenFlow PACKET_IN (Pkt In) message
OpenFlow Switch (2/2)

• The packets are transferred between the controller and the switch through secure channel.

• When the controller has a data packet to forward out through the switch, it uses the OpenFlow PACKET_OUT message. Two paths are possible:
  (1) Controller directly specifies the output port.
  (2) Controller defer the forwarding decision to the packet-matching logic.
OpenFlow Controller

• The OpenFlow control plane differs from the legacy control plane in **three** key ways:
  - It can program different data plane elements with a **common and standard language**, OpenFlow.
  - It exists on a separate **hardware device** than the forwarding plane.
  - The controller can program **multiple data plane elements from a single control plane instance**.
The Controller-Switch Secure Channel
(1/3)

• The communications between controller and switch are secured by
  - Transport Layer Security (TLS) based asymmetrical encryption
  - unencrypted TCP connections

• The connections may be in-band or out-of-band.
The Controller-Switch Secure Channel (2/3)

- **In-band:**
  - The OpenFlow messages from the controller arriving via port K
    ✓ which is part of the OpenFlow data plane.
  - These packets will be handled by the OpenFlow packet-matching logic.
  - The flow tables will have been constructed so that this OpenFlow traffic is forwarded to the LOCAL virtual port
    ✓ results in the messages being passed to the secure channel process
The Controller-Switch Secure Channel (3/3)

• Out-of-band:
  - The secure channel connection enters the switch via port Z
    ✓ *not* switched by OpenFlow data plane
  - Some legacy network stack will deliver the OpenFlow messages via the secure channel to the secure channel process in the switch
  - The out-of-band secure channel is relevant only in the case of an OpenFlow-hybrid switch
# Openflow Controllers

<table>
<thead>
<tr>
<th>Use-Cases</th>
<th>Trema</th>
<th>Nox/Pox</th>
<th>RYU</th>
<th>Floodlight</th>
<th>ODL</th>
<th>ONOS***</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Virtualization by Virtual Overlays</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>PARTIAL</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Hop-by-hop Network Virtualization</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>OpenStack Neutron Support</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Legacy Network Interoperability</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>PARTIAL</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Service Insertion and Chaining</td>
<td>NO</td>
<td>NO</td>
<td>PARTIAL</td>
<td>NO</td>
<td>YES</td>
<td>PARTIAL</td>
</tr>
<tr>
<td>Network Monitoring</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Policy Enforcement</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>PARTIAL</td>
<td>YES</td>
<td>PARTIAL</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Traffic Engineering</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>YES</td>
<td>PARTIAL</td>
</tr>
<tr>
<td>Dynamic Network Taps</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Multi-Layer Network Optimization</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>PARTIAL</td>
<td>NO</td>
<td>PARTIAL</td>
</tr>
<tr>
<td>Transport Networks - NV, Traffic-Rerouting, Interconnecting DCs, etc.</td>
<td>NO</td>
<td>NO</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>NO</td>
</tr>
<tr>
<td>Campus Networks</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>PARTIAL</td>
<td>YES</td>
</tr>
<tr>
<td>Routing</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

https://www.researchgate.net/figure/Comparison-among-SDN-controllers_fig5_281979574
Openflow building blocks

- Monitoring / debugging tools
  - oftrace
  - oflops
  - openseer

- Stanford Provided
  - ENVI (GUI)
  - LAVI
  - n-Casting
  - Expedient

- Controller
  - NOX
  - Beacon
  - Trema
  - Maestro
  - ONIX

- Slicing Software
  - FlowVisor
  - FlowVisor Console

- Commercial Switches
  - HP, NEC, Pronto, Juniper.. and many more

- Stanford Provided
  - Software Ref. Switch
  - NetFPGA
  - Broadcom Ref. Switch
  - OpenWRT
  - PCEngine WiFi AP
  - Open vSwitch

- OpenFlow Switches
  - Broadcom Ref. Switch
  - PCEngine WiFi AP
  - Open vSwitch

https://www.slideshare.net/openflow/openflow-tutorial
OpenFlow Example

Software Layer

Hardware Layer

Flow Table

<table>
<thead>
<tr>
<th>MAC src</th>
<th>MAC dst</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>*</td>
<td>port 1</td>
</tr>
</tbody>
</table>

Controller

5.6.7.8

5.6.7.8

1.2.3.4

資料來源：OpenFlow/SDN tutorial OFC/NFOEC March, 2012
OpenFlow usage

OpenFlow offloads control intelligence to a remote software

資料來源：OpenFlow/SDN tutorial OFC/NFOEC March, 2012
OpenFlow usage (cont.)

- Alice’s code:
  - Simple learning switch
  - Per Flow switching
  - Network access control/firewall
  - StaAc “VLANs”
  - Her own new rouAng protocol: unicast, mulAcast, mulApAth
  - Home network manager
  - Packet processor (in controller)
  - IPvAlice

- VM migration
- Server Load balancing
- Mobility manager
- Power management
- Network monitoring and visualization
- Network debugging
- Network slicing

… and much more you can create!
## Flow Table

- Flow table in switches, routers, and chipsets

<table>
<thead>
<tr>
<th>Flow</th>
<th>Rule (exact &amp; wildcard)</th>
<th>Action</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow N</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Flow Table in switches, routers, and chipsets*
Flow Table

- A flow entry consists of
  - Match fields
    - Match against packets
  - Action
    - Modify the action set or pipeline processing
  - Stats
    - Update the matching packets

<table>
<thead>
<tr>
<th>Match Fields</th>
<th>Action</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Port</td>
<td>Layer 2</td>
<td></td>
</tr>
<tr>
<td>Src MAC</td>
<td>Layer 3</td>
<td></td>
</tr>
<tr>
<td>Dst MAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eth Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vlan Id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP Tos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP Proto</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP Src</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP Dst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP Src Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP Dst Port</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Layer 2
1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline

Layer 3
Layer 4

Lab210

資料來源：國立清華大學資工系．鍾葉青教授—虛擬化技術：Network Virtualization Software Defined Network
## Flow Table Entries

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Packet + byte counters</td>
</tr>
</tbody>
</table>

1. Forward packet to zero or more ports
2. Encapsulate and forward to controller
3. Send to normal processing pipeline
4. Modify Fields
5. Any extensions you add!

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>VLAN ID</th>
<th>VLAN pcp</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP ToS</th>
<th>IP Prot</th>
<th>L4 sport</th>
<th>L4 dport</th>
</tr>
</thead>
</table>

+ mask what fields to match

資料來源：OpenFlow/SDN tutorial OFC/NFOEC March,2012
## Examples

### Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>00:1f..</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Flow Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>port3</td>
<td>00:20..</td>
<td>00:1f..</td>
<td>0800</td>
<td>vlan1</td>
<td>1.2.3.4</td>
<td>5.6.7.8</td>
<td>4</td>
<td>17264</td>
<td>80</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Firewall

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>22</td>
</tr>
</tbody>
</table>
# Examples

## Routing

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>port6</td>
</tr>
</tbody>
</table>

## VLAN Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>00:1f..</td>
<td>*</td>
<td>vlan1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>port6, port7, port9</td>
</tr>
</tbody>
</table>
OpenFlow v1.0-1.5

- OpenFlow (v1.0-1.5) Features
- OpenFlow (v1.0-1.3) Details
OpenFlow recap

Flow table

Packet → Redirect to controller → Apply actions, forward → Drop

資料來源：IEEE CAMAD 2014 — From dumb to smarter switches in software defined networks : an overview of data plane evolution
Models can be perfect and clean, reality is dirty!

• The match/action model can ideally be used to program any network behavior and to get rid of protocol limitations at any level

• But unfortunately, with OF:
  - Matches can be done only on a set of predefined header fields (Ethernet, IPv4, MPLS, VLAN tag, etc.)
  - Actions are limited to a rather small set
  - Header manipulation (like adding label/tags, rewriting of fields, etc.) is limited to standard schemes

• As a result, OF is not really protocol independent and standards (including OF standards) are still necessary
Where do OF limitations come from?

- OpenFlow has been designed having in mind current specialized HW architecture for switches
- Specialized HW is still fundamental in networking
  - General purpose HW (CPU) and soft-switches are still 2 order of magnitude slower
  - Architectures based network processors are also at least 1 order of magnitude slower
- The reference HW model for OF flow tables is TCAM (Ternary Content Addressable Memory)

![Diagram showing flow table and actions](image-url)
Where do OF limitations come from?

- TCAMs however are typically expensive components that are used by manufacturers only when strictly necessary.
- Less expensive memory components based on predefined search keys are often used for most of the common functions of a switch.
- OF success depends on its “vendor neutral” approach where implementations issues are completely opaque (including reuse of standard modules for e.g. MAC and IP forwarding).
- Specialized ASICs are typically complex with a number of hard limitations on table types, sizes, and match depth.
Evolution of the AL in OpenFlow: OF 1.1

- Single tables are costly
  - all possible combinations of header values in a single long table
- Solution: **Multiple Match Tables (MMT)**
- New actions of **MMT**:
  - **Add metadata**: parameters added and passed to next table
  - **Goto table**: possibility to go to specific tables for further processing

資料來源: IEEE CAMAD 2014 — From dumb to smarter switches in software defined networks: an overview of data plane evolution
Evolution of the AL in OpenFlow: OF 1.1

• Packets of the same flow are applied the same actions unless the table entry is modified by the controller

• Not good for some common and important cases (e.g. multicast, multipath load balancing, failure reaction, etc.)

• Solution: **Group tables**
  - Goto table “group table n”
  - List of buckets of actions
  - All or some of the buckets are executed depending on the following types of Group tables:
    - All (multicast)
    - Select (multipath)
    - Fast-failover (protection switching)
Evolution of the AL in OpenFlow: OF 1.1

• Fast failover

• Note that this is the first “stateful” behavior in the data plane introduced in OF !!!
Evolution of the AL in OpenFlow: OF 1.2

- Support for IPv6, new match fields:
  - source address, destination address, protocol number, traffic class, ICMPv6 type, ICMPv6 code, IPv6 neighbor discovery header fields, and IPv6 flow labels
- Extensible match (Type Length Value)
- Experimenter extensions
- Full VLAN and MPLS support
- Multiple controllers
**Evolution of the AL in OpenFlow: OF 1.3**

- Initial traffic shaping and QoS support
  - **Meters**: tables (accessed as usual with “goto table”) for collecting statistics on traffic flows and applying rate-limiters

<table>
<thead>
<tr>
<th>Meter Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meter identifier</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**Type**

- Rate
- Counters
- Type/argument

---

資料來源：IEEE CAMAD 2014 — From dumb to smarter switches in software defined networks : an overview of data plane evolution
Evolution of the AL in OpenFlow: OF v1.4 and v1.5

• OpenFlow v1.4
  - More extensible wire protocol
  - **Synchronized tables**
    - tables with synchronized flow entries
  - **Bundles**
    - similar to transactional updates in DB
  - Support for optical ports

• OpenFlow v1.5
  - Packet type identification process (Ethernet packet, PPP packet) egress table.
# Comparison with OpenFlow Version

<table>
<thead>
<tr>
<th>Protocol Version</th>
<th>Main Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenFlow 1.0</td>
<td>Single-table, IPv4</td>
</tr>
<tr>
<td>OpenFlow 1.1</td>
<td>Multi-table, Group-table, MPLS, VLAN</td>
</tr>
<tr>
<td>OpenFlow 1.2</td>
<td>Multi-Controller, IPv6</td>
</tr>
<tr>
<td><strong>OpenFlow 1.3 (mainstream, long-term support, stable)</strong></td>
<td>Meter Table, Version negotiation capability</td>
</tr>
<tr>
<td>OpenFlow 1.4</td>
<td>Synchronized tables, Protocol message perfection</td>
</tr>
<tr>
<td>OpenFlow 1.5</td>
<td>Packet type identification process (Ethernet packet, PPP packet) egress table.</td>
</tr>
</tbody>
</table>

資料來源：https://blog.csdn.net/qq_29229567/article/details/88797395
OpenFlow 1.0
Ports and Port Queues

• An OpenFlow V.1.0 port corresponds to a physical port.

• Sophisticated switches have supported multiple queues per physical port for different QoS levels.

• OpenFlow 1.0 embraces the QoS concept and permits a flow to be mapped to an already defined queue at an output port.
Flow Table

• A flow table consists of flow entries.
• A flow entry consists of
  - header fields used as match criteria to determine whether an incoming packet matches this entry
  - counters used to track statistics relative to this flow
  - actions prescribing what the switch should do for a matched packet
Packet Matching (1/3)

• 12 match fields may be used:
  - Switch input port,
  - VLAN ID, VLAN priority,
  - Ethernet source address, Ethernet destination address, Ethernet frame type
  - IP source address, IP destination address, IP protocol, IP Type of Service (ToS) bits
  - TCP/UDP source port, TCP/UDP destination port
Packet Matching (2/3)

- Flow entries are processed in order, and once a match is found, no further match attempts are made against that flow table.
- OpenFlow 1.0 is silent about which of these 12 match fields are required versus those that are optional.
- The ONF has clarified this confusion by defining three types of conformance:
  - Full conformance means that all 12 match fields are supported.
  - Layer two conformance means that only layer two header fields are supported.
  - Layer three conformance means that only layer three header fields are supported.
Packet Matching (3/3)

- If no flow entry is matched, it is called table miss.
  - In this case, the packet is forwarded to the controller.
- OpenFlow V1.0 was designed as an abstraction of the way that existing switching hardware works.
- In later versions, we will see that the specification outpaces the reality of today’s hardware.
Actions and Packet Forwarding (1/4)

• 5 special virtual ports defined in V.1.0: {LOCAL, ALL, CONTROLLER, IN_PORT, TABLE}
  - LOCAL indicates that the packet needs to be processed by the local OpenFlow control software.
    ✓ LOCAL can be used for in-band OpenFlow messages.
  - ALL is used to send a packet out all ports except the input port.
**Actions and Packet Forwarding (2/4)**

- CONTROLLER indicates that the switch should forward this packet to the controller.
- IN_PORT instructs the switch to forward the packet back out of the port on which it arrived.
- TABLE only applies to packets that the controller sends to the switch.

- The packets are then processed by normal OpenFlow packet processing pipeline.
Actions and Packet Forwarding (3/4)

• In V.1.0, there are two optional virtual ports: NORMAL and FLOOD.
  - A packet forwarded to NORMAL port is sent to legacy forwarding logic in the case of a hybrid switch.
  - FLOOD instructs the switch to send a copy of the packet along minimum spanning tree, except the input port.
Actions and Packet Forwarding (4/4)

• For a table miss, the virtual port CONTROLLER is used.
• There are two optional actions in V.1.0: enqueue and modify field.
  - Enqueue specifies a queue to a particular port.
  - Modify-field informs the switch how to modify certain header fields.

The action box has three fundamental options:
(A) Forward the packet out, possibly modifying certain header fields first
(B) Drop the packet
(C) Pass the packet to the controller through a OpenFlow PACKET_IN (Pkt In) message
Messaging Between Controller and Switch

- If the switch knows the IP address of the controller, the switch will initiate this connection.
- Each message between controller and switch starts with the OpenFlow header.
OpenFlow Message Types and Protocol Session

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Category</th>
<th>Subcategory</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELLO</td>
<td>Symmetric</td>
<td>Immutable</td>
</tr>
<tr>
<td>ECHO_REQUEST</td>
<td>Symmetric</td>
<td>Immutable</td>
</tr>
<tr>
<td>ECHO_REPLY</td>
<td>Symmetric</td>
<td>Immutable</td>
</tr>
<tr>
<td>VENDOR</td>
<td>Symmetric</td>
<td>NA</td>
</tr>
<tr>
<td>FEATURES_REQUEST</td>
<td>Controller-switch</td>
<td>Switch configuration</td>
</tr>
<tr>
<td>FEATURES_REPLY</td>
<td>Controller-switch</td>
<td>Switch configuration</td>
</tr>
<tr>
<td>GET_CONFIG_REQUEST</td>
<td>Controller-switch</td>
<td>Switch configuration</td>
</tr>
<tr>
<td>GET_CONFIG_REPLY</td>
<td>Controller-switch</td>
<td>Switch configuration</td>
</tr>
<tr>
<td>SET_CONFIG</td>
<td>Controller-switch</td>
<td>Switch configuration</td>
</tr>
<tr>
<td>PACKET_IN</td>
<td>Async</td>
<td>NA</td>
</tr>
<tr>
<td>FLOW_REMOVED</td>
<td>Async</td>
<td>NA</td>
</tr>
<tr>
<td>PORT_STATUS</td>
<td>Async</td>
<td>NA</td>
</tr>
<tr>
<td>EOFRR</td>
<td>Async</td>
<td>NA</td>
</tr>
<tr>
<td>PACKET_OUT</td>
<td>Controller-switch</td>
<td>Cmd from controller</td>
</tr>
<tr>
<td>FLOW_MOD</td>
<td>Controller-switch</td>
<td>Cmd from controller</td>
</tr>
<tr>
<td>PORT_MOD</td>
<td>Controller-switch</td>
<td>Cmd from controller</td>
</tr>
<tr>
<td>STATS_REQUEST</td>
<td>Controller-switch</td>
<td>Statistics</td>
</tr>
<tr>
<td>STATS_REPLY</td>
<td>Controller-switch</td>
<td>Statistics</td>
</tr>
<tr>
<td>BARRIER_REQUEST</td>
<td>Controller-switch</td>
<td>Barrier</td>
</tr>
<tr>
<td>BARRIER_REPLY</td>
<td>Controller-switch</td>
<td>Barrier</td>
</tr>
<tr>
<td>QUEUE_GET_CONFIG_REQUEST</td>
<td>Controller-switch</td>
<td>Queue configuration</td>
</tr>
<tr>
<td>QUEUE_GET_CONFIG_REPLY</td>
<td>Controller-switch</td>
<td>Queue configuration</td>
</tr>
</tbody>
</table>
Initialization Phase

- The **HELLO** messages are exchanged to determine the highest OpenFlow version supported by the peers, where the lower of the two is used.
- The controller uses the **FEATURES** message pair to interrogate the switch about the supported features.
- The controller modifies existing flow entries in the switch via the **FLOW_MOD** message.
Operation Phase

- The `PACKET_IN` message is the way the switch passes data packets back to the controller for exception handling.
- The controller uses `PACKET_OUT` to send data packets to the switch for forwarding out through the data plane.
Monitoring Phase

• **ECHO** messages are used by either side to ascertain that the connection is still alive and to measure the current latency or bandwidth of the connection.

• **PORT_STATUS** is used to communicate changes in port status.

• Statistics are obtained from the switch via the **STATS** message pair.
Example: Controller Programming Flow Table

(1/3)

- Adding a flow entry

Adding a flow for packets entering the switch on any port, with source IP addresses 192.168.1.1 and destination IP address 209.1.2.1, source TCP port 20, and destination port 20. All other match fields have been wildcarded. The output port is specified as P.

All Ethernet frames entering the switch on input port K with a destination Ethernet address of 0x000CF1569AD should be output on output port N.
Example: Controller Programming Flow Table (2/3)

- Modifying a flow entry

At time $t_b$, the controller sends a FLOW_MOD (MODIFY) command for flow entry zero. The controller seeks to modify the corresponding flow entry such that there is a one-hour (3600-second) idle time on that flow.
Example: Controller Programming Flow Table (3/3)

- A packet arriving at the switch through port 2 with source IPv4 address of 192.168.1.1 and destination IPv4 address of 209.1.2.1.
- The packet-matching function scans the flow table starting at flow entry 0 and finds a match in flow entry F.
- Flow entry F stipulates that a matching packet should be forwarded out port P.
Example: Switch Forwarding Packet to Controller
OpenFlow 1.1 Additions
OpenFlow V.1.1

- OpenFlow 1.1 had little impact other than as a stepping stone to OpenFlow 1.2.
- SDN community waited for V.1.2 (the first version by ONF) before creating implementations.
- Five major new features
  - Multiple flow tables
  - Groups
  - MPLS and VLAN tag support
  - Virtual ports
  - Controller connection failure
Multiple Flow Tables

• In V.1.1, it is possible to defer further packet processing to subsequent matching in other flow tables.

• **Instruction** is introduced to be associated with a flow entry.

• Flow tables can be chained by **GOTO instructions**.

```
enum ofp_instruction_type {
  OFFIT_GOTO_TABLE = 1, /* Setup the next table in the lookup pipeline */
  OFPIT_WRITE_METADATA = 2, /* Setup the metadata field for use later in pipeline */
  OFPIT_WRITE_ACTIONS = 3, /* Write the action(s) onto the datapath action set */
  OFPIT_APPLY_ACTIONS = 4, /* Applies the action(s) immediately */
  OFPIT_CLEAR_ACTIONS = 5, /* Clears all actions from the datapath action set */
  OFPIT_EXPERIMENTER = 0xFFFF /* Experimenter instruction */
};
```
**Multiple Flow Tables**

- For an incoming packet, an action set is initialized and then modified by instructions through the processing pipeline.
- When the pipeline ends, the actions in the action set are executed in the following order:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Copy TTL inward</td>
<td>4. Copy TTL outward</td>
<td>7. QoS: Apply QoS actions (e.g. set_queue)</td>
</tr>
<tr>
<td>2. Pop</td>
<td>5. Decrement TTL</td>
<td>8. Group: Apply actions to relevant action buckets if a group action is specified</td>
</tr>
</tbody>
</table>

- If there is neither a group nor output action, the packet is dropped.
Multiple Flow Tables

• Note that actions specified by an Apply-Actions instruction are immediately executed.

• When a matched flow entry does not specify a GOTO flow table, the processing pipeline completes, and actions in the current action set are then executed.

• In normal cases, the final action is to forward a packet to an output port, to the controller, or to a group table.

• The flexibility of multiple flow tables comes at a price for adapting existing hardware switches.
Example: Multiple Flow Tables

- The packet first matches entry 1 in Table 0.
  - An action “output to port P” is added to the action set.
  - The pipeline then jumps to Table K according to the GOTO instruction.
- The packet matches entry F in Table K.
  - Since there is no GOTO instruction here, actions in the action set are executed.
**Groups**

- **Group table** consists of group entries.
  - Each entry consists of one or more action buckets.
- Refinements on flooding, such as **multicast**, can be achieved in V.1.1 by defining groups as specific sets of ports.
- One group’s buckets may forward to other groups, providing the capability to **chain groups** together.
Groups
MPLS and VLAN Tag Support

• Multiprotocol label switching (MPLS) and virtual local area networks (VLAN) are supported by adding PUSH and POP actions.
  - When a PUSH action is executed, a new header (or tag) is inserted in front of the current outermost header.
  - On the other hand, POP is used to remove the current outermost header.

• The matching logic is replaced by V.1.2, and will be discussed later.
Virtual Ports

• A V.1.1 switch classifies ports into standard ports and reserved virtual ports.

• Standard ports consist of
  - Physical ports
  - Switch-defined virtual ports

• Reserved virtual ports consist of
  - ALL,
  - CONTROLLER,
  - TABLE,
  - IN_PORT,
  - LOCAL (optional),
  - NORMAL (optional)
  - FLOOD (optional)
Controller Connection Failure

• Two modes are introduced for loss of connection between switch and controller:
  
  - In fail secure mode,
    ✓ the switch continues to operate as a normal V.1.1 switch except that all messages destined for the controller are dropped.
  
  - In fail standalone mode,
    ✓ the switch additionally ceases its OpenFlow pipeline processing and continues to operate in its native, underlying switch or router mode.
OpenFlow 1.2 Additions
OpenFlow V.1.2

- Eight major new features
  1. Extensible match support
  2. Extensible set_field packet-rewriting support
  3. Extensible context expression in “packet-in”
  4. Multiple controller enhancements
  5. Extensible error messages via experimenter error type
  6. IPv6 support
  7. Simplified behavior of flow-mod request
  8. Removed packet parsing specification
Extensible Match Support

- A generic and extensible packet-matching capability has been added in V.1.2 via the Openflow Extensible Match (OXM) descriptors.
  - OXM defines a set of type-length-value (TLV) pairs that can describe or define virtually any of the header fields.

```c
/* Fields to match against flows */
struct ofp_match {
    uint16_t type;    /* One of OFPMT_* */
    uint16_t length;  /* Length of ofp_match */
    uint32_t in_port; /* Input switch port */
    uint32_t wildcards; /* Wildcard fields */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address */
    uint8_t dl_src_mask[OFP_ETH_ALEN]; /* Ethernet source address mask */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address */
    uint8_t dl_dst_mask[OFP_ETH_ALEN]; /* Ethernet destination address */
    uint16_t dl_vlan; /* Input VLAN id */
    uint8_t dl_vlan_pcp; /* Input VLAN priority */
}
```

OpenFlow 1.1

```c
struct ofp_match {
    uint16_t type;    /* One of OFPMT_* */
    uint16_t length;  /* Length of ofp_match (excluding padding) */
    /* Followed by: */
    /* - Exactly (length - 4) (possibly 0) bytes containing OXM TLVs, then */
    /* - Exactly ((Length + 7)/8*8 - length) (between 0 and 7) bytes of */
    /*   all-zero bytes */
    /* In summary, ofp_match is padded as needed, to make its overall size */
    /* a multiple of 8, to preserve alignment in structures using it. */
    uint8_t ooxm_fields[4]; /* OXMs start here - Make compiler happy */
    /* */
    OFP_ASSERT(sizeof(struct ofp_match) == 8);
}
```

OpenFlow 1.2
Extensible Match Support

OXM TLV Layout
Extensible Match Support

- The ability to match on any combination of header fields is provided within the OPENFLOW_BASIC match class.
- The EXPERIMENTER match class opens up the opportunity for matching on fields in the packet payload
  - providing a near-limitless horizon for new definitions of flows.

```c
enum ofp_oxm_class {
    OFPXMC_NXM_0     = 0x0000, /* Backward compatibility with NXM */
    OFPXMC_NXM_1     = 0x0001, /* Backward compatibility with NXM */
    OFPXMC_OPENFLOW_BASIC = 0x8000, /* Basic class for OpenFlow */
    OFPXMC_EXPERIMENTER = 0xFFFF, /* Experimenter class */
};
```
Extensible set_field Packet Rewriting Support

• It is allowed to set the value of any packet header field that may be used for matching.

• For EXPERIMENTER match class,
  - it is possible to modify any packet fields (including the payload) that is not part of the standard OXM header fields.

```c
/* Action structure for OFPAT_SET_FIELD. */
struct ofp_action_set_field {
  uint16_t type; /* OFPAT_SET_FIELD. */
  uint16_t len; /* Length is padded to 64 bits. */
  /* Followed by:
   * - Exactly oxm_len bytes containing a single OXM TLV, then
   * - Exactly ((oxm_len + 4) + 7)/8*8 - (oxm_len + 4) (between 0 and 7)
   * bytes of all-zero bytes
   */
  uint8_t field[4]; /* OXM TLV - Make compiler happy */
};
OFPP_ASSERT(sizeof(struct ofp_action_set_field) == 8);
```
Extensible Context Expression in PACKET_IN

- The OXM encoding is also used to extend the PACKET_IN message sent from the switch to the controller.

OpenFlow 1.1

```
/* Packet received on port (datapath -> controller). */
struct ofp_packet_in {
    struct ofp_header;
    uint32_t buffer_id; /* ID assigned by datapath. */
    uint32_t in_port; /* Port on which frame was received. */
    uint32_t in_phy_port; /* Physical Port on which frame was received. */
    uint16_t total_len; /* Full length of frame. */
    uint8_t reason; /* Reason packet is being sent (one of OFPR_*) */
    uint8_t_table_id; /* ID of the table that was looked up */
    uint8_t_data[0]; /* Ethernet frame, halfway through 32-bit so the IP header is 32-bit aligned. Amount of data is inferred from the length field in the header. Because of padding offsetof(struct ofp_packet_in, data) = sizeof(struct ofp_packet_in) - 2. */
};

OFPP_ASSERT(sizeof(struct ofp_packet_in) == 24);
```

OpenFlow 1.2

```
/* Packet received on port (datapath -> controller). */
struct ofp_packet_in {
    struct ofp_header;
    uint32_t buffer_id; /* ID assigned by datapath. */
    uint16_t total_len; /* Full length of frame. */
    uint8_t_reason; /* Reason packet is being sent (one of OFPR_*) */
    uint8_t_table_id; /* ID of the table that was looked up */
    struct ofp_match match; /* Packet metadata. Variable size. */
    /* Followed by: */
    /* - Exactly 2 all-zero padding bytes, then */
    /* - An Ethernet frame whose length is inferred from header.length. */
    /* The padding bytes preceding the Ethernet frame ensure that the IP */
    /* header (if any) following the Ethernet header is 32-bit aligned. */
    /* */
    /*    */;
};

OFPP_ASSERT(sizeof(struct ofp_packet_in) == 24);
```
Multiple Controllers

• In V.1.2, the switch may be configured to maintain simultaneous connections to multiple controllers.

• If a message pertains to multiple controllers, it is duplicated and a copy sent to each controller.

• Three different roles of controllers: slave, master and equal
  - In slave mode, the controller may only request data from the switch.
  - Both master and equal modes allow the controller the full ability to program the switch.
  - Only one controller in master mode is allowed, while other controllers are in slave mode.
OpenFlow 1.3 Additions
OpenFlow V.1.3

- OpenFlow V.1.3 was released on April 13, 2012.
  - This release was a major milestone, especially for ASIC designers.
  - It is likely that the real-life chips that support V.1.3 will have to limit the number of flow tables to a manageable number.

- Thirteen major new features

| 1. Refactor capabilities negotiation | 6. Auxiliary connections | 11. Cookies in PACKET_IN |
| 4. Per-flow meters | 9. Rework tag order |
| 5. Per-connection event filtering | 10. Tunnel-ID metadata |
More Flexible Table-miss Support

• A table miss was configurable as one of three options:
  - dropping,
  - forwarding to the controller,
  - and continuing matching in the next table.

• V.1.3 expands on this limited handling capability via the introduction of the table-miss flow entry.
  - The table-miss flow entry is of the lowest priority (zero) and all match fields are wildcards.
  - The advantage of this approach is that full semantics of flow entry (including instructions and actions) are applicable for a table miss.
  - For example, a table-missed packet may be passed to another flow table by a GOTO instruction for further processing.
Per-Flow Meters

- Meters are defined on a per-flow basis and reside in a meter table.
- V.1.3 instructions may direct packets to a meter identified by a meter ID.
- V.1.3 meters are only rate-limiting meters.
- There may be multiple meter bands attached to a given meter.
Per-Flow Meters

• When a packet is processed by a meter, at most one band is used.
• This band is selected based on the highest bandwidth rate band that is lower than the current measured bandwidth for that flow.
• If the current measured rate is lower than all bands, no band is selected, and no action is taken.
• If a band is selected, the action prescribed by the band type field is taken.
Example: Enforcing QoS via Meter Bands

• The packet from Port 2 matches a flow entry with an instruction which directs the packet to Meter 3.
  - If the bandwidth limits are exceeded, the packet is dropped.
  - Otherwise, the packet undergo further processing in the pipeline.
Per Connection Event Filtering

• In the previous versions, all controllers must receive the same kind and quantity of asynchronous notifications from the switch.
• V.1.3 introduces a SET_ASYNC message that allows the controller to specify the sorts of async messages it is willing to receive from a switch.
Auxiliary Connections

- As deployment of OpenFlow grows, performance considerations have become increasingly important.
- V.1.3 allows multiple connections per controller-switch channel.
- The major advantage is to achieve greater overall throughput, both in control plane and data plane.
Cookies in PACKET_IN

• In the case of PACKET_IN messages, it is somewhat wasteful to require the switch to match over and over for the same flow.

• V.1.3 allows the switch to pass a cookie with the PACKET_IN message.
  - The switch maintains the cookie in a new field in the flow entry.
  - The cookie allows the switch to cache the flow entry pointed to by this cookie, and prevent the full packet-matching process.

<table>
<thead>
<tr>
<th>Header Fields</th>
<th>Field value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Field value</td>
</tr>
<tr>
<td>Counters</td>
<td>Field value</td>
</tr>
<tr>
<td>Actions/Instructions</td>
<td>Field value</td>
</tr>
<tr>
<td>Timeouts</td>
<td>Field value</td>
</tr>
<tr>
<td>Cookie</td>
<td>Field value</td>
</tr>
</tbody>
</table>
Thanks for your attention!