Sharif University of Technology

Computer Engineering Department

Software-Defined Networking

Ali Movaghar

Mohammad Hosseini

OpenFlow – Part 2

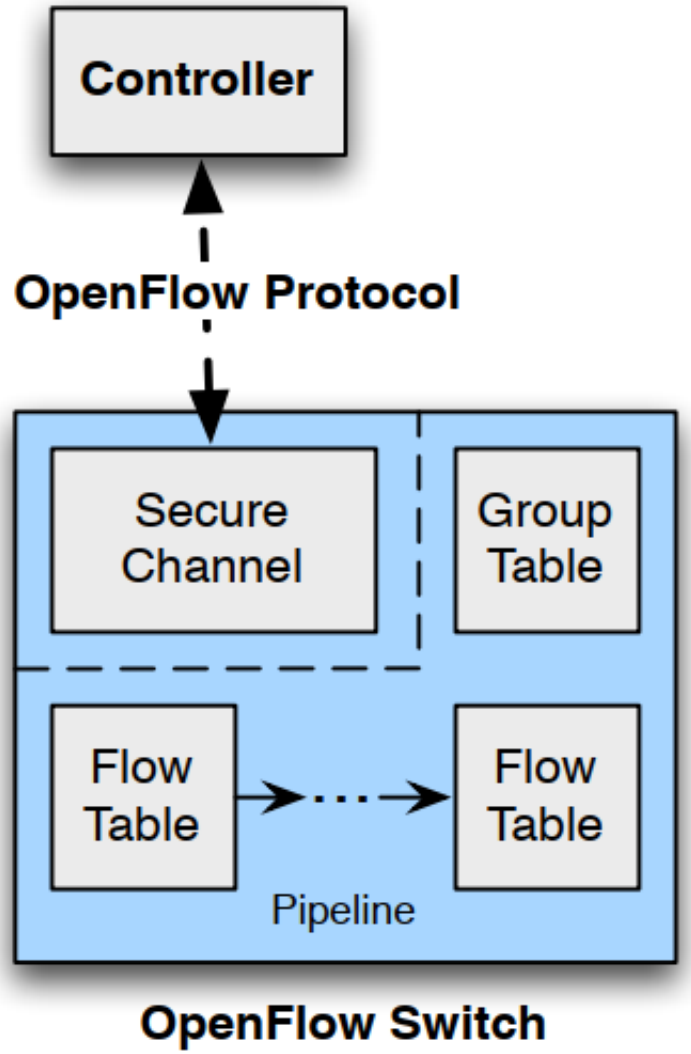
TA: Iman Rahmati & Farbod Shahinfar

OpenFlow 1.1

- Released in 2011
- Wire protocol version: 0x02
- Major changes:
  - Multiple flow tables in a pipeline
  - Instructions to control pipeline processing
  - Group table
  - MPLS and improved VLAN support
  - Controller connection failure handling

Components

- An OpenFlow switch consists of *one or more flow tables* and *a group table*, which perform packet lookups and forwarding.
- Pipeline: the set of linked tables that provide matching, forwarding, and packet modifications in an OpenFlow switch.



Flow Table

❖ Flow table: contains a set of flow entries.
Each flow entry consists of *match fields*, *counters*, and *a set of instructions* to apply to matching packets.

Match fields	Counters	Instructions

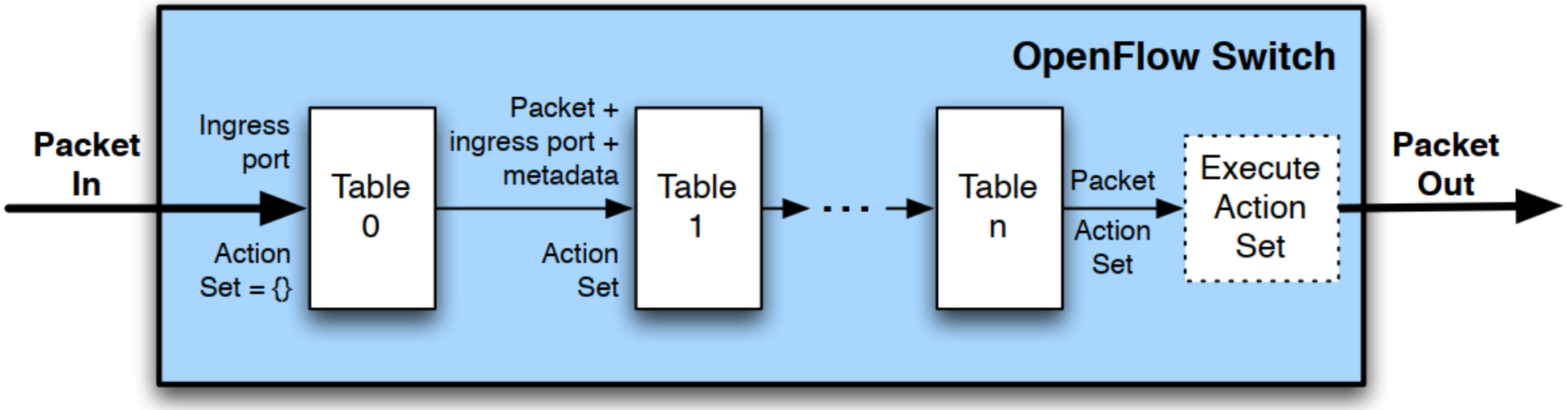
❖ **Instructions**: describe packet forwarding/modification (**actions**), group table processing, and pipeline processing.

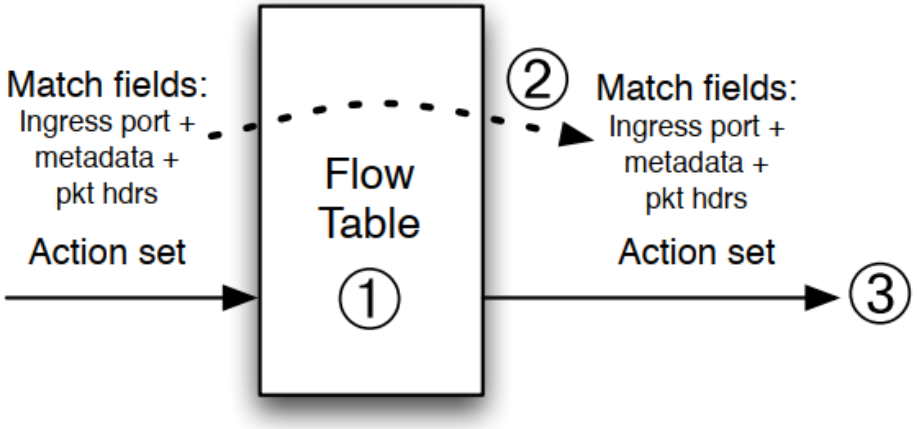
❖ Match fields: consist of the *ingress port*, *packet headers*, and optionally ***metadata*** specified by a previous table.

MPLS *label* and *traffic class* fields were added

a maskable register value used to carry information from one table to the next

Pipeline





① Find highest-priority matching flow entry

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

③ Send match data and action set to next table

Pipeline

- Pipeline processing always starts at the first flow table: the packet is first matched against entries of flow table 0.
- If the packet matches a flow entry in a flow table, the corresponding instruction set is executed.
- The instructions in the flow entry may direct the packet to another flow table (using the Goto Instruction).
- A flow entry can only direct a packet to a flow table number which is greater than its own flow table number.
- If the matching flow entry does not direct packets to another flow table, pipeline processing stops at this table. When pipeline processing stops, the packet is processed with its associated action set.
- Each flow table may not support every match field, every instruction, or every action. The table features request enables the controller to discover what each table of the switch supports.

Table Miss

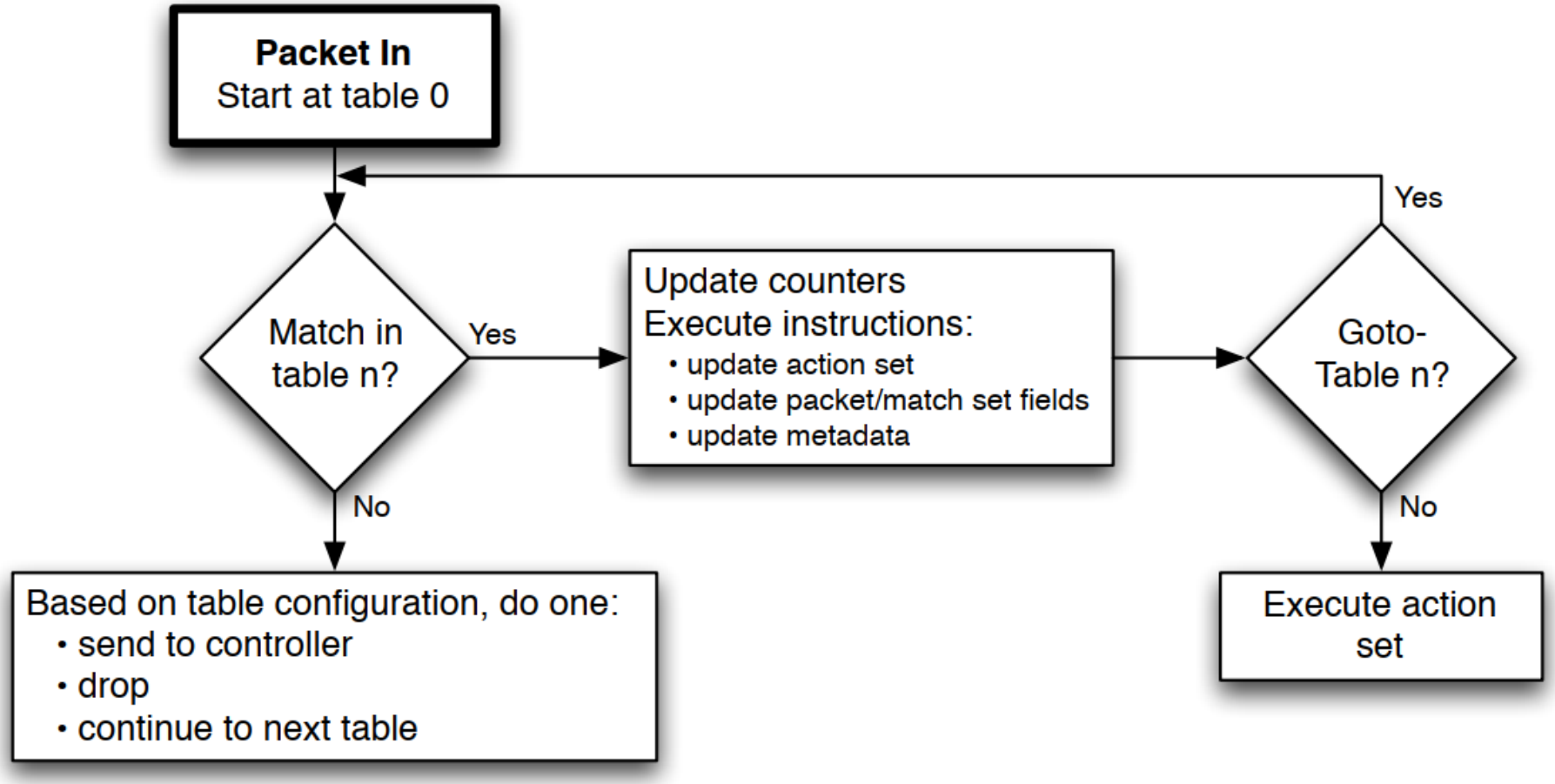
- If the packet does not match a flow entry in a flow table, this is a **table miss**.

- The behavior on table miss depends on the table configuration:

  - Sending packets to the controller over the control channel via a packet-in message.
  - Dropping the packets
  - Processing the packets by the next sequentially numbered table.

  - Flow tables can be configured by a new **controller-to-switch** message named TABLE_MOD.

Packet flow through an OpenFlow switch



Group Table

- Flow entries can point to a group, which specifies additional processing.
- An OpenFlow group is an abstraction that facilitates more complex and specialized packet operations that cannot easily be performed through a flow table entry.
- A group table consists of group entries.

Group Identifier	Group Type	Counters	Action Buckets

- Each group entry contains a list of *action buckets* with specific semantics dependent on *group type*.
- The group table can be configured by a new **controller-to-switch** message named GROUP_MOD.

Group Table

❖ Group Types:

- **all**: Execute all buckets in the group. This group is used for multicast or broadcast forwarding. The packet is cloned for each bucket; one packet is processed for each bucket of the group.

- **select**: Execute one bucket in the group. Packets are sent to a single bucket in the group, based on a switch-computed selection algorithm (e.g. hash on some user-configured tuple or simple round robin). Designed for load-balancing. Each bucket of this type can have a ***weight***.

- **indirect**: Execute the one defined bucket in this group. Allows multiple flows or groups to point to a common group identifier, supporting faster, more efficient convergence (e.g. next hops for IP forwarding).

- **fast failover**: Execute the first live bucket. Each action bucket is associated with a specific port and/or group that controls its liveness. Enables the switch to change forwarding without requiring a round trip to the controller.

New Actions

- **Set-Queue:** setting the ***queue id*** for a packet. When the packet is forwarded to a port using the output action, the queue id determines which queue attached to this port is used for forwarding the packet.

- **Group:** processing the packet through the specified group.

- **Push-[*Tag*] / Pop-[*Tag*]:** a set of optional actions for adding or removing tags such as **VLAN** and **MPLS** headers. Newly pushed tags should always be inserted as the outermost tag in the outermost valid location for that tag. Pop actions, remove the outermost tag of the selected header type.

- **[Set-Field actions]:** some new optional Set-Field actions were added:
  - Set MPLS label/traffic_class/TTL, decrement MPLS TTL
  - Set IPv4 ECN/TTL, decrement IPv4 TTL
  - Copy TTL outwards/inwards (IP and MPLS)

Instructions

- Each flow entry contains a set of instructions that are executed when a packet matches the entry. These instructions result in changes to the **packet**, **action set** and/or **pipeline processing**.
  - Apply-Actions *action(s)*: Applies the specific ***action(s)*** immediately, without any change to the ***Action Set***. This instruction may be used to modify the packet between two tables or to execute multiple actions of the same type.
  - Clear-Actions: Clears all the actions in the ***action set*** immediately.
  - Write-Actions *action(s)*: Merges the specified set of ***action(s)*** into the current ***action set***. If an action of the given type exists in the current set, overwrite it, otherwise add it.
  - Write-Metadata *metadata/mask*: Writes the masked ***metadata*** value into the metadata field.
  - Goto-Table *next-table-id*: Indicates the next table in the processing pipeline. The ***table-id*** must be greater than the current *table-id*.
- The instruction set associated with a flow entry contains a maximum of one instruction of each type. The instructions of the set execute in the order specified by this above list.

Action Set

- An **action set** is associated with each packet.
- This set is empty by default.
- A flow entry can modify the action set using *Write-Action* or *Clear-Action* instructions associated with a particular match.

- The action set is carried between flow tables.
- When an instruction set does not contain a *Goto-Table* instruction, pipeline processing stops and the actions in the action set are executed.

- An action set contains a maximum of one action of each type.
- When multiple actions of the same type are required, e.g. pushing multiple MPLS labels or popping multiple MPLS labels, the *Apply-Actions* instruction may be used.

Action Set

- The actions in an action set are applied **in the order specified in the standard**, regardless of the order that they were added to the set.
- The output action in the action set is executed last.
- If both an output action and a group action are specified in an action set, the output action is ignored and the group action takes precedence.

Action List

- The ***Apply-Actions*** instruction and the ***Packet-out*** message include an **action list**.
- The actions of an action list are executed in the order specified by the list, and are applied immediately to the packet.
- The effect of the actions is **cumulative**. For example, if the action list contains two Push VLAN actions, two VLAN headers are added to the packet.
- If the action list contains an output action or a group action, a **copy** of the packet is forwarded in **its current state** to the desired port or group.
- After the execution of the action list in an *Apply-Actions* instruction, pipeline execution continues on **the modified packet**

Connection Interruption

- In the case that a switch loses contact with the current controller, The switch must immediately enter either "fail secure mode" or "fail standalone mode", depending upon the switch implementation and configuration.

- **Fail secure mode**: the only change to switch behavior is that packets and messages destined to the controllers are dropped. Flow entries should continue to expire according to their timeouts.

- **Fail standalone mode**: the switch processes all packets using the **NORMAL** port. The switch acts as a legacy Ethernet switch or router

The OpenFlow Switch Categories

- OpenFlow-compliant switches come in two types:
  - **OpenFlow-only**: switches that support only OpenFlow operation. In those switches, all packets are processed by the OpenFlow pipeline, and can not be processed otherwise
  - **OpenFlow-hybrid**: switches that support both OpenFlow operation and ***normal*** Ethernet switching operation, i.e. traditional L2 Ethernet switching, VLAN isolation, L3 routing, ACL and QoS processing.

OpenFlow 1.2

- Released in 2011
- Wire protocol version: 0x03
- Major changes:
  - Support for IPv6
  - OpenFlow Extensible Match
  - Support for multiple controllers

OpenFlow Standard Ports

- **Physical ports**: switch defined ports that correspond to a hardware interface of the switch.
- **Reserved ports:** ports defined by this specification.
  - They specify **generic forwarding actions** such as sending to the *controller*, *flooding*, or forwarding using non-OpenFlow methods, such as "*normal*" switch processing.
- **Logical ports**: switch defined ports that do not correspond directly to a hardware interface of the switch.
  - Logical ports are higher level abstractions that may be defined in the switch using non-OpenFlow methods (e.g. link aggregation groups, tunnels, loopback interfaces).
  - Logical ports can optionally be used to insert a network service or complex processing in the OpenFlow switch.
  - Packets sent to logical ports are either consumed by the logical port or eventually sent over a physical port. In version 1.5, packets sent to a logical port can be **recirculated back** to the OpenFlow switch after the logical port processing. This can be a property of a logical port.

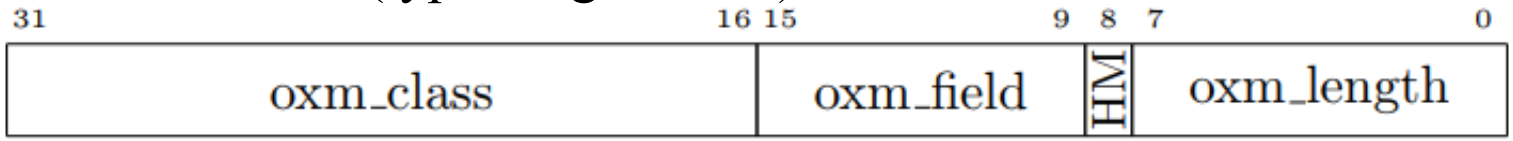
New format for match fields

- Flow match fields are now described using the OpenFlow Extensible Match (OXM) format, which is a *type-length-value* (TLV) format.

- Objectives:
  - Classifying match fields and making them structured
  - Making it easy to add new match fields by ONF members

OpenFlow Extensible Match (OXM)

- Each OXM is a TLV (type-length-value)



Name		Width	Usage
oxm_type	oxm_class	16	Match class: member class or reserved class
	oxm_field	7	Match field within the class
	oxm_hasmask	1	Set if OXM include a bitmask in payload
	oxm_length	8	Length of OXM payload

- *oxm_class* is an OXM match class containing related match types. *oxm_field* is a class-specific value, identifying one of the match types within the match class.
- The combination of *oxm_class* and *oxm_field* are collectively *oxm_type*.
- Two types of OXM match classes:
  - ONF reserved classes: used for the OpenFlow specification itself.
    - **OFPXMC_OPENFLOW_BASIC**: contains the basic set of OpenFlow match fields. (IPv6, ARP, ICMP header fields [and TCP_FLAGS in version 1.5] were added)
    - GPRS Tunneling Protocol (GTP), Network Service Header (NSH)
  - ONF member classes: allocated by the ONF on an as needed basis. They identify an ONF member and can be used arbitrarily by that member. Support for ONF member classes is **optional**

New Actions

- A new action named **Set-Field** replaced the previous actions for setting the header fields.
- The Set-Field action sets a header field described using a single **OXM TLV** structure.
- The value of *oxm_hasmask* must be zero and no *oxm_mask* is included. (In version 1.5, *oxm_mask* is supported by the Set-Field action)

- TTL modification actions are still separate actions:
  - SET_NW_TTL
  - DEC_NW_TTL
  - SET_MPLS_TTL
  - DEC_MPLS_TTL

Multiple Controllers

- The switch may establish communication with a single controller, or may establish communication with **multiple controllers**.
- It improves **reliability**, as the switch can continue to operate in OpenFlow mode if one controller or controller connection fails.
- The hand-over between controllers is entirely managed by the controllers themselves, which enables **fast recovery from failure** and also **controller load balancing**.
- When OpenFlow operation is initiated, the switch must connect to all controllers it is configured with, and try to maintain connection with all of them concurrently.

Multiple Controllers

- There are three **controller roles**: *EQUAL*, *SLAVE* and *MASTER*.
- The default role of a controller is EQUAL. A controller can request its role to be changed by the new controller-to-switch message named ROLE-REQUEST. The switch informs a controller of a change of its role by the asynchronous ROLE-STATUS message.

  - **EQUAL**: The controller has **full access** to the switch and is equal to other controllers in the same role. The controller receives all the switch **asynchronous** messages, and it can send **controller-to-switch** commands to modify the state of the switch.
  - **SLAVE**: The controller has **read-only access** to the switch, and it is denied ability to send controller-to-switch commands that modify the state of the switch. By default, the controller does not receive switch asynchronous messages.
  - **MASTER**: This role is similar to EQUAL and has **full access** to the switch, the difference is that the switch ensures it is the only controller in this role. When a controller changes its role to MASTER, the switch changes the current controller with the role MASTER to have the role SLAVE, but does not affect controllers with role EQUAL.