Sharif University of Technology

Computer Engineering Department

Software-Defined Networking

Ali Movaghar

Mohammad Hosseini

OpenFlow – Part 3

TA: Iman Rahmati & Farbod Shahinfar

OpenFlow 1.3

- Version 1.3.0 released in 2012
- Revision 1.3.5 released in 2015

- Wire protocol version: 0x04

- Major changes:
  - Table miss strategy
  - Meter Table was added
  - Many implementation improvements, bug fixes, and clarifications

Table Miss

- ❖ The flow entry that wildcards all fields (all fields omitted) and has priority equal to 0 is called the **table-miss flow entry**.
- ❖ Every flow table must support a table-miss flow entry to process table misses.
- ❖ The table-miss flow entry specifies how to process packets unmatched by other flow entries in the flow table, and may, for example, send packets to the controller, drop packets or direct packets to a subsequent table.
- ❖ If the table-miss flow entry does not exist, by default packets unmatched by flow entries are dropped (discarded).

Meter Table

- A meter table consists of meter entries.

- Meters enable OpenFlow to implement various simple **QoS** operations, such as **rate-limiting**.

- A meter measures the rate of packets assigned to it and enables controlling the rate of those packets.

- Any flow entry can specify a meter using the new instruction “Meter” and direct the packet to it:
  - Meter *meter_id*
  - As the result of the metering, the packet may be dropped (depending on meter configuration and state). The meter measures and controls the rate of the aggregate of all flow entries to which it is attached.
  - In version 1.5, metering is done using the action “meter” instead of the instruction. As the result, multiple meters can be attached to a flow entry, and meters can be used in group buckets.

Meter Entry

- Meters are added/modified using a new controller-to-switch message named METER_MOD

Meter Identifier	Flags	Meter Bands	Counters

- Meter identifier: an unsigned integer uniquely identifying the meter
- Flags: a bitmap determining the meter settings such as its measurement unit (kb/s or packet/s)
- Meter bands: an unordered list of meter bands.
  - Each band specifies the rate at which the band applies and the way packets should be processed.
  - Each meter band includes:
    - **meter type**: defines how packets are processed (such as DROP or DSCP remark)
    - **rate**: defines the lowest rate at which the band can apply
  - The meter applies the meter band with the highest configured rate that is lower than the current measured rate.

Connection

❖ The default OpenFlow transport port was changed to 6653 (assigned by *IANA*)

❖ Optionally, the switch may allow the controller to initiate the connection.

❖ Auxiliary Connections: The OpenFlow channel may be composed of a **main connection** and multiple **auxiliary connections**.

- ➢ Auxiliary connections are created by the OpenFlow switch and may be configured by the controller.
- ➢ Each connection from the switch to the controller is identified by the switch ***Datapath ID*** and an ***Auxiliary ID***.
- ➢ They are helpful to improve the switch processing performance.
- ➢ The switch may service auxiliary connections with different priorities.
- ➢ Auxiliary connections are mostly useful to carry packet-in and packet-out messages.
- ➢ A barrier message applies only to the connection where it is used.

Asynchronous-Configuration

- New controller-to-switch messages for **Asynchronous-Configuration**:
  - **SET_ASYNC**, GET_ASYNC_REQUEST, GET_ASYNC_REPLY

- The Asynchronous-Configuration messages are used by the controller to set an additional filter on the asynchronous messages that it wants to receive on its OpenFlow channel.

- This is mostly useful when the switch connects to **multiple controllers**.

Other changes

- MULTIPART messages replaced STATS messages, and they are used to request statistics or state information from the switch.

- A cookie field was added to the PACKET_IN message.
- This field takes its value from the flow that sends the packet to the controller.
- Having the cookie in the PACKET_IN enables the controller to more efficiently classify PACKET_IN messages, rather than having to match the packet against the full flow table.

OpenFlow 1.4

- ❖ Version 1.4.0 released in 2013
- ❖ Revision 1.4.1 released in 2015

- ❖ Wire protocol version: 0x05

- ❖ Major changes:
  - ➢ Eviction, and vacancy events
  - ➢ The bundle mechanism
  - ➢ Flow monitoring

Eviction

- In previous versions, when a flow table is full, new flow entries are not inserted in the flow table and an error is returned to the controller.

- Reaching that point is pretty problematic, as the controller needs time to operate on the flow table and this may cause a disruption of service.

- **Eviction** adds a mechanism enabling the switch to automatically eliminate entries of lower importance to make space for newer entries.

- Eviction can be enabled for a flow table by the *TABLE_MOD* message.

- If eviction is enabled on a flow table, and if that flow table is full, an add request will use the eviction mechanism to try to make space for the new flow entry.

Eviction

- Three eviction process flags have defined for each flow table:
  - IMPORTANCE: the eviction process will use the flow entry ***importance*** field which is set by *FLOW_MOD* message for each flow.
    - A flow entry with lower importance will always be evicted before a flow entry with higher importance.
    - If a new flow entry to insert is less important than all existing flow entries, no eviction takes place.
  - LIFETIME: the eviction process will use the flow entry remaining lifetime, the shortest time to expiration, to perform eviction.
    - If the flag LIFETIME is the only flag set, eviction will be performed strictly in order of remaining lifetime, and permanent flow entries are never removed
  - OTHER: the eviction process will use other switch-defined factors, such as internal constraints, to perform eviction.

Vacancy Events

- Vacancy events add a mechanism enabling the controller to get an early warning based on a capacity threshold chosen by the controller.

- This allows the controller to react in advance and avoid getting the table full.

- Vacancy events can be enabled for a flow table by the *TABLE_MOD* message. The message can also set the vacancy thresholds (up/down thresholds which is a hysteresis mechanism) for each table.

- VACANCY_UP/VACANCY_DOWN events are sent to the controller using the TABLE_STATUS asynchronous message.

Bundle

- Bundle: a sequence of OpenFlow modification requests from the controller that is applied as a single operation.

- If all modifications in the bundle succeed, all of the modifications are retained, but if any errors arise, none of the modifications are retained.

- The first goal of bundles is to group related state changes on a switch so that all changes are applied together or that none of them are applied.
- The second goal is to better synchronize changes across a set of switches: bundles can be prepared and pre-validated on each switch and then applied by the controller approximately at the same time. It is useful in **consistent** network updates.

Bundle

❖ The controller can issue the following sequence of messages to apply a sequence of modifications (messages) together using a bundle:

1. `OFPBCT_OPEN_REQUEST bundle_id`
2. `OFPT_BUNDLE_ADD_MESSAGE bundle_id` *modification 1*
3. `OFPT_BUNDLE_ADD_MESSAGE bundle_id` ...
4. `OFPT_BUNDLE_ADD_MESSAGE bundle_id` *modification n*
5. `OFPBCT_CLOSE_REQUEST bundle_id`
6. `OFPBCT_COMMIT_REQUEST bundle_id`

❖ The messages included in a bundle are pre-validated (in terms of syntax, features support, and resource availability) as they are stored in the bundle. If a message validation error or a bundle error condition arises, an error message is returned.

❖ It minimizes the probability of errors when the bundle is applied using the commit message.

❖ If one or more message part of the bundle can not be applied without error, the commit fails and all messages that are part of the bundle will be discarded without being applied. When the commit fails, the switch generates an error message corresponding to the message that could not be applied.

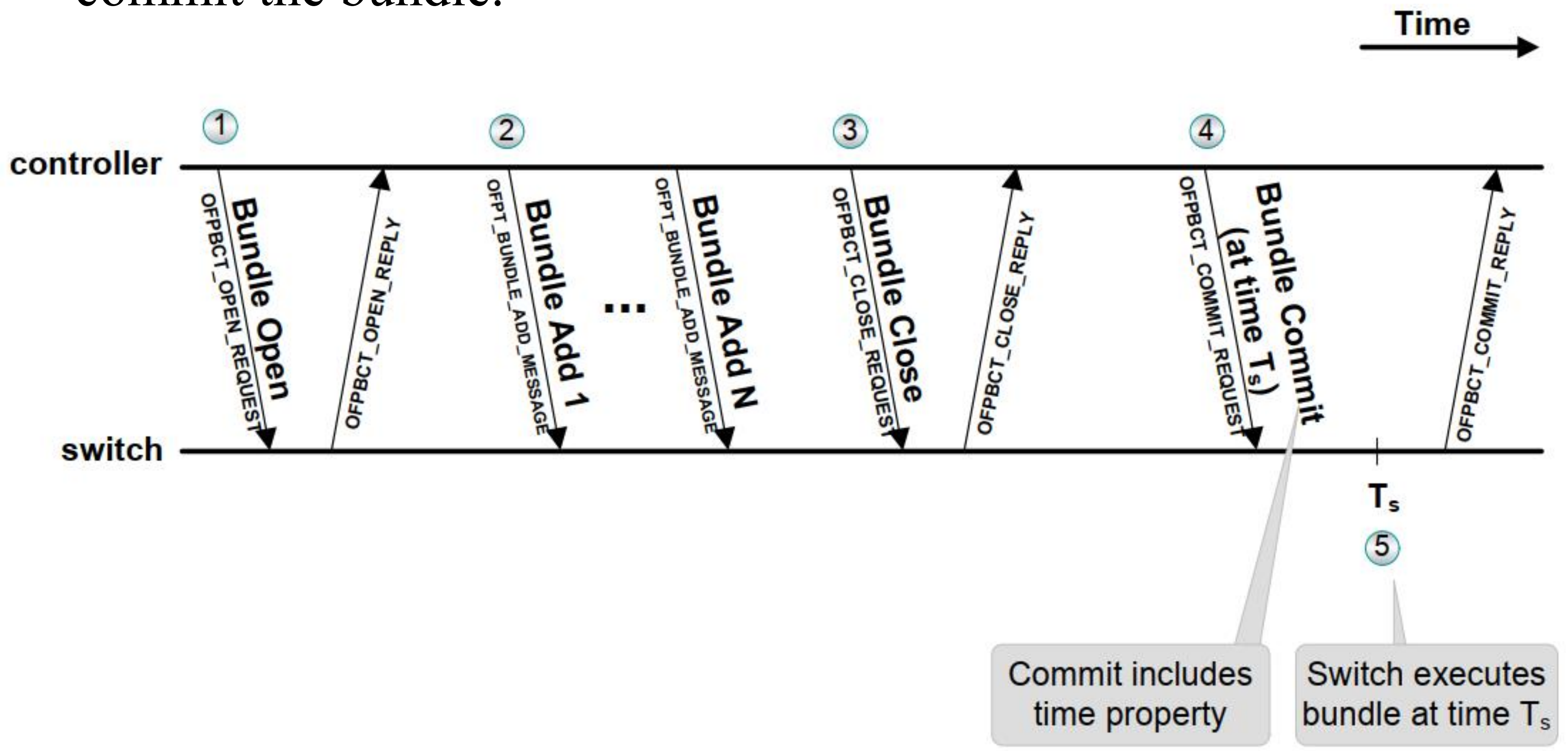
Bundle

- Committing a bundle is controller atomic: a controller querying the switch never sees the intermediate state, it sees either the state of the switch with none or with all of the modifications contained in the bundle.

- If the optional flag *OFPBF_ATOMIC* is specified, committing a bundle becomes packet atomic: a given packet from an input port or packet-out request is either processed with none or with all of the bundle modifications.

Scheduled Bundles

- In version 1.5, a bundle commit message may include an *execution time*, specifying when the switch is expected to commit the bundle.



- The controller may cancel a scheduled bundle by sending a BUNDLE_CONTROL message with type DISCARD_REQUEST.

Flow Monitoring

- In a multi-controller deployment, flow monitors enable a controller to be aware of changes made to the flow tables by other controllers.

- A controller can create a number of flow monitors, each flow monitor matching a subset of flow entries in some flow tables.

- Parameters of a flow monitor:
  - Table_ID
  - Flags (ADD/REMOVE/MODIFY flows)
  - Match_fields (can include wildcards)

- Whenever a change to a flow table matches some outstanding flow monitor, the switch must send a notification to the controller using an OFPMP_FLOW_MONITOR multipart reply.

OpenFlow 1.5

- Version 1.5.0 released in 2014
- Revision 1.5.1 released in 2015

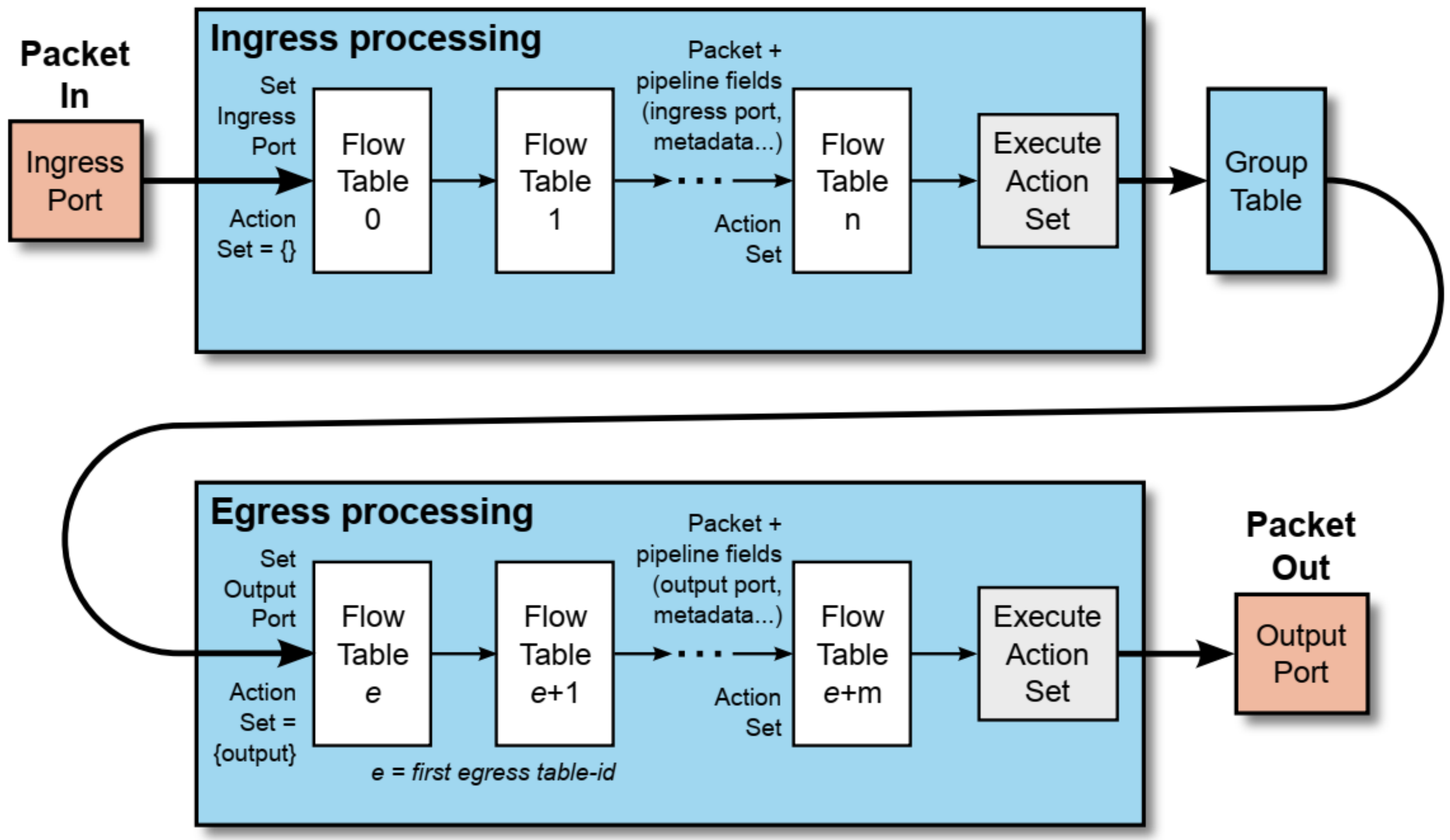
- Wire protocol version: 0x06

- Major changes:
  - Egress Tables
  - Extensible Flow Entry Statistics (OXS)
  - Statistics trigger
  - Copy-Field action

Egress Tables

- In previous versions of the specification, all processing was done in the context of the input port. Version 1.5 introduces **Egress Tables**, enabling processing to be done in the context of the output port.
- When a packet is output to a port, it will start processing at the first egress table where flow entries can define processing and redirect the packet to other egress tables.
- The behavior of egress table and egress flow entries is mostly similar to ingress.
- A new OXM field, OXM_OF_ACTSET_OUTPUT, enables egress flow entry to match the outgoing port.
- It is an **optional** feature which simplifies packet processing. (How come?)

Egress Tables



❖ Pipeline fields are carried from ingress to egress tables.

❖ Group processing and reserved port substitution happen before egress tables.

❖ The action set for egress processing is initialized at the beginning of egress processing with an output action for the current output port

Egress Tables

- Flow tables used for ingress processing can only direct packets via the Goto-Table instruction to ingress flow tables, and cannot direct packets to egress flow tables using the Goto-Table instruction.

- When the **ingress action set** contains an **output** action or a group action forwarding the packet to a port, the packet starts egress processing on that port.

- If the **list of actions** contains an **output** action, a clone (copy) of the packet is forwarded in its current state to the desired port where it starts egress processing.

- If the output action references the ALL reserved port, a clone of the packet starts egress processing for each relevant port.

Egress Tables

- ❖ In the egress tables, adding output/group actions to the action set is forbidden by write-action instruction, so that the packet output port can not be changed.

- ❖ However, the egress tables may optionally **support** the output/group actions in apply-action instruction. They forward clones of the packet to the specified ports, and those clones start egress processing from **the first egress table**.
  - ➢ This can be used for example for selective egress **mirroring**.

- ➢ Modifying fields such as Source MAC or Source IP address as a function of the output port is another application which can be facilitated by egress tables.

Extensible Flow Entry Statistics

- ❖ Previous versions of the specification use a fixed structure for flow entry statistics.
- ❖ Version 1.5 introduces a flexible encoding, OpenFlow eXtensible Statistics (OXS), to encode any arbitrary flow entry statistics.
- ❖ The existing flow statistics are now described (redefined) using OXS format, which is a compact TLV format.

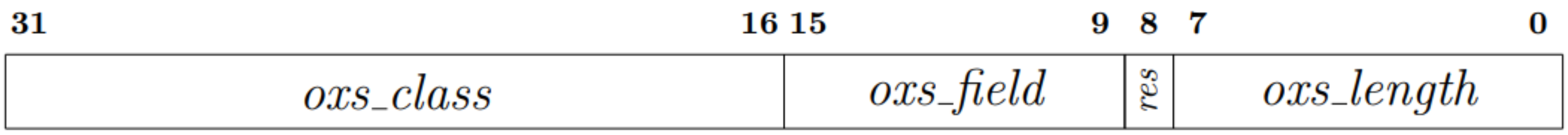


Figure 10: OXS TLV header layout.

Name		Width	Usage
oxs_type	oxs_class	16	Stat class: member class or reserved class
	oxs_field	7	Stat field within the class
	oxs_reserved	1	Reserved for future use
	oxs_length	8	Length of OXS payload

- ❖ The class OFPXSC_OPENFLOW_BASIC contains the basic set of OpenFlow stat fields: DURATION, IDLE_TIME, PACKET_COUNT, BYTE_COUNT

Flow Entry Statistics Trigger

- Polling flow entry statistics can induce high overhead and utilization for the switch.
- A new statistics trigger mechanism enable statistics to be automatically sent to the controller based on various statistics thresholds.

New Action

- **Copy-Field** ***src_field dst_field***
  - It is typically used to copy data
    - from a header field to a packet register pipeline field
    - from a packet register pipeline field to a header field
    - from a header field to another header field
  - Optional
  - A switch may not support all combinations of copies between header or pipeline fields.

- The packet register fields OXM_OF_PKT_REG(N) are used to store temporary values and information alongside the packet through pipeline processing.
- They are defined as a class of OXM (OFPXMC_PACKET_REGS).
- However, they usually cannot be matched in tables.
- Each packet register is 64 bits wide.
- They can be used with the set-field and copy-field actions.

Controller Connection Status

- ❖ The Controller Connection Status enables controller to know the status of all the connections from the switch to controllers.
- ❖ A controller can request the status, the roles and the control channels of other controllers configured on the switch.
- ❖ This allows a controller to detect control network partitioning or monitor the status of other controllers.