Software Defined Networking
Technical Report for Course Project

Karthik V.Nath Vaithianathan and Shashank Rao

Abstract— **The previous paper presented had provided the basic overview of software defined networking (SDN) as well as network function virtualization (NFV) concepts and basic architecture of the same. This paper provides an in-depth review into the technical aspects of the OpenFlow protocol, which is one of the most prominent open-source means of communication between the control and data planes in an SDN architecture.**

I. OpenFlow Standards

OpenFlow standards are specified by the Open Networking Foundation (ONF), a non-profit organization founded in early 2011 by Microsoft, Google, Verizon, Yahoo!, Duetsche Telekom, and Facebook. Currently, ONF has around 150 member organizations, collaborating to promote and develop open SDN standards.

OpenFlow is described in a number of documents - Openflow Switch Specification, OpenFlow Management and Configuration Protocol, OpenFlow Notifications Framework, OpenFlow Controller-Switch NDM Specification, OpenFlow Table Type Patterns, OpenFlow Conformance Test Specification. This paper mostly describes the OpenFlow switch protocol.

II. OpenFlow Components

The fundamental components of OpenFlow are:

- OpenFlow Controller
- OpenFlow Logical Switch, which contains ports, OpenFlow Secure Channel and flow table
- Group table
- Meter table
- Pipeline

The following section gives a brief description of each component:

A. Controller

The OpenFlow component of the controller is responsible for communicating instructions to the switch across the Secure Channel, but the OpenFlow protocol has no say in HOW the controller determines what instructions to send. All of that happens at a higher layer within or above the controller. The instructions might be determined by an automated orchestration component, by direct operator intervention, or by traditional path determination protocols such as OSPF or BGP. In fact, OpenFlow need not even be the only southbound communications protocol used by the controller. It might even communicate with other switches, or even the same switch, using some other open or proprietary protocol.

B. Switch

Switch is where most of the actions, with regards to packet processing, takes place. A set of tables - flow tables, group tables, and a meter table - are populated with instructions from the controller through the OpenFlow channel. Packets enter and exit processing through ports.

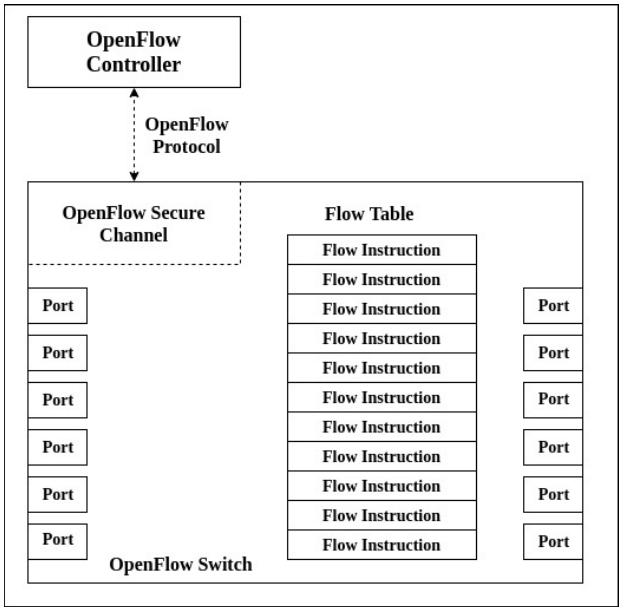

Fig. 1. OpenFlow Logical switch - A single OpenFlow capable switch can support multiple such OpenFlow logical switches

A switch can also be an OpenFlow-only switch (can process and forward packets only using the OpenFlow protocol) or a hybrid switch (supports both OpenFlow and traditional Ethernet switching including L2 switching, VLANs, L3 IPv4 and IPv6). Whether a switch is OpenFlow-only or hybrid makes a difference in some port functions.

C. Ports

Ports on an OpenFlow switch serve the same input/output purpose that they do on any switch. Packets are received on an ingress or input port, are processed, and switched to an egress or output port. A port can be added, changed, or removed in the switch configuration using, for example, OF-CONFIG or directly from the controller. A port state can also change, such as when the link goes down or comes up. Port changes do not change the flow entries pointing to that port. So for example, if a port goes down or is removed and a flow entry directs packets to that port, the packets are dropped.

For this reason, port changes must be communicated to the controller so that the controller can clean up any flow entries related to that port. OpenFlow defines three types of standard ports:

1) Physical ports: These correspond directly to the hardware interfaces on a switch. If a physical switch supports multiple OpenFlow logical switches, the hardware interfaces might be shared among several or all of the logical switches.
2) Logical ports: These include different kinds of logical interfaces such as tunnel interfaces, loopback interfaces, null interfaces, MPLS LSPs, and link aggregation groups. The packet handling of a logical port, such as encapsulation/de-encapsulation and whether it maps to a hardware interface, is implementation-specific and independent of OpenFlow.
3) Reserved ports: These are used for internal packet processing, for special functions such as flooding, or in a hybrid switch, for handoff from an OpenFlow logical switch to the 'normal' switching process. Reserved ports can be *required* or *optional*.

D. Secure Channel

The OpenFlow channel is the communications interface between the switch and the controller. Precedent to the connection startup, the switch must be configured with the controller's IP address so that the devices may find each other.

1) Channel Connections: The OpenFlow (OF) connection operates over TCP, and both the switch and the controller listen on default port 6653. The switch initiates the connection to the controller on startup or restart, originating traffic to the default port 6653 or a user-defined port to an ephemeral controller TCP port. Optionally, the switch can allow the controller to initiate the session. The connection is usually encrypted over Transport Layer Security (TLS) Protocol. Immediately after the TCP session is established and certificates are exchanged by TLS, the controller and switch exchange Hello messages to negotiate the OF version to use. The highest version that both sides can support is selected. At this point, the connection is up and controller and switch can exchange messages. The two sides use Echo Request/Reply process to monitor the connection and connection latency.

2) Connection Interruption: If one side cannot support the version advertised by the other side, it sends an error message and connection is closed. Similarly, if the failure of an established connection is detected either because echo replies are not received, because TLS session times out, or because of some other evidence of disconnect, the switch will enter one of the two states:

- Fail Secure Mode: The switch continues to operate, but does not try to send messages or packets to the controller. Entries in the flow tables continue to time out as they normally would.
- Fail Standalone Mode: The switch reverts to operating as a non-OpenFlow switch. This mode is used only by hybrid switches.

The switch also operates in one of these two modes when it starts and before it establishes a session with the controller.

3) Connection Reestablishment: When a switch reestablishes a session with a controller after a connection interruption, the existing entries in the flow table continue to be used. The controller can optionally request the switch to send it all flow entries, using a flow stats message, so that the controller can read the entries and synchronize its state to the switch accordingly. The controller also has the option of deleting the switch's flow entries and then synchronizing the switch by sending it a fresh set of flow entries using flow modification messages. The first option runs the risk of having inaccurate forwarding information and the second option disrupts all forwarding until the switch and controller are synchronized.

4) Reliable Message Delivery: The reliability of OpenFlow message delivery depends on the underlying TCP and TLS mechanisms. It does not use its own acknowledgements of messages, and it does not assume ordered delivery of messages. In fact, the switch can reorder the received messages from the controller to optimize performance.

5) Message Categories: OpenFlow messages between the controller and switch fall into one of the three categories:

- Controller-to-switch messages are used by the controller to manage the switch. The controller can add, modify, or delete flow tables; query the switch for features and statistics; configure the switch; set switch port properties; and send packets out a specified switch port.
- Asynchronous messages are sent from the switch to the controllers. Some examples are - messages whose packet or packet header does not match any flow entry and therefore needs to be processed at the controller; notification regarding change in flow state or port status; or an error messages.
- Symmetric messages can be sent by either the controller or switch. Examples are - hellos (keepalives), echo requests and replies, or experimenter messages that enable functionality.

6) Connecting to Multiple Controllers: A realistic OpenFlow design connects each switch to multiple controllers to eliminate single points of failure at the controller and at the connection to the controller. A switch connecting to multiple controllers must be able to categorize the relationship of the controllers to each other from the switch's perspective. A controller can play one of three roles:

- *Equal* is the default role. The switch exchanges the same messages with each controller, and does not distinguish between them, and does not load-balance among the controllers or arbitrate between the controllers. The switch relies on controllers to co-ordinate the instructions sent to the switch.
- *Master* is the same as *Equal* in terms of relationship between the switch and the controller, but the difference is that only a single controller can be *Master*.
- *Slave* is the role of all switches except the *Master*, when controllers are setup in a Master/Slave configuration. A

Slave controller cannot send any Controller-to-switch messages that would cause a change to the switch; nor can receive any asynchronous messages except Port Status messages. It can only send messages that query information from the switch or inform the switch of its role.

A switch cannot dictate on its own what role a controller plays, but rather, depends on Controller-to-switch messages called Role Request from the controllers to tell the switch what its role is. Although OpenFlow does not specify Master/Slave election mechanism between controllers, it does specify a 64-bit number called Generation-ID, in the Role Request message, that allows the switch to keep track of the most recent Role Request messages during a role change. This avoids a situation where multiple Slave controllers might declare themselves as the Master and the switch does not know who to believe.

7) Auxiliary Connections: Similar to how the switch can connect to more than one controller, we can have multiple connections between a switch and a controller. This not only provides redundancy, but also enables load balancing of messages and can improve overall connection performance between controller and switch. When there are multiple connections, one connection is the main and others are auxiliary connections. The main connection must be established first, and others are auxiliary connections. Two identifiers are used with auxiliary connections - Datapath ID and Auxiliary ID.

E. Flow Table

The following figure shows flow table entry format:

Match Fields	**Priority**	**Counters**	**Instructions**	**Timeouts**	**Cookie**	**Flags**

Fig. 2. The Flow Entry Format

- Match field: Specifies conditions under which packet is matched
- Priority field: Sets precedence of flow entry
- Counter field: Record statistics of matching packets
- Instruction field: To specify actions to be performed on matching packets
- Timeouts field: Specifies maximum amount of time or idle time before the entry is expired by the switch
- Cookie field: Used by the controller to filter flow entries
- Flags field: For altering the way flows are managed

There might be multiple flow entries that contain match conditions for a packet; the entry with the most conditions matching the packet is the matching entry, and the actions in that entry's instruction field are followed. The packet might be forwarded out of the switch on a physical or logical port, or it might be forwarded to the controller, flooded, or referred to default switch processing through a Flood or Normal reserved ports.The Timeouts field consists of two timer values for deleting flow entries - hard timeout and idle timeout. While a switch can remove a flow entry based on these timeouts, a controller can actively delete an entry with a Delete message.

The flow table in a switch is not the same thing as Forwarding Information Base (FIB). A FIB is a simple set of forwarding instructions mapping, at a minimum, a destination address to an outgoing port. It supports destination-based switching. An OpenFlow table is a sequential set of instructions matching multiple fields, and taking some action based on that match - it supports flow-based switching. If the matching condition in a flow entry is the destination address, and if the action is to forward matching packets out a port, then the flow entry behaves just like FIB. Thus, a flow table is a superset of FIB.

F. Pipeline

A switch with a single flow table doesn't scale. To remedy the limitations of a single-table switch, multiple tables and a pipeline processing mechanism were introduced, which allowed the user to create a hierarchy of processing options using a sort of "if-then-goto" logic. The OpenFlow pipeline is a series of flow tables, numbered sequentially starting with 0. All incoming packets must be processed through table 0, and then may be forwarded to some numerically higher table along with metadata that can be matched at the next table. The following figure shows how pipeline processing works.

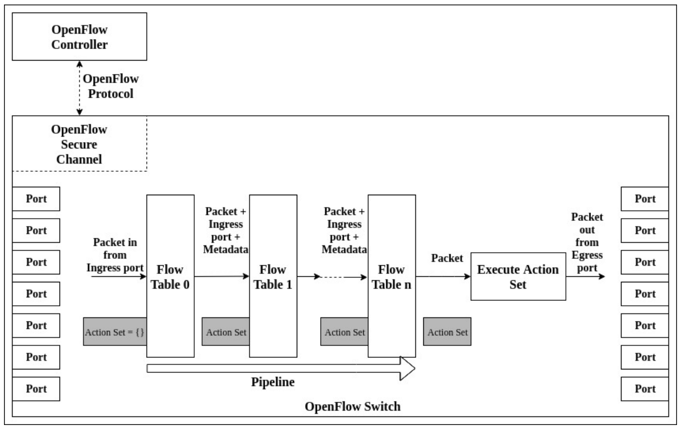

Fig. 3. Figure showing how pipeline is setup as well as information shared from table to table during pipeline processing

When a packet arrives on some port, the ingress port is recorded and the packet is sent to flow table 0. An Action Set is associated with the flow, which is initially an empty set. If a matching flow entry is found, the associated instructions in the entry are either executed or added to the Action Set. One of the instructions can be a Go-To instruction, which sends the packet to another, numerically higher table along with the associated Action Set, metadata (inter-table information) and the ingress port ID. The process is repeated at each table, and after the last table in the pipeline has processed the packet, the instructions in the Action Set are executed sequentially.

A Table-miss occurs if the packet does not match any flow entry at a given table. The switch has to know what to do with a packet in this event, so the controller should add a

Table-miss entry to each flow table. If Table-miss entry is not included in the table, the default action when no match is found in the table is to drop the packet. The figure shows only a single pipeline process, but different match conditions can be made to point to different tables, thus giving rise to a complex series of processing actions.

G. Group Table

Like a flow table, a group table consists of a set of entries (or group entries), each of which represents some group of packets that should be treated in the same way. A group is a means of applying a common set of output actions to aggregate flows; that is, by designating a packet as a member of a group, actions can be efficiently applied or changed across multiple flows. The following figure gives the Group Entry format:

Group Identifier	**Group Type**	**Counters**	**Action Buckets**

Fig. 4. Group Entry Format

Each group entry consists of:

- A 32-bit group identifier
- A group type, which is one of - All, Select, Indirect, Fast Failover
- A counter field to record statistics of matching packets
- Action buckets, which contain set of actions to be executed

H. Meter Table

A meter table allows OpenFlow to create a simple Quality Of Service (QoS) mechanism that measures the rate of a flow and then imposes a specified rate limit. Each entry on the meter table represents a meter. A meter is associated directly with a flow entry and measures the rate of matching flow.

III. The OpenFlow Protocol

A. Message Exchange between Switch and Controller

1) Message Header: The following figure shows the OpenFlow message header:

Version (8 bytes)	**Type (8 bytes)**	**Length (16 bytes)**
Transaction Identifier (XID) (32 bytes)		

Fig. 5. OpenFlow Message Header Format

- Version: The OpenFlow Switch Specification standard to which the message is compliant.
- Type: The type of message; Table 1 lists the different message types, along with how the message types are used to perform several OpenFlow operations.
- Length: Indicates the total length of the message
- Transaction ID (XID): Similar to a sequence number; its job is to identify a specific message, so that if multiple messages of the same type are received, a switch or controller can determine which one is more recent.

2) Message Structures: There are a number of structures that are used within OpenFlow messages to describe functional elements of the switch:

- Port structures: which define physical, logical and reserved ports. Details of the port include a 32-bit port number, the port type, port state and port speed for physical ports.
- Queue structures: which describe queues on the output port of the data path. Each queue is defined by - a 32 bit port number, a 32 bit queue ID, minimum and maximum rate properties and flow match structures (which describe matching packets or flow entries)

B. Flow Processing

The heart of OpenFlow is the processing and switching of data flows. A flow table is a list of one or more entries, and each entry has a match condition, an action to take on any packets that match, and possibly an accounting function to count how many matches have occurred. The list of flow entries making up a flow table, or multiple flow tables tied together in a pipeline, comprise flow processing.

1) Match Parameters: OpenFLow match structures are built using a Type/Length/Value (TLV) format, which makes changing or expanding match parameters between versions much easier. OpenFlow match conditions fall into one of the following categories:

- Flow Match: A flow is usually identified by a combination of parameters such as incoming port, L2 and L3 source and destination addresses, Class of Service (CoS) bits, and upper layer ports.
- Header Match: In addition to flows, a flow entry can match specific packets based on the contents of its Layer 2,3,4 headers.
- Pipeline Match: Pipeline match fields are the information attached to a packet, other than the packet header, for pipeline processing.
- Experimenter flow match: This is an optional category that supports experimentation.

2) Instructions And Actions: When a match is found, things start happening. An individual flow table consists of multiple flow entries, and and individual flow entry consists of, among other things, a set of match criteria and one or more Instructions to be executed when the match occurs (Fig.2). As shown in the figure below, this set of one or more instructions in a flow entry is an Instruction set. An individual Instruction in the Instruction Set might or might not have a list of actions - an Action List - associated with it. An Action Set is passed between flow tables during flow processing (as seen in Fig. 3). An Action List, on the other hand, is associated within a flow entry of a single flow table.

Instructions:

There are six types of instructions. Some instruction types have actions associated with them, and others do not, but

Table 1 - OpenFlow Message Types and Operations

Function	Messages Used	Type Number	Message Category	Description
Handshake	FEATURES REQUEST	6	Controller/Switch Message	Identify a switch at startup and describe the switch capabilities
	FEATURES REPLY	6	Controller/Switch Message	
Switch Configuration	GET CONFIG REQUEST	7	Controller/Switch Message	The controller can request configuration information and make configuration changes
	GET CONFIG REPLY	8	Controller/Switch Message	
	SET CONFIG	9	Controller/Switch Message	
Flow table Configuration	TABLE MOD	17	Controller/Switch Message	Change a flow table configuration
Modify Flow Entry	FLOW MOD	14	Controller/Switch Message	Add, change or remove flow entries, and specify parameters such as timeouts
Modify Group Entry	GROUP MOD	15	Controller/Switch Message	Add, change or remove group table entries, and specify parameters such as action buckets
Port Modification	PORT MOD	16	Controller/Switch Message	Modify behavior of a physical port
Meter Modification	METER MOD	29	Controller/Switch Message	Add, change or remove meter table, and specify parameters such as rate and burst size
Multipart Data Request	MULTIPART REQUEST	18	Controller/Switch Message	Request data from the switch that might not fit into a single normal OpenFlow message, which is limited in size to 64KB, such as flow, queue or port statistics or switch manufacturer information
	MULTIPART REPLY	19	Controller/Switch Message	
Queue Configuration	QUEUE GET CONFIG REQUEST	22	Controller/Switch Message	Request Queue information
	QUEUE GET CONFIG REPLY	23	Controller/Switch Message	
Packet Send	PACKET OUT	13	Controller/Switch Message	Send a packet through the datapath
Barrier	BARRIER REQUEST	20	Controller/Switch Message	Controller requests notifications about completed operations or ensures dependencies
	BARRIER REPLY	21	Controller/Switch Message	
Role change	ROLE REQUEST	24	Controller/Switch Message	The controller notifies the switch of its role (Equal, Master, Slave) or a change in its role
	ROLE REPLY	25	Controller/Switch Message	
Set asynchronous configuration	GET ASYNC REQUEST	26	Controller/Switch Message	The controller specifies the asynchronous messages it wants to receive, other than error notifications, or queries for current asynchronous messages it will be sent
	GET ASYNC REPLY	27	Controller/Switch Message	
	SET ASYNC	28	Controller/Switch Message	
Packet In	PACKET IN	10	Async Message	The switch sends a received packet to the controller, and specifies the reason the packet is being sent (such as no flow match or invalid TTL)
Flow Removed	FLOW REMOVED	11	Async Message	Switch notifies the controller that a flow entry has been removed due to timeout or some other reason
Port Status	PORT STATUS	12	Async Message	Switch notifies the controller that a port has been added, changed or removed
Error	ERROR	1	Async Message	The switch notifies the controller of a problem and provides information about the nature of the problem
Hello	HELLO	0	Symmetric Message	Used during version negotiation, and as a keepalive
Echo	ECHO REQUEST	2	Symmetric Message	A "ping" mechanism used for such things as latency or bandwidth measurement
	ECHO REPLY	3	Symmetric Message	
Experimenter	EXPERIMENTER	4	Symmetric Message	A message with undefined content that can be used by experimenters

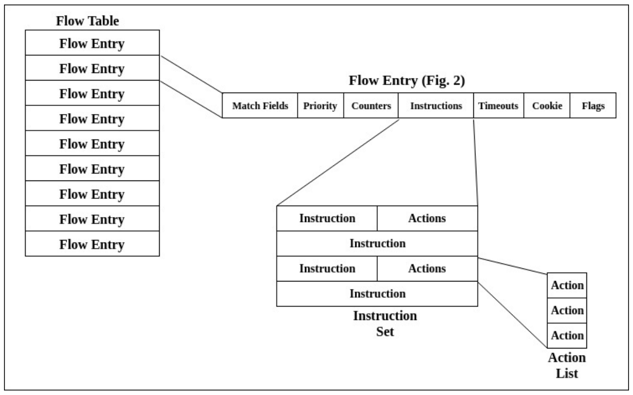

Fig. 6. Relationship of Instructions, an Instruction set, Actions and an Action List

all instructions cause a change of some type - to a packet, an action set, or to a pipeline processing. Instructions types can be optional or required. A controller can query a switch to discover which, if any, optional instructions the switch supports. If a switch cannot execute an instruction in a flow entry given to it by a controller, it rejects the flow entry and sends an error message to the controller. The following are the six instruction types:

- Write-Actions (Required type): Adds specified Action List into the Action Set; if one of the specified actions already exists in the Action Set, overwrite it.
- Apply-Actions (Optional type): Executes the specified Action List immediately, without changing the Action Set.
- Clear-Actions (Optional type): Clears all actions in the Action Set immediately.
- Goto-Table (Required type): Indicates the Table ID of the next table in the processing pipeline; a matching flow entry without a Goto-Table instruction indicates the end of the pipeline, and the Action Set is executed.
- Meter (Optional type): Sends the packet to the specified meter.
- Write-Metadata (Optional type): Write the masked metadata value into the metadata field for pipeline processing.

Instruction Set:

An Instruction Set comprises the 'Instructions' field in a flow entry, and consists of one or more individual Instructions. An individual Instruction Set can contain only one Instruction of each type. For example, an Instruction Set can contain an Apply-Actions instruction, a Write-Actions instruction, and a Goto-Table instruction, but cannot contain two different Goto-Table instructions. Thus, Instruction Set can never contain more than six Instructions. This makes sense, since, we should not be able to send a single matching packet to more than one downstream table in the pipeline, or to more than one meter, or associate more than one metadata value with the packet. If Action Set needs to cleared in the pipeline, we need to do it only once for any matching packet. If multiple Actions need to be executed or multiple Actions need to be written to an Action Set, we can associate multiple Actions in a single Action List for each of those instructions. Although a given Instruction Set might not contain Instructions of all six types, Instructions in a set are always executed in the following order - Meter, Apply-Actions, Clear-Actions, Write-Actions, Write-Metadata, Goto-Table.

Action Lists:

An Action List consists of one or more individual actions, and is associated with either an Apply-Actions instruction, Write-Actions instruction or the PACKET OUT message.

- The Apply-Actions instruction immediately executes the actions in the Action List. The actions are executed consecutively, in the order they appear on the list. Execution of the actions can change the matched packets in some way, but doesn't change the Action Set associated with the packet for pipeline processing.
- The Write-Actions instruction writes actions on the Action List to the Action Set, used on pipeline processing. The actions on the list are not executed by this instruction.
- The controller uses the PACKET OUT message to send a packet into the switch's dataflow. It can contain an Action List that instructs the switch to modify the packet in some way or to send the packet to a group or to an output port.

Actions:

Like Instructions, a switch is required to support certain Actions while support of others is optional. The Action format includes a header that specifies the Action type number, and a length. Following are some example of Action types:

- OUTPUT "port": Forwards the packet to a specified OpenFlow port (physical, logical or reserved). The port is represented with a 32-bit number.
- COPY_TTL_OUT: Copy the value of the TTL field in the next-to-outermost header to the TTL field outermost header. The copy can be IP-to-IP, MPLS-to-MPLS, or IP-to-MPLS.
- COPY_TTL_IN: Copy the value of the TTL field in the outermost header to the TTL field next-to-outermost header. The copy can be IP-to-IP, MPLS-to-MPLS, or MPLS-to-IP.
- SET_MPLS_TTL "mpls_ttl": Change the value of the TTL field in the outermost MPLS header to the specified value.
- DEC_MPLS_TTL: Decrement the value of the TTL field in the outermost MPLS header by 1.
- PUSH_VLAN "ethertype": Push a new VLAN header (C-TAG or S-TAG) into the packet. If a VLAN header already exists, the pushed VLAN tag becomes the outermost tag. The ethertype field must be 0x8100 or 0x88a8.
- POP_VLAN: Pop the outermost VLAN header from the packet.
- PUSH_MPLS "ethertype": Push a new MPLS shim header into the packet. If an MPLS header already exists, the pushed MPLS header becomes the outermost label. The ethertype field must be either 0x8847 or 0x8848.
- POP_MPLS: Pops the outermost MPLS label or shim header from the packet.
- SET_QUEUE "queue_id": Map the packet to the specified port queue, regardless of the value of IP DSCP or VLAN PCP bits.
- GROUP "group_id": Process the packet in the specified group.
- SET_NW_TTL "ip_ttl": Change the value of the IPv4 TTL field or the IPv6 Hop Limit field to the specified value, and updates the IP checksum accordingly.
- DEC_NW_TTL: Decrement the value of IPv4 TTL field or IPv6 Hop Limit field by 1, and update IP checksum accordingly.
- SET_FIELD: Overwrite a field in the packet to a value specified in the Type/Length/Value (TLV) expression. The action applies to the outermost header, if there are multiple headers. The header CRC value is recalculated if applicable.
- DROP: DROP is not an explicit or specified action; rather, it is a default action taken when an Instruction Set or Action Bucket is empty.

There are more actions other than the ones listed that perform changes on the matched packet. An Action List can include multiple PUSH or POP (VLAN, MPLS or PBB) actions, to add or remove multiple tags.

Action Sets:

An Action Set is used in pipeline processing. An incoming packet has an empty Action Set attached to it, and as the packet is processed across one or more tables, matching flow entries use Write-Actions instructions to add actions to the Action Set or Clear-Actions instructions to delete actions from the Action Set. After the last flow table on the pipeline has processed the packet, the actions in the Action Set are executed.

An Action Set can contain only one action of each type (except for the SET_FIELD action).

Action Buckets:

An Action Bucket contains one or more Action Sets. They are associated with groups. They consist of the Action Sets of various packets set to a given group, and a matching Group Entry (Fig. 4) specifies one or more Action Buckets to be executed.

Counters:

OpenFlow maintains multiple counters for each of the following - flow table, flow entry, port, queue, group, group bucket, meter and meter band. The counters wrap when they exceed the maximum bits, and there is no overflow indicator. Each flow entry, port, group queue, and meter has a Duration counter that tracks the amount of time the entity has been installed in the switch (in seconds).

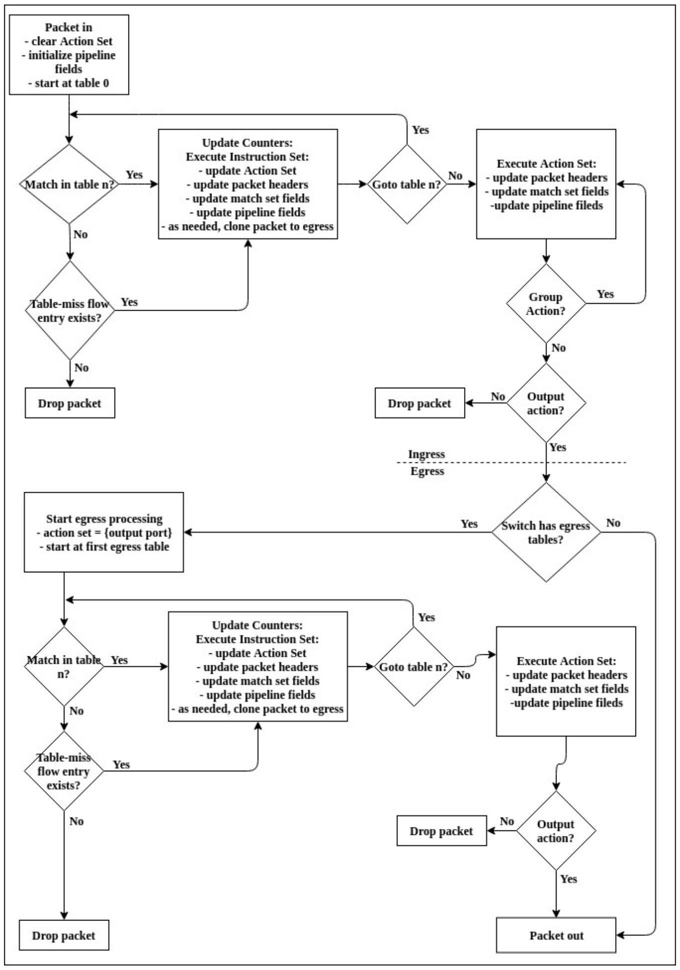

Fig. 7. Simplified flowchart detailing packet flow through an OpenFlow Switch

Figure 7 shows a simplified flowchart detailing the flow of a packet through an OpenFlow switch; this is a simplified summary of all the processes described before.

IV. Conclusion

During the course of the project, we have summarized the entire architecture as well as protocols of OpenFlow protocol - starting with each of the basic component of OpenFlow, and building it up until the entire flow processing was understood. This has enabled us to gain deep understanding in the data plane side of SDN architecture, as well as understanding of the Southbound API's functionalities. As a result of the project, we will be easily be able to understand and apply our knowledge to implement the data plane of any SDN architecture.

References

[1] William Stallings, Foundations of Modern Networking - SDN, NFV, QoE, IoT and Cloud; Pearson Publications: November, 2015

[2] Doug Marschke, Jeff Doyle and Pete Mayer, SDN: Anatomy of OpenFlow; Lulu publications: March, 2015

[3] Open Data Center Alliance. Open Data Center Alliance Master Usage Model: Software-Defined Networking Rev. 2.0. White Paper. 2014.

[4] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. ONF White Paper, April 13, 2012.

[5] ISG NFV. Network Functions Virtualization: An Introduction, Benefits, Enablers, Challenges & Call for Action. ISG NFV white paper, October 2012.

[6] ETSI GS NFV Architectural Framework 002 v1.1.1, October-2013