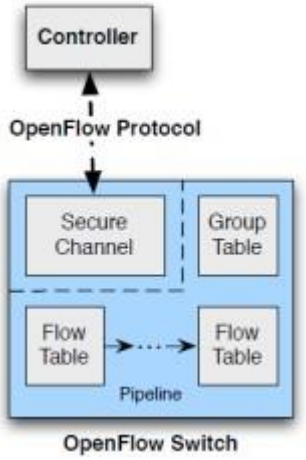
Course Title: Telecommunication Engineering
Assignment Name: OpenFlow Protocol
Assignment No: 02
Name: Tusar Sarker
ID: IT- 16015

Theory:

This speciation covers the components and the basic functions of the switch, and the OpenFlow protocol to manage an OpenFlow switch from a remote controller.



Switch Components:

An OpenFlow Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and an OpenFlow channel to an external controller .The switch communicates with the controller and the controller manages the switch via the

OpenFlow protocol:

Using the OpenFlow protocol, the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets) and

proactively. Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets. Matching starts at the first flow table and may continue to additional flow tables. Flow entries match packets in priority order, with the first matching entry in each table being used. If a matching entry is found, the instructions associated with the specific flow entry are executed. If no match is found in a flow table, the outcome depends on configuration of the table-miss flow entry: for example, the packet may be forwarded to the controller over the OpenFlow channel, dropped, or may continue to the next flow table. Instructions associated with each flow entry either contain actions or modify pipeline processing.

Actions included in instructions describe packet forwarding, packet modification and group table processing. Pipeline processing instructions allow packets to be sent to subsequent tables for further processing and allow information, in the form of metadata, to be communicated between tables. Table pipeline processing stops when the instruction set associated with a matching flow entry does not specify a next table; at this point the packet is usually modified and forwarded.

Flow entries may forward to a port. This is usually a physical port, but it may also be a logical port defined by the switch or a reserved port defined by this specification. Reserved ports may specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as normal switch processing while switch-defined logical ports may specify link aggregation groups, tunnels or loopback interfaces .Actions associated with flow entries may also direct packets to a group, which specifies additional processing. Groups represent sets of actions for flooding, as well as more complex forwarding semantics (e.g. multipath, fast reroute,

and link aggregation). As a general layer of indirection, groups also enable multiple flow entries to forward to a single identifier (e.g. IP forwarding to a common next hop). This abstraction allows common output actions across flow entries to be changed efficiently.

OpenFlow specification terms:

This section describes key OpenFlow specification terms:

 Byte: an 8-bit octet.

 Packet: an Ethernet frame, including header and payload.

 Port: where packets enter and exit the OpenFlow pipeline. May be a physical port, a logical port defined by the switch, or a reserved port defined by the OpenFlow protocol.

 Pipeline: the set of linked flow tables that provide matching, forwarding, and packet modifications in an OpenFlow switch.

 Flow Table: A stage of the pipeline, contains flow entries.

 Flow Entry: an element in a flow table used to match and process packets. It contains a set of match fields for matching packets, a priority for matching precedence, a set of counters to track packets, and a set of instructions to apply.

 Match Field: A field against which a packet is matched, including packet headers, the ingress port, and the metadata value. A match field may be wild carded (match any value) and in some cases bit masked.

 Metadata: a maskable register value that is used to carry information from one table to the next.

 Instruction: Instructions are attached to a flow entry and describe the OpenFlow processing that happen when a packet matches the flow entry. An instruction either modifies pipeline processing, such as direct the packet to another flow table, or contains a set of actions to add to the action set, or contains a list of actions to apply immediately to the packet.

 Action: an operation that forwards the packet to a port or modifies the packet, such as decrementing the TTL field. Actions may be specified as part of the instruction set associated with a flow entry or in an action bucket associated with a group entry. Actions may be accumulated in the Action Set of the packet or applied immediately to the packet.

 Action Set: a set of actions associated with the packet that are accumulated while the packet is processed by each table and that are executed when the instruction set instructs the packet to exit the processing pipeline.

 Group: a list of action buckets and some means of choosing one or more of those buckets to apply on a per-packet basis.

 Action Bucket: a set of actions and associated parameters, defined for groups.

 Tag: a header that can be inserted or removed from a packet via push and pop actions.

 Outermost Tag: the tag that appears closest to the beginning of a packet.

 Controller: An entity interacting with the OpenFlow switches using the OpenFlow protocol.

 Meter: a switch element that can measure and control the rate of packets. The meter trigger a meter band if the packet rate or byte rate passing through the meter exceed a predefined threshold. If the meter band drops the packet, it is called a Rate Limiter.

Question 5.1: How the RYU GUI interface can be improved? Provide some ideas.

Answer: Best Practices for Designing an Interface:-

- Keep the **interface** simple. ...
- Create consistency and use common **UI** elements. ...
- Be purposeful in page layout. ...
- Strategically use color and texture. ...
- Use typography to create hierarchy and clarity. ...
- Make sure that the system communicates what's happening. ...
- Think about the defaults.

Question 5.2: Explain at least two the advantage and disadvantage of using mininet or Zodiac FX?

Answer: Advantages of using mininet:-

- Provides a simple and inexpensive network testbed for developing OpenFlow applications

• Enables multiple concurrent developers to work independently on the same topology

• Supports system-level regression tests, which are repeatable and easily packaged

• Enables complex topology testing, without the need to wire up a physical network

• Includes a CLI that is topology-aware and OpenFlow-aware, for debugging or running network-wide tests

• Supports arbitrary custom topologies, and includes a basic set of parametrized topologies

• is usable out of the box without programming, but

• also Provides a straightforward and extensible Python API for network creation and experimentation

Disadvantages of using mininet:

Mininet is great but it has some limit.Those are:-

- Running on a single system is convenient, but it imposes resource limits: if your server has 3 GHz of CPU and can switch about 10 Gbps of simulated traffic, those resources will need to be balanced and shared among your virtual hosts and switches.
- Mininet uses a single Linux kernel for all virtual hosts; this means that you can't run software that depends on BSD, Windows, or other operating system kernels. (Although you can attach VMs to Mininet.)
- Mininet won't write your OpenFlow controller for you; if you need custom routing or switching behavior, you will need to find or develop a controller with the features you require.
- By default your Mininet network is isolated from your LAN and from the internet – this is usually a good thing! However, you may use the NAT object and/or the --natoption to connect your Mininet network to your LAN via Network Address Translation. You can also attach a real (or virtual) hardware interface to your Mininet network (seeexamples/hwintf.py for details.)
- By default all Mininet hosts share the host file system and PID space; this means that you may have to be careful if you are

running daemons that require configuration in /etc, and you need to be careful that you don't kill the wrong processes by mistake. (Note the bind.py example demonstrates how to have per-host private directories.)

- Unlike a simulator, Mininet doesn't have a strong notion of virtual time; this means that timing measurements will be based on real time, and that faster-than-real-time results (e.g. 100 Gbps networks) cannot easily be emulated.

Question 5.3: Explain how the open flow tables are created?

Answer: Flow Tables:

Using the OpenFlow switch protocol, the controller can add, update, and delete flow entries in *flow tables*, both reactively (in response to packets) and proactively.

Reactive Flow Entries: are created when the controller dynamically learns where devices are in the topology and must update the flow tables on those devices to build end-to-end connectivity. For example, since the switches in a pure OpenFlow environment are simply forwarders of traffic, all rational logic must first be dictated and programmed by the controller. So, if a host on switch A needs to talk to a host switch B, messages will be sent to the controller to find out how to get to this host. The controller will learn the host MAC address tables of the switches and how they connect, programming the logic into the *flow tables* of each switch. This is a reactive flow entry.

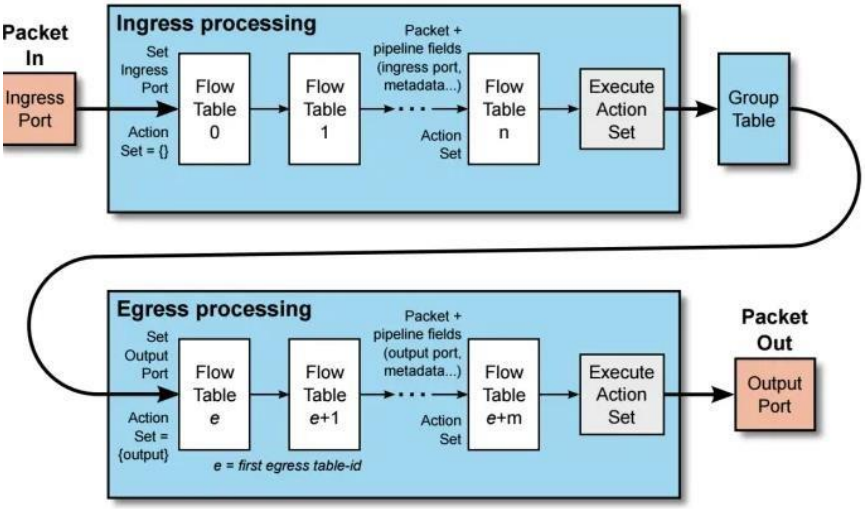
Proactive Flow Entries are programmed before traffic arrives. If it's already known that two devices should or should not communicate, the controller can program these *flow entries* on the OpenFlow endpoints ahead of time**.**

Traffic Matching, Pipeline Processing, and Flow Table Navigation:

In an OpenFlow network, each OpenFlow switch contains at least 1 *flow table* and a set of *flow entries* within that table. These *flow entries* contain *match fields*, *counters* and *instructions* to apply to matched packets.

Typically, you'll have more than a single *flow table*, so it's important to note that matching starts at the **first** flow table and may continue to additional flow tables of the pipeline. The packet will first start in table 0 and check those entries based on priority. Highest priority will match first (e.g. 200, then 100, then 1). If the flow needs to continue to another table, *goto* statement tells the packet to go to the table specified in the instructions.

This pipeline processing will happen in two stages, *ingress processing* and *egress processing*.



If a matching entry is found, the instructions associated with the specific flow entry are executed. If no match is found in a flow table, the outcome depends on the configuration of the Table-miss flow entry.

Table-miss flow entry:

The Table-miss flow entry is the last in the table, has a priority of 0 and a match of anything. It's like a catch-all, and the actions to be taken depend on how you configure it. You can forward the packet to the controller over the OpenFlow Channel, or you could drop the packet, or continue with the next flow table.

Conclusion: The **OpenFlow** protocol allows a server to instruct network switches where to send data packets. In a non-**OpenFlow** or legacy switch, packet forwarding (the data path) and route determi- nation (the control path) occur on the same device. A switch using the **OpenFlow** protocol separates the data path from the control path.

OpenFlow specifies the communication between the control plane and the data plane. It implements one of the first standards in SDN enabling the SDN controller to interact directly with the forwarding nodes – switches and routers – in a network.

The control logic triggers queue settings and forwarding tables through OpenFlow. It is also possible to collect traffic traces and destination addresses from the nodes. The queues in the nodes are assumed to be configurable in terms of bandwidth, which could be specified as a minimum/maximum rate pair . Version 1.3 of OpenFlow supports slicing, that is, multiple configurable queues per port. Similarly, the buffer sizes assigned to the queues should be configurable. In principle, it is sufficient to be able to specify a maximum buffer size per queue.

This is particularly important to limit the impact of self-similar traffic on the rest of the network .

The update of packet forwarding tables in switches and routers is a standard operation, where globally optimal routes are determined by the central logic. We also assume that traffic traces are available, either via OpenFlow or some other means. Full traces rather than statistics are required for reasonable accuracy on short time scales.

OpenFlow implements the Orchestrator-Forwarder interface for manipulating flow tables and requests traffic statistics and network state information from the nodes to model the network topology and for load monitoring. For large networks, we expect to have multiple orchestrators, and OpenFlow is used to share information between these entities.
We will need to classify traffic efficiently, and ideally use lower-order protocol header fields to extract this information. Examples are traffic class header field in MPLS, the type of service field in IPv4, traffic class in IPv6, and possibly the source IP address and source and/or destination ports.