In class Test 3　2254411 Enze.Zhou

1. **What is the initial flow entry (before attacking) installed in the switch of this lab?**

Answer：It is the table-miss flow entry. Defined in the switch_features_handler function:

```
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # install table-miss flow entry
    #
    # We specify NO BUFFER to max_len of the output action due to
    # OVS bug. At this moment, if we specify a lesser number, e.g.,
    # 128, OVS will send Packet-In with invalid buffer_id and
    # truncated packet data. In that case, we cannot output packets
    # correctly.  The bug has been fixed in OVS v2.1.0.
    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                      ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)
```

2. **Explain what this initial flow entry does.**

Answer: What it does is:

(1) If the priority is 0, this rule is used after all other flow table rules become invalid, that is, it is the default rule when other flow table entries do not match

(2) Match all packets (empty matching rule)

(3) All unmatched packets are forwarded to the controller, who decides whether to discard the packets, forward the packets, or install specific flow table rules for subsequent packets

(4) Using NO_BUFFER to process the complete packet, the switch will send the complete packet to the controller

(5) If there is no table-miss rule, the unmatched packets will be directly discarded, and the table-miss rule will deal with the unmatched packets as a last resort

3. **Show the flow ‘match’ rule (in the lab11.py) used to cope with the above flooding traffic.**

Answer: The above flooding traffic is the UDP flooding traffic, the ‘match’ rule is:

match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, in_port=in_port, ipv4_src=srcip, ipv4_dst=dstip, ip_proto=protocol, udp_src=u.src_port, udp_dst=u.dst_port,)

```
            #  If UDP Protocol
            elif protocol == in_proto.IPPROTO_UDP:
                u = pkt.get_protocol(udp.udp)
                match =
parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, in_port=in_port,
ipv4_src=srcip, ipv4_dst=dstip, ip_proto=protocol, udp_src=u.src_port,
udp_dst=u.dst_port,)
```

For the other packets, each has a corresponding ‘match’ rule, like:

```
                # if ICMP Protocol
                if protocol == in_proto.IPPROTO_ICMP:
                    match =
parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, in_port=in_port,
ipv4_src=srcip, ipv4_dst=dstip, ip_proto=protocol)

                #  if TCP Protocol
                elif protocol == in_proto.IPPROTO_TCP:
                    t = pkt.get_protocol(tcp.tcp)
                    match =
parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, in_port=in_port,
ipv4_src=srcip, ipv4_dst=dstip, ip_proto=protocol, tcp_src=t.src_port,
tcp_dst=t.dst_port,)
```

4. Explain how the flooding command exhausts the switch’s flow table?

Answer: According to “h1 hping3 h2 -c 10000 -s --flood --rand-source” this command, a large number of packets with random source IP addresses are generated through --rand-source. Since ipv4_src matches the rule Packet, the switch sends a packet-in message to the controller for each unique source IP address. Since there is no matching rule, the controller installs a new flow table rule for each source IP address. However, the flow table capacity is limited, and this process consumes the switch memory rapidly and fills the flow table quickly. When the flow table is full, performance degrades significantly, the switch cannot install new flow table rules, the switch flow table is exhausted, and it begins to discard packets.