BÀI THÍ NGHIỆM

4

ĐỊNH TUYẾN SỬ DỤNG THUẬT TOÁN BELLMAN

I. MỤC ĐÍCH THÍ NGHIỆM

Bài thí nghiệm này giúp sinh viên làm quen với cách tạo một module trong POX Controller và cách sử dụng các thuật toán để tạo ra module định tuyến lớp 3 sử dụng thuật toán Bellman

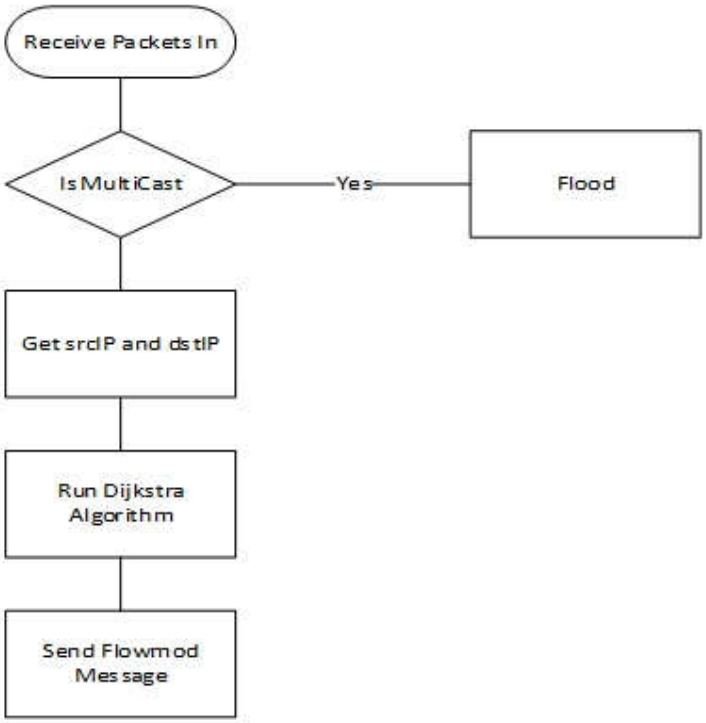
II. THẢO LUẬN

Module định tuyến lớp 3 là module sẽ giúp định hình đường đi cho các gói tin trong mạng, khi chạy module định tuyến lớp 3 sẽ không cần chạy module l2_learning.

Sau đây là các bước thực hiện của module định tuyến lớp 3.

- ✓ Bước 1: sinh viên tạo một topology bằng Mininet
- ✓ Bước 2: Khi nhận 1 packet đến, module định tuyến sẽ kiểm tra xem có phải là bản tin MultiCast không, nếu là bản tin multicast thì sẽ gửi lệnh flood gói tin xuống cho switch thực hiện, nếu không thì module sẽ lấy địa chỉ IP nguồn và IP đích của gói tin, sau đó sẽ chạy thuật toán Dijkstra để tìm đường đi cho gói tin này. Thuật toán Dijkstra sẽ lấy thông tin về topo từ module *discovery,* giả sử đã biết IP của tất cả các host trong mạng. Sau khi có đường đi cho gói tin, module định tuyến sẽ tiến hành tạo các gói tin flow_mod để gửi xuống switch.

Toàn bộ quá trình được thể hiện qua lưu đồ thuật toán sau:



Hình 1. Lưu đồ thuật toán

❑ **Một số hàm cần dùng**

- **connection.send():** hàm để gửi một bản tin OpenFlow xuống switch
- **ofp_action_output class:** Class này cung cấp hai bản tin: *ofp_packet_out* và *ofp_flow_mod*. *ofp_packet_out* sẽ gửi các bản tin điều khiển switch, *ofp_flow_mod* sẽ gửi một bản tin yêu cầu switch thêm một flow-entry vào bảng flow-table của switch. Các bản tin này có định nghĩa port mà người dùng cần forward gói tin. Ví dụ khi người dùng muốn flood gói tin thì sẽ dùng hàm:

```
out_action = of.ofp_action_output(port = of.OFPP_FLOOD)
```

- **ofp_match_class:** lớp này cung cấp các phương thức giúp người dùng định nghĩa các trường header của gói tin khớp. Một số trường thường dùng

✓ **dl_src:** địa chỉ MAC nguồn

✓ **dl_dst:** địa chỉ MAC đích

✓ **in_port:** port mà gói tin đến

Ví dụ tạo một đối tượng *match* các packet đến từ port 3

```
match = of.ofp_match()
match.in_port = 3
```

Ví dụ hàm gửi gói tin đầy đủ

```
def send_packet (self, buffer_id, raw_data, out_port, in_port):
  # tạo một đối tượng msg để gửi gói tin
  msg = of.ofp_packet_out()
  # tạo action flood
  action = of.ofp_action_output(port = of.OFPP_FLOOD)
  msg.actions.append(action)
  # gửi goi tin đến switch
  self.connection.send(msg)
```

- **ofp_flow_mod**: đây là gói tin sẽ gửi các trường header match xuống để switch thêm vào trong flow-table.

Các trường được định nghĩa

✓ **idle_timeout**: khoảng thời gian trước khi remove một flow-entry khi không có bất cứ gói tin nào match

✓ **hard_timeout**: khoảng thời gian trước khi remove một flow-entry

✓ **actions**: một danh sách các hành động sẽ thực hiện khi có một gói tin khớp

✓ **priority**: với giá trị cao hơn thì độ ưu tiên cao hơn

✓ **in_port**: port mà gói tin đến

III. YÊU CẦU VỀ THIẾT BỊ

Trong bài thí nghiệm này, cần một PC cài sẵn Mininet, POX Controller

IV. TRÌNH TỰ THÍ NGHIÊM

Sinh viên điền thêm những đoạn code còn thiếu vào đoạn code sau

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_dpid
from pox.lib.util import str_to_bool
from pox.lib.packet.arp import arp
from pox.lib.packet.ipv4 import ipv4
from pox.openflow.discovery import Discovery
import pox.lib.packet as pkt
```

```
import time
log = core.getLogger()
_flood_delay = 0
class L3Routing (object):
  def __init__ (self, connection, transparent):
    self.connection = connection
    self.transparent = transparent
    self.graph={}
    self.listSWitch=[]
    self.listHost=[]
    #'switch':'port' trong đó 'port' là tên port mà host dùng để kết nối
    #với switch. Sinh viên cần sửa cho giống với đồ hình mạng
    self.host={'1':3, '2':3, '3':3,'4':3}
    self.path=[]
    self.macToPort = {}
    # Lắng nghe các gói tin đến bằng cách lắng nghe kết nối giữa switch
    # và controller
    connection.addListeners(self)
    self.hold_down_expired = _flood_delay == 0
  def _handle_PacketIn (self, event):
    """
    Xử lý các gói tin đến
    """
    packet = event.parsed
    def bellman_ford(graph, source, dst):
        distance, predecessor = dict(), dict()
        for node in graph:
        distance[node], predecessor[node] = float('inf'), None
        distance[source] = 0
        for _ in range(len(graph) - 1):
        for node in graph:
```

```
                    for neighbour in graph[node]:
                        if distance[neighbour] > distance[node] + 1:
                        distance[neighbour], predecessor[neighbour] = distance[node] + 1, node
        for node in graph:
        for neighbour in graph[node]:
                    assert distance[neighbour] <= distance[node] + graph[node][neighbour], "Negative weight cycle."
        print predecessor
        check =True
        while check:
            for node in predecessor:
                if node == dst:
                    if dst == source:
                        self.path.append(dst)
                        check=False
                        break
                    else:
                        self.path.append(node)
                        dst= predecessor[node]
   def flood (message = None):
    """ Floods gói tin """
    msg = of.ofp_packet_out()
    if time.time() - self.connection.connect_time >= _flood_delay:
      if self.hold_down_expired is False:
       self.hold_down_expired = True
       log.info("%s: Flood hold-down expired -- flooding",
          dpid_to_str(event.dpid))
      if message is not None: log.debug(message)
      #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
      # OFPP_FLOOD is optional; on some switches you may need to change
```

```
      # this to OFPP_ALL.
      msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    else:
      pass
      #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
    msg.data = event.ofp
    msg.in_port = event.port
    self.connection.send(msg)
  if not self.transparent:
    if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
      drop()
      return
  if packet.dst.is_multicast:
    flood()
  else:
    for l in core.openflow_discovery.adjacency:
      sw_src = l.__str__().split("->")[0].split(".")[0].split("-")[5][1].strip()
      port_src= l.__str__().split("->")[0].split(".")[1].strip()
      sw_dst = l.__str__().split("->")[1].split(".")[0].split("-")[5][1].strip()
      port_dst=l.__str__().split("->")[1].split(".")[1].strip()
      log.debug("SW src: %s SW dst: %s Port src: %s Port dst: %s" %(sw_src,
sw_dst, port_src, port_dst))
      if sw_src in self.listSWitch:
        list = self.graph[sw_src]
        list[sw_dst]=int(port_src)
        self.graph[sw_src]=list
      else:
        tlist={}
        tlist[sw_dst]=int(port_src)
        self.graph[sw_src]= tlist
        self.listSWitch.append(sw_src)
```

```
if isinstance (packet.next, arp):
  arp_packet = packet.find(pkt.arp)
  src_ip = arp_packet.protosrc.toStr().split(".")[3]
  dst_ip = arp_packet.protodst.toStr().split(".")[3]
  dpid = dpid_to_str(event.dpid).split("-")[5][1]
if isinstance(packet.next, ipv4):
  ip_packet = packet.find(pkt.ipv4)
  if ip_packet is not None:
    src_ip = ip_packet.srcip.toStr().split(".")[3]
    dst_ip = ip_packet.dstip.toStr().split(".")[3]
log.debug("IP src= %s IP dst= %s" %(src_ip, dst_ip))
self.path=[]
bellman_ford(self.graph,dst_ip,src_ip)
print self.path
dpid = dpid_to_str(event.dpid).split("-")[5][1]
for index in range(len(self.path)):
  if dpid is self.path[index]:
    if self.path[index] is self.path[-1]:
      /*
       add your logic here
      */
      msg.idle_timeout = 10
      msg.hard_timeout = 30
      msg.actions.append(of.ofp_action_output(port = self.host[dpid]))
      msg.data = event.ofp
      self.connection.send(msg)
    else:
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet, event.port)
      msg.idle_timeout = 10
      msg.hard_timeout = 30
```

```
        /*
        add your logic here
        */
        self.connection.send(msg)
class l3_routing (object):
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent
  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    L3Routing(event.connection, self.transparent)
def launch (transparent=False, hold_down=_flood_delay):
  """
  Starts an L3 routing.
  """
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")
  core.registerNew(l3_routing, str_to_bool(transparent))
```

Sau khi điền code, sinh viên chạy và in ra kết quả pingall.

V. KẾT LUẬN

Qua bài thí nghiệm này, sinh viên đã làm quen với cách tạo một module trên POX, cách xử lý các gói tin và tạo flow-entry bằng POX để thêm vào flow-table của switch.

VI. CÂU HỎI KIỂM TRA

1. Sử dụng các thuật toán định tuyến khác thay vào code mẫu ở trên ?

……..

……..

……..

……..

2. Ở module định tuyến tại sao cần phải xử lý gói tin ARP?

……..

……..

……..

……..