Báo cáo Lab mininet

Thành viên nhóm 4: Phạm Xuân Dương-22025518

▼ LAB CUỐI KỲ – XÂY DỰNG SDN GATEWAY HOÀN CHỈNH (ROUTER + FIREWALL + MONITORING)

▼ Code topo

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel, info

class MyTopo(Topo):
    def build(self):
        # ===== Switches =====
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')

        # ===== Hosts & Links (Cố định Port cho Host là 1, 2) =====

        # Subnet A (s1): Port 1, 2 cho Host
        h1 = self.addHost('h1', ip='10.0.1.2/24', defaultRoute='via 10.0.1.
1')
        h2 = self.addHost('h2', ip='10.0.1.3/24', defaultRoute='via 10.0.1.
1')
        self.addLink(h1, s1, port2=1) # h1 → s1-eth1
        self.addLink(h2, s1, port2=2) # h2 → s1-eth2
```

```
        # Subnet B (s2): Port 1, 2 cho Host
        h3 = self.addHost('h3', ip='10.0.2.2/24', defaultRoute='via 10.0.
2.1')
        h4 = self.addHost('h4', ip='10.0.2.3/24', defaultRoute='via 10.0.
2.1')
        self.addLink(h3, s2, port2=1) # h3 → s2-eth1
        self.addLink(h4, s2, port2=2) # h4 → s2-eth2

        # Subnet C (s3): Port 1, 2 cho Host
        h5 = self.addHost('h5', ip='10.0.3.2/24', defaultRoute='via 10.0.
3.1')
        h6 = self.addHost('h6', ip='10.0.3.3/24', defaultRoute='via 10.0.
3.1')
        self.addLink(h5, s3, port2=1) # h5 → s3-eth1
        self.addLink(h6, s3, port2=2) # h6 → s3-eth2

        # ===== Inter-Switch Links (Cố định Port liên kết là 3, 4) ====
=
        info( '*** Creating links between switches with fixed ports\n' )

        # s1(3) <-----> s2(3)
        self.addLink(s1, s2, port1=3, port2=3)

        # s2(4) <-----> s3(3)
        self.addLink(s2, s3, port1=4, port2=3)

        # s3(4) <-----> s1(4)
        self.addLink(s3, s1, port1=4, port2=4)

# Register topology with Mininet
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

- Lệnh khởi tạo topo:

```
sudo mn --custom multi_router_topo.py --topo mytopo --controller=rem
ote --mac
```

▼ Code controller

```
# -*- coding: utf-8 -*-
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.arp import arp
from pox.lib.packet.ipv4 import ipv4
from pox.lib.packet.icmp import icmp
from pox.lib.addresses import IPAddr, EthAddr
import pox.lib.util as poxutil
from pox.lib.recoco import Timer
from pox.openflow.discovery import Discovery
import socket

log = core.getLogger()

# ============================================
# 1. CONFIGURATION
# ============================================

MONITOR_IP = "127.0.0.1"
MONITOR_PORT = 6666

# Protocols
IP_TCP_PROTOCOL = ipv4.TCP_PROTOCOL
IP_UDP_PROTOCOL = ipv4.UDP_PROTOCOL
IP_ICMP_PROTOCOL = ipv4.ICMP_PROTOCOL
ICMP_ECHO_REQUEST = 8
ICMP_ECHO_REPLY = 0

# Routing Table
routing_table = [
    {'network': '10.0.1.0/24', 'gateway': '10.0.1.1'},
    {'network': '10.0.2.0/24', 'gateway': '10.0.2.1'},
    {'network': '10.0.3.0/24', 'gateway': '10.0.3.1'},
]
```

```
# Router Interfaces (SỬA MAC ĐỂ TRÁNH TRÙNG VỚI HOST)
router_interfaces = {
    '10.0.1.1': EthAddr('00:00:00:00:01:01'),
    '10.0.2.1': EthAddr('00:00:00:00:02:01'),
    '10.0.3.1': EthAddr('00:00:00:00:03:01'),
}

# ACL Rules
ACL = [
  ("ingress_s1", "TCP", 22, "DENY", "Block SSH to s1"),
  ("ingress_s2", "TCP", 80, "DENY", "Block HTTP to s2"),
  ("ingress_s2", "UDP", 53, "ALLOW", "Allow DNS")
]

# Mappings
switch_name_to_dpid = { "s1": 1, "s2": 2, "s3": 3 }
dpid_to_switch_name = {v: k for k, v in switch_name_to_dpid.items
()}

gateway_ip_to_dpid = {
    '10.0.1.1': 1,
    '10.0.2.1': 2,
    '10.0.3.1': 3
}

# ================================================
# 2. GLOBAL STATE
# ================================================
arp_cache = {}
packet_queue = {}
connections = {}
ip_to_location = {}
adjacency = {}

monitor_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRA
M)

# ================================================
```

```
# 3. HELPER FUNCTIONS
# ============================================

def ip_in_network(ip, network):
    try:
        ip_addr = IPAddr(ip)
        net_addr, prefix = network.split('/')
        mask = (0xFFFFFFFF << (32 - int(prefix))) & 0xFFFFFFFF
        return (ip_addr.toUnsigned() & mask) == (IPAddr(net_addr).toU
nsigned() & mask)
    except: return False

def find_gateway_for_ip(dst_ip):
    for entry in routing_table:
        if ip_in_network(dst_ip, entry['network']): return entry['gatewa
y']
    return None

def is_switch_port(dpid, port):
    if dpid in adjacency:
        if port in adjacency[dpid].values(): return True
    return False

def learn_location(ip, dpid, port):
    # Chỉ học vị trí host, không học từ cổng nối switch
    if is_switch_port(dpid, port): return
    if ip not in ip_to_location or ip_to_location[ip] != (dpid, port):
        ip_to_location[ip] = (dpid, port)
        log.debug("Learned: %s at s%s-eth%s" % (ip, dpid, port))

def get_next_hop_port(src_dpid, dst_dpid):
    if src_dpid == dst_dpid: return None
    queue = [(src_dpid, [])]
    visited = set()
    while queue:
        (node, path) = queue.pop(0)
        if node in visited: continue
        visited.add(node)
```

```
        if node == dst_dpid:
            if not path: return None
            next_node = path[0]
            if src_dpid in adjacency and next_node in adjacency[src_dpi
d]:
                return adjacency[src_dpid][next_node]
            return None
        if node in adjacency:
            for neighbor in adjacency[node]:
                queue.append((neighbor, path + [neighbor]))
    return None

# ================================================
# 4. MONITORING (FIXED VERSION)
# ================================================

def send_stats_request():
    # Gửi request đến tất cả switch đang kết nối
    # log.info("Monitor: Sending stats request to %d switches..." % le
n(connections))
    for connection in connections.values():
        connection.send(of.ofp_stats_request(body=of.ofp_flow_stats_r
equest()))

def handle_flow_stats(event):
    tcp_bytes = 0
    udp_bytes = 0
    icmp_bytes = 0
    total_bytes = 0 # Biến mới để đếm tổng

    # Lấy DPID từ connection (SỬA LỖI QUAN TRỌNG)
    dpid = event.connection.dpid

    for f in event.stats:
        # Cộng dồn tổng lưu lượng của mọi flow
        total_bytes += f.byte_count

        # Kiểm tra Protocol (Lưu ý: Flow Routing priority 50 không có n
```

```
w_proto nên sẽ không vào if này)
        if f.match.nw_proto == 6:
            tcp_bytes += f.byte_count
        elif f.match.nw_proto == 17:
            udp_bytes += f.byte_count
        elif f.match.nw_proto == 1:
            icmp_bytes += f.byte_count

    # Format tin nhắn có thêm cột Total
    msg = "Switch s%s | Total: %-8s | TCP: %-8s | UDP: %-8s | ICM
P: %-8s" % (
        dpid, total_bytes, tcp_bytes, udp_bytes, icmp_bytes
    )

    # Debug log để kiểm tra xem hàm này có chạy không
    # log.info("Monitor: Sending to dashboard → %s" % msg)

    # Gửi UDP (Bỏ try-except để bắt lỗi nếu có)
    monitor_sock.sendto(msg.encode('utf-8'), (MONITOR_IP, MONIT
OR_PORT))

# ================================================
# 5. PACKET HANDLING (FIXED ARP LOGIC)
# ================================================

def handle_arp_request(packet, packet_in, event):
    arp_packet = packet.payload
    target_ip = str(arp_packet.protodst)
    source_ip = str(arp_packet.protosrc)

    learn_location(source_ip, event.dpid, packet_in.in_port)
    arp_cache[source_ip] = arp_packet.hwsrc

    # === CASE 1: ARP cho Gateway (Router) ===
    if target_ip in router_interfaces:
        reply_mac = router_interfaces[target_ip]
        # Gửi ARP Reply
        arp_reply = arp()
```

```
        arp_reply.hwsrc = reply_mac
        arp_reply.hwdst = arp_packet.hwsrc
        arp_reply.opcode = arp.REPLY
        arp_reply.protosrc = IPAddr(target_ip)
        arp_reply.protodst = IPAddr(source_ip)

        ether = ethernet(type=ethernet.ARP_TYPE, src=reply_mac, dst
=arp_packet.hwsrc)
        ether.payload = arp_reply

        msg = of.ofp_packet_out()
        msg.data = ether.pack()
        msg.actions.append(of.ofp_action_output(port=packet_in.in_po
rt))
        event.connection.send(msg)
        # log.info("ARP: Replied for gateway %s" % target_ip)
        return

    # === CASE 2: ARP cho Host khác (Intra-subnet) ===
    # Tìm xem target_ip thuộc subnet nào
    gw = find_gateway_for_ip(target_ip)
    if not gw: return

    target_subnet_dpid = gateway_ip_to_dpid.get(gw)

    # QUAN TRỌNG: Chặn ARP Broadcast lan sang subnet khác (Rout
er Boundary)
    if event.dpid != target_subnet_dpid:
        # log.debug("ARP: Dropped leakage request for %s on s%s" %
(target_ip, event.dpid))
        return

    # === CASE 3: Nếu đúng switch, FLOOD để host kia tự trả lời
    # (KHÔNG Proxy ARP ở đây, để tránh trùng MAC Router)
    msg = of.ofp_packet_out()
    msg.data = packet_in.data
    msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
    event.connection.send(msg)
```

```
    # log.debug("ARP: Flooded request for %s on s%s" % (target_ip,
event.dpid))

def handle_arp_reply(packet, packet_in, event):
    arp_packet = packet.payload
    src_ip = str(arp_packet.protosrc)
    dst_ip = str(arp_packet.protodst) # IP của người nhận (ví dụ h1)

    # 1. Học vị trí và Cập nhật Cache
    arp_cache[src_ip] = arp_packet.hwsrc
    learn_location(src_ip, event.dpid, packet_in.in_port)
    log.info("ARP Learned: %s → %s" % (src_ip, arp_packet.hwsrc))

    # 2. Xử lý gói tin đang chờ (Queue) - Giữ nguyên logic cũ
    if src_ip in packet_queue:
        log.info("Processing queued packets for %s" % src_ip)
        for item in packet_queue[src_ip]:
            handle_ip_packet(item['packet'], item['packet_in'], item['eve
nt'])
        del packet_queue[src_ip]

    # ============================================
==================
    # 3. FIX LỖI: Chuyển tiếp ARP Reply về cho Host yêu cầu (h1)
    # ============================================
==================

    # Nếu gói tin này gửi cho Router (Gateway), ta không cần forward
(vì ta đã xử lý ở trên rồi)
    if dst_ip in router_interfaces:
        return

    # Nếu gói tin gửi cho Host khác (h1), ta cần đẩy nó xuống Switch
    # Tìm xem h1 đang ở đâu
    if dst_ip in ip_to_location:
        (dst_dpid, dst_port) = ip_to_location[dst_ip]

        # Nếu h1 cùng nằm trên switch này (Intra-subnet thường là vậy)
```

```
        if dst_dpid == event.dpid:
            msg = of.ofp_packet_out()
            msg.data = packet_in.data
            msg.actions.append(of.ofp_action_output(port=dst_port))
            event.connection.send(msg)
            # log.debug("ARP: Forwarded reply to %s on port %s" % (dst
_ip, dst_port))
            return

    # Nếu không biết h1 ở đâu (hiếm khi xảy ra vì h1 vừa gửi Request),
    # ta FLOOD để đảm bảo h1 nhận được.
    msg = of.ofp_packet_out()
    msg.data = packet_in.data
    msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
    event.connection.send(msg)

def send_arp_request_smart(target_ip):
    # Chỉ dùng khi Router cần tìm Host (Inter-subnet)
    gw = find_gateway_for_ip(target_ip)
    if not gw: return
    target_dpid = gateway_ip_to_dpid.get(gw)
    if not target_dpid or target_dpid not in connections: return

    src_mac = router_interfaces[gw]
    r = arp()
    r.hwsrc = src_mac
    r.hwdst = EthAddr("ff:ff:ff:ff:ff:ff")
    r.opcode = arp.REQUEST
    r.protosrc = IPAddr(gw)
    r.protodst = IPAddr(target_ip)
    e = ethernet(type=ethernet.ARP_TYPE, src=src_mac, dst=r.hwds
t)
    e.payload = r

    msg = of.ofp_packet_out()
    msg.data = e.pack()
    msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
    connections[target_dpid].send(msg)
```

```
    log.info("Smart ARP: Router requesting %s on s%s" % (target_ip,
target_dpid))

def handle_ip_packet(packet, packet_in, event):
    ip_packet = packet.payload
    if not isinstance(ip_packet, ipv4): return

    src_ip = str(ip_packet.srcip)
    dst_ip = str(ip_packet.dstip)

    learn_location(src_ip, event.dpid, packet_in.in_port)

    # 1. Packet gửi tới Router (Ping Gateway)
    if dst_ip in router_interfaces:
        if ip_packet.protocol == IP_ICMP_PROTOCOL and ip_packet.pa
yload.type == ICMP_ECHO_REQUEST:
            icmp_req = ip_packet.payload
            icmp_rep = icmp(type=ICMP_ECHO_REPLY, code=0, payload
=icmp_req.payload)
            ip_rep = ipv4(protocol=IP_ICMP_PROTOCOL, srcip=ip_packe
t.dstip, dstip=ip_packet.srcip, payload=icmp_rep)
            eth_rep = ethernet(type=ethernet.IP_TYPE, src=packet.dst, d
st=packet.src, payload=ip_rep)
            msg = of.ofp_packet_out(data=eth_rep.pack())
            msg.actions.append(of.ofp_action_output(port=packet_in.in_
port))
            event.connection.send(msg)
        return

    # 2. Routing Logic
    # Nếu chưa biết MAC đích → Queue và gửi ARP
    if dst_ip not in arp_cache or dst_ip not in ip_to_location:
        if dst_ip not in packet_queue:
            packet_queue[dst_ip] = []
            send_arp_request_smart(dst_ip)
        packet_queue[dst_ip].append({'packet': packet, 'packet_in': pa
cket_in, 'event': event})
        return
```

```
    dst_mac = arp_cache[dst_ip]
    (dst_dpid, host_port) = ip_to_location[dst_ip]
    src_gw = find_gateway_for_ip(dst_ip)
    src_mac = router_interfaces[src_gw]

    out_port = None
    if event.dpid == dst_dpid:
        out_port = host_port
    else:
        out_port = get_next_hop_port(event.dpid, dst_dpid)
        if out_port is None: return # Drop để tránh loop nếu chưa có đư
ờng

    # Gửi Packet (Rewrite MAC)
    msg = of.ofp_packet_out()
    msg.data = packet_in.data
    msg.actions.append(of.ofp_action_dl_addr.set_src(src_mac))
    msg.actions.append(of.ofp_action_dl_addr.set_dst(dst_mac))
    msg.actions.append(of.ofp_action_output(port=out_port))
    event.connection.send(msg)

    # Cài Flow
    fm = of.ofp_flow_mod()
    fm.match.dl_type = ethernet.IP_TYPE
    fm.match.nw_dst = IPAddr(dst_ip)

    # Phân loại Flow theo giao thức (TCP/UDP/ICMP) để Monitor đếm
được
    fm.match.nw_proto = ip_packet.protocol

    fm.priority = 50
    fm.idle_timeout = 100
    fm.actions.append(of.ofp_action_dl_addr.set_src(src_mac))
    fm.actions.append(of.ofp_action_dl_addr.set_dst(dst_mac))
    fm.actions.append(of.ofp_action_output(port=out_port))
    event.connection.send(fm)
```

```
    # log.info("Routing: %s → %s via s%s-eth%s" % (src_ip, dst_ip, e
vent.dpid, out_port))

# ============================================
# 6. INIT & ACL
# ============================================

def install_acl(connection, dpid):
    sw_name = dpid_to_switch_name.get(dpid)
    if not sw_name: return
    acl_key = "ingress_" + sw_name
    for (rule_sw, proto, port, action, desc) in ACL:
        if rule_sw == acl_key:
            msg = of.ofp_flow_mod()
            msg.priority = 100
            msg.match.dl_type = ethernet.IP_TYPE
            msg.match.nw_proto = IP_TCP_PROTOCOL if proto == "TCP"
else IP_UDP_PROTOCOL
            msg.match.tp_dst = port
            if action == "DENY": msg.actions = []
            elif action == "ALLOW": msg.actions.append(of.ofp_action_o
utput(port=of.OFPP_CONTROLLER))
            connection.send(msg)
            log.info("Firewall: %s on %s" % (desc, sw_name))

def _handle_ConnectionUp(event):
    connections[event.dpid] = event.connection
    log.info("Switch s%s connected" % event.dpid)
    install_acl(event.connection, event.dpid)

def _handle_LinkEvent(event):
    l = event.link
    if l.dpid1 not in adjacency: adjacency[l.dpid1] = {}
    adjacency[l.dpid1][l.dpid2] = l.port1
    # log.info("Link: s%s → s%s (port %s)" % (l.dpid1, l.dpid2, l.port
1))

def _handle_PacketIn(event):
```

```
    if not event.parsed: return
    if event.parsed.type == ethernet.ARP_TYPE:
        if event.parsed.payload.opcode == arp.REQUEST: handle_arp_r
equest(event.parsed, event.ofp, event)
        else: handle_arp_reply(event.parsed, event.ofp, event)
    elif event.parsed.type == ethernet.IP_TYPE:
        handle_ip_packet(event.parsed, event.ofp, event)

def launch():
    def start_discovery(): core.openflow_discovery.addListeners(cor
e)
    core.call_when_ready(start_discovery, "openflow_discovery")
    core.openflow_discovery.addListenerByName("LinkEvent", _handl
e_LinkEvent)
    core.openflow.addListenerByName("ConnectionUp", _handle_Con
nectionUp)
    core.openflow.addListenerByName("PacketIn", _handle_PacketIn)

    # Đăng ký sự kiện thống kê
    core.openflow.addListenerByName("FlowStatsReceived", handle_
flow_stats)

    # Timer gửi request mỗi 5s
    Timer(5, send_stats_request, recurring=True)

    log.info("Triangle Router Started")
```

- Lệnh khởi tạo controller

```
./pox.py openflow.discovery controller
```

▼ Code monitor

```
# -*- coding: utf-8 -*-
# monitor.py
import socket
import sys
import time
```

```
# Cấu hình Dashboard
LISTEN_IP = "127.0.0.1"
LISTEN_PORT = 6666

def start_server():
    # Tạo UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    try:
        sock.bind((LISTEN_IP, LISTEN_PORT))
    except socket.error as e:
        print("Error: Cannot bind to port %s. Is another monitor runnin
g?" % LISTEN_PORT)
        sys.exit(1)

    print("==========================================
=======")
    print("   TRAFFIC MONITOR DASHBOARD (UDP Listener)      ")
    print("==========================================
=======")
    print("Waiting for data from Controller at %s:%s..." % (LISTEN_IP,
LISTEN_PORT))
    print("Press Ctrl+C to exit.")
    print("------------------------------------------------")

    try:
        while True:
            # Nhận dữ liệu (tối đa 4096 bytes)
            data, addr = sock.recvfrom(4096)
            message = data.decode('utf-8')

            # In ra màn hình với thời gian thực
            current_time = time.strftime("%H:%M:%S")
            print("[%s] %s" % (current_time, message))

    except KeyboardInterrupt:
        print("\nMonitor stopped.")
```

```
    finally:
        sock.close()

if __name__ == "__main__":
    start_server()
```

- Monitor:

```
# Tại thư mục chứa file monitor_dashboard.py
python monitor_dashboard.py
```

▼ Kịch bản kiểm tra:

Kịch bản kiểm tra (Test Case)

Khi hệ thống đã lên, hãy kiểm tra theo trình tự này để thấy sự hiệu quả của thuật toán tìm đường:

1. **Chờ 5-10 giây:** Để module `Discovery` tìm thấy hết các link (s1-s2, s2-s3, s1-s3). Bạn sẽ thấy log `Discovery: Link detected...` trên controller.
2. **Pingall:**
   - Gõ `mininet> pingall`.
   - Lần đầu có thể có vài dấu `X` (do ARP learning delay).
   - Gõ `pingall` lần 2: Sẽ thấy kết quả thông suốt.
3. **Kiểm tra đường đi (Tránh Loop):**
   - `mininet> h1 traceroute h5` (h1 ở s1, h5 ở s3).
   - Nếu thuật toán BFS đúng, nó sẽ đi trực tiếp **s1 → s3** (1 hop giữa các router).
   - Nếu nó đi s1 → s2 → s3 (2 hops), có thể do link cost chưa tối ưu hoặc thuật toán chọn ngẫu nhiên, nhưng miễn là đi được là tốt.
4. **Kiểm tra Firewall:**
   - Trên h1: `nc -zv 10.0.2.2` 80 → Mong đợi: Timeout (HTTP bị chặn).
   - Trên h3: `nc -zv 10.0.1.2 22` → Mong đợi: Timeout (SSH bị chặn).
   - Để kiểm tra đúng luật `UDP 53` mà bạn đã cài, bạn **bắt buộc** phải chỉ định `netcat` sử dụng giao thức UDP bằng flag `-u`.

```
h1# nc -ul 53
h3# nc -zvu 10.0.1.2 53
```

Kết luận: Bộ code này đã sẵn sàng (Ready to Run). Bạn có thể tự tin nộp bài hoặc demo.

▼ Trick debug

- Check interface:

```
mininet> links
```

- Check entry table (flow table)

```
mininet> sh ovs-ofctl dump-flows s1
```

- Vào từng ter của host qua xterm, dò trong khi pingall:

```
arp -n  #Check dia chi MAC
```

```
tcpdump -i hx-eth0 -n -e #Thay hx thanh cac host
```

- ping các host

```
mininet>pingall
mininet> h1 ping -c1 h2/s1/s2
```