1. **VXLAN คืออะไร และทำงานอย่างไรในเครือข่าย Layer 2 Overlay?**

   VXLAN (Virtual eXtensible LAN) เป็นเทคโนโลยีที่ช่วยขยายเครือข่าย Layer 2 ให้สามารถทำงานข้าม Layer 3 ได้โดยใช้ encapsulation ผ่าน UDP. VXLAN ใช้ VXLAN Network Identifier (VNI) ขนาด 24-bit เพื่อแทน VLAN ID ทำให้สามารถรองรับเครือข่ายเสมือน (Virtual Network) ได้มากถึง 16 ล้านเครือข่าย (เทียบกับ VLAN ที่มีได้แค่ 4096)

   VXLAN อาศัยอุปกรณ์ที่เรียกว่า VTEP (VXLAN Tunnel End Point) ซึ่งทำหน้าที่ encapsulate และ decapsulate VXLAN packet. ในกรณีนี้ Leaf switch ทำหน้าที่เป็น VTEP โดย encapsulate traffic ระหว่าง VLAN 10 และ VLAN 20 ให้สามารถวิ่งข้าม Spine ไปยังอีก Leaf ได้

2. **Packet ที่แสดงใน Wireshark มี VXLAN Network Identifier (VNI) เป็น 5020 หมายถึงอะไร?**

   VNI 5020 ใน Wireshark Capture ระบุว่า Packet นี้เป็น VXLAN frame ที่ encapsulate traffic ของ VLAN ที่อยู่ใน Virtual Network ID 5020. จาก Topology, VLAN 20 ใช้ subnet 192.168.20.0/24, และ Host 192.168.20.10 กับ 192.168.20.11 อยู่ใน VLAN นี้

   ดังนั้น, VXLAN frame ที่มี VNI 5020 น่าจะเป็น packet ที่ encapsulate traffic ของ VLAN 20 เพื่อให้สามารถส่งข้ามเครือข่าย Layer 3 ผ่าน VXLAN Tunnel ระหว่าง Leaf switches

3. **จาก Topology และ Wireshark Packet Capture, VXLAN ทำงานอย่างไรระหว่าง Leaf Switches?**

   a. Host H2 (192.168.20.10) ส่ง Packet ไปหา H6 (192.168.20.11)

   b. Leaf switch ที่เชื่อมต่อกับ H2 ตรวจสอบว่าเป้าหมายอยู่ที่ Leaf อื่น และต้องส่งผ่าน VXLAN Tunnel.

   c. Leaf switch encapsulate Layer 2 frame เป็น VXLAN packet โดยใส่ VNI = 5020 และส่งไปยัง VTEP ของ Leaf อีกตัวผ่าน Spine

   d. เมื่อ Leaf switch ปลายทางได้รับ VXLAN packet มันจะ decapsulate และส่งไปยัง H6 ใน VLAN 20

4. **MAC Address ที่ปรากฏใน VXLAN Frame ของ Wireshark Capture บ่งบอกถึงอะไร?**

   Source MAC: เป็น MAC ของ Host ที่ส่ง

   Destination MAC: เป็น MAC ของ Host ปลายทาง

   แต่ใน VXLAN header จะมี outer Ethernet header ซึ่งใช้ MAC ของ Leaf switches แทน

   Outer Source MAC: เป็น MAC ของ Leaf switch ที่ encapsulate packet

   Outer Destination MAC: เป็น MAC ของ Leaf switch ปลายทางที่ต้องรับ packet

5. **VXLAN ใช้ UDP Port 4789 ใน Wireshark Packet หมายถึงอะไร?**

   UDP Port 4789 เป็นค่า port ที่ IANA กำหนดให้ VXLAN ใช้เป็นค่าเริ่มต้นสำหรับ encapsulation Packet ที่ใช้ UDP Port 4789 แสดงว่าเป็น VXLAN Encapsulated Packet

   ใน Capture, Frame ที่มี UDP Port 4789 หมายความว่าเป็น VXLAN traffic ที่ถูก encapsulate และกำลังถูกส่งระหว่าง Leaf switches

6. **ในกรณีนี้ VXLAN Packet ถูกส่งจาก 10.0.0.4 ไปยัง 10.0.0.1 แสดงถึงอะไร?**

   จาก Topology:

10.0.0.1 และ 10.0.0.4 เป็น Loopback interfaces ของ Leaf switches

Loopback นี้ถูกใช้เป็น VTEP (VXLAN Tunnel End Point)

เมื่อ Wireshark แสดงว่า VXLAN packet ถูกส่งจาก 10.0.0.4 ไปยัง 10.0.0.1, หมายความว่า Leaf ที่มี 10.0.0.4 เป็น VTEP ได้ encapsulate packet และส่งไปยัง Leaf ที่มี 10.0.0.1 เป็น VTEP ปลายทาง ผ่าน Spine

7. **VXLAN Packet ใน Capture นี้สามารถระบุได้ว่ามีการใช้ Multicast หรือ Unicast สำหรับการ Forward Packet หรือไม่?**

   ใน VXLAN, การส่ง packet ไปยัง Leaf อื่นสามารถใช้ได้ 2 วิธีหลัก:

   • Multicast VXLAN – ใช้ Multicast Group เพื่อส่ง Broadcast/Unknown/Multicast (BUM) traffic ไปยัง Leaf หลายตัว

   • Unicast VXLAN (EVPN) – ใช้ BGP EVPN ในการทำ MAC Address Learning

   จาก Wireshark Capture:

   มีการใช้ Multicast Address 224.0.0.13 และ 224.0.0.5 ซึ่งเกี่ยวข้องกับ OSPF และ PIM

   ไม่มีการใช้ VXLAN Multicast Address เช่น 239.x.x.x

   ดังนั้น, ในกรณีนี้ VXLAN อาจใช้ Unicast EVPN มากกว่าการใช้ Multicast VXLAN.

8. **มีข้อสังเกตใดเกี่ยวกับ OSPF Hello Packet ใน Wireshark Capture และความสัมพันธ์กับ VXLAN?**

   Wireshark Capture แสดง OSPF Hello Packet ที่ถูกแลกเปลี่ยนระหว่าง 10.1.1.1, 10.1.1.2, และ 10.1.1.3, ซึ่งเป็นการสร้าง Neighbor Relationship ระหว่าง Spine และ Leaf ผ่าน Interface P2P

   ความสัมพันธ์ของ OSPF กับ VXLAN:

   OSPF ถูกใช้สำหรับการสร้าง Underlay Network เพื่อให้ Spine และ Leaf รู้จักเส้นทางไปยัง VTEP อื่น

   เมื่อ OSPF ทำงาน, เส้นทางของ VTEP (10.0.0.1, 10.0.0.4) ถูกแจกจ่ายให้ทุก Leaf รู้จัก

   เมื่อมี VXLAN traffic, Leaf switches จะรู้ว่า VTEP ปลายทางอยู่ที่ไหน และสามารถส่ง VXLAN Encapsulated Packet ไปยัง Leaf อื่นได้

9. Spine & Leaf คือโครงสร้างการออกแบบเครือข่ายที่ใช้ในดาต้าเซ็นเตอร์ โดยมักจะใช้เพื่อจัดการกับการขยายตัวและความต้องการที่สูงในการส่งข้อมูลในระบบที่มีจำนวนอุปกรณ์มากขึ้น เช่น ในเครือข่ายขนาดใหญ่ที่ต้องรองรับจำนวนการใช้งานและข้อมูลที่เพิ่มขึ้นอย่างรวดเร็ว

- **Spine switches** คือ switches ที่เชื่อมต่อกับทุก **Leaf switches** แต่ไม่เชื่อมต่อกันเองโดยตรง พวกมันทำหน้าที่เป็น "backbone" หรือแกนกลางในการส่งข้อมูลระหว่าง Leaf switches
- **Leaf switches** คือ switches ที่เชื่อมต่อกับอุปกรณ์ต่าง ๆ เช่น เซิร์ฟเวอร์หรือคอมพิวเตอร์ และจะส่งข้อมูลไปยัง Spine switches เพื่อให้ข้อมูลไปถึงปลายทางที่ต้องการ

ข้อดีของการออกแบบนี้คือ:

1. การส่งข้อมูลสามารถทำได้อย่างรวดเร็วและไม่มีการชนกันของข้อมูลในเครือข่าย
2. ระบบสามารถขยายได้ง่าย เพียงแค่เพิ่ม Spine หรือ Leaf switch โดยไม่ทำให้ระบบเกิดปัญหาการหนาแน่นของข้อมูล
3. โครงสร้างนี้ช่วยให้การจัดการกับการเติบโตของข้อมูลและการเชื่อมต่อของอุปกรณ์เป็นเรื่องที่ง่ายขึ้น

การทำงานในระบบนี้จะใช้ **VXLAN (Virtual Extensible LAN)** สำหรับการขยายเครือข่าย Layer 2 ข้าม Layer 3 เพื่อให้สามารถส่งข้อมูลระหว่าง Leaf switches ได้แม้ว่าเครือข่ายจะกระจายข้ามหลาย ๆ ชั้น (Spine)

# BGP Opendaylight

1. Run topo

(Optional 3 routers)

```
from mininet.net import Containernet
from mininet.node import RemoteController, Docker, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Link

def topology():

"Create a network with some docker containers acting as hosts."
net = Containernet(controller=RemoteController)

info('*** Adding switch\n')
r1 = net.addDocker('r1', ip='172.30.1.1/24', dimage="knet/urouter:1.4")
r2 = net.addDocker('r2', ip='172.30.2.1/24', dimage="knet/urouter:1.4")
r3 = net.addDocker('r3', ip='172.30.3.1/24', dimage="knet/urouter:1.4")
h1 = net.addDocker('h1', ip='172.30.1.2/24', defaultRoute='via 172.30.1.1' , dimage="knet/host-ubuntu:1-2")
h2 = net.addDocker('h2', ip='172.30.2.2/24', defaultRoute='via 172.30.2.1' ,dimage="knet/host-ubuntu:1-2")
h3 = net.addDocker('h3', ip='172.30.3.2/24', defaultRoute='via 172.30.3.1' ,dimage="knet/host-ubuntu:1-2")

s3 = net.addSwitch('s3', failMode='standalone')
s4 = net.addSwitch('s4', failMode='standalone')

info('*** Creating links\n')

net.addLink(h1, r1)
net.addLink(h2, r2)
net.addLink(h3, r3)

net.addLink(r1, s3, params1={"ip": "10.10.10.1/24"})
net.addLink(r2, s3, params1={"ip": "10.10.10.2/24"})
net.addLink(r3, s4, params1={"ip": "10.20.20.2/24"})
net.addLink(r2, s4, params1={"ip": "10.20.20.1/24"})

info('*** Starting network\n')
net.start()

#copy the bird config files
s3.cmd("sudo docker cp r1.conf mn.r1:/etc/bird.conf")
s3.cmd("sudo docker cp r2.conf mn.r2:/etc/bird.conf")
s3.cmd("sudo docker cp r3.conf mn.r3:/etc/bird.conf")
s4.cmd("sudo docker cp r1.conf mn.r1:/etc/bird.conf")
s4.cmd("sudo docker cp r2.conf mn.r2:/etc/bird.conf")
s4.cmd("sudo docker cp r3.conf mn.r3:/etc/bird.conf")
r1.cmd("bird -c /etc/bird.conf")
r2.cmd("bird -c /etc/bird.conf")
r3.cmd("bird -c /etc/bird.conf")
```

```
info('*** Running CLI\n')
CLI(net)
info('*** Stopping network')
net.stop()

if __name__ == '__main__':
setLogLevel('info')
topology()
```

# R3 conf

```
log "/var/log/bird.log" all;
debug protocols all

router id 10.20.20.2;

protocol direct {
    interface "*";
}

protocol kernel {
    learn;
    scan time 20;
    export all;
    import all;
}

protocol device {
    scan time 10;
}

# BGP Configuration
protocol bgp R2 {
    export all;
    import all;
    local as 64514;
    neighbor 10.20.20.1 as 64512;
}
```

2. Run opendaylight and install feature:install odl-bgpcep-bgp
3. Push :

```
curl --user "admin:admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X POST \
http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-instance/global-bgp/openconfig-network-instance:protocols/ -d @bgp_router.xml
################################################################################
```

```
curl -v --user "admin:admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X POST
http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-instance/global-
bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-odl-router/bgp/neighbors/ -d
@bgp_neighbor.xml
```

4. Verify :

```
curl -v --user "admin:admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X GET
http://localhost:8181/restconf/operational/bgp-rib:bgp-rib/rib/bgp-odl-router/ | xmllint --format -
```

5. Command checks on containernet :

```
r1 birdc show protocols
r1 birdc ip route
r1 birdc show status
h1 ping h2
h1 traceroute h2
```

# OVSDB-PORT

```
#create host named red
sudo ip netns add red
sudo ip link add red-veth0 type veth peer name red-veth1
sudo ip link set red-veth1 netns red
sudo ip netns exec red ip addr add 10.10.20.1/24 dev red-veth1
sudo ip netns exec red ip link set red-veth1 up

#create host named blue
sudo ip netns add blue
sudo ip link add blue-veth0 type veth peer name blue-veth1
sudo ip link set blue-veth1 netns blue
sudo ip netns exec blue ip addr add 10.10.20.2/24 dev blue-veth1
sudo ip netns exec blue ip link set blue-veth1 up

sudo ifconfig red-veth0 up
sudo ifconfig blue-veth0 up
```

```
#Adding a Switch
sudo ovs-vsctl add-br s1
sudo ovs-vsctl add-port s1 red-veth0
sudo ovs-vsctl add-port s1 blue-veth0

#Adding a Normal flow
```

```
sudo ovs-ofctl -O OpenFlow13 add-flow s1 actions=NORMAL
#Testing
sudo ip netns exec blue ping 10.10.20.1
#Destory the setup
#delete
sudo ip netns delete red
sudo ip netns delete blue
sudo ovs-vsctl del-br s1
```

## Add host 3 to switch

```
sudo ip netns add green
sudo ip link add green-veth0 type veth peer name green-veth1
sudo ip link set green-veth1 netns green
sudo ip netns exec green ip addr add 10.10.20.3/24 dev green-veth1
sudo ip netns exec green ip link set green-veth1 up

sudo ifconfig green-veth0 up
sudo ovs-vsctl add-port s1 green-veth0
```

## Sample Management commands

```
sudo ovs-vsctl add-br s1
sudo ovs-vsctl show
sudo ovs-vsctl add-port s1 veth0
sudo ovs-vsctl show
sudo ovs-vsctl add-port s1 veth1
sudo ovs-vsctl show
sudo ovs-vsctl del-port s1 veth0
sudo ovs-vsctl set-controller s1 tcp: 192.168.56.101:6633
sudo ovs-vsctl set-controller s2 tcp:192.168.56.101:6633
sudo ovs-vsctl del-br s1

Note:
sudo ip link add veth0 type veth peer name veth1
sudo ifconfig veth0 up
sudo ifconfig veth1 up
sudo ip link delete veth0
```