

# **Overlay Network and VXLAN**

组员： N/A

# Q1: Time To Live (TTL) field in ping command

## 1.1 Definition of TTL

TTL (Time To Live) is a field in the header of an IP packet that specifies the maximum number of hops that the packet can travel in the network. It is used to prevent packets from endlessly circulating in the network.

When a packet is sent, the TTL field in the IP header is initialized to a certain value (e.g., 64 for Linux systems). Each time the packet passes through a router, the TTL value is decremented by 1. When the TTL value reaches 0, the packet is discarded.

## 1.2 TTL in ping command

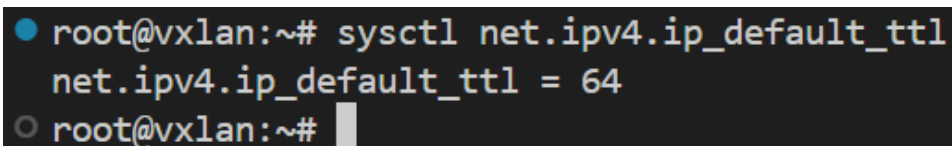
In the ping command, the TTL (Time to Live) field helps determine the number of hops between the source and destination.

When a ping command is issued, an ICMP echo request packet is sent to the destination, with the TTL field initially set to 64 by default on Linux systems. As the packet travels through the network, each router it passes will decrement the TTL value by 1. Once the packet reaches the destination, the TTL value is read, and an ICMP echo reply is sent back to the source, including the TTL value that was received.

Therefore, if the sender of the ping command initializes the TTL field to  $a$  and receives a reply with TTL  $b$ , the number of hops between the source and destination is  $a - b$ . Pinging different destinations will likely result in different TTL values, depending on the “distance” between the source and destination.

## 1.3 Examples of ping command

In the following examples, we use the ping command to send ICMP echo requests to the destination IP address. The default TTL value is 64.



```
● root@vxlan:~# sysctl net.ipv4.ip_default_ttl
net.ipv4.ip_default_ttl = 64
○ root@vxlan:~#
```

Figure 1: Default TTL value of 64 in ping command

First we send a ping request to the loopback address. Since the destination is the local machine, the TTL value is unchanged.

```
● root@vxlan:~# ping 127.0.0.1 -c 1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.029 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.029/0.029/0.029/0.000 ms
○ root@vxlan:~#
```

Figure 2: Ping request to loopback address

Next, we send a ping request to `www.baidu.com`. The TTL value 45, which means that the number of hops between the source and destination is  $64 - 45 = 19$ .

```
● root@vxlan:~# ping www.baidu.com -c 1
PING www.a.shifen.com (182.61.200.6) 56(84) bytes of data.
64 bytes from 182.61.200.6 (182.61.200.6): icmp_seq=1 ttl=45 time=27.8 ms

--- www.a.shifen.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 27.765/27.765/27.765/0.000 ms
○ root@vxlan:~#
```

Figure 3: Ping request to `www.baidu.com`

A ping request to a closer destination will have a lower TTL value. In this example to `home-sjtu.jcloud.sjtu.edu.cn`, the TTL value is 60, indicating that there are only 4 hops between the source and destination.

```
● root@vxlan:~# ping home-sjtu.jcloud.sjtu.edu.cn -c 1
PING home-sjtu.jcloud.sjtu.edu.cn (111.186.60.34) 56(84) bytes of data.
64 bytes from 111.186.60.34 (111.186.60.34): icmp_seq=1 ttl=60 time=0.808 ms

--- home-sjtu.jcloud.sjtu.edu.cn ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.808/0.808/0.808/0.000 ms
○ root@vxlan:~#
```

Figure 4: Ping request to `home-sjtu.jcloud.sjtu.edu.cn`

To verify the number of hops between the source and destination, we can use the `traceroute` command. The output shows the IP addresses of the routers along the path and the number of hops to reach the destination, which is consistent with the TTL value in the ping command.

```
root@vxlan:~# traceroute home-sjtu.jcloud.sjtu.edu.cn
traceroute to home-sjtu.jcloud.sjtu.edu.cn (111.186.60.34), 30 hops max, 60 byte packets
 1 _gateway (192.168.1.1)  0.366 ms  0.336 ms  0.319 ms
 2 10.119.0.1 (10.119.0.1)  0.752 ms  0.851 ms  0.946 ms
 3 10.3.2.169 (10.3.2.169)  1.344 ms  1.979 ms  2.552 ms
 4 10.3.2.170 (10.3.2.170)  1.452 ms  1.418 ms  1.845 ms
 5 111.186.60.34 (111.186.60.34)  0.684 ms  0.669 ms  0.680 ms
root@vxlan:~#
```

Figure 5: Traceroute to home-sjtu.jcloud.sjtu.edu.cn

## Q2: Overlay Network: WireGuard

### 2.1 Introduction to WireGuard

WireGuard is a Virtual Private Network (VPN) protocol that can be used to create point-to-point connections between devices. When two devices establish a WireGuard connection, they create a secure tunnel through which data can be transmitted, regardless of the underlying network infrastructure.

The protocol is implemented on the third layer of the OSI model (network layer). When the virtual network is established, WireGuard gives each device a unique virtual IP address, allowing them to communicate as if they were on the same local network.

When device  $A$  sends a packet to device  $B$  using the virtual IP address, WireGuard will encrypt the packet and encapsulate it in a WireGuard protocol header. The encrypted packet is then sent over the underlying physical network to device  $B$  using the real IP address of device  $B$ . Upon receiving the packet, device  $B$  decrypts it and decapsulates the WireGuard header to obtain the original packet.

### 2.2 Example of using WireGuard

In this example, we use ping commands to explain the operation of WireGuard. The network topology is shown below.

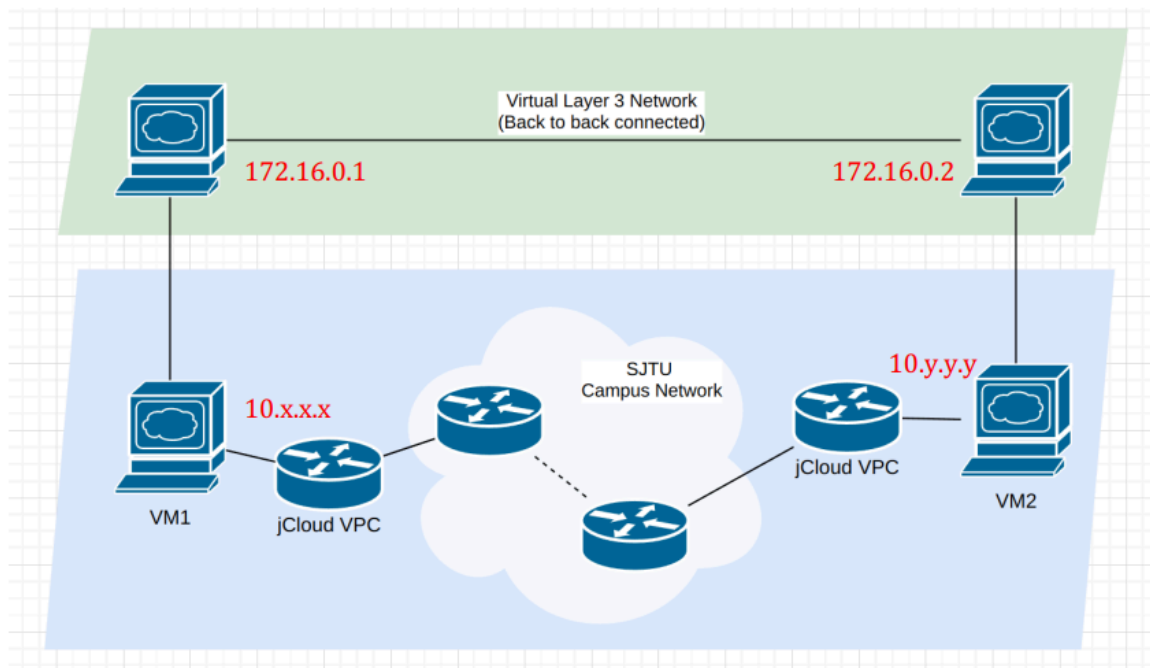


Figure 6: Network topology for WireGuard

Suppose that VM1 sends a ping request to VM2 using the virtual IP address 172.16.0.2.

- VM1 first creates an ICMP echo request packet with the destination IP address 172.16.0.2.
- WireGuard on VM1 recognizes that the destination IP address is within the virtual network and intercepts the packet.
- WireGuard on VM1 encrypts the packet and encapsulates it in a WireGuard header. It then sends the encrypted packet to VM2 using the real IP address of VM2, which is 10.y.y.y.
- Upon receiving the packet, WireGuard on VM2 decrypts it and decapsulates the WireGuard header to obtain the original ICMP echo request packet.
- VM2 processes the ICMP echo request and sends an ICMP echo reply back to VM1 using the same process.

In the process, the ICMP packets are transmitted over the overlay network created by WireGuard. The IP addresses used in the packets are virtual IP addresses.

On the contrary, the WireGuard packets are transmitted over the underlay network (the physical network). The IP addresses used in the

WireGuard packets are the real IP addresses of the devices, otherwise the packets cannot be delivered to the correct destination.

With WireGuard, communication in the overlay network is secure and private. All packets in the overlay network have no knowledge of the underlay network, and the underlay network also cannot access the data in the overlay network due to encryption.

## 2.3 Experiment with WireGuard

In the experiment we first establish a WireGuard connection between VM1 and VM2. The real IP addresses of the two VMs are respectively 10.119.14.12 and 10.119.13.193.

```
root@vxlan:~/computer-networking# wg show
interface: wg1
  public key: WuEfTho3RD34NWSRkiEpaeqdlRUf/VKLbu86w60B9HM=
  private key: (hidden)
  listening port: 51820

peer: MIqnTDFOuX1QbFTILDgTsQycyjDKPTpZQF9eiGVtEQc=
  endpoint: 10.119.13.193:51820
  allowed ips: 0.0.0.0/0
  latest handshake: 2 minutes, 17 seconds ago
  transfer: 3.12 MiB received, 118.36 MiB sent
  persistent keepalive: every 1 minute
root@vxlan:~/computer-networking#
```

Figure 7: Establishing WireGuard connection between VM1 and VM2

After the connection is established, we can ping VM2 from VM1 using the real IP address and the virtual IP address.

```
root@vxlan:~/computer-networking# ping 10.119.13.193 -c 1
PING 10.119.13.193 (10.119.13.193) 56(84) bytes of data.
64 bytes from 10.119.13.193: icmp_seq=1 ttl=62 time=2.84 ms

--- 10.119.13.193 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.844/2.844/2.844/0.000 ms
root@vxlan:~/computer-networking# ping 172.16.0.2 -c 1
PING 172.16.0.2 (172.16.0.2) 56(84) bytes of data.
64 bytes from 172.16.0.2: icmp_seq=1 ttl=64 time=2.85 ms

--- 172.16.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.850/2.850/2.850/0.000 ms
root@vxlan:~/computer-networking#
```

Figure 8: Ping VM2 from VM1 using real IP address and virtual IP address

In the first ping command, we use the real IP address 10.119.13.193 and the TTL value is 62. This indicates that there are 2 hops between VM1 and VM2 in the underlay network.

In the second ping command, we use the virtual IP address 172.16.0.2 and the TTL value is 64. This is because the ICMP packets are transmitted over the overlay network created by WireGuard, unaware of the routers in the underlay network. In the virtual network, the connection is direct between VM1 and VM2.

By using WireShark to capture the packets, we can see the difference between the ICMP packets sent using the real IP address and the virtual IP address.

When using the real IP address, the ICMP packets are transmitted directly over the underlay network. We can see a request ICMP packet from VM1 to VM2 and a reply ICMP packet from VM2 to VM1.

→	3 11.835863	192.168.1.11	10.119.13.193	ICMP	104 Echo (ping) request id=0x0008, seq=1/256, ttl=64 (reply in 4)
←	4 11.837600	10.119.13.193	192.168.1.11	ICMP	104 Echo (ping) reply id=0x0008, seq=1/256, ttl=62 (request in 3)

Figure 9: ICMP packets using real IP address

When using the virtual IP address, the ICMP packets are transmitted over the overlay network created by WireGuard. In the sender side, we can see that the temporal relation between the packets is as follows.

- VM1 sends an ICMP echo request packet to 172.16.0.2.
- WireGuard on VM1 intercepts the packet and encapsulates it in a WireGuard header. Then the WireGuard packet is sent to real IP address 10.119.13.193.
- The request is processed in VM2 and an ICMP echo reply packet is sent back to VM1.
- VM1 receives a WireGuard packet from real IP address 10.119.13.193. This is the reply from VM2.
- WireGuard on VM1 decrypts the packet and decapsulates the WireGuard header to obtain the original ICMP echo reply packet.

→	17 36.318497	172.16.0.1	172.16.0.2	ICMP	104 Echo (ping) request id=0x0009, seq=1/256, ttl=64 (reply in 20)
	18 36.318537	192.168.1.11	10.119.13.193	WireGu...	176 Transport Data, receiver=0xC44E1E90, counter=3, datalen=96
	19 36.319987	10.119.13.193	192.168.1.11	WireGu...	176 Transport Data, receiver=0x3A7EC1AA, counter=1, datalen=96
←	20 36.320021	172.16.0.2	172.16.0.1	ICMP	104 Echo (ping) reply id=0x0009, seq=1/256, ttl=64 (request in 17)

Figure 10: ICMP packets using virtual IP address (sender side)

In the receiver side, we can see that the temporal relation is reversed, as is expected.

53	85.325218	10.119.13.193	192.168.1.11	WireGu...	176 Transport Data, receiver=0x2CAD968F, counter=0, datalen=96
→	54	85.325292	172.16.0.2	172.16.0.1	ICMP 104 Echo (ping) request id=0x0008, seq=1/256, ttl=64 (reply in 55)
←	55	85.325329	172.16.0.1	172.16.0.2	ICMP 104 Echo (ping) reply id=0x0008, seq=1/256, ttl=64 (request in 54)
	56	85.325347	192.168.1.11	10.119.13.193	WireGu... 176 Transport Data, receiver=0x08900C49, counter=1, datalen=96

Figure 11: ICMP packets using virtual IP address (receiver side)

## Q3: Mininet over WireGuard

### 3.1 Setting up Mininet environment

Mininet is a network emulator that allows users to create virtual networks for testing and research purposes. In this experiment, we set up a Mininet environment with one switch and two hosts connected to the switch, on each VM.

VM1 creates a Mininet network with the subnet 192.168.3.0/24, where h1, h2 and s1 respectively holds the IP addresses 192.168.3.1, 192.168.3.2 and 192.168.3.3. The environment can be created using the following commands.

```
# exptopo.py
host1 = self.addHost('h1', mac='00:00:00:00:00:01',
ip='192.168.3.1/24')
host2 = self.addHost('h2', mac='00:00:00:00:00:02',
ip='192.168.3.2/24')
# bash of VM1
python3 exptopo.py
ip address add 192.168.3.3/24 dev s1
ip link set s1 up
sysctl -w net.ipv4.ip_forward=1
screen -c mininet.screenrc
```

We set up the Mininet environment on VM2 in a similar way, with its subnet being 192.168.4.0/24. Then we establish a route between the two Mininet networks using WireGuard.

```
# bash of VM1
ip route add 192.168.4.0/24 via 172.16.0.2
# bash of VM2
ip route add 192.168.3.0/24 via 172.16.0.1
# bash of h1 and h2 on VM1
```



```
ip route add default via 192.168.3.3
# bash of h1 and h2 on VM2
ip route add default via 192.168.4.3
```

After the setup, we can ping between all hosts in the Mininet environment. An example of pinging from h1 in VM1 to h2 in VM2 is shown below.

```
root@vxlan:~/computer-networking# ping 192.168.4.2 -c 1
PING 192.168.4.2 (192.168.4.2) 56(84) bytes of data:
64 bytes from 192.168.4.2: icmp_seq=1 ttl=62 time=4.97 ms

--- 192.168.4.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.974/4.974/4.974/0.000 ms
root@vxlan:~/computer-networking#
```

Figure 12: Ping from h1 in VM1 to h2 in VM2

### 3.2 Analysis of Mininet over WireGuard

In the previous chapter, we have established a WireGuard connection between VM1 and VM2 and successfully pinged between the two VMs using the virtual IP addresses. In this chapter, we use WireShark to capture the packets and analyze the forwarding process of the packet.

When pinging from h1 in VM1 to h2 in VM2, the ICMP packets should go through a path similar to the following.

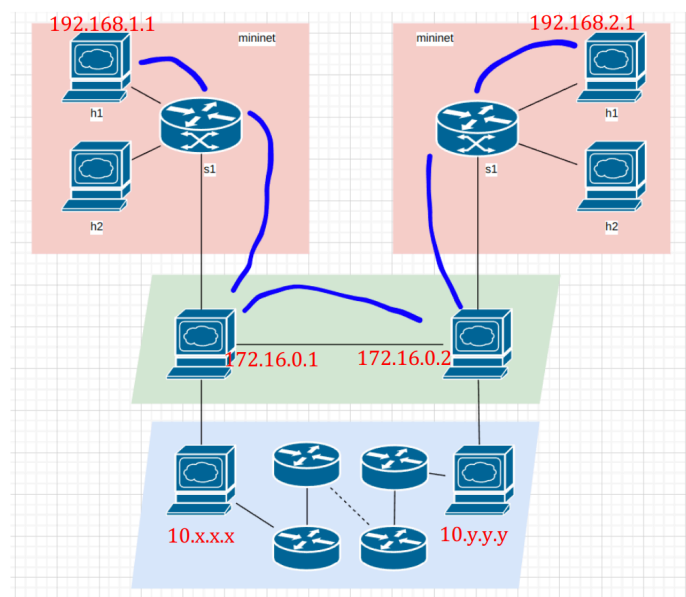


Figure 13: Path of ICMP packets

In the sender side, we can see that the ICMP packets are transmitted over the overlay network created by WireGuard. The temporal relation between the packets is as follows.

9	14.529404	192.168.3.1	192.168.4.2	ICMP	104 Echo (ping) request	id=0xa237, seq=1/256, ttl=64 (no response found!)
10	14.530062	192.168.3.1	192.168.4.2	ICMP	104 Echo (ping) request	id=0xa237, seq=1/256, ttl=64 (no response found!)
11	14.530079	192.168.3.1	192.168.4.2	ICMP	104 Echo (ping) request	id=0xa237, seq=1/256, ttl=63 (reply in 14)
12	14.530123	192.168.1.11	10.119.13.193	WireGuard	176 Transport Data, receiver=0x749F8200, counter=1, datalen=96	
13	14.534289	10.119.13.193	192.168.1.11	WireGuard	176 Transport Data, receiver=0xAD06F00A, counter=1, datalen=96	
14	14.534312	192.168.4.2	192.168.3.1	ICMP	104 Echo (ping) reply	id=0xa237, seq=1/256, ttl=63 (request in 11)
15	14.534316	192.168.4.2	192.168.3.1	ICMP	104 Echo (ping) reply	id=0xa237, seq=1/256, ttl=62
16	14.534641	192.168.4.2	192.168.3.1	ICMP	104 Echo (ping) reply	id=0xa237, seq=1/256, ttl=62

Figure 14: ICMP packets using virtual IP address (sender side)

The forwarding process of the packets is as follows.

- h1 in VM1 sends an ICMP echo request packet to 192.168.4.2. This is the ICMP packet in line 9.
- h1 does not know the route to 192.168.4.2. We previously set `ip route add default via 192.168.3.3` on h1, so h1 uses this default route to send the packet to s1.
- s1 is a switch in the Mininet environment. It forwards the packet to VM1. The ICMP packet in line 10 is the packet received by VM1.
- In VM1 we set `ip route add 192.168.4.0/24 via 172.16.0.2`, so VM1 knows that the packet should be sent to 172.16.0.2. This is the ICMP packet in line 11.
- The packet sent to 172.16.0.2 is intercepted by WireGuard and transmitted using WireGuard protocol. The WireGuard packet in line 12 is the packet sent to VM2 using the real IP address of VM2.
- After a while, VM1 receives a WireGuard packet from VM2. This is the WireGuard packet in line 13.
- WireGuard on VM1 decrypts the packet and decapsulates the WireGuard header to obtain the original ICMP echo reply packet. This is the ICMP packet in line 14.
- VM1 knows the route back to 192.168.3.1 is through s1, so it sends the ICMP echo reply packet to s1. This is the ICMP packet in line 15.
- Finally, s1 forwards the packet to h1. The ICMP packet in line 16 is the packet received by h1.

The Interface index field in the packet also reveals the forwarding process of the packet. For example, we can check the packets in line 10,

11 and 12 to see the interface index of the packets. These packets are all relevant to VM1.

```
> Frame 10: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface sshdump.exe, id 0
√ Linux cooked capture v2
  Protocol: IPv4 (0x0800)
  Interface index: 11
  Link-layer address type: Ethernet (1)
  Packet type: Unicast to us (0)
  Link-layer address length: 6
  Source: 00:00:00_00:00:01 (00:00:00:00:00:01)
  Unused: 0000
```

Figure 15: Interface index of packet in line 10

```
> Frame 11: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface sshdump.exe, id 0
√ Linux cooked capture v2
  Protocol: IPv4 (0x0800)
  Interface index: 3
  Link-layer address type: zero header length (65534)
  Packet type: Sent by us (4)
  Link-layer address length: 0
  Unused: 0000000000000000
```

Figure 16: Interface index of packet in line 11

```
> Frame 12: 176 bytes on wire (1408 bits), 176 bytes captured (1408 bits) on interface sshdump.exe, id 0
√ Linux cooked capture v2
  Protocol: IPv4 (0x0800)
  Interface index: 2
  Link-layer address type: Ethernet (1)
  Packet type: Sent by us (4)
  Link-layer address length: 6
  Source: fa:16:3e:a8:25:26 (fa:16:3e:a8:25:26)
  Unused: 0000
```

Figure 17: Interface index of packet in line 12

We also check the interface indices of VM1 using the command `ip address`. Part of the output is shown below.

```
valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc fq_codel state UP group default qlen 1000
   link/ether fa:16:3e:a8:25:26 brd ff:ff:ff:ff:ff:ff
   altname enp0s3
   inet 192.168.1.11/24 metric 100 brd 192.168.1.255 scope global dynamic ens3
      valid_lft 81424sec preferred_lft 81424sec
   inet6 fe80::f816:3eff:fea8:2526/64 scope link
      valid_lft forever preferred_lft forever
3: wg1: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1370 qdisc noqueue state UNKNOWN group default qlen 1000
   link:none
   inet 172.16.0.1/24 scope global wg1
      valid_lft forever preferred_lft forever
8: s1-eth1@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master ovs-system state UP group default qlen 1000
```

Figure 18: Interface index of VM1

```

link/ether c2:12:a8:da:31:80 brd ff:ff:ff:ff:ff:ff
11: s1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 42:29:7b:34:da:48 brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.3/24 scope global s1
        valid_lft forever preferred_lft forever
    inet6 fe80::4029:7bff:fe34:da48/64 scope link
        valid_lft forever preferred_lft forever

```

Figure 19: Interface index of VM1

In the packet in line 10, the interface index is 11, which corresponds to the interface s1 in VM1. This means that the packet is received from s1.

In the packet in line 11, the interface index is 3, which corresponds to the interface wg1 in VM1. This means that the packet is sent to VM2 using the virtual network created by WireGuard.

In the packet in line 12, the interface index is 2, which corresponds to the interface ens3 in VM1. This means that the packet is intercepted by WireGuard and transmitted using the real IP address of VM2. The packet is sent to ens3 because 192.168.1.11 is the gateway of the physical subnet VM1 is in.

On the receiver side, the packets are forwarded in a similar way, but in the reverse order.

### 3.3 Performance tests

In the Mininet environment, we can perform performance tests to measure the throughput of the network. We use the `iperf` tool to measure the bandwidth between various hosts.

The results of the performance tests are shown below.

```

root@vxlan:~/computer-networking# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[  1] local 192.168.1.11 port 5001 connected with 10.119.13.193 port 49338
[ ID] Interval      Transfer    Bandwidth
[  1] 0.0000-0.1102 sec  127 MBytes  105 Mbits/sec

```

Figure 20: Bandwidth between VM1 and VM2 using campus IP

```

root@vxlan:~/computer-networking# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 1] local 172.16.0.1 port 5001 connected with 172.16.0.2 port 44010
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.1738 sec  122 MBytes  100 Mbits/sec

```

Figure 21: Bandwidth between VM1 and VM2 using WireGuard

```

root@vxlan:~/computer-networking# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 1] local 192.168.3.1 port 5001 connected with 192.168.3.2 port 43394
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-9.9995 sec  30.7 GBytes  26.4 Gbits/sec

```

Figure 22: Bandwidth between h1 and h2 in the same VM

```

root@vxlan:~/computer-networking# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 1] local 192.168.3.1 port 5001 connected with 192.168.4.2 port 39842
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.5531 sec  126 MBytes  100 Mbits/sec

```

Figure 23: Bandwidth between h1 in VM1 and h2 in VM2

The bandwidth between VM1 and VM2 using the campus IP is 105Mbits/s, while the bandwidth using WireGuard is 100Mbits/s. The difference is due to the overhead introduced by WireGuard.

The overhead is mainly caused by the additional 80 bytes WireGuard header in each packet. With the same bytes transferred, the ratio of actual data is lower when using WireGuard. The encryption and encapsulation process typically do not account for a significant portion of the overhead, because the speed of CPUs and memory is much faster than the network speed.

Yet it can be seen that WireGuard is very efficient in terms of overhead. The throughput only drops by around 5% when using WireGuard compared to the physical network.

Inside the same Mininet network, the bandwidth between h1 and h2 is 24.6Gbits/s. This is because the packets are transmitted directly between

the two hosts without actually going through the physical network. The speed may largely depend on the performance of the CPU and memory.

When transferring data between h1 in VM1 and h2 in VM2, the bandwidth is 100Mbits/s, which is the same as the bandwidth between VM1 and VM2 using WireGuard. This is because the speed inside the Mininet network (24.6Gbits/s) is much faster than the speed of the WireGuard connection (100Mbits/s). The throughput between h1 and h2 is limited by the bottleneck of the network, in this case the WireGuard connection.

## Q4: VXLAN

### 4.1 Establishing VXLAN connection

VXLAN (Virtual Extensible LAN) is a network virtualization technology that allows users to create virtual networks over an existing Layer 3 infrastructure. In this experiment, we use VXLAN to connect two Mininet networks on different VMs.

The network topology is shown below.

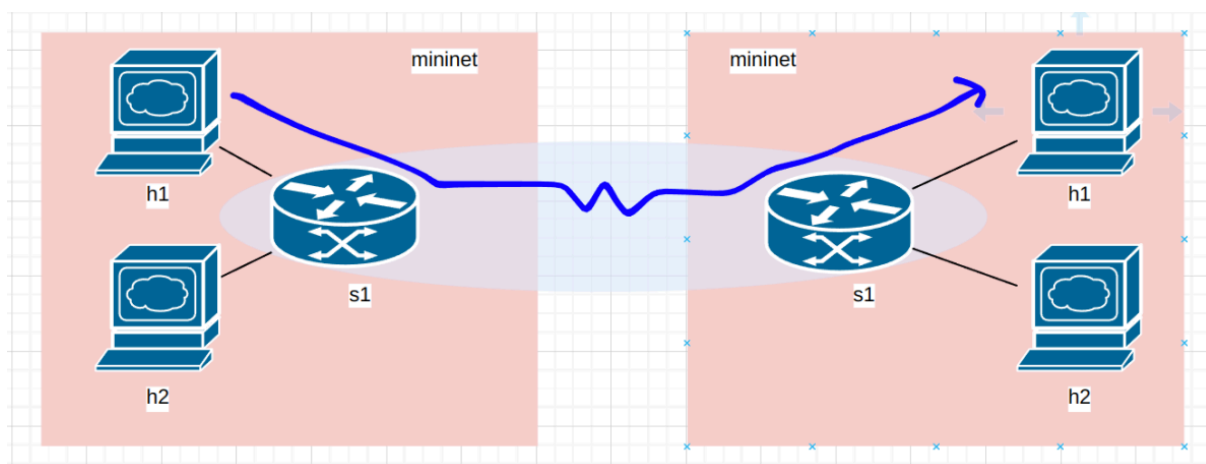


Figure 24: Network topology for VXLAN

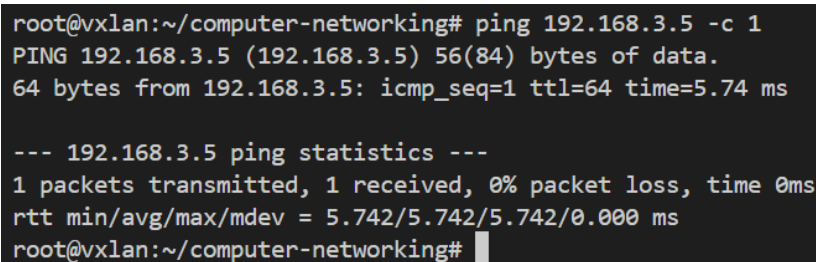
To set up the VXLAN connection, we first change the Mininet settings to make the two Mininet networks use the same subnet. We set the subnet of the Mininet network to 192.168.3.0/24. The two hosts on VM1 have IP addresses 192.168.3.1 and 192.168.3.2. The two hosts on VM2 have IP addresses 192.168.3.4 and 192.168.3.5. The MAC addresses of the hosts are set to avoid conflicts.

Then we create a VXLAN tunnel between VM1 and VM2. The commands are shown below.

```
# bash of VM1
ovs-vsctl add-port s1 vxlan0 -- set interface vxlan0
type=vxlan options:remote_ip=10.119.13.193 options:key=123
# bash of VM2
ovs-vsctl add-port s1 vxlan0 -- set interface vxlan0
type=vxlan options:remote_ip=10.119.14.12 options:key=123
```

The two VMs are connected through the VXLAN tunnel through the switch s1. Both sides need to use the same key to establish the connection.

After the setup, we can ping between all hosts in the Mininet environment. An example of pinging from h1 in VM1 to h2 in VM2 is shown below.



```
root@vxlan:~/computer-networking# ping 192.168.3.5 -c 1
PING 192.168.3.5 (192.168.3.5) 56(84) bytes of data:
64 bytes from 192.168.3.5: icmp_seq=1 ttl=64 time=5.74 ms

--- 192.168.3.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.742/5.742/5.742/0.000 ms
root@vxlan:~/computer-networking#
```

Figure 25: Ping from h1 in VM1 to h2 in VM2

In the previous chapter, we connected the two Mininet networks by manually configuring the routing tables in the two VMs. That means that the two Mininet subnets are different subnets connected by a series of routers.

VXLAN, however, connects the two Mininet networks by creating a tunnel between the two VMs. The two Mininet networks are in the same subnet, and the hosts in the two networks can communicate directly without going through a router. The VXLAN tunnel acts like a switch that connects the two networks.

## 4.2 Analysis of VXLAN

We ping from h1 in VM1 to h2 in VM2. The temporal relation between the packets on the sender side is as follows.



33	10.563620	192.168.3.1	192.168.3.5	ICMP	104 Echo (ping) request	id=0xf646, seq=1/256, ttl=64 (no response found!)
34	10.564004	192.168.3.1	192.168.3.5	ICMP	104 Echo (ping) request	id=0xf646, seq=1/256, ttl=64 (reply in 37)
35	10.564014	192.168.3.1	192.168.3.5	ICMP	154 Echo (ping) request	id=0xf646, seq=1/256, ttl=64 (reply in 36)
36	10.565129	192.168.3.5	192.168.3.1	ICMP	154 Echo (ping) reply	id=0xf646, seq=1/256, ttl=64 (request in 35)
37	10.565129	192.168.3.5	192.168.3.1	ICMP	104 Echo (ping) reply	id=0xf646, seq=1/256, ttl=64 (request in 34)
38	10.565433	192.168.3.5	192.168.3.1	ICMP	104 Echo (ping) reply	id=0xf646, seq=1/256, ttl=64

Figure 26: ICMP packets using VXLAN (sender side)

In order to analyze the forwarding process of the packets, we need to check the interface indices of the packets. The interface indices of the packets in line 33, 34 and 35 are shown below.

```
> Frame 33: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface sshdump.exe, id 0
✓ Linux cooked capture v2
  Protocol: IPv4 (0x0800)
  Interface index: 17
  Link-layer address type: Ethernet (1)
  Packet type: Unicast to another host (3)
  Link-layer address length: 6
  Source: 00:00:00_00:00:01 (00:00:00:00:00:01)
  Unused: 0000
```

Figure 27: Interface index of packet in line 33

```
> Frame 34: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface sshdump.exe, id 0
✓ Linux cooked capture v2
  Protocol: IPv4 (0x0800)
  Interface index: 21
  Link-layer address type: Ethernet (1)
  Packet type: Sent by us (4)
  Link-layer address length: 6
  Source: 00:00:00_00:00:01 (00:00:00:00:00:01)
  Unused: 0000
```

Figure 28: Interface index of packet in line 34

```
> Frame 35: 154 bytes on wire (1232 bits), 154 bytes captured (1232 bits) on interface sshdump.exe, id 0
✓ Linux cooked capture v2
  Protocol: IPv4 (0x0800)
  Interface index: 2
  Link-layer address type: Ethernet (1)
  Packet type: Sent by us (4)
  Link-layer address length: 6
  Source: fa:16:3e:a8:25:26 (fa:16:3e:a8:25:26)
  Unused: 0000
```

Figure 29: Interface index of packet in line 35

The interface index of VM1 is shown below.

```
valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc fq_codel state UP group default qlen 1000
   link/ether fa:16:3e:a8:25:26 brd ff:ff:ff:ff:ff:ff
   altname enp0s3
   inet 192.168.1.11/24 metric 100 brd 192.168.1.255 scope global dynamic ens3
      valid_lft 78795sec preferred_lft 78795sec
   inet6 fe80::f816:3eff:fea8:2526/64 scope link
      valid_lft forever preferred_lft forever
3: wg1: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1370 qdisc noqueue state UNKNOWN group default qlen 1000
```

Figure 30: Interface index of VM1



```

valid_lft forever preferred_lft forever
17: s1-eth1@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master ovs-system state UP group default qlen 1000
    link/ether 12:64:bb:b9:f9:0d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::1064:bbff:feb9:f90d/64 scope link
        valid_lft forever preferred_lft forever
18: s1-eth2@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master ovs-system state UP group default qlen 1000
    link/ether c6:51:8c:45:c4:4d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::c651:8c45:c44d:0000/64 scope link
        valid_lft forever preferred_lft forever

```

Figure 31: Interface index of VM1

```

link/ether c6:51:8c:45:c4:4d brd ff:ff:ff:ff:ff:ff link-netnsid 0
21: vxlan_sys_4789: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65000 qdisc noqueue master ovs-system state UNKNOWN group default qlen 1000
    link/ether 02:4d:60:95:d1:d1 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::e8c9:39ff:fe57:3df5/64 scope link
        valid_lft forever preferred_lft forever

```

Figure 32: Interface index of VM1

In the packet in line 33, the interface index is 17, which corresponds to the interface s1-eth1 in VM1. This means that the packet is sent from h1, through the switch s1, and then to VM1.

In the packet in line 34, the interface index is 21, which corresponds to the interface vxlan\_sys\_4789 in VM1. This means that VM1 are sending the packet to VM2 through the VXLAN tunnel.

In the packet in line 35, the interface index is 2, which corresponds to the interface ens3 in VM1. This means that the packet is sent to the physical network through the jcloud gateway.

From the interface indices, we can see that the forwarding process is similar to that of WireGuard. The devices, connections and routing tables are automatically configured by the VXLAN protocol, and the two Mininet networks are connected as if they were in the same subnet.

Another difference between VXLAN and WireGuard is that VXLAN do not use a special protocol header to encapsulate the packets. Instead, VXLAN uses a UDP header to encapsulate the packets. The packet structure in line 35 is shown below. It can be seen that the original IPv4 packet is encapsulated in a UDP packet. Therefore it has two layers of IPv4 headers.

```

> Frame 35: 154 bytes on wire (1232 bits), 154 bytes captured (1232 bits) on interface sshdump.exe, id 0
> Linux cooked capture v2
> Internet Protocol Version 4, Src: 192.168.1.11, Dst: 10.119.13.193
> User Datagram Protocol, Src Port: 40298, Dst Port: 4789
> Virtual eXtensible Local Area Network
> Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:05 (00:00:00:00:00:05)
> Internet Protocol Version 4, Src: 192.168.3.1, Dst: 192.168.3.5
> Internet Control Message Protocol

```

Figure 33: Packet structure of VXLAN

A possible reason for the difference is that WireGuard is designed for VPN connections that require encryption and security, while VXLAN only needs to establish a tunnel between two networks. Using WireGuard protocols implies that the packets are encrypted.

The packets on the receiver side are forwarded in a similar way, but in the reverse order.

### 4.3 Performance tests

We perform performance tests to measure the throughput of the network. The results are shown below.

```
root@vxlan:~/computer-networking# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 1] local 192.168.3.1 port 5001 connected with 192.168.3.2 port 54222
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-9.9991 sec 29.6 GBytes 25.4 Gbits/sec
```

Figure 34: Bandwidth between h1 and h2 in the same VM

```
root@vxlan:~/computer-networking# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 1] local 192.168.3.1 port 5001 connected with 192.168.3.5 port 45428
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.3180 sec 124 MBytes 101 Mbits/sec
```

Figure 35: Bandwidth between h1 in VM1 and h2 in VM2

The bandwidth between h1 and h2 in the same VM is 25.4Gbits/s, similar to the result in the WireGuard experiment. This is consistent with the fact that the Mininet network is unchanged.

The bandwidth between h1 in VM1 and h2 in VM2 is 101Mbits/s, slightly higher than the result in the WireGuard experiment. This is because the UDP encapsulation used by VXLAN and the WireGuard encapsulation have similar overheads. VXLAN does not need to encrypt or decrypt the packets, so the computing overhead is lower.

### 4.4 Mininet over VXLAN over WireGuard (Bonus )

In the final experiment, we establish a VXLAN connection between the two Mininet networks on different VMs. But instead of connecting the

two VMs directly, we use the WireGuard IP addresses as the remote IP addresses of the VXLAN tunnel.

The commands to set up the VXLAN connection are shown below.

```
# bash of VM1
ovs-vsctl add-port s1 vxlan0 -- set interface vxlan0
type=vxlan options:remote_ip=172.16.0.2 options:key=123
# bash of VM2
ovs-vsctl add-port s1 vxlan0 -- set interface vxlan0
type=vxlan options:remote_ip=172.16.0.1 options:key=123
```

The temporal relation between the packets on the sender side is as follows.

27	28.019704	192.168.3.1	192.168.3.5	ICMP	104 Echo (ping) request	id=0x4174, seq=1/256, ttl=64 (no response found!)
28	28.020069	192.168.3.1	192.168.3.5	ICMP	104 Echo (ping) request	id=0x4174, seq=1/256, ttl=64 (reply in 33)
29	28.020088	192.168.3.1	192.168.3.5	ICMP	154 Echo (ping) request	id=0x4174, seq=1/256, ttl=64 (reply in 32)
30	28.020121	192.168.1.11	10.119.13.193	WireGuard	224 Transport Data, receiver=0xADDCA343, counter=5, datalen=144	
31	28.022771	10.119.13.193	192.168.1.11	WireGuard	224 Transport Data, receiver=0x052B4F69, counter=7, datalen=144	
32	28.022803	192.168.3.5	192.168.3.1	ICMP	154 Echo (ping) reply	id=0x4174, seq=1/256, ttl=64 (request in 29)
33	28.022803	192.168.3.5	192.168.3.1	ICMP	104 Echo (ping) reply	id=0x4174, seq=1/256, ttl=64 (request in 28)
34	28.022934	192.168.3.5	192.168.3.1	ICMP	104 Echo (ping) reply	id=0x4174, seq=1/256, ttl=64

Figure 36: ICMP packets using VXLAN over WireGuard (sender side)

The temporal relation is nothing special. The only difference is that the packets sent between VM1 and VM2 have to be encrypted and encapsulated in WireGuard packets.

We also test the performance of the network. The throughput between h1 and h2 in the same VM is about the same as the previous experiments. The throughput between h1 in VM1 and h2 in VM2 is shown below.

```
root@vxlan:~/computer-networking# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 1] local 192.168.3.1 port 5001 connected with 192.168.3.4 port 55258
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.1638 sec  117 MBytes  96.6 Mbits/sec
```

Figure 37: Bandwidth between h1 in VM1 and h2 in VM2 using VXLAN over WireGuard

The throughput is 96.6Mbits/s, slower than respectively using VXLAN and WireGuard. This is because the packets are first encapsulated in VXLAN UDP packets, then encrypted and encapsulated in WireGuard

packets. The overhead of the two protocols is added together, which results in a lower throughput.

## **Q5: Subnets in WireGuard and VXLAN**

### **5.1 Difference between WireGuard and VXLAN**

Both WireGuard and VXLAN are overlay network technologies that create virtual networks over an existing physical network. However, in L3 routing, we set up different IP subnets, whereas in VXLAN, we need the same IP subnet. This is because the two technologies have different purposes and mechanisms.

L3 routing is used to forward packets between different IP subnets. In this case, each L3 interface is typically assigned a unique IP subnet. L3 switches, which act as routers, use a routing table to determine the next hop. Each subnet defines its own network boundaries and controls the domain. Routers (or L3 switches in this lab) rely on subnet information to distinguish between networks and determine how to route traffic. If devices are set to the same subnet, an error will occur when configuring the routing table on the VM, because the routing table cannot distinguish between the two devices.

On the other hand, VXLAN is used to extend Layer 2 domains across Layer 3 networks. To devices within the VXLAN network, it appears as if they are on the same Layer 2 subnet. However, when devices are set to different subnets, they cannot communicate within VXLAN because VXLAN operates only at Layer 2 and does not provide routing for Layer 3 traffic. Two subnets are simply two different Layer 2 domains that are not connected. Therefore, in VXLAN, devices in the same VXLAN network must be in the same IP subnet.

### **5.2 Experiment with wrong configurations**

If we try to set up a WireGuard connection between two devices in different subnets, the connection will fail. The reason is that the routing table cannot determine the next hop for the packets. The error message is shown below.

```
⊗ root@vxlan:~/computer-networking# ip route add 192.168.3.0/24 via 172.16.0.2  
RTNETLINK answers: File exists
```

Figure 38: Error message when setting up WireGuard connection using the same subnet

This error message means “trying to add a route that already exists or a gateway that is already configured”. The “already exists” here refers to the Mininet network which occupies the subnet 192.168.3.0/24. The WireGuard connection is trying to set up a route to the same subnet, which is not allowed.

Suppose that this command does successfully set up a route entry in the routing table. When ping from VM1 to VM2, there will be two possible ways to route the packets. The first way is to send the packets directly to the Mininet network, and the second way is to send the packets through the WireGuard connection, i.e. 172.16.0.2. The routing table cannot determine which way to choose, so the connection will fail. Therefore it is necessary to reject configurations that lead to ambiguous routing.

If we try to set up a VXLAN connection between two devices in different subnets, there will not be immediate errors. The VXLAN will create two different Layer 2 domains for each Mininet respectively, and the two domains will not be connected. The other side is simply unreachable. The error message is shown below.

```
root@vxlan:~/computer-networking# ping 192.168.4.1 -c 1  
ping: connect: Network is unreachable
```

Figure 39: Error message after setting up VXLAN connection between different subnets

## Q6: MAC address in WireGuard and VXLAN

### 6.1 MAC address in WireGuard

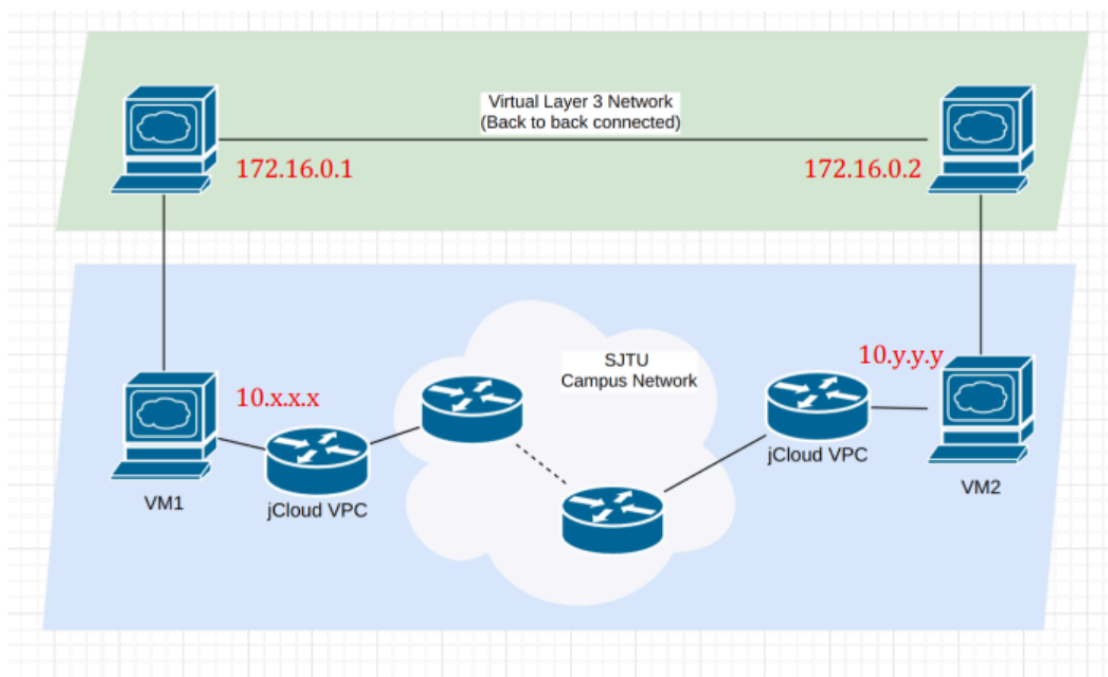


Figure 40: Network topology for WireGuard

WireGuard operates at the network layer (Layer 3) of the OSI model. As is analyzed in the previous chapter, WireGuard operates like a router that connects two subnets. The MAC addresses in two different subnets are not relevant. Therefore nothing will happen if we set the MAC addresses of the devices in different subnets to the same value.

The following example shows a typical ARP process.

18	19.147498	00:00:00_00:00:02	ARP	48	Who has 192.168.3.1? Tell 192.168.3.2
19	19.148432	00:00:00_00:00:02	ARP	48	Who has 192.168.3.1? Tell 192.168.3.2
20	19.148447	00:00:00_00:00:01	ARP	48	192.168.3.1 is at 00:00:00:00:00:01
21	19.148732	00:00:00_00:00:01	ARP	48	192.168.3.1 is at 00:00:00:00:00:01

Figure 41: ARP process in a local network

In this example, h1 has the MAC address 00:00:00:00:00:01 and the IP address 192.168.3.1, and h2 has the MAC address 00:00:00:00:00:02 and the IP address 192.168.3.2.

h2 initially does not know the MAC address of h1. When h2 wants to send a packet to h1, it sends an ARP request to the broadcast MAC address ff:ff:ff:ff:ff:ff asking “Who has the IP address 192.168.3.1? Tell 192.168.3.2” (line 18). h1 then receives the ARP request (line 19) and replies with an ARP reply (line 20) saying “192.168.3.1 is at 00:00:00:00:00:01”. The reply is sent to 192.168.3.2

and h2 gets the ARP reply (line 21). After that, h2 knows the MAC address of h1 and can send packets to h1.

ARP protocols can not travel across routers. When h1 and h2 are in different subnets, the ARP request from h2 will not reach h1. Therefore, the MAC address of h1 is not relevant to h2 and vice versa. The MAC addresses of the devices in different subnets can be set to the same value without causing any problems.

## 6.2 MAC address in VXLAN

VXLAN operates at the data link layer (Layer 2) of the OSI model. VXLAN creates a virtual network that connects two Layer 2 domains. The MAC addresses of the devices in the same VXLAN network must be unique, because the MAC address is used to identify the devices in the same Layer 2 domain.

The following experiment shows what happens when the MAC addresses of the devices in the same VXLAN network are set to the same value. In this experiment, we set the MAC address of h1 in VM1 and VM2 share the same value 00:00:00:00:00:01, and the MAC address of h2 in VM1 and VM2 share the same value 00:00:00:00:00:02.

When ping from h1 in VM1 to h1 in VM2, the following packets are captured. And the ping command fails after several attempts.

22	17.543383	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
23	17.543924	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
24	17.543929	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
25	17.543942	00:00:00_00:00:01	Broadcast	ARP	98 Who has 192.168.3.4? Tell 192.168.3.1
26	18.572301	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
27	18.572830	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
28	18.572835	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
29	18.572854	00:00:00_00:00:01	Broadcast	ARP	98 Who has 192.168.3.4? Tell 192.168.3.1
30	19.596298	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
31	19.596862	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
32	19.596867	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
33	19.596884	00:00:00_00:00:01	Broadcast	ARP	98 Who has 192.168.3.4? Tell 192.168.3.1

Figure 42: packets in VM1



23	22.782745	00:00:00_00:00:01	Broadcast	ARP	98 Who has 192.168.3.4? Tell 192.168.3.1
24	22.782745	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
25	22.783343	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
26	22.783348	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
27	22.783362	00:00:00_00:00:01		ARP	48 192.168.3.4 is at 00:00:00:00:00:01
28	23.811272	00:00:00_00:00:01	Broadcast	ARP	98 Who has 192.168.3.4? Tell 192.168.3.1
29	23.811272	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
30	23.811921	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
31	23.811927	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
32	23.811940	00:00:00_00:00:01		ARP	48 192.168.3.4 is at 00:00:00:00:00:01
33	24.835256	00:00:00_00:00:01	Broadcast	ARP	98 Who has 192.168.3.4? Tell 192.168.3.1
34	24.835256	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
35	24.835897	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
36	24.835903	00:00:00_00:00:01		ARP	48 Who has 192.168.3.4? Tell 192.168.3.1
37	24.835916	00:00:00_00:00:01		ARP	48 192.168.3.4 is at 00:00:00:00:00:01

Figure 43: packets in VM2

h1 in VM1 (192.168.3.1) wants to know the MAC address of h1 in VM2 (192.168.3.4), so it repeatedly sends ARP requests to the broadcast address asking who has it. However, no ARP reply is received. After several attempts, h1 in VM1 gives up and the ping command fails.

From figure 43, we can see that when the ARP request packet reaches h1 in VM2, it sends out its ARP reply saying that 192.168.3.4 is at 00:00:00:00:00:01. This part is correct.

However, the ARP reply's destination is the MAC address of h1 in VM1, also 00:00:00:00:00:01. Therefore, the ARP reply should be sent to h1 in VM1, but it is actually received by h1 in VM2. The reply is then discarded because it is not the expected destination. h1 in VM1, which is waiting for the ARP reply, never receives it and the ping command fails.

Therefore, the MAC addresses of the devices in the same VXLAN network must be unique. If the MAC addresses are set to the same value, the devices may not be able to communicate with each other.

## Acknowledgement

We would like to thank the teaching assistants for their guidance and help during the lab. The handout and the lab manual are detailed and very helpful. We would also like to thank the authors of the tools and software used in the lab, such as WireGuard, VXLAN, Mininet, etc. They provide us with a good platform to learn and experiment with network technologies. Finally, we would like to thank the professors for organizing the lab and providing us with the opportunity to learn and practice.