

# Overlay Network and VXLAN (With Bug)

计算机网络 CS339

李子龙 518070910095

2021 年 12 月 7 日

## 目录

1	建立网络	2
2	Wireshark 抓包	3
3	iperf 测试	4
4	ping 测试	5

An overlay network can be thought of as a computer network on top of another network.

VXLAN is often described as an overlay technology because it allows to stretch Layer 2 connections over an intervening Layer 3 network by encapsulating (tunneling) Ethernet frames in a VXLAN packet that includes IP addresses.

## 1 建立网络

先克隆虚拟机<sup>1</sup>，然后分别在 VM1 和 VM2 上分别运行 `vm1topo.py` 和 `vm2topo.py`（与 `vm1topo.py` 类似，略过），得到如图 1 所示的拓扑结构，注意拓扑已经与教程中

Listing 1: `vm1topo.py`

```
1 from mininet.cli import CLI
2 from mininet.link import TCLink
3 from mininet.node import CPULimitedHost
4 from mininet.topo import Topo
5 from mininet.net import Mininet
6 from mininet.log import lg, info
7 from mininet.util import dumpNodeConnections, run
8
9 vm1ip = "192.168.4.131"          #
10 vm2ip = "192.168.4.132"        #
11
12 class VM1Topo(Topo):
13     "Topology of VM1."
14
15     def build(self):
16         switch1 = self.addSwitch('s1', ip='10.0.0.101')
17         host1 = self.addHost('h1', cpu=.25, ip='10.0.0.1')
18         host2 = self.addHost('h2', cpu=.25, ip='10.0.0.2')
19         self.addLink(host1, switch1, use_htb=True)
20         self.addLink(host2, switch1, use_htb=True)
21
22 if __name__=="__main__":
23     topo = VM1Topo()
24     net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink, autoStaticArp=
25         True)
26     net.start()
27     dumpNodeConnections(net.hosts)
28
29     run('ifconfig s1 10.0.0.101/8 up')
30     run('ovs-vsctl del-br br1')
31     run('ovs-vsctl add-br br1')
32     run('ifconfig ens33 0 up')
33     run('ovs-vsctl add-port br1 ens33')
34     run('ifconfig br1 ' + vm1ip + ' /24 up')
35
36     # set up VxLAN
37     run('ovs-vsctl add-port s1 vxlan0 -- set interface vxlan0 type=vxlan
38         options:remote_ip=' + vm2ip + ' option:key=100 ofport_request=10')
39
40     # config MTU
41     run('ifconfig vxlan_sys_4789 mtu 1450')
42     s1 = net.switches[0]
43     h1, h2 = net.hosts
44     h1.cmdPrint('ifconfig h1-eth0 mtu 1450')
45     h2.cmdPrint('ifconfig h2-eth0 mtu 1450')
46     s1.cmdPrint('ifconfig s1-eth1 mtu 1450')
```

<sup>1</sup>感谢 VMWare Workstation 的快照技术，如果重新启动虚拟机，网络配置将会让其无法上网，必须返回原点重新配置。

```

45 s1.cmdPrint('ifconfig s1-eth2 mtu 1450')
46
47 # TBD: set up flow-table manually
48
49 CLI(net)
50 net.stop()

```

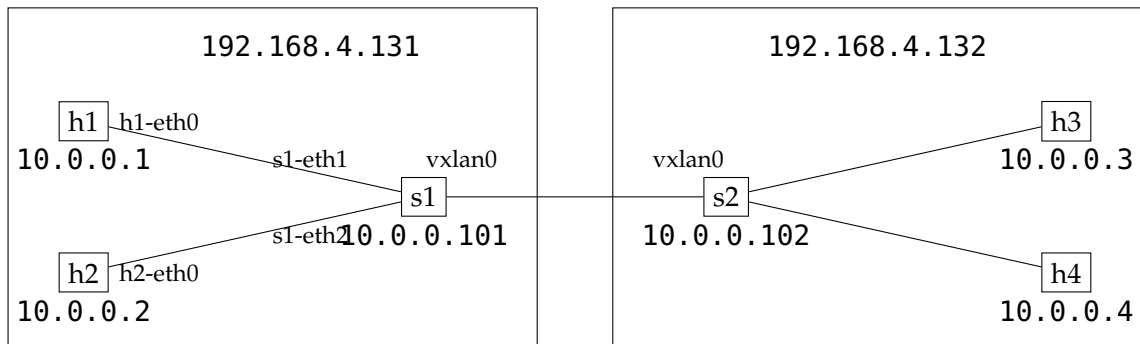


图 1: 拓扑结构

## 2 WIRESHARK 抓包

Use Wireshark to monitor the interfaces s1 and eth0, and describe your findings.

使用 Wireshark 抓取传输时数据，得到如图 2 所示的包。外层为 UDP 报文，显示的实际的 IP 地址；中间为 VXLAN 层；再内侧为 Overlay 的 ICMP 报文，显示的是 Overlay IP 地址。

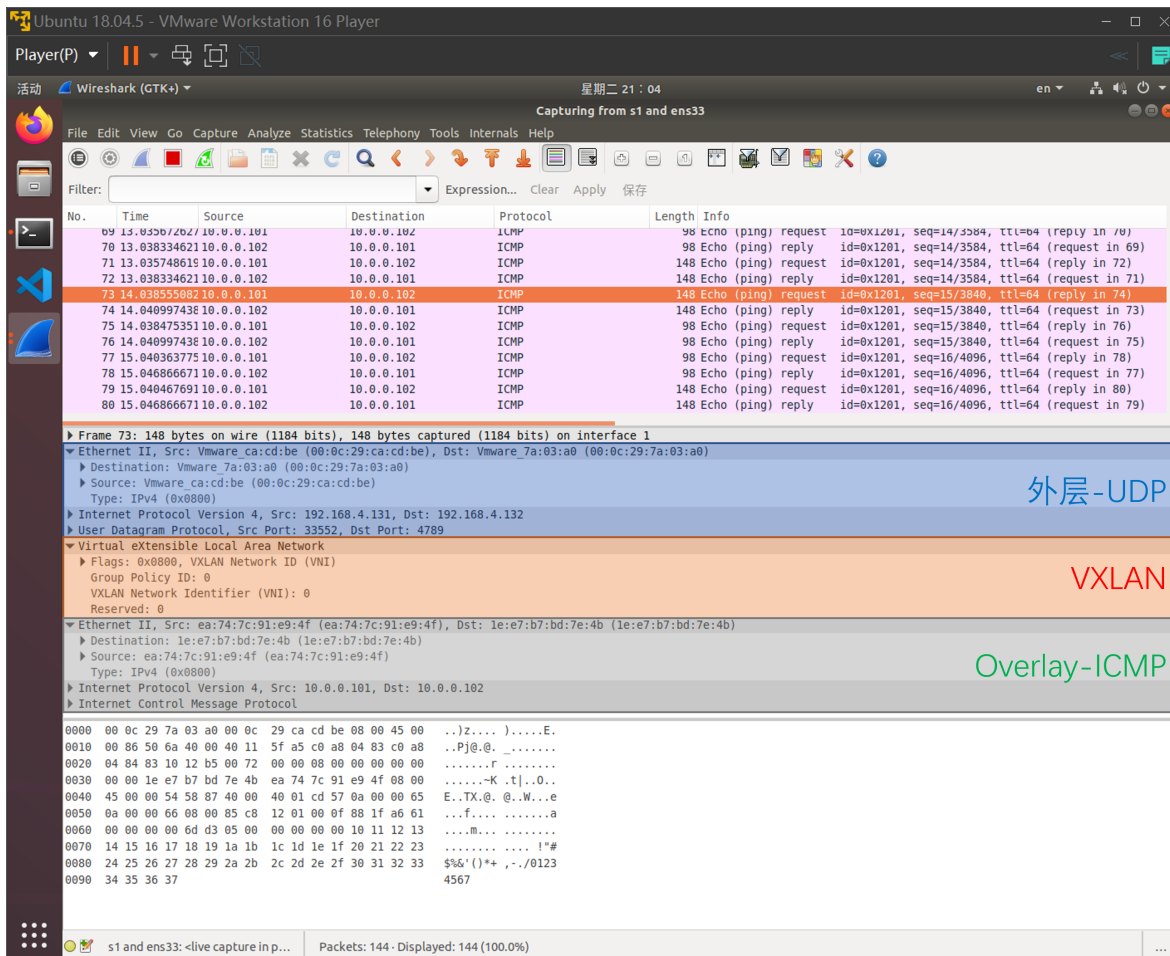


图 2: Wireshark 抓包

### 3 IPERF 测试

Use iperf to test the network bandwidth between the two virtual machines

- Test the bandwidth between 192.168.56.127 and 192.168.56.128
- Test the bandwidth between 10.0.0.1/10.0.0.2/10.0.0.101 and 10.0.0.102 (hint: you may need to specify a reasonable MTU size in order for your iperf to work in this case. Please also think about why.)

Compare the above results and explain the reason.

## 4 PING 测试

Similar to Q2, use ping to test the network latency and analyze your results.

如果不调整 MTU 会导致从 h1 ping 出去的时候只有两个包被接收了，之后会提示“没有可用的缓冲区空间”，如图 3 所示。这是因为 VXLAN 会添加 50 – 54 Bytes 的额外头部，导致超出 MTU 限制。现在的 bug 是，已经调整 MTU 后，仍然无法 ping 通。

The image shows a Wireshark packet capture interface. The main pane displays a list of captured packets. The bottom pane shows the packet details for the selected packet (Frame 35). The packet details pane shows the Ethernet II, Src: Vmware ca:cd:be (00:0c:29:ca:cd:be), Dst: Vmware 7a:03:a0 (00:0c:29:7a:03:a0) and the Address Resolution Protocol (reply) details. The packet bytes pane shows the raw data of the packet.

Packet List:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Vmware ca:cd:be	Vmware 7a:03:a0	ARP	42	Who has 192.168.4.132? Tell 192.168.4.131
2	0.000842261	Vmware 7a:03:a0	Vmware ca:cd:be	ARP	60	192.168.4.132 is at 00:0c:29:7a:03:a0
3	3.323384763	aa:4c:1d:e3:c6:7f	Broadcast	ARP	92	Who has 10.0.0.102? Tell 10.0.0.1
4	3.324738678	6e:0d:d4:ac:46:4b	aa:4c:1d:e3:c6:7f	ARP	92	10.0.0.102 is at 6e:0d:d4:ac:46:4b
5	3.323534645	aa:4c:1d:e3:c6:7f	Broadcast	ARP	42	Who has 10.0.0.102? Tell 10.0.0.1
6	3.32371218	aa:4c:1d:e3:c6:7f	Broadcast	ARP	42	Who has 10.0.0.102? Tell 10.0.0.1
7	3.324738678	6e:0d:d4:ac:46:4b	aa:4c:1d:e3:c6:7f	ARP	42	10.0.0.102 is at 6e:0d:d4:ac:46:4b
8	3.332389262	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=1/256, ttl=64 (reply in 9)
9	3.333481168	10.0.0.102	10.0.0.1	ICMP	98	Echo (ping) reply id=0xb16, seq=1/256, ttl=64 (request in 8)
10	3.322576721	aa:4c:1d:e3:c6:7f	Broadcast	ARP	42	Who has 10.0.0.102? Tell 10.0.0.1
11	3.325479289	6e:0d:d4:ac:46:4b	aa:4c:1d:e3:c6:7f	ARP	42	10.0.0.102 is at 6e:0d:d4:ac:46:4b
12	3.325488417	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=1/256, ttl=64 (reply in 13)
13	3.335797162	10.0.0.102	10.0.0.1	ICMP	98	Echo (ping) reply id=0xb16, seq=1/256, ttl=64 (request in 12)
14	3.323533342	aa:4c:1d:e3:c6:7f	Broadcast	ARP	42	Who has 10.0.0.102? Tell 10.0.0.1
15	3.332408128	10.0.0.1	10.0.0.102	ICMP	148	Echo (ping) request id=0xb16, seq=1/256, ttl=64 (reply in 16)
16	3.333481168	10.0.0.102	10.0.0.1	ICMP	148	Echo (ping) reply id=0xb16, seq=1/256, ttl=64 (request in 15)
17	4.325663401	10.0.0.1	10.0.0.102	ICMP	148	Echo (ping) request id=0xb16, seq=2/512, ttl=64 (reply in 18)
18	4.330186674	10.0.0.102	10.0.0.1	ICMP	148	Echo (ping) reply id=0xb16, seq=2/512, ttl=64 (request in 17)
19	4.325617838	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=2/512, ttl=64 (reply in 20)
20	4.330186674	10.0.0.102	10.0.0.1	ICMP	98	Echo (ping) reply id=0xb16, seq=2/512, ttl=64 (request in 19)
21	4.324886372	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=2/512, ttl=64 (reply in 22)
22	4.331543775	10.0.0.102	10.0.0.1	ICMP	98	Echo (ping) reply id=0xb16, seq=2/512, ttl=64 (request in 21)
23	5.327048503	10.0.0.1	10.0.0.102	ICMP	148	Echo (ping) request id=0xb16, seq=3/768, ttl=64 (no response found!)
24	5.327001917	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=3/768, ttl=64 (no response found!)
25	5.326960909	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=3/768, ttl=64 (no response found!)
26	6.336699662	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=4/1024, ttl=64 (no response found!)
27	6.336754676	10.0.0.1	10.0.0.102	ICMP	148	Echo (ping) request id=0xb16, seq=4/1024, ttl=64 (no response found!)
28	6.336728027	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=4/1024, ttl=64 (no response found!)
29	7.360782395	10.0.0.1	10.0.0.102	ICMP	148	Echo (ping) request id=0xb16, seq=5/1280, ttl=64 (no response found!)
30	7.360738862	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=5/1280, ttl=64 (no response found!)
31	7.360673473	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=5/1280, ttl=64 (no response found!)
32	8.384558945	10.0.0.1	10.0.0.102	ICMP	148	Echo (ping) request id=0xb16, seq=6/1536, ttl=64 (no response found!)
33	8.471136528	6e:0d:d4:ac:46:4b	aa:4c:1d:e3:c6:7f	ARP	92	Who has 10.0.0.1? Tell 10.0.0.102
34	8.471169416	Vmware 7a:03:a0	Vmware ca:cd:be	ARP	60	Who has 192.168.4.131? Tell 192.168.4.132
35	8.471493434	Vmware ca:cd:be	Vmware 7a:03:a0	ARP	42	192.168.4.131 is at 00:0c:29:ca:cd:be
36	8.486394359	aa:4c:1d:e3:c6:7f	6e:0d:d4:ac:46:4b	ARP	92	10.0.0.1 is at aa:4c:1d:e3:c6:7f
37	8.384525383	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=6/1536, ttl=64 (no response found!)
38	8.471136528	6e:0d:d4:ac:46:4b	aa:4c:1d:e3:c6:7f	ARP	42	Who has 10.0.0.1? Tell 10.0.0.102
39	8.486354471	aa:4c:1d:e3:c6:7f	6e:0d:d4:ac:46:4b	ARP	42	10.0.0.1 is at aa:4c:1d:e3:c6:7f
40	8.384490517	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0xb16, seq=6/1536, ttl=64 (no response found!)
41	8.477272885	6e:0d:d4:ac:46:4b	aa:4c:1d:e3:c6:7f	ARP	42	Who has 10.0.0.1? Tell 10.0.0.102
42	8.477311823	aa:4c:1d:e3:c6:7f	6e:0d:d4:ac:46:4b	ARP	42	10.0.0.1 is at aa:4c:1d:e3:c6:7f
43	9.409295709	10.0.0.1	10.0.0.102	ICMP	148	Echo (ping) request id=0xb16, seq=7/1792, ttl=64 (no response found!)

Packet Details (Frame 35):

- Ethernet II, Src: Vmware ca:cd:be (00:0c:29:ca:cd:be), Dst: Vmware 7a:03:a0 (00:0c:29:7a:03:a0)
- Address Resolution Protocol (reply)

Packet Bytes:

```
0000 00 0c 29 7a 03 a0 00 0c 29 ca cd be 08 06 00 01 ..12.... ).....
0010 00 00 06 04 00 02 00 0c 29 ca cd be c0 a8 04 83 ..... ).....
0020 00 0c 29 7a 03 a0 c0 a8 04 84 ..12.... ).....
```

图 3: 无缓冲区空间抓包情况

更进一步的测试表明，同一条路径上 s1-h3，如果是 s1 为主动ping方，可以ping通，而如果是 h3 为主动方，则ping不通，如图 4 所示。疑似子网内的主机无法被 VxLAN 识别以正确发包，或者有机制造缺陷，导致只能ping通两次而没有释放缓冲区。

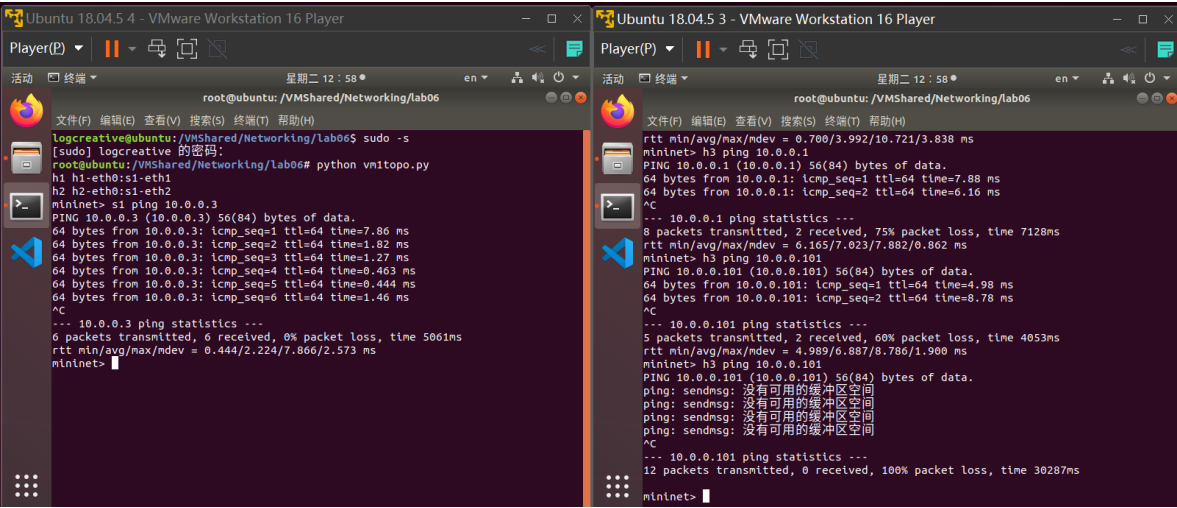


图 4: 同一路径不同发送主动方测试

下面暂时解决方法是直接将服务请求方全部放在交换机上。以及可能会手写控制器<sup>[1]</sup>。

表 1: 运行环境

操作系统	Ubuntu 18.04.5
虚拟软件	VMWare Workstation
克隆设置	链接克隆与完整克隆皆已测试

参考文献

[1] XJTU\_QYQ. 利用Mininet进行VxLAN验证实验[EB/OL]. 2018. [https://blog.csdn.net/m0\\_37313888/article/details/82748230](https://blog.csdn.net/m0_37313888/article/details/82748230).