

指令学习

1. 简单拓扑设计

实验过程

问题

2. 使用 python 建立拓扑

实验过程

问题

3. NAT 转换

实验过程

问题

4. 子网和掩码

实验过程

问题

Bug记录

Bug1

Bug2

补充资料

参考链接

指令学习

`mn --link=tc, bw=10, delay='1ms', loss=0` 设置链路属性

`--link`：链路属性可以是默认Link及TCLink。将链路类型指定为TC后，可以进一步指定具体参数。具体参数命令显示如下：

```
1 sudo mn --link tc, bw=[bandwidth], delay=[delay time], loss=[loss rate], max_QUE_size=[queue size]
```

bw表示链路带宽，使用Mbit/s为单位表示；延迟delay以字符串形式表示，如‘5ms’、‘100us’和‘1s’；loss 表示数据分组丢失率的百分比，用 0~100的一个百分数表示；max_queue_size表示最大排队长度，使用数据分组的数量表示。

iperf命令 是一个网络性能测试工具。

iperf可以测试TCP和UDP带宽质量。iperf可以测量最大TCP带宽，具有多种参数和UDP特性。iperf可以报告带宽，延迟抖动和数据包丢失。利用iperf这一特性，可以用来测试一些网络设备如路由器，防火墙，交换机等的性能。

Node sta2给sta1发送信息：iperf -s命令

```
1 iperf -s -p 5566 -i 1
```

-s表示发送数据包

-p表示端口号

-i表示间隔的时间（单位：s）

sta1监听5566端口，接受信息：iperf -c命令

```
1 perf -c 10.0.0.3 -p 5566 -t 15
```

-t表示持续接受15s的消息

10.0.0.3是sta1的IP

```
1 h1 ping -c 1000 i 0.01 h2
```

- 1 -d 使用Socket的SO_DEBUG功能。
- 2 -c <完成次数> 设置完成要求回应的次数。
- 3 -f 极限检测。
- 4 -i<间隔秒数> 指定收发信息的间隔时间。
- 5 -I<网络界面> 使用指定的网络接口送出数据包。
- 6 -l<前置载入> 设置在送出要求信息之前，先行发出的数据包。
- 7 -n 只输出数值。
- 8 -p<范本样式> 设置填满数据包的范本样式。
- 9 -q 不显示指令执行过程，开头和结尾的相关信息除外。
- 10 -r 忽略普通的Routing Table，直接将数据包送到远端主机上。
- 11 -R 记录路由过程。
- 12 -s<数据包大小> 设置数据包的大小。
- 13 -t<存活数值> 设置存活数值TTL的大小。
- 14 -v 详细显示指令的执行过程。
- 15 -w <deadline> 在 deadline 秒后退出。
- 16 -W <timeout> 在等待 timeout 秒后开始执行。

1. 简单拓扑设计

实验过程

使用iperf进行测试，一个终端机作为客户端，一个终端机作为服务器端，在服务器端每秒输出测试报告，在客户端去连接服务器端，观察输出报告。

h1 server

```
root@ubuntu-linux-20-04-desktop:/home/parallels# iperf -s -p 5566 -i 1
iperf: option requires an argument -- i
-----
Server listening on TCP port 5566
TCP window size: 85.3 KByte (default)
[ 6] local 10.0.0.1 port 5566 connected with 10.0.0.2 port 34776
[ ID] Interval Transfer Bandwidth
[ 6] 0.0- 1.0 sec 1.14 MBytes 9.60 Mbits/sec
[ 6] 1.0- 2.0 sec 1.14 MBytes 9.57 Mbits/sec
[ 6] 2.0- 3.0 sec 1.14 MBytes 9.53 Mbits/sec
[ 6] 3.0- 4.0 sec 1.14 MBytes 9.52 Mbits/sec
[ 6] 4.0- 5.0 sec 1.14 MBytes 9.57 Mbits/sec
[ 6] 5.0- 6.0 sec 1.14 MBytes 9.53 Mbits/sec
[ 6] 6.0- 7.0 sec 1.14 MBytes 9.55 Mbits/sec
[ 6] 7.0- 8.0 sec 1.14 MBytes 9.55 Mbits/sec
[ 6] 8.0- 9.0 sec 1.14 MBytes 9.52 Mbits/sec
[ 6] 9.0-10.0 sec 1.14 MBytes 9.56 Mbits/sec
[ 6] 10.0-11.0 sec 1.14 MBytes 9.57 Mbits/sec
[ 6] 11.0-12.0 sec 1.14 MBytes 9.52 Mbits/sec
[ 6] 12.0-13.0 sec 1.14 MBytes 9.58 Mbits/sec
[ 6] 13.0-14.0 sec 1.14 MBytes 9.55 Mbits/sec
[ 6] 14.0-15.0 sec 1.14 MBytes 9.55 Mbits/sec
[ 6] 15.0-16.0 sec 1.13 MBytes 9.49 Mbits/sec
[ 6] 16.0-17.0 sec 1.14 MBytes 9.55 Mbits/sec
[ 6] 17.0-18.0 sec 1.14 MBytes 9.52 Mbits/sec
[ 6] 18.0-19.0 sec 1.14 MBytes 9.52 Mbits/sec
[ 6] 19.0-20.0 sec 1.14 MBytes 9.53 Mbits/sec
[ 6] 20.0-21.0 sec 1.13 MBytes 9.48 Mbits/sec
[ 6] 0.0-21.5 sec 24.5 MBytes 9.54 Mbits/sec
```

h2 client

```
root@ubuntu-linux-20-04-desktop:/home/parallels# iperf -c 10.0.0.1 -p 5566 -t 15
-----
Client connecting to 10.0.0.1, TCP port 5566
TCP window size: 102 KByte (default)
-----
[ 5] local 10.0.0.2 port 34776 connected with 10.0.0.1 port 5566
[ ID] Interval Transfer Bandwidth
[ 5] 0.0-15.9 sec 24.5 MBytes 12.9 Mbits/sec
```

将 loss 设为 1，连接到环境后，使用 h1 ping -c 1000 i 0.01 h2，重复多次实验，观察总的丢包率。

Time 1

```
--- 10.0.0.2 ping statistics ---
1000 packets transmitted, 967 received, 3.3% packet loss, time 10135ms
rtt min/avg/max/mdev = 4.103/4.814/5.854/0.208 ms
```

Time 2

```
--- 10.0.0.2 ping statistics ---
1000 packets transmitted, 961 received, 3.9% packet loss, time 10189ms
rtt min/avg/max/mdev = 4.122/4.798/6.758/0.230 ms
```

Time 3

```
--- 10.0.0.2 ping statistics ---
1000 packets transmitted, 968 received, 3.2% packet loss, time 10183ms
rtt min/avg/max/mdev = 4.066/4.812/5.464/0.229 ms
```

Average packet loss: 3.46%

问题

观察输出报告，回答为什么 Bandwidth 的值只会接近于 10，而到达不了

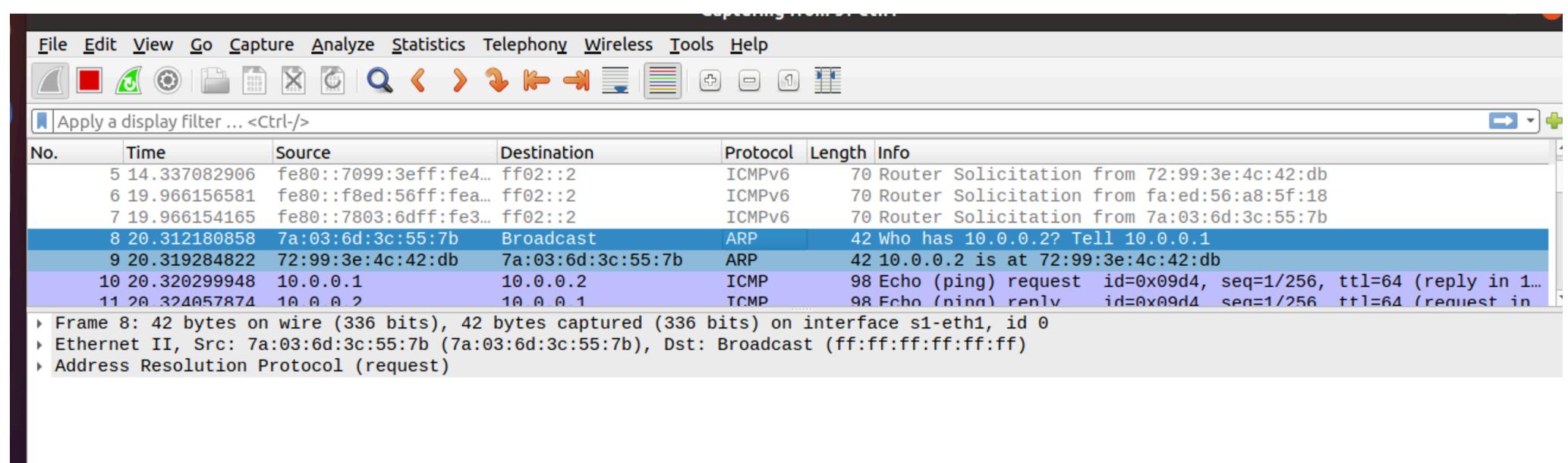
- 物理带宽限制：由于硬件限制、网络拥塞等原因，实际链路的物理带宽可能小于设定的10
- 虚拟化软件限制：在使用mininet这类虚拟化软件构建网络时，由于虚拟化的特性，在数据包的传输过程中可能出现delay, jitter等问题，从而影响数据的实际到达时间和带宽。
- 系统资源限制：在进行测试时，如果系统的CPU、内存等资源已经达到负载极限，可能会导致数据传输的速度变慢，从而影响带宽测试结果。
- 传输协议限制：不同的传输协议对带宽的利用率不同，例如TCP是一种流控协议，对带宽的利用效率可能比UDP低，所以在使用不同的传输协议时，测试结果也会有所差异。

观察丢包率，回答丢包率大概为多少，为什么不是设置的 1%

根据三次实验结果可知，平均丢包率约为3.46%。关于为什么loss并非设置值的可能原因：

- 丢包率是指在指定的时间内，发出去的数据包中未到达目的地的比例。如果在发送数据包时，数据包的发送速度过快超过了设备处理能力，就有可能会丢失部分数据包。因此，可能需要适当降低发送速度。
- 运行Mininet的操作系统对网络性能的影响。网络驱动程序的性能可能会受到限制，导致网络丢包率升高。

在第一次用 h1 ping h2 时，使用 wireshark 抓包，观察 其中的 ARP 协议，回答为什么询问的时候 destination 显示的是 broadcast



为了发送文件，服务器端需要从客户端的IP地址确认客户端的MAC地址，具体方式是通过ARP协议询问子网上所有的路由与主机的ARP表，以确定对应要解析的IP地址对应的MAC地址。正如wireshark info 里说的那样，通过广播 broadcast，h1服务器询问所有子网内的路由和主机是否有包含目的IP的ARP表，从而获得h2客户端的MAC地址。

二. 使用 python 建立拓扑

实验过程

首先使用 1.py 创建网络拓扑，然后环境中测试 ping 封包，观察现象

```
parallels@ubuntu-linux-20-04-desktop:~/Downloads/ComputerNetwork_lab2$ sudo python3 1.py
[sudo] password for parallels:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.114 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.407 ms
```

在 1.py 的基础上，我们增加一个 r1(需自己实现代码)，作为 2.py，创建网络拓扑，并再次测试 ping 封包，观察现象

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
r1 -> h1 h2
*** Results: 33% dropped (4/6 received)
```

问题

请对上述的命令进行简要介绍

ifconfig：该命令用于查看和配置网络接口。'ifconfig eth0' 可以查询第一个接口的设置信息，'ifconfig eth0 down' 则是关闭此网卡。

ip addr add：该命令用于添加 IP 地址，例如 'ip addr add 192.168.1.1/24 dev eth0' 表示为网卡 eth0 配置 IP 地址为 192.168.1.1，掩码为 /24。

在命令 "ip addr add 192.168.2.1/24 brd + dev h2-eth0" 中，**brd** 是参数之一，表示为该网络接口设置一个广播地址，"+" 表示由 Linux 内核自动生成。dev 参数则是要添加 IP 的网络接口名称。

具体来说，“brd”后面的“+”代表在内核中自动计算该网段的广播地址，并将其配置到网络接口上。使用广播地址可以确保数据包被发送到同一物理网络中的所有主机，使得通信更加高效。

ip route add：该命令用于添加路由表规则，通常用于设置默认路由，例如 'ip route add default via 10.0.1.1' 表示所有无法匹配的数据包都将通过目标地址为 10.0.1.1 的网关转发出去。

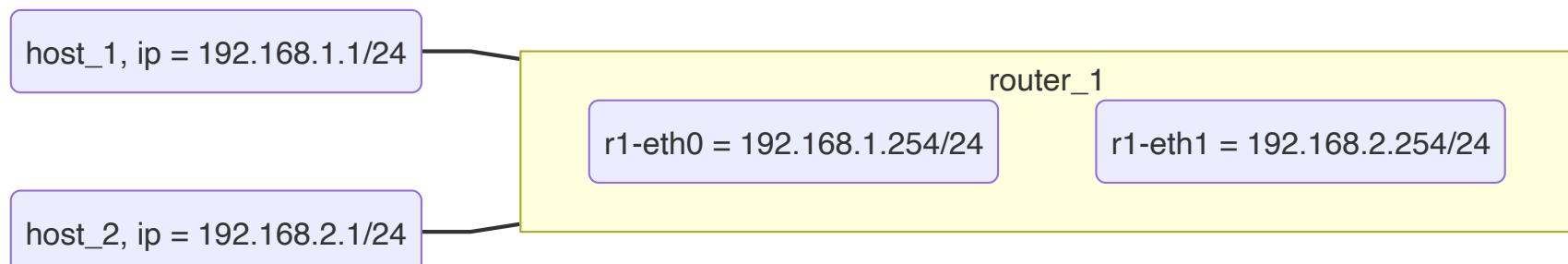
echo：该命令用于输出指定字符串或者文件重定向，例如 'echo 1 > /proc/sys/net/ipv4/ip_forward' 表示将 1 输出到 proc 文件系统下的 ipv4 目录中。

请你画出 1.py 和 2.py 构建的网络拓扑结构，包括端口 ip

py.1



py.2



其中， h_1 和 h_2 的 IP 地址分别为 192.168.1.1/24 和 192.168.2.1/24， r_1 的 eth_0 和 eth_1 分别对应 h_1 和 h_2 网段的 IP 地址。

3. NAT 转换

实验过程

利用 python 脚本，在 2.py 的基础上，编写 3.py，其拓扑结构如下图所示，并设置各元件，是的 h_1 和 h_2 能够相互 ping 通

```

parallels@ubuntu-linux-20-04-desktop:~/Downloads/ComputerNetwork_lab2$ sudo python3 3.py
mininet> h1 ping h2
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
64 bytes from 192.168.2.1: icmp_seq=1 ttl=62 time=0.085 ms
64 bytes from 192.168.2.1: icmp_seq=2 ttl=62 time=0.120 ms
64 bytes from 192.168.2.1: icmp_seq=3 ttl=62 time=0.051 ms
64 bytes from 192.168.2.1: icmp_seq=4 ttl=62 time=0.040 ms
64 bytes from 192.168.2.1: icmp_seq=5 ttl=62 time=0.045 ms
64 bytes from 192.168.2.1: icmp_seq=6 ttl=62 time=0.044 ms
64 bytes from 192.168.2.1: icmp_seq=7 ttl=62 time=0.062 ms
^C
--- 192.168.2.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6139ms
rtt min/avg/max/mdev = 0.040/0.063/0.120/0.026 ms
  
```

在 3.py 的基础上，我们编写 4.py，在路由器 1 中加入 NAT 转换

问题

使用 wireshark 对上面两个拓扑中 h_1 ping h_2 进行抓包，观察封包的不同

从以下两个拓扑的抓包情况可以看出，有以下几点不同：

- 由于外网和内网的划分，Source 和 Destination 的 IP 已经不同
- 未使用 NAT 前，需要四个使用 ARP 的数据报来传输信息，使用 NAT 后，只需要两个

-py.3

Capturing from h1-eth0

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000000	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=1/256, ttl=64 (r)
2	0.000082204	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=1/256, ttl=62 (r)
3	1.016072839	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=2/512, ttl=64 (r)
4	1.016252038	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=2/512, ttl=62 (r)
5	2.036351083	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=3/768, ttl=64 (r)
6	2.036381706	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=3/768, ttl=62 (r)
7	3.059713165	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=4/1024, ttl=64 (r)
8	3.059818993	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=4/1024, ttl=62 (r)
9	4.084236936	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=5/1280, ttl=64 (r)
10	4.084365970	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=5/1280, ttl=62 (r)
11	5.079778684	5e:ce:0b:9f:55:2b	4e:16:ba:23:6c:f0	ARP	42	Who has 192.168.1.254? Tell 192.168.1.1
12	5.079673397	4e:16:ba:23:6c:f0	5e:ce:0b:9f:55:2b	ARP	42	Who has 192.168.1.1? Tell 192.168.1.254
13	5.079877178	5e:ce:0b:9f:55:2b	4e:16:ba:23:6c:f0	ARP	42	192.168.1.1 is at 5e:ce:0b:9f:55:2b
14	5.079917093	4e:16:ba:23:6c:f0	5e:ce:0b:9f:55:2b	ARP	42	192.168.1.254 is at 4e:16:ba:23:6c:f0
15	5.108007704	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=6/1536, ttl=64 (r)
16	5.108116323	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=6/1536, ttl=62 (r)
17	6.131662768	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=7/1792, ttl=64 (r)
18	6.131689183	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=7/1792, ttl=62 (r)
19	7.155785931	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=8/2048, ttl=64 (r)
20	7.155949214	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=8/2048, ttl=62 (r)
21	8.180542807	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=9/2304, ttl=64 (r)
22	8.180575180	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=9/2304, ttl=62 (r)
23	9.204620840	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=10/2560, ttl=64 (r)
24	9.204697086	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=10/2560, ttl=62 (r)
25	10.233067809	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=11/2816, ttl=64 (r)
26	10.233377584	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=11/2816, ttl=62 (r)
27	11.251712824	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=12/3072, ttl=64 (r)
28	11.252033307	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=12/3072, ttl=62 (r)
29	12.253941768	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=13/3328, ttl=64 (r)
30	12.254026555	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=13/3328, ttl=62 (r)

Capturing from h2-eth0

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000000	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=1/256, ttl=62 (r)
2	0.000046956	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=1/256, ttl=64 (r)
3	1.016137836	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=2/512, ttl=62 (r)
4	1.016200791	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=2/512, ttl=64 (r)
5	2.036341459	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=3/768, ttl=62 (r)
6	2.036350208	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=3/768, ttl=64 (r)
7	3.059760704	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=4/1024, ttl=62 (r)
8	3.059784120	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=4/1024, ttl=64 (r)
9	4.084302058	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=5/1280, ttl=62 (r)
10	4.084330681	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=5/1280, ttl=64 (r)
11	5.079708479	4a:82:20:9f:b9:c0	de:23:65:e4:5d:e6	ARP	42	Who has 192.168.2.254? Tell 192.168.2.1
12	5.079731603	de:23:65:e4:5d:e6	4a:82:20:9f:b9:c0	ARP	42	Who has 192.168.2.1? Tell 192.168.2.254
13	5.079877887	4a:82:20:9f:b9:c0	de:23:65:e4:5d:e6	ARP	42	192.168.2.1 is at 4a:82:20:9f:b9:c0
14	5.079868221	de:23:65:e4:5d:e6	4a:82:20:9f:b9:c0	ARP	42	192.168.2.254 is at de:23:65:e4:5d:e6
15	5.108031828	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=6/1536, ttl=62 (r)
16	5.108076450	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=6/1536, ttl=64 (r)
17	6.131649061	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=7/1792, ttl=62 (r)
18	6.131658352	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=7/1792, ttl=64 (r)
19	7.155847886	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=8/2048, ttl=62 (r)
20	7.155904217	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=8/2048, ttl=64 (r)
21	8.180532849	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=9/2304, ttl=62 (r)
22	8.180543266	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=9/2304, ttl=64 (r)
23	9.204640923	192.168.1.1	192.168.2.1	ICMP	98	Echo (ping) request id=0x1cd5, seq=10/2560, ttl=62 (r)
24	9.204663213	192.168.2.1	192.168.1.1	ICMP	98	Echo (ping) reply id=0x1cd5, seq=10/2560, ttl=64 (r)

-py.4

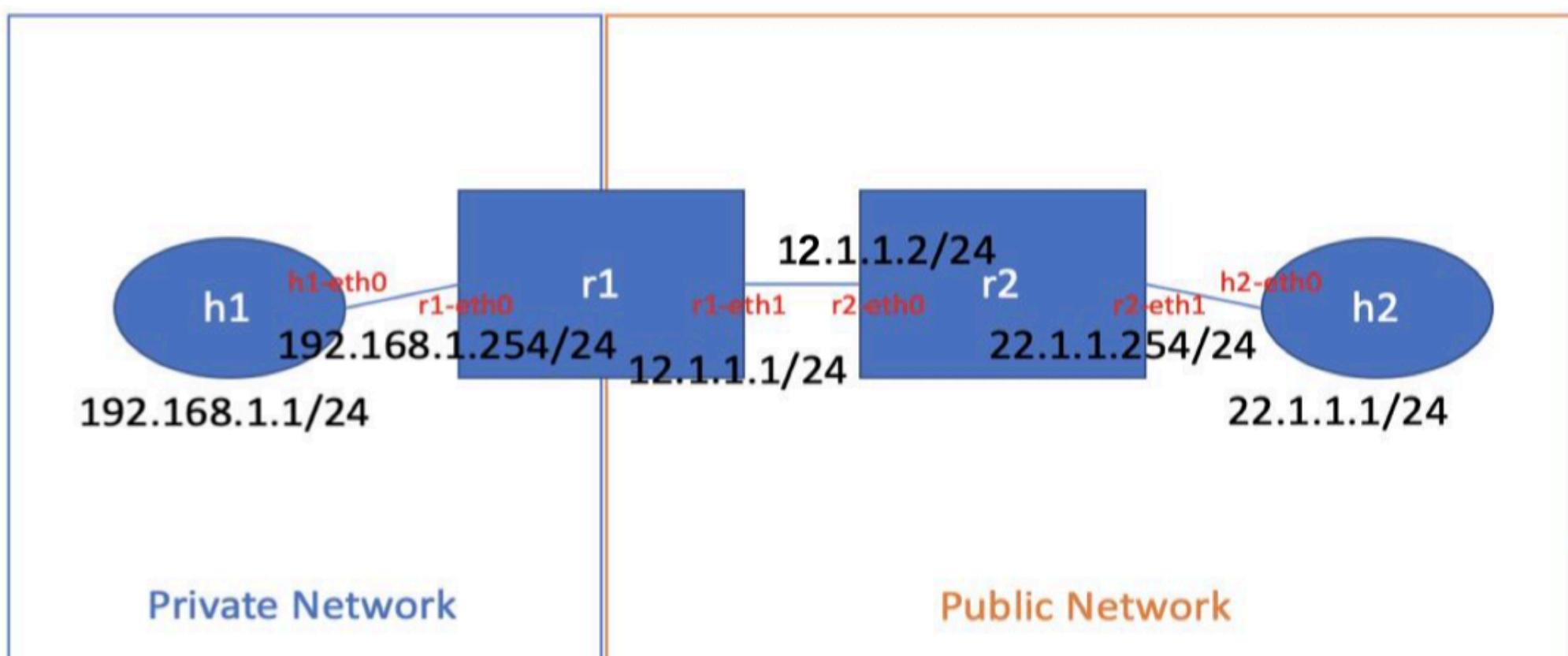
1 h1 ping h2

Capturing from h1-eth0

74	34.784707381	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=35/8960, ttl=62
75	35.810118480	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=36/9216, ttl=64
76	35.810382653	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=36/9216, ttl=62
77	36.832779086	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=37/9472, ttl=64
78	36.832949522	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=37/9472, ttl=62
79	37.856628053	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=38/9728, ttl=64
80	37.856855565	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=38/9728, ttl=62
81	38.881152988	8a:18:9b:e0:47:03	66:55:de:75:48:89	ARP	42 Who has 192.168.1.1? Tell 192.168.1.254	
82	38.881195774	66:55:de:75:48:89	8a:18:9b:e0:47:03	ARP	42 192.168.1.1 is at 66:55:de:75:48:89	
83	38.881237394	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=39/9984, ttl=64
84	38.881284263	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=39/9984, ttl=62
85	39.909372536	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=40/10240, ttl=64
86	39.909433237	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=40/10240, ttl=62
87	40.928845212	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=41/10496, ttl=64
88	40.929053227	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=41/10496, ttl=62
89	41.929932196	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=42/10752, ttl=64
90	41.930113131	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=42/10752, ttl=62
91	42.944513366	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=43/11008, ttl=64
92	42.944595897	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=43/11008, ttl=62
93	43.969223773	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=44/11264, ttl=64
94	43.969298347	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=44/11264, ttl=62
95	44.997539459	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=45/11520, ttl=64
96	44.997610159	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=45/11520, ttl=62
97	46.020049128	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=46/11776, ttl=64
98	46.020160531	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=46/11776, ttl=62
99	47.041409245	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=47/12032, ttl=64
100	47.041492318	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=47/12032, ttl=62
101	48.064989881	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=48/12288, ttl=64
102	48.065229477	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=48/12288, ttl=62
103	49.089843899	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=49/12544, ttl=64
104	49.089907432	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=49/12544, ttl=62
105	50.113364092	192.168.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=50/12800, ttl=64
106	50.113364092	22.1.1.1	192.168.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=50/12800, ttl=62

Capturing from h2-eth0

215	100.052779515	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=99/25344, ttl=64
216	101.061909223	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=100/25600, ttl=6
217	101.061994673	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=100/25600, ttl=6
218	102.081233760	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=101/25856, ttl=6
219	102.081287505	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=101/25856, ttl=6
220	103.108584216	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=102/26112, ttl=6
221	103.108653127	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=102/26112, ttl=6
222	104.128710788	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=103/26368, ttl=6
223	104.128735036	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=103/26368, ttl=6
224	105.152972646	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=104/26624, ttl=6
225	105.153044931	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=104/26624, ttl=6
226	106.177991984	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=105/26880, ttl=6
227	106.178058103	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=105/26880, ttl=6
228	107.206242317	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=106/27136, ttl=6
229	107.206305228	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=106/27136, ttl=6
230	108.225176606	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=107/27392, ttl=6
231	108.225205770	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=107/27392, ttl=6
232	109.249037598	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=108/27648, ttl=6
233	109.250307855	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=108/27648, ttl=6
234	109.280582136	ea:a1:75:d7:ac:96	02:d7:a8:ea:b9:99	ARP	42 Who has 22.1.1.1? Tell 22.1.1.254	
235	109.280617882	02:d7:a8:ea:b9:99	ea:a1:75:d7:ac:96	ARP	42 22.1.1.1 is at 02:d7:a8:ea:b9:99	
236	110.251222831	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=109/27904, ttl=6
237	110.251286617	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=109/27904, ttl=6
238	111.264768080	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=110/28160, ttl=6
239	111.264776329	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=110/28160, ttl=6
240	112.292732739	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=111/28416, ttl=6
241	112.292760903	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=111/28416, ttl=6
242	113.314809507	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=112/28672, ttl=6
243	113.314878084	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=112/28672, ttl=6
244	114.340165702	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=113/28928, ttl=6
245	114.340238029	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=113/28928, ttl=6
246	115.360712437	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=114/29184, ttl=6
247	115.360731811	22.1.1.1	12.1.1.1	ICMP	98 Echo (ping) reply	id=0x1c25, seq=114/29184, ttl=6
248	116.387522711	12.1.1.1	22.1.1.1	ICMP	98 Echo (ping) request	id=0x1c25, seq=115/29440, ttl=6
249	116.387592247	22				



内网中的主机h1想要向IP为22.1.1.1/24的主机h2发送文件，首先通过ARP获取h2主机IP对应的MAC地址。然后h1主机指派了某个端口，然后先将数据报发送到LAN中。NAT路由器，也即r1，会为数据报分配新的端口，然后将源IP替代为一个外网的IP地址。然后将数据报转发到router2，最后发送到h2主机上。

四. 子网和掩码

实验过程

拓扑结构1

```
mininet> h1 ping h2
PING 10.0.0.130 (10.0.0.130) 56(84) bytes of data.
64 bytes from 10.0.0.130: icmp_seq=1 ttl=64 time=0.214 ms
64 bytes from 10.0.0.130: icmp_seq=2 ttl=64 time=0.245 ms
64 bytes from 10.0.0.130: icmp_seq=3 ttl=64 time=0.228 ms
64 bytes from 10.0.0.130: icmp_seq=4 ttl=64 time=0.090 ms
64 bytes from 10.0.0.130: icmp_seq=5 ttl=64 time=0.175 ms
64 bytes from 10.0.0.130: icmp_seq=6 ttl=64 time=0.211 ms
64 bytes from 10.0.0.130: icmp_seq=7 ttl=64 time=0.204 ms
64 bytes from 10.0.0.130: icmp_seq=8 ttl=64 time=0.224 ms
64 bytes from 10.0.0.130: icmp_seq=9 ttl=64 time=0.127 ms
64 bytes from 10.0.0.130: icmp_seq=10 ttl=64 time=0.211 ms
64 bytes from 10.0.0.130: icmp_seq=11 ttl=64 time=0.246 ms
64 bytes from 10.0.0.130: icmp_seq=12 ttl=64 time=0.195 ms
64 bytes from 10.0.0.130: icmp_seq=13 ttl=64 time=0.076 ms
64 bytes from 10.0.0.130: icmp_seq=14 ttl=64 time=0.085 ms
64 bytes from 10.0.0.130: icmp_seq=15 ttl=64 time=0.160 ms
```

拓扑结构2

```
mininet> h1 ping h2
ping: connect: Network is unreachable
```

问题

上面两个拓扑结构中 h1 与 h2 的连通性如何，并解释原因

从实验过程的图中可以看出，拓扑结构1中h1和h2可以连通，拓扑结构2中h1和h2无法连通。

原因是拓扑结构1中的两个主机在同一个子网中。同一个子网中的两个主机可以使用 Ping 命令互相通信，这是因为它们具有相同的网络前缀，并且使用相同的默认网关进行通信。在一个局域网（LAN）中，所有的主机都连接到同一个交换机或路由器上，被视为同一个网络。当两台主机想要互相通信时，它们可以通过交换机或路由器来查找对方的 MAC 地址和 IP 地址，然后将数据包发送给对方。因此，只要两台主机的 IP 地址位于同一个子网，并且共享同一个默认网关和交换机，它们就可以互相连通并交换信息。

而拓扑结构2中的两个主机不在一个子网内，从而无法连通。

加入一个路由器，使得不能相互通信的 h1 和 h2 可以相互通信

具体见代码 [subnet3.py](#)

Bug记录

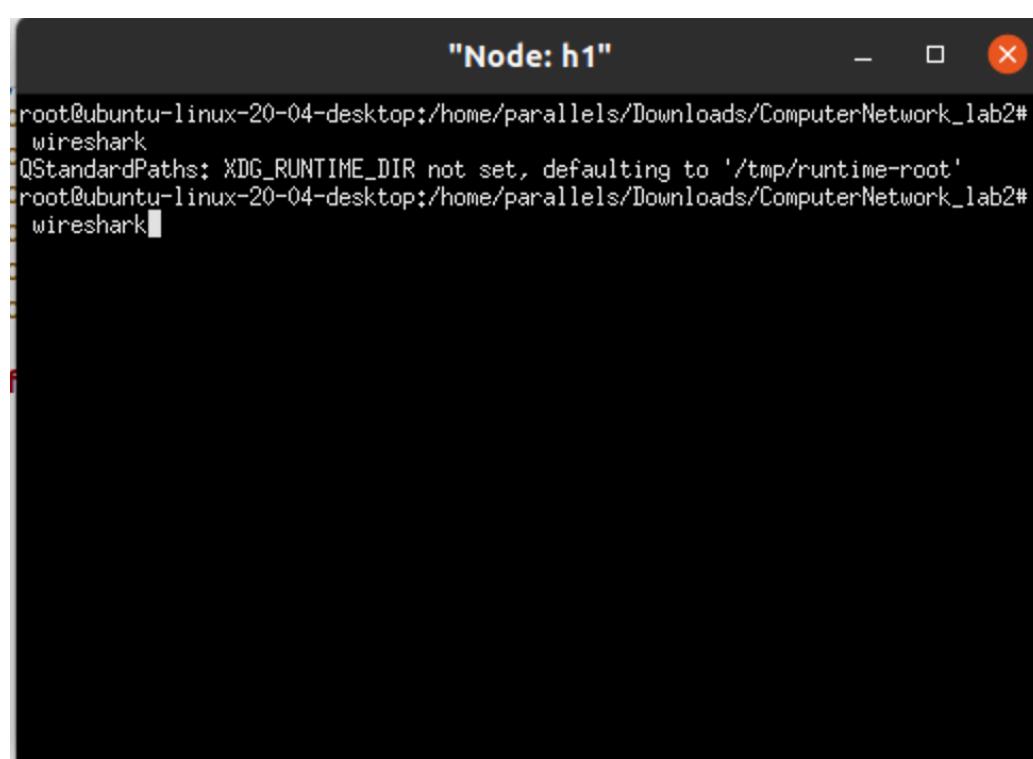
Bug1

```
1 router.cmd('echo 1 > /proc/sys/net/ipv4/ip_forward')
```

设置router的时候，不能没有这一行代码，否则无法ping通

Bug2

使用wireshark抓包，但是没有发现相应的网卡接口。这是因为wireshark要在设备的xterm里打开！



补充资料

在网络中，数据包在传输过程中可能会遇到各种各样的问题，例如链路带宽限制、延迟、丢包等。其中，“延迟”和“时延抖动（jitter）”是常见的影响因素。

mininet是一款轻量级的网络模拟器，使用它可以方便地搭建虚拟网络环境并进行相关测试。在mininet模拟过程中，由于网络拓扑结构和数据传输方式等因素的复杂性，会导致一些网络性能上的问题，如“延迟”和“时延抖动”。

1. 延迟：指数据包从发送方到接收方所需的时间。在mininet模拟的网络环境中，如果模拟拓扑结构、网络设备等配置不当，或者系统资源紧张等原因，都可能导致数据包的延迟增加，并影响网络应用的实时性和用户体验。
2. 时延抖动：指相邻两个数据包之间的延迟差异。在网络传输过程中，数据包的到达时间受多种因素的影响，如网络拥塞、路由选择等，从而导致数据包到达的时间存在波动。这种波动即称为“时延抖动”。时延抖动的大小和变化趋势会直接影响到多媒体应用（如VoIP、视频会议等）的质量和用户的体验。

参考链接

1. <https://www.cnblogs.com/lht333/p/16695664.html>
2. <https://blog.csdn.net/ZhrXg/article/details/126833559>
3. <https://blog.csdn.net/ZhrXg/article/details/126833559>
4. [Linux ping指令](#)
5. [ifconfig 用法总结](#)