

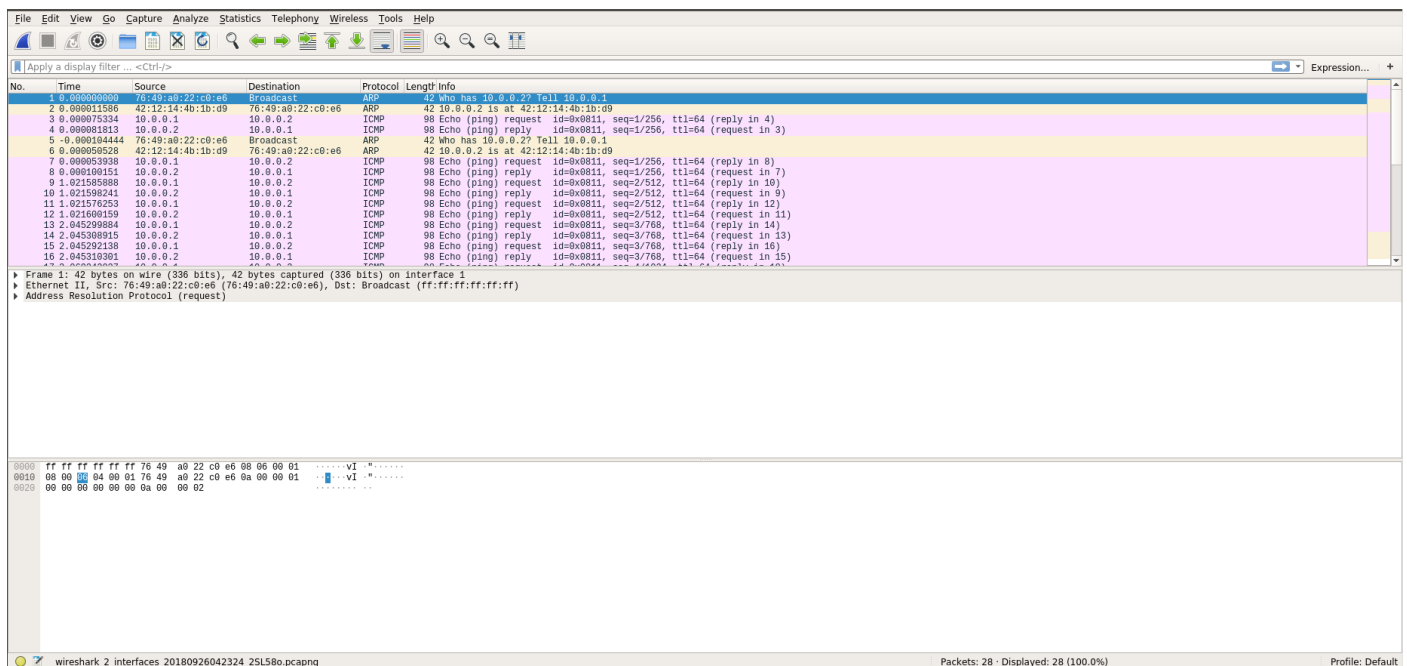
# Part.I

1. 安裝 mininet，可藉由 **apt-get install mininet** 或其他方式完成安裝。
2. 使用“mn”可建立基本的虛擬拓樸。
3. 有幾個基本指令可以顯示現在的虛擬拓樸中節點資訊或鏈結的訊息等，如“nodes”、“net”、“dump”等，請嘗試使用這些指令，並觀察顯示的訊息，可使用 **help** 查詢 mininet 可支援的指令。

```
kelvin@ubuntu:~/Desktop$ sudo -i
root@ubuntu:~# mn
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> nodes
available nodes are:
h1 h2 s1
mininet> nets
*** Unknown command: nets
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=9288>
<Host h2: h2-eth0:10.0.0.2 pid=9290>
<OVSBridge s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=9295>
```

4. 請打開 **wireshark**，擷取兩個虛擬 host 的網卡，再使用 mininet 從 **h1 ping h2**，將你所看到的 wireshark 畫面擷取下來。

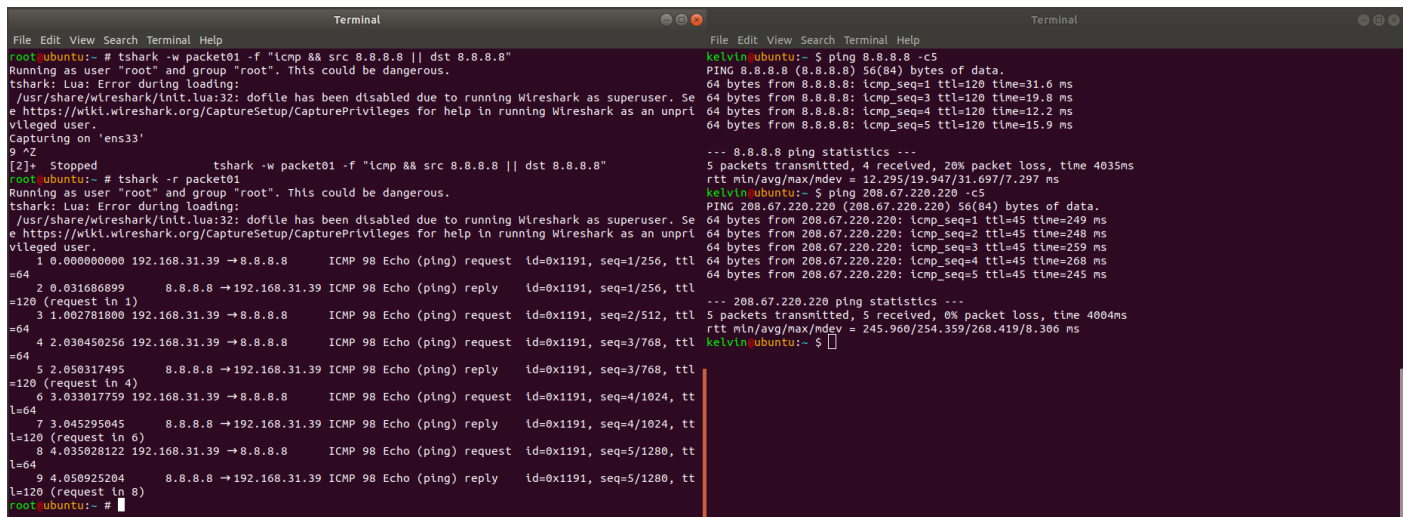


# Part.II

## 1. tshark

請使用 **tshark** 指令搭配正確的參數達到以下的要求：

- 抓取”icmp”的封包， 並且來源或目的是”8.8.8.8”
- 將擷取的封包儲存為一個檔案， 名稱為”packet01”
- 下達正確的指令後，請 **ping 8.8.8.8** 及 **ping 208.67.220.220**
- 結束後使用 **tshark** 指令查看擷取的檔案
- 以上請寫出正確的指令並截圖證明



The image shows two terminal windows. The left window shows the execution of `tshark -w packet01 -f "icmp && src 8.8.8.8 || dst 8.8.8.8"` and `tshark -r packet01`. The right window shows the execution of `ping 8.8.8.8 -c5` and `ping 208.67.220.220 -c5`, followed by the output of `tshark` showing captured ICMP packets.

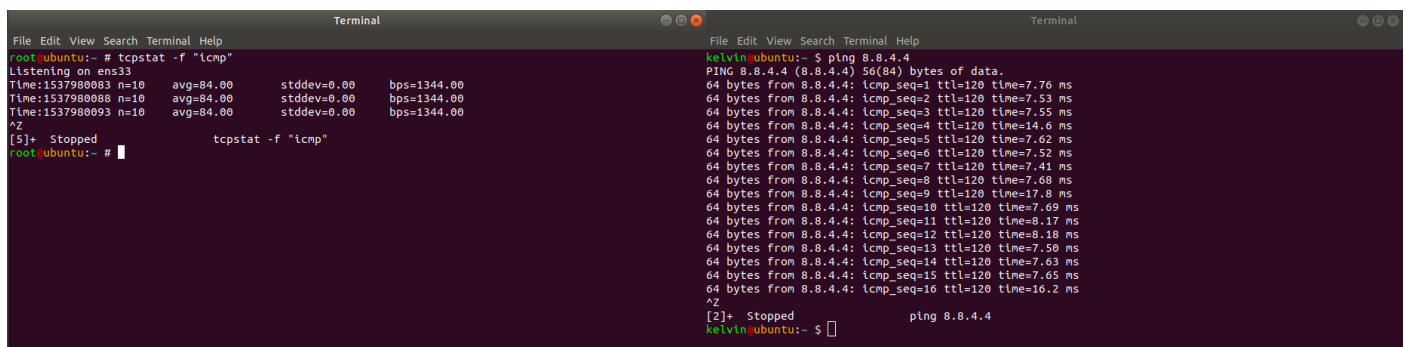
截取指令(root)：**tshark -w packet01 -f "icmp && host 8.8.8.8"**

讀取指令(root)：**tshark -r packet01**

## 2. tcpstat

請使用 **tcpstat** 指令搭配正確的參數達到以下的要求：

- 抓取”icmp”的封包
- ping** 任意位址
- 完成後中斷 **tcpstat**，將顯示的結果截圖，並寫出正確的指令



The image shows two terminal windows. The left window shows the execution of `tcpstat -f "icmp"` and the output showing statistics for ICMP traffic. The right window shows the execution of `ping 8.8.4.4` and the output showing ping results.

指令：**tcpstat -f "icmp"**

### 3. tcpdump & tcpstat & gnuplot

a. 使用 **tcpdump** 擷取網路封包，每台電腦的網卡代號可能不同

tcpdump -i eth1 -w rawdata.dmp

b. 開啟瀏覽器瀏覽網頁約一分鐘

c. 中斷 **tcpdump**

d. 使用 **tcpstat** 將擷取的檔案做格式化

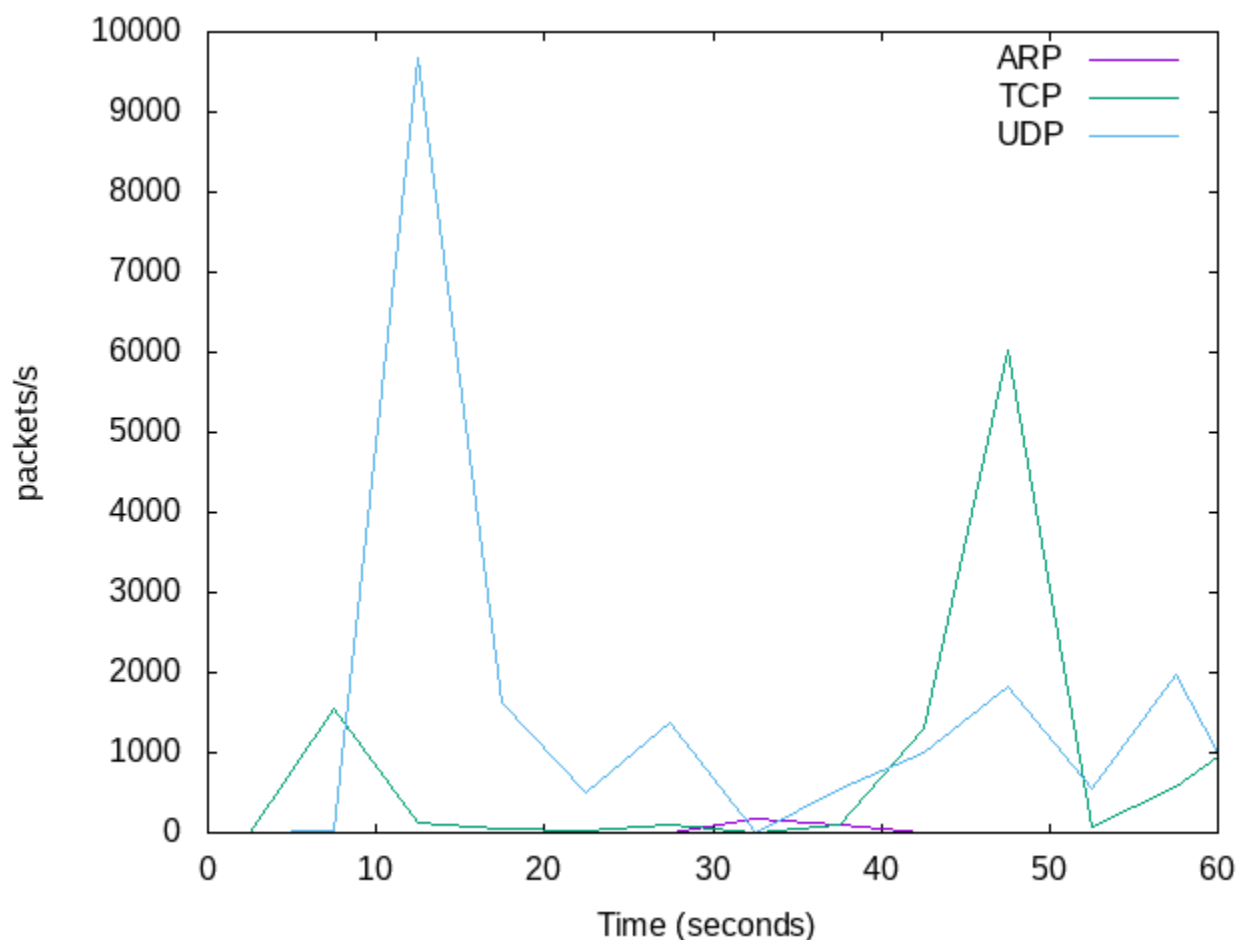
tcpstat -r rawdata.dmp -o "%r %A %T %U %I %B\n" > tcpstat.log

e. 使用 **vim** 寫一個 **script**，名稱為"script1"，**script** 內容如下圖

```
set terminal png
set style data lines
set xlabel "Time (seconds)"
set ylabel "packets/s"
plot [00:60] "tcpstat.log" using 1:2 title "ARP", \
            "tcpstat.log" using 1:3 title "TCP", \
            "tcpstat.log" using 1:4 title "UDP"
```

f. 利用 **gnuplot** 繪圖，產生如下的圖表

gnuplot script1 > graph1.png



#### 4.mininet & iperf & gnuplot

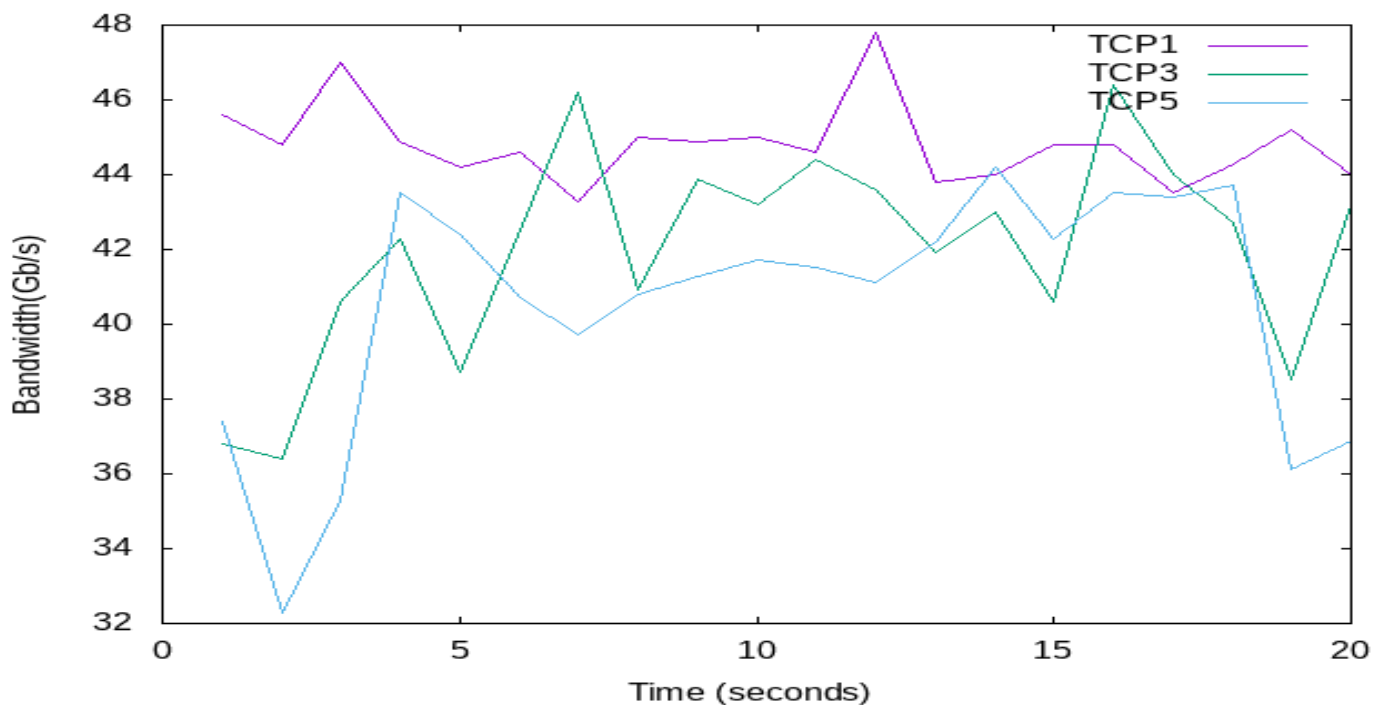
4-1. 請說明 **iperf** 的用途，以及在什麼情況下你會需要使用它？

**iperf** 是一個測試網路性能的工具，可以測試 TCP 和 UDP 的頻寬性能，可以根據需要調整參數，根據結果回報頻寬、延遲、封包丟失等資料。

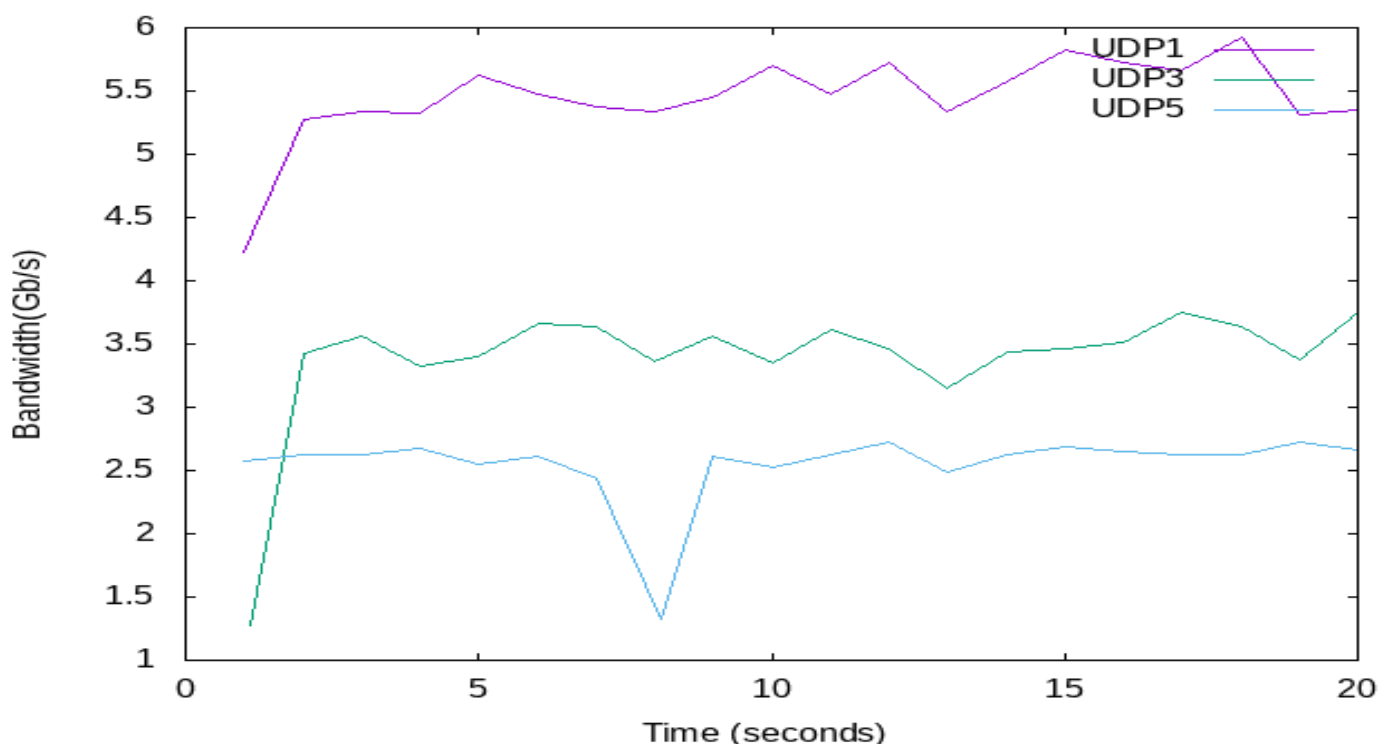
4-2. 請在 **mininet** 下設計 **topology**，使用 **iperf3** 指令開啟 **Server** 和 **Client**，使 **Server** 及 **Client** 之間有 0~4 個節點，並測量使用 **TCP** 及 **UDP** 傳輸時不同數量節點的頻寬變化，將結果存成檔案。

4-3. 請參考 Part2 第三小題自行修改 **script**，將第二步 **tcp** 及 **udp** 的結果使用參考指令處理後，利用 **gnuplot** 繪製兩張結果圖。

**TCP:**



**UDP:**



#### 4-4. 請說明 TCP 及 UDP 產生結果差異的原因。

從兩個結果皆可看出節點越多頻寬會有些微的下降，UDP 下降的幅度更是顯著，而 UDP 頻寬遠低於 TCP 的原因在於，UDP 並不會確認封包是否送達，而伺服器將掉包視為無效的頻寬，因此沒有送到的部分將會變成頻寬的浪費，導致最後頻寬無法上升。

### 5. netperf

請利用 netperf 完成以下要求【自行使用 mininet 產生 hosts，利用 xterm <host> 指令開啟個別 host 視窗，即可進行量測】

- 測量 Client 與 Server 間的 TCP 網路效能
- 測量 Client 與 Server 間的 UDP 網路效能
- 請寫出正確的指令並截圖證明

server: netserver

client(TCP): netperf -H 10.0.0.1 -l 20

client(UDP): netperf -t UDP\_STREAM -H 10.0.0.1 -l 20



The image shows two terminal windows side-by-side. The left window, titled "host: h1", shows the execution of netserver and ifconfig commands. The right window, titled "host: h2", shows the execution of netperf for both TCP and UDP tests, displaying detailed performance metrics.

```
root@kelvin-vm:~# netserver
Starting netserver with host 'IN(6)ADDR_ANY' port '12865' and family AF_UNSPEC
root@kelvin-vm:~# ifconfig
h1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::18aa:83ff:fe2c:d615 prefixlen 64 scopeid 0x20<link>
    ether 1a:aa:83:c:d6:15 txqueuelen 1000 (Ethernet)
    RX packets 15163267 bytes 126799368395 (126.7 GB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1217034 bytes 80329700 (80.3 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kelvin-vm:~#
```

```
root@kelvin-vm:~# netperf -H 10.0.0.1 -l 20
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.0.0.1 () po
rt 0 AF_INET : demo
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput
bytes bytes bytes secs. 10^6bits/sec

87380 87380 87380 20.71 46218.03
root@kelvin-vm:~# netperf -H 10.0.0.1 -l 20 -t UDP_STREAM
MIGRATED UDP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.0.0.1 () po
rt 0 AF_INET : demo
Socket Message Elapsed Messages
Size Size Time Okay Errors Throughput
bytes bytes bytes secs # # 10^6bits/sec

212992 65507 20.73 291437 0 7366.28
212992 65507 20.73 284124 0 7181.44

root@kelvin-vm:~#
root@kelvin-vm:~#
root@kelvin-vm:~#
root@kelvin-vm:~#
```