

ELEC-E7130 - Internet Traffic Measurements and Analysis  
Assignment 3 Network capture  
Haibi Peng 875552

## Task 1: Analyze flow data

### Solution:

#### 1. Describe how you generated flow data

For Task 1, I chose to do it on the Aalto computer by VDI. So at first, I set the environment using the provided command line code:

```
source /work/courses/unix/T/ELEC/E7130/general/use.sh
```

Then I could use the packet data analysis tool *CoralReef*. It is a CLI tool. Next, I use the tool in the set *crl\_flow* to generate the flow data:

```
crl_flow -Ci=3600 -cl -Tf60 -O %i.t2 -Cai=1 flow.pcap
```

This step generated 6 *.t2* files. I then used *awk* tool to combine useful data together into one file:

```
1. awk '{line[NR]=$0} END {for(i=0;i<=NR-3;i++) print line[i]}' 0.t2 >> flowdata.t2
2. awk '{line[NR]=$0} END {for(i=28;i<=NR-3;i++) print line[i]}' 1.t2 >> flowdata.t2
3. awk '{line[NR]=$0} END {for(i=28;i<=NR-3;i++) print line[i]}' 2.t2 >> flowdata.t2
4. awk '{line[NR]=$0} END {for(i=28;i<=NR-3;i++) print line[i]}' 3.t2 >> flowdata.t2
5. awk '{line[NR]=$0} END {for(i=28;i<=NR-3;i++) print line[i]}' 4.t2 >> flowdata.t2
6. awk '{line[NR]=$0} END {for(i=28;i<=NR-3;i++) print line[i]}' 5.t2 >> flowdata.t2
7. awk '{line[NR]=$0} END {for(i=28;i<=NR-3;i++) print line[i]}' 6.t2 >> flowdata.t2
```

Then I got a file *flowdata.t2* including all useful information from *flow.pcap*, which was used for selection(top-ten host pairs, etc) task. For basic statistics of flow data, I generated a *.csv* file by using *awk*:

```
1. awk '{line[NR]=$0} END {for(i=28;i<=NR-3;i++) print line[i]}' 0.t2 >> flowdata.csv
2. awk '{line[NR]=$0} END {for(i=29;i<=NR-3;i++) print line[i]}' 1.t2 >> flowdata.csv
3. awk '{line[NR]=$0} END {for(i=29;i<=NR-3;i++) print line[i]}' 2.t2 >> flowdata.csv
4. awk '{line[NR]=$0} END {for(i=29;i<=NR-3;i++) print line[i]}' 3.t2 >> flowdata.csv
5. awk '{line[NR]=$0} END {for(i=29;i<=NR-3;i++) print line[i]}' 4.t2 >> flowdata.csv
6. awk '{line[NR]=$0} END {for(i=29;i<=NR-3;i++) print line[i]}' 5.t2 >> flowdata.csv
7. awk '{line[NR]=$0} END {for(i=29;i<=NR-3;i++) print line[i]}' 6.t2 >> flowdata.csv
```

Part of the data looks like below:

#src	dst	pro	ok	sport	dport	pkts	bytes	flows	first	latest
216.53.2 50.61	202.132. 209.187	6	1	443	62147	9	3618	1	1491968654. 08889	1491968654. 20812
216.53.2 50.125	163.35.1 16.144	17	1	443	55436	29	5611	1	1491966751. 87016	1491966857. 98093
216.53.2 50.113	163.35.2 47.23	6	1	443	60886	1	40	1	1491968175. 889	1491968175. 889
216.53.2 50.125	163.35.9 4.234	6	1	443	56189	11	6060	1	1491966316. 44205	1491966316. 62398
216.53.2 50.70	202.132. 209.187	6	1	443	59936	10	5012	1	1491968016. 81972	1491968064. 65697

...	...	...	...	...	...	...	...	...	...	...
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

## 2. Provide descriptive statistics

For total number of flows, I used **grep** tool to get the IPv4 flows from *flowdata.t2*:

```

1. % grep "#      IPv4 flows: " flow.t2
2. #      IPv4 flows: 103696
3. #      IPv4 flows: 107536
4. #      IPv4 flows: 108416
5. #      IPv4 flows: 112925
6. #      IPv4 flows: 114909
7. #      IPv4 flows: 111652
8. #      IPv4 flows: 2859

```

By adding them together, I could get the total number of flows, which should be:

IPv4 flow 0	103696
IPv4 flow 1	107536
IPv4 flow 3	108416
IPv4 flow 3	112925
IPv4 flow 4	114909
IPv4 flow 5	111652
IPv4 flow 6	2859
Total flows	661993

For minimum, median, mean and maximum flow sizes in bytes and packets, I used Python scripts to do the calculation works:

```

1. def caldata(datafile, datakind):
2.     with open(datafile, 'r') as f:
3.         rawdata = csv.reader(f)
4.         data=[]
5.         for i in rawdata:
6.             i = i[0].split('\t')
7.             if i[0]!='#src':
8.                 continue
9.             else:
10.                 if datakind == 'bytes':
11.                     data.append(int(i[7]))
12.                 else:
13.                     data.append(int(i[6]))
14.     dt=sorted(data)
15.     med=dt[len(dt)/2]
16.     mean=sum(data)/len(data)
17.     Max=max(data)
18.     Min=min(data)
19.     results=[Min, med, mean, Max]
20.     return results
21.

```

Finally, I put the calculated results in a .csv file. So the descriptive statistics are:

	bytes	packets
Minimum	40	1
Median	2413	9
Mean	42014	44
Maximum	5669964196	3980504

### 3. Provide table of top-ten host pairs

For this task, I used **t2\_top** tool in the set of **CoralReef**.

Based on number of flows:

```
t2_top -Sf -n 10 < flowdata.t2 > top10flows.t2
```

whose results are:

```
1. # begin Tuple Table (expired) for subif: 0[0] (497211 entries)
2. #KEYs  pkts  bytes  flows  (top 10 sorted by flows)
3. 216.53.250.116 163.35.138.218 6 1 443 51596 498 116859 47
4. 216.53.250.116 163.35.138.218 6 1 443 51077 543 173942 41
5. 216.53.250.116 163.35.138.218 6 1 443 51290 495 123899 38
6. 216.53.250.116 163.35.139.159 6 1 443 58322 355 71776 36
7. 216.53.250.115 163.35.92.106 6 1 443 49537 190 22268 31
8. 216.53.250.116 163.35.237.167 17 1 443 64347 834 592621 31
9. 216.53.250.115 163.35.92.106 6 1 443 49856 162 20034 31
10. 216.53.250.115 163.35.92.106 6 1 443 49927 165 20306 31
11. 216.53.250.116 163.35.138.218 6 1 443 50606 2523 2492001 27
12. 216.53.250.125 163.35.99.123 17 1 443 49153 854 813396 27
13. # end of text table
```

Based on number of bytes:

```
t2_top -Sb -n 10 < flowdata.t2 > top10bytes.t2
```

whose results are:

```
1. # begin Tuple Table (expired) for subif: 0[0] (497211 entries)
2. #KEYs  pkts  bytes  flows  (top 10 sorted by bytes)
3. 216.53.250.113 163.35.92.189 6 1 443 52109 3980505 5669964248 2
4. 216.53.250.113 163.35.92.189 6 1 443 52113 1415834 2017698520 2
5. 216.53.250.113 163.35.92.189 6 1 443 52114 661635 943101550 2
6. 216.53.250.113 163.35.95.39 6 1 443 54734 482032 684602705 1
7. 216.53.250.113 163.35.95.25 6 1 443 49246 319457 454169576 2
8. 216.53.250.125 163.35.137.89 17 1 443 56974 88957 118725206 1
9. 216.53.250.113 163.35.139.140 17 1 443 58195 80156 109838344 1
10. 216.53.250.125 163.35.137.89 17 1 443 57670 57254 76630623 1
11. 216.53.250.125 163.35.137.89 17 1 443 56984 50399 67548325 1
12. 216.53.250.125 163.35.137.89 17 1 443 61155 39377 53121691 1
13. # end of text table
```

According to the results, there are many same pairs in both cases.

#### 4. Provide Top100 plots and evaluate them

For this part, I used two different method to deal with the problem.

- Command line:

I used the provided sample command line in *netcap\_intro*, which was:

```
1. cat flowdata.txt | awk '$1<$2{print $1,$2;next}{print $2,$1;}' |
2. sort | uniq -c | sort -r -n > topflows.cnt
```

Then I got a file containing how many times the host pairs occurred, from the most to the least, and I converted it into *.csv* file using *awk*, part of which are as follows:

times	src	dst
5082	163.35.11.79	216.53.250.2
4249	202.140.204.237	216.53.250.13
4047	163.35.205.38	216.53.250.125
4020	202.132.209.187	216.53.250.61
3770	163.35.235.74	216.53.250.125
3349	202.132.209.187	216.53.250.13
3324	202.140.204.237	216.53.250.61
3165	202.140.204.71	216.53.250.13
3123	202.140.204.237	216.53.250.53
...	...	...

And next I could plot the data by using Python scripts.

- Python script

First, I used *awk* command to get the *src*, *dst* and *flows* column from original flow and converted them into *.csv* file. I used *pandas* library in Python to pair the hosts and counted the numbers. Then I sorted the results. The core codes are included here:

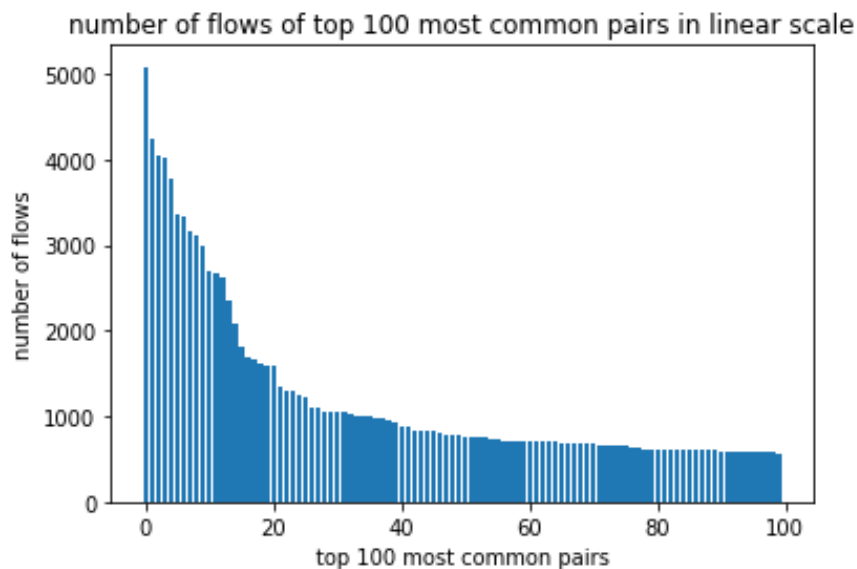
```
1. data = pd.read_csv('/u/88/pengh1/unix/Desktop/Assignment3/Task1/data/top100flow.csv')
2. grp= data.groupby(['src','dst'])#make pairs of the hosts
3. grp = list(grp)[3:-1]#convert into list
4. pairs=[len(grp[i][1]) for i in range(len(grp))]
5. grp0=bubbleSort(pairs, grp)#sort based on the number of flows
6. top = [len(grp[i][1]) for i in range(len(grp))]
7. print('flows of top 100 most common pairs:', top[0:100])
```

The final output looked like this:

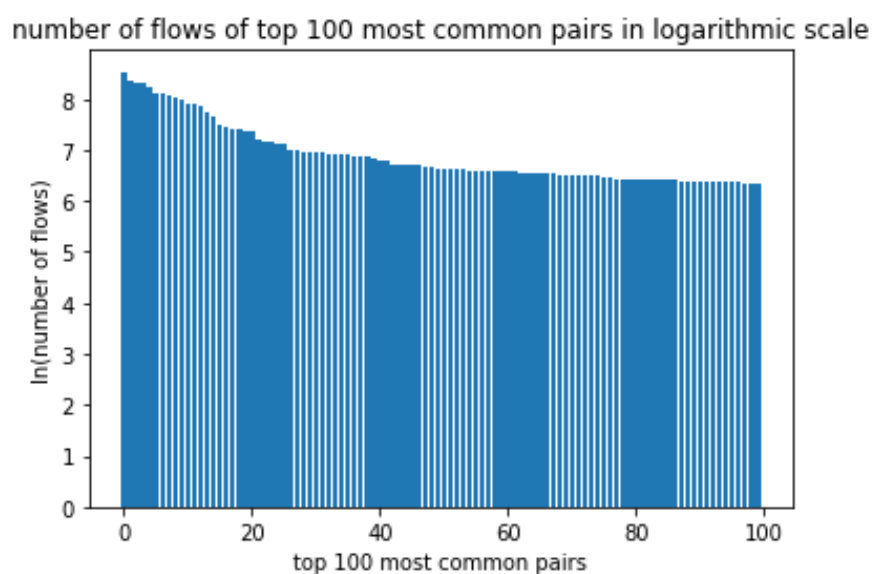
```
In [12]: runfile('/u/88/pengh1/unix/Desktop/Assignment3/Task1/data/groupby.py', wdir='/u/88/pengh1/
unix/Desktop/Assignment3/Task1/data')
('flows of top 100 most common pairs:', [5082, 4249, 4047, 4020, 3770, 3349, 3324, 3165, 3123, 2981,
2689, 2673, 2633, 2339, 2090, 1806, 1690, 1662, 1626, 1594, 1584, 1349, 1295, 1288, 1250, 1219,
1088, 1088, 1056, 1054, 1048, 1040, 1015, 1012, 1008, 989, 975, 968, 957, 930, 887, 877, 836, 836,
828, 824, 811, 789, 782, 769, 760, 756, 752, 747, 733, 733, 719, 718, 716, 715, 714, 711, 704, 704,
697, 687, 686, 683, 680, 675, 674, 664, 661, 657, 657, 650, 644, 621, 619, 617, 616, 616, 615, 613,
610, 609, 606, 601, 599, 598, 594, 593, 587, 587, 582, 579, 579, 577, 575, 566])
```

Basically they are in line with the results of command line.

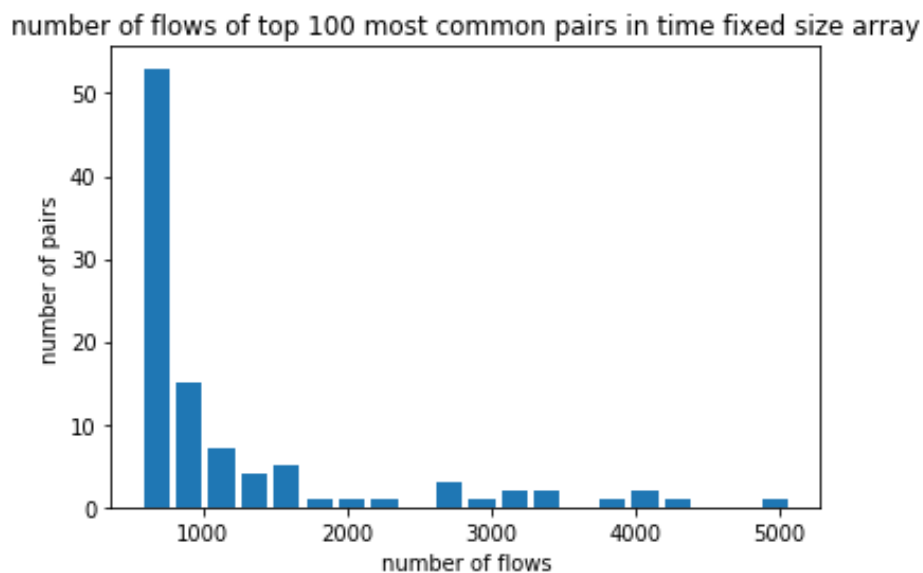
Plot of the number of flows of top 100 most common pairs in linear scale:



Plot of the number of flows of top 100 most common pairs in logarithmic scale:



5. Provide Top100 plots with fixed-array approach.



## 6. Discussion on resource memory requirements.

The running time / memory consumption of command line(method 1) method was:

```
1. % /usr/bin/time -v bash pairs.sh
2. Command being timed: "bash pairs.sh"
3. User time (seconds): 2.99
4. System time (seconds): 0.20
5. Percent of CPU this job got: 90%
6. Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.51
```

The running time / memory consumption of Python script(method 2) method was:

```
1. % /usr/bin/time -v python groupby.py
2. Command being timed: "python groupby.py"
3. User time (seconds): 40.37
4. System time (seconds): 0.40
5. Percent of CPU this job got: 96%
6. Elapsed (wall clock) time (h:mm:ss or m:ss): 0:42.16
```

Apparently, using command line was far more efficient than python script. I believe it had something to do with the algorithm used in each method. I think a more efficient way to do this task is to use hash function, but I was not able to successfully complete one.

## Task 2: Packet capture

### Solution:

#### 1. Describe your measurement setup (tools and workflow).

##### - Session 1:

For network data traffic capture, I used *dumpcap* tool. First, I use *-D* option to check the available interfaces:

```
1. root@debian-gnu-linux-vm:/home/parallels/Desktop/Assignment3/Task2/Data1# dumpcap -D
2. 1. enp0s5
3. 2. any
4. 3. lo (Loopback)
5. 4. nflog
6. 5. nfqueue
7. 6. usbmon1
8. 7. usbmon2
9. 8. usbmon3
10. 9. usbmon4
```

I chose to capture the network traffic from enp0s5 interface, so the command was:

```
sudo dumpcap -a duration:3600 -i enp0s5 -s 96 -g -P -w data1.pcap
```

Among which, *-a duration:3600* means stop after 3600 seconds, a.k.a., one hour; *-g* enables group read access on the output file(s); *-P* denotes to use libpcap(.pcap) format instead of pcapng. Then I got *data1.pcap* file for further analysis.

##### - Session 2:

Also, I used the same command to capture data, while this time the duration changed to 15 minutes. Therefore, the command should be:

```
sudo dumpcap -a duration:900 -i enp0s5 -s 96 -g -P -w data2.pcap
```

Then I got *data2.pcap* file for further analysis.

As for *iperf3* and *ping* test, I used similar bash scripts from Assignment2. And the settings were as follows:

```
1. #!/bin/bash
2.
3. function rand(){
4. min=$1
5. max=$((($2-$min+1))
6. num=$((date +%s%N))
7. echo $((($num%$max+$min))
8. }
9.
10. port=$((rand 5200 5210))
11. d=$(date -Isec | tr -d : | sed s/+.*///)
12.
```

```

13. ping -c 10 ok1.iperf.comnet-student.eu -O -
    D >> /home/parallels/Desktop/Assignment3/Task2/Data2/iperfserver1/ICMP/is1-icpm-$d.txt &
14. ping -c 10 blr1.iperf.comnet-student.eu -O -
    D >> //home/parallels/Desktop/Assignment3/Task2/Data2/iperfserver2/ICMP/is2-icpm-$d.txt &
15. ing -c 10 pna-es.ark.caida.org -O -
    D >> /home/parallels/Desktop/Assignment3/Task2/Data2/researchserver1/rs1-icpm-$d.txt &
16. ping -c 10 per-au.ark.caida.org -O -
    D >> /home/parallels/Desktop/Assignment3/Task2/Data2/researchserver2/rs2-icpm-$d.txt &
17. ping -c 10 cjj-kr.ark.caida.org -O -
    D >> /home/parallels/Desktop/Assignment3/Task2/Data2/researchserver3/rs3-icpm-$d.txt &
18. iperf3 -c ok1.iperf.comnet-student.eu -t 10 -
    p $port >> /home/parallels/Desktop/Assignment3/Task2/Data2/iperfserver1/iperf/iperf-s-$d.txt &
19. iperf3 -c blr1.iperf.comnet-student.eu -t 10 -
    p $port >> /home/parallels/Desktop/Assignment3/Task2/Data2/iperfserver2/iperf/iperf-s-$d.txt

```

Among which, the duration of every ping and iperf3 session was both 10 seconds. For iperf3 session, it chose the port randomly from 5200 to 5210.

The crontab tool was used to control the frequency of script running, which are as follows:

```

1. SHELL=/bin/bash
2.
3. */3 * * * * /home/parallels/Desktop/Assignment3/Task2/Data2/activemeasurements.sh >> /home/parallels/Desktop/Assignment3/Task2/Data$

```

In this case, the script would be run every three minutes, so for 15 minutes we could get 5 files for iperf or ping results.

## 2. Summary of capture data for both sessions.

The size of trace file of data1.pcap was 21.4MB, while data2.pcap was 44.0MB.


data1.pcap Properties

Basic

Permissions

Open With

Tags



Name:

data1.pcap

Type:

Packet Capture (PCAP) (application/vnd....)

Size:

21.4 MB (21,429,170 bytes)

Parent Folder:

/home/parallels/Desktop/Assignment3/...


data2.pcap Properties

Basic

Permissions

Open With

Tags



Name:

data2.pcap

Type:

Packet Capture (PCAP) (application/vnd....)

Size:

44.0 MB (44,033,798 bytes)

Parent Folder:

/home/parallels/Desktop/Assignment3/...

For number of packets in trace file, I used the same method in Task1 to obtain the number of packets, and the results are as follows:

```

- Session 1:
- root@debian-gnu-linux-
  vm:/home/parallels/Desktop/Assignment3/Task2/Data1# grep "# IPv4 pkts: " capturefile1.t2
- # IPv4 pkts: 172803
- # IPv4 pkts: 36502
- # IPv4 pkts: 0
- root@debian-gnu-linux-
  vm:/home/parallels/Desktop/Assignment3/Task2/Data1# grep "# IPv6 pkts: " capturefile1.t2

```



```

- # IPv6 pkts: 1368
- # IPv6 pkts: 209
- # IPv6 pkts: 0
- Session 2:
- root@debian-gnu-linux-
  vm:/home/parallels/Desktop/Assignment3/Task2/Data2/dt# grep "# IPv4 pkts: " capturefile2.t2
- # IPv4 pkts: 525943
- # IPv4 pkts: 0
- root@debian-gnu-linux-
  vm:/home/parallels/Desktop/Assignment3/Task2/Data2/dt# grep "# IPv6 pkts: " capturefile2.t2
- # IPv6 pkts: 352
- # IPv6 pkts: 0

```

By adding them together, I could get the total number of flows, which should be:

	Session 1	Session 2
IPv4 pkts 1	172803	525943
IPv4 pkts 2	36502	0
IPv4 pkts 3	0	N/A
IPv6 pkts 1	1368	352
IPv6 pkts 2	209	0
IPv6 pkts 3	0	N/A
Total packets	210882	526295

For total size of packets in trace file, I used the same method to obtain them for both size, and the results are as follows:

- Session 1:

```

1. root@debian-gnu-linux-
  vm:/home/parallels/Desktop/Assignment3/Task2/Data1# grep "# IPv4 bytes: " capturefile1.t2
2. # IPv4 bytes: 170730522
3. # IPv4 bytes: 40438989
4. # IPv4 bytes: 0
5. root@debian-gnu-linux-
  vm:/home/parallels/Desktop/Assignment3/Task2/Data1# grep "# IPv6 bytes: " capturefile1.t2
6. # IPv6 bytes: 120943
7. # IPv6 bytes: 18532
8. # IPv6 bytes: 0

```

- Session 3:

```

1. root@debian-gnu-linux-
  vm:/home/parallels/Desktop/Assignment3/Task2/Data2/dt# grep "# IPv4 bytes: " capturefile2.t2
2. # IPv4 bytes: 685793005
3. # IPv4 bytes: 0
4. root@debian-gnu-linux-
  vm:/home/parallels/Desktop/Assignment3/Task2/Data2/dt# grep "# IPv6 bytes: " capturefile2.t2
5. # IPv6 bytes: 34475
6. # IPv6 bytes: 0

```

By adding them together, I could get the total size of packes, which should be:

	Session 1	Session 2
IPv4 bytes 1	170730522	685793005
IPv4 bytes 2	40438989	0
IPv4 bytes 3	0	N/A
IPv6 bytes 1	120943	34475
IPv6 bytes 2	18532	0
IPv6 bytes 3	0	N/A
Total bytes	211308986	685827480

3. Were there differences between capture file statistics and counters?

The comparison of statistics from capture file and counters is as follows:

	capture file	counters
number of packets / session 1	210882	210884
total size of packets / session 1	211308986	214261418
number of packets / session 2	526295	526295
total size of packets / session 2	685827480	693195610

According to the table, we could see that there were some differences between capture file statistics and counters numbers. For example, for the number of packets of session 1, the data from capture file was slightly less than from counters. And also for the total size of packets of both sessions, the data from capture file were somehow smaller than from counters.

4. Any observations on those two sessions?

When transferred the original *.pcap* files into *.t2* files, it would be split into several *.t2* files. And according to the results of *grep* command, some of the packets or bytes data were 0, just like the codes and results mentioned above.

## Task 3: Analyze captured traffic

### Solution:

#### 1. Describe your analysis setup. Include code snippets.

For Task3, I used two different kinds of tools to analyze captured traffic, which were Wireshark and CoralReef. For question III, IV I used CoralReef to obtain the results. For other questions, I mainly used Wireshark to solve the problem, while question VI and VIII I used *tcpstat*.

First, I converted the *.pcap* file to *.t2* file:

```
crl_flow -Ci=3600 -cl -Tf60 -O %i.t2 -Cai=1 data1.pcap
```

This step generated 3 *.t2* files. I then used *awk* tool to combine useful data together into one file:

```
1. awk '{line[NR]=$0} END {for(i=0;i<=NR-3;i++) print line[i]}' 0.t2 >> capturefile1.t2
2. awk '{line[NR]=$0} END {for(i=28;i<=NR-3;i++) print line[i]}' 1.t2 >> capturefile1.t2
3. awk '{line[NR]=$0} END {for(i=28;i<=NR-3;i++) print line[i]}' 2.t2 >> capturefile1.t2
```

Then I used *t2\_top* tool in the set of **CoralReef**.

Based on number of bytes:

```
t2_top -Sf -n 15 < flowdata.t2 > top15bytes.t2
```

Based on number of packets:

```
t2_top -Sf -n 15 < flowdata.t2 > top10packets.t2
```

For *tcpstat* tool, by reading the documents, one could choose the needed information and output them into files. The command is as follows:

```
tcpstat -o "%s\t%N\t%n\t%V\t%I\t%T\t%U\t%C\t%b\t%p\n" -r data1.pcap >> data1.csv
```

which would output the following data:

time	bytes	packets	IPv6 packets	IPv4 packets	TCP packets	UDP packets	ICMP and ICMPv6 packets	bit per second	packet per second

#### 2. Answers to questions

I. How many IP (and IPv6 if any) hosts are communicating?

Using Wireshark-Statistics-Endpoint, we could see the number of IPv4 and IPv6 hosts:

Therefore, there are 161 IPv4 hosts and 5 IPv6 hosts are communicating.

Wireshark · Endpoints · data1.pcap								
Ethernet · 7		IPv4 · 161		IPv6 · 5		TCP · 939		UDP · 1483
Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	
3.219.203.195	52	12 k	25	8,472	27	3,958	—	
10.211.55.1	4,294	657 k	1,417	427 k	2,877	229 k	—	
10.211.55.2	29	6,605	29	6,605	0	0	—	
10.211.55.8	209,276	214 M	49,821	6,976 k	159,455	207 M	—	
14.18.245.239	47	16 k	23	10 k	24	5,416	—	
34.107.159.67	1,336	1,085 k	925	1,052 k	411	32 k	—	
34.120.202.204	56	10 k	29	6,163	27	4,650	—	

II. How many hosts were tried to contact to, but communication failed for a reason or another? Can you identify different subclasses of failed communications?

Using Wireshark-Analyze-Expert Information, we could see the abnormal or failed connections:

Wireshark · Expert Information · data1.pcap					
Packet	Summary	Group	Protocol	Count	
Warning	DNS query retransmission. Original request in frame 168991	Protocol	mDNS	2	
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	15	
Warning	Previous segment(s) not captured (common at capture start)	Sequence	TCP	5	
Warning	ACKed segment that wasn't captured (common at capture start)	Sequence	TCP	25	
Warning	Connection reset (RST)	Sequence	TCP	435	
Warning	Ignored Unknown Record	Protocol	SSL	132340	
Note	This frame is a (suspected) retransmission	Sequence	TCP	2	
Note	ACK to a TCP keep-alive segment	Sequence	TCP	1244	
Note	TCP keep-alive segment	Sequence	TCP	1255	
Note	Duplicate ACK (#1)	Sequence	TCP	14	
Chat	Possible traceroute: hop #9, attempt #1	Sequence	UDP	8	
Chat	TCP window update	Sequence	TCP	11	
Chat	GET /cgi-bin/frame_html?sid=lrndGiEnTQzWZ0	Sequence	HTTP	80	
Chat	Connection establish acknowledge (SYN+ACK): server port 443	Sequence	TCP	790	
Chat	Connection establish request (SYN): server port 443	Sequence	TCP	792	
Chat	M-SEARCH * HTTP/1.1\r\n	Sequence	SSDP	120	
Chat	Connection finish (FIN)	Sequence	TCP	1256	

The subclasses are as such:

- ACK to a TCP keep-alive segment
- TCP keep-alive segment
- Duplicate ACK(#1)
- ...

III. Top 15 hosts by byte counts.

```

1. # begin Tuple Table (expired) for subif: 0[0] (3081 entries)
2. #KEYs   pkts   bytes   flows   (top 15 sorted by bytes)
3. 185.10.104.115  10.211.55.8 6   1   443 34450   31347   45546178   1
4. 109.105.109.207 10.211.55.8 6   1   443 55674   21819   31504884   1
5. 109.105.109.205 10.211.55.8 6   1   443 41670   17091   24691055   1
6. 185.10.104.120  10.211.55.8 6   1   443 54906   14886   21070983   1
7. 109.105.109.205 10.211.55.8 6   1   443 40914   8382    12100139   1
8. 58.254.150.35   10.211.55.8 6   1   443 38340   7907    10916078   1
9. 173.194.150.203 10.211.55.8 6   1   443 47904   3401    4885998    1
10. 173.194.150.203 10.211.55.8 6   1   443 47172   3243    4610852    1
11. 109.105.109.207 10.211.55.8 6   1   443 55610   2716    3857778    1
12. 109.105.109.204 10.211.55.8 6   1   443 37394   2245    3225619    1
13. 173.194.150.203 10.211.55.8 6   1   443 47656   2162    3112256    1
14. 58.254.150.35   10.211.55.8 6   1   443 37992   2131    2884801    1
15. 173.194.150.203 10.211.55.8 6   1   443 47976   1129    1603594    1
16. 173.194.150.203 10.211.55.8 6   1   443 48016   1133    1596605    1
17. 173.194.150.203 10.211.55.8 6   1   443 48014   1119    1584420    1

```

IV. Top 15 hosts by packet counts.

```

1. # begin Tuple Table (expired) for subif: 0[0] (3081 entries)
2. #KEYs    pkts    bytes    flows    (top 15 sorted by pkts)
3. 185.10.104.115  10.211.55.8 6 1 443 34450 31347 45546178 1
4. 109.105.109.207 10.211.55.8 6 1 443 55674 21819 31504884 1
5. 109.105.109.205 10.211.55.8 6 1 443 41670 17091 24691055 1
6. 185.10.104.120  10.211.55.8 6 1 443 54906 14886 21070983 1
7. 109.105.109.205 10.211.55.8 6 1 443 40914 8382 12100139 1
8. 58.254.150.35   10.211.55.8 6 1 443 38340 7907 10916078 1
9. 10.211.55.8     58.254.150.35 6 1 38340 443 4375 203508 1
10. 173.194.150.203 10.211.55.8 6 1 443 47904 3401 4885998 1
11. 173.194.150.203 10.211.55.8 6 1 443 47172 3243 4610852 1
12. 10.211.55.8     185.10.104.115 6 1 34450 443 3159 134072 1
13. 10.211.55.8     109.105.109.207 6 1 55674 443 2934 230232 1
14. 216.58.211.14   10.211.55.8 6 1 443 54932 2755 1136275 1
15. 109.105.109.207 10.211.55.8 6 1 443 55610 2716 3857778 1
16. 10.211.55.8     109.105.109.205 6 1 41670 443 2340 188058 1
17. 109.105.109.204 10.211.55.8 6 1 443 37394 2245 3225619 1

```

V. Top 10 TCP and top 5 UDP port numbers (by packet count).

top 10 UDP port numbers: using Wireshark-Statistics-Endpoints-TCP

Wireshark · Endpoints · data1.pcap							
Ethernet · 7		IPv4 · 161		IPv6 · 5		TCP · 939	
UDP · 1483							
Address	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
185.10.104.115	443	36,126	47 M	32,329	47 M	3,797	225 k
10.211.55.8	34450	34,506	46 M	3,159	178 k	31,347	45 M
109.105.109.205	443	29,911	38 M	25,871	37 M	4,040	383 k
109.105.109.207	443	28,807	36 M	24,728	35 M	4,079	373 k
10.211.55.8	55674	24,753	32 M	2,934	271 k	21,819	31 M
173.194.150.203	443	22,296	25 M	17,640	25 M	4,656	302 k
10.211.55.8	41670	19,431	25 M	2,340	220 k	17,091	24 M
185.10.104.120	443	17,036	21 M	14,895	21 M	2,141	117 k
10.211.55.8	54906	17,017	21 M	2,131	116 k	14,886	21 M
58.254.150.35	443	15,864	14 M	10,228	14 M	5,636	352 k

top 5 UDP port numbers: using Wireshark-Statistics-Endpoints-UDP

Wireshark · Endpoints · data1.pcap							
Ethernet · 7		IPv4 · 161		IPv6 · 5		TCP · 939	
UDP · 1483							
Address	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
10.211.55.1	53	4,284	653 k	1,412	425 k	2,872	228 k
fe80::21c:42ff:fe00:18	53	1,459	150 k	25	8,476	1,434	142 k
239.255.255.250	1900	120	25 k	0	0	120	25 k
224.0.0.251	5353	30	6,788	0	0	30	6,788
10.211.55.2	5353	29	6,605	29	6,605	0	0

VI. Top 10 fastest TCP connections

After sorting the output data from *tcpstat* command by bit per second, the top 10 fastest TCP connections are as follows:

time	bytes	packets	IPv6 packets	IPv4 packets	TCP packets	UDP packets	ICMP and ICMPv6 packets	bit per second	packet per second
1601848261.43178	12225222	9056	4	9052	9032	22	2	<b>19560355.2</b>	1811.2
1601848226.43178	10092553	8525	8	8517	8509	16	0	<b>16148084.8</b>	1705
1601848071.43178	8439353	6961	8	6953	6931	30	0	<b>13502964.8</b>	1392.2
1601848406.43178	7786979	6128	10	6118	6092	36	0	<b>12459166.4</b>	1225.6
1601848361.43178	7085207	5380	8	5372	5364	16	0	<b>11336331.2</b>	1076
1601848446.43178	6937585	5184	0	5184	5184	0	0	<b>11100136</b>	1036.8
1601848066.43178	5241393	4424	10	4414	4391	33	0	<b>8386228.8</b>	884.8
1601848086.43178	3729985	3080	2	3078	3074	6	0	<b>5967976</b>	616
1601848101.43178	3695678	3047	0	3047	3043	4	0	<b>5913084.8</b>	609.4
1601848001.43178	3512008	3869	10	3859	3824	45	0	<b>5619212.8</b>	773.8

## VII. Top 10 longest (by time) TCP connections

Using Wireshark and entering the filter expression *tcp.analysis.ack\_rtt*, we could see the top 10 longest TCP connections:

tcp.analysis.ack_rtt				
No.	Time	Source	Destination	Protocol
2108...	3590.713240	172.217.21.174	10.211.55.8	TCP
2108...	3590.713104	10.211.55.8	172.217.21.174	TLSv1.2
2108...	3590.711882	10.211.55.8	172.217.21.174	TCP
2108...	3590.688489	172.217.21.174	10.211.55.8	TCP
2108...	3590.688478	172.217.21.174	10.211.55.8	TCP
2108...	3584.292652	10.211.55.8	109.105.109.205	TCP
2108...	3584.292549	10.211.55.8	109.105.109.205	TCP
2108...	3584.292233	10.211.55.8	109.105.109.205	TCP
2108...	3584.292049	10.211.55.8	109.105.109.205	TCP
2108...	3584.291411	10.211.55.8	109.105.109.205	TCP

## VIII. Bit and packet rate over time (e.g. tcpstat)

Part of the data are as follows:

time	bytes	packets	IPv6 packets	IPv4 packets	TCP packets	UDP packets	ICMP and ICMPv6 packets	bit per second	packet per second
1601846836.43178	506416	502	0	502	495	7	0	<b>810265.6</b>	<b>100.4</b>
1601846841.43178	258399	346	4	342	322	24	0	<b>413438.4</b>	<b>69.2</b>
1601846846.43178	206315	354	12	342	318	34	2	<b>330104</b>	<b>70.8</b>
1601846851.43178	820391	870	12	858	825	45	0	<b>1312625.6</b>	<b>174</b>
1601846856.43178	1445273	1505	10	1495	1458	47	0	<b>2312436.8</b>	<b>301</b>
1601846861.43178	1611462	1710	10	1700	1676	34	0	<b>2578339.2</b>	<b>342</b>
1601846866.43178	409307	466	4	462	440	26	0	<b>654891.2</b>	<b>93.2</b>
1601846871.43178	9232	31	2	29	21	10	0	<b>14771.2</b>	<b>6.2</b>
...	...	...	...	...	...	...	...	...	...

### 3. Did byte and packet count top hosts differ?

There are 8 out of 15 pairs hosts in byte and packet count top hosts are the same, only just differ in ranking. It might be somehow likely to say that bytes are in line with the packets counts.

#### 4. Any interesting observations?

From the table in *VIII. Bit and packet rate over time*, I found that:

total packets = IPv4 packets + IPv6 packets = TCP packets + UDP packets + ICMP and ICMPv6 packets

And if one sort the packets from largest to smallest, he might find that basically more packets means more bytes:

time	bytes	packets	IPv6 packets	IPv4 packets	TCP packets	UDP packets	ICMP and ICMPv6 packets	bit per second	packet per second
1601848261.43178	<b>12225222</b>	<b>9056</b>	4	9052	9032	22	2	19560355.2	1811.2
1601848226.43178	<b>10092553</b>	<b>8525</b>	8	8517	8509	16	0	16148084.8	1705
1601848071.43178	<b>8439353</b>	<b>6961</b>	8	6953	6931	30	0	13502964.8	1392.2
1601848406.43178	<b>7786979</b>	<b>6128</b>	10	6118	6092	36	0	12459166.4	1225.6
1601848361.43178	<b>7085207</b>	<b>5380</b>	8	5372	5364	16	0	11336331.2	1076
1601848446.43178	<b>6937585</b>	<b>5184</b>	0	5184	5184	0	0	11100136	1036.8
1601848066.43178	<b>5241393</b>	<b>4424</b>	10	4414	4391	33	0	8386228.8	884.8
1601848001.43178	<b>3512008</b>	<b>3869</b>	10	3859	3824	45	0	5619212.8	773.8
...	...	...	...	...	...	...	...	...	...

So basically one could say that bytes are in line with the packets counts.

[illegible]



For *icmp2csv.sh* script, it actually used *tshark* tool, the script is as follows:

```
1. #!/bin/sh
2. export LANG=C
3. tshark -n -r ${1:-capture.pcap} -Y icmp.code==0 \
4. -T fields -E header=y -E separator=, -E occurrence=f \
5. -e frame.time_epoch -e ip.src -e ip.dst -e icmp.type \
6. -e icmp.ident -e icmp.seq -e ip.len -e ip.hdr_len -e icmp.resptime
```

And part of the results are as follows:

frame.time_epoch	ip.src	ip.dst	icmp.type	icmp.ident	icmp.seq	ip.len	ip.hdr_len	icmp.resptime
1601845931.70317	10.211.55.8	195.148.124.36	8	3151	1	84	20	
1601845931.70359	10.211.55.8	150.183.95.135	8	3155	1	84	20	
1601845931.70535	10.211.55.8	142.93.213.224	8	3152	1	84	20	
1601845931.70566	195.148.124.36	10.211.55.8	0	3151	1	84	20	2.496
1601845931.99749	150.183.95.135	10.211.55.8	0	3155	1	84	20	293.897
1601845932.04623	142.93.213.224	10.211.55.8	0	3152	1	84	20	340.88
1601845932.70423	10.211.55.8	150.183.95.135	8	3155	2	84	20	
1601845932.70426	10.211.55.8	195.148.124.36	8	3151	2	84	20	
1601845932.70691	195.148.124.36	10.211.55.8	0	3151	2	84	20	2.647
...	...	...	...	...	...	...	...	...

- For active measurement data, I mainly used Python script(same method in Assignment 2) to extract useful information from the files.

For iperf results, part of the result output are as follows:

timestamp	typeofmeasurement	target	bitrate	timeelapsed	bytestransfered	failed
2020-10-05T002101	iperf	iperf.netlab.hut.fi (195.148.124.36)	135 Mbits/sec	10	161 MBytes	False
2020-10-05T001201	iperf	iperf.netlab.hut.fi (195.148.124.36)	147 Mbits/sec	10	175 MBytes	False
2020-10-05T000901	iperf	iperf.netlab.hut.fi (195.148.124.36)	153 Mbits/sec	10	183 MBytes	False
			0	0	0	True
2020-10-05T001801	iperf	iperf.netlab.hut.fi (195.148.124.36)	101 Mbits/sec	10	121 MBytes	False

For ping results, part of the result output are as follows:

timestamp	typeofmeasurement	target	delay	failed
2020-10-05T001501	ICMP echo request	iperf.netlab.hut.fi (195.148.124.36)	2.915	False
2020-10-05T001801	ICMP echo request	iperf.netlab.hut.fi (195.148.124.36)	4.596	False
2020-10-05T001201	ICMP echo request	iperf.netlab.hut.fi (195.148.124.36)	3.042	False
2020-10-05T000901	ICMP echo request	iperf.netlab.hut.fi (195.148.124.36)	3.193	False
2020-10-05T002101	ICMP echo request	iperf.netlab.hut.fi (195.148.124.36)	3.059	False

## 2. Answers to questions above.

I. How much there was traffic that was not iperf or ping traffic?

As far as we know, perf traffic was TCP traffic and ping traffic was ICMP traffic, meanwhile the total packets(traffic) consist of TCP packets, ICMP packets and UDP packets, so the if the traffic was not iperf nor ping traffic, it should be UDP traffic.

I used Python script to calculate how much is the UDP traffic from the output file of *tcptrace* command:

```
1. import re
2.
3. file = '/home/parallels/Desktop/Assignment3/Task2/Data2/dt/trace2.txt'
4. datasent1 = re.findall(re.compile(r'    data bytes sent:      (.{3})'), open(file).read())
5. datasent2 = re.findall(re.compile(r'    data bytes sent:      (.{3})'), open(file).read(
    ))
6.
7. bytes1 = []
8. bytes2 = []
9. for i in range(len(datasent1)):
10.     bytes1.append(datasent1[i].split(' ')[-1])
11.     bytes2.append(datasent1[i].split(' ')[-1])
12.
13. bytes1 = [i for i in bytes1 if i != '']
14. bytes2 = [i for i in bytes2 if i != '']
15.
16. totalUDPbytes = sum([int(i) for i in bytes1]) + sum([int(i) for i in bytes2])
17. print('total UDP traffic in bytes:', totalUDPbytes)
```

The final output, a.k.a. total UDP traffic in bytes was: **296526** bytes

```
1. runfile('/home/parallels/Desktop/Assignment3/Task2/Data2/dt/UDP.py', wdir='/home/parallels/Desktop
/Assignment3/Task2/Data2/dt')
2. total UDP traffic in bytes: 296526
```

II. Compare iperf results from active and passive measurements. Provide a table.

iperf results	active measurements	passive measurements
bit rate	server1: <b>134 Mbps</b> /server2: <b>17.1Mbps</b>	<b>6.236 Mbps</b>

The throughput of active measurements is apparently bigger than passive measurements.

III. Compare ping results from active and passive measurements. Provide a table.

Using Wireshark, by filtering ICMP packets:

Measurement	Captured	Displayed
Packets	526295	487 (0.1%)
Time span, s	888.832	735.863
Average pps	592.1	0.7
Average packet size, B	1317	98
Bytes	693195610	47726 (0.0%)
Average bytes/s	779 k	64
Average bits/s	6,239 k	518

And by filtering *icmp.no\_resp* :

Measurement	Captured	Displayed
Packets	526295	13 (0.0%)
Time span, s	888.832	544.076
Average pps	592.1	0.0
Average packet size, B	1317	98
Bytes	693195610	1274 (0.0%)
Average bytes/s	779 k	2
Average bits/s	6,239 k	18

ping results	active measurements	passive measurements
delay	iperf server1: <b>3.361 ms</b> iperf server2: <b>377.156 ms</b> research server1: <b>99.735 ms</b> research server2: <b>288.658 ms</b> research server1: <b>306.377 ms</b>	<b>450.81 ms</b>
packet loss ratio	iperf server1: 0 iperf server2: 25% research server1: 0 research server2: 0 research server3: 0	<b>13/487=2.67%</b>

3. Were there any systematic bias on active and passive measurements?

As far as I think, based on the comparison, there might be some systematic bias on active and passive measurements, since throughput of passive measurements was less than active measurements, and delay of passive measurements was also larger than active measurements, while it was not clear who performed better on packet loss, which might need bigger amount of data to carry out further investigation.