ELEC-E7130
Internet Traffic Measurements and Analysis

Final Assignment
From measurements to conclusions

Haibi Peng 875552

30th November 2020

Task 1: Flow data

Acquiring flow data

As is introduced in the current and previous assignment handout, I acquired the flow data using the commands as follows:

```
source /work/courses/unix/T/ELEC/E7130/general/use.sh
cd $TRACE
ls
```

In Task 1, I chose the ***flow-continue*** dataset to finish the assignments.

Data pre-processing

In this part, I am supposed to make a sampling process to extract relevant data based on subnetwork. Since the last digit of my student is 2, I chose ***133.60.168.0/24*** as my subnetwork that would be utilized in extracting process.

The commands I used to extract relevant data are as follows:

```
#!/bin/bash

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

gawk '$1~/^133\.60\.168\./||$2~/^133\.60\.168\./' $TRACE/flow-
continue/*.t2 >> /u/88/pengh1/unix/Desktop/FinalAssignment/Task1/my_subnetwork.t2
```

Then I got a ***.t2*** file ***my_subnetwork.t2***. And short samples taken from the distilled data are as follows:

src	dst	proto	valid	sport	dport	pkt	bytes	flows	start	end
74.121.108.127	133.60.168.49	6	1	1242	445	2	96	1	1491924577.679660000	1491924580.882079000
201.248.233.31	133.60.168.236	6	1	4713	445	2	96	1	1491923001.264597000	1491923004.341330000
45.222.129.200	133.60.168.247	6	1	3696	445	2	96	1	1491923639.973742000	1491923642.942049000
122.59.238.182	133.60.168.105	6	1	48839	23	1	40	1	1491923216.591854000	1491923216.591854000
216.105.82.10	133.60.168.198	6	1	1847	445	2	96	1	1491926315.308767000	1491926318.248969000
178.47.230.195	133.60.168.227	6	1	2251	23	1	40	1	1491923907.302938000	1491923907.302938000
203.127.201.65	133.60.168.29	6	1	2659	445	2	96	1	1491925782.228185000	1491925785.269885000
77.181.192.140	133.60.168.92	6	1	39949	445	2	96	1	1491924420.190987000	1491924423.237415000
196.219.99.82	133.60.168.225	6	1	380	23	1	44	1	1491924069.849576000	1491924069.849576000
67.4.3.214	133.60.168.206	6	1	2392	445	2	96	1	1491922863.584239000	1491922866.643288000
…	…	…	…	…	…	…	…	…	…	…

Data analysis

1.1: Plot traffic volume

Plot traffic volume as a function of time with at least two sufficiently different time scales (for example, bits per second in function of seconds and bits per second in function of minutes).

Solution:

In this part, the main thought of mine is trying to fit together all the flows that match the conditions. First, I sorted the dataset (***my_subnetwork.t2***) by the column of ***start*** timestamp:

```
sort -n -k 10 -t '	' my_subnetwork.t2 | awk '{print $8,$10,$11}' > my_subnetwork_sort.txt
```

Second, I subtracted the ***start*** value of the first flow from all the timestamps in ***start*** and ***end*** column so that I could plot them from ***0*** timestamp:

```
awk '{print $1,($2-1491922800.265839000),($3-1491922800.265839000)}' my_subnetwork_sort.txt > my_subnetwork_uni.txt
```

Third, according to the time slot and order, such as 0-1s, 1-2s ,…, I added the bytes of the flows that matched the conditions, which are as follows:

Suppose that the start time of the seconds is T0 and end time of the seconds is T1, the start and end timestamp of the flows are Ts and Te, respectively, then we have:

If [T0<Ts<T1 or T0<Te<T1 or (Ts<T0 and T1<T0)]:

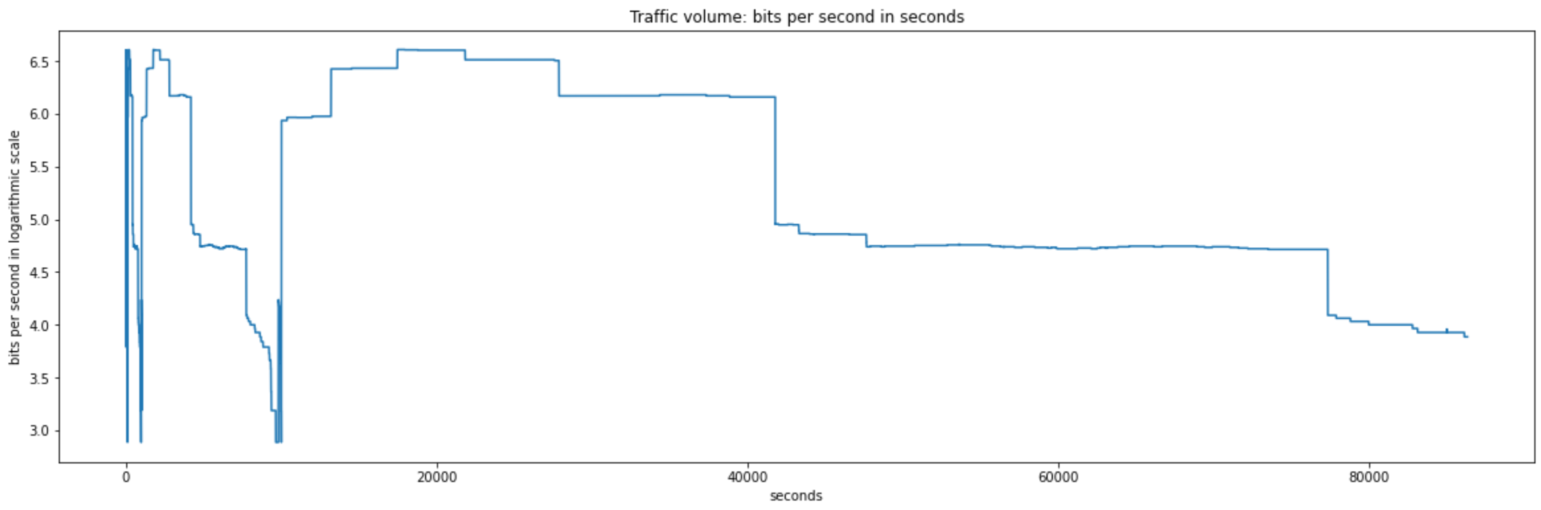
Add the bytes of the flow into the traffic volume in this seconds

According to the following algorithm, we can obtain the throughput of each seconds in the data flow:

```
#!/bin/bash

time=86340

for ((i = 0; i <= $time; i++))
do
    Bytes=0
    cat my_subnetwork_uni.txt | awk 'NR>1'| while read bytes start end
    do
        if [[ $start > $i && $start < $i+1 ]] || [[ $end > $i && $end < $i+1 ]] || [[ $start < $i && $end > $i+1 ]]
        then
            Bytes=$[Bytes+$bytes]
        fi
        if [[ $start > $i+1 ]]
        then
            echo $Bytes >> seconds.csv
            break
        fi
    done
done
```

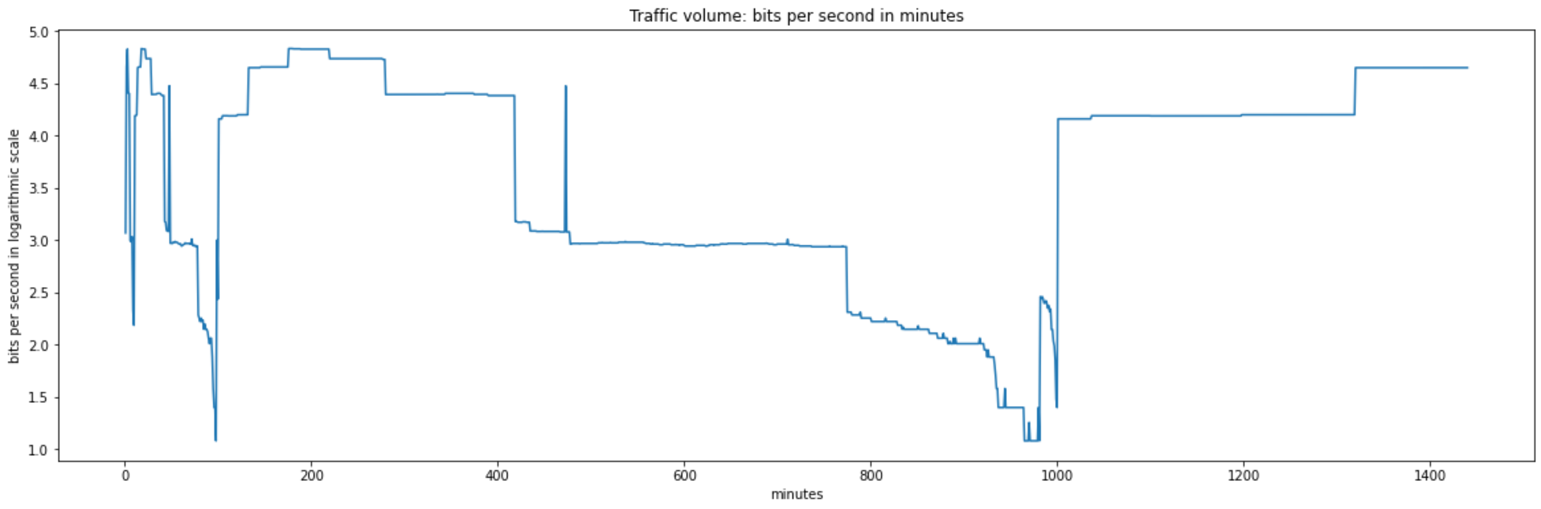
Among the code, the ***time = 86340*** came from the largest end timestamp ***86339.6*** after the subtracting process. So that there were ***86340*** seconds in total.

Then, I used Python to plot the throughput data from the ***seconds.csv*** file, and the result of bits per second in function of seconds is as below:



bits per second in function of seconds (logarithmic scale)

From the same method, we can get the plot of bits per second in function of minutes is as below:



bits per second in function of minutes (logarithmic scale)

1.2: Flows by port numbers

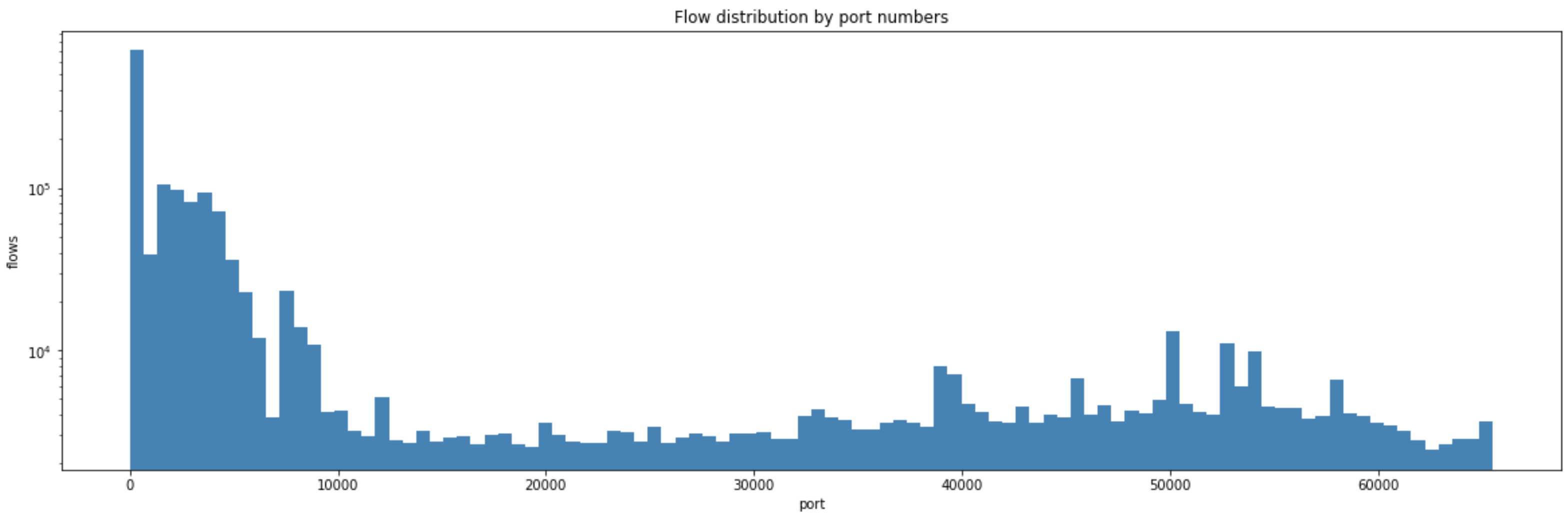
Visualise flow distribution by port numbers. If you use histograms, consider if port numbers are continuous or discrete values.

Solution:

In this part, I first used this command to extract bytes and ports data from the flows:

```
awk '{print $5","$6","$9}' ./dataprepro/my_subnetwork.t2 > ./1.2/1.2ports.csv
```

Then, I used Python to plot the histgram:



Flow distribution by port numbers

When using the ***plt.hist()*** function in Python, I set the parameter ***bins = 100*** and ***log = 1***, since the port number is discrete and the difference of the port frequency varied quite dramatically.

1.3: OD-pairs

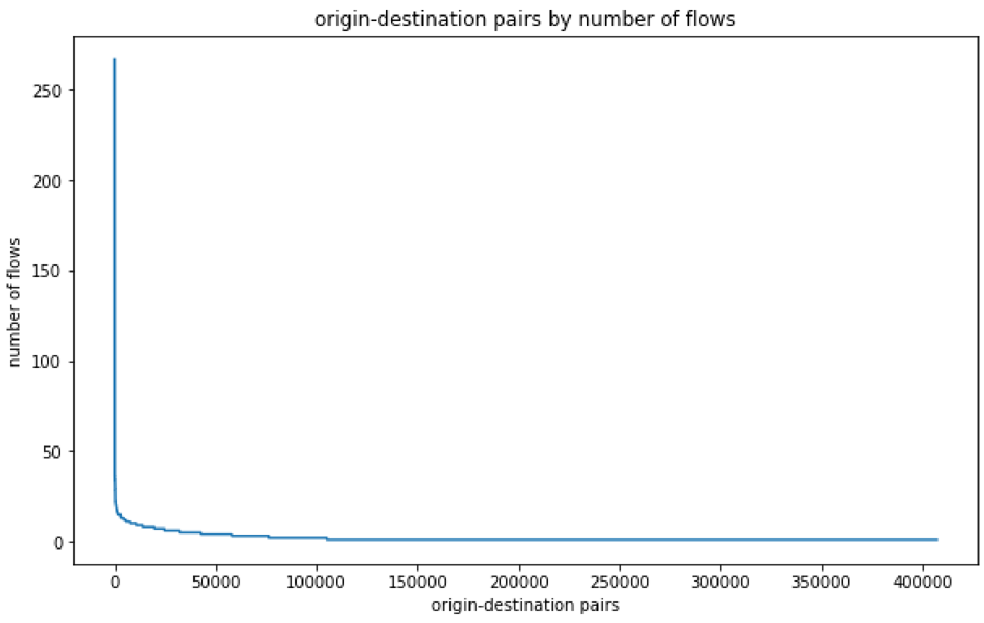
Plot origin-destination pairs by both by data volume (=bytes) and by number of flows (Zipf type plot).

Solution:

1. For number of flows, I simply used shell commands to pair and sort the OD-pairs:

```
cat ./dataprepro/my_subnetwork.t2 | awk '$1<$2{print $1,$2;next}{print $2,$1;}' | sort | un
iq -c | sort -r -n > topflows.csv
```

And then I used Python to plot the Zipf type plot:



origin-destination pairs by number of flows

2. For data volume (number of bytes), it was a little bit complex to pair the origins and destinations, extract the corresponding number of bytes and add them together at the same time. So at first, I did a pre-process to the dataset:

```
awk '{print $1,$2,$8}' ./dataprepro/my_subnetwork.t2 | sort | uniq -c | sort -r -
n | awk '{print $2,$3,$1*$4}' > bytes.txt
```

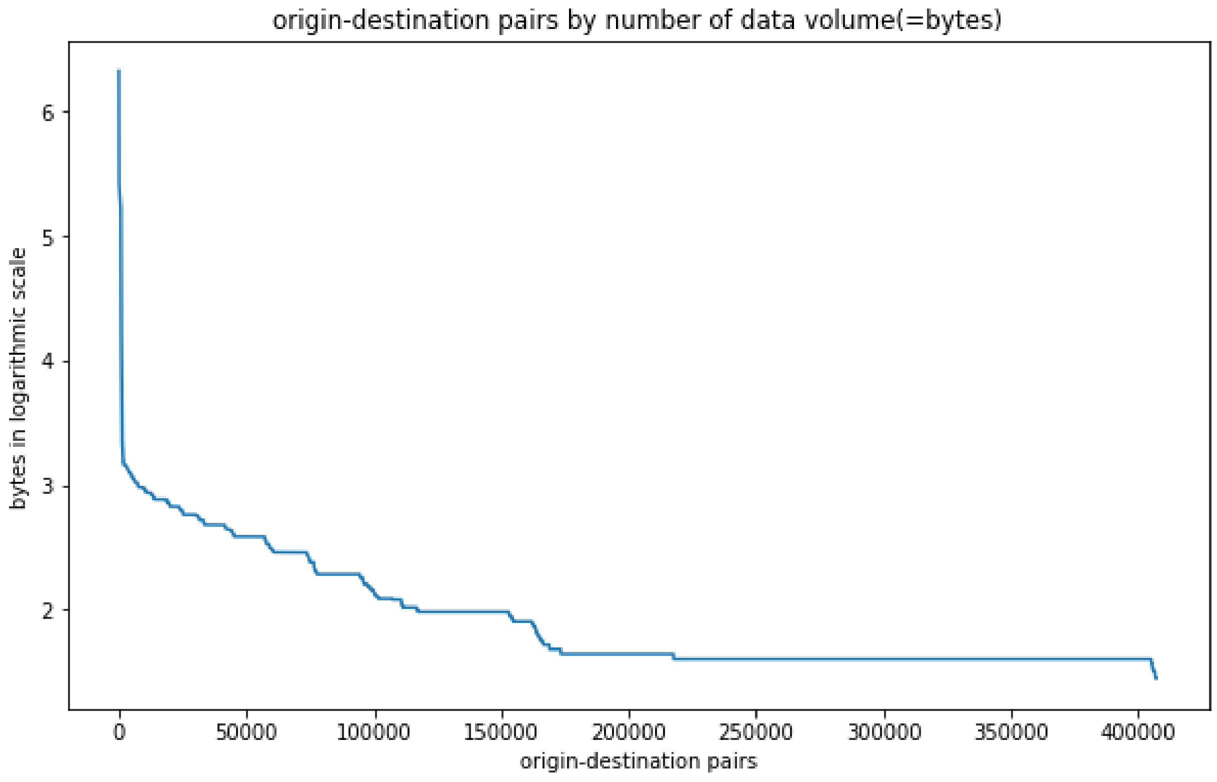
What I did here was to pair and sort the origins and destinations that has the same bytes in them, then I multiplied the frequency and bytes for each line and saved them in ***.csv*** files. But there was still a problem that those lines with the same origin and destination were not fully counted (still got OD unpaired), since they had different number of bytes. What I needed to do next was to add the bytes of same OD-pairs together and sort again. And I wrote an algorithm and did that using Python:

```
pairsbytes=[]
while (len(my_data)!=0):
  for i in my_data[1:]:
    if ((my_data[0][0]==i[0] and my_data[0][1]==i[1]) or (my_data[0][0]==i[1] and my_data[0
][1]==i[0])):
      my_data[0][2]+=i[2]
      my_data.remove(i)
  pairsbytes.append(my_data[0])
  my_data.pop(0)
with open('./1.3OD-pairs-bytes.csv', 'w', newline='') as csvfile:
    writer  = csv.writer(csvfile)
    for row in pairsbytes:
        writer.writerow(row)
```

Then I checked the number of lines of the result file ***1.3OD-pairs-bytes.csv***, and it was the same as the result of number of flows, which also suggested that the algorithm and results were correct.

```
pengh1@vdiubuntu102 ~/Desktop/FinalAssignment/Task1/1.3
 % cat 1.3OD-pairs-flows.csv | wc -l
407238
pengh1@vdiubuntu102 ~/Desktop/FinalAssignment/Task1/1.3
 % cat 1.3OD-pairs-bytes.csv | wc -l
407238
```

And the plot is as below:



origin-destination pairs by number of bytes (traffic volume)

1.4: Per user data volume

Compute the aggregate data volume for each user and draw a histogram to visualise distribution of user aggregated data. In other words, make one histogram that contains all users, no need to identify users from each other. (user would be one IP address within your assigned subnetwork)

Solution:

In this part, I used similar method with 1.3 to calculate the data volume.

```
#!bin/bash

```

```
gawk '$1~/^133\.60\.168\./' ../dataprepro/my_subnetwork.t2 | awk '{print $1,$8}' | sort | uniq -c | sort -r -n | awk '{print $2","$1*$3}' > source.csv
gawk '$2~/^133\.60\.168\./' ../dataprepro/my_subnetwork.t2 | awk '{print $2,$8}' | sort | uniq -c | sort -r -n | awk '{print $2","$1*$3}' > destination.csv
```

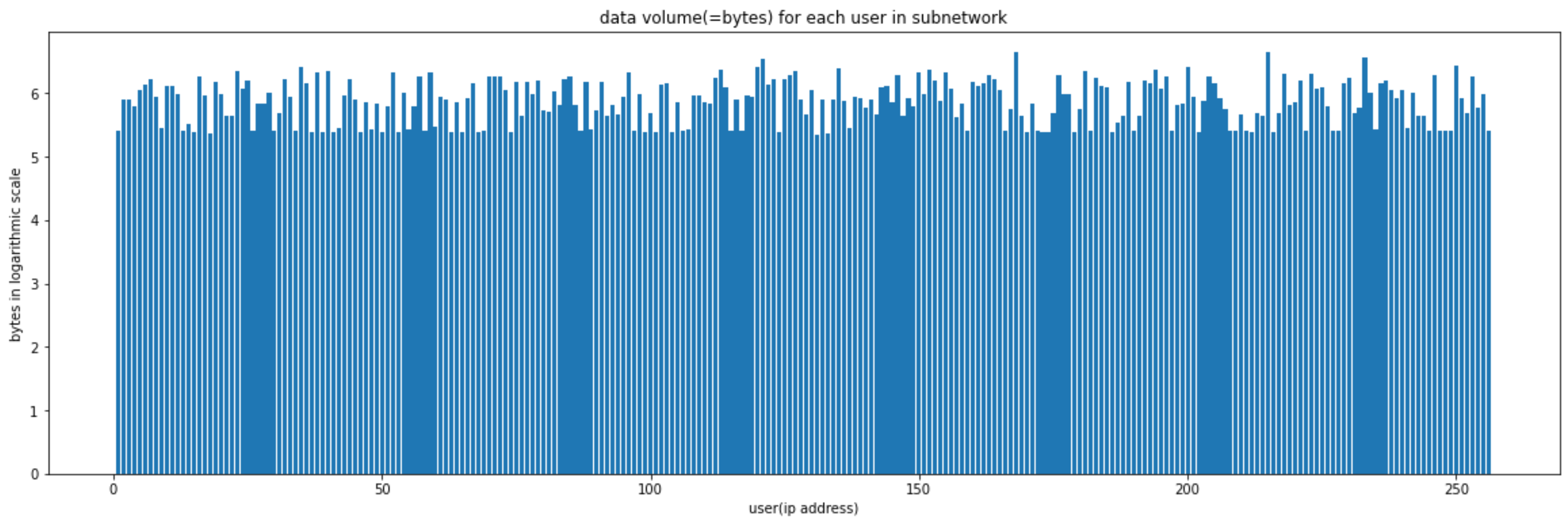
I extracted *source/bytes* and *destination/bytes* separately, then paired and sorted (by ip address and bytes) them. Next, I multiplied the frequency and bytes in each line and saved them in ***.csv*** files. And also, there was still a same problem that those lines with the same ip address were not fully counted (still got ip address unpaired), since they had different number of bytes. So I did some similar thing in Python to add bytes of same ip address together and sorted them by the ip address:

```python
userdata = srcedata + dstdata
user=[]
while (len(userdata)!=0):
  for i in userdata[1:]:
    if (userdata[0][0]==i[0]):
      userdata[0][1]+=i[1]
      userdata.remove(i)
  user.append(u[0])
  userdata.pop(0)
user.sort(key=lambda x:x[0],reverse=False)
```

Then I print the length of the ***user*** list, I got:

```python
print(len(user))
256
```

That showed the result was correct since there were at most 256 ip address in the subnetwork. And the plot is showed as below:



data volume(=bytes) for each user in subnetwork

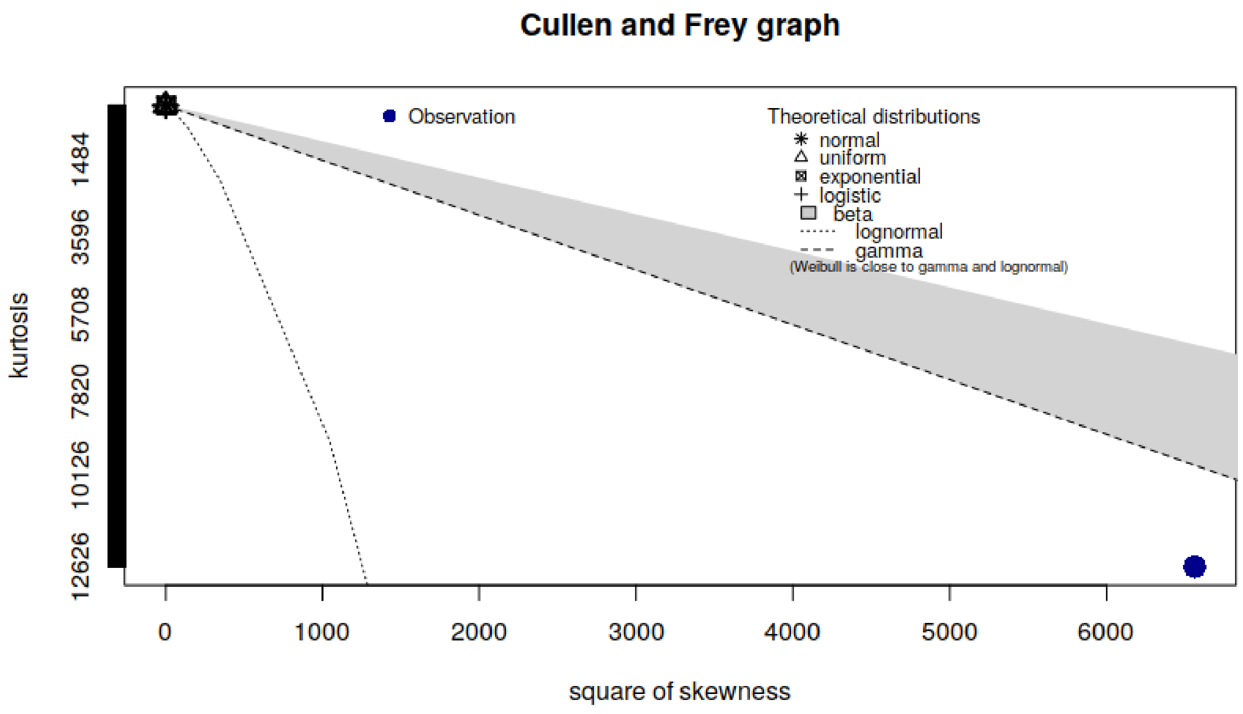
1.5: Flow length distribution

Plot flow length distribution, its empirical cumulative distribution function, and key summary statistics. Fit a suitable distribution for the flow lengths and validate it.

Solution:

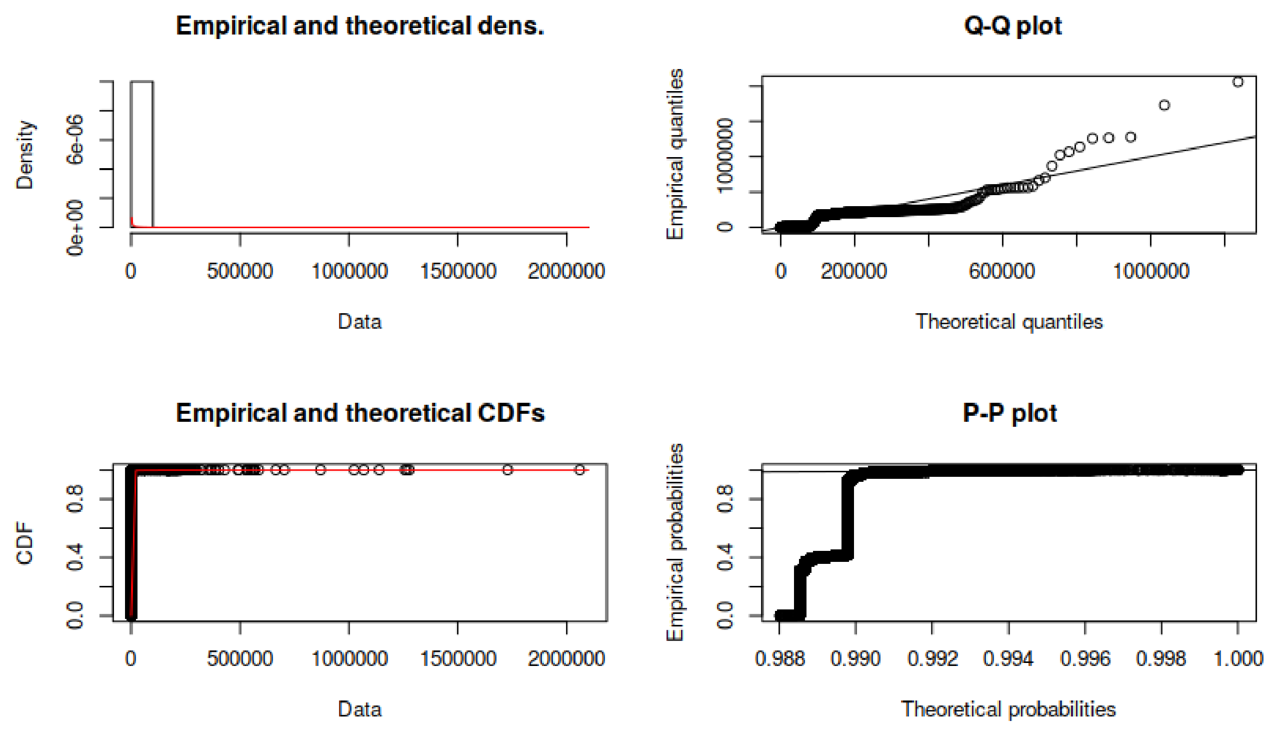
For this part, first I used the ***descdist()*** function in the ***fitdistplus*** package in R to preliminarily check the data set is closest to what distribution.

```
> descdist(my_subnetwork[,8], discrete = FALSE, boot = NULL, method = "unbiased", graph = TRUE, obs.col = "darkblue", obs.pch = 16, boot.col = "orange")
summary statistics
------
min:  28   max:  2060067
median:  96
mean:  303.5228
estimated sd:  7994.316
estimated skewness:  81.01046
estimated kurtosis:  12625.39
```



As we can see, the Observation point was most closed to the line for gamma distribution. So basically we could say the data set follows the gamma distribution. And the validation was done below:

```
> fit.gamma<-fitdist(my_subnetwork[,1],"gamma","mme")
> plot(fit.gamma)
```



As we can see, the dataset fits the gamma distribution generally well. The key summary statistics and the estimated parameters of the distribution are as follows:

```
summary statistics
------
min:  28   max:  2060067
median:  96
mean:  303.5228
estimated sd:  7994.316
estimated skewness:  81.01046
estimated kurtosis:  12625.39
```

```
> fitdist(my_subnetwork[,8], "gamma","mme")
Fitting of the distribution ' gamma ' by matching moments
Parameters:
          estimate
shape 1.441519e-03
rate  4.749295e-06
```

1.6: Flow sampling

For this task, use FS1 and take ALL flow data into account (i.e., not limiting the scope solely on your subnetwork).

Make two random selections from all flows by sampling flows from the 24h flow data: first selection to only include IPv4 traffic and the other only IPv6. Define your sampling process such that you will get about the same number of flows for this all flow data as in your assigned subnetwork. Document your selection process.

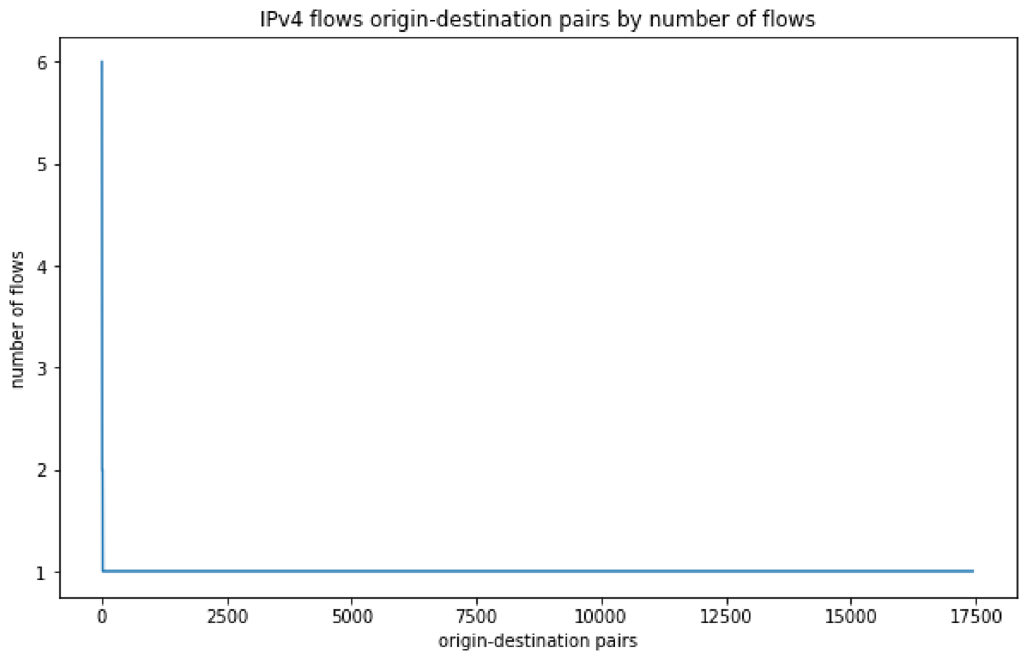
Select one of previous tasks (1.1-1.5) and perform same analysis for the both sampled data sets you just collected. Compare the results to the original task where you used your subnetwork (FS2) only. Can you say characteristics of your subnetwork is representative? Is there difference between IPv4 and IPv6?

Solution:

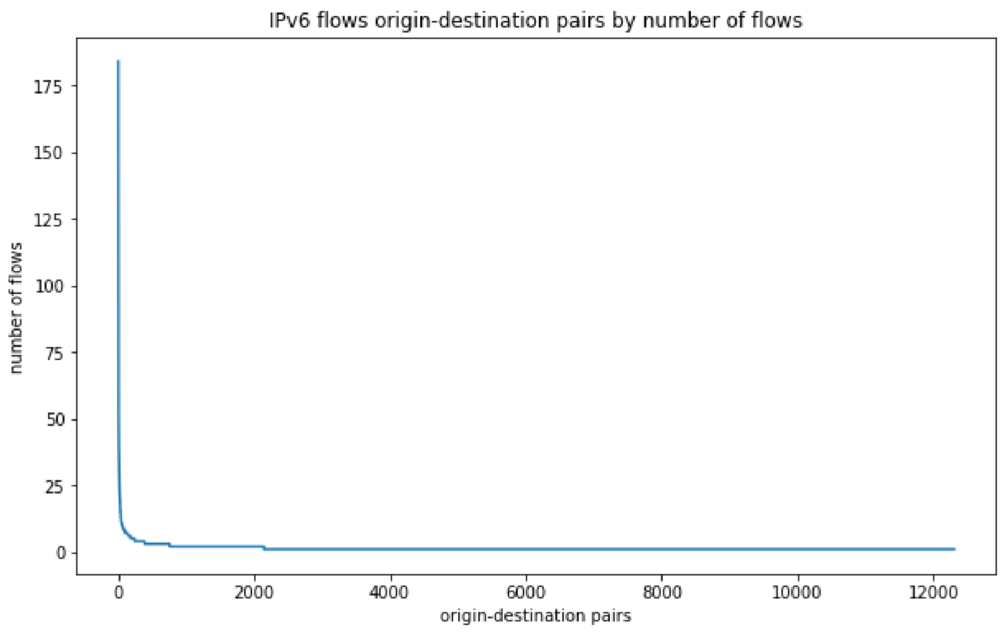
In this part, I first used regular expression to match IPv4 and IPv6 flows. Then I did a sampling process to extract appropriate amount of flows from the whole large dataset:

```bash
#!bin/bash

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

gawk '$1~/^.*\..*\..*\..*/||$2~/^.*\..*\..*\..*/' $TRACE/flow-continue/*.t2 | awk 'BEGIN{srand()}rand()<0.00047' > /u/88/pengh1/unix/Desktop/FinalAssignment/Task1/1.6/ipv4.t2

gawk '$1~/^.*:.*:.*:.*:.*:.*:.*:.*/||$2~/^.*:.*:.*:.*:.*:.*:.*:.*/' $TRACE/flow-continue/*.t2 | awk 'BEGIN{srand()}rand()<0.047' > /u/88/pengh1/unix/Desktop/FinalAssignment/Task1/1.6/ipv6.t2
```

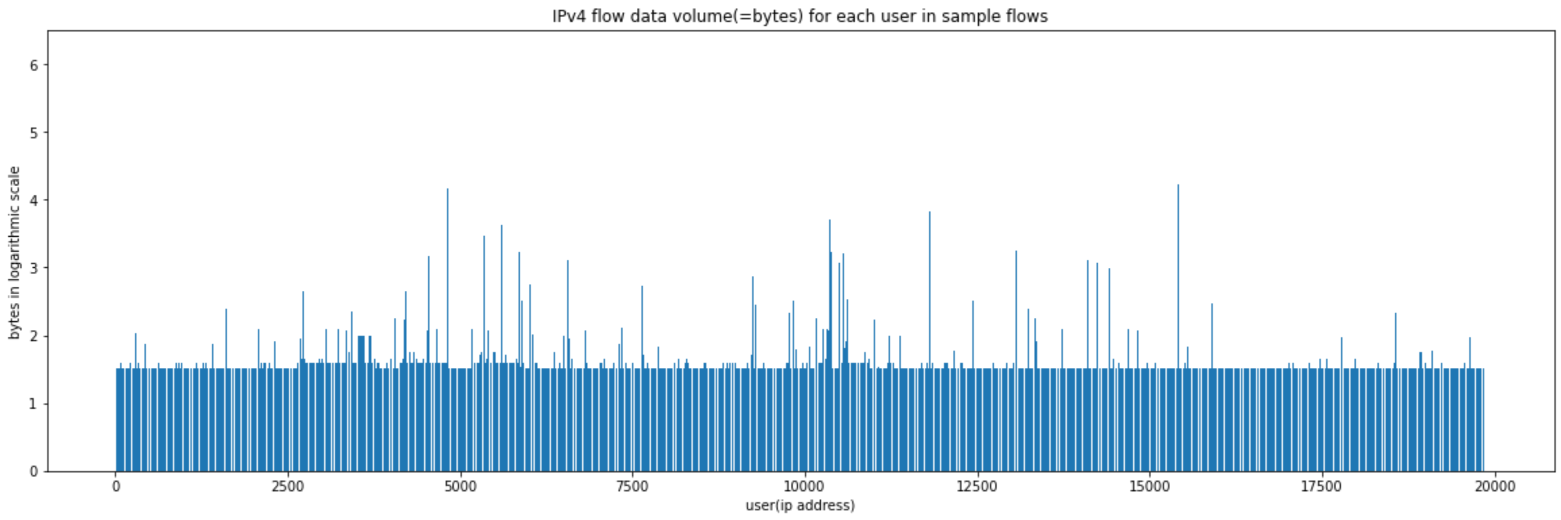
Next, I did same process as in the 1.4 for IPv4 and IPv6 data respectively, and part of 1.3 as well. And the results are as follows:



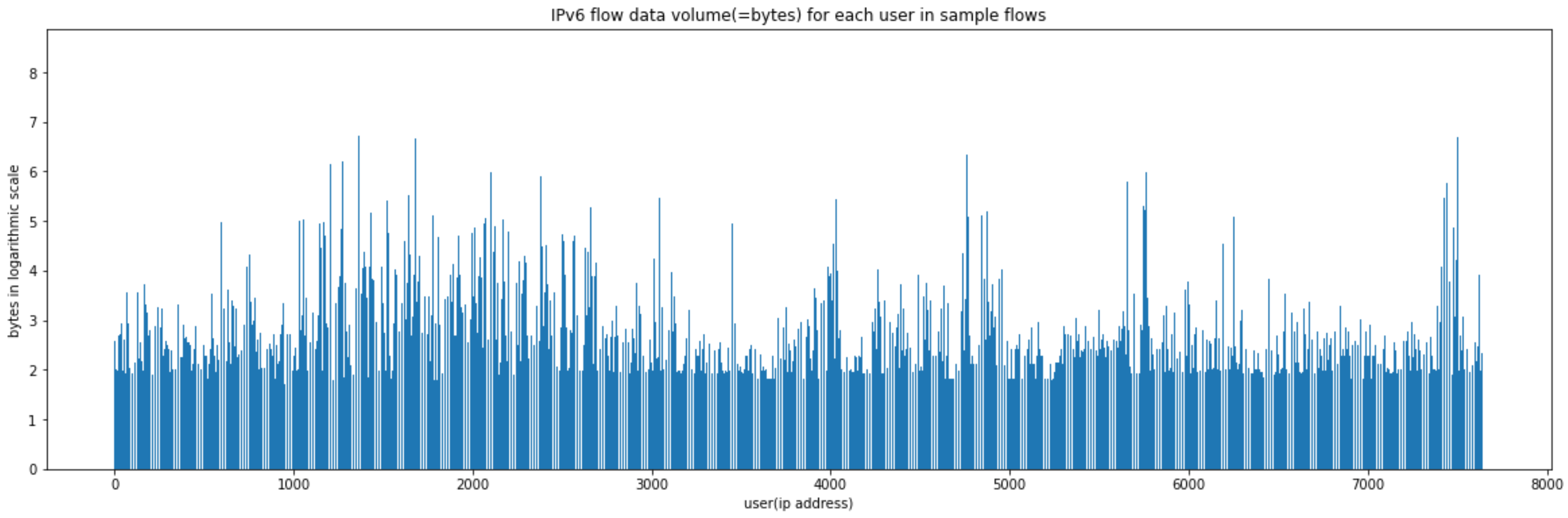
IPv4 flows origin-destination pairs by number of flows



IPv6 flows origin-destination pairs by number of flows



IPv4 flow data volume(=bytes) for each user in sample flows



IPv6 flow data volume(=bytes) for each user in sample flows

Based on the plots, it is possible to say that the characteristics of my subnetwork is somehow representative, since the plot of volume for each user and OD-pairs are similar to the plot of IPv4 data. We can see that in both plot the data volume for each user stay relatively stable around a certain value, while there are small fluctuations and small amount of outliers in the data, which is normal for data traffic. But as for IPv6 data, the fluctuations in the plot are relatively larger than in IPv4 data. So the characteristics of IPv4 and IPv6 are not the same.

1.7: Conclusions

Based on results above, explain your conclusions on data for:

1. Traffic volume at different time scales. Are there any recognizable patterns?

As we can see in the plots, the traffic volume in seconds time scale fluctuates dramatically in 0-10000 seconds, then it increases in 10000-20000 seconds. After 20000 seconds, it decreased somehow less dramatically until the end.

In minutes time scale, the data volume has the same pattern as in seconds time scale in 0-100 minutes, where it fluctuates dramatically, then from 100-200 minutes it increases. After 200 minutes it decrease until around 980 minutes, during which a tiny upgoing fluctuation occurs. However, not like in seconds time scale, it increases again from 980 minutes afterwards until the end.

Basically, there are big fluctuations in data volume in both time scales, and the differences are quite apparent to observe.

2. What are the 5 most common applications (study the port numbers)?

After sorting the ports by frequency, the result is as follows:

```
mostcommon = sorted(portdict.items(),key=lambda x:x[1],reverse=True)
print(mostcommon[:5])
[(445, 473676), (23, 148454), (22, 29529), (7547, 19341), (1433, 17236)]
```

So the 5 most common port numbers are: 445, 23, 22, 7547 and 1433, which represent respectively:

port 445: Common Internet File System (CIFS)

port 23: Telnet

port 22: Secure Shell (SSH)

port 7547: CPE WAN Management Protocol (CWMP)

port 1433: default port for SQL Server

3. What kind of users there are in the network? Speculate on what kind of network this network could be based on traffic volumes and user profiles. Is your subnetwork different from larger population?

Of course, there are both IPv4 and IPv6 users in the network, while the number of IPv4 users are far more than IPv6 users. And based on traffic volumes and user profiles, it is speculated that the network is used to transfer and store files remotely between different computers.

As far as I am concerned, my subnetwork is somehow similar to the larger population, since the patterns in user profile is somehow similar with each other. And more investigations are needed to see the specific differences between them.

Task 2: Capturing packets

Acquiring packet capture data

As is introduced in the previous assignment handout, I acquired the packet capture data on my own computer using ***dumpcap*** tool, the commands as follws:

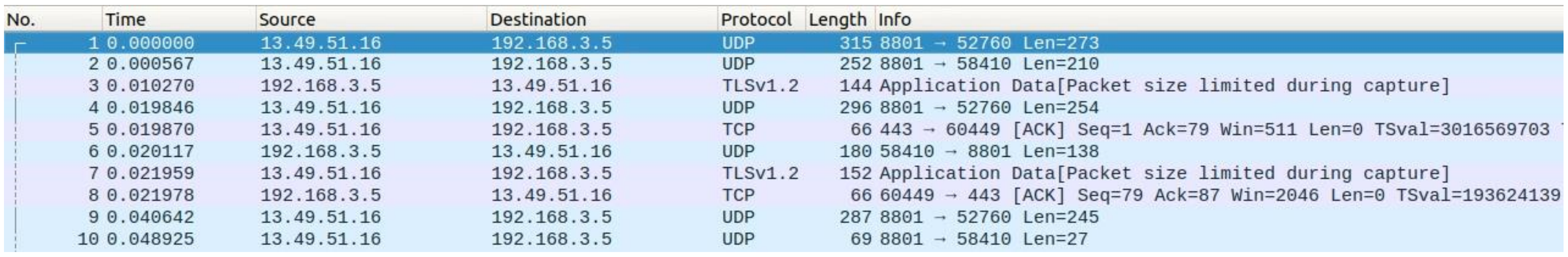
```
dumpcap -D
1. enp0s5
2. any  v
3. lo (Loopback)
4. nflog
5. nfqueue
6. usbmon1
7. usbmon2
8. usbmon3
9. usbmon4
```

I chose ***en0s5*** to carry out the packet capture, and the measurement setting are as follows:

First packet	2020-11-13 11:00:03
Last packet	2020-11-13 15:00:01
Elapsed	03:59:57
Packets	5552719

```
sudo dumpcap -a duration:14400 -i enp0s5 -s 96 -g -P -w data.pcap
Capturing on 'enp0s5'
File: data.pcap
Packets captured: 5552719
```

And the sample of the data from the .pcap file is as follows:



```
No.  Time     Source        Destination   Protocol Length Info
   1 0.000000 13.49.51.16   192.168.3.5   UDP        315 8801 → 52760 Len=273
   2 0.000567 13.49.51.16   192.168.3.5   UDP        252 8801 → 58410 Len=210
   3 0.010270 192.168.3.5   13.49.51.16   TLSv1.2    144 Application Data[Packet size limited during capture]
   4 0.019846 13.49.51.16   192.168.3.5   UDP        296 8801 → 52760 Len=254
   5 0.019870 13.49.51.16   192.168.3.5   TCP         66 443 → 60449 [ACK] Seq=1 Ack=79 Win=511 Len=0 TSval=3016569703
   6 0.020117 192.168.3.5   13.49.51.16   UDP        180 58410 → 8801 Len=138
   7 0.021959 13.49.51.16   192.168.3.5   TLSv1.2    152 Application Data[Packet size limited during capture]
   8 0.021978 192.168.3.5   13.49.51.16   TCP         66 60449 → 443 [ACK] Seq=79 Ack=87 Win=2046 Len=0 TSval=193624139
   9 0.040642 13.49.51.16   192.168.3.5   UDP        287 8801 → 52760 Len=245
  10 0.048925 13.49.51.16   192.168.3.5   UDP         69 8801 → 58410 Len=27
```

Data pre-processing

Cleaning the data packets (PS1)

1. For task 2.1, I first converted the data.pcap file into packet.t2 file using ***crl_flow*** command:

```
crl_flow -Ci=14400 -Tf60 data.pcap > packet.t2
```

Then, I extracted the lines that only contained flow data using regular expressions:

```
gawk '$1~/^.*\..*\..*\..*/||$1~/^.*:.*:.*/' data0.t2 > /u/88/pengh1/unix/Desktop/FinalAssignment/Task2/PS1/packet.t2
```

Next, I extracted the columns that contains source port, destination port and packets:

```
awk '{print $5","$6","$7}' ./packet.t2 > ./packetports.csv
```

And finally the ***packetports.csv*** was what I used to finish the task.

2. For task 2.2, I extracted the lines that contained source, destination and bytes:

```
sort -n -k 10 -t '   ' packet.t2 | awk '{print $8,$10,$11}' > packet_sort.txt
```

And basically the following steps were based on the ***packet_sort.txt*** file.

3. For task 2.3, I just did the task using R so I did not need to extract extra data from the ***packet.t2*** file.

Here I just list some samples of ***packet.t2*** file, which also could represent other data files:

src	dst	pro	ok	sport	dport	pkts	bytes	flows	first	latest
192.168.3.1	192.168.3.5	17	1	53	15482	1	133	1	1605267032.199536000	1605267032.199536000
192.168.3.1	192.168.3.5	17	1	53	23441	1	512	1	1605269467.260965000	1605269467.260965000
192.168.3.5	216.58.211.3	17	1	55171	443	4	2048	1	1605258767.319682000	1605258767.390160000
192.168.3.59	216.58.211.10	6	1	58584	443	18	2751	1	1605270772.699924000	1605271012.796204000
192.168.3.1	192.168.3.5	17	1	53	46847	1	149	1	1605267321.033430000	1605267321.033430000
192.168.3.5	224.0.0.251	17	1	5353	5353	2	660	1	1605271987.694980000	1605271987.797308000
192.168.3.1	192.168.3.5	17	1	53	23653	1	155	1	1605258983.643622000	1605258983.643622000
192.168.3.1	192.168.3.5	17	1	53	23759	1	196	1	1605265908.002572000	1605265908.002572000
192.168.3.1	192.168.3.5	17	1	53	31718	1	348	1	1605270994.976234000	1605270994.976234000
192.168.3.59	202.77.129.232	6	1	46884	443	11	1057	1	1605261620.449549000	1605261684.797647000
…	…	…	…	…	…	…	…	…	…	…

Converting packet trace to flow data (PS2)

For PS2 data, I first converted the data.pcap file into data.t2 file using ***crl_flow*** command:

```
crl_flow -Ci=14400 -cl -Tf60 -Cai=1 data.pcap > data0.t2
```

Then, I extracted the lines that only contained flow data using regular expressions:

```
gawk '$1~/^.*\..*\..*\..*/||$1~/^.*:.*:.*/' data0.t2 > /u/88/pengh1/unix/Desktop/FinalAssig
nment/Task2/PS2/data.t2
```

And finally the ***data.t2*** was what I used to finish the task.

The sample data are as follows:

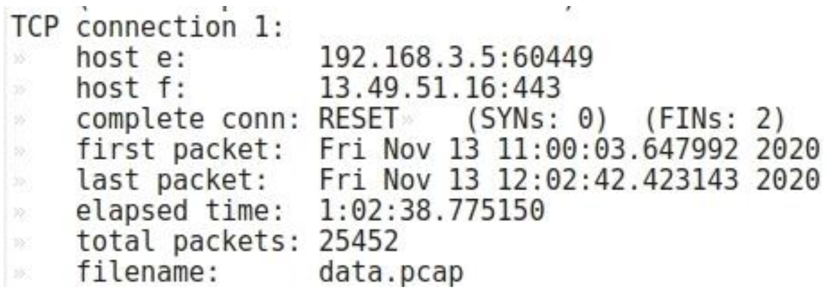
src	dst	pro	ok	sport	dport	pkts	bytes	flows	first	latest
172.217.22.174	192.168.3.5	17	1	443	62353	5	2131	1	1605259779.238599000	1605259780.057167000
192.168.3.59	192.168.3.1	17	1	51612	53	2	118	1	1605258286.517044000	1605258286.517071000
192.168.3.5	192.168.3.1	17	1	19266	53	1	64	1	1605259532.297663000	1605259532.297663000
17.253.5.202	192.168.3.5	6	1	80	61589	5	3515	1	1605261960.923585000	1605261961.283000000
162.62.97.64	192.168.3.5	6	1	80	61634	5	750	1	1605262860.742617000	1605262861.030427000
172.217.22.174	192.168.3.5	17	1	443	57153	7	2761	1	1605262533.273178000	1605262533.540711000
192.168.3.5	192.168.3.1	17	1	19242	53	1	71	1	1605260523.141455000	1605260523.141455000
192.168.3.1	192.168.3.5	17	1	53	28885	1	356	1	1605258542.139190000	1605258542.139190000
192.168.3.5	192.168.3.1	17	1	19223	53	1	75	1	1605259283.717666000	1605259283.717666000
192.168.3.1	192.168.3.5	17	1	53	5114	1	325	1	1605268585.917182000	1605268585.917182000
…	…	…	…	…	…	…	…	…	…	…

TCP connection statistics (PS3)

For tasks in PS3, I used ***tcptrace*** command to TCP connection statistics (PS3):

```
tcptrace -l -r -D -n -u  data.pcap > ./PS3/data.txt
```

The sample summary data of first connection are as follows:



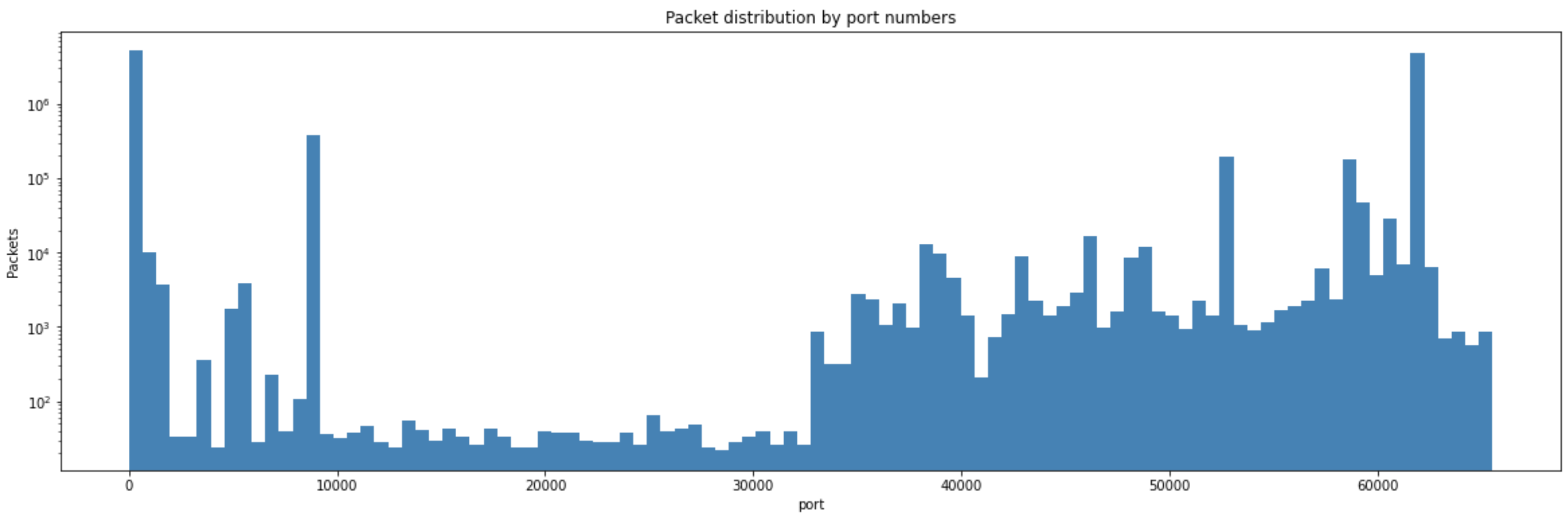
```
TCP connection 1:
	host e:        192.168.3.5:60449
	host f:        13.49.51.16:443
	complete conn: RESET	(SYNs: 0)  (FINs: 2)
	first packet:  Fri Nov 13 11:00:03.647992 2020
	last packet:   Fri Nov 13 12:02:42.423143 2020
	elapsed time:  1:02:38.775150
	total packets: 25452
	filename:      data.pcap
```

Data analysis

Packet data PS1

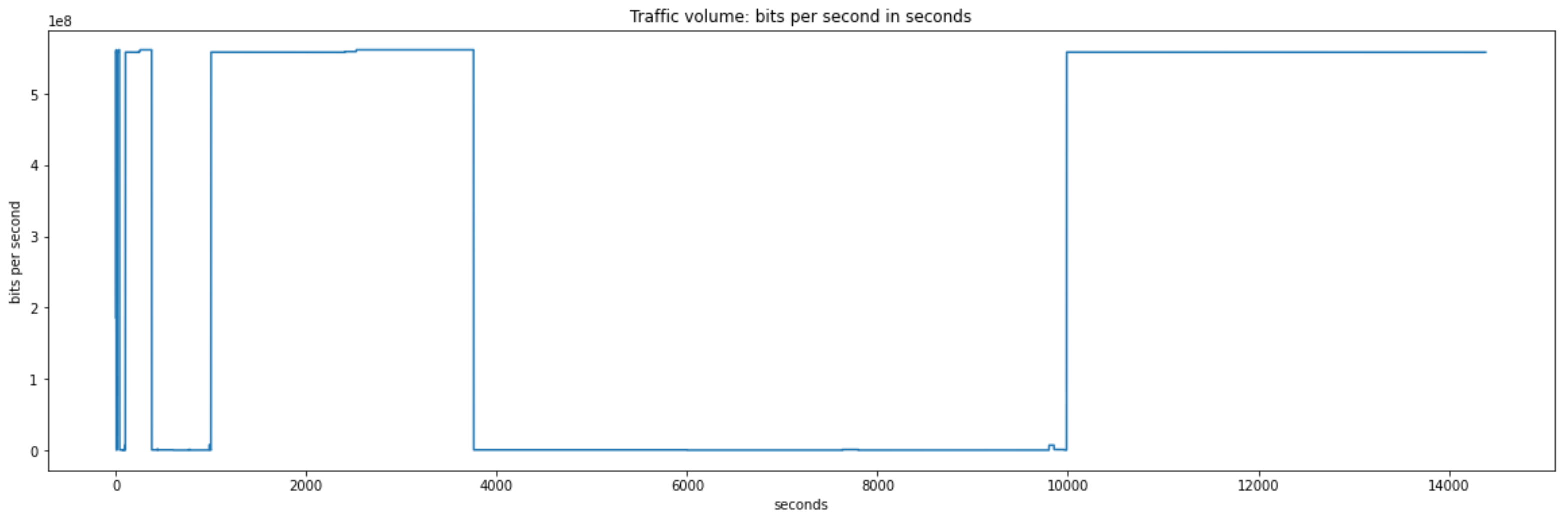
2.1: Visualize packet distribution by port numbers.

Based on the data pre-processed, I used Python to plot the figure. The codes used is basically similar to that in task 1.2. And the result is at below:

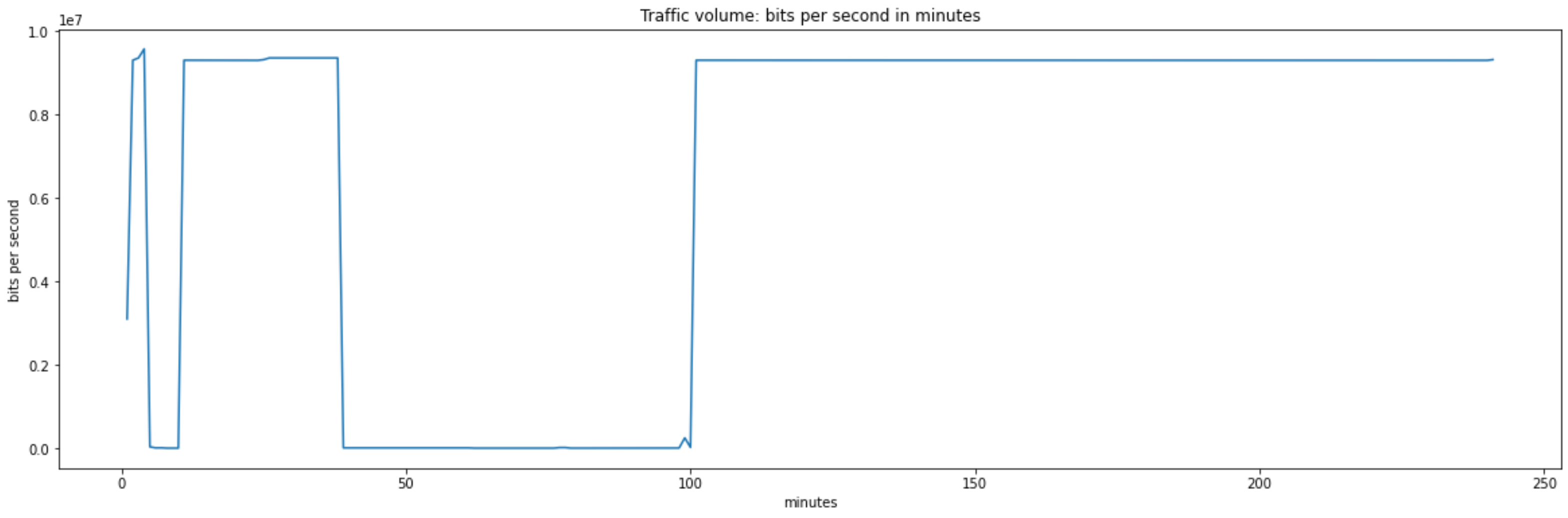


2.2: Plot traffic volume as a function of time with at least two sufficiently different time scales.

Basically, the method used in this task was the same as that in task 1.1. So the results are as follows:



bits per second in function of seconds



bits per second in function of minutes

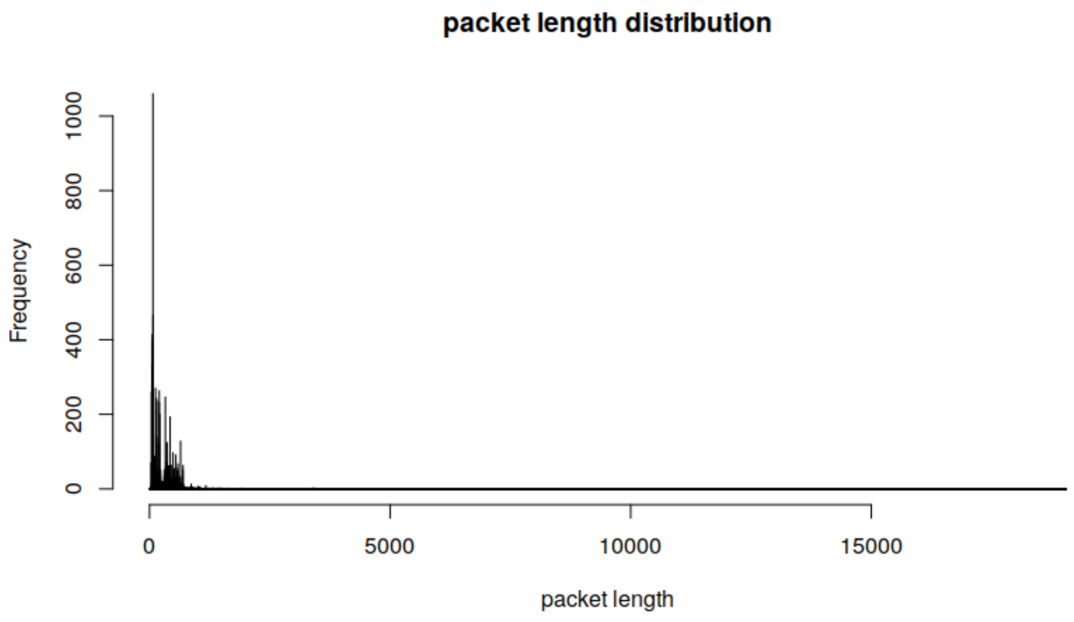
2.3: Plot packet length distribution (use bins of width 1 byte), its empirical cumulative distribution function and key summary statistics.

I did this task in R. I assumed the packet length was obtained by:

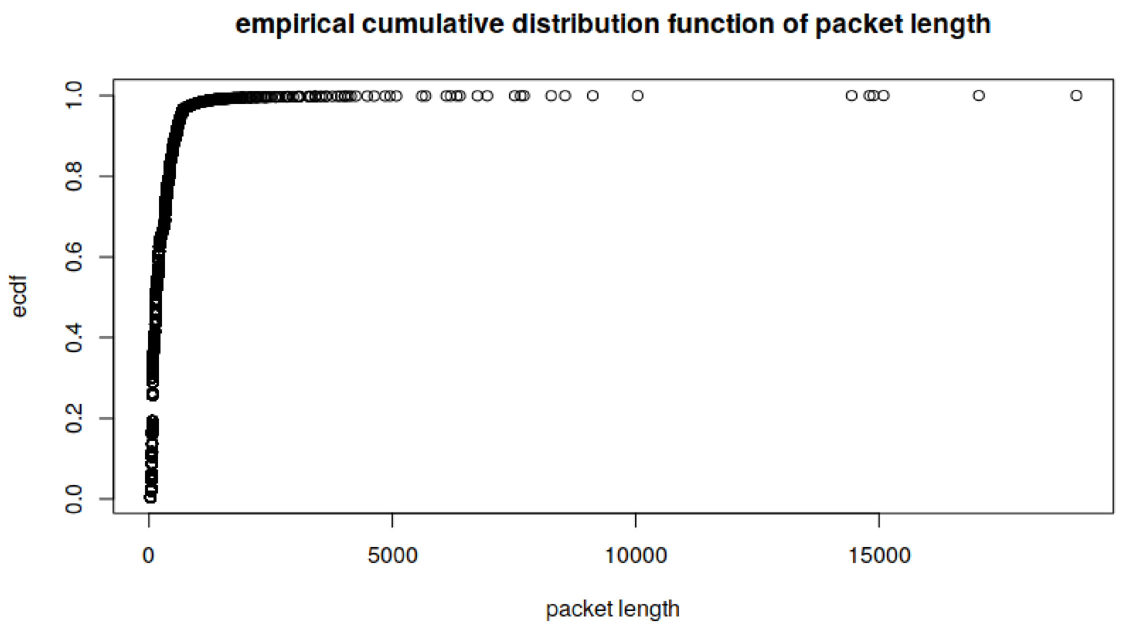
$$packet\ length = {}^{bytes}/_{packets}$$

Therefore, the commands I used are as follows:

```
> hist(packet[,8]/packet[,7], breaks = 16416, main = "packet length distribution", xlab = "
packet length")
```



```
> x<-packet[,8]/packet[,7]
> y<-ecdf(x)
> plot(x,y(x),main = "empirical cumulative distribution function of packet length", xlab =
  "packet length", ylab = "ecdf")
```



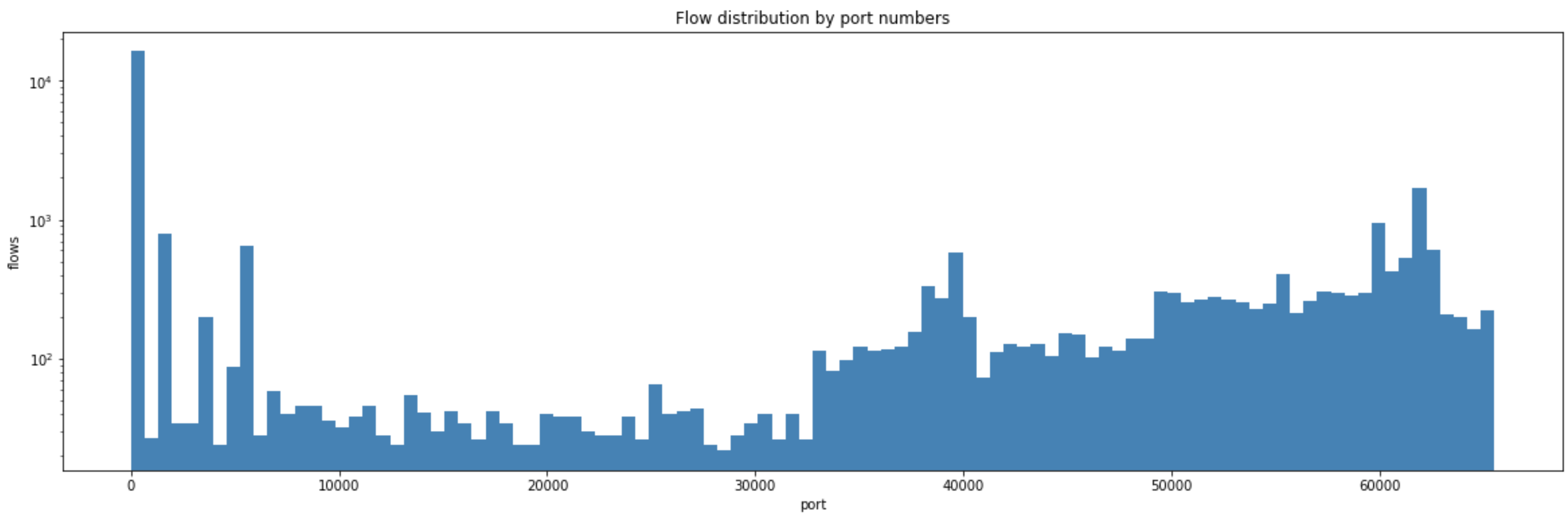
The key summary data are as follows:

```
> summary(packet[,8]/packet[,7])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   32.0    72.0   149.0   264.1   358.5 19052.0
```

Flow data PS2

2.4: Visualise flow distribution by port.

Based on the data pre-processed, I used Python to plot the figure. The codes used is basically similar to that in task 1.2 abd 2.1. And the result is at below:



2.5: Visualise flow distribution by country.

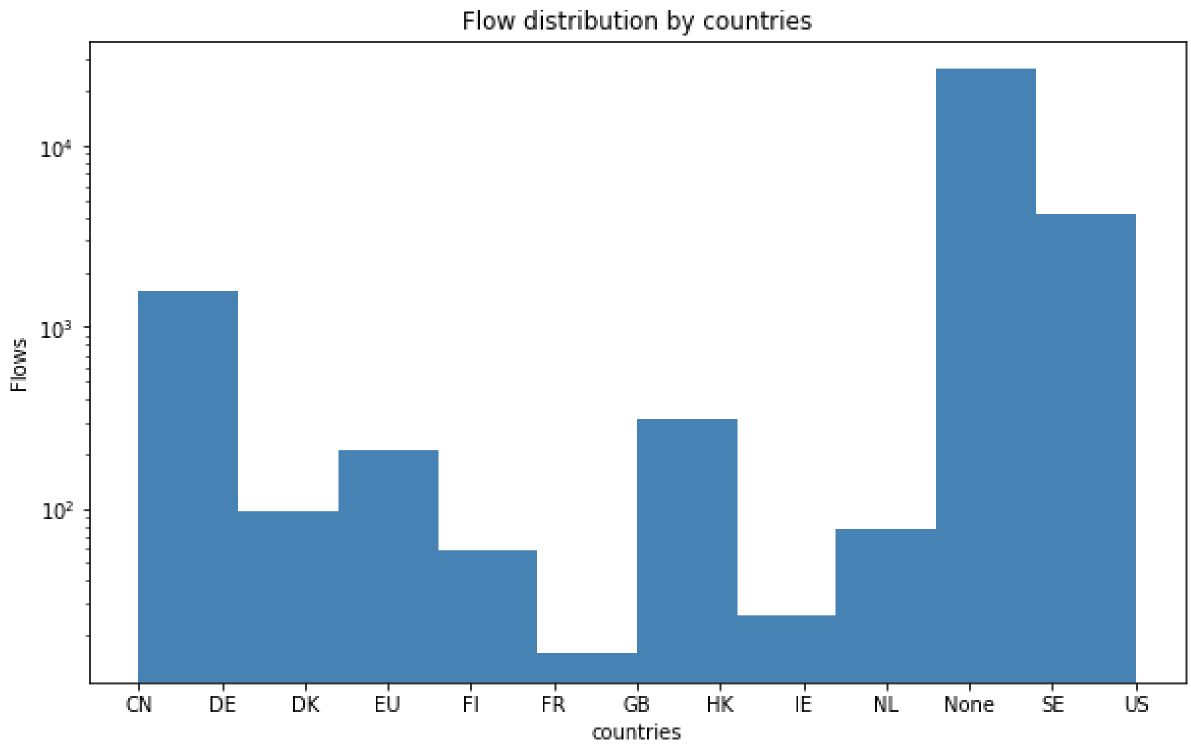
I did this in shell and Python. First, I extracted the source and destination ip addresses into a ***.csv*** file in one column:

```
awk '{print $1}' data.t2 >> ip.csv
awk '{print $2}' data.t2 >> ip.csv
```

Then I used Python script to convert the ip into countries and plot the distribution:

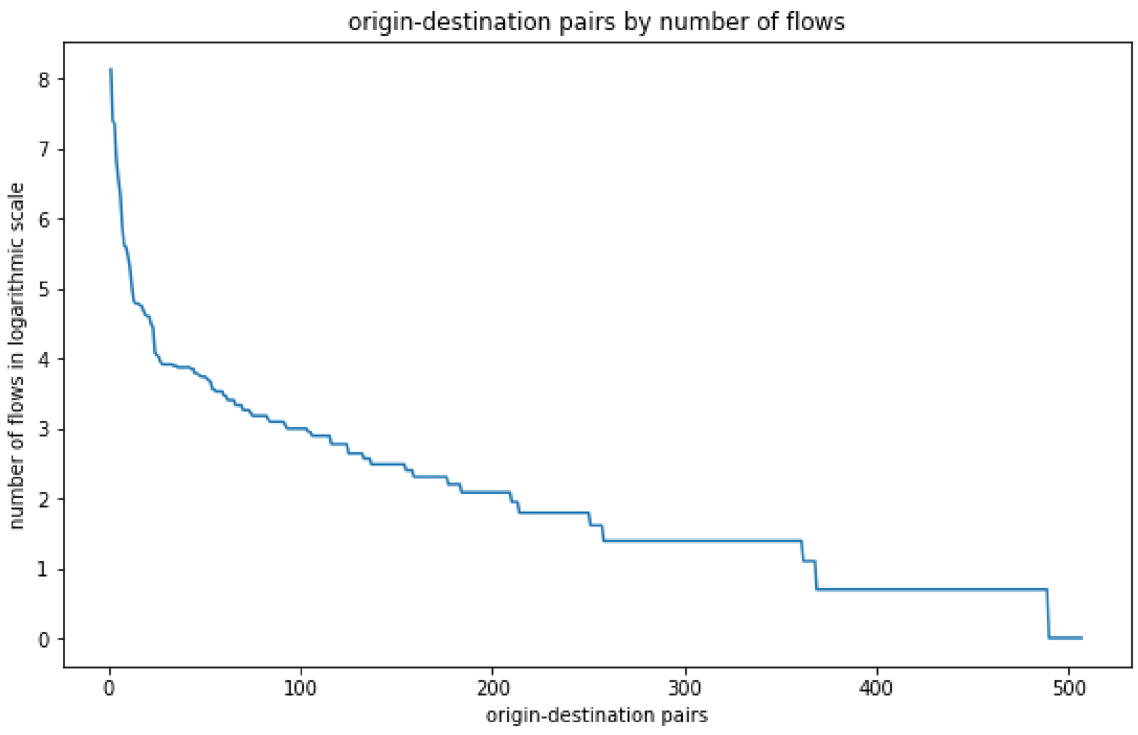
```
import GeoIP
import csv
from matplotlib import pyplot as plt

gi = GeoIP.new(GeoIP.GEOIP_MEMORY_CACHE)

ip=[]
with open('./ip.csv', 'r') as f:
    reader = csv.reader(f)
    for i in reader:
        ip.append(i[0])

countries = []
for n in ip:
  countries.append(gi.country_code_by_addr(n))

plt.gcf().set_size_inches(10, 6)
plt.title("Flow distribution by countries")
plt.xlabel("countries")
plt.ylabel("Flows")
plt.hist(x = countries, log = 1, color = 'steelblue')
```

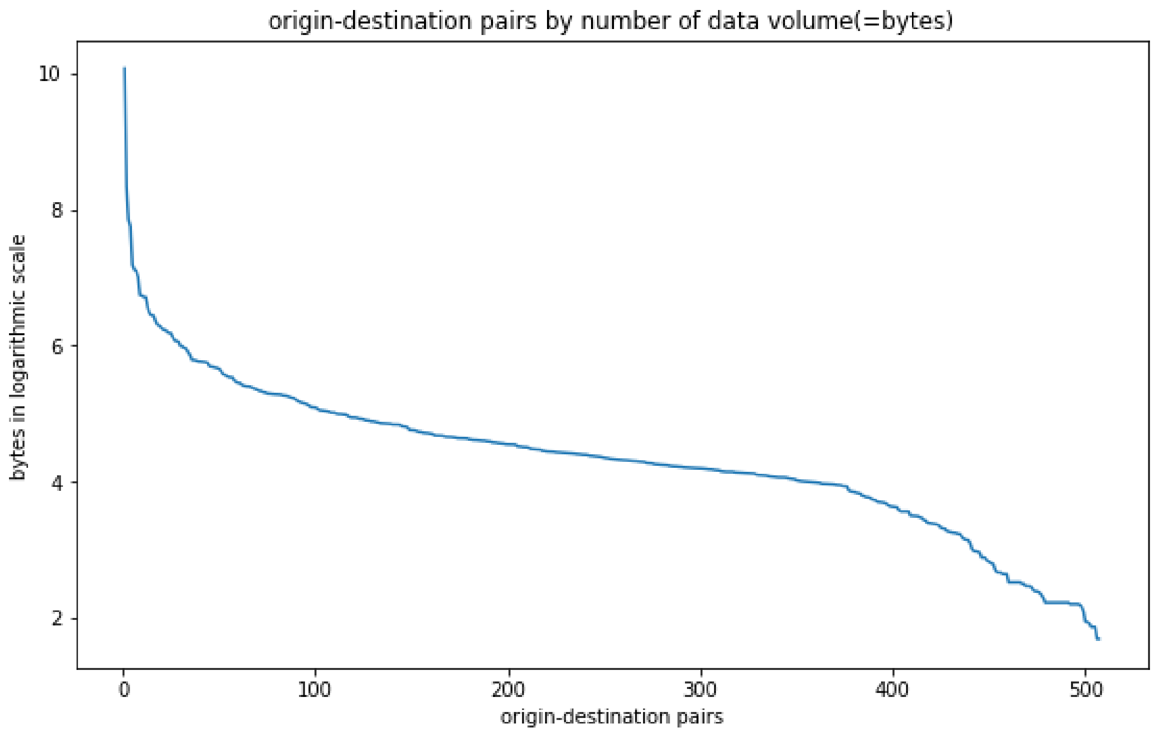
And the result is as follows:



2.6: Plot origin-destination pairs by both by data volume and by flows (Zipf type plot).

Basically, the method used in this task was the same as that in task 1.3. So the results are as follows:

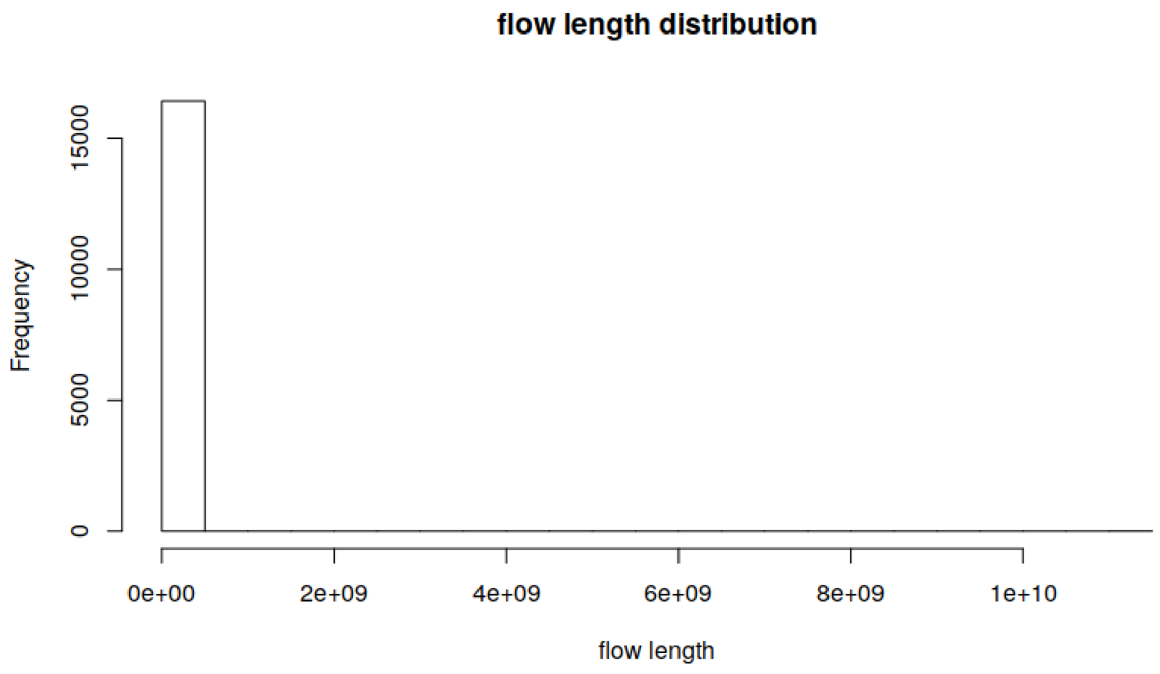




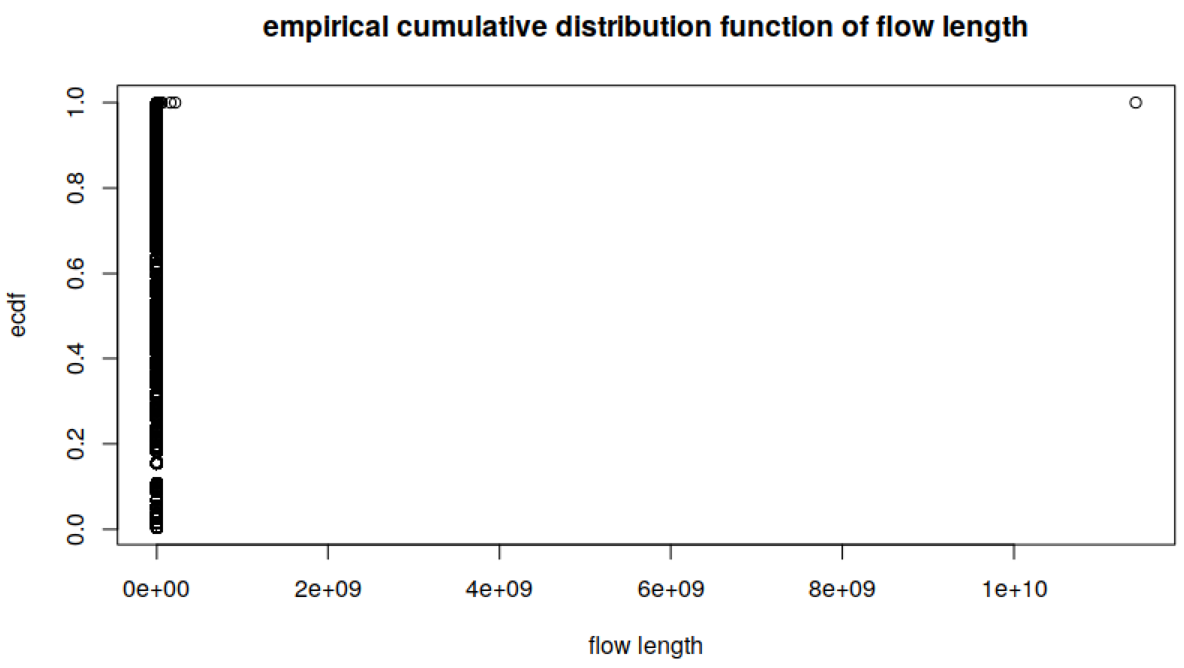
2.7: Plot flow length distribution, its empirical cumulative distribution function and key summary statistics.

I did this task in R. And the commands and results are as follows:

```
> hist(data[,8], breaks = 20, main = "flow length distribution", xlab = "flow length")
```



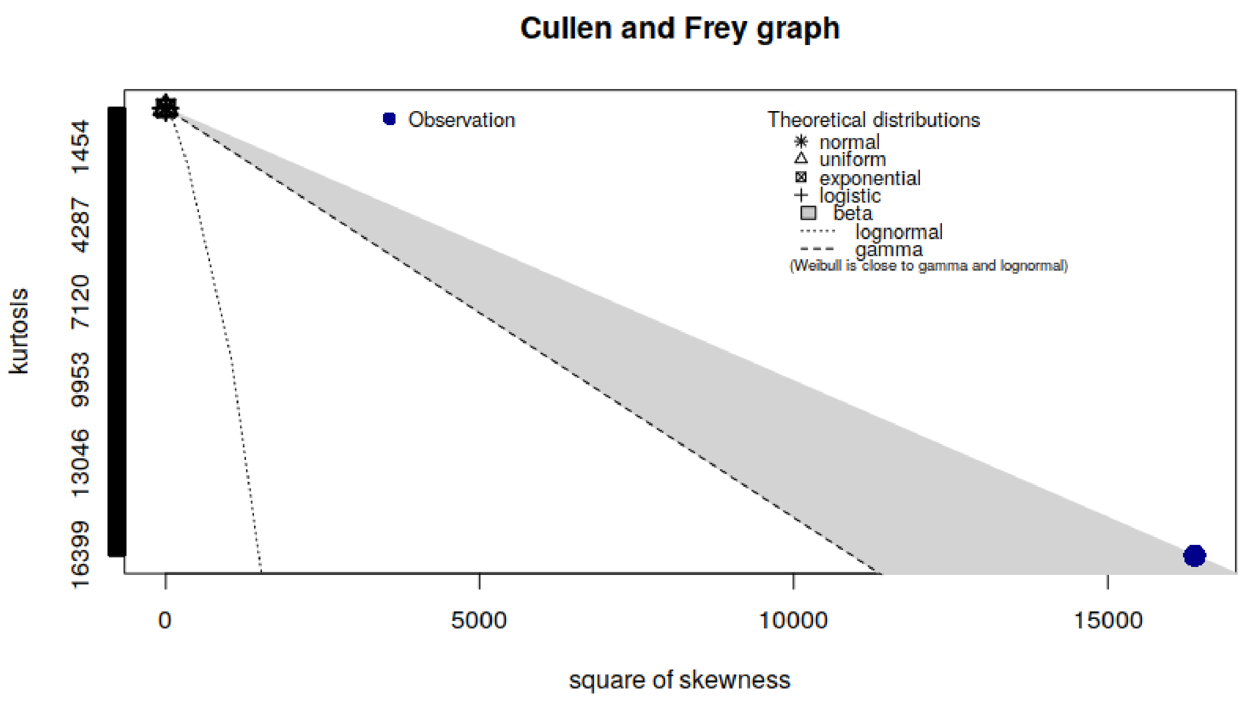
```
> x<-data[,8]
> y<-ecdf(x)
> plot(x,y(x),main = "empirical cumulative distribution function of packet length", xlab = "packet length", ylab = "ecdf")
```



2.8: Fit a distribution for the flow lengths and validate the model.

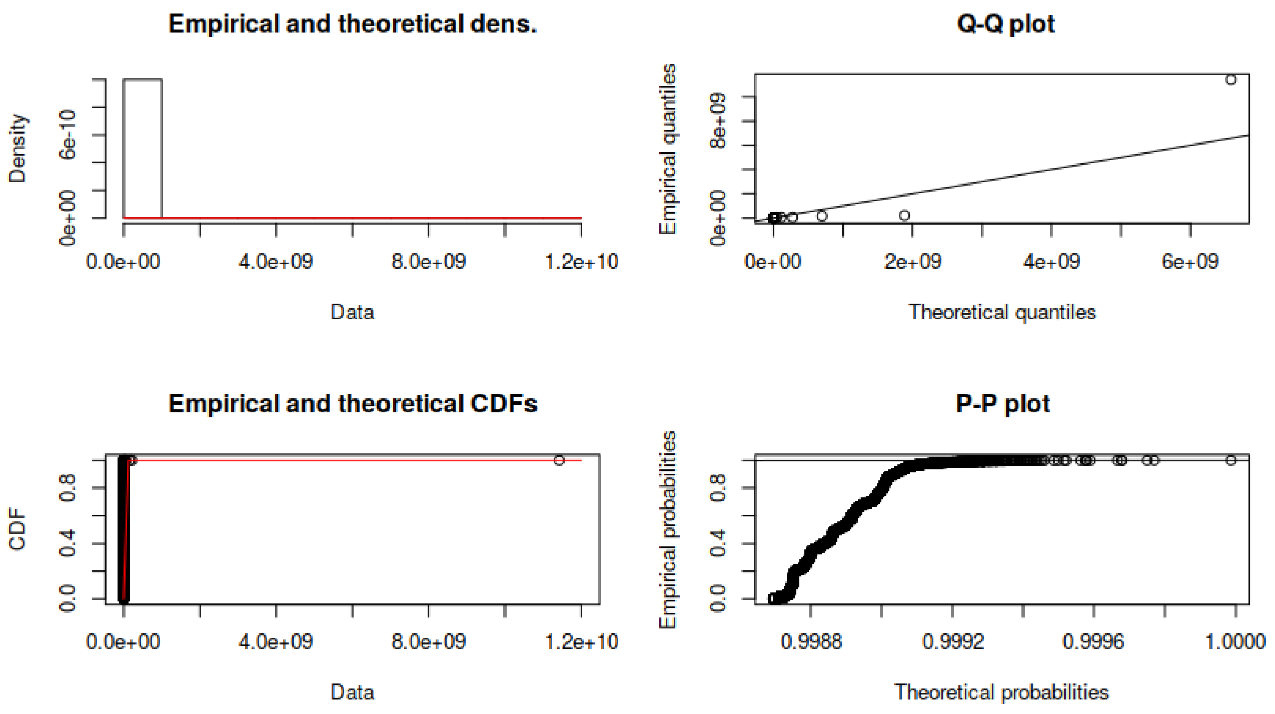
For this part, first I used the ***descdist()*** function in the ***fitdistplus*** package in R to preliminarily check the data set is closest to what distribution.

```
> descdist(data[,8], discrete = FALSE, boot = NULL, method = "unbiased", graph = TRUE, obs.col = "darkblue", obs.pch = 16, boot.col = "orange")
summary statistics
------
min:  32   max:  11424977388
median:  442
mean:  738269.4
estimated sd:  89197773
estimated skewness:  128.0077
estimated kurtosis:  16398.76
```



As we can see, the Observation point was most closed to the line for gamma distribution. So basically we could say the data set follows the gamma distribution. And the validation was done below:

```
> fit.gamma<-fitdist(data[,8],"gamma","mme")
> plot(fit.gamma)
```



2.9: Compare the number of flows with 1, 10, 60, 120 and 1800 second timeouts.

First, I generated different .t2 files regarding different timouts:

```
#!bin/bash

crl_flow -Ci=14400 -cl -Tf1 -Cai=1 data.pcap > data1.t2
crl_flow -Ci=14400 -cl -Tf10 -Cai=1 data.pcap > data10.t2
crl_flow -Ci=14400 -cl -Tf60 -Cai=1 data.pcap > data60.t2
crl_flow -Ci=14400 -cl -Tf120 -Cai=1 data.pcap > data120.t2
crl_flow -Ci=14400 -cl -Tf1800 -Cai=1 data.pcap > data1800.t2
```

Then I extracted flow data from the .t2 files:

```
#!bin/bash

gawk '$1~/^.*\..*\..*\..*/||$1~/^.*:.*:.*/' data1.t2 > data1_flow.t2
gawk '$1~/^.*\..*\..*\..*/||$1~/^.*:.*:.*/' data10.t2 > data10_flow.t2
gawk '$1~/^.*\..*\..*\..*/||$1~/^.*:.*:.*/' data60.t2 > data60_flow.t2
gawk '$1~/^.*\..*\..*\..*/||$1~/^.*:.*:.*/' data120.t2 > data120_flow.t2
gawk '$1~/^.*\..*\..*\..*/||$1~/^.*:.*:.*/' data1800.t2 > data1800_flow.t2
```

Next I counted how many flows were there in each files:

```
#!bin/bash

echo "number of flow with 1 second timeout is: `cat data1_flow.t2 | wc -l`"
echo "number of flow with 10 second timeout is: `cat data10_flow.t2 | wc -l`"
echo "number of flow with 60 second timeout is: `cat data60_flow.t2 | wc -l`"
echo "number of flow with 120 second timeout is: `cat data120_flow.t2 | wc -l`"
echo "number of flow with 1800 second timeout is: `cat data1800_flow.t2 | wc -l`"
```

And the results are as follows:

```
pengh1@vdiubuntu104 ~/Desktop/FinalAssignment/Task2/PS2/2.9
 % bash countrow.sh
number of flow with 1 second timeout is: 53412
number of flow with 10 second timeout is: 32056
number of flow with 60 second timeout is: 16430
number of flow with 120 second timeout is: 15644
number of flow with 1800 second timeout is: 13706
```

As we can see, when the timeout is longer, the number of the flows is smaller.

TCP connection data PS3

2.10: Round-trip times and their variance.

For this task, I first extracted the round-trip time using ***grep*** and ***awk*** commands:

```
grep 'RTT avg:' data.txt | awk '{print $3,$7}' > rtt0.txt
```

Then I used R to calculate its variance:

```
>   View(rtt0)
> colnames(rtt0)<-c("src to dst","dst to src")
> summary(rtt0[,1])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0    24.5    42.0   124.9   234.4   622.8
> sd(rtt0[,1])
[1] 125.8001
> summary(rtt0[,2])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   0.100   4.203  10.200  69.300
> sd(rtt0[,2])
[1] 6.200801
```

Note that the **rtt0[,1]** represents the round-trip time of source ip to destination ip, while **rtt0[,2]** represents the round-trip time of destination ip to source ip.

2.11: Total traffic volume during the connection

I used R to finish this task. I assume the total traffic volume was obtained by:

$$total\ traffic\ volume = {\sum bytes}\Big/{Max(end\ timestamp) - Min(start\ timestamp)}$$

Therefore, the commands I used and the results are as follows:

```
> sum(data[,8])*8/(max(data[,11]-min(data[,10])))
[1] 6734491
> sum(data[,8])*8/(max(data[,11]-min(data[,10])))/(1024)^2
[1] 6.422511
```

As we can see, the total traffic volume in Bps is 673491 bits per second, which is 6.422511 Mbits per second. It is basically the same as my actual downloading speeds, while it sometimes varies.

Conclusions

1. Traffic volume at different time scales. Are there any recognizable patterns?

Basically, the plots of two different time scales are quite similar to each other. The traffic volume varied from the beginning. Then it went high and stayed stable, for seconds time scale it was from around 1000 seconds to 3900 seconds, while for seconds time scale it was from around 10 minutes to 40 minutes. Next, it suddenly became quite low (nearly to 0) and stayed stable, for seconds time scale it was from around 3900 seconds to 10000 seconds, while for seconds time scale it was from around 40 minutes to 100 minutes. And finally, it went normal and stayed stable, for seconds time scale it was from around 10000 seconds to 14400 seconds, while for seconds time scale it was from around 100 minutes to 240 minutes.

I assumed that it was because I was using my laptop from 0 minutes to 40 minutes (from 11:00 to 11:40), then I left my computer and went out for dinner until 100 minutes (from 11:40 to 12:40) so the traffic volume was quite low. And then I started to use my computer again so the traffic volume went up again and remained stable until the capture was end. So basically the traffic volume plot matched the real situation.

2. Characteristics of top 5 most common applications used (studies of the port numbers).

I used Python to determine the top 5 most common port numbers, which are as follows:

port 445: Common Internet File System (CIFS): Microsoft-DS (Directory Services) SMB file sharing
port 61614: TCP/UDP: Dynamic and/or Private Ports / TCP: Xsan. Xsan Filesystem Access (Apple)
port 8801: TCP/UDP: EMC2 (Legato) Networker or Sun Solcitice Backup (Official) / UDP: QuickTime Streaming Server (Apple)
port 52760: TCP/UDP: Dynamic and/or Private Ports / TCP: Xsan. Xsan Filesystem Access (Apple)
port 58410: TCP/UDP: Dynamic and/or Private Ports / TCP: Xsan. Xsan Filesystem Access (Apple)

3. Comparison of above results with result from data set FS2.

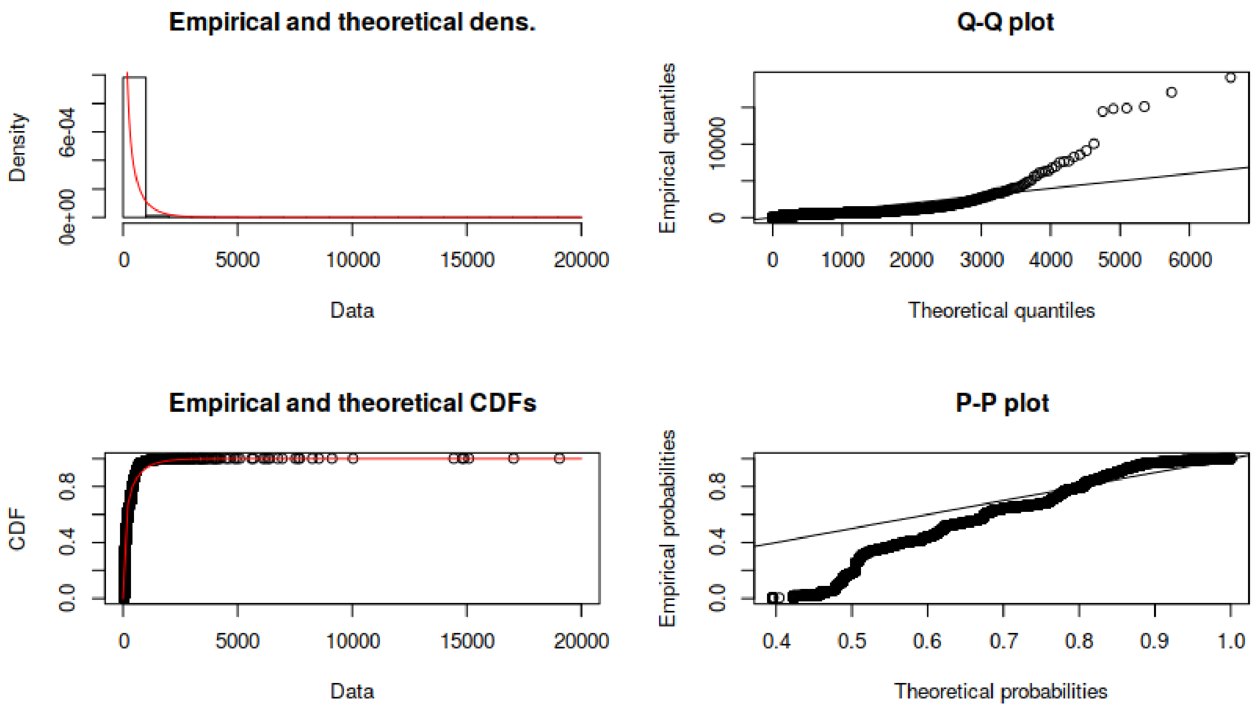
Regarding the traffic volume, the patterns of both cases are easy to recognize but the trends are quite different. And since the capture duration of FS2 (24 hours) is longer than the capture duration of my own data (4 hours), there are more fluctuations and variations in FS2 patterns, while my own capture data is somehow monotonous.

Regarding the top 5 most common applications used, the conclusion for FS2 is that it might be used for file transmission among different computers, and from my side I mainly used my virtual machine to play Youtube videos and used my home computer normally while capturing the packets. So basically according to the results, both interpretations make sense in network traffic view.

4. Differences of flow and packet measurements in the example case.

The flow and packet distributions by port numbers are somehow similar, but there are still some differences. For example, the top 5 most common port numbers for measurement of flow data are 53 (DNS), 443 (HTTP), 0 (dynamic port in programming APIs), 135 (DCE endpoint resolution) and 80 (HTTP), which are quite different from measurement of packet data (445, 61614, 8801, 52760, 58410, mainly rely on transport layer protocols). Besides, regarding the distribution of flow length and packet length, both of the distributions are assumed to be fit in gamma distribution, however, the packet length is better fit in the gamma distribution (the plot is at below).

In my perspective of view, the behaviours of the flows and packets are somehow similar but they are actually not necessarily the same, where the flow behaviours take place on application layer and packet behaviour take place on transport layer.



Packet length distribution fit in gamma distribution

5. Your findings on retransmissions.

First, I extracted retransmissions data from the TCP connection statistics:

```
grep 'avg retr time:' data.txt | awk '{print $4,$9}' > retr.txt
```

Then I used R to process the retransmissions data:

```
> colnames(retr)<-c("src to dst","dst to src")
> summary(retr[,1])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    0.0     0.0     0.0   534.4     0.0 25540.5 
> sd(retr[,1])
[1] 2330.534
> summary(retr[,2])
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
   0.000    0.000    0.000    7.383    0.000 3939.600 
> sd(retr[,2])
[1] 122.8786
```

It is interesting that the average retransmissions time in source ip to destination ip were quite longer than in the source ip to destination ip. According to my observations, the source ip mainly worked on TCP/UDP protocols and the destination ip mainly worked on HTTP protocol. Since the video contents are mainly transported by UDP protocol, which does not guarantee reliable transmissions and there might be quite a lot timeout retransmissions.

Task 3: Analysing active measurements

For Task 3, I made the measurements from Aalto servers, mainly on the VDI VMware Ubuntu 18.04 Nvidia. I collected the data for about 15 days, but due to unexpected Aalto server failure, the 14-day data actually consisted of two parts of data, one was from 9.29 to 10.08, and the other one was from 10.28 to 11.04. Therefore, there were some difference between these two timeframes, but as far as I know it is allowed to have the whole dataset consisted of two parts as long as it shows useful information.

3.1 Latency data plots (AS1.x)

• Provide box plots including all successful latency measurements from AS1.x data sets (one box per data set; ignore lost packets). Make sure numerical values could be seen. What observations can be made, for example differences between sites? Were there differences in AS1.d_N_ and AS1.n*N?

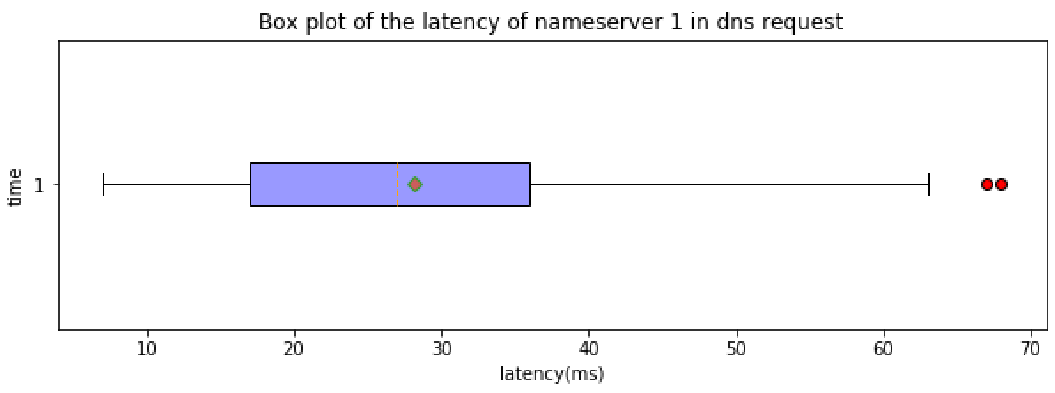
Solution and results:

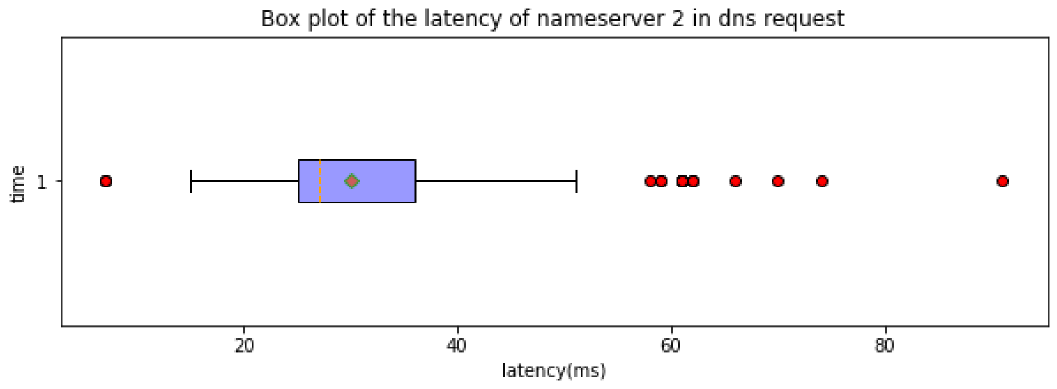
For this part, I used the Python scripts to extract useful data from the measurement files. I will included it in the submission files. An example code used for box plots is as below:

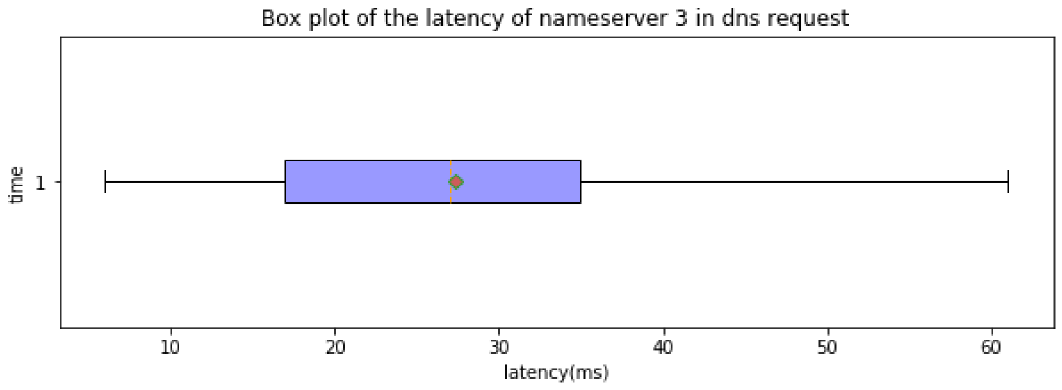
```
plt.gcf().set_size_inches(10, 3)
plt.boxplot(x = [x for x in ns1pingy], patch_artist=True, showmeans=True,
            showfliers=True, vert=False, boxprops = {'color':'black','facecolor':'#9999ff'},
             flierprops = {'marker':'o','markerfacecolor':'red','color':'black'},
             meanprops = {'marker':'D','markerfacecolor':'indianred'},
             medianprops = {'linestyle':'--','color':'orange'})
plt.xlabel('latency(ms)')
plt.ylabel('time')
plt.title('Box plot of the latency of nameserver 1 in ping request with lost packets')
plt.show()
```

And the results are as follows:

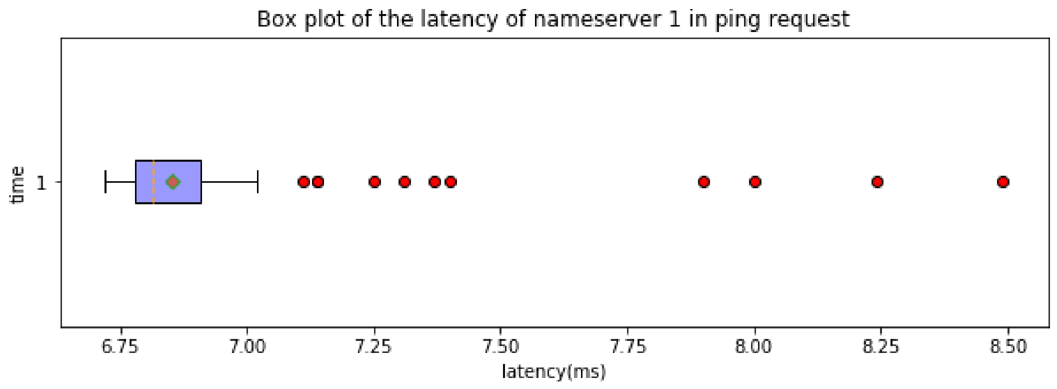
Name servers, DNS:

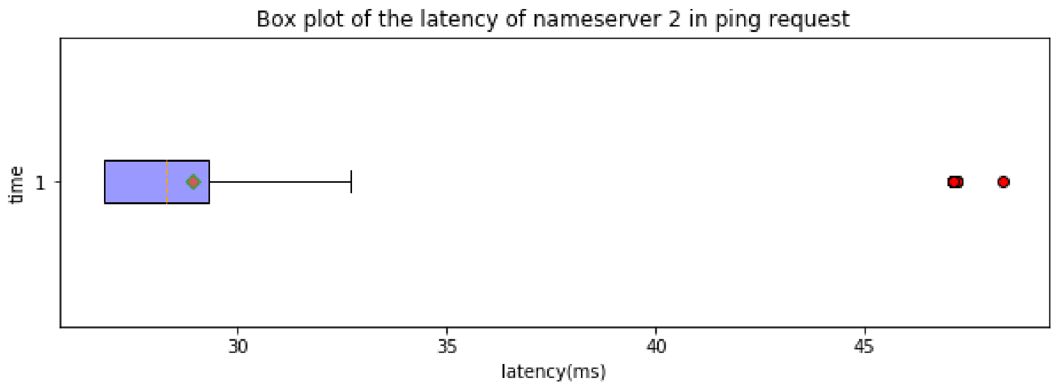


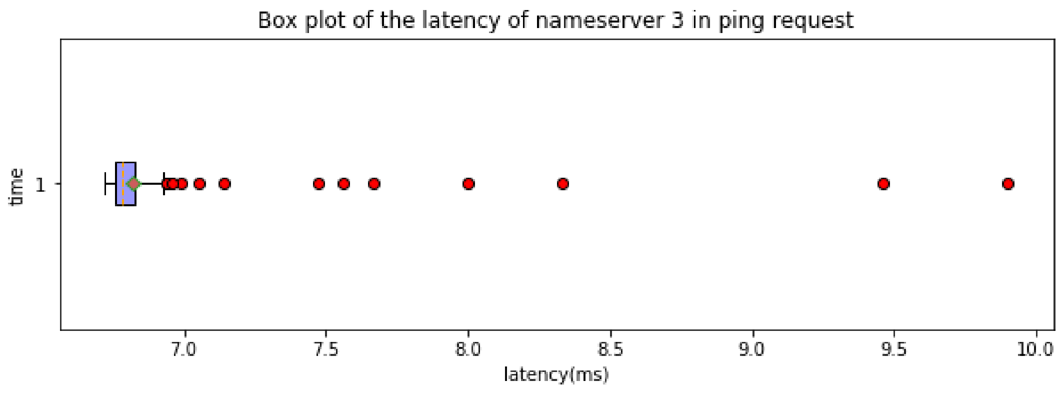




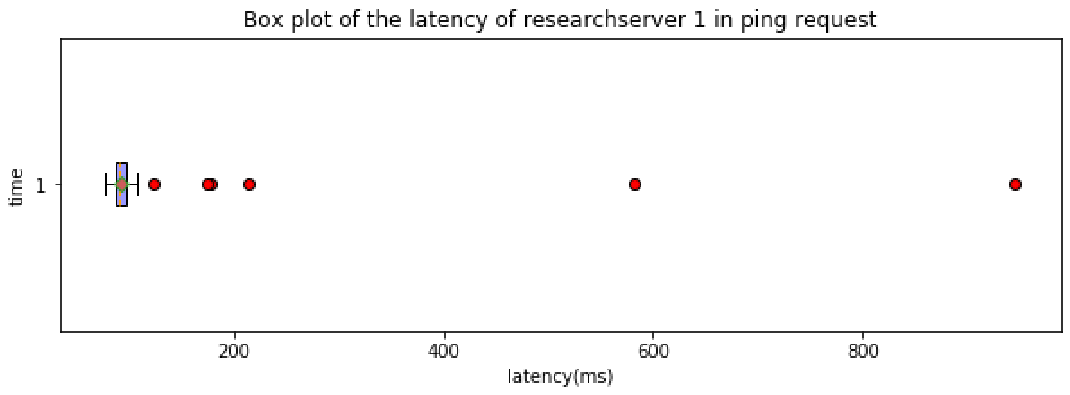
Name servers, ICMP:

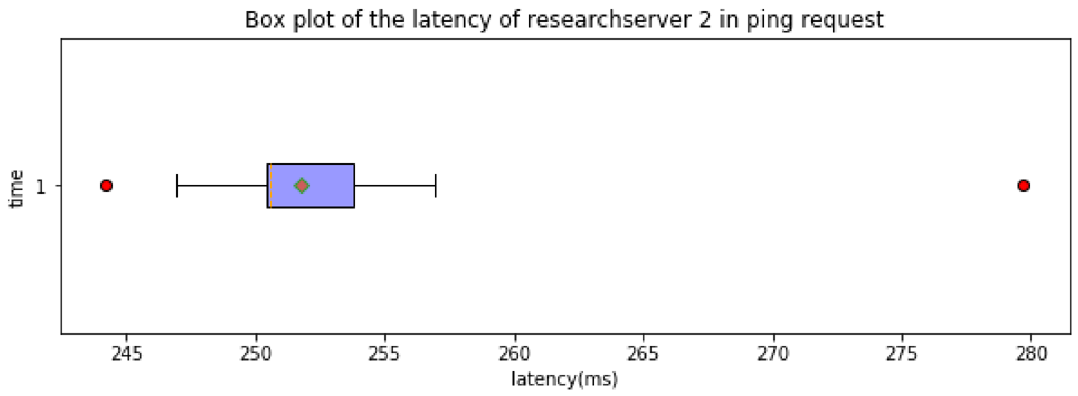


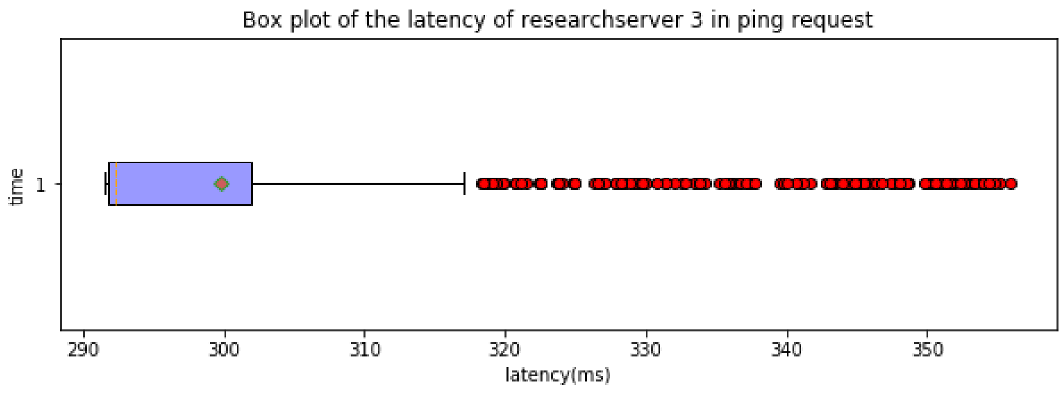




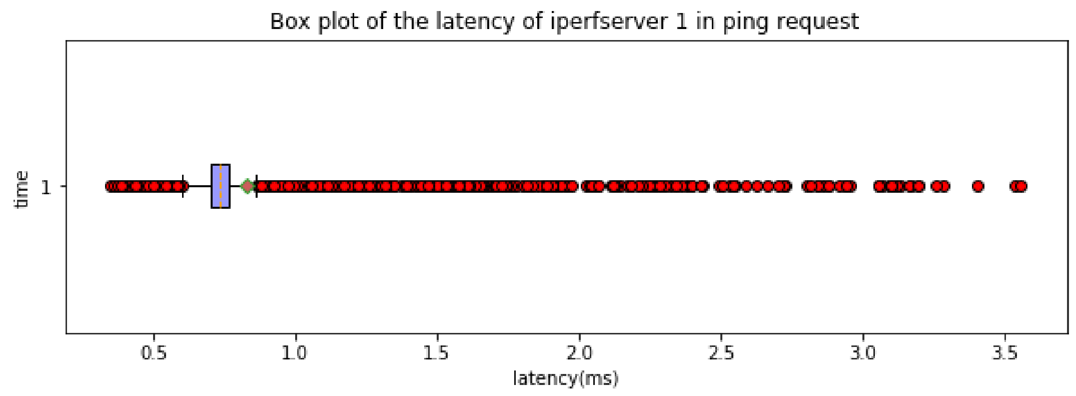
Research servers, ICMP:

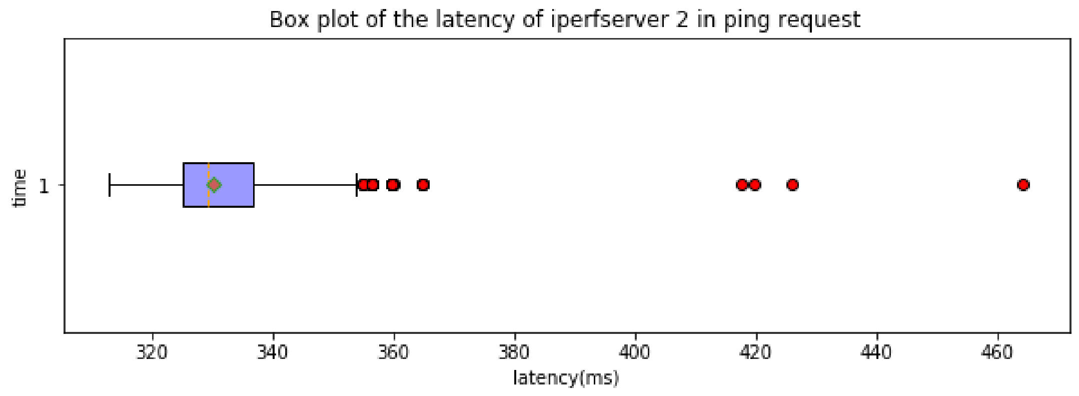






Iperf servers, ICMP





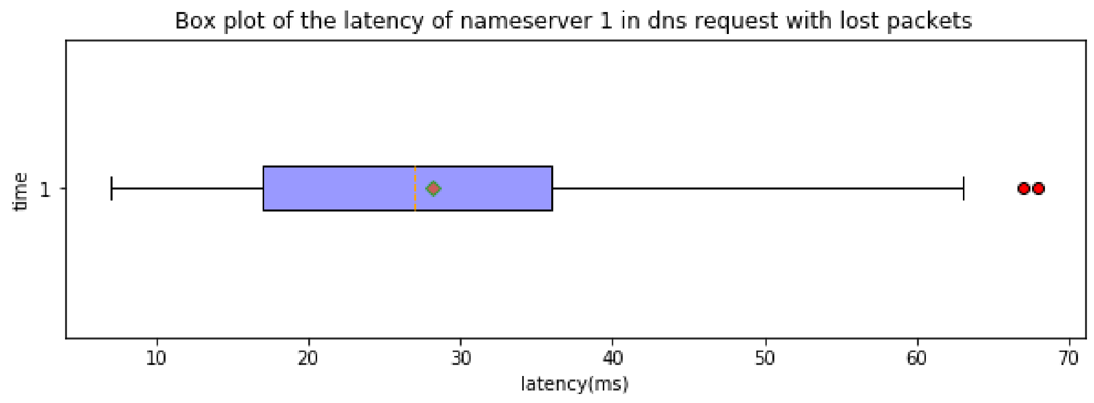
Observations:

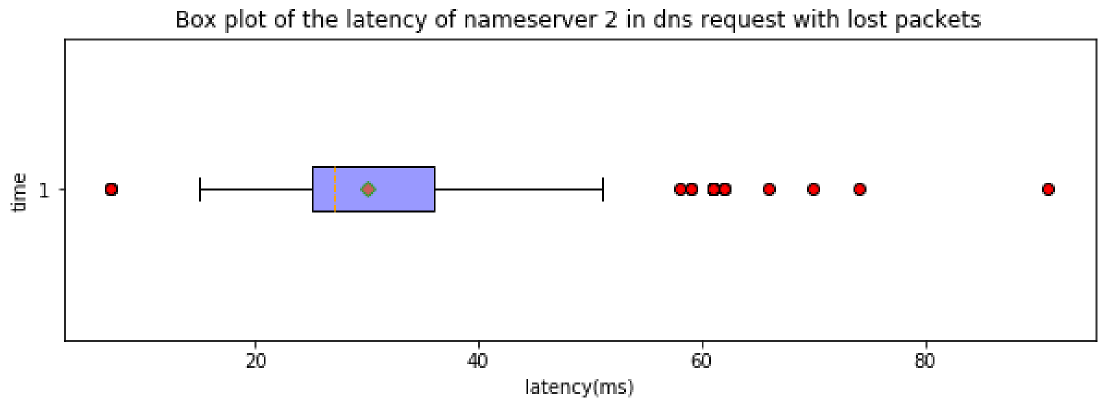
In the same kind of server, there are still differences between sites. For example, the latency spans of name servers for ICMP requests are quite different (6.5-8.5ms for n1, 25-50ms for n2, 6.5-10ms for n3), and the distributions of them also vary, where in n1 and n3 there are more outliers outside the acceptable latency range than n2. As for sites from different servers, the differences are even larger. The differences between latency spans are larger than between same kind of server. For example, the latency of i2 ranges 310-470ms, while n2 ranges 25-50ms and r1 ranges 100-900ms.

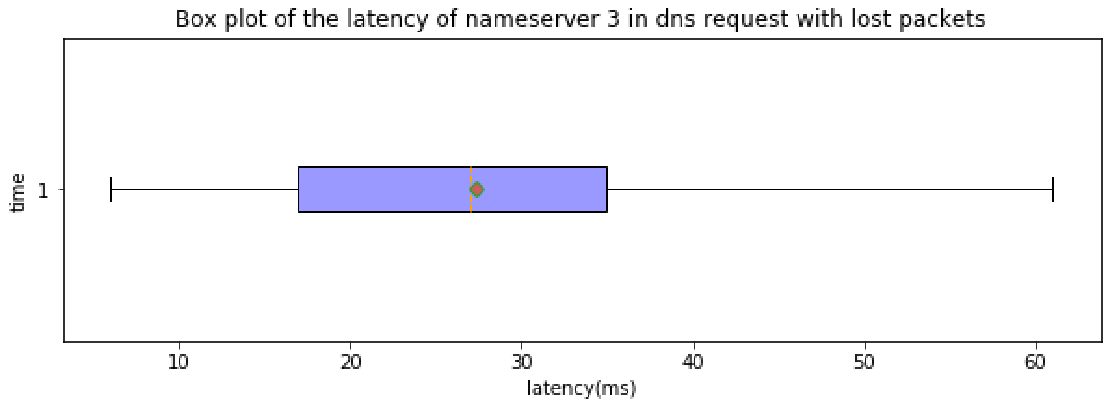
As for AS1.d_N_ and AS1.n*N, the latencies of DNS and ICMP requests are different for each site. Each has different span and numbers of outliers. For name server 1, d1 shows more stability than n2; for name server 2, n2 is more stable than d2, and for name server 3, the n2 is also more stable than d2.

• Another graph but this time consider also the lost packets. One option is define all lost packets to have some maximum delay (like 2 seconds, also any packet delayed more than 2 seconds would be shown as 2 s) and make single box plot for each dataset. There can be other options too.

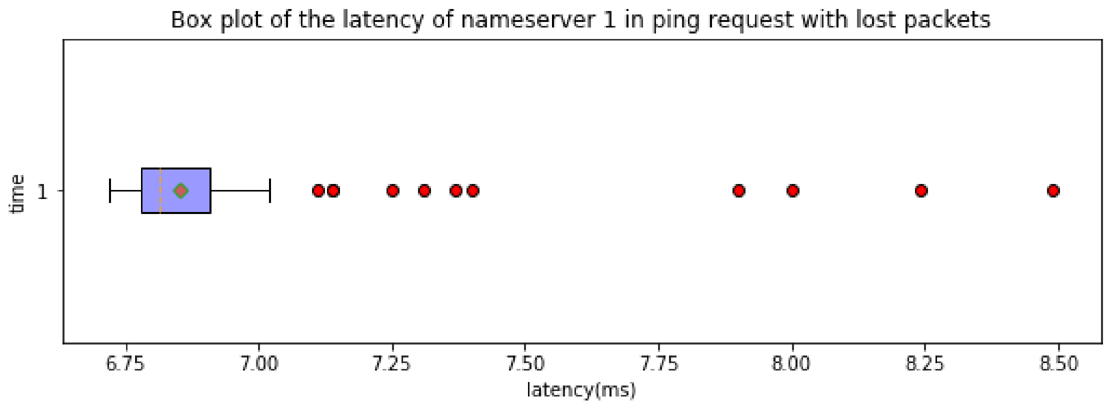
Name servers, DNS:

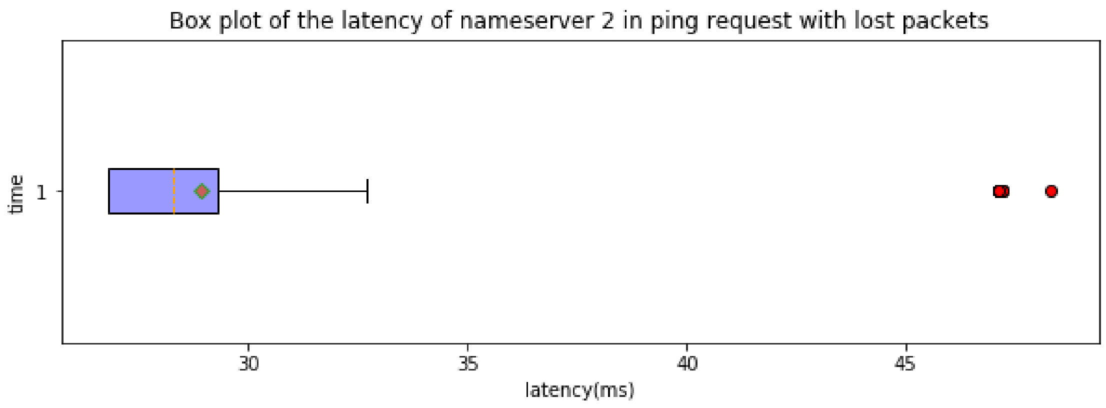


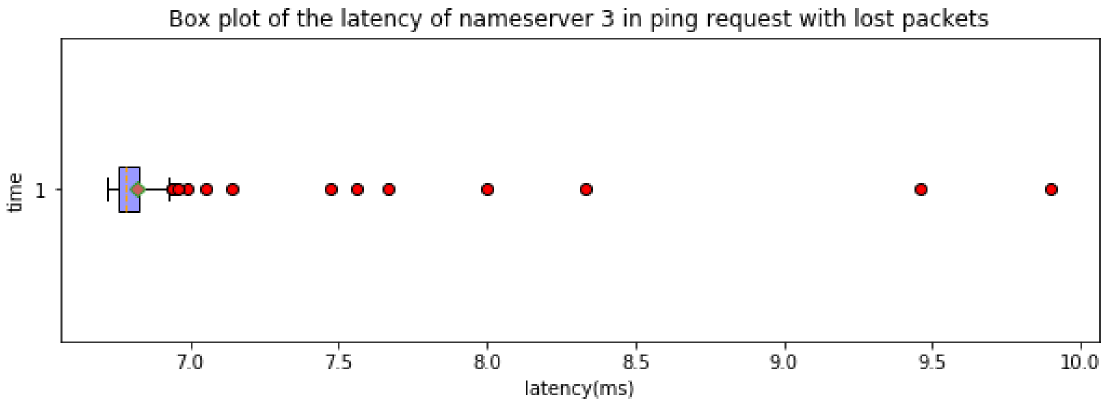




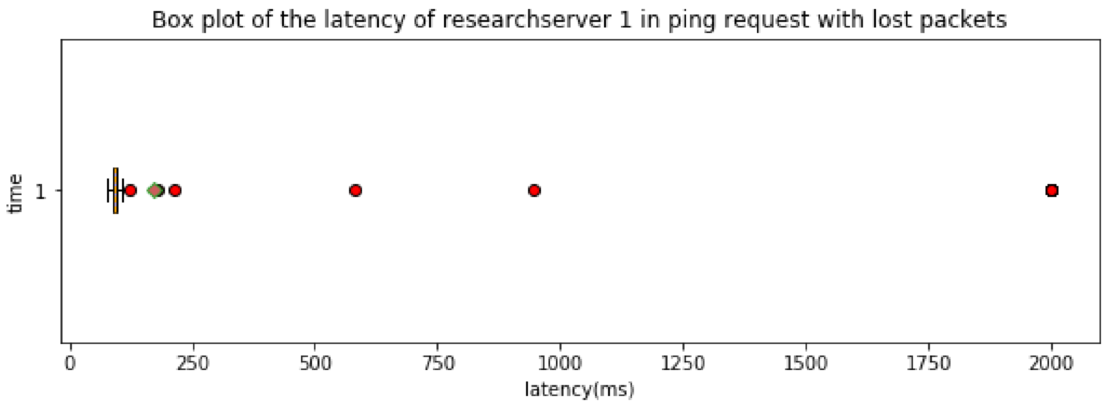
Name servers, ICMP:

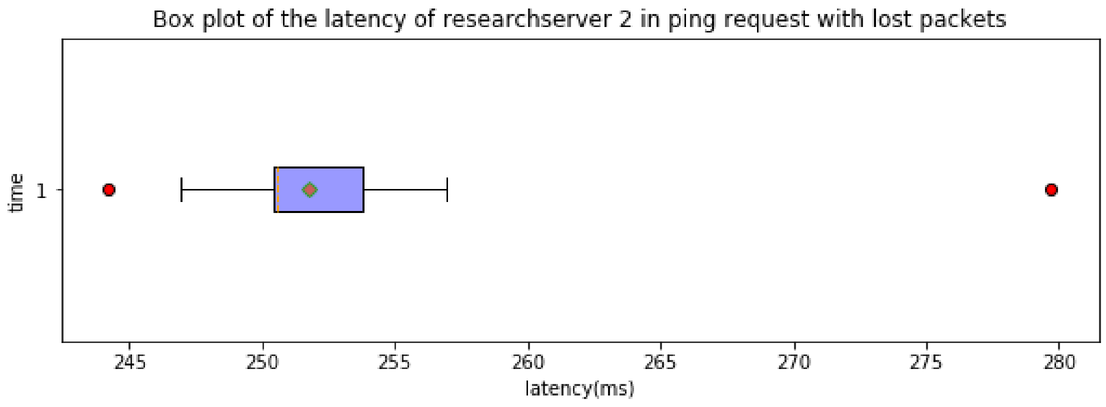


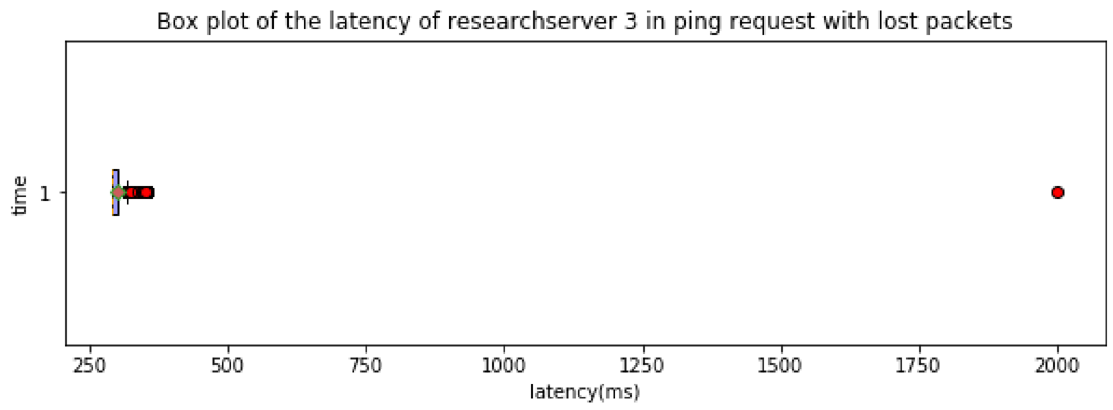




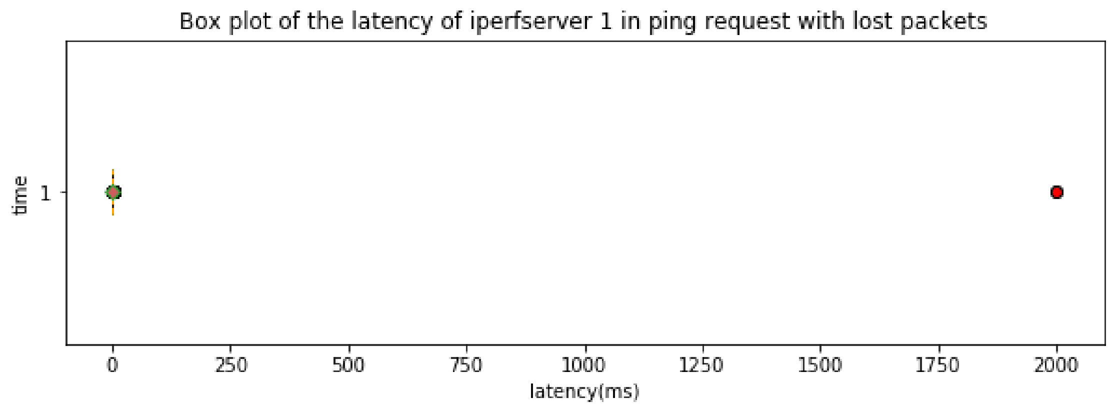
Research servers, ICMP:

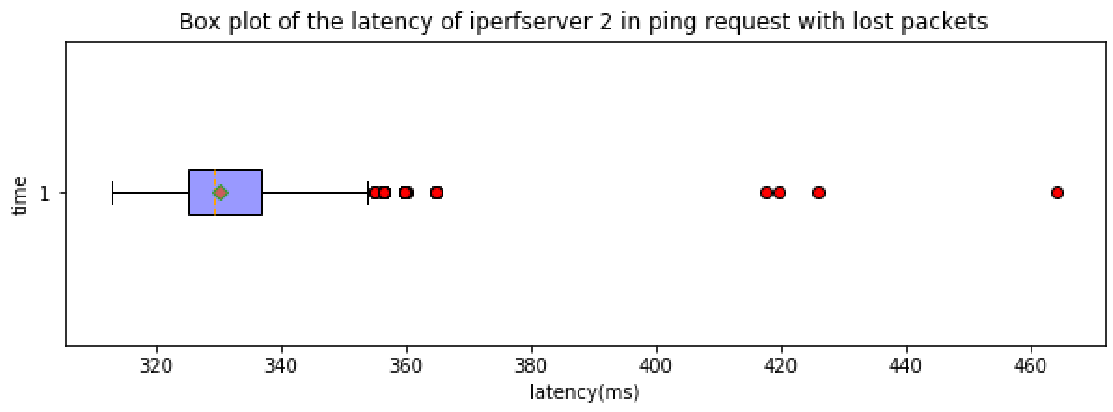






Iperf servers, ICMP:

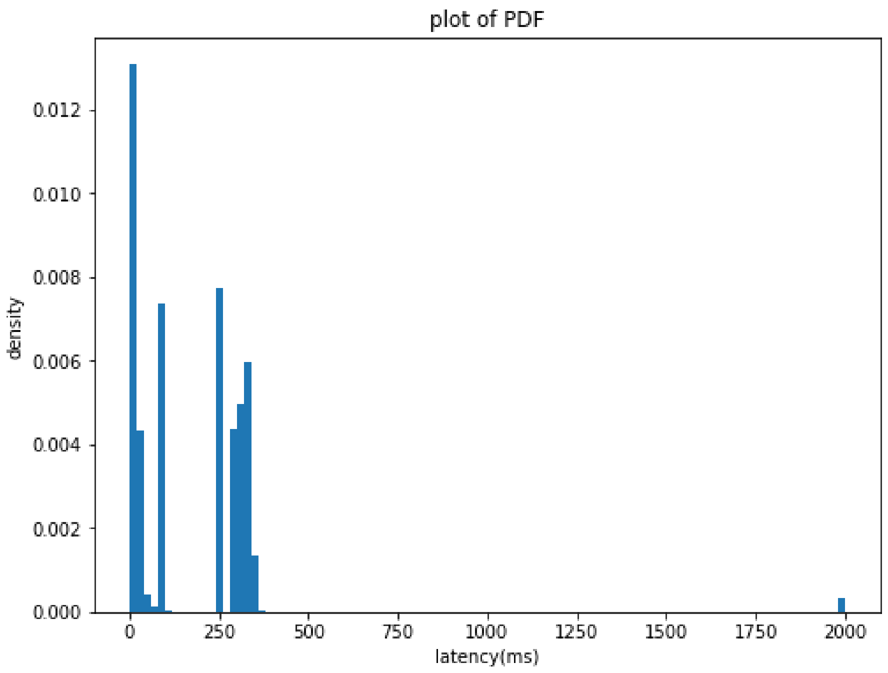


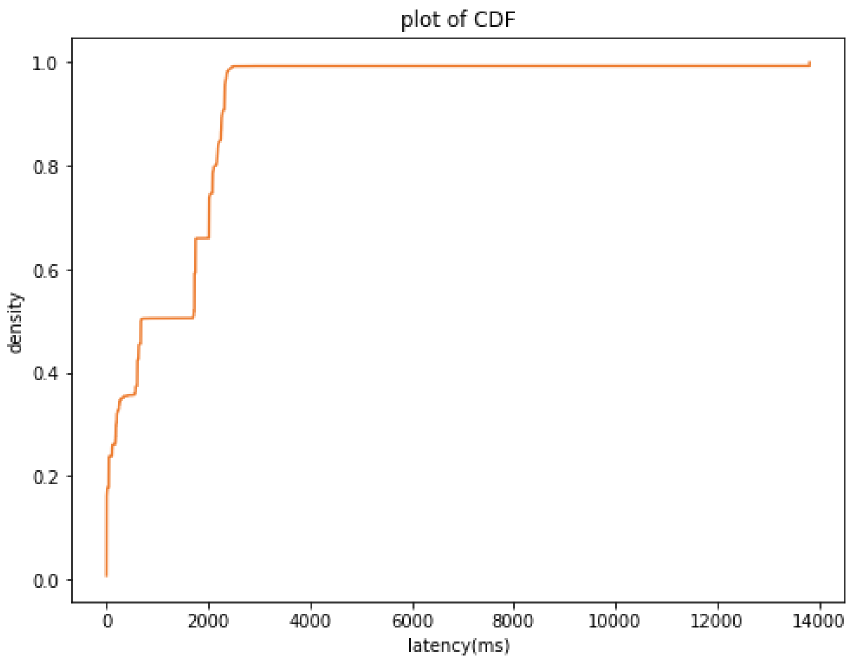


• Provide PDF and CDF plots including all AS1.x delay distributions.

In this part, the code used to plot the PDF/CDF and results area as follows:

```
data = ns1pingy+ns2pingy+ns3pingy+ns1dnsy+ns2dnsy+ns3dnsy+rs1pingy+rs2pingy+rs3pingy+is1pingy+is2pingy

#PDF plot
plt.gcf().set_size_inches(8, 6)
plt.ylabel('density')
plt.xlabel('latency(ms)')
plt.title('plot of PDF')
plt.hist(data, bins = 100, normed= True)
plt.show(block=True)

#CDF plot
plt.gcf().set_size_inches(8, 6)
hist, bin_edges = np.histogram(data, bins = len(data))
cdf = np.cumsum(hist)
CDF = [float(i)/float(len(data)) for i in cdf.tolist()]
plt.ylabel('density')
plt.xlabel('latency(ms)')
plt.title('plot of CDF')
plt.plot(CDF, '-', color='#ED7D31')
```





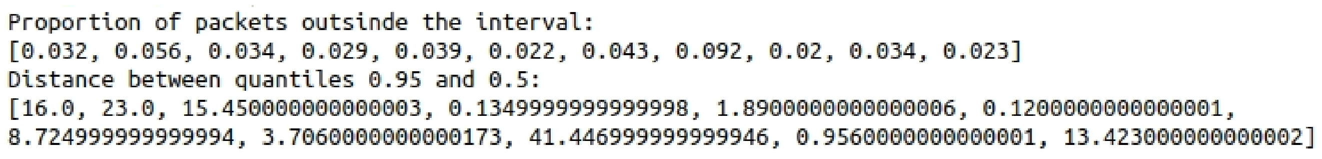
• Characterise delay distributions according to ITU-T Y.1541 in a tabular form for all AS1.x.

According to the chapter 5.1 (page 44) of material ***elec-7130.pdf*** provided in the course, ITU-T Y.1541 defines following ways to define delay distribution:

1. Select acceptable delay interval in advance and count the proportion of packets that fall outside this interval.
2. Distance between two quantiles like 0.95 and 0.5.

Based on these descriptions, the codes used to solve the problem and results are as follows:

```python
def calcinterval(datalist,a,b):
    interval=[]
    data=0
    for i in datalist:
        data=float(i[3])
        if (data<a) or (data>b):
            interval.append(data)
    out=round(float(len(interval))/float(len(datalist)),3)
    return out

def distpercentile(dataList):
    ltc=[]
    for i in dataList:
        ltc.append(float(i[3]))
    dst=round(np.percentile(ltc, 95), 3)-round(np.percentile(ltc, 50), 3)
    return dst

outside = [calcinterval(DNSinfo1,10,50), calcinterval(DNSinfo2,10,50), calcinterval(DNSinfo3,10,50), calcinterval(ICMPinfo1,6,7), calcinterval(ICMPinfo2,20,40), calcinterval(ICMPinfo3,6,7), calcinterval(RICMPinfo1,50,200), calcinterval(RICMPinfo2,250,300), calcinterval(RICMPinfo3,250,350), calcinterval(IICMPinfo1,0,2), calcinterval(IICMPinfo2,250,350)]
print('Proportion of packets outsinde the interval:')
print(outside)

dstper = [distpercentile(DNSinfo1), distpercentile(DNSinfo2), distpercentile(DNSinfo3),
        distpercentile(ICMPinfo1), distpercentile(ICMPinfo2), distpercentile(ICMPinfo3),
        distpercentile(RICMPinfo1), distpercentile(RICMPinfo2), distpercentile(RICMPinfo3),
        distpercentile(IICMPinfo1), distpercentile(IICMPinfo2)]
print('Distance between quantiles 0.95 and 0.5:')
print(dstper)
```



```
Proportion of packets outsinde the interval:
[0.032, 0.056, 0.034, 0.029, 0.039, 0.022, 0.043, 0.092, 0.02, 0.034, 0.023]
Distance between quantiles 0.95 and 0.5:
[16.0, 23.0, 15.450000000000003, 0.1349999999999998, 1.8900000000000006, 0.1200000000000001,
8.724999999999994, 3.7060000000000173, 41.446999999999946, 0.9560000000000001, 13.423000000000002]
```

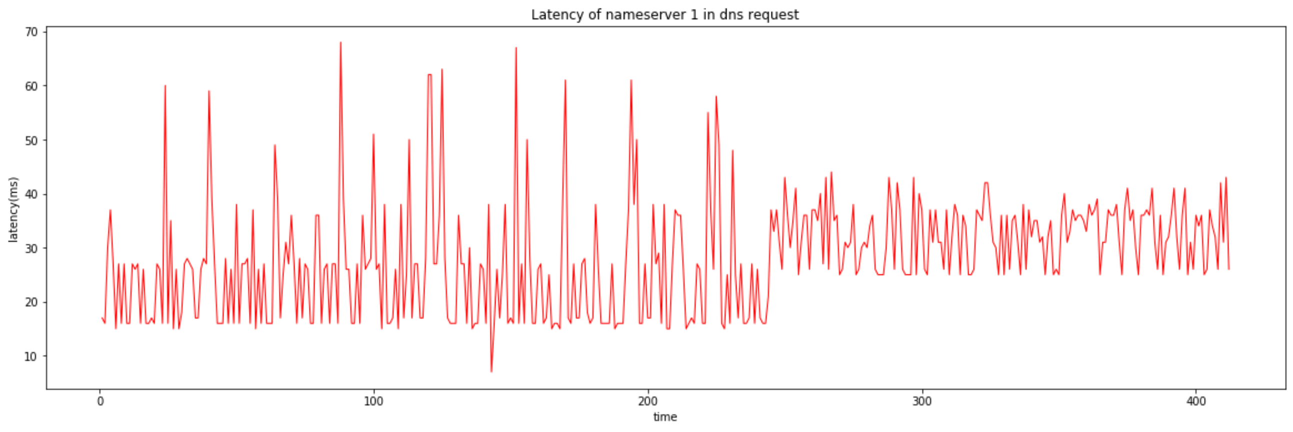
Dataset	Delay interval (ms)	Proportion of packets outside the interval	Distance between quantiles 0.95 and 0.5 (ms)
AS1.d1	10-50	3.2%	16.0
AS1.d2	10-50	5.6%	23.0

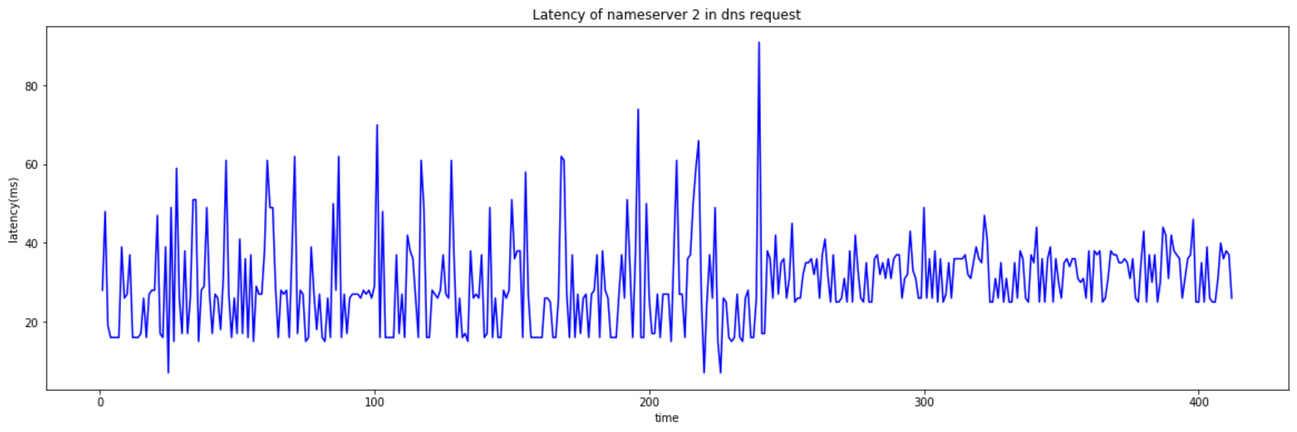
AS1.d3	10-50	3.4%	15.45
AS1.n1	6-7	2.9%	0.14
AS1.n2	20-40	3.9%	1.89
AS1.n3	6-7	2.2%	0.12
AS1.r1	50-200	4.3%	8.73
AS1.r2	250-300	9.2%	3.71
AS1.r3	250-350	2.0%	41.45
AS1.i1	0-2	3.4%	0.96
AS1.i2	250-350	2.3%	13.42

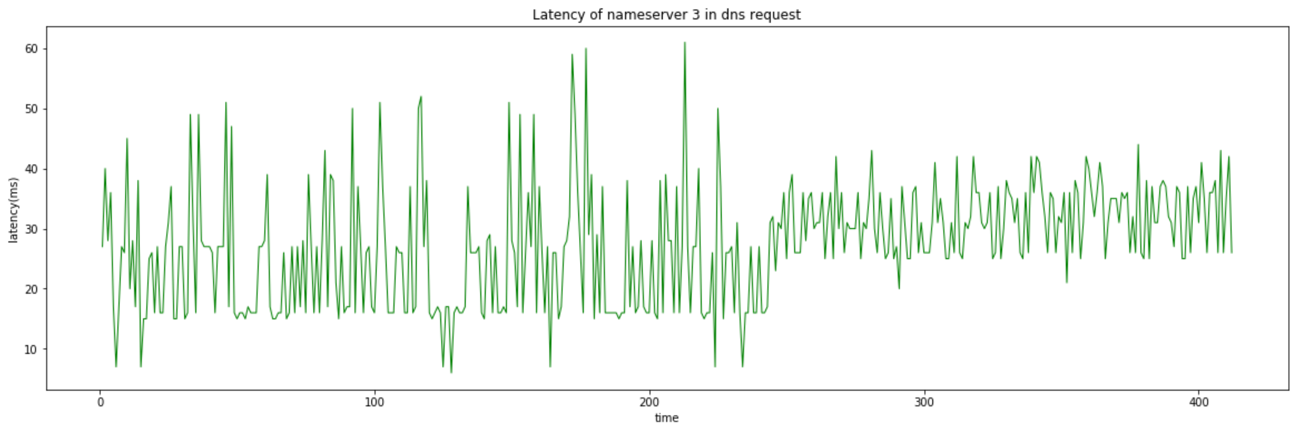
3.2 Latency data time series

• Plot time series of each data set AS1.x. Consider appropriate scaling for comparison. Any observations for e.g. diurnal patterns?

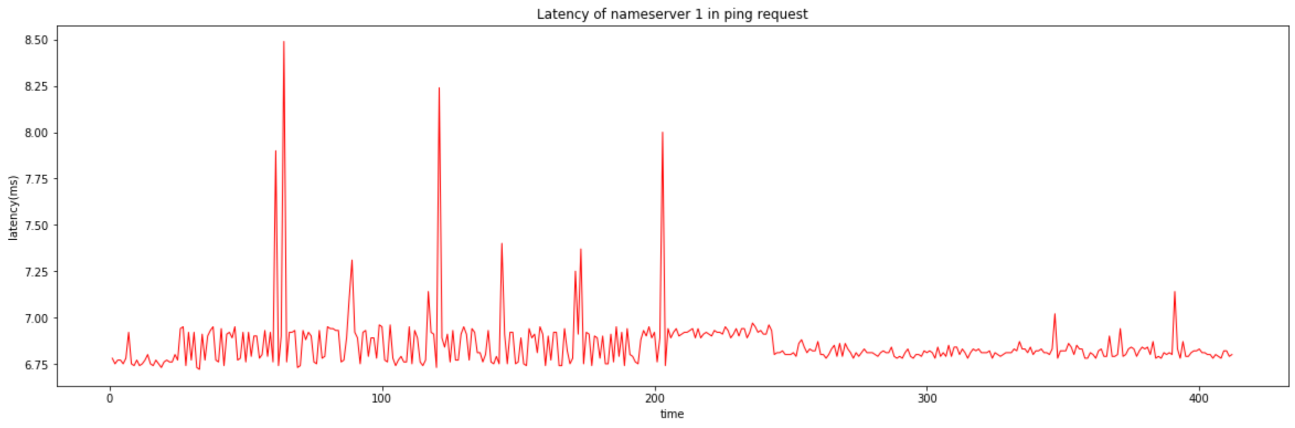
Name servers, DNS:

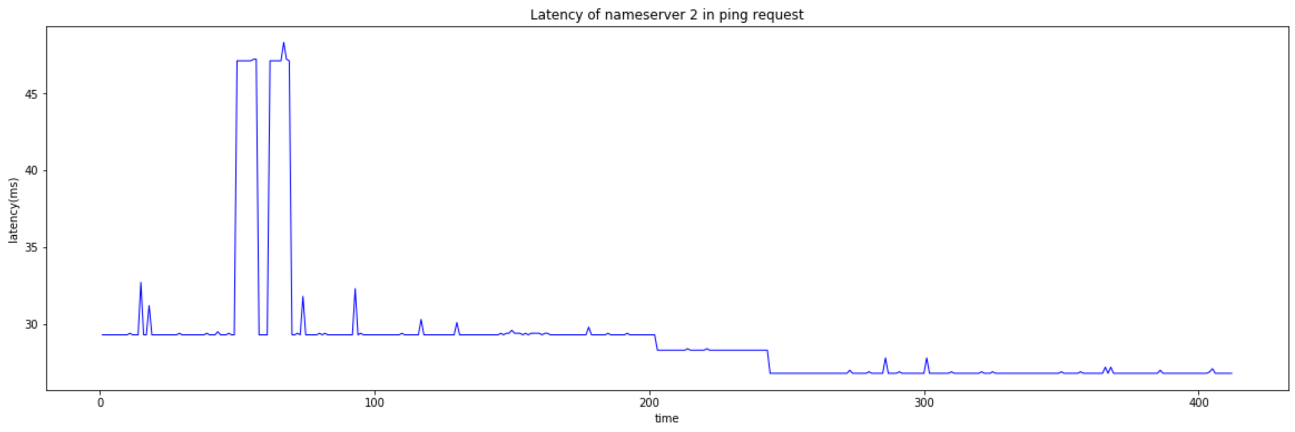


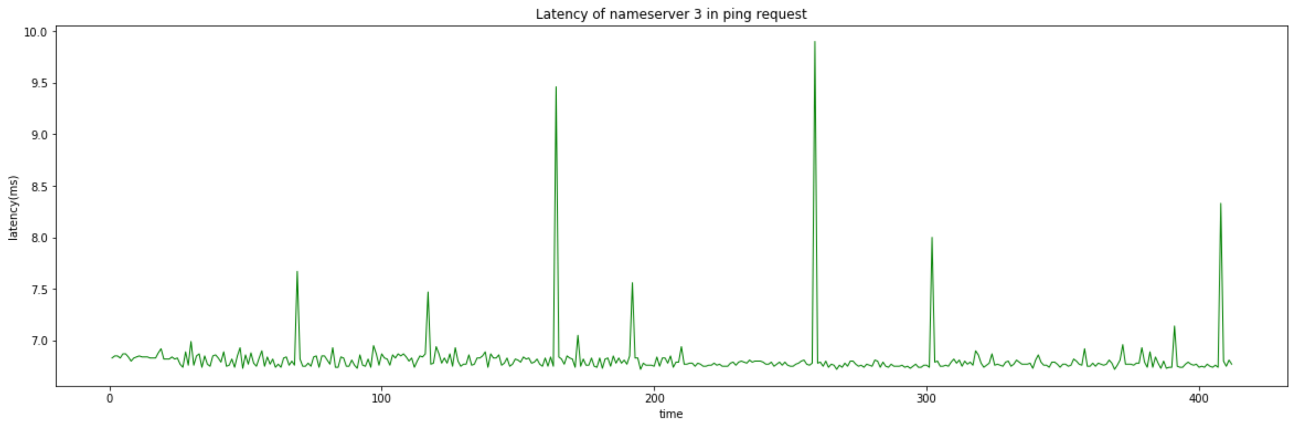




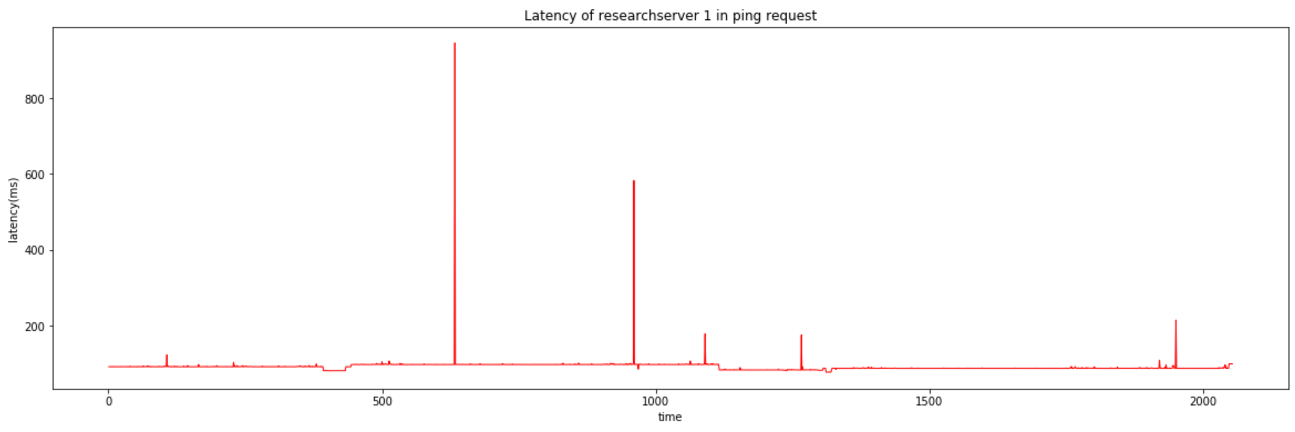
Name servers, ICMP:

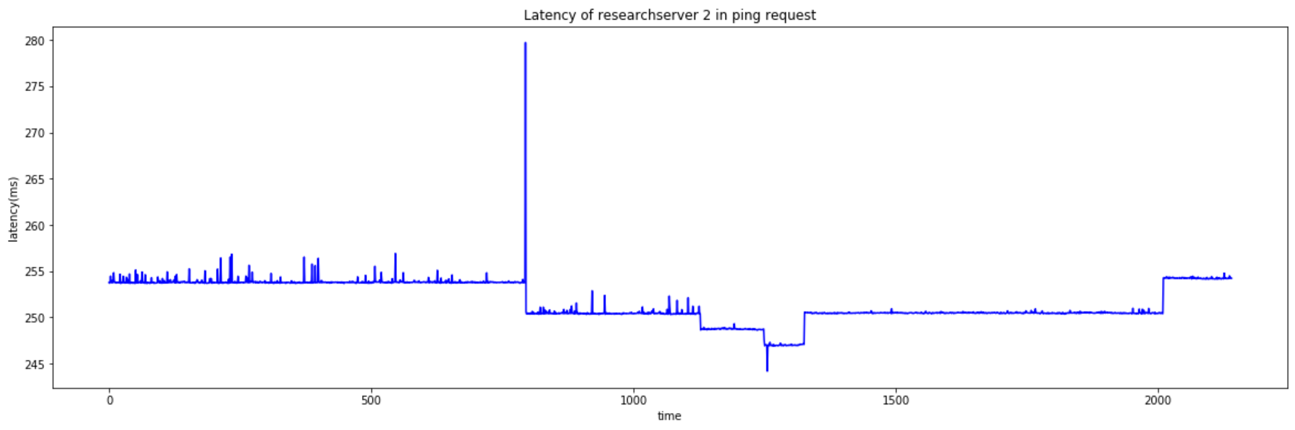


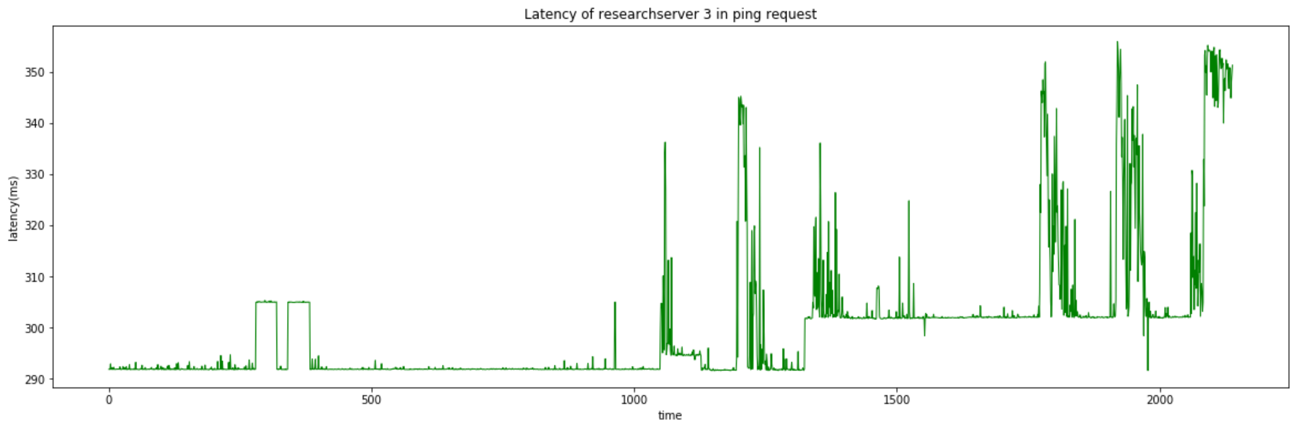




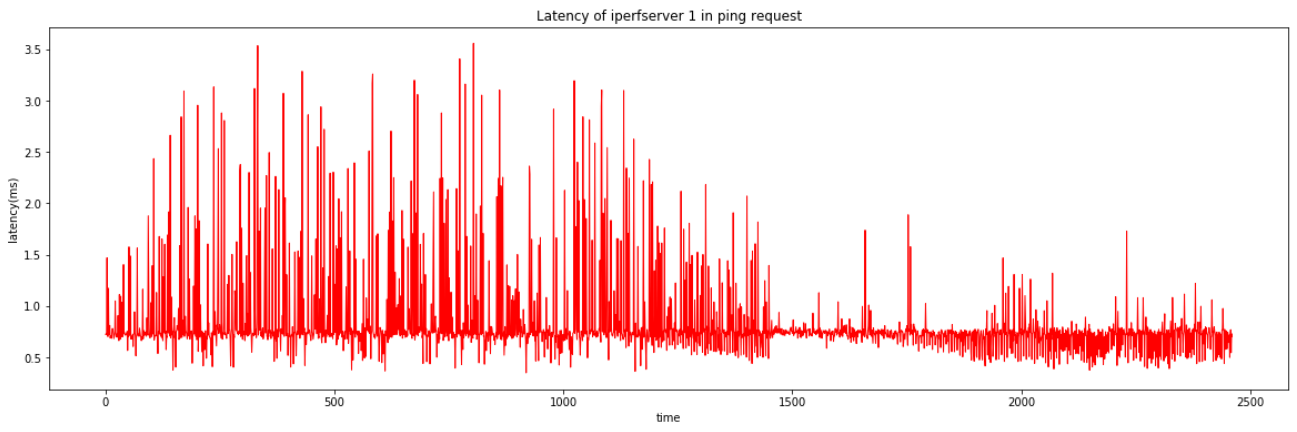
Research servers, ICMP:

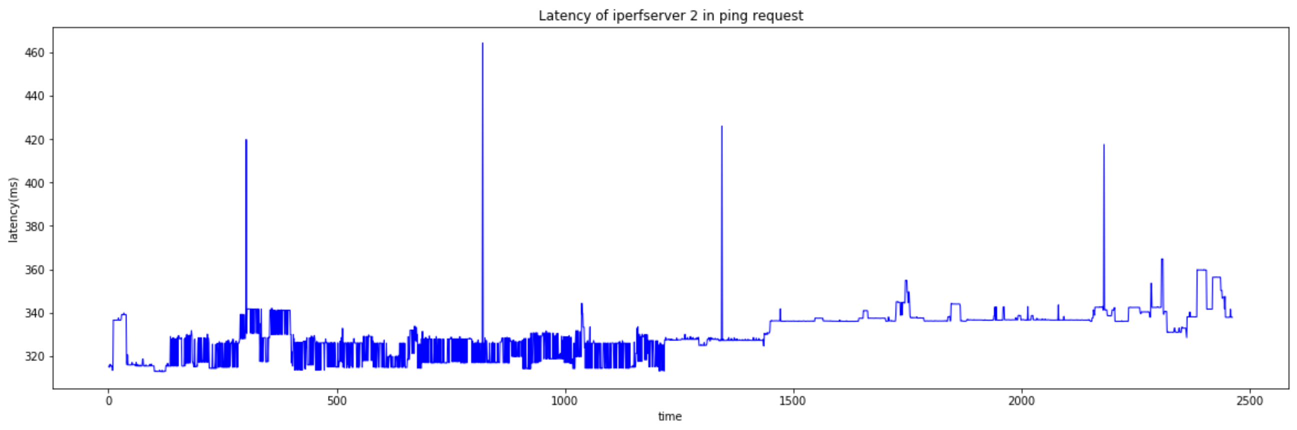






Iperf servers, ICMP:





Observations:

For name servers, the diurnal patterns of DNS requests are more apparent, where the fluctuations are more frequent and the latencies in the day and night are easy to recognize. Another observation is that the latency variation in 9.28-10.08 was larger than in 10.28-11.04, which might be the result of server optimization or fixation between some points between 10.08-10.28. As for ICMP requests, the latency fluctuations are comparatively smaller than DNS requests so the latency remains stable to some degree, while the occasional outliers are easier to observe. And the diurnal patterns are not very apparent since the plots seem more stable. Also, except n3 dataset, for the n1 and n2 datasets the latency was a bit lower in 10.28-11.04 than in 9.28-10.08.

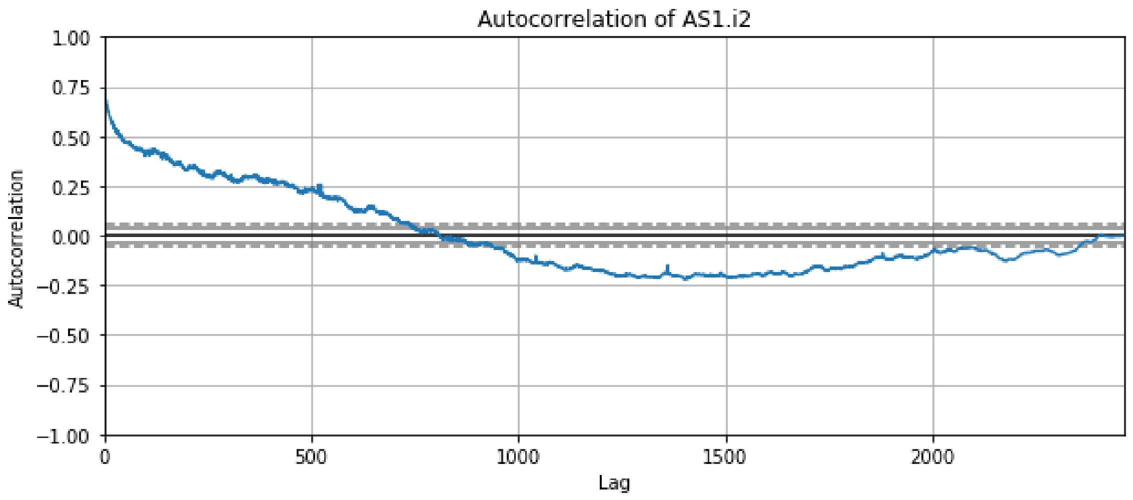
For research servers, we can only see diurnal patterns in r2 dataset from 9.28-10.08 (not so apparent but are still there) and in r3 from 10.28-11.04. As for other timeframe, the latency remain stable and only occasional outliers can be seen. And it looks weird that in r2 dataset, the latency has four different levels, which I could not reasonably interpret it.

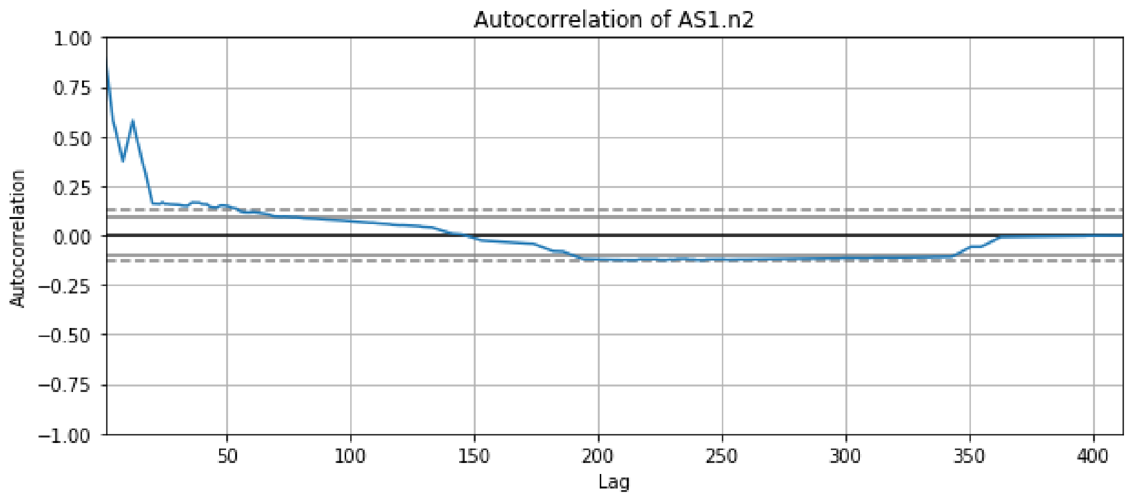
For iperf servers, the diurnal patterns are easier to see in the whole i1 dataset and i2 dataset in 9.28-10.08 (not so apparent but are still there). For i1 dataset, the latency is lower in 10.28-11.04, while for i2 dataset it seems the latency is lower in 9.28-10.08.

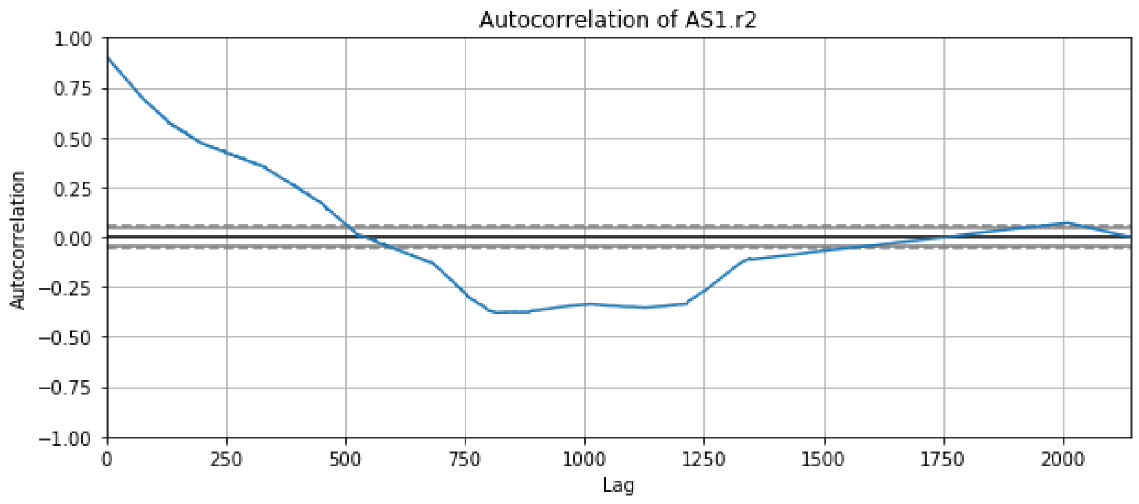
• Select AS1.i2 and minimum two other most interesting data sets from AS1.x. Make an autocorrelation plot. Any observations?

I chose AS1.n2 and AS1.r2 to do the plot since they are all ICMP requests. The codes used to plot autocorrelation and the results are as follows:

```
def autocorrelation(datalist,Dataname):
    dl=[]
    for i in datalist:
        dl.append(float(i[3]))
    s = pd.Series(dl)
    plt.figure(figsize=(10,4))
    autocorrelation_plot(s)
    plt.title('Autocorrelation of '+ Dataname)
    plt.show()

autocorrelation(IICMPinfo2, 'AS1.i2')
autocorrelation(ICMPinfo2, 'AS1.n2')
autocorrelation(RICMPinfo2, 'AS1.r2')
```







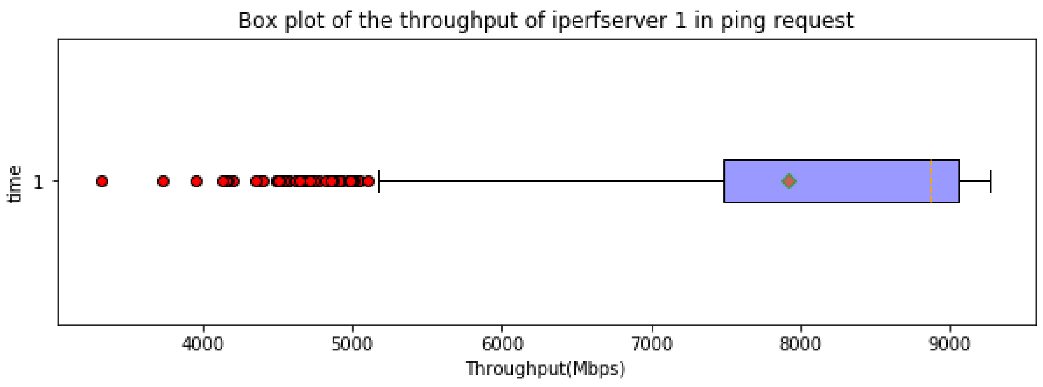
Observations:

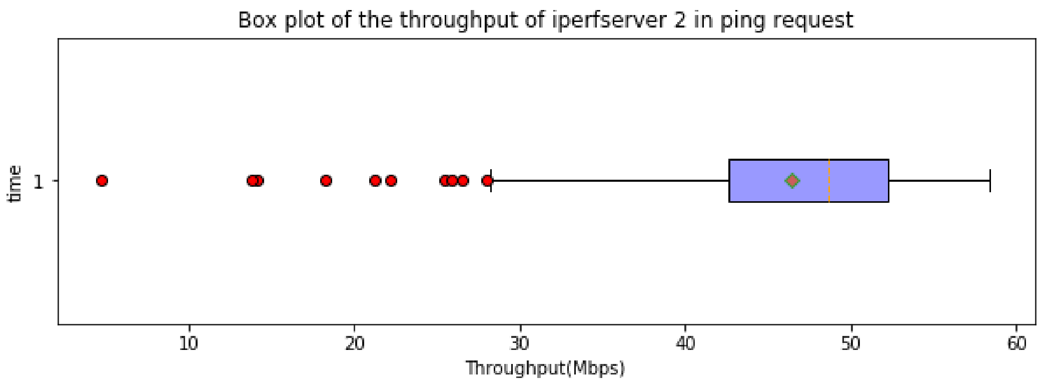
According to the autocorrelation plot, more recent values (smaller lags, less than 800 for i2, less than 145 for n2 and less than 550 for r2) are stronger correlated with the current value than older values (larger lags, more than the values above), and at extremely large lags (at the end) the correlation decays to 0. The r2 dataset has the most autocorrelation and i2 has the least.

3.3 Throughput

• Plot throughput measurements as box plots for both AS2.x data sets

The code used is similar to that used above, so here are just the results:





• From throughput, compute and tabulate for both data sets representative values using

– mean

– harmonic mean

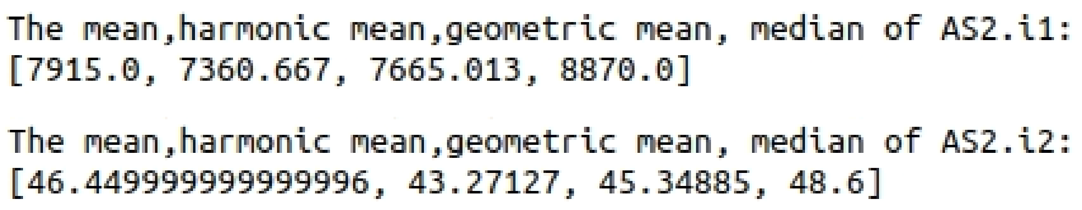
– geometric mean

– median

The codes used to compute the means are as follows:

```python
def calcvalue(dataList):
    dl=[]
    for i in dataList:
        if (i[2]!=None) and (len(i[2].split(' '))==2):
            d = i[2].split(' ')
            if d[1] == 'Mbits/sec':
                dl.append(float(d[0])/10)
            else:
                dl.append(float(d[0]))
    DL=sorted(dl, key=float)
    mean=round(sum(dl)/len(dl), 3)
    harmean=round(float(len(dl))/float(sum([1/i for i in dl])), 6)
    geomean=round(np.prod(dl)**(float(1)/float(len(dl))), 6)
    med=DL[(len(DL))/2]
    return [mean*1000,harmean*1000,geomean*1000,med*1000]

i1value=calcvalue(iperfinfo1dn)
i2value=calcvalue(iperfinfo2dn)
print('The mean,harmonic mean,geometric mean, median of AS2.i1:')
print(i1value)
print('The mean,harmonic mean,geometric mean, median of AS2.i2:')
print(i2value)
```

And the results are as follows:



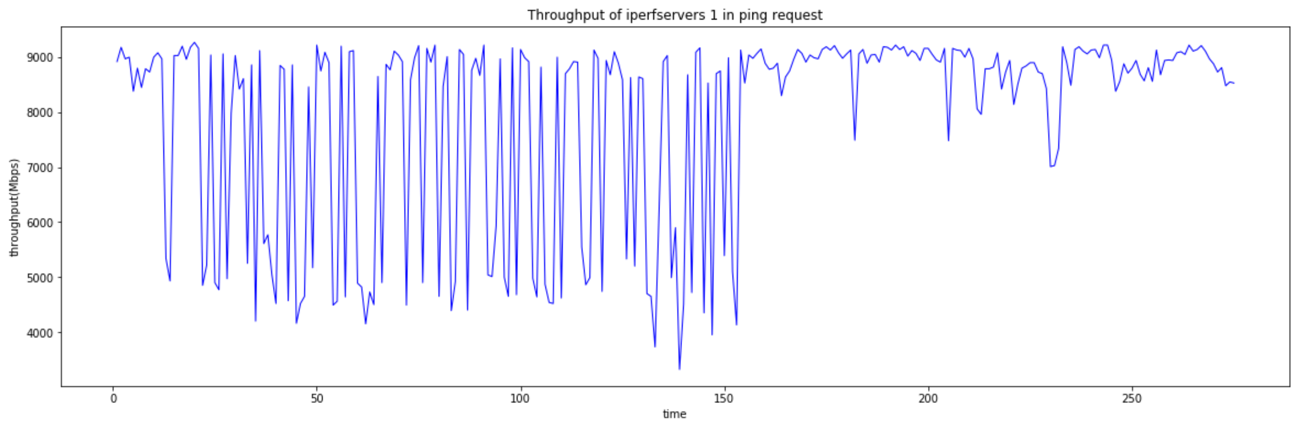
```
The mean,harmonic mean,geometric mean, median of AS2.i1:
[7915.0, 7360.667, 7665.013, 8870.0]

The mean,harmonic mean,geometric mean, median of AS2.i2:
[46.449999999999996, 43.27127, 45.34885, 48.6]
```

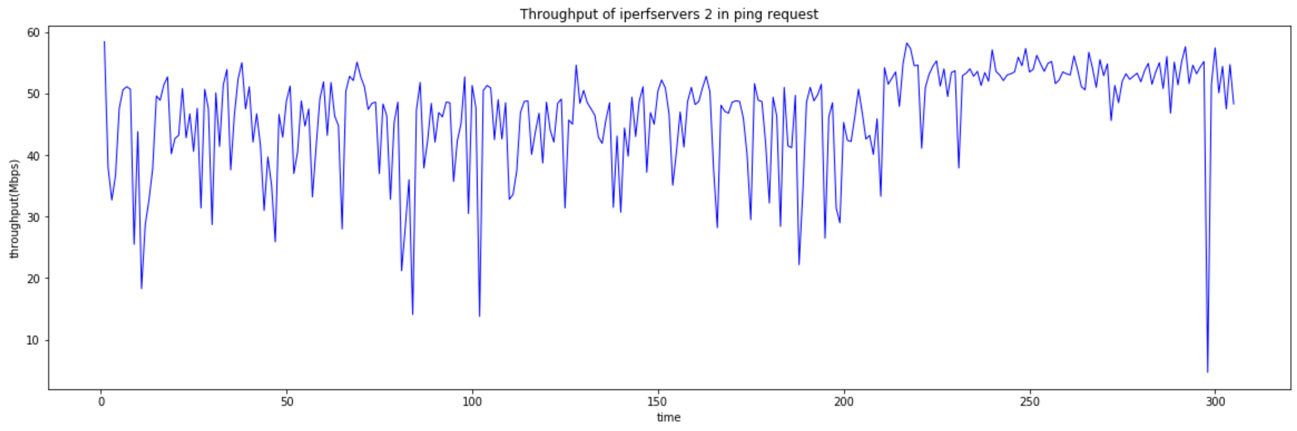
Dataset	mean (Mbps)	harmonic mean (Mbps)	geometric mean (Mbps)	median (Mbps)
AS2.i1	7915	7360.67	7365.01	8870
AS2.i2	46.45	43.27	45.35	48.6

3.4 Thoughput time series

• Plot time series of each data set AS2.x. Consider appropriate scaling for comparison. Any observations for e.g. diurnal patterns?

The throughput time series plots are as follows:





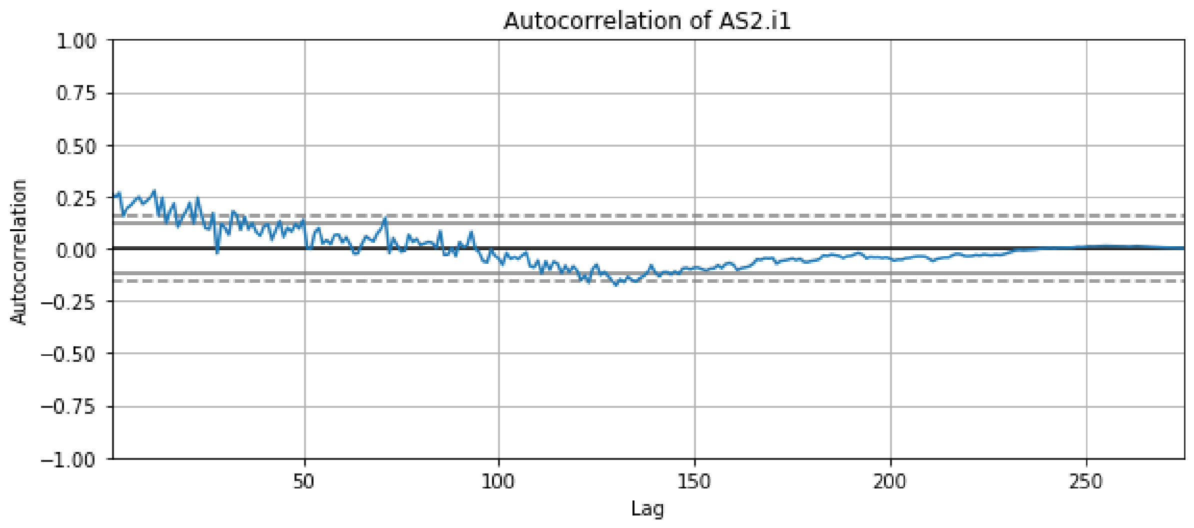
Observations:

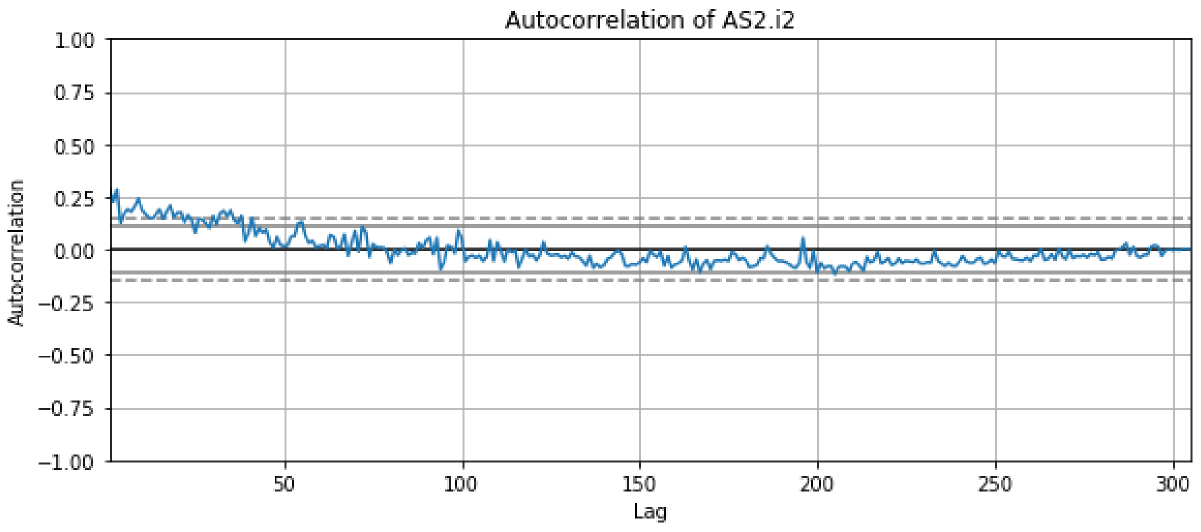
For i1 dataset, the diurnal patterns are easy to find in .9.28-10.08, where there have apparent fluctuations in throughput, while in 10.28-11.04 it is not so apparent to observe, where it remains stable and occasional big variations can be seen.

For i2 dataset, the situation is quite similar to the i1 dataset to some degree, while in 10.28-11.04 the throughput becomes higher than in the previous timeframe.

• Make autocorrelation plot on AS2.x data sets. Any observations? Compare also to 3.2.

The code used is similar to that used above, so here are just the results:





Observations:

As we can see from the plots, the autocorrelation values for both datasets are not so optimistic and they fall to around 0 quite early. So as far as I am concerned, there is not much autocorrelation in throughput datasets.

As for compared with 3.2, the autocorrelation pattern is quite similar for i1 dataset for latency and throughput, both fall from a positive value to below 0, and stay under 0 then gradually approaching 0. So there may exist some correlation between latency and throughput.

Conclusion

Discuss on conclusion on Task 3 for at least following topics:

• Describe the system your made measurements from measurement. What kind of impact it had for measurements?

The measurement setup for my measurements is as follows:

For latency measurements:

	name server	research server	iperf server
1	a.cctld.us	pna-es.ark.caida.org	ok1.iperf.comnet-student.eu
2	b.cctld.us	per-au.ark.caida.org	blr1.iperf.comnet-student.eu
3	c.cctld.us	cjj-kr.ark.caida.org	N/A
Measurement type	DNS query/ICMP echo request	5 ICMP echo requests	5 ICMP echo requests
Frequency	Once an hour	Every 10 minitues	Every 10 minitues
Configuration	Sending minute: 875552%60=32 Add time stamp to ping using -O and -D options	Sending minute: 875552%10=2	Sending minute:875552%10=2 Randomly choose port for each test in iperf;

For throughput measurements:

	By special measurement tool
Tools	***iperf3***
Frequency	Once an hour
Configuration	Randomly choose port for each test in iperf; Option *-t 10* for 10 seconds Option *-R* set client as reciever

The scripts and crontab commands are quite similar to those used in Assignment 2, so it will be meaningless to put it here again. However, this measurement system, including its request time point, request frequency or different options in those commands, would have some different effect on the measurement results. For example, the time point of ping request for iperf servers may should be different, in case that those requests conflict with each other, which might result in lost packets or other problems.

• Did there exists some correlation between path length (number of routers, it can be check with traceroute and/or with TTL value of ICMP Echo Responses) and measurement stability? If you happened to record also TTL value, did it change over time?

I managed to extract TTL value from different datasets using this command:

```
gawk '$7~/^ttl=.*/' *.txt | awk '{print $7}' > ../ttl.txt
```

And the results are as follows:

datasets	AS1.n1	AS1.n2	AS1.n3	AS1.r1	AS1.r2	AS1.r3	AS1.i1	AS1.i2
TTL	55/56	50/53	55/56	44-46	52	47/48	60	45

And according to the box plots and time series plots, it is found that when the TTL values are smaller (less number of routers), the measurement shows better stability than those has larger TTL values. For example, for both latency and throughput, the fluctuations in time series plots are smaller in AS1.i2 than in AS1.i1. It is also the same as AS1.r*N datasets.

• Did throughput and latency have any correlation?

And according to the box plots, time series plots, and autocorrelation plots of throughput and latency for both AS1.i2 and AS1.i1, it could be concluded that higher throughput tend to have higher latency, since the time series and autocorrelation plots for both throughput and latency have similar trends with each other.

Final conclusions

After you have completed the Task 1-3, you are now almost done. Based on these tasks, answer the following questions.

• How was your own traffic (Task 2) different from the data provided (Task 1)? What kind of differences you can identify? What could be a reason for that?

First, the data amount of both datasets is quite different. My own traffic data only contains the traffic for 4 others, while the data provided from Task 1 contains traffic for 24 hours, which could be more resourceful than mine.

Second, for traffic volume, there are more recognizable patterns in data provided in Task 1, since there are more apparent variations in the traffic volume plot of data provided. However, the traffic volume trends in my own traffic data is relatively monotonous. And combined with the results of most common port numbers for both datasets, the reason for the difference in this part might be that these two networks were used for different purposes or the behaviors in the networks were different, since the network in Task 1 might be used for data transmission while my network was mainly used for playing Youtube videos when capturing the traffic data.

Third, the number of users was different in both datasets. There were much more OD-pairs in provided data in Task 1 than my own captured data. But the flows and bytes for per user of my own data were somehow a lot more than those of data provided, which I assumed the reason might also be the purposes of the networks, since when watch videos there were more packets and flows than normal file transmissions.

- **Comparing RTT latency about TCP connections (2.10), were active latency measurements around the same magnitude or was another much larger than the other?**

The results of 2.10 are as follows:

```
>   View(rtt0)
> colnames(rtt0)<-c("src to dst","dst to src")
> summary(rtt0[,1])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0    24.5    42.0   124.9   234.4   622.8
> sd(rtt0[,1])
[1] 125.8001
> summary(rtt0[,2])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   0.100   4.203  10.200  69.300
> sd(rtt0[,2])
[1] 6.200801
```

Note that the **rtt0[,1]** represents the round-trip time of source ip to destination ip, while **rtt0[,2]** represents the round-trip time of destination ip to source ip.
According to the results, we could conclude that: the values in active latency measurements varied quite much in both directions, since the difference between the 3rd quantile and max value was quite large, and it was also the same with median and mean values. Also, the standard variance, it was almost the same with or even bigger than the mean value of both directions, which meant that the deviation among the data was quite big.

- **How do you rate complexity of different tasks? Was some tasks more difficult or laborious than others? Did data volume cause any issues with your analysis?**

As far as I am concerned, Task 1 should be the most difficult or laborious among the three tasks, since it is the first one and many small tasks are new to me, and the second place should be Task 2 and third place was Task 3. However, in Task 2 there are a lot of small tasks that has the same requirement as in Task 1, so I could use the methods devised in Task 1 to finish Task 2, in which case there were less new works in Task 2. As for Task 3, the scripts used were nearly the same with those used in Assignment 2-5, so there were quite less new works needed to be done.
Data volume did somehow cause a bit difficulty in analyzing the diurnal patterns. For one thing, the super large data files in Task 1 made it extremely slow to process them and compute the needed outcomes. For example, in Task 1.3 (OD-pairs), it took the VDI Nvidia server 6 hours to finish the computing in Python scrips. So it might need algorithm optimization or server with better computing performance. For another, the data used in Task 3 were actually collected from two timeframes, each one for about 7-8 days. But fortunately, there were still some recognizable patterns in the plots and results.