Project 1 report

I. Implementation and Code Overview

To run the script there are two options as follows:

```
$python3 bottleneck_link_rate_measurement_real.py <lowbound> <upperbound>

$python3 bottleneck_link_rate_measurement_real.py
```

Disclaimer: The code only runs with python 3.7 and upwards due to the use of time.time_ns

If the upper bound is lower than 200 or the upper bound higher than 1500 these default values will be used.

Such s achieved by reading and testing the arguments.

```
lower_bound = 200
upper_bound = 1500

try:
    script, first, second = argv
    if(int(first) > int(second)):
        print("Upper bound cannot be higher than lower bound, using default values")
    elif(int(first) < 200):
        print("Lower bound cannot be lower than 200, using default values")
    elif(int(second) > 1500):
        print("Upper bound cannot be higher than 1500, using default values")
    else:
        lower_bound = int(first)
        upper_bound = int(second)
except:
    print("No args, using default values")
```

In order to connect to the server a socket is created with the corresponding address and port.

```
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
address = "129.187.223.200"
port = 2000
```

As requested, the connection is tested by sending a packet containing a "TEST" string and receiving a socket in response.

```
packet = "TEST".ljust(400);
sock.sendto(packet.encode(),(address,port));
data = sock.recv(400);
print('message received:', data);
```

Then, two packets are sent back to back and the receiving time of its response recorded in order to calculate their bottleneck. To calculate such the packet length is calculated by multiplying the length of the packet by 8 bits and adding the UDP overhead (64 bits) and by converting the delay from nanoseconds to seconds.

```
packet = "".ljust(1000);
sock.sendto(packet.encode(),(address,port));
sock.sendto(packet.encode(),(address,port));
data = sock.recv(1000);
ts1 = int(time.time_ns());
data = sock.recv(1000);
ts2 = int(time.time_ns());
delay= abs(ts2 - ts1);
print('The delay for the two small back to back packets is ',  delay, 'nanoseconds');
if (delay == 0):
    bottleneck = 0;
else:
    delay = delay*10**(-9);
    bottleneck = len(packet)/delay;
print('The bottleneck is ',  bottleneck);
```

In order to calculate the bottleneck for different sizes of packets a cycle is performed and for each packet size a cycle of 200 measures for the delay is performed. In this cycle not only is the bottleneck calculated but also the mean bottleneck for each packet size, the delays that are 0 for each packet size and the amount of times that the timeout is bigger then 2 for each packet size.

```
r1 = 200;
r2 = 1500;
pace = int((r2 - r1)/5)

bottlenecks = [[],[],[],[],[],[]];
bottlenecks_means = [];
delays = []
sock.settimeout(2);
n = 0;

timeouts = []

for j in range (r1,r2+1,pace):
    packet = "".ljust(j);
    packet_size = len(packet)*8 + 64; #packet len to bits + UDP overhead
    bottleneck_sum = 0;
    delay_0 = 0;
    time_out = 0;
    i = 0;
    while i <= 200:
        #print(i);
        sock.sendto(packet.encode(),(address,port));
        sock.sendto(packet.encode(),(address,port));
        data = None;
        read1 = True
        read2 = True
        try:
            data = sock.recv(j);
        except socket.timeout:
            read1 = False
            time_out = time_out + 1;
        ts1 = int(time.time_ns());
        try:
            data = sock.recv(j);
        except socket.timeout:
            read2 = False
            time_out = time_out + 1;
        ts2 = int(time.time_ns());

        if(i >= 0 and (not read1 or not read2)):
            continue

        delay= abs(ts2 - ts1);
        if (delay == 0):
            i = i - 1;
            delay_0 = delay_0 + 1;

        else:
            delay = delay*10**(-9);
            bottleneck = packet_size/delay;
            bottlenecks[n].append(bottleneck);
            bottleneck_sum = bottleneck_sum + bottleneck;
        i = i + 1;
    bottlenecks_means.append(bottleneck_sum/201);
    delays.append(delay_0);
    timeouts.append(time_out);
    n = n + 1;
    #print(n);
    #print(j);
```

Since we are dealing with a real system, the are many times in which a timeout leaves the program hanging or there is too much cross traffic and such is prevented for timeouts bigger than 2 seconds by trying the connection again. The same try again principle is applied when the delays are equal to 0.

Then, the bottlenecks for each packet size and the mean bottlenecks per packet size are plotted as well as the timeouts that occur at each packet size and the delays that are 0 for each packet size.

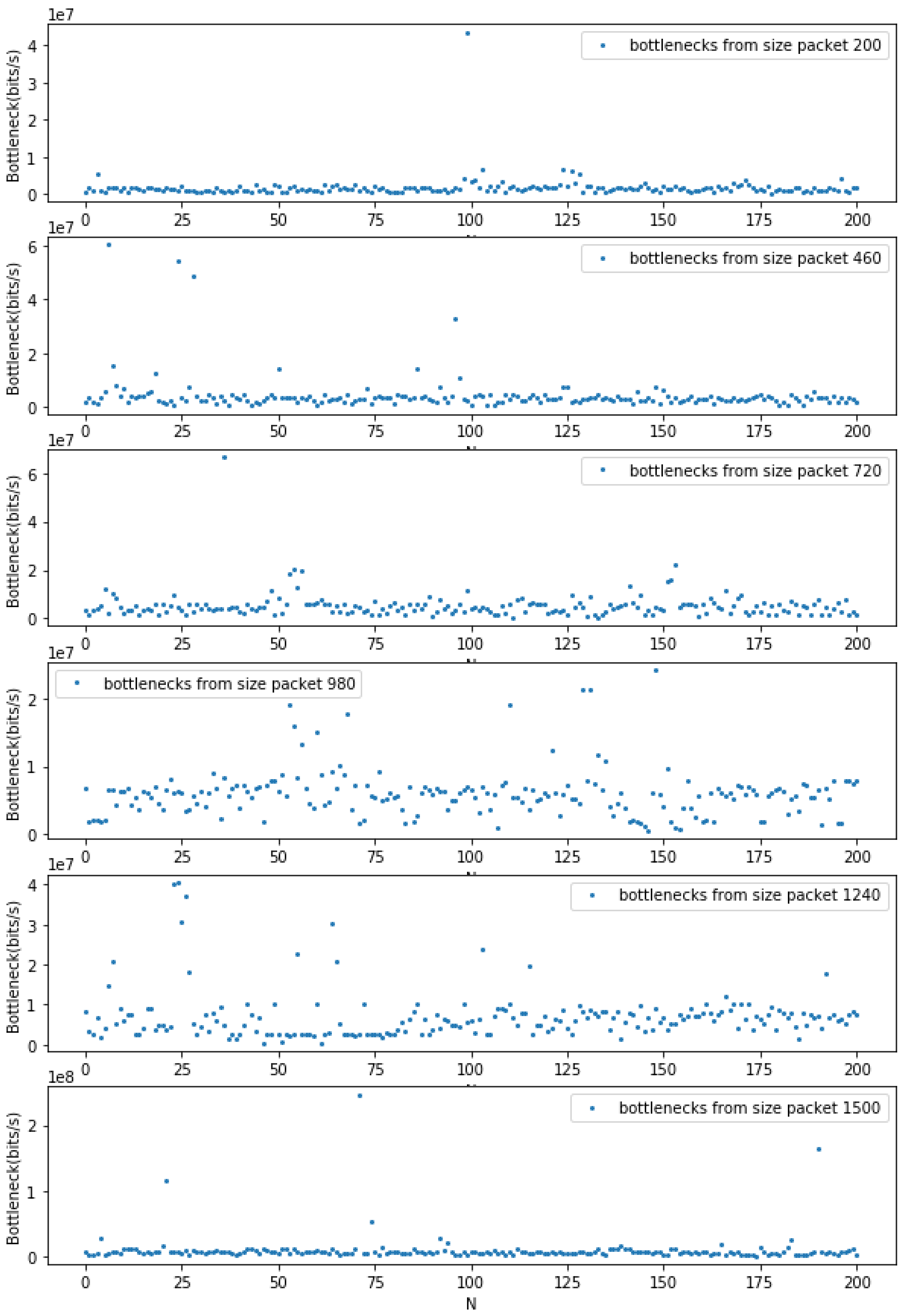
```
#plot the bottlenecks for each packet size
x = list(range(0,201))
fig = plt.figure(figsize=(12,30));
#p = [];
for i in range(len(bottlenecks)):
    #print (len(bottlenecks[i]));
    p = r1 + i*pace;
    l = 'bottlenecks from size packet ' + str(p);
    y = bottlenecks[i];
    ax = fig.add_subplot(6, 1, i + 1)
    ax.plot(x,y, 'o', linewidth=2, markersize=2, label=l);
    ax.set_ylabel('Bottleneck(bits/s)')
    ax.set_xlabel('N')
    ax.legend()
    #print (i);

#plot the bottleneck mean for each packet size
x = list(range(r1,r2+1,pace));
y = bottlenecks_means;
fig = plt.figure(figsize = (10,10));
ax = fig.add_subplot(1, 1, 1)
```

```
l = "Mean Bottleneck link per packet size"
ax.plot(x,y, linewidth=4, markersize=6, label=l);
ax.set_ylabel('Mean bottleneck(bits/s)')
ax.set_xlabel('Packet Size(bytes)')
ax.legend()

#plot the timeouts for 2s
y = timeouts;
fig = plt.figure(figsize = (10,10));
ax = fig.add_subplot(1, 1, 1)
l = "Timeouts per packet size"
ax.plot(x,y, 'o', linewidth=4, markersize=6, label=l);
ax.set_ylabel('N')
ax.set_xlabel('Packet Size')
ax.grid()
ax.legend()

#plot the delays that were 0 for each packet size
y = delays;
fig = plt.figure(figsize = (10,10));
ax = fig.add_subplot(1, 1, 1)
l = "Delays equaling 0 for each packet size"
ax.plot(x,y, 'o', linewidth=4, markersize=6, label=l);
ax.set_ylabel('N')
ax.set_xlabel('Packet Size')
ax.grid()
ax.legend()
```

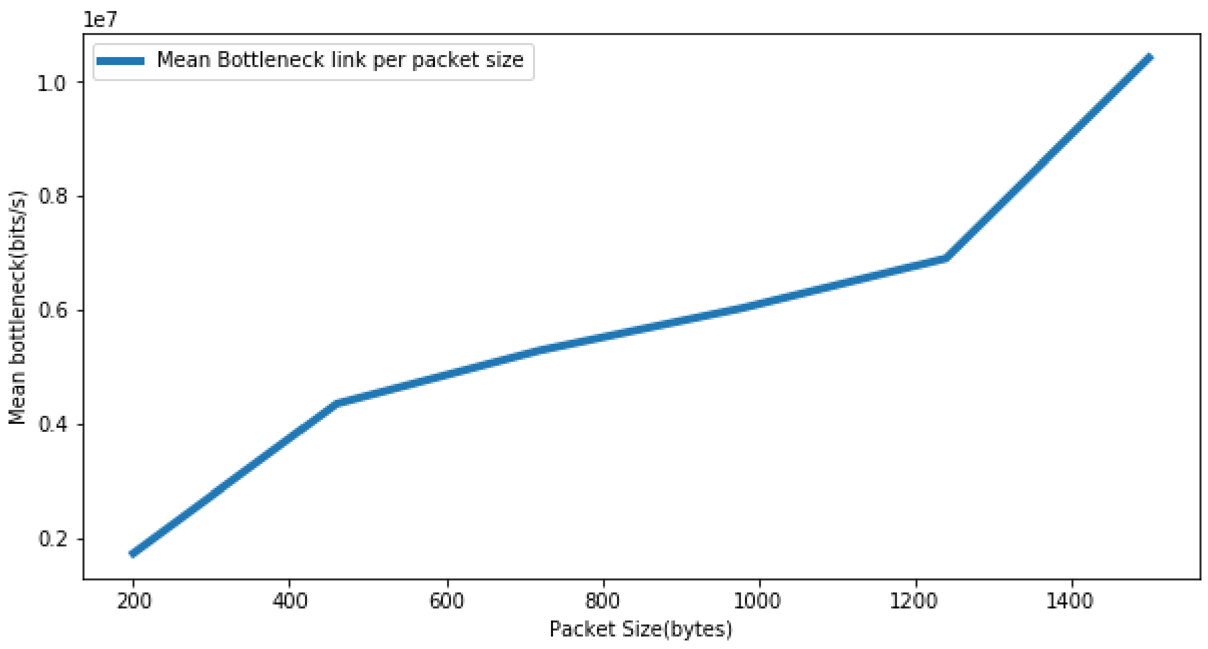
II. Figures and Results

Running the code with the limits of the packet sizes being, respectively 200 and 1500 and having a pace o 0.25 times the difference of those the produced plots are as follows.

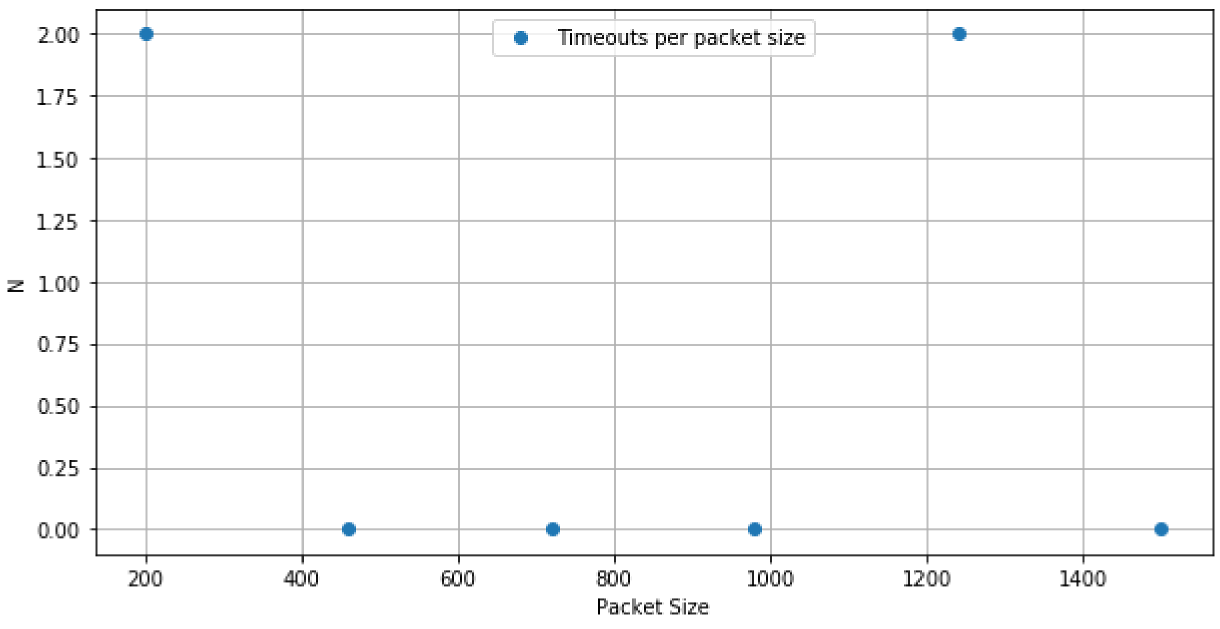
In this first picture we can analyse the different bottlenecks for 201 attempts at measuring the delays for each packet size. We can analyse from the picture that they are not constant and this can be due both to cross traffic as well as a slower internet link connection.



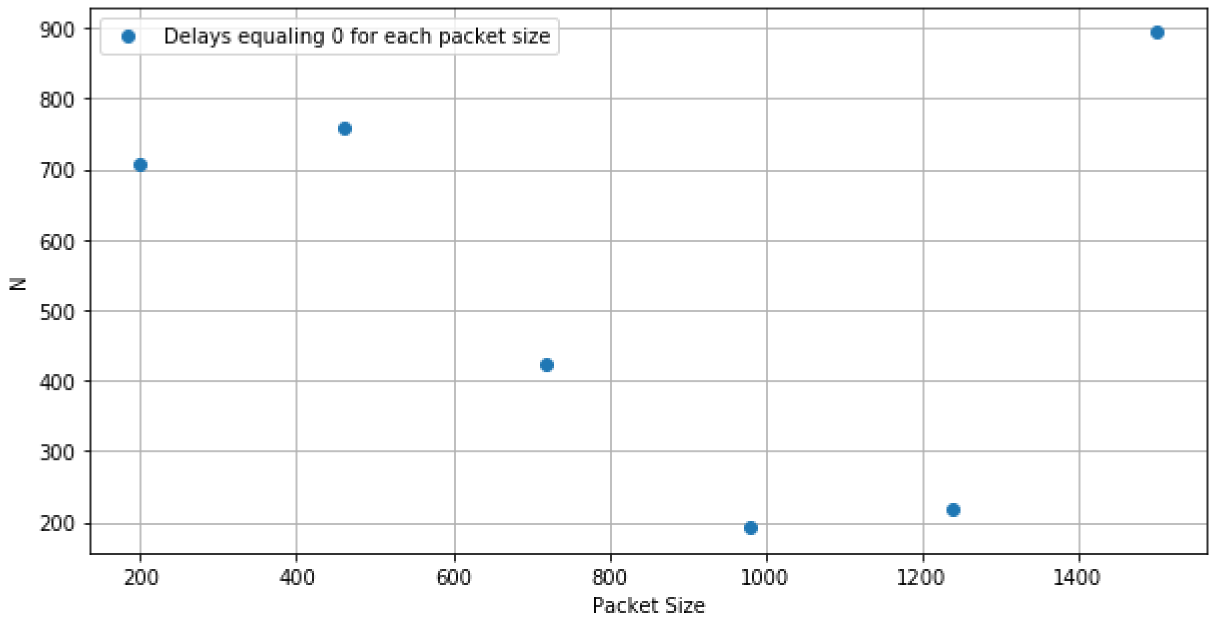
It is also quite clear that as the packet size augments, so do the bottlenecks and that can be see in the next plot:



One can also evaluate the usefulness of using a socket.timeout to prevent the program from hanging in the next plot:



And the usefulness of not including delays equalling 0 in the statistics in the next one where we can also analyse that usually they decrease with the packet size as the bottleneck is lower but also due, mostly, to the imprecision on the measurements. However we can also check that a lot of delays come back as 0 at 1500bytes for it is at the limit of what the network server can handle.



III. Questions f.:

Q: In order to get the result as accurate as possible, what is the suitable packet size in your experiment?

For the result to be accurate one must not exceed the 1500 bytes packet size that the network can handle but also, it is convenient to have reasonable packet sizes (from 200 bytes) as to have enough precision measuring it.