ECS 152A: Computer Networks

Spring 2022

Project 1

(150 points)

Due Date: Monday May 9th, 2022 (by 11:59 PM – before midnight)

Team: The project is to be done in a team of at most 2 students. You *cannot* discuss your code/data with other classmates (*except* your project partner)

All submissions (including your code) will be checked for ***plagiarism*** against other submissions as well as the public Internet. Plagiarized submissions will be entitled to ***zero*** points.

Project 1 consists of two parts:

1. Implementing UDP Ping Client and Iperf Client
2. Implementing HTTP Client

Part 1: Implementing UDP Ping client & Iperf Client (*50 points*)

---

1. UDP Ping Client Code file name: udp_ping_[name1]_[student_id1]_[name2]_[student_id2].py (*5 points*)

You have to implement a "UDP ping client program" in Python version >= 3. Note this shall not be the same as Ping studied in discussion sessions, which was ICMP-based Ping.

Your UDP client program should send a *ping* to the UDP server (let's call it ***Pong Server***).

Following are the details about the ***Pong Server***:

- ❖ Server IP: 173.230.149.18
- ❖ Server Port: 12000
- ❖ Message received from the server: PING

Details about ***Ping Client***:

- ❖ Message sent to the server: ping
- ❖ Number of pings to send: 10

Your UDP Ping client will send 10 pings to the pong server. Your program should count the number of times the message is sent, stop and calculate the following statistics. You will compute and print the following:

❖ The total number of ping/pong packets lost **(*1 points*)**
❖ The total number of pong packets successfully received **(*1 points*)**
❖ RTT of each ping **(*1 points*)**
❖ Overall minimum, maximum, average, and sum of RTT for 10 pings. **(*1 points*)**

Consider RTT (Round Trip Time) as the time duration between when the client sends a ping and client receives a pong from the server.

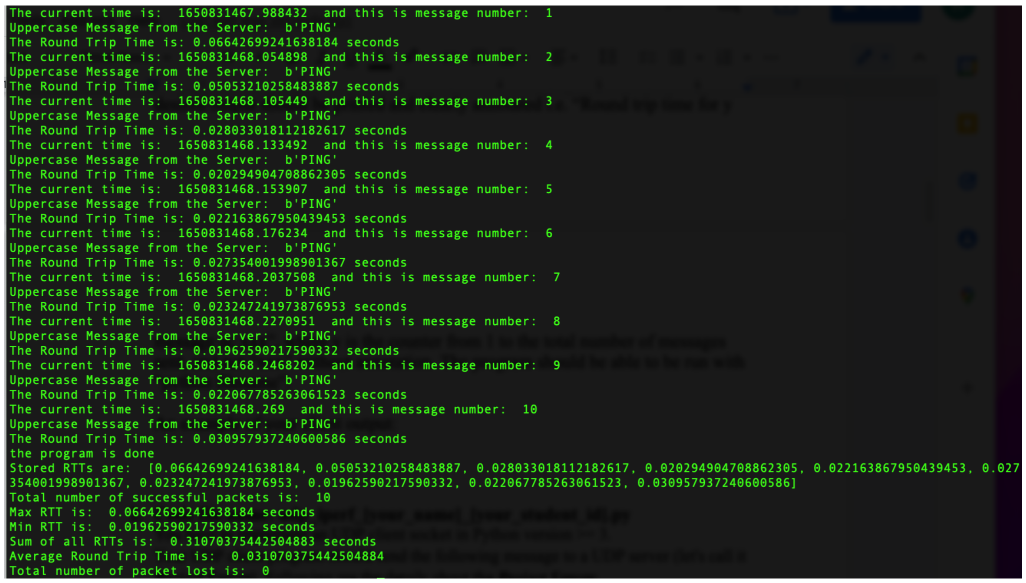
In your program, you will implement a timeout, i.e. if your client does not receive the message from the server in the "timeout_seconds" time, then it will sleep for a duration computed by the Exponential Backoff function provided below, and then retry the task. Every time the client retries, the time to attempt a new try will increase exponentially until you reach 10 minutes. **(*1 points*)**

Implement Exponential Backoff in your ping client to avoid congestion:

❖ timeout_seconds = 10 seconds
❖ Exponential Backoff function:
timeout_seconds = (timeout_seconds * $2^{\text{timeout\_count}}$ + random.uniform(0, 1))
❖ The upper limit of timeout: 10 minutes

All of the desired output should be printed and clearly annotated i.e. "Round trip time for ping y is: xxx", where y is the counter from 1 to the total number of pings sent. Do not print numbers without annotation. The program should be able to be run with the command "python3 <filename>.py" without needing any additional command line input.

An example output for your ping client:



```
The current time is:  1650831467.988432  and this is message number:  1
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.06642699241638184 seconds
The current time is:  1650831468.054898  and this is message number:  2
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.05053210258483887 seconds
The current time is:  1650831468.105449  and this is message number:  3
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.028033018112182617 seconds
The current time is:  1650831468.133492  and this is message number:  4
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.020294904708862305 seconds
The current time is:  1650831468.153907  and this is message number:  5
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.022163867950439453 seconds
The current time is:  1650831468.176234  and this is message number:  6
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.027354001998901367 seconds
The current time is:  1650831468.2037508  and this is message number:  7
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.023247241973876953 seconds
The current time is:  1650831468.2270951  and this is message number:  8
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.01962590217590332 seconds
The current time is:  1650831468.2468202  and this is message number:  9
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.022067785263061523 seconds
The current time is:  1650831468.269  and this is message number:  10
Uppercase Message from the Server:  b'PING'
The Round Trip Time is: 0.030957937240600586 seconds
the program is done
Stored RTTs are:  [0.06642699241638184, 0.05053210258483887, 0.028033018112182617, 0.020294904708862305, 0.022163867950439453, 0.027
354001998901367, 0.023247241973876953, 0.01962590217590332, 0.022067785263061523, 0.030957937240600586]
Total number of successful packets is:  10
Max RTT is:  0.06642699241638184 seconds
Min RTT is:  0.01962590217590332 seconds
Sum of all RTTs is:  0.31070375442504883 seconds
Average Round Trip Time is:  0.031070375442504884
Total number of packet lost is:  0
```

2. **UDP Iperf Client**
   Code file name:
   udp_iperf_[name1]_[student_id1]_[name2]_[student_id2].py (*15 points*)

You have to implement a "UDP iperf client program" in Python version >= 3. Your UDP client program should send a message to the UDP server (let's call it ***Iperf Server***).

Following are the details about the ***Iperf Server***:

❖ Server IP: 173.230.149.18
❖ Server Port: 5005
❖ Data server will send: a txt file

Details about ***Iperf Client***:

❖ Message your UDP client program will send: ping
❖ Target file size: 1300000 bytes
❖ Create a txt file to write the received data to

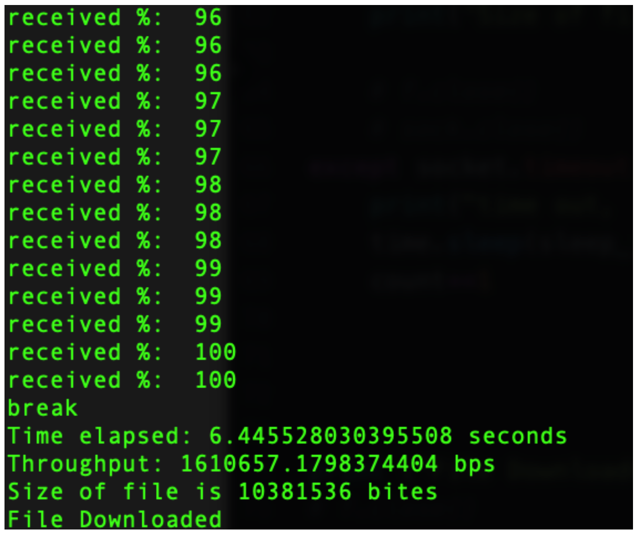
In this iperf client file you will program a UDP Iperf client to send the given message to the Iperf Server. As a response, your client will receive a text file from Iperf Server. Your UDP client should stop receiving the data from the server when target file size is reached.

You will compute and print the following:

❖ Print the percentage of data received **(*3 points*)**
❖ Print the size of the file (bite) **(*3 points*)**
❖ The total time it took to download the whole file (second) **(*3 points*)**
❖ The throughput (the size of the received file / the total time it took to receive it) **(*6 points*)**

Your Iperf client will also implement the exponential timeout as explained for the ping client. The Iperf client program should also be able to run with "python3 <filename>.py" without needing any additional command-line input.

An example of your client output:



```
received %:  96
received %:  96
received %:  96
received %:  97
received %:  97
received %:  97
received %:  98
received %:  98
received %:  98
received %:  99
received %:  99
received %:  99
received %:  100
received %:  100
break
Time elapsed: 6.445528030395508 seconds
Throughput: 1610657.1798374404 bps
Size of file is 10381536 bites
File Downloaded
```

3. Report:

udp_[name1]_[student_id1]_[name2]_[student_id2].pdf (*30 points*)

At the beginning of the page, specify the following:

1. Full Name of student 1 (Student ID) (Discussion Group)
2. Full name of student 2 (Student ID) (Discussion Group)
3. Name of the code files submitted

Report the following details related to the UDP socket in your report:

Question I: Ping

1. Attach a screenshot of the execution output of your udp_ping.py. In the screenshot, you have to include the line where the file is executed, i.e. root@xxx: python udp_ping_[your_name]_[your_student_id].py, and not just the output of the code. **(*2 points*)**
2. The total number of ping packets delivered and lost? **(*4 points*)**
3. What is the minimum, maximum, average, and sum of ping round trip times (RTT)? **(*4 points*)**
4. Find the messages your UDP ping client sent and received to/from UDP Server on Wireshark using filters. Attach a screenshot (including the filters applied to generate the required output) and briefly describe. **(*5 points*)**

Question II: Iperf

5. Attach a screenshot of the output of your udp_iperf.py. In the screenshot, you have to include the line where the file is executed, i.e. root@xxx: python udp_iperf_[your_name]_[your_student_id].py, and not just the output of the file. **(*2 points*)**
6. How many packets did you receive? Identify, attach a screenshot (including the filters) of Wireshark and briefly describe. **(*5 points*)**
7. What is the total time it took to receive the file? **(*4 points*)**
8. What is the throughput? **(*4 points*)**

Part 2: Implementing HTTP Client (*100 points*)

You have to implement an HTTP client for a pared-down version of **HTTP/1.1 using TCP sockets**. This project should be completed using Python version >= 3.

Your client program should craft and perform HTTP requests to the HTTP server (let's call the HTTP Server hosting the HTML file as ***Project Server***).

Following are the details about the ***Project Server***:

- Server IP: 173.230.149.18
- Server Port: 23662
- Server Root Directory contains: ecs152a.html (HTML file)
- Project Server implements and returns the following standard HTTP headers:
    - The **Connection** header
    - The **Content-length** header
    - The **Content-type** header

  Note: The Project Server implementation of Content-length does not consider headers.
- Project Server returns the desired response only if the HTTP request is made with the correct format of the custom header.
    - HTTP requests from your client program should include the following custom header: "**X-Client-project**"
    - Project Server will recognize this custom header only if its value is "*project-152A-part2*".

Implementation Details:

Your task is to implement and compare the following two flavors of HTTP discussed in the class. You should implement all the details mentioned in steps 1-5 for both of these flavors and as independent Python programs. All versions should be non-pipelined, non-parallel, and non-multiplexed.

1. Non-persistent HTTP
2. Persistent HTTP

Your code should not have any command line inputs (hardcode everything you require with interpretable variable names and/or comments). It should run with the command: "python3 <filename>.py". Following are the code files to be submitted.

A. **non_persistent_http_[name1]_[student_id1]_[name2]_[student_id2].py (*25 points*)**
B. **persistent_http_[name1]_[student_id1]_[name2]_[student_id2].py (*25 points*)**

1. Your HTTP client should be programmed to first request ***ecs152a.html*** from the Project Server. The response should be saved as an HTML file (.html extension) with the same name

as received from the Project Server. (***5 points***)

2. Next, assuming that HTTP clients perform a sequential load of resources, parse the HTML response, and issue subsequent requests to fetch and download/save all the images referenced in the HTML file (consider all images with **`<img>`** tag). You can use HTML parsers (e.g., HTMLParser or BeautifulSoup in Python) to parse the HTML. You should verify the success of the image downloaded by opening the saved images, which are typically 9x6 pixels. All images should be submitted as a zip file as explained in the Submission Page. (***5 points***)

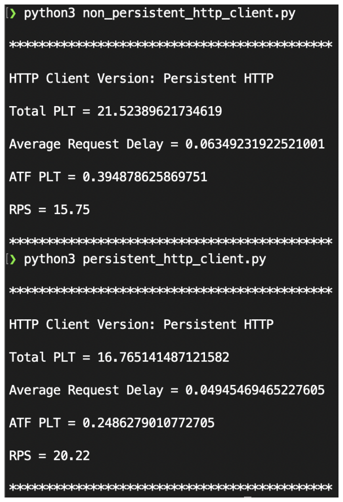
3. Implement and include the necessary standard and custom HTTP headers in your requests.

4. By default `socket.recv(buffer_size)` executes in BLOCKING MODE and blocks the further execution of the program indefinitely in certain situations (like if it incorrectly senses that some data is still pending to be received). To avoid blocking, your HTTP client should monitor the size of the data received from the Project Server. For both the HTTP flavors, use `buffer_size` (i.e., the size of the receiving buffer) as **4096**.

5. Compute the following performance metrics:

    a. The **Total Page Load Time (PLT)** of *ecs152a.html*
    b. **Above the Fold** **Page Load Time (ATF PLT)** of *ecs152a.html.* Refer to the definition in the "In web design" section of the above link, if you are not familiar with "above the fold".
    c. **Average Request Delay**: Average time between making a request and receiving the response. If any request requires establishing a new connection then that time should also be included while computing the request delay.
    d. **Requests per Second (RPS)**

The following is an example of the expected output that your codes should print on the terminal when executed:



```
❯ python3 non_persistent_http_client.py

*********************************************

HTTP Client Version: Persistent HTTP

Total PLT = 21.52389621734619

Average Request Delay = 0.06349231922521001

ATF PLT = 0.394878625869751

RPS = 15.75

*********************************************
❯ python3 persistent_http_client.py

*********************************************

HTTP Client Version: Persistent HTTP

Total PLT = 16.765141487121582

Average Request Delay = 0.04945469465227605

ATF PLT = 0.2486279010772705

RPS = 20.22

*********************************************
```

Project Server has limited compute. As a result, your HTTP Client may take a longer time to complete its execution during Peak hours. Based on the time the requests are made to the Project server at Peak hours v/s Non-peak hours, you will see different outputs for the same version of your code. Hence, you should execute codes for both the flavors of HTTP during the same time for reporting your numbers and associated output. However, while testing and/or building your individual codes, you can run them at any time. At peak hours, you may observe the performance metrics as follows:

- Connection: Non-Persistent
  RPS: 0.7, Total PLT: 483.12474942207336, ATF PLT: 4.5390403270721436
- Connection: Persistent
  RPS: 8.85, Total PLT: 38.30596137046814, ATF PLT: 4.459162950515747

Part 2 Project Report Details

Report file name: http_[name1]_[student_id1]_[name2]_[student_id2].pdf (*40 points*)

In the beginning of the page, specify the following:

1. Full Name of student 1 (Student ID) (Discussion Group)
2. Full name of student 2 (Student ID) (Discussion Group)
3. Name of the code files submitted
4. Screenshot of the Terminal Window displaying the output as shown in the sample screenshot above for both the codes.

Analyze and answer the following questions in the report with detailed justifications and explanations along with supporting calculations, execution screenshots and Wireshark screenshots wherever asked:

Question I: TCP Protocol

Report the following details related to the *TCP Protocol*:

a. The total number of TCP sockets used by both flavors of your HTTP Client implementations to perform all the steps. Justify your answer. **(*5 points*)**
b. The total number of ports used to perform all the steps. Why? **(*3 points*)**

Question II: HTTP Protocol

Answer the following questions with respect to *HTTP Protocol* for any flavor of HTTP:

a. The total number of HTTP server(s) that your client program will connect to in order to perform all the steps. Insert a screenshot of Wireshark showing the contacted servers along with the requests sent to it. **(2 points)**
b. Report Host Name, IP4 or IPv6 Address, and the total number of GET requests issued to each of the server(s). Verify these details from Wireshark by attaching the screenshot. **(*6 points*)**
c. What is the total number of images downloaded by your HTTP Client? **(*2 points*)**
d. Report the total size of *all resources* (including the HTML file) fetched using GET requests in *Megabytes*. **(*2 points*)**

e. What happens if you perform HTTP requests with *Custom Header:* ***X-Client-project*** to some server other than the Project Server. What status code and response will you receive? Explain with an example from the HTML file. **(*3 points*)**

Question III: HTTP Performance

Answer the following questions about *HTTP performance.* Ignore the time taken to parse the HTML and close the connection.

a. Report the *Above the fold* page load time metric of ecs152a.html for both flavors of HTTP clients. What resources does this time constitute of? **(*4 points*)**
b. Report *Total* page load time of ecs152a.html for both flavors. Which flavor has a higher Total PLT? Why? Suggest at least one way in which your HTTP client program can improve the total PLT? **(*4 points*)**
c. Which HTTP flavor does the Project Server implement by default - Persistent or Non-persistent? How did you reach your conclusion? **(*3 points*)**
d. Change `buffer_size` in `socket.recv(buffer_size)` function to **256** and report the Total PLT for either Persistent or Non-persistent HTTP Client. Compare it with the Total PLT observed with `buffer_size` of **4096**. Which option is better, explain? **(*6 points*)**

Testing Environment:

All submissions will be tested on Python 3+.

Late Submission Policy:

No late submissions are allowed. However, if you barely miss the deadline, you can get partial points upto 24 hours. The percentage of points you will lose is given by the equation below. This will give you partial points up to 24 hours after the due date and penalizes you less if you narrowly miss the deadline.

$$Total\ Marks\ you\ get\ =\ (Actual\ Marks\ you\ would\ get\ if\ NOT\ late)\ \times\ \left[1\ -\ \frac{hours\ late}{24}\right]$$

Late Submissions (later than 24 hours from the due date) will result in zero points, *unless you have our prior permission or documented accommodation*.

——————————————— *Best of luck* ———————————————

Submission Page

Submission Checklist: Submit the following documents on Canvas:

1. Replace placeholder text below with actual file names and names of team members before printing. Include this page as a scanned PDF after signing the plagiarism statement below. *Your submission will be rejected without inclusion of this signed statement.*

Part 1: ***50 points***

2. **UDP Ping Client Program (.py) (*5points*)**
   udp_ping_[name1]_[student_id1]_[name2]_[student_id2].py
3. **UDP Iperf Client Program (.py) (*15 points*)**
   udp_iperf_[name1]_[student_id1]_[name2]_[student_id2].py
4. **Report for Part 1 (.pdf) (*30 points*)**
   udp_[name1]_[student_id1]_[name2]_[student_id2].pdf

Part 2: ***100 points***

5. **Non-persistent HTTP Program (.py) (*25 points*)**
   non_persistent_http_[name1]_[student_id1]_[name2]_[student_id2].py
6. **Persistent HTTP Program (.py) (*25 points*)**
   persistent_http_[name1]_[student_id1]_[name2]_[student_id2].py
7. **HTML File Download from Project Server (.html) (5 points)**
   ecs152a_[name1]_[student_id1]_[name2]_[student_id2].html
8. **A Zip file containing all the Images referenced in the HTML File (.zip) (5 points)**
   images_[name1]_[student_id1]_[name2]_[student_id2].zip
9. **Report for Part 2 (.pdf) (40 points)**
   http_[name1]_[student_id1]_[name2]_[student_id2].pdf

I certify that all submitted work is my own work. I have completed all of the assignments on my own without assistance from others except as indicated by appropriate citation. I have read and understand the university policy on plagiarism and academic dishonesty. I further understand that official sanctions will be imposed if there is any evidence of academic dishonesty in this work. I certify that the above statements are true.

Team Member 1:

____________________	____________________	____________________
Full Name (Printed)	Signature	Date

Team Member 2:

____________________	____________________	____________________
Full Name (Printed)	Signature	Date