Policies and Guidelines

- Assignments deadline come with a ***72 hours grace period***. If you submit the assignment during the grace period, it will be accepted with a ***20% penalty***. Submissions after the grace period will be accepted with a ***50% penalty***. ***No submission will be accepted after the last day of classes on Thursday, May 4, 2023, at 11:59 pm***.
- Every project has required submission guidelines. Please read the submission requirements carefully. Any deviations from specifications on projects will result in point deductions or incomplete grades.
- ***Instructors will not 'fix' your code to see what works***. If your code does not run due to any type of errors, it is considered non-functioning.
- ***You cannot resubmit 'fixed' code after the deadline***, regardless of how small the error is.
- If you use the University resources for development and/or testing, then keep in mind that others in the University may need to use the same resources, so please use these resources wisely. You should also follow the rules and restrictions associated with using the University resources.
- Using any external/third-party library and/or framework to implement the project requirements is not permitted.

Academic Honesty Expectations and Violation Penalty

- ***The programming assignments are team assignments; each team cannot have more than two members***. Each team must do the vast majority of the work on its own. It is permissible to consult with other teams to ask general questions, help discover and fix specific bugs, and talk about high-level approaches in general terms. ***Giving or receiving answers or solution details from other teams is not permissible***.
- You may research online for additional resources; however, ***you may not use code explicitly written to solve the problem you have been given***. ***You may not have anyone else help you write the code or solve the problem***. You may use code snippets found online, providing that they are appropriately and clearly cited within your submitted code.
- ***If you use a distributed version control service such as GitHub to maintain your project, you should always set your project repository to private***. If you make your project repository public during the semester or even after the semester, it may cause a violation of the academic honesty policy if other students find and use your solution.

- If you are involved in plagiarism or cheating, you will receive a score of "0" for the involved project for the first offense. You will receive an "F" grade for the course and be reported to the university for any additional offense.
- Students are required to strictly follow the rules and guidelines laid out in the Watson School Student Academic Honesty Code at the following link: http://www.binghamton.edu/watson/about/honesty-policy.pdf
- Please also review Computer Science Faculty Letter to Students Regarding Academic Honesty at the following link: http://www.cs.binghamton.edu/~ghyan/courses/CSHonestyLetterToStudents.pdf
- Cheating and copying will **NOT** be tolerated. Anything submitted as a programming assignment must be the student's original work.
- We reserve the right to use plagiarism detection tools to detect plagiarism in the programming assignments.

Implementation

Part 1: Simple UDP Pinger

In this project, you will learn the basics of socket programming for UDP in Python. You will learn how to send and receive datagram packets using UDP sockets and also how to set a proper socket timeout. Throughout the project, you will gain familiarity with a Ping application and its usefulness in computing statistics such as packet loss rate.

You will first study a simple Internet ping server written in Python and implement a corresponding client. The functionality provided by these programs is similar to the functionality provided by standard ping programs available in modern operating systems. However, these programs use a more straightforward protocol, UDP, rather than the standard Internet Control Message Protocol (ICMP) to communicate with each other. The ping protocol allows a client machine to send a packet of data to a remote machine and have the remote machine return the data to the client unchanged (an action referred to as echoing). Among other uses, the ping protocol allows hosts to determine round-trip times to other machines.

You are given a complete template code for the server-side UDP Pinger. Your task is to write the client-side UDP Pinger.

Before running your client-side ping code, you should compile and run the server-side ping code. In the server code, 30% of the client's packets are simulated to be lost. You should study the server code carefully, as it will help you write your client's ping code. UDP provides applications with an unreliable transport service. Messages may get lost in the network due to router queue overflows, faulty hardware, or other reasons. Because packet loss is rare or even non-existent in typical campus networks, the server echo program in this project injects artificial loss to simulate the effects of network packet loss. The server creates a variable randomized integer that determines whether a particular incoming packet is lost or not.

The server echo program sits in an infinite loop listening for incoming UDP packets from the client. When a packet comes in, and if a randomized integer is greater than or equal to 4, the server creates its response message and sends it back to the client.

You need to implement the following client program. The client should send a ping to the server every ***three seconds*** and terminate the program after ***three minutes***. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network or vice versa. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should get the client to wait up to ***one second*** for a reply. If no reply is received within one second, your client program should assume that the packet was lost during transmission across the network. You will need to look up the Python documentation to find out how to work with timers and how to set the timeout value on a datagram socket.

Specifically, your client program should:

1. Print the ping message and send it using UDP to the server (Note: Unlike TCP, you do not need to establish a connection first since UDP is a connectionless protocol.).
2. Print the response message from the server, if any.
3. Calculate and print the round trip time (RTT), in seconds, of each packet if the server responds.
4. Otherwise, print "Client ping timed out."

The client ping messages in this project are formatted in a simple way. The client message is one line consisting of ASCII characters in the following format:

```
ping,SEQUENCE-NUMBER,TIMESTAMP
```

Where "SEQUENCE-NUMBER" starts at one and increments for each successive ping message sent by the client, and "TIMESTAMP" is the time when the client sends the message.

The server echo messages in this project are also formatted in a simple way. The server message is one line consisting of ASCII characters in the following format:

```
echo,SEQUENCE-NUMBER,TIMESTAMP
```

Where "SEQUENCE-NUMBER" is the same as the sequence number in the client ping message, and "TIMESTAMP" is the time when the server sends the message.

You should only compute the RTT on the client side, and when the sequence number in a ping and an echo messages match.

Part 2: Standard UDP Pinger

Currently, the program calculates each packet's round-trip time (RTT) and prints it out individually. Modify this to correspond to the way the standard ping program works. You will need to print the following at the end of all pings from the client and right before the program terminates:

- The minimum RTT.
- The maximum RTT.
- The total number of RTTs (number of successful ping-echo pairs).
- The packet loss rate (in percentage).
- The average RTTs.

Part 3: Integrate the Standard UDP Pinger into Project-2

Integrate your Standard UDP Pinger implementation into your Project-2 implementation. You should integrate the client side of the UDP Pinger with the proxy server such that the client ping code entirely runs on a separate thread. You should also integrate the server side of the UDP Pinger with the web server such that the server echo code entirely runs on a separate thread.

To properly handle thread termination, you should modify the server echo code such that it doesn't run indefinitely. Instead, the server echo code should terminate if it doesn't receive any ping packet from the client for ***30 seconds***. In this case, the server echo code should print "Server echo timed out." and then end, which should also terminate its thread.

Demonstration

You are expected to demo part 3 of the project across two machines connected to the campus network. You should run the web server, which includes the server echo code, on machine one. The proxy server, including the client ping code, should run on machine two. The client side of the web server (browser) should also run on machine two. You will need to set the IP and port numbers correctly. We will set up a second test machine for students working alone and with access to a single machine.

During the demo, we expect to see the UDP Pinger working as described in this document and simultaneously with the web server and proxy server. We may request multiple objects from the browser while the ping code runs, which should continue running smoothly without issues.

Submission

- Complete the "README.md". You may write “n/a” where applicable (bugs, references, etc.).
- Please submit the following files for Project-3 on Brightspace using the same file name and extension as here:
  - udppinger-client1.py (part 1)
  - udppinger-client2.py (part 2)
  - webserver3.py (part 3)
  - proxyserver3.py (part 3)
  - Screenshots (PNG) at the client for part 1 and part 2 verifying that your ping programs work as expected.
  - Screenshots (PNG) at the client (browser), and proxy server for part 3 verifying that the terminal prints the correct ping output and the browser renders the requested files correctly.
  - README.md
- You must submit your project before the deadline to be considered on time. Otherwise, it will be considered late even if your project was completely working before the deadline. ***Please note that on Brightspace, we maintain all your submissions but only grade your most recent submission.***

Grading Rubric

Total: 100 points

- Part 1: 15 points
  - Implementation: 15 points
- Part 2: 15 points
  - Implementation: 15 points
- Part 3: 55 points
  - Demonstration: 30 points
  - Implementation: 25 points
- Submission: 15 points
  - README.md: 10 points
  - All other requirements: 5 points
- If submission didn’t pass the plagiarism test(s): -100 points.