Final Project

Introduction:

Your task is to design and implement a system that simulates TCP packets using a UDP connection by extending the user space of your application while maintaining reliability and supporting the HTTP protocol on top of UDP. UDP is a connectionless protocol that does not guarantee the delivery or order of packets (see 3.3.1). This presents several challenges when attempting to transfer data reliably. socket — Low-level networking interface — Python 3.12.3 documentation

One solution should include mechanisms for error detection and correction, packet retransmission, and flow control. Error detection and correction can be achieved using checksums (see 3.3.2) or other error-detecting codes. Packet retransmission can be implemented using acknowledgments and timeouts. Flow control (see 3.5.5) can be achieved using sliding window protocols or other congestion control mechanisms.

In addition to these reliability mechanisms, your solution should also support the HTTP protocol. This will involve parsing HTTP requests and responses and handling the various methods and headers defined by the protocol.

Think of the trade-offs between performance and reliability when designing your solution. Your system should be able to handle packet loss, duplication, and reordering while maintaining a reasonable level of performance.

Your solution will be evaluated on its effectiveness in achieving reliable data transfer over UDP and its support for the HTTP protocol. You should provide a detailed description of your design and implementation, including any assumptions you have made and any limitations of your solution. Also, you have to write test cases for your code.

Objectives:

- Connection Establishment:
  - Implement a handshake mechanism between the client and server to establish a connection using UDP packets.

- Reliable Data Transfer:
  - Ensure reliable data transfer with acknowledgments (ACKs) and retransmissions for lost packets.
- HTTP/1.0 Support: HTTP headers - HTTP | MDN (mozilla.org)
  - Enable the client to send various HTTP/1.0 requests (GET, POST.) to the server. Connection management in HTTP/1.x - HTTP | MDN (mozilla.org)
  - Allow the server to respond with HTTP/1.0 responses. Status (not Found and OK) HTTP response status codes - HTTP | MDN (mozilla.org)
- Error Handling and Testing:
  - Handle errors such as invalid packets, timeouts, and connection termination gracefully.

Requirements:

- HTTP 1.0 (see 2.2.2, https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x).
- Make special consideration to HTTP Headers (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers), you may have to implement some of them.
- It is good to watch the flow of your connection using Wireshark or https://www.postman.com/downloads/ . Seeing the HTTP messages will help you understand better what to do.
- At least status messages OK and NOT FOUND are required (see https://developer.mozilla.org/en-US/docs/Web/HTTP/Status).
- Use python UDP socket (see https://docs.python.org/3/library/socket.html).
- Most of your work will be on a newly created Class that should implement the transition described before. Name it as you like.
- Then use that class to implement an HTTP server and client.
- Only the GET and POST methods are required (see 2.2.3, https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST , https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET).
- Use Stop-and-wait (see fig. 3.18).
- Calculate the checksum of packets before sending them and include them in the packet. Check the checksum when receiving and if it is not correct, the packet is dropped. You will need a method to simulate a

false checksum to see what happens. The sender then realizes that there is no ack received from the receiver and timeout and resends it again. Think of what you should include from the packet while calculating the checksum. (eg: CRC-32) Cyclic redundancy check - Wikipedia

- Implement packet loss and packet corruption and implement methods that are specially created for this purpose. (eg: 2% packet loss, 10% corruption)
- Take special consideration to the following: retransmission, duplicate packet, sequence number, handshake, flags (eg. SYN, SYNACK, ACK, FIN), and timeouts.

Bonus

- Persistent HTTP
  - Implement persistent connections to allow multiple HTTP requests and responses to be sent over a single connection.
  - Handle connection reuse and keep-alive headers to optimize performance.
- Window Implementation Implement a sliding window mechanism to control the flow of data packets between the client and server.
  - Manage the window size dynamically based on network conditions and congestion control.

What To Submit

- Your source code, documented, and your hard-coded test cases are ready to run (do not put them within the report).
- A PDF report and any screenshots you are required to attach.
- Enclose all of that into one folder compress this folder as *.zip then change the extension of the file from *.zip to *.zip.pdf and submit it via the form.
- Name the folder as follows. studentID-sp24-nw-proj.zip.pdf (eg: 1234-sp24-nw-proj.zip.pdf).
- Submission Link: https://forms.office.com/r/etLYPEYA01

Policies

- No late.
- Copies will be heavily penalized.
- Your code should be clean, readable, and highly documented.
- You must submit at least one paper explaining your work testing your code and stating your conclusions (if any conclusions exist).
- Online submission on time is a mandatory step for your submission.
- You will be penalized if you don't attend the discussion on time (or at all), so stick with your timeslot.
- You should obey the submission format explained above.