Phase 4 Design Document

This is the document for the Phase 4 of the project. Phase 4 requires to implement RDT 3.0 which add some new features in addition to Phase 3. New features like handling packet or ACK loss by using a countdown timer. Overall, it sends one file from client side in the form of packets which contains data along with sequence number and checksum; and server receives those packets, extract data, checksum and sequence number. Once server is done, it sends the sequence number of the received packet as ACK to client. Client receive the ACK and check if it is correct or not. If ACK is correct, it sends the next packet else it will wait for the timer to timeout. Timeout will resend the packet. Similar mechanism for the data loss case. Once all packets are being received by the server, it will save it as an image onto the laptop.

We have written two codes – one for the sender side and other for the receiver side. Each code has one class – client () and server () respectively.

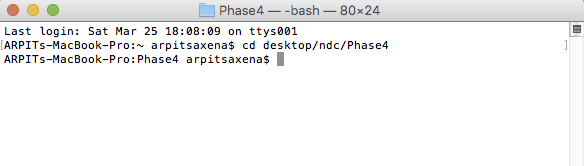
1. Server () – This class contains eight functions and is responsible for the function of the sender side. It provides user 5 options to choose from: 1. No Error 2. ACK Error 3. Data Error 4. ACK Loss 5. Data Loss. Once you select the option, it will divide the data into chunks and create packet for each chunk by adding sequence number, checksum for that chunk and then send that packet to receiver. Once it sends the packet, it will start the timer. It will now wait to receive the ACK for the sent packet till the timeout happens. If it receives the ACK before timeout and user has opted for option 2 or option 4, sender will corrupt the ACK, based on error percentage and wait for the timeout. Only timeout can resend the packet. Once it sent all packets and received all ACKs for the sent packets, it will display the time it took to send the whole file. Time of execution starts when it has sent its first packet till it has received the last ACK.
2. Client () – This class is mentioned in the code written for receiver side and contains 6 functions. It receives many parameters before receiving the chunks like which option has been selected by the user, chunk size, checksum size, and corruption error percentage only if option 3 or option 5 has been selected. Once all required parameters have been received, it will start receiving packets. It will extract the sequence number, checksum and data. It will then send the ACK which is the sequence number of the received packet. It will also corrupt the received data based on the error percentage if the user has opted for option 3 or option 5. Once it receives all packets, it will join the data together, encode it jpg format and save it in the desktop.

Functions:

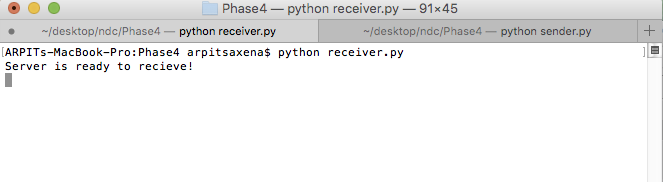
1. \_\_init\_\_ - this is init function defined for both the classes and it holds the initialization of global variables. We also created socket within the init function for both client and server.
2. make\_chunks – The primary role of this function is to divide the image into chunks. It takes the whole file data as the argument.
3. make\_packets – it will receive chunk as an argument and then call checksum function. It will create packet by concatenating the chunk, checksum value and sequence number.
4. Checksum – this function first convert the chunk to hexadecimal format and then add each hex value to other. We converted the data to hex so that we make sure that the sum is of length 16 bits. It is present in both codes.
5. Time\_out – If time-out happens, then this function will be called to resend the packet by calling send\_data function.
6. Send\_data – For every packet to be sent, this function will be called. It sends the checksum size, packet size and then packet. Once packet is sent, it starts the timer and wait for the ack. If it receives the ACK before timeout, it will check the ack. If option 2 or option 4 has been selected then depending upon the error percentage, it will corrupt the ACK and then wait for timeout. Once timeout happens, it will call Time\_out function to resend the packet.
7. serverSend – This function first asks the user about the mode in which he wants to send the file. If he chooses the Error mode, then it will ask the user to enter the error percentage. It will send some parameters to receiver before sending the packets. It will then open an image and decode it using base64 library. It divides the image into several chunks of fixed size, make packets and call send\_data. It will also calculate the time taken for the execution.
8. send\_msg – This function is responsible for sending out the messages to the either sender or receiver. It is present on both the sides and takes the message as the argument.
9. make\_corrupt – The role of this function is to corrupt the chunk we received. It will simply overwrite the chunk with some garbage value.
10. Extract\_header – it will extract the data, checksum and sequence from the packet it received. It has 2 arguments – packet and checksum size.
11. clientRecv - It will first receive all required parameters and will receive the packets. If option 3 has been chosen, then will randomly corrupt the data and then send the wrong ACK. If everything goes fine, then we append the data to a list. Once it receives all the packets and send all the corresponding ACKs, it will write the data as image onto the desktop again.

Step by Step Execution:

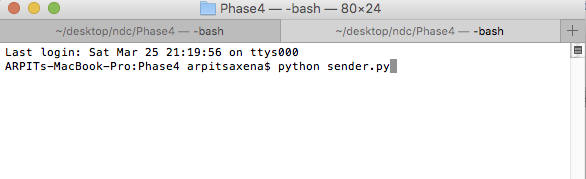
1. Open the terminal and change the current directory to the desired directory.



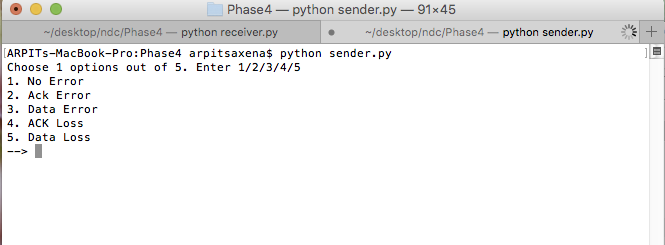
1. Run the client code by writing the command – python followed by the filename and the moment we hit enter, code will execute and client is waiting to receive the message from server side.

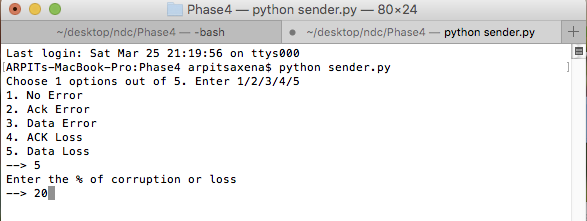


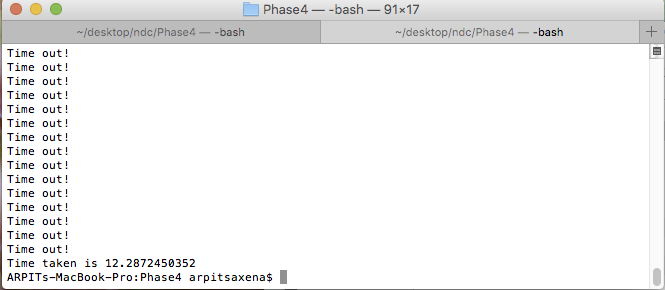
1. Open another terminal to execute the server code.



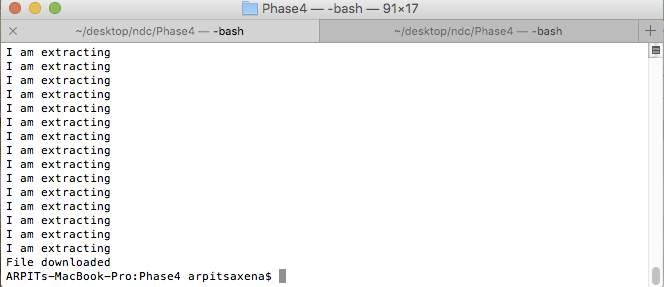
1. We will again use the command – python followed by the filename. Next we hit the enter and it will ask the option to choose. And User supposed to enter 1 or 2 or 3 or 4 or 5.



1. Once you enter the option, it will ask to enter the % of corruption on client side if it is any option other than 1.   
   



1. This is the client side which received packets, each packet size. Then it will receive all the packets, assemble them and save it back as the image on the relative folder.



Graph of Compilation time VS Error percentage.

