The University of Queensland

School of Information Technology and Electrical Engineering

Semester One, 2022

COMS3200 Computer Networks I

Assignment 1

Due: 3:00pm 27th April, 2022

Marks: 100

Weight: 25% of your final grade.

Revision: 1.0

Introduction

This assignment introduces the Network Edge & Core, the top three Networking layers (from OSI models) such as Application Layer, Transport Layer, and a small part of Network Layer. After finishing the assignment, you should be able to distinguish between Packet and Circuit Switching, understand End-to-End Delay, be able to interpret and analyse network packets and do network programming.

This assignment contains three (3) parts: Part A, Part B and Part C. For Part A and B, you will need to submit you answers through the Blackboard Quiz. For Part C, you will submit your code via COMS3200 subversion (SVN). All submitted work will be marked automatically. And it's strongly recommended that you have read this specification very carefully.

Note that this assignment is individual work. Using answers (or code) that you did not calculate (or write) is against course rules and may lead to a misconduct charge.

Part A (34 marks)

Answer each of the following questions in the associated quiz on Blackboard, following the specified instructions. All answers will be automatically marked.

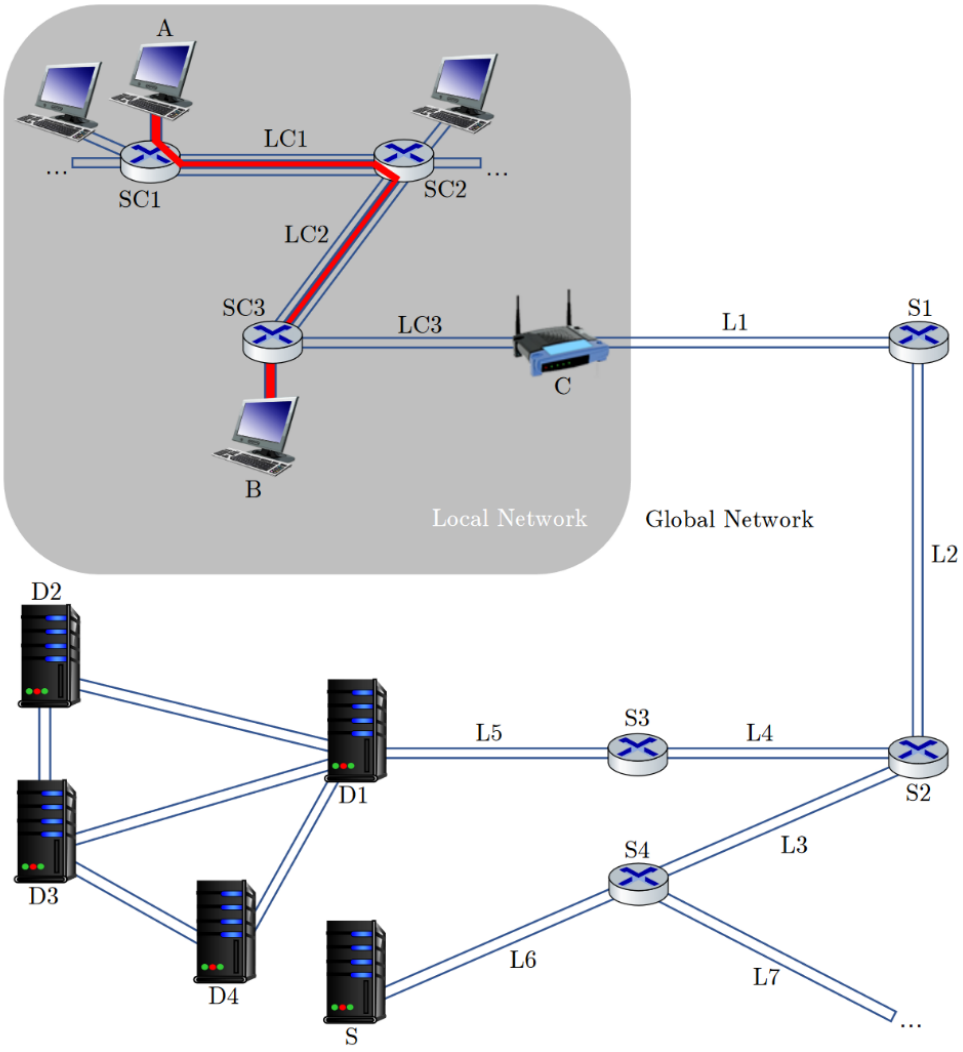

Figure 1. A simple network configuration map.

Figure 1 shows a simple network map that includes two edges: a Local Network (represented as the grey area) and a Global Network (white area).

In the Local Network, there are the three circuit switches (SC1, SC2 and SC3). They are interconnected by two links (LC1 and LC2). Each of these circuit links has three slots. Each host is directly connected to one of the switches. In this part, you only consider the communication scenarios between Host A and Host B. The network reserves one slot on each of the two links. In Figure 1, the dedicated end-to-end connection uses the second slot in both links LC1 and LC2 (see the red coloured line). We assume

that device C (which is located between is LC3 and L1) is the combination of modem and router in the Local Network and connected to SC3 through LC3.

In the Global Network, packet-switching technique is used in the entire network. We have a scenario where a client requests a web page from a remote server on a remote island via a slow satellite link above the earth in a geostationary orbit. There is one client C, one server S, and four DNS servers. D1 is a Local DNS Server, D2 is a Root DNS Server, D3 is a Top-level Domain DNS Server, and D4 is an Authoritative DNS Server. We don't need to know where the Link L7 is connected in this assignment.

Scenario 1 – Local Network (6 marks)

Host A sends a file of 2,790,000 bits to Host B at the time $t_0 = 0$ mili-second (msec). Link LC1 and link LC2 use time-division multiplexing, all of them have one frame, and it is one second that is divided by three slots. The total bit rate is 9 Mbps (from SC1 to SC3). It needs 500 msecs to establish an end-to-end circuit before Host A begins to transmit the file. And the second slot of the circuits in LC1 and LC2 is allocated for the transmission between Host A and Host B. At the time $t_0 + 500$ msecs, the first slot of the circuits in LC1 and LC2 starts to transmit. At the time t_1, Host B has received all bits of the file from Host A. After 400 msecs from t_1, Host B sends a file of 990,000 bits to Host A. Host A has received all bits of the file from Host B at time t_2.

There are some assumptions for the Local Network:

- There are no propagation delays in this networking.
- There are no transmission delays between Host A and SC1, as well as Host B and SC3.
- The network starts without data transmission.
- The slot of the circuits is allocated for the transmission can be used in both directions.
- All switches have no processing delays (this is unrealistic though, we assume to simplify calculations).

Answer the questions below in mili-seconds (msecs) <u>rounded to three decimal places</u>.

Question 1: What is the value of t_1? (3 marks)

Question 2: What is the value of t_2? (3 marks)

Scenario 2 – Global Network (21 marks)

A program on one of the hosts in the Local Network starts the process of retrieving a webpage from the Server S. The following events happen sequentially:

1. Client C sends a DNS request to D1.
2. D1 sends a response with Server S's IP address to Client C.

3. C sends a TCP request (SYN) to open a connection to Server S.
4. Server S acknowledges (SYN/ACK) the TCP request and opens the connections with Client C.
5. Client C sends a HTTP GET (which also includes the TCP ACK) request to Server S. This packet is considered just a HTTP GET packet (i.e. 1000 bytes in total)
6. Server S sends the web page to Client C, segmented into 7 packets, which includes the HTTP response plus the HTML file.
7. After receiving each data packet, Client C sends an ACK message back to Server S. Note that S does not need to wait for an ACK before sending the next data packet. After receiving the last data packet acknowledgement, S sends a TCP FIN packet to close the connection.
8. After receiving FIN from Server S without any processing delays, Client C sends one FIN/ACK packet back to S. After receiving FIN/ACK without any processing delays, S sends a final ACK packet back to C. When this final ACK packet is received at C the connection is finally closed.

There are some assumptions for the Global Network:

- The network is packet-switched.
- The network starts with no other packets in queues - only packets that are a part of this question.
- All events outlined are successful (i.e., no corrupted packets, retransmissions, etc)
- Apart from the switches, there are no other processing delays.
- Each link is bidirectional and can concurrently handle bits travelling in opposite directions.
- All required records are stored in the DNS servers.
- C, D1 and S have no processing delays (this is unrealistic, just to simplify calculations).
- All DNS packets are 100 bytes long, including all headers, queries, responses, etc.
- All TCP SYN, ACK, FIN, SYN/ACK, and FIN/ACK packets are 200 bytes long, including all headers, preamble, etc.
- All HTTP GET and response packets are 1000 bytes long, including headers, preamble, etc.
- 1 Mbps = 10^6 bps; 1 Kbps = 10^3 bps (bps is bits per second).
- 1 GB = 1024 KBs; 1 KB = 1024 Bs; 1 B = 8 bs (B denotes byte and b is bit, respectively).

Answer the following questions below in mili-seconds (msecs) <u>rounded to three decimal places</u>.

Question 3: In event 1, how long does it take for D1 to receive the DNS request? (2 marks)
Question 4: In event 2, how long does it take for C to receive the response? (2 marks)
Question 5: In event 3, how long does it take for S to receive the SYN packet? (2 marks)
Question 6: In event 4, how long does it take for C to receive the SYN/ACK packet? (2 marks)
Question 7: In event 5, how long does it take for S to receive the HTTP GET packet? (2 marks)
Question 8: In event 6, how long does it take for C to receive the **first** HTTP packet? (2 marks)
Question 9: In event 6, how long does it take for C to receive the **second** HTTP packet? (3 marks)
Question 10: In event 6, how long does it take for C to receive the **last** HTTP packet? (3 marks)
Question 11: How long does it take for C to receive the ACK packet (event 8) after sending the ACK packet for the last HTTP packet (event 7)? (3 marks)

Some information about Transmission Links and Switches that you may need in your calculations.

Table 1. Tranmission Links

Link	Connect Between		Transmission Rate	Length	Propagation Speed
L1	C	S1	100 Mbps	240 m	2 x 10^8 m/s
L2	S1	S2	100 Kbps	42,000 km	3 x 10^8 m/s
L3	S2	S4	100 Mbps	60 m	3 x 10^8 m/s
L4	S2	S3	500 Mbps	3,300 km	3 x 10^8 m/s
L5	S3	D1	200 Mbps	500 m	2 x 10^8 m/s
L6	S4	S	50 Mbps	60 m	2 x 10^8 m/s

Table 2. Switches and Processing Delay

Switch	Link Associated	Processing Delay
S1	L1, L2	1 msec
S2	L2, L3, L4	2 msec
S3	L4, L5	0.5 msec
S4	L3, L6, L7	0.25 msec

Scenario 3 – The Internet's Directory Service (7 marks)

In this scenario, we consider on the Global Network only. The following events happen sequentially:

1. Client C sends a DNS query message of a host name to D1.
2. D1 receives the DNS message, then forward the query message to D2.
3. D2 receives the DNS message, then sends a DNS message to D1.
4. D1 receives the DNS message, then sends the DNS message to D3.
5. D3 receives the DNS message, then sends a DNS message to D1.
6. D1 receives the DNS message, then sends the DNS message to D4.
7. D4 receives the DNS message, then sends a DNS message to D1.
8. D1 receives the DNS message, then forwards the message to Client C.

Answer the following questions.

Question 12: Which on these events can combined together to form an recursive DNS query? (2 marks)

Question 13: Which on these events can combined together to form an iterative DNS query? (2 marks)

Question 14: In event 3, what is DNS record type in the message that D2 sends to D1? (1 mark)

Question 15: In event 5, what is DNS record type in the message that D3 sends to D1? (1 mark)

Question 16: In event 8, what is DNS record type in the message that D1 sends to C? (1 mark)

Part B (25 marks)

Answer each of the following questions in the associated quiz on Blackboard, following the specified instructions. All answers will be automatically marked.

In this part, you need to know how to use Wireshark in analysing a packet capture file, names **a1.pcap**. This packet capture shows a client downloading a javascript resource file from a server. Some of the relevant IETF RFCs may help you understand more of the following questions. In particular:

1. RFC 793 for Transmission Control Protocol
2. RFC 791 for Internet Protocol
3. RFC 2018 for TCP Selective Acknowledgment Options
4. RFC 7323 for TCP Extensions for High Performance

Wireshark displays TCP sequence numbers as a value relative to the first sequence number. You will need to disable this to answer any questions that ask for a raw sequence number. It is highly recommended that you turn off TCP Reassembly to understand the order of packet transmission.

Answer the following questions.

Question 17: What is the web browser being used by the sender? (1 mark)

Question 18: What is the web server software being used in the remote server? (1 mark)

Question 19: Is protocol offloading being used at the client? (1 mark)

Question 20: How many routers are there between the client and server? (1 mark)

Question 21: What is the raw TCP sequence number of...

a. the packet initiating the TCP connection (1 mark)
b. the acknowledgement from the server for the above packet? (1 mark)

Question 22: In frame 4, what are the values of the 8 TCP flags (CWR to FIN) as a single 8-digit binary number? (1 mark)

Question 23: In frame 4, the GET request is for a JavaScript file resource. What is the address of the web page that requested this JavaScript file resource? (2 marks)

Question 24: How long should the received JavaScript file be considered fresh? (2 marks)

Question 25: What is the initial value of the window scale shift count indicated by...

a. the client? (1 mark)
b. The server? (1 mark)

Question 26: The GET packet from the client to the server has an advertised window size of 46. What is the true value window size in bytes? (2 marks)

Question 27: How many bytes are lost totally while transmission is indicated in the pcap file? (2 marks)

Question 28: How many duplicate acknowledgments are sent by the client regarding the missing bytes? (2 marks)

Question 29: In which frame is the lost frame retransmitted? (2 marks)
Question 30: In which frame does the receiver indicate that all missing frames have been received? (2 marks)
Question 31: From frame 1 to 3 (inclusive), how many frames have incorrect TCP checksum value? (2 marks)

Hints:

- Frame 4 shows a HTTP GET request from the client to the server, that could help you on answering question 22, 23, and 24.
- In frame 30, some data is lost. Question 27, 28, 29, and 30 are focused on the lost frames.

Part C (41 marks)

You have recently been hired by the multinational tech giant COMS3200 Inc., who have identified your in-depth knowledge in the field of secure transport-layer protocols. **You have been tasked to develop a network server capable of sending messages to a client that comply with the desired protocol.** This part will implement a network simulation in the application layer, called RUSHBServer.

Again, this is an individual assignment. You can feel free to discuss aspects of socket programming and the specification with fellow students. Directly help (or seek help from) other students with the actual coding of your solution is considered cheating, as well as looking at another student's code (even if it's in printed or electronic form). All of your submitted code will be subject to automated checks for plagiarism and collusion on MOSS, and detected plagiarism or collusion will be reported for misconduct proceedings. If you're having trouble, please seek help from a member of tutors. Don't be tempted to copy another student's code, and don't commit any code to your repository unless it's your work.

Simulation

Each process (a running instance) of RUSHB server will simulate file storage server. When your server receives a client's file request, it should locate the requested file in its local working directory and return the file contents over one or more packets. When complete, the server should close the connection with the guest that have sent the file. Your server also needs to be capable of simultaneously dealing with multiple clients. The server should try its best on improving the flow-control and guaranteeing the clients to get all the content reliably and uncorrupted by using the company's selected protocol, RUSHB (Reliable UDP Substitute for HTTP Beta).

RUSHB Structure

The RUSHB protocol is a HTTP-like **stop-and-wait** protocol that uses UDP in conjunction with some idea of the RDT (Reliable Data Transfer) protocols. It is expected that the RUSHB protocol is able to handle packet corruption and loss.

A RUSHB packet can be expressed as the following structure (|| is concatenation):

IP HEADER || UDP HEADER || RUSHB HEADER || ASCII PAYLOAD

The data segment of the packet is a string of ASCII plaintext. A single RUSHB packet must be no longer than 1500 bytes, including the IP and UDP headers (i.e. the maximum length of the data section, or the concatenation of RUSHB HEADER and ASCII PAYLOAD, is 1472 bytes). Any packets smaller than 1500 bytes need to be padded with 0 bits up to that size. In detail, Figure 2 describes the header structure of a RUSHB packet.

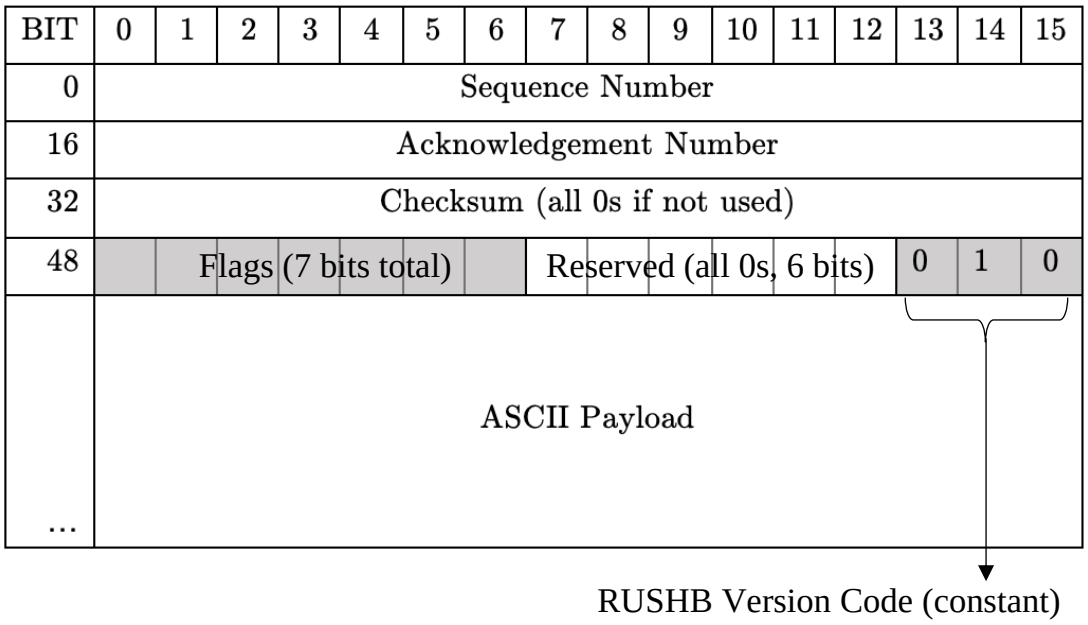

Figure 2. Structure of a RUSHB Packet

A client and server independently maintain sequence numbers. The first packet sent by either endpoint (a client and server) should have a sequence number of 1, and subsequent packets should have a sequence number of 1 higher than the previous packet (note that unlike TCP, RUSHB sequence numbers are based on the number of packets as opposed to the number of bytes).

When the ACK flag (see Figure 3: Flags structure) is set, the acknowledgement number should contain the sequence number of the packet that is being acknowledged. When a packet is retransmitted, it

should use the original sequence number of the packet being retransmitted. Packet that is not retransmission (including NAKs) should increment the sequence number. The Flags Header is broken down in Figure 3.

BIT	0	1	2	3	4	5	6
FLAG	ACK	NAK	GET	DAT	FIN	CHK	ENC

Figure 3. The Flags Header structure

The following scenario describes a simple RUSHB communication session. The square brackets denote the flags set in each step (e.g., [FIN/ACK] denotes the FIN and ACK flags having the value 1 and the rest having the value 0). Note that RUSHB, unlike TCP, is not connection-oriented. There is no handshake to initialise the connection, but there is one to close the connection.

Scenario A (simple communication):

1. The client sends a request [GET] to the server. The sequence number of this packet is 1.
2. The data section (ASCII payload) of this packet contains the name of a resource (e.g. file.txt).
3. The server receives [GET] message, then transmits the requested resource to the client over (possibly) multiple [DAT] packets. Remember, RUSHB protocol is a HTTP-like **stop-and-wait** protocol. The first packet from the server has a sequence number of 1.
4. The client acknowledges by sending an [DAT/ACK] packet to each received data packet. The Acknowledgement Number of each packet is the Sequence Number of the packet being acknowledged.
5. After receiving the last acknowledgement [DAT/ACK] from the client, the server sends [FIN] message to end the connection.
6. The client receives [FIN] message from the server, then sends back [FIN/ACK] to the server.
7. After receiving the last acknowledgement [FIN/ACK] from the client, the server send [FIN/ACK] again to the client and close the connection.

For all the packets with [DAT] flag is set to 0, the payload must be filled with all 0s bit only. Please note that it is just a simple example of RUSHB protocol; the server also needs to deal with optimised data flow control and multiple clients (that will be described later).

RUSHB server is capable of checksum. During the initial [GET], clients can negotiate requests for checksum. This is done using [CHK] for checksum in the very first [GET] packet. The first [DAT] from the server will indicate if negotiation was successful by setting the corresponding [CHK] flag. Once negotiated, these options are valid for all packets until close the connection with that client.

[ENC] is encryption flag. In this assignment, you do not have to implement [ENC], thus this flag can be left as 0.

RUSHB checksum uses the standard Internet checksum on the payload only. As per the RFC, the checksum field is the 16-bit one's complement of the one's complement addition of all 16-bit words of the payload (see example below). Once [CHK] is negotiated, all packets that have invalid checksums are considered corrupt.

For example, this is how we calculate:

ASCII Payload	abcde
Calculation	0x6261 is ab 0x6463 is cd 0x0065 is e ---------- 0xc729
Result	~(0xc729) = 0x38d6

Figure 4. Example checksum calculation

Submission

Your server program must be written in Python or C. Your program should be able to be invoked from a UNIX command line as follows. It is expected that any Python programs can run with version 3.6, any C programs have to be complied with the standard of C99. **No external libraries (e.g, scapy) or extensions (e.g, gnu99) is permitted to use in your program.**

Python	python3 RUSHBSvr.py
C	make ./RUSHBSvr

Figure 5. Filename and Command-Line syntax for the submission

No late submissions will be marked for this assignment under any circumstances. Submission must be made electronically by committing using subversion. In order to mark your assignment, the markers will check out /trunk/ass1/ from your repository on source.eait.uq.edu.au. Please do not create subdirectories under /trunk/ass1/. **The marking script will check for any such directories before attempting to compile, and will stop running if it found one**. Code checked in to any other part of your repository will not be marked.

IMPORTANT NOTICE: As the assignment is auto-marked, it is very important that the filename and command-line syntax exactly matches the specification above. Specification adherence is critical for passing. If your program is failed to executed because of the wrong syntax, you will receive a 0 for the assignment without any exceptions.

Tasks

1. Basic Server (5 marks)

To receive marks in this section you need to write a program that is able to:

1. Listen on an unused port for a client's message
2. Successfully close the connection

When invoked, your program should let the OS choose an unused localhost (127.0.0.1) port and your program should listen on that port. It should output that port as a raw base-10 integer to stdout. For example, if you use C for your program and port 12345 was selected, your program invocation is:

```
make
./RUSHBSvr
12345
```

If you use Python for your program and port 23456 was selected, your program invocation is:

```
python3 RUSHBSvr.py
23456
```

Any lines in stdout after the port number can be used for debugging. For this section, your program does not need to respond to the [GET] request from the client. Upon hearing from a client, your program can immediately signal the end of the connection (as described in the example above). Once the [FIN] handshake has been completed, your server should close the connection with the client. Note that in this case, unlike TCP, closing the connection clears the state (or cache) associated client with the client. The current client could send [GET] request and get accepted by the server again, but any other packets that associated with the previous session (such as [ACK] of the previous file) should be ignored. You do not need to implement [CHK] or [ENC] for this part.

2. File Tranmission (6 marks)

To receive marks in this section you need to write a program that is able to:

1. Perform all features outlined in the above section
2. Successfully transmit a requested file over one or more packets
3. Receive ACKs from the client during transmission

When your server receives a GET packet, it should locate the file being requested and return the file contents over one or more DAT packets. When complete, the server should close the connection (as in the above section). **If the file being requested does not exist, closes the connection.** It is expected

that this file is stored in your program working directory. You do not need to implement [CHK] or [ENC] for this part.

3. Retransmission (12 marks)

To receive marks in this section you need to have a program that is able to:

1. Perform all features outlined in the above sections
2. Retransmit any packet on receiving a NAK for that packet
3. Retransmit any packet that has not been acknowledged within 4 seconds of being sent

A client will send a DAT/NAK packet should it receive a corrupted packet or a packet with a sequence number it wasn't expecting (the NAK packet's acknowledgement number will contain the sequence number it was expecting). In this case, your program should retransmit the packet with that sequence number. If a [DAT], [FIN], or [ACK] packet gets lost during transmission, your program should retransmit it after 4 seconds without acknowledgement (an packet should has its own timer, saparately). If a [NAK] is received, the timer should reset (for the packet related to that [NAK] only). How you choose to handle timeouts is up to you, however it must work on moss.

Please note there could be some unexpected behaviours when dealing with concurrency (clocking and frequency), please research carefully before doing this task. You can only use standard libraries, please make a thread on Ed if you are not sure if a library is permitted for use or not. You do not need to implement [CHK] or [ENC] for this part.

4. Packet Integrity [CHK] (6 marks)

To receive marks in this section you need to have a program that is able to:

1. Perform all features outlined in the above sections
2. Implement the [CHK] mode negotiation and ignore packets with corrupt checksums if checksum mode is in use

When a packet with incorrect checksum arrives, your program should ignore it and continue to run without error. Any retransmission timer should also not stop or reset. You must also ensure that if checksum was negotiated at the start, all packets must have a valid checksum. You do not need implement [ENC] for this part.

5. Multiple-clients Handling (12 marks)

To receive marks in this section you need to have a program that is able to:

1. Perform all features outlined in the above sections
2. Handle multiple connection with clients

Your server should handle multiple connections with the clients. It should not mismatch the packets of one client with another. When closing the connection with a client, your server should do the action with that client only, and the transmission with others should continue as usual.

Marking

Marks will be awarded for functionality only. The tests will mostly check your program's behaviour and packet structure. Provided that your code compiles (see above), you will earn marks based on the number of features your program correctly implements. Partial marks may be awarded for partially meeting the functionality requirements. Not all features are of equal difficulty. If your program does not allow a feature to be tested, you will receive 0 marks for that feature, even if you claim to have implemented it. For example, if your program can never open a file, we can not determine if your program would have loaded input from it. The markers will make no alterations to your code (other than to remove code without academic merit). Your program should not crash or lock up/loop indefinitely (without any reason). Your program should not delay for unreasonably long times.

Tips and hints

- Please start your assignment early.
- Drawing a finite state machine is a good way to design a program’s work flow.
- Check if the process send packets successfully by using Wireshark (listen on loopback).
- Use RUSHBSimpleClient to debug your program, and test on MOSS frequently using RUSHBSimpleTest (will be released soon).
- Ask us if you need any assistances on the course EdStem.

Specification updates

It is possible that this specification contains errors or inconsistencies, or missing information. It is possible that clarifications will be issued via the course website. Any such clarifications posted five (5) days or more before the due date will form part of the assignment specification. If you find any inconsistencies or omissions, please notify the tutors and lecturer.

Updates

1.0 First release