The University Of Queensland
School of Information Technology and Electrical Engineering
Semester One, 2021

COMS3200 - Assignment 1

Due: 3:00pm 24th March, 2021
Marks: 100
Weighting: 20% of your final grade
Revision: 1.1

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 programing.

This assignment contains three (3) parts. For Part A and B, you will need to submit you answers through Blackboard Quiz. for Part C, you will submit your code via COMS3200 subversion (SVN). All submitted works are marked automatically. And it's strongly recommended that you have read this spec carefully before start working, as avoiding any mistakes.

This assignment is to be your 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 (30 marks)

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

The figure below illustrates a simple network map that included two (2) edges: Local Network (grey area) and Global Network (white area).

In Local Network, the three (3) circuit switches: SC1, SC2 and SC3 are interconnected by two (2) links: LC1 and LC2. Each of these circuit links has three (3) slots. The hosts are each directly connected to one of the switches. In this part, we only need to know the communication scenario between Host A and Host B. The network has reserved one slot on each of the two links. In the figure below, 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 is the combination of modem and router in the Local Network, are connected together and connect to SC3 through LC3. The data transmission from device C in Local Network through L1 is considered the data from client C in the Global Network view.

In Global Network, packet-switches are used in the entirety of the 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 (1) client C, one (1) server S, and four (4) DNS servers: D1 is a Local DNS Server, D2 is a Root DNS Server, D3 is a Top-level Domain (TLD) DNS Server, and D4 is Authoritative DNS Server. We don't need to know where the Link L7 connect to in this assignment.

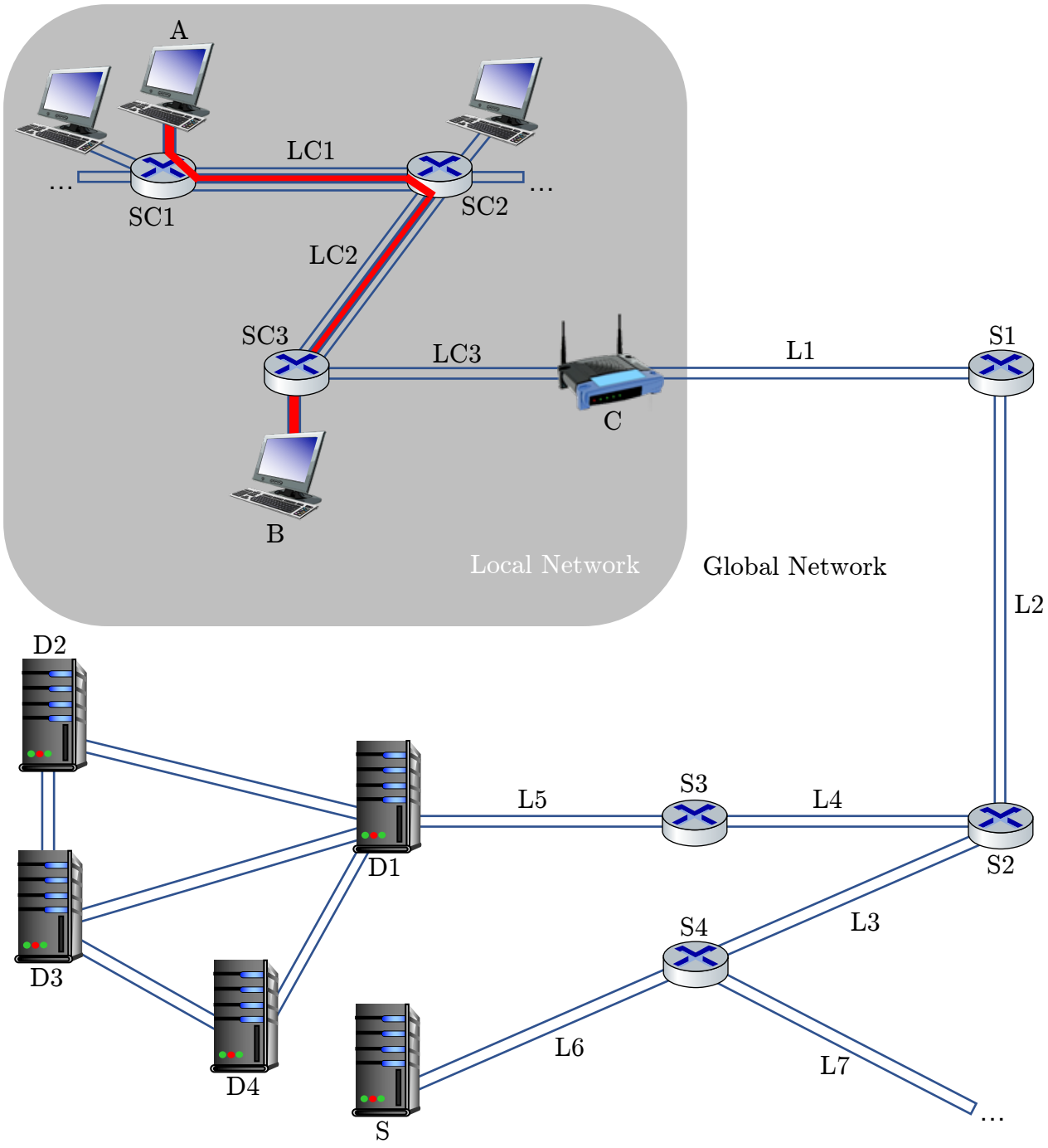

Figure 1. A simple Network Map

Scenario 1 – Local Network (8 marks)

Host A sends a file of 2,880,000 bits to Host B at the time $t_0 = 0$ mili-second (msec). Link LC1 and link LC2 uses time-division multiplexing (TDM) with three (3) slots and have a bit rate of 9 Mbps. It needs 500 msecs to establish an end-to-end circuit before Host A can begin to transmit the file and the second (2^{nd}) 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 (1^{st}) slot of the circuits in LC1 and LC2 start to transmit. At the time t_1, Host B has received all bits of the file from Host A. After 200 msecs from t_1, Host B sends a file of 960,000 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:

1. There is no propagation delays in this networking.
2. There is no transmission delays between A and SC1, as well as B and SC3.
3. The network starts with no data are transmissing at the time $t_0 = 0$.
4. The slot of the circuits is allocated for the transmission can be used in both direction.
5. All switches have no processing delays (this is unrealistic, just to simplify calculations).

Task: Answer the questions below in mili-seconds (msecs) rounded to three decimal places.
Question 1: What is the value of t_1? (4 marks)
Question 2: What is the value of t_2? (4 marks)

Scenario 2 – Global Network (17 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. At the time $t_0 = 0$ mili-second (msec), Client C sends a DNS request to D1. D1 receives the DNS request from Client C successfully at the time t_1.
2. Right after t_1 without any processing delays, D1 sends a response with Server S's IP address to Client C. Client C receives the DNS response from D1 successfully at the time t_2.
3. Right after t_2 without any processing delays, C sends a TCP request (SYN) to open a connection to Server S. Server S receives the TCP SYN request from Client C successfully at the time t_3.
4. Right after t_3 without any processing delays, Server S acknowledges (SYN/ACK) the TCP request and opens the connections with Client C. Client C receives the TCP (SYN/ACK) acknowledgement from Server S successfully at the time t_4.
5. Right after t_4 without any processing delays, Client C sends a HTTP GET request to Server S (which includes the ACK back to S for its SYN). Server S receives the HTTP GET packet from Client C successfully at the time t_5.
6. Right after t_5 without any processing delays, Server S sends the web page to Client C, which requires 7 packets, which includes the HTTP response plus the HTML file. Client C receive the first (1^{st}) packet of the HTTP response from Server S successfully at the time t_6 and the seventh (or last) packet of the HTTP response at the time t_7.
7. After receiving each data packet without any processing delays, 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 without any processing delays, S sends a TCP FIN packet to close the connection. 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 without any processing delays, the connection is finally closed at the time t_8.

There are some assumptions for the Global Network:

1. The network is packet-switched.
2. The network starts with no other packets in queues - only packets that are a part of this question.
3. Each link is bidirectional and can concurrently handle bits travelling in opposite directions.
4. All required records are stored in the DNS servers.
6. C, D1 and S have no processing delays (this is unrealistic, just to simplify calculations).
5. DNS packets and TCP SYN, ACK, FIN, SYN/ACK, and FIN/ACK packets are 100 bytes long, including all headers, preamble, etc.
6. HTTP GET and HTTP response packets are 1000 bytes long, including all headers, preamble, etc.
7. 1 Mbps = 10^6 Bps; 1 Kbps = 10^3 Bps (Bps is bits per second).
8. 1 GB = 1024 KBs; 1 KB = 1024 Bs; 1 B = 8 bs (b is bit).

Task: Answer the questions below in mili-seconds (msecs) rounded to three decimal places.

Question 1: What is the value of t_1? (1 mark)
Question 2: What is the value of t_2? (1 mark)
Question 3: What is the value of t_3? (2 marks)
Question 4: What is the value of t_4? (2 marks)
Question 5: What is the value of t_5? (2 marks)
Question 6: What is the value of t_6? (2 marks)
Question 7: What is the value of t_7? (3 marks)
Question 8: What is the value of t_8? (4 marks)

There are some information about Tranmission Links and Switches that will assist you on finishing the task:

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	7 Kbps	36,000 km	3 x 10^8 m/s
L3	S2	S4	100 Mbps	40 m	2 x 10^8 m/s
L4	S2	S3	100 Mbps	3,200 km	2 x 10^8 m/s
L5	S3	D1	100 Mbps	500 m	2 x 10^8 m/s
L6	S4	S	80 Mbps	60 m	2 x 10^8 m/s

Switches

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

Scenario 3 – The Internet's Directory Service (5 marks)

In this scenario, we now 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 containing the IP address of D3 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 containing the IP address of D4 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 containing the IP address of the required host name to D1.
8. D1 receives the DNS message, then forwards the message to Client C.

Task: Answer the questions below.
Question 1: Which on these events above that can combined together to form an recursive DNS query? (1 mark)
Question 2: Which on these events above can combined together to form an iterative DNS query? (1 mark)
Question 3: In the event 3, what is the DNS record type in the message (or packet) that D2 sends to D1? (1 mark)
Question 4: In the event 5, what is the DNS record type in the message (or packet) that D3 sends to D1? (1 mark)
Question 5: In the event 8, what is the DNS record type in the message (or packet) that D1 sends to C? (1 mark)

Part B (20 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.

Task: Answer the questions below.

Question 1: What is the web browser being used by the sender? (1 mark)
Question 2: What is the web server software being used in the remote server? (1 mark)
Question 3: Is protocol offloading being used at the client? (1 mark)
Question 4: How many routers are there between the client and server? (2 marks)
Question 5: 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 6: 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 7: 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? (1 mark)
Question 8: How long should the received javascript file be considered fresh? (1 mark)
Question 9: What is the initial value of the window scale shift count indicated by...

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

Question 10: 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 11: How many bytes are lost totally while tranmissing is indicated in the pcap file? (2 marks)
Question 12: How many duplicate acknowledgments are sent by the client regarding the missing bytes? (1 mark)
Question 13: In which frame is the lost frame retransmitted? (1 mark)
Question 14: In which frame does the receiver indicate that all missing frames have been received? (1 mark)
Question 15: From frame 1 to 3 (inclusive), how many frames have incorrect checksum value? (1 mark)

Hint:

1. Frame 4 shows a HTTP GET request from the client to the server, that could help you on answering question 6,7 and 8.
2. In frame 30, some data is lost, question 11, 12, 13 and 14 are focused in the lost frames.

Part C (50 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 the teaching staff. 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 RUSHBServer 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 the RDT (Reliable Data Transfer) protocol. It is expected that the RUSHB protocol is able to handle packet corruption, loss and encryption.

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. 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). Packets smaller than 1500 bytes need to be padded with 0 bits up to that size. In detail, the following figure describes the header structure of a RUSHB packet.

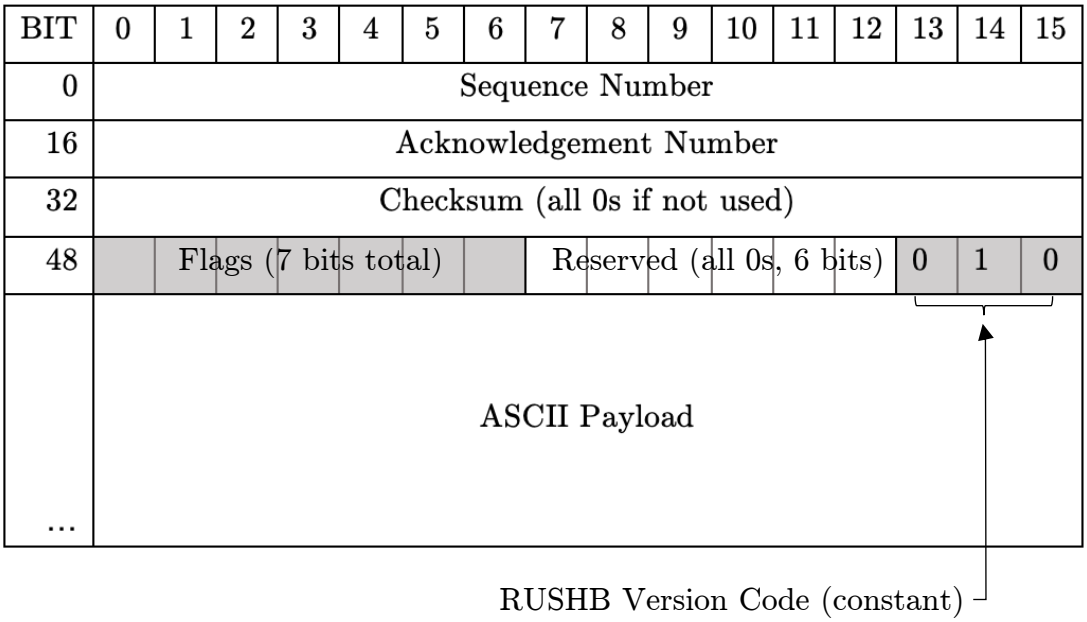

Figure 2. Structure of a RUSHB Packet

The client and server independently maintain sequence numbers. The first packet sent by either endpoint 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 Flags section in the next page) 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 to the figure 3 below.

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. Square brackets denote the flags set in each step (for example, [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] packet to the server.
   - The sequence number of this packet will always be 1.
   - The data section (ASCII payload) of this packet will contain the name of a resource (e.g. file.txt).
2. The server receives the [GET] message, then transmits the requested resource to the client over (possibly) multiple [DAT] packets.
   - The first packet from the server has a sequence number of 1.
3. The client acknowledges having received each data packet, by sending an [DAT/ACK] packet to each received one.
   - The acknowledgement number of this packet should be the sequence number of the packet being acknowledged.
4. After receiving the last acknowledgement [DAT/ACK] from the client, the server send [FIN] message to end the connection.
5. The client receives the [FIN] message from the server, then sends back [FIN/ACK] to the server.
6. 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).

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

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 checksum (note that network packets use big-endian as the byte order for its protocol):

ASCII Payload	`abcde`

Calculation	`0x6261 is ab`
	`0x6463 is cd`
	`0x0065 is  e`
	`------------`
	`0xc729`

Result	`~(0xc729) = 0x38d6`

Figure 4. Example checksum calculation

RUSHB protocol used a simple asymmetric encryption algorithm called RSA to (perhaps) secure the data transferred between host and clients. To simplify the use of this algorithm, the server and the client will both (share the same key) use [n; e] = [249; 11] as its public key and [n; d] = [249; 15] as its private key. We assume that **the client always knows the server public key, and the server always knows the client public key** (to reduce the difficulty of the assignment).

When the server receives the message with [ENC] from the client, the server will perform decryption using the algorithm below (in Pseudocode):

```
message = ''
for each letter i in payload:
    if i == 0 then break
    convert_to_dec(i)
    value = (i^d) % n
    message += convert_to_ascii(value)
```

Also, the server will perform encryption and send the message with encrypted payload (in Pseudocode):

```
encrypted = ''
for each letter i in payload:
    if i == 0 then break
    convert_to_dec(i)
    value = (i^e) % n
    encrypted += convert_to_ascii(value)
payload = encrypted
```

You do not need to know how RSA works; you just need to know how to use them as the following example:

1. The client sends a request [GET/ENC] packet to the server, assume that the packet is valid an uncorrupted. The packet payload is: ¸Ü†Ü

2. The server receives the packet, then analyses the received payload as the figure below.

ASCII Payload	¸Ü†Ü
Calculation	¸ -> 247 -> (247^15)%249=100 -> d Ü -> 220 -> (220^15)%249=97 -> a † -> 134 -> (134^15)%249=116 -> t Ü -> 220 -> (220^15)%249=97 -> a
Result	data

Figure 5. Decryption process at server

3. The server understands that the file it needs to send is called `data`, so it retrieves the content of the file `data` and send the encrypted messages to the client with [ENC] flag using the encryption algorithm above.

The following rules takes place when [ENC] is negotiated for all packets:

1. Encryption and decryption processes is only for ASCII Payload.
2. If [CHK] is negotiated, then the checksum computation takes precedence after the encryption/decryption has finished.
3. [ENC] request is only available on the client side, and server only follows the client requests.

Submission

Your server program must be written in Python, Java, C, 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 Java programs can run with version 8, any C programs have to be complied with the standard of C99 or GNU99. No external libraries or extensions is permitted for use in your program.

Python	`python3 RUSHBSvr.py`
Java	`make` `java RUSHBSvr`
C or C++	`make` `./RUSHBSvr`

Figure 6. 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 may delete any such directories before attempting to compile. 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 (10 marks)

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

- Listen on an unused port for a client's message
- 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 with GNU99 for your program and port 26041 was selected, your program invocation would look like this:

```
./3200Svr
26041
```

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 (5 marks)

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

- Perform all features outlined in the above section
- Successfully transmit a requested file over one or more packets
- 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 (10 marks)

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

- Perform all features outlined in the above sections
- Retransmit any packet on receiving a NAK for that packet
- 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 need not implement [CHK] or [ENC] for this part.

4 Packet Integrity [CHK] (5 marks)

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

- Perform all features outlined in the above sections
- 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 need not implement [ENC] for this part.

5 Secure Communication [ENC] (5 marks)

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

- Perform all features outlined in the above sections
- Implement the [ENC] mode negotiation and encode outgoing and decode incoming packets when ENC mode is in use

You must also ensure that if ENC was negotiated at the start, all packets are to be encoded before sending and decoded before reading.

6 Multiple-clients Handling (10 marks)

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

- Perform all features outlined in the above sections
- 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.

7 Internet Attack Prevention (5 marks)

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

- Perform all features outlined in the above sections
- Prevent Denial-of-Service attack from one client
- Prevent Denial-of-Service attack from multiple clients (distributed)

You should only do this part when finished all the parts above. Your server should detect and block the client if it receives equal or more than ten (10) [GET] or invalid packets from a client within 2 seconds from that client. Also, it should temporarily stop receiving and sending packets for 10 seconds if it gets more than thirty (30) [GET] or invalid packets from any client within 5 seconds. After 10 seconds of freezing, your program should refresh and start accepting new coming connections, and those clients that perform the attacks will be blocked from your server. Blocking a client means ignoring all incoming packet from that client. **Please note there could be some unexpected behaviours when dealing with concurrency**

(clocking and frequency), please research carefully before doing this task. Your program needs to distinguish between an authentic client and an attacker. A packet is called invalid when the server ignores them, such as a packet that is not in RUSHB protocol, or [ACK] and [NAK] with the wrong acknowledge number, or [GET] packet with a nonexistent file, etc. For example, client A sends a valid [GET/CHK] to the server, while server is transmissting the data to client A, client A send 9 packets with invalid checksum numbers, total of 10 packets happened in 2 seconds, and thus, the server blocks client A.

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.

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 teaching staff.

Tips and hints

1. Start your assignment early.
2. Finite state machine is a good way to design a program's work flow.
3. Check if the process send packets successfully by using Wireshark (listen on loopback).
4. Use RUSHBSimpleClient to debug your program, and test them on moss frequently using RUSHBSimpleTest.
5. Ask a teaching staff if you need any assistances.

Updates

0.1: initial release
0.2: fix checksum errors, add clarifications about RUSHB packet
0.3: edit and add clarification in part A, part C submission changed to RUSHBSvr
0.4: add clarification in part A
0.5: fix RSA errors
1.0: official release
1.1: fix typos