COMS3200/7201 Assignment 1

Due: 20th April 2020, 20:00

100 marks total (20% weight for the final grade (100%))

Part A (20 marks total)

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

A client requests a webpage from a remote server on a remote island via a slow satellite link above the earth in geostationary orbit. The goal of this exercise is to use the following information to calculate the time for the request to be completed.

The scenario is in no way intended to be realistic.

There is a client (C), a server (V), and a DNS server (D). These are connected by three switches (S1, S2, and S3) and five transmission links (L1 to L5) where L2 is the satellite link. The following tables provide further information about the network (bps == bits per second, 1 kbps == 10^3 bps, 1 Mbps == 10^6 bps).

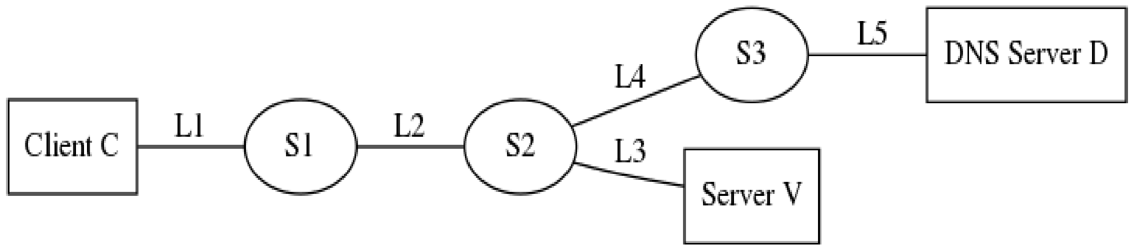
Transmission Links

Link Name	**Connected to**	**Connected to**	**Transmission Rate**	**Length**	**Propagation Speed**
L1	C	S1	100Mbps	300m	2×10^8 m/s
L2	S1	S2	5kbps	36,000km	3×10^8 m/s
L3	S2	V	100Mbps	75m	2×10^8 m/s
L4	S2	S3	100Mbps	3,000km	2×10^8 m/s
L5	S3	D	100Mbps	600m	2×10^8 m/s

Switches

Switch	**Links on Ports**	**Processing Delay**
S1	L1, L2	1ms
S2	L2, L3, L4	1ms
S3	L4, L5	250μs

The basic network topology is as below:



Assumptions

You may make the following assumptions about the network:

- The network is packet-switched
- The network starts with no other packets in queues - only packets that are a part of this question
- Each link is bidirectional and can concurrently handle bits travelling in opposite directions
- All required records are stored in the DNS server
- C, D, and V have no processing delays (while this is unrealistic, this simplifies calculations)
- DNS packets and TCP SYN, ACK, FIN, SYN/ACK, and FIN/ACK packets are 100 bytes long, including all headers, preamble, etc
- HTTP GET and HTTP response packets are 1000 bytes long, including all headers, preamble, etc

Scenario

At time t = 0, a program on the client (C), starts the process of retrieving a webpage from the server (V). The following events happen sequentially:

1. C sends a DNS request to D
2. D sends a response with V's IP address to C
3. C sends a TCP request (SYN) to open a connection to V
4. V acknowledges (SYN/ACK) the TCP request and opens the connections
5. C sends a HTTP GET request to V (which includes the ACK back to V for its SYN)
6. V sends the web page to C, which requires 5 packets, which includes the HTTP response plus the HTML file.
7. After receiving each data packet, C sends an ACK message back to V. Note that V does not need to wait for an ACK before sending the next data packet.
8. After receiving the last data packet acknowledgement, V sends a TCP FIN packet to close the connection.
9. After receiving FIN, C sends one FIN/ACK packet back to V.
10. After receiving FIN/ACK, V sends a final ACK packet back to C.
11. When this final ACK packet is received at C, the connection is finally closed.

Questions

Answer each of the following questions in the associated quiz on blackboard, following the specified instructions.
All answers should be in milliseconds(ms) rounded to four decimal places.

1. At what time does D receive the DNS request from Client C? (2 marks)

2. At what time does C receive the DNS response from D? (2 marks)

3. At what time does V receive the TCP SYN request from Client C? (2 marks)

4. At what time does C receive the TCP SYN/ACK acknowledgement from Server V? (2 marks)

5. At what time does V receive the HTTP GET packet from Client C? (2 marks)

6. At what time does C receive the first packet of the HTTP response from Server V? (3 marks)

7. At what time does C receive the fifth (last) packet of the HTTP response from Server V? (3 marks)

8. At what time is the connection between Client C and Server V closed? (4 marks)

HINT: Make sure you correctly convert between units

Part B (25 marks)

This question requires you to analyse the Wireshark file Assignment1b.pcap. This packet capture shows a client downloading a javascript resource file from a server, using TCP as a transport layer protocol. You will need to read some of the relevant IETF RFCs to understand some of the following questions. In particular:

- RFC793 is the base TCP spec
- RFC 791 describes the IP TTL option
- RFC2018 describes the TCP SACK option
- RFC7323 describes the Window Scale option

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

1. What is the web browser being used by the sender? (1mark)
2. What is the web server software being used in the remote server? (1 mark)
3. Is protocol offloading being used at the client ? (1 mark)
4. How many routers are there between the client and server? (1 mark)

Recall that 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 of TCP reassembly to understand the order of packet transmission.

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)

Frame 4 shows a HTTP GET request from the client to the server. Questions 6 and 7 focus on this frame.

6. What are the values of the 8 TCP flags (CWR to FIN) as a single 8-digit binary number? For example, if FIN was set and the others unset, you would write 00000001. (2 marks)
7. 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)
8. How long should the received javascript file be considered fresh (in seconds) ? (1 mark)

The following questions relate to window scaling. You should read RFC7323 before attempting these questions.

9. What is the initial value of the window scale shift count indicated by
    a. the client? (1 mark)
    b. the server? (1 mark)

10. The GET packet from the client to the server has an advertised window size of 46. What is the true window size in bytes? (3 marks)

In packet 30, some data is lost. Questions 11-13 focus on these lost frames.

11. How many bytes are lost? (4 marks)
12. How many duplicate acknowledgments are sent by the client regarding the missing bytes? (2 marks)
13. In which frame is the lost frame retransmitted? (2 marks)
14. In which frame does the receiver indicate that all missing frames have been received? (2 marks)

Part C (55 marks)

The RUSH2 protocol (Reliable UDP Substitute for HTTP version 2) is a HTTP-like stop-and-wait protocol that uses UDP in conjunction with the RDT rules. You have recently been hired by the multinational tech giant COMS3200 Inc, who have identified your deep knowledge in the field of secure transport-layer protocols. You have been tasked to develop a network server capable of sending RUSH2 messages to a client. It is expected that the RUSH2 protocol is able to handle packet corruption, loss and encryption. Your server program must be written in Python, Java, C, or C++.

NOTE: The RUSH2 protocol is not a real networking protocol. It has been created purely for this assignment.

Program Invocation

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, and any Java programs can run with version 8.

Python	C/C++	Java

`Python3 RUSH2Svr.py`	`make`
	`./RUSH2Svr`
		`make`
		`java RUSH2Svr`

IMPORTANT: As the assignment is auto marked, it is important that the filenames and command line syntax exactly matches the specification. Specification adherence is critical for passing.

RUSH2 Packet Structure

A RUSH2 packet can be expressed as the following structure (|| is concatenation):

IP-header || UDP-header || RUSH2-header || ASCII-payload

The data segment of the packet is a string of ASCII plaintext. Single RUSH2 packets must be no longer than 1500 bytes, including the IP and UDP headers (i.e. the maximum length of the data section is 1464 bytes). Packets that are smaller than 1500 bytes should be padded with 0 bits up to that size. The following table describes the header structure of a RUSH2 packet:

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Sequence Number															
16	Acknowledgement Number															
32	Checksum (all 0, if not used)															
48	Flags (7 flags)							Reserved (should be 0, 8 bits)								1

RUSH2 version code

The following sections describe each header in this packet further.

Sequence and Acknowledgement Numbers

Sequence numbers are independently maintained by the client and server. 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, RUSH2 sequence numbers are based on the number of packets as opposed to the number of bytes). When the ACK flag 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. Any packet that isn't a retransmission (including NAKs) should increment the sequence number.

Flags

The Flags section of the header is broken down into the following:

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

The purpose of these flags is described in the example below.

RUSH2 Example

The following situation describes a simple RUSH2 communication session. Square brackets denote the flags that are 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 RUSH2, unlike TCP, is not connection-oriented. There is no handshake to initialise the connection, but there is one to close the connection.

1. The client sends a request packet to the server [GET]
   - The sequence number of this packet will always be 1
   - The data section of this packet will contain the name of a resource (e.g. file.txt)
2. The server transmits the requested resource to the client over (possibly) multiple packets [DAT]
   - The first packet should have a sequence number of 1
3. The client acknowledges having received each data packet [DAT/ACK]
   - The acknowledgement number of this packet should be the sequence number of the packet being acknowledged
4. After receiving the last acknowledgement, the server signals the end of the connection [FIN]
5. The client acknowledges the connection termination [FIN/ACK]
6. The server acknowledges the client's acknowledgement and terminates the connection [FIN/ACK]

All RUSH2 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 RUSH2 Server will indicate if negotiation was successful by setting the appropriate [CHK], [ENC] flags. Once negotiated, these options are valid for all packets until a [FIN/ACK] closes the connection.

RUSH2 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 sum of all 16-bit words in the header". Once [CHK] is negotiated, all packets that have invalid checksums are considered corrupt.

The following are two examples of the checksum process.

ASCII payload	`abcde`	`at 12.30`
16 bit words	`0x6261 0x6463 0x65`	`0x7461 0x3120 0x2e32 0x3033`
Checksum	`0x38d6`	`0xfc18`

RUSH2 uses symmetric-key cryptography based on Caesar cipher with a key of 3. The entire ASCII table is rotated by the key value. This is used to encode the payload when using [ENC]. Note that the padding is not encoded and remains as 0 bits. In order to carry out the encode/decode process, RUSH2 encodes the ASCII string, one character at a time.

The following are two examples of the encryption outcomes.

ASCII payload	`ab &c`	`12.5%`
Encrypted content	`de#)f`	`4518(`

The following takes place when [ENC] is negotiated for all packets until [FIN/ACK].

1. Client encodes the ASCII plaintext payload
2. Client computes and adds the checksum if [CHK] was negotiated
3. Client sends the packet
4. On receiving a packet, the checksum is checked if [CHK] was negotiated.
5. The packet is decoded to obtain the original payload

Note that [CHK] takes precedence over [ENC]. This results in any corrupt packet being discarded early with enhances the RUSH2 performance.

Your Task

Basic Server Functionality (5 marks)

To receive marks in this section you need to have 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 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 Python was used and port 54321 was selected, your program invocation would look like this:

```
python3 RUSH2Svr.py
54321
```

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. Upon hearing from a client, your program can immediately signal the end of the connection (as described in the example). Once the FIN handshake has been completed, your program should terminate.

You may always assume that only one client will connect to the server at a time. For this section and the next you may also assume that no packets are corrupted or lost during transmission. You need not implement [CHK] or [ENC] for this part.

File Transmission (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 section
- Successfully transmit a requested file over one or more packets
- Receive (but not handle) ACKs from the client during transmission

When your server receives a GET packet, it should locate the file being requested and return the file's contents over one or more DAT packets. When complete, the server should close the connection (as in the above section). You may assume that the file being requested always exists (it is expected that this file is stored in your program's working directory). You need not implement [CHK] or [ENC] for this part.

Retransmission (15 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. If a NAK is received the timer should reset. How you choose to handle timeouts is up to you, however it must work on a UNIX machine (moss). Achieving this through multithreading, multiprocessing, or signals is fine provided if you only use standard libraries. You need not implement [CHK] or [ENC] for this part.

Packet integrity (15 marks)

RUSH2 is capable of detecting packet corruption using checksums.

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 CHK 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 CHK was negotiated at the start, all packets must have a valid checksum. You need not implement [ENC] for this part.

Secure communication (10 marks)

RUSH2 is also capable of packet level encryption.

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 and decoded.

Tips for Success

- Revisit the lectures and labs on reliable data transfer and TCP, ensuring you are familiar with the fundamentals
- Frequently test your code on moss
- Ensure your base functionality is working before attempting the more difficult tasks
- Start early - there will be limited help during the mid-semester break so any questions will need to be asked in labs beforehand

Library Restrictions

- The only communication libraries you may use are standard socket libraries which open UDP sockets
- You can't use any libraries that aren't considered standard for your language (i.e. if you have to download a library to use it should be considered as non-standard. You will need to implement CHK and ENC using standard libs only, writing your own code. Scapy is not permitted)
- If you are unsure about whether you may use a certain library, please ask the course staff on Piazza

Submission

Submit all files necessary to run your program. At a minimum, you should submit a file named `RUSH2Svr.py`, `RUSH2Svr.c`, `RUSH2Svr.cpp` or `RUSH2Svr.java`. If you submit a C/C++ or Java program, you should also submit a makefile to compile your code into a binary named `RUSH2Svr` or a .class file named `RUSH2Svr.class`.

IMPORTANT: If you do not adhere to this (e.g. submitting a C/C++/Java program without a Makefile, or a .class file instead of a .java file), you will receive 0 for this part of the assignment.

Marking

Your code will be automatically marked on a UNIX machine, **so it is essential that your program's behaviour is exactly as specified above.** Your program should complete all tasks within a reasonable time frame (for example a single packet should not take more than one second to construct and send) - there will be timeouts on all tests and it is your responsibility to make sure your code is not overly inefficient. It is expected that you will receive a small sample of tests and a basic RUSH2 client program before the submission deadline.

There are no marks for coding style.

Academic Misconduct

Students are reminded of the University's policy on student misconduct, including plagiarism. See the course profile and the School web page http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism.

Late Submission and Extensions

Please view the ECP for policies on late submissions and extensions as well as updates based on COVID-19 situation.

Version History

24/03/2020	Initial release	1.0
26/03/2020	Corrected Internet checksum example	1.1
09/04/2020	Improved RUSH2 header image, added checksum missing values	1.2