CSEE4119 Computer Networking HW3

Tong Wu, tw2906

Question 1

Part a

The purpose of adding sequence number to the packet sending between sender and receiver is to avoid receiver receiving duplicate packet.

Assume a situation that the sequence number is not been introduced yet, receiver receive a lossless packet and send a packet including ACK to the sender. However the ACK packet is corrupted (bit flipped), then the sender cannot check the checksum so it retransmit a duplicate packet to the receiver, and the receiver will not discard duplicate packet, which will cause data corruption.

With the sequence number added into the packet, the receiver will wait the packet with sequence number 1 after it ACK the packet with sequence number 0. If the ACK packet is corrupted and sender send a duplicate seq 0 packet, receiver will discard this packet and send a ACK packet for seq 0 packet again. It then prevent the duplicate receiving packet in receiver.

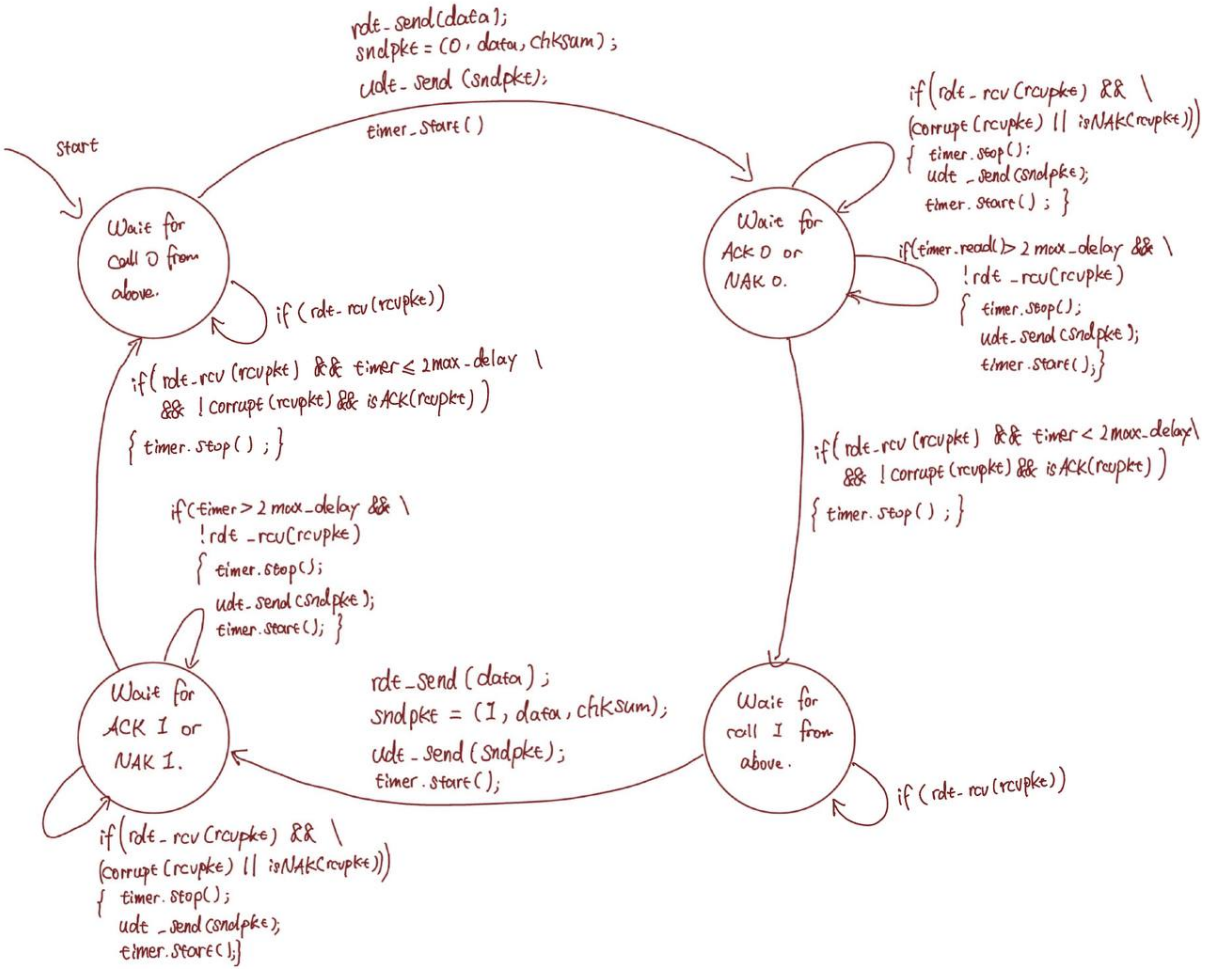
Part b

The underlying channel will not only corrupt packet, but also lost packet. In this case, the sender will not receive ACK packet so it will never know whether the packet is sent lossless or corrupted, the process will be stuck since the packet is loss. Timer for sender is to guarantee a continuous communication even a packet is loss. The sender will start countdown according to the timer once the packet is start sending to the receiver. If the ACK packet from receiver cannot be reached in the timer, then the sender will send a duplicate packet. No need to worry about the duplicate problem since it has been solved in rdt2.1.

Part c

Since the maximum delay is known, so it is possible to calculate the maximum wait time that sender can tolerance. Assume that the maximum delay includes a single way of RTT plus the processing time of the packet in both sender and receiver side. So the maximum wait time equal to $2\times$ maximum delay.

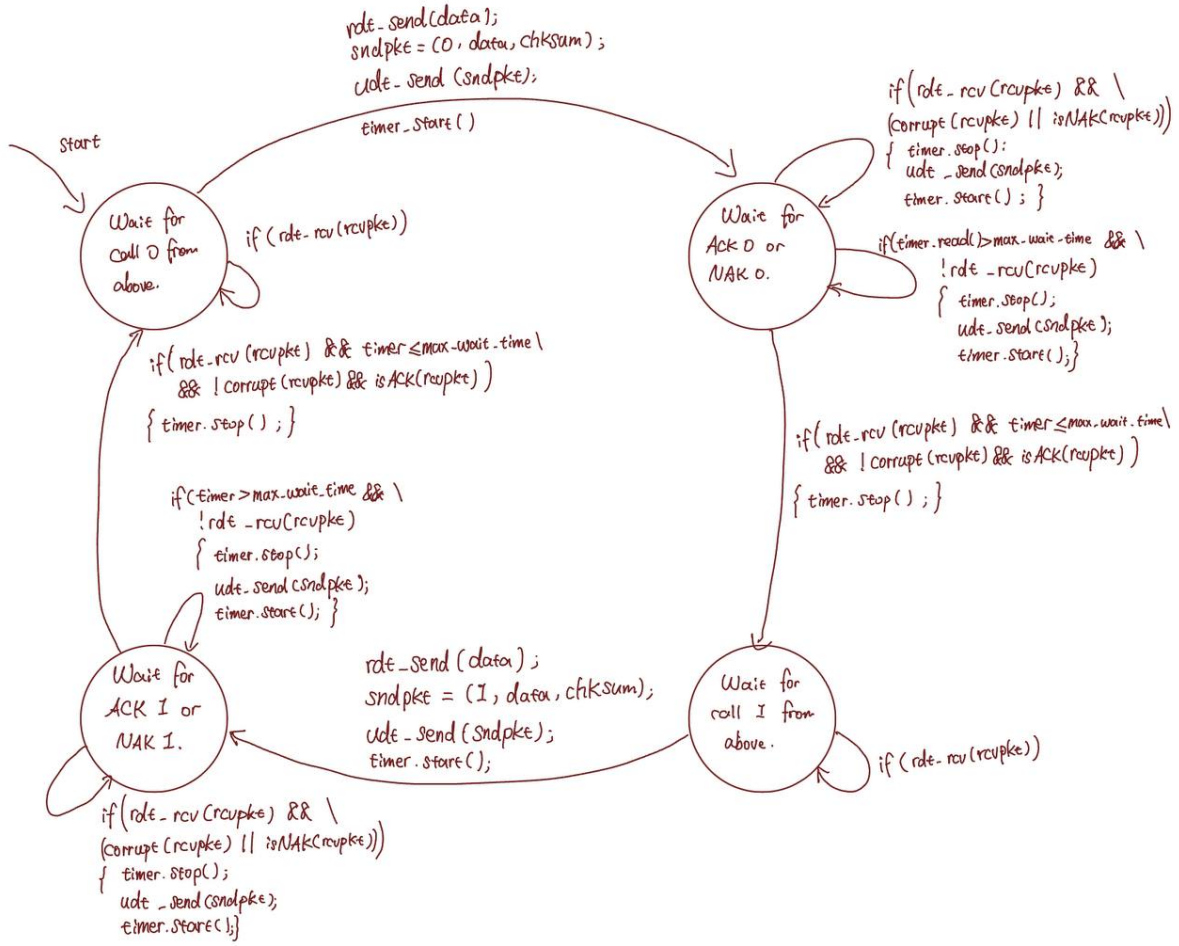
The new finite state machine can be drawn and has been attached below.



When the sender is waiting for the ACK 0/1 or NAK 0/1, the extra if condition is that the timer is larger than the twice of maximum delay, which indicates the time includes RTT and processing time that sender sends packet to receiver, plus the RTT and processing time that receiver sends packet to sender, and the receive packet is still empty. In this case, the sender will retransmit the packet to the receiver and restart the timer. It can then deal with the packet loss and maximum delay time problem.

Part d

If the maximum delay cannot be known, then the FSM in part c cannot work. Instead, a custom maximum wait time need to be defined by sender side. The new finite state machine can be drawn and has been attached below.



Where the max_wait_time should be defined by sender, which is the maximum tolerance time that sender can wait for the ACK packet.

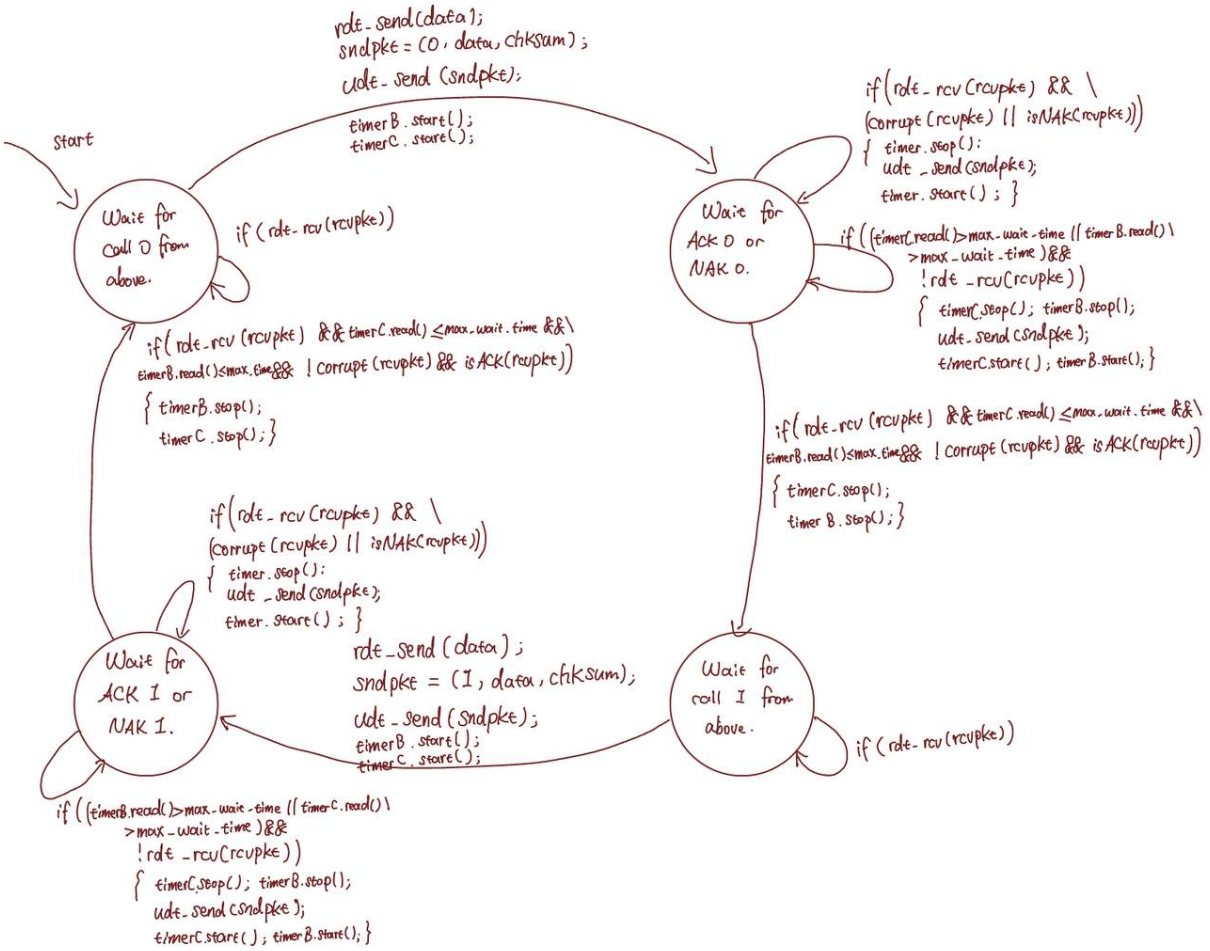
Part e

For a broadcast channel, if sender A upload a packet on the channel, then both B and C will receive the packet and return a ACK packet. The sender A need to confirm that both B and C receive the packet with lossless, so the tuple of the ‘senpkt’ of the receiver (B, C) should now in four, including sequence number, data, checksum, and the source hostname. Where hostname is used to identify the packet source.

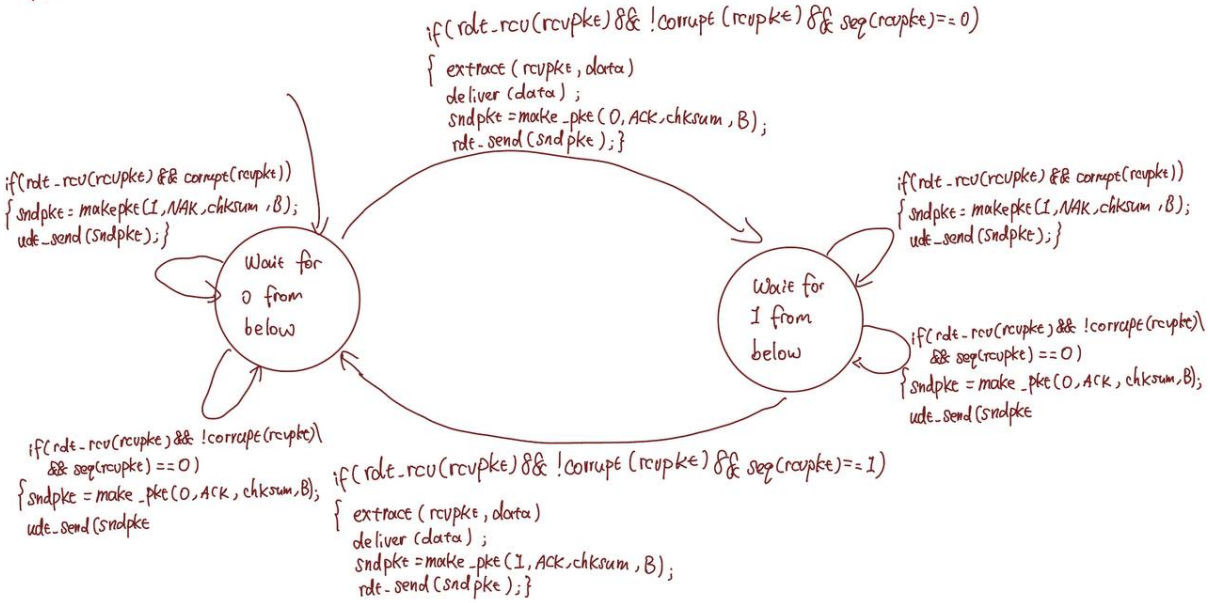
```
string sndpkt = (seq_num, data, checksum, hostname);
```

The FSM diagrams of sender and receiver are attached below, which the function used in the FSM should also be rewritten.

Sender (A):



Receiver (B, C):



Explanation:

The sender A first sends a packet with sequence number 0, data and checksum to the receiver B and C, also start the timer for B and C. Where timer is a class and ‘timerB’ and ‘timerC’ are objects for the class timer.

```
rdt.send(data);
sndpkt =  make_pkt(0, data, chksum, A);
udt.send(sndpkt);
timerB.start();
timerC.start();
```

Then for the sender side, it will wait until ACK/NAK from B and C is received. For the receiver side, it received packet from sender with sequence number 0, which meet the condition of FSM's first state. Then the receiver will demultiplexing the packet and check the checksum to determine whether ACK or NAK will be sent.

```
if (rdt_rcv(rcvpkt)) {
    if (corrupt(rcvpkt))
    {
        # For receiver B as an example, replace B to C for receiver C
        sendpkt = make_pkt(0, NAK, checksum, B);
        udt.send(sndpkt);
    } else
    {
        if (seq(rcvpkt) == 1)
        {
            # For receiver B as an example, replace B to C for receiver C
            sendpkt = make_pkt(1, ACK, checksum, B);
            udt.send(sndpkt);
        }
        else if (seq(rcvpkt) == 0) {state++;}
        else {error();}
    }
}
```

The sender side analysis the ACK packet from B and C if it is reached in maximum waiting time, and determine the next step. Note that the sender need to identify the source hostname of the packet, and goes to the next state only if both host send ACK and is received by sender A.

```
if (timerB.read() > max_wait_time || timerC.read() > max_wait_time)
{
    # Timeout, retransmit duplicate packet
    udt.send(sndpkt);
    # Restart timer
    timerB.restart();
    timerC.restart();
} else
{
    if (rdt_rcv(rcvpkt))
    {
        if (corrupt(rcvpkt) || isNAK(rcvpkt))
        {
            # Packet corrupted, retransmit duplicate packet
            udt.send(sndpkt);
            # Restart timer
            timerB.restart();
            timerC.restart();
        } else
        {
            if (seq(rcvpkt) == 1) {udt.send(sndpkt);}
```

```
            else if(seq(rcvpkt) == 0)
            {
                if (hostname(rcvpkt) == B)
                {
                    isBcompleted;
                    timerB.stop();
                }
                else if (hostname(rcvpkt) == C)
                {
                    isCcompleted;
                    timerC.stop();
                }
                else {error();}
            }
            else {error();}
        }
    }
}
if (isBcompleted && isCcompleted) {state++;}
```

Then the sender will sends a data with sequence number 1, the processes afterwards should be similar with the process introduced above.

Part f

Now consider transmit one packet first.

The time that receiver is successfully receiving packet can be defined as $\frac{L}{R}$, the idle time can be defined as RTT. So the total time that transmit a single packet should be $RTT + \frac{L}{R}$. There is a probability for failure to receiving a packet, so the average total time should plus the failure time $p(RTT + \frac{L}{R})$, which therefore should be $RTT + \frac{L}{R} + p(RTT + \frac{L}{R})$. So the utilization-under-failures should calculated as:

$$\frac{\frac{L}{R}}{RTT+\frac{L}{R}+p(RTT+\frac{L}{R})} = \frac{L/R}{(p+1)(RTT+L/R)}$$

Now consider timeout and retransmission.

The timeout is $1RTT$, while the total journey for a packet and a ACK packet is also $1RTT$, so the probability of losing a packet will not interrupt the current transmission of each packet. So the utilization-under-failures should remain as before.

If there is N packets, according to stop-and-wait operation, the sender will wait until the ACK packet received. So the utilization-under-failures should be:

$$\frac{NL/R}{N(p+1)(RTT+L/R)}$$

Part g

The utilisation can be calculated by $\frac{D_{trans}}{RTT+L/R}$, where $D_{trans} = L/R$. In this question, the sender will send all N packets and wait for an ACK from receiver indicates receive all N packets.

The utilisation is written below, where the time of transmission should be multiplied by N and the total time is still $RTT + L/R$ since it is pipelining.

$$\frac{NL/R}{RTT+L/R}$$

Part h

Each packet has independent probability p to fail, according to the principle of probability, for N packet, the probability that one of N packet fail is $\binom{N}{1} \times p$, where $\binom{N}{1} = N$, so $\binom{N}{1} \times p = Np$

The busy time still be NL/R, and the total time is $RTT + L/R + Np(RTT + L/R)$. So the utilisation should be:

$$\frac{NL/R}{RTT+L/R+Np(RTT+L/R)} = \frac{NL/R}{(Np+1)(RTT+L/R)}$$

Part i

Section i

Subsection 1

In rdt3.0, the D_{trans} is equal to L/R, where the L is the link speed, and R is length of packet. So it can be calculated as:

$$D_{trans} = \frac{L}{R} = \frac{4000}{10^6} = \frac{1}{250} = 4\,ms$$

For the utilisation, calculated by the equation $U_{sender} = \frac{L/R}{(p+1)(RTT+L/R)}$:

$$U_{sender} = \frac{4}{(1.01)(10+4)} = \frac{200}{707} \approx 0.283$$

Subsection 2

Using answer in part h, the utilisation of sender can be calculated as $\frac{NL/R}{(Np+1)(RTT+L/R)}$

$$U_{sender} = \frac{10\times4}{(10\times0.01+1)\times(10+4)} = \frac{200}{77} \approx 2.597$$

Section ii

Subsection 1

$$U_{sender} = \frac{L/R}{(p+1)(RTT+L/R)} = \frac{4}{(1.3)(10+4)} = \frac{20}{91} \approx 0.220$$

Subsection 2

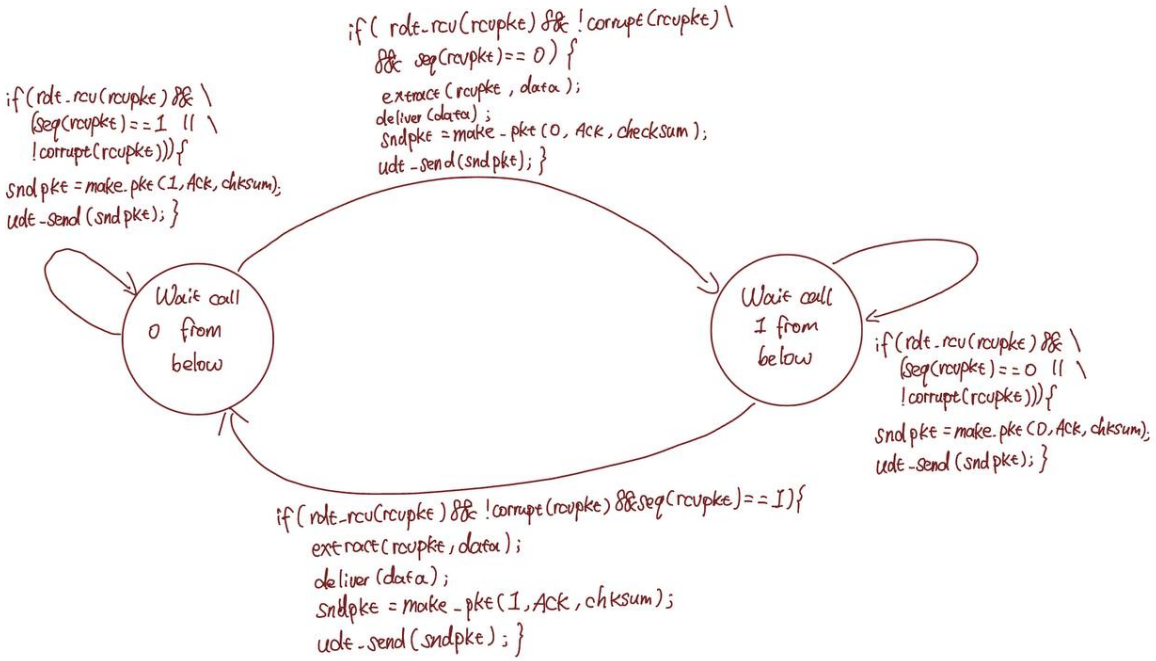
$$U_{sender} = \frac{NL/R}{(Np+1)(RTT+L/R)} = \frac{10\times4}{(10\times0.3+1)(10+4)} = \frac{5}{7} \approx 0.714$$

Part j

Under the high-loss situation, the transmit-N approach may not suitable, since it will retransmit all packets if there is one of the packet is loss or corrupted. It will take longer time then expected. The utilisation for transmit-N approach is decrease 350% from probability 0.01 to probability 0.3.

Question 2

Part a



Part b

Section a

Sequence number size defines how many different unique number that can sender sends in a specific round. Window size defines the maximum number of continuous packets that sender can sends if no ACK of these packets are received, also give the receiver a window to allow the packets in.

For selective repeat (SR), the receiver will send ACK packet for each packet that it received. If one of the ACK packet doesn't received by the sender, then sender will retransmit this packet after timeout. Imagine a extreme condition but have to be consider is that, all ACK packets sends from receiver lost. In this case, the window size will continue swipe forward, assume the window size is w and sequence number size is n. The packet sequence numbers that the receiver is allowed to receive is $[w+1,\ w+2,\ w+3,\ \ldots,\ w+w]$, which shows that the maximum sequence number that receiver can receive is $w+w=2w$. The max receive number cannot bigger than n, since if do so, the receiver will not know next following packet will be the last sequence or the current sequence because $2w$ is already in a new sequence. Hence the objective is to prevent the receiver swipe the receive window to the initial sequence number, so the max receive number $2w$ must small or equal to the sequence number size n, which can be written as $2w \leq n$. So the maximum of the window size is $\frac{n}{2}$.

Section b

For Go-Back-N approach, the receiver will send ACK packet for each packet that it received. If one of the ACK packet doesn't received by the sender, then sender will retransmit this and following packets after timeout. Also need to imagine a extreme condition same as in section a, which is all ACK packets lost. In this case, the sender will sends all packets that it need to send in a window size. Assume window size is w and sequence number size is

n, then the sender will retransmit $[1, 2, 3, \ldots, w]$, and the receiver is now waiting for the new sequence of packet, stating from $[1, 2, 3, \ldots, n]$. In this case, w cannot equal or greater than n since if it is, the receiver will ACK all packets in a sequence and start receiving a new sequence starting with number 1. However, the sender will not know whether receiver ACKed all packets or not, it just know that all ACK packets are not received so sender will retransmit from sequence number 1. Receiver will then identify the new coming packet as for the new sequence, which is not allowed. So the window size w must smaller than sequence number size n, so the maximum of the window size is $n - 1$.

Part c

Section i

True

Assume the situation that all ACK packets are lost, and sequence number size is $[0, 1, 2, 3, 4]$ and window size is 2. After receiver sends ACK for packet 1, the allowance window now is $[2, 3]$, while the sender will retransmit packet 0 since it did not receive the ACK for packet 0, which is fall outside the receiver's window.

Section ii

False

It is impossible for the sender to receive a ACK outside its window since the sender's window is changing by the ACK from receiver.

Section iii

True

The sender window in GBN will only swipe forward after receiving the first packet in current sequence, and the receiver will keep sending the previous packet before the loss packet when subsequent packets are coming. So assume the packet 1 is loss, then receiver will keep sending ACK 0 when it receive the subsequent packets. In this case the sender window is $[1, 2, 3, \ldots, w]$, where subsequent ACK 0 will outside the window.

Section iv

False

Continue use the example in section iii, the receiver expect the sender sends packet number 1, which is in the window that sender has.

Question 3

Part a

For the TCP communication, assume that host A and B always use 0 as the initial sequence number after the handshaking. In this case and each packet has 's' bits of data, `packet(s)` , the first packet that two hosts send must has a tuple `dest_addr=B, souce_port=x, dest_addr=A, dest_port=x, seq=0, ack=s, data=''` . Where the source port and destination port is both X which is indicates in the question, sequence number is 0, and the ack is 's' shows that it is ready for receiving packets with seq 's' since this packet has s bit of data and next packet will comes with sequence number s . Host C as attack node, can try sends packet with tuple `dest_addr=B, souce_port=x, dest_addr=A,`

`dest_port=x, seq=0, ack=s, data=''` to host A multiple times, as each TCP connection established the first packet will be this but may not sure whether comes from host A or B.

So it cannot be 100% of success but it can has a relatively higher probability of success than using random initial sequence number. For random initial sequence number, attacker node C cannot 'know' which sequence number that host A and B are using even for the first packet sending. So if C wants to spoof packet(s), it will be harder to guessing a correct sequence number.

Part b

`rwnd, cwnd`

`rwnd` is a TCP state variable for the TCP receive window to deal with the size of the bucket (the capacity of receiver).

`cwnd` is a TCP state variable for the TCP congestion window to deal with the narrow part of the pipe of network.

Use these two attributes to tell host to send how much of packets can avoid full-drop from receiver and congestion from the network.

Part c

Slow-start in the initial start-up of the connection is in order to guarantee the speed of network. Slow-start is able to find a best connection speed through continuously increase the value of `cwnd` without congest the network.

Part d

During the slow start, the increase trend for sending rate of B in each round is `cwnd=cwnd*2` .

Part e

1. If `cwnd > ssthresh` , which means the receive window is larger than the threshold, then B will exit slow-start and start congestion avoidance.
2. If `dupACKcount == 3` , which means there is three packets received by C after B's packet lost or delayed, then C sends duplicate ACK packet to B. In this case, B will:
    1. retransmit missing segment
    2. `ssthresh=cwnd/2`
    3. `cwnd=cwnd/2`

Part f

During the congestion avoidance, the increase trend for sending rate of B in each round is `cwnd++`

Part g

BDP:

Bandwidth delay product can be calculated by multiplying the link speed and the RTT.

$$\text{BDP} = B \times D = 500Mbit/s \times 50ms = 2.5 \times 10^7\ bit = 25\ Mbit = 3.125MByte$$

TCP handshaking phase:

The outgoing packet size is the Ethernet Packet Size without the TCP payload, which is $1500\ bytes - 1460\ bytes = 40\ bytes$. In 3-way handshaking, there will send 3 packets to establish the handshake. The total time cost in TCP handshaking phase can be calculated below:

File upload time: $\frac{40}{500M} \times 8 = 6.4 \times 10^{-7} s$

Transmit time: $1 \times RTT$

$$\text{TotalTime} = RTT + 6.4 \times 10^{-7} s = 50ms + 6.4 \times 10^{-4} ms$$

File transmission phase:

Note that the `ssthresh` can be assumed as infinity, and the BDP is $25\ Mbit = 3.125 MByte$. Using slow-start:

First transmission: `cwnd=32`

Upload size: $32 \times 1500B = 48000B$

Upload time: $\frac{48000}{500M} \times 8 = 7.68 \times 10^{-4} s = 0.768ms$

Total time: $0.096ms + RTT = 0.768ms + 50ms = 50.768ms$

Receive window remains: $3.125MB - 48000B = 3077000B$

File size remains: $1MB - 1460B \times 32 = 953280B$

Second transmission: `cwnd=64`

Upload size: $64 \times 1500B = 96000B$

Upload time: $\frac{96000}{500M} \times 8 = 1.536 \times 10^{-3} s = 1.536ms$

Total time: $1.536ms + RTT = 1.536ms + 50ms = 51.536ms$

Receive window remains: $3.125MB - 48000B - 96000B = 2981000B$

File size remains: $1MB - 1460B \times 32 - 1460B \times 64 = 859840B$

Third transmission: `cwnd=128`

Upload size: $128 \times 1500B = 192000B$

Upload time: $\frac{192000}{500M} \times 8 = 3.072 \times 10^{-3} s = 3.072ms$

Total time: $3.072ms + RTT = 3.072ms + 50ms = 53.072ms$

Receive window remains: $3.125MB - 48000B - 96000B - 192000B = 2789000B$

File size remains: $1MB - 1460B \times 32 - 1460B \times 64 - 1460B \times 128 = 672960B$

Fourth transmission: `cwnd=256`

Upload size: $256 \times 1500B = 384000B$

Upload time: $\frac{384000}{500M} \times 8 = 6.144 \times 10^{-3} s = 6.144ms$

Total time: $6.144ms + RTT = 0.768ms + 50ms = 56.144ms$

Receive window remains:
$3.125MB - 48000B - 96000B - 192000B - 384000B = 2405000B$

File size remains:
$1MB - 1460B \times 32 - 1460B \times 64 - 1460B \times 128 - 1460 \times 256 = 299200B$

Fifth transmission: `cwnd=512`

Note that the file size remain can only further transmit $\lceil 299200/1460 \rceil = 205$ packets

Upload size: $205 \times 1500B = 307500B$

Upload time: $\frac{307500}{500M} \times 8 = 4.92 \times 10^{-3}s = 4.92ms$

Total time: $4.92ms + RTT = 54.92ms$

Receive window remains:
$3.125MB - 48000B - 96000B - 192000B - 384000B - 307500B = 2097500$

Total time:

$$50ms + 6.4 \times 10^{-4}ms + 50.768ms + 51.536ms + 53.072ms + 56.144s + 54.92ms = 316.44064ms$$

Part h

According to the previous answer, with the MSS=1460, and RTT=50ms. The loss rate of 200 Mbps will be $2 \times 2 \times 10^{-8} = 4 \times 10^{-8}$

The TCP throughput can be calculated as $\frac{1.22 \times 1460}{50m \times \sqrt{4 \times 10^{-8}}} = 178.12Mbps$

If the initial `cwnd` is low, it will take much longer time to transmit a whole file since the start up phase takes quite long time.

Part i

The timeout shows the serious situation of the network congestion straightforwardly, so it need to be solved by drop the `cwnd` to zero.

While for the triple duplicate ACK, it just means the some packet is lost during the transfer, but may not clearly indicate it is caused by the congestion. However, cut `cwnd` to half is an insurance action.

Question 4

Part a

For congestion avoidance mode:

Round 5 - Round 21

Round 25 - Round 42

Part b

`ssthresh=16`, since it changes to congestion avoidance with `cwnd=16`, so `ssthresh=cwnd=16`.

Part c

During the round 21, the window size `cwnd` cut to half and switch the mode to slow-start.

It can indicate that there must has one or more packet lost, since only triple duplicate ACK will active the fast recovery, which is cut `cwnd` a half and goes to slow-start.

Part d

Packet timeout

Part e

May not. It is also caused by start to transmit a new file, which needs to start a new connection and starting with initial `cwnd` .

Part f

`ssthresh=16` , before round 21, the `ssthresh=16` , in round 21, triple duplicate ACK occurs and `ssthresh=cwnd/2=32/2=16` . Round 21 to 23 is the fast recovery time, so `ssthresh` keep 16.

Part g

Before round 41, the `ssthresh` is always 16, so recover to 16 from `cwnd = 1` needs 4 rounds.

Part h

Fast recovery. After the triple duplicate ACK occurs, it changes mode into fast recovery doing slow-start.

Part i

The ACK 20000 should first appear in round 19, and duplicate appear 3 times until round 21 to trigger a fast recovery. So the SEQ in round 24 should add all packets sends from round 19-23, which is $20000 + 30 + 31 + 32 + 19 + 20 = 20132$

Part j

Plus maximum packet sending form transmission rounds 1 to 24, which is
$1 + 2 + 4 + 8 + (16 + 17 + \ldots + 32) + 19 + 20 + 24 = 78 + \frac{(16+32)\times 17}{2} = 78 + 408 = 486$

Part k

After 3 rounds (round 24), the new ACK received, so change mode to congestion avoidance and `cwnd=ssthresh=16` .

Question 5

Part a

Buffer bloat problem caused by overlarge buffer in router or switch. The loss-based congestion control will monitor the status of packet loss then adjust the send window.

The buffer size of router or switch, represented by `ssthresh` , and the change of host's congestion window, `cwnd` is depends on the value of `ssthresh` . In reno, slow-start will be used when `cwnd < ssthresh` in order to quickly fill the buffer as a quickest speed, then change to congestion avoidance to ensure best speed and latency. However, a overlarge buffer which is not coupling with capacity link of the network will not improve the delay problem, but will make a backlog of packets and increase the delay.

The reason is that the congestion control algorithm will not change to congestion avoidance mode unless the `cwnd >= ssthresh`, while `ssthresh` is overlarge so the algorithm will keep the slow-start mode.

Part b

The reason that TCP and UDP cannot be updated as quickly as application may that, the application layer need to acquire much more other 'packets' from peripherals or localhost, which are not required to output to the transport layer, and these packets needs more frequent update rate than the normal rate that UDP and TCP requires.

QUIC is efficient since it assembled part of protocols in application, security and transport layer [1], and can multiplexing the streams from a connection, and reassembled to the application [1].

Part c

The BBR uses the maximum bandwidth and RTT for the most recent packet acknowledgement to measure the send rate [2].

Part d

BBR make sure that in-flight packets is a small multiple of the BDP, which can lower the buffer using rate and results low delay [3].

Part e

TCP Vegas determine the `cwnd` by using RTT. Measure the expected and current throughput, and compare its different with the maximum and minimum threshold, then enlarge or decrease the `cwnd` according to the compare result [4]. TCP Vegas improve the performance in network but cannot balance it with the fairness [5].

In order to improve the fairness, BBR uses pacing rate to control the inter-packet spacing [6].

Part f

The router/switch on the network layer using drop-tail to queuing the packets in the buffer, which means the last coming packet will be dropped if the buffer of the router/switch is full [7]. BBR v2 use the loss packet as a signal, with an explicit target loss rate ceiling [8].

Since the improvement in BBR v2, the BBR's throughput does not depend on the number of loss packets or flows, but depends on the router loss packet signal, which makes it more likely a congestion algorithm.

References

[1] Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., Yang, F., Kouranov, F., Swett, I., Iyengar, J., Bailey, J., Dorfman, J., Roskind, J., Kulik, J., Westin, P., Tenneti, R., Shade, R., Hamilton, R., Vasiliev, V., … Shi, Z. (2017). The QUIC Transport Protocol: Design and Internet-Scale Deployment. *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* , 183–196.

[2] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., & Jacobson, V. (n.d.). *BBR Congestion-Based Congestion Control* . 34.

[3] Vargas, S., Drucker, R., Renganathan, A., Balasubramanian, A., & Gandhi, A. (2021). BBR Bufferbloat in DASH Video. *Proceedings of the Web Conference 2021* , 329–341.

[4] Basic concept of TCP-Vegas. (2020, December 31). *GeeksforGeeks* . https://www.geeksforgeeks.org/basic-concept-of-tcp-vegas/

[5] *Fairness Comparisons Between TCP Reno and TCP Vegas for Future Deployment of TCP Vegas* . (2016, January 3). https://web.archive.org/web/20160103040648/http://www.isoc.org/inet2000/cdproceedings/2d/2d_2.htm

[6] Cao, Y., Jain, A., Sharma, K., Balasubramanian, A., & Gandhi, A. (2019). When to use and when not to use BBR: An empirical analysis and evaluation study. *Proceedings of the Internet Measurement Conference* , 130–136.

[7] Fumie Costen. (n.d.). *EEEN30024 Data Networking 2021–22 1st Semester. Theory 5: Network Layer* . The University of Manchester.

[8] Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., Vasiliev, V., Jha, P., Seung, Y., Mathis, M., & Jacobson, V. (n.d.). *BBR v2 A Model-based Congestion Control* . 36.