Jason Lai
204995126
Dis 1A

CS 118
April 26, 2020

Homework 3

1 Problem 1

The sender side of *rdt3.0* simply ignores (that is, takes no action on) all received packets that are either in error or have the wrong value in the acknum field of an acknowledged packet. Suppose that in such circumstances, *rdt3.0* were simply to retransmit the current data packet. Would the protocol still work? (Hint: Consider what would happen if there were only bit errors; there are no packet losses but premature timeouts can occur. Consider how many times the *nth* packet is sent, in the limit as n approaches infinity).

Yes, the protocol would still work, since rdt3.0 retransmits on a timeout if the sent packet or ACK packet was lost, so retransmitting on errors would be feasible, albeit inefficiently in some cases. For instance, if we consider the hint, then in the event that a sent packet was received correctly, but the corresponding ACK had errors, then the sender will retransmit the packet when in reality, it's unnecessary. However, the real issue arises if there are consecutive ACKs with errors, which increases network traffic due to the amount of retransmitted packets and their subsequent ACKs being sent back and forth. With this increase network traffic, there may be greater network delay, which can lead to premature timeouts on the sender side, which in turn leads to even more retransmitted packets and therefore greater network traffic, thus exacerbating the problem. This means that the number of times that the *nth* packet is sent could increase indefinitely as n approaches infinity.

2 Problem 2

Consider a reliable data transfer protocol that uses only negative acknowledgments. Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why? Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

If the sender sends data only infrequently, then a NAK-only protocol would not be preferable to one that uses ACKS. The reason is because if a sent packet were to be lost, the receiver would have no way of knowing until it receives the next packet that has a sequence number it was not expecting, and since data is sent infrequently, this could take a long time. Only then, would the receiver send a NAK back to the sender asking to retransmit the lost packet, which is a long amount of time to recover from packet loss as opposed to using ACKs with a countdown timer.

If the sender has a lot of data to send and the end-to-end connection experiences few losses, then the NAK-only protocol would be preferable to one that uses ACKS. The reason is because, with frequent data transfer, the receiver can now realize packet losses much faster, and if the end-to-end connection experiences few losses, then the number of NAKs sent back to the sender would be relatively few in number. Furthermore, the network traffic would be less congested than with a reliable data transfer protocol that uses ACKs, since NAKs are not sent back to the sender with every received packet like ACKs are.

3 Problem 3

Consider the GBN protocol with a sender window size of 6 and a sequence number range of 1,024. Suppose that at time t, the next in-order packet that the receiver is expecting has a sequence number of k. Assume that the medium does not reorder messages. Answer the following questions:

(a) What are the possible sets of sequence numbers inside the senders window at time t? Justify your answer.

(b) What are all possible values of the ACK field in all possible messages currently propagating back to the sender at time t? Justify your answer.

a) The possible sets of sequence numbers inside the senders window at time t are the packets in the range $[k,\ k+5]$ and the packets in the range $[k-6,\ k-1]$.

This is because we have to consider the two cases where the sender receives the ACK for the previous $N = 6$ packets before k or not. If the sender receives a ACK for the previous N packets, then the next N packets in the range $[k,\ k+N-1]$ are in the window. If the sender did not receive an ACK for the previous N packets, then the sender must retransmit all N packets from the previous batch, so the packets in the range $[k-N, k-1]$ are in the window.

b) All possible values of the ACK field in all possible messages currently propagating back to the sender at time t are in the range $[k-7,\ k-1]$.

This is because we know for a sender window size of $N = 6$, if the receiver is expecting a packet with sequence number k, then the sender sent and the receiver received the packets $[k-N, k-1]$. And if the sender sent the packets $[k-N, k-1]$, that means that the sender sent and already received an ACK for the N packets before that in the range $[k-2N,\ k-N-1]$. However, depending on the situation, the sender may have generated a duplicate cumulative ACK for the packets $[k-2N,\ k-N-1]$ which may be a message currently propagating back to the sender with the value $k-N-1$ in the ACK field. Furthermore, if the receiver is expecting packet k, this does not mean that the sender has a received an ACK for the previous N packets. So, for the packets in the range $[k-N, k-1]$, the ACK(s) for these packets could still be propagating back to the sender. However, if the network is experiencing congestion causing delay or if there is some packet loss, the receiver will generate a cumulative ACK for all packets it has received so far from the range $[k-N,\ k-1]$. This means that the ACK field for this ACK could be any of the values from $[k-N,\ k-1]$, since any of the packets may have been lost or are taking a long time to arrive, and the receiver's timer would then timeout, causing a cumulative ACK to be generated. Therefore, the all possible values of the ACK field in all possible messages currently propagating back to the sender at time t are in the range $[k-N-1,\ k-1]$.

4 Problem 4

Follow the same problem setting in Page 62 of Slides Chapter3-2020.ppt. Suppose packet size is 4KB (*i.e.* 4000 bytes), bandwidth is 8Mbps, and one-way propagation delay is 20 msec. Assume there is no packet corruption and packet loss.

(a) Suppose sener window size is 5, will the sender be kept busy? If yes, explain why. If not, What is the effective throughput?

(b) What is the minimum sender window size to achieve full utilization? Then how many bits would be needed for the sequence number field?

a) No, the sender will not be kept busy, as the utilization of the sender would be 0.333, meaning that the sender is only busy sending 33.33% of the time. The effective throughput is 454.55 kilobytes per second, as it take $RTT + \frac{L}{R} = 0.044$ seconds from the time that the first packet of a 5-packet window is sent to the time that the first packet of the next 5-packet window is sent. Since we only send 5 packets of size 4KB in this amount of time, our throughput is 454.55KBps.

b) To achieve full utilization, the minimum sender window size would need to be 11 and the sequence number field would need 4 bits.

The window size would need to be 11, as this would mean that once the pipeline has reached a steady state where the pipeline is completely full, the sender is constantly busy. This is because, by the time an entire window of 11 packets is sent ($\frac{11L}{R} = 0.044$ seconds), the ACK for the first packet of that window will have arrived at the sender ($RTT + \frac{L}{R} = 0.044$ seconds). This means the sender will be able to send the first packet of the next window as soon as the 11th packet of the previous window is fully sent. With no packet loss, we would need sequence numbers from 0 to 10; this would require 4 bits in the sequence number field, since we need 4 bits to represent the numbers 9 and 10. The reason we only need 11 sequence numbers is because, with no packet loss or corruption, we will never send duplicate packets, and therefore not need to worry about confusing the receiver with whether a packet with a certain number is from a retransmitted batch or part of the next batch.

Work:

$n = 5$ packets
$L = 4000 \times 8 = 32000$ bits
$R = 8 \times 1 \times 10^6 = 8 \times 10^6$ bps
$RTT = 0.02 \times 2 = 0.04$ seconds

a)

$$U_{\text{sender}} = \frac{\frac{nL}{R}}{RTT + \frac{nL}{R}} = \frac{\frac{5 \times 32000}{8 \times 10^6}}{0.04 + \frac{5 \times 32000}{8 \times 10^6}} = 0.333$$

$$\text{Throughput} = \frac{5L}{RTT + \frac{L}{R}} = \frac{5 \times 32000}{0.04 + \frac{32000}{8 \times 10^6}} = \frac{3636363.636}{1000} = \frac{3636.3636}{8} = 454.55 \text{ KBps}$$

b)

$$\frac{nL}{R} = RTT + \frac{L}{R} => \frac{n32000}{8 \times 10^6} = 0.04 + \frac{32000}{8 \times 10^6}$$

$$=> n = 11 \text{ packets}$$

5 Problem 5

Answer True or False to the following questions and briefly justify your answer:

(a) With the Selective Repeat protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.

(b) With Go-Back-N, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.

(c) The Stop-and-Wait protocol is the same as the SR protocol with a sender and receiver window size of 1.

(d) Selective Repeat can buffer out-of-order delivered packets, while GBN cannot. Therefore, SR saves network communication cost (by transmitting less) at the cost of additional memory.

a) True. Consider a sender with a window size of n, the sender sends n packets in the range $[k,\ k+n-1]$ to the receiver. The receiver receives the packets $[k,\ k+n-1]$ and sends ACKs for each of the n packets. However, then consider that the sender's timer times out, so the sender resends all of the n packets before the ACKs arrive. Then the ACKs arrive, so the sender's window will slide to the next n packets in the range $[k+n,\ k+2n-1]$. Meanwhile, the receiver will receive the duplicate set of packets $[k,\ k+n-1]$ and sends a duplicate set of ACKs for them to the sender. The sender will then receive this second set of ACKs for the packets $[k,\ k+n-1]$, while the window is currently contains the range $[k+n,\ k+2n-1]$, so it has received ACKs for packets that fall outside of its current window.

b) True. Like in part **a)**, the sender's timer may timeout before the cumulative ACK for its previous window of packets arrives, so it sends a duplicate set of packets to the receiver. The sender may then receive the cumulative ACK and slide its window to the next set of packets, while the receiver will receive the duplicate set of packets and sends a second cumulative ACK for the duplicate packets back to the sender. The sender will then receive this ACK for a set of packets that falls outside of its current window.

c) True. In essence, Selective Repeat is the Stop-and-Wait protocol with pipelining. If Selective Repeat is used with a window size of only 1 for the sender and receiver, then it is the same as Stop-and-Wait, since Selective Repeat will only send 1 packet at a time and will wait for the receiver's ACK before sliding the window and sending the next packet. In principle, this is the same as Stop-and-Wait, which also sends 1 packet at a time, and waits for the receiver's ACK before sending the next packet.

d) True. The receiver in Go-Back-N protocol does not buffer out-of-order delivered packets and instead discards them. The sender must then resend all packets starting in-order

from the first packet which was not included in a cumulative ACK, so even the out-of-order packets that the receiver received successfully will be retransmitted. However, in Selective Repeat protocol, the receiver can buffer out-of-order packets while it waits for the sender to retransmit any missing packets. Since the receiver sends ACKs individually for each packet, the sender will only retransmit packets for which it has not received an individual ACK for and has timed out on, so the sender does not need to resend out-of-order packets that came after a lost packet, since those out-of-order packets will have had an ACK sent to the sender. Therefore, Selective Repeat does save network communication cost by only sending packets that were lost in transmission at the cost of additional memory to buffer out-of-order packets, unlike Go-Back-N.