Chap 3 Homework 3

Zilong Wang 18281218 CS1804 wangzilong@bjtu.edu.cn

(P5) a. Suppose you have the following 2 bytes: 01001010 and 01111001. What is the 1s complement of sum of these 2 bytes?

b. Suppose you have the following 2 bytes: 11110101 and 01101110. What is the 1s complement of sum of these 2 bytes?

c. For the bytes in part (a), give an example where one bit is flipped in each of the 2 bytes and yet the 1s complement doesn't change.

Answer:

a. Adding the two bytes gives 10011101. Taking the one's complement gives 01100010

b. Adding the two bytes gives 00011110; the one's complement gives 11100001.

c. first byte = 00110101 ; second byte = 01101000.

(P6) Consider our motivation for correcting protocol rdt2.1. Show that the receiver, shown in

the figure on the following page, when operating with the sender show in Figure 3.11, can lead the sender and receiver to enter into a deadlock state, where each is waiting for an event that will never occur.

Answer: Suppose the sender is in state "Wait for call 1 from above" and the receiver (the receiver shown in the homework problem) is in state "Wait for 1 from below." The sender sends a packet with sequence number 1, and transitions to "Wait for ACK or NAK 1," waiting for an ACK or NAK. Suppose now the receiver receives the packet with sequence number 1 correctly, sends an ACK, and transitions to state "Wait for 0 from below," waiting for a data packet with sequence number 0. However, the ACK is corrupted. When the rdt2.1 sender gets the corrupted ACK, it resends the packet with sequence number 1. However, the receiver is waiting for a packet with sequence number 0 and (as shown in the home work problem) always sends a NAK when it doesn't get a packet with sequence number 0. Hence the sender will always be sending a packet with sequence number 1, and the receiver will always be NAKing that packet. Neither will progress forward from that state.

(P8) In protocol rdt3.0, the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?

Answer: To best Answer this question, consider why we needed sequence numbers in the first

place. We saw that the sender needs sequence numbers so that the receiver can tell if a data packet is a duplicate of an already received data packet. In the case of ACKs, the sender does not need this info (i.e., a sequence number on an ACK) to tell detect a duplicate ACK. A

duplicate ACK is obvious to the rdt3.0 receiver, since when it has received the original ACK it transitioned to the next state. The duplicate ACK is not the ACK that the sender needs and hence is ignored by the rdt3.0 sender.

(P9) Give a trace of the operation of protocol rdt3.0 when data packets and acknowledgment packets are garbled. Your trace should be similar to that used in Figure 3.16

Answer: Suppose the protocol has been in operation for some time. The sender is in state "Wait
for call from above" (top left hand corner) and the receiver is in state "Wait for 0 from below". The scenarios for corrupted data and corrupted ACK are shown in Figure 1.

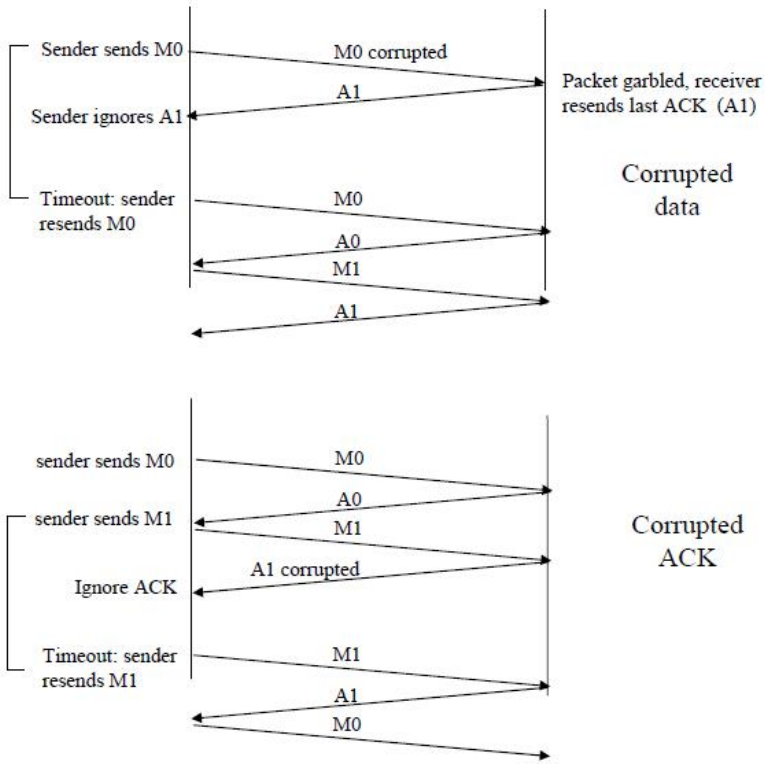


Figure 1: rdt 3.0 scenarios: corrupted data, corrupted ACK

(P10) Consider a channel that can lose packets but has a maximum delay that is known. Modify protocol rdt2.1 to include sender timeout and retransmit. Informally argue why your protocol can communicate correctly over this channel.

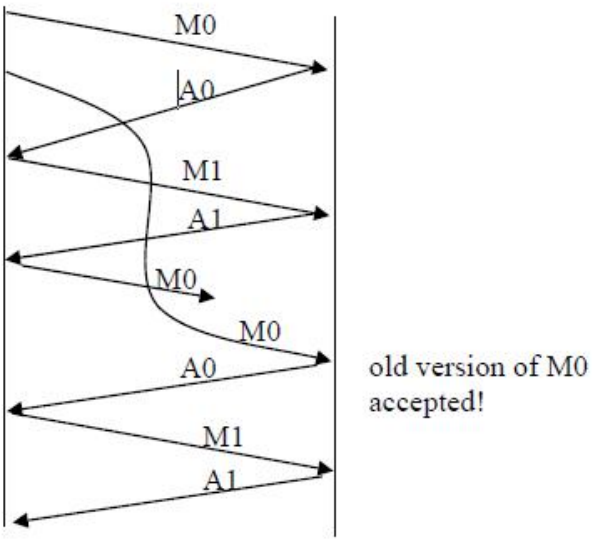
Answer: Here, we add a timer, whose value is greater than the known round-trip propagation delay. We add a timeout event to the "Wait for ACK or NAK0" and "Wait for ACK or NAK1" states. If the timeout event occurs, the most recently transmitted packet is retransmitted. Let us see why this protocol will still work with the rdt2.1 receiver.

• Suppose the timeout is caused by a lost data packet, i.e., a packet on the senderto-receiver channel. In this case, the receiver never received the previous transmission and, from the receiver's viewpoint, if the timeout retransmission is received, it look *exactly* the same as if the original transmission is being received.

• Suppose now that an ACK is lost. The receiver will eventually retransmit the packet on a timeout. But a retransmission is exactly the same action that is take if an ACK is garbled. Thus the sender's reaction is the same with a loss, as with a garbled ACK. The rdt 2.1 receiver can already handle the case of a garbled ACK.

(P11) Consider the rdt3.0 protocol. Draw a diagram showing that if the network connection between the sender and receiver can reorder messages (that is, that two messages propagating in the medium between the sender and receiver can be reordered), then the alternating-bit protocol will not work correctly (make sure you clearly identify the sense in which it will not work correctly). Your diagram should have the sender on the left and the receiver on the right, with the time axis running down the page, showing data (D) and acknowledgement (A) message exchange. Make sure you indicate the sequence number associated with any data or acknowledgement segment.

Answer:



(P12) 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 ack-num field of an acknowledgement 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 timeout can occur. Consider how many times the nth packet is sent, in the limit as n approaches infinity.)

Answer: The protocol would still work, since a retransmission would be what would happen if the packet received with errors has actually been lost (and from the receiver standpoint, it never knows which of these events, if either, will occur). To get at the more subtle issue behind this question, one has to allow for premature timeouts to occur. In this case, if each extra copy of the packet is ACKed and each received extra ACK causes another extra copy of the current packet to be sent, the number of times packet *n* is sent will increase without bound as *n* approaches infinity.

(P13) Consider a reliable data transfer protocol that uses only negative acknowledgements. 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?

Answer:

In a NAK only protocol, the loss of packet *x* is only detected by the receiver when packet *x+1* is received. That is, the receivers receives *x-1* and then *x+1,* only when *x+1 is* received does the receiver realize that *x* was missed. If there is a long delay between the transmission of x and the transmission of *x+1,* then it will be a long time until *x* can be recovered, under a NAK only protocol.

On the other hand, if data is being sent often, then recovery under a NAK-only scheme could happen quickly. Moreover, if errors are infrequent, then NAKs are only occasionally sent (when needed), and ACK are never sent – a significant reduction in feedback in the NAK-only case over the ACK-only case.

(P14) Consider the cross-country example shown in Figure 3.17. How big would the window size have to be for the channel utilization to be greater than 80 percent?

Answer: It takes 8 microseconds (or 0.008 milliseconds) to send a packet. in order for the sender
to be busy 90 percent of the time, we must have *util* = 0.9 = (0.008*n*) / 30.016 or *n* approximately 3377 packets.

(P15) Consider a scenario in which Host A wants to simultaneously send packets to Host B and C. A is connected to B and C via a broadcast channel—a packet sent by A is carried by the channel to both B and C. Suppose that the broadcast channel connecting A, B, and C can independently lose and corrupt packets (and so, for example, a packet sent from A might be correctly received by B, but not by C). Design a stop-and-wait-like error-control protocol for reliable transferring packets from A to B and C, such that A will not get new data from the upper layer until it knows that B and C have correctly received the current packet. Give FSM descriptions of A and C. (Hint: The FSM for B

should be essentially be same as for C.) Also, give a description of the packet format(s) used.

Answer:

In our solution, the sender will wait until it receives an ACK for a pair of messages (seqnum and seqnum+1) before moving on to the next pair of messages. Data packets have a data field and carry a two-bit sequence number. That is, the valid sequence numbers are 0, 1, 2, and 3. (Note: you should think about why a 1-bit sequence number space of 0, 1 only would not work in the solution below.) ACK messages carry the sequence number of the data packet they are acknowledging.

The FSM for the sender and receiver are shown in Figure 2. Note that the sender state records whether (i) no ACKs have been received for the current pair, (ii) an ACK for seqnum (only) has been received, or an ACK for seqnum+1 (only) has been received. In this figure, we assume that the seqnum is initially 0, and that the sender has sent the first two data messages (to get things going). A timeline trace for the sender and receiver recovering from a lost packet is shown below:

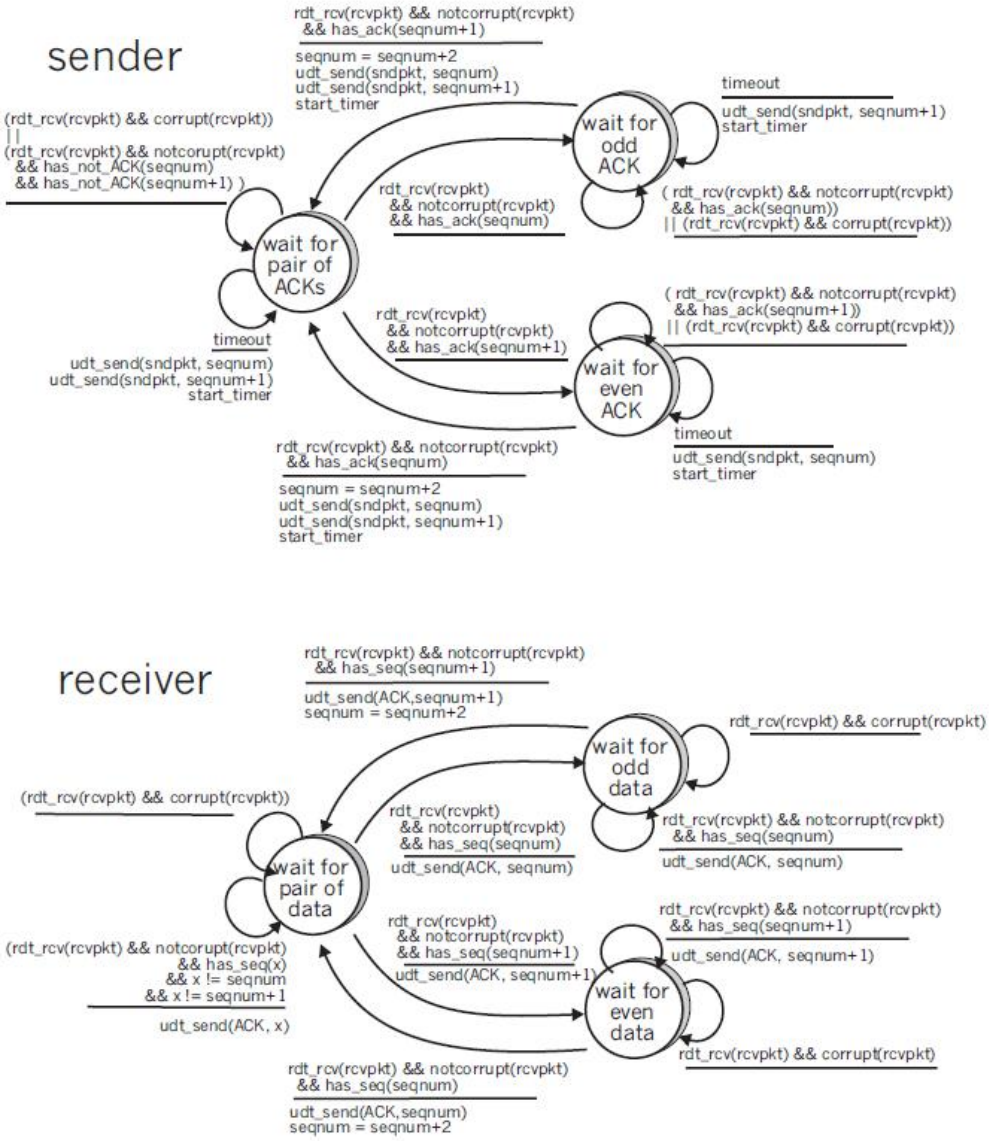


Figure 2: Sender and receiver for Problem 3.15

Sender	Receiver
make pair (0,1)	
send packet 0	
	Packet 0 drops
send packet 1	
	receive packet 1
	buffer packet 1
	send ACK 1
receive ACK 1	
(timeout)	
resend packet 0	
	receive packet 0

deliver pair (0,1)
send ACK 0

receive ACK 0

(P16) Consider a scenario in which Host A and Host B want to send messages to Host C. Hosts A and C are connected by a channel that can lose and corrupt (but not reorder)message. Hosts B and C are connected by another channel (independent of the channel connecting A and C) with the same properties. The transport layer at Host C should alternate in delivering messages from A and B to the layer above (that is, it should first deliver the data from a packet from A, then the data from a packet from B, and so on). Design a stop-and-wait-like error-control protocol for reliable transferring packets from A to B and C, with alternating delivery at C as described above. Give FSM descriptions of A and C. (Hint: The FSM for B should be essentially be same as for A.) Also, give a description of the packet format(s) used.

Answer:

This problem is a variation on the simple stop and wait protocol (rdt3.0). Because the channel may lose messages and because the sender may resend a message that one of the receivers has already received (either because of a premature timeout or because the other receiver has yet to receive the data correctly), sequence numbers are needed. As in rdt3.0, a 0-bit sequence number will suffice here.

The sender and receiver FSM are shown in Figure 3. In this problem, the sender state indicates whether the sender has received an ACK from B (only), from C (only) or from neither C nor B. The receiver state indicates which sequence number the receiver is waiting for.

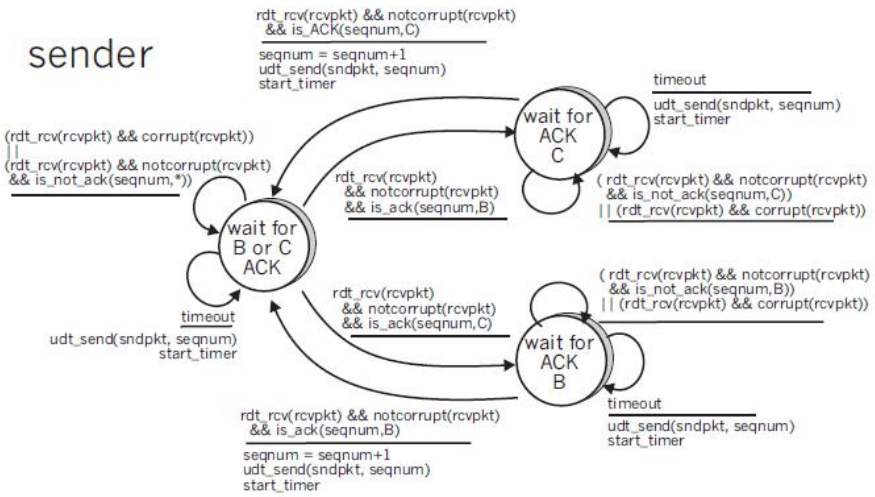


Figure 3. Sender and receiver for Problem 3.16

(P17) In the generic SR protocol that we studied in Section 3.4.4, the sender transmits a message as soon as it is available (if it is in the window) without waiting for an acknowledgment. Suppose now that we want an SR protocol that sends messages two at a time. That is , the sender will send a pair of messages and will send the next pair of messages only when it knows that both messages in the first pair have been receiver correctly.

Suppose that the channel may lose messages but will not corrupt or reorder messages. Design an error-control protocol for the unidirectional reliable transfer of messages. Give an FSM description of the sender and receiver. Describe the format of the packets sent between sender and receiver, and vice versa. If you use any procedure calls other than those in Section 3.4(for example, udt_send(), start_timer(), rdt_rcv(), and so on) ,clearly state their actions. Give an example (a timeline trace of sender and receiver) showing how your protocol recovers from a lost packet.

Answer:

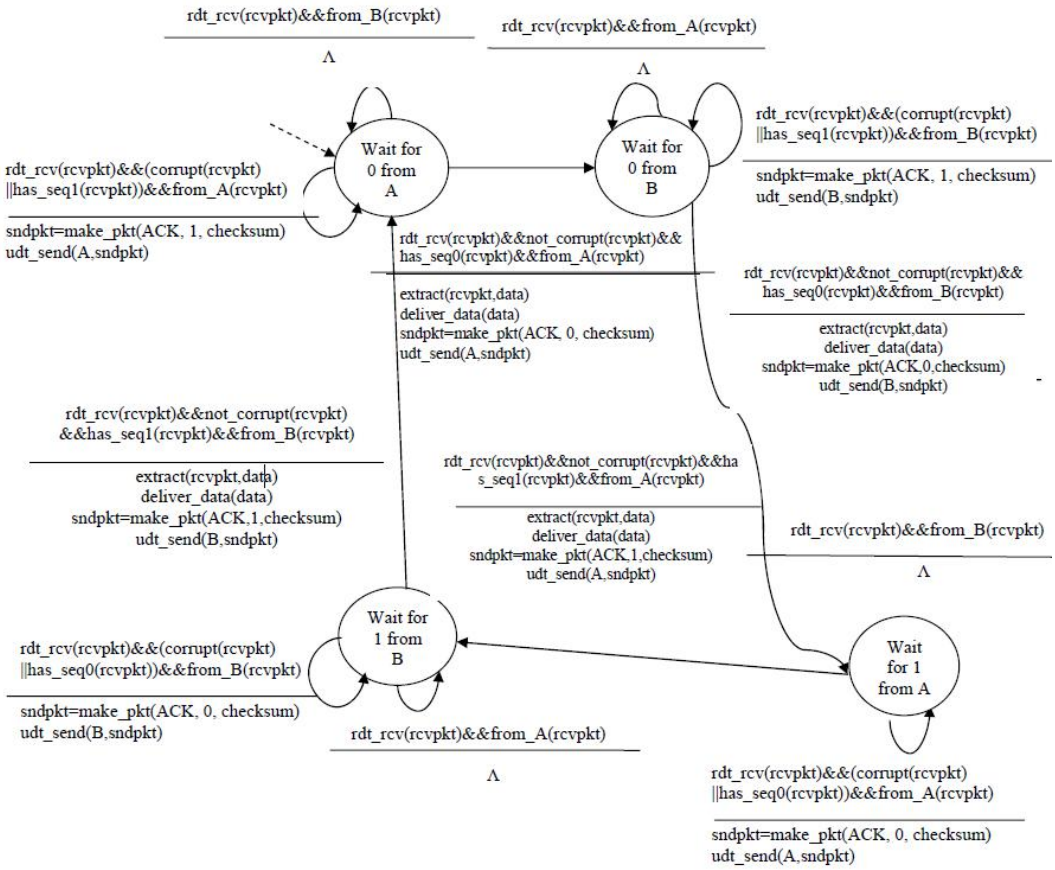


Figure 4: Receiver side FSM for 3.17

(P18) Consider the GBN protocol with a sender window size of 3 and a sequence number range of 1024. 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 number inside the sender's 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.

Answer:

a. Here we have a window size of N=3. Suppose the receiver has received packet k-1, and has ACKed that and all other preceeding packets. If all of these ACK's have been received by sender, then sender's window is [k, k+N-1]. Suppose next that none of the ACKs have been received at the sender. In this second case, the sender's window contains k-1 and the N packets up to and including k-1. The sender's window is thus [k-N,k-1]. By these arguments, the senders window is of size 3 and begins somewhere in the range [k-N,k].

b. If the receiver is waiting for packet k, then it has received (and ACKed) packet k-1 and

the N-1 packets before that. If none of those N ACKs have been yet received by the sender, then ACK messages with values of [k-N,k-1] may still be propagating back. Because the sender has sent packets [k-N, k-1], it must be the case that the sender has already received an ACK for k-N-1. Once the receiver has sent an ACK for k-N-1 it will never send an ACK that is less that k-N-1. Thus the range of in-flight ACK values can range from k-N-1 to k-1.