**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.**

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 sender to 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

**P15. 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 98 percent? Suppose that the size of a packet is 1,500 bytes, including both header fields and data.**

round-trip propagation delay between A and B (RTT) = 30 ms
Transmission rate of the link between A and B (R) = 1Gbps = 10^9bps
size of the data packet (L)= 1500 bytes (1500x8 bits)
the time required to transmit the packet over the 1 Gbps link Dtrans = L/R = 1500 * 8 bits/packet / 10^9 bits/sec = 12ms = 0.012us
Chanel Utilization = N * (L/R) / (L/R + RTT)
N = 2450.98
Window size ≈ 2451 packets

**P16. Suppose an application uses rdt 3.0 as its transport layer protocol. As the stop-and-wait protocol has very low channel utilization (shown in the crosscountry example), the designers of this application let the receiver keep sending back a number (more than two) of alternating ACK 0 and ACK 1 even if the corresponding data have not arrived at the receiver. Would this application design increase the channel utilization? Why? Are there any potential problems with this approach? Explain.**

- Suppose an application uses rdt 3.0 as its transport layer protocol. Then It stop-and-wait protocol is used very low utilization in the application.
- The application allows the receiver to send the acknowledgements and sends the next data packet. It is used as a pipelined data in the channel.
- There is a chance of missing data before it reaches the receiver.
- So the sender (who is using rtd 3.0 protocol) will not retransmit the data. The missing or lost some

data.

- Thus, designing of the application need to adopt some mechanism to overcome this problem.

**P18. 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 received 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.**

In my solution, the sender will wait until it receives an ACK for a pair of messages (seq num 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 seq num (only) has been received, or an ACK for seqnum+1 (only) has been received. In this figure, we assume that the seq num is initially 0, and that the sender has sent the first two data messages (to get things going).

**P22. Consider the GBN protocol with a sender window size of 4 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 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.**

(a)
Consider the following data:
Window size (N) = 4
Range of sequence number = 1024
Case 1:
• Assume that the sequence number of the packet being expected by the receiver is k and assume that the receiver has received and acknowledged all the k-1 packets.

• The sender's window will be in the range of [ k, k+N-1] sequence numbers when all the k-1 acknowledgements have been received without loss.
Case 2:
• If the sender does not receive any of the acknowledgements, then the sender's window will be in the range of [ k-N, k-1] sequence numbers.
• Since it did not receive any acknowledgements, it will try to send all the k-1 and N packets.
• So, the sender's window will be in the range of [ k-N, k-1] sequence numbers.
Hence, the possible sets of sequence numbers inside the sender's window at time t are in the range [ k-N, k].

(b)
• When the receiver is waiting for a packet with sequence number k and if it starts receiving the N-1 earlier packets, then the possible values of the acknowledgement(ACK) field will be [ k-N, k-1].
• It is so because when the sender has not received the N ACKs, then the ACK messages may be propagating back.
• As the sender has sent all the k-N packets and received ACK for k-N-1 from the receiver, it will never send an ACK less than the n-N-1 ACK.
Therefore, all possible values of the ACK field in all messages currently propagating back to the sender at time t are in the range of ACK values and they will be in range between k-N-1 and k-1.

**P24. Answer true or false to the following questions and briefly justify your answer:**
**a. With the SR protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.**
**b. With GBN, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.**
**c. The alternating-bit protocol is the same as the SR protocol with a sender and receiver window size of 1.**
**d. The alternating-bit protocol is the same as the GBN protocol with a sender and receiver window size of 1.**

a) True. Suppose the sender has a window size of 3 and sends packets 1, 2, 3 at t 0. At t1 (t1 > t 0) the receiver ACKS 1, 2, 3. At t 2 (t 2 > t1) the sender times out and resends 1, 2,3. At t 3 the receiver receives the duplicates and re-acknowledges 1, 2, 3. At t 4 the sender receives the ACKs that the receiver sent at t1 and advances its window to 4, 5, 6. At t 5 the sender receives the ACKs 1, 2, 3 the receiver sent at t2. These ACKs are outside its window.
b) True. By essentially the same scenario as in (a)
c) True. Note that with a window size of 1, SR, GBN, and the alternating-bit protocol are functionally equivalent. The window size of 1 precludes the possibility of out-of-order packets (within the window). A cumulative ACK is just an ordinary ACK in this situation, since it can only refer to the single packet within the window.
d) True. The explanation is the same as (c).