COMP 431 Internet Protocols & Services

Fall 2018  
Kevin Jeffay

Worksheet 8, October 15

1) Consider the rdt 2.2 receiver and the creation of a new packet in the self-transition (*i.e*., the transition from the state back to itself) in the “wait for 0 from below” and the “wait for 1 from below” states: “*sndpkt* = *make\_pkt*(*ACK,0, checksum*)” and “*sndpkt* = *make\_pkt*(*sndpkt,ACK,1, checksum*)”*.*

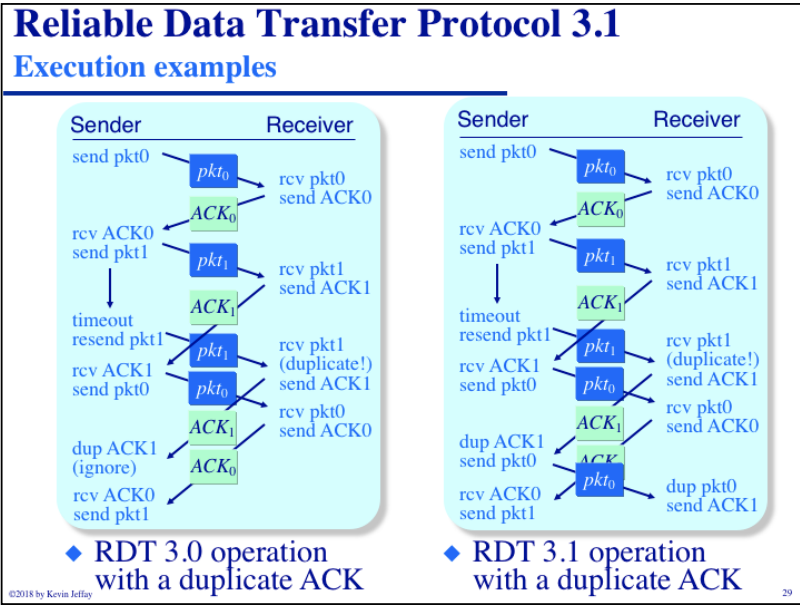
Would the protocol work correctly if this action were removed from the self-transition in the “*wait for rdt\_rcv* *seq* 1”? Justify your answer. What if this event were removed from the self-transition in the “*wait for rdt\_rcv* *seq* 0” state?

It wouldn’t work because the sender would have no idea that the recipient received the packet, as it is receiving no further acknowledgement, so it has no idea when to send another packet, meaning both sender and receiver are both waiting for another transmission. This means they are in a state of deadlock.

2) Unlike the sender state machine for rdt 2.2, the sender state machine for rdt 3.0 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 acknowledgment packet. Suppose that in the “wait for ACK X” state the sender now retransmits the previously transmitted packet upon receipt of a duplicate or corrupted ACK (as was done in rdt 2.2) and restarts the timer. Would the protocol still work? Hint: Technically the protocol still “works” but it becomes very inefficient under certain circumstances.

Under

certain circumstances, you will enter a state where you will send multiple copies of every single packet. Because the receiver is designed to deal with multiple arrivals, it will throw the duplicates away, but it’s obviously not a good thing because you’re sending multiple packets.

Because the sender is retransmitting every time, and never ignore a duplicate packet, you end up generating two copies of every packet. You add in a duplicate every time there is a premature timeout.