COMP 431 Internet Protocols & Services

Spring 2017  
Kevin Jeffay

Worksheet 9, February 16

1) 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 ACK packets do not require sequence numbers?

The receivers aren’t any different than 2.2, but the state for ACK can be inferred because the client and the server would be in the same sequence state. Sequence numbers are only needed in data packets to determine if a packet is a duplicate. On the server’s side, duplicate ACKs would occur in [wait for rdt\_send 1] and will ignore it. Having a sequence number in the ACK would be redundant.

2) Consider the rdt2.2 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?

This protocol wouldn’t work because it is dependent on sitting in states until good data is sent. In both cases, the protocol would break because the error checking would be removed, the states can only change when the error checking passes.

Rdt2.2 MUST have explicit response retransmit or continue, or it will deadlock. Sender and receiver are waiting on each other.

In rdt3.0, the sender will continually retransmit.

3) 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 acknowledgment packet. Suppose that in such circumstances, rdt3.0 were simply to retransmit the current data packet. Would the protocol still work?

~~The protocol would still work because even if the receiver initially received the packet correctly, it would still send back an ACK in the event of a duplicate.~~

If you ignore issues except for timeout, performance will improve.

Anytime a sender/receiver receives a packet, it generates a packet.

If nothing is ever lost, a premature timeout means there will be another packet. One premature timeout, there will be two copies of every single packet.

In this case, each packet will be sent multiples of time.

In terms of protocol, it sort of works since it is reliable, but has horrible mechanism

4) Consider the rdt 3.0 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 acknowledgment (A) message exchange. Make sure you indicate the sequence number associated with any data or acknowledgment segment.

Out of order pathology. These reliability protocols are too simple for out of order pathologies.

