CSC358 Tutorial 4

Julian Sequeira and KyoKeun Park

February 7, 2020

University of Toronto Mississauga

Question 1: Concept Review

(a) What are the different roles of the transport layers and the network layer?

(b) What are the differences between TCP and UDP?

(c) What do we mean by "UDP is connectionless"?

(d) In *`rdt3.0`*, what are purposes of ACKs, timeouts, and sequence numbers?

For the answers, review the lectures, books, go to office hours, and use the discussion board!

Question 2: Receiver FSM of *rdt3.0*

In the lecture slides, we showed and discussed the sender's FSM of ***rdt3.0***, but we omitted the receiver FSM. In this question, you will complete the FSM for the receiver side of protocol ***rdt3.0***. To get started, think about what modifications need to be made to the receiver's FSM in ***rdt2.2***.

Question 2: Receiver FSM of *`rdt3.0`*

Recall: timeout functionality was added to ***`rdt3.0`***

So when the sender sends a packet with sequence number X, it can timeout and send the same packet with that same sequence number X

So the receiver can potentially get **duplicate packets**, the FSM needs to handle that.

Question 2: Receiver FSM of *rdt3.0*

Recall: timeout functionality was added to ***rdt3.0***

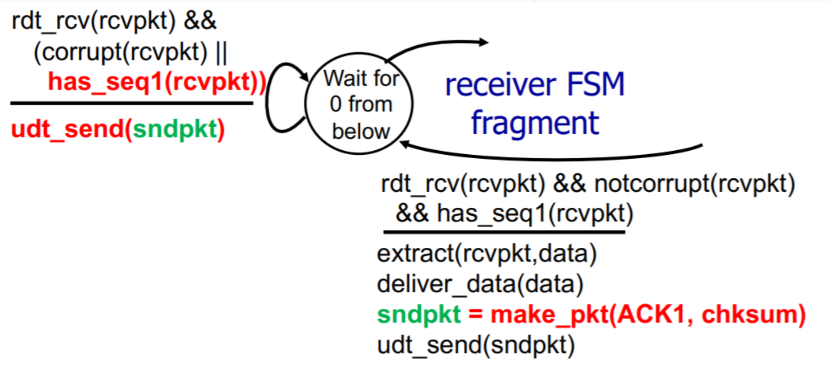
So when the sender sends a packet with sequence number X, it can timeout and send the same packet with that same sequence number X

So the receiver can potentially get **duplicate packets**, the FSM needs to handle that.

The receiver's FSM turns out to be the same one used in ***rdt2.2***, as it also has the possibility of duplicate packets.

If you're waiting for a packet with seqnum 1, but you get a seqnum of 0, re-send the ACK for 0 (your previous ACK may have been lost, so the sender timed out and sent you 0 again)

Question 2: Receiver FSM of `rdt3.0`



rdt_rcv(rcvpkt) &&
  (corrupt(rcvpkt) ||
    has_seq1(rcvpkt))

udt_send(sndpkt)

Wait for
0 from
below

receiver FSM
fragment

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
  && has_seq1(rcvpkt)

extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK1, chksum)
udt_send(sndpkt)

Question 3: Design a *rdt* protocol for 1-to-2 transmission

Consider a scenario in which Host A wants to simultaneously send packets to Hosts 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 C). Design a stop-and-wait-like error-control protocol for reliably transferring packets from A to B and C, such that A will not get new data from the upper layer until it knows that both B and C have correctly received the current packets. Give FSM descriptions of A, B and C (Hint: the FSMs for B and C should be essentially the same). In particular, think about the following questions:

(a) What are the states of the sender's FSM?

(b) What are the state of the receiver's FSM?

(c) Is it necessary to have sequence numbers?

(d) Is it necessary to have ACK or NAK, or both?

(e) Is it necessary to have timeout?

(f) Could this protocol be similar to one of the `rdt` protocols that we learned in class?

Question 3: Design a *rdt* protocol for 1-to-2 transmission

HINT: Try to modify the ***rdt3.0*** FSM

Question 3: Design a *rdt* protocol for 1-to-2 transmission

HINT: Try to modify the `rdt3.0` FSM

Do we need sequence numbers?

Question 3: Design a *rdt* protocol for 1-to-2 transmission

HINT: Try to modify the *rdt3.0* FSM

Do we need sequence numbers?

Yes. We can't avoid possibly sending duplicates to B or C, so they need to be able to recognize a duplicate via sequence numbers

Question 3: Design a *rdt* protocol for 1-to-2 transmission

HINT: Try to modify the *rdt3.0* FSM

Do we need sequence numbers?

Yes. We can't avoid possibly sending duplicates to B or C, so they need to be able to recognize a duplicate via sequence numbers

How many sequence numbers bits do we need?

Question 3: Design a *rdt* protocol for 1-to-2 transmission

HINT: Try to modify the *rdt3.0* FSM

Do we need sequence numbers?

Yes. We can't avoid possibly sending duplicates to B or C, so they need to be able to recognize a duplicate via sequence numbers

How many sequence numbers bits do we need?

We're designing a stop-and-wait protocol, so A waits for packet X to be successfully received by both B and C before sending Packet X+1. So a 1 bit sequence number (for 2 values) is enough.

Question 3: Design a *rdt* protocol for 1-to-2 transmission

HINT: Try to modify the *rdt3.0* FSM

Do we need sequence numbers?

Yes. We can't avoid possibly sending duplicates to B or C, so they need to be able to recognize a duplicate via sequence numbers

How many sequence numbers bits do we need?

We're designing a stop-and-wait protocol, so A waits for packet X to be successfully received by both B and C before sending Packet X+1. So a 1 bit sequence number (for 2 values) is enough.

Is it necessary to have ACK or NAK, or both?

Question 3: Design a *rdt* protocol for 1-to-2 transmission

HINT: Try to modify the *`rdt3.0`* FSM

Do we need sequence numbers?

Yes. We can't avoid possibly sending duplicates to B or C, so they need to be able to recognize a duplicate via sequence numbers

How many sequence numbers bits do we need?

We're designing a stop-and-wait protocol, so A waits for packet X to be successfully received by both B and C before sending Packet X+1. So a 1 bit sequence number (for 2 values) is enough.

Is it necessary to have ACK or NAK, or both?

We need ACKS so the receivers can confirm they've gotten non-corrupt packets. NAKs are not necessary in this case if we use timeouts

Question 3: Design a rdt protocol for 1-to-2 transmission

Is it necessary to have timeouts?

Question 3: Design a rdt protocol for 1-to-2 transmission

Is it necessary to have timeouts?

Timeouts are needed since packets can be arbitrarily lost between sender and receiver. A NAK is not enough- the NAK may get lost between the receiver and sender. Or the receiver simply doesn't receive the packet in the first place.

Question 3: Design a *rdt* protocol for 1-to-2 transmission

Is it necessary to have timeouts?

Timeouts are needed since packets can be arbitrarily lost between sender and receiver. A NAK is not enough- the NAK may get lost between the receiver and sender. Or the receiver simply doesn't receive the packet in the first place.

Receiver FSM:

Similar to q2, but include some information about which host you are (B or C) in the response packets to A, so A knows which host received the packet

Question 3: Design a *rdt* protocol for 1-to-2 transmission

Is it necessary to have timeouts?

Timeouts are needed since packets can be arbitrarily lost between sender and receiver. A NAK is not enough- the NAK may get lost between the receiver and sender. Or the receiver simply doesn't receive the packet in the first place.

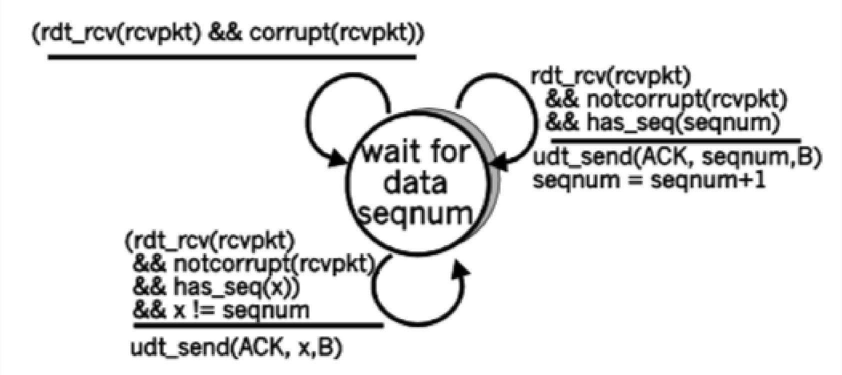
Receiver FSM:

Similar to q2, but include some information about which host you are (B or C) in the response packets to A, so A knows which host received the packet

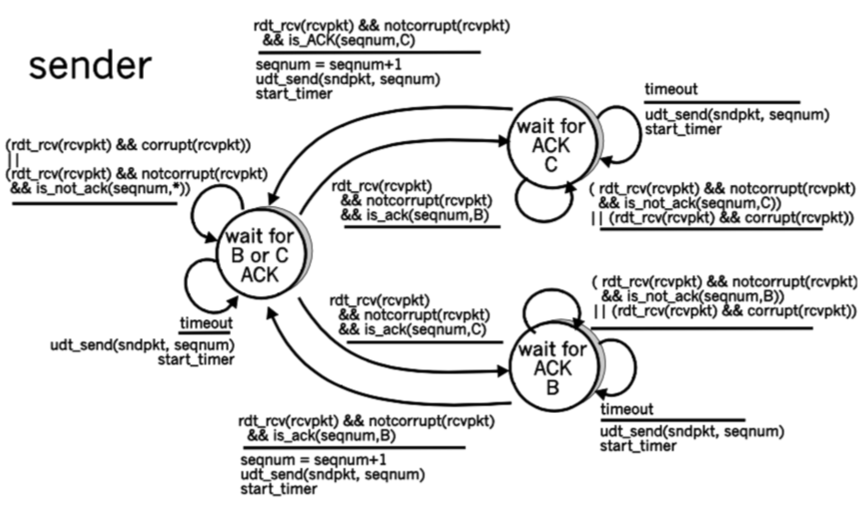
Sender FSM:

- Can only increment the sequence number and send the next packet once both B and C send back ACKS
- One state to wait for either B or C
- One state to wait for C after having received B's ACK
- One state to wait for B after having received C's ACK

Question 3: Receiver FSM



Question 3: Sender FSM



sender
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && is_ACK(seqnum,C)
seqnum = seqnum+1
udt_send(sndpkt, seqnum)
start_timer
wait for ACK C
timeout
udt_send(sndpkt, seqnum)
start_timer
(rdt_rcv(rcvpkt) && corrupt(rcvpkt))
||
(rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && is_not_ack(seqnum,*))
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && is_ack(seqnum,B)
(rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && is_not_ack(seqnum,C))
|| (rdt_rcv(rcvpkt) && corrupt(rcvpkt))
wait for B or C ACK
timeout
udt_send(sndpkt, seqnum)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && is_ack(seqnum,C)
(rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && is_not_ack(seqnum,B))
|| (rdt_rcv(rcvpkt) && corrupt(rcvpkt))
wait for ACK B
timeout
udt_send(sndpkt, seqnum)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && is_ack(seqnum,B)
seqnum = seqnum+1
udt_send(sndpkt, seqnum)
start_timer