P23

A

Server with a transfer rate to client of max (d_{min}) u_s/N

Transferring files in parallel to maximize the amount we can transfer at once

Time for client to receive files is F/transfer rate

So time to get file is $F/(u_s/N) = FN/u_s$

B

Server with a transfer rate to client of min (d_{min}) u_s/N

Have to use max since $d_{min} <= u_s/N$, so I will denote the max as d_{min}

Transferring files in parallel to maximize the amount we can transfer at once

Time for client to receive files is F/transfer rate

With transfer rate being d_{min}, the distribution time would be F/ d_{min}

So the time to get the file would be F/ d_{min}

C

The minimum distribution time in general is given by

$$\max\{\frac{NF}{u_s}, \frac{F}{d_{min}}\}$$

This is shown by combining A and B since d_{min} can be below (A) or above (B) $d_{min.}$ To get the full range of d_{min} we need to add those two together the first half is the max equation is A and the second is B. This will give us the max no matter what d_{min} is.

P24

A

Since the distribution time is a F/ u_s and $u_s <= (u_s+ u_1+...+ u_N)/N$. The first equation is bits/rate so with the rate being u_s we can assume that the u_1 through u_N is equal to u_s. This means that the peer to peer file transfer is happening for each packet at rate u_s and with F number of bits in each packet, that rate to transfer each packet is F/ u_s . So it is p2p at a rate of u_s.

This means the server must upload F bits to N peers, to end in a total of NF bits. This cannot happen faster than total of the upload rates that I will denote as u_{total} so the it would be NF/u_{total} . This is

equivalent to $NF/(Nu_s/N) = NF/u_s$ Since us is less than $(u_s+ u_1+…+ u_N)/N$ I will use the smallest N of 1 wich results in F/u_s

B

Since the distribution time is $N F/ (u_s+ u_1+…+ u_N)$ and $u_s >= (u_s+ u_1+…+ u_N)/N$.

The Amount of time it takes for a file to download is bits/rate so $F/((u_s+ u_1+…+ u_N)/N)$ which is $N F/ (u_s+ u_1+…+ u_N)$. This would mean that

This means the server must upload F bits to N peers, to end in a total of NF bits. This cannot happen faster than total of the upload rates that I will denote as u_{total} so the it would be NF/u_{total}. This is equivalent to $N F/ (u_s+ u_1+…+ u_N)$.

C

The minimum distribution time is given by

$$\max\{\frac{F}{u_s}, \frac{NF}{(u_s + u_1 + \cdots + u_N)}\}$$

because the first part F/u_s is the given rate for the first file and the other part is for the rest of the files in the peer to peer system. Taking the max of this would give us the minimum distribution time in general because its either the first file that has one of the following files. The equations change each time a new download happens with F/u_s being the first and as the number N grows, the second equation grows, becoming smaller and the new minimum distribution time.

P3

1s compliment of the 3 8 bit bytes:

10101100, 10011001, 10001011

83, 102, 116 = 301 = 00101101 with overflow

1s compliment = 11010010

Adding all 4 together at receiver = 11111111 with overflow

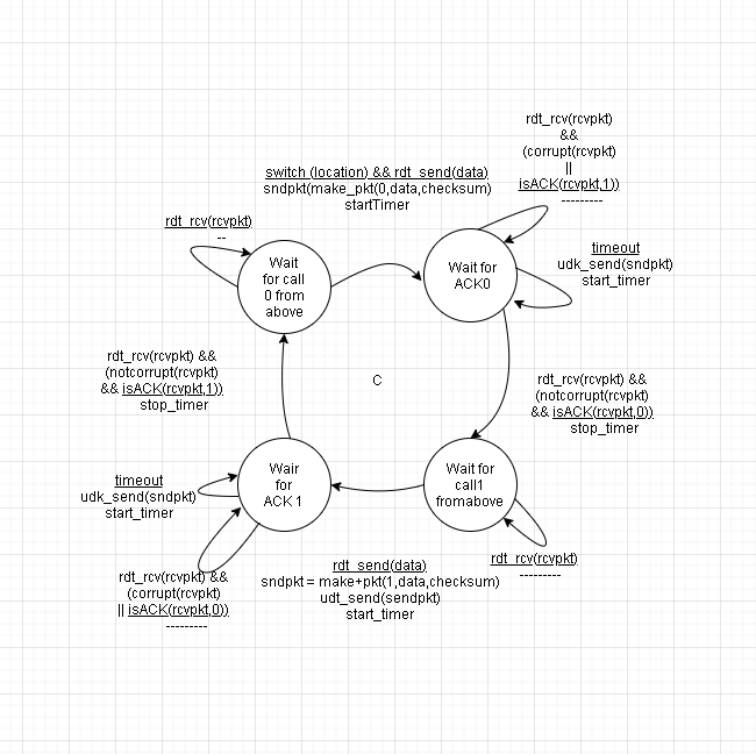
If there were any errors, it would not be 11111111 and the receiver would detect it. We don't use the sum because then it wouldn't always be 11111111. It would detect both 1 and 2 bit errors

P16

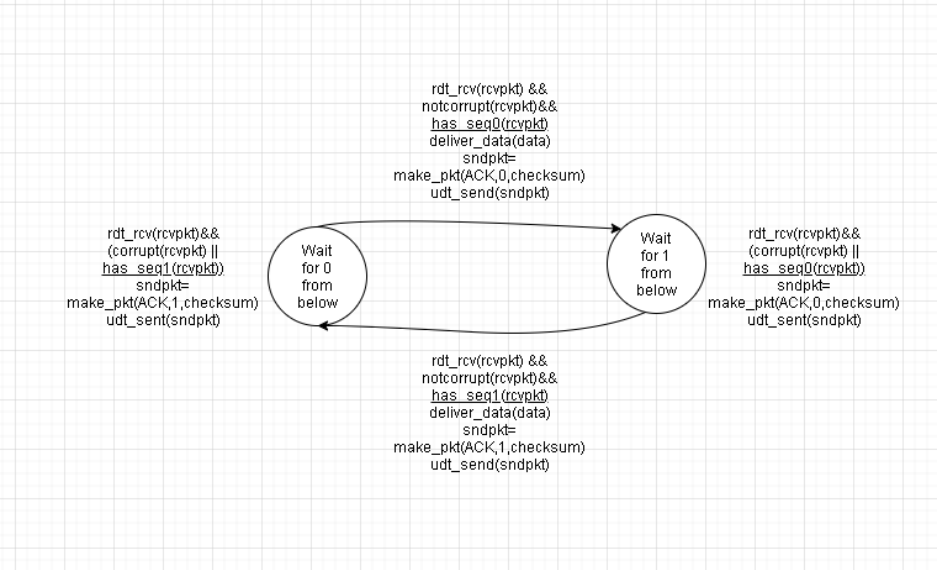
No because the sender will not move on to the next packet because it will go off of the last ACK that it received and it will keep sending the same packet each time. IT wouldn't help efficiency either since by the time the ACK packet gets the the sender, the sender will go off of the last received ACK and send the packet again at the same speed therefore not increasing channel utilization.

P20

This describes a RDT 3.0 System quite well so I modeled a rdt 3.0 sender FSM like in the books and slides for C and added the function switch(location) which takes the location A or B and swaps it and sends the packet to the new location to be sent to.

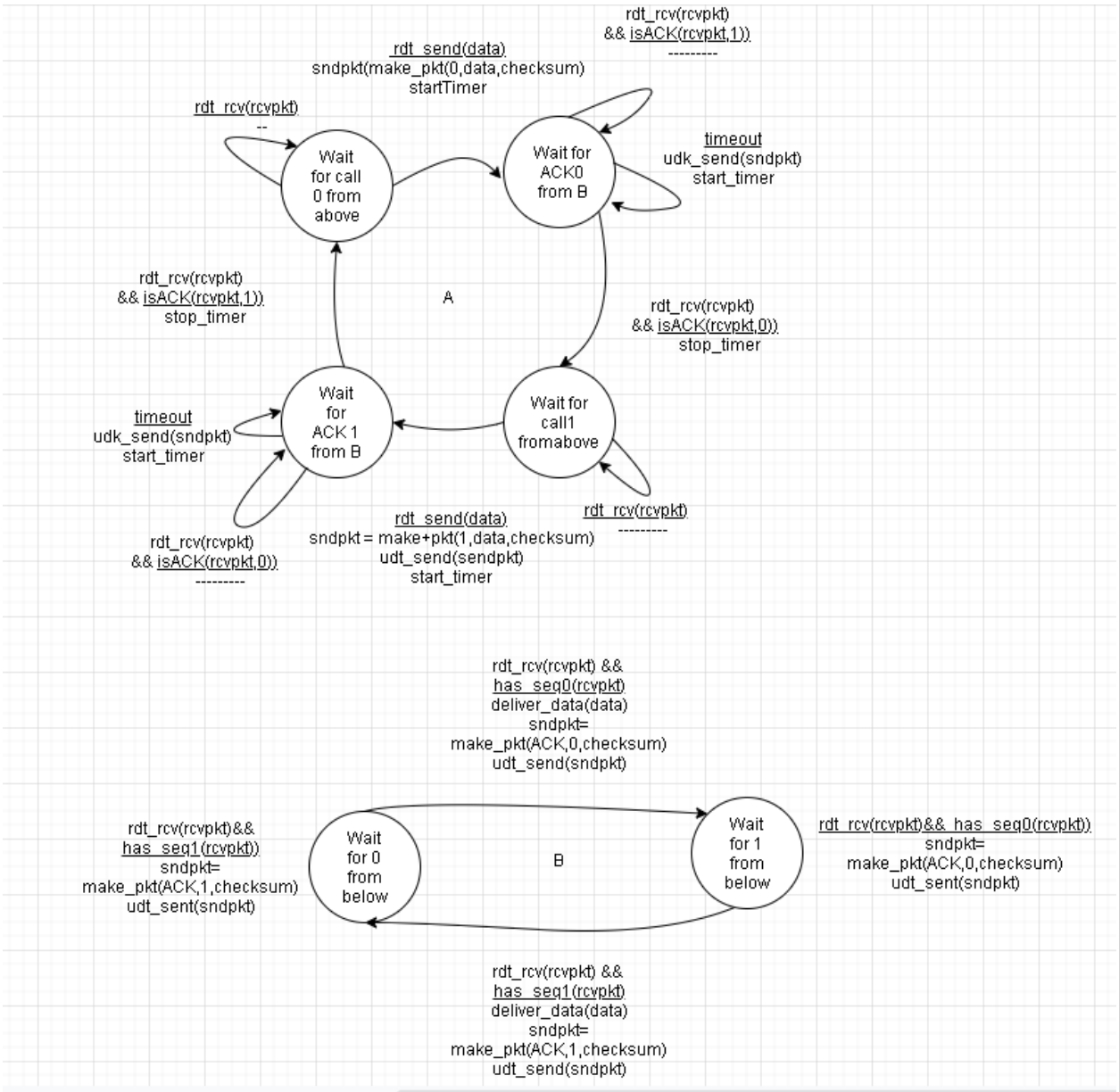


A or B essentially the same and the problem just asked for A



P21

I am going to use a rdt 3.0 FSM modified by taking out the corruption checking since the files will not be corrupted only lost. The description mentioned that the delay is unknown and variable so I figured the timer would work even if there is a delay bigger than the timer and sent the packet again, then it would ignore the duplicate packet but the packet would still get delivered. This could possibly be solved with rdt 2.0 since we wouldn't have bit errors errors but I already had a rdt 3.0 FSM so I just modified it.


rdt_rcv(rcvpkt)
&& isACK(rcvpkt,1))
---
rdt_send(data)
sndpkt(make_pkt(0,data,checksum)
startTimer
rdt_rcv(rcvpkt)
--
Wait
for call
0 from
above
Wait for
ACK0
from B
timeout
udk_send(sndpkt)
start_timer
rdt_rcv(rcvpkt)
&& isACK(rcvpkt,1))
stop_timer
A
rdt_rcv(rcvpkt)
&& isACK(rcvpkt,0))
stop_timer
Wait
for
ACK 1
from B
Wait for
call1
fromabove
timeout
udk_send(sndpkt)
start_timer
rdt_rcv(rcvpkt)
---
rdt_rcv(rcvpkt)
&& isACK(rcvpkt,0))
---
rdt_send(data)
sndpkt = make+pkt(1,data,checksum)
udt_send(sendpkt)
start_timer
rdt_rcv(rcvpkt) &&
has_seq0(rcvpkt)
deliver_data(data)
sndpkt=
make_pkt(ACK,0,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt)&&
has_seq1(rcvpkt))
sndpkt=
make_pkt(ACK,1,checksum)
udt_sent(sndpkt)
Wait
for 0
from
below
B
Wait
for 1
from
below
rdt_rcv(rcvpkt)&& has_seq0(rcvpkt))
sndpkt=
make_pkt(ACK,0,checksum)
udt_sent(sndpkt)
rdt_rcv(rcvpkt) &&
has_seq1(rcvpkt)
deliver_data(data)
sndpkt=
make_pkt(ACK,1,checksum)
udt_send(sndpkt)