CPE 464 - Selective-Reject Program Design
Due Monday, May 17, 2021 at 11:59 pm

Angela Kerlin

Design assignment: You may work on this in groups (up to 3 people). You only need to turn in one assignment for the group but you must put each group members name and a picture of the group (a picture of a zoom screen works) on the **first** page. All group members must be present for all work (except the final write-up). If you work in a group, you must provide a picture of your group on the first page with some flow diagrams or state diagrams in the picture!

Once the design work (e.g packet flows, state diagrams and program tree/common code) is done you MUST stop working as a group. Your program code needs to be your own.

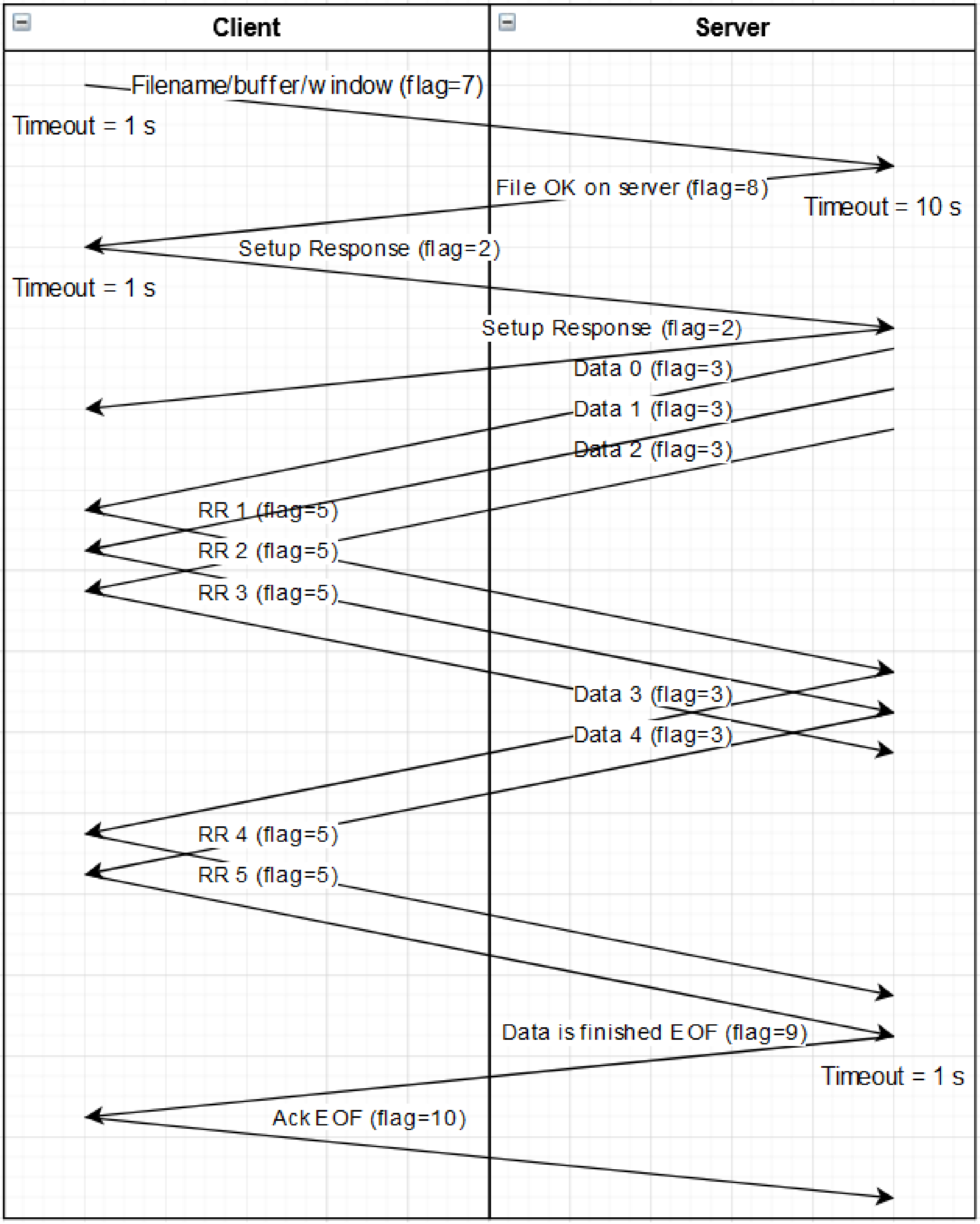
NOTE – handwritten/hand-drawn is fine. This assignment is a tool to help you successfully write your rcopy and server programs.

PART I – Design questions

1) STOP – Read the entire program #3 spec before continuing. I recommend you underline the important parts as your read it.

2) Give a packet flow diagram of the following scenarios (use a diagram similar to the ones done in class – labeled arrows between rcopy and server.) and discuss what implications this has on your rcopy and server programs (during the transmission of the file **data**):

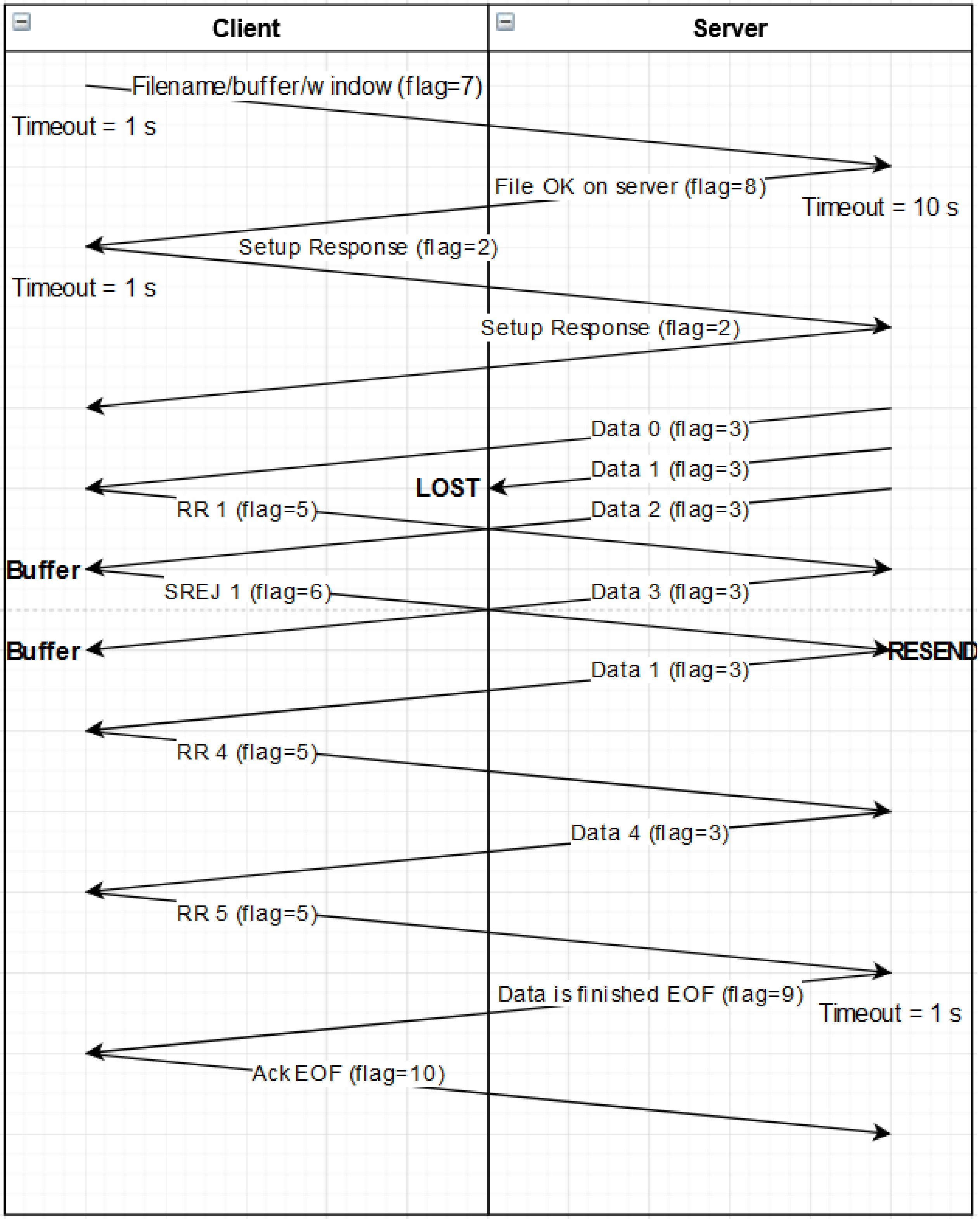
a. Normal flow – nothing lost (no errors in data or flow control packets)



Note: Window size = 3

(2)

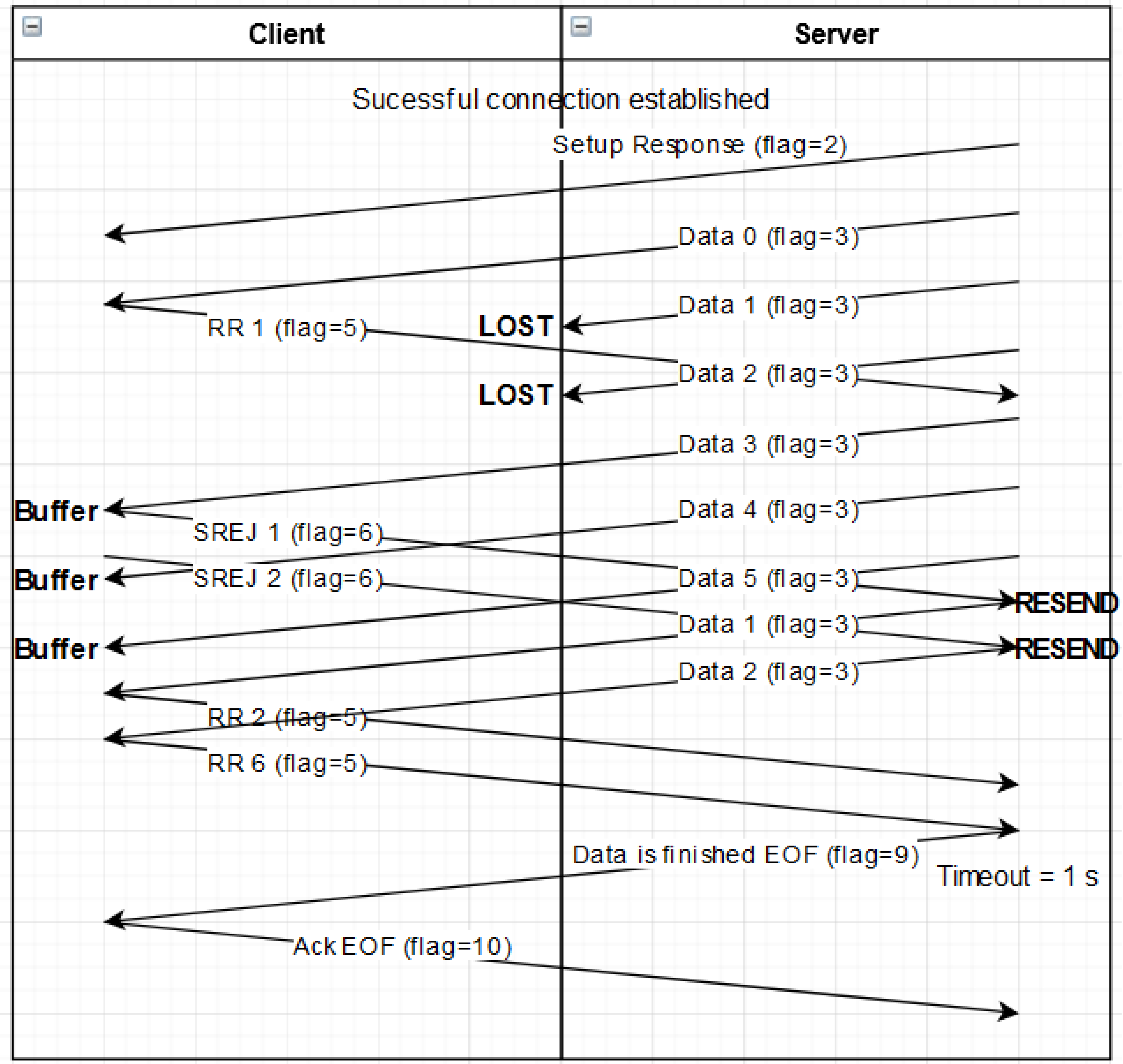
b. One data packet lost in a window of size 3 (include how its recovered)



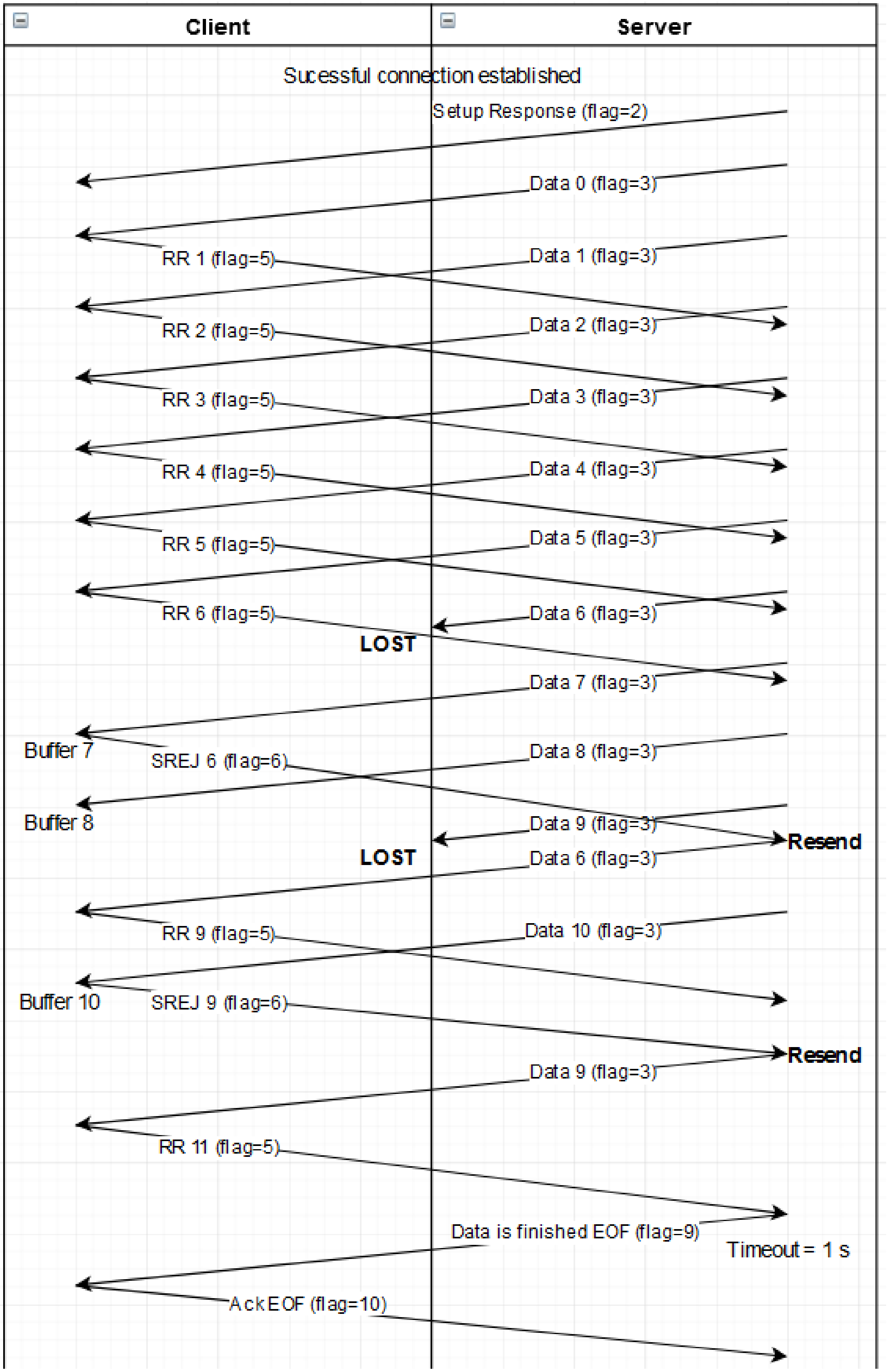
When data packet 1 is lost, the receiver will send a selective reject on 1 once it realizes its missing a packet (when packet 2 arrives). When the sender receives SREJ 1 it will resend packet 1.

(3)

c. Multiple packets lost in a window (look at both sequential packets being lost and non-sequential e.g packets 6 and 9 being lost with window size of 10)

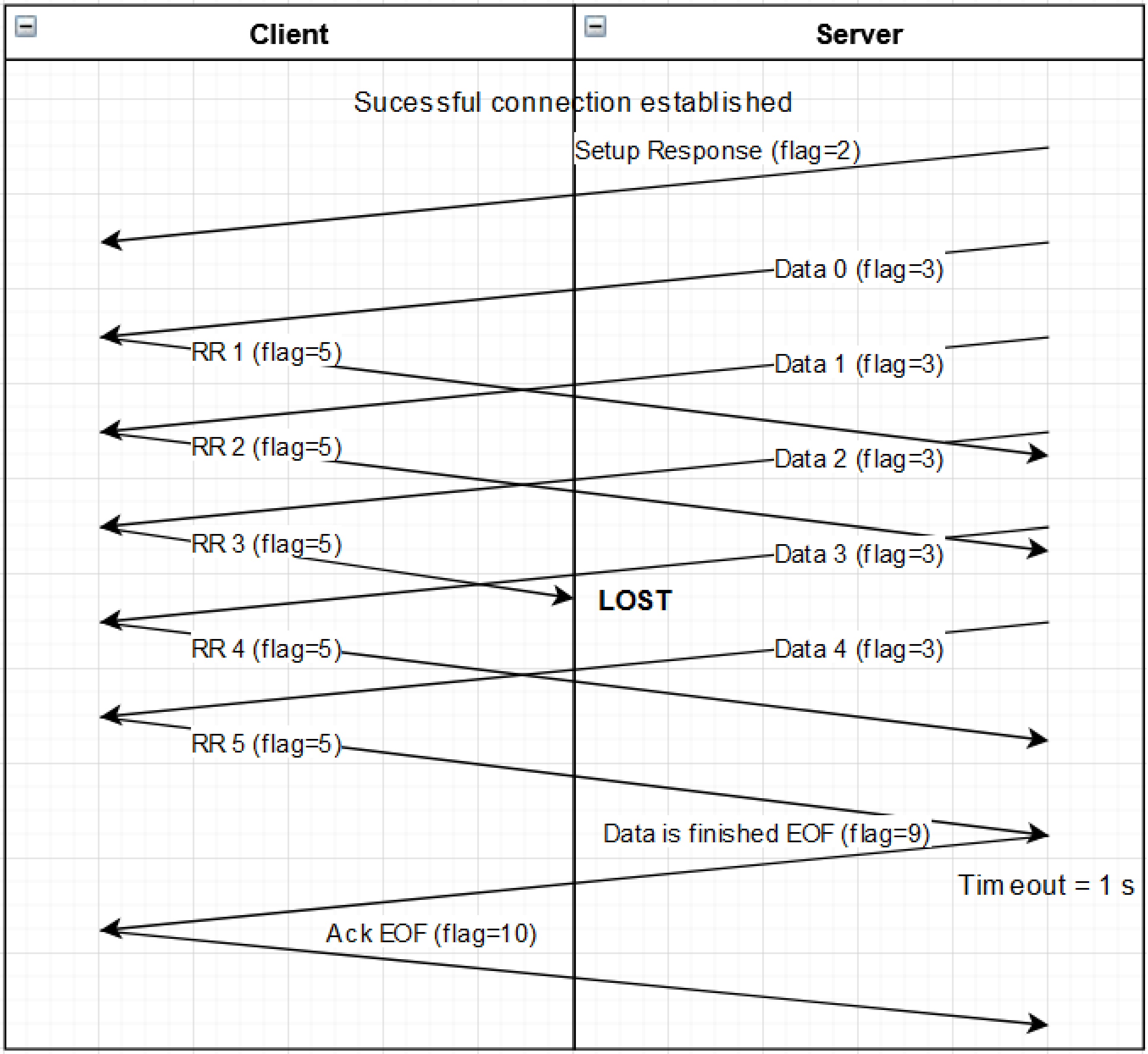


Packets 1 and 2 are consecutively lost. When packet 3 is received, the reciever must send SREJ for packets 1 and 2 and buffer until they are recieved. Sender will resend these packets when SREJ is received.



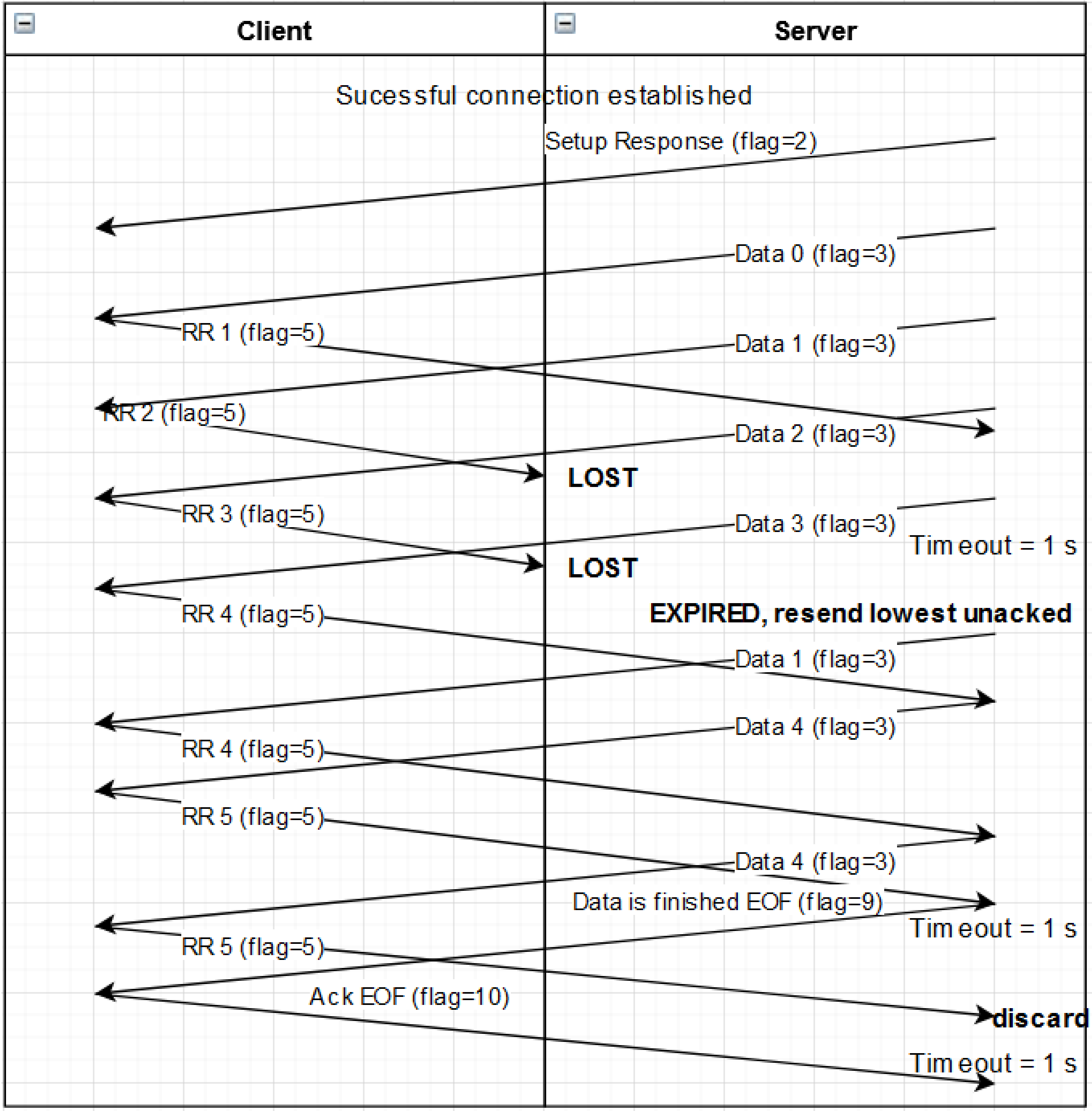
Packets 6 and 9 are non consectutively lost. Upon receiving packet 7 out of order, reciever will SREJ 6. Until it recieves packet 6, it buffers packets 7 and 8. Upon receiving packet 10, reciever will SREJ 9. Until it recieves the retranmit of packet 9, it buffers packet 10.

d. RR lost for packet 2



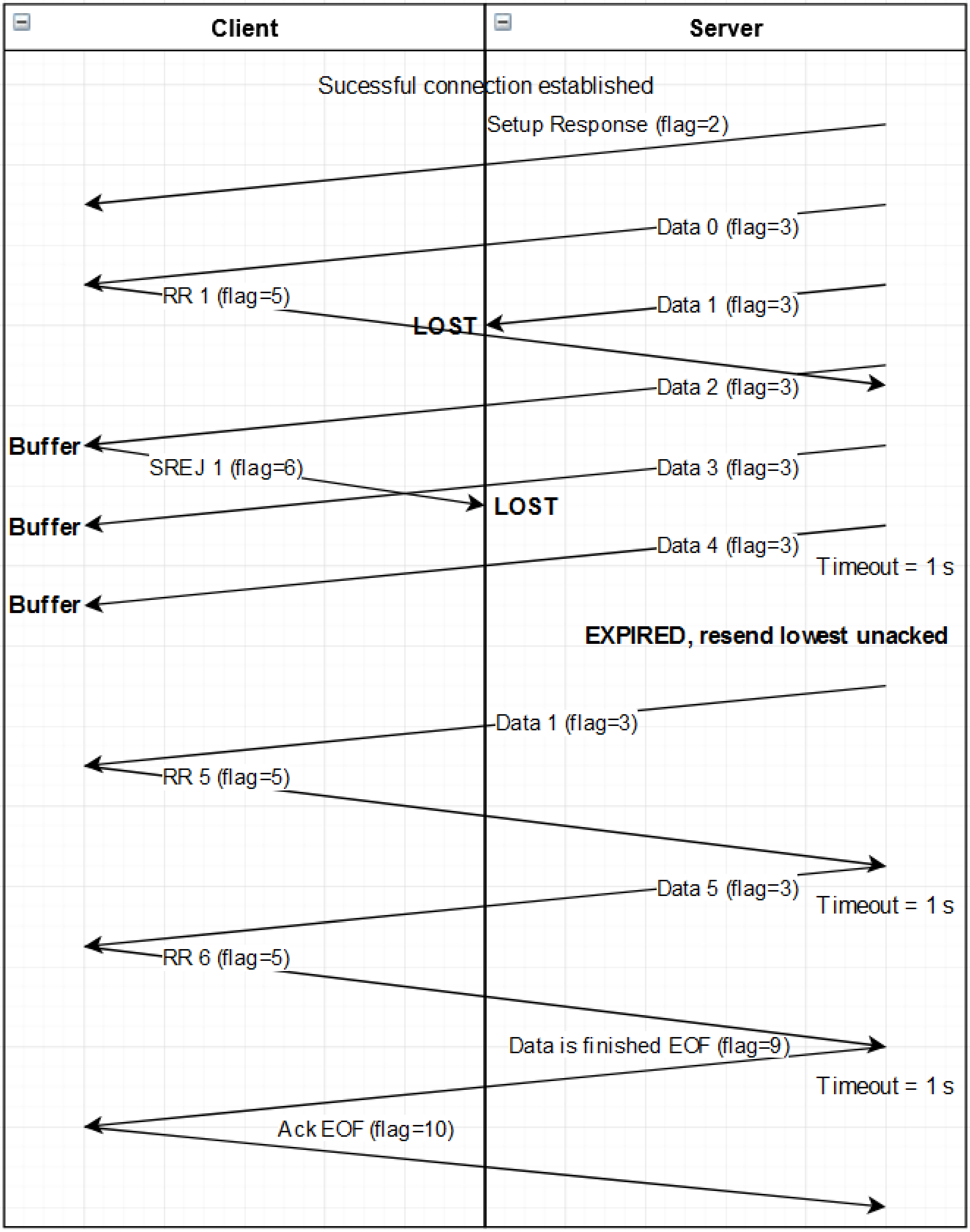
Window size = 3. The RR after the second packet is lost, but the RR's are coming in so fast that it has no real impact on the program. The sender gets RR 4 before the window runs out, so it assumes data 2 was received as well as 3.

e. Multiple RR’s lost (e.g. RR 2 and RR 3 lost)



Window size = 3. When the two RR’s are lost, the window appears filled on the sender size, so the sender calls select(1) to wait for an RR. I assumed that this took more than a second, so the timer expired and it resends the lowest unacked packet. (If it received RR 4 earlier, the case would look much like problem d.) Because this happened so close to the end of the tranmission, the EOF timer recieves an RR instead of an Ack EOF, which is discarded.

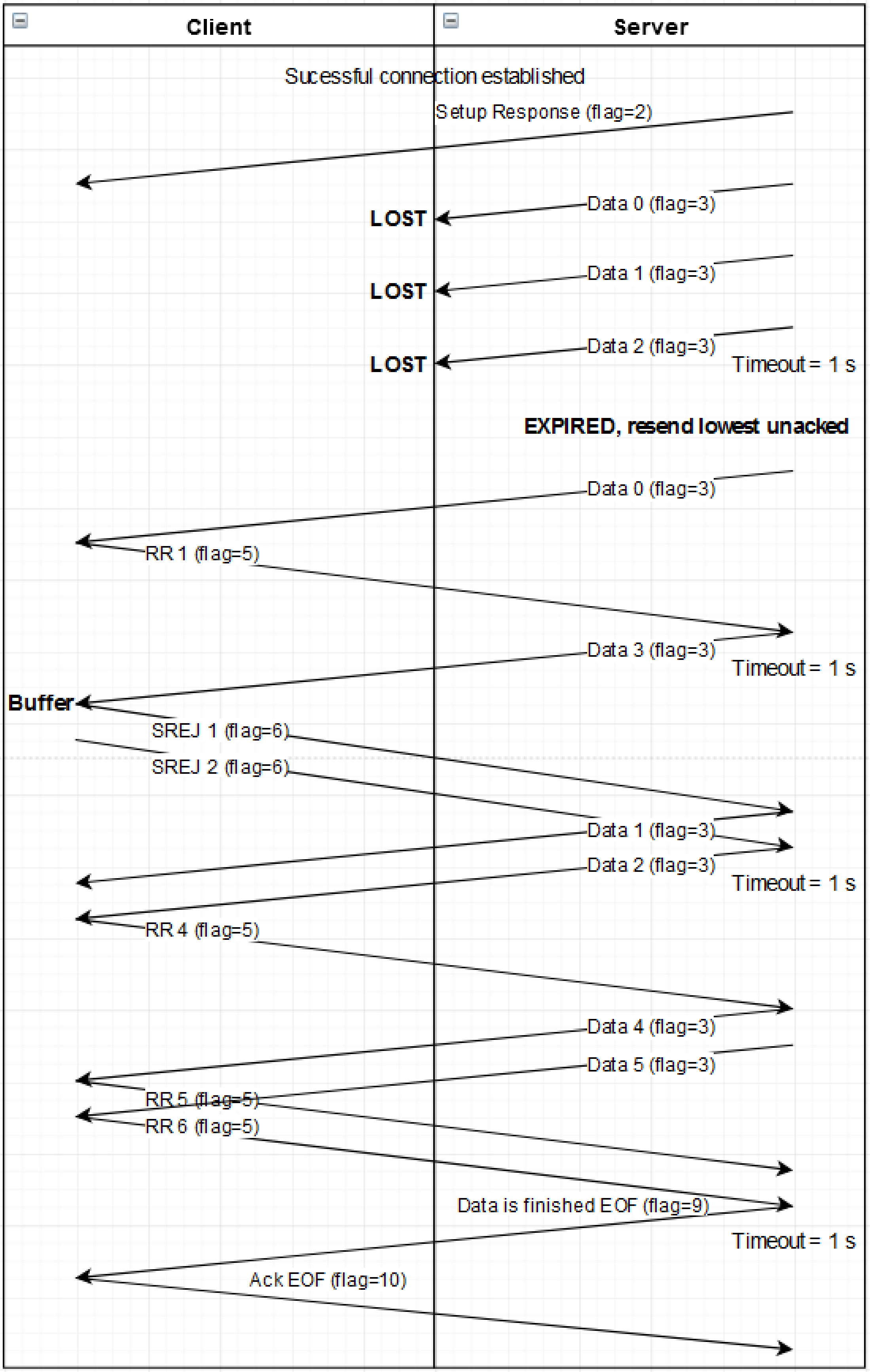
f. SREJ lost



Window size = 4. When SREJ 1 is lost, rcopy will go silent and the sender will just continue to send a whole window's worth of data. When it cannot send anymore, it will set a timer for 1 sec. When that expires, it resends the lowest unacked data, which is the packet rcopy is waiting on.

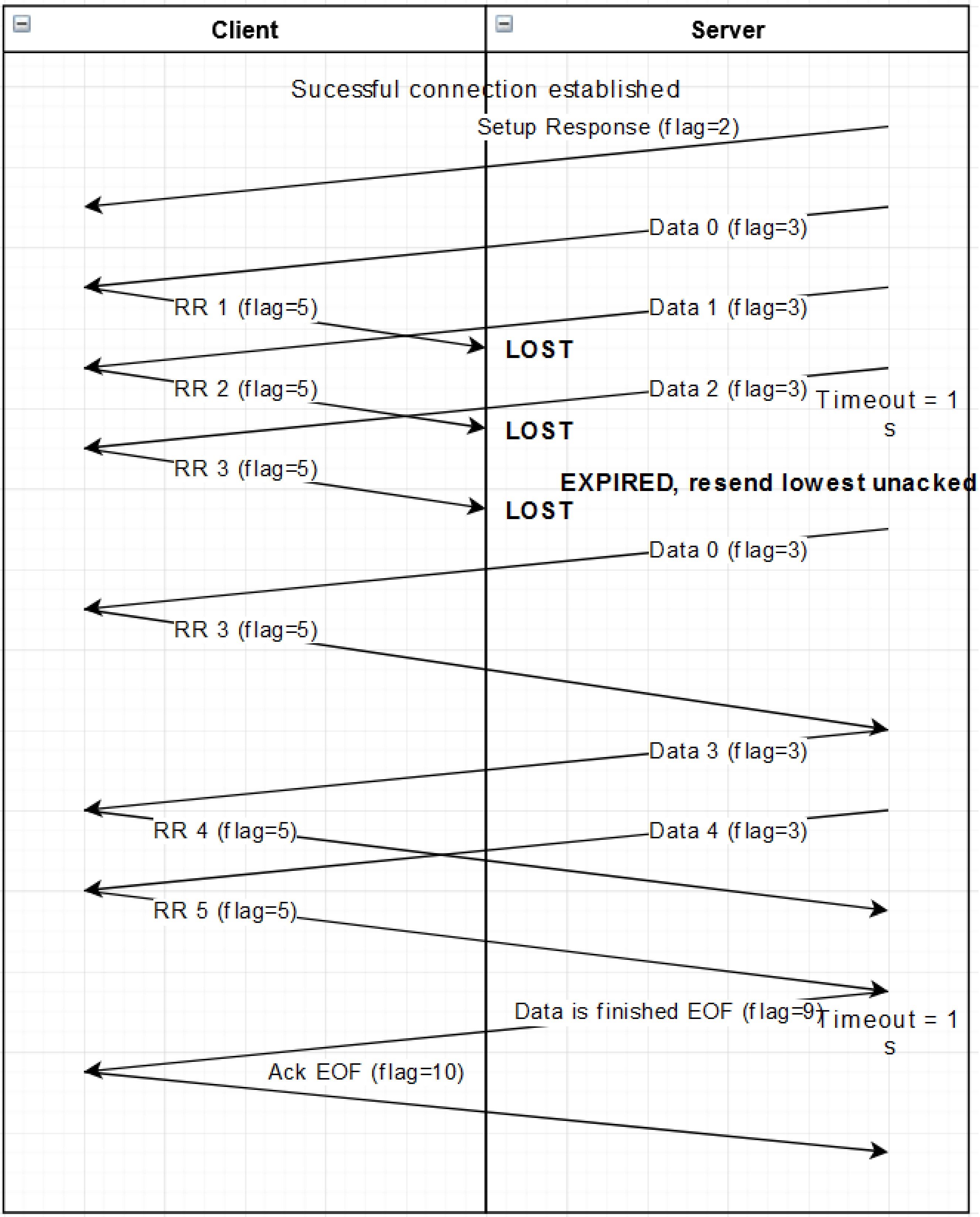
(8)

g. Entire window of data being lost



Window size = 3. When the whole window is lost, the programs get fairly confused. Server will resend the first data packet after timing out, and the client will ack it freeing up 1 slot in the window. The server then sends packet 3 which triggers 2 SREJ for the client, then it all gets caught up.

h. Entire window of RR's being lost



Window size = 3. When a whole window of RR's are lost, the server just needs to timeout and resend the lowest data packet to receive one RR for the entire window.

3) Looking back at question 1, which of those scenarios may cause your window to close? (Explain each case also explain what will need to be retransmitted… I know the window will close if the window size is to small… so don't just say that. Which of the above situations will cause the window to close because of the loss packet(s)[1]?

The window may close if:

1. An entire window of RR's is lost. The server just needs to timeout once then resend the lowest unacked packet to trigger a resend of one RR that will clear the whole window.
2. An entire window of data is lost. The server will need to timeout to resend the first packet, send the last packet in the new window, then the client will send SREJ for every other missed packet.
3. A SREJ is lost. The server will just continue sending packets until the window closes, then it will timeout and resend the lowest packet (which will be the missing packet). If multiple SREJ are lost in a row, the process will repeat until the client receives all lost packets.

4) Regarding the following possible scenarios for receiving data. Explain **how** this scenario can happen and list **what action** you will take with the data and how you will reply to the sender (it is possible that one or more of these scenarios cannot happen):

a. Data frame number is a duplicate of one you have already received

   This can happen if your RR was lost. Discard the packet and send the RR again.

b. Data frame number is greater than the one you expected but is still in your window size.

   A data packet was lost. Send a SREJ for each sequence number lower than the one received but higher than the last one received. Buffer the data frame received.

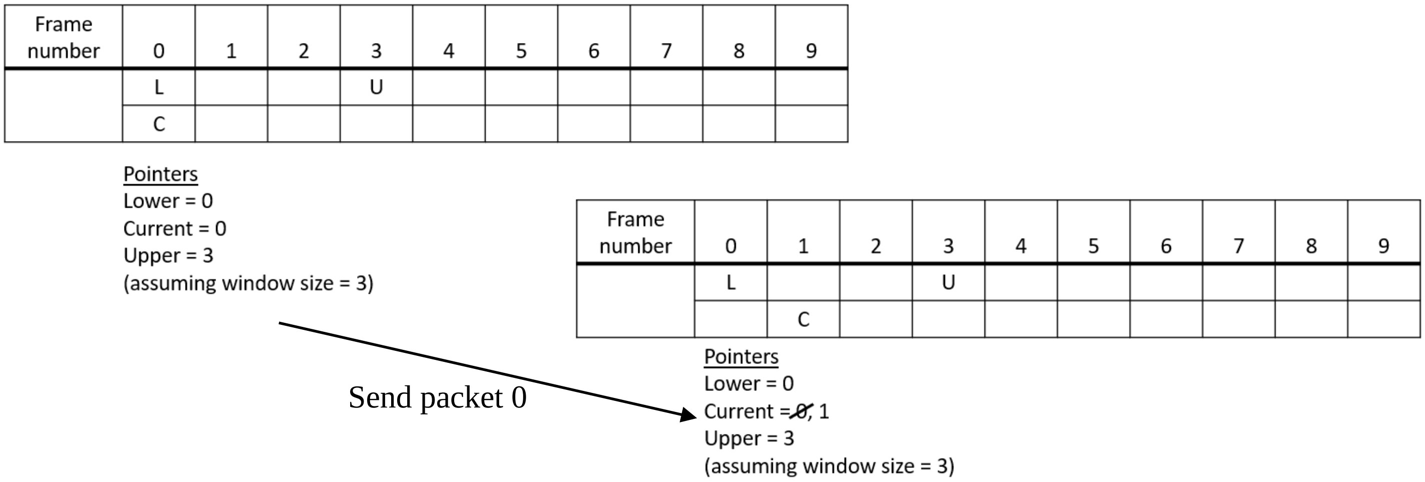
c. Data frame number is the one you are expecting but you have already received frames with higher numbers.

   This packet was sent in response to a SREJ. RR the highest received frame (if you have sent multiple SREJ, then RR up until the next sequence number waiting on) then unbuffer the higher frames. If you do not have an outstanding SREJ, discard the packet and do not respond.

[1] The window is closed if you have sent window size number of packets and have not received any acknowledgment for those packets.

5) Managing the Server's window: Walk through the following examples using the technique I showed in class using the Lower, Upper and Current indexes for the server's window. I am ok with hand drawn and you don't need to repeat everything, just cross out the numbers as your window moves. I just want you to understand how the window indexes work.

Example (note - I show 2 tables so I can demonstrate the movement of the pointers, you can just use the same table and cross out the pointers as they change. This example below assumes a window size of 3.



Frame number	0	1	2	3	4	5	6	7	8	9
	L			U						
	C									

Pointers
Lower = 0
Current = 0
Upper = 3
(assuming window size = 3)

Send packet 0

Frame number	0	1	2	3	4	5	6	7	8	9
	L			U						
		C								

Pointers
Lower = 0
Current = ~~0~~, 1
Upper = 3
(assuming window size = 3)

a. Walk through the scenario: send frame 0, 1, 2 and then receive RR 1, receive RR 3, send frame 3 and 4. (window size = 3)

Frame num	0	1	2	3	4	5	6	7	8	9
Initial	L, C			U						
Send(0)	L	C		U						
Send(1)	L		C	U						
Send(2)	L			C,U						
RR(1)		L		C	U					
RR(3)				L,C			U			
Send(3)				L	C		U			
Send(4)				L		C	U			

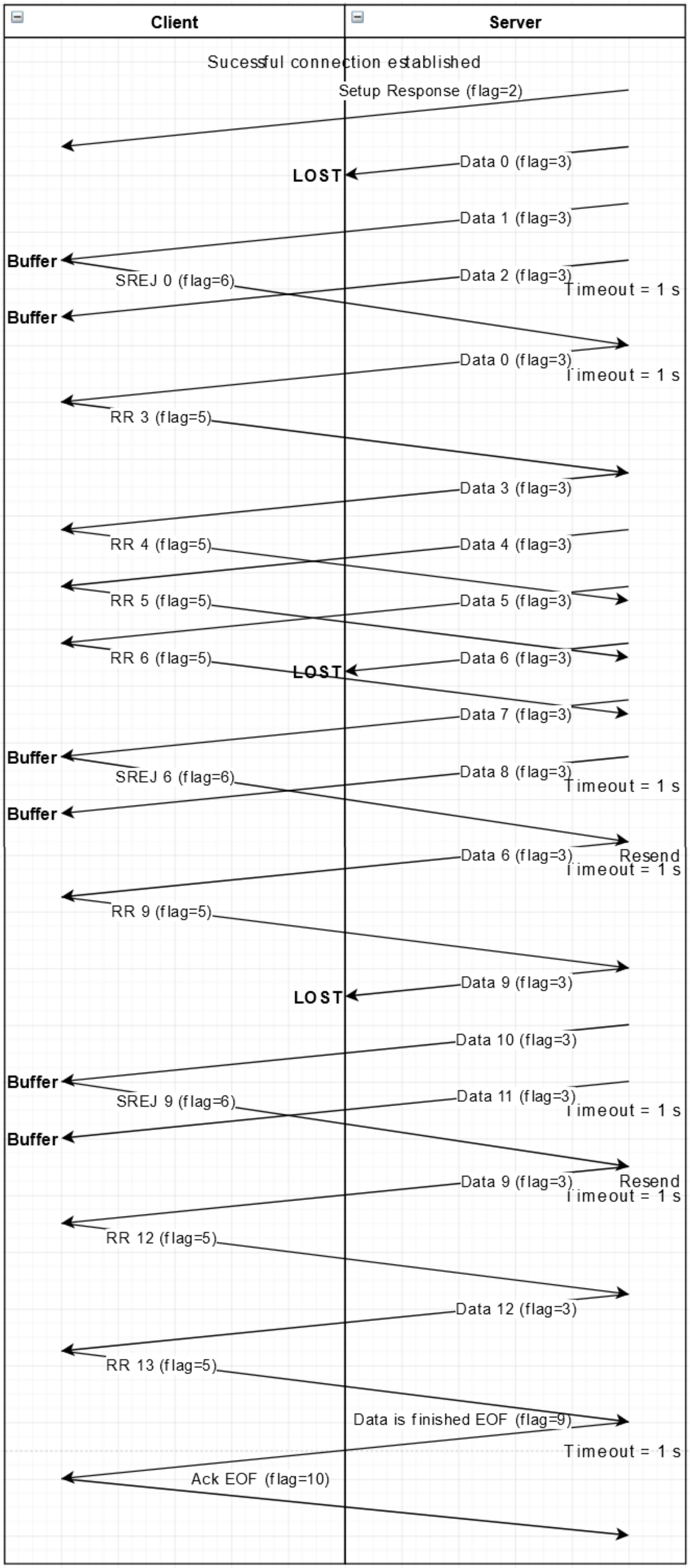
b. Walk through the scenario: send frame 0, 1, receive RR 1, send frames 2, 3, receive SREJ 2, resend 2, receive RR 4. (window size = 3)

Frame num	0	1	2	3	4	5	6	7	8	9
Initial	L, C			U						
Send(0)	L	C		U						
Send(1)	L		C	U						
RR(1)		L	C		U					
Send(2)		L		C	U					
Send(3)		L			C,U					

SREJ(2)		L			C,U					
Resend(2)		L			C,U					
RR(4)					L,C			U		

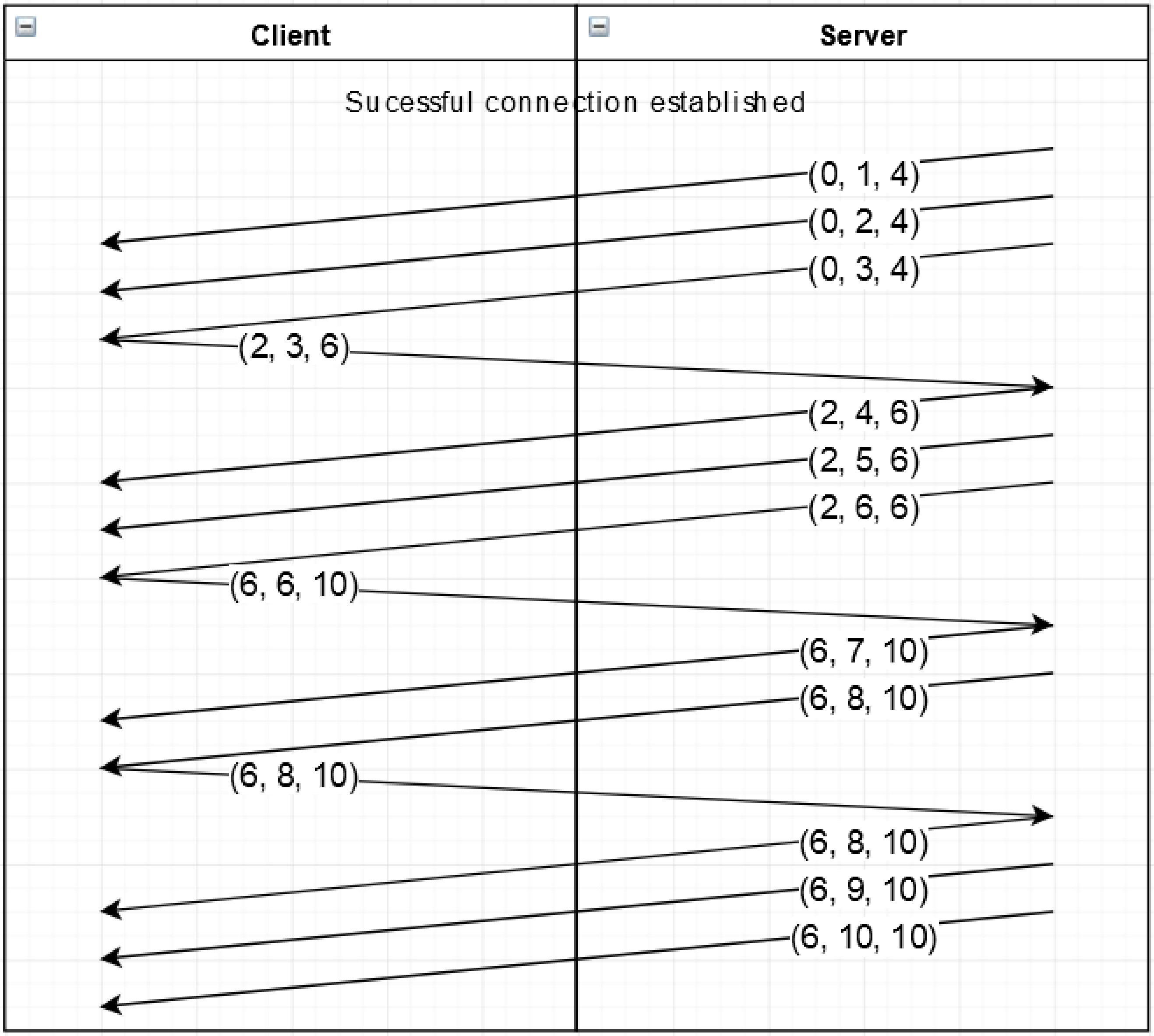
6) Assume your program receives data frames 1,2,3,4,5 and then receives frame 7 and 8 and then frames 10, 11 and 12 (some recovered packet(s) is/are needed to make this work). Draw a packet flow diagram of your solution using a window size of 3. Discuss how your implementation will handle this situation. How are RR's and SREJ's sent? What packets are sent/resent?

Packets 0, 6, and 9 are lost so they must be resent. RR's are sent on every received packet except when waiting on a SREJ resent packet. An RR for the greatest packet number received after a SREJ resent packet is received.


Client
Server
Sucessful connection established
Setup Response (flag=2)
Data 0 (flag=3)
LOST
Data 1 (flag=3)
Buffer
SREJ 0 (flag=6)
Data 2 (flag=3)
Timeout = 1 s
Buffer
Data 0 (flag=3)
Timeout = 1 s
RR 3 (flag=5)
Data 3 (flag=3)
RR 4 (flag=5)
Data 4 (flag=3)
RR 5 (flag=5)
Data 5 (flag=3)
RR 6 (flag=5)
Data 6 (flag=3)
LOST
Data 7 (flag=3)
Buffer
SREJ 6 (flag=6)
Data 8 (flag=3)
Timeout = 1 s
Buffer
Data 6 (flag=3)
Resend
Timeout = 1 s
RR 9 (flag=5)
Data 9 (flag=3)
LOST
Data 10 (flag=3)
Buffer
SREJ 9 (flag=6)
Data 11 (flag=3)
Timeout = 1 s
Buffer
Data 9 (flag=3)
Resend
Timeout = 1 s
RR 12 (flag=5)
Data 12 (flag=3)
RR 13 (flag=5)
Data is finished EOF (flag=9)
Timeout = 1 s
Ack EOF (flag=10)

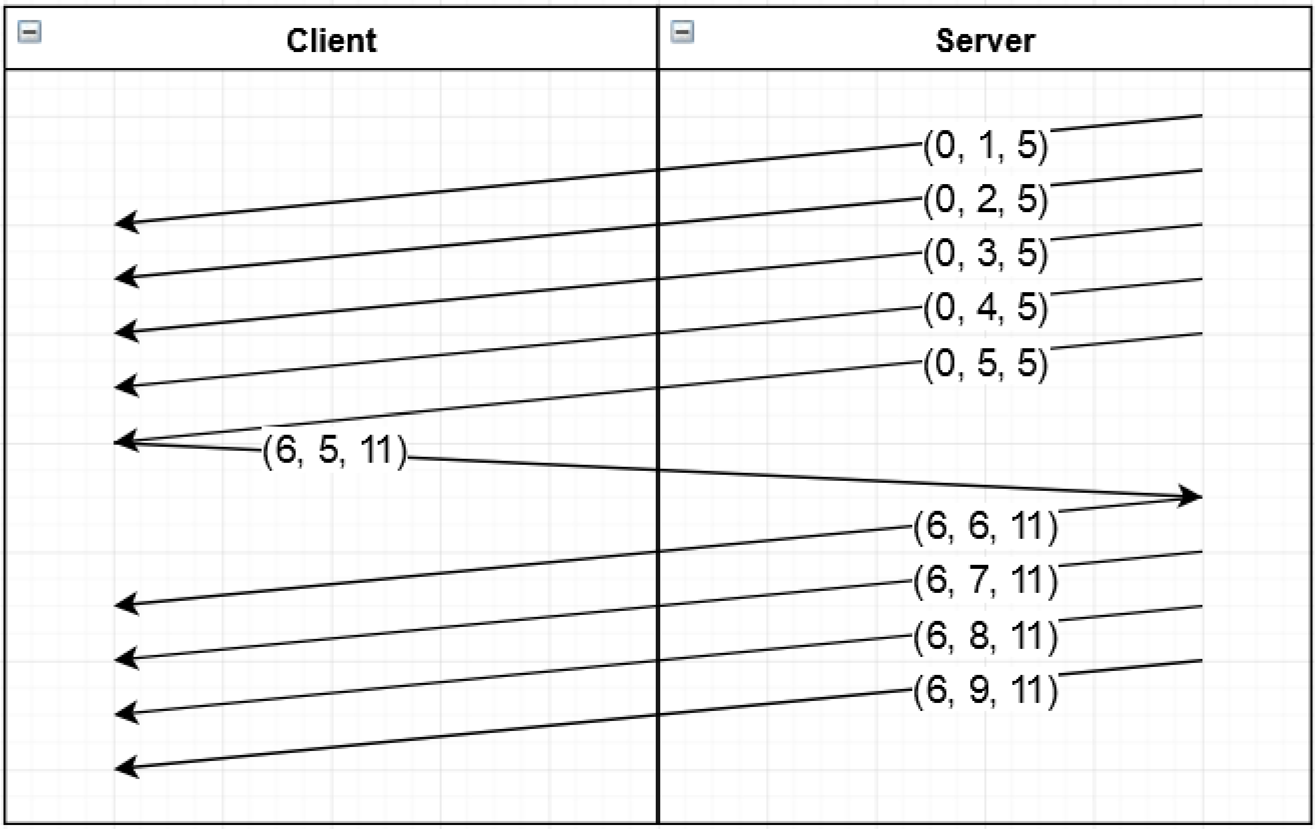
7) Draw a packet flow diagram for the following scenario but include for sender of the data **only** the value for the bottom of the window, lower edge of the window and upper edge of the window for each send and receive. You can use a notation like (Bottom of window, current frame, upper edge)… so (3,3,6) would mean the bottom of the window is 3 the current to send is 3 and the upper edge of the window is 6. (sender is the one sending the file data)[2]

   a. With a window size of 5: sender sends data packets 1-3, sender receives RR 2, sender sends packets 4,5,6, sender receives RR 6, sender sends packets 7 and 8, sender receives SREJ 7, sender sends 7, 9 and 10

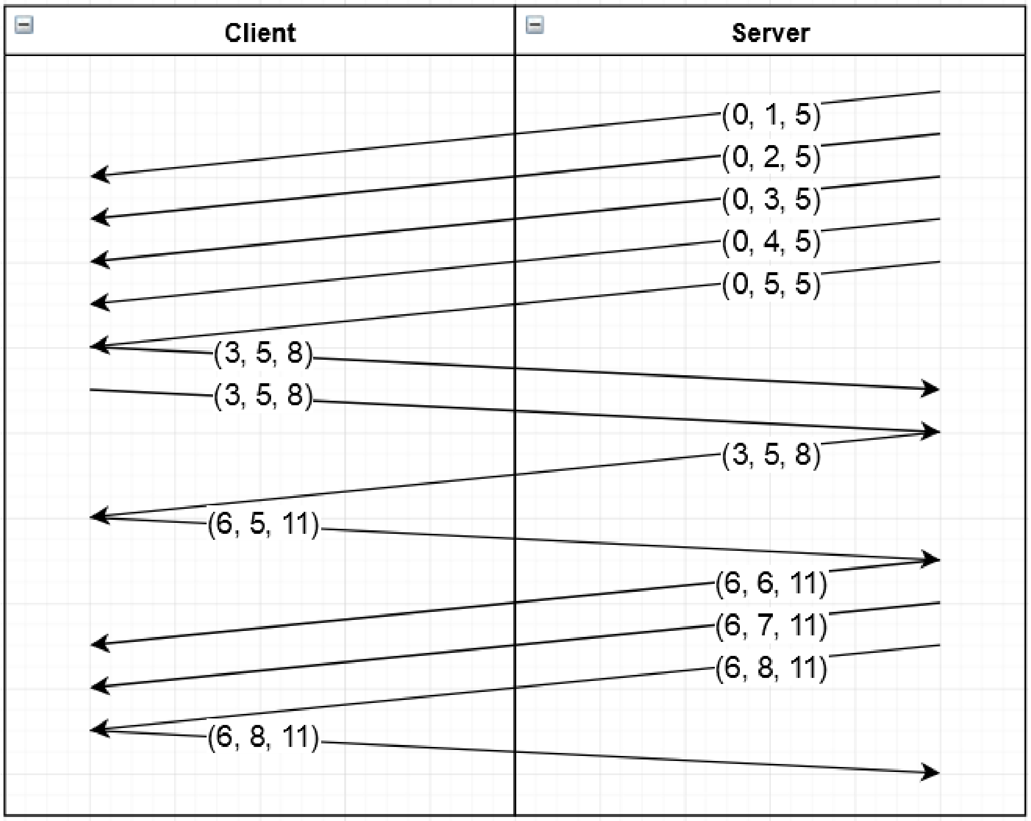


[2] While this problem only lists a few RRs, I would recommend you send an RR every time a good packet (in order, error free) arrives. It is more complex to accumulate RRs and only send them every few packets. But, this is a design decision and is up to you.

b. With a window size of 5: sender sends data packets 1-5, sender receives nothing and timeouts, sender resends packet 1, sender receives RR 6, sender sends 6, 7, 8, 9



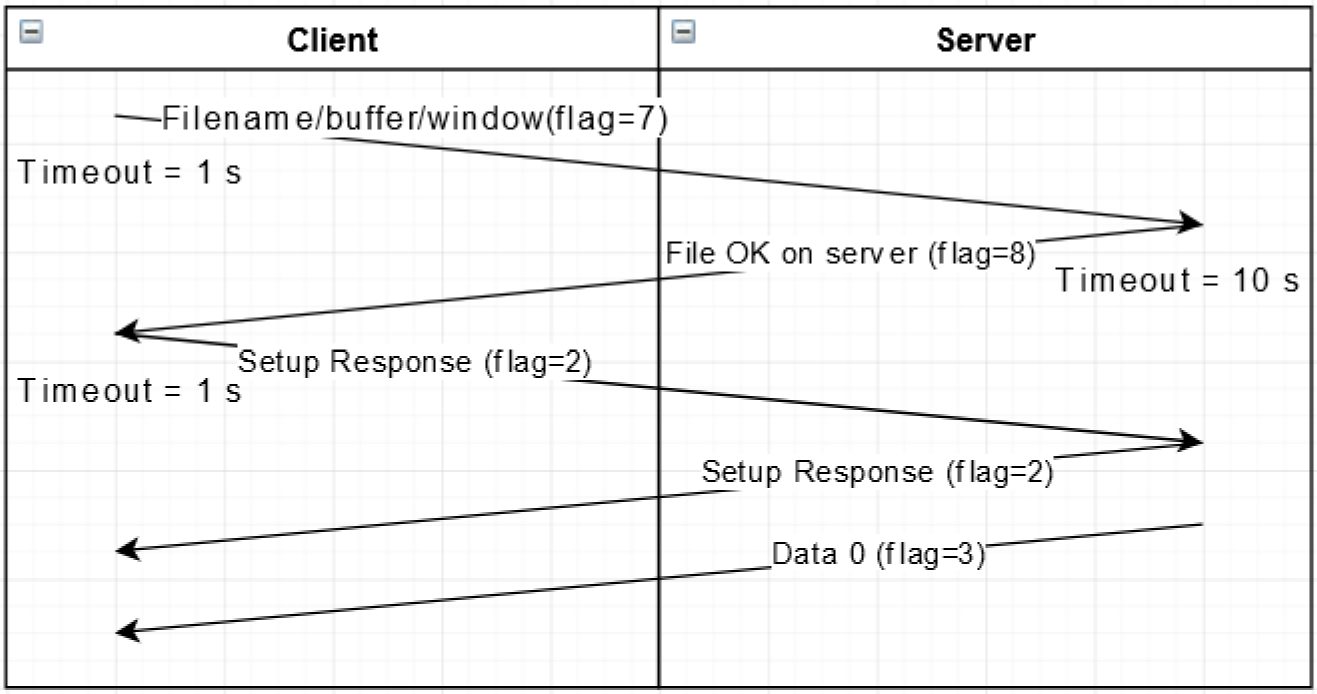
c. With a window size of 5: sender sends data packets 1-5, sender receives RR 3, sender receives SREJ 3, sender resends packet 3, sender receives RR 6, sender sends 6, 7, 8, sender receives SREJ 6



(16)

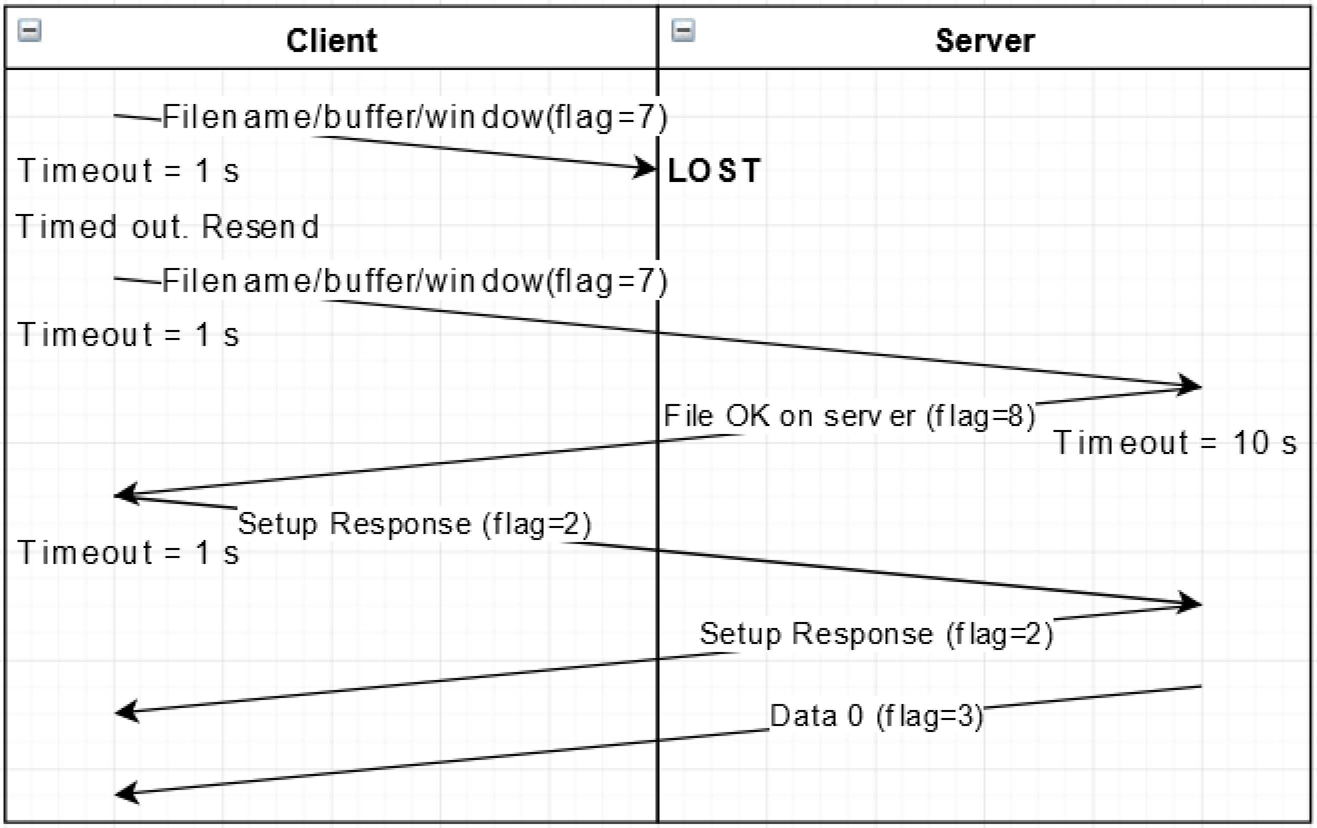
8) Draw a packet flow diagram for the following scenarios for the connection establishment and then **filename** exchange (end the diagram with the first data packet being successfully received).

a. No packets lost



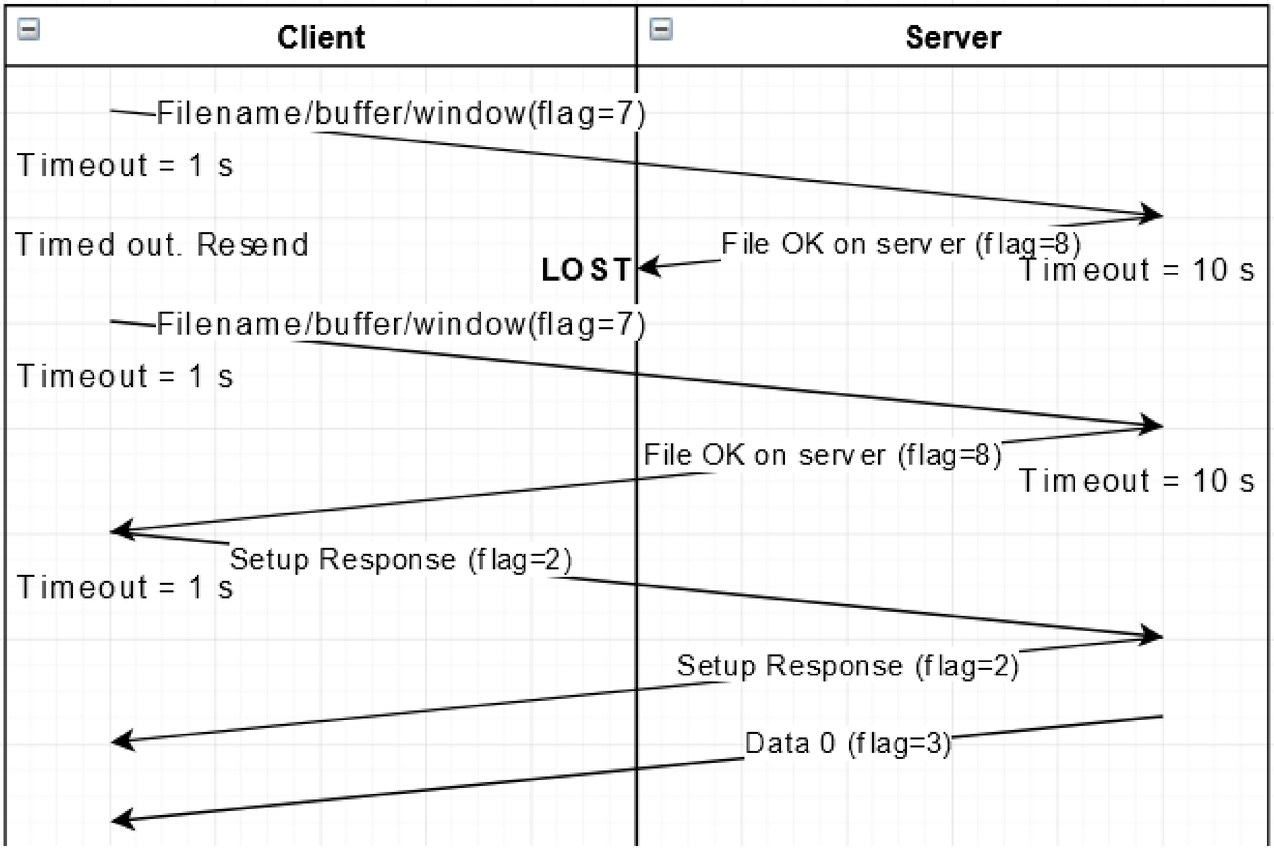
b. First packet (flag=1) is lost from rcopy
I do not use flag=1. See 8c for first packet lost.

c. Second packet sent by rcopy is lost (filename packet).

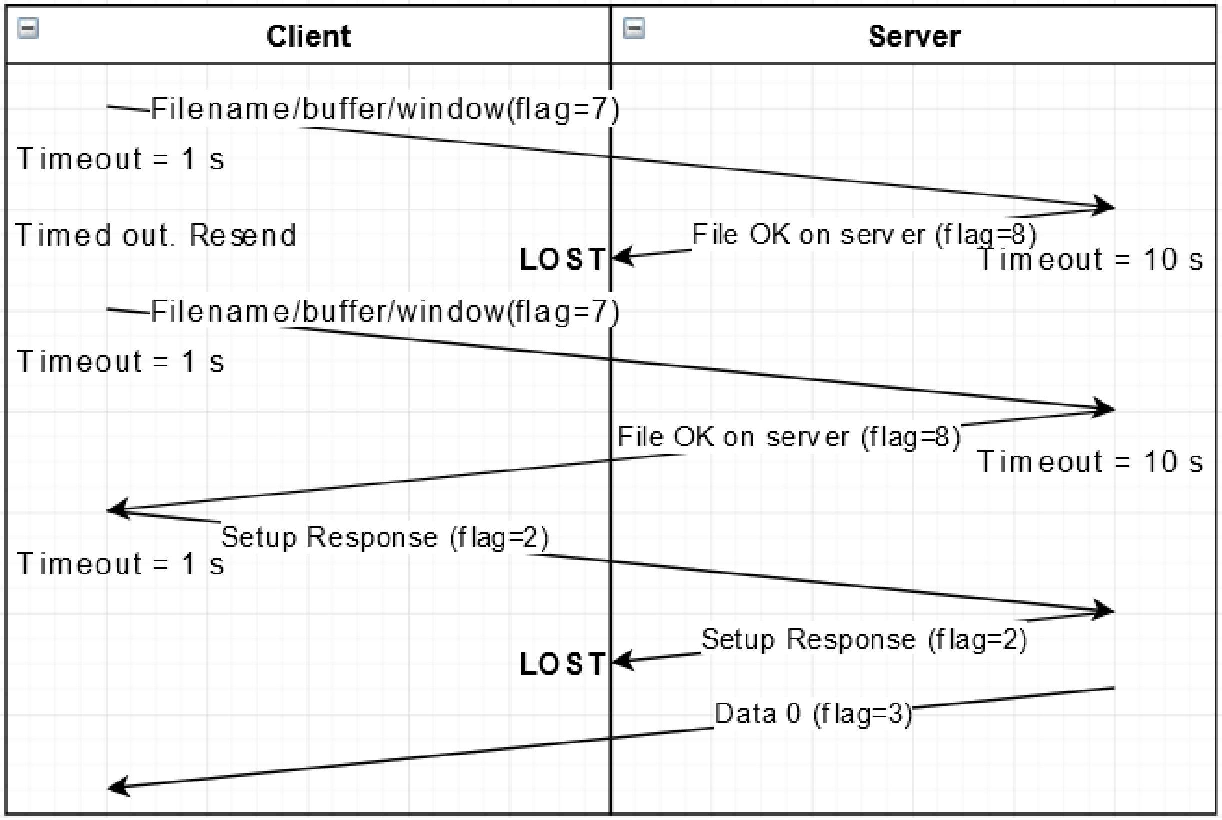


d. First packet sent by the server is lost. (flag = 2)

I do not use flag=2 in my initial setup. Below is first packet sent by server (flag=8) lost.

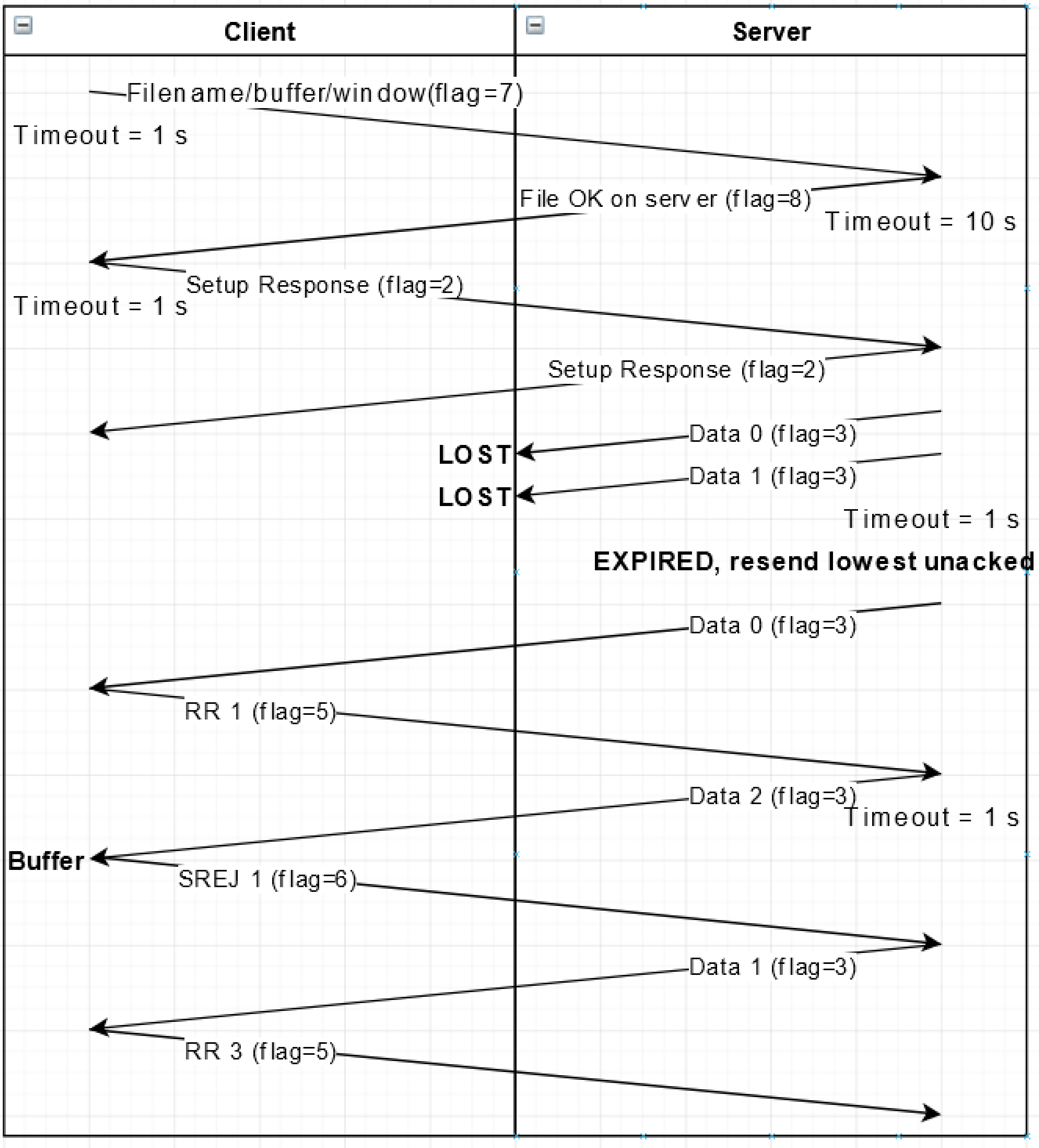


e. First two packets sent by the server are lost. (flag = 2 packets)

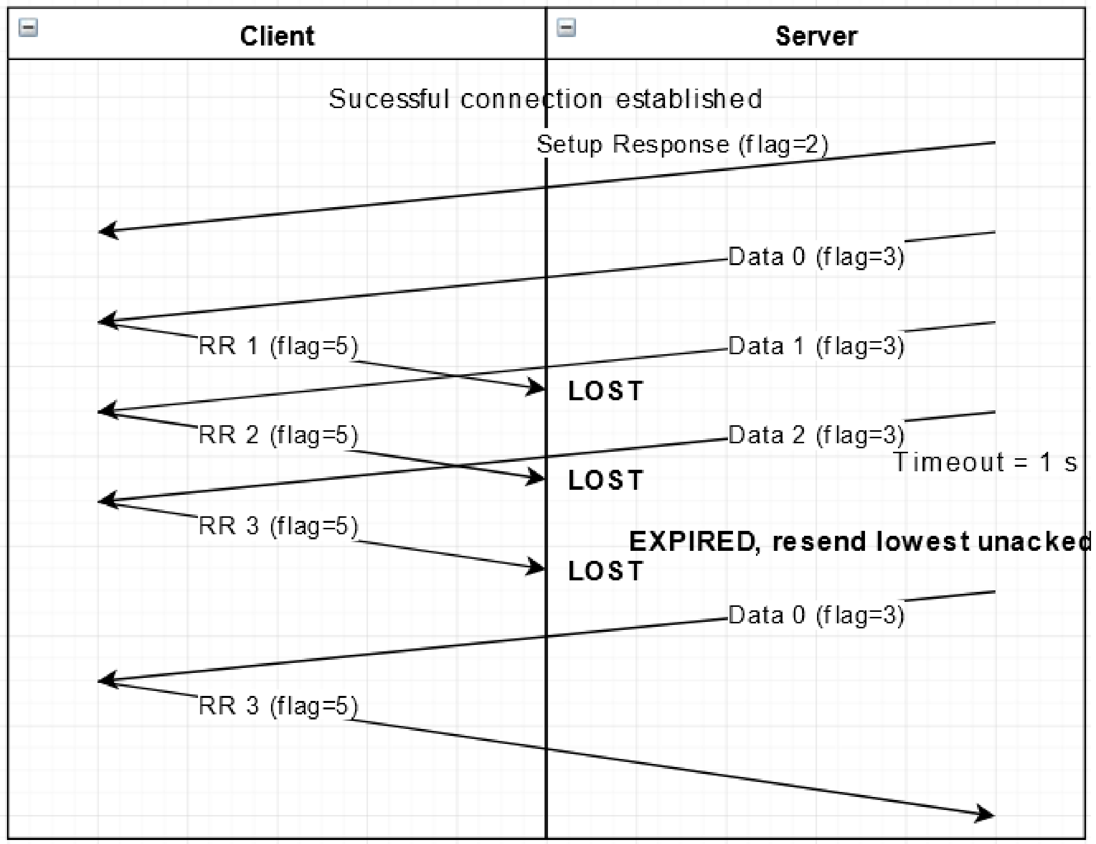


I think I'm not quite using setup response as intended, but it makes more sense to me to use flag=2 than to create a new flag. I use it to cleanly end the setup process, not initialize it. If the setup response is lost and a data packet is received instead, the client assumes setup response was lost and moves on to accepting data.

f. After filename exchange and you start to send data, what happens if the entire first window of data is lost

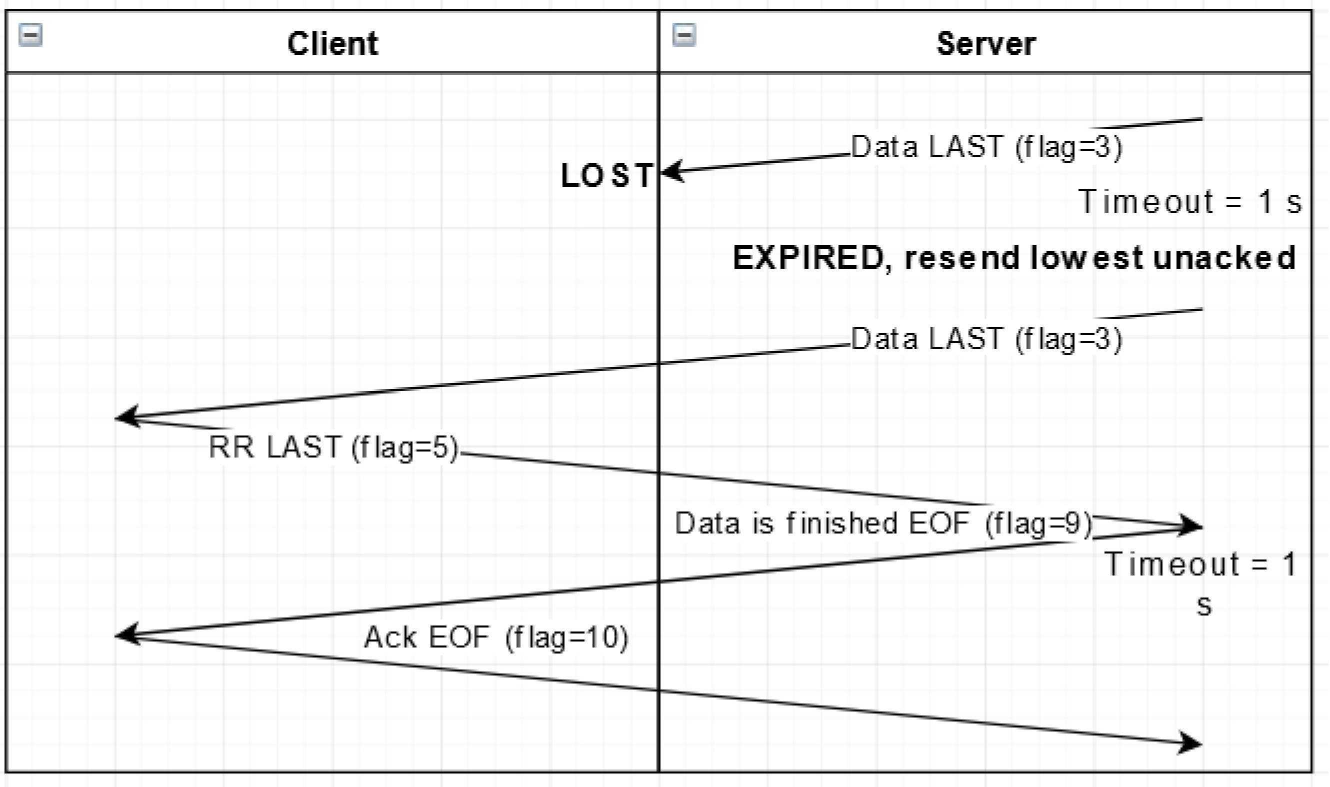


g. After filename exchange and you start to send data what happens if an entire window of RRs are lost.

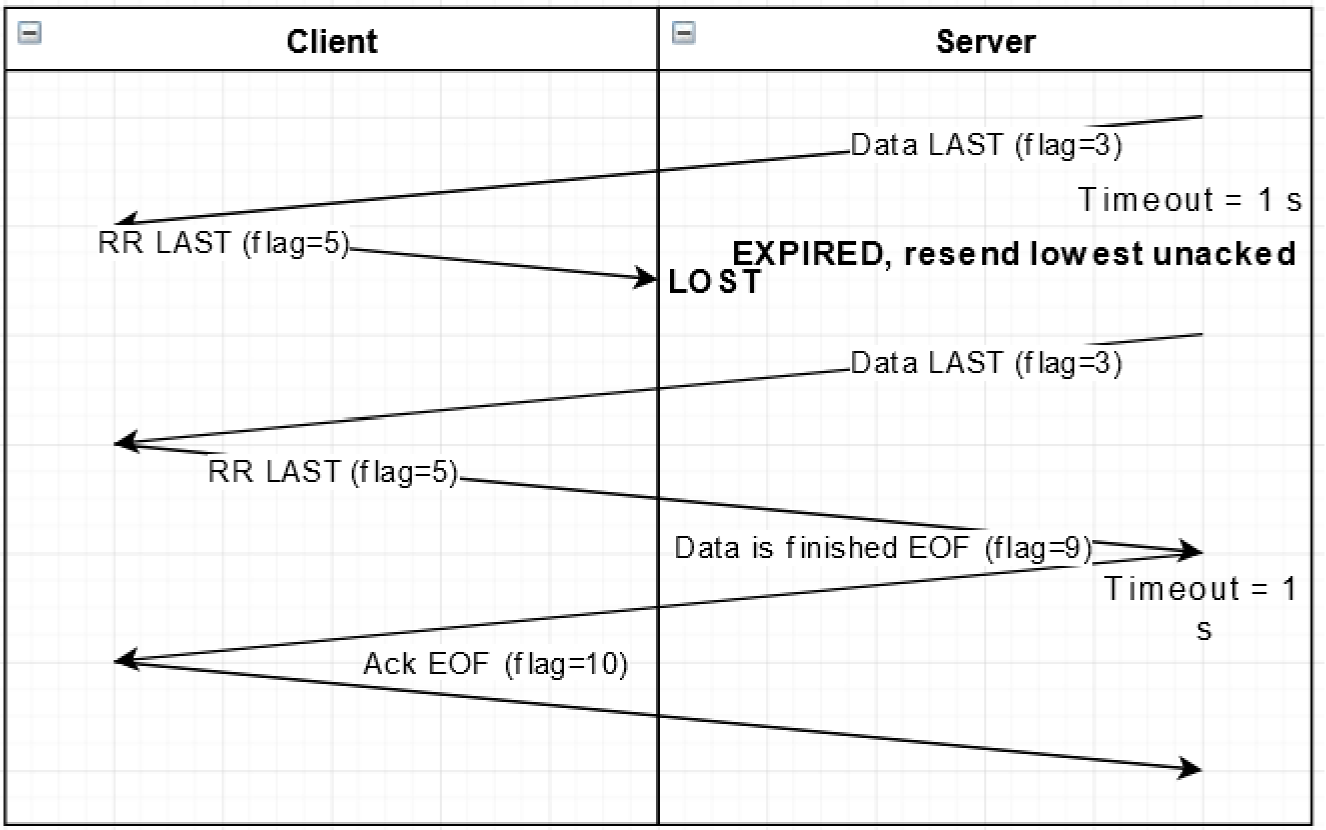


9) Give a packet flow diagram on how you will handle the last packet of the file.

a. Last data packet is lost



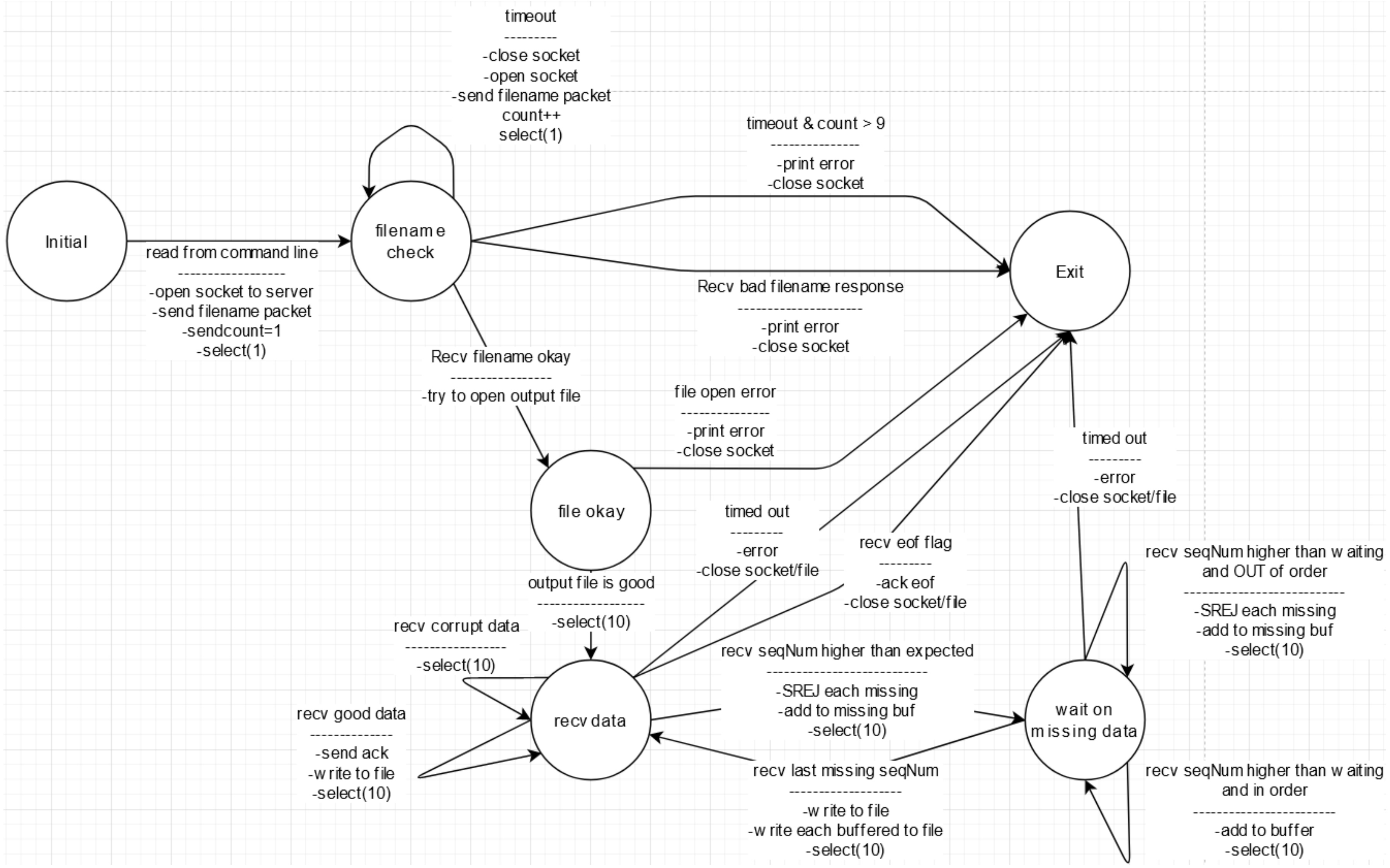
b. RR for last data packet is lost.



c. The 2nd and 5th data packet from the end is lost (so the final data packet gets there but the one before that does not… and the one 4 before that does not). Your window size is 7.

Part II – State diagrams

Turn in a state diagram for Rcopy and another for server (so two diagrams). These diagrams should start with the filename exchange and cover the sending/acking of the last packet. Only worry about the file transfer between one rcopy program and one server process/thread.


Initial
read from command line

-open socket to server
-send filename packet
-sendcount=1
-select(1)
filenam e
check
timeout

-close socket
-open socket
-send filename packet
count++
select(1)
timeout & count > 9

-print error
-close socket
Recv bad filename response

-print error
-close socket
Exit
Recv filename okay

-try to open output file
file okay
file open error

-print error
-close socket
output file is good

-select(10)
recv data
recv corrupt data

-select(10)
recv good data

-send ack
-w rite to file
-select(10)
timed out

-error
-close socket/file
recv eof flag

-ack eof
-close socket/file
recv seqNum higher than expected

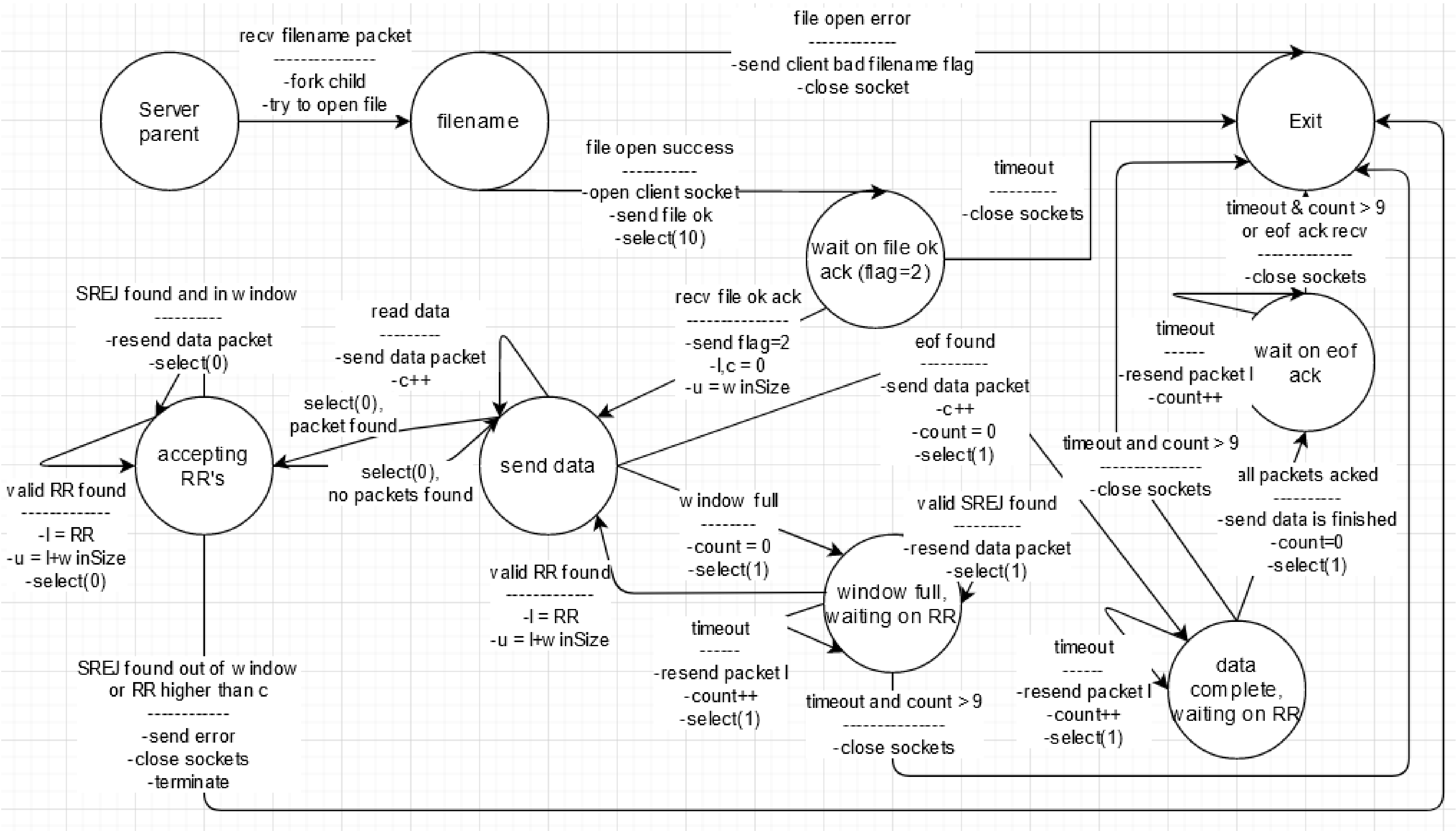
-SREJ each missing
-add to missing buf
-select(10)
recv last missing seqNum

-w rite to file
-w rite each buffered to file
-select(10)
wait on
m issing data
timed out

-error
-close socket/file
recv seqNum higher than w aiting
and OUT of order

-SREJ each missing
-add to missing buf
-select(10)
recv seqNum higher than w aiting
and in order

-add to buffer
-select(10)


Server parent
recv filename packet

-fork child
-try to open file
filename
file open error

-send client bad filename flag
-close socket
Exit
file open success

-open client socket
-send file ok
-select(10)
wait on file ok ack (flag=2)
timeout

-close sockets
timeout & count > 9
or eof ack recv

-close sockets
SREJ found and in window

-resend data packet
-select(0)
read data

-send data packet
-c++
recv file ok ack

-send flag=2
-l,c = 0
-u = winSize
eof found

-send data packet
-c++
-count = 0
-select(1)
timeout

-resend packet l
-count++
wait on eof ack
select(0),
packet found
accepting RR's
send data
select(0),
no packets found
timeout and count > 9

-close sockets
all packets acked

-send data is finished
-count=0
-select(1)
valid RR found

-l = RR
-u = l+w inSize
-select(0)
window full

-count = 0
-select(1)
valid SREJ found

-resend data packet
-select(1)
valid RR found

-l = RR
-u = l+w inSize
window full,
waiting on RR
timeout

-resend packet l
-count++
-select(1)
timeout

-resend packet l
-count++
-select(1)
data complete,
waiting on RR
SREJ found out of window
or RR higher than c

-send error
-close sockets
-terminate
timeout and count > 9

-close sockets

Part III – Windowing Library

As part of program #3, you are going to implement a library (so a .c and .h file) that implements the windowing functionality (including the window data structure and accessor code) needed by both rcopy and server. Note the needs for managing the window data structure in the rcopy and server programs are slightly different but do overlap. You window must be managed as a circular queue.

a. Reread the program requirements regarding the **window library**.

b. Starting with the server, look at your answers to part I and II above and walk through using a window data structure on your server (e.g look at the upper, lower, and current variables, what happens when a packet is sent, an RR arrives…) and answer the following three questions[3]:

- What actions do you need to take to manage the data structure (e.g. add a packet to you window data structure)

  To add a packet to the circular array, you need to add it in the array at index c and increment c.

- What questions do you need to ask about your window (e.g. is the window closed?)

  You need to check if there is room in the window before you add it to the structure.

- What data do you need to maintain to manage your window?

  You need the window size, current index, lower index, and upper index.

c. Now look at rcopy, look at your answers to part I and II above and walk through using a window data structure in rcopy, answer the following three questions:

- What actions do you need to take to manage the data structure (e.g. add a packet to you window data structure)

  If the packet is within the expected window, you can add it to the data structure at the seqNum % window size index.

- What questions do you need to ask about your window

  You need to ask what packets are missing from your window, what is the lowest missing packet, and how many missing packets are there.

- What data do you need to maintain to manage your window?

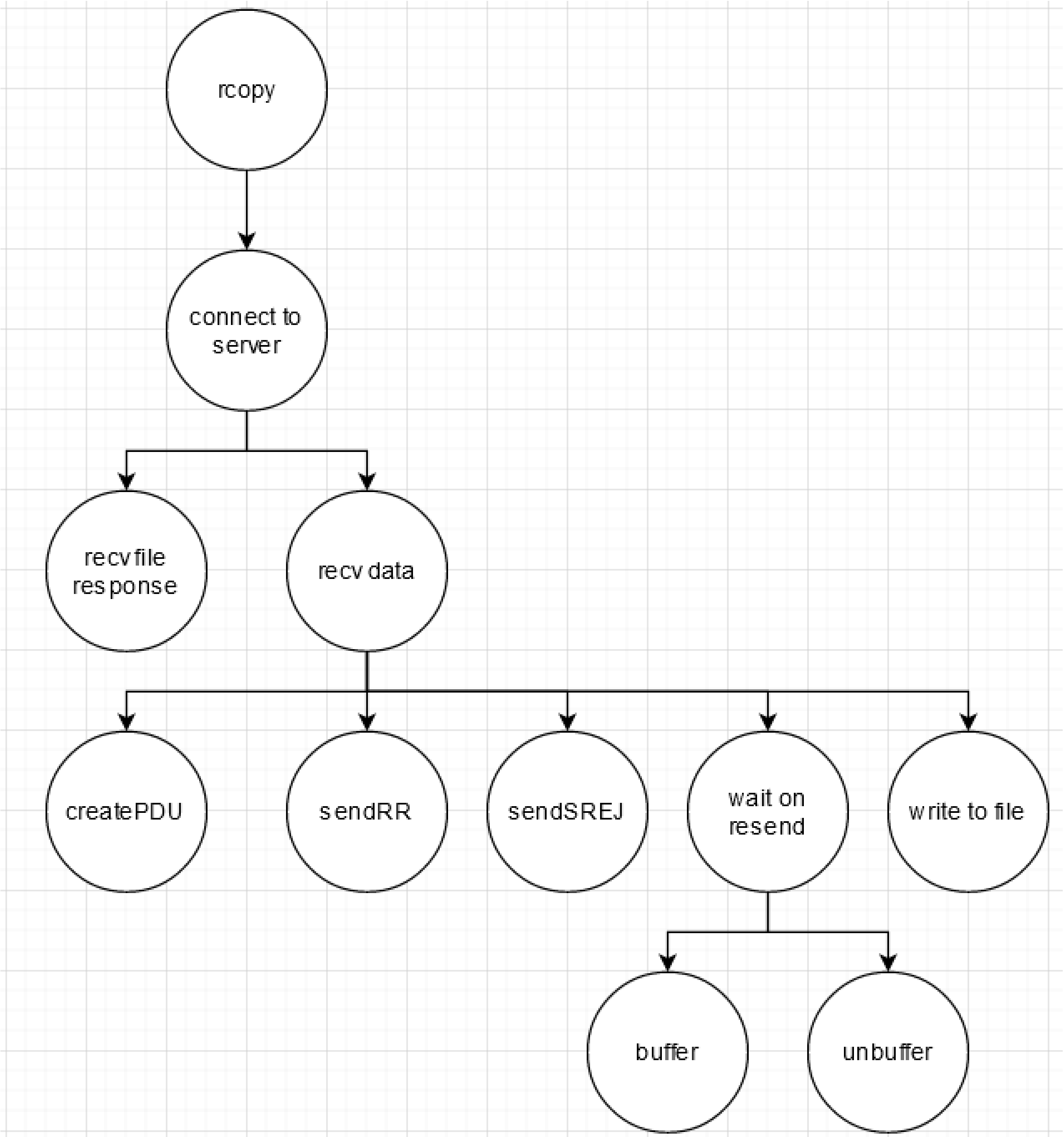
  You need to know the last acked packet and what packets are missing.

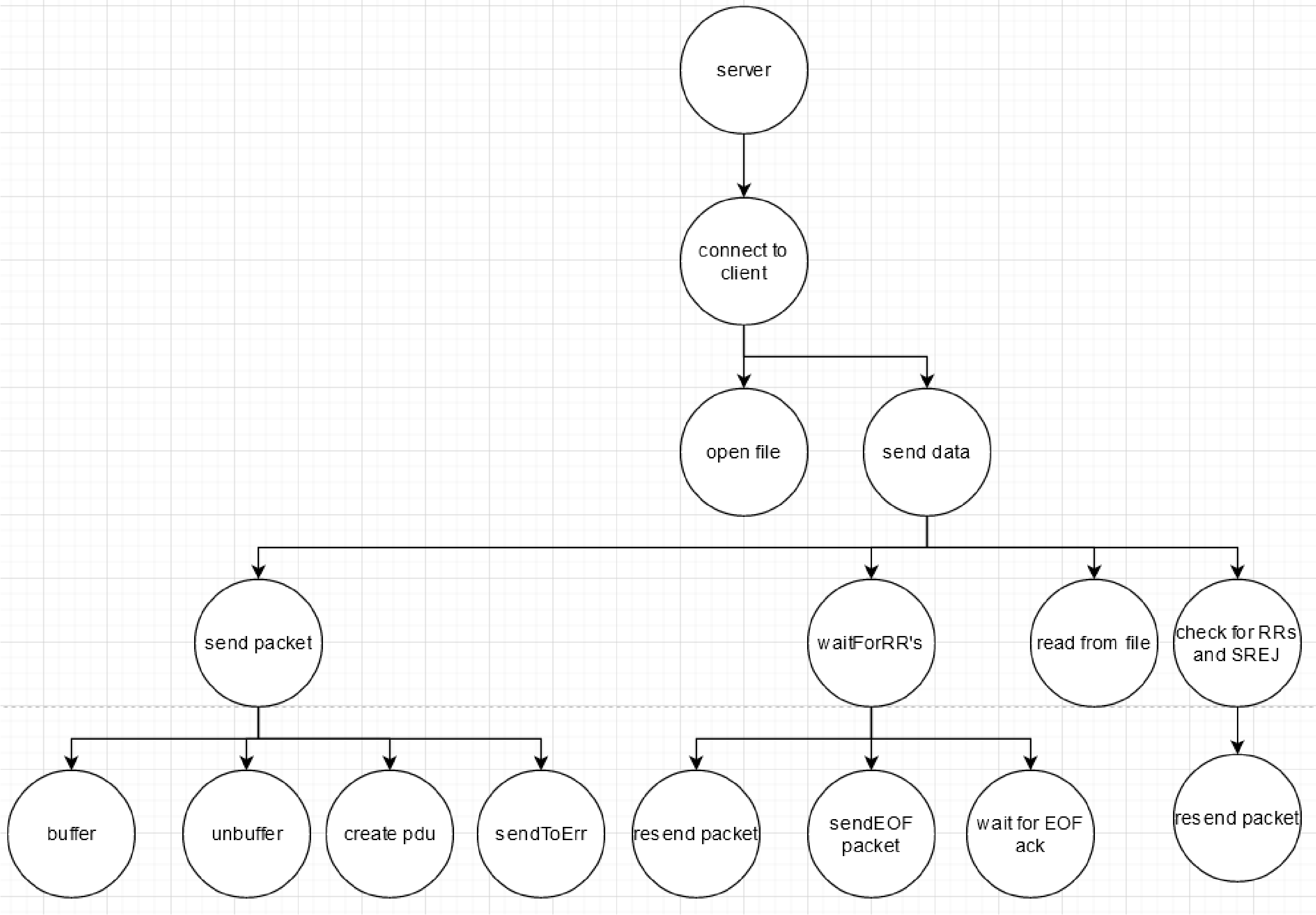
Part IV – Function hierarchy drawing and listing common code

This section has two parts:

a. **Function hierarchy drawing:** Turn in a <u>hierarchical (e.g. a tree of functions) drawing</u> of each of your programs (one for rcopy one for server). This is a course grain overview of your implementation. Many of the branches in this tree should be at least 4 nodes in depth. The goal is to identify the worker function (lower functions used to support your higher level logic) that are common between the rcopy and server and also within rcopy and server.

[3] Remember, you are designing and implementing a library to manage your windowing. This means you need to figure out what **data** you need for your window and what **accessor functions** you need to utilize your window in your rcopy and server code.


rcopy
connect to server
recv file response
recv data
createPDU
sendRR
sendSREJ
wait on resend
write to file
buffer
unbuffer



b. Common Code: List the functionality that is common between the programs

Looking at your design including your hierarchical drawings, what functionality is common between the two programs (rcopy and server)? This is functionality that will be written in a separate code file (.c and related .h) and linked into rcopy and server as part of creating your executables (i.e. in your Makefile).

- Windowing/buffering
- Sending and receiving PDU's
- file management
- mallocing/freeing
- creating and interpreting PDU's

Make a copy of your packet flows, state diagrams and code breakdown before you turn them in… you will not get them back in time to help you write the program.