LAB4: - SLIDING WINDOW PROTOCOL

CS212 (Computer Networks)

Kanwar Raj Singh (1903122)

Afzal Hussain (2103104)

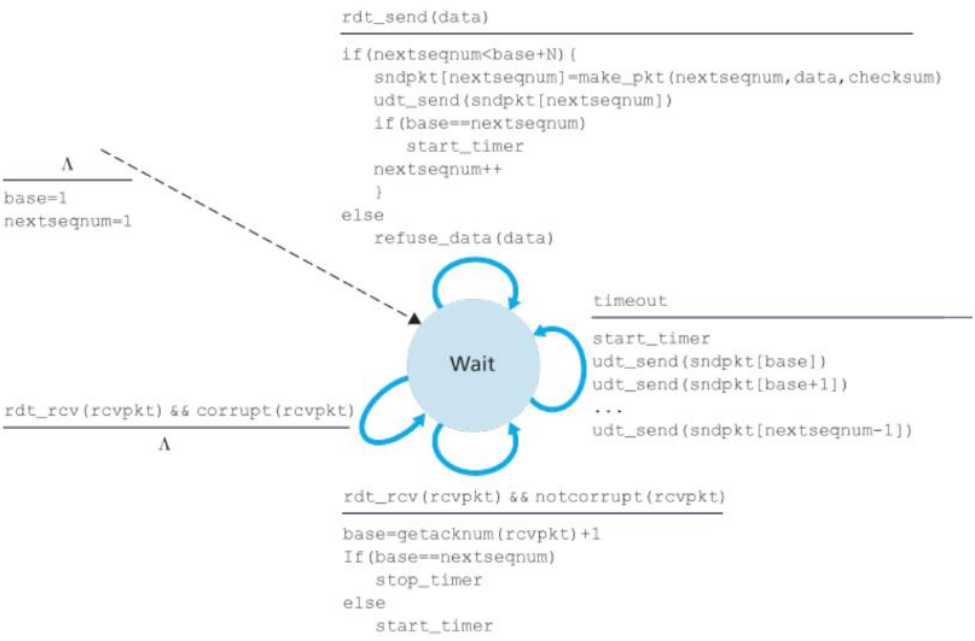


Figure 3.20 Extended FSM description of the GBN sender

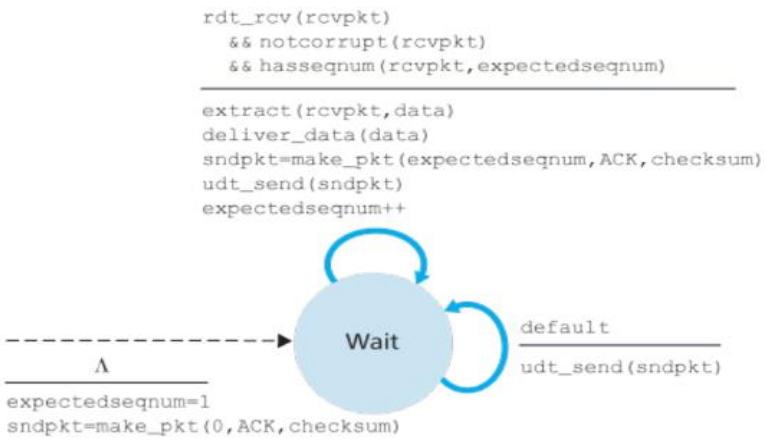


Figure 3.21 Extended FSM description of the GBN receiver

Q1.

(a) Yes, it’s matching almost properly.

(b) One major difference I observe between the Python implementation for the RDT sender/receiver classes and the FSM description in the textbook is that the Python implementation uses a combination of Go-Back-N and Selective Repeat ARQ protocols for reliable data transfer, whereas the FSM description only describes a basic Stop-and-Wait ARQ protocol.

In the Python implementation, the sender maintains a sliding window of packets to be transmitted and waits for acknowledgments from the receiver before sliding the window forward. The receiver, on the other hand, buffers out-of-order packets and sends cumulative acknowledgments to the sender indicating the last correctly received packet.

In contrast, the FSM description only includes a basic Stop-and-Wait ARQ protocol, where the sender sends one packet at a time and waits for an acknowledgment before sending the next packet.

Overall, the Python implementation is more complex and sophisticated than the basic FSM description in the textbook

(c) Raw Output after running Given Code snippet.

```
rajkanwar@Kanwarraj:/mnt/c/Users/Student/Desktop/CS212/LAB4$ python3 Testbench.py
TIME: 0 Current window: [1, 2, 3, 4, 5] base = 1 nextseqnum = 1
--------------------
TIME: 1 SENDING APP: trying to send data 1
TIME: 1 RDT_SENDER: rdt_send() called for nextseqnum= 1  within current window. Sending new packet.
TIME: 1 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, packet_length=100 bits, corrupted=False)
TIME: 1 TIMER STARTED for a timeout of  5
TIME: 1 Current window: [1, 2, 3, 4, 5] base = 1 nextseqnum = 2
--------------------
--------------------
TIME: 27 TIMER RESTARTED for a timeout of  5
TIME: 27 RDT_SENDER: Got an ACK 7 . Updated window: [8, 9, 10, 11, 12] base = 8 nextseqnum = 11
TIME: 27 TIMER RESTARTED for a timeout of  5
TIME: 27 RDT_SENDER: Got an ACK 8 . Updated window: [9, 10, 11, 12, 13] base = 9 nextseqnum = 11
TIME: 27 TIMER RESTARTED for a timeout of  5
TIME: 27 RDT_SENDER: Got an ACK 9 . Updated window: [10, 11, 12, 13, 14] base = 10 nextseqnum = 11
TIME: 27 SENDING APP: trying to send data 11
TIME: 27 RDT_SENDER: rdt_send() called for nextseqnum= 11  within current window. Sending new packet.
TIME: 27 DATA_CHANNEL : udt_send called for Packet(seq_num=11, payload=11, packet_length=100 bits, corrupted=False)
TIME: 27 Current window: [10, 11, 12, 13, 14] base = 10 nextseqnum = 12
--------------------
Receiving application received 10 messages. Halting simulation.
================================================
 SIMULATION RESULTS:
================================================
Total number of messages sent by the Sending App= 14
Total number of messages received by the Receiving App=10
Total number of DATA packets sent by rdt_Sender=29
Total number of re-transmitted DATA packets=15 (51.72% of total packets sent)
Total number of ACK packets sent by rdt_Receiver=21
Total number of re-transmitted ACK packets=11 (52.38% of total packets sent)
Utilization for the DATA channel=8.53%
Utilization for the  ACK channel=0.62%
```

Yes, it is running for both Pl and Pc having non-zero values(0.1).

Q2. Output after Implementing the testbench as given in question is attached below

```
Receiving application received 1000 messages. Halting simulation.
================================================
 SIMULATION RESULTS:
================================================
Total number of messages sent by the Sending App= 1008
Total number of messages received by the Receiving App=1000
Total number of DATA packets sent by rdt_Sender=6289
Total number of re-transmitted DATA packets=5281 (83.97% of total packets sent)
Total number of ACK packets sent by rdt_Receiver=5048
Total number of re-transmitted ACK packets=4048 (80.19% of total packets sent)
Utilization for the DATA channel=154.63%
Utilization for the  ACK channel=1.24%



================================================
5 SIMULATION AVERAGE RESULTS:
================================================
Time taken =  [4124, 4606, 4500, 4100, 4067]
Channel Utilisation =  [151.67313288069835, 154.27702996092054, 153.6, 154.14634146341464, 154.6348659945906]
Retransmission Rate =  [83.91686650679456, 85.80073177596398, 85.47453703703704, 84.03481012658229, 83.9720146287168]

Average time taken for receiving application to receive 1000 msgs:  4279.4
Average channel utilisation for DATA channel:  153.6662740599248
Average retransmission rate:  84.63979201501894
```

a) Attached below:

```
================================================
5 SIMULATION AVERAGE RESULTS:
================================================
Time taken =  [4124, 4606, 4500, 4100, 4067]
```

b) Channel utilization attached below:

```
Channel Utilisation =  [151.67313288069835, 154.27702996092054, 153.6, 154.14634146341464, 154.6348659945906]
```

Average across five simulation runs is:

```
Average channel utilisation for DATA channel:  153.6662740599248
```

c) Re-transmitted packet:

```
Retransmission Rate =  [83.91686650679456, 85.80073177596398, 85.47453703703704, 84.03481012658229, 83.9720146287168]
```

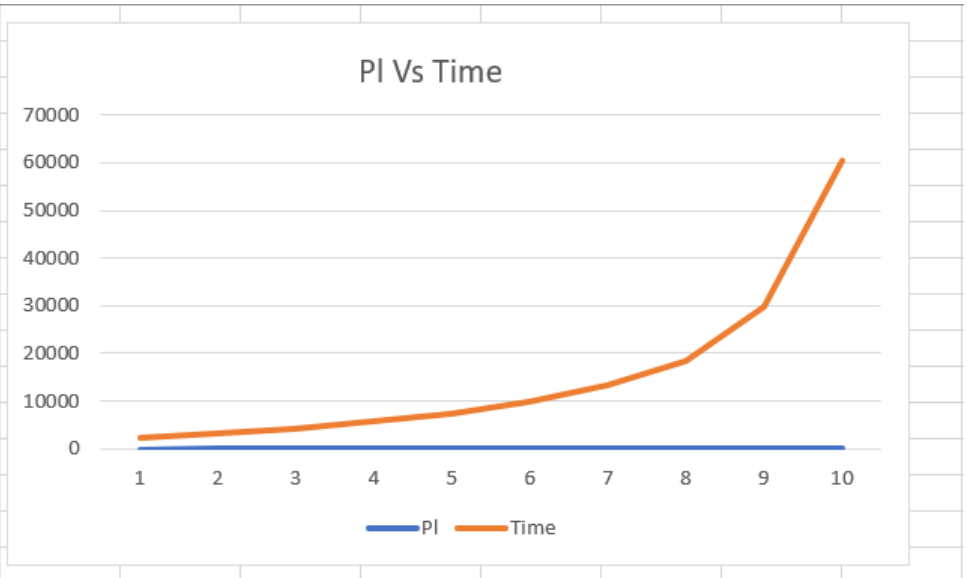
Average Re-transmitted packet:

```
Average retransmission rate:  84.63979201501894
```

Q3.

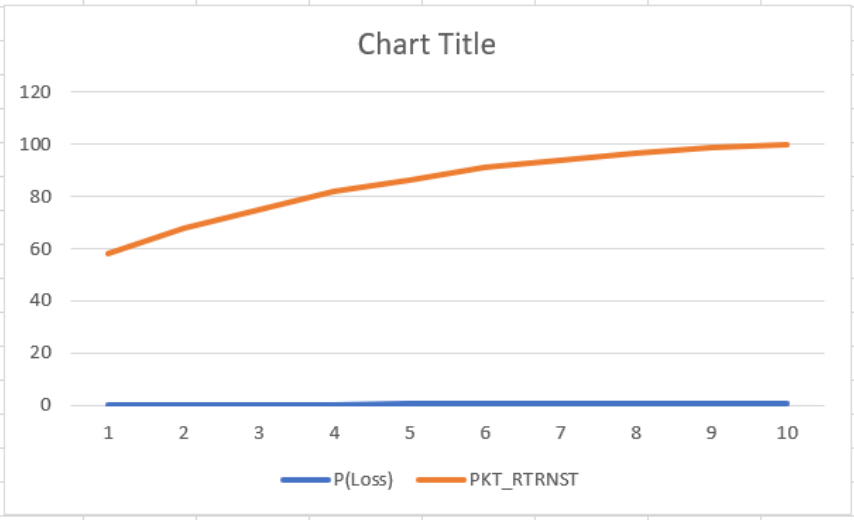
a) Pc = 0.1 and Graph between Pl and Time T is attached below:

Pl	Time
0	2235
0.1	3090
0.2	4330
0.3	5662
0.4	7405
0.5	9892
0.6	13395
0.7	18523
0.8	29838
0.9	60507



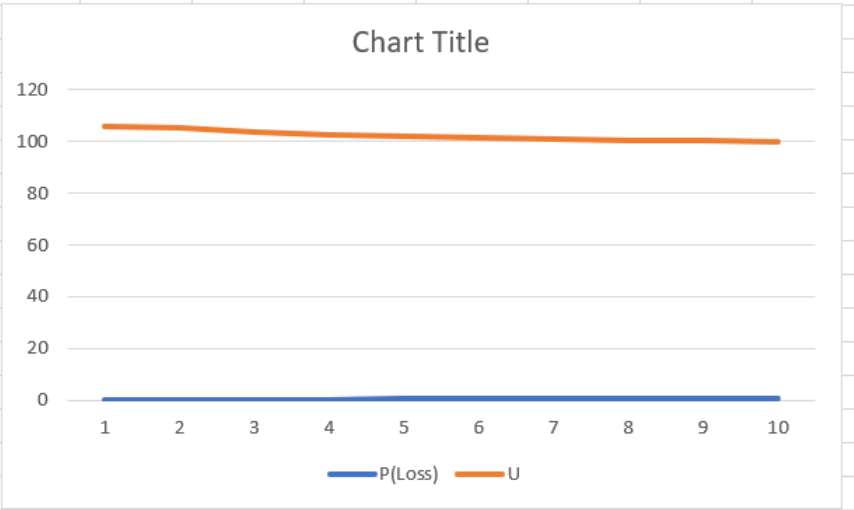
b) Plot between percentage of packet that were re-transmitted and P(loss)

P(Loss)	PKT_RTRNST
0	58.11
0.1	67.73
0.2	74.98
0.3	81.72
0.4	86.54
0.5	91.34
0.6	93.93
0.7	96.69
0.8	98.69
0.9	99.66



c) Plot Between Data Channel Utilization and P(loss)

P(Loss)	U
0	105.91
0.1	105.13
0.2	103.79
0.3	102.8
0.4	102.04
0.5	101.54
0.6	100.96
0.7	100.51
0.8	100.25
0.9	100.06



Q4.

a) To quantify "best performance" in this case, we can consider metrics such as throughput, latency, and reliability. Throughput refers to the amount of data that can be transmitted per unit time, while latency measures the time delay between the transmission of a packet and its successful reception. Reliability refers to the probability that a packet is successfully delivered without errors. Ideally, we would like to maximize throughput, minimize latency, and maximize reliability.

b) If the window size N is too small, the transmission rate will be low, and the protocol will not be able to take advantage of the available channel capacity. On the other hand, if N is too large, the protocol may cause congestion, resulting in packet loss and increased latency. The value of N is limited by the channel capacity, the propagation delay, and the processing capabilities of the sender and receiver.

c) To achieve the best performance, we can use the following values for N, K, and timeout:

N = 20: This value provides a good balance between throughput and reliability. With a packet length of 1000 bits and a transmission rate of 1000 bits per second, it would take 1 second to transmit a single packet. Therefore, a window size of 20 packets would allow the sender to transmit 20 packets in 20 seconds, achieving a throughput of 20,000 bits per second. This value also provides sufficient redundancy to recover from packet loss.

K = 2: This value allows for selective repeat, which enables the receiver to request retransmission of only the lost packets, reducing the latency and conserving network resources.

Timeout = 4: This value should be set to a multiple of the round-trip time (RTT) to account for the propagation delay. With a propagation delay of 2, the RTT is 4. Therefore, a timeout value of 4 provides sufficient time for the sender to receive an ACK before retransmitting a packet.

Overall, these values should provide a good balance between throughput, latency, and reliability while minimizing packet loss and congestion. However, the specific values may need to be adjusted based on the actual network conditions and performance requirements.

Q5.

1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
2. *Timeout.* Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to `send_base`, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

Sender Side For SR

1. *Packet with sequence number in* `[rcv_base, rcv_base+N-1]` *is correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (`rcv_base` in Figure 3.22), then this packet, and any previously buffered and consecutively numbered (beginning with `rcv_base`) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 3.26. When a packet with a sequence number of `rcv_base=2` is received, it and packets 3, 4, and 5 can be delivered to the upper layer.
2. *Packet with sequence number in* `[rcv_base-N, rcv_base-1]` *is correctly received.* In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.
3. *Otherwise.* Ignore the packet.

Receiver side for SR

Q6.

a) for Pl = 0 and Pc = 0, Output attached below;

```
Receiving application received 1000 messages. Halting simulation.
================================================
 SIMULATION RESULTS:
================================================
Total number of messages sent by the Sending App= 1001
Total number of messages received by the Receiving App=1000
Total number of DATA packets sent by rdt_Sender=1001
Total number of re-transmitted DATA packets=0 (0.00% of total packets sent)
Total number of ACK packets sent by rdt_Receiver=1000
Total number of re-transmitted ACK packets=0 (0.00% of total packets sent)
Utilization for the DATA channel=9.99%
Utilization for the  ACK channel=1.00%
```

b) Pl=0, Pc=0.5 for both DATA and ACK channels

```
TIME: 6983 RECEIVING APP: received data 1000


Receiving application received 1000 messages. Halting simulation.
================================================
 SIMULATION RESULTS:
================================================
Total number of messages sent by the Sending App= 1001
Total number of messages received by the Receiving App=1000
Total number of DATA packets sent by rdt_Sender=7141
Total number of re-transmitted DATA packets=6140 (85.98% of total packets sent)
Total number of ACK packets sent by rdt_Receiver=3541
Total number of re-transmitted ACK packets=0 (0.00% of total packets sent)
Utilization for the DATA channel=10.23%
Utilization for the  ACK channel=0.51%
```

C) Pl=0.5, Pc=0.5 for both DATA and ACK channels

```
Total simulation time has elapsed. Halting simulation.
================================================
 SIMULATION RESULTS:
================================================
Total number of messages sent by the Sending App= 677
Total number of messages received by the Receiving App=677
Total number of DATA packets sent by rdt_Sender=20115
Total number of re-transmitted DATA packets=19438 (96.63% of total packets sent)
Total number of ACK packets sent by rdt_Receiver=5029
Total number of re-transmitted ACK packets=0 (0.00% of total packets sent)
Utilization for the DATA channel=10.06%
Utilization for the  ACK channel=0.25%
```

D) For Window Size = 10;

```

Receiving application received 1004 messages. Halting simulation.
================================================
 SIMULATION RESULTS:
================================================
Total number of messages sent by the Sending App= 1005
Total number of messages received by the Receiving App=1004
Total number of DATA packets sent by rdt_Sender=36455
Total number of re-transmitted DATA packets=35450 (97.24% of total packets sent)
Total number of ACK packets sent by rdt_Receiver=9273
Total number of re-transmitted ACK packets=0 (0.00% of total packets sent)
Utilization for the DATA channel=19.96%
Utilization for the  ACK channel=0.51%
```

Window Size = 5

```
Total simulation time has elapsed. Halting simulation.
================================================
 SIMULATION RESULTS:
================================================
Total number of messages sent by the Sending App= 656
Total number of messages received by the Receiving App=655
Total number of DATA packets sent by rdt_Sender=20105
Total number of re-transmitted DATA packets=19449 (96.74% of total packets sent)
Total number of ACK packets sent by rdt_Receiver=5048
Total number of re-transmitted ACK packets=0 (0.00% of total packets sent)
Utilization for the DATA channel=10.05%
Utilization for the  ACK channel=0.25%
```

7) I set following values to check GBN and SR

◦ Pc=0.2, Pl=0.2, channel propagation delay = 2 seconds

◦ Packet length for DATA packets = 1000 bits, packet length for ACK packets = 10 bits

◦Transmission rate = 1000 bits per second

◦ Window size(N)=10, Range of sequence numbers(K)=32

Output of SR protocol:

```
Receiving application received 1002 messages. Halting simulation.
================================================
 SIMULATION RESULTS:
================================================
Total number of messages sent by the Sending App= 1006
Total number of messages received by the Receiving App=1002
Total number of DATA packets sent by rdt_Sender=4904
Total number of re-transmitted DATA packets=3898 (79.49% of total packets sent)
Total number of ACK packets sent by rdt_Receiver=3083
Total number of re-transmitted ACK packets=0 (0.00% of total packets sent)
Utilization for the DATA channel=194.60%
Utilization for the  ACK channel=1.22%
```

Output of GBN protocol:

```
Receiving application received 1000 messages. Halting simulation.
================================================
 SIMULATION RESULTS:
================================================
Total number of messages sent by the Sending App= 1005
Total number of messages received by the Receiving App=1000
Total number of DATA packets sent by rdt_Sender=6288
Total number of re-transmitted DATA packets=5283 (84.02% of total packets sent)
Total number of ACK packets sent by rdt_Receiver=5028
Total number of re-transmitted ACK packets=4028 (80.11% of total packets sent)
Utilization for the DATA channel=152.84%
Utilization for the  ACK channel=1.22%
```

Yes, SR protocol show better performance as compare to GBN.

Selective Repeat (SR) and Go-Back-N (GBN) are two protocols used for reliable data transmission over unreliable channels. Both protocols have their own advantages and disadvantages, and their performance depends on various factors such as channel conditions, packet loss probability, and network congestion.

In general, SR protocol performs better than GBN protocol in terms of efficiency and throughput, especially when the channel conditions are good and the packet loss probability is low. This is because SR protocol allows the receiver to acknowledge and store out-of-order packets, whereas GBN protocol discards all out-of-order packets and requires retransmission of all packets from the lost packet onwards.

However, if the packet loss probability is high or the network is congested, the performance of both protocols deteriorates, with GBN protocol showing slightly better performance in such scenarios. This is because GBN protocol uses cumulative acknowledgments and thus requires fewer retransmissions compared to SR protocol, which may lead to network congestion and increased packet loss.

In summary, the performance of SR and GBN protocols depends on various factors, and it is difficult to make a general comparison between the two. In most cases, SR protocol performs better than GBN protocol, but in high packet loss or congested scenarios, GBN protocol may show better performance.

Q.8) Following output I got, after setting up N == K;

```
TIME: 46 RDT_RECEIVER: Currently buffered packets:
TIME: 46 RDT_RECEIVER: Packet with seq_num= 8  and payload= 40
TIME: 46 RDT_RECEIVER: Packet with seq_num= 4  and payload= 4
TIME: 46 RDT_RECEIVER: Packet with seq_num= 13  and payload= 13
TIME: 46 RDT_RECEIVER: Packet with seq_num= 14  and payload= 14
TIME: 46 RDT_RECEIVER: Packet with seq_num= 5  and payload= 37
TIME: 46 RDT_RECEIVER: Packet with seq_num= 10  and payload= 10
TIME: 46 RDT_RECEIVER: Packet with seq_num= 15  and payload= 15
TIME: 46 RDT_RECEIVER: Packet with seq_num= 6  and payload= 6
TIME: 46 RDT_RECEIVER: Packet with seq_num= 11  and payload= 11
TIME: 46 RDT_RECEIVER: Packet with seq_num= 12  and payload= 44
TIME: 46 RDT_RECEIVER: Packet with seq_num= 18  and payload= 18
TIME: 46 RDT_RECEIVER: Packet with seq_num= 9  and payload= 41
TIME: 46 RDT_RECEIVER: Packet with seq_num= 20  and payload= 20
TIME: 46 RDT_RECEIVER: Packet with seq_num= 16  and payload= 16
TIME: 46 RDT_RECEIVER: Packet with seq_num= 17  and payload= 17
TIME: 46 RDT_RECEIVER: Packet with seq_num= 21  and payload= 21
TIME: 46 RDT_RECEIVER: Packet with seq_num= 27  and payload= 27
TIME: 46 RDT_RECEIVER: Packet with seq_num= 23  and payload= 23
TIME: 46 RDT_RECEIVER: Packet with seq_num= 28  and payload= 28
TIME: 46 RDT_RECEIVER: Packet with seq_num= 26  and payload= 26
TIME: 46 RDT_RECEIVER: Packet with seq_num= 24  and payload= 24
TIME: 46 RDT_RECEIVER: Packet with seq_num= 7  and payload= 39


TIME: 46 SENDING APP: trying to send data 46
TIME: 46 DATA_CHANNEL : udt_send called for Packet(seq_num=14, payload=46, packet_length=1000 bits, corrupted=False)
TIME: 46 TIMER STARTED for a timeout of  5 for packet 14
```

When implementing the Selective Repeat (SR) protocol, the range of sequence numbers must be large enough to avoid the possibility of wrapping around and reusing old sequence numbers. The range of sequence numbers is defined by the value of K, which is equal to the number of distinct sequence numbers that can be assigned.

The window size, represented by the variable N, determines the number of packets that can be transmitted without waiting for an acknowledgment. In the SR protocol, the receiver maintains a buffer to store out-of-order packets, and the window size determines the number of packets that can be stored in this buffer.

It is recommended that the value of K be at least twice as large as the window size, that is, N <= K/2. This is because if the window size N is equal to or greater than K/2, it is possible that some sequence

numbers may be reused before their corresponding packets have been acknowledged, which can lead to errors in the protocol.

If N = K, then the window size is equal to the range of sequence numbers, which means that the sender will use up all the sequence numbers before receiving any acknowledgments. This can result in the sender reusing old sequence numbers, which can cause confusion at the receiver and lead to errors in the protocol.

By ensuring that N <= K/2, the range of sequence numbers is kept sufficiently large to avoid wrapping around and reusing old sequence numbers, while at the same time limiting the window size to a safe and efficient value that will not cause errors in the protocol.