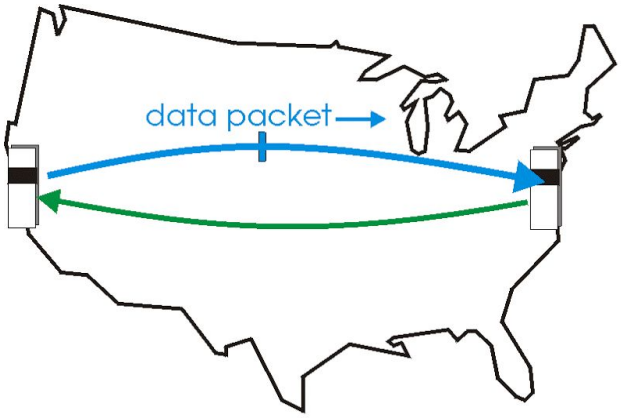
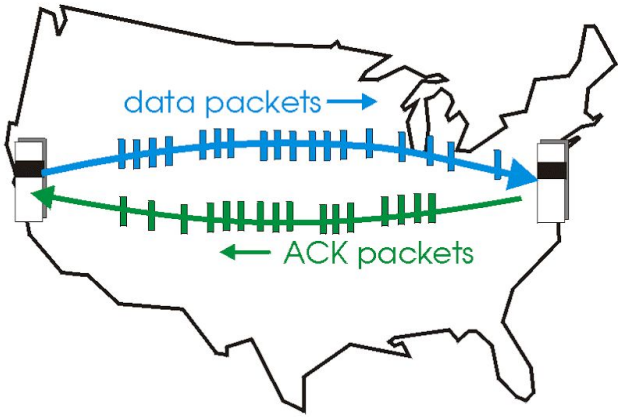
Reliable Data Transfer

Stop-and-Wait, Selective-Repeat and Go-Back-N



(a) a stop-and-wait protocol in operation



(b) a pipelined protocol in operation

Monica George Abdoh 3494
Fatma Nader 3346

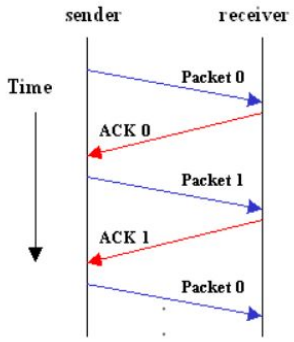
Stop and wait:

Known as ***"positive acknowledgement with retransmission"*** *, it is* is the fundamental technique to provide reliable transfer under unreliable packet delivery system.

How the protocol work:

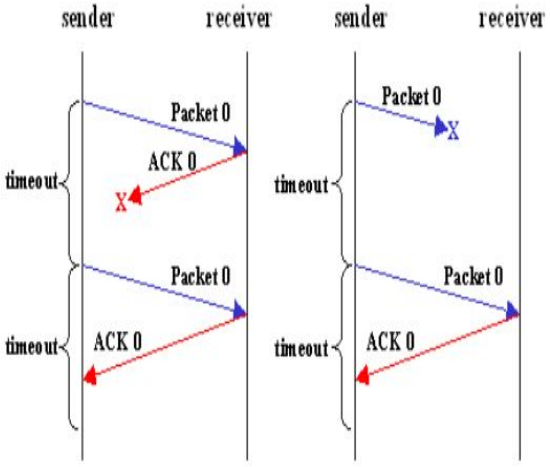
1)Normal way

After transmitting one packet, the sender waits for an ***acknowledgment (ACK)*** from the receiver before transmitting the next one.
this way, the sender can recognize that the previous packet is transmitted successfully and we could say "stop-n-wait" guarantees reliable transfer between nodes.
the sender keeps a record of each packet it sends.
,to avoid confusion caused by delayed or duplicated ACKs, "stop-n-wait" sends each packets with unique sequence numbers and receives that numbers in each ACKs.



2)Timeout

If the sender doesn't receive ACK for previous sent packet after a certain period of time, the sender *times out* and *retransmits* that packet again. There are two cases when the sender doesn't receive ACK; One is when the ACK is lost and the other is when the frame itself is not transmitted.



➔ How the protocol is implemented:

In the client side:

1) The client enters username and password in a packet sent to server.
2) A filename containing info of client is opened to know from it IP, well known port for server, client port, file to request and window size.
3) Client choose one of 3 modes to work with.
4) The first packet contains filename with the mode to work on.
5) A packet is received containing the thread server that will respond to the client and packets needed to handle the file.
6) The client starts the function that gets the file from server:
   The function loops until the sequence number is equal to the packets needed.
   A packet is received and check if there is error by comparing the checksum with the one calculated or comparing the sequence number received with the expected one. If there is an error it loops till the packet is received again correctly.
   If packet is received correctly an acknowledgement is sent containing the sequence number of packet.

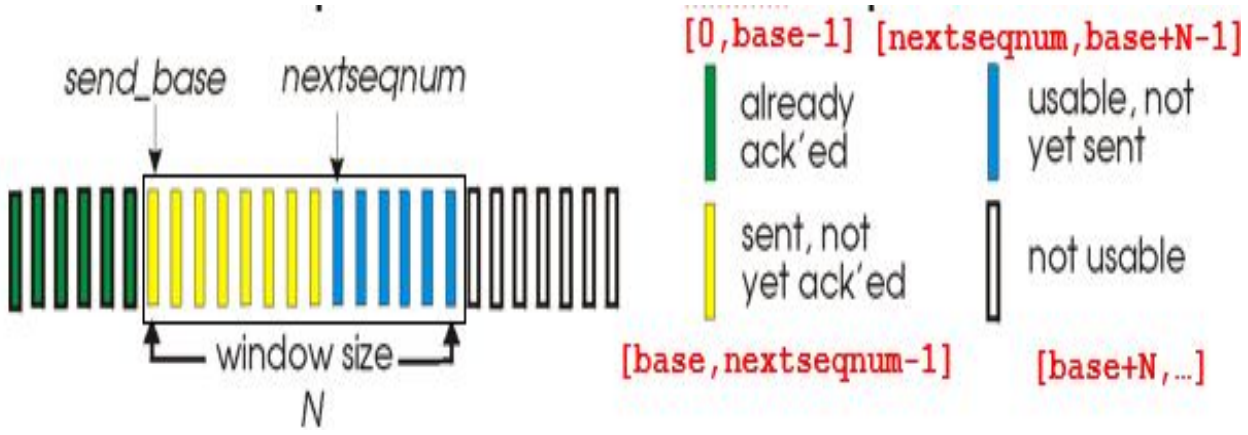
In the server side:

1) Gets information from server file containing server port, window size, random seed and probability of packet loss.
2) The seed is used to get the number of packet to be lost in the sending.
3) Server checks the username and password with an hashmap saved statically.
4) If username and password are correct an ack, then receive filename and mode client choose to run in.
5) The server makes a new responder (thread) for handling the client, the thread type is made upon the mode chosen.
6) A packet is sent containing the new server port for client to send to it and packets needed.
7) Function send file loops till the packets needed finishes, it sends the packet containing it's length in first 2 bytes, checksum in following 8 bytes, the last 4 bytes are the sequence number.
8) Server waits for an ack, function receive ack will return if the ack is positive or negative if there is a timeout, as long as it gives false the packet is resent.

2) Go back N:

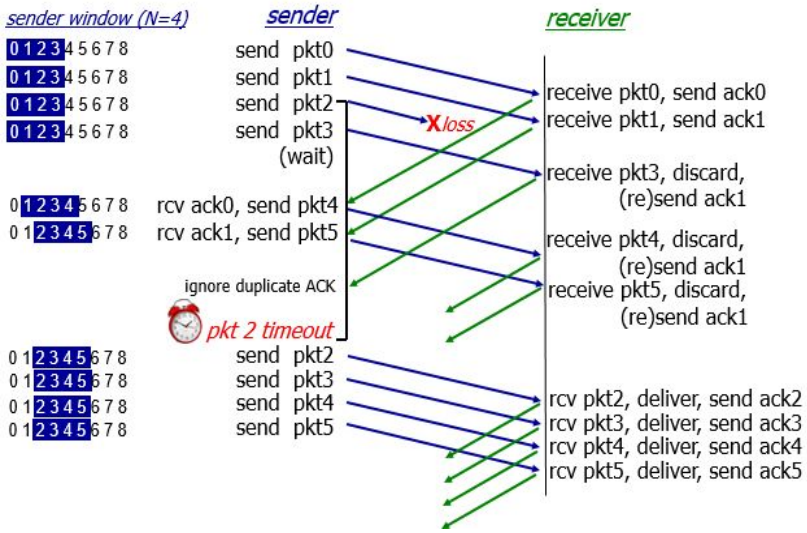
k-bit seq # in pkt header

“window” of up to N, consecutive unacked packets allowed



The scenario:

Sender can have up to N unacked packets in pipeline.
Receiver only sends cumulative ack (doesn’t ack is there’s a gap)
Sender has timer for oldest unacked packet, when it expires all unacked packets are retransmitted.



➔How the protocol is implemented:

In the client side:

1,2,3 are the same as in Stop and wait

4) Get file function will loop till the packets needed achieved, it receive the packet and check the sequence number with the expected, if not equal or there's an error the packet with the sequence number before is resent and wait till it's received correctly.

In the server side:

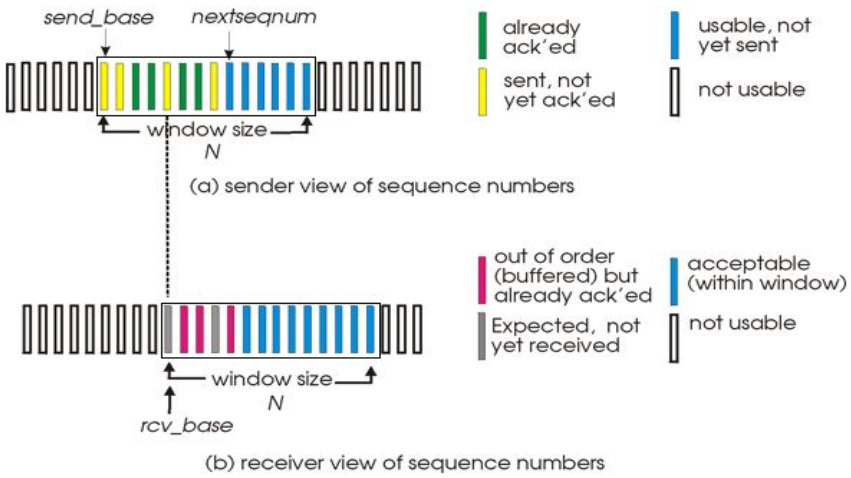
1,2,3,4,5,6 are the same as before.

7) The function send file loops till packets needed equal sequence number if pipelined not full, send packets till it's full, else wait for ack when an ack is received break to send next packet.

8) When ack is received it checks if it's equal to last ack +1 if equal break and send as a new place is empty in pipeline, else resend the packet.

9)If all packets are sent, wait to receive all unreceived acks.

3) Selective Repeat:



Sender:

if next available seq # in window, send pkt.

If timeout happen, resend pkt n, restart timer.

ACK(n) in [sendbase,sendbase+N], mark pkt n as received.

if n smallest unACKed pkt, advance window base to next unACKed seq #.

Receiver:

If pkt n in [rcvbase, rcvbase+N-1], send ACK(n)

If out-of-order: buffer

If in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N,rcvbase-1], ACK(n)

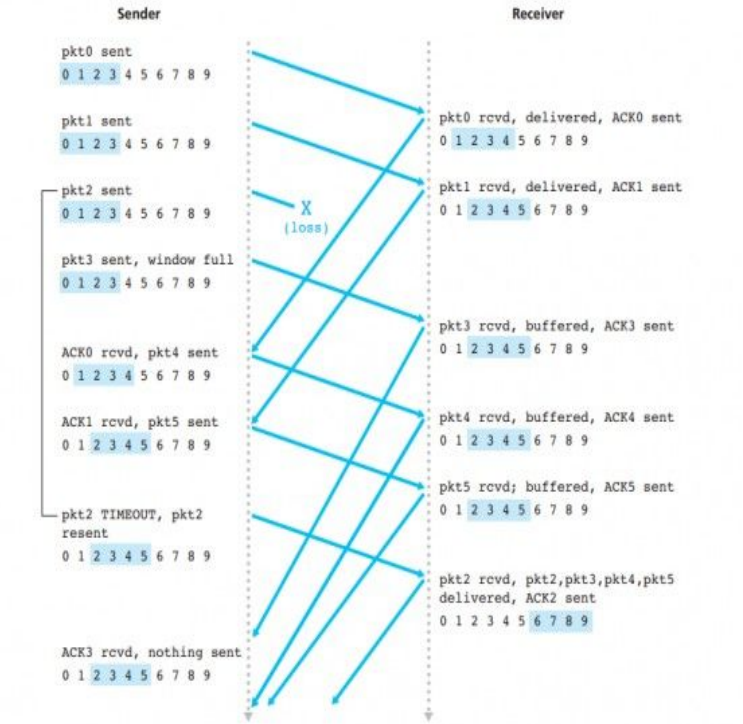


Figure 3.26 ♦ SR operation

➔ How the protocol is implemented:

In the client side:

1,2,3 are the same

4)The get file function initialize the array list that will contain all packets waiting for.
5) The window base is set to -1 and loops till it comes to packets needed -1.
6) When receiving a packet it return the new window base.
7) If sequence number of packet > window base and < window base +window size and the checksum equal to the one calculated then receive the packet.
8)An ack is sent to server.
9)If sequence number= window base, shift the base, and if equal to packets needed break as it's the last one.

In the server side:

1,2,3,4,5,6 are the same.
7)The send file check if pipeline is not full it sends packets and add it to the array list
8) A timer is set for each packet, when it timeouts an interrupt happens and resend the packet again.
9)If pipeline is full, waits for an ack, if ack is the base of window then shift the window.
10)After sending all, waits for the last acks that are still not received.

➢Instructions to run the code:

1) Run the ServerRun class
2) Run the ClientRun class
3) Enter a username and password from this list:

Username	**Password**
fatma	12
Monica	2
Mohamed	7
ahmed	49
maha	34

4)Put server.in and client.in file in the project folder as follows:

Server.in.txt

```
9876
50
5
0.5
```

Client.in.txt

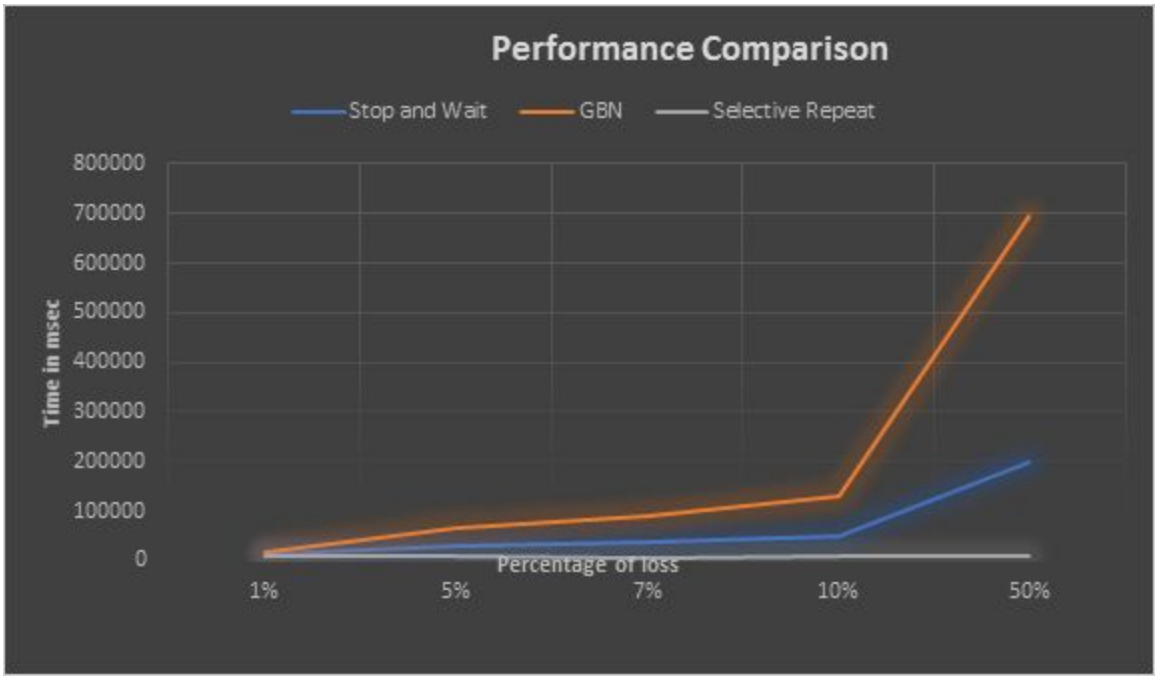
```
localhost
9876
51459
4.ppt
50
```

5)Choose the file of client to read from it the information.
6)When the run finish it asks for new name to save with it the file.

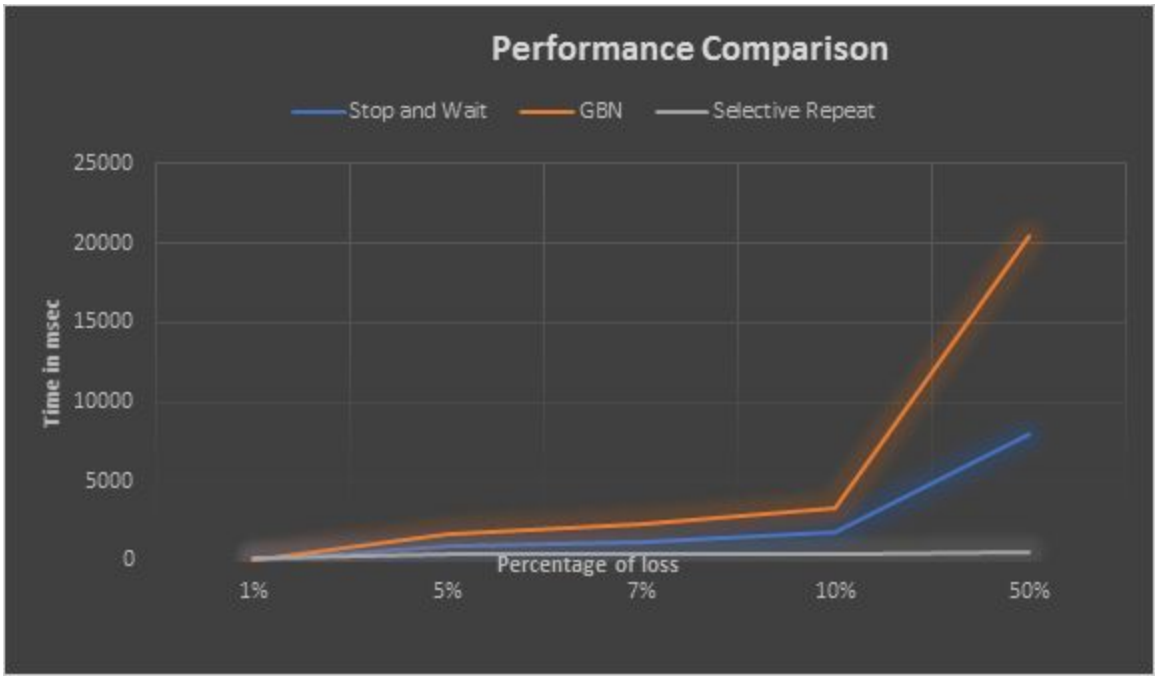
✓ Compares and Analysis:

Sending same file with different probability of loss:

★ File Size = 3,553,792 bytes , Packet Size = 500,
Packets = 7108□, Wait Time = 200ms



★ File Size = 38,253 bytes, Packet Size = 500,
Packets = 77, Wait Time = 200ms



Conclusion: Selective Repeat is the Fastest.