* Elementary data link ~~co~~ protocols — Unrestricted simplex protocol
protocol-1

Assumptions.
1. data are transmitted ~~in~~ one direction only half/full duplex.
2. Both transmitting and receiving network layers are always ready.
3. processing time is ignored. (can process infinitely quickly)
4. Infinite buffer space is available.
5. Communication channel is error free.

most unrealistic protocol.

(5) ~~cases~~

The protocol consists of two distinct procedures, a sender and a receiver. The sender run on data link layer on source m/c and receiver run on data link layer on destination m/c. No sequence number and ack are used here.

The sender process ~~a~~ runs an infinite loop to pump data out onto the line as fast as ~~possible~~ it can. Thus, it fetch the packet from the network layer, construct the frame and then send the frame on its way.

The receiver will wait for an event to occur (the only possibility is arrival of frame). When the frame arrives, it removes the newly arrived frame from the buffer and ~~puts~~ send (after removing the header from the frame) it to the network layer. ~~Th~~ then it waits for the next frame to come.

A simplex stop-and-wait protocol — (protocol-2)

1. The buffer space is not infinite,
2. processing time is not ignored.
3. Data traffic is still simplex (only one direction)
4. Channel is still error free.

First 2 assumption imply that flow control is required. But the error control is still not required.

Suppose, the receiver requires Δt time to get the frame from the physical layer and to send it to the network layer. Then sender then must transmit at an average rate less that ~~stops~~ one frame per Δt time. How to know Δt? Interval between frame arrival and its being processed may vary considerably. If network designer can

Computes the worst-case behavior of the receiver, they can program the sender to transmit slowly that even if every frame suffers the maximum delay, there will be no overruns. This is too conservative approach and highly inefficient. (Rate-based flow control).

The other approach is to have the receiver provide feedback to the sender (feedback based flow control). After having passed ~~#~~ a packet to its network layer, the receiver sends a little dummy frame (ack) back to the sender. After having sent a frame, the sender wait to get the ack from the receiver. After getting the ack, it sends the next frame. This ~~approach~~ protocol is known as stop-and-wait.

Although the data traffic is simplex (sender to receiver), the ack is comming from the other direction (receiver to sender). Then the communication channel needs to be capable of bidirectional data transfer. There two transfers not not happenings simultaneously, so half-duplex is sufficient.

A simplex protocol for noisy channel —

1. Channels make errors — frame may be damaged or lost completely. But it assumes that receiver can detect it. If receiver can not detect, the protocol fails. It is still simplex.

One way — Just add a timer in protocol 2. The sender will send a ~~packet~~ frame but the receiver will ack only if it receives correctly. A damaged frame arrives at the receiver will be discarded and no ack will be

9) sent to the sender. After a while, the sender would time out and send the frame again. This process will be repeated until the frame finally arrives correctly. what is the problem?

If ack get lost? But the frame actually reached the receiver correctly! the sender will resend the correct frame again, lot of duplicate frame!

1. A send a packet to B. The packet is correctly received by B and B sends an ack to A.
2. The ack get lost (the channel who can lost data frame can also lost ack frame).
3. After timeout, A sends the packet again.

what is the requirement to deal this —?
The receiver must be able to distinguish a frame whether it is first time or retransmission.
(the sequence number).

The receiver now can check the sequence number of each arriving frame to check if it is a new frame or a duplicate frame. Duplicate will be discarded.

How is the minimum number of bits needed for sequence number? - Only ambiguity is between frame m and m+1. Depending on whether the ack frame lost or not, the sender may try to send m or m+1. The sender is sending frame m+2 means ~~m+1 is received~~ ack of m+1 received correctly and this intern imply frame m has been correctly send and ack also received correctly and so on. Thus 1 bit is sufficient and the sequence number will be incremented by one in modulo 2 (0 becomes 1 and 1 becomes 0).

(8) this protocol in which sender wait for positive ack before advancing to the ~~next~~ frame, it is known as PAR (positive ack with retransmission) or ARQ (atomatic repeat request.

Sliding window protocol

Data will be transmitted in both direction simultaneously (full-duplex). In this protocol, data and ack will be intermixed and send in ~~the same~~ both direction.

sender	receiver	
data →	← data	Seperate circuit for data
← ack	ack →	" " ack

~~Although~~ Interleaving data and control frames on the same circuit is an improvement over having two separate circuit. In this case, a kind field in the header will tell whether it is data frame or ack frame. Another improvement might be - when a data frame arrives, instead of immediately sending a separate ack frame, the receiver waits until the network layer passes next ~~frame~~ packet to be sent to the sender. The ack will then be attached to the outgoing data frame using ack field in the header. In effect, the ack gets a free ride on the next outgoing data frame. The technique of temporarily delaying the ack so that it can be hooked onto next outgoing data frame is known as piggybacking.

(9) Advantage is clear — better utilization of bandwidth. A seperate ack frame needs a header, ack, checksum etc. But with piggyback it can be sent at the cost of only one bit ack-field.

The problem? - How long should the data link layer wait for a packet onto which to piggyback the ack? If ~~data~~ sender has to wait longer than timeout the frame will be retransmitted !! The whole purpose is gone. If the data link layer can foretell the future, it would know when the next packet will be going, and can accordingly decide either to wait for piggyback or to send as seperate ack frame. Of course, the data link layer cannot foretell the future, so it must take some adhoc scheme such as waiting for a fixed amount of time with the intention of piggyback and after that send as separate ack frame.

We will now discuss three different ~~proto~~ bidirectional protocols that belong to a class of sliding window protocols. All of these protocols uses sequence number from 0 to 2^n-1 so the sequence number fits exactly in an n-bit field. The three differ among themselves in terms of efficiency, complexity and buffer requirements. When n = 1, ~~...~~ the protocols is commonly known as stop-and-wait sliding window. The earlier stop-and-wait protocol that we have discussed was simplex. this time it is ~~a~~ full-duplex.

(10) The basic Concept of Sliding Window protocol in its general sense is that at any instant of time, the Sender maintains a set of Sequence numbers Corresponding to frames it is permitted to send. These frames are said to fall within the Sending window. Similarly, the receiver also maintains a receiving window corresponding to the set of frames it is permitted to accept/receive. Note that Sender's window and receiver's window need not have the same lower and upper limits or even have the same size.

Although these protocols give the data link layer the freedom about the order of frames it may send or receive, it must be noted that packets must be deliver to the destination network layer in the Same order they were passed to the sender's data link layer. Thus, sort of reshuffling may be required to perform at the receiver if they are not sent by the sender in proper order. When n=1, data link layer accepts frames in order, but n>1 it is not necessarily so. the Sequence numbers within Sender's window represent frames that have been sent or can be sent but are not yet acknowledged. When a new packet arrives from the network layer, it is given the next highest sequence number and the upper edge is advanced by one. When an acknowledgement comes in, the lower edge is advanced by one. In this way the window continuously maintains a list of unacknowledged frames. Since frames currently within Sender's window may be lost/damaged, the Sender must keep them in memory for a possible retransmission. Thus, if the Sender's window size is n, it requires n buffer space to hold the unacknowledged frames. If window ever grows to its maximum size, the Sender data link layer forcibly shut off the network layer untill another buffer space becomes free.

(11) The receive data link layer's window corresponds to the frames it may accept. Any frame falling outside its window is discarded. When a frame whose sequence number is equal to the lower edge of the window is received, it is passed to the network layer and an acknowledgement is generated and the window is rotated by one. Unlike the sender's window, the receiver's window always remains at its initial size.

1. One-bit sliding window (Stop-and-wait sliding window):- In this case, the sender transmits a frame and waits for its acknowledgement before sending the next one. Under normal circumstances, one of two data link layers goes first and transmit the first frame. Thus, the starting m/c fetches the first packet from its network layer, builds a frame from it, and send it. When this frame arrives, the receiving data link layer checks to see if it is a duplicate or new frame. If the frame is the new one, it is passed to the network layer, otherwise it is discarded. The ack field contains the number of the last frame received without error. If this number agrees with the sequence number of the frame ~~stored~~ the sender is trying to send, the sender knows it is done with the frame stored in the buffer and can fetch the next packet from the network layer. If the sequence number disagrees, it must continue trying to send the same frame. It is assumed that whenever a frame is received there is a frame to send back. That means, piggybacking can be used to send the ack.

— page 214-215-216.

2. Go Back to N :- A maximum of max-seq frames and not (max-seq +1) frames may be outstanding at any instant, even though there are (max-seq +1) distinct sequence number.

(12) Why? * Sender sends 0-7 frames
* A piggybacked ack for frame 7 comes back to the sender
* The sender sends another eight (0-7) frames.
* Now another piggyback ack for frame 7 comes in.

Question: did all eight frames of second batch arrive successfully or all eight lost? In both cases the receiver would be sending frame 7 as the ack. For this reason, max number of outstanding frames may be restricted to (maxseq).

- when an ack for frame n comes, frames n-1, n-2, --etc are automatically acked.

- we assume that there is always reverse traffic on which ~~to~~ piggyback ack. If not, no ack can be sent.

- Logically it needs multiple timer, one per outstanding frame since there are multiple outstanding frames. Timeout for each frame may be independent of other.

Selective Repeat — accepts frames out of order but passes them in the network layer in order. Each frame has a timer, when that timer expires, only that frame is retransmitted, not all outstanding frames. Non-sequential receive introduces the following problem —

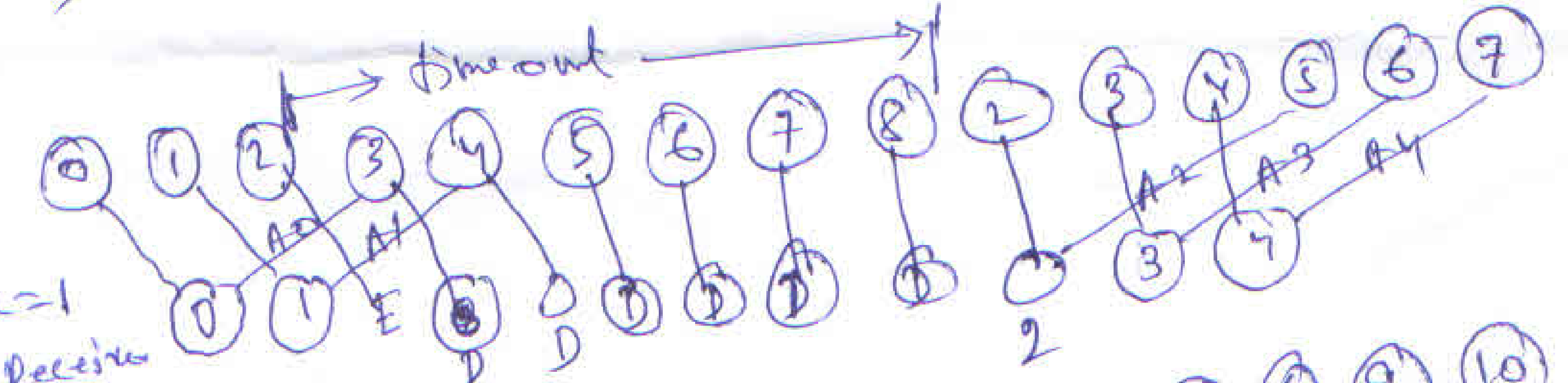
* 3 bit sequence number, so 0-~~6~~ frames are sent and then wait for ack.
* All seven frames arrive correctly, so the receiver acked and advanced its window to 7 0 1 2 - - 5.
* If that ack lost, the sender will send 0 again after
all acks are lost   time out and

③ checking whether it falls within the receiver's window. Since, ~~it~~ frame 0 is within the new window, so it will be accepted. ~~The receiver sends a piggybacked ack for frame 6 since frames 0-6 have been received so far. The sender understand that all 0-6 frames reach correctly. So it advances its window to 7, 0, 1, 2, 3, 4, 5. frame 7 will be accepted by the receiver and it will be passed to the network layer. Then it will receive frame 0. But receiver has a frame 0 already and thus it will send frame 0 this wrong frame 0 to the network layer~~

This frame 0 is actually original ~~frame 0~~ but receiver will think it is new frame (frame 8)?? The reason is there is overlapp between sending and receiving windows. That is why receiver unable to distinguish the duplicate frame 0 with frame 8 (in mind this is also frame 0). To avoid the overlapp, max size will be $2^n/2 = 2^{n-1}$.

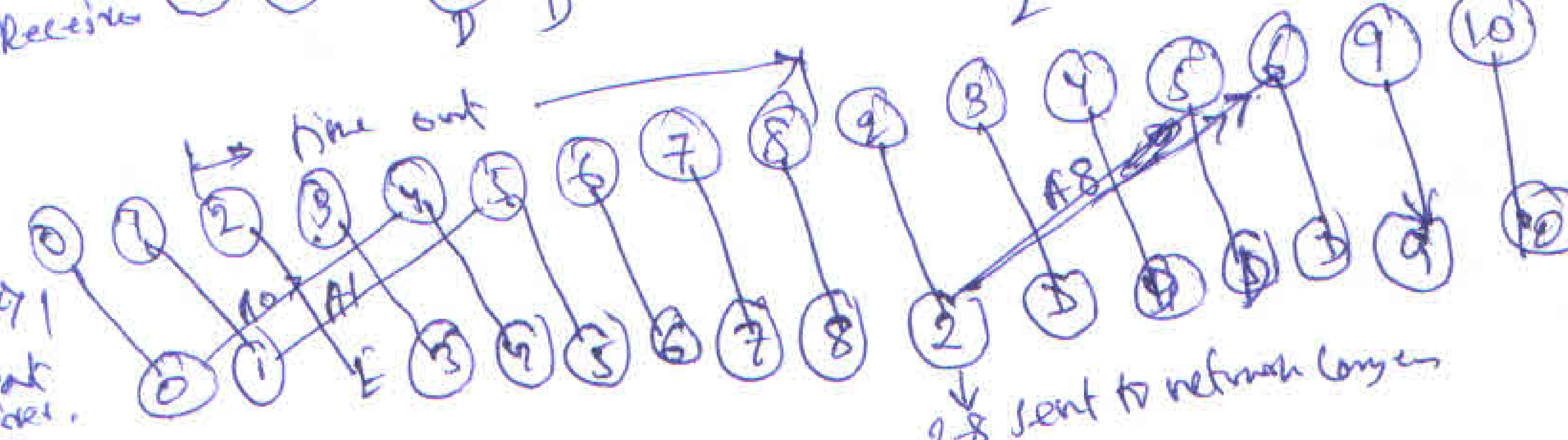
Go back-to-N

No out of order. Receiver window size = 1 no buffer required at receiver

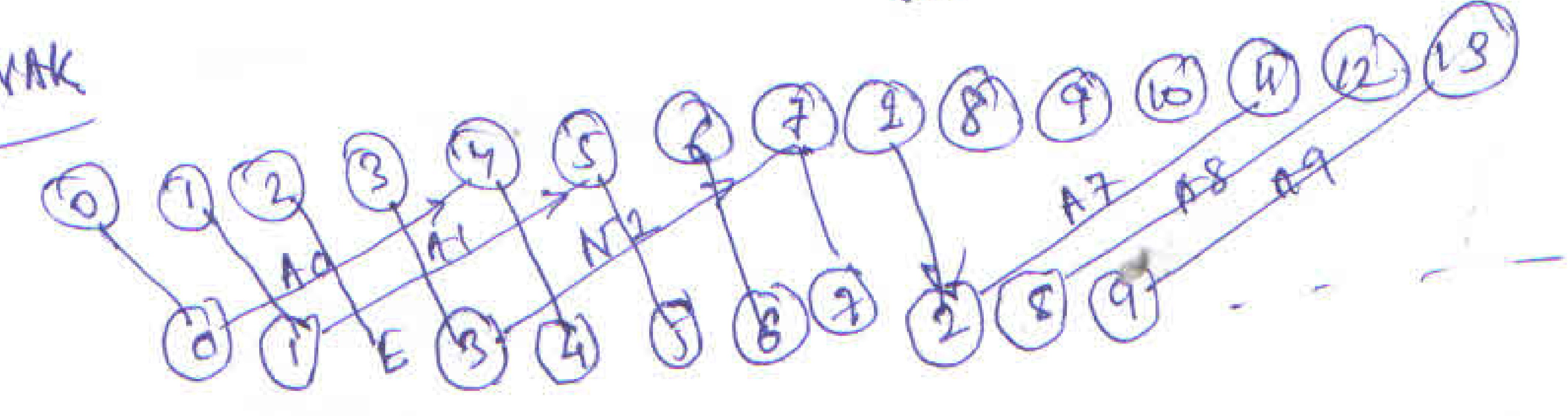


Selective Repeat

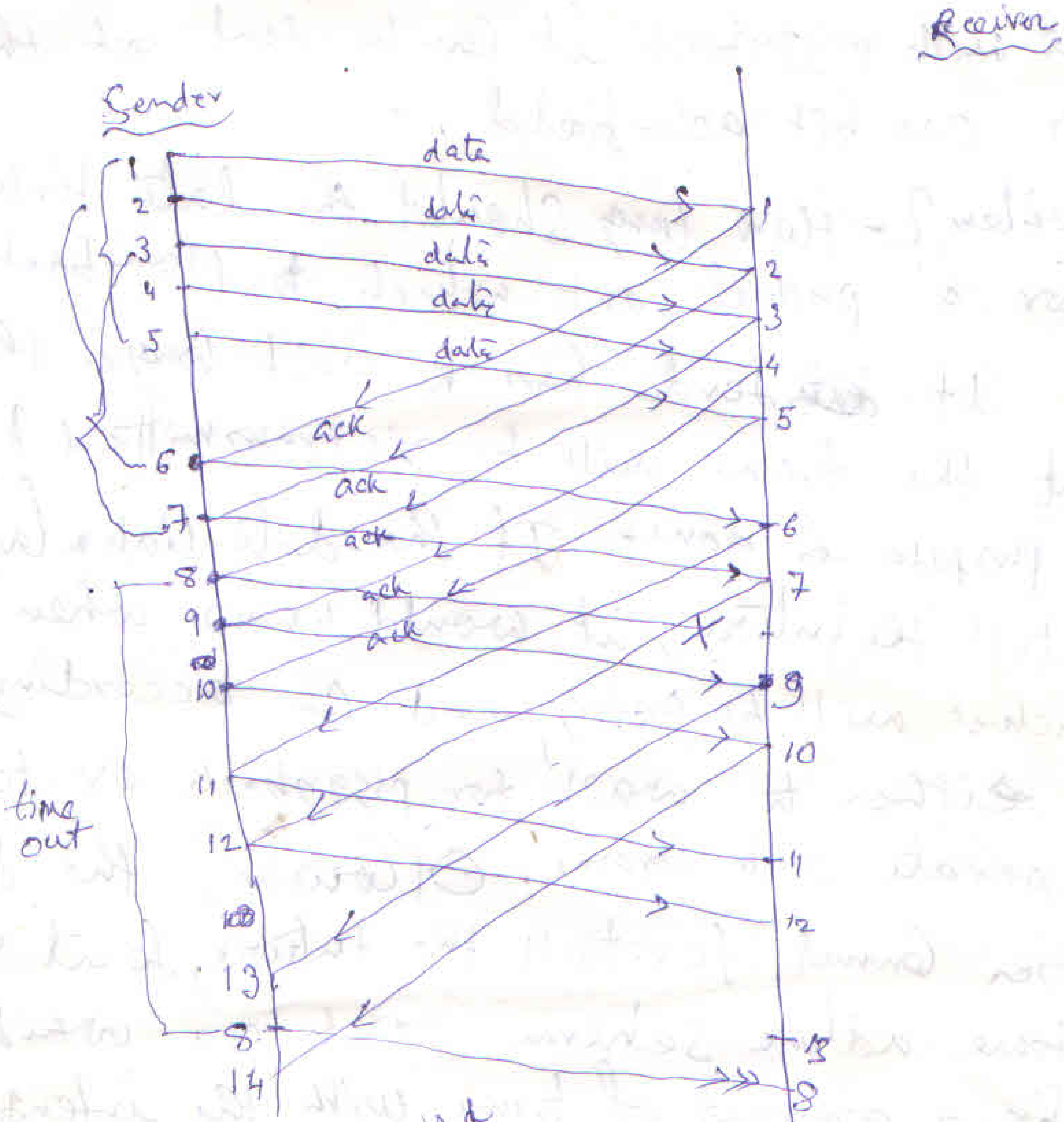
Out of order. Receiver window size > 1 buffer required at receiver.



Selective repeat with NAK



Setting Window Size



roundtrip & window size $n=5$

Setting n: window size: Let RTT be the mean delay between sending a packet and getting the ack. Let B packet/sec is the rate of the bottleneck link between sender and receiver. So the processing rate can not exceed the bit rate B of the slowest link between sender and receiver. then $n = B \times RTT$ will ensure that the protocol comes close to achieve a throughput equal to the available bit rate when no packet/ack losses are involved. the quantity $(B.RTT)$ is called the bandwidth-delay product and is crucial in determining performance of the sliding-window protocol. If $n < B.RTT$, the queue at the link is empty and does not cause any queueing delay. However RTT includes propagation, transmission and processing delays. When $n > B.RTT$, the RTT experienced by the connection includes the queueing delay as well. Hence, setting $n = B.RTT$ will provide maximum possible throughput in the absence of data/ack losses. When packet loss occurs, window size n needs to higher to get maximum throughput.

Throughput of sliding window protocol:

Let λ be probability that a data packet or ack is lost and packet losses are i.i.d. The number of transmission required by any packet before its ack is received. With probability $(1-\lambda)$, we need one transmission, with prob $\lambda(1-\lambda)$ we need two transmission and so on. Thus, expected number of transmission required is $(1-\lambda).1+2\lambda(1-\lambda)+3\lambda^2(1-\lambda)+\dots = \frac{1}{1-\lambda}$. Hence on expected case, we need $\frac{1}{1-\lambda}$ transmission to send one packet and get it acked. Thus, the throughput is $\frac{1}{\frac{1}{1-\lambda}} = 1-\lambda$.

Timeout interval determination: If time out is large the packet continue to flow as long as acks are arriving (no losses). However, as packet (ack) are lost the effective window size is falling and eventually the protocol will stall until the sender retransmits. Hence, longer the time out, bigger the stalls experienced. Moreover, larger the time out, the receiver's buffer has to be bigger as packets needs to be delevered in order. Hence the tradeoff.