

# **Lectures 3 & 4**

**6.263/16.37**

## **The Data Link Layer: ARQ Protocols**

**Eytan Modiano**

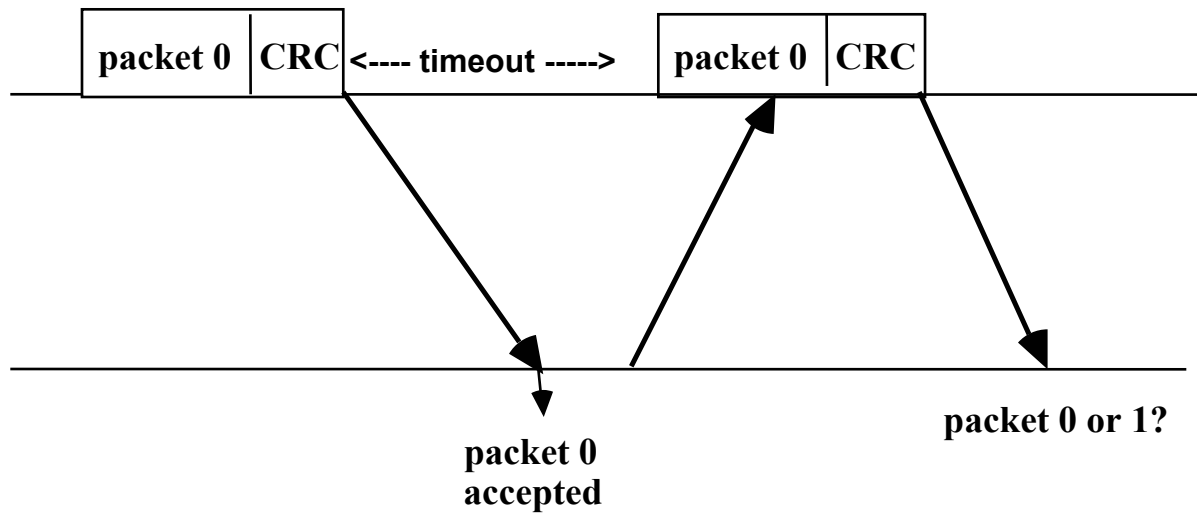
**MIT, LIDS**

# Automatic Repeat ReQuest (ARQ)

- When the receiver detects errors in a packet, how does it let the transmitter know to re-send the corresponding packet?
- Systems which automatically request the retransmission of missing packets or packets with errors are called ARQ systems.
- Three common schemes
  - Stop & Wait
  - Go Back N
  - Selective Repeat

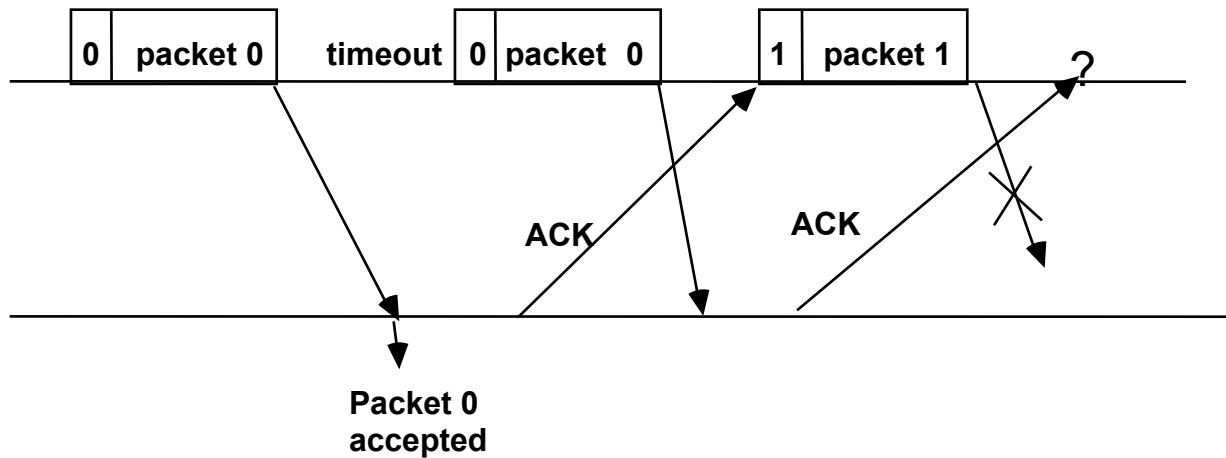


# The Use Of Timeouts For Lost Packets Requires Sequence Numbers



- Problem: Unless packets are numbered the receiver cannot tell which packet it received
- Solution: Use packet numbers (sequence numbers)

# Request Numbers Are Required On ACKs To Distinguish Packet ACKed



- **REQUEST NUMBERS:**

- Instead of sending "ack" or "nak", the receiver sends the number of the packet currently awaited.
- Sequence numbers and request numbers can be sent modulo 2.

This works correctly assuming that

- 1) Frames travel in order (FCFS) on links
- 2) The CRC never fails to detect errors
- 3) The system is correctly initialized.

# Stop and Wait Protocol Algorithm at sender (node A)

**(with initial condition SN=0)**

- 1) Accept packet from higher layer when available;  
assign number SN to it
- 2) Transmit packet SN in frame with sequence # SN
- 3) Wait for an error free frame from B
  - i. if received and it contains  $RN > SN$  in the  
request # field, set SN to RN and go to 1
  - ii. if not received within given time, go to 2

# Stop and Wait

## Algorithm at receiver (node B)

**(with initial condition  $RN=0$ )**

- 1) Whenever an error-free frame is received from A with a sequence # equal to RN, release received packet to higher layer and increment RN.
- 2) At arbitrary times, but within bounded delay after receiving any error free frame from A, transmit a frame to A containing RN in the request # field.

# Correctness of stop & wait with integer SN, RN

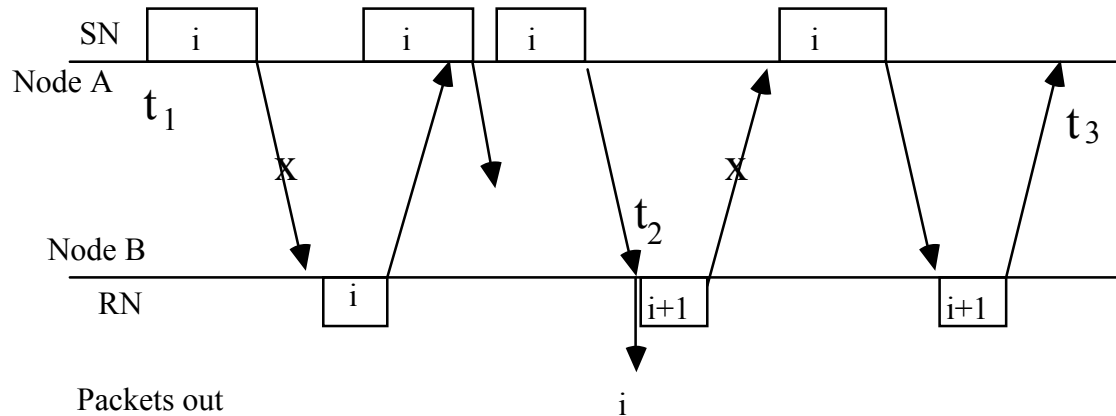
- Assume, for A to (from) B transmission, that
  - All errors are detected as errors
  - Initially no frames are on link, SN=0, RN=0
  - Frames may be arbitrarily delayed or lost
  - Each frame is correctly received with at least some probability  $q > 0$ .
- Split proof of correctness into two parts:
  - SAFETY: show that no packet is ever released out of order or more than once
  - LIVENESS: show that every packet is eventually released



# Safety

- No frames on link initially, packet 0 is first packet accepted at A, it is the only packet assigned  $SN=0$ , and must be the packet released by B if B ever releases a packet
- Subsequently (using induction) if B has released packets up to and including  $n-1$ , then  $RN$  is updated to  $n$  when  $n-1$  is released, and only  $n$  can be released next

# LIVENESS



$t_1$  = time at which A first starts to transmit packet i

$t_2$  = time at which B correctly receives & releases i, and increases RN to i+1

$t_3$  = time at which SN is increased to i+1

Will prove that  $t_1 < t_2 < t_3 < \infty$ .  $\Rightarrow$  Liveness

# Liveness Argument

- Let  $SN(t)$ ,  $RN(t)$  be values of SN and RN at time  $t$

**From the algorithm,**

(1)  $SN(t)$  and  $RN(t)$  are increasing in  $t$  and  $SN(t) \leq RN(t)$  for all  $t$

(2) From safety (since  $i$  has not been sent before  $t_1$ )  $RN(t_1) \leq i$  and  $SN(t_1) = i$

- From (1) and (2),  $RN(t_1) = SN(t_1) = i$
- RN is incremented at  $t_2$  and SN at  $t_3$ , so  $t_2 < t_3$
- A transmits  $i$  repeatedly up to  $t_3$ , and thus to  $t_2$  when it is correctly received. Since  $q > 0$ ,  $t_2$  is finite
- B transmits  $RN=i+1$  repeatedly until correctly received at  $t_3$ , and  $q > 0$  implies that  $t_3$  is finite.

# Correctness of Stop & Wait with binary (finite) SN, RN

- Assume that frames travel on link in order

Note that with integer SN, RN, either

$$\text{SN}=\text{RN} \quad (\text{from } t_1 \text{ to } t_2) \quad \text{or} \quad (3)$$

$$\text{SN}=\text{RN}-1 \quad (\text{from } t_2 \text{ to } t_3) \quad (4)$$

Since frames travel in order, the sequence numbers arriving at B and the request numbers arriving at A are increasing, so a single bit can resolve the ambiguity between (3) and (4)

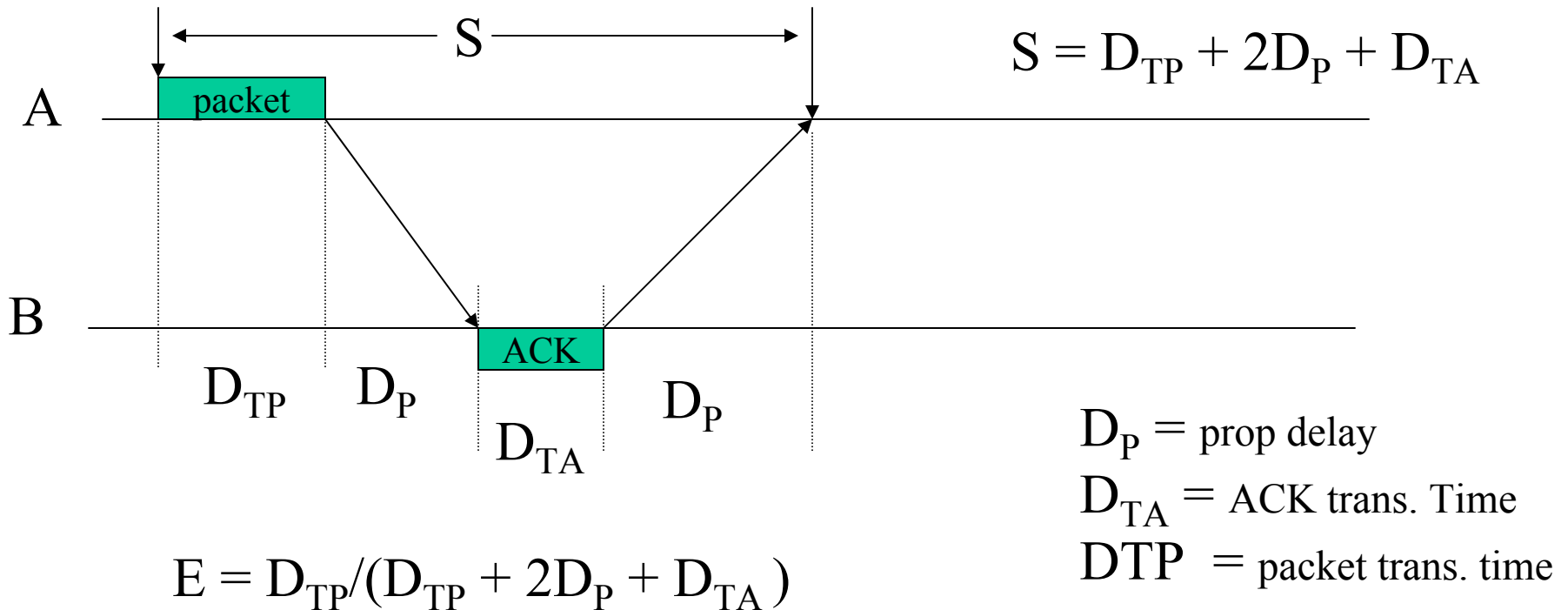
- RN = 0 and SN = 1 or RN = 1 and SN = 0  
=> received packet is an old packet
- RN = 0 and SN = 0 or RN = 1 and SN = 1  
=> received packet is new

# Efficiency of stop and wait

Let  $S$  = total time between the transmission of a packet and reception of its ACK

$D_{TP}$  = transmission time of the packet

Efficiency (no errors) =  $D_{TP}/S$



# Stop and wait in the presence of errors

Let  $P$  = the probability of an error in the transmission of a packet or in its acknowledgment

$$S = D_{TP} + 2D_P + D_{TA}$$

TO = the timeout interval

$X$  = the amount of time that it takes to transmit a packet and receive its ACK. This time accounts for retransmissions due to errors

$$E[X] = S + TO \cdot P / (1 - P),$$

$$\text{Efficiency} = D_{TP} / E[X]$$

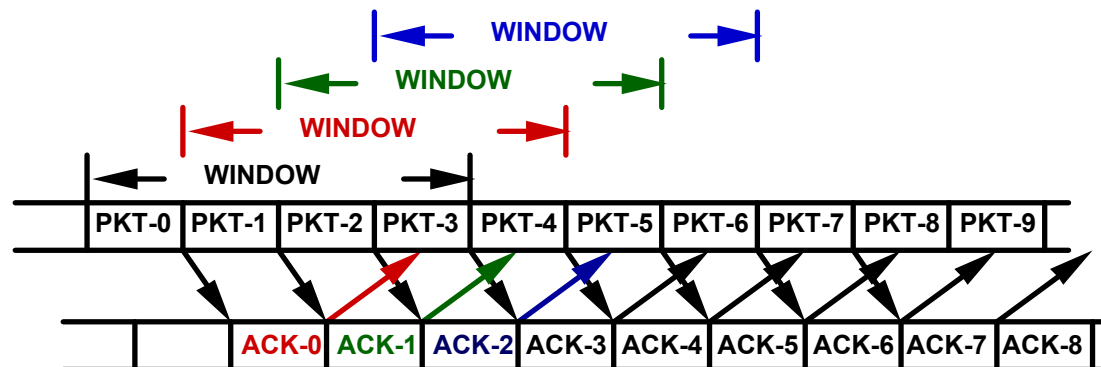
Where,

TO =  $D_{TP}$  in a full duplex system

TO =  $S$  in a half duplex system

# Go Back N ARQ (Sliding Window)

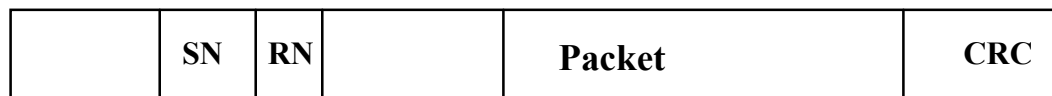
- Stop and Wait is inefficient when propagation delay is larger than the packet transmission time
  - Can only send one packet per round-trip time
- Go Back N allows the transmission of new packets before earlier ones are acknowledged
- Go back N uses a window mechanism where the sender can send packets that are within a “window” (range) of packets
  - The window advances as acknowledgements for earlier packets are received



# Features of Go Back N

- Window size = N
  - Sender cannot send packet  $i+N$  until it has received the ACK for packet  $i$
- Receiver operates just like in Stop and Wait
  - Receive packets in order
  - Receiver cannot accept packet out of sequence
  - Send  $RN = i + 1 \Rightarrow$  ACK for all packets up to and including  $i$
- Use of piggybacking
  - When traffic is bi-directional RN's are piggybacked on packets going in the other direction
    - Each packet contains a SN field indicating that packet's sequence number and a RN field acknowledging packets in the other direction

<--Frame Header ----->





# Go Back N ARQ

- The transmitter has a "window" of N packets that can be sent without acknowledgements
- This window ranges from the last value of RN obtained from the receiver (denoted  $SN_{\min}$ ) to  $SN_{\min}+N-1$
- When the transmitter reaches the end of its window, or times out, it goes back and retransmits packet  $SN_{\min}$

Let  $SN_{\min}$  be the smallest number packet not yet ACKed

Let  $SN_{\max}$  be the number of the next packet to be accepted from the higher layer (i.e., the next new packet to be transmitted)

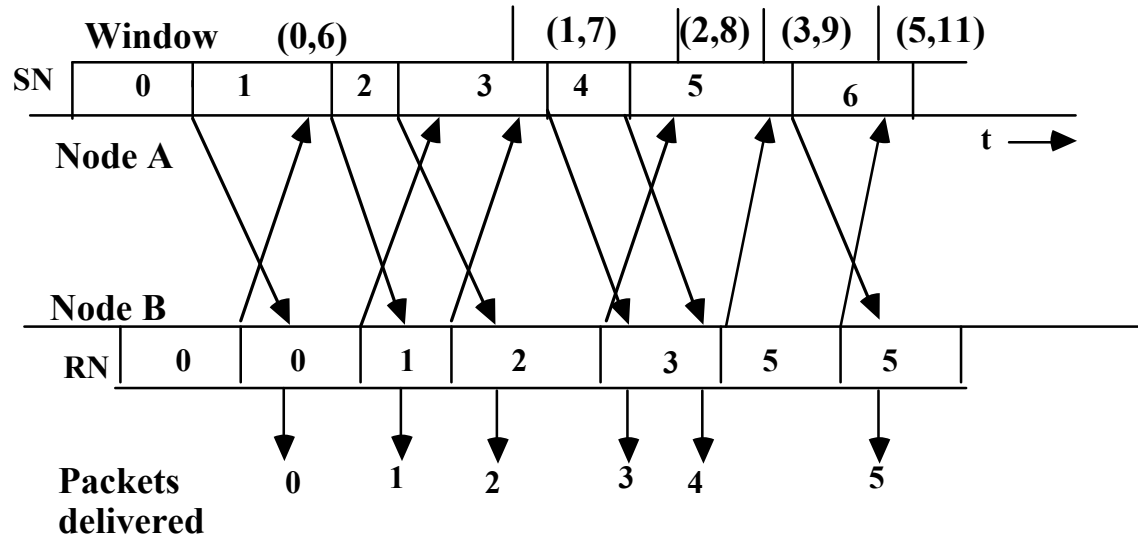
# Go Back N Sender Rules

- $SN_{min} = 0; SN_{max} = 0$
- Repeat
  - If  $SN_{max} < SN_{min} + N$  (entire window not yet sent)
    - Send packet  $SN_{max}$  ;
    - $SN_{max} = SN_{max} + 1$ ;
  - If packet arrives from receiver with  $RN > SN_{min}$ 
    - $SN_{min} = RN$ ;
  - If  $SN_{min} < SN_{max}$  (there are still some unacknowledged packets) and sender cannot send any new packets
    - Choose some packet between  $SN_{min}$  and  $SN_{max}$  and re-send it
- The last rule says that when you cannot send any new packets you should re-send an old (not yet ACKed) packet
  - There may be two reasons for not being able to send a new packet
    - Nothing new from higher layer
    - Window expired ( $SN_{max} = SN_{min} + N$ )
  - No set rule on which packet to re-send
    - Least recently sent

# Receiver Rules

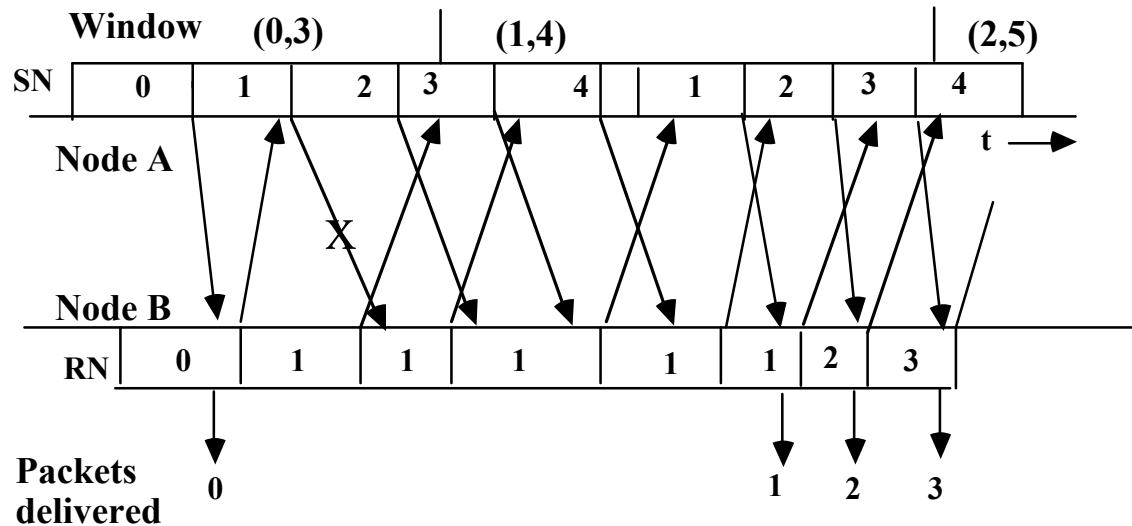
- $RN = 0$ ;
- Repeat
  - When a good packet arrives, if  $SN = RN$ 
    - Accept packet
    - Increment  $RN = RN + 1$
- At regular intervals send an ACK packet with  $RN$ 
  - Most DLCs send an ACK whenever they receive a packet from the other direction
    - Delayed ACK for piggybacking
- Receiver reject all packets with  $SN$  not equal  $RN$ 
  - However, those packets may still contain useful  $RN$  numbers (see homework assignment)

# Example of Go Back 7 ARQ



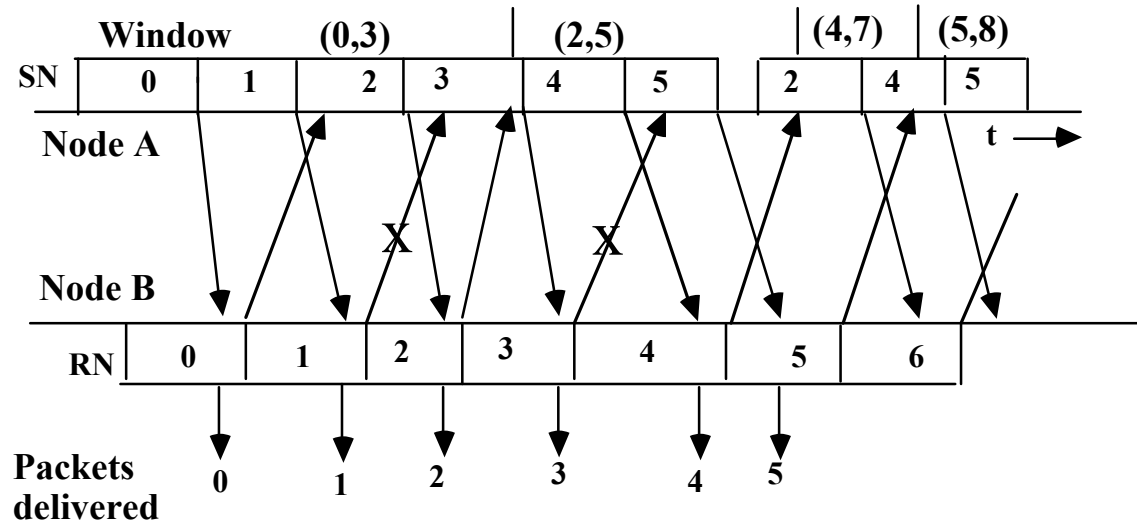
- Note that packet RN-1 must be accepted at B before a frame containing request RN can start transmission at B

# RETRANSMISSION BECAUSE OF ERRORS FOR GO BACK 4 ARQ



- Note that the timeout value here is take to be the time to send a full window of packets
- Note that entire window has to be retransmitted after an error

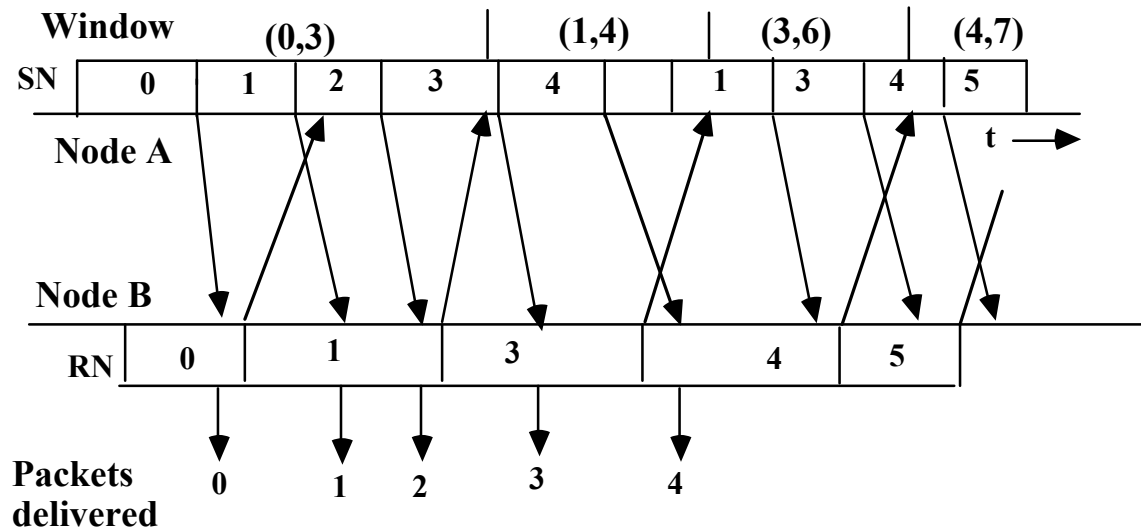
# RETRANSMISSION DUE TO FEEDBACK ERRORS FOR GO BACK 4 ARQ



- When an error occurs in the reverse direction the ACK may still arrive in time. This is the case here where the packet from B to A with RN=2 arrives in time to prevent retransmission of packet 0
- Packet 2 is retransmitted because RN = 4 did not arrive in time, however it did arrive in time to prevent retransmission of packet 3
  - Was retransmission of packet 4 and 5 really necessary?

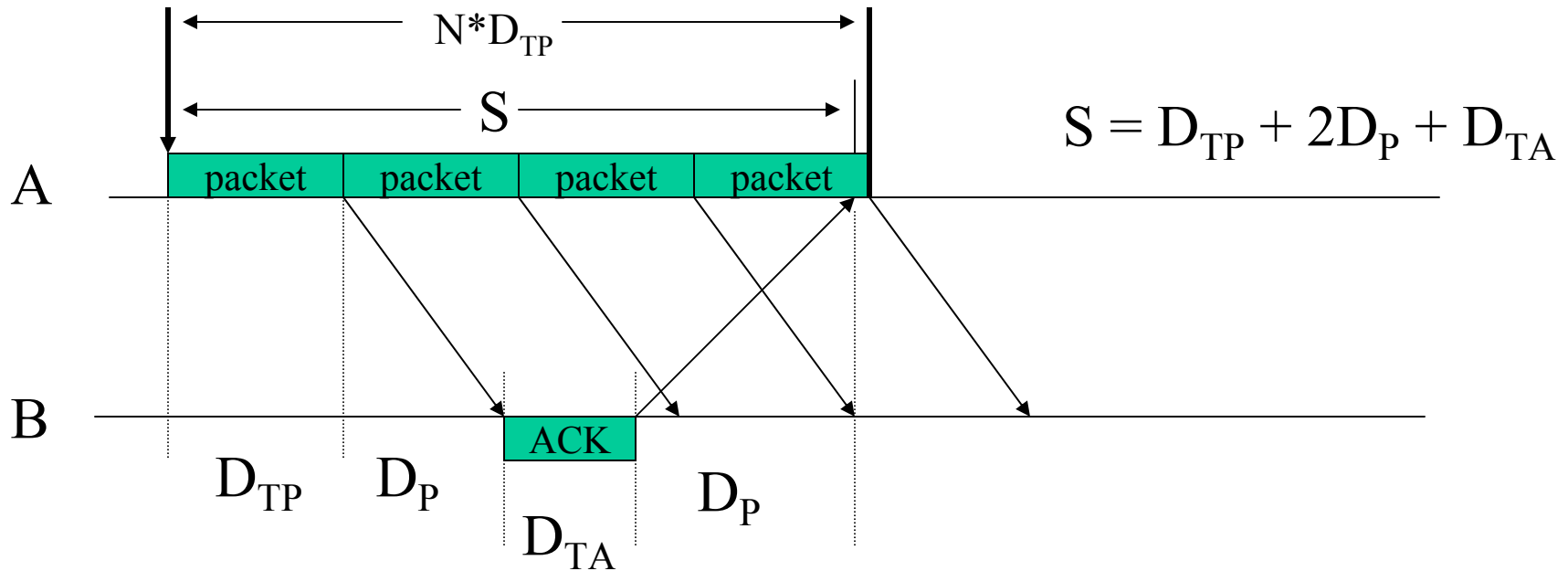
Strictly no because the window allows transmission of packets 6 and 7 before further retransmissions. However, this is implementation dependent

# EFFECT OF LONG FRAMES



- Long frames in feedback direction slow down the ACKs
  - This causes a transmitter with short frames to wait or go back
- Notice again that the retransmission of packets 3 and 4 was not strictly required because the sender could have sent new packets within the window
  - Again, this is implementation dependent

# Efficiency of Go Back N



- We want to choose  $N$  large enough to allow continuous transmission while waiting for an ACK for the first packet of the window,

$$N > S / D_{TP}$$

- Without errors the efficiency of Go Back N is,

$$E = \min\{1, N \cdot D_{TP} / S\}$$



# Efficiency of Go Back N with transmission errors

## Approximate analysis

Assume:  $N = \left\lceil \frac{S}{D_{TP}} \right\rceil$        $TO = N * D_{TP}$

- When an error occurs the entire window of N packets must be retransmitted

Let X = the number of packets sent per successful transmission

$$E[X] = 1*(1-P) + (X+N)*P$$

$$= 1 + N*P/(1-P)$$

$$\text{Efficiency} = 1/E[X]$$

# Go Back N Requirements

- Go Back N is guaranteed to work correctly, independent of the detailed choice of which packets to repeat, if
  - 1) System is correctly initialized
  - 2) No failures in detecting errors
  - 3) Packets travel in FCFS order
  - 4) Positive probability of correct reception
  - 5) Transmitter occasionally resends  $S_{n_{\min}}$  (e.g., upon timeout)
  - 6) Receiver occasionally sends RN

# Notes on Go Back N

- Requires no buffering of packets at the receiver
- Sender must buffer up to N packets while waiting for their ACK
- Sender must re-send entire window in the event of an error
- Packets can be numbered modulo M where  $M > N$ 
  - Because at most N packets can be sent simultaneously
- Receiver can only accept packets in order
  - Receiver must deliver packets in order to higher layer
  - Cannot accept packet  $i+1$  before packet  $i$
  - This removes the need for buffering
  - This introduces the need to re-send the entire window upon error
- The major problem with Go Back N is this need to re-send the entire window when an error occurs. This is due to the fact that the receiver can only accept packets in order

# Selective Repeat Protocol (SRP)

- Selective Repeat attempts to retransmit only those packets that are actually lost (due to errors)
  - Receiver must be able to accept packets out of order
  - Since receiver must release packets to higher layer in order, the receiver must be able to buffer some packets
- Retransmission requests
  - Implicit
    - The receiver acknowledges every good packet, packets that are not ACKed before a time-out are assumed lost or in error
    - Notice that this approach must be used to be sure that every packet is eventually received
  - Explicit
    - An explicit NAK (selective reject) can request retransmission of just one packet
    - This approach can expedite the retransmission but is not strictly needed
  - One or both approaches are used in practice

# SRP Rules

- Window protocol just like GO Back N
  - Window size  $W$
- Packets are numbered Mod  $M$  where  $M \geq 2W$
- Sender can transmit new packets as long as their number is with  $W$  of all un-ACKed packets
- Sender retransmit un-ACKed packets after a timeout
  - Or upon a NAK if NAK is employed
- Receiver ACKs all correct packets
- Receiver stores correct packets until they can be delivered in order to the higher layer

# Need for buffering

- Sender must buffer all packets until they are ACKed
  - Up to  $W$  un-ACKed packets are possible
- Receiver must buffer packets until they can be delivered in order
  - I.e., until all lower numbered packets have been received
  - Needed for orderly delivery of packets to the higher layer
  - Up to  $W$  packets may have to be buffered (in the event that the first packet of a window is lost)
- Implication of buffer size =  $W$ 
  - Number of un-ACKed packets at sender  $\leq W$   
Buffer limit at sender
  - Number of un-ACKed packets at receiver cannot differ by more than  $W$   
Buffer limit at the receiver (need to deliver packets in order)
  - Packets must be numbered modulo  $M \geq 2W$  (using  $\log_2(M)$  bits)

# EFFICIENCY

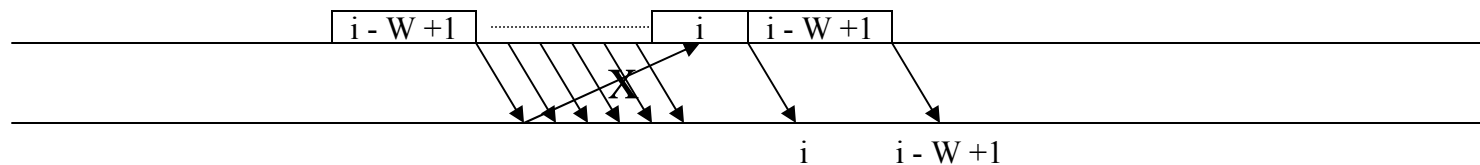
- For ideal SRP, only packets containing errors will be retransmitted
  - Ideal is not realistic because sometimes packets may have to be retransmitted because their window expired. However, if the window size is set to be much larger than the timeout value then this is unlikely
- With ideal SRP, efficiency =  $1 - P$ 
  - $P$  = probability of a packet error
- Notice the difference with Go Back N where

$$\text{efficiency (Go Back N)} = 1 / (1 + N * P / (1 - P))$$

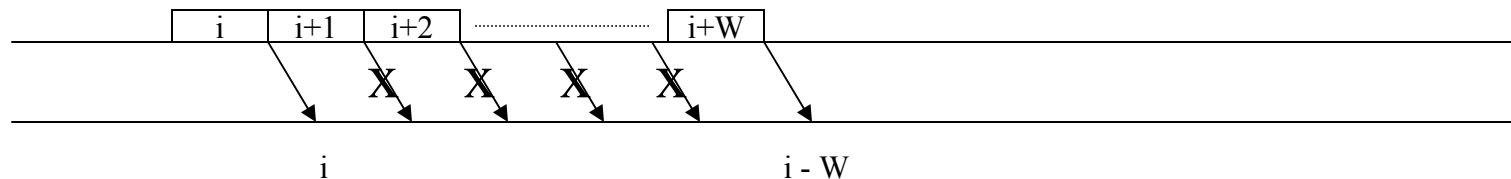
- When the window size is small performance is about the same, however with a large window SRP is much better
  - As transmission rates increase we need larger windows and hence the increased use of SRP

# Why are packets numbered Modulo $2W$ ?

- Lets consider the range of packets that may follow packet  $i$  at the receiver



Packet  $i$  may be followed by the first packet of the window ( $i - W + 1$ ) if it requires retransmission



Packet  $i$  may be followed by the last packet of the window ( $i + W$ ) if all Of the ACKs between  $i$  and  $i + W$  are lost

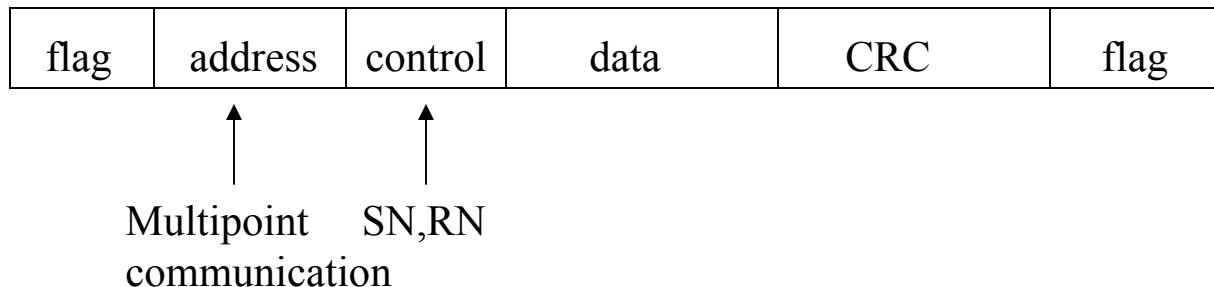
- Receiver must differentiate between packets  $i - W + 1 \dots i + W$ 
  - These  $2W$  packets can be differentiated using Mod  $2W$  numbering



# STANDARD DLC's

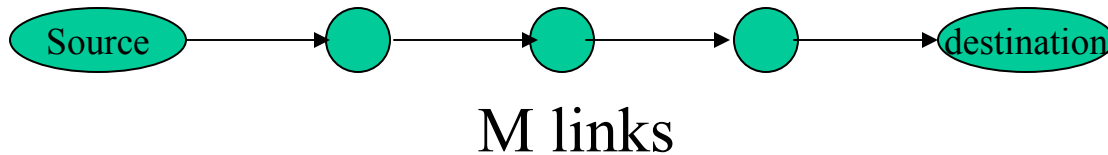
- HDLC, LAPB (X.25), and SDLC are almost the same
  - HDLC/ SDLC developed by IBM for IBM SNA networks
  - LAPB developed for X.25 networks
- They all use bit oriented framing with flag = 01111110
- They all use a 16-bit CRC for error detection
- They all use Go Back N ARQ with N = 7 or 127 (optional)

## SDLC packet



- Older protocols (used for modems, e.g., xmodem) used stop and wait and simple checksums

# Optimal packet size based on pipelining effect



- Packet must be completely received before being forwarded to next node
- Delay for sending N packets over M links (pipelining delay)

$$D = N \cdot D_{TP} + (M-1) \cdot D_{TP}$$

- Each packet contains K bits of data and a header of size H bits
  - CRC, flags, SN's, etc.
  - Total packet size K+H bits
- In order to transmit a message of L bits we need L/K packets
- Time to transmit message over M links,

$$R = \text{data rate} \quad D = \frac{L}{K} \left( \frac{K+H}{R} \right) + (M-1) \left( \frac{K+H}{R} \right)$$

## Optimal packet size

Transmission  
Delay

Pipelining  
delay

$$D = \frac{L}{K} \left( \frac{K+H}{R} \right) + (M-1) \left( \frac{K+H}{R} \right)$$

- Small packets reduce the pipelining delay but increase the transmission delay due to additional headers
- Large packets reduce header overhead but increase the pipelining delay

- Optimal packet size,  $K_{opt} = \sqrt{\frac{LH}{M-1}}$

- Approach may be appropriate for high-speed multi-hop networks
- Alternative approach may optimize the packet size to minimize link layer retransmissions due to errors
  - Large packet are more likely to contain transmission errors