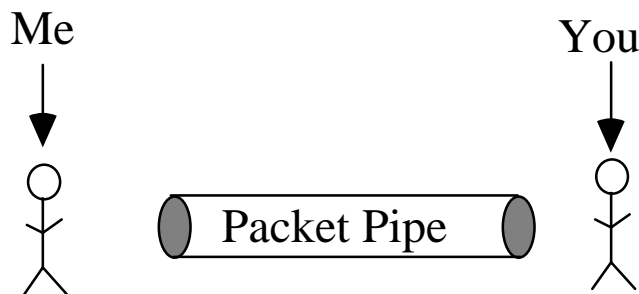


Lecture 5a - Automatic Repeat Request (BG 2.4)



Consider DLC & Frame Transmission
(Transport Layer can also use ARQ)

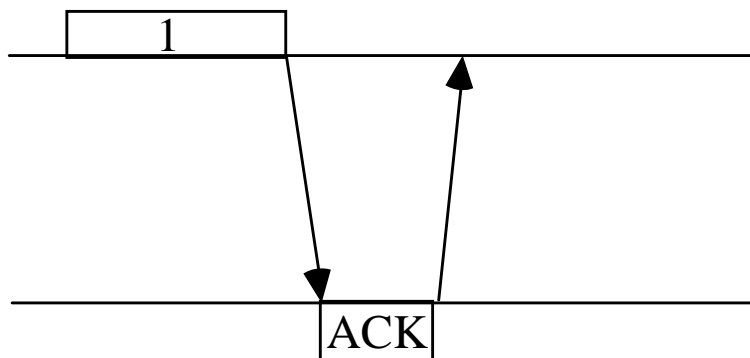
- Issue 1:
 - I send a packet
 - Packet may be:
 - lost (how do I know this?)
 - corrupted (you can ask me to resend it, but what if your request is corrupted or lost?)
 - duplicated, due to retransmission of an earlier, corrupted packet (how do I know this?)
 - Thus the DLC protocol must have rules to detect & recover from losses/corruption.
 - (Detecting corruption is via CRC in trailer - subject of another lecture)

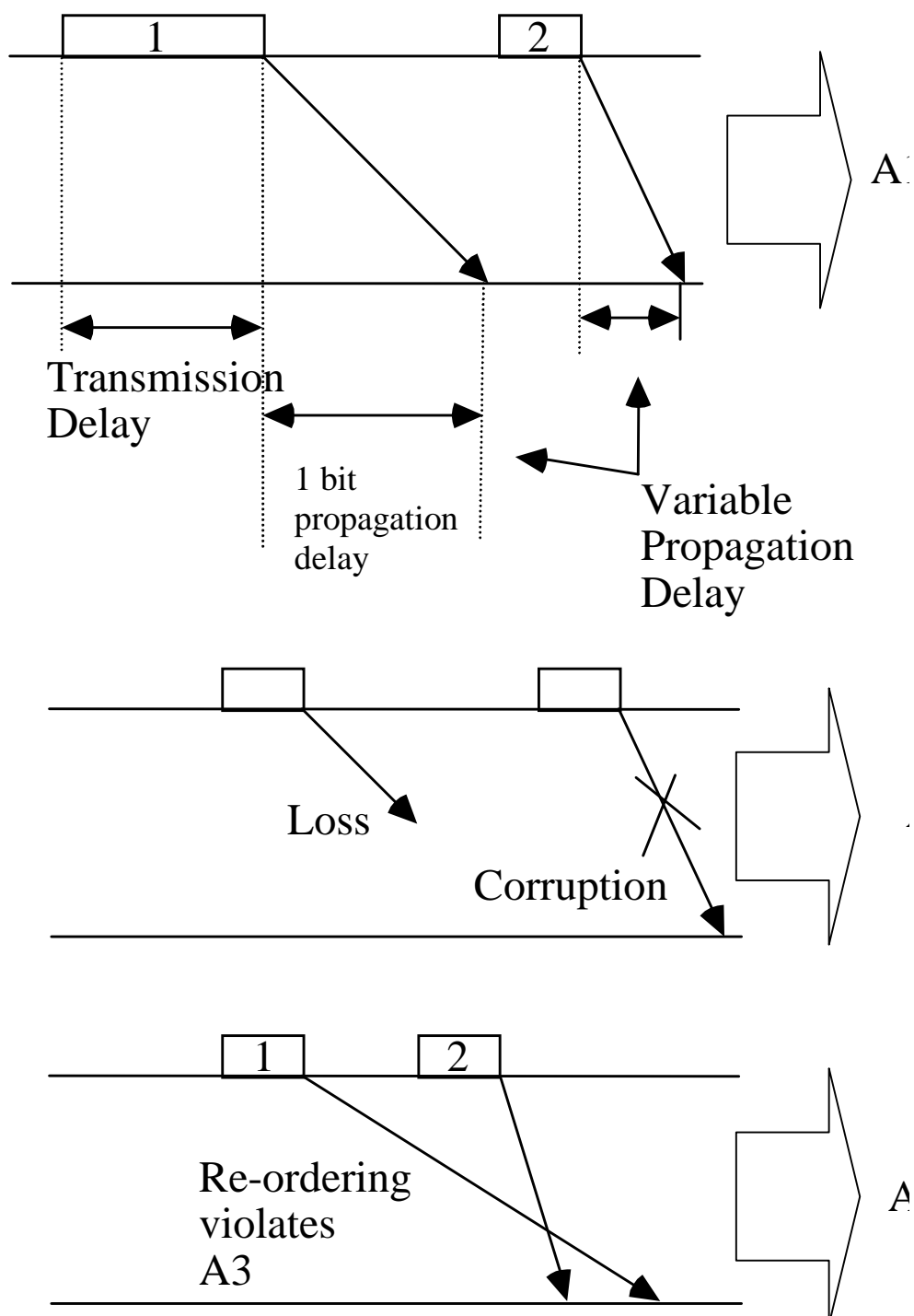
- Second Issue: Efficiency

What fraction of pipe capacity is consumed by

- Retransmission?
- "Artificial Delay"? (sender waits for an ACK)
- Overhead (non-data bits sent by protocol)
- Assumptions in constructing an ARQ protocol:
 - A1: Each frame undergoes a variable propagation delay
 - A2: Frame may be lost or corrupted
 - A3: Frames always arrive in order sent

Picture with negligible bit delay





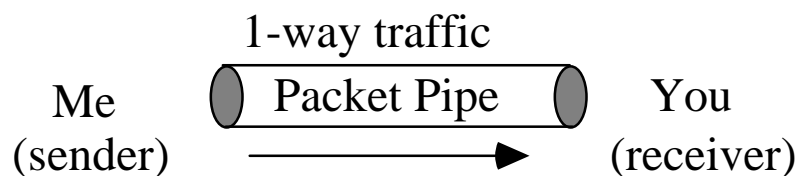
Look at notation used in Figure 2.17 in the text.

- ARQ Protocols

- Stop and Wait (BG 2.4.1)
- Sliding window a.k.a. go back-*n* (BG 2.4.2)
- Selective Repeat (BG 2.4.3)

- BG 2.4.1: Stop and Wait

- Consider 1 - Way traffic:



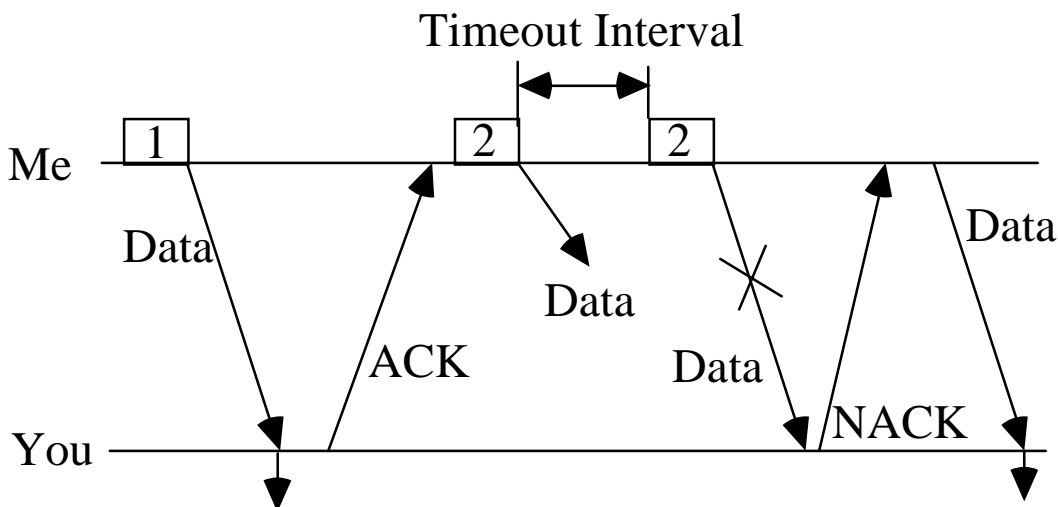
- 3 Packet Types (header bits identify type)
 - Data
 - ACK
 - NACK (sometimes used)

Stop and Wait (without sequence numbers):

- I do:
 1. Accept packet from layer 3 (possibly waiting for packet to arrive)
 2. Transmit frame containing packet
 3. Start a timer
 4. Wait for event
 5. If ((event == timer pop) or (event == NACK))
 - then { resend packet and goto 4 }
 - else if (event == receive error free ACK)
 - then { goto 1 }
 - else if (event == receive corrupted ACK)
 - then { noop }

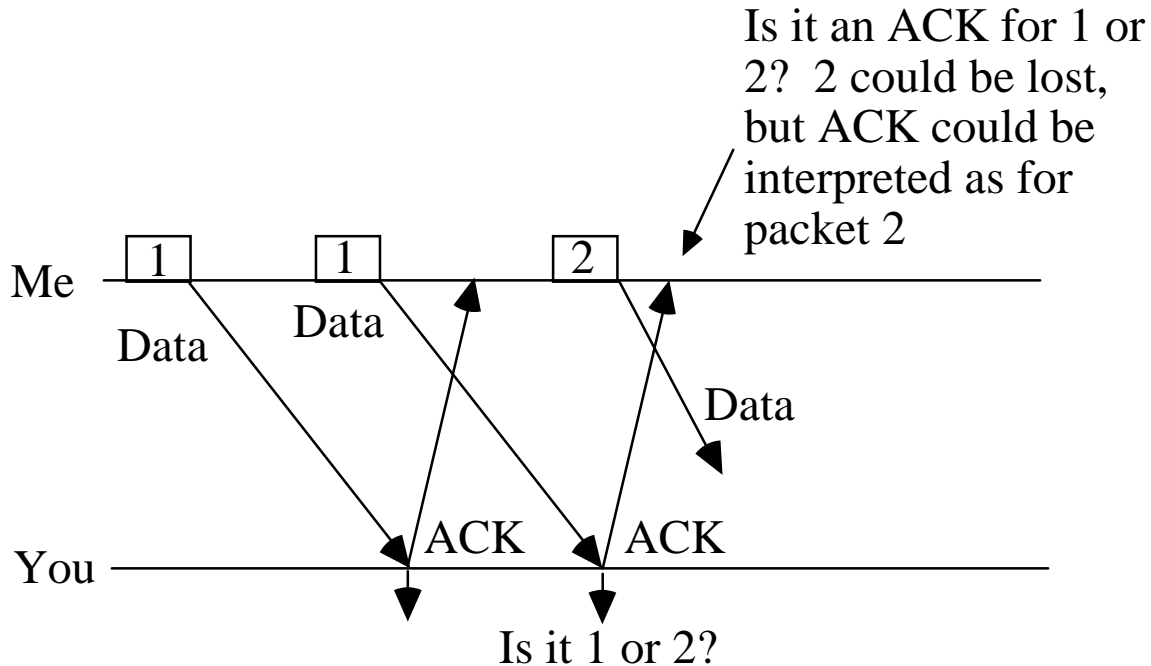
- You do:
 1. Wait for frame from link
 2. If (uncorrupted DATA frame is received)
 - then {
 - send ACK
 - release packet to layer
 - }
 - else {send NACK}

Illustration of Time-Out



- Note on timeout value:
 - Too short -> lots of unnecessary resends
 - Too much artificial waiting reduces pipe capacity
 - We set timeout to be a small (e.g., close to 1) multiple of the round trip delay (RTD)
- For a cross country hop, when ARQ is used at transport layer, estimating RTD is a big problem. Research in early 1990's successfully addressed this

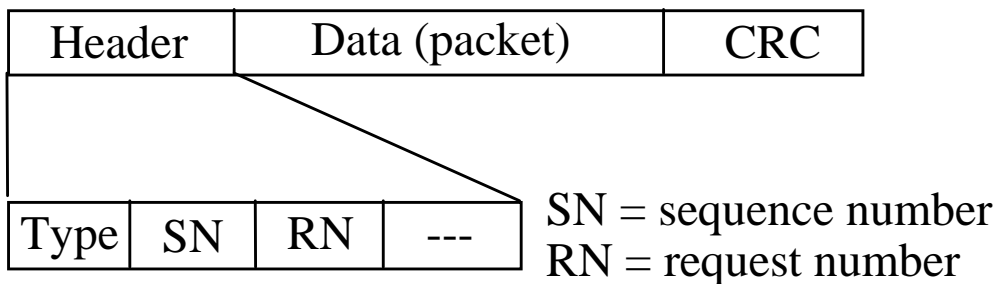
Problem with above algorithm:



- Solution: Each packet (Data, ACK, NACK) has two *sequence numbers*:

- SN = id of packet currently being sent
- RN = id of next packet expected

Each end point maintains SN, RN, too.



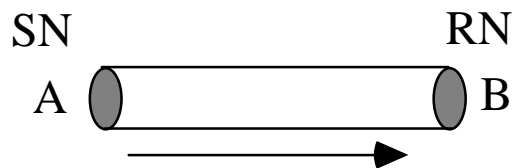
Q. Why does each packet contain both SN and RN?

A. Two way communication piggybacks ACKS/NACKS on opposite direction traffic

Note:

Receiver typically delays ACK/NACK a bit in hopes that returning data arrives from local transport to piggyback ACK

- Algorithm (initially $SN = RN = 0$):



A - to - B (no NACKS in this algorithm)

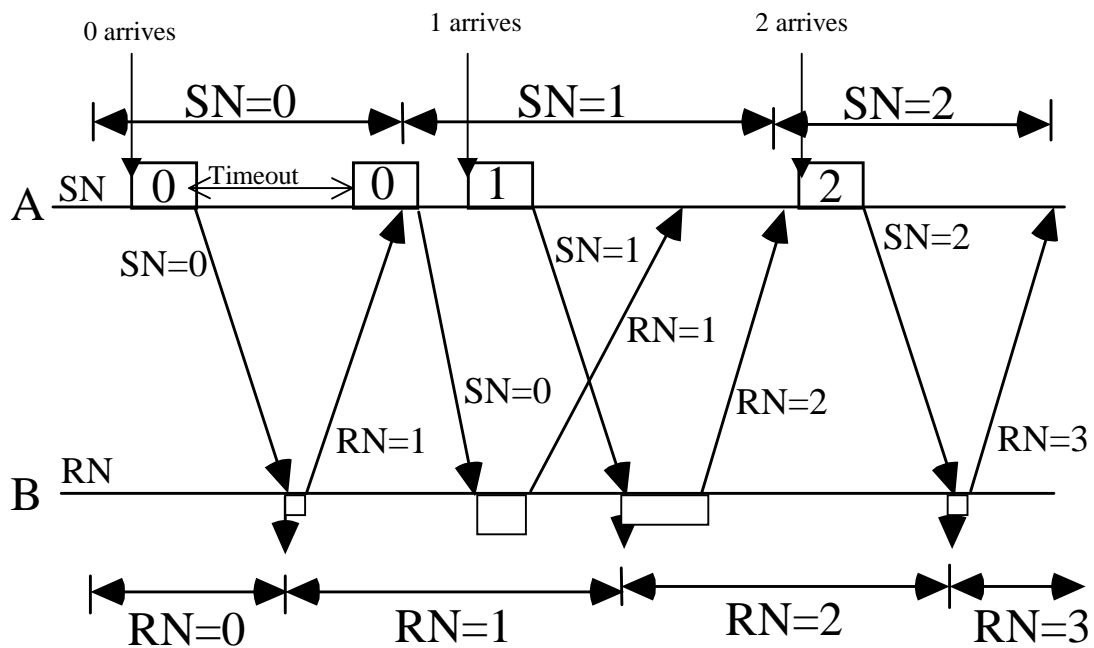
- Node A does:

- 1) $SN = 0$
- 2) Wait for & accept packet from layer 3
- 3) A) Transmit (SNth) packet in frame
 with sequence number SN;
 B) start timer
- 4) Wait for event:
 if (event == timer pop)
 then goto 3
 else if (event == receive error free frame w/ $RN > SN$)
 then { $SN = SN + 1$;
 goto step 2 }

- Node B does:

- 1) $RN = 0$
- 2) Wait for frame from link
 if (frame is uncorrupted & $SN = RN$)
 then { $RN = RN + 1$;
 release packet to layer 3 }
- 3) At arbitrary times, but within bounded delay after
 last receipt of error free frame from A: transmit
 a frame to A containing request # = RN (could
 be ACK or return direction Data)
- 4) Goto step 2 (repeat steps 2, 3, 4 forever)

A -> B Example



Correctness of Stop and Wait (general comments)

- Proof is critical for protocol specs -> zero defects. Spec written once, implemented many times by many companies for many years.
- If spec has bug, 2 company's protocols won't interoperate.
- Must also prove that an implementation matches spec.
- Correctness familiar to:
 - EE's: hardware verification
 - CS's: software correctness
- Much research in last 10-15 years on verification methods (by temporal logic, state machines, Petri nets; also on formal specification methods [e.g., LOTUS, ESTELLE])

- Need to prove:
 - *liveness*: continues forever to produce results: transport of packets from A to B continues forever (prove bounded delay); a “there exists a state” property; also, no deadlock
 - *safety*: never produce incorrect result (all packets released by B to its transport in same order sent by A); a “for all states” property
- Key to safety proof:
 - Induction on RN, the packet number released.
 - Liveness: See Fig. 22. Prove $t_1 < t_2 < t_3$. Requires

- fact that packet is delivered w/o corruption with probability $q > 0$
- Looks at SN, RN as functions of time:
 $SN(t), RN(t)$

Alternating Bit Protocol

The proof on pp. 69-71 shows that all sequence numbers can be represented using just 0 & 1, with arithmetic mod 2

(That's because only packets SN and SN-1 can be received in duplicate. Also, if receiver's waiting for RN, it could only get RN or RN-1.)

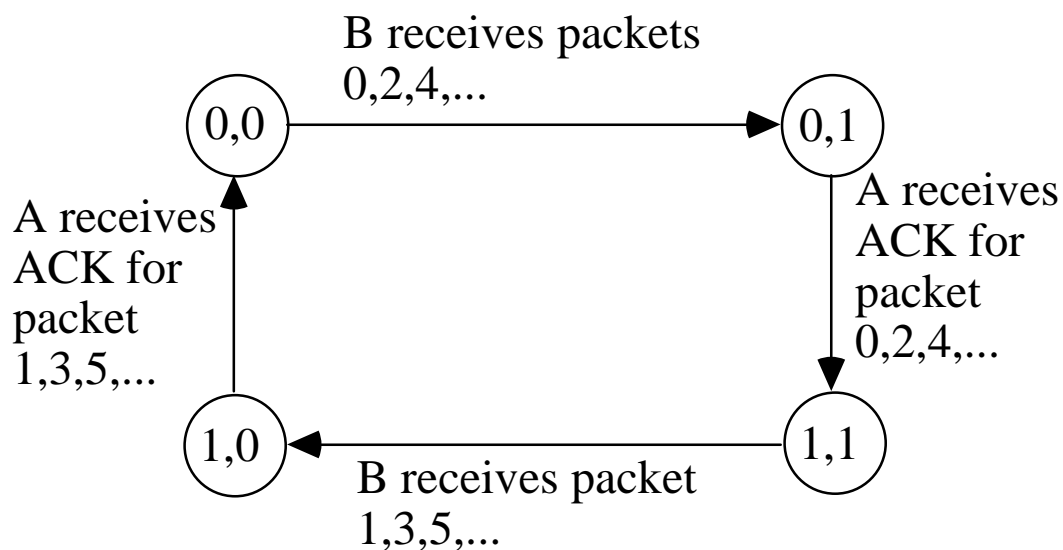
- The ability to prove a bound on sequence numbers range is critical to selecting number of bits for SN, RN in frame.

Common protocol representation (for A->B)

Can we use one picture to represent the two algorithms?

State = (Node A's SN mod 2, Node B's RN mod 2)

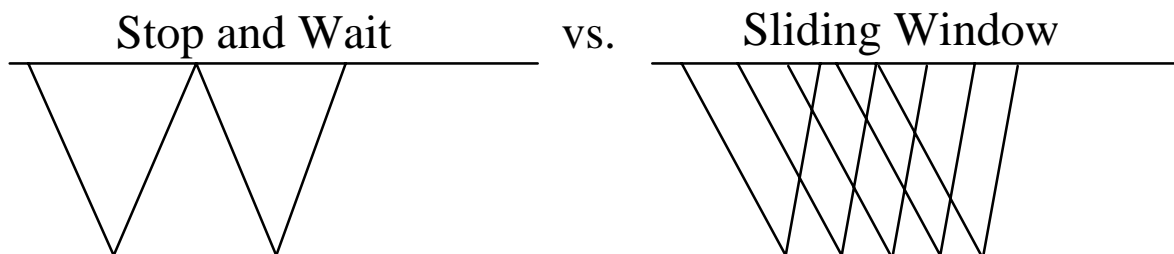
= (packet # being sent, packet # B is awaiting)



Efficiency of Stop-and-wait

In practice, stop & wait rarely used:

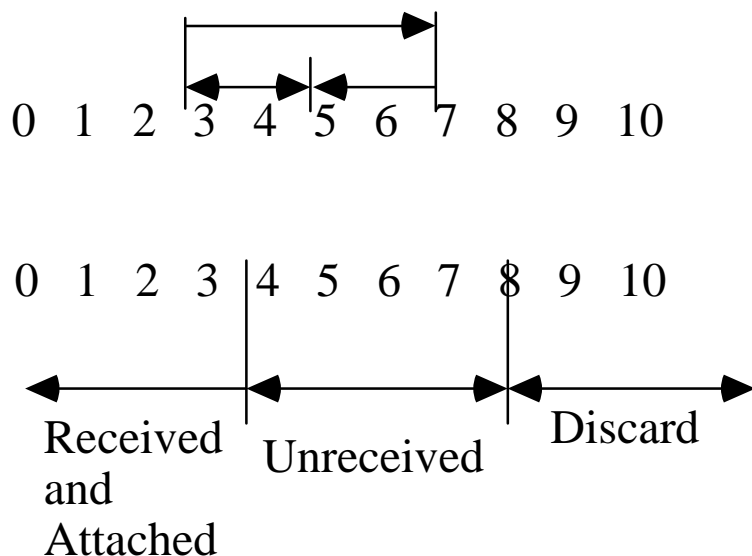
For efficiency, you want to keep pipe busy 100% of time in event no errors occur. Long propagation delay reduces efficiency (killer on satellite links -> 2 pkts/sec!).



(BG 2.4.2) Go-back N

(BG) 2.4.3 Selective Repeat

- Modification of Go back N:
 - Receiver buffers packets received out of order
 - Receive window:



- Typically receiver window size = sender size
- BG: “Receiver requests retransmission of any packets missing from received sequence.”
- TCP: Same meaning of RN ACK’s - send upper half of window (wastefully)