Summary So Far:

① We want to understand how to provide reliability
   → We do not care about speed of delivery... we only want correctness
      [in order pkts, no missing or duplicates]

② Try scenario: one pkt at a time under channel error models.

① No error → just transmit

② Bit error → Need ACKs / NACKs
   But ACKs / NACKs can also be corrupt.
   Then retransmit packet
   But if ACK was corrupt, retransmit will cause duplicates at receiver
   If NACK is corrupt, retransmit is OK
   Also need pkt sequence no. to identify duplicates
   But just two sequence nos. enough.
3. Bit error + pkt loss $\rightarrow$ Need timeout
   $\rightarrow$ Tx retransmits pkts after timeout.

4. Bit error + pkt loss + delay $\rightarrow$
   - ACK delays causing illusion that pkts have been delivered to Rx but it has not.
   - ACK delays also causing pkts to get dropped by Rx since Rx thinks it's a duplicate.
   - What about Packet delays?
Packet Delays.

Main problem:
Too small seq. # space.

A seg. # space solves this problem completely.
Estimate max. delay a packet could encounter in the network = $T_{\text{max}}$

$$\text{Seq. # space} \geq \frac{T_{\text{max}}}{E[\text{RTT}]}$$

$\#$ of bits in seq. $\#$ part of header = $\log_2 \left( \frac{T_{\text{max}}}{E[\text{RTT}]} \right)$

$\text{RTT} = \text{Round Trip time.}$
only focussed on correctness (bit error, loss, delay)
But what about efficiency? or throughput?

Efficiency = $\frac{T.T(Rx)}{RTT}$
P.D = f(packet size)?
Big packet size $\Rightarrow$ much higher prob. of pkt error.

$$p(\text{bit}_1 = \text{correct}) \cdot p(\text{bit}_2 = \text{correct}) \cdot \ldots \cdot p(\text{bit}_n = \text{correct})$$

$$= p_{\text{correct}}^n$$

Why not make pkts = 1 bit

$\Rightarrow$ Poor goodput because even for 1 bit packet you need all the headers

Typically 1500 bytes $\approx 200$ bytes (control overhead)
Correct reliable design $\rightarrow$ Stop & go.

For better efficiency $\rightarrow$ Pipelining

Pipelined protocols
So, to maximize efficiency, I should be willing to send out \( \left( \frac{RTT}{T.T} \right) \) pkts into the network w/o waiting for ACKs.
Itt.

\[ n = \frac{RTT (\text{Bottleneck})}{T.T. (\text{Bottleneck})} \]
Transport Layer

- Sockets
  - MUX/DeMUX
- Reliable pkt transfer
  - Principles
    - Correctness
      - Stop & Go
  - Performance
    - Pipelining
      - GBN
      - SACK
  - Real World
    - TCP
      - cc
    - FC
Go Back N (GBN) pipelining protocol

Main Frame → Internet → terminals

Did not have memory.

Transmit (Tx)

Duplicate ACK

Receive (Rx)

Did not have memory.

P1

P2

P3

P4

A1

A2

A2

A2

app
Tx must transmit P₃ onwards.

GBN: Tx transmits a window of pkts that can have pending ACKs (pipelining).

Rx ACKs the last correctly received in-order packet.

Rx has no memory so drops all 000 (out of order) pkts.
$W = 3$

$123$

$234$

$345$

Rx drops $P_3$ because it is 000 and Rx has no buffer to hold $P_3$. Sends ACK $A_1$

Send $A_2$

Dup ACKs for packet 1 does not matter because Tx knows $P_2$ has surely
been delivered.
\[
\text{Prob. of error (L + h bits)}
\]

\[
P_{\text{bit flip}} = P_e
\]

\[
\mathbb{E}[\text{throughput}] = \left( \frac{L}{L+h} \right) + (1-P_e)\left( \frac{L}{L+h} \right) + \ldots
\]