Summary So Far:

- 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]

- Toy 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 ⟶ Need timeout

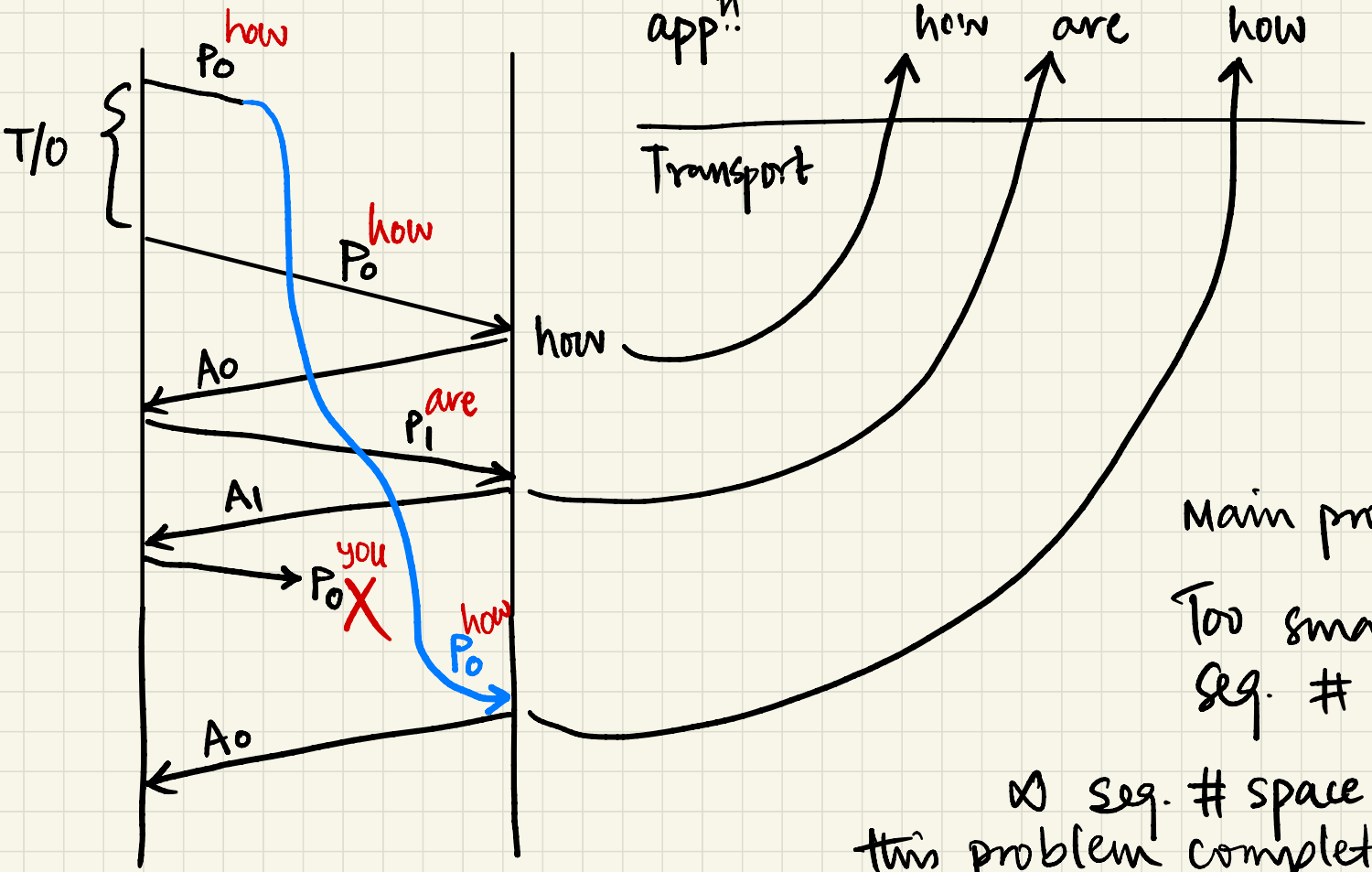
↳ T_x retransmits pkts after timeout.

④ Bit error + pkt loss + delay ⟶

- ACK delays causing illusion that pkts have been delivered to R_x but it has not
- ACK delays also causing pkts to get dropped by R_x since R_x thinks its a duplicate.
- What about Packet delays ?

Packet Delays.

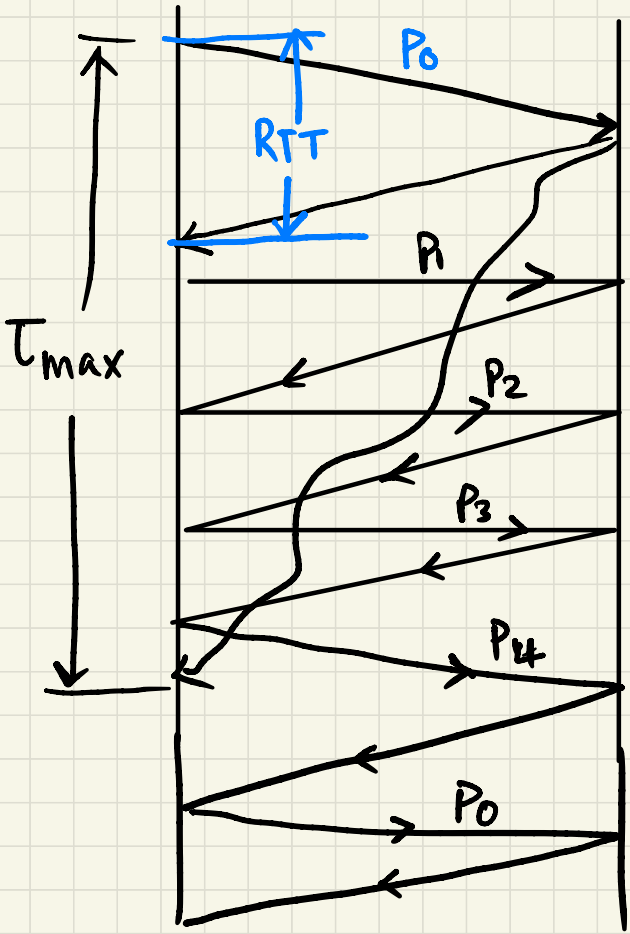
in order, duplicate, missing.



Main problem:

Too small seq. # space.

∞ seq. # space solves this problem completely.



Estimate max. delay a packet could encounter in the network $= \tau_{max}$

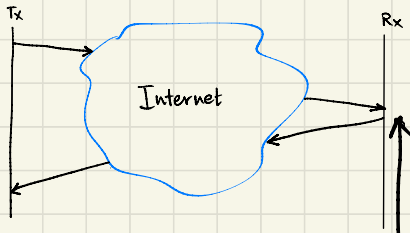
$$\boxed{\text{Seq. \# space} \geq \frac{\tau_{max}}{E[RTT]}}$$

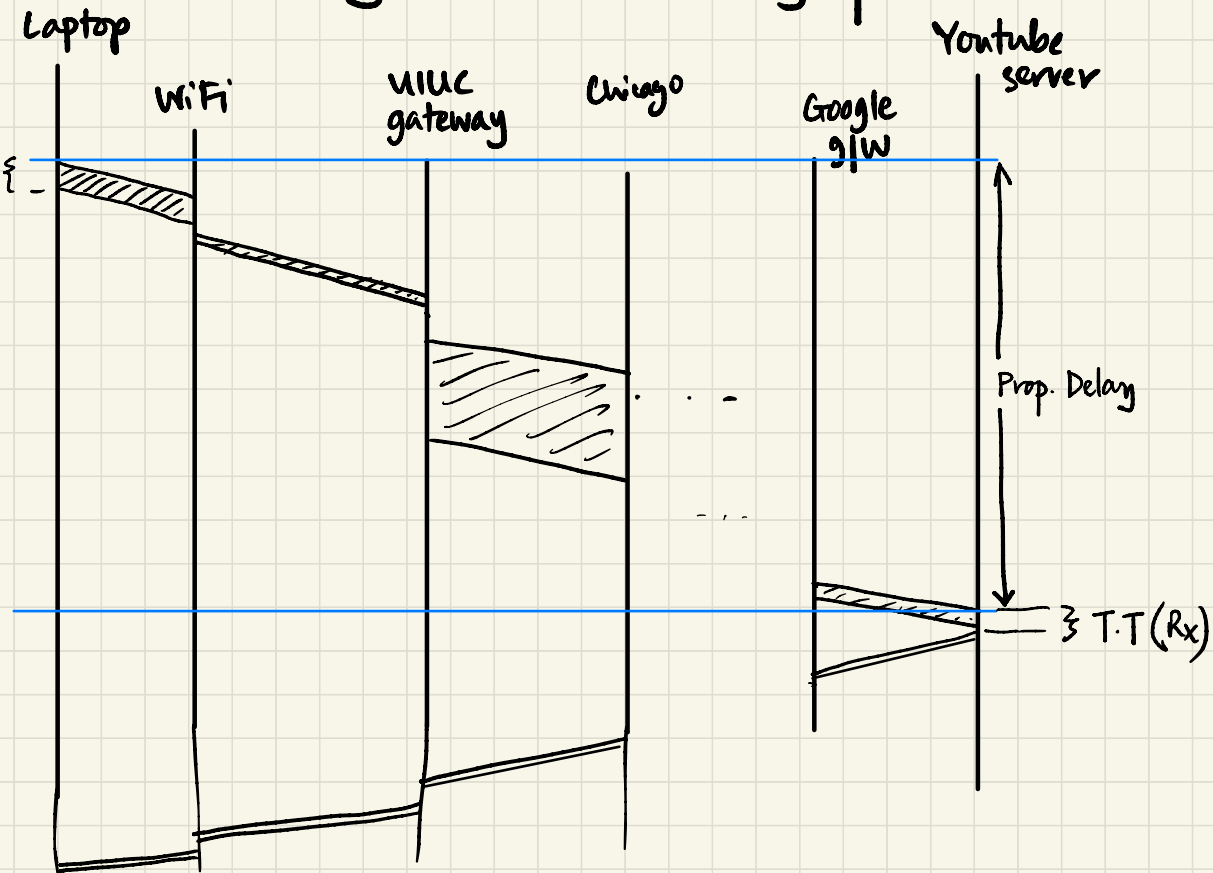
# of bits in seq. # part of header $= \log_2 \left(\frac{\tau_{max}}{E[RTT]} \right)$

RTT = Round Trip time.

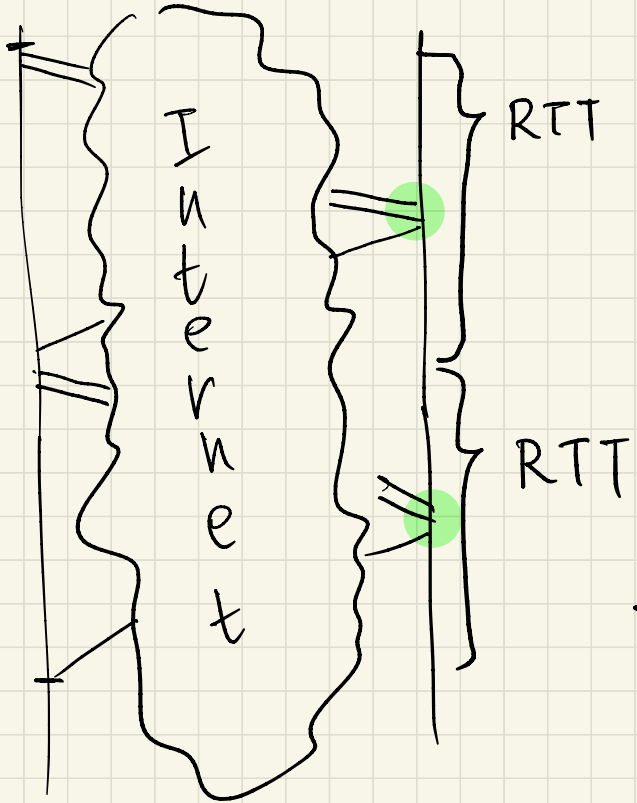
Only focussed on correctness (bit error, loss, delay)

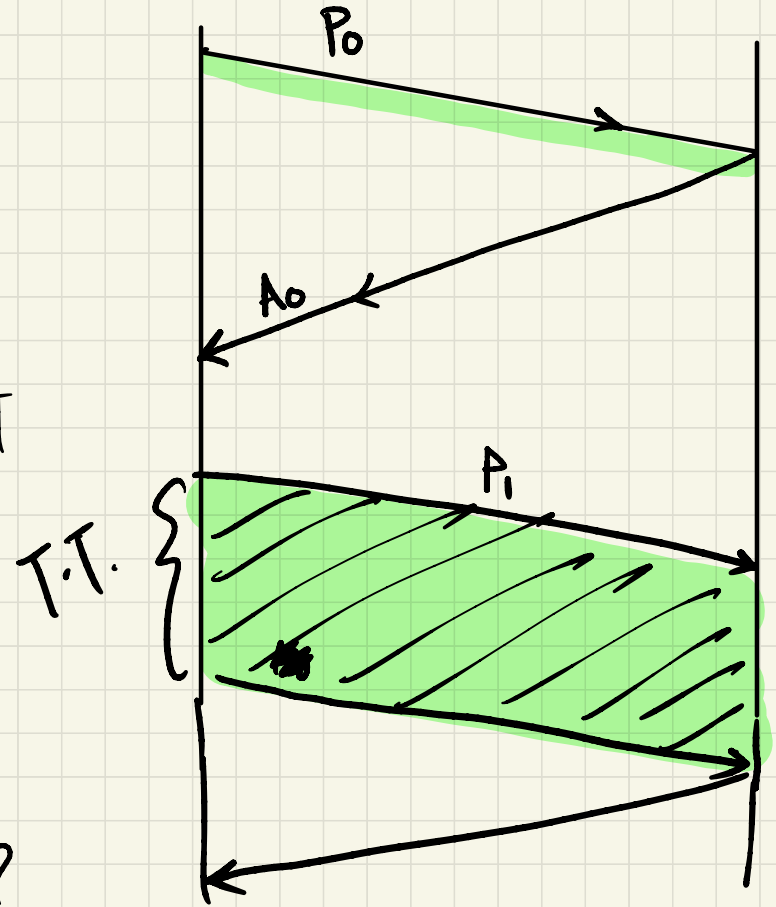
But what about efficiency ? or throughput ?



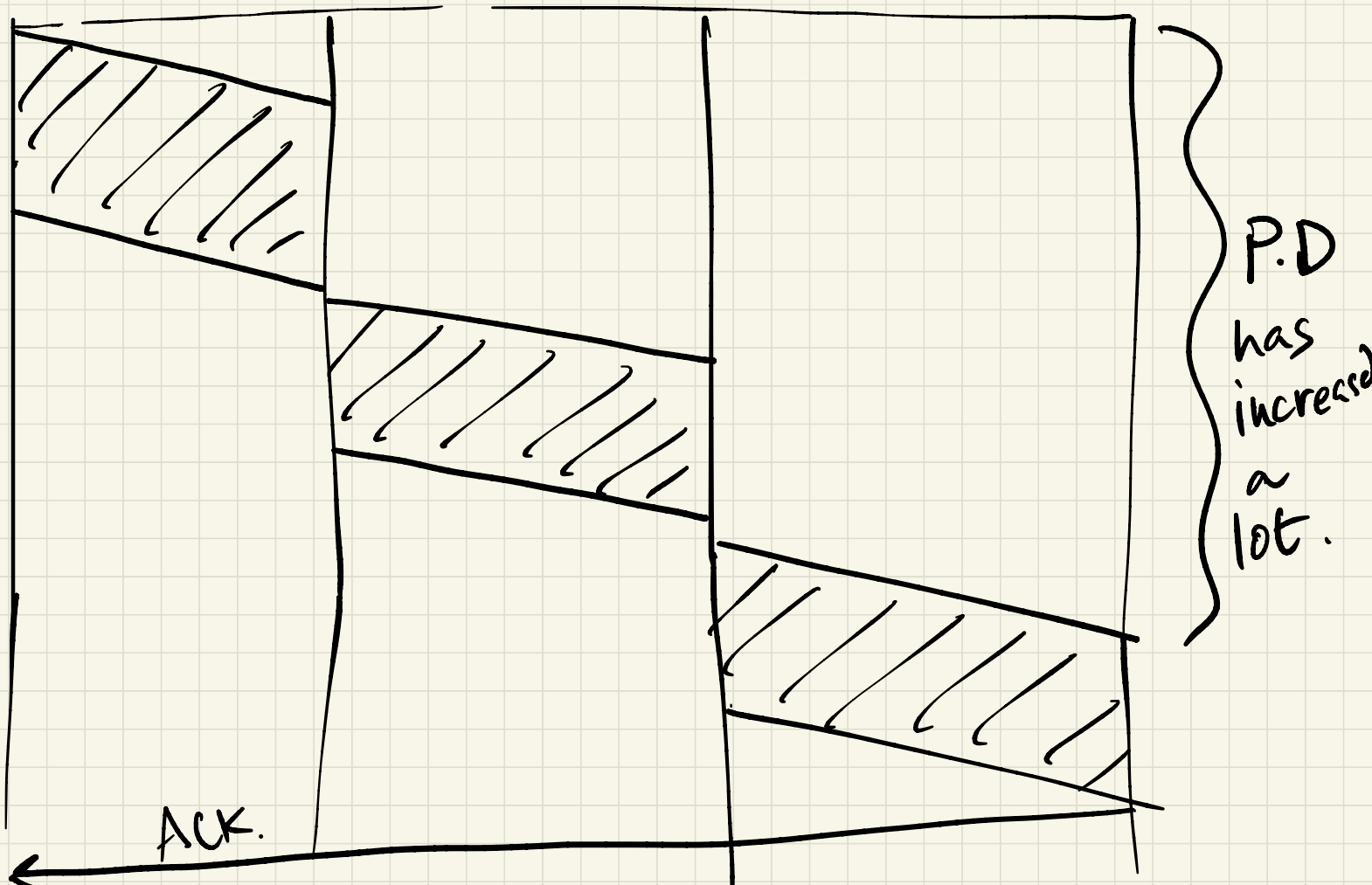


$$\text{Efficiency} = \frac{T.T(R_x)}{RTT}$$





P.D = f(packet size) ?


P.D
has
increased
a
lot.
ACK.

Big packet size $\Rightarrow$ much higher prob. of pkt error.

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

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

⊖ Why not make pkts = 1 bit

↳ Poor goodput because even for 1 bit packet you need all the headers $\approx$ 200 bytes (control overhead)

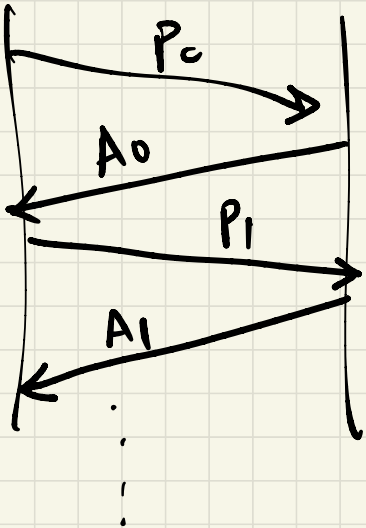
Typically 1500 bytes.

?

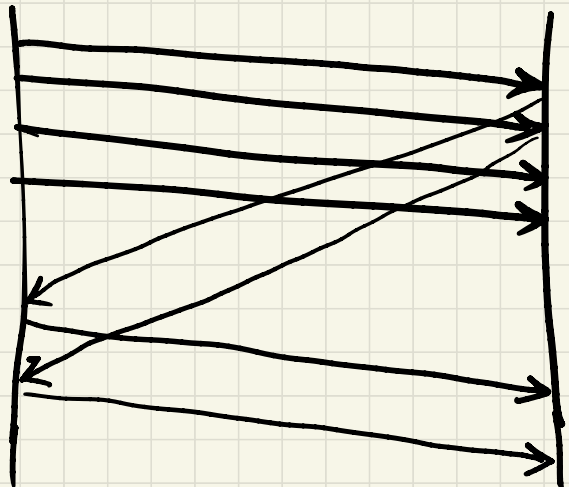
⊖ Correct reliable design → Stop & go.

↓

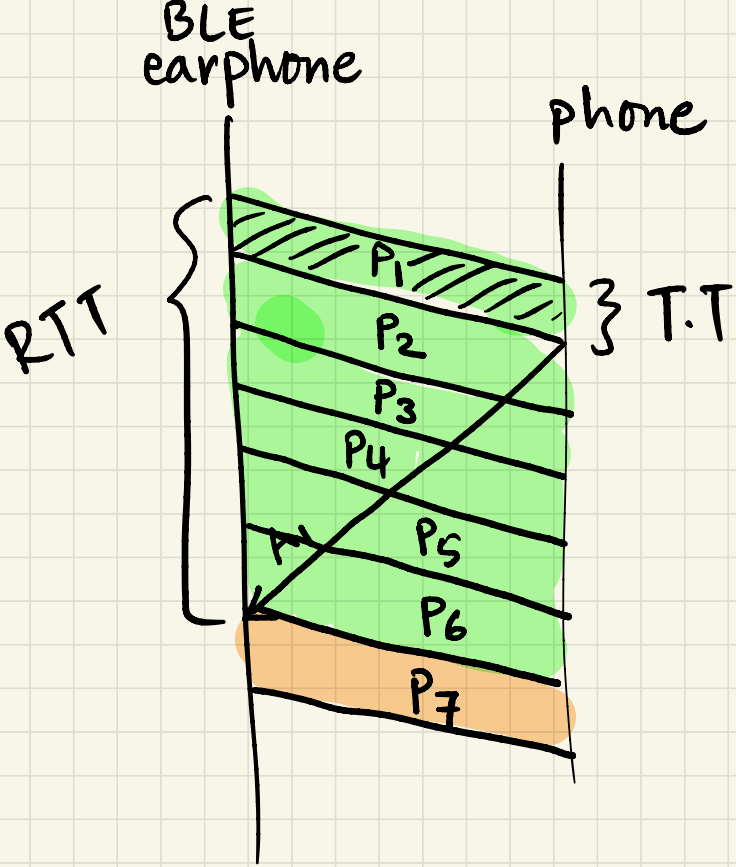
Poor efficiency.



For better efficiency. → Pipelining

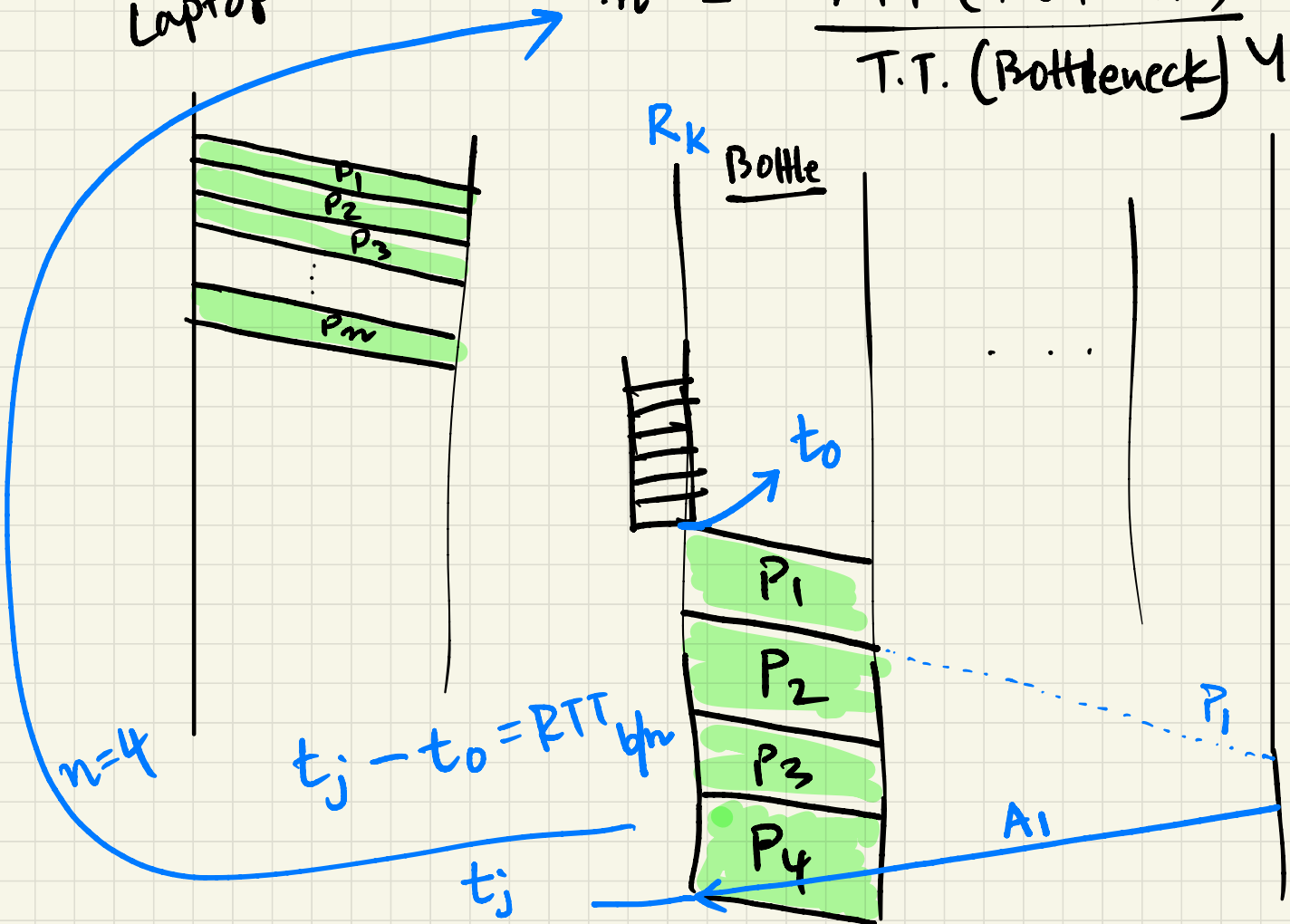


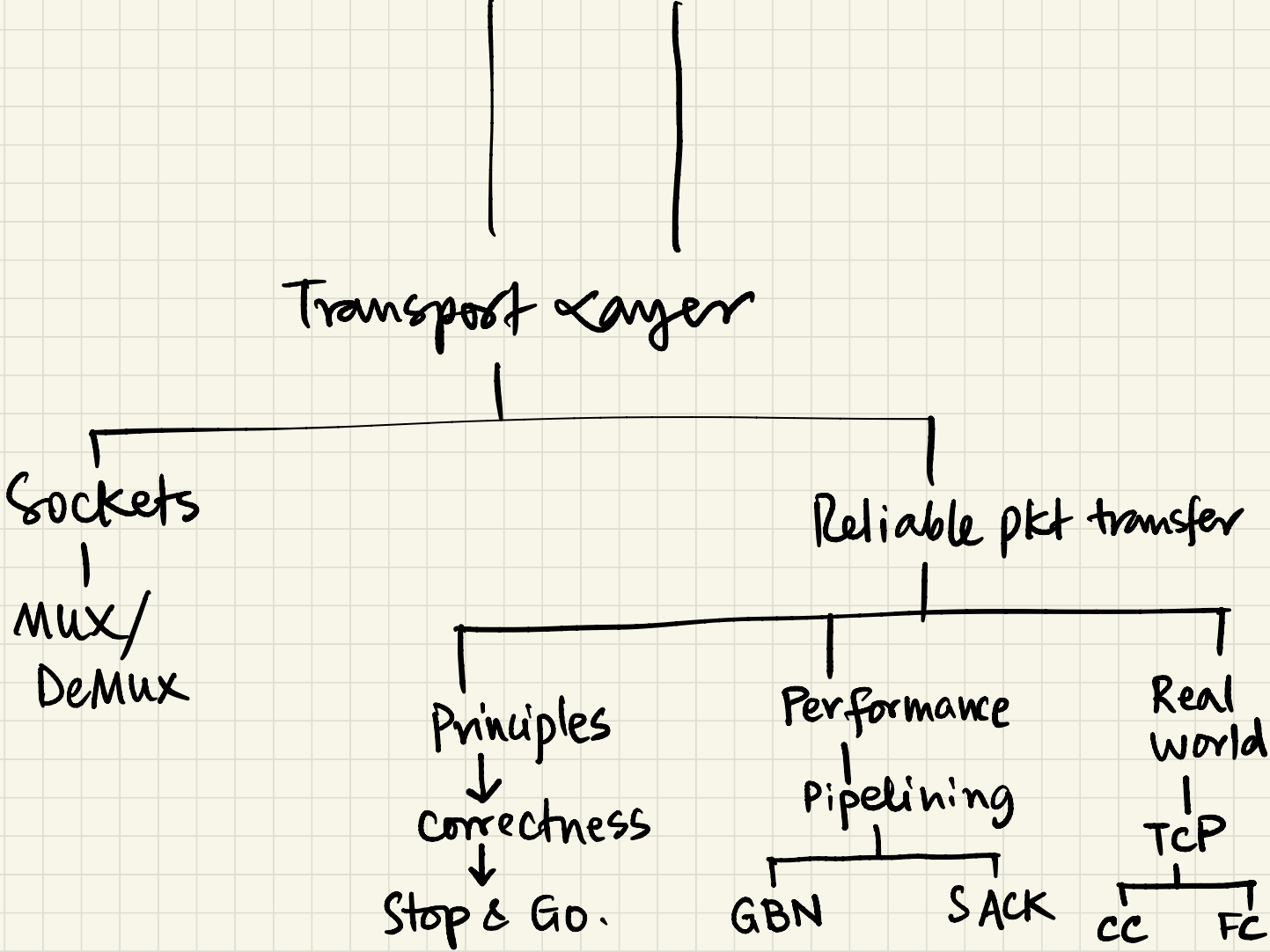
Pipelined protocols



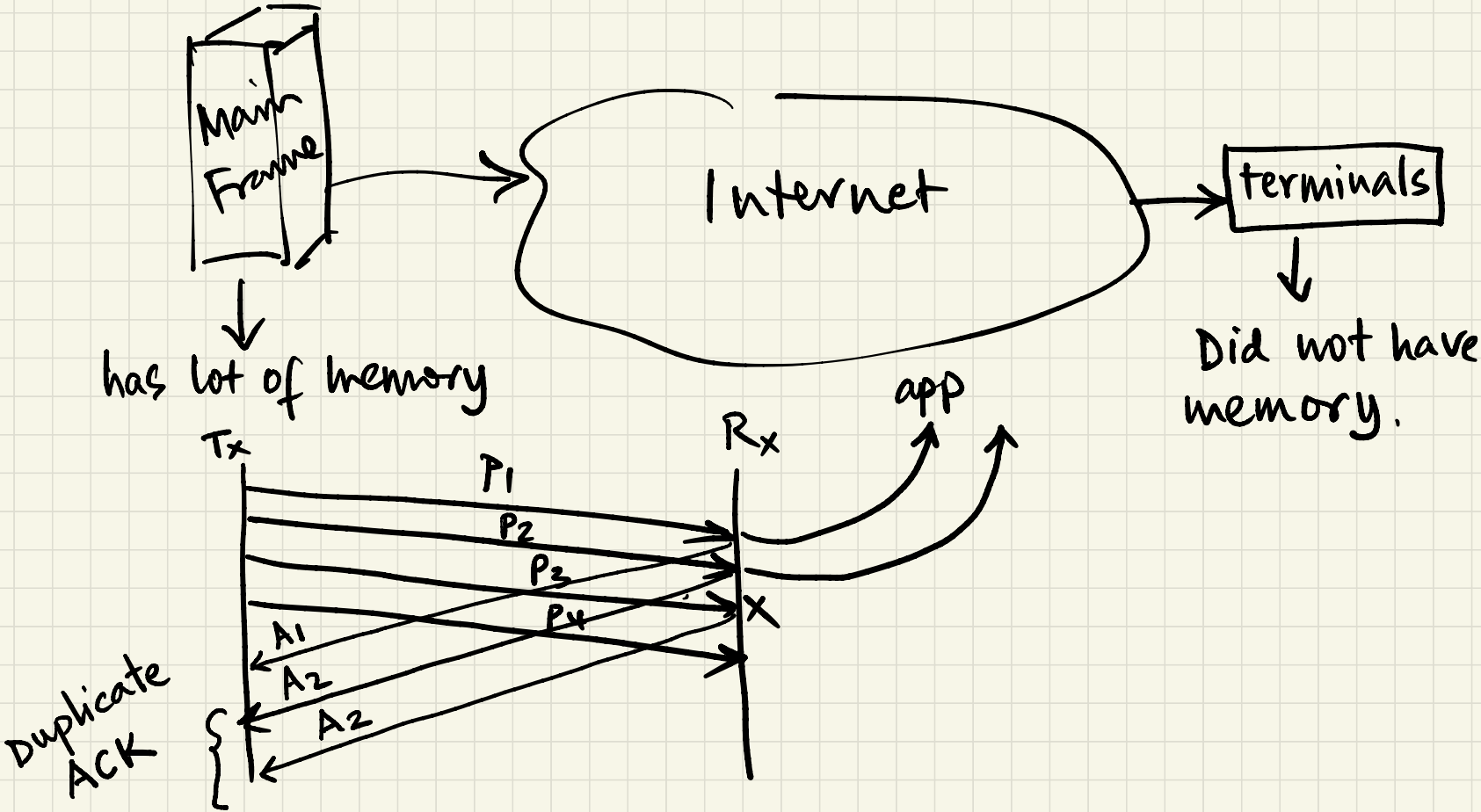
So, to maximize efficiency, I should be willing to send out $\left(\frac{RTT}{T.T}\right)$ pkts into the network w/o waiting for ACKs.

$$n = \frac{RTT\ (\text{Bottneck})}{T.T.\ (\text{Bottleneck})}$$ Yontwoc




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

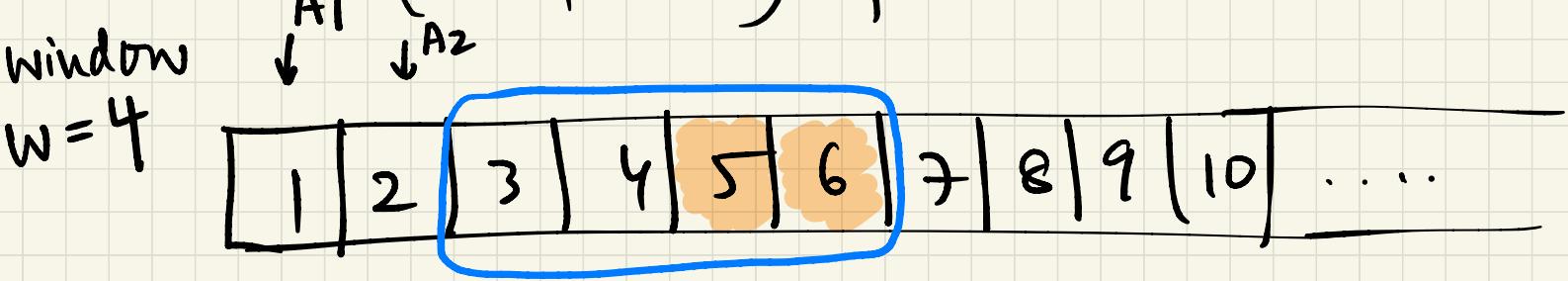


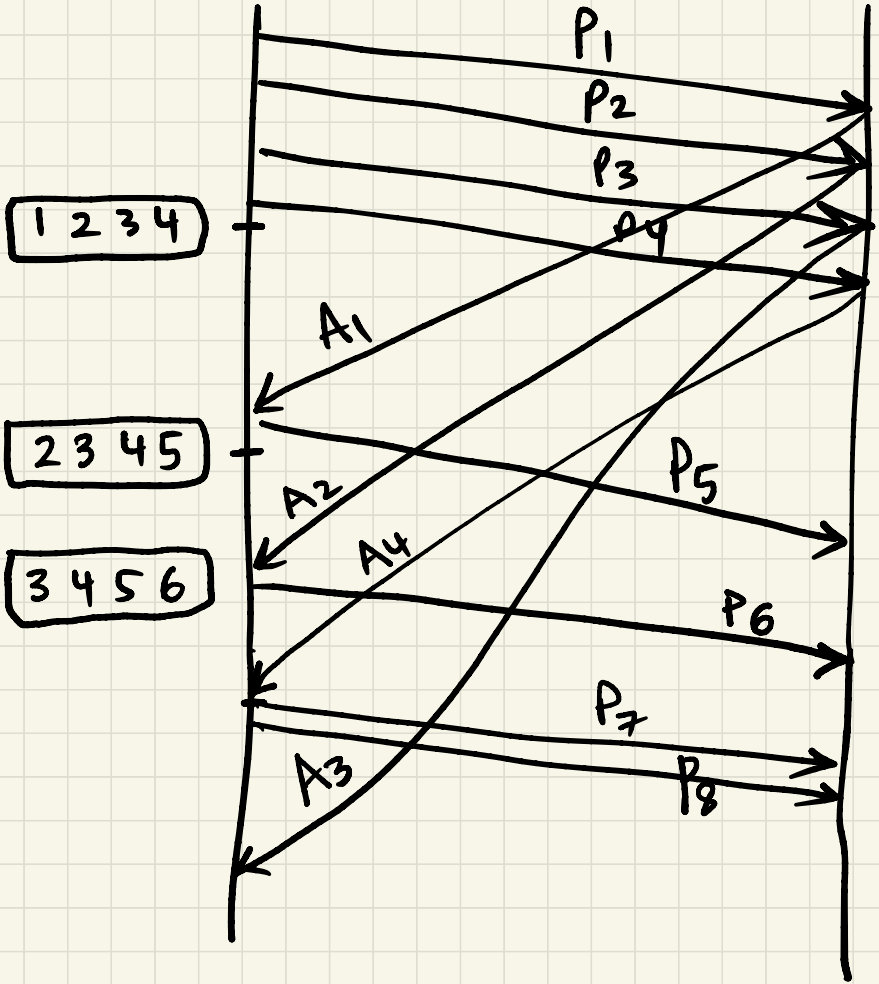
T_x must transmit P_3 onwards.

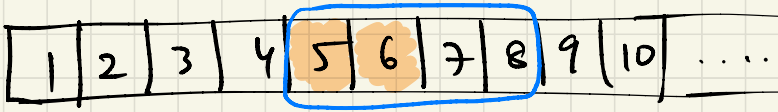
GBN : T_x transmits a window of pkts that can have pending ACKs (pipelining)

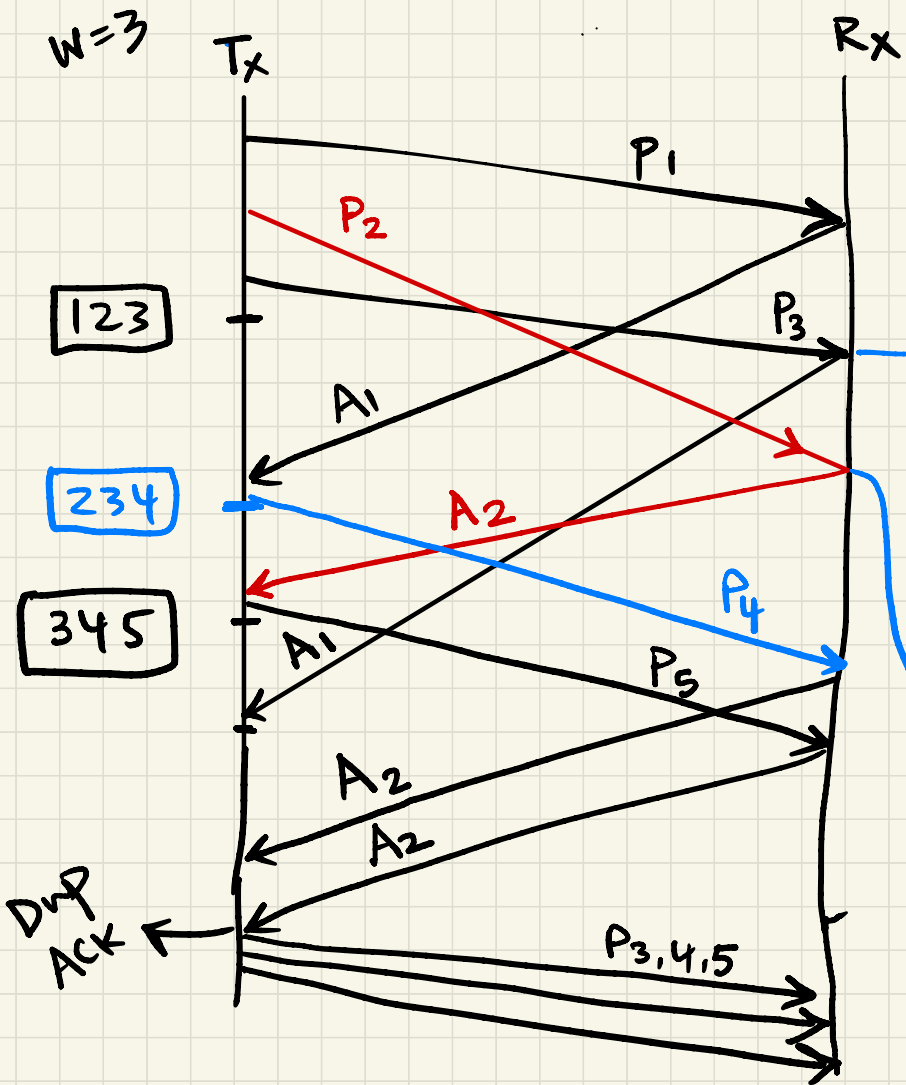
R_x ACKs the last correctly received in-order packet.

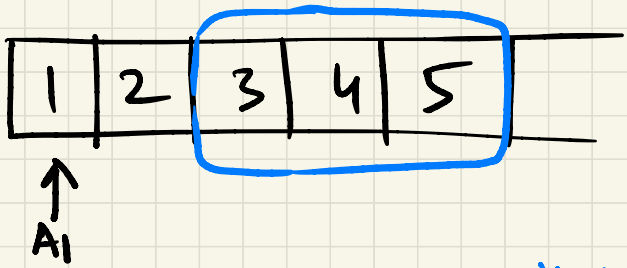
R_x has no memory so drops all OOO (out of order) pkts.




P1
P2
P3
P4
1 2 3 4
A1
2 3 4 5
P5
A2
A4
3 4 5 6
P6
P7
A3
P8


1
2
3
4
5
6
7
8
9
10
. . . .





Rx drops P_3 because it is OOO and Rx has no buffer to hold P_3.

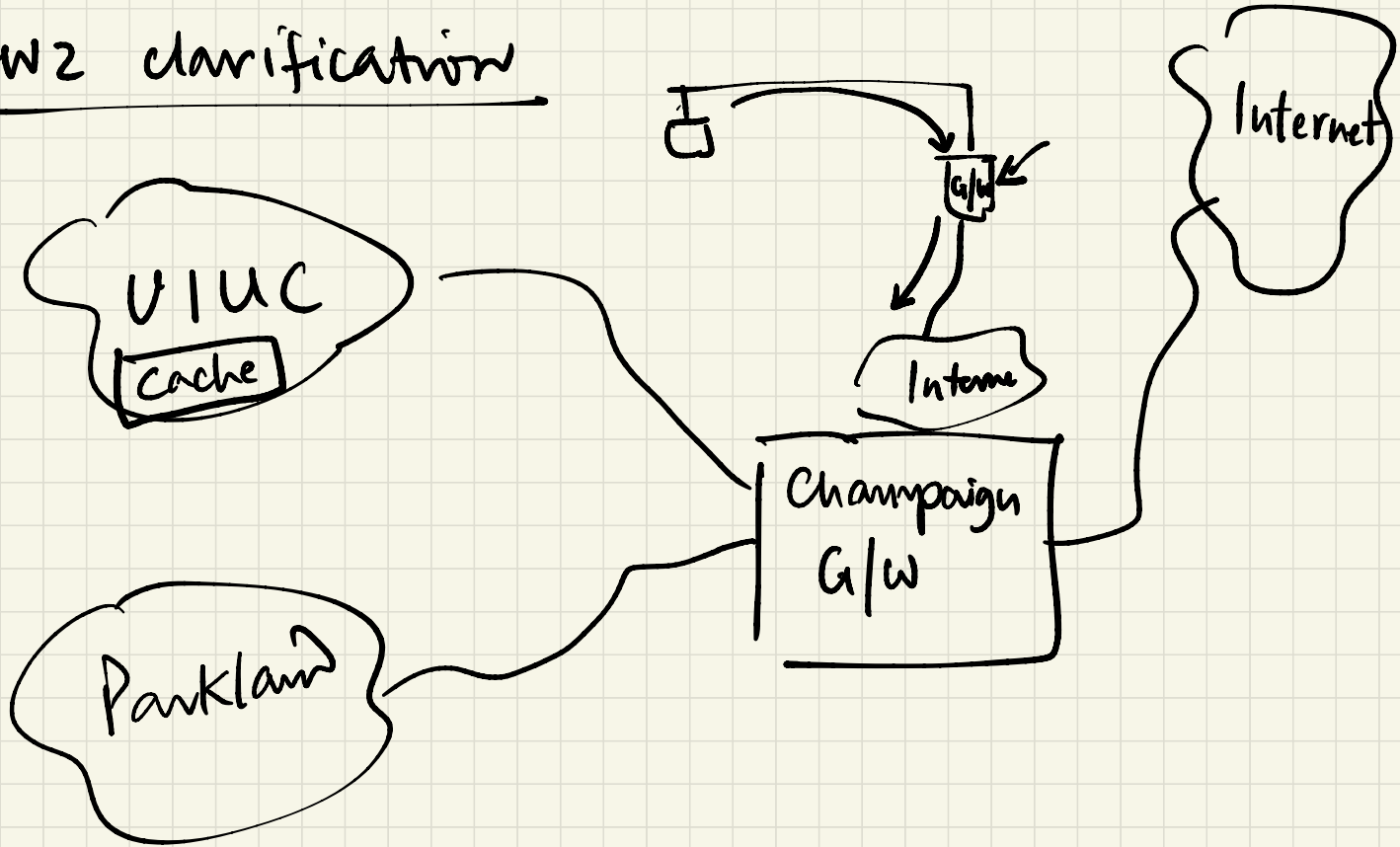
Sends ACK A_1

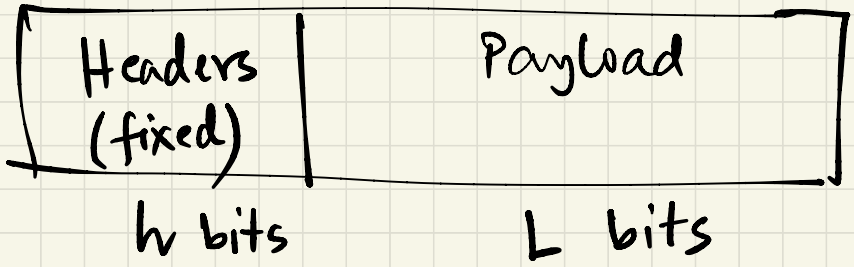
Send A_2

Dup ACKs for packet 1 does not matter because Tx knows P_2 has surely

been delivered.

HW2 clarification





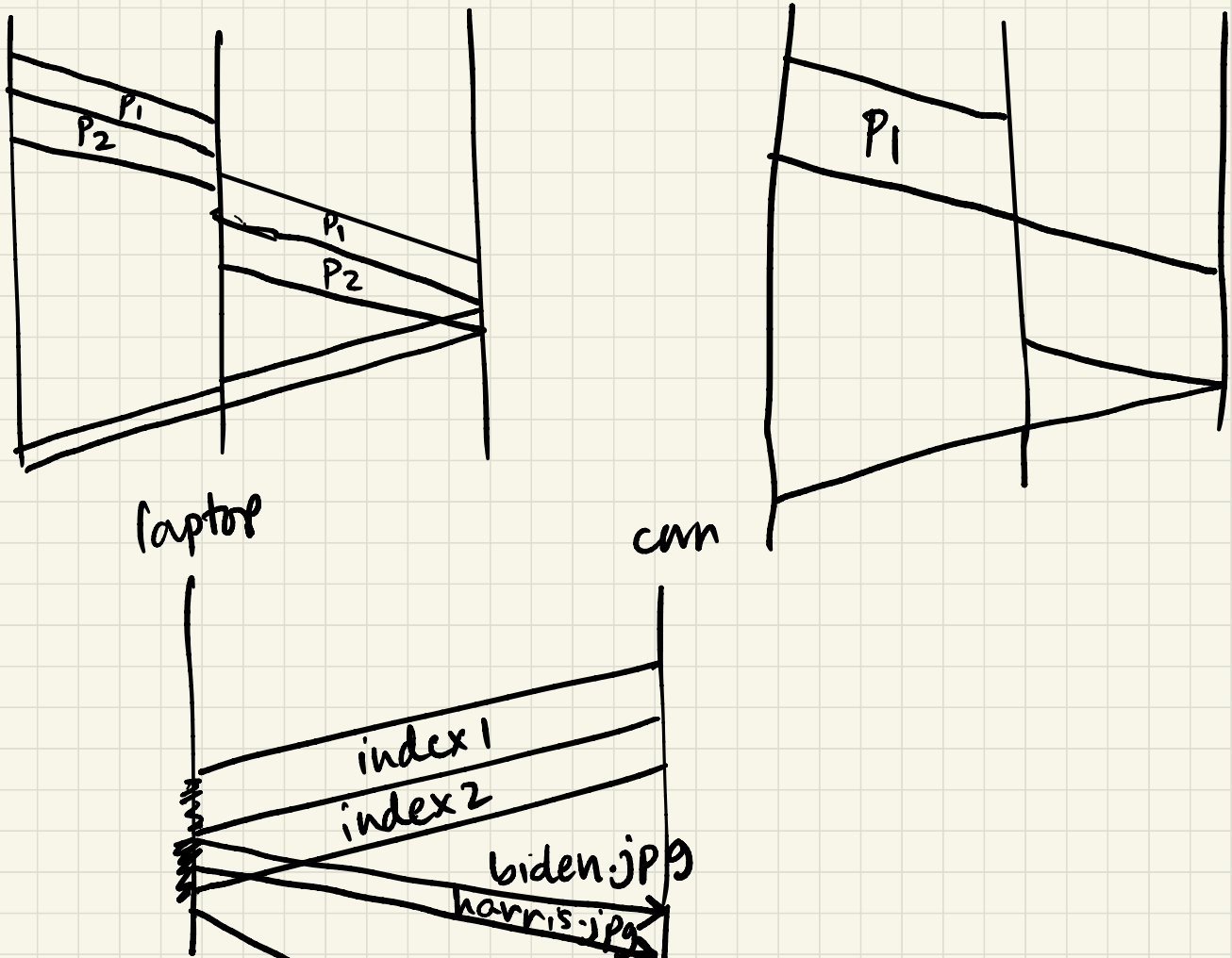
$$\left(\frac{L}{L+h}\right)$$

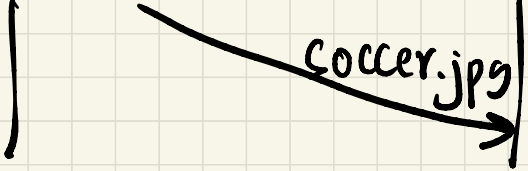
Prob. of error (L+h bits)

$$P_{\text{bit flip}} = P_f$$

E[throughput]

$$\left(\frac{L}{L+h}\right) + (1-P_e)\frac{L}{(L+h)} + \cdots\cdots$$


P1
P2
P1
P2
P1
laptop
cam
index1
index2
biden.jpg
harris.jpg


coccer.jpg