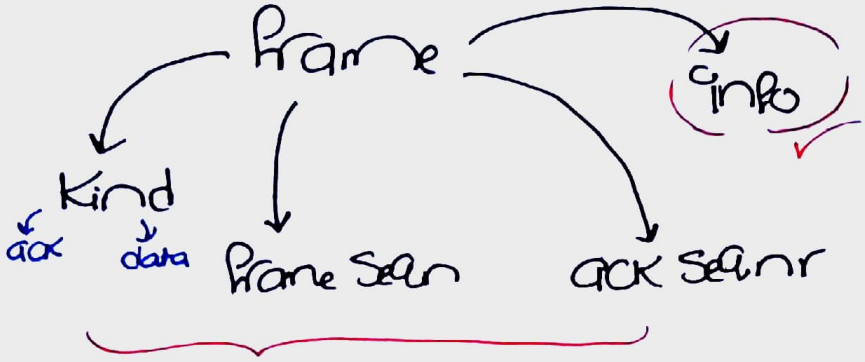
CN Sheet 3

* Unrestricted Simplex Protocol
(No errors, ∞ buffers, ∞ Processing, ∞ Packets..)

1. Why don't we need to initialize frame header ?



- there are no errors, never need to retransmit hence never need ack (likewise, buffers are ∞ don't need to confirm having empty buffer)

- Also implies we won't need seq. numbers (frames will always arrive at the right order undamaged & with no duplication or loss)

2. Why no timer?

→ The receiver sends back nothing so nothing to set timer on for the sender.

→ receiver will always receive intended message

- Receiver will eventually stop waiting (when receiving) and no error can cause it to wait forever.

- Usually need timer to prevent deadlock (Sender waiting for ack, receiver waiting next frame, ack gets lost)

14. Consider unrestricted simplex protocol

→ Suppose we start assuming that one type of error is possible

- Packet duplication in reverse channel (i.e., from receiver to sender)

what do we need to change in the protocol?

→ Nothing

→ Receiver sends nothing back to sender (not even control data (acks))

(there are no packets to duplicate)

2 x 0 = 0

* Simplex Stop & Wait
→ Stops assuming ∞ buffers & Processing

3. Why no Sequence numbers?

Network Packet

- Sender Sends frame then Stops & waits
- receiver wakes up Sender with dummy ack after it receives and finishes Processing.

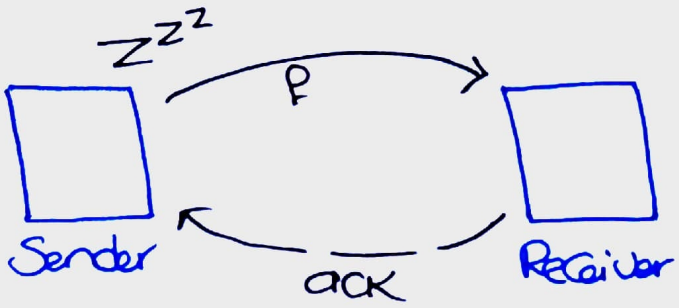
→ Clearly, its a transmission of one frame at a time
- Packets will be received at the exact order the Sender's network layer chose
- None of them ever dropped, damaged or duplicated (Perfect in-order delivery)

Hence, no need for Sequence numbers
- They are usually needed to distinguish the received frame with the one expected
→ here they're always the same.

4. Why is the ack 'dummy' with no seq nr?

- Same reason, we ack. one frame at a time So there're no other acks that we need to distinguish.
(receiver will never ack. the wrong frame or have the ack get lost)

5. Why no Sender timer? Receiver timer?



- guaranteed not to wait forever
  → error free channel & ack. will arrive eventually.
- guaranteed not to wait forever
  → Sender's frame will eventually arrive, channel is error free.

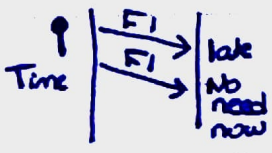
- No deadlock is possible (in other words).

* Simplex Protocol for Noisy Channel

→ Channel no longer error free + Prop. delay
→ errors are perfectly detected

7. Why Seq nrs on frames & why just one bit?

- Receiver may get same frame twice
  → e.g. if its ackn. gets lost.
- Noisy channel can cause frames to arrive later (Previous Send round)
  → need to check if its the expected frame

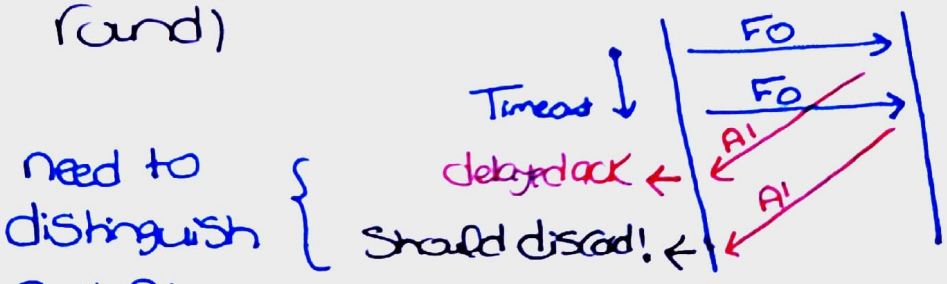


- It's only one bit as
  - → Sender never sends a new frame before the previous is acknowledged.
    - So receiver only needs to distinguish between if it has received it earlier or not to protect from duplication
    - .... or not takes 1 bit.

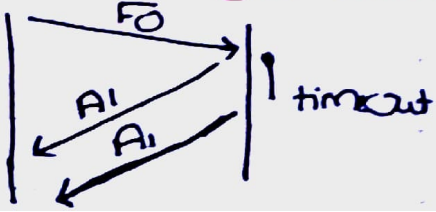
8. Why we need sequence nrs for ack?

→ Sender needs to check that receiver is acknowledging the right frame (most recently sent)

- This might be violated if
  - → A delayed ack. arrives (from previous round)



→ If we allow receiver to set timers on ack. (we didn't in lecture)
- then if an earlier ack. arrives later (after timeout) we get same prob.



9. Why Sender needs timer?

- Channel is subject to errors
  - → Frame may get lost or receiver might discard it if damaged or out of order.
  - → If the Sender has no way to detect that this happened we enter a deadlock State.
    - Sender waiting for an ack. forever
    - Receiver never getting the frame it should ack.

Sender timeout = break ↵

10. Why receiver needs timer?

- Same argument as above but for acknowledgement frames.
  - Sender waiting for ack. forever
  - Receiver never getting the next frame
    - ↳ receiver timeout = break (resend lost ack.)

- Note
  - → In the actual Protocol
    - We only used Sender timer
    - In the two answers above we assumed the other end had no timer
    - In general, only need one of them

* Link Capacity

11. Digital Transmission System between Cairo and Alexandria.

Frame of 512 Byte

- Randtrip delay is 250 MS using a T1-line of 1500 KBIS
  - B capital is for byte
- OverHead can be neglected
- Total throughput & efficiency using Stop & wait ARQ?

⇒ Given

$2T_P = 250\,MS$ (Randtrip delay)

$B = 1500\ KBIS$

$L = 512\ B$

- Worked $\eta = \frac{T_t}{T_t + 2T_P}$ and $B_{eff} = \eta B$

$\rightarrow T_t = \frac{L}{B} = \frac{512B}{1500KBIS} = 3.4 \times 10^{-4} S$

$\eta = \frac{3.4 \times 10^{-4}}{3.4 \times 10^{-4} + 250 \times 10^{-6}} = 0.576$

Throughput = B_{eff} = $\eta \cdot B$ = 0.58×1500
= 870 KBPS

* Design Problems

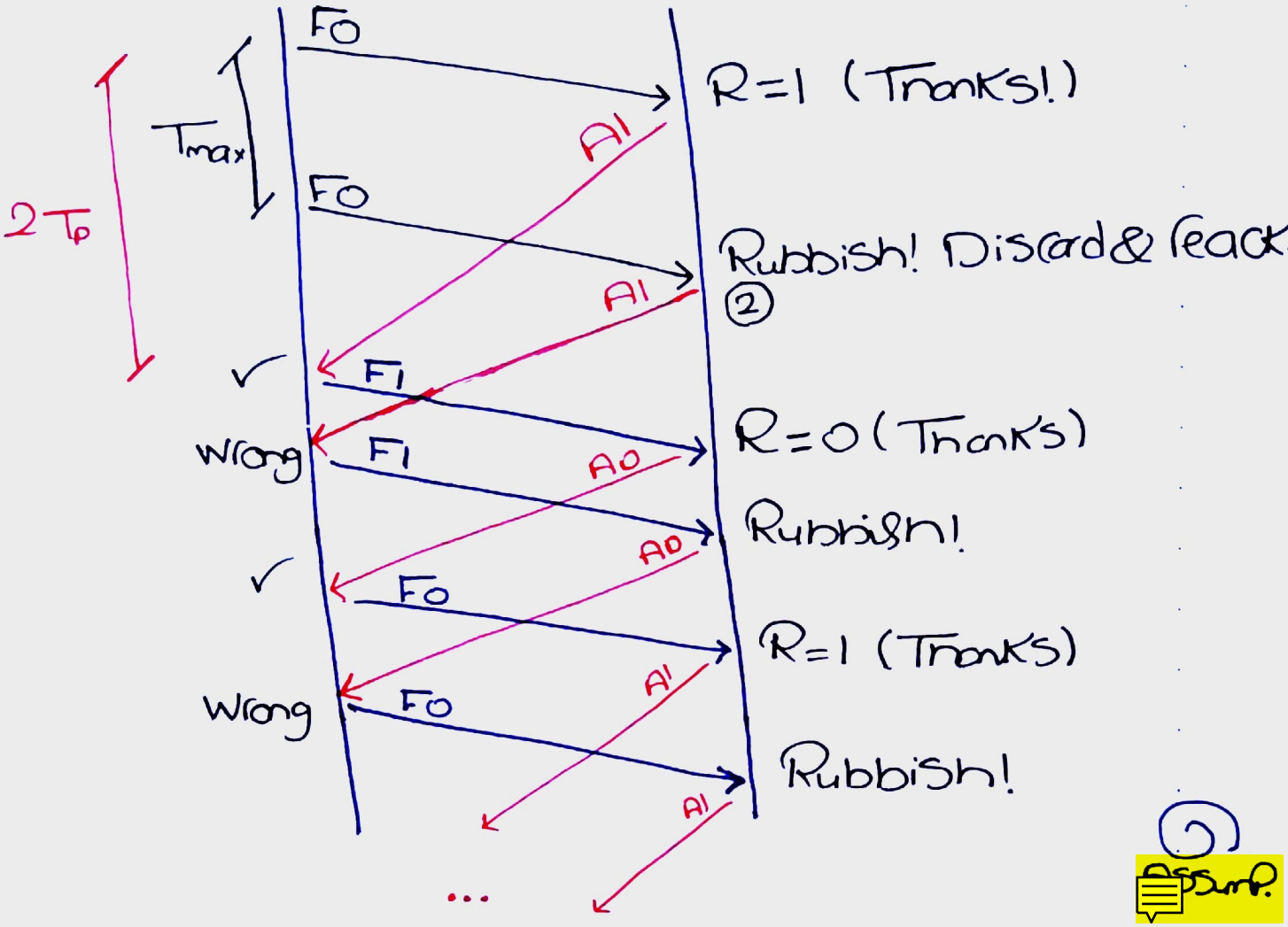
12.

- Suppose Sender timeout (T_{max}) is smaller than the round-trip delay ($2T_p$)
- would this cause any problems? Even if it only happened once (for one packet)?

⇒ Recall that roundtrip time is the time taken for the message that has left the Sender to reach receiver + time for ack to get back to Sender.

- Clearly, if Sender has a timeout interval smaller than this
  → it will resend the message even before the ack arrives

- So if we assume that this happens only for one round



- As can be seen, in this case
  - → we assumed it only happened at 1st round (i.e., the ack at ② arrived before timeout); otherwise, it gets even much worse.
  - → The consequence is that all further frames and acks were duplicated which is a big reduction in efficiency (50% $\eta_{tot.}$) compared to normal case
- This, however, has not affected Correctness & Protocol (no wrong frames/acks were accepted).

13. Consider Simplex Protocol for noisy channel

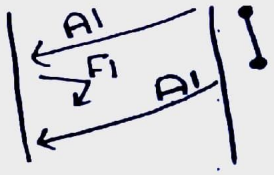
→ If Sender gets duplicate ack, it resends frame. why?

- According to the slides, it doesn't. It just discards and relies on time out to send fram again (Sufficient).

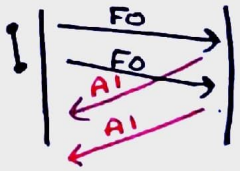
hmm.

- Possible Causes of duplicate ACK

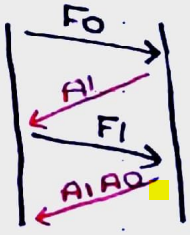
→ Frame was lost and receiver timed out (resend ack.)



→ ACK. was delayed (from Previous round)



→ ACK was modified by channel



→ ACK. duplication by channel

- If there's a reason to resend the frame then its because besides timeout, this is the only way to restor synchronisation between Sender & receiver

→ i.e., for receiver to consider sending the correct ack