Computer Networks
COL 334/672

Principles of Reliable Communication

Slides adapted from KR

Sem 1, 2025-26

Recap: Transport services and protocols

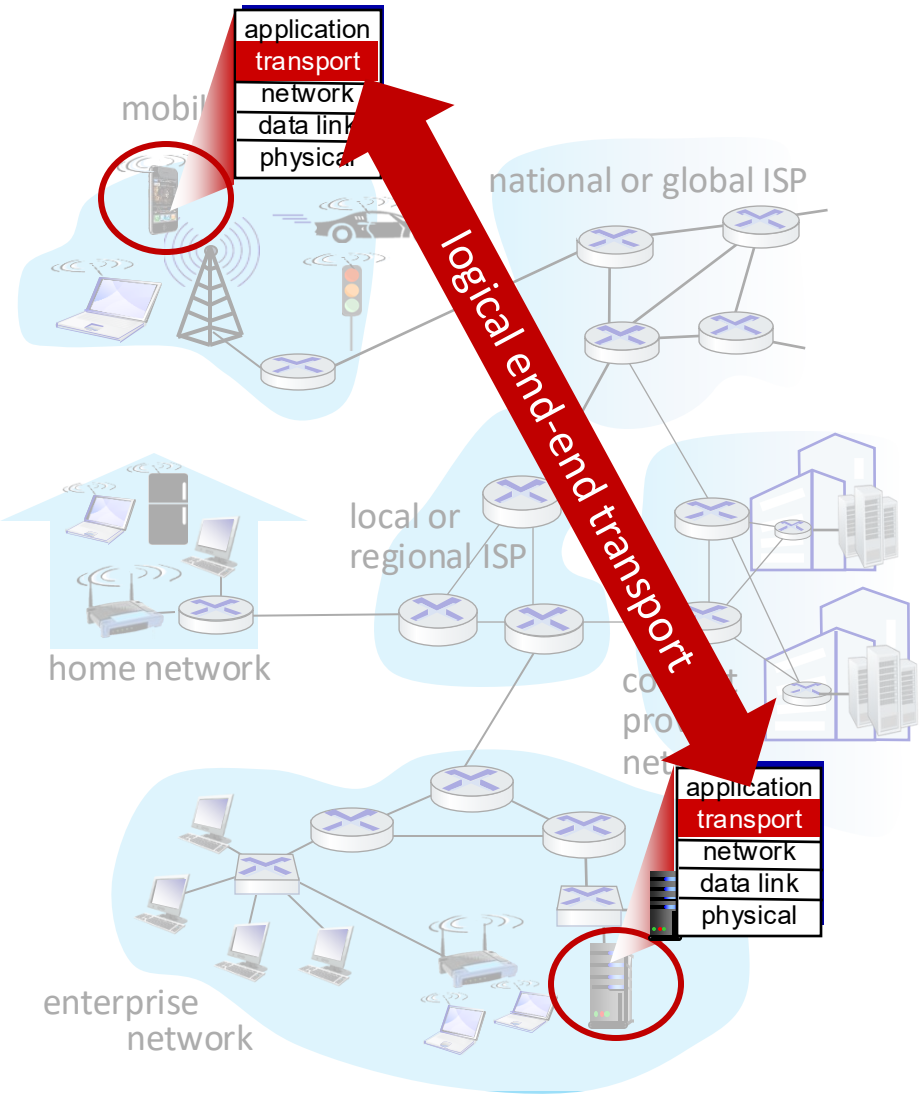
- provide *logical communication* between application processes running on different hosts

→ UDP and TCP → Part of the operating system

① Multiplexing / Demultiplexing (Port #s + IP address)

① Reliability + In-order delivery of data
② Flow control
③ Congestion control

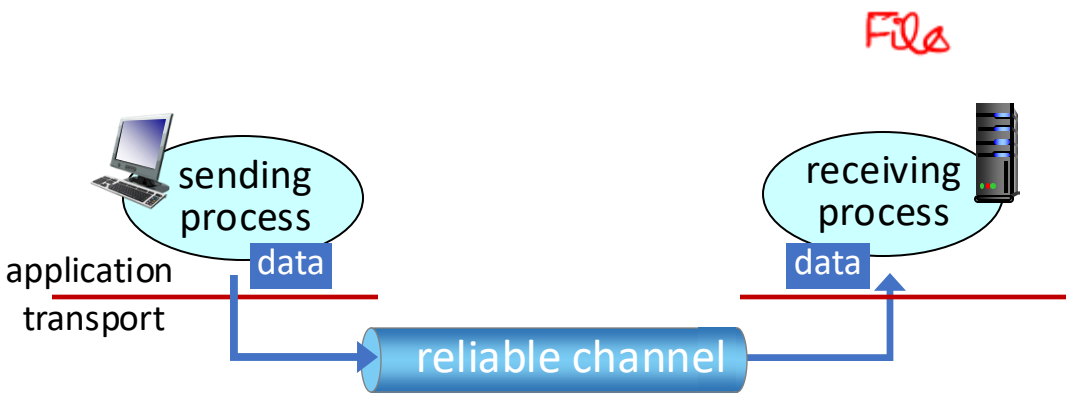
TCP



Transport-layer services

- Multiplexing/demultiplexing
- Reliable delivery
- Flow control
- Congestion control

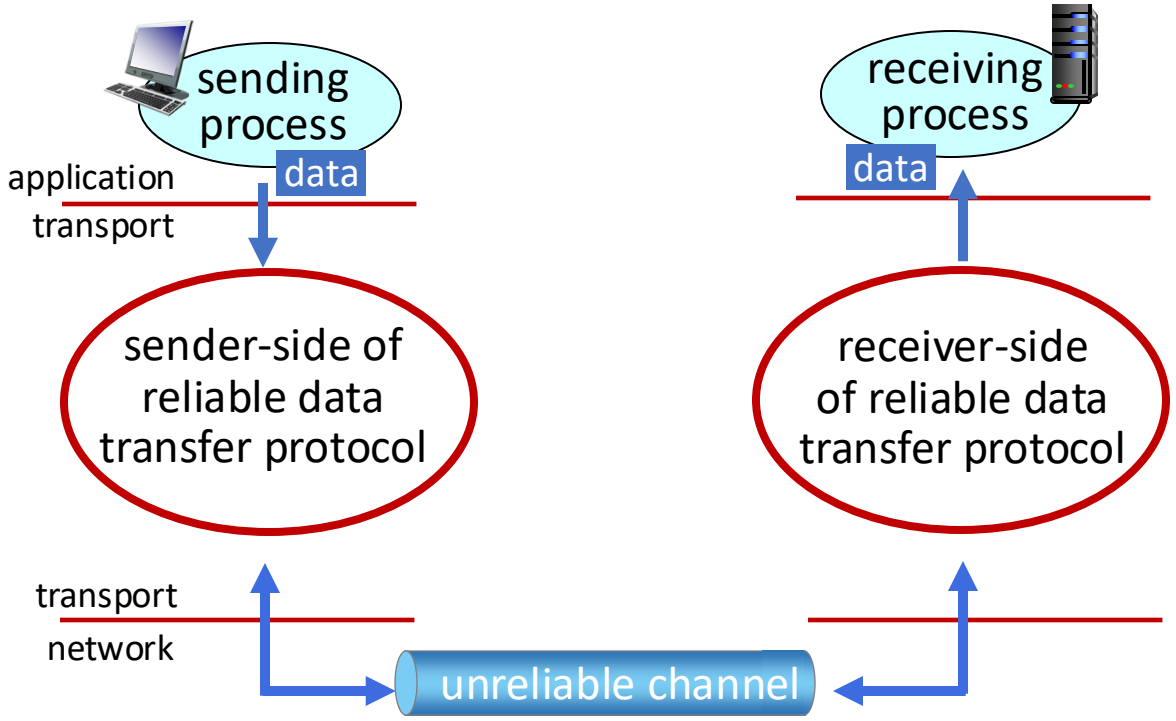
Principles of reliable data transfer



reliable service *abstraction*

Principles of reliable data transfer

reliable service *abstraction*



reliable service *implementation*

Principles of Reliable Data Transfer

[① Use Acknowledgements from the receiver]

[Automatic Repeat Request Protocols (ARQ)]

② Error correction [Fountain coding (Raptor codes)]

(n bytes + r bytes) → OK if you receive k bytes → ① If loss is very high, Error correction is not enough

minimize the

↓ k ≤ n+r

② Efficiency is low

[Forward Error Correction]

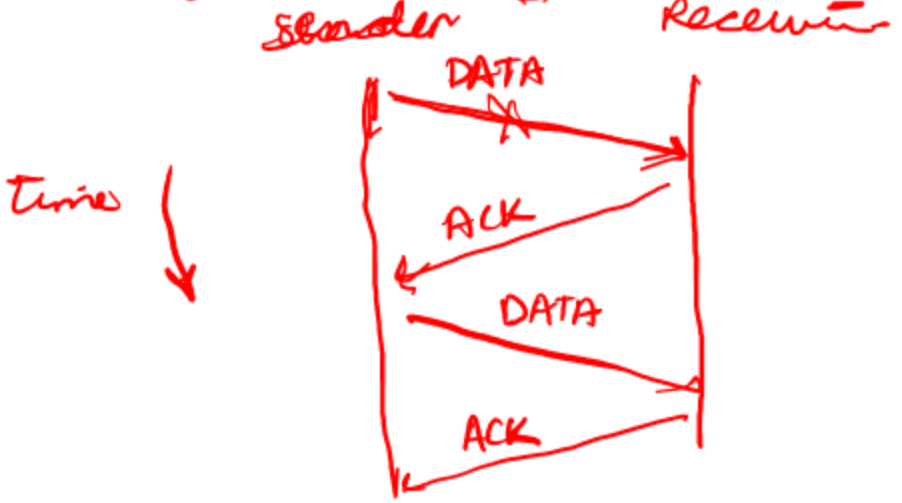
Principles of reliable data transfer

- Forward Error Correction (FEC)

  ① For audio calls → (In appn layer)

  ② When RTTs are very high (Satellite n/w) (In link layer)

- Automatic Repeat Request Protocols (ARQ)



Data bytes can be arbitrary size

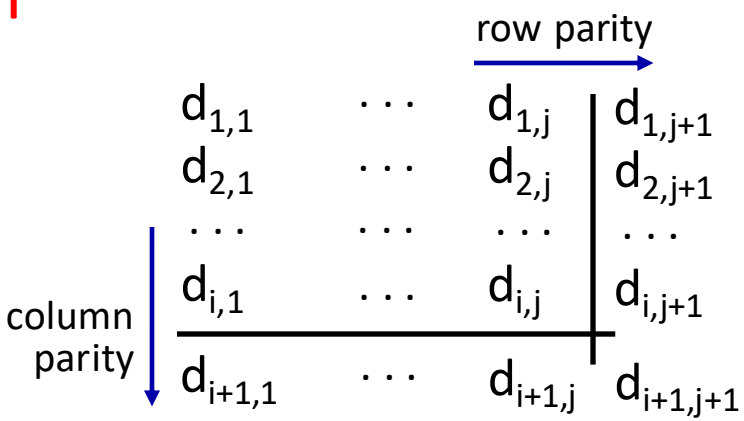
ACKs are very small

Sender: if NO ACK within To, then retransmit

Error correction code

- Also known as Forward Error Correction
- Using 2D parity

  Can detect *and* correct errors (without retransmission!)
  - detect *and correct* single bit errors
- Always useful?
  - When cost of retransmissions are high
  - When there are frequent bit errors



no errors:

1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
1	0	1	0	1	0

detected and correctable single-bit error:

1	0	1	0	1	1
1	0	1	1	0	0
0	1	1	1	0	1
1	0	1	0	1	0

parity error

ARQ Protocol: Stop and Wait

- Transmit one segment, wait for an acknowledgement
  - If no ack and timer expires, resend

① ② ③

② Add seq #s to identify the packet (use 1 bit) & ACKs

③ Receiver ACKs every packet (even if it is duplicate)

[Next seq # sender should transmit]

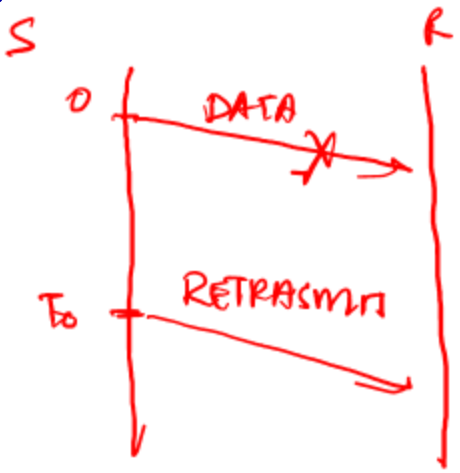
only 1 bit is needed

Identifier (0, 1)

DATA [0]
ACK [1]
DATA [1]
ACK [0]

Q: How to set To ?

CASE-1



CASE-2

DROP the packet, timeout at the sender

CASE-3

If ACK is dropped

seq #s / ID for segments

CASE-4

ACK arrives after To

(Need to ID ACKs as well.)

Stop and Wait: How much should we wait?

$T_0 > \alpha . RTT \qquad \alpha > 1$

if α is very large, performance is an issue [wait a long time before retransmit]

α is very small, retransmissions (redundant/wasted bandwidth)

Stop and Wait: Performance Analysis

Packet size: 1000 bytes

RTT is : 100 msec (Propagation delay)

$$\text{Throughput} = \frac{L}{\text{Time taken}} = \frac{L}{\text{prop delay} + \text{Trans delay}}$$

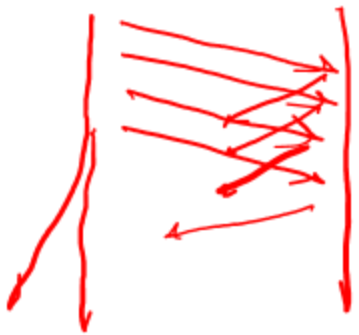
(negligible)

$$= \frac{L}{RTT}$$

$$= \frac{1000 \times 8}{100 \times 10^{-3}} = 80 \text{ kbps}$$

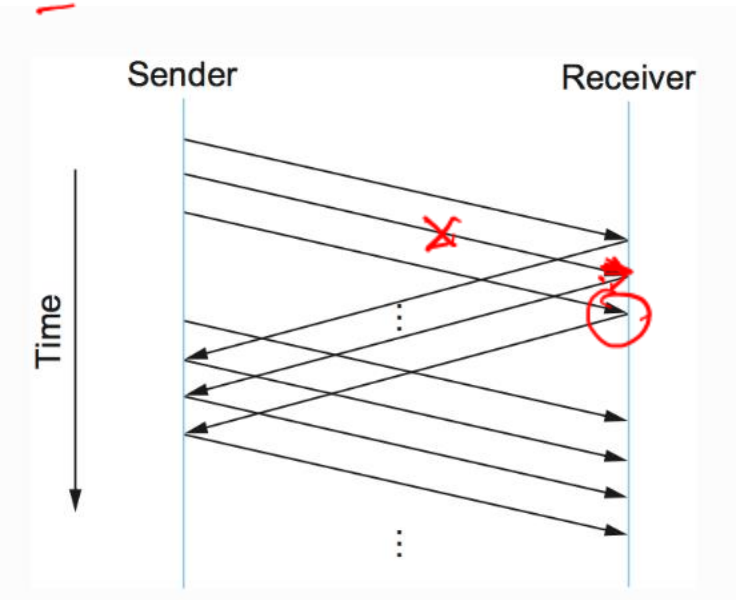
(Utilization is very low)

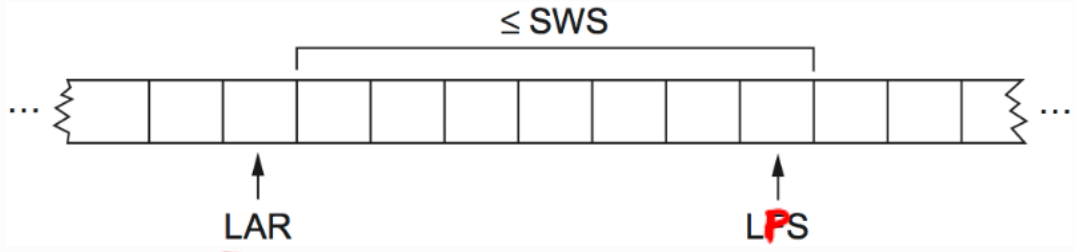
Solution : Allow multiple unacked packet



Pipelining or Sliding Window Protocol

LPS − LAR ≤ SWS





Last Ack received (LAR)

Latest packet sent (LPS)

Two main implications

1. Use more bits (K>1) for seq #s
2. Need to keep buffer at sender & possibly at the receiver