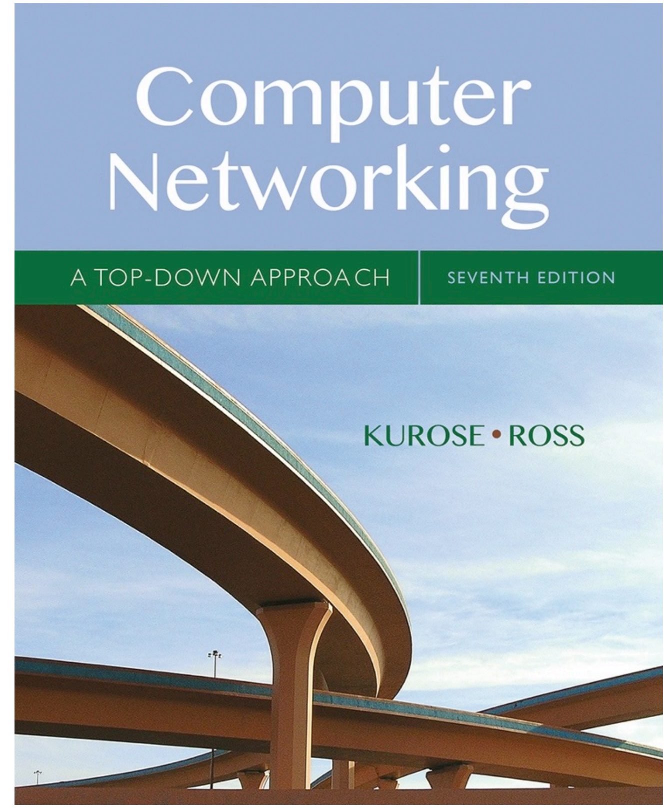
Chapter 3
Transport Layer



Computer Networking: A Top Down Approach

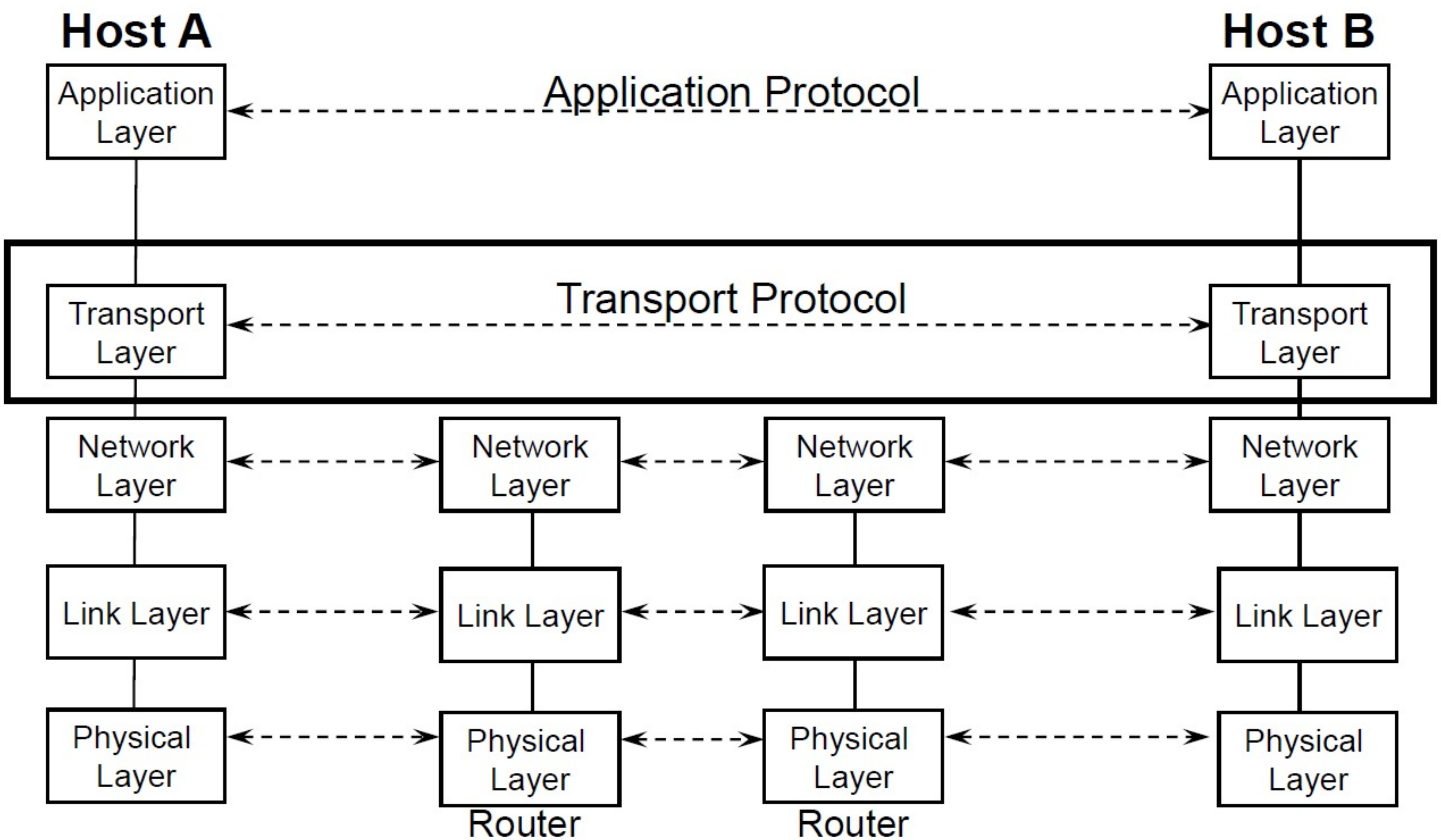
7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

Chapter 3: Transport Layer

our goals:

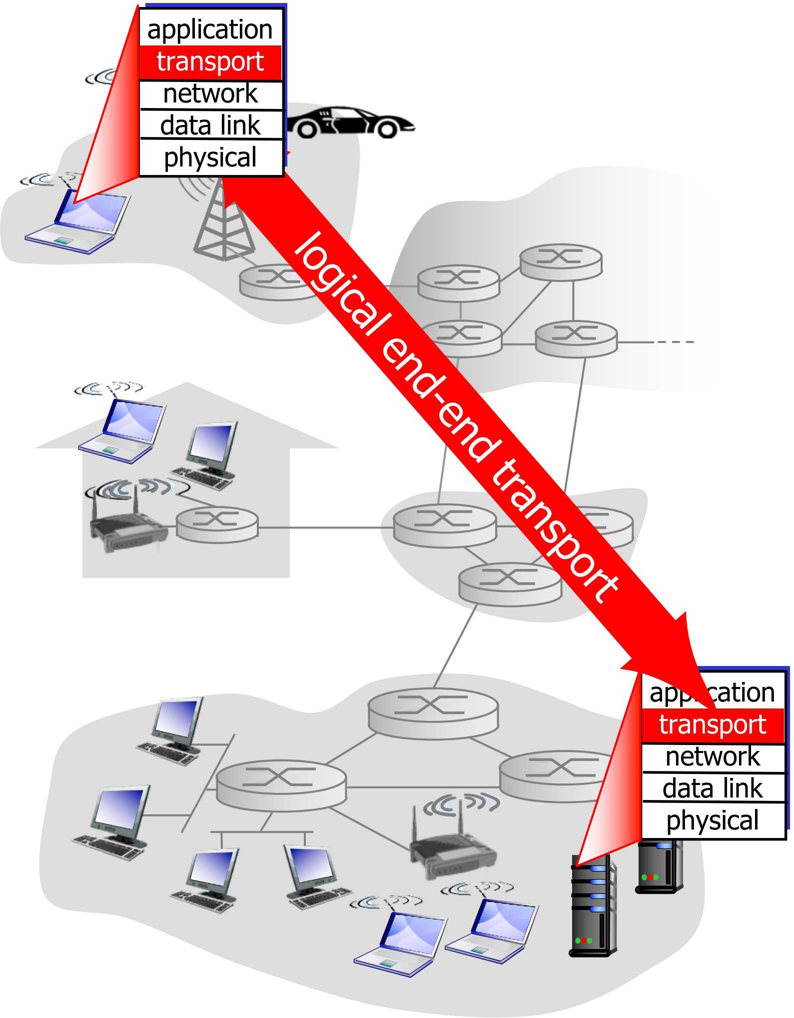
- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - flow control
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport

Internet Protocol Stack



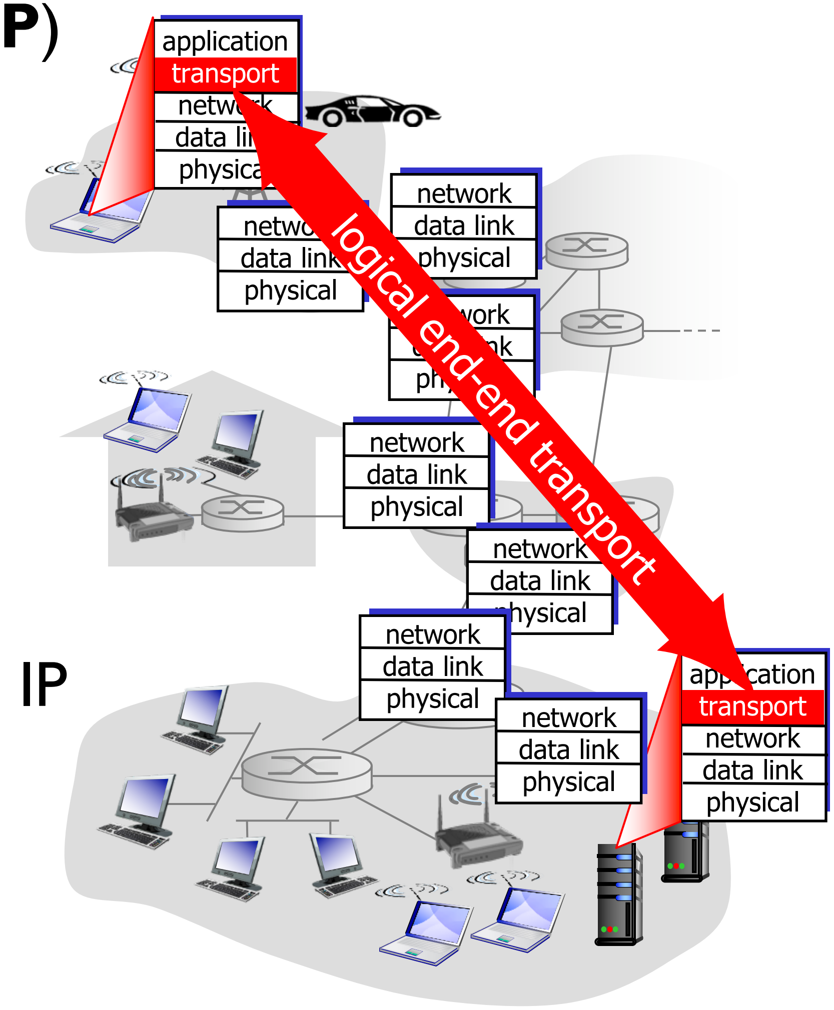
Transport services and protocols

- provide *logical communication* between **app processes** running on **different hosts**
- transport protocols run in end systems
  - **send side:** breaks app messages into *segments*, passes to network layer
  - **rcv side:** reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: **TCP and UDP**



Internet transport-layer protocols

- Transmission Control Protocol (**TCP**)
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- User Datagram Protocol: **UDP**
  - unreliable, unordered delivery
  - Simple extension of "best-effort" IP
- Services not available:
  - delay guarantees
  - bandwidth guarantees



Multiplexing/demultiplexing

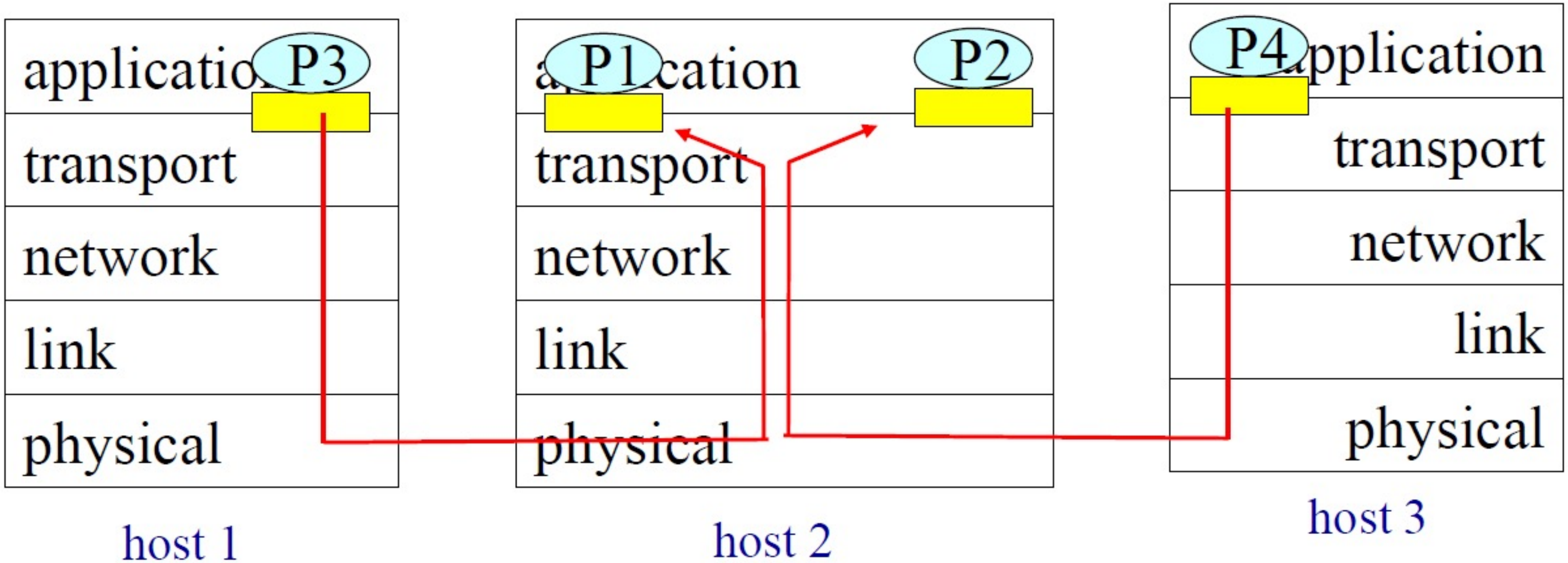
Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

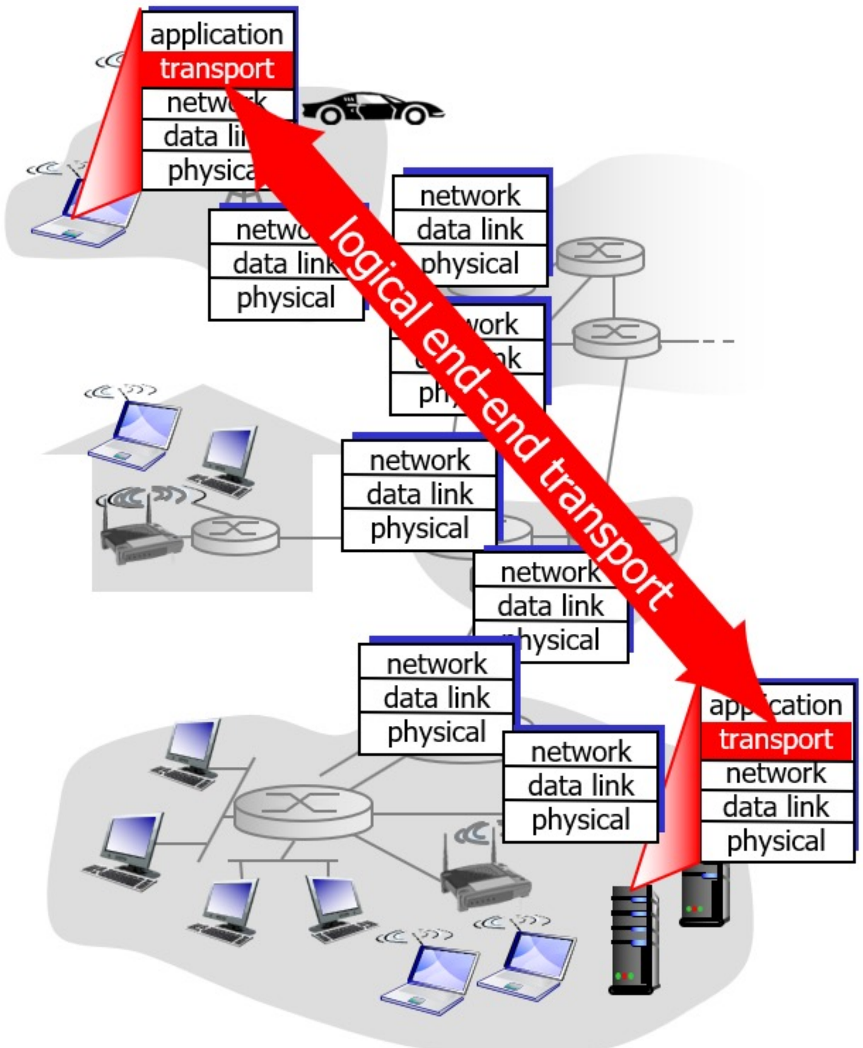
Demultiplexing at rcv host:

delivering received segments to correct socket

= socket   = process

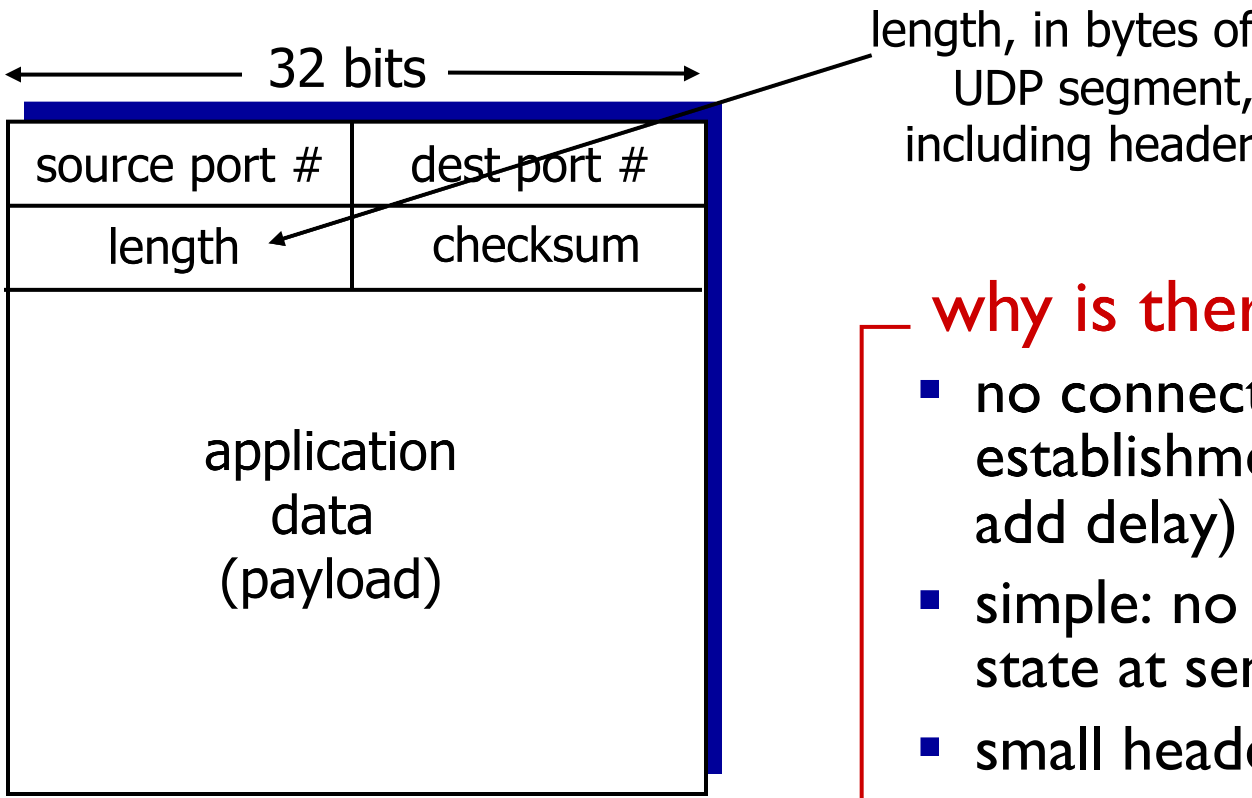


UDP: User Datagram Protocol [RFC 768]



- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

UDP: segment header



UDP segment format

why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

Reliable Data Transfer

- Problem: Reliability
  - A reliable link even though packets can be corrupted or get lost
- Where can packets be corrupted or lost?
  - In the network
  - At the receiver
- Solution: keep track of the packets
  - Not as simple as one would expect
  - Reliable Transmission & Flow Control

Reliable Transmission & Flow Control

- What to do when there is a packet loss?
  - On the link: Retransmit
  - At the receiver: Flow Control

- Sender needs to know if a packet was lost, How?
  - ACK (Positive & Negative)

- Sender needs to retransmit, How?
  - Timeouts

Some Flow Control Algorithms

1. Flow control for the ideal network
2. Stop and Wait for noiseless channels
3. Stop and Wait for noisy channels
4. Sliding window protocols
5. Sliding window with error control
   - Go Back N
   - Selective Repeat

1. Flow control in the ideal network

Assumptions:

(1) Error free transmission link,

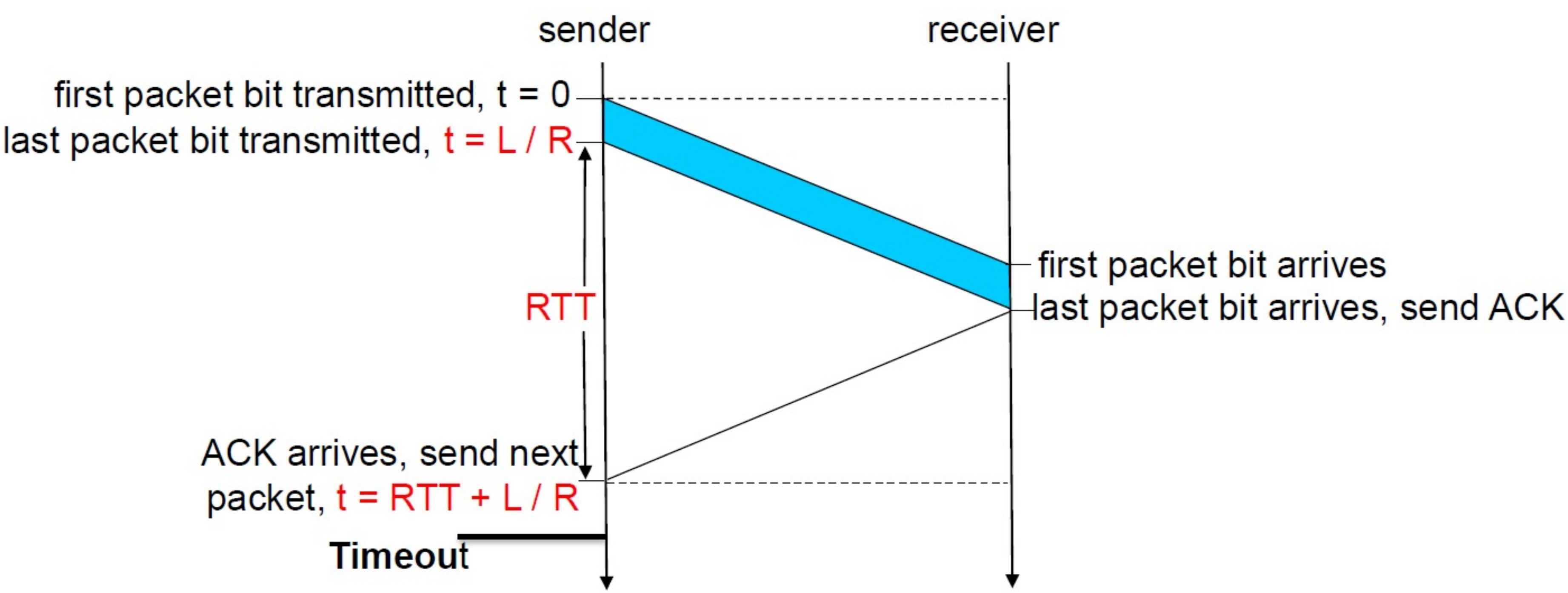
(2) Infinite buffer at the receiver

No acknowledgement necessary

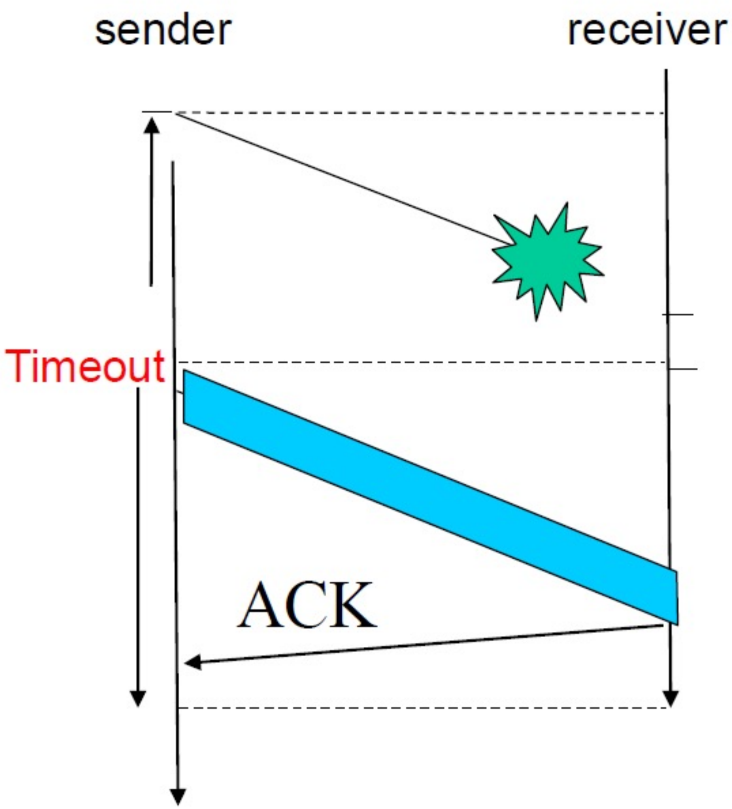
Since the data link is error-free and the receiver can buffer as many packet as it likes, no packet will ever be lost

2. Stop-and-Wait Noiseless Channel

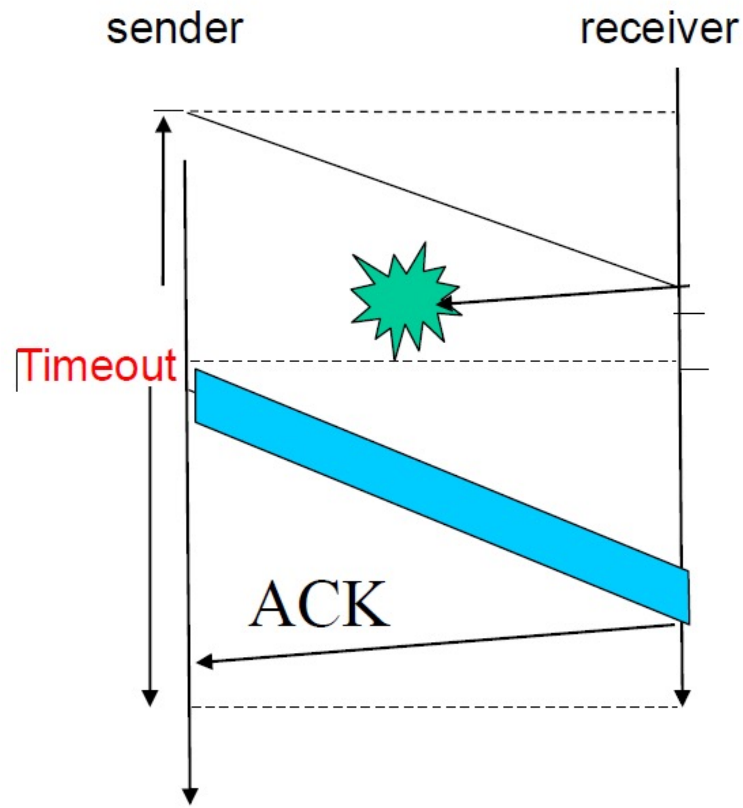
Packet Length = L; Bandwidth =R; RTT = 2*Prop Delay



3. Stop-and-Wait Noisy Channel



Packet retransmitted



Packet retransmitted

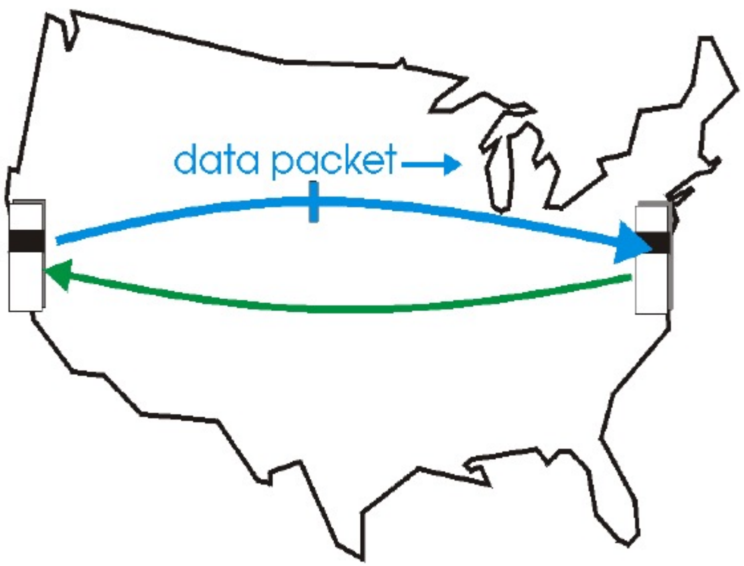
Is Stop and Wait the best we can do?

Stop and Wait is an effective form of flow control, but…

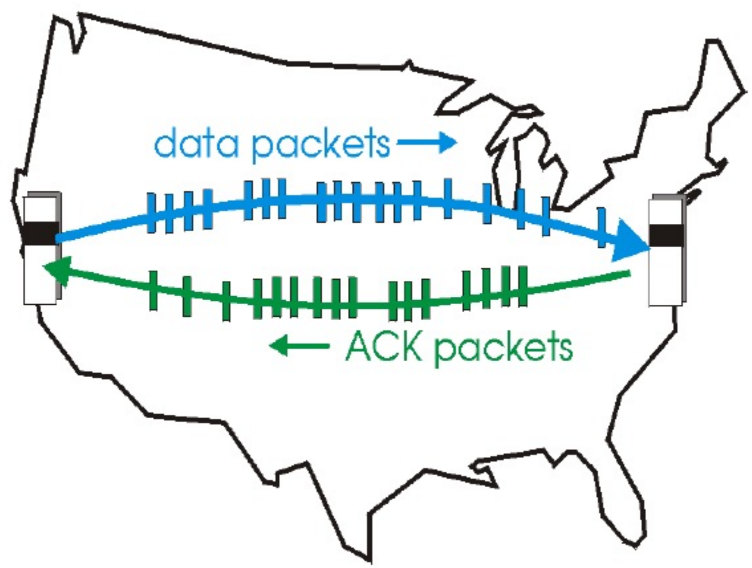
It's not very efficient.

1. Only one data frame can be in transit on the link at a time
2. When waiting for an acknowledgement, the sender cannot transmit any frames

Better solution? Pipelined Protocol : Sliding Window

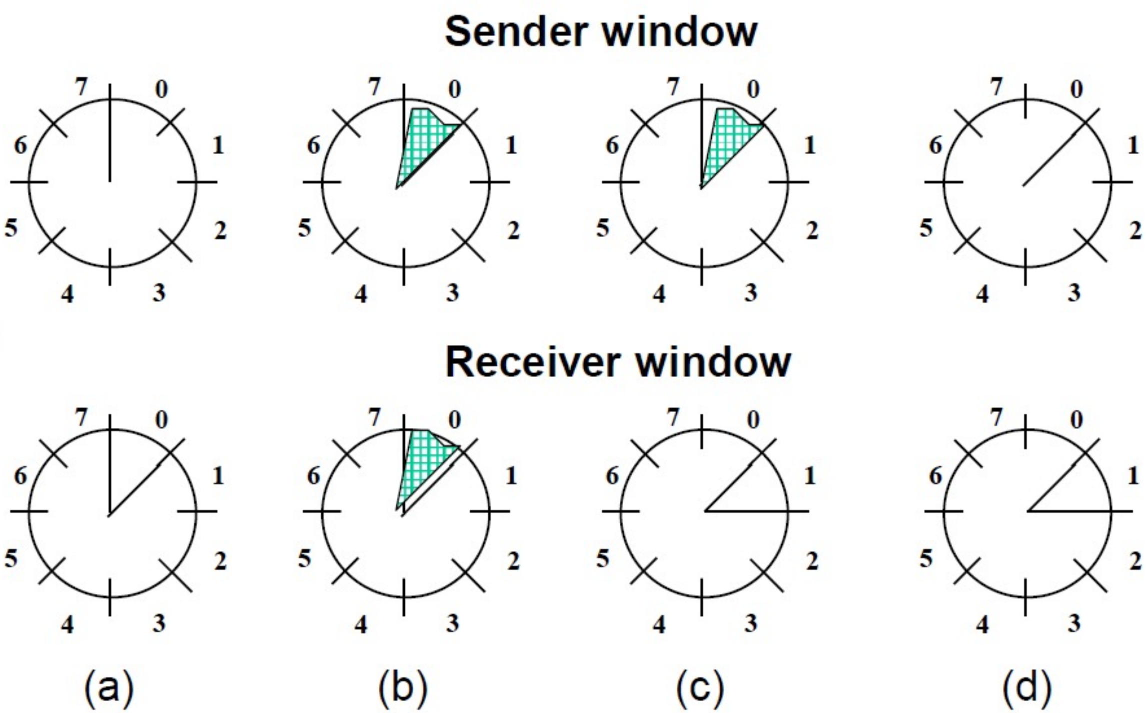


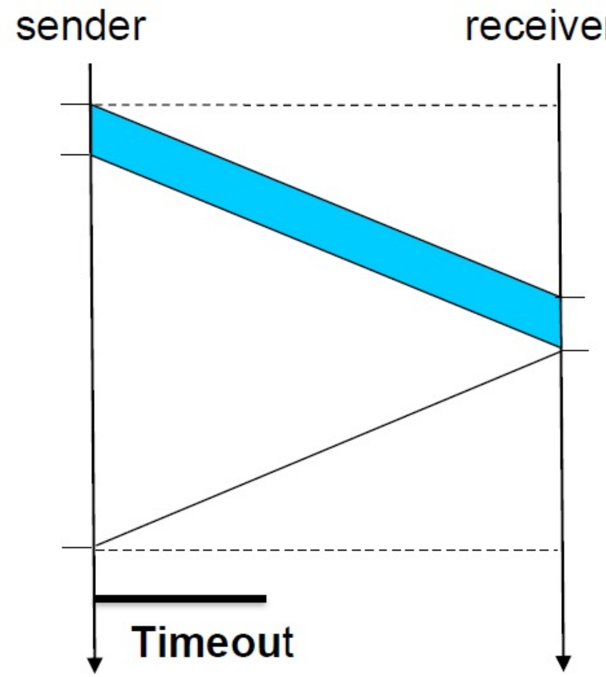
(a) a stop-and-wait protocol in operation



(b) a pipelined protocol in operation

Sliding Window example





(a) Initial state, no frames transmitted, receiver expects frame 0
(b) Sender transmits frame 0, receiver buffers frame 0
(c) Receiver ACKS frame 0
(d) Sender receives ACK, removes frame 0

This protocol behaves identically to stop and wait for a noisy channel

Sliding Window with Maximum Sender Window Size SWS

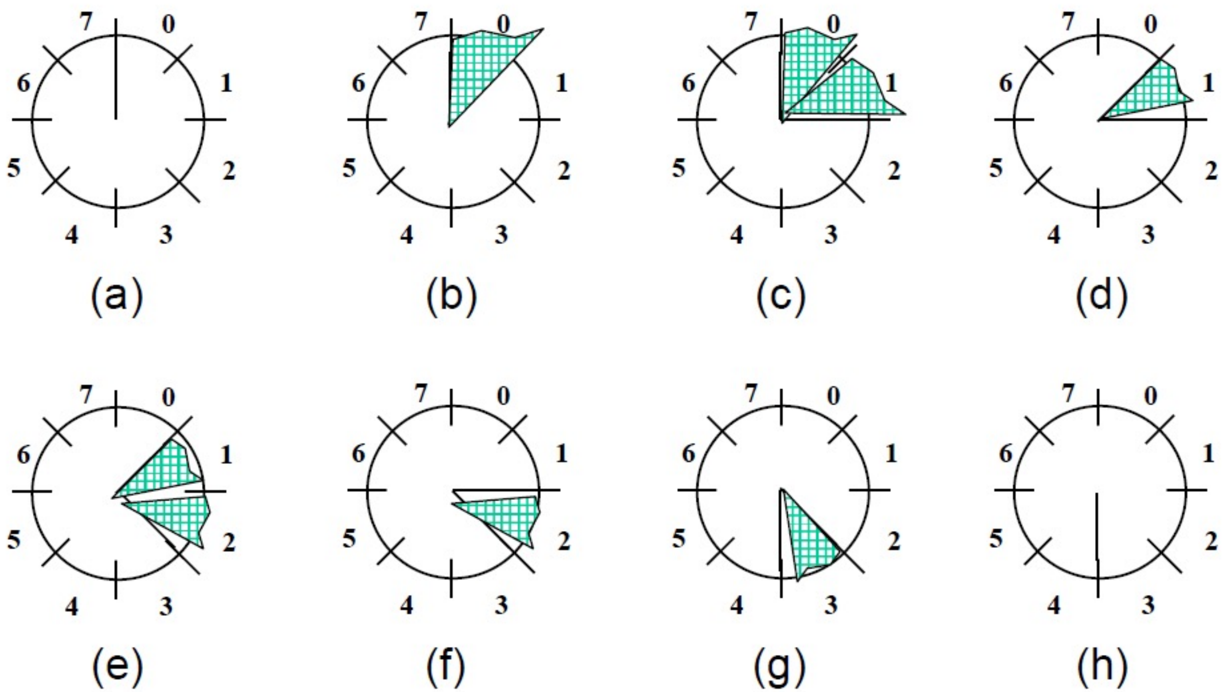
Sender Window size: The maximum number of frames the sender may transmit without receiving any acknowledgements

With a maximum window size of *SWS*, the sender can transmit up to *SWS* frames before "being blocked"

This allows the sender to transmit several frames before waiting for an acknowledgement

LFS - LAR <= SWS

Sender-Side Window with W_S=2



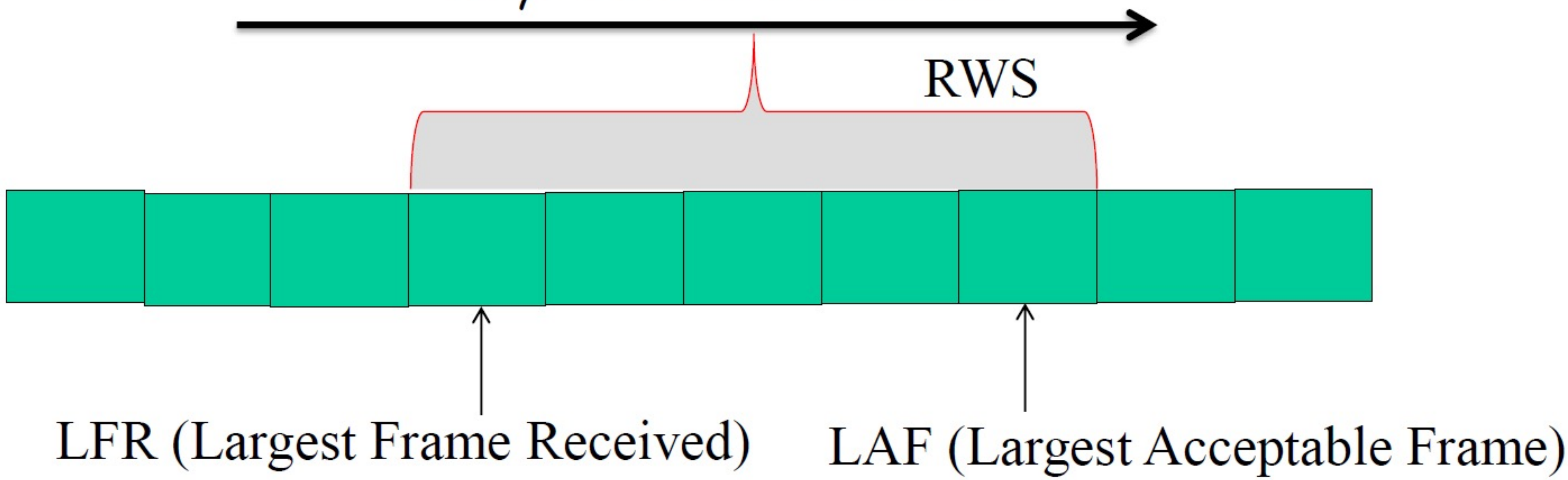
(a) Initial window state	(e) Send frame 2
(b) Send frame 0	(f) ACK for frame 1 arrives
(c) Send frame 1	(g) ACK for frame 2 arrives, send frame 3
(d) ACK for frame 0 arrives	(h) ACK for frame 3 arrives

Sliding Window with Maximum Receiver Window Size

Receiver Window size: The maximum number of frames the receiver may receive before returning an acknowledgement to the sender

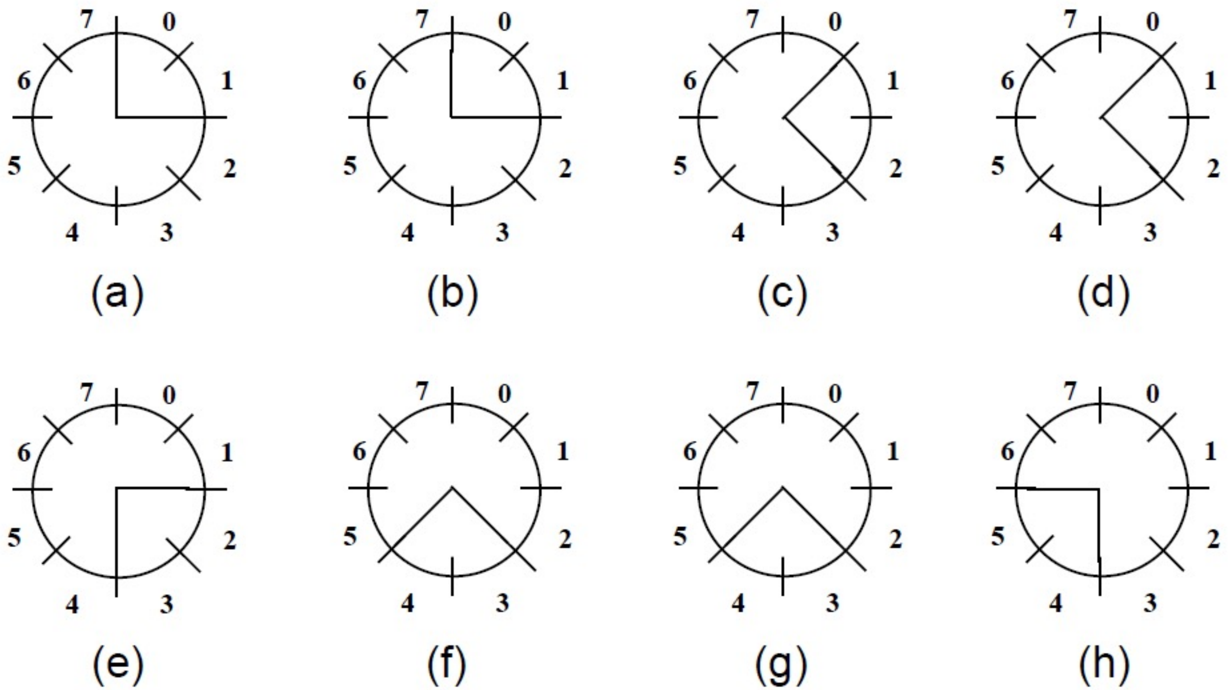
With a maximum window size of RWS, the receiver rejects packets if SeqNum <= LFR or SeqNum > LAF

Why? Outside the window



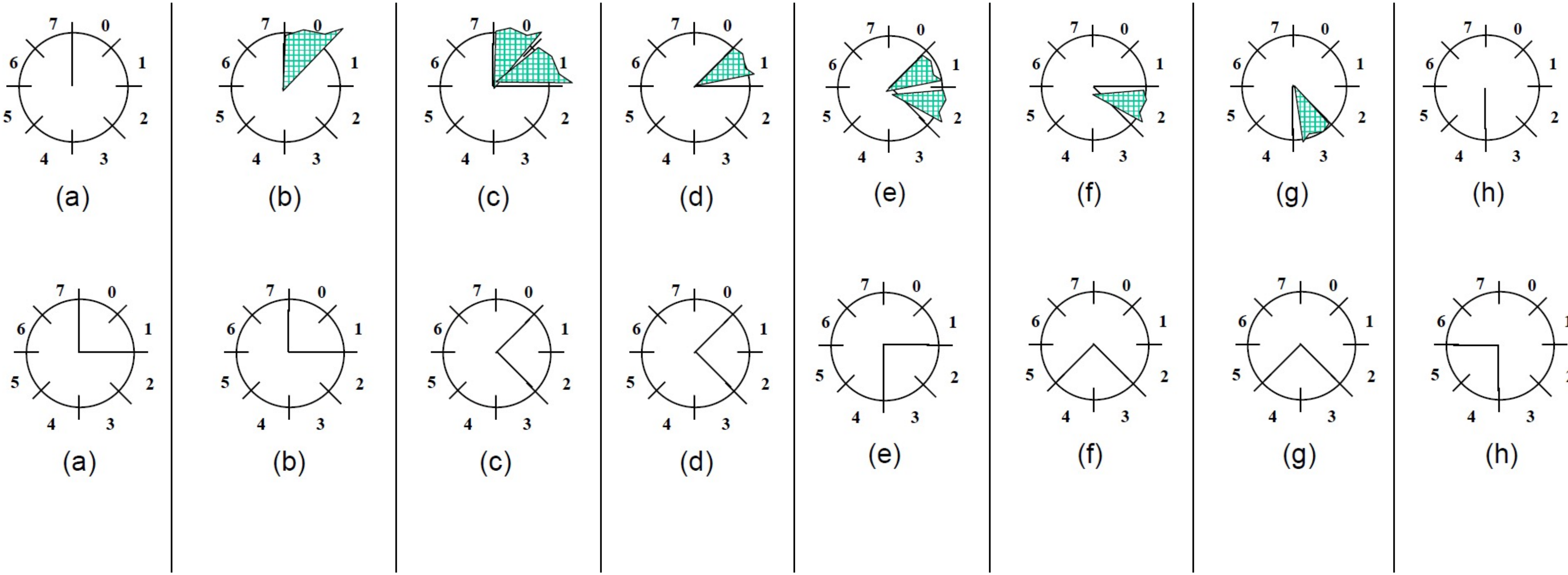
LAF - LFR <= RWS

Receiver-Side Window with W_R=2



(a) Initial window state	(e) Frame 1 arrives, ACK frame 1
(b) Nothing happens	(f) Frame 2 arrives, ACK frame 2
(c) Frame 0 arrives, ACK frame 0	(g) Nothing happens
(d) Nothing happens	(h) Frame 3 arrives, ACK frame 3

Sender-Side Window with W_S=2



Receiver-Side Window with W_R=2