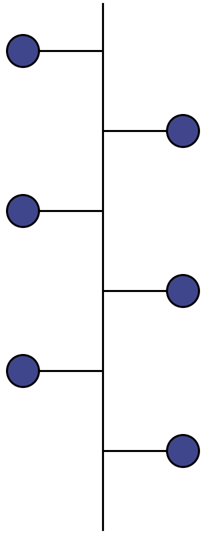
Introduction to Communication Networks and Distributed Systems



Unit 12: Transport Layer

Prof. Dr.-Ing. Adam Wolisz

TKN Telecommunication Networks Group

9. Transport layer (Layer 4)

- Overview

  9.1 Ports

  9.2 User Datagram Protocol (UDP)

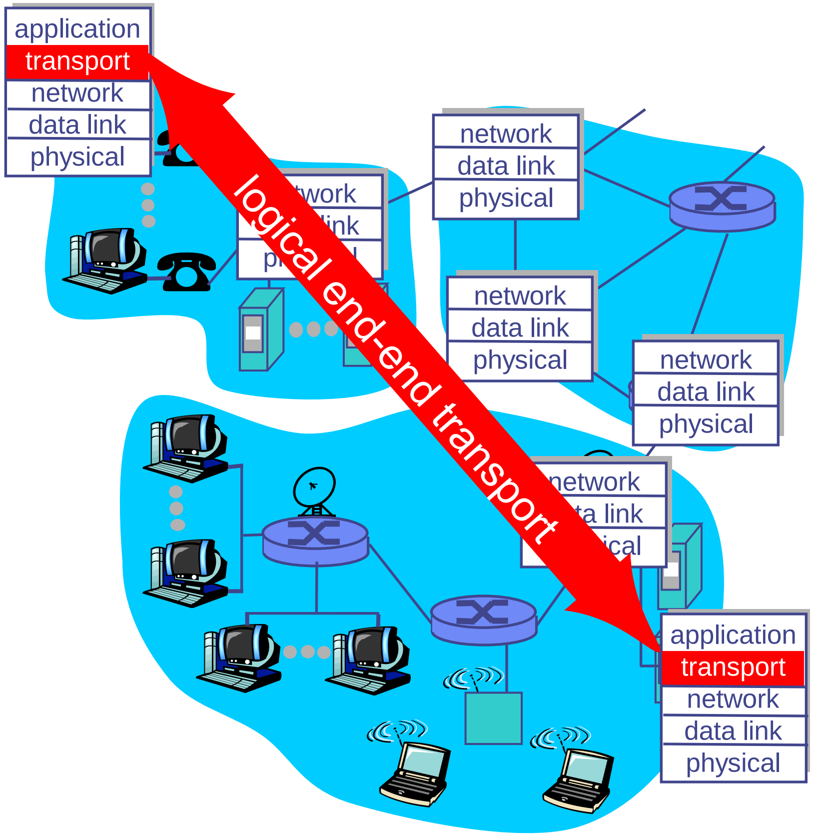
  9.3 Transmission Control Protocol (TCP)

  9.4 Flow Control

  9.5 The Problem of Congestion

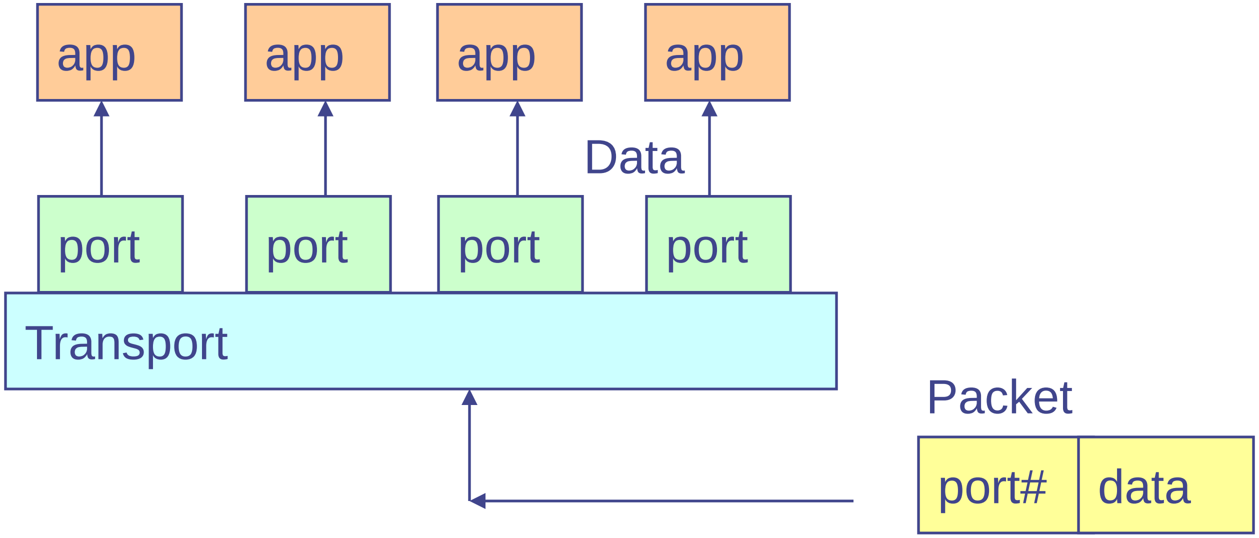
Key features: Transport services and protocols

- Provide logical communication between application processes running on different hosts
  - ⇒Transport protocols run in end systems
- Transport vs. network layer services
  - network layer = transfer between end systems
  - transport layer = transfer between processes
- Transport protocols TCP and UDP
  - Processes as sender / receiver, ports for addressing processes in the system
  - Quality of service: if quality requirement is not met, the communication fails



5.1 Understanding Ports [Buya]

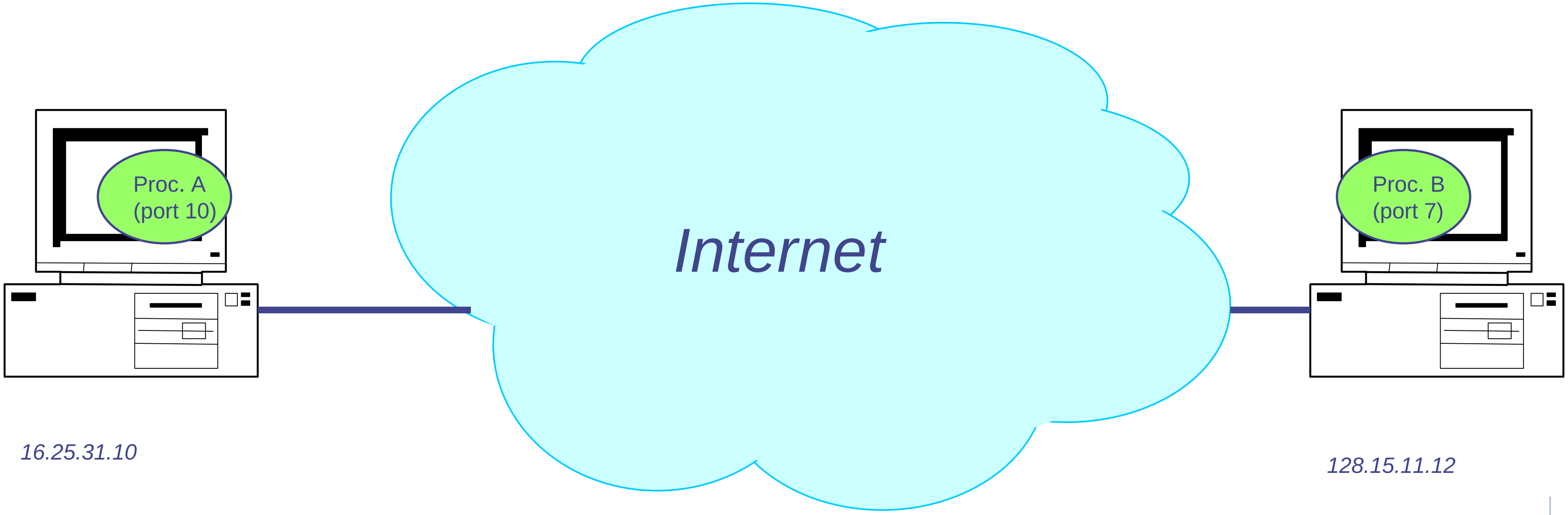
- Port is represented by a positive (16-bit) integer value
- Some ports have been reserved to support common/well known services: http 80/tcp; ftp 21/tcp; telnet 23/tcp; smtp 25/tcp;
- User level process/services generally use port number value >= 1024



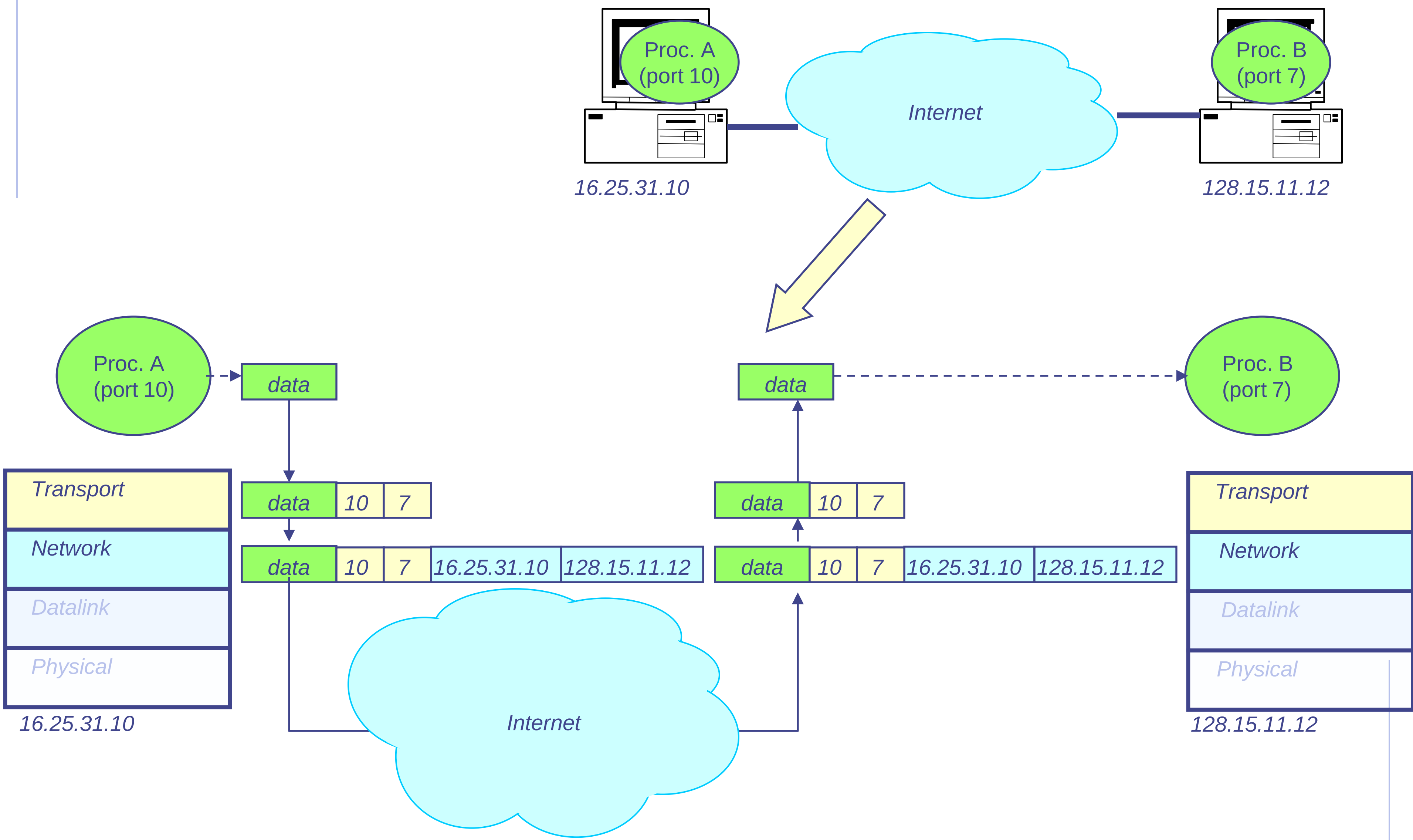
Internet End-to-End View

[Stoica]

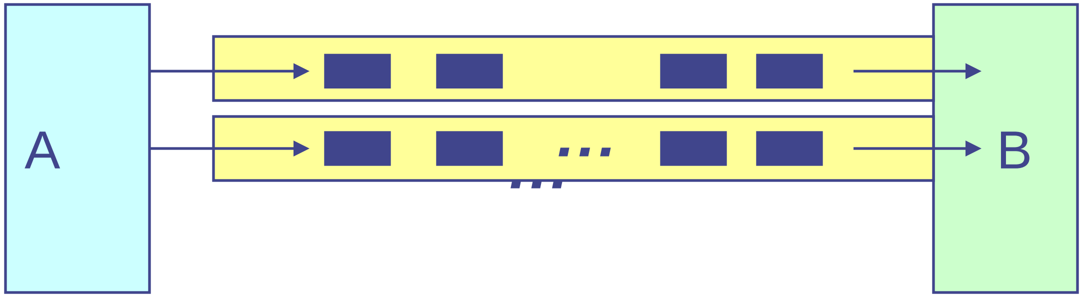
- Process A sends a packet to process B



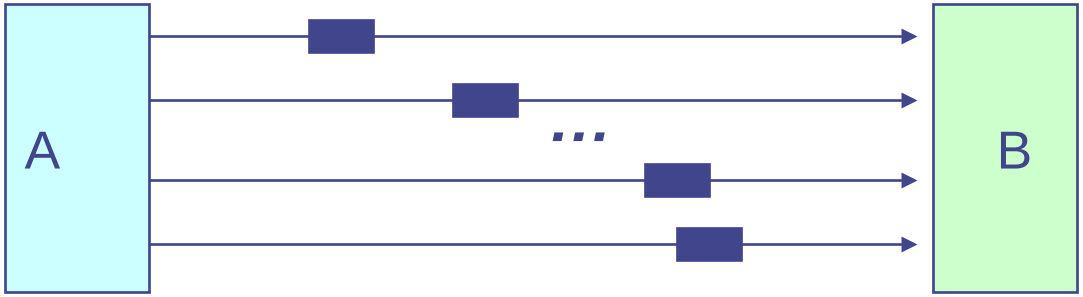
End-to-End Layering View



Connection-Oriented vs. Connection-less Communication



- Connection-Oriented communication with connect, data exchange, release connection $\Rightarrow$ service requires preliminary setup phase, e.g., to determine receiver



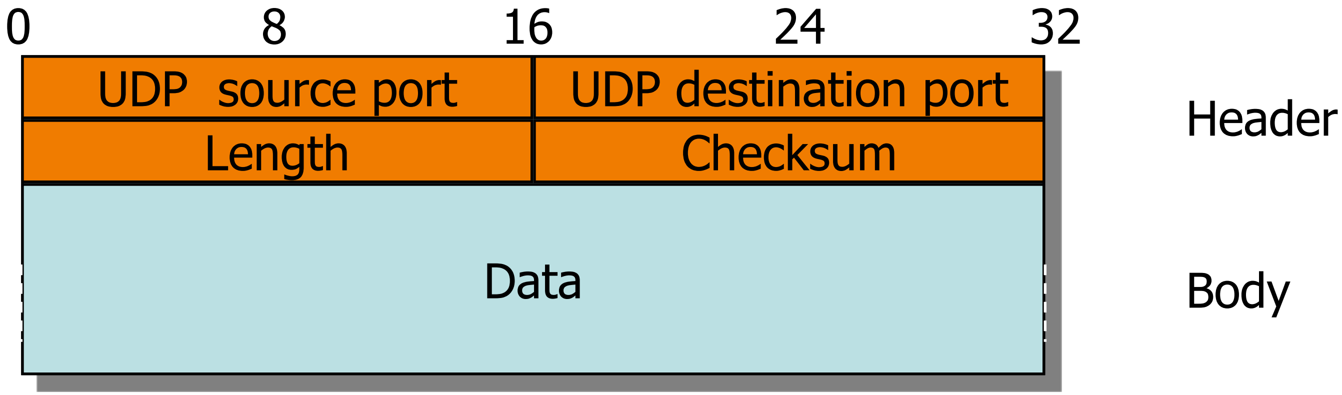
- Connection-less communication: Invocation of a service primitive can happen at any time, with all necessary information provided in the invocation

5.2 User Datagram Protocol (UDP) Characteristics

- UDP is a connection-less datagram service
  - ⇒There is no connection establishment
  - ⇒Packets may show up at any time
- UDP packets are self-contained
- UDP is unreliable
  - No acknowledgements to indicate delivery of data
  - Checksums cover the header, and only optionally cover the data
  - Contains no mechanism to detect missing or mis-sequenced packets
  - No mechanism for automatic retransmission
  - No mechanism for flow control, and so can over-run the receiver

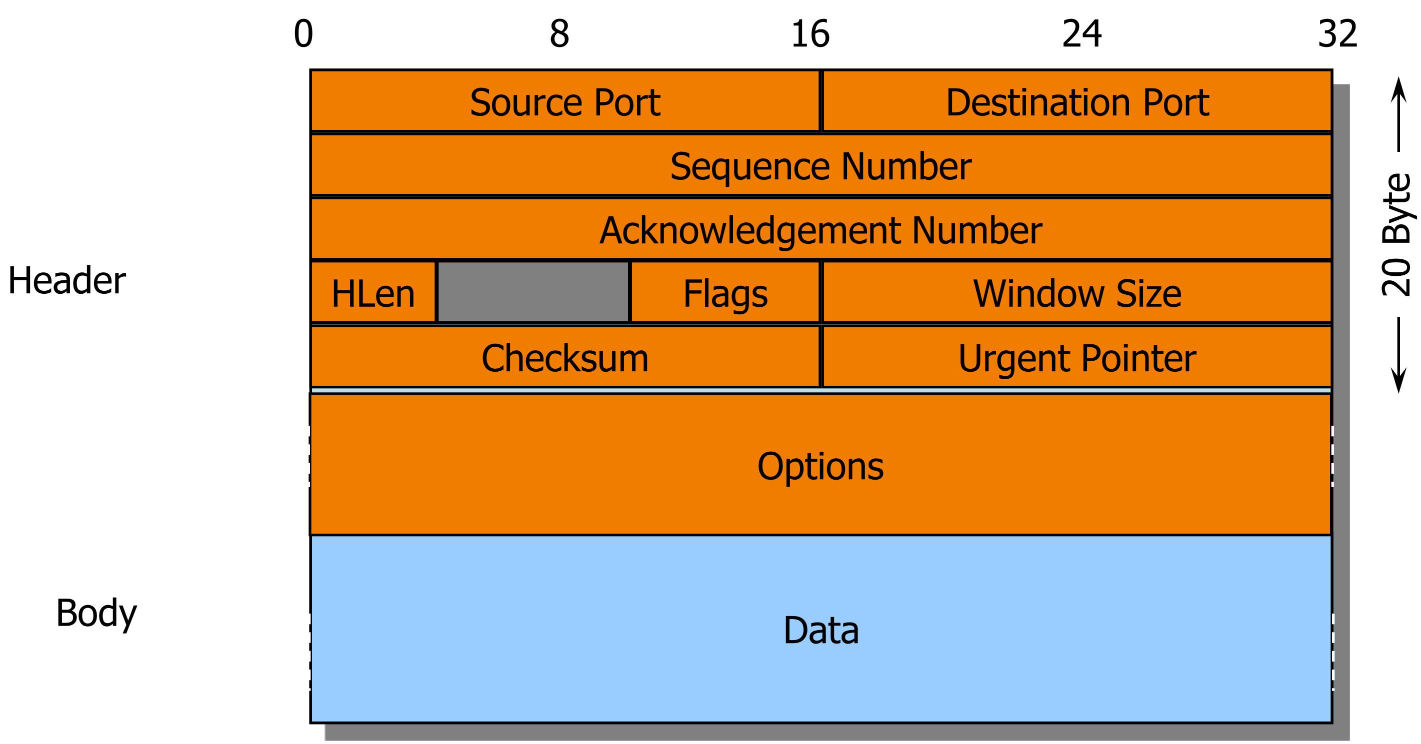
User-Datagram Protocol (UDP) Packet format

- Why do we have UDP?
  - It is used by applications that don't need reliable delivery, or
  - Applications that have their own special needs, such as streaming of real-time audio/video

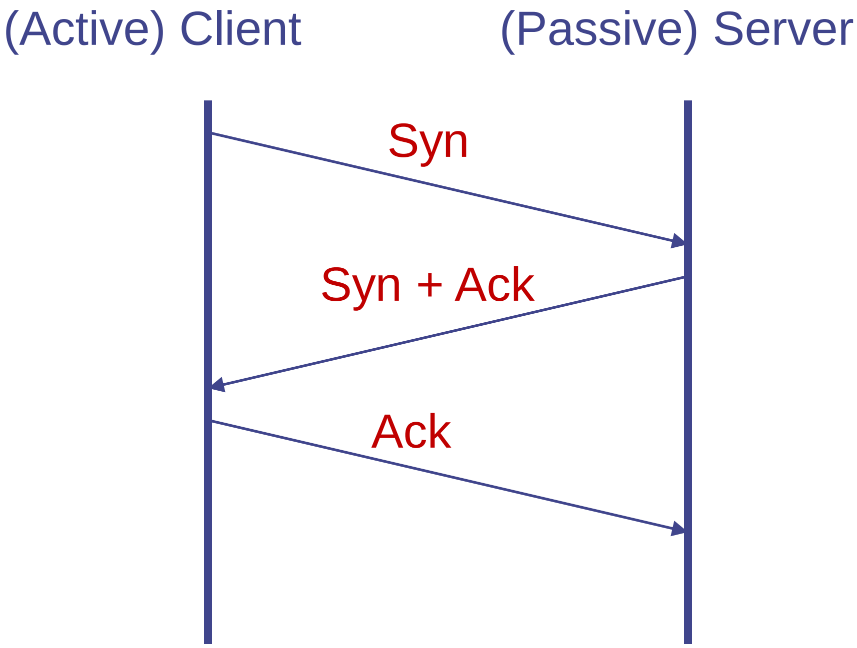


5.3 Transmission Control Protocol (TCP)

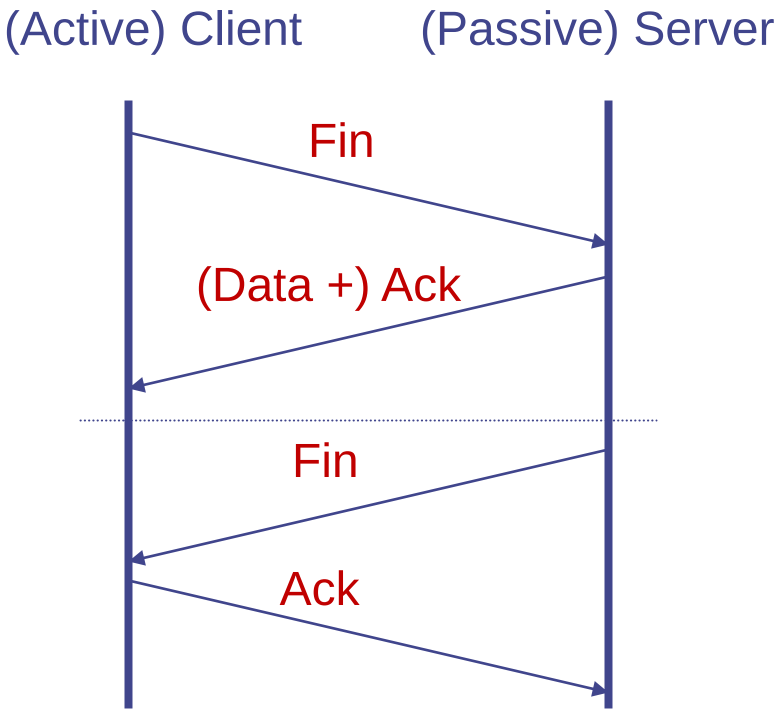
- TCP provides a stream-of-bytes service
- TCP is connection-oriented: 3-way handshake for connection setup
- TCP is reliable!
- Flow control prevents over-run of receiver
- TCP uses congestion control to share network capacity among users



TCP is connection-oriented

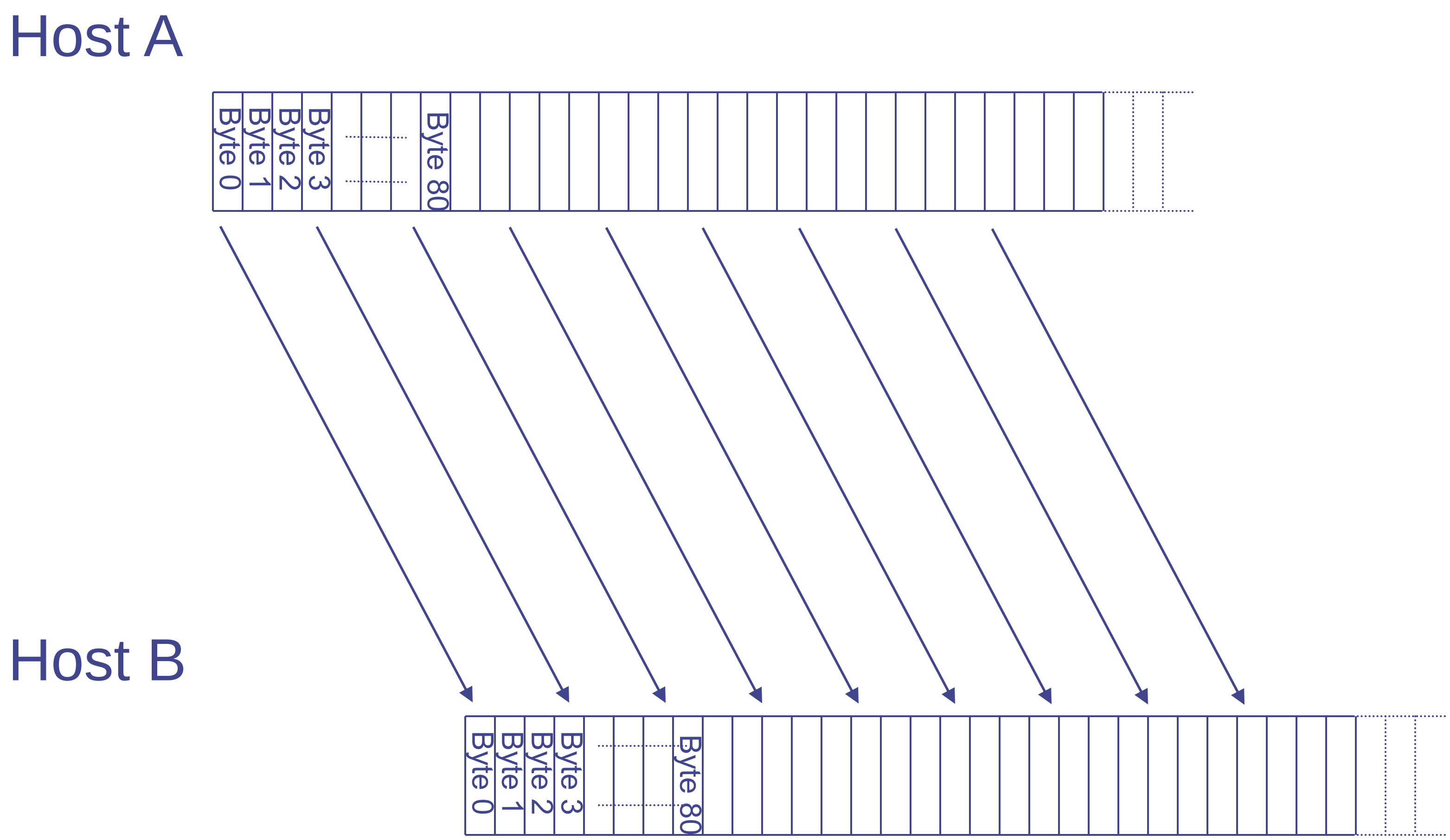


Connection Setup
3-way handshake

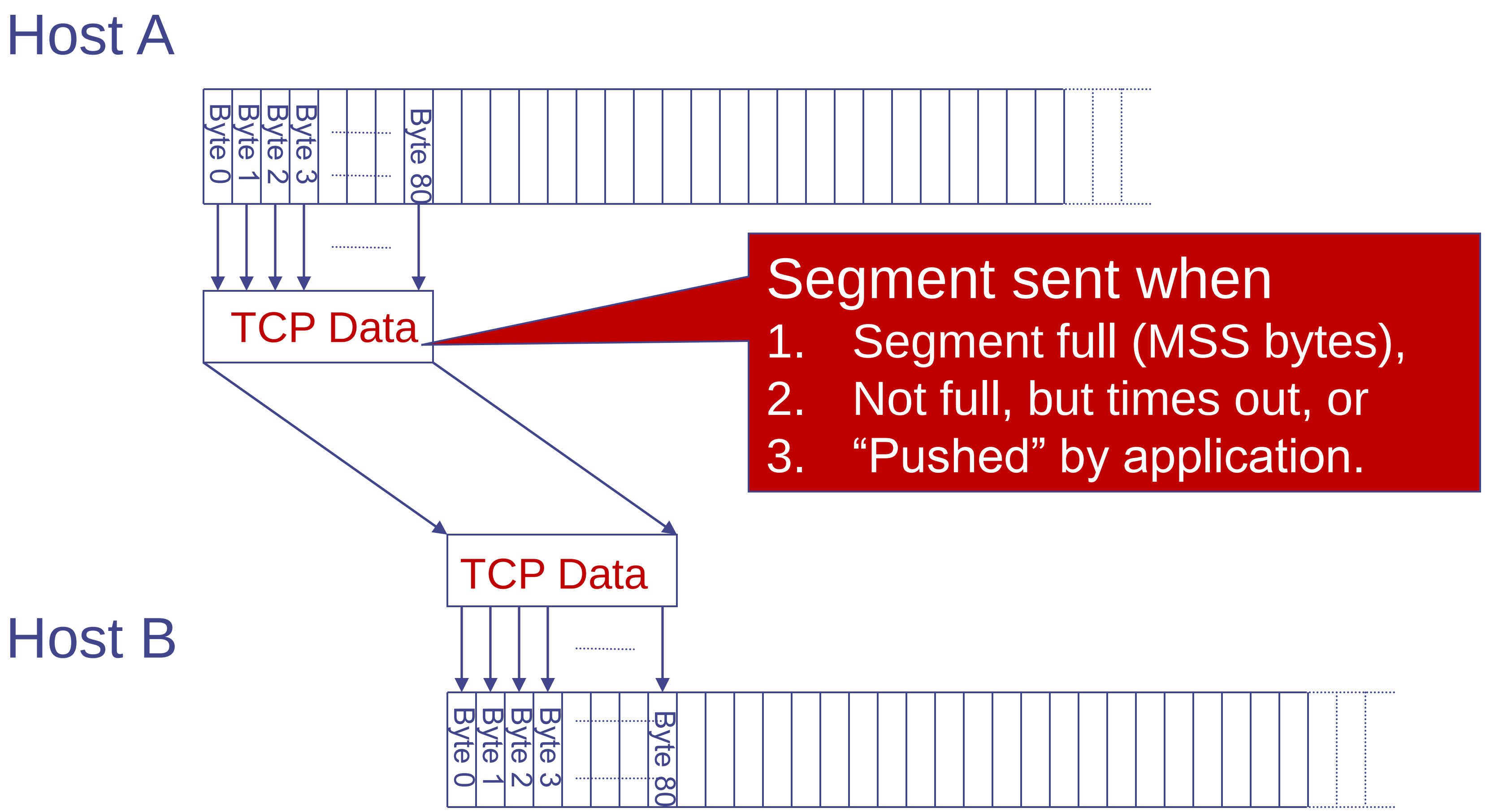


Connection Close/Teardown
2 x 2-way handshake

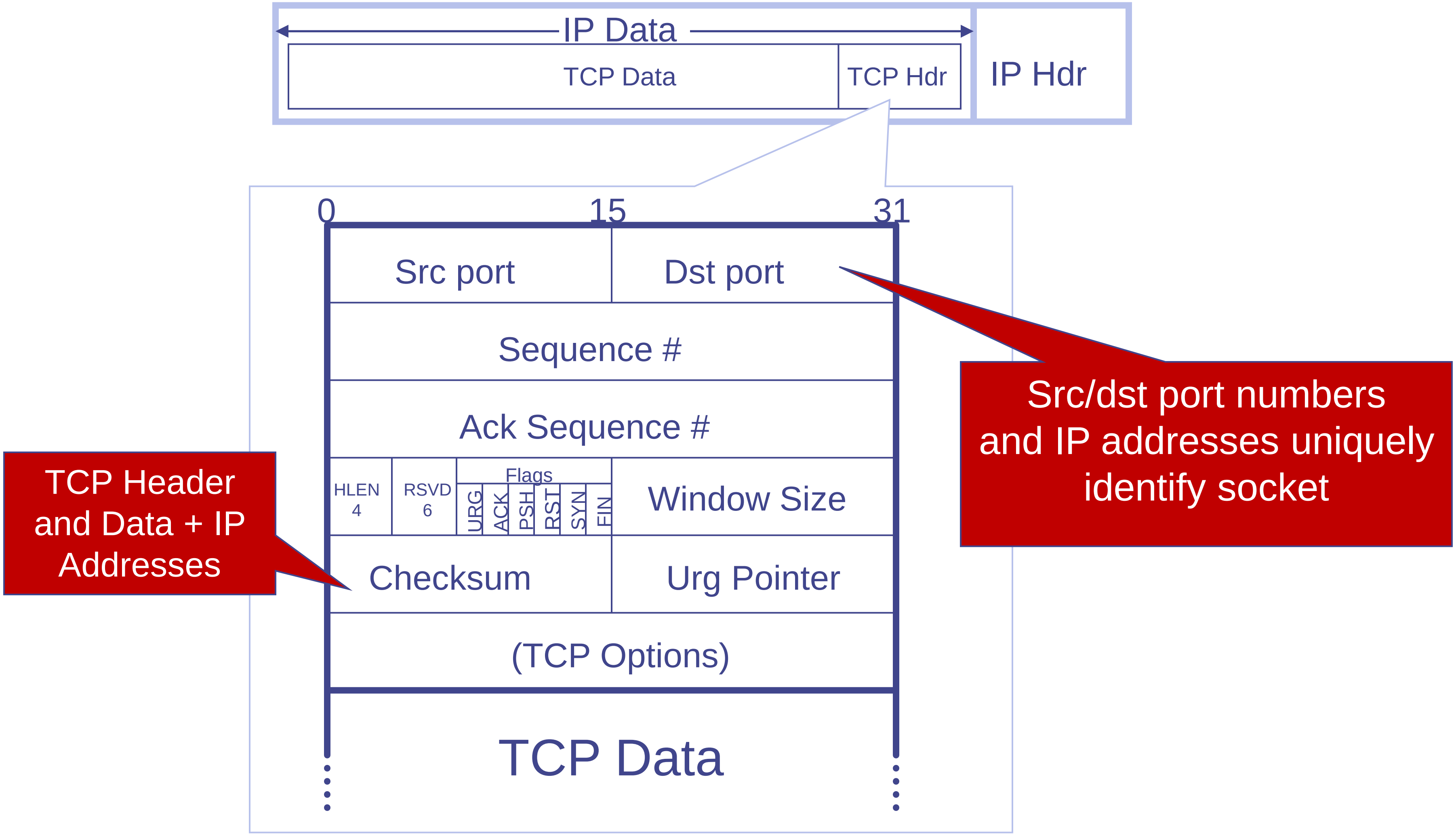
TCP supports a “stream of bytes” service



…which is emulated using TCP "segments"



The TCP Segment Format



What Problems Reliable Transport Solution Try to Solve?

- Best effort network layer
  - Packets can get corrupted
  - Packets can get lost
  - Packets can get re-ordered

Mechanisms used in Reliable Transport

- Packets can get corrupted
  - CRC or Checksum to detect, retransmission to recover
  - Error correction code to recover

- Packets can get lost
  - Acknowledgement + Timeout to detect, retransmission to recover

- Packets can get re-ordered
  - Sequence number to detect, receiver buffer to re-order

Automatic Repeat Request (ARQ) Algorithms

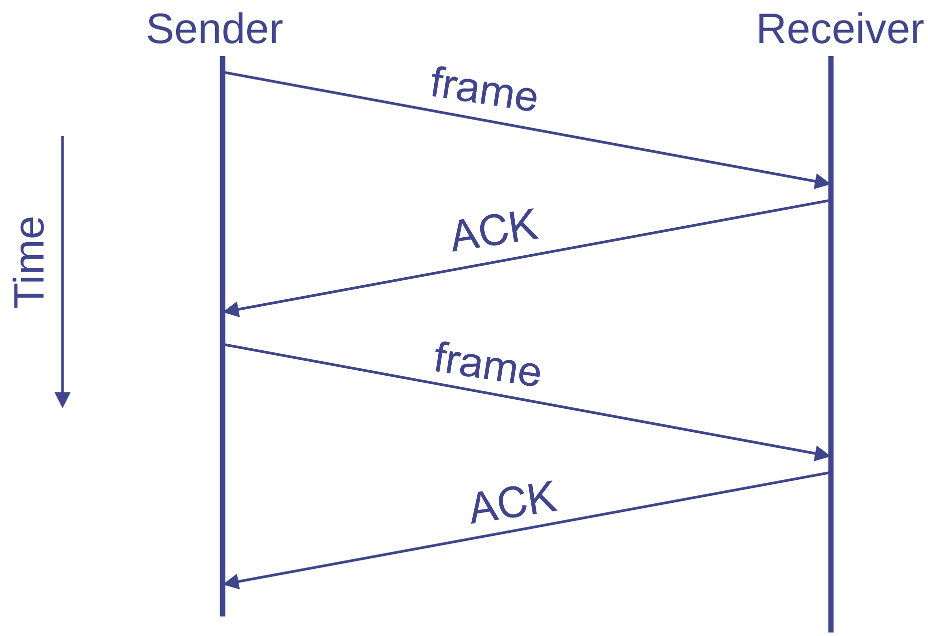
- Used in DIFFERENT LAYERS, not in Transport Layer only!

- Use two basic techniques
  - Acknowledgements (ACKs)
  - Timeouts

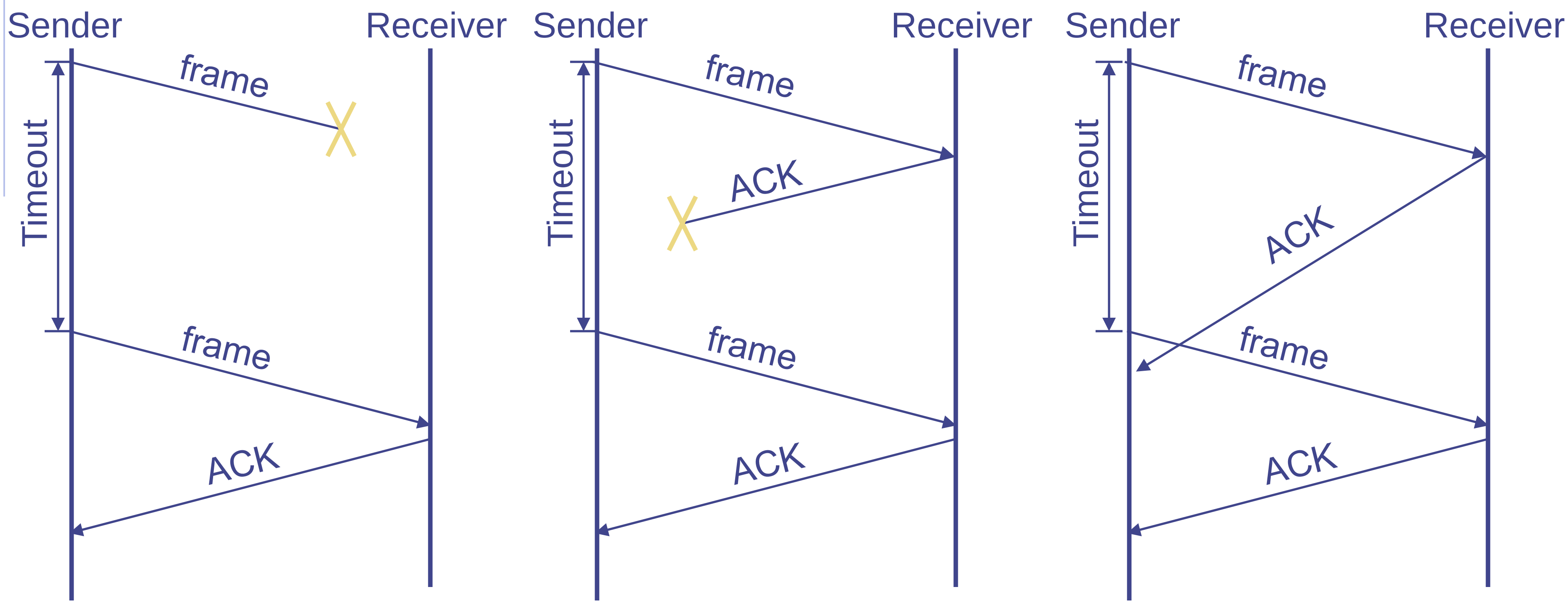
- Two examples
  - Stop-and-Wait
  - Sliding window

Send-and-Wait

- Receiver: send an acknowledge (ACK) back to the sender upon receiving a packet (frame)
- Sender: excepting first packet, send a packet only upon receiving the ACK for the previous packet



What Can Go Wrong?



Frame lost → resent it on Timeout

ACK lost → resent packet

Need a mechanisms to detect duplicate packet

ACK delayed → resent packet

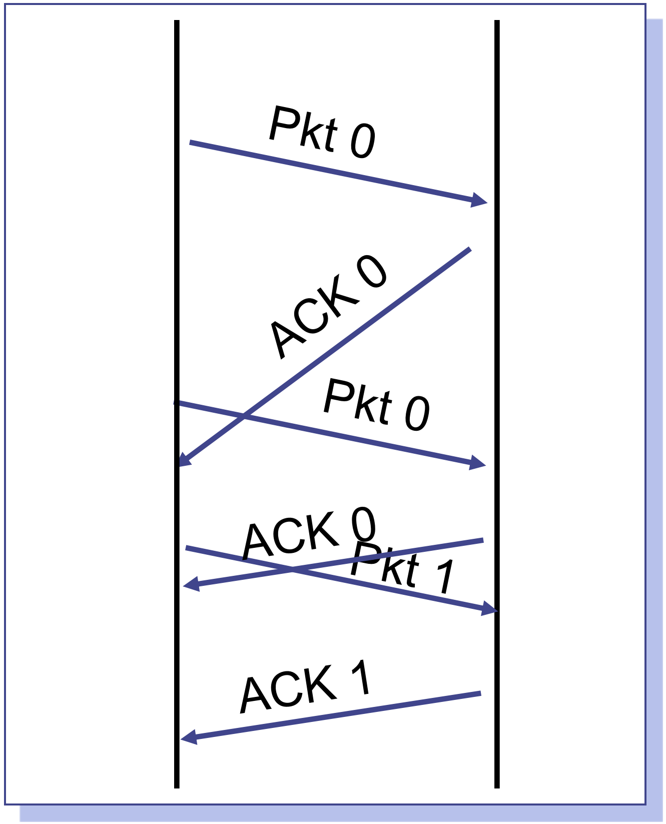
Need a mechanism to differentiate
between ACK for current and previous packet

How to Recognize Retransmissions?

- Use sequence numbers
  - both packets and acks

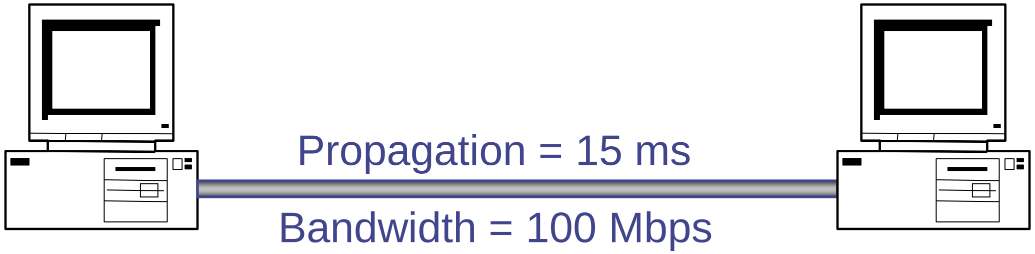
- Sequence # in packet is finite
  - ⇒How big should it be?
  - ⇒For stop and wait?

- One bit – won't send seq #1 until received ACK for seq #0



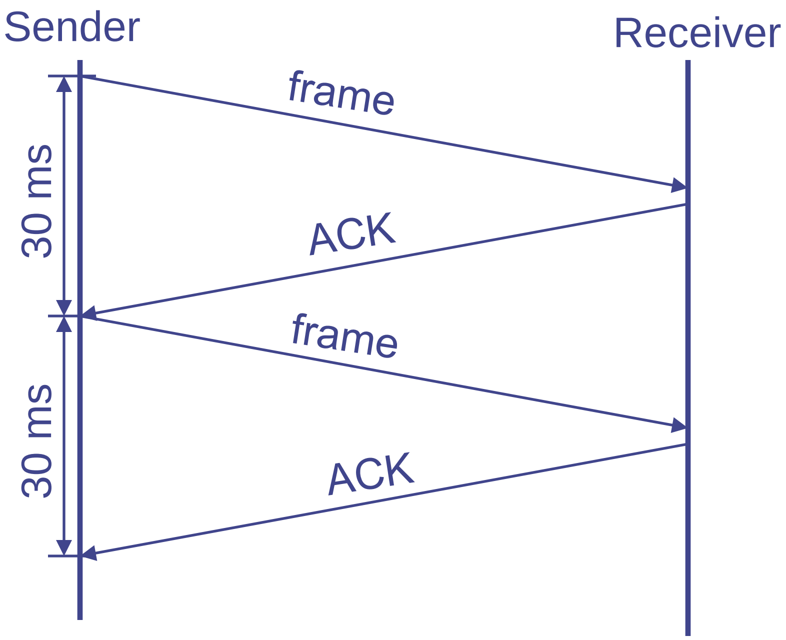
Stop-and-Wait Disadvantage

- May lead to inefficient link utilization
- Assume following example
  - One-way propagation = 15 ms
  - Bandwidth = 100 Mbps
  - Packet size = 1000 bytes → transmit = $(8*1000)/10^8 = 0.08$ms
  - Neglect queue delay → Latency = approx. 15 ms; RTT = 30 ms



Stop-and-Go Disadvantage

- Send a message every 30 ms
  - $\Rightarrow$Throughput = (8*1000)/0.03 = 0.2666 Mbps
  - $\Rightarrow$The protocol uses less than 0.3% of the link capacity!

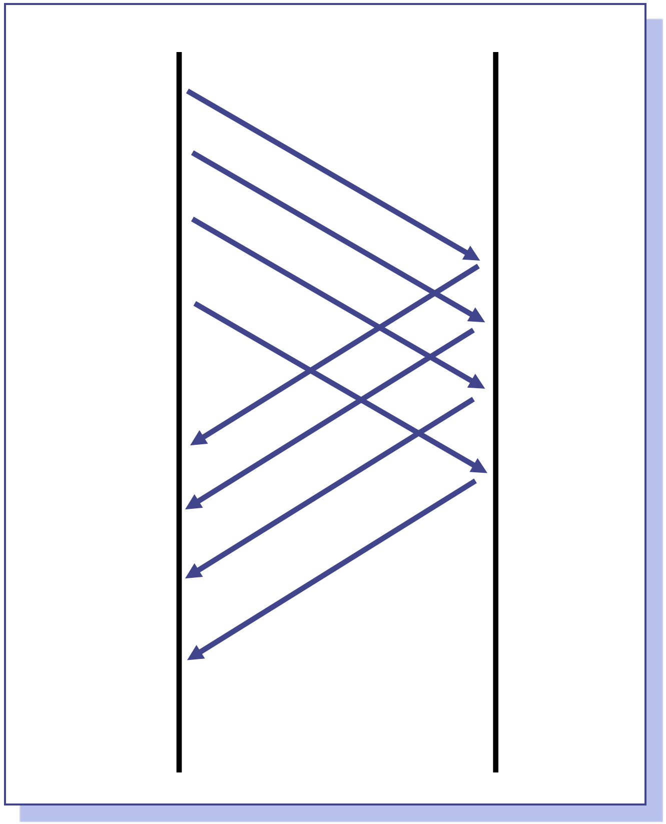


How to Keep the Pipe Full?

- Send multiple packets without waiting for first to be acked
  - Number of packets in flight = window

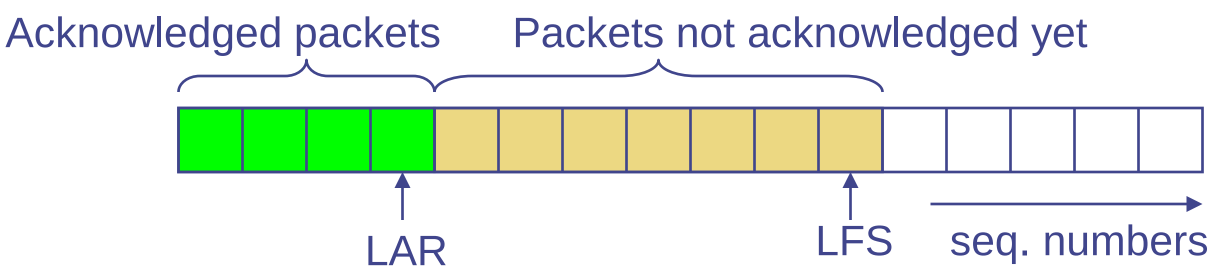
- Reliable, unordered delivery
  - Several parallel stop & waits
  - Send new packet after each ack
  - Sender keeps list of unack'ed packets; resends after timeout
  - Receiver same as stop & wait

- How large a window is needed?



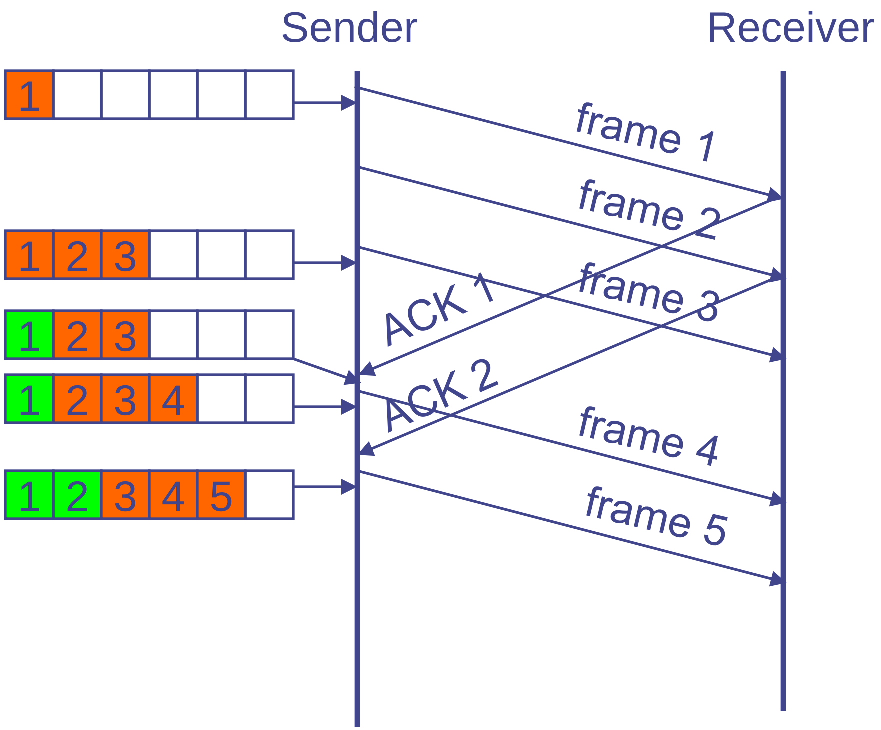
Sliding Window Protocol: Sender

- Each packet has a sequence number
  - Assume infinite sequence numbers for simplicity
- Sender maintains a window of sequence numbers
  - SWS (sender window size) – maximum scope of packets that can be sent without receiving an ACK
  - LAR (last ACK received)
  - LFS (last frame sent)



Example

- Assume SWS = 3



Note: usually ACK contains the sequence number of the first packet in sequence expected by receiver

Sliding Window Protocol: Receiver

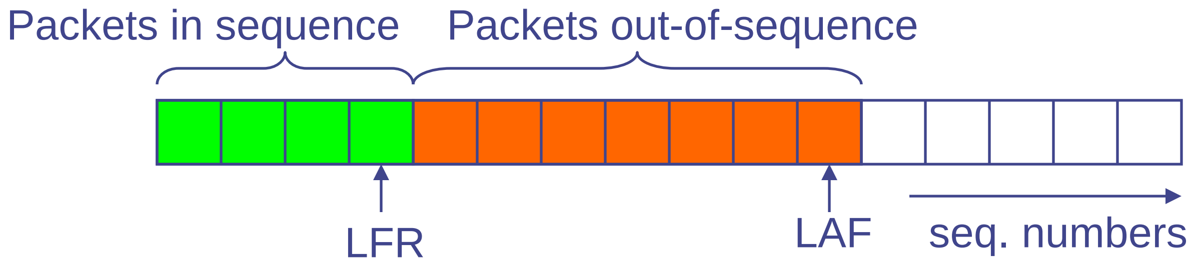
- Receiver maintains a window of sequence numbers
  - RWS (receiver window size) = maximum number of out-of-sequence packets that can received
  - LFR (last frame received) = last frame received in sequence
  - LAF (last acceptable frame)
  - LAF – LFR <= RWS

Sliding Window Protocol: Receiver

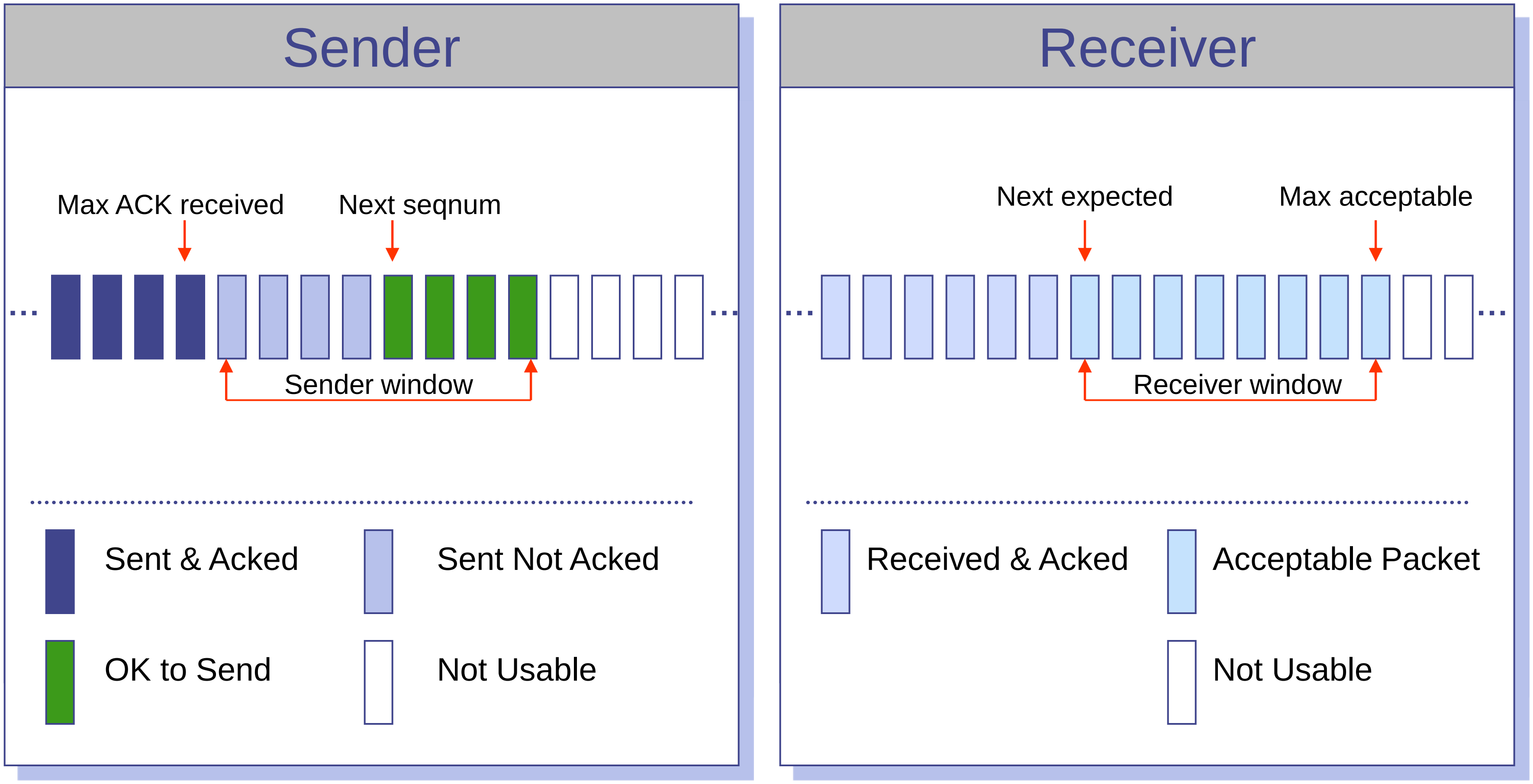
- Let seqNum be the sequence number of arriving packet

- If (seqNum <= LFR) or (seqNum >= LAF)
  - Discard packet

- Else
  - Accept packet
  - ACK largest sequence number seqNumToAck, such that all packets with sequence numbers <= seqNumToAck were received



Sender/Receiver State



Sequence Numbers

- How large do sequence numbers need to be?
  - Must be able to detect wrap-around
  - Depends on sender/receiver window size

- Example
  - Max seq = 7, send win=recv win=7
  - If pkts 0..6 are sent successfully and all acks lost
    - Receiver expects 7,0..5, sender retransmits old 0..6!!!

- Max sequence must be ≥ send window + recv window

Cumulative ACK + Go-Back-N

- On reception of new ACK (i.e. ACK for something that was not acked earlier)
  - Increase sequence of max ACK received
  - Send next packet

- On reception of new in-order data packet (next expected)
  - Hand packet to application
  - Send cumulative ACK – acknowledges reception of all packets up to sequence number
  - Increase sequence of max acceptable packet

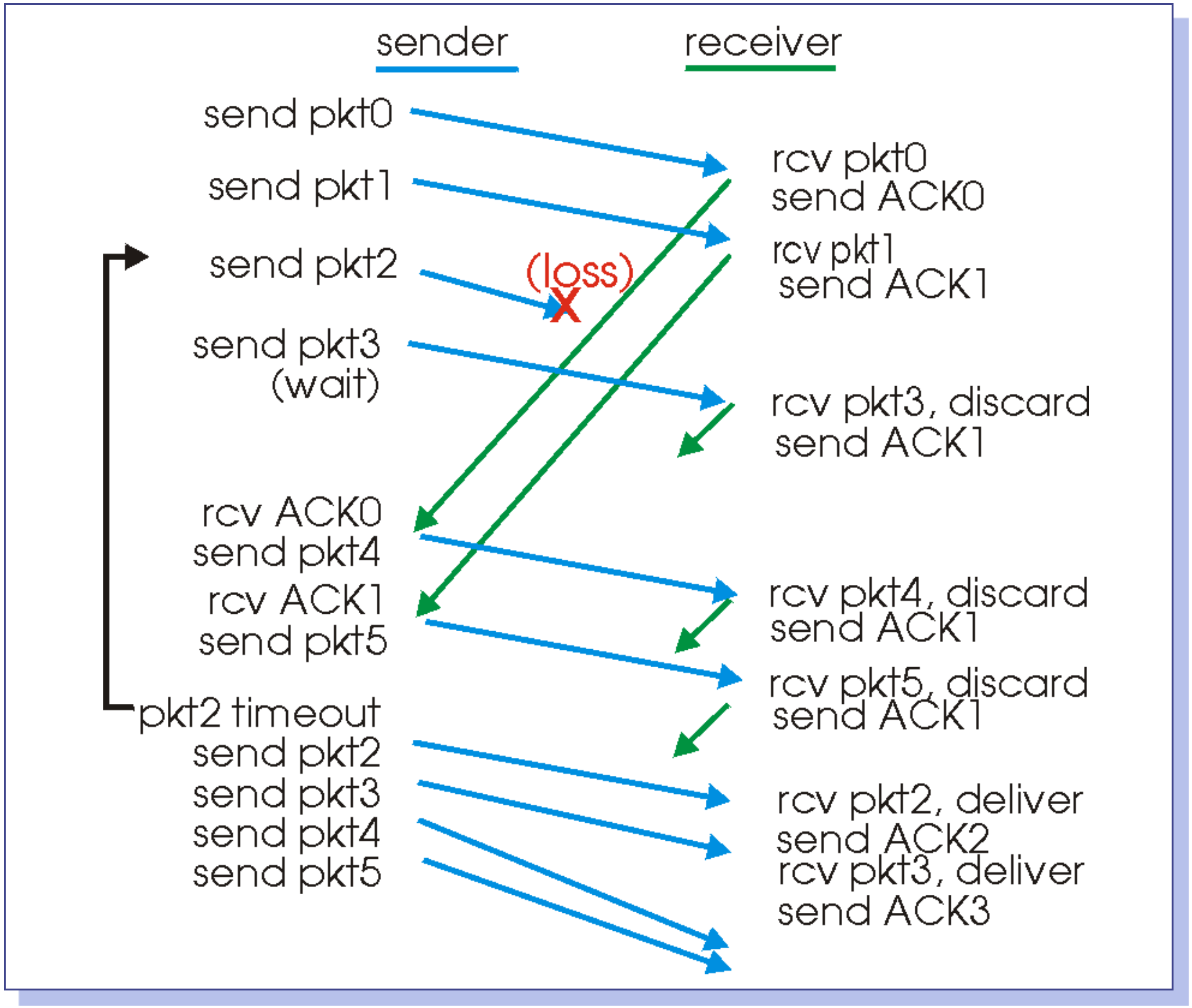
Loss Recovery

- On reception of out-of-order packet
  - Send nothing (wait for source to timeout)
  - Cumulative ACK (helps source identify loss)

- Timeout (Go-Back-N recovery)
  - Set timer upon transmission of packet
  - Retransmit all unacknowledged packets

- Performance during loss recovery
  - No longer have an entire window in transit
  - Can have much more clever loss recovery

Go-Back-N in Action



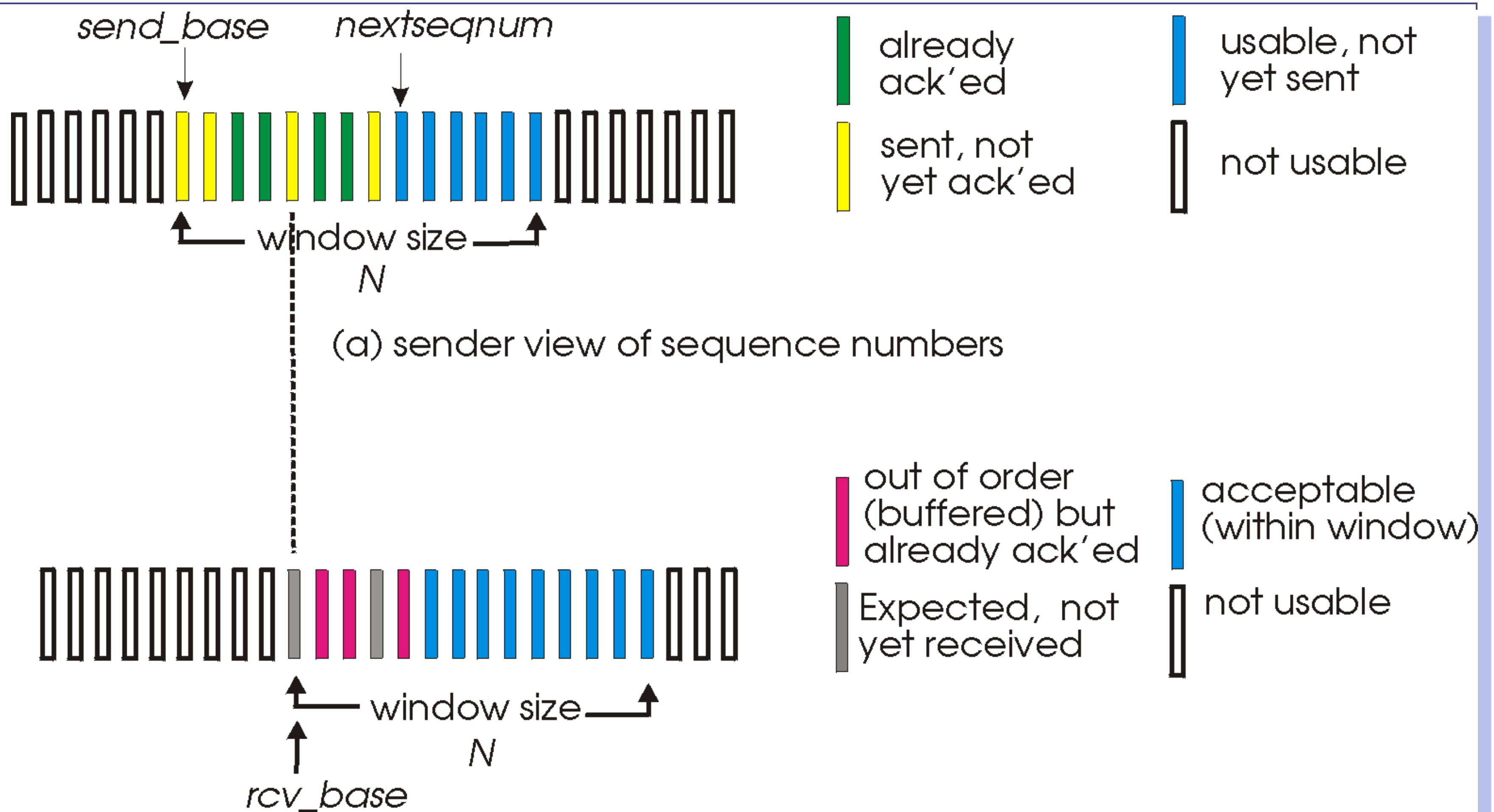
Selective Ack + Selective Repeat

- Receiver individually acknowledges all correctly received packets
  - Buffers packets, as needed, for eventual in-order delivery to upper layer

- Sender only resends packets for which ACK not received
  - Sender timer for each unACKed packet

- Sender window
  - N consecutive seq #’s
  - Again limits seq #s of sent, unACKed packets

Selective Repeat: Sender, Receiver Windows



(a) sender view of sequence numbers

(b) receiver view of sequence numbers

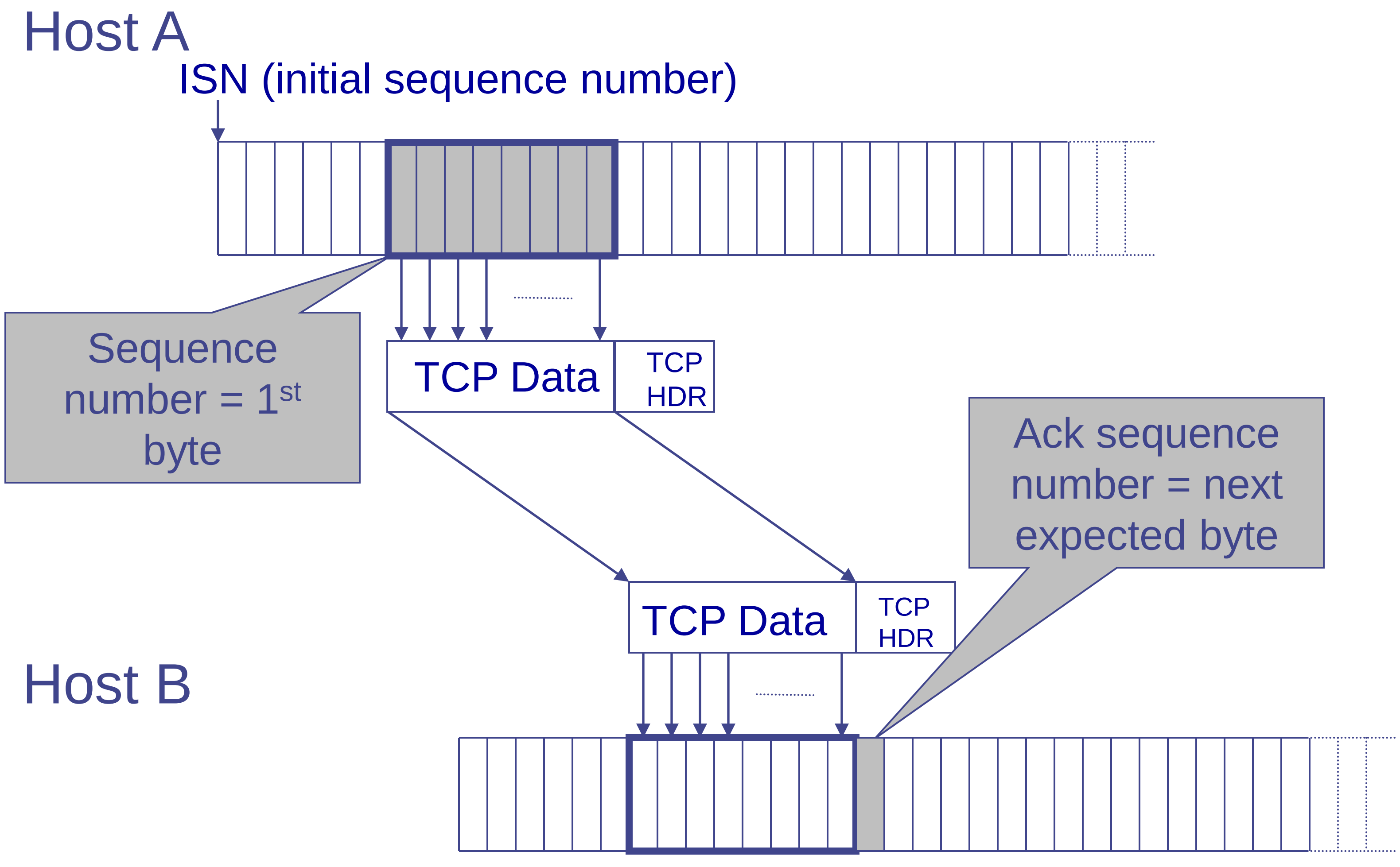
Summary of ARQ Protocols

- Mechanisms
  - Sequence number
  - Timeout
  - Acknowledgement

- Sender window: fill the pipe

- Receiver window: handle out-of-order delivery

TCP Sequence Numbers



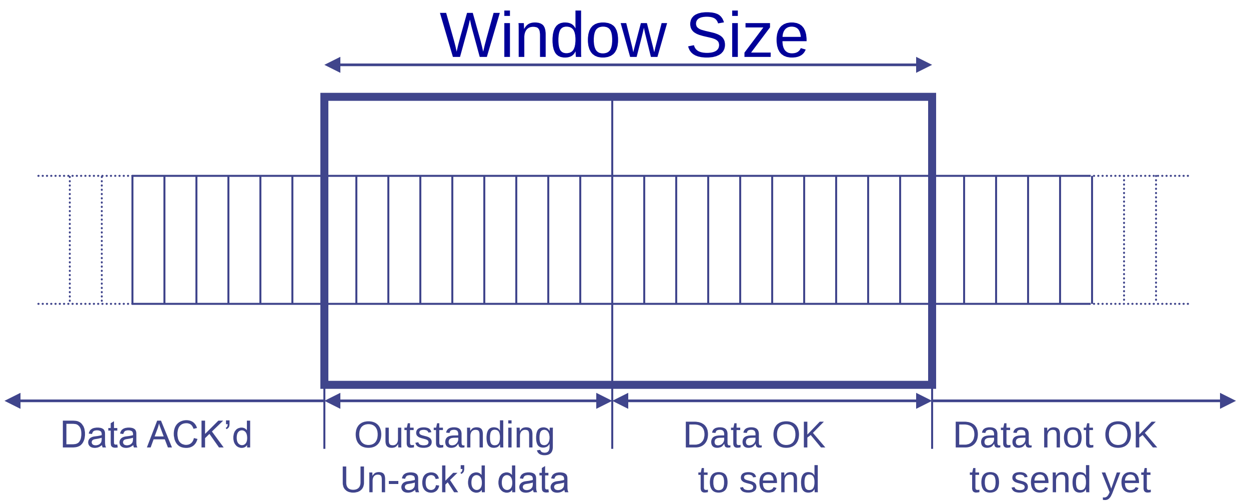
TCP Sliding Window

- How much data can a TCP sender have outstanding in the network?

- How much data should TCP retransmit when an error occurs? Just selectively repeat the missing data?

- How does the TCP sender avoid over-running the receiver’s buffers?

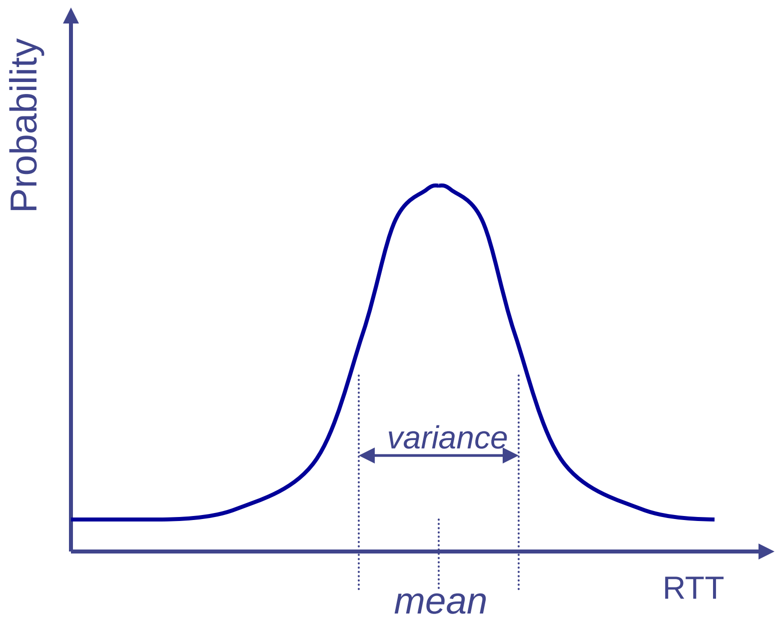
TCP Sliding Window



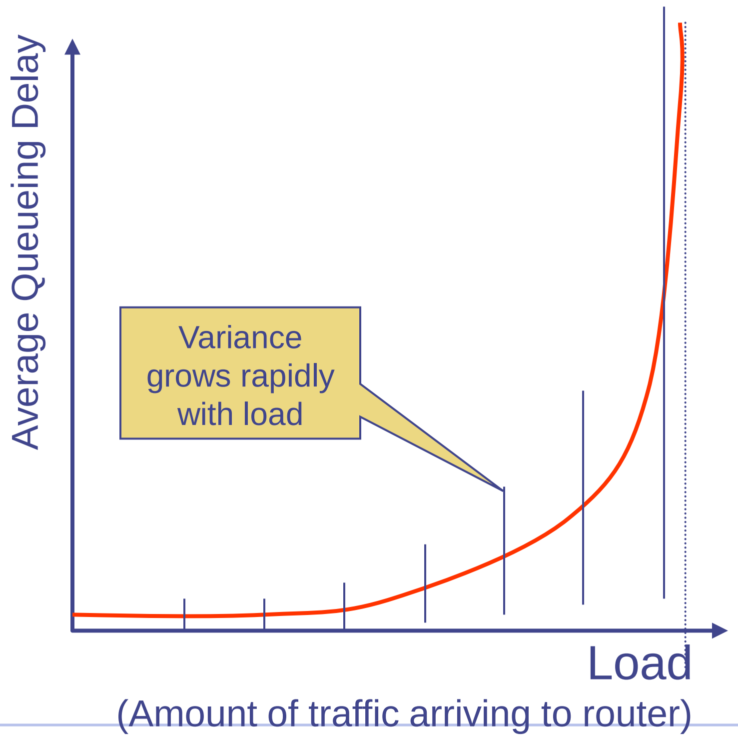
- Window is meaningful to the sender
- Current window size is “advertised” by receiver (usually 4k – 8k Bytes when connection set-up)
- TCP’s Retransmission policy is “Go Back N”

TCP: Retransmission and Timeouts

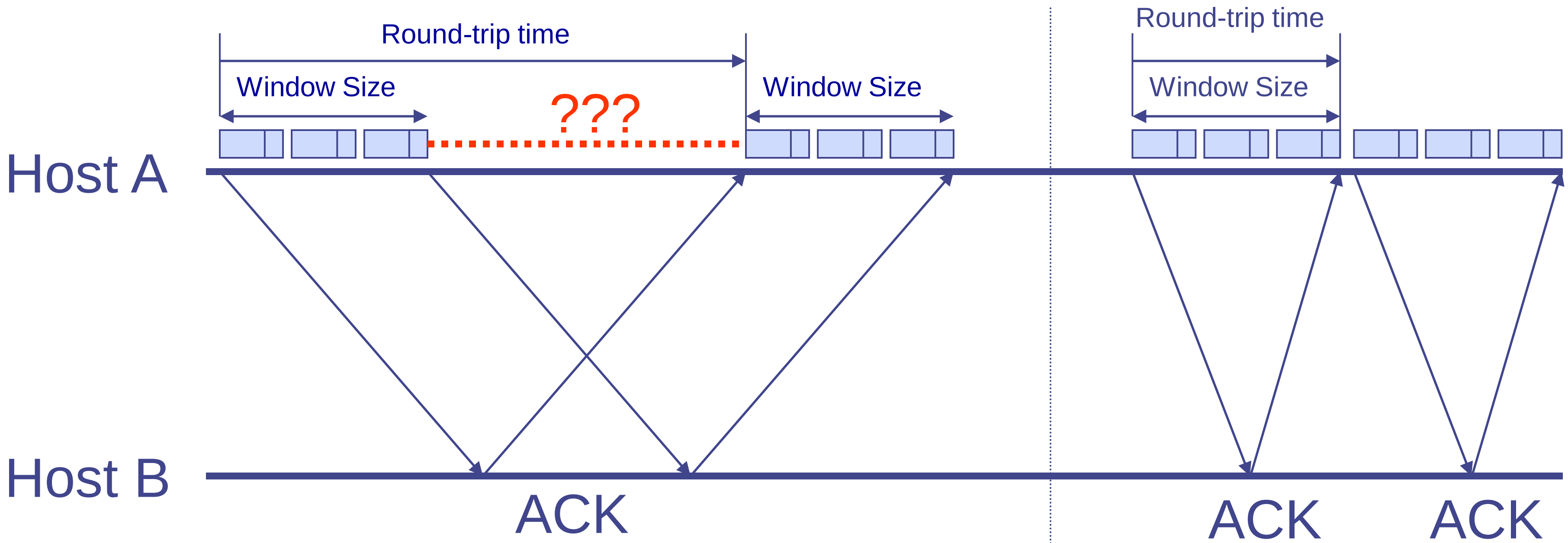
- There will be some (unknown) distribution of RTTs
- Estimate Retransmission TimeOut (RTO) to minimize probability of false timeout



- Router queues grow when there is more traffic, until they become unstable
- As load grows, variance of delay grows rapidly



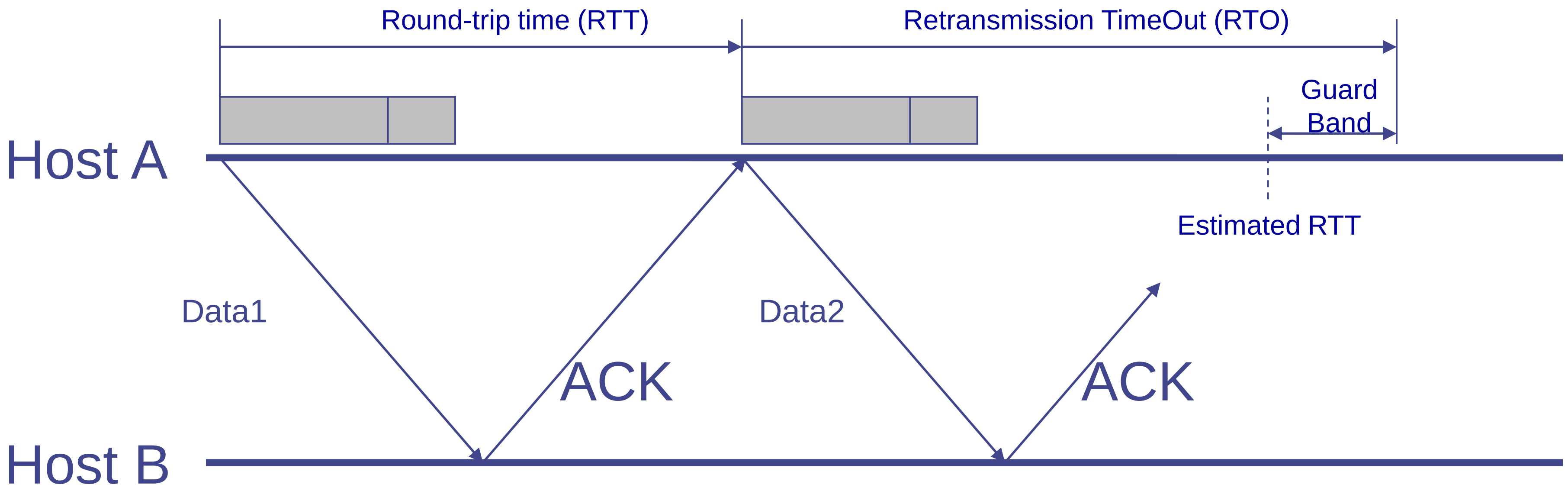
TCP Sliding Window



(1) RTT > Window size

(2) RTT = Window size

TCP: Retransmission and Timeouts



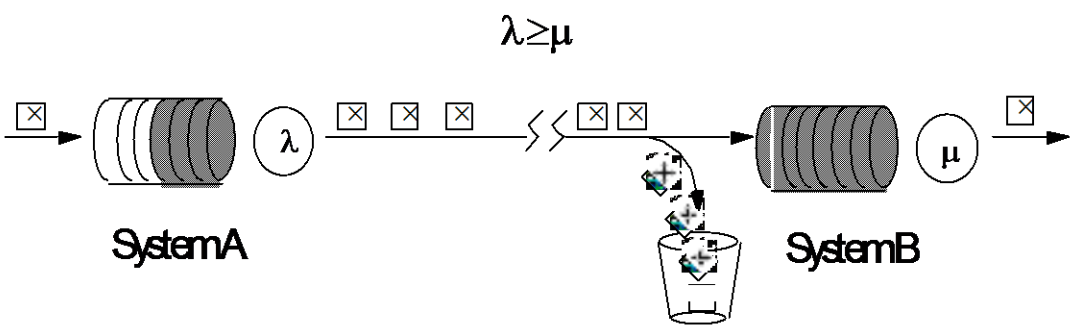
TCP uses an adaptive retransmission timeout value:

Congestion
Changes in Routing

RTT changes frequently

5.4 Flow Control: Motivation

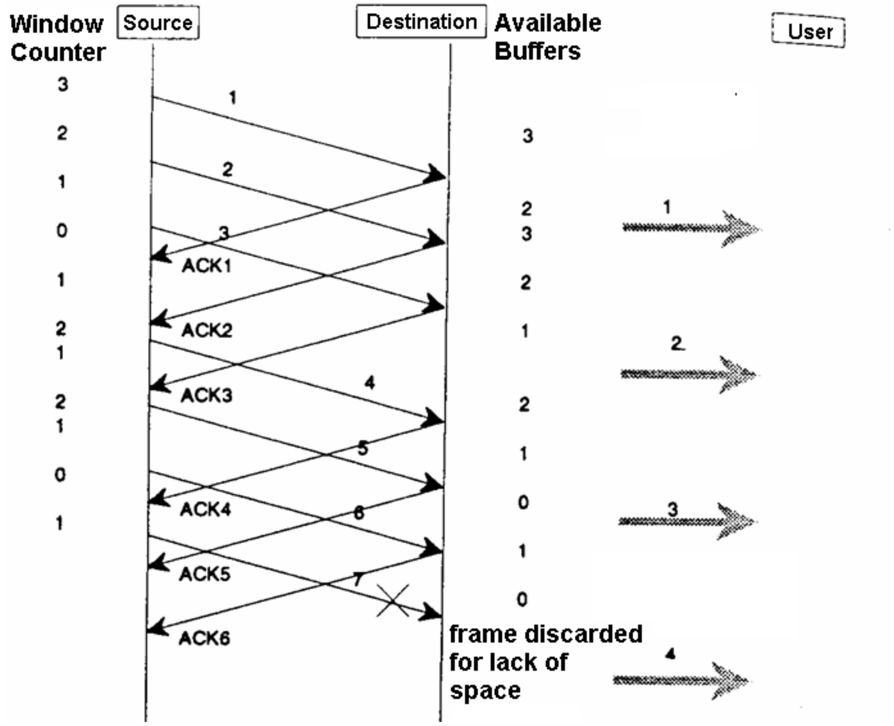
- Mismatch between the sender processing speed (in packets/s) and receiver processing speed



- Some control mechanism is needed !
- Flow Control is typically used to insure that a source does not overwhelm a destination with more traffic than it can handle
- Major approaches
  - Window based flow control
  - Rate based flow control

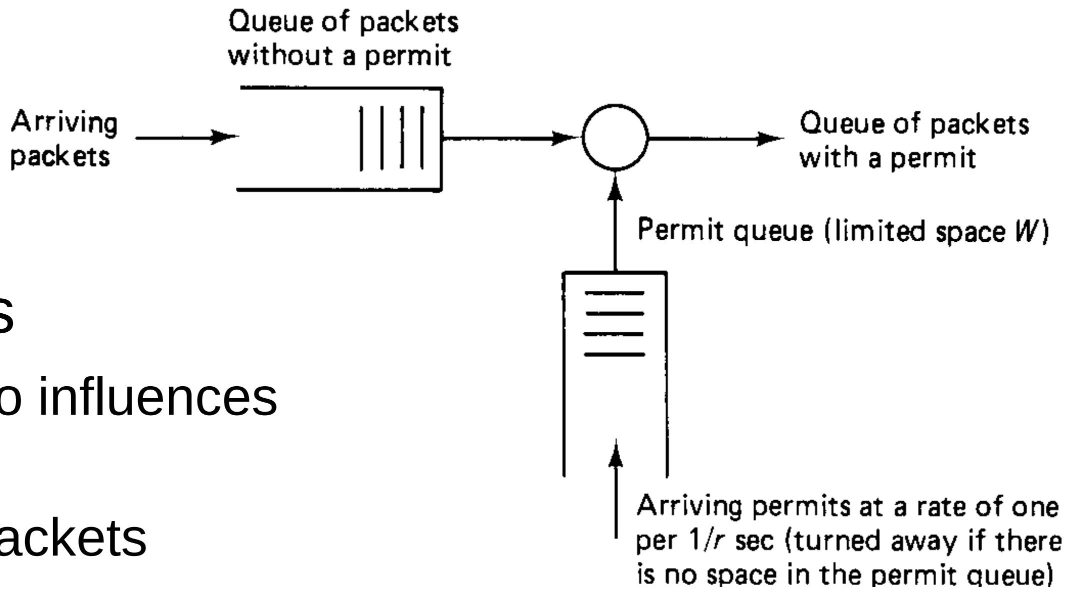
Generation of Permits upon Reception by Destination

- Permit will be sent immediately upon reception of packet (ack == permit)
- User consumes the received data at his discretion
- Packets may be discarded because of a lack of buffering space
- Excessive buffering capacity at sender necessary



Rate control

- Very often also referred to as traffic shaping or network access control
- Controls the amount of data per time unit
- One of several possible algorithms and implementations is Token Bucket:



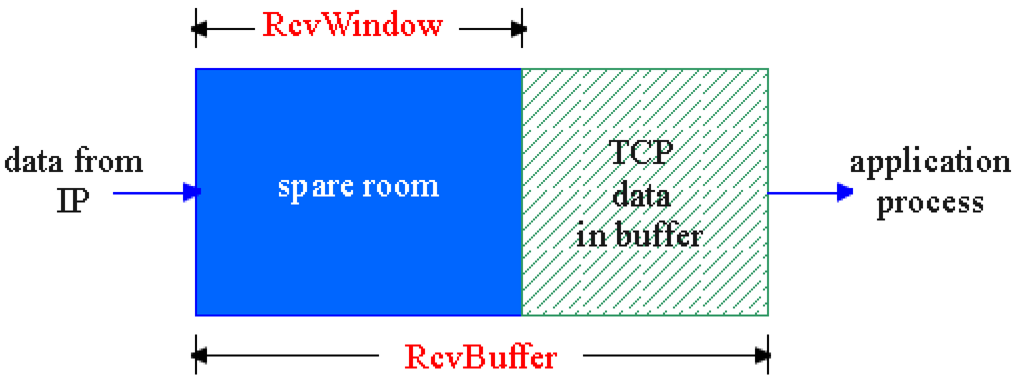
- Number of advantages
  - Open loop approach (no influences from round trip time)
  - Use of fewer network packets
- Applied in XTP and ATM networks

TCP Flow Control

flow control
sender won't overrun receiver's buffers by transmitting too much, too fast

RcvBuffer = size or TCP Receive Buffer

RcvWindow = amount of spare room in Buffer

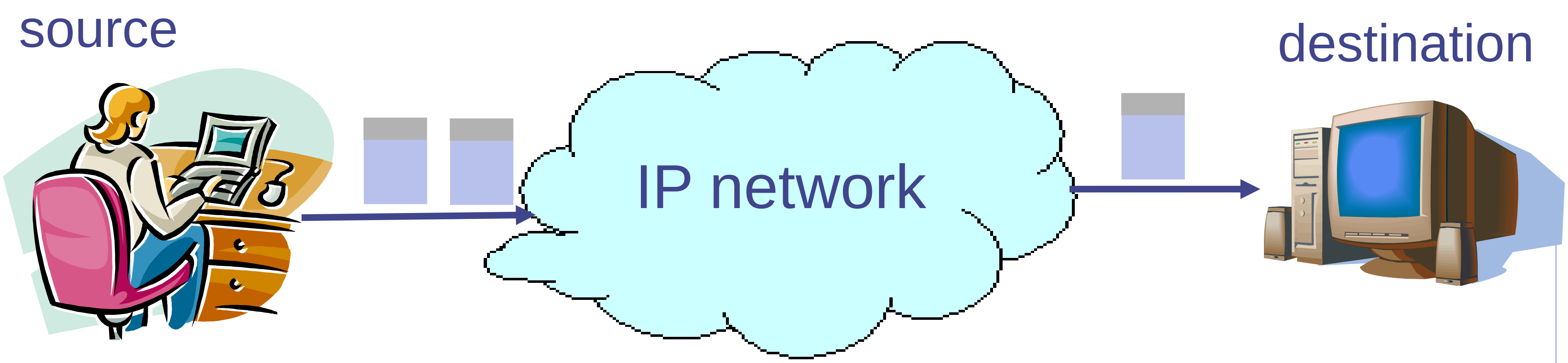


receiver buffering

- Receiver
  - Explicitly informs sender of (dynamically changing) amount of free buffer space
  - RcvWindow field in TCP segment
- Sender
  - keeps the amount of transmitted, unACKed data less than most recently received RcvWindow

IP Best-Effort Design Philosophy

- Best-effort delivery
  - Let everybody send
  - Try to deliver what you can
  - … and just drop the rest

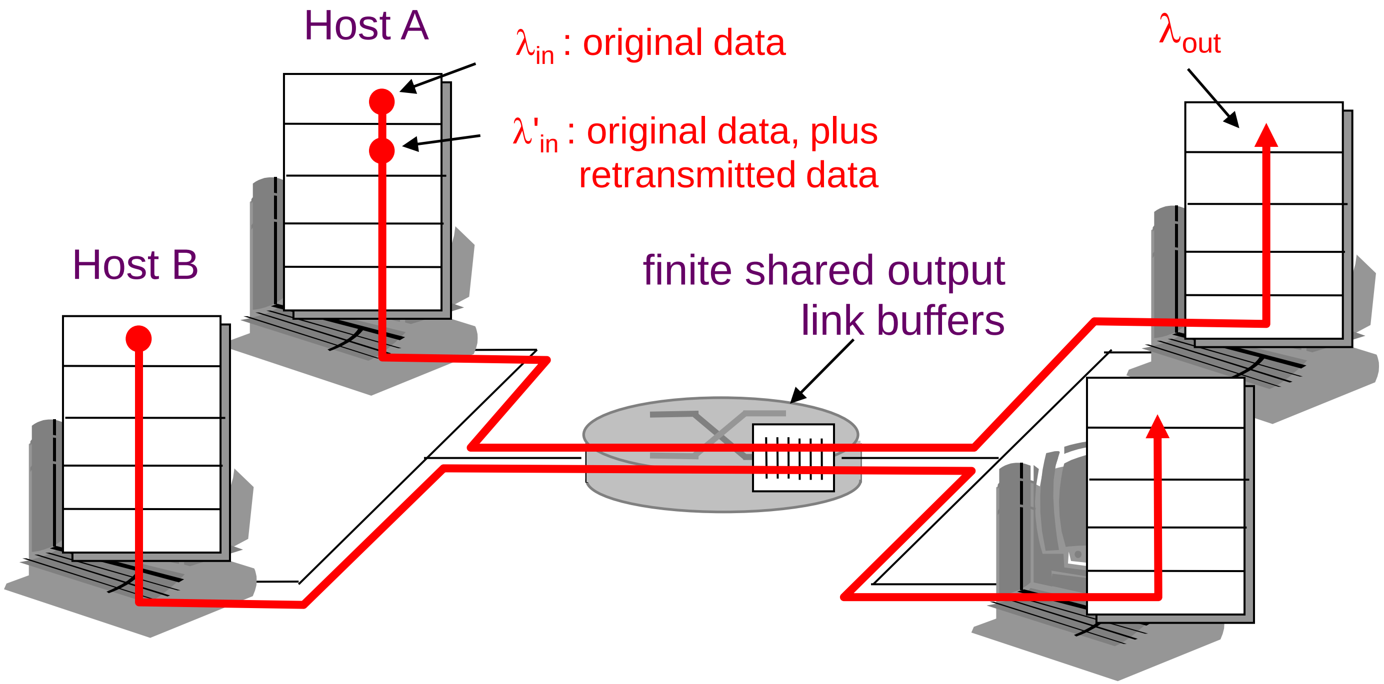


Causes/costs of congestion

- One router, finite buffers, sender retransmission of lost packet

  ⇒more work (retrans) for given "goodput"

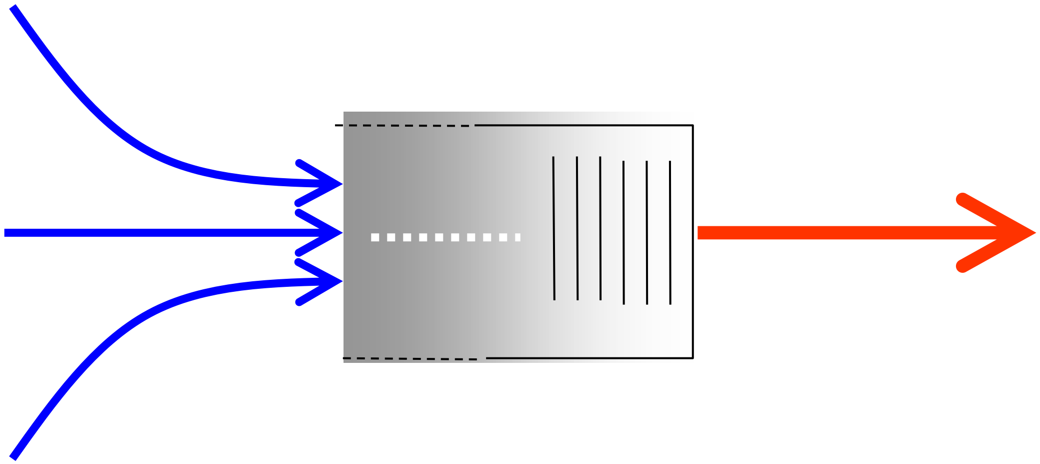
  ⇒unneeded retransmissions: link carries multiple copies of packet



Congestion is Unavoidable

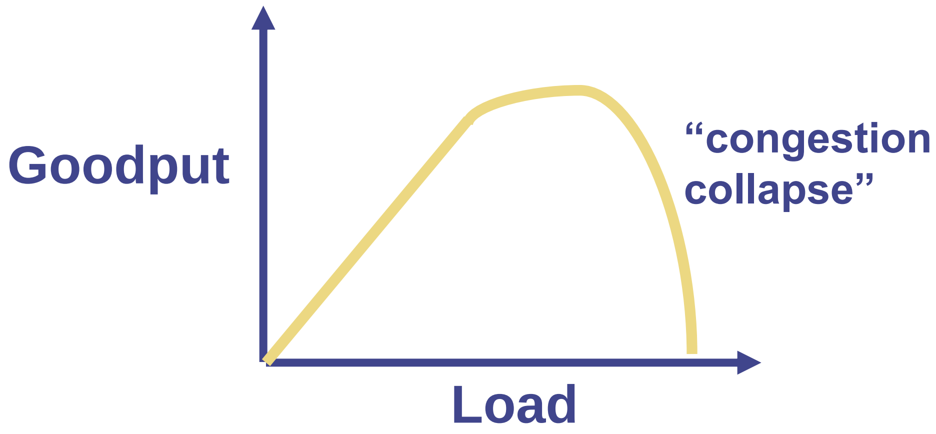
- Two packets arrive at the same time
  - The node can only transmit one
  - … and either buffer or drop the other

- If many packets arrive in short period of time
  - The node cannot keep up with the arriving traffic
  - … and the buffer may eventually overflow



5.5 The Problem of Congestion

- What is congestion?
  - Load is higher than capacity
- What do IP routers do?
  - Drop the excess packets
- Why is this bad?
  - Wasted bandwidth for retransmissions



Increase in load that results in a decrease in useful work done.

Ways to Deal With Congestion

- Ignore the problem
  - Many dropped (and retransmitted) packets
  - ⇒Causes congestion collapse
- Reservations, like in circuit switching
  - Pre-arrange bandwidth allocations
  - Requires negotiation before sending packets
- Pricing
  - Don't drop packets for the high-bidders
  - Requires a payment model
- Dynamic adjustment (TCP)
  - Every sender infers the level of congestion
  - And adapts its sending rate, for the greater good

How it Looks to the End Host

- Packet delay
  - Packet experiences high delay
- Packet loss
  - Packet gets dropped along the way
- How does TCP sender learn this?
  - Delay
    - Round-trip time estimate
  - Loss
    - Timeout
    - Triple-duplicate acknowledgment (three ACKs in a row with one or more missing ACKs between them $\Rightarrow$ Lost packet)

What Can the End Host Do?

- Upon detecting congestion
  - Decrease the sending rate (e.g. divide in half)
  - End host does its part to alleviate the congestion
- But, what if conditions change?
  - Suppose there is more bandwidth available
  - Would be a shame to stay at a low sending rate
- Upon *not* detecting congestion
  - Increase the sending rate, a little at a time
  - And see if the packets are successfully delivered

TCP Congestion Window

- Each TCP sender maintains a congestion window
  – Maximum number of bytes to have in transit
  – I.e., number of bytes still awaiting acknowledgments

- Adapting the congestion window
  – Decrease upon losing a packet: backing off
  – Increase upon success: optimistically exploring
  – Always struggling to find the right transfer rate

- Both good and bad
  – Pro: avoids having explicit feedback from network
  – Con: under-shooting and over-shooting the rate

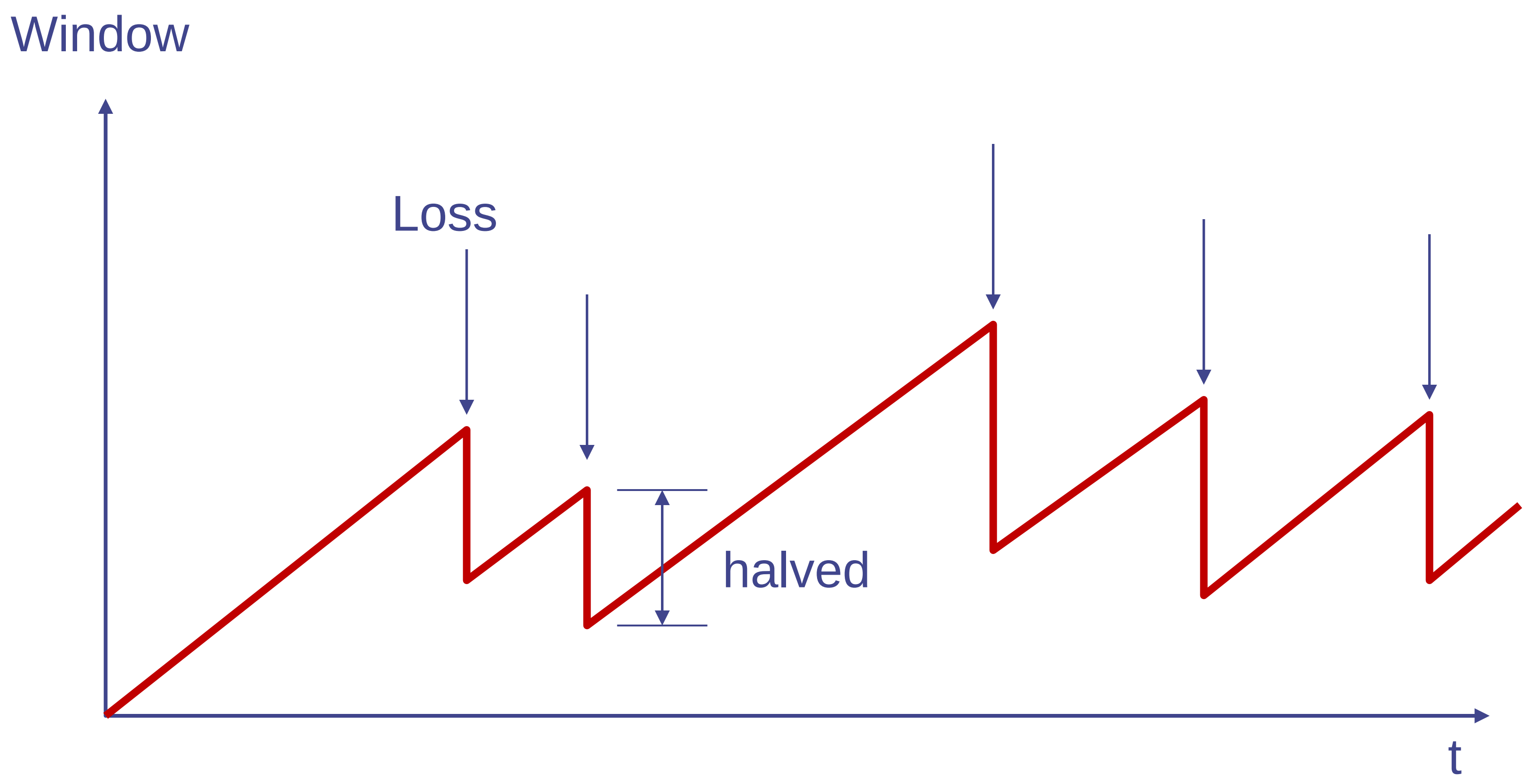
Additive Increase, Multiplicative Decrease

- How much to increase and decrease?
  - Increase linearly, decrease multiplicatively
  - A necessary condition for stability of TCP
  - Consequences of over-sized window are much worse than having an under-sized window
    - Over-sized window: packets dropped and retransmitted
    - Under-sized window: somewhat lower throughput

- Multiplicative decrease
  - On loss of packet, divide congestion window in half

- Additive increase
  - On success for last window of data, increase linearly

Leads to the TCP "Sawtooth"



Practical Details

- Congestion window
  - Represented in bytes, not in packets
  - Packets have MSS (Maximum Segment Size) bytes

- Increasing the congestion window
  - Increase by MSS on success for last window of data

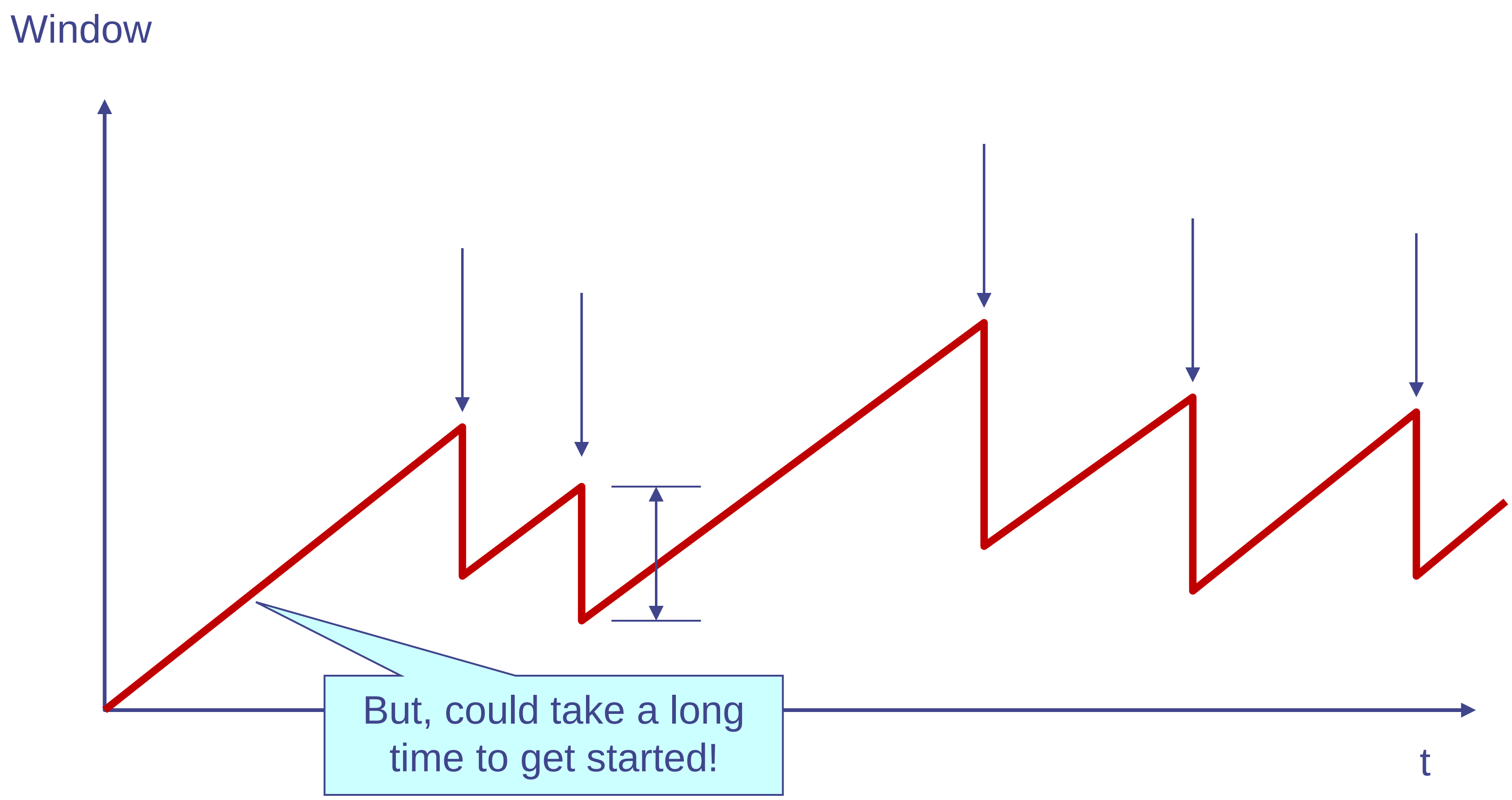
- Decreasing the congestion window
  - Never drop congestion window below 1 MSS

Receiver Window vs. Congestion Window

- Flow control
  - Keep a *fast sender* from overwhelming *a slow receiver*
- Congestion control
  - Keep a *set of senders* from overloading the *network*
- Different concepts, but similar mechanisms
  - TCP flow control: receiver window
  - TCP congestion control: congestion window
  - TCP window: min{congestion window, receiver window}

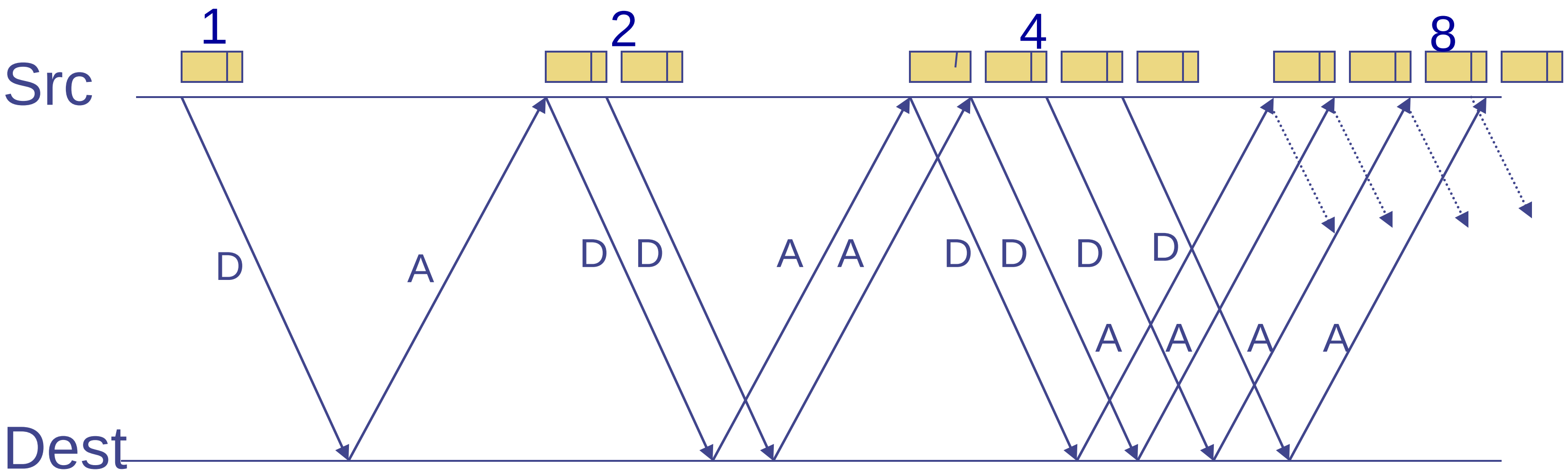
How Should a New Flow Start

Need to start with a small CWND to avoid overloading the network

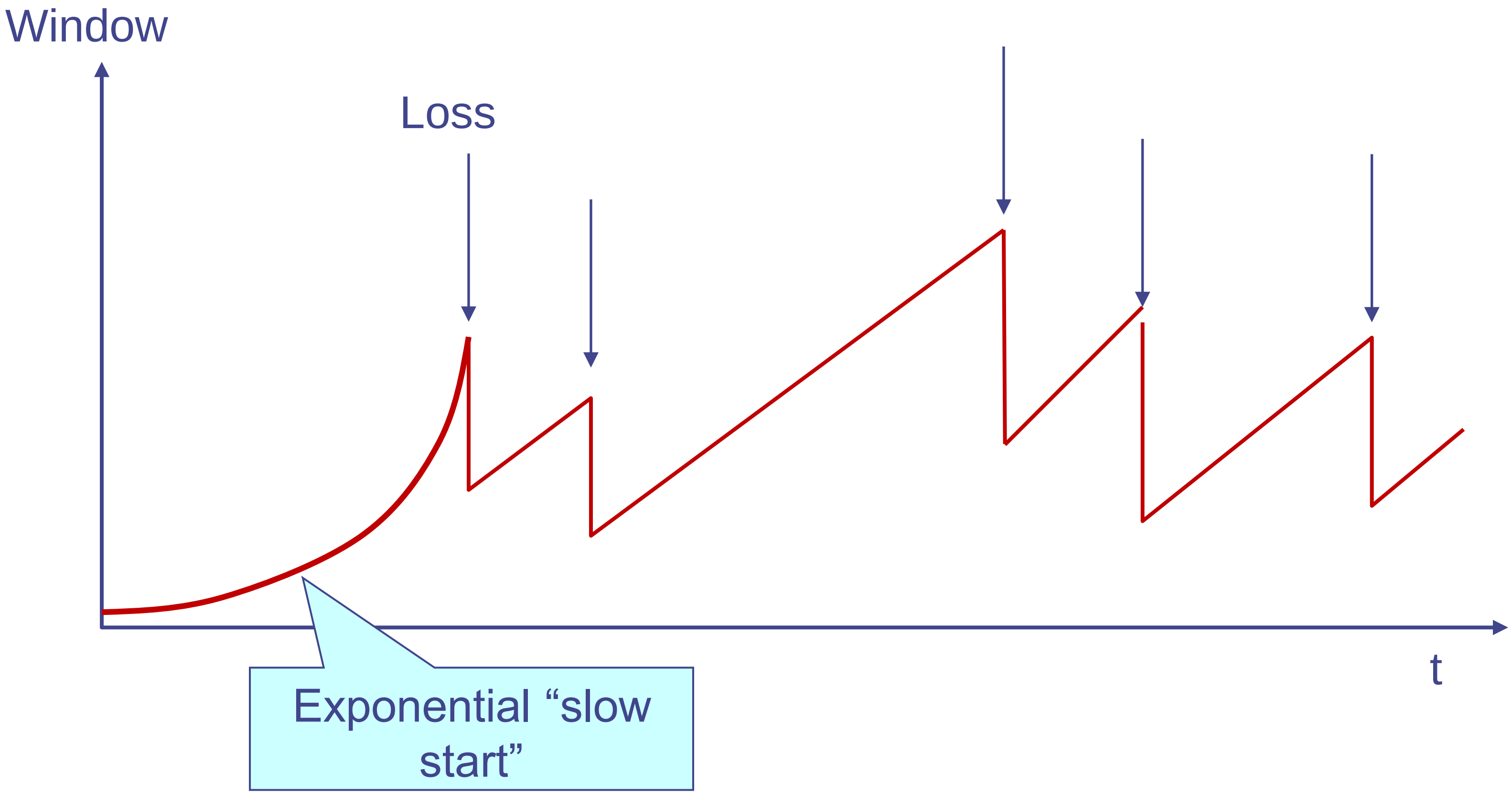


"Slow Start" in Action

Double CWND per round-trip time

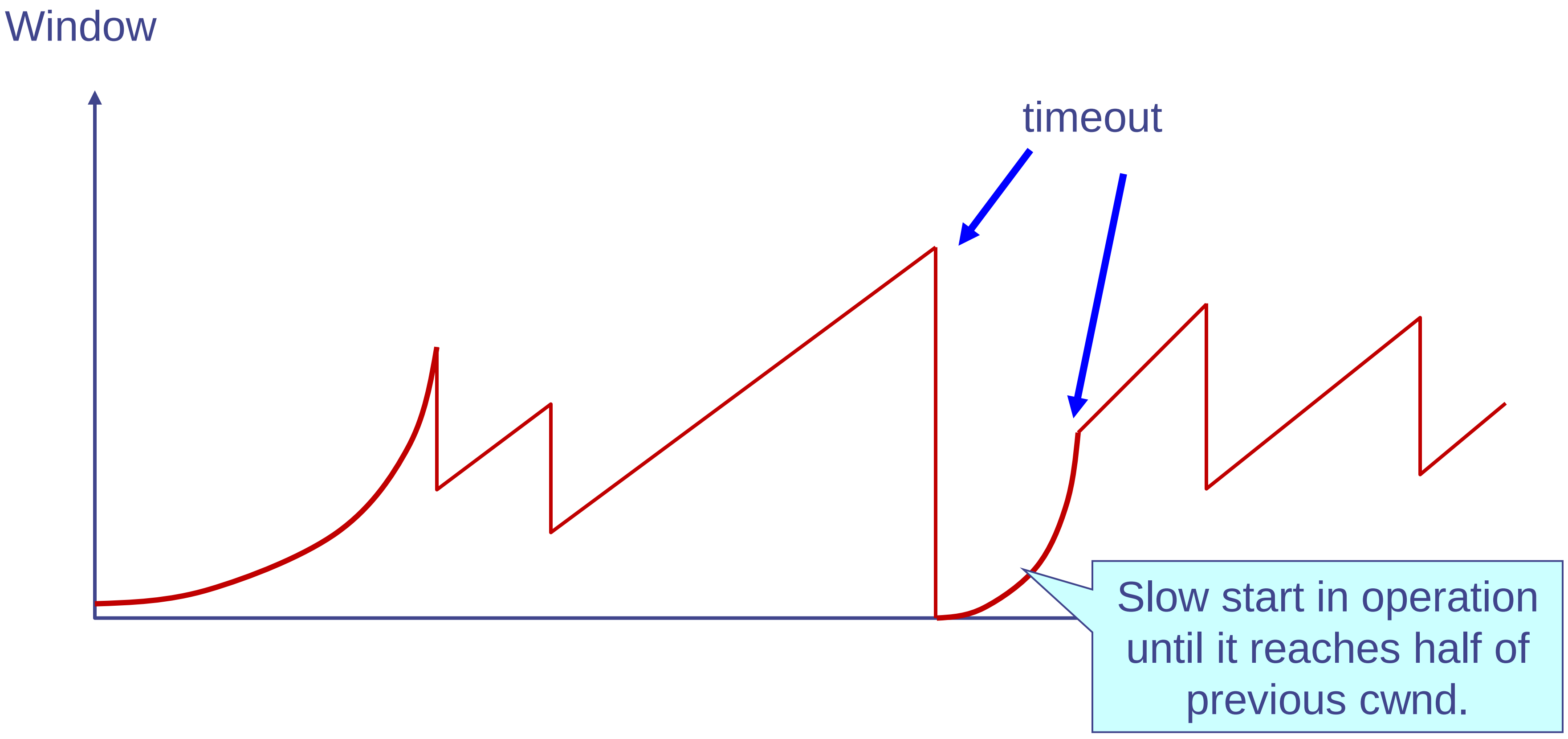


Slow Start and the TCP Sawtooth



Why is it called slow-start? Because TCP originally had no congestion control mechanism. The source would just start by sending a whole receiver window's worth of data.

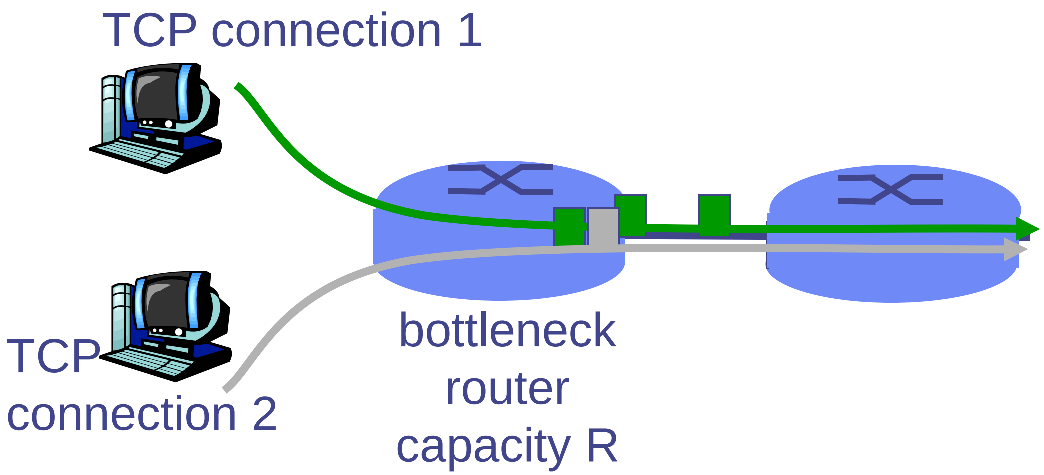
Repeating Slow Start After Timeout



Slow-start restart: Go back to CWND of 1, but take advantage of knowing the previous value of CWND.

TCP Fairness

- Fairness goal
  - If N TCP sessions share same bottleneck link, each should get 1/N of link capacity



What About Cheating?

- Some folks are more fair than others
  - Running multiple TCP connections in parallel
  - Modifying the TCP implementation in the OS
  - Use the User Datagram Protocol

- What is the impact
  - Good guys slow down to make room for you
  - You get an unfair share of the bandwidth

- Possible solutions?
  - Routers detect cheating and drop excess packets?
  - Peer pressure?
  - ???