Transmission Control Protocol (TCP)

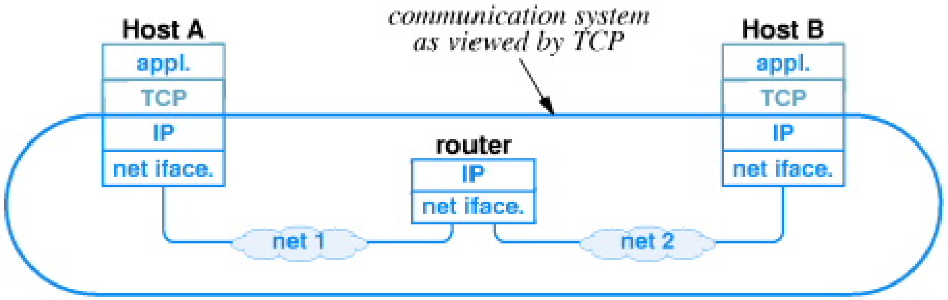
- Standardized by IETF as RFC 793
- Most popular layer 4 protocol
- Connection-oriented protocol
- Conceptually between applications and IP
- Provides reliable data delivery by using IP unreliable datagram delivery
- Compensates for loss, delay, duplication and similar problems in Internet components
- Reliable delivery is high-level, facilitates application development

TCP Characteristics

- Connection-oriented
  - Application requests connection to destination then uses connection to deliver data
- Point-to-point
  - A TCP connection has two endpoints
- Reliability
  - TCP guarantees data will be delivered without loss, duplication or transmission errors
- Full duplex
  - The endpoints of a TCP connection can exchange data in both directions simultaneously
- Stream interface
  - Application delivers data to TCP as a continuous stream, with no record boundaries; TCP makes no guarantees that data will be received in same blocks as transmitted
- Reliable connection startup
  - Three-way handshake guarantees reliable, synchronized startup between endpoints
- Graceful connection shutdown
  - TCP guarantees delivery of all data after endpoint shutdown by application

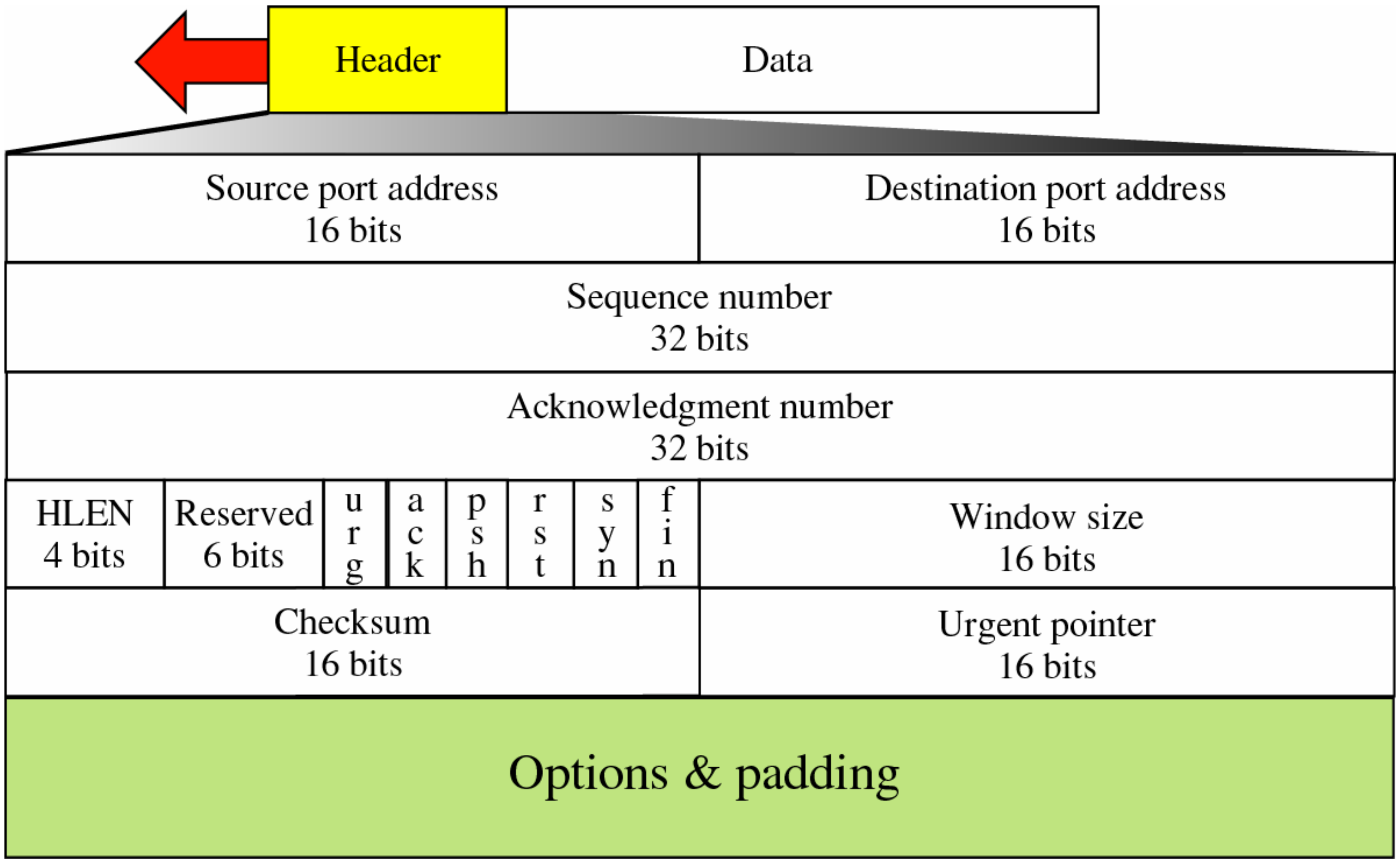
TCP and Layering

- TCP on one computer uses IP to communicate with TCP on another computer



- Protocol implemented entirely at the ends
- Protocol has evolved over time and will continue to do so
  – Nearly impossible to change the header
  – Uses options to add information to the header
  – Change processing at endpoints
  – Backward compatibility is what makes it TCP

TCP Header (1/2)



TCP Header (2/2)

- TCP and UDP ports are essentially the same, but are assigned separately (different services)
- Flow control is achieved using

- Sequence Numbers
- Sliding window mechanism

- Credit based flow control is also employed

- Window is used by receiver to advertise how much buffer space is left

- Flags

- URG: Urgent pointer is valid; send urgent data, bypass normal flow control
- ACK: Acknowledgment field is valid
- PSH: This segment requests a PUSH
    * Forces sender to send segment immediately
    * Forces receiver to forward it to destination process
- RST: Abort connections quickly
- SYN: Synchronize sequence numbers
- FIN: Sender has reached end of his byte stream

Achieving Reliability

- Reliable data transmission
- Reliable connection setup
- Reliable connection shutdown

Reliable Data Transmission

- Positive acknowledgement

  - Receiver returns short message when data arrive
  - Call an *acknowledgement*

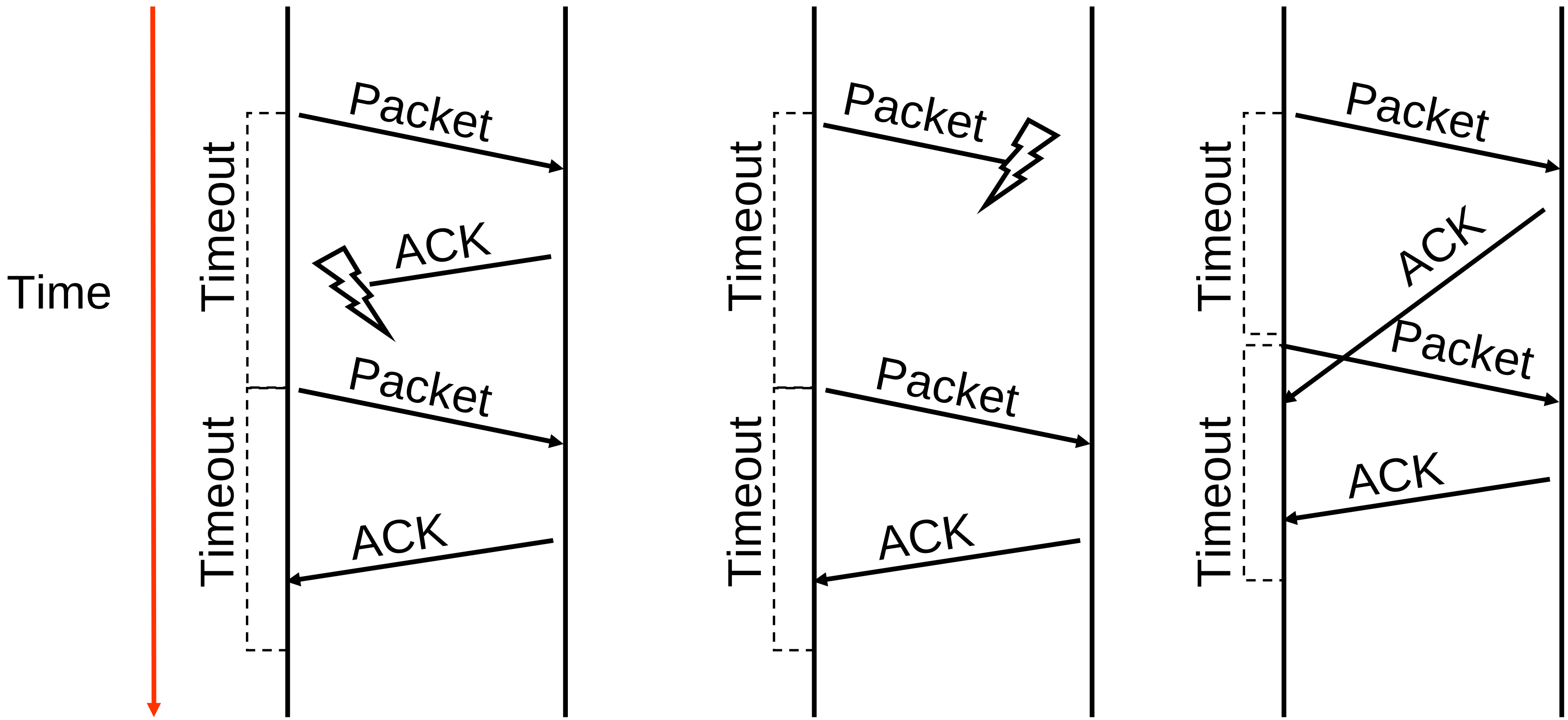
- Retransmission

  - Sender starts timer whenever message is transmitted
  - If timer expires before acknowledgement arrives, sender retransmits message

TCP Error Recovery

- Automatic Repeat Request (ARQ):
  - Receiver sends acknowledgement (ACK) when it receives packet
  - Sender waits for ACK and timeouts if it does not arrive within some time period

Stop and Wait



ACK lost   Packet lost   Early timeout

Retransmission Timeout

- How long should TCP wait before retransmitting?
- Time for acknowledgement to arrive depends on:

  - Distance to destination
  - Current traffic conditions
- Multiple connections can be open simultaneously
- Traffic conditions change rapidly
- So, TCP must be able to handle a variety of retransmission timeouts that can change rapidly

Adaptive Retransmission

- Keep estimate of round trip time (RTT) on each connection
- Use current estimate to set retransmission timer
- Known as *adaptive retransmission*
- Key to TCP's success

Keep the Pipe Full

- Send multiple packets without waiting for first to be ACK’ed

 - Number of pkts in flight == window size

- How large a window is needed

 - Round trip delay * bandwidth = capacity of pipe

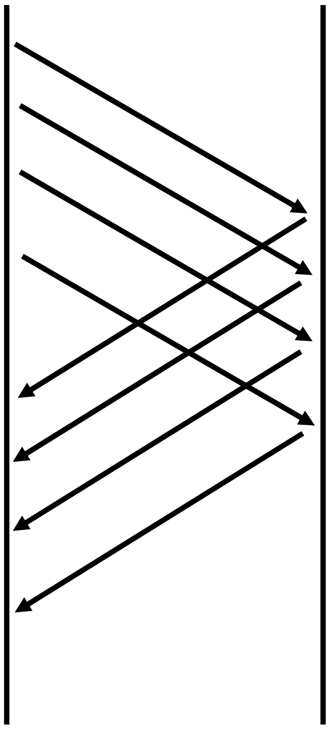
- Reliable, unordered delivery

 - Several parallel stop and waits

 - Send new packet after each ACK

 - Sender keeps list of unACK’ed packets; resends after timeout

 - Receiver same as stop and wait



TCP Flow Control

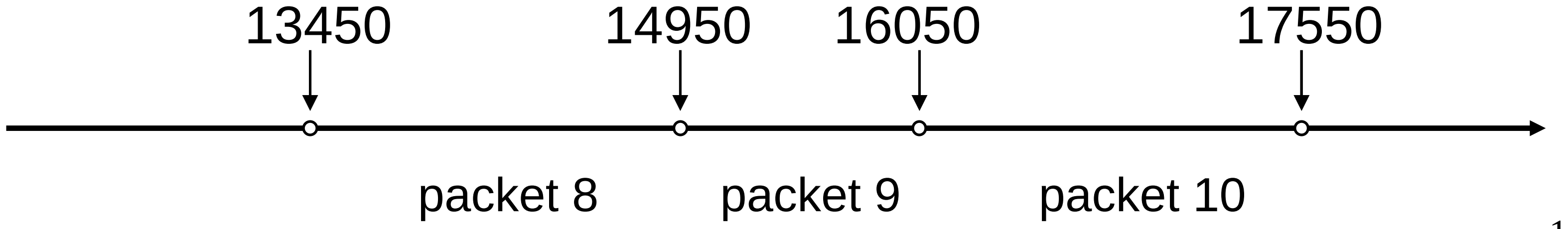
- TCP is a sliding window protocol
  - For window size *n*, can send up to *n* bytes without receiving an acknowledgement
  - When the data is acknowledged then the window slides forward
- Each packet advertises a window size
  - Indicates number of bytes the receiver has space for

Window Advertisement

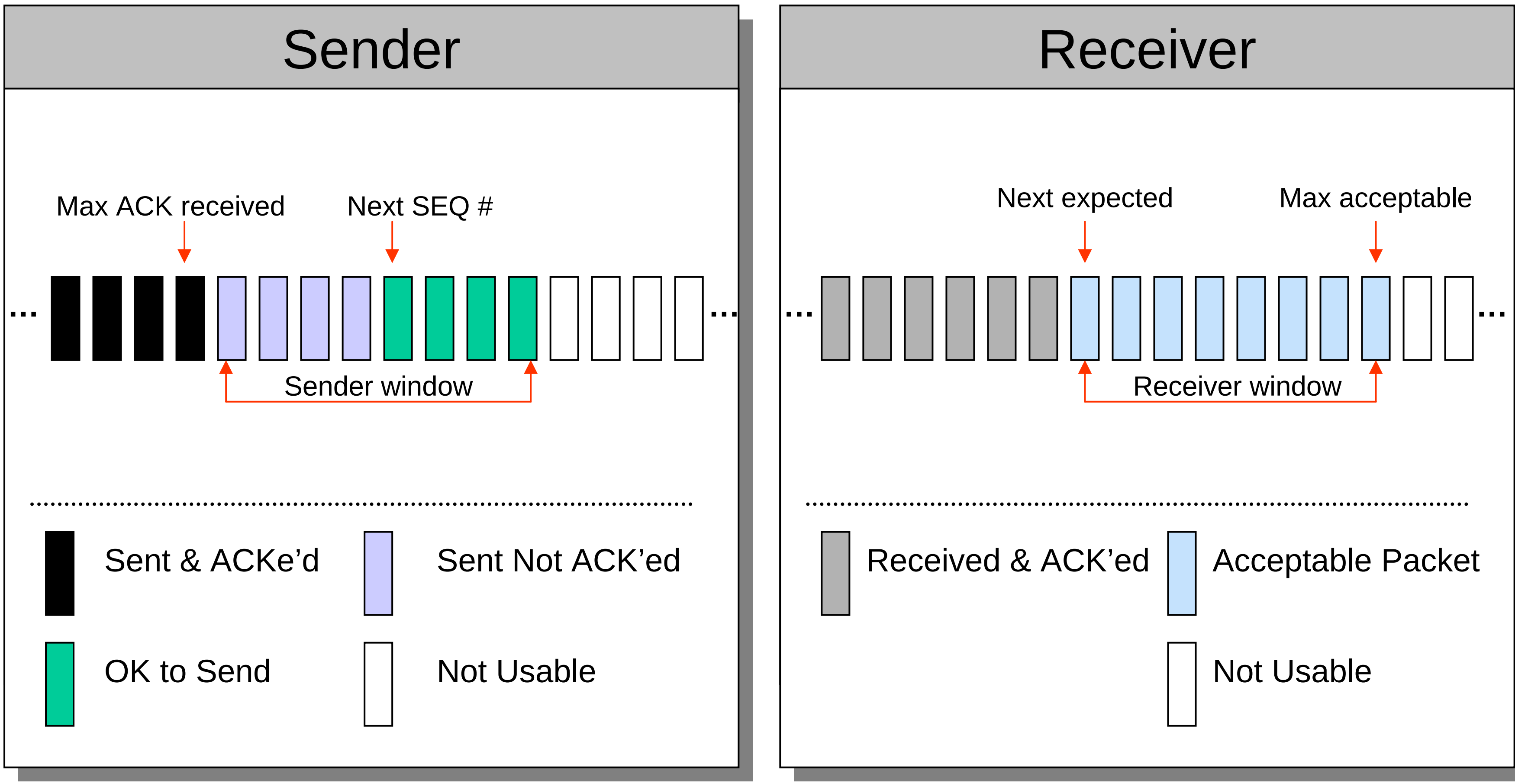
- Receiver
  - Advertises available buffer space (*window*)
- Sender
  - Can send up to entire window before ACK arrives
- Each acknowledgement carries new window information
  - Called *window advertisement*
  - Can be zero (called *closed window*)
- Interpretation: I have received up through *X* and can take *Y* more octets

Sequence Number Space

- Each byte in byte stream is numbered
  - 32 bit value
  - Wraps around
  - Initial values selected at start up time
- TCP breaks up the byte stream in packets
  - Sender divides data stream into individual segments, each no longer than the *sender maximum segment size* (SMSS)
- Each packet has a sequence number
  - Indicates where it fits in the byte stream
- Receiver sends a cumulative ACK notifying the sender that all of the data preceding that segments's SEQ has been received



Sender and Receiver State



Window Sliding

- On reception of new ACK (i.e. ACK for something that was not ACK'ed earlier)

 - Increase sequence of max ACK received

 - Send next packet

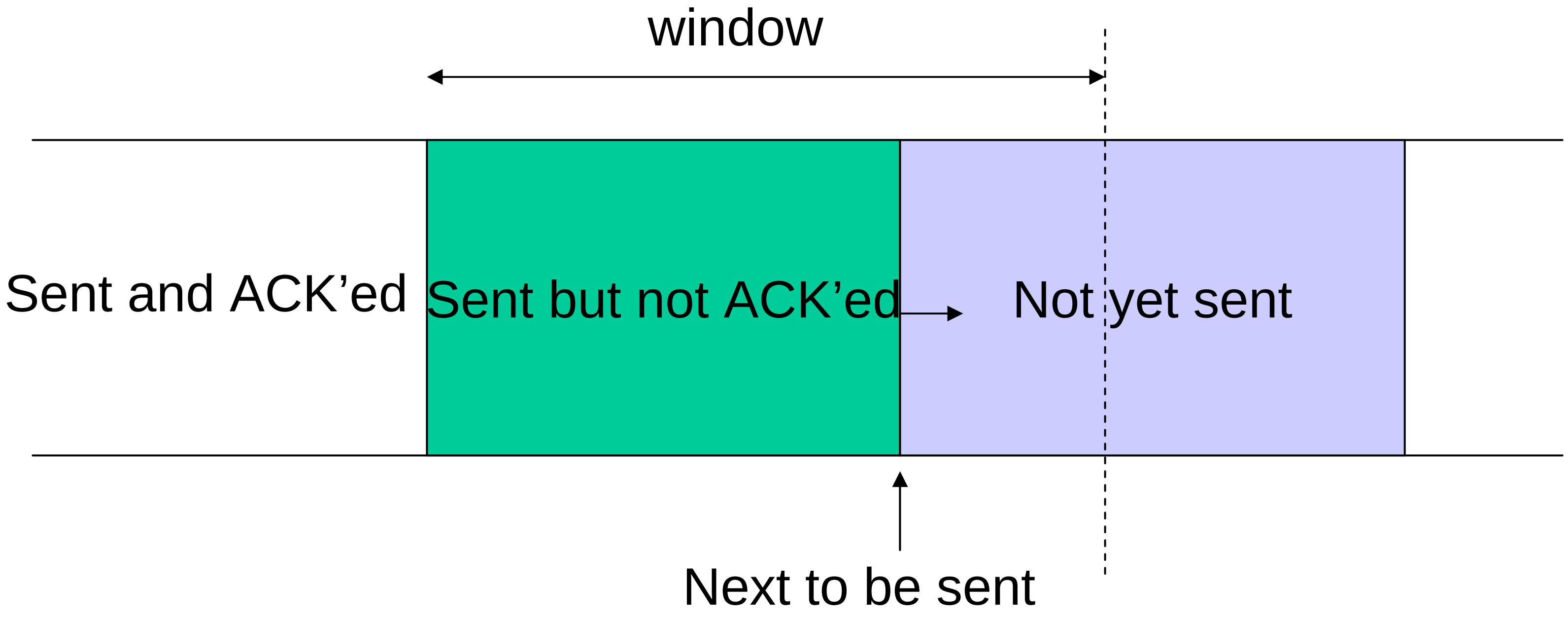
- On reception of new in-order data packet (next expected)

 - Hand packet to application

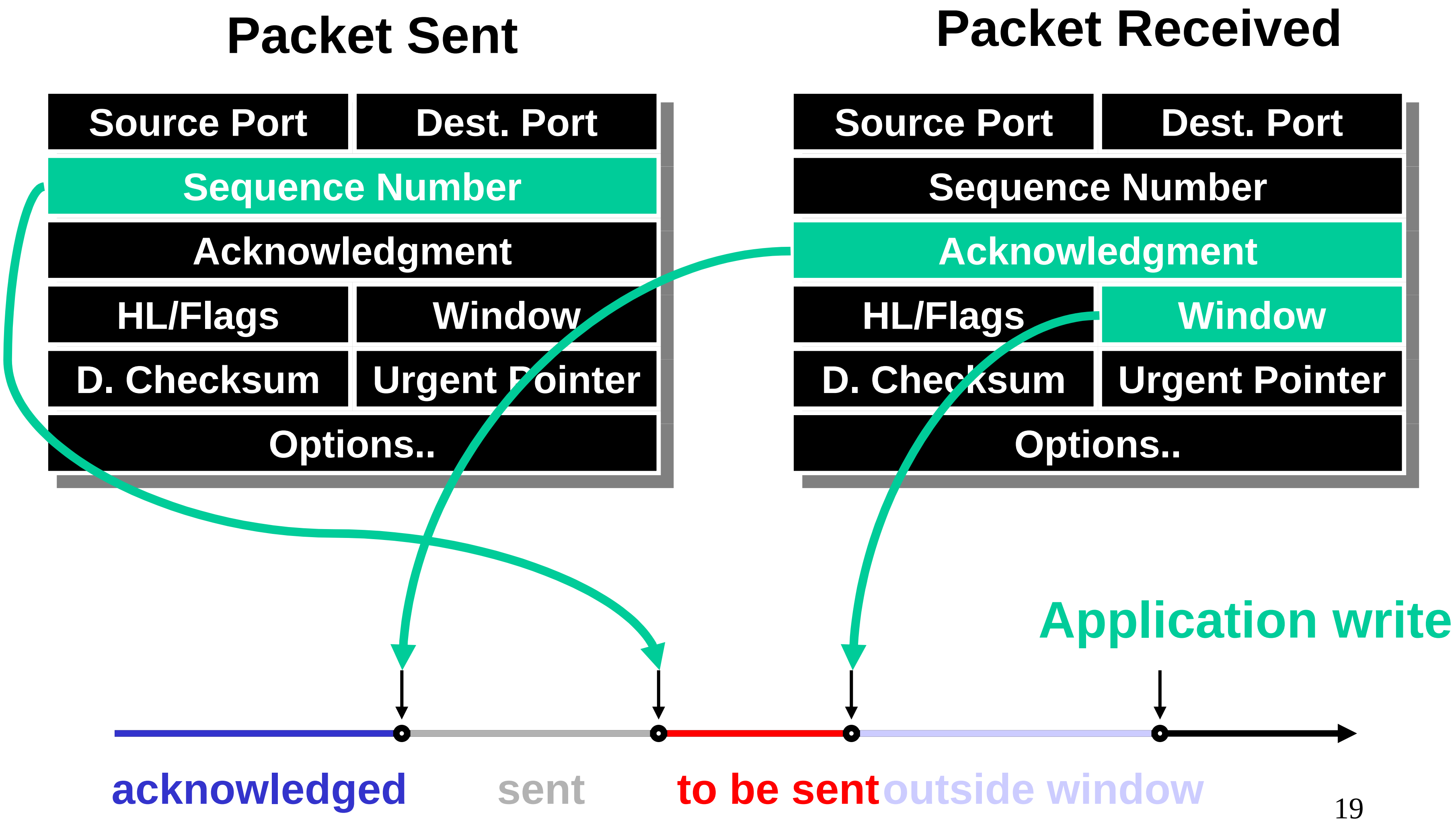
 - Send cumulative ACK – acknowledges reception of all packets up to sequence number

 - Increase sequence of max acceptable packet

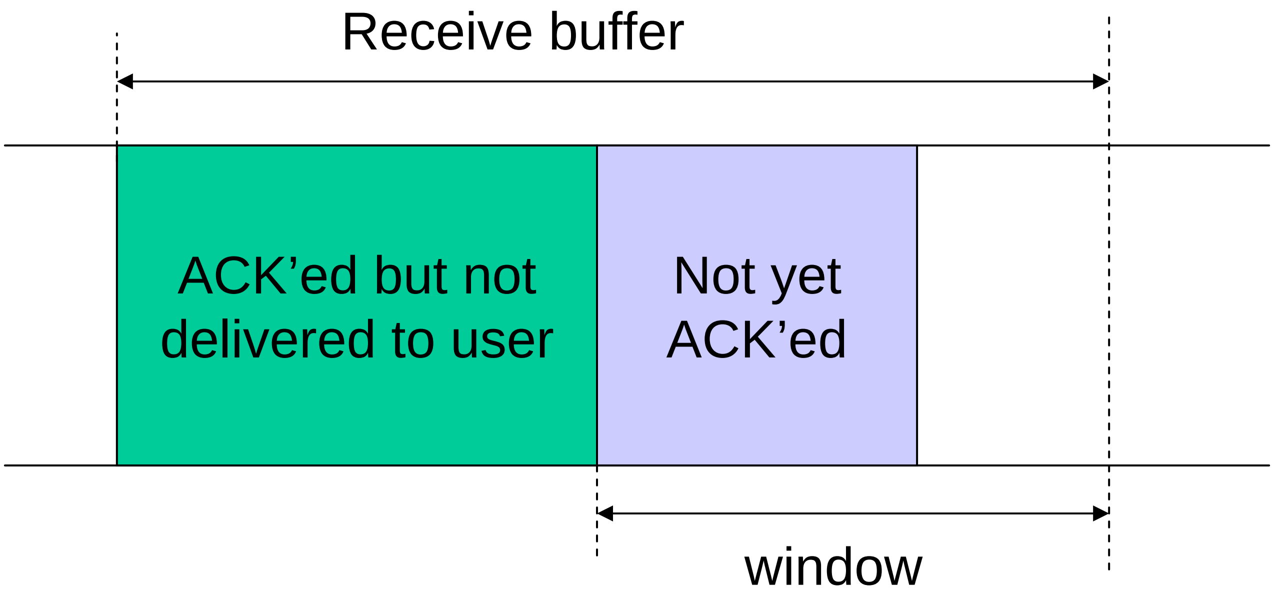
Sender’s Side (1/2)



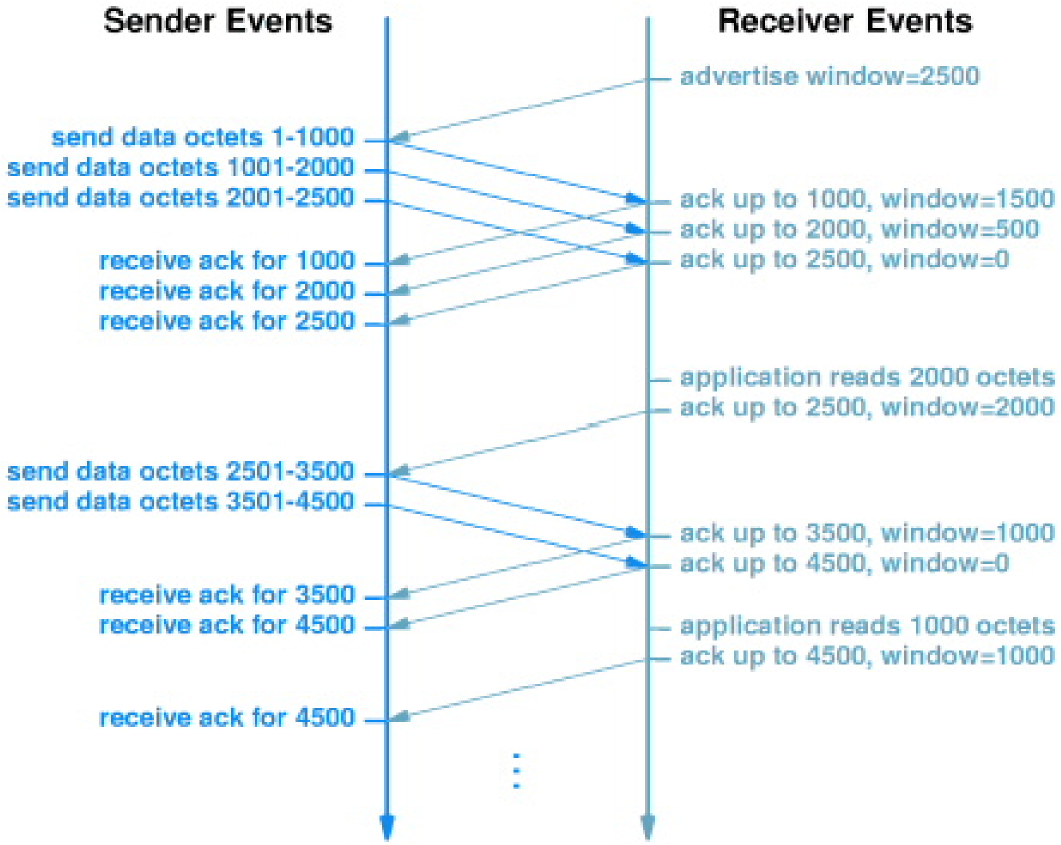
Sender’s Side (2/2)



Receiver's Side



Example


Sender Events
Receiver Events
advertise window=2500
send data octets 1-1000
send data octets 1001-2000
send data octets 2001-2500
ack up to 1000, window=1500
ack up to 2000, window=500
ack up to 2500, window=0
receive ack for 1000
receive ack for 2000
receive ack for 2500
application reads 2000 octets
ack up to 2500, window=2000
send data octets 2501-3500
send data octets 3501-4500
ack up to 3500, window=1000
ack up to 4500, window=0
receive ack for 3500
receive ack for 4500
application reads 1000 octets
ack up to 4500, window=1000
receive ack for 4500

TCP Congestion Control

- A mechanism which:
  - Uses network resources efficiently
  - Preserves fair network resource allocation
  - Prevents or avoids congestion collapse
- Congestion collapse is not just a theory
  - Has been frequently observed in the Internet many times

Congestion Collapse

- Definition: *Increase in network load results in decrease of useful work done*
- Many possible causes

- Illegitimate retransmissions of packets still in flight
  * Classical congestion collapse
  * Solution: Better timers and TCP congestion control
- Undelivered packets
  * Packets consume resources and are dropped elsewhere in network
  * Solution: Congestion control for ALL traffic

Other Congestion Collapse Causes

- Fragments
  - Mismatch of transmission and retransmission units
  - Solutions
    * Make network drop all fragments of a packet
    * Do path MTU discovery
- Control traffic
  - Large percentage of traffic is for control
    * Headers, routing messages, DNS, etc.
- Stale or unwanted packets
  - Packets that are delayed on long queues

Approaches Towards Congestion Control

- Two broad approaches towards congestion control:

- End-end congestion control:
  - No explicit feedback from network
  - Congestion inferred from end-system observed loss, delay
  - Approach taken by TCP

- Network-assisted congestion control:
  - Routers provide feedback to end systems
    - * Single bit indicating congestion (TCP/IP ECN, ATM)
    - * Explicit rate sender should send at

The TCP Approach (1/2)

- TCP interprets packet drops as signs of congestion and slows down
  - This is an assumption: Packet drops are not a sign of congestion in all networks
    * E.g. wireless networks
- Periodically probes the network to check whether more bandwidth has become available
- Diversity in networks makes TCP approach a good solution
  - Dropping packets is universally a natural response to congestion
  - But many open issues: how to isolate poorly behaved sources, diversity in TCP implementations, etc.

The TCP Approach (2/2)

- Underlying design principle: Packet conservation
  - At equilibrium, inject packet into network only when one is removed
- Why was this not working?
  - Connection doesn’t reach equilibrium
  - Illegitimate retransmissions
  - Resource limitations prevent equilibrium
- Packet loss is seen as sign of congestion and results in a multiplicative rate decrease
  - Factor of 2
- TCP periodically probes for available bandwidth by increasing its rate

The TCP Approach: Questions

- How can this be implemented?
  - Operating system timers are very coarse – how do you accurately calculate the transmission rate?
- How does TCP know what is a good initial rate to start with?
  - Should work both for a low bandwidth link (10s of Kbs or less) and for supercomputer links (2.4 Gbs and growing)

TCP Congestion Control Implementation

- Implemented using a congestion window (cwnd) that limits how much data can be in the network
  - TCP also keeps track of how much data is in transit
- Data can only be sent when the amount of outstanding data is less than the congestion window
  - The amount of outstanding data is increased on a ``send'' and decreased on ``ack''
  - (last sent – last ACKed) < congestion window
- Window limited by both congestion and buffering
  - Sender's maximum window = min (advertised window, cwnd)

Congestion Avoidance (1/2)

- If loss occurs when cwnd = W
  - Network can handle 0.5W ~ W segments
  - Set cwnd to 0.5W
- Upon receiving ACK
  - Increase cwnd by 1/cwnd

Congestion Avoidance (2/2)