CPSC 471: Computer Communications

UDP and TCP

Figures from Computer Networks: A Systems Approach, version 6.02dev (Larry L. Peterson and Bruce S. Davie)

Transport Protocol Expectations

- Guarantee message delivery
- In-order delivery of messages
- Delivers one copy of each message
- Supports arbitrarily large messages
- Supports sender/receiver synchronization
- Allows the receiver to apply flow control to the sender
- Supports multiple application processes on each host

Underlying Network Limitations

- Drop messages
- Reorder messages
- Deliver duplicate copies of message
- Limit messages to some finite size
- Deliver messages after an arbitrarily long delay

- A best-effort level of service

A Simple Demultiplexing Protocol

- Extend host-to-host delivery service to a process-to-process communication service
- Adds no other functionality to the best-effort service of the underlying network
- Add a level of demultiplexing
  - Many processes run on a host
  - Allow multiple application processes to access the network
- This is UDP

UDP: User Datagram Protocol

- Address hosts with a port and an address
- Where is the host address?
- How does a process learn the port number of the process it wishes to send to?
  - Contact a server process at a well-known port
    - DNS → #53
    - Mail → #25
  - Contact a port mapper service

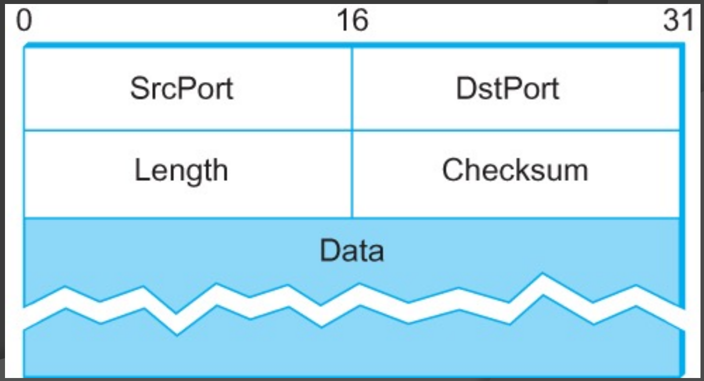

Figure 125

UDP continued

- Port implemented by a message queue
  - No flow control mechanism in UDP
  - Process blocks until message available
- Socket API is an implementation of ports
- Checksum
  - Verify message delivered to correct destination

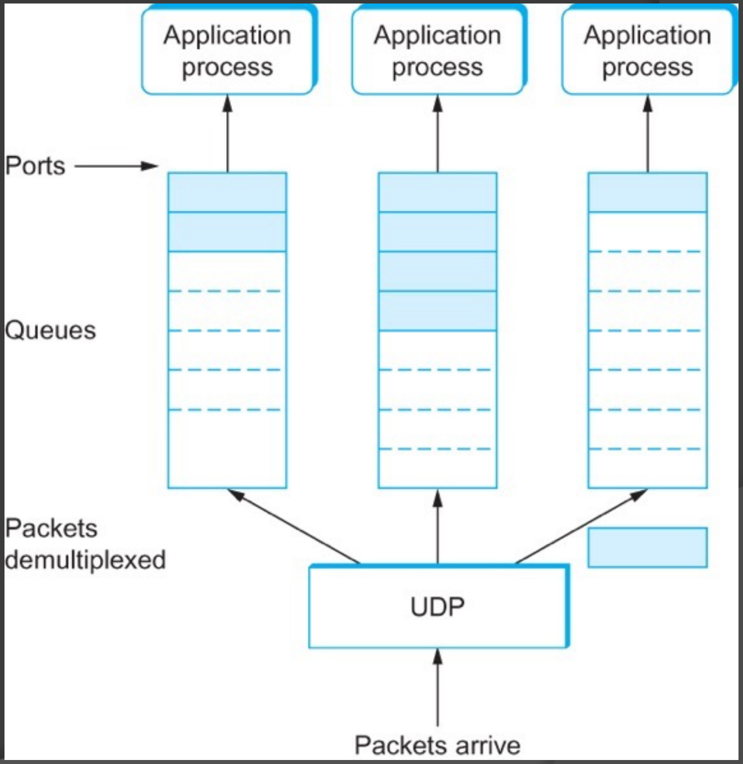

Figure 126

Highlights of TCP: Transmission Control Protocol

- Guarantees reliable, in-order delivery of a byte stream
- Full-duplex
- Flow-control mechanism
- Demultiplexing mechanism
- Congestion-control mechanism
  - Flow control vs. congestion control

TCP Sliding Window Differences

- Logical connection running on 2 hosts
  - Instead of single link connecting 2 hosts
- Requires explicit connection establishment and teardown phases
- Timeouts (for retransmissions) must be adaptive
  - Single link has fixed RTTs
  - TCP connections can have variation in RTTs

TCP Sliding Window Differences continued

- Packets may be reordered as they cross the Internet
  - How late can a packet arrive at its destination?
    - Maximum Segment Lifetime (MSL) = 120 sec
- Window sizes can vary
  - TCP must learn what resources the receiver has
    - Flow control

TCP Sliding Window Differences continued

- TCP sender does not know what links will be traversed going to receiver
  - Slow links
  - Network congestion
- TCP assumes underlying network is
  - Unreliable
  - Delivers messages out of order
- Could use sliding window on a hop-to-hop basis
  - Assumptions for nodes?

TCP Segments

- TCP transmits segments
  - Not individual bytes

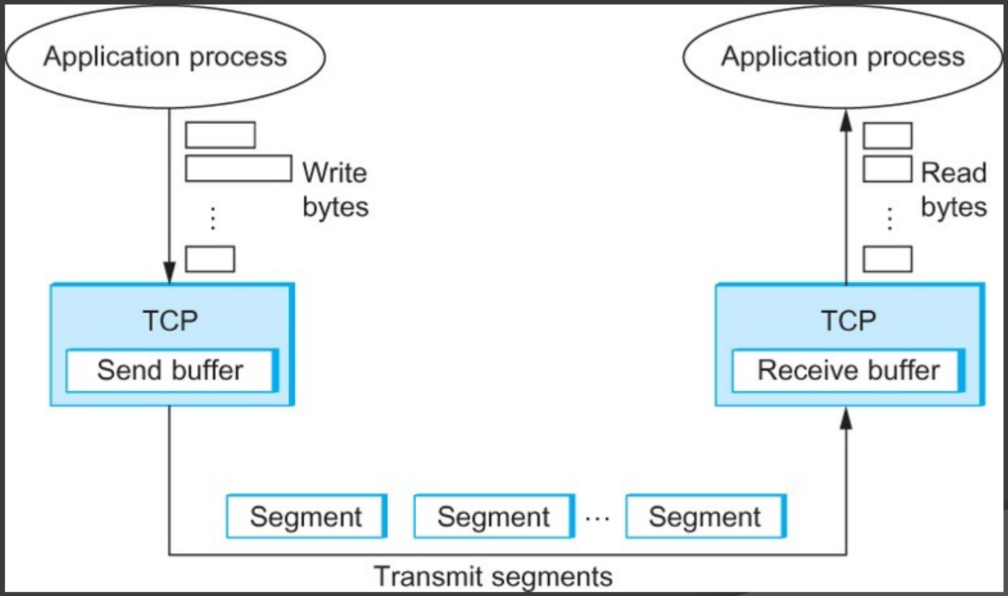

Figure 127

TCP Header Format

- Port numbers and IP addresses form the demux key
- Sequence number for 1st byte of data in seg.
- Flow Control
  - Acknowledgent
  - AdvertisedWindow

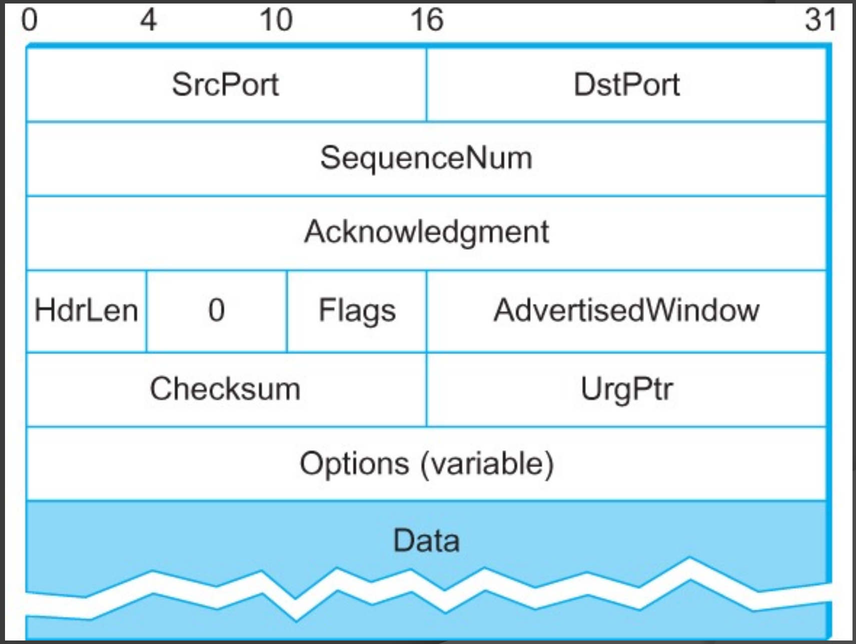

Figure 128

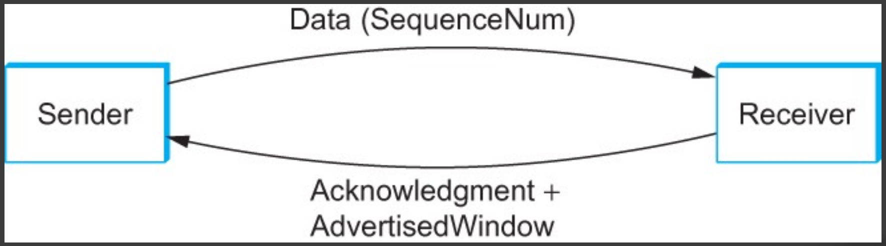

Figure 129

TCP Flags

- SYN: establishing a TCP connection
- FIN: terminating a TCP connection
- ACK: if Acknowledgement field valid
- URG: segment has urgent data
  - UrgPtr points to beginning of non-urgent data
- PUSH: sender used Push operation
- RESET: receiver wishes to abort connection

TCP Connection Establishment/Termination

- Connection setup
  - One side actively opens
  - Other side passively opens
- Connection teardown
  - Each side must close down connection independently

Three-Way Handshake

- Two sides agree on starting sequence numbers
- Why not start with sequence number 0?

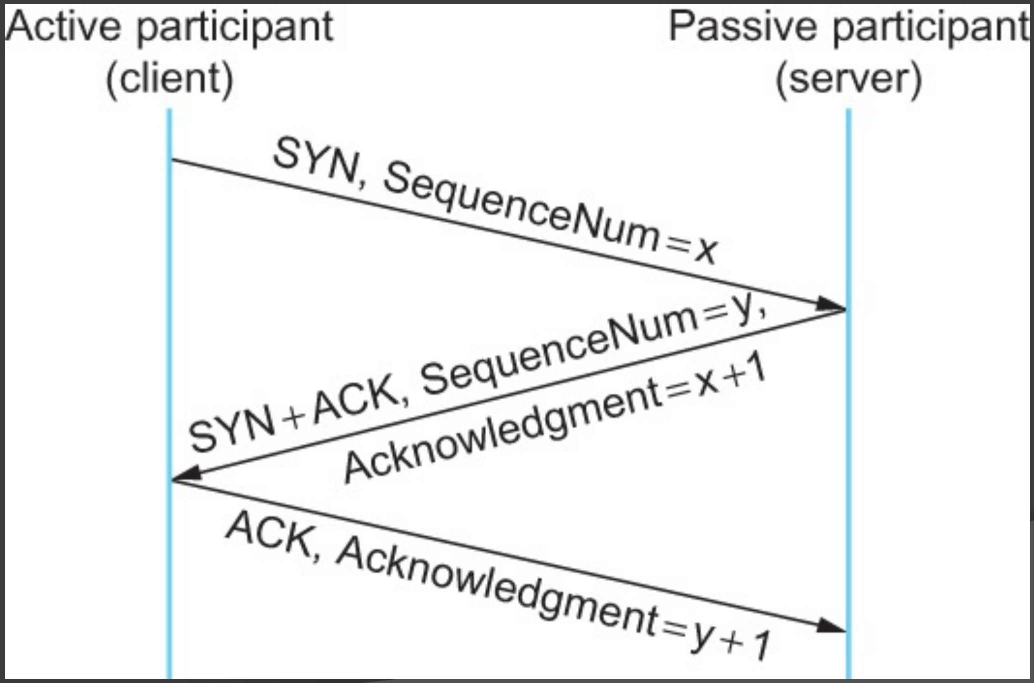

Figure 130

TCP State Transition Diagram

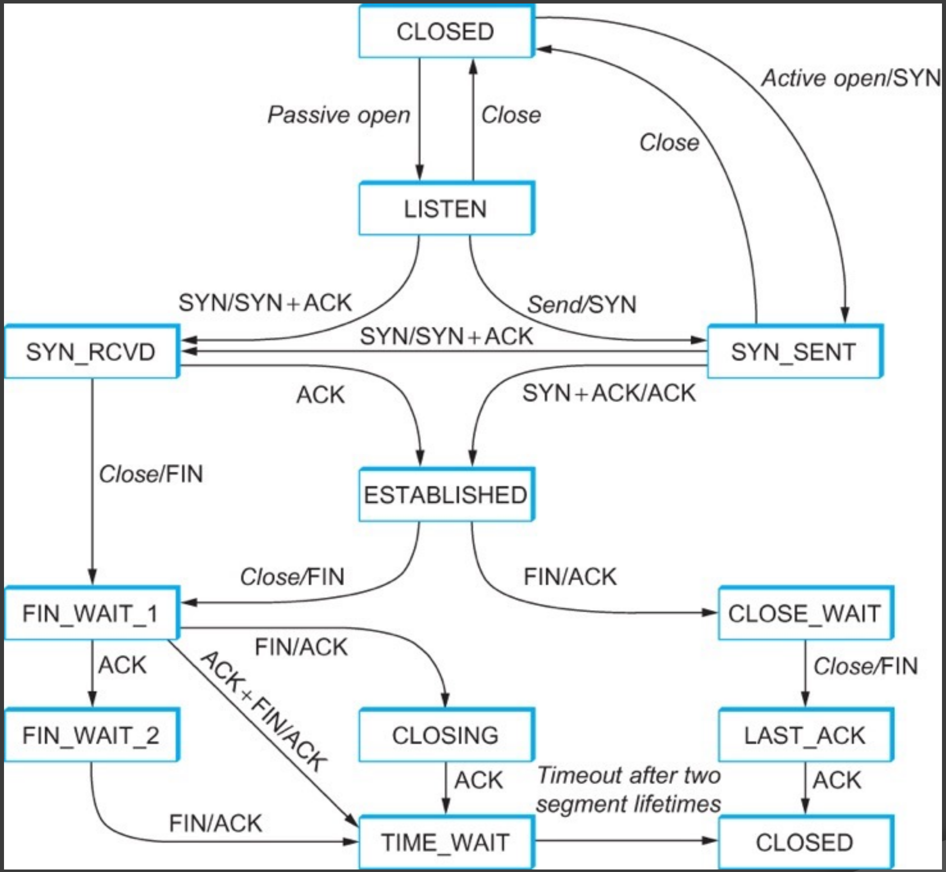

Figure 131

TCP Flow Control

- Receiver advertises a window size
  - Not a fixed-size sliding window
- Sender can have no more than AdvertisedWindow bytes of unacknowledged data
- Receiver chooses AdvertisedWindow based on amount of buffer space available

Send/Receive Buffers

- Send buffer contains data that was
  - sent but not acknowledged
  - written by sending app but not transmitted
- Receive buffer contains data that arrived
  - out of order
  - in order but not yet read by the application process

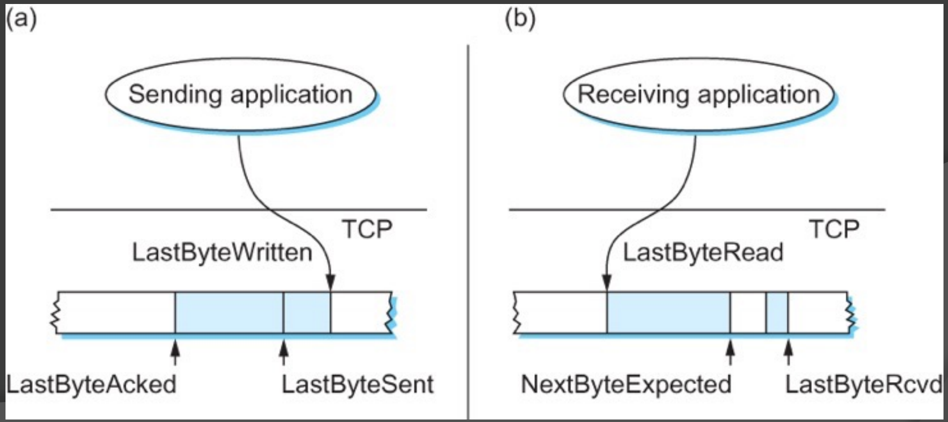

Figure 132

Send/Receive Buffers continued

- Send buffer
  - LastByteAcked ≤ LastByteSent
  - LastByteSent ≤ LastByteWritten
- Receive buffer
  - LastByteRead < NextByteExpected
  - NextByteExpected ≤ LastByteReceived + 1

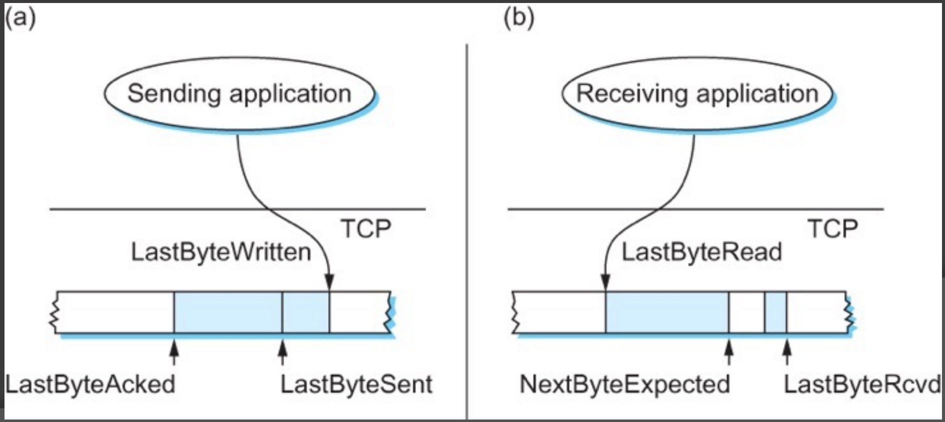

Figure 132

Flow Control

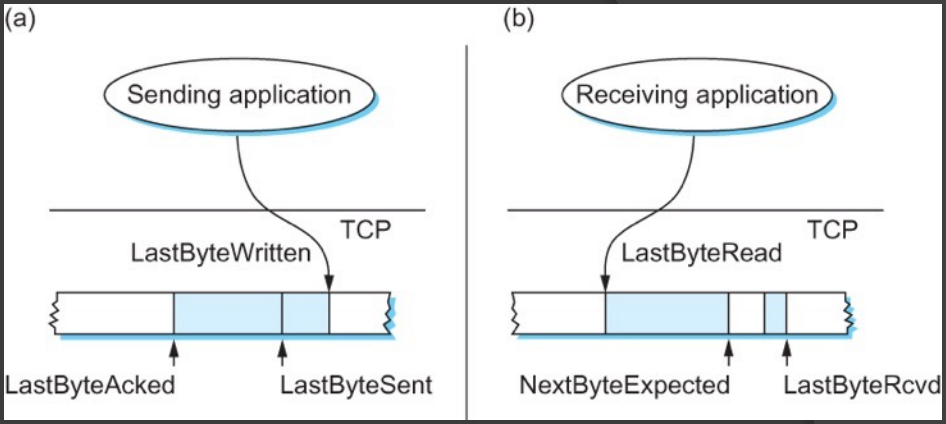

Figure 132

- Buffers have finite size
- Send window-size amount of data
  - Receiver throttles sender
- Receiver keeps
  - LastByteReceived – LastByteRead ≤ MaxRcvBuffer
  - Advertises Window Size of
    - MaxRcvBuffer – ((NextByteExpected – 1) – LastByteRead)

Shrink/Growth of Advertised Window

- Cumulative acknowledgement
- LastByteRcvd
- LastByteRead

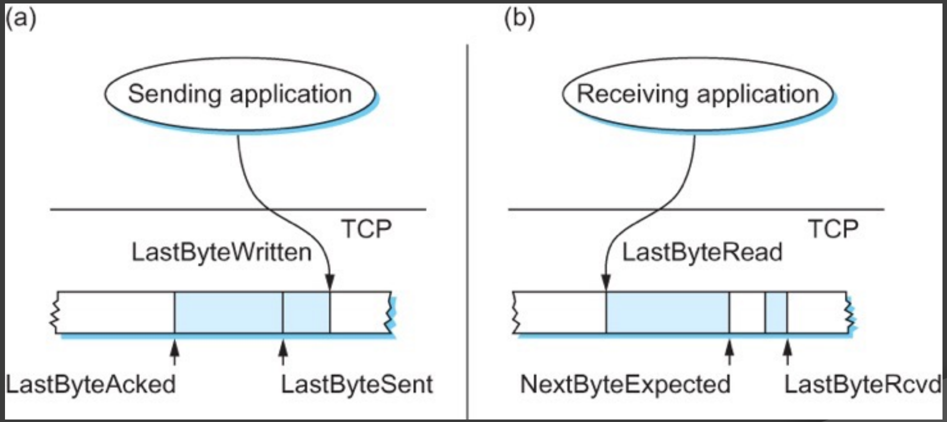

Figure 132

Flow Control continued

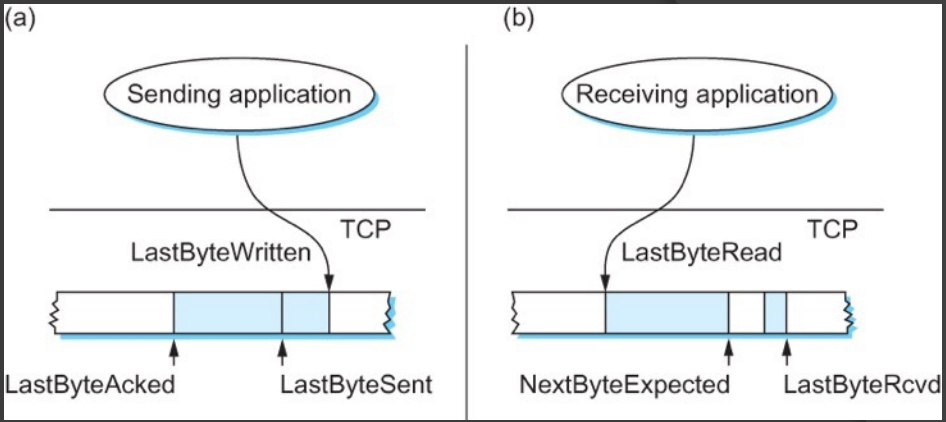

Figure 132

- Sender ensures
  - LastByteSend – LastByteAcked ≤ AdvertisedWindow
  - Computes effective window size of
    - AdvertisedWindow – (LastByteSent – LastByteAcked)
- Sender ensures local app does not overflow send buffer
  - LastByteWritten – LastByteAcked ≤ MaxSendBuffer
  - TCP blocks the sending process

Flow Control continued

- Can the sender's window size ever be larger than the receiver's window size? Why not?

Flow Control continued

- How does a slow process stop a fast-sending process?

Flow Control continued

- How does the sender know the advertised window is no longer 0?
- TCP sends segment in response to a received data segment
- If Advertised Window == 0
  - Sender not allowed to send more data

Flow Control continued

- Sender sends a 1-byte segment periodically
  - Avoids halting the transmission

Wraparound

- 32-bit sequence number may wrap around
  - Too small?
- Ensure wrap-around does not occur within MSL
  - Maximum segment lifetime
  - Typically, 120 sec
    - Note, for 10GigE, wraparound occurs in 3 sec
- Note, sequence numbers may still go from 2^{32}-1 to 0, even if sending a small amount of data. Why?

Keeping the Pipe Full

- 16-bit advertised window
  - Too small
- Must be large enough for full delay x bandwidth product worth of information to be transmitted

TCP Segment Transmission

- How many bytes in a TCP segment?
  - Maintains a maximum segment size (MSS)
    - Usually set to largest segment without IP fragmentation occurring
  - Sending process explicitly tells TCP to send
    - Push operation
  - A periodic timer

- What if sender has MSS bytes to send and receiver can only accept MSS/2 bytes?

Silly Window Syndrome

- Early TCP implementations sent half-full segment
- How to combine small segments?

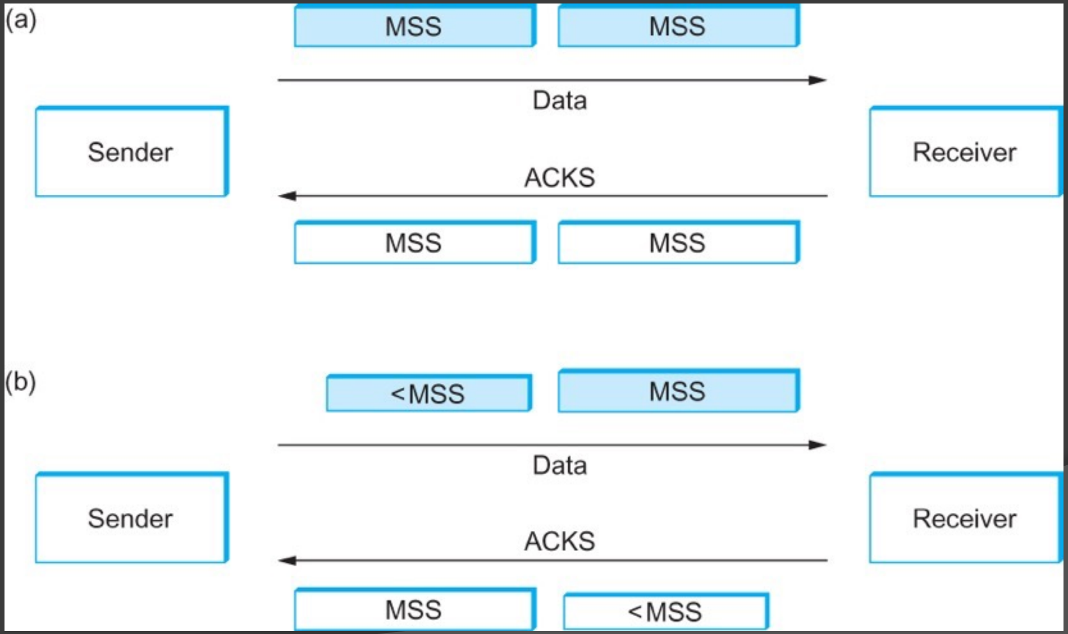

Figure 133

Silly Window Syndrome continued

- May want to wait for large window size
  - But wait how long?
    - Too short → Silly Window Syndrome
    - Too long → Hurt interactive app performance
- Nagle's Algorithm (self-clocking solution)
  - Send full segment if the window allows
  - Send smaller segment if no segments are in transit
  - Wait for an ACK before sending next segment if there is anything in flight

Nagle's Algorithm Example

- Suppose you are sending 8 bytes of data at a rate of 1 byte per second over a TCP connection with a RTT of 3.6 sec.
- Construct a timeline of this data transfer

Solution to Nagle's Algorithm Example (1/2)

- Assume we send: 12345678
- At t = 0 sec: Send "1" (first byte) since no segments are in transit
- At t = 1 sec: Wait to send "2" (second byte) since first byte is in transit
- At t = 1.8 sec: "1" (first byte) is received and receiver sends an ACK
- At t = 2 sec: Wait to send "3" since first byte is in transit (not yet received ACK)
- At t = 3 sec: Wait to send "4" since first byte is in transit (not yet received ACK)
- At t = 3.6 sec: Sender receives ACK for "1" (first byte) and sends "234" since no segments are in transit

Solution to Nagle's Algorithm Example (2/2)

- At t = 4 sec: Wait to send "5" since "234" is in transit
- At t = 5 sec: Wait to send "6" since "234" is in transit
- At t = 5.4 sec: "234" is received and receiver sends an ACK
- At t = 6 sec: Wait to send "7" since "234" is in transit (not yet received ACK)
- At t = 7 sec: Wait to send "8" since "234" is in transit (not yet received ACK)
- At t = 7.2 sec: Sender receives ACK for "234" and sends "5678" since no segments are in transit
- At t = 9 sec: "5678" is received and receiver sends an ACK
- At t = 10.8 sec: Sender receives ACK for "5678"

Adaptive Retransmission

- Timeout is a function of expected RTT
  - Range in RTTs in the Internet
  - Variation over time of RTTs between hosts
- Keep a running average of RTT
  - Calculate timeout as a function of this
- Issues:

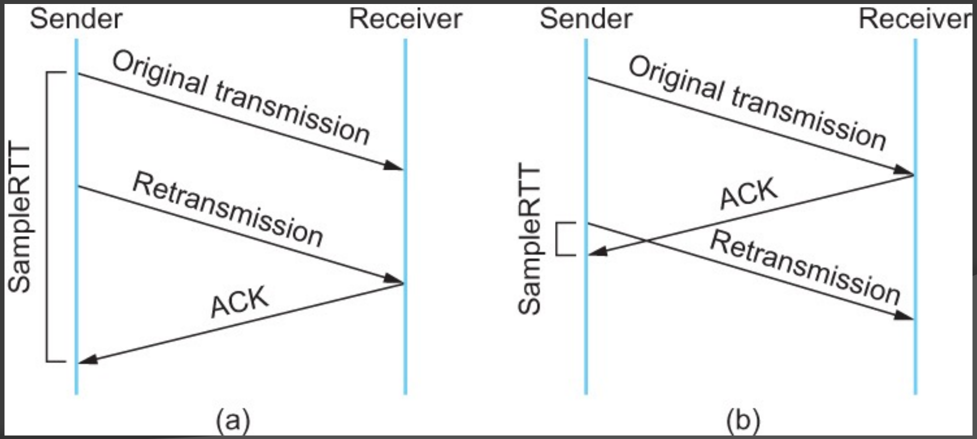

Figure 134

Adaptive Retransmission continued

- Karn/Partridge Algorithm
  - Only measure the sample RTT for segments that have only been sent once
  - Exponential backoff
- Jacobson/Karels Algorithm
  - Calculates both the mean and variation in the mean

Record Boundaries

- TCP does not inject record boundaries in the bytes
  - Sender may write 8, 2, then 20 bytes
  - Receiver may read 5 bytes for six times
- Use the urgent data feature to signify special data (record marker)
- Sending application can use the Push command
- Application can insert its own record boundaries

TCP Extensions

- Options that can be added to TCP header
- Improve TCP’s timeout mechanism
  - Use timestamp of actual system clock
- Sequence number wrap-around
  - Use 32-bit timestamp with 32-bit sequence number
- Advertise a larger window
  - Fill high-speed network’s larger delay x bandwidth pipes
  - Add a scaling factor for the advertised window

TCP Extensions continued

- Add selective acknowledgements to cumulative acknowledgements
  - Allows sender just to retransmit missing segments
  - Versus
    - Retransmitting just the timed-out segment
    - Retransmitting the segment plus all subsequent frames
- TCP continues to perform well as network speeds increase
  - Due to extensions

Stream-Oriented Protocols vs. Request-Reply Protocols

- Byte-oriented vs. message-oriented
- Reliable vs. unreliable
- Reply-request message
  - Requires 9 TCP segments
- Upper bound on size for message-oriented protocols

TCP Alternative Design Choices

- TCP delivers bytes in order
  - Stream Control Transmission Protocol (SCTP)
    - Provides partially ordered service
- Explicit setup/teardown phases
  - Sender may reject connection before data arrives
- Window-based protocol
  - Could have rate-based design
    - E.g., receiver advertises it can accommodate 100 packets/sec