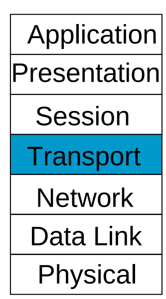
CSE 461: Sliding Windows & ARQ

Next Topic

- We begin on the Transport layer
- Focus
  - How do we send information <u>reliably</u>?
- Topics
  - The Transport layer
  - Acknowledgements and retransmissions (ARQ)
  - Sliding windows



The Transport Layer

- Builds on the services of the Network layer
- Communication between processes running on hosts
  - Naming/Addressing
- Stronger guarantees of message delivery
  - Reliability

Example – Common Properties

TCP

- Connection-oriented
- Multiple processes
- Reliable byte-stream delivery
  - In-order delivery
  - Single delivery
  - Arbitrarily long messages
- Synchronization
- Flow control
- Congestion control

IP

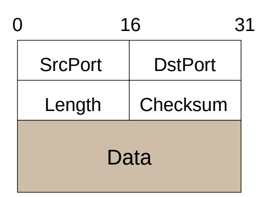
- Datagram oriented
- Lost packets
- Reordered packets
- Duplicate packets
- Limited size packets

What does it mean to be "reliable"

- How can a sender "know" the sent packet was received?
  - sender receives an acknowledgement
- How can a receiver "know" a received packet was sent?
  - sender includes sequence number, checksum
- Do sender and receiver need to come to consensus on what is sent and received?
  - When is it OK for the receiver's TCP/IP stack to deliver the data to the application?

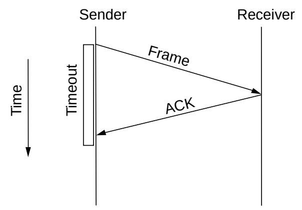
Internet Transport Protocols

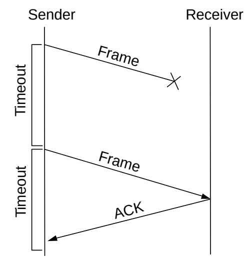
- UDP
  - Datagram abstraction between processes
  - With error detection



- TCP
  - Bytestream abstraction between processes
  - With reliability
  - Plus congestion control (next week)

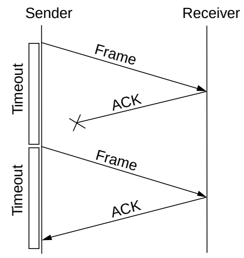
Automatic Repeat Request (ARQ)

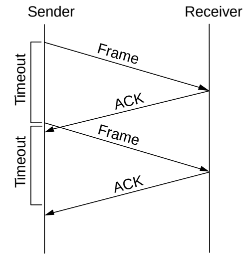




- Packets can be corrupted or lost. How do we add reliability?
- Acknowledgments (ACKs) and retransmissions after a timeout
- ARQ is generic name for protocols based on this strategy

The Need for Sequence Numbers

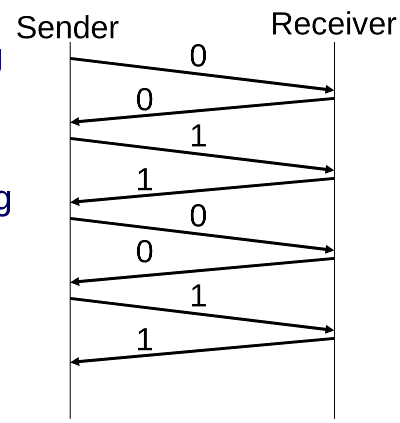




- In the case of ACK loss (or poor choice of timeout) the receiver can't distinguish this message from the next
  - Need to understand how many packets can be outstanding and number the packets; here, a single bit will do

Stop-and-Wait

- Only one outstanding packet at a time
- Also called alternating bit protocol

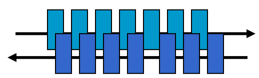


Limitation of Stop-and-Wait

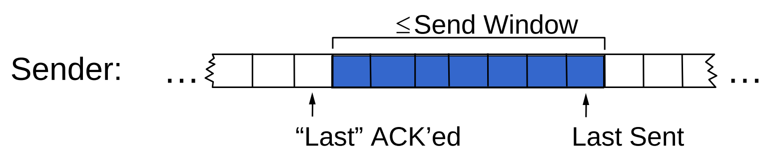
- Lousy performance if trans. delay << prop. delay
  - Max BW: B
  - Actual BW: M/2D
    - Example: B = 100Mb/s, M=1500Bytes, D=50ms
    - Actual BW = 1500Bytes/100ms --> 15000 Bytes/s --> ~100Kb/s
    - 100Mb vs 100Kb?

More BW Please

- Want to utilize all available bandwidth
  - Need to keep more data "in flight"
  - How much? Remember the bandwidth-delay product?
- Leads to Sliding Window Protocol
  - "window size" says how much data can be sent without waiting for an acknowledgement

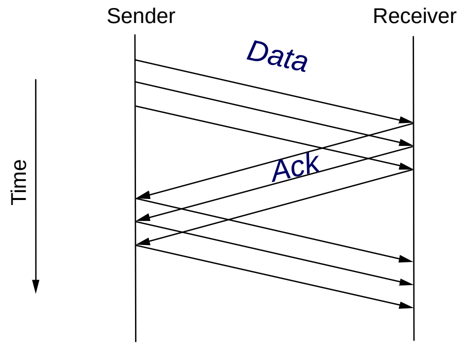


Sliding Window – Sender



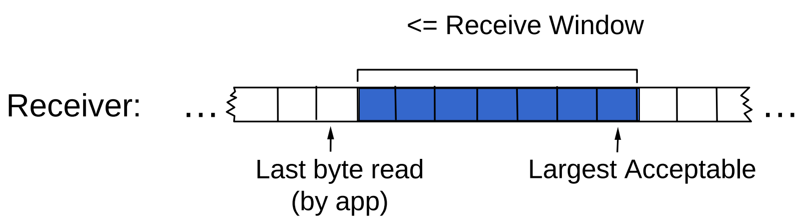
- Window bounds outstanding data
  - Implies need for buffering at sender
    - Specifically, must buffer unack'ed data
- "Last" ACK applies to in-order data
  - Need not buffer acked data
- Sender maintains timers too
  - Go-Back-N: one timer, send all unacknowledged on timeout
  - Selective Repeat: timer per packet, resend as needed

Sliding Window – Timeline



- Receiver ACK choices:
  - –Individual
    - Each packet acked
  - –Cumulative (TCP)
    - Ack says "got everything up to X-1…"
    - really, "my ack means that the next byte I am expecting is X"
  - –Selective (newer TCP)
    - Ack says "I got X through Y"
  - – Negative
    - Ack says "I did not get X"

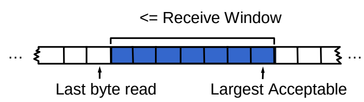
Sliding Window – Receiver



- Receiver buffers too:
  - data may arrive out-of-order
  - or faster than can be consumed by receiving process
- No sense having more data on the wire than can be buffered at the receiver.
  - In other words, receiver buffer size should limit the sender's window size

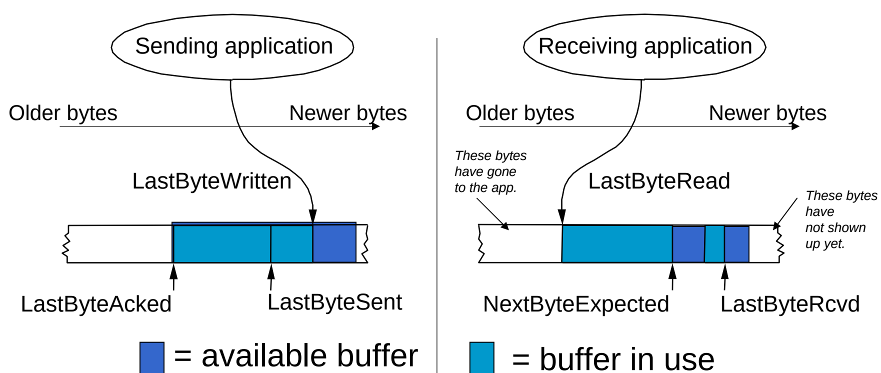
Flow Control

- Sender must transmit data no faster than it can be consumed by receiver
  - Receiver might be a slow machine
  - App might consume data slowly



- Accomplish by adjusting the size of sliding window used at the sender
  - sender adjusts based on receiver's feedback about available buffer space
  - the receiver tells the sender an "Advertised Window"

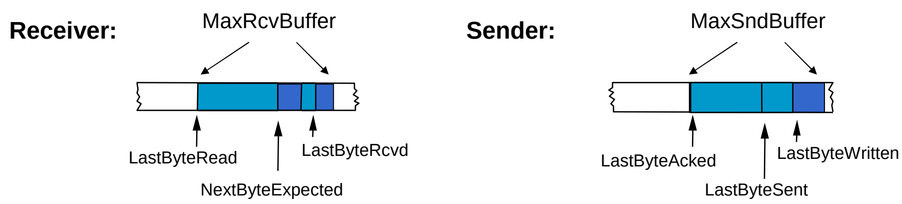
Sender and Receiver Buffering



LastByteAcked <= LastByteSent
LastByteSent <= LastByteWritten

LastByteRead < NextByteExpected
NextByteExpected <= LastByteRvcd+1
== if data arrives in order
else start of first gap.

Flow Control



Receiver's goal: always ensure that LastByteRcvd - LastByteRead <= MaxRcvBuffer

- in other words, ensure it never needs to buffer more than MaxRcvBuffer data

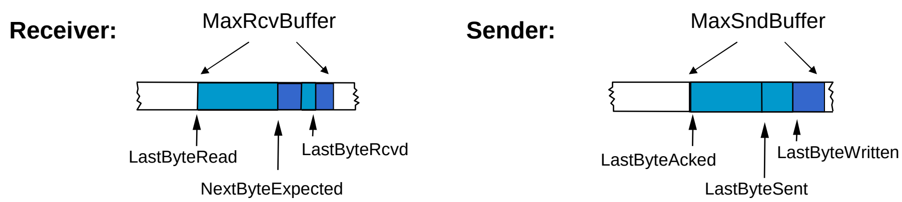
To accomplish this, receiver advertises the following window size:

- AdvertisedWindow = MaxRcvBuffer - ((NextByteExpected - 1) - LastByteRead)
- *"All the buffer space minus the buffer space that's in use."*

Flow control on the receiver

- As data arrives:
  - receiver acknowledges it so long as all preceding bytes have also arrived
  - ACKs also carry a piggybacked AdvertisedWindow
  - So, an ACK tells the sender:
    1. All data up to the ACK'ed seqno has been received
    2. How much more data fits in the receiver's buffer, as of receiving the ACK'ed data
- AdvertisedWindow:
  - shrinks as data is received
  - grows as receiving app. reads the data from the buffer

Flow Control On the Sender



Sender's goal: always ensure that LastByteSent - LastByteAcked <= AdvertisedWindow

- in other words, don't sent that which is unwanted

Notion of "EffectiveWindow": how much new data it is OK for sender to currently send

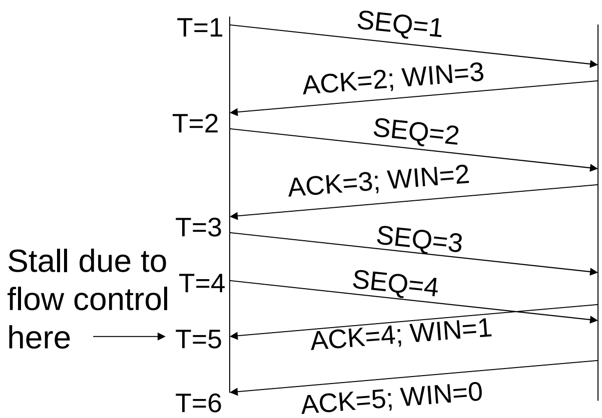
- EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)

OK to send that which there is room for, which is that which was advertised (AdvertisedWindow) minus that which I've already sent since receiving the last advertisement.

Sending Side

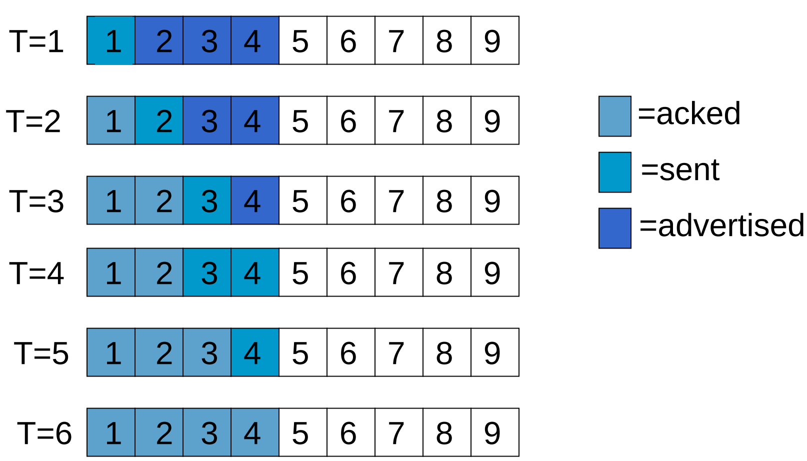
- As acknowledgements arrive:
  - advance LastByteAcked
  - update AdvertisedWindow
  - calculate new EffectiveWindow
    - If EffectiveWindow > 0, it is OK to send more data
- One last detail on the sender:
  - sender has finite buffer space as well
    - LastByteWritten - LastByteAcked <= MaxSendBuffer
  - OS needs to block application writes if buffer fills
    - i.e., block write(y) if
      (LastByteWritten - LastByteAcked) + y > MaxSendBuffer

Example – Exchange of Packets



Receiver has buffer of size 4 and application doesn't read

Example – Buffer at Sender



Packet Format

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

16 bit window size gets
Cramped with large
Bandwidth x delay

16 bits --> 64K
BD ethernet: 122KB
STS24 (1.2Gb/s): 14.8MB

32 bit sequence number
must not wrap around faster
than the maximum packet
lifetime. (120 seconds)
-- 622Mb/s link: 55 seconds

Sliding Window Functions

- Sliding window is a mechanism
- It supports multiple functions:
  - Reliable delivery
    - *If I hear you got it, I know you got it.*
    - ACK (Ack # is "next byte expected")
  - In-order delivery
    - *If you get it, you get it in the right order.*
    - SEQ # (Seq # is "the byte this is in the sequence")
  - Flow control
    - *If you don't have room for it, I won't send it.*
    - Advertised Receiver Window
    - AdvertisedWindow is amount of free space in buffer

Key Concepts

- Transport layer allows processes to communicate with stronger guarantees, e.g., reliability
- Basic reliability is provided by ARQ mechanisms
  - Stop-and-Wait through Sliding Window plus retransmissions