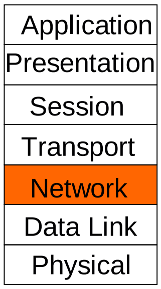
CSE/EE 461

Sliding Windows and ARQ

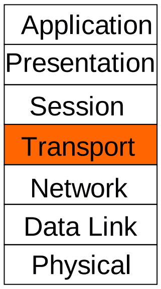
Last Time

- We finished up the Network layer
  - Internetworks (IP)
  - Routing (DV/RIP, LS/OSPF)
- It was all about routing: how to provide end-to-end delivery of packets.



This Time

- We begin on the Transport layer
- Focus
  - How do we send information reliably?
- 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
- Byte-stream delivery
  - In-order delivery
  - Single delivery
  - Arbitrarily long messages
- Synchronization
- Flow control
- Reliable delivery

IP

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

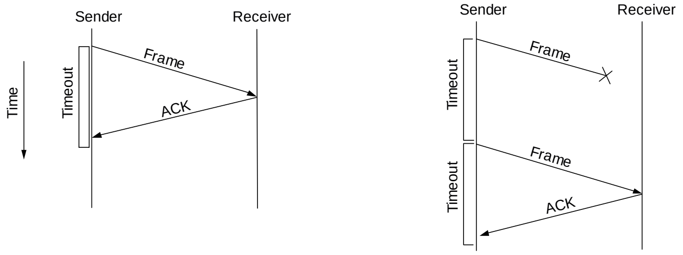
What does it mean to be "reliable"

- Some options:
  1. A packet sent is a packet received
  2. A packet not sent is a packet not received
  3. A packet received is a packet sent
  4. A packet not received is a packet not sent
  5. An acknowledged packet means the packet was received
  6. *A received acknowledgement for a packet sent means the packet was received*
  7. An unreceived acknowledgment for a packet sent means the packet was not received

Internet Transport Protocols

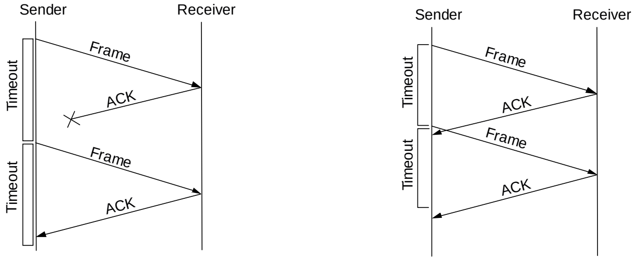
- UDP
  - Datagram abstraction between processes
  - With error detection
- TCP
  - Bytestream abstraction between processes
  - With reliability
  - Plus congestion control (later!)

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

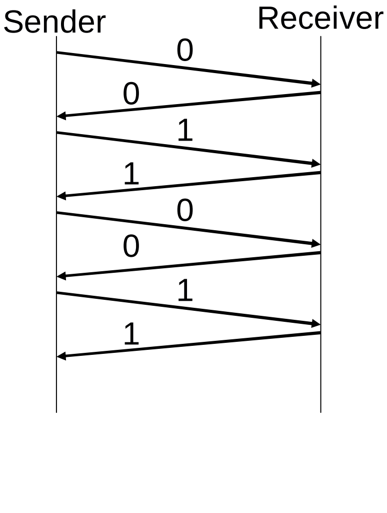


- In the case of ACK loss (or poor choice of timeout) the receiver can’t distinguish this message from the next
  - Need to understand how many packets can be outstanding and number the packets; here, a single bit will do

Stop-and-Wait

- Only one outstanding packet at a time

- Also called alternating bit protocol

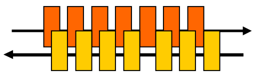


Limitation of Stop-and-Wait

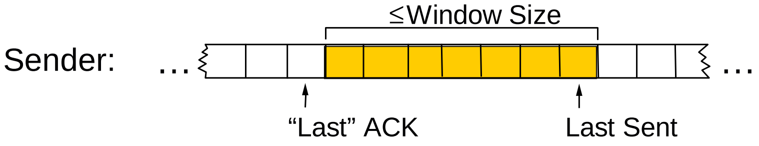
- Lousy performance if wire time << prop. delay
  - Max BW: B
  - Actual BW: M/2D
    - Example: B = 100Mb/s, M=1500Bytes, D=50ms
    - Actual BW = 1500Bytes/100ms --> 15000 Bytes/s --> 100Kb/s
    - 100Mb vs 100Kb?

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

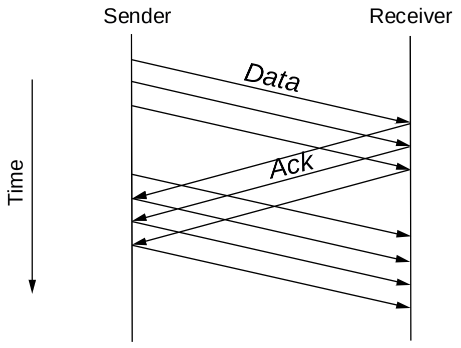


Sliding Window – Sender



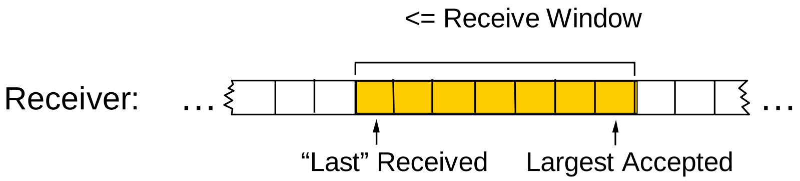
- Window bounds outstanding data
  - Implies need for buffering at sender
    - Specifically, must buffer unacked data
- "Last" ACK applies to in-order data
  - Need not buffer acked data
- Sender maintains timers too
  - Go-Back-N: one timer, send all unacknowledged on timeout
  - Selective Repeat: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

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)
    - Acks says "I got X through Y"
  - Negative
    - Acks says "I did not get X"

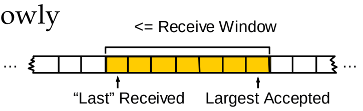
Sliding Window – Receiver



- 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 limits the window size

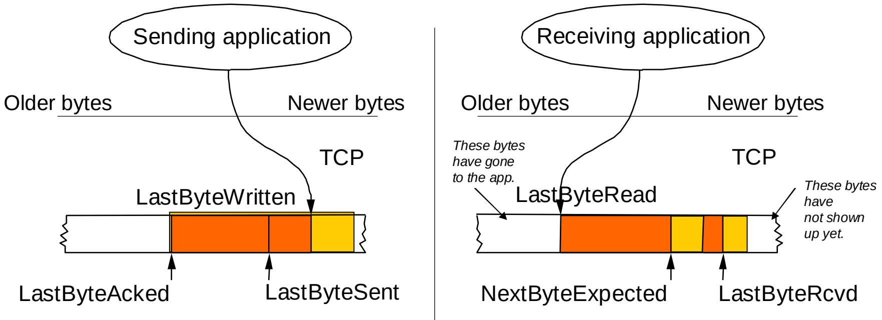
Flow Control

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



- Implement by adjusting the size of the sliding window used at the sender based on receiver feedback about available buffer space
  - This is the purpose of the Advertised Window field

Sender and Receiver Buffering



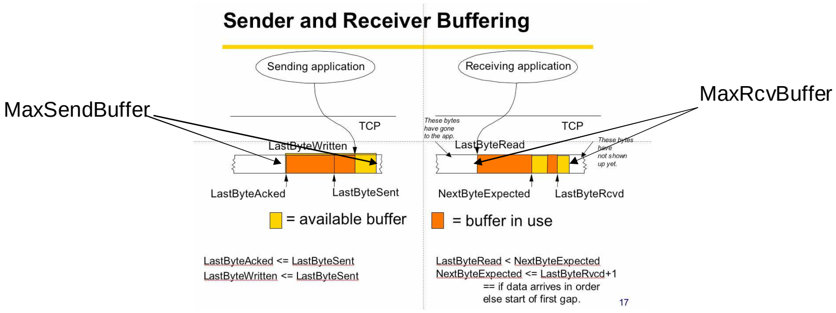
🟨 = available buffer

🟧 = buffer in use

LastByteAcked <= LastByteSent
LastByteWritten >= LastByteSent

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

Flow Control



Sender and Receiver Buffering

🟨 = available buffer

🟧 = buffer in use

LastByteAcked <= LastByteSent
LastByteWritten <= LastByteSent

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

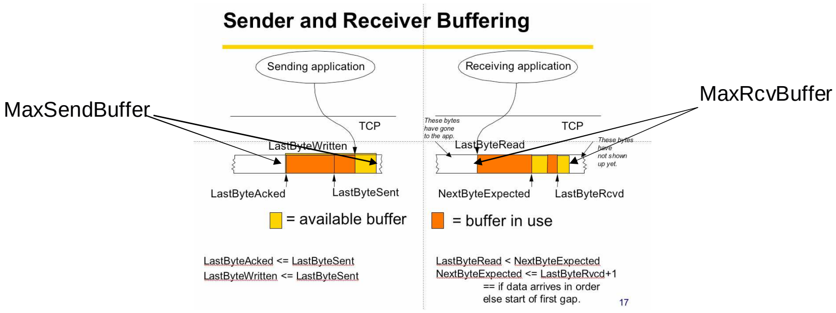
17

LastByteRcvd - LastByteRead <= MaxRcvBuffer

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

As data arrives, receiver acknowledges it so long as all preceding bytes have also arrived. Advertised Window potentially shrinks depending on how fast receiving app is drawing out Data.

Flow Control On the Sender



Sender and Receiver Buffering

🟨 = available buffer 🟧 = buffer in use

LastByteAcked <= LastByteSent
LastByteWritten <= LastByteSent

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

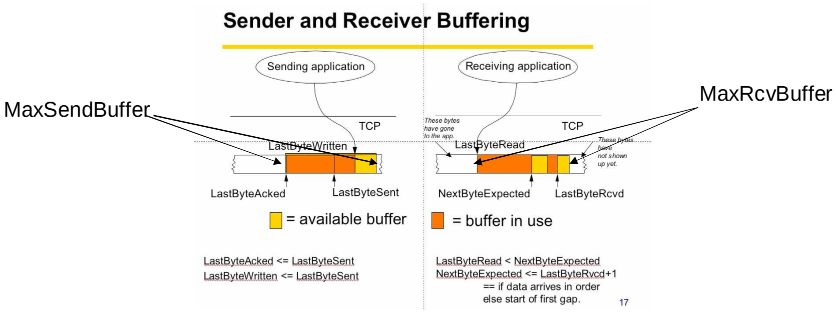
17

LastByteSent - LastByteAcked <= AdvertisedWindow *‘don’t send that which is unwanted.’*

EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)

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

Sending Side -- One last detail



Sender and Receiver Buffering

🟨 = available buffer 🟧 = buffer in use

LastByteAcked <= LastByteSent
LastByteWritten <= LastByteSent

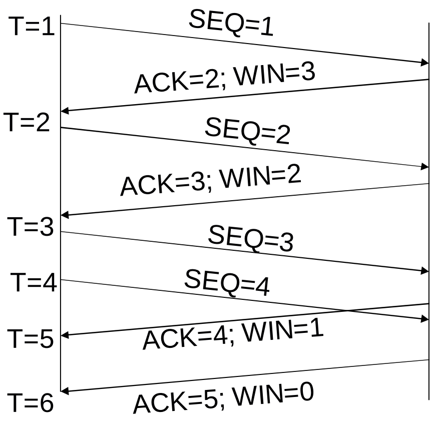
LastByteRead < NextByteExpected
NextByteExpected <= LastByteRvcd+1
== if data arrives in order
else start of first gap.

17

LastByteWritten - LastByteAcked <= MaxSendBuffer
Can only hang on to unsent and unacked data if there’s room for it.

==> *BLOCK write(y) if*
(LastByteWritten - LastByteAcked) + y > MaxSendBuffer

Example – Exchange of Packets



Stall due to flow control here

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

Example – Buffer at Sender

T=1	🟧 1	🟨 2	🟨 3	🟨 4	5	6	7	8	9
T=2	🟩 1	🟧 2	🟨 3	🟨 4	5	6	7	8	9
T=3	🟩 1	🟩 2	🟧 3	🟨 4	5	6	7	8	9
T=4	🟩 1	🟩 2	🟧 3	🟧 4	5	6	7	8	9
T=5	🟩 1	🟩 2	🟩 3	🟧 4	5	6	7	8	9
T=6	🟩 1	🟩 2	🟩 3	🟩 4	5	6	7	8	9

🟩 =acked

🟧 =sent

🟨 =advertised

Packet Format

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

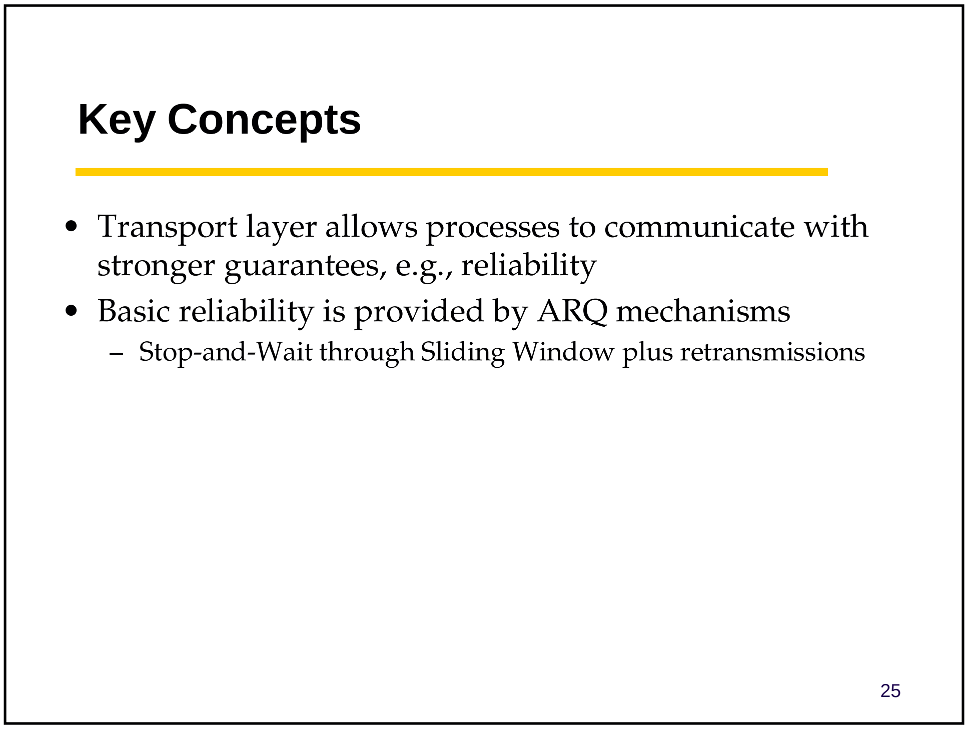
16 bit window size gets
Cramped with large
Bandwidth x delay

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