CSCI-1680
Transport Layer II

Data over TCP: Flow Control

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, David Mazières, Phil Levis, John Jannotti

Administrivia

- TCP Gearup I TONIGHT (10/26) 5-7pm, CIT368
  - How the project works, how to think about sockets
  - Stuff you need for milestone 1
- TCP milestone 1: Schedule on/before Thursday, November 2
  - Email later today for signups
- HW2: Due Mon, Oct 30
  - Last problem helpful for milestone 1

The story so far

Stop and Wait: Simplest TCP sender/receiver

The story so far

Stop and Wait: Simplest TCP sender/receiver

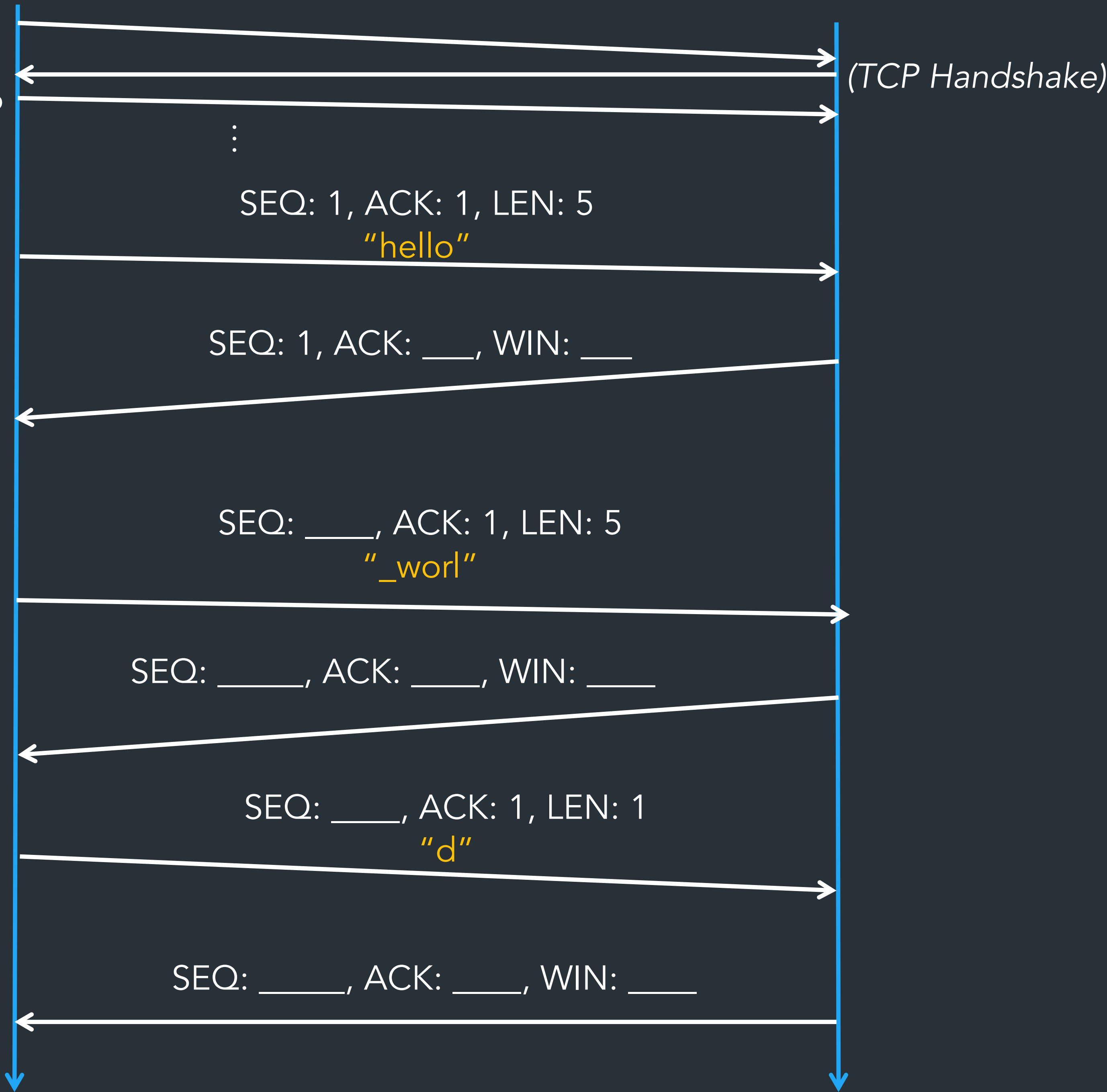
Key features

- SEQ/ACK numbers denote where sender/receiver are in data stream
- Only one segment is "in flight" at a time

Warmup: Stop and Wait

What are the values for the SEQ and ACK fields?

`conn.Write(“hello_world”)`



(TCP Handshake)

⋮

SEQ: 1, ACK: 1, LEN: 5
“hello”

SEQ: 1, ACK: ___, WIN: ___

SEQ: ____, ACK: 1, LEN: 5
“_worl”

SEQ: _____, ACK: ____, WIN: ____

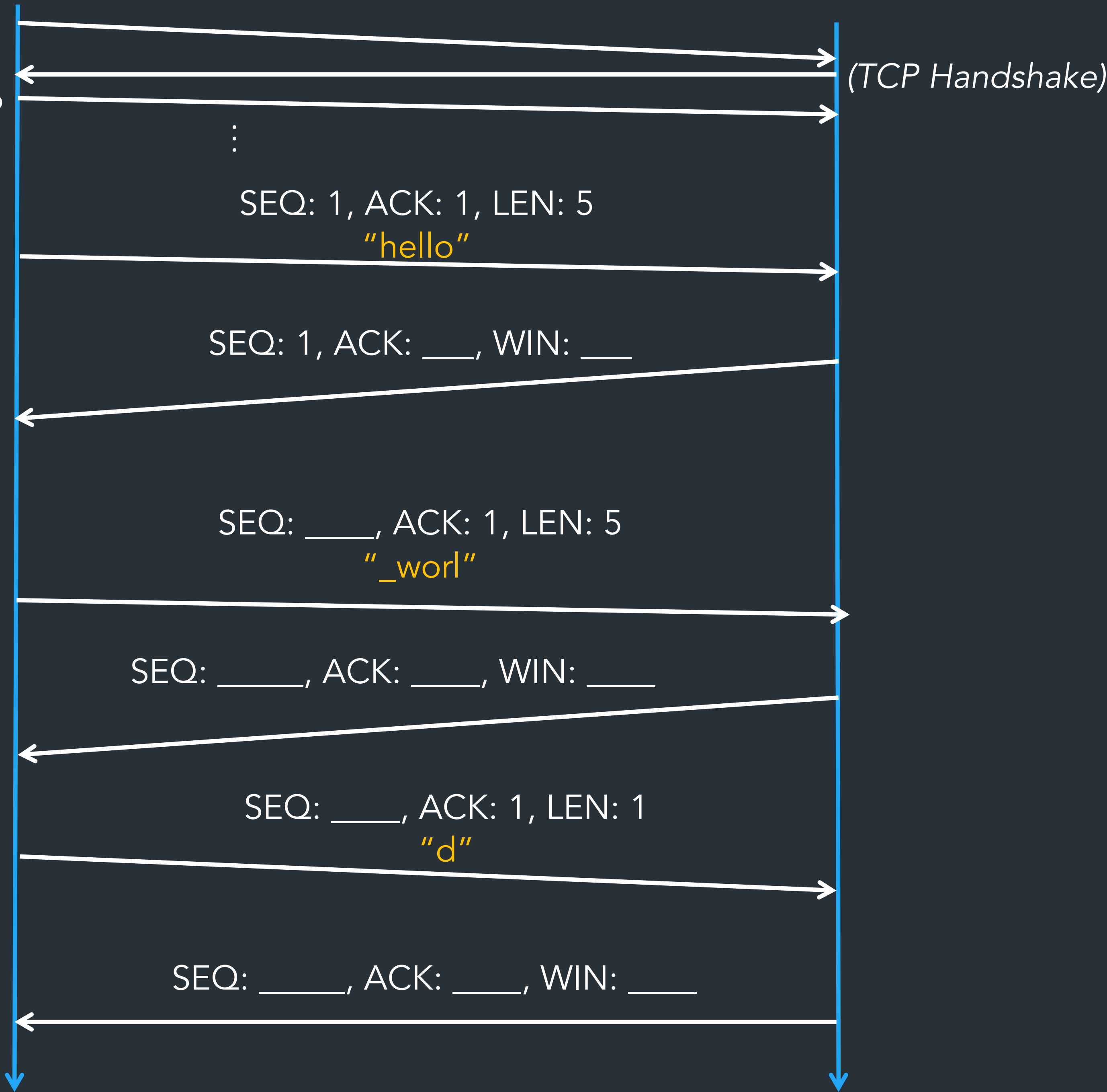
SEQ: ____, ACK: 1, LEN: 1
“d”

SEQ: _____, ACK: ____, WIN: ____

Warmup: Stop and Wait

What are the values for the SEQ and ACK fields?

`conn.Write(“hello_world”)`


(TCP Handshake)
⋮
SEQ: 1, ACK: 1, LEN: 5
“hello”
SEQ: 1, ACK: ___, WIN: ___
SEQ: ____, ACK: 1, LEN: 5
“_worl”
SEQ: _____, ACK: ____, WIN: ____
SEQ: ____, ACK: 1, LEN: 1
“d”
SEQ: _____, ACK: ____, WIN: ____

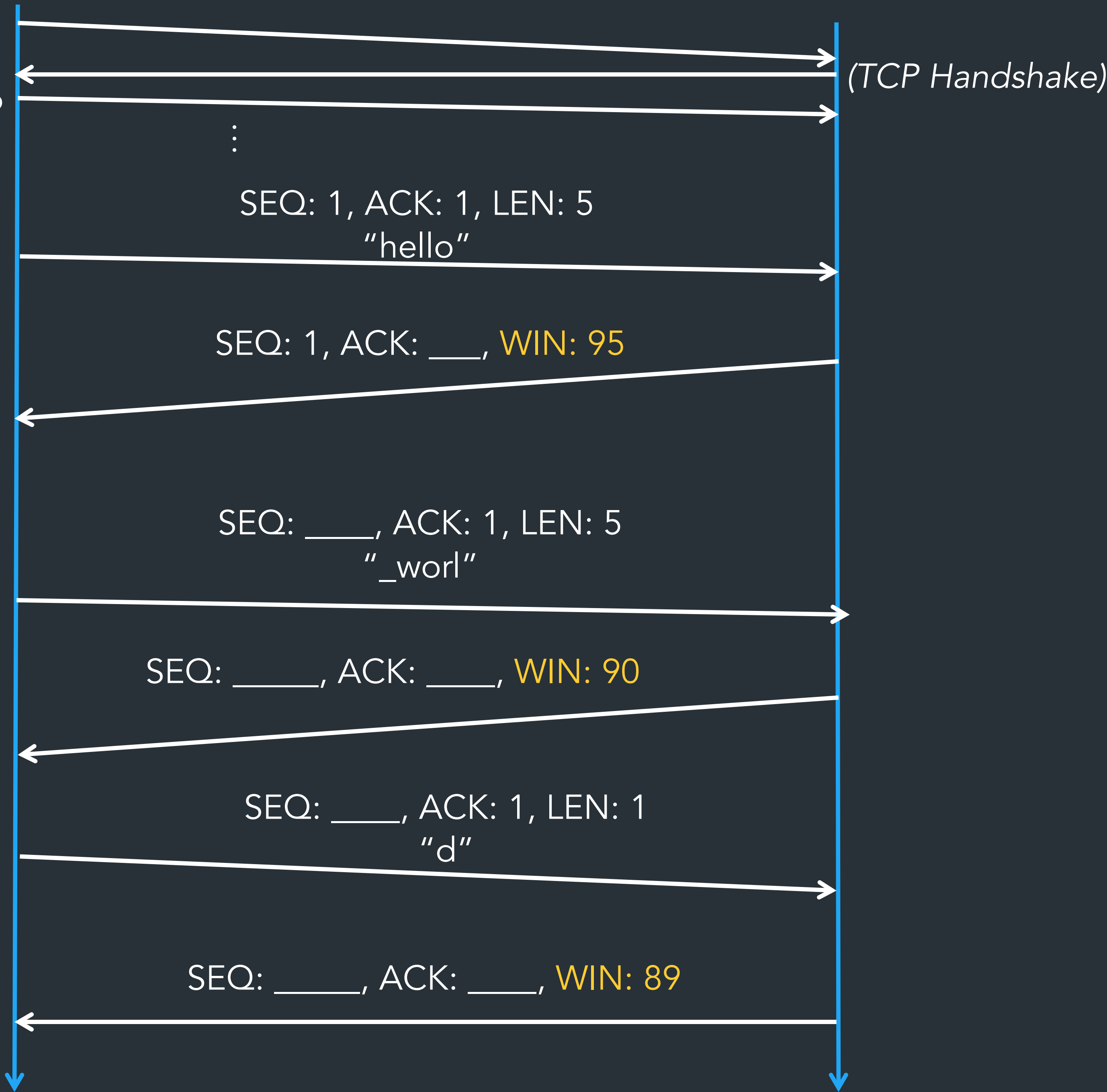
Key features

- SEQ: Position of this segment in the data stream
- ACK: Next sequence number the receiver expects to receive (ACK N == “I have up to (N – 1)”)

Warmup: Stop and Wait

What are the values for the SEQ and ACK fields?

`conn.Write("hello_world")`


(TCP Handshake)
⋮
SEQ: 1, ACK: 1, LEN: 5
"hello"
SEQ: 1, ACK: ___, WIN: 95
SEQ: ____, ACK: 1, LEN: 5
"_worl"
SEQ: _____, ACK: ____, WIN: 90
SEQ: ____, ACK: 1, LEN: 1
"d"
SEQ: _____, ACK: ____, WIN: 89

Key features

- SEQ: Position of this segment in the data stream
- ACK: Next sequence number the receiver expects to receive (ACK N == "I have up to (N – 1)")

Advertised window: how much space the receiver has left in its receive buffer
=> Window (WIN) field in TCP header

Topics for today

- Flow control: Sliding window
- Computing RTO
- Connection termination

TCP and buffering

Recall: TCP stack responsibilities

- Sender: breaking application data into segments
- Receiver: receiving segments, reassembling them in order

TCP and buffering

Recall: TCP stack responsibilities

- Sender: breaking application data into segments
- Receiver: receiving segments, reassembling them in order

TCP stack needs to buffer data for both parts

- <u>Sender</u>: data waiting to be sent, not yet ACK’d
- <u>Receiver</u>: data not yet read by app, out-of-order segments

TCP and buffering

Recall: TCP stack responsibilities

- Sender: breaking application data into segments
- Receiver: receiving segments, reassembling them in order

TCP stack needs to buffer data for both parts

- Sender: data waiting to be sent, not yet ACK'd
- Receiver: data not yet read by app, out-of-order segments

Remember: in reality, both sides can send and receive!
=> All sockets have both a send and receive buffer

Sliding window: in abstract terms

- Window of size w
- Can send at most w packets before waiting for an ACK

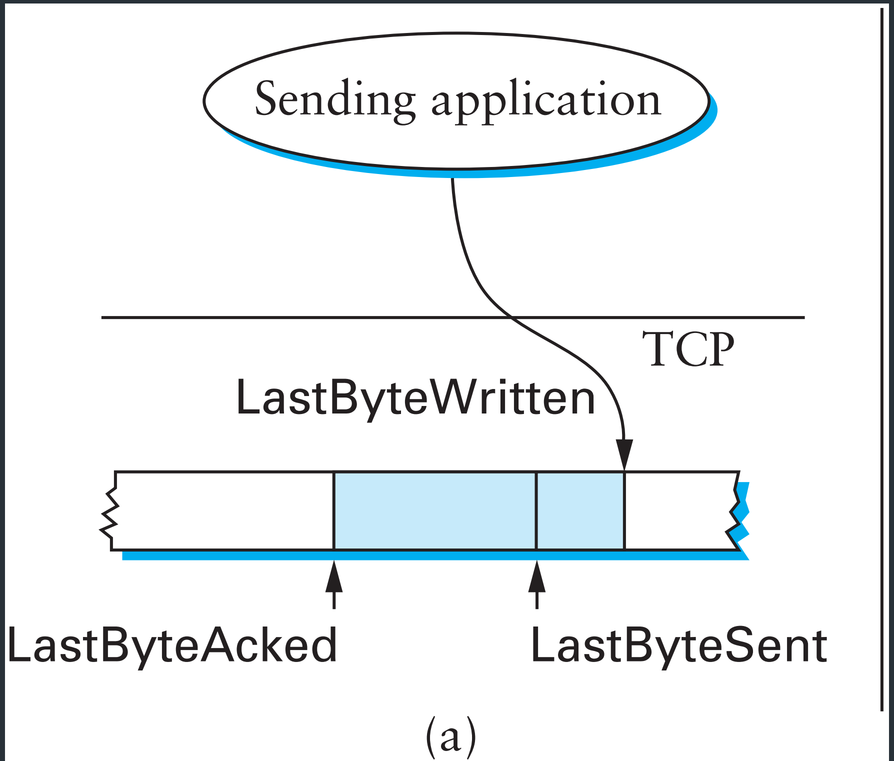
<u>Goals</u>

- Network “pipe” always filled with data
- ACKs come back at rate data is delivered => “self-clocking”

Sender example

Receiver example

Flow Control: Sender



Invariants

- LastByteSent – LastByteAcked <= AdvertisedWindow
- EffectiveWindow = AdvertisedWindow – (BytesInFlight)
- LastByteWritten – LastByteAcked <= MaxSendBuffer

Useful Sliding Window Terminology:
RFC 9293, Sec 3.3.1

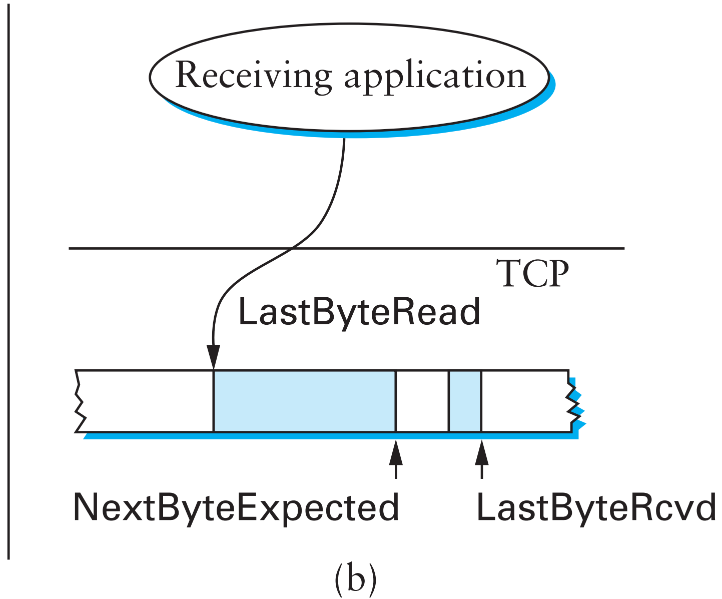
Flow control: receiver

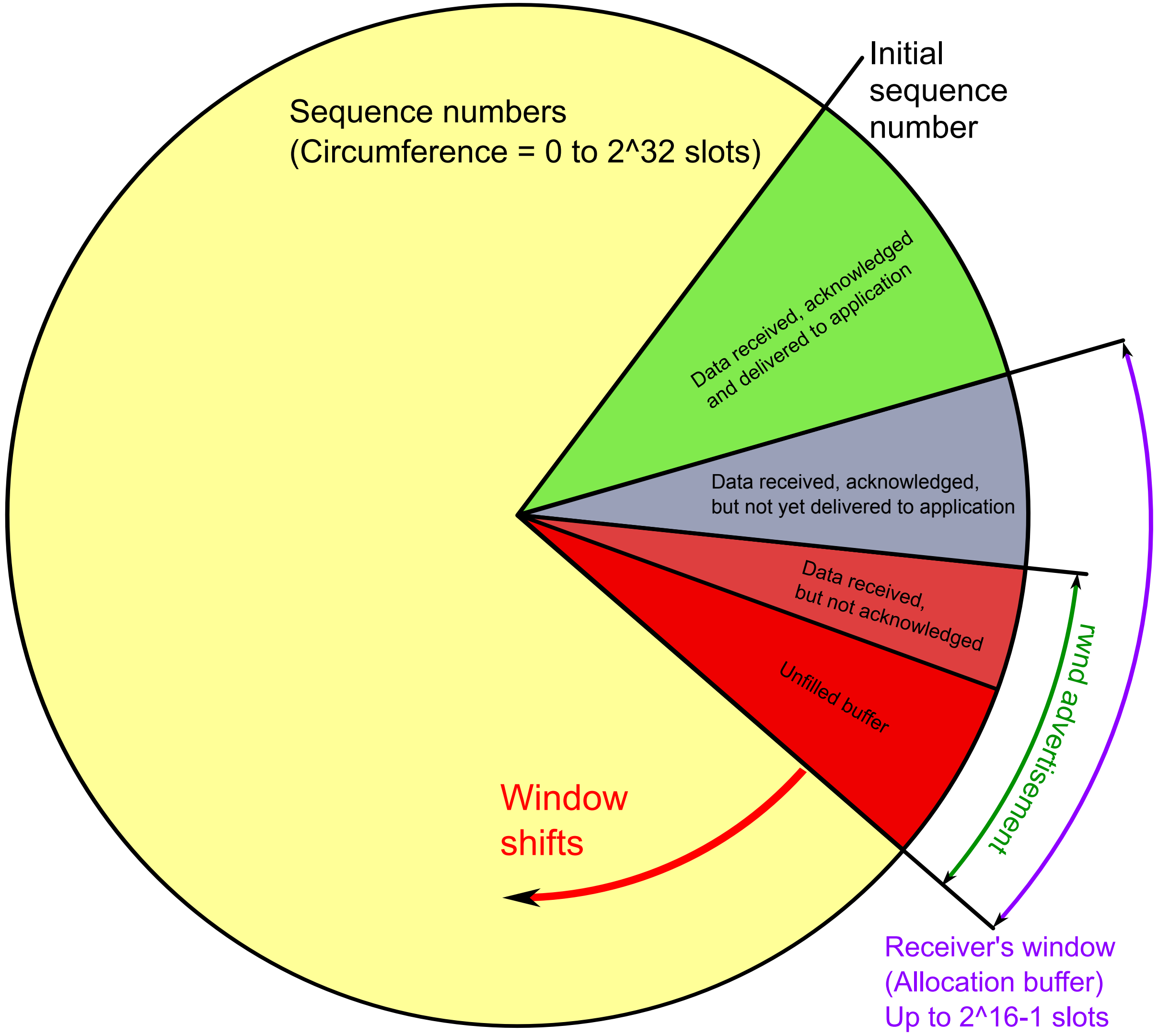
Useful Sliding Window Terminology: RFC 9293, Sec 3.3.1

- Can accept data if space in window
- Available window =
  BufferSize– ((NextByteExpected-1) - LastByteRead

- On receiving segment for byte S
  - if s is outside window, ignore packet
  - if s == NextByteExpected:
    - Deliver to application (Update LastByteReceived)
    - If next segment was early arrival, deliver it too
  - If s > NextByteExpected, but within window
    - Queue as early arrival

- Send ACK for highest contiguous byte received, available window





Sequence numbers
(Circumference = 0 to 2^32 slots)

Initial
sequence
number

Data received, acknowledged
and delivered to application
Data received, acknowledged,
but not yet delivered to application
Data received,
but not acknowledged
Unfilled buffer
rwnd advertisement
Window
shifts

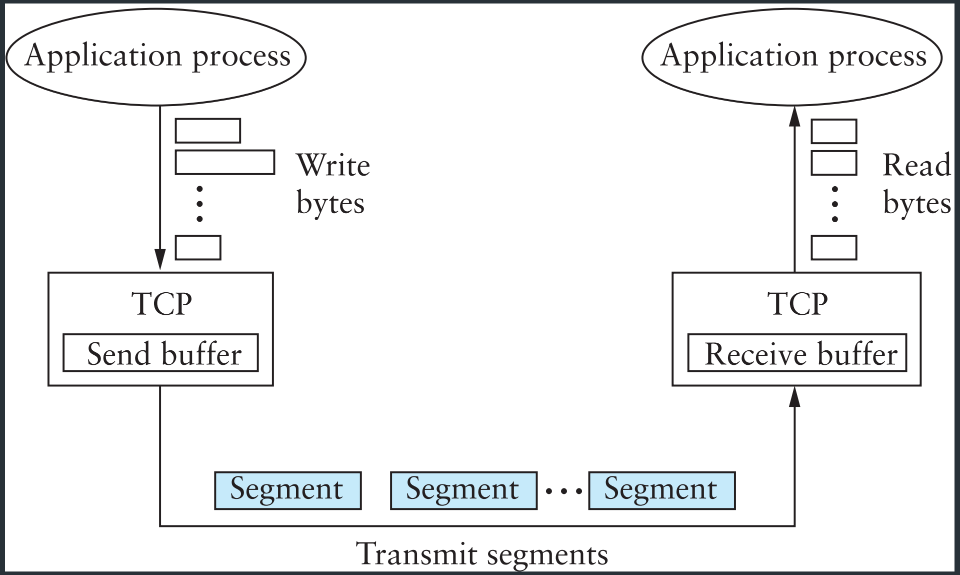
Receiver's window
(Allocation buffer)
Up to 2^16-1 slots

Some Visualizations

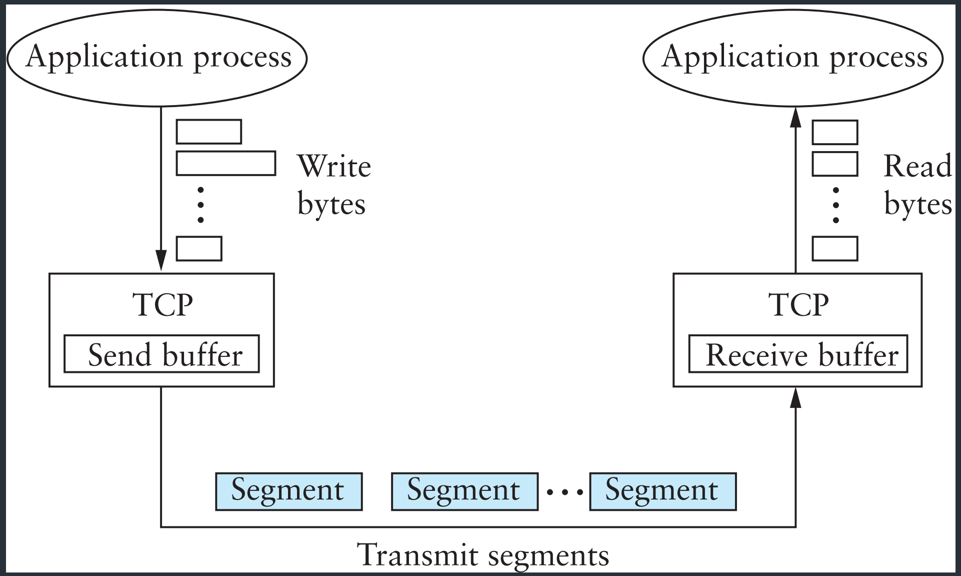
- Normal conditions: https://www.youtube.com/watch?v=zY3Sxvj8kZA

- With packet loss: https://www.youtube.com/watch?v=lk27yiITOvU

What happens if the receiving app never reads from its buffer?



What happens if the receiving app never reads from its buffer?



⇒ Receive buffer fills up => Advertised window drops to 0
⇒ Send buffer fills up
⇒ Eventually, sending app can’t send anymore

What happens if the receiving app never reads from its buffer?

Problem: need a way for sender to know when space is available again!

What happens if the receiving app never reads from its buffer?

Problem: need a way for sender to know when space is available again!

Resolution: zero window probing

- Sender periodically sends 1-byte segments
- Receiver sends back ACK with advertised window (even if it has no room for segment

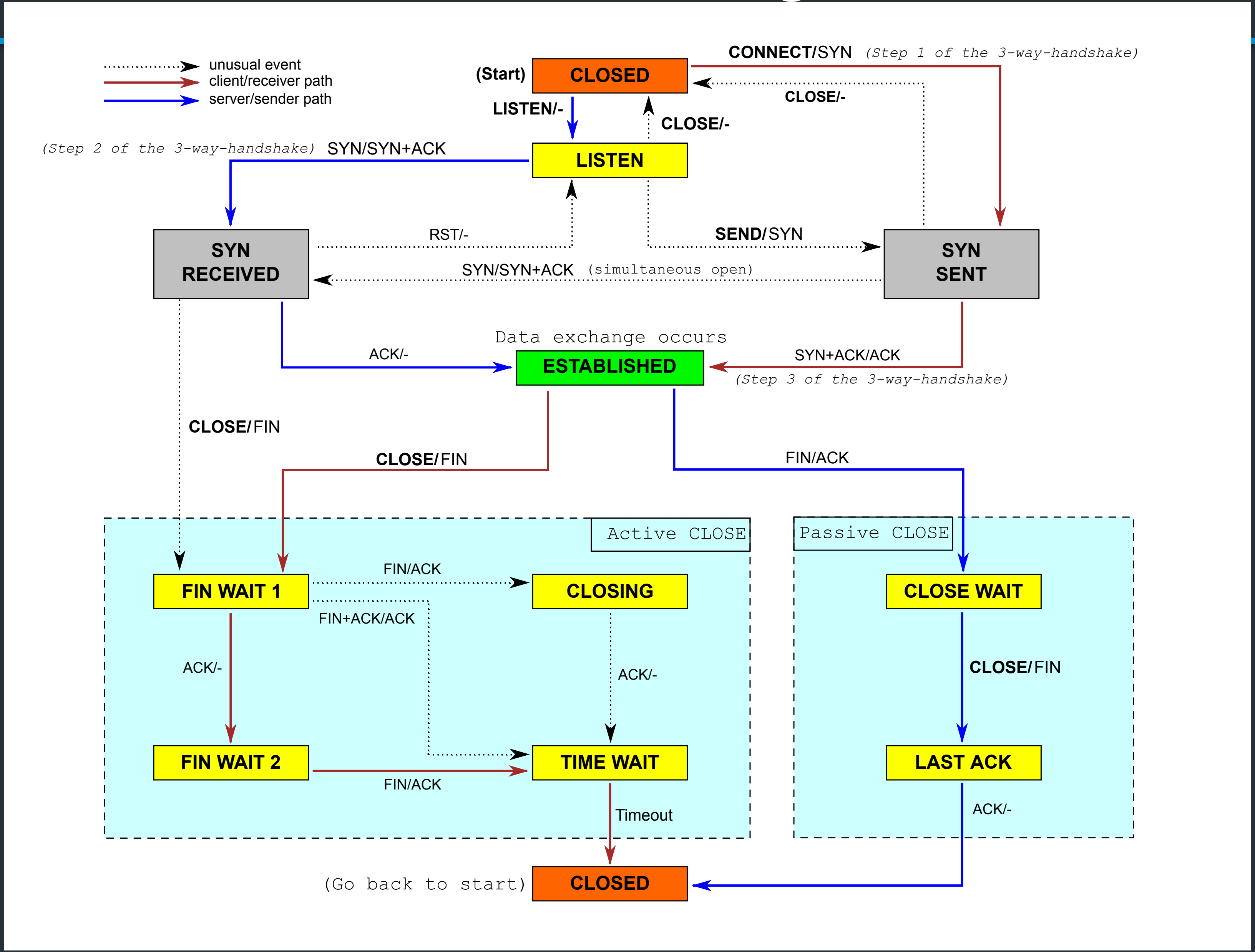
What happens if the receiving app never reads from its buffer?

Problem: need a way for sender to know when space is available again!

Resolution: <u>zero window probing</u>

- Sender periodically sends 1-byte segments
- Receiver sends back ACK with advertised window (even if it has no room for segment
- Sender can resume sending when win != 0 (preferably when win >= MSS)

TCP State Diagram



How do ACKs work?

- ACK contains *next expected sequence number*
- Sender: if one segment is missed but new ones received, send duplicate ACK
- Receiver retransmits when:
  - Receive timeout (RTO) expires
  - Possibly other conditions, for certain TCP variants (eg. 3 dup ACKs)

- How to set RTO?

What’s a good timeout value?

- 0.5s? 1s? 0.01s?

What’s a good timeout value?

- 0.5s? 1s? 0.01s?

=> If timeout too short, packet might still be in flight (network latency, etc.)

=> If timeout too long, affects throughput

What’s a good timeout value?

- 0.5s? 1s? 0.01s?

=> If timeout too short, packet might still be in flight (network latency, etc.)

=> If timeout too long, affects throughput

=> How long *should* it take a packet to arrive at other side?

What’s a good timeout value?

- 0.5s? 1s? 0.01s?

=> If timeout too short, packet might still be in flight (network latency, etc.)

=> If timeout too long, affects throughput

⇒How long ***should*** it take a packet to arrive at other side?

1RTT!

=>Can measure RTT, use to set RTO

Computing RTO

Strategy: *measure* expected RTT based on ACKs received

- Use exponentially weighted moving average (EWMA)

Computing RTO

Strategy: *measure* expected RTT based on ACKs received

Use exponentially weighted moving average (EWMA)

- RFC793 version ("smoothed RTT"):

$$SRTT = (\alpha * SRTT_{Last}) + (1 - \alpha)* RTT_{Measured}$$

$$RTO = max(RTO_{Min}, min(\boldsymbol{\beta} * SRTT, RTO_{Max}))$$

α = "Smoothing factor": .8-.9

$\boldsymbol{\beta}$ = "Delay variance factor": 1.3—2.0

RTO_{Min} = 1 second

RFC793, Sec 3.7
RFC6298 (slightly more complicated, also measures variance)

Using the RTO timer

Recommended by RFC6298

- Maintain ONE timer per connection
- When segment is sent => set timer to expire after t_{RTO}
- When ACK is received with new data, reset the timer

Using the RTO timer

Recommended by RFC6298

- Maintain ONE timer per connection
- When segment is sent => set timer to expire after t_{RTO}
- When ACK is received with new data, reset the timer

When the timer expires:

- Retransmit earliest unacknowledged segment
- RTO = 2 * RTO (up to some max)
- If no data after N retransmissions => give up, terminate connection

This is only the beginning…

- Problem 1: what if ACK is for a retransmitted segment?
  - Solution: don’t update RTT if segment was retransmitted

- Problem 2: RTT can have high variance
  - Initial implementation doesn’t account for this (modern version, RFC6298)
  - Congestion control: modeling network load