CSCI-1680
Transport Layer II

Data over TCP: Flow Control

Nick DeMarinis

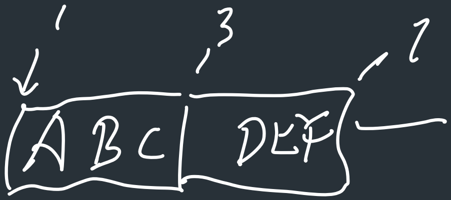
Based partly on lecture notes by Rodrigo Fonseca, David Mazières, Phil Levis, John Jannotti

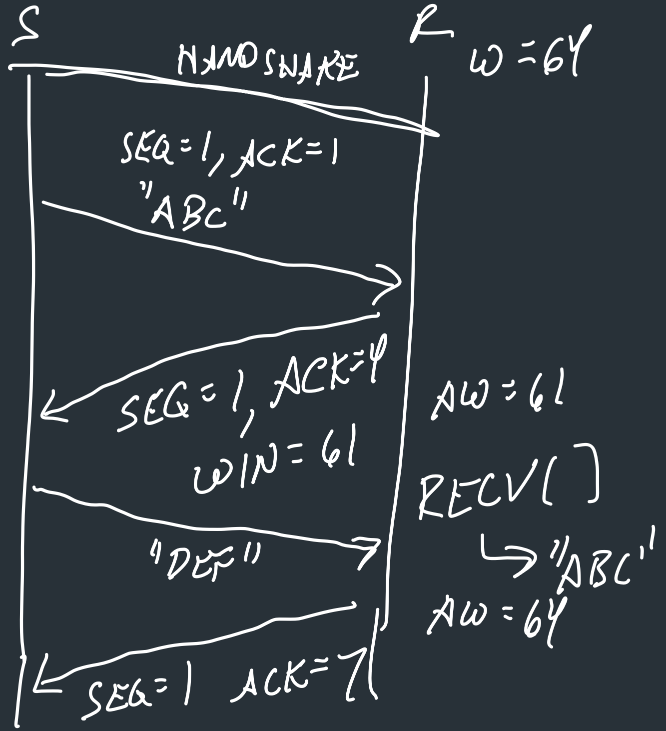
Administrivia

- Make sure you have signed up for IP grading
- TCP milestone I: Thurs, Apr 14
- TCP gear-up early next week, look for details

Topics for today

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

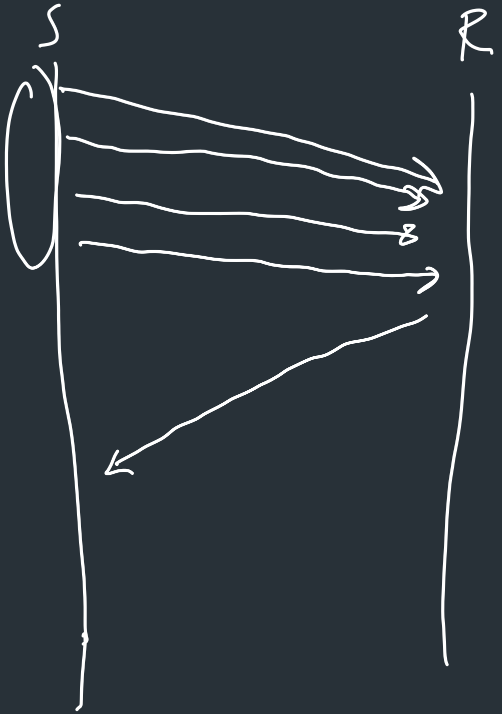




Sliding window: in abstract terms

6ℓ

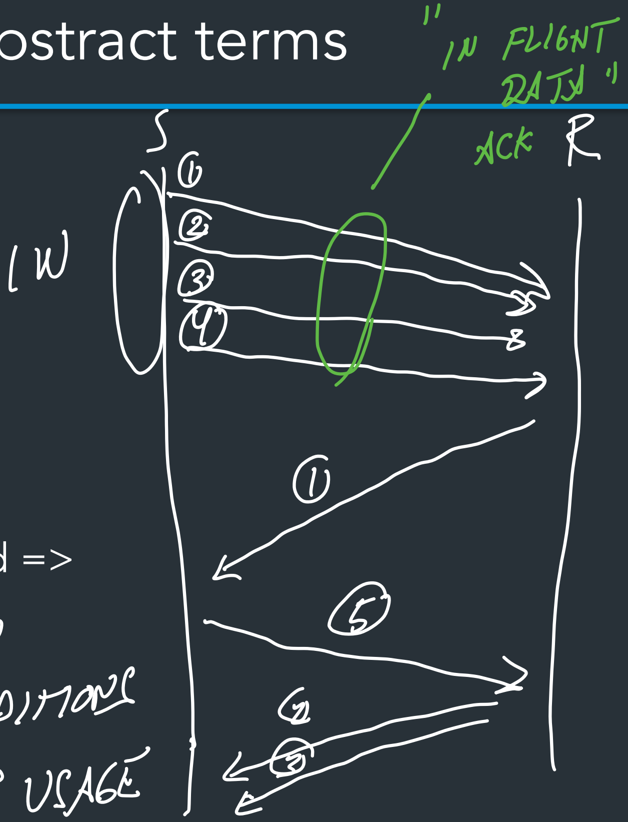
- Window of size w
- Can send at most w packets before waiting for an ACK



Sliding window: in abstract terms

- Window of size w
- Can send at most w packets before waiting for an ACK
- Goal
  - Network "pipe" always filled with data
  - ACKs come back at rate data is delivered => "self-clocking"

↳ DEPENDS ON
- NETWORK CONDITIONS
- RECEIVER/APP USAGE



SENDER

SEND ADDS DATA TO BUFFER

LAST BYTE SENT → NEXT SEQ NUM

LAST BYTE ACK'D BY RECEIVER

↳ OLDEST SEQ NUM IN FLIGHT

"BYTES IN FLIGHT" < WINDOW ADVERTIZED BY RECEIVER

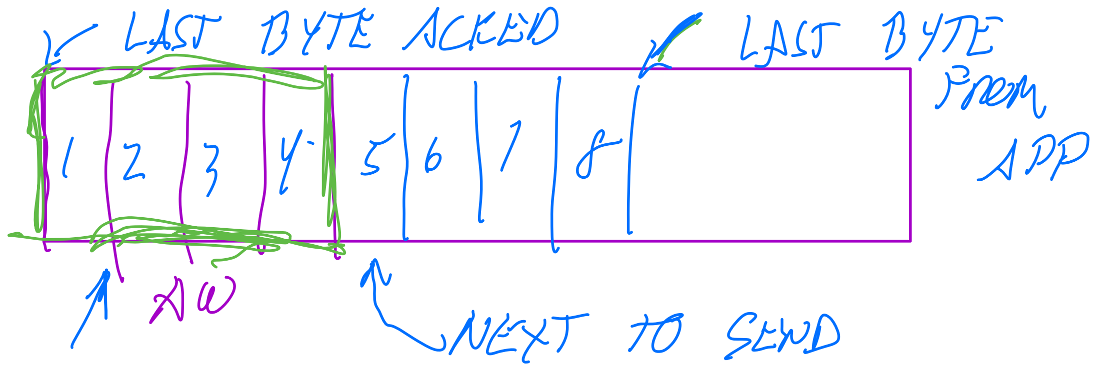
ON ACK FOR A

IF $LBA \not< LBS$ = OUT OF WINDOW, DROP, SINCE (OLD) INVALID

OTHERWISE

LBA += (SIZE OF SEGMENT)

IF ACK FULLY COVERS AN UNACKED SEGMENT, CAN DEQUEUE.



- RECEIVER ADVERTISES HOW MUCH DATA IT CAN ACCEPT (ADVERTIZED WINDOW, AW)
- SEND SEGMENTS TO FILL UP TO AW)
  - FOR EACH ONE, RECORD TIMESTAMP FOR RETRANSMIT

IN FLIGHT = 1,2,3,4 LBA = 0
NBS = 5

EACH SEGMENT SHOULD BE IDEALLY MSS BYTES ≈ (≈1000 BYTES)

IF YOU GET ACK FOR SEQ A

- RECEIVER HAS DATA UP TO SEQ A ("CUMULATIVE ACKS)
- LBA = A
- CAN NOW SEND MORE DATA (UP TO AW)

IN 1 BYTE SEGMENTS

LBA = 10   W = 4   [$\boxed{11,}$ 12, 13, 14]

IF YOU GET ACK 12

W = [12, 13, 14, 15]

IF 10 BYTE SEGMENTS

LBA = 10   W = 40 BYTES

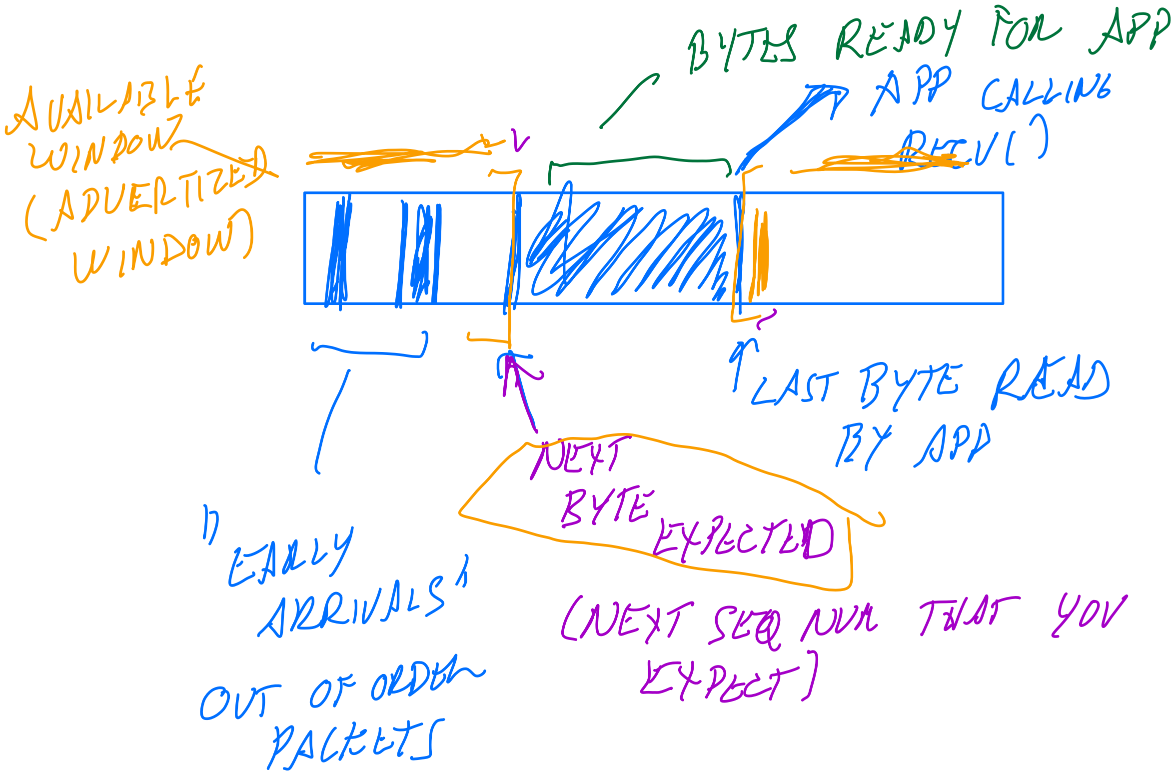
IF ACK FOR 21

- R HAS DATA UP TO 20

SEGMENTS IN FLIGHT: [~~10~~, 20, 30, 40]
50

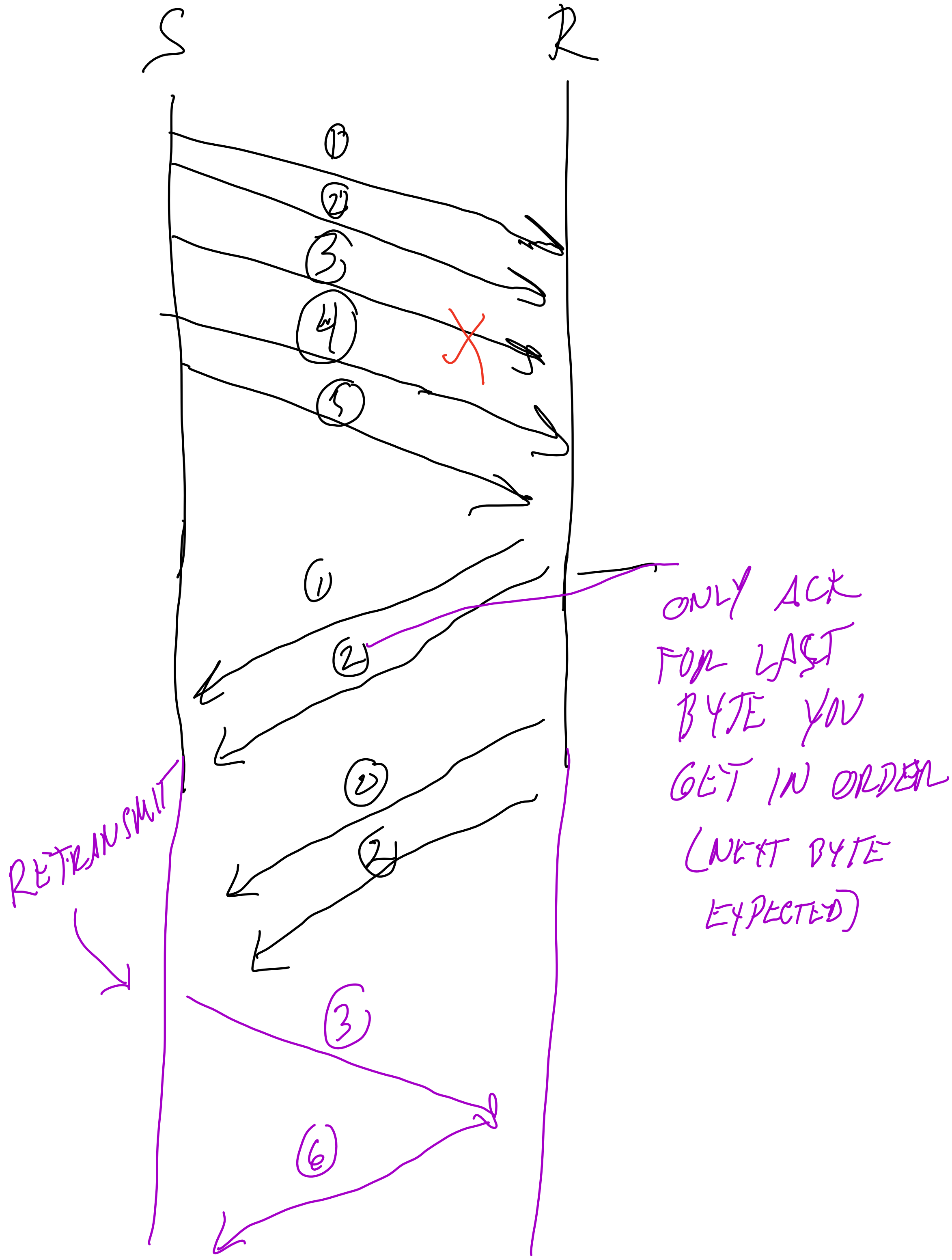
IF ACK FOR 15:
MIGHT NEED TO RETRANSMIT
PART OF SEGMENT

FOR EACH SEGMENT IN FLIGHT:
- TIMESTAMP OF LAST SENT TIME
- RETRANSMIT IF THIS EXPIRES



AVAILABLE/ADVERTIZED WINDOW

- WHAT YOU SEND BACK TO SENDER TO INDICATE HOW MUCH IT CAN SEND
- CAN BE 0



S
R
1
2
3
4
5
1
2
2
2
ONLY ACK FOR LAST BYTE YOU GET IN ORDER (NEXT BYTE EXPECTED)
RETRANSMIT
3
6

NEXT BYTE EXPECTED =
NEXT SEQ NUM YOU EXPECT
TO GET

VALID
RANGE OF SEQUENCE NUMBERS

(NEXT, NEXT SEQ + AVAIL WINDOW)

IF YOU GET SEG W/ SEQ S

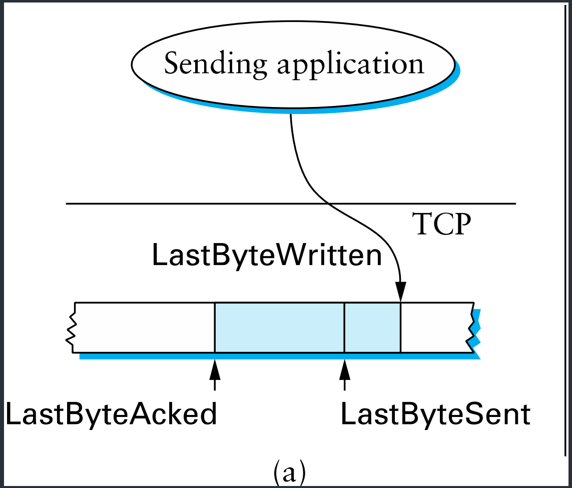
IF S ≥ NBE AND S < NBE + AVAIL WINDOW

- ADD TO BUFFER AT POSITION S
- NBE += SEGMENT SIZE
- CHECK EARLY ARRIVAL QUEUE, MOVE UP TO NEXT CONTIGUOUS BLOCK.

Sender example

Receiver example

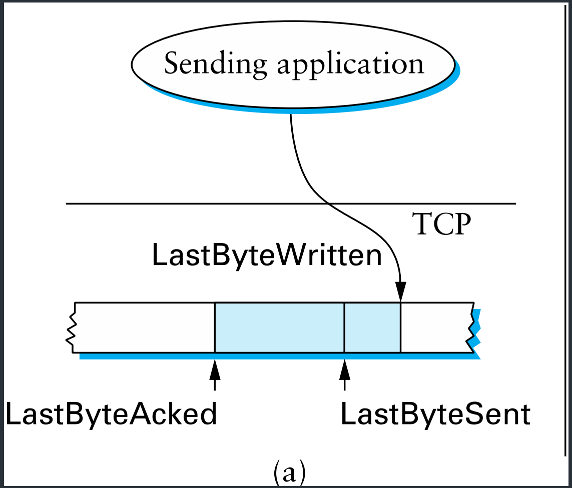
Flow Control: Sender



Invariants

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

Flow Control: Sender



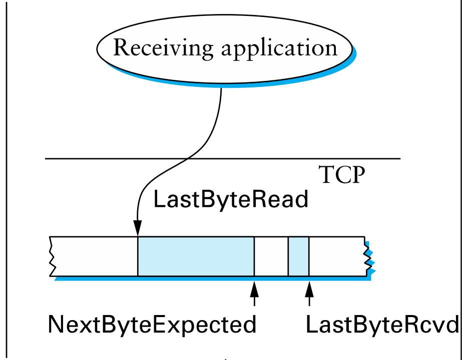
Invariants

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

Useful Sliding Window Terminology:
RFC 793, Sec 3.3

Flow control: receiver

- 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



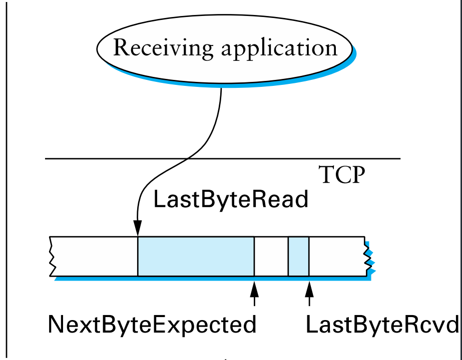
(b)

- Send ACK for highest contiguous byte received, available window

Flow control: receiver

Useful Sliding Window Terminology:
RFC 793, Sec 3.3

- 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

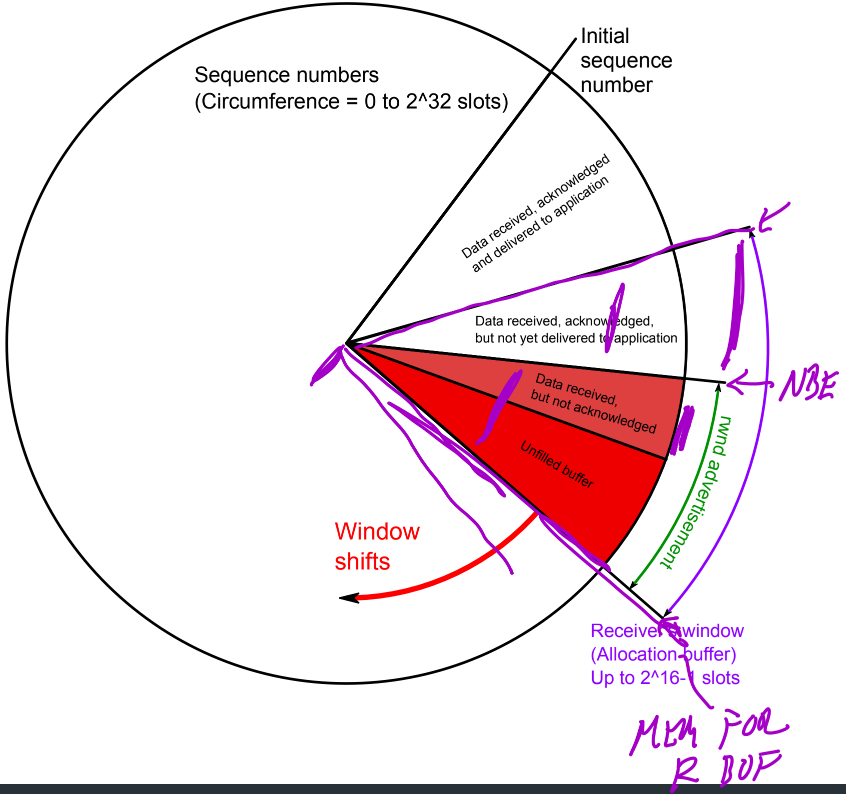


(b)

Flow Control

- Advertised window can fall to 0
  - How?
  - Sender eventually stops sending, blocks application

- Resolution: zero window probing: sender sends 1-byte segments until window comes back > 0


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 window
(Allocation buffer)
Up to 2^16-1 slots
NBE
MEM FOR
R BUF

Some Visualizations

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

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

How do ACKs work?

- ACK contains next expected sequence number
- If one segment is missed but new ones received, send duplicate ACK
- Retransmit when:
  - Receive timeout (RTO) expires
  - Receive 3 Duplicate ACKs
- How to set RTO?

When to time out?

RFC793, Sec 3.7

Should expect an ACK within one Round Trip Time (RTT)

- Problem: RTT can be highly variable

- Strategy: expected RTT based on ACKs received
  - Use exponentially weighted moving average (EWMA)
  - RFC793 version ("smoothed RTT"):

When to time out?

RFC793, Sec 3.7

Should expect an ACK within one Round Trip Time (RTT)

- Problem: RTT can be highly variable

- Strategy: expected RTT based on ACKs received
  - Use exponentially weighted moving average (EWMA)
  - RFC793 version ("smoothed RTT"):

NEW MEASURED VALUE

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

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

MAX MIN

α = "Smoothing factor": .8-.9

β = "Delay variance factor": 1.3—2.0

HOW MUCH NEW SAMPLE AFFECTS AVG

This is only the beginning…

- Problem 1: what if segment is a retransmission?

This is only the beginning…

- Problem 1: what if segment is a retransmission?
  - Solution: don't update RTT if segment was retransmitted

This is only the beginning…

- Problem 1: what if segment is a retransmission?
  - Solution: don’t update RTT if segment was retransmitted
- Problem 2: RTT can have high variance
  - Initial implementation doesn’t account for this

This is only the beginning…

- Problem 1: what if segment is a retransmission?
  - Solution: don't update RTT if segment was retransmitted
- Problem 2: RTT can have high variance
  - Initial implementation doesn't account for this
  - Congestion control: modeling network load

When to Transmit?

Nagle's algorithm

- Goal: reduce the overhead of small packets

```
if (there is data to send) and (window >= MSS)
    Send a MSS segment
else
    if there is unAcked data in flight
            buffer the new data until ACK arrives
    else
            send all the new data now
```

- Receiver should avoid advertising a window <= MSS after advertising a window of 0

Delayed Acknowledgments

- Goal: Piggy-back ACKs on data
  - Delay ACK for 200ms in case application sends data
  - If more data received, immediately ACK second segment
  - Note: never delay duplicate ACKs (if missing a segment)

Delayed Acknowledgments

- Goal: Piggy-back ACKs on data
  - Delay ACK for 200ms in case application sends data
  - If more data received, immediately ACK second segment
  - Note: never delay duplicate ACKs (if missing a segment)

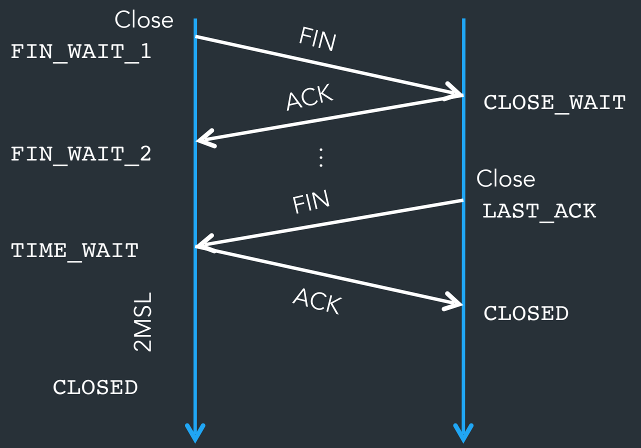
- Warning: can interact badly with Nagle for some applications
  - Nagle waits for ACK until send => Temporary deadlock
  - App can disable Nagle with `TCP_NODELAY`
  - App should also avoid many small writes

Summary: flow control

- Flow control provides correctness: reliable, in order delivery
- Need more for performance
  - What if the network is the bottleneck?
- Sending too fast will cause queue overflows, heavy packet loss
- Need more for performance: congestion control

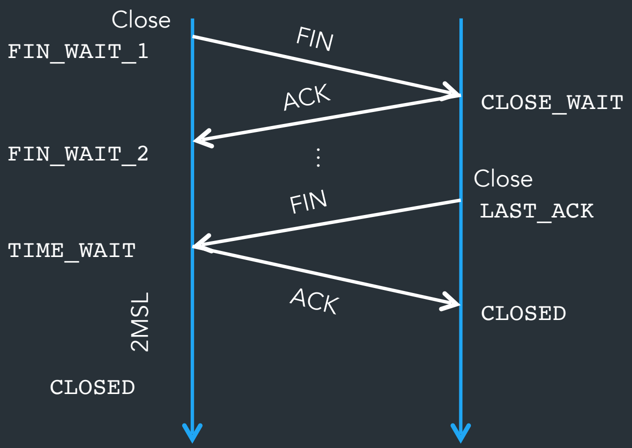
Connection Termination

- When you have no more data to send, send a FIN
  - Sent by `close()` or `shutdown()`
- Both sides close connection separately!

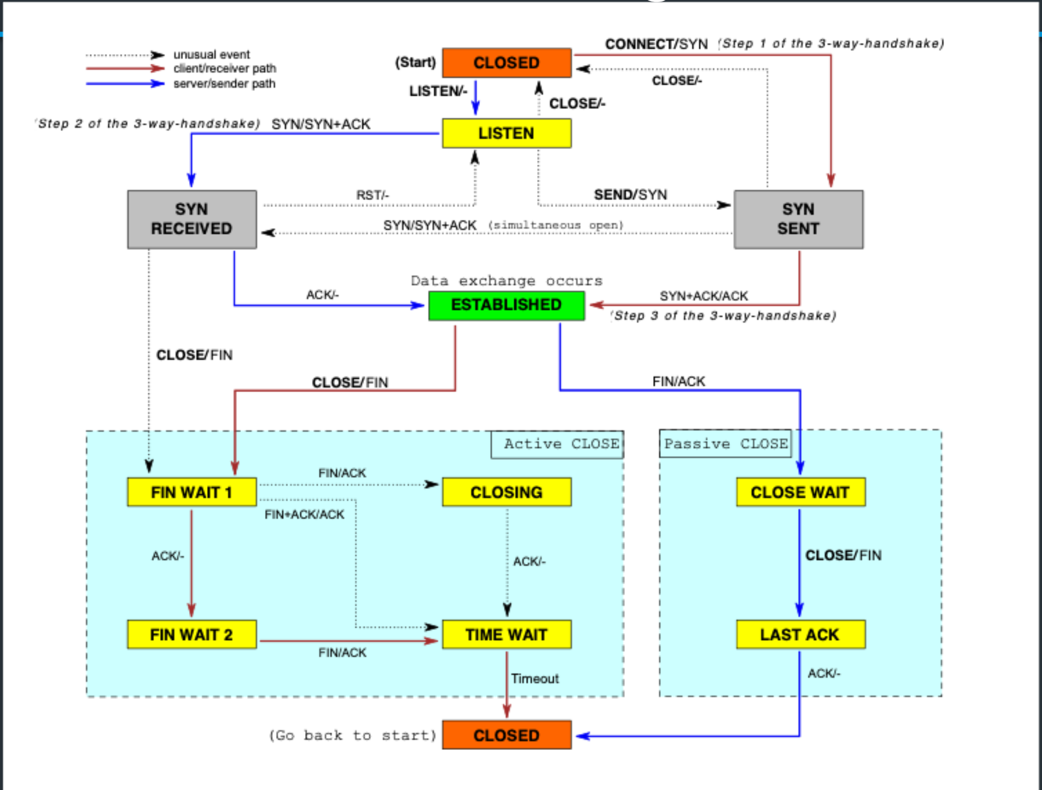


Connection Termination

- When you have no more data to send, send a FIN
  - Sent by `close()` or `shutdown()`
- Both sides close connection separately!
- `TIME_WAIT`: initiating side should wait for 2*MSL before deleting TCB
  - MSL = Longest time a segment might be delayed (configurable, ~1min)



TCP State Diagram



AIMD Implementation

- In practice, send MSS-sized segments
  - Let window size in bytes be w (a multiple of MSS)
- Increase:
  - After w bytes ACKed, could set w = w + MSS
  - Smoother to increment on each ACK
    - w = w + MSS/(# acks/w) = w + MSS/(w/MSS)
      = w + MSS^2/w
- Decrease:
  - After a packet loss, w = w/2
  - But don't want w < MSS
  - So react differently to multiple consecutive losses
  - Back off exponentially (pause with no packets in flight)