Don’t panic: TCP gearup III

Overview

- Final TCP stuff
- Any questions you have

Roadmap

Milestone I

- Start of your API and TCP stack
- Listen and establish connections => create sockets/TCB
- TCP handshake
- accept, connect, and start of ls REPL commands

Roadmap

Milestone II

- Basic sending and receiving using your sliding window/send receive buffers
- Plan for the remaining features

Roadmap

Final deadline

- Retransmissions (+ computing RTO from RTT)
- Out-of-order packets
- Sending and receiving files (sf, rf) ←
- Zero-window probing
- Connection teardown (CL)

Sendfile/Recvfile

I recommend starting here!!
=> Can to help test other features!
=> Can help find concurrency bugs early

Using your socket API, send/recv a file

Sendfile

- Open a file, VConnect, call VWrite in a loop

Recvfile

- Listen on a port, Open a file, call VRead in a loop

=> This is the ultimate test: your implementation should be similar to how you'd use a real socket API!

Demo!

A common thing to notice when you start sf/rf, sometimes you start seeing bugs from IP

=> Run reference with YOUR router, OUR HOST

=> Could help you root out a problem at the interface level

Relevant materials

- Lecture 16 (3/19): Sliding window, retransmissions, zero window probing
- Lecture 17 (3/31): connection teardown

- "Getting started" and "Testing guide" in TCP docs
  => Can configure reference to drop packets
  => How to test each part (following guidance here)
  => How to debug stuff

So how do we get there?

Retransmissions

More info: Lecture 16, RFC6298

Usually, make a "retransmission queue"

- When segment sent, add segment to queue with some metadata

  => What to store? You decide!

=> Start RTO timer => one timer per socket

= > When you get an ACK, reset the timer

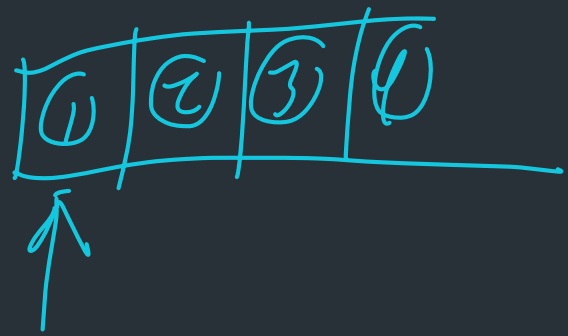
Retransmissions

More info: Lecture 16, RFC6298

Usually, make a "retransmission queue"

- When segment sent, add segment to queue with some metadata

  => What to store? You decide!



=> Start RTO timer => one timer per socket
= > When you get an ACK, reset the timer

When timer expires
=> Retransmit earliest segment
=> RTO = 2*RTO up to some max
=> Fail after some N retransmissions

RFC6298 is your friend!
Use it! (edge cases, etc.)

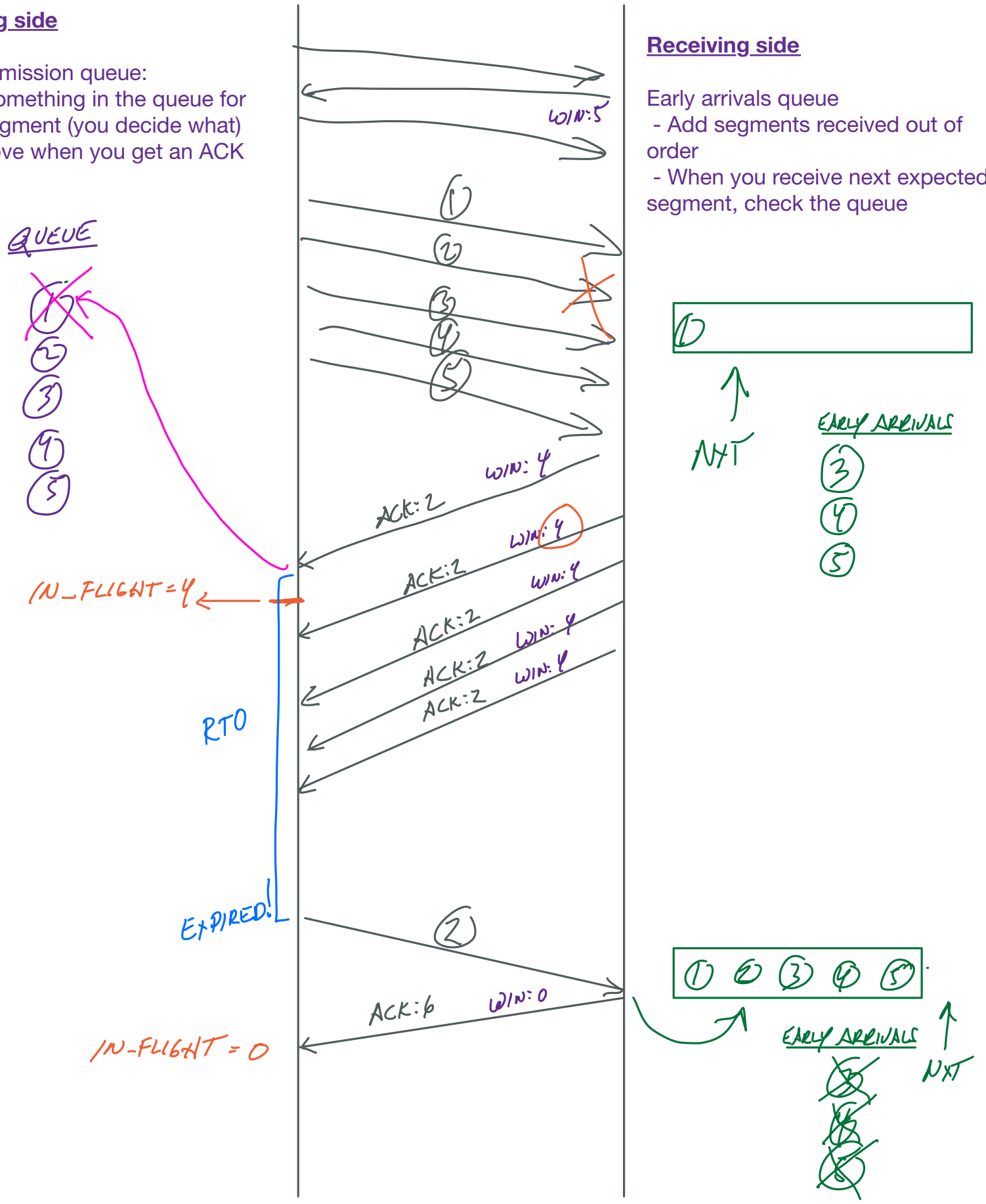
Scratch notes for retransmissions/early arrivals: see recording for a live drawing, lecture 15 for more

Sending side

Retransmission queue:
- Put something in the queue for each segment (you decide what)
- Remove when you get an ACK

Receiving side

Early arrivals queue
- Add segments received out of order
- When you receive next expected segment, check the queue



RTO?

More info: Lecture 16, RFC6298

RTO = Retransmission Timeout (RTO)

=> Based on expected RTT: "how long until you SHOULD get an ACK?"

When you get an ACK (FOR NEW DATA DATA), update RTO

RTT = one measurement
Use to compute SRTT
=> Smoothed RTT => RTO
=> Compute by weighted moving average

RTT_new = RTT_old*(alpha) + RTT_latest*(1-alpha)

Example upper/lower bounds
RTOmin ~= 100ms
RTOmax ~= 5sec

RTO?

More info: Lecture 16, RFC6298

RTO = Retransmission Timeout (RTO)

=> Based on expected RTT: "how long until you SHOULD get an ACK?"

When you get an ACK, update RTO

=> Smoothed weighted moving average of recent RTTs

Note: this is different from RFC (because our network is all on localhost)

=> You may need to tune for your implementation/ your computer

Example upper/lower bounds

RTOmin ~= 100ms

RTOmax ~= 5sec

Computing RTO

Strategy: *measure* expected RTT based on ACKs received

Use exponentially weighted moving average (EWMA)

- RFC793 version ("smoothed RTT"):

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

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

α = "Smoothing factor": .8-.9

β = "Delay variance factor": 1.3—2.0

[redacted]

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

Out of order segments

More info: Lecture 16

Usually, make a "early arrival queue"

- When segment arrives, add to queue if it's not the next segment
  => What to store? You decide!
- As more segments arrive, check the top of the queue to see if it fills in any gaps

Zero window probing (ZWP)

More info: Lecture 16

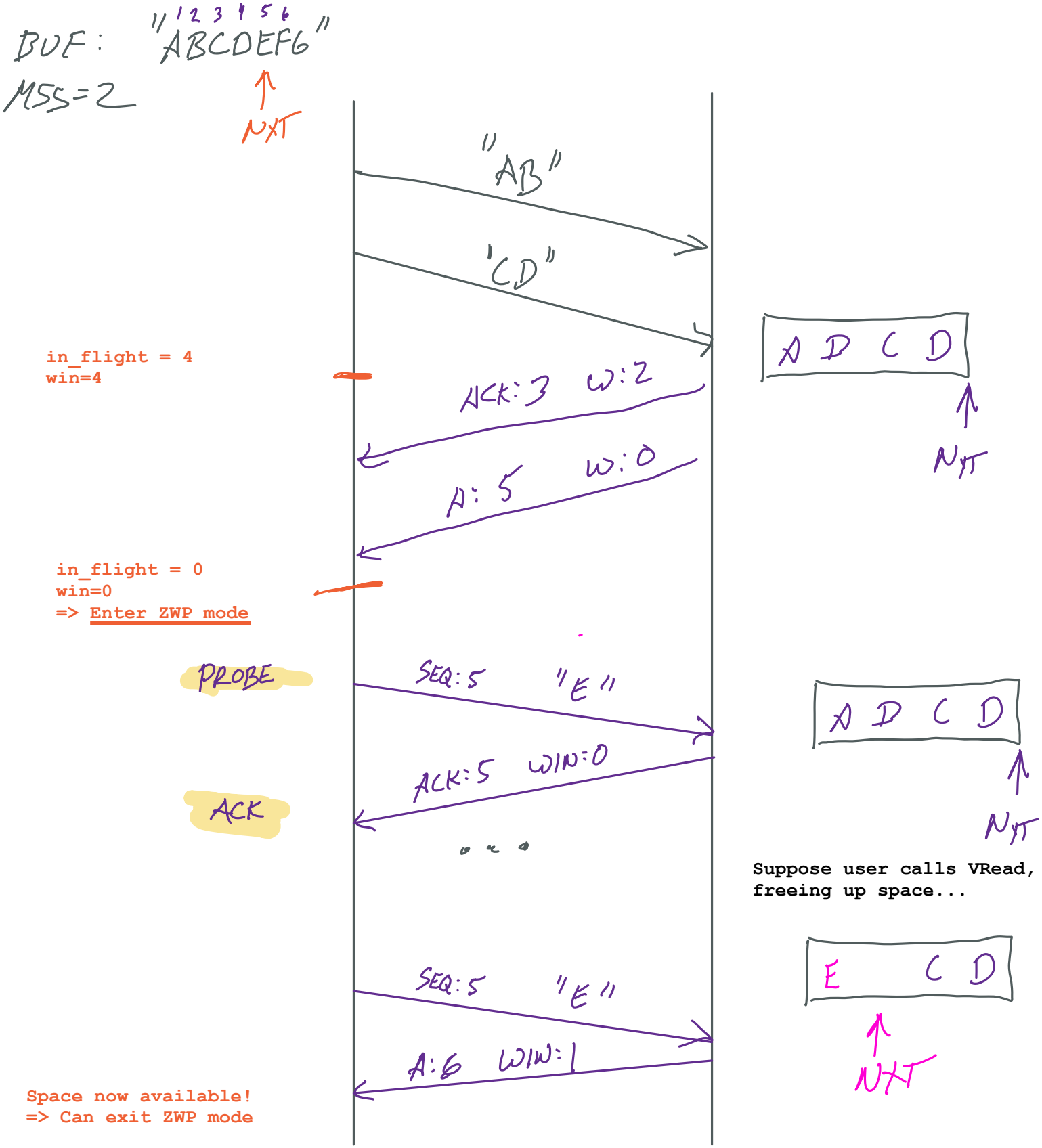
When receiver's window is full, sender enters zero window probing mode

- Stop sending segments
- At a periodic intervals, send 1 byte segments until receiver sends back window > 0 bytes

Send 1 byte of real data (whatever next data is in buffer)
=> Send one byte out of window ON PURPOSE
=> Receiver will send ACK (without adjusting its NXT), include updated window size

See example on next page

Scratch notes for ZWP: see recording for a live drawing, lecture 16 for more
Also, there's another example at end of these notes



Zero window probing

More info: Lecture 16

When receiver's window is full, sender enters zero window probing mode

- Stop sending segments
- At a periodic intervals, send 1 byte segments until receiver sends back window > 0 bytes

How to test?

- On one side, listen on a port: a 9999
- On other side, send a file

NO ONE IS READING!

Connection teardown

More info: Lecture 17

4-way connection close process => see the lecture for details

- VClose just starts the connection close process
  => TCB not deleted until connection goes to CLOSED state

How to test?

- Don’t leave sendfile/recv file to the end--try to test as you go
  - You WILL find bugs. Breathe. It’s going to be okay.

- Try to test each part individually, as shown here

Don’t be afraid to write some test code!
=> Eg. To test retransmissions, comment out your ACK processing and see what happens

Testing with packet loss

New REPL command in vrouter reference (out soon):

```
> drop 0.01   // Drop 1% of packets
> drop 0.5    // Drop 50% of packets (way too aggressive)


> drop 1   // Drop ALL packets (equivalent to “down”)


> drop 0  // Drop no packets
```

Also: can set by running vrouter with `--drop`

Custom vnet_run configurations

With ~30s of work, you can set up a config file for vnet_run to easily let you...

- Run custom configurations of vhost/vrouter (your h1, reference h2, etc.)
- Automatically configure drop rate at startup (save on typing!)
- Turn on logging

=> See recording for a demo (also "Custom vnet_run configurations" in Docs > "Tools and resources)

Reference implementation

- Our implementation of TCP
- Try it and compare with your version!

Note: we're using a new reference this year

- We've tested as best we can, but there may be bugs
- See Ed FAQ, docs FAQ for list of known bugs
- Let us know if you have issues!

⇒ If the spec disagrees with the reference implementation, the spec wins--don't propagate buggy behavior (please help us find any discrepancies!)

Closing thoughts

Do not underestimate these last parts--it will take time to debug and test them.

When stuck, take a break and come back to it. It will help.

=> Do NOT wait until the last minute.

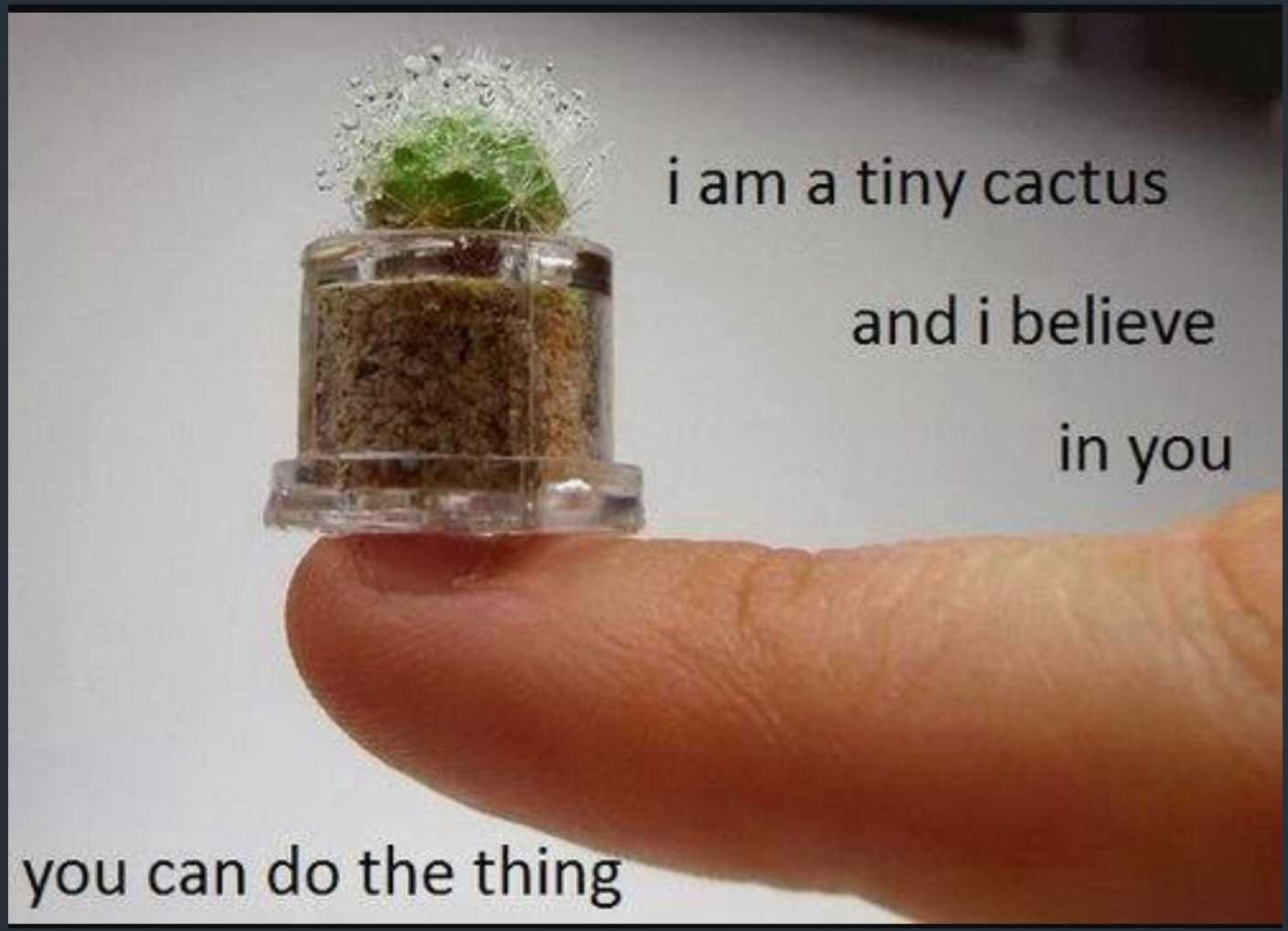
Don’t panic.

SUN	MON	TUE	WED	THU	FRI	SAT
29	30	31	Apr 1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25

You got this!!!!! But make sure you pace yourself.

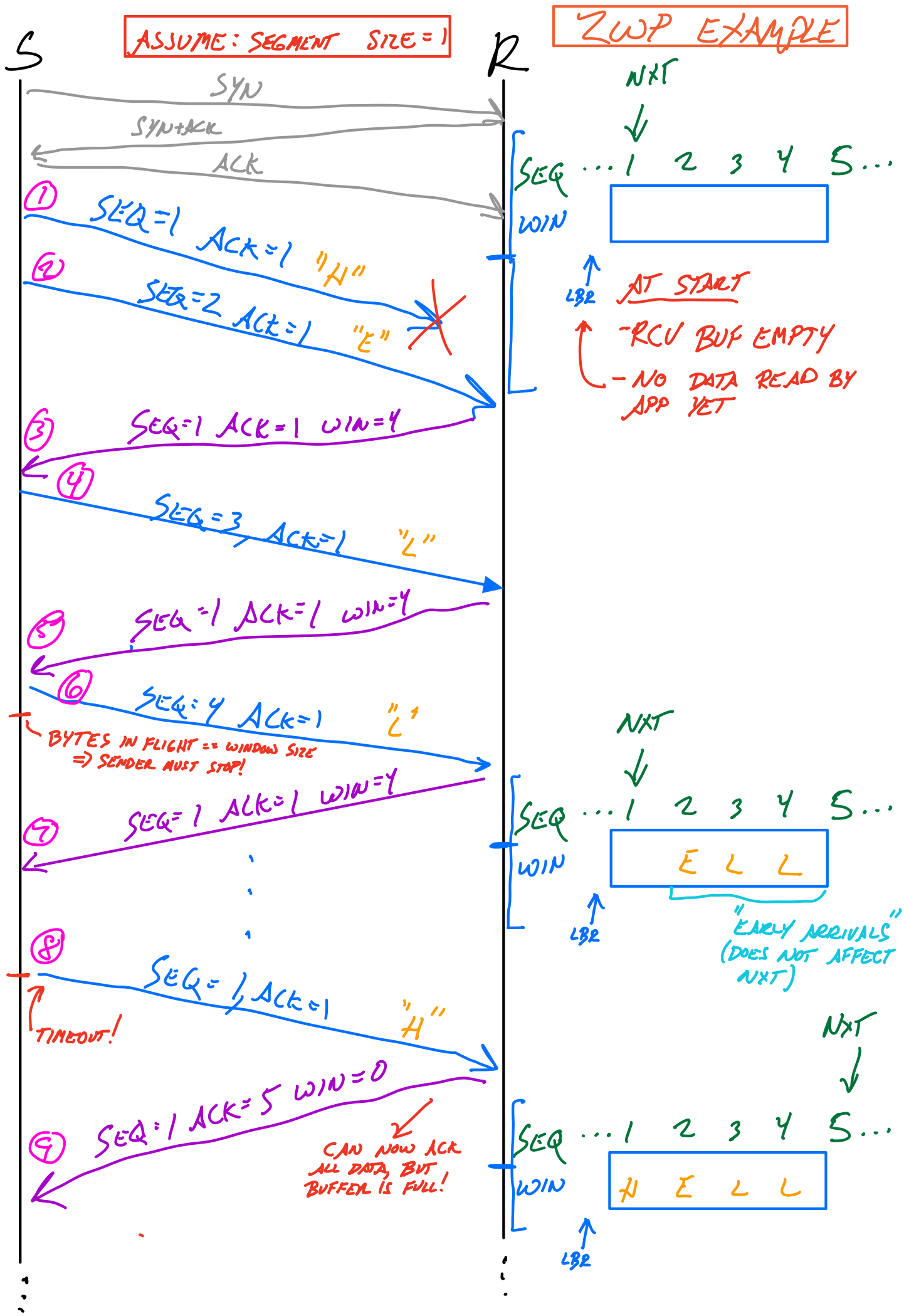
You are here

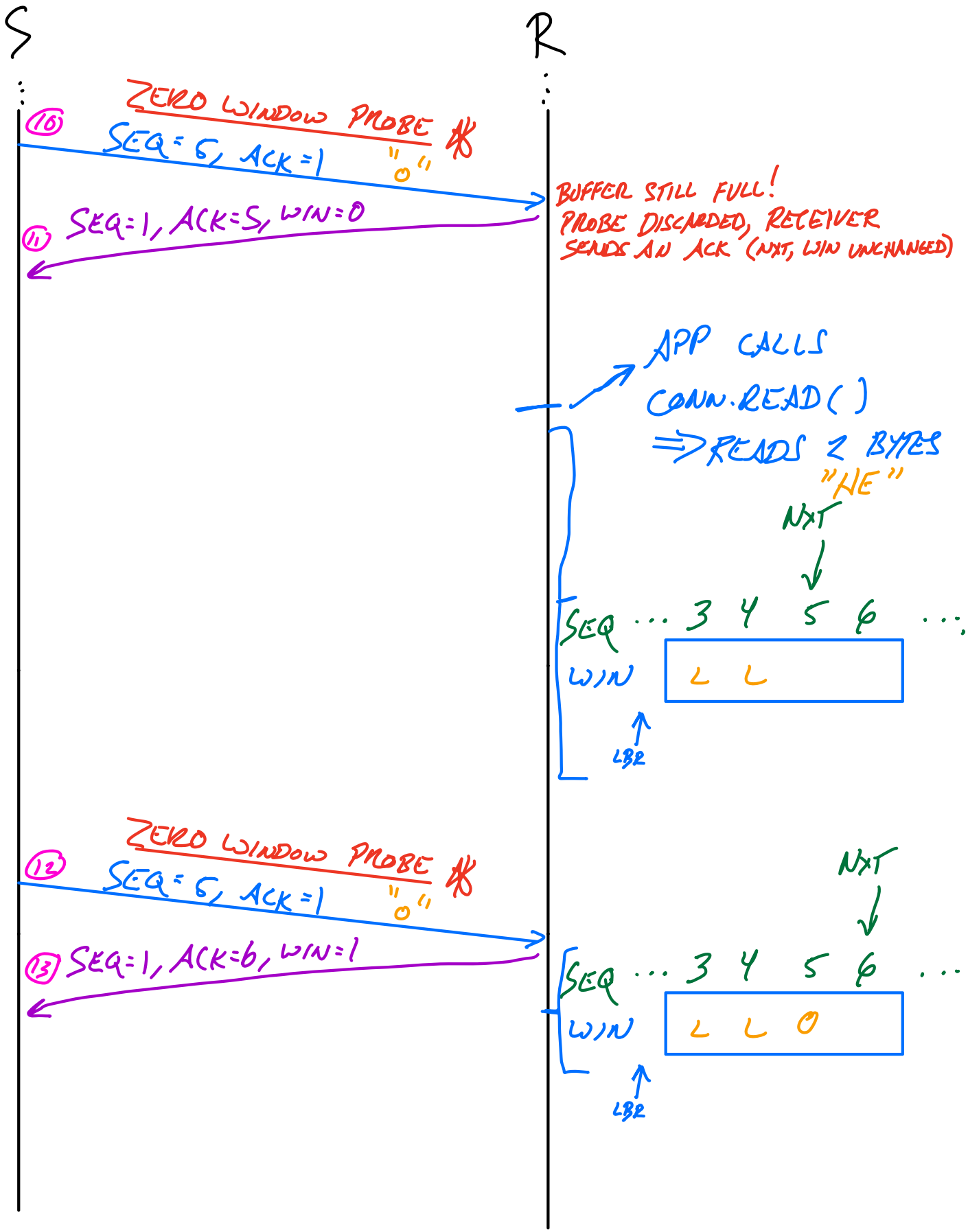
TCP due


i am a tiny cactus
and i believe
in you
you can do the thing

The next page has an old example for zero window probing and retransmissions—it's a bit more involved than we discussed in the gearup but should be useful for seeing how it works and interacts with your buffers.

After that is an annotated example of how zero-window probing should look in wireshark


ASSUME: SEGMENT SIZE = 1
ZWP EXAMPLE
S
R
SYN
SYN+ACK
ACK
1 SEQ=1 ACK=1 "H"
2 SEQ=2 ACK=1 "E"
3 SEQ=1 ACK=1 WIN=4
4 SEQ=3 ACK=1 "L"
5 SEQ=1 ACK=1 WIN=4
6 SEQ=4 ACK=1 "L"
BYTES IN FLIGHT == WINDOW SIZE
=> SENDER MUST STOP!
7 SEQ=1 ACK=1 WIN=4
8 SEQ=1 ACK=1 "H"
TIMEOUT!
9 SEQ=1 ACK=5 WIN=0
CAN NOW ACK ALL DATA, BUT BUFFER IS FULL!
NXT
SEQ
WIN
LBR
... 1 2 3 4 5 ...
AT START
- RCV BUF EMPTY
- NO DATA READ BY APP YET
E L L
"EARLY ARRIVALS" (DOES NOT AFFECT NXT)
H E L L



* NOTE: ZERO WINDOW PROBES ARE ALWAYS ONE BYTE, REGARDLESS OF THE SEGMENT SIZE. IN THIS EXAMPLE, WE HAVE BEEN USING 1-BYTE SEGMENTS THROUGHOUT — THIS IS JUST A COINCIDENCE!