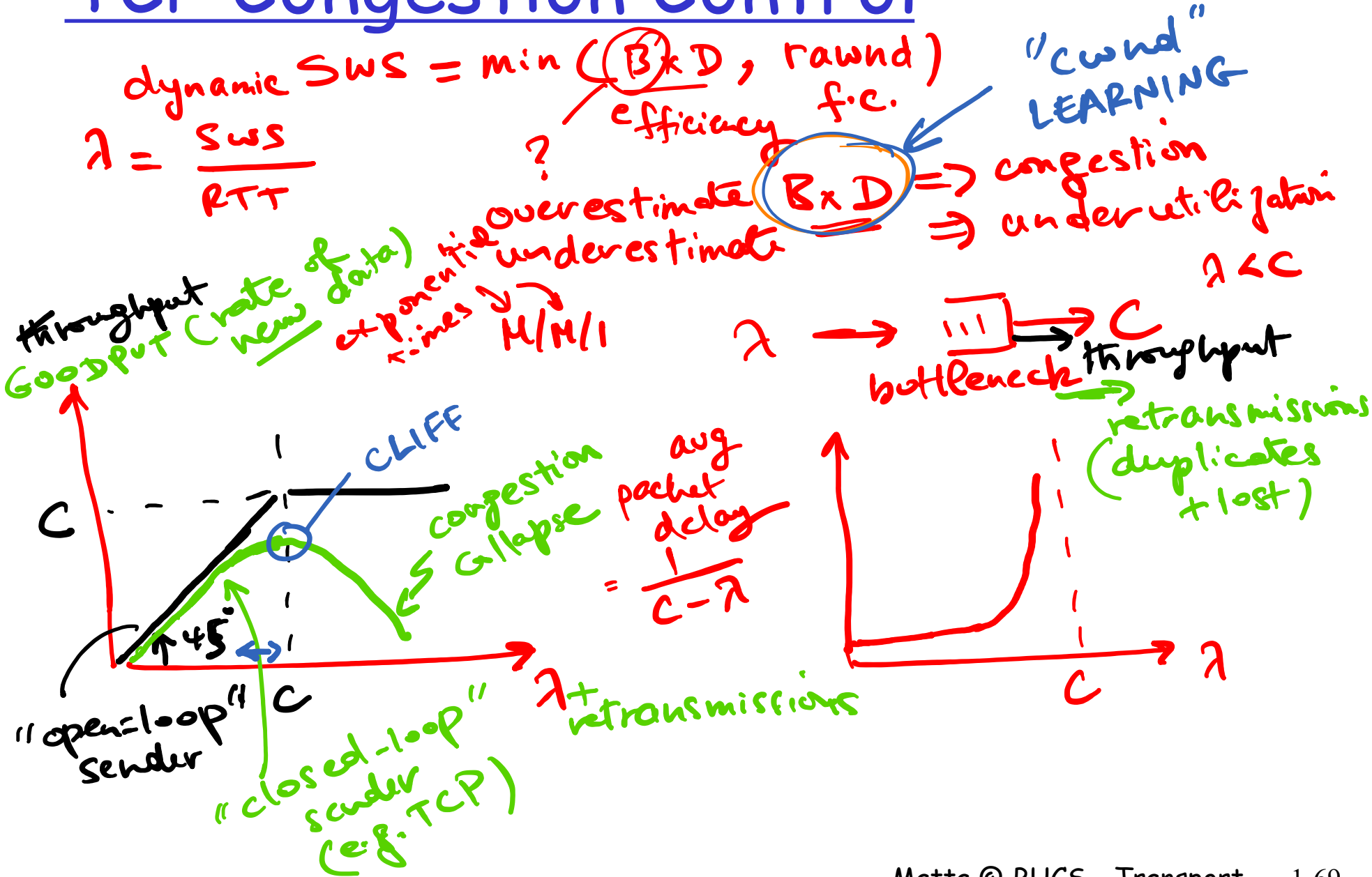
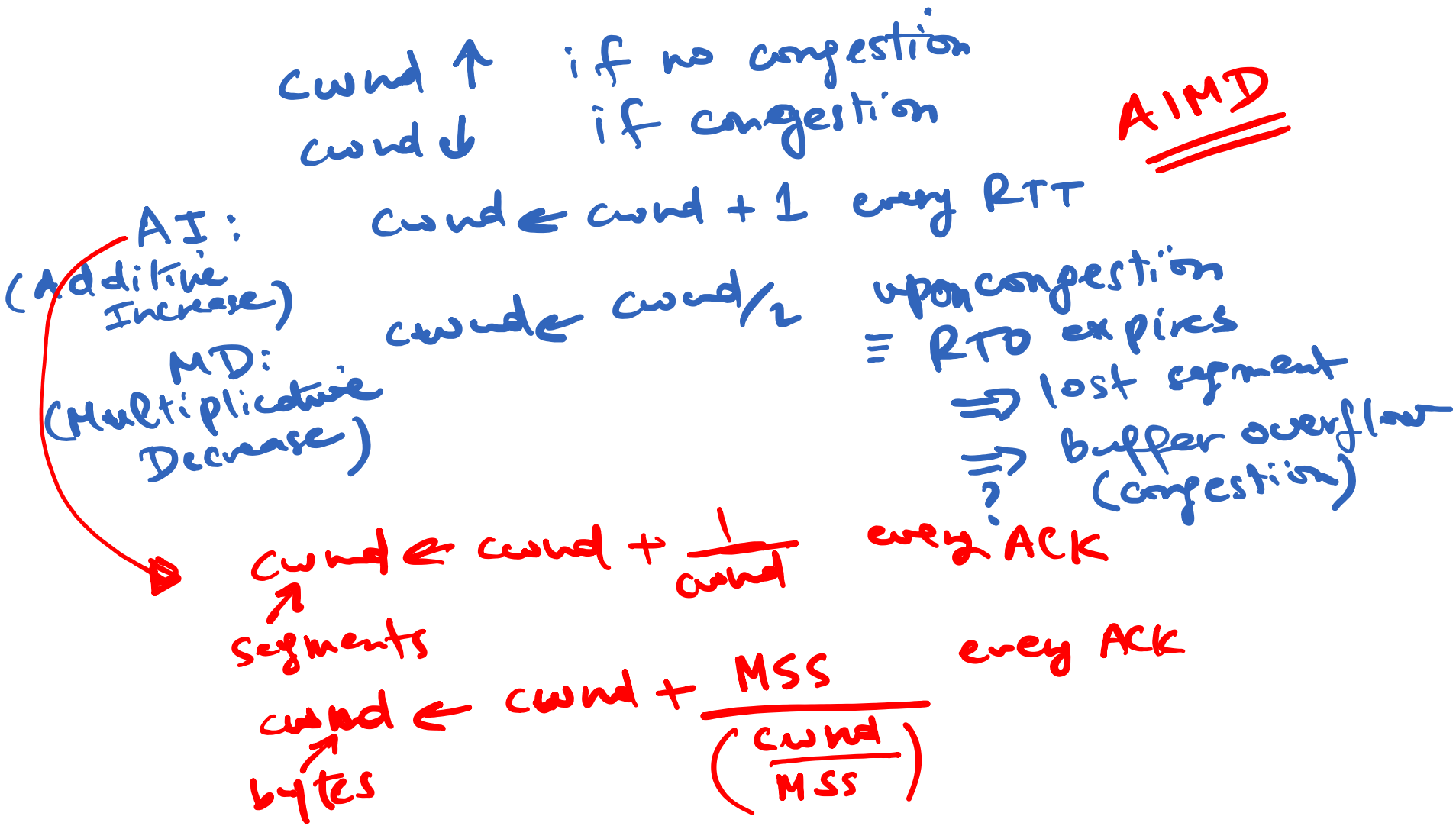
TCP Congestion Control



TCP Congestion Control



TCP Congestion Control

Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
- New state variable per connection: CongestionWindow
  - limits how much data source has in transit

MaxWin = MIN(CongestionWindow, AdvertisedWindow)

- Idea:
  - increase CongestionWindow when congestion goes down
  - decrease CongestionWindow when congestion goes up
- *Question:* how does the source determine whether or not the network is congested?
- *Answer:* a timeout occurs
  - timeout signals that a segment was lost
  - segments are seldom lost due to transmission error
  - lost segment implies congestion

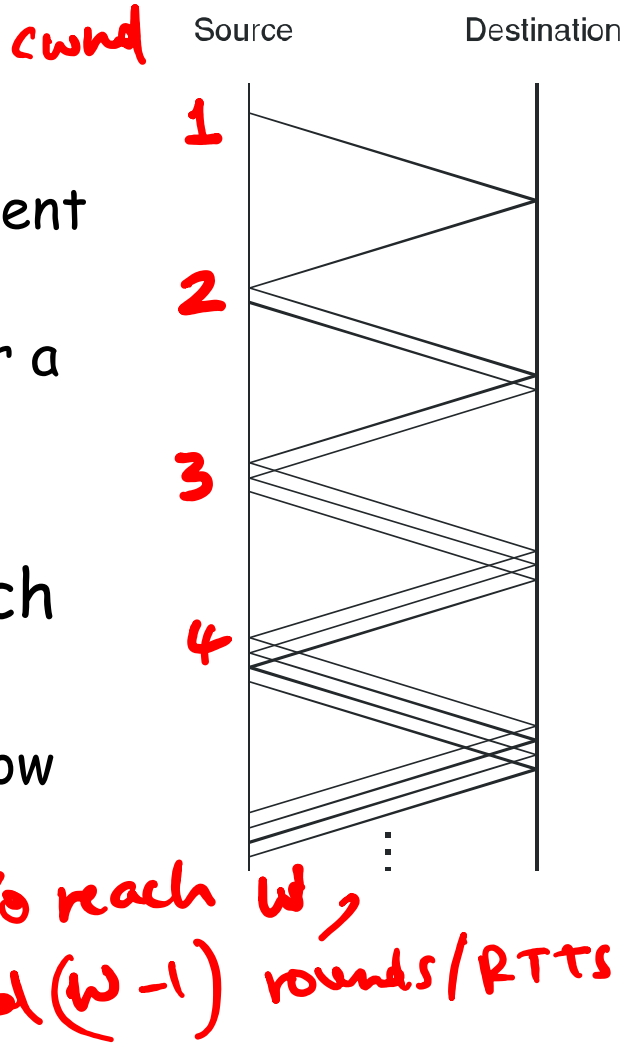
AIMD

- Algorithm:
  - increment CongestionWindow by one segment per RTT (*linear increase*)
  - divide CongestionWindow by two whenever a timeout occurs (*multiplicative decrease*)

- In practice: increment a little for each ACK

  Increment = (MSS x MSS)/CongestionWindow

  CongestionWindow += Increment

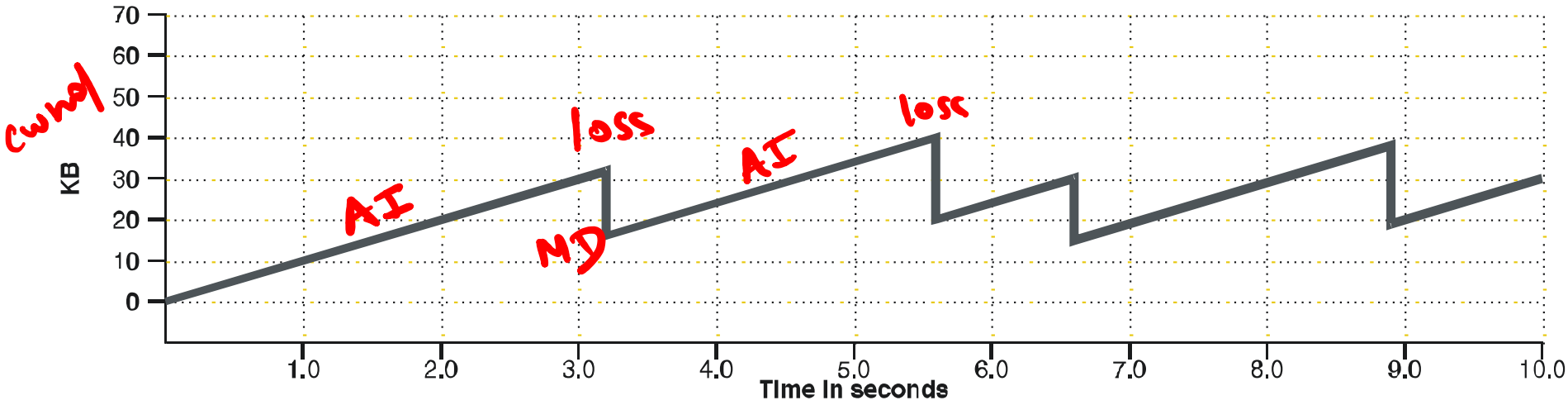


Sawtooth behavior

AIMD

TCP is a feedback control system

- Example trace



Slow Start

Exponential Increase

cwnd ← 2^{*}cwnd every RTT

cwnd ← cwnd + cwnd every RTT

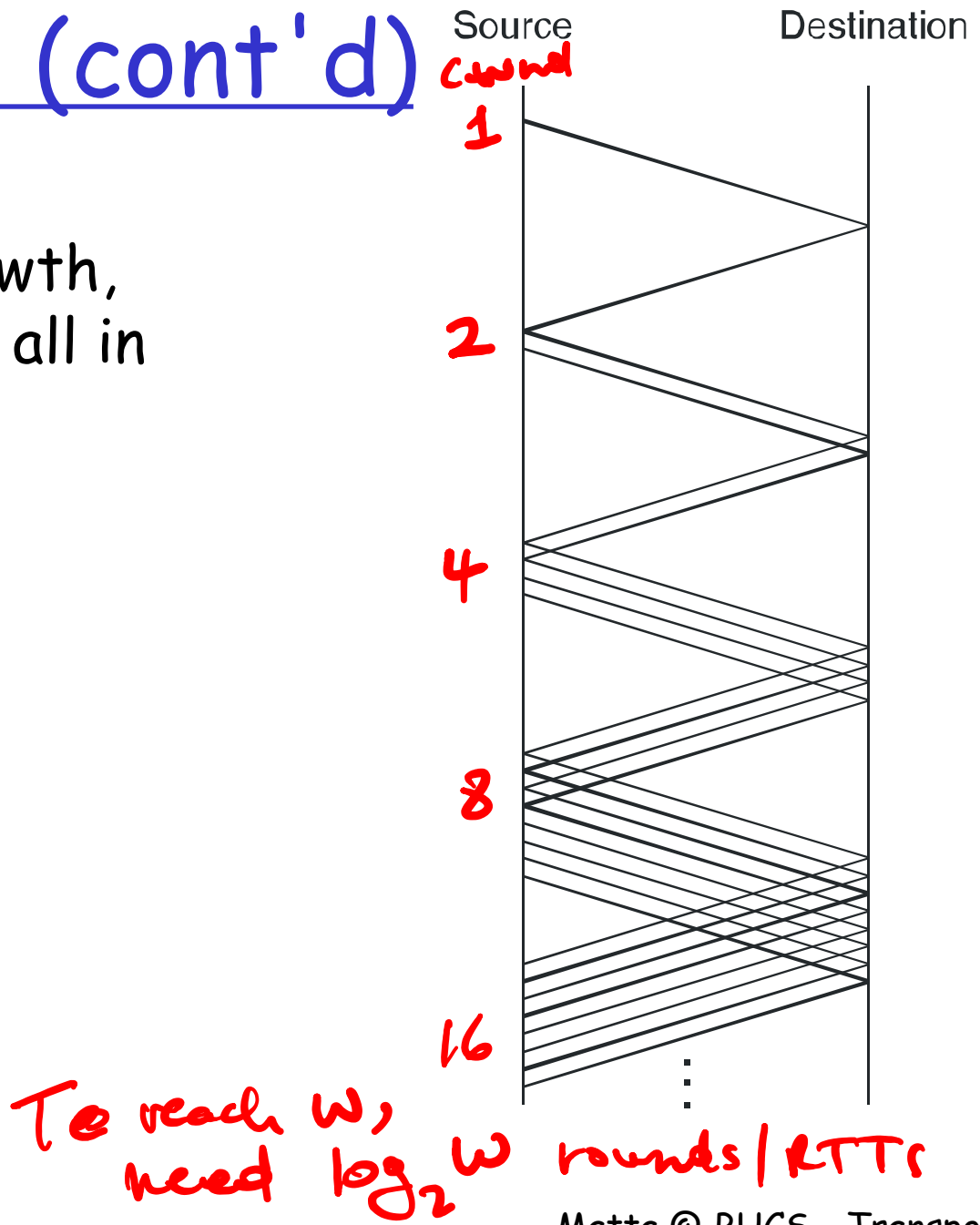
SS : cwnd ← cwnd + 1 every ACK

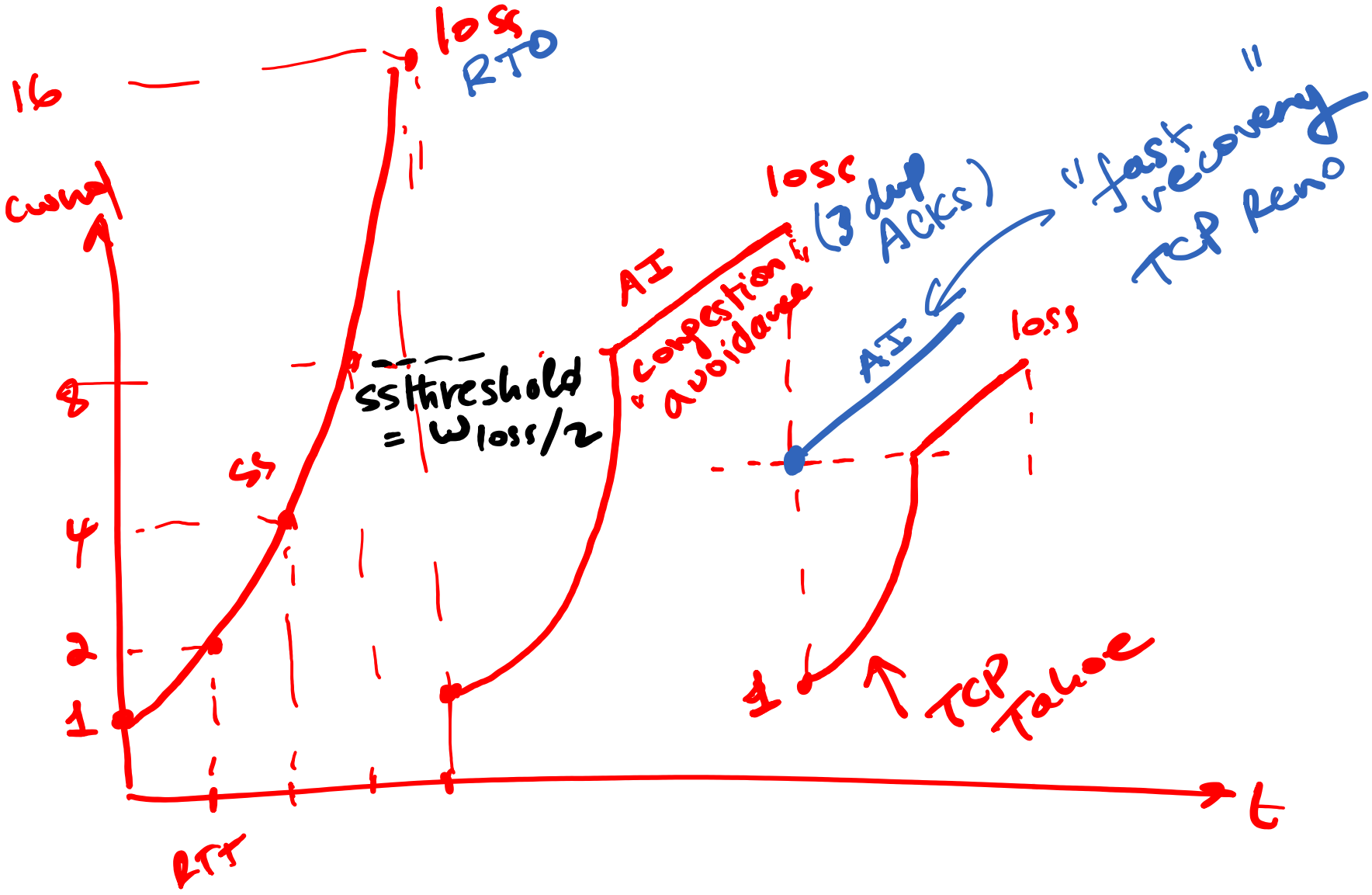
Slow Start

- Objective: determine the available capacity in the first place
  - when first starting connection
  - when connection recovers after a timeout
- Idea:
  - begin with CongestionWindow = 1 segment
  - double CongestionWindow each RTT (increment by 1 segment for each ACK)

Slow Start (cont'd)

- Exponential growth, but slower than all in one blast

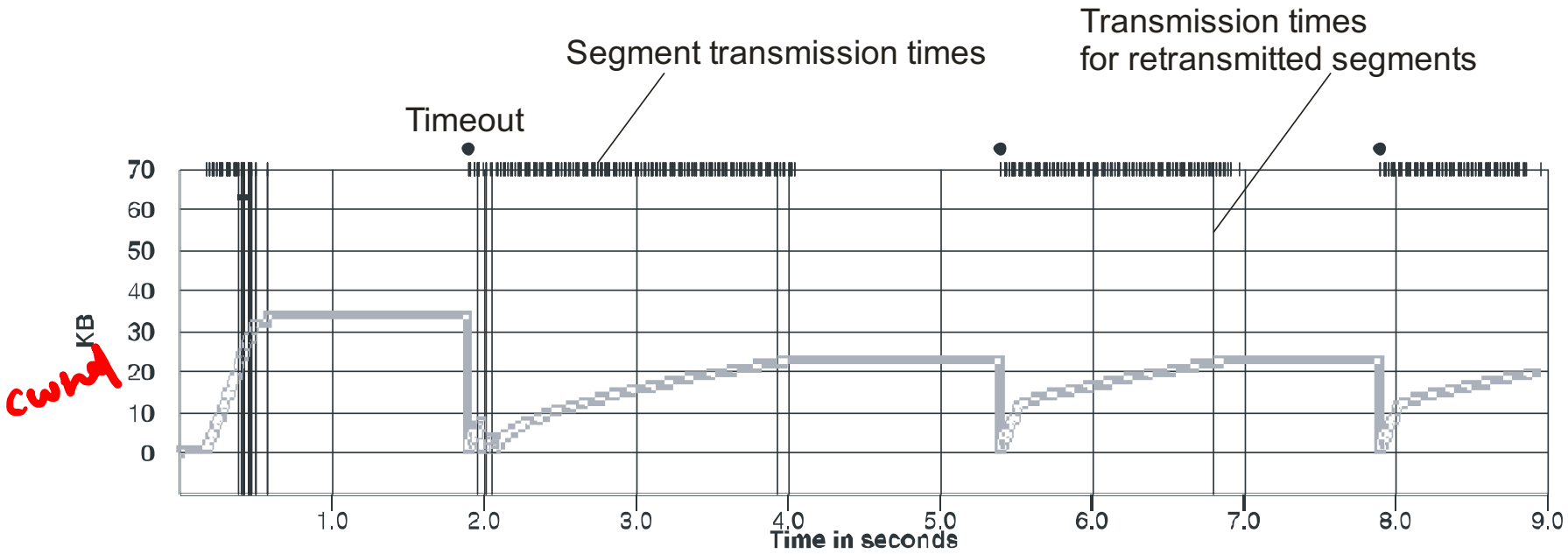



16
loss
RTO
cwnd
8
4
2
1
ssthreshold
= wloss/2
SS
AI
congestion
avoidance
loss
(3 dup
ACKs)
"fast recovery"
TCP Reno
AI
loss
1
TCP Tahoe
t
RTT

TCP Congestion Algorithm

On a timeout, half the current window size is recorded in ssthresh

```
if (cwnd < ssthresh) // if we're still doing
                     // slow-start, open window exponentially
    cwnd += 1
else // otherwise do Congestion Avoidance
     // linear increase
    cwnd += 1/cwnd
```



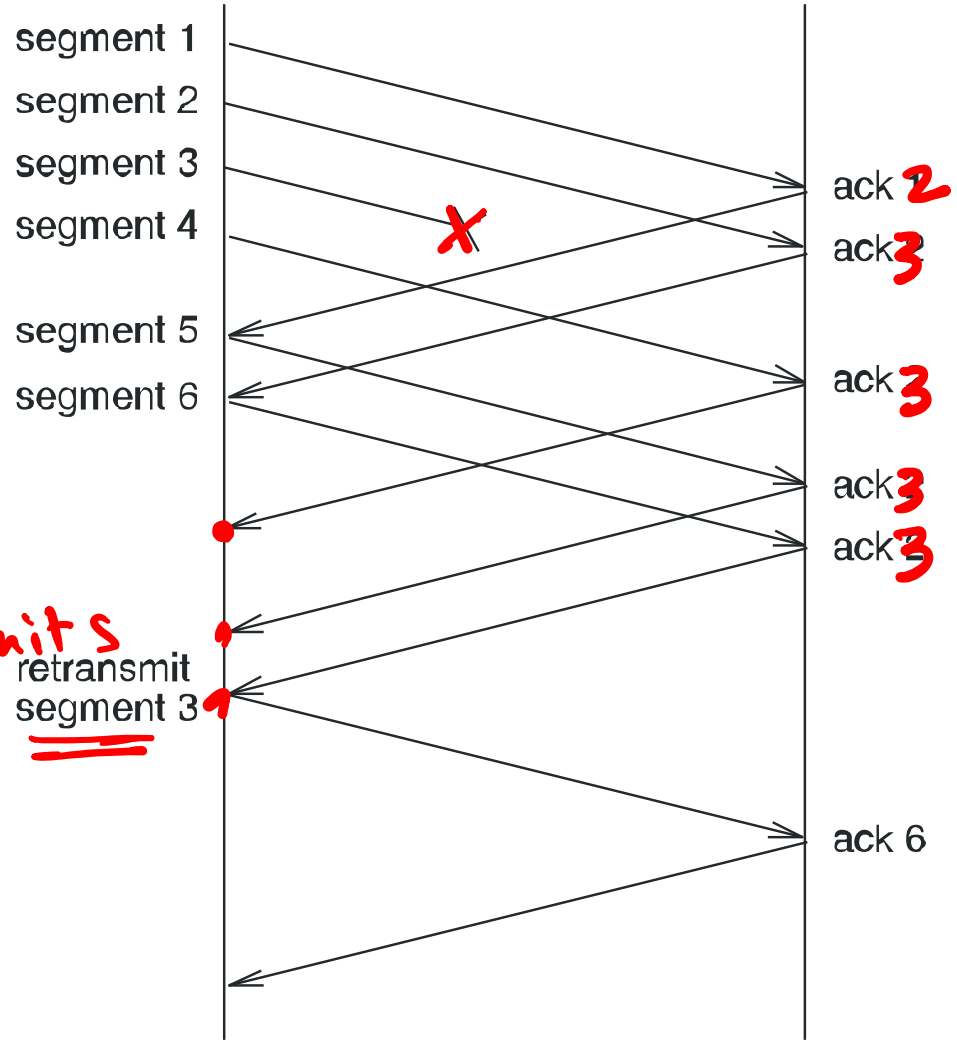
- Example trace
- cwnd stays flat if no ACKs are received
- Problem: lose up to half a CongestionWindow 's worth of data during slow start

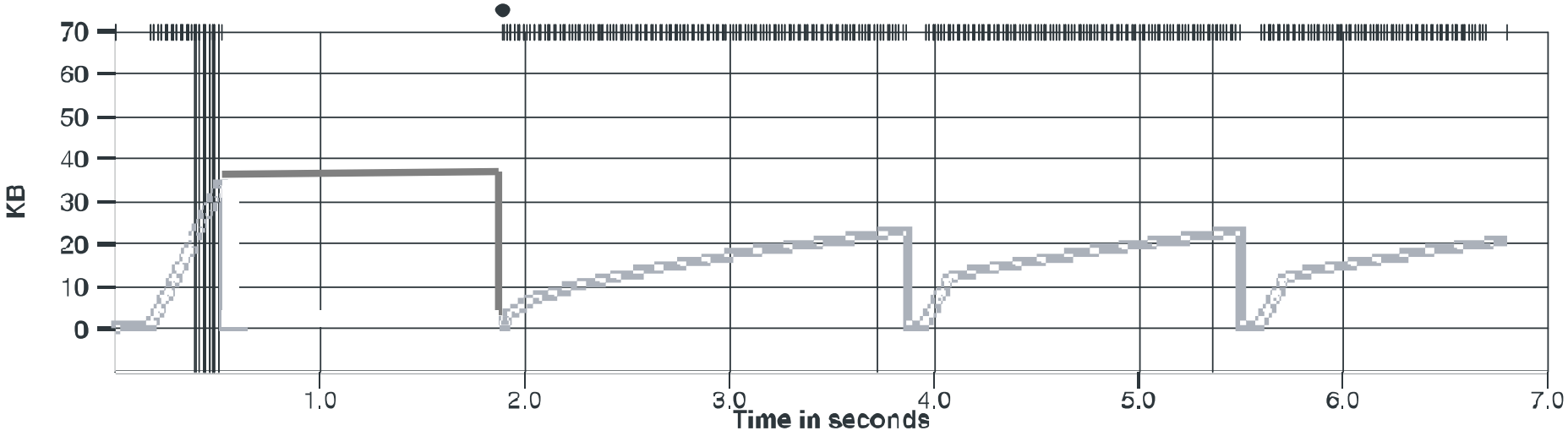
Fast Retransmit and Fast Recovery

- Problem: coarse-grain TCP timeouts lead to idle periods
- Fast retransmit: use duplicate ACKs to trigger retransmission

Internet channel non-FIFO

3 dup ACKs
⇒ fast retransmits segment 3





- Long periods during which cwnd stays flat are eliminated

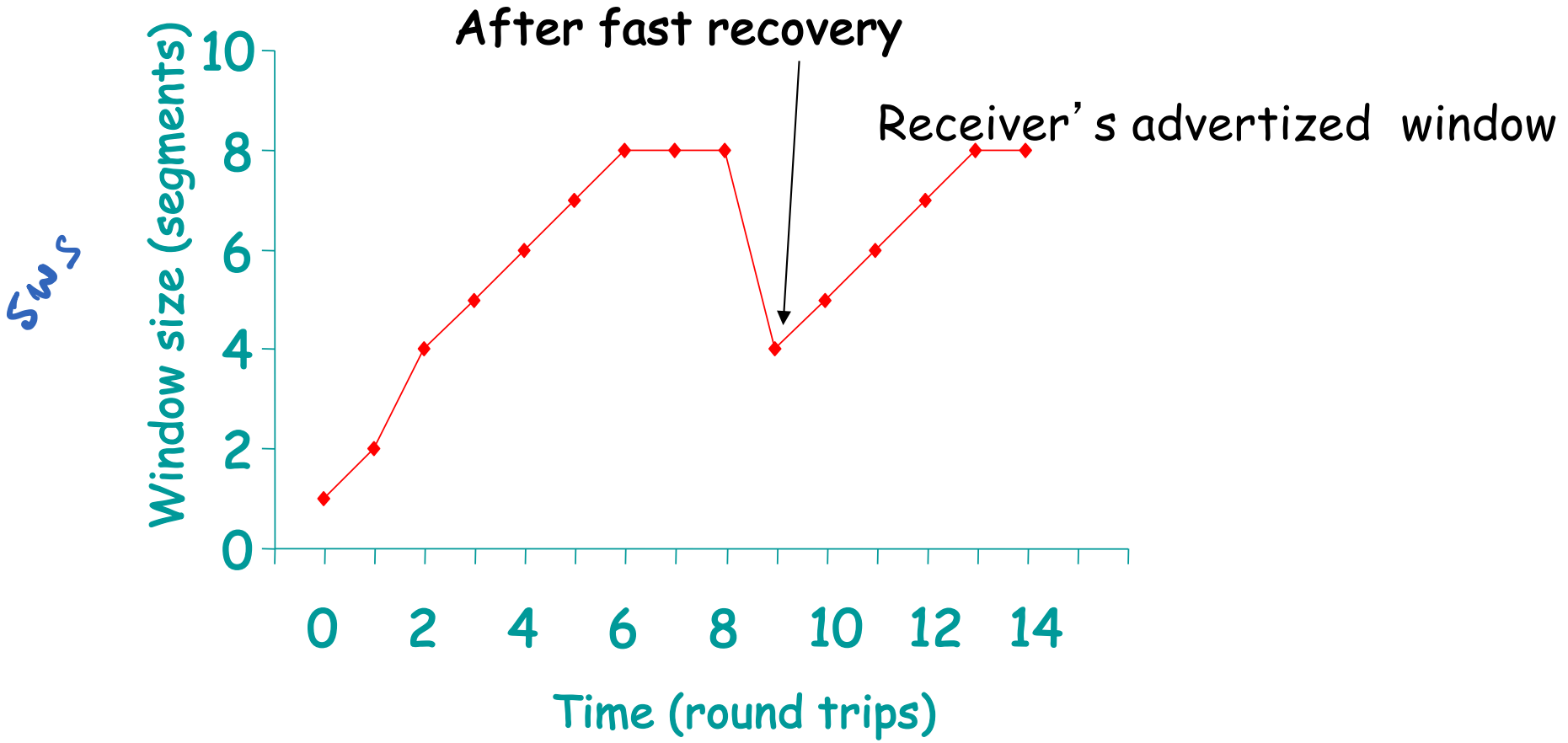
loss event
RTO ⇒ "severe congestion" ⇒ cwnd ← 1 then SS, AI
3 dup ACKs ⇒ "low/moderate congestion" ⇒ cwnd ← cwnd/2 then AI

Fast Recovery

- Fast recovery: remove the slow start phase; go directly to half the last successful CongestionWindow
- TCP Tahoe includes all mechanisms *except* fast recovery
- TCP Reno adds fast recovery

Fast Recovery

swS = min(cwnd, rawnd)



- After fast retransmit and fast recovery, window size is reduced by half

Putting it Together: TCP Reno

On every ACK

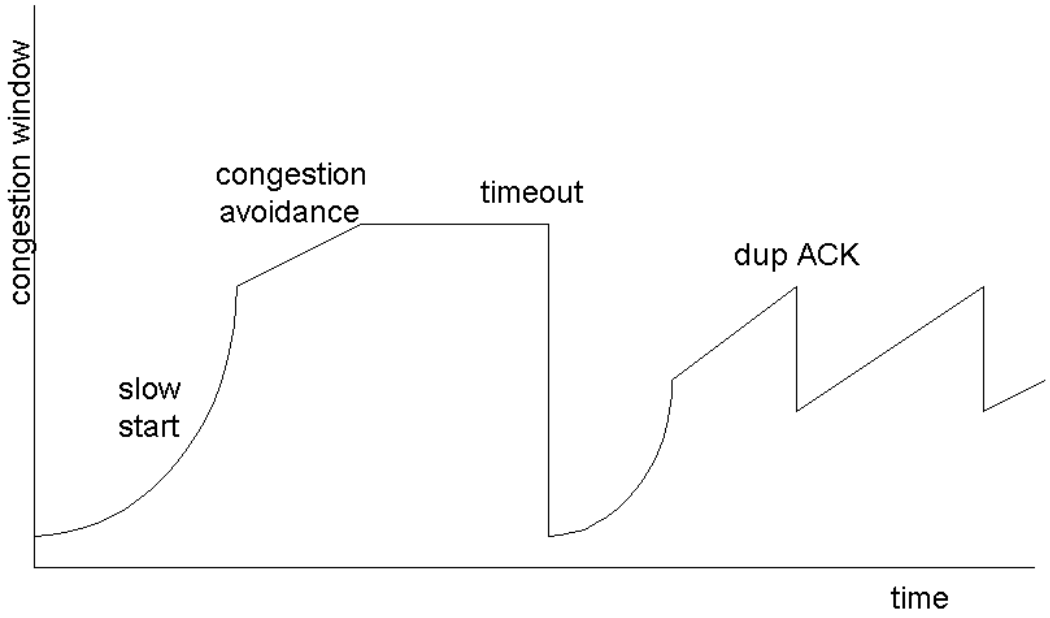
```
if (window < threshold)      // slow start phase
    window += 1              // double window every RTT
else                         // congestion avoidance
    window += 1 / window     // increment by 1 every RTT
```

On timeout

```
threshold = window / 2
window = 1
```

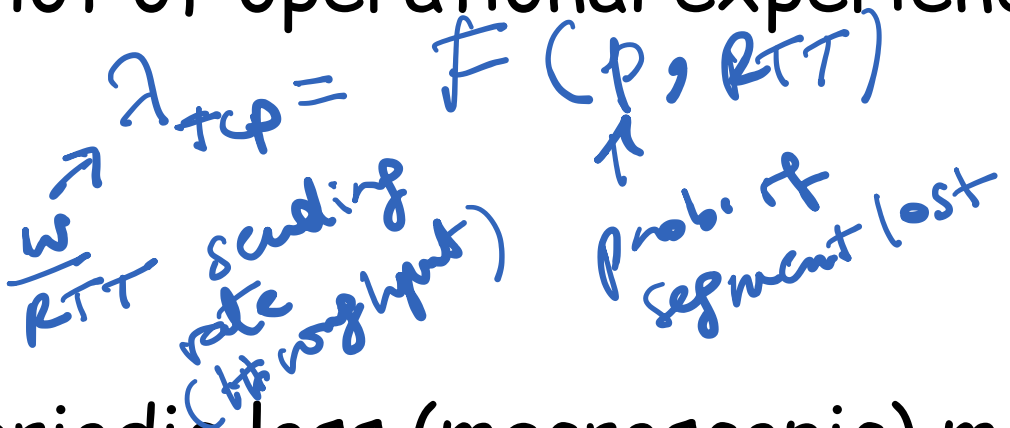
On duplicate acknowledgments

```
threshold = window = window / 2
                    // fast recovery
```



TCP Performance

- Effective over a wide range of capacities
- A lot of operational experience



- Periodic loss (macroscopic) model shows that throughput is inversely proportional to
  - square root of loss probability *p*
  - RTT
  - average sending rate = sqrt(1.5/*p*) / RTT