6.033, Spring 2014

TCP Congestion Control

Dina Katabi & Sam Madden

nms.csail.mit.edu/~dina

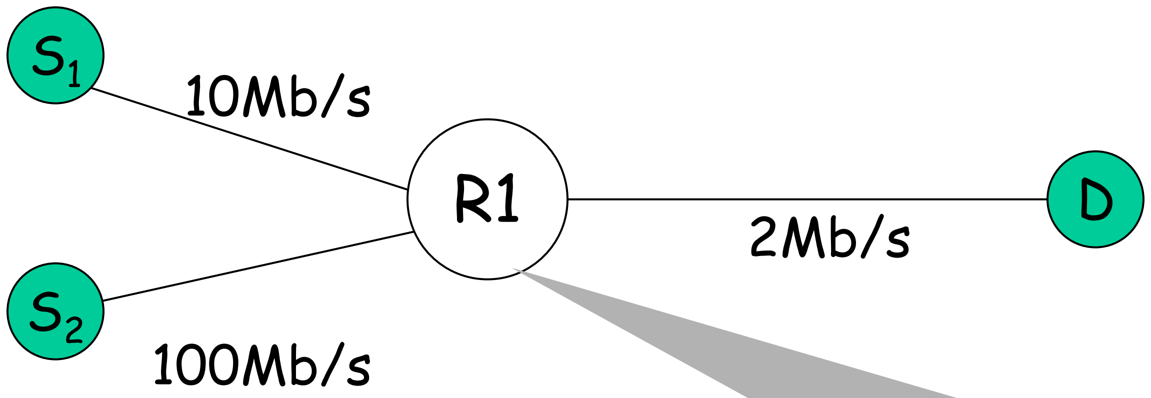
Sharing the Internet

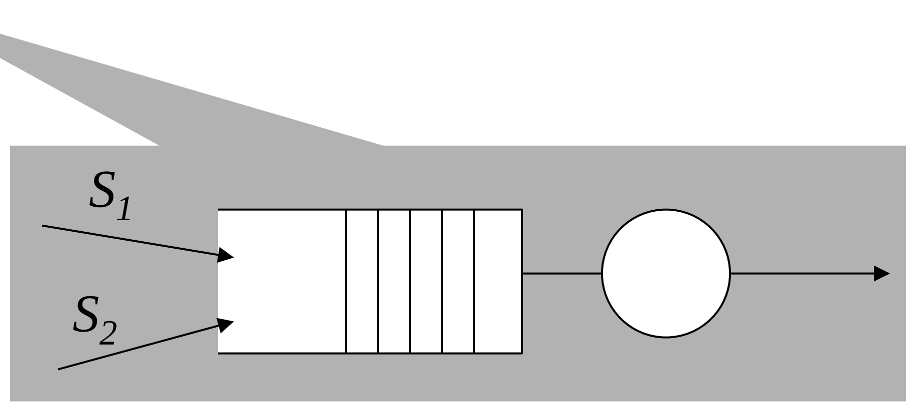
How do you manage resources in a huge system like the Internet, where users with different interests share the same links?

Difficult because of:

- Size
  - Billions of users, links, routers
- Heterogeneity
  - bandwidth: 9.6Kb/s (then modem, now cellular), 10 Tb/s
  - latency: 50us (LAN), 133ms (wired), 1s (satellite), 260s (Mars)

Congestion

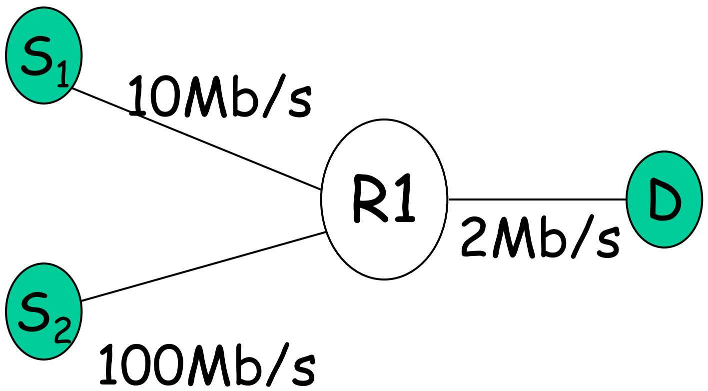




- Sources compete for link bandwidth, and buffer space
- Why a problem?
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delays (queuing in router buffers)
  - In many situations will result in < 2 Mb/s of throughput for the above topology (congestion collapse)

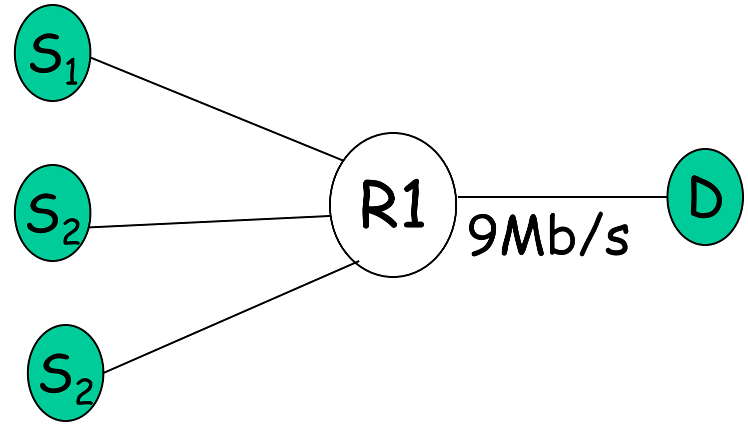
Objectives of Congestion Control Efficiency & Fairness

- Efficiency
  - Maximize link utilization
  - Minimize queue size (i.e., delay)
  - Minimize packet drops
- Many solutions!
  - (S_1= 1 Mb/s, S_2= 1 Mb/s) and (S_1=1.5 Mb/s, S_2=0.5 Mb/s) are both efficient.
- Want Fairness



Fairness

- Max-Min Fairness
  - At each bottleneck, user gets min(user's demand, fair share)
  - User's rate is the minimum max-min fair rate along the path



Demands $\begin{cases} \rho_1 = 1\text{Mb/s} \\ \rho_2 = 7\text{Mb/s} \\ \rho_3 = \infty \end{cases}$

Max-min Fair Rates $\begin{cases} \mu_1 = 1\text{Mb/s} \\ \mu_2 = 4\text{Mb/s} \\ \mu_3 = 4\text{Mb/s} \end{cases}$

TCP

TCP

- TCP provides reliability & congestion control
- Reliable transmission ensures the receiver's application receives the correct and complete data sent by the sender's application
  - TCP recovers from lost packets, eliminates duplicates and ensures in-order packet delivery to the application
  - Reliability was discussed in 6.02
- Congestion control
  - Sender reacts to congestion and discovers its fair and efficient send rate

TCP Cong. Cont.

- Basic Idea:
  - Send a few packets. If a packet is dropped decrease rate. If no drops, increase rate
  - How does TCP detect drops?
    - Packets have sequence numbers
    - Receiver acks the next expected sequence number (i.e., if received 1, it acks 2 saying it is expecting 2 to arrive next. Note that TCP implementations ack bytes but for simplicity we talk about acking packets.)

TCP controls throughput via the congestion window

- Congestion window is the number of outstanding packets, i.e., number of packets sender can send without waiting for an ack
- TCP is <u>window-based</u>: sources change their sending rate by modifying the window size: "cwnd"
  - Avg. Throughput = (Avg. cwnd) /(Avg. RTT)
- Why not changing rate directly?
  - Window protocols are easy to implement (no need for accurate timers, i.e., works for slow machines an sensors)

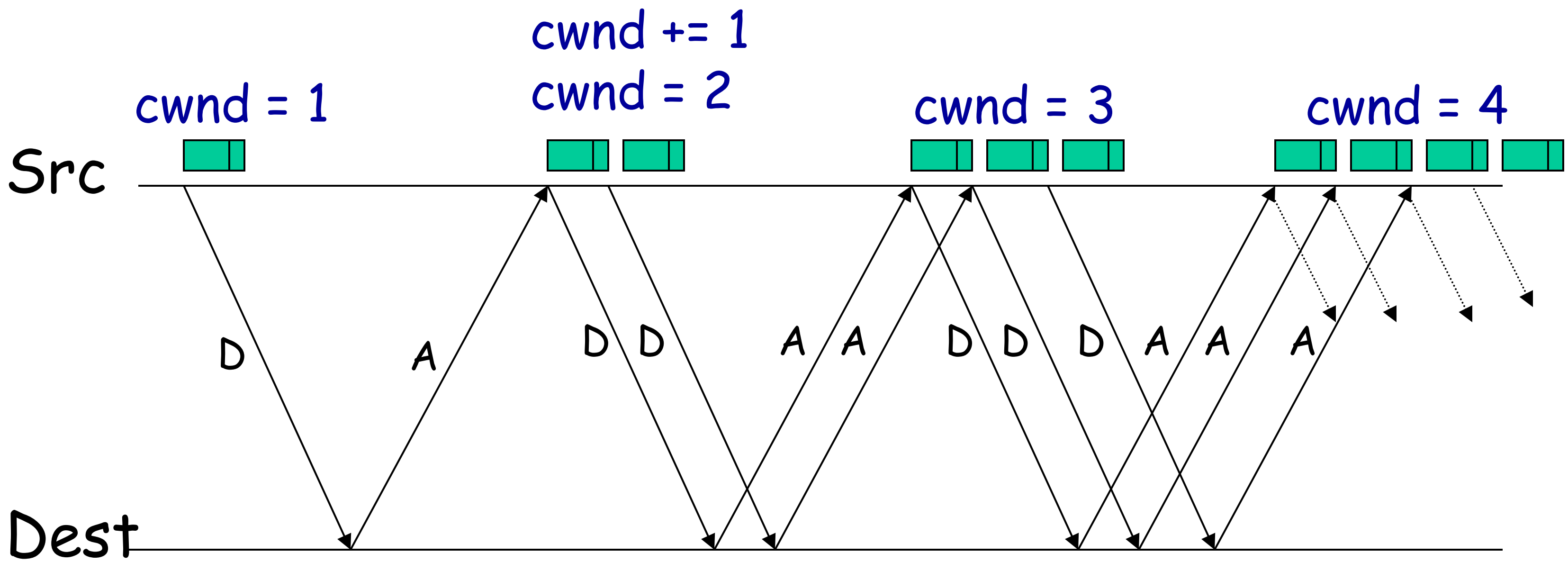
How much should TCP Increase/decrease?

- Probe for the correct sending window
- Additive Increase / Multiplicative Decrease (AIMD)
  - Every RTT:

    No loss: cwnd = cwnd + 1

    A loss: cwnd = cwnd /2

Additive Increase

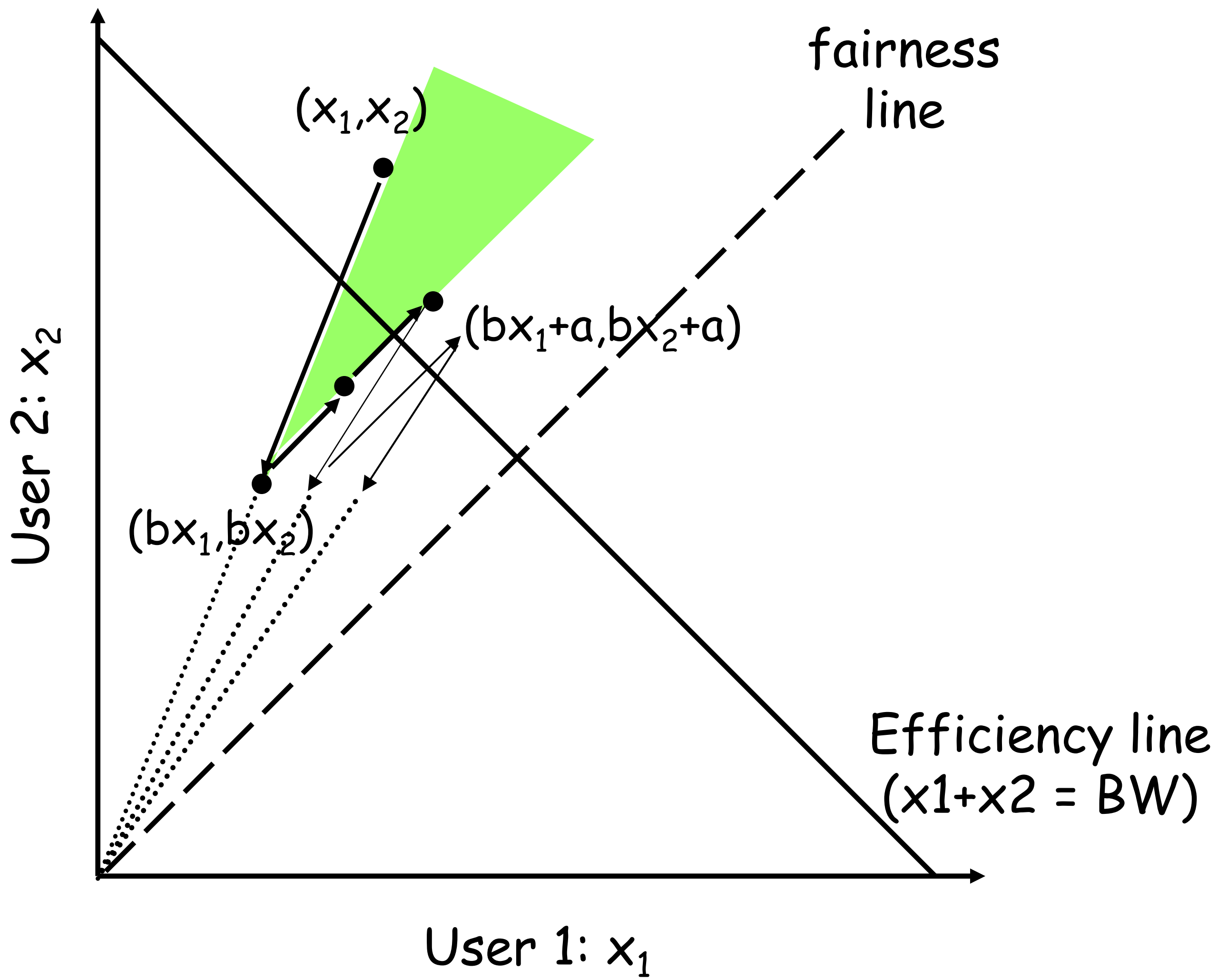


Actually,

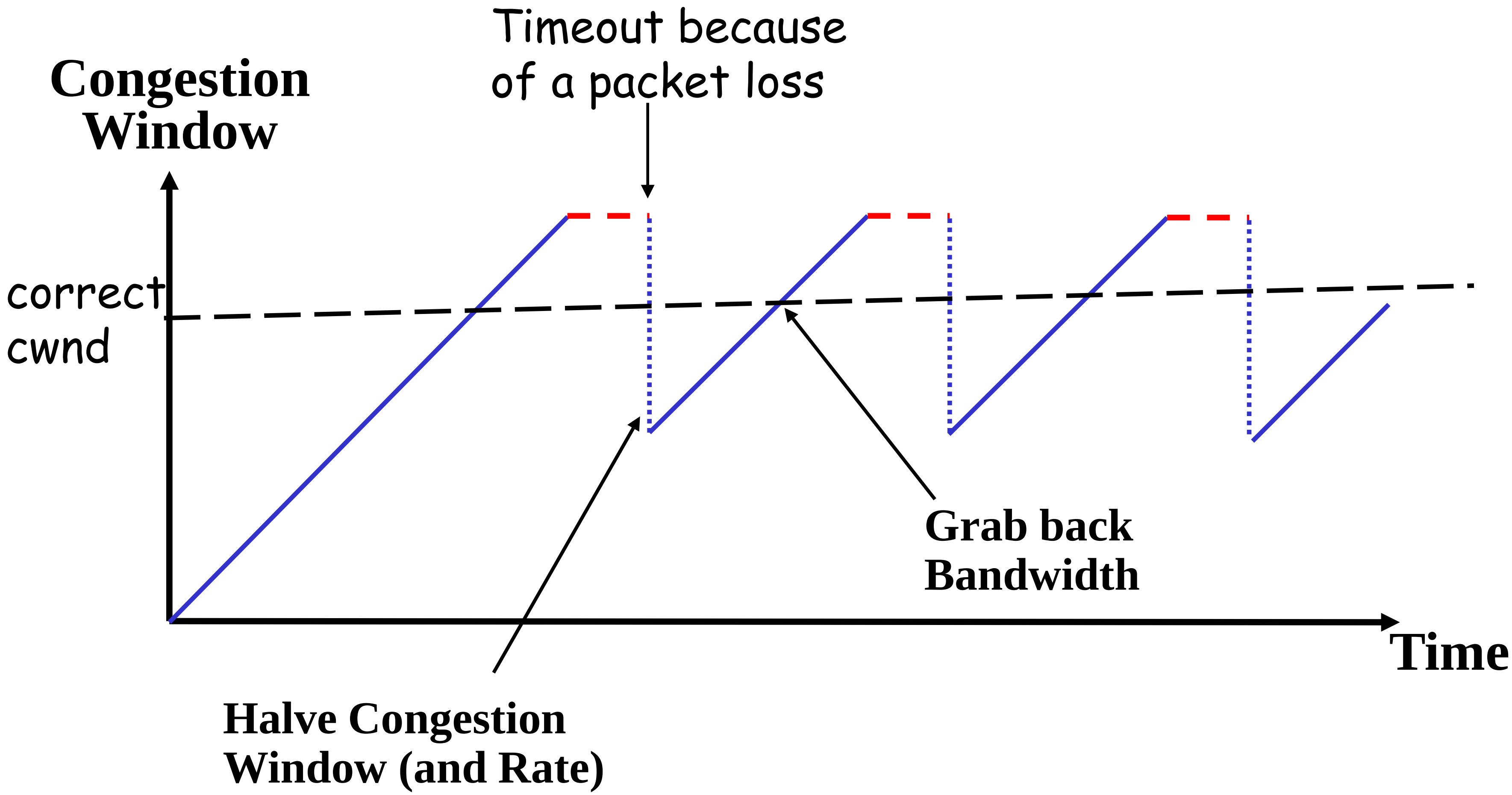
On ack arrival: cwnd = cwnd + 1/cwnd

On timeout: cwnd = cwnd /2

AIMD Leads to Efficiency and Fairness



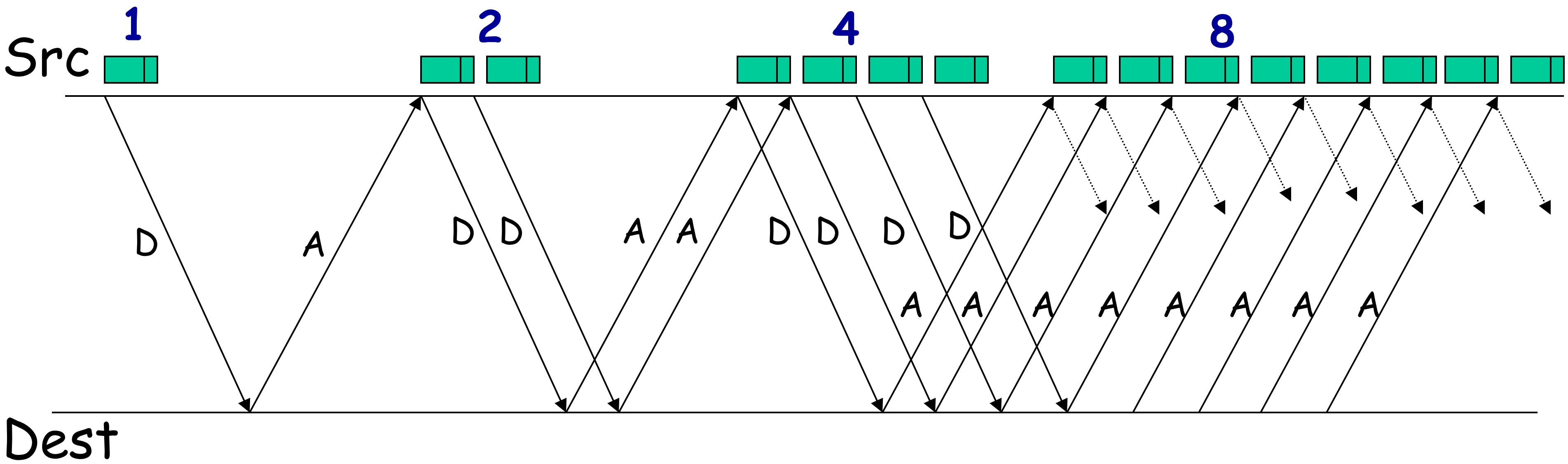
TCP AIMD



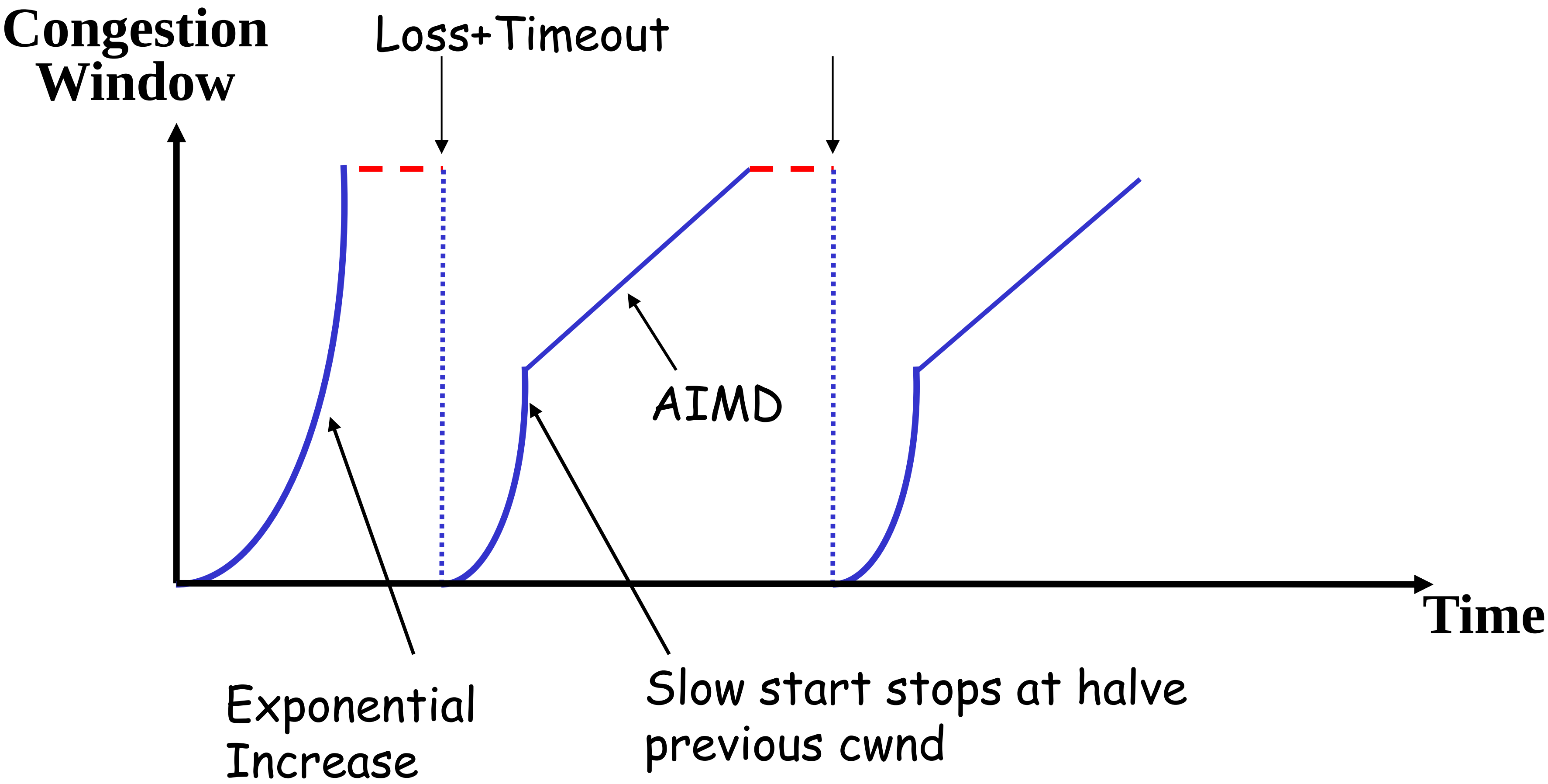
Need the queue to absorb these saw-tooth oscillations

"Slow Start"

- Cold start a connection at startup or after a timeout
- At the beginning of a connection, increase exponentially
  - On ack arrival: cwnd += 1



Adding Slow Start



Tweaking TCP

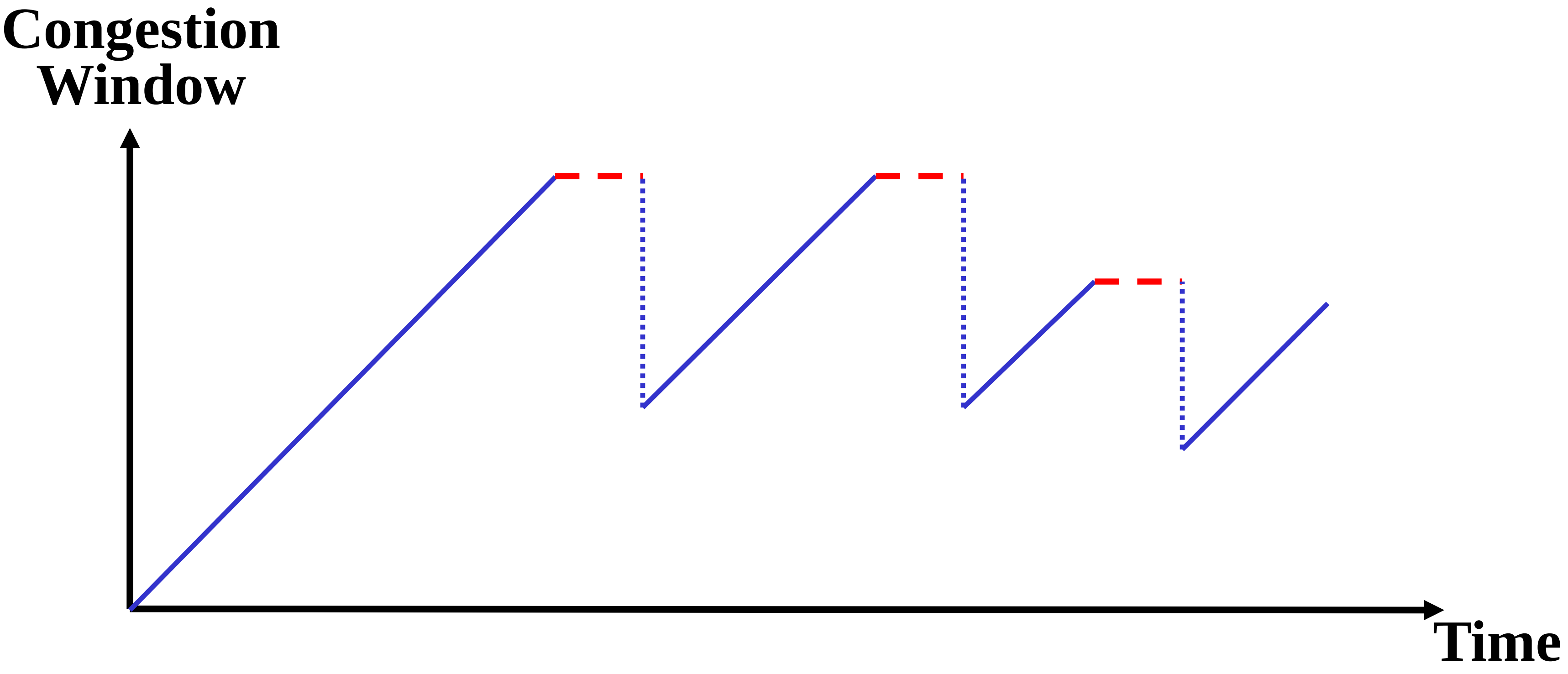
Fast Retransmit

- Timeouts are too slow
- When packet is dropped, receiver still acks the next in-order packet
- Use 3 duplicate ACKs to indicate a drop
  - Why 3? When this does not work?

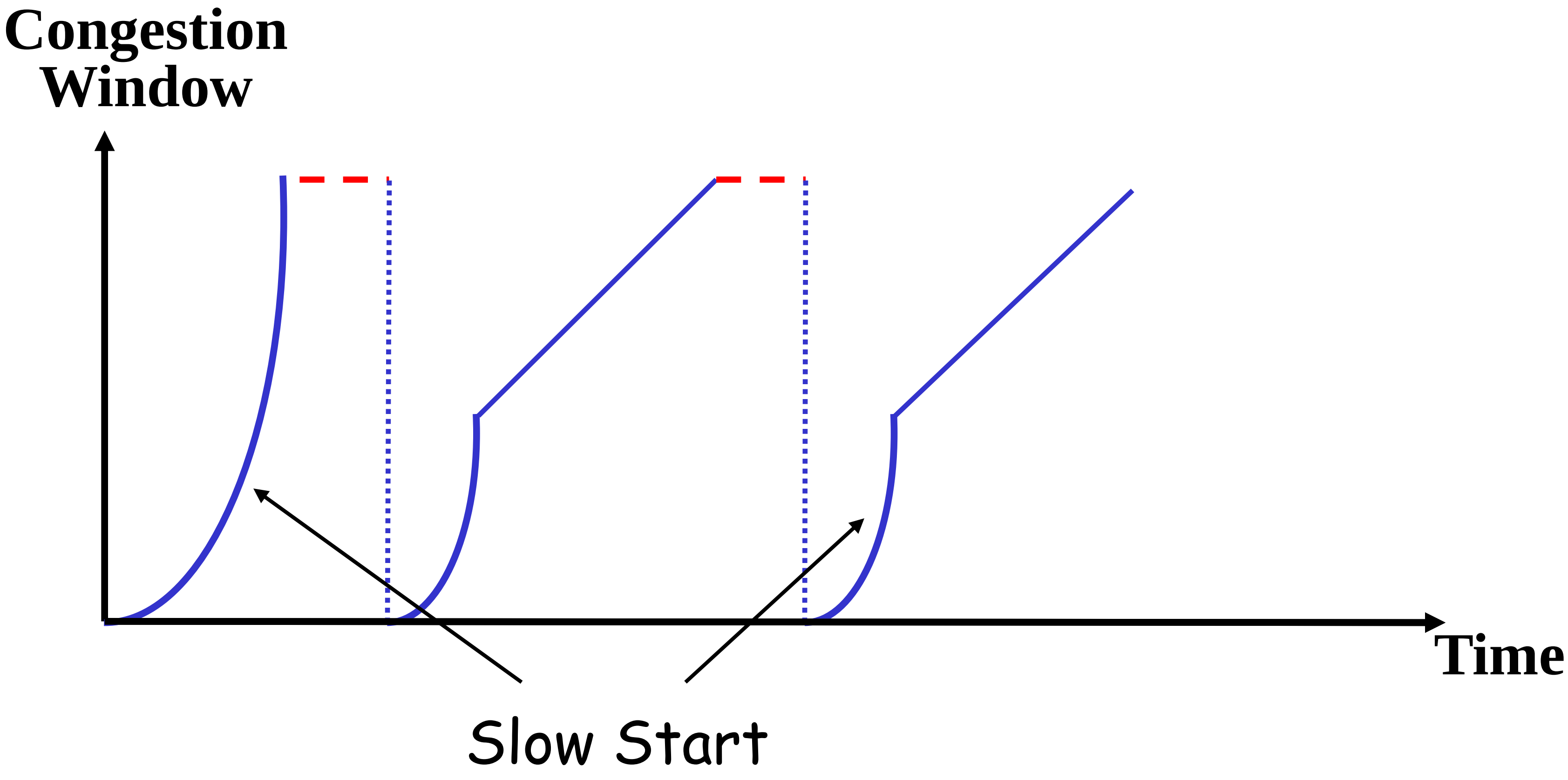
Fast Recovery

- If there are still ACKs coming in then no need for slow-start
- Divide cwnd by 2 after fast retransmit
- Increment cwnd by 1/cwnd for each dupack

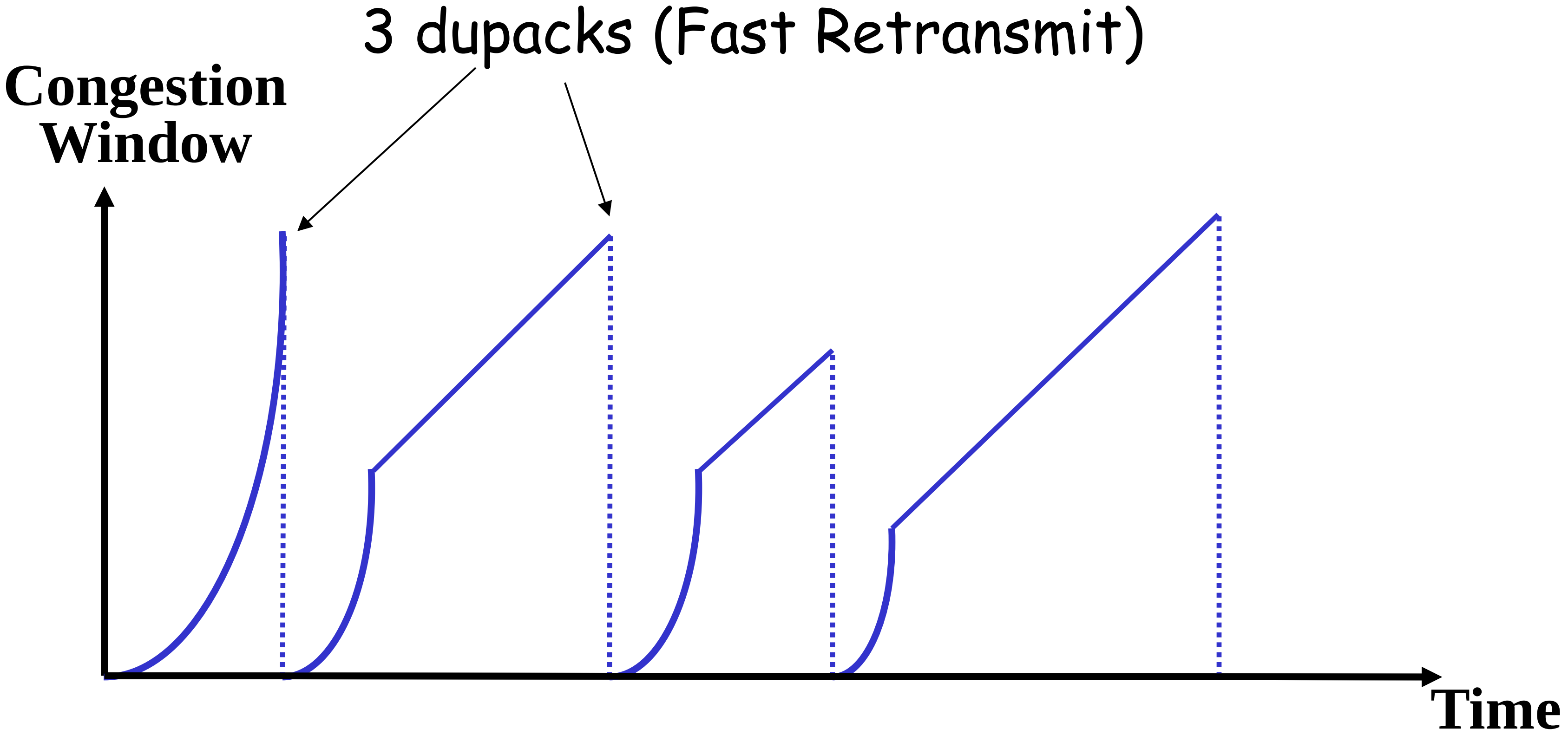
Putting It Together



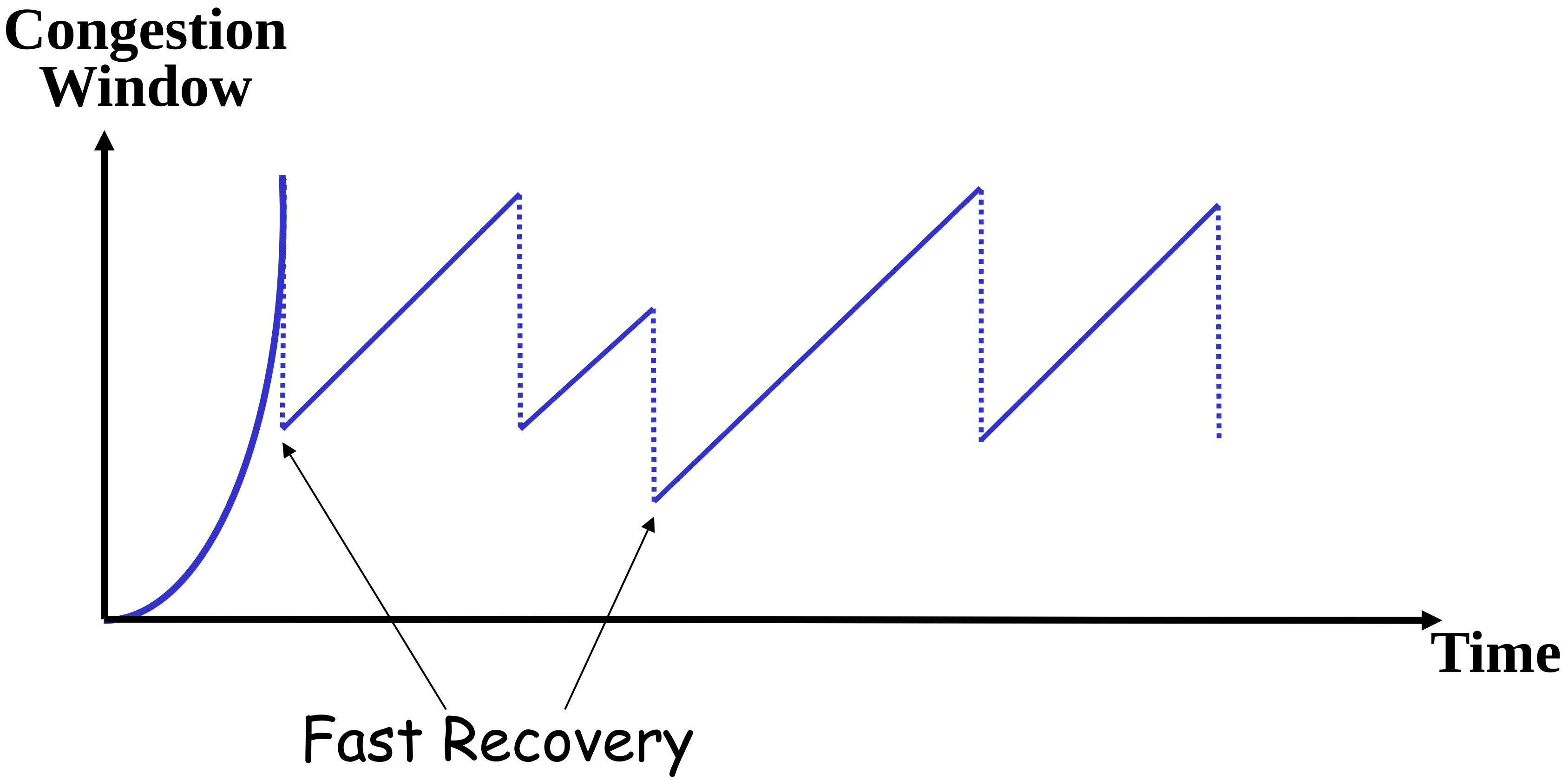
Putting It Together



Putting It Together



Putting It Together



TCP Steady-State Throughput as Function of Loss Rate

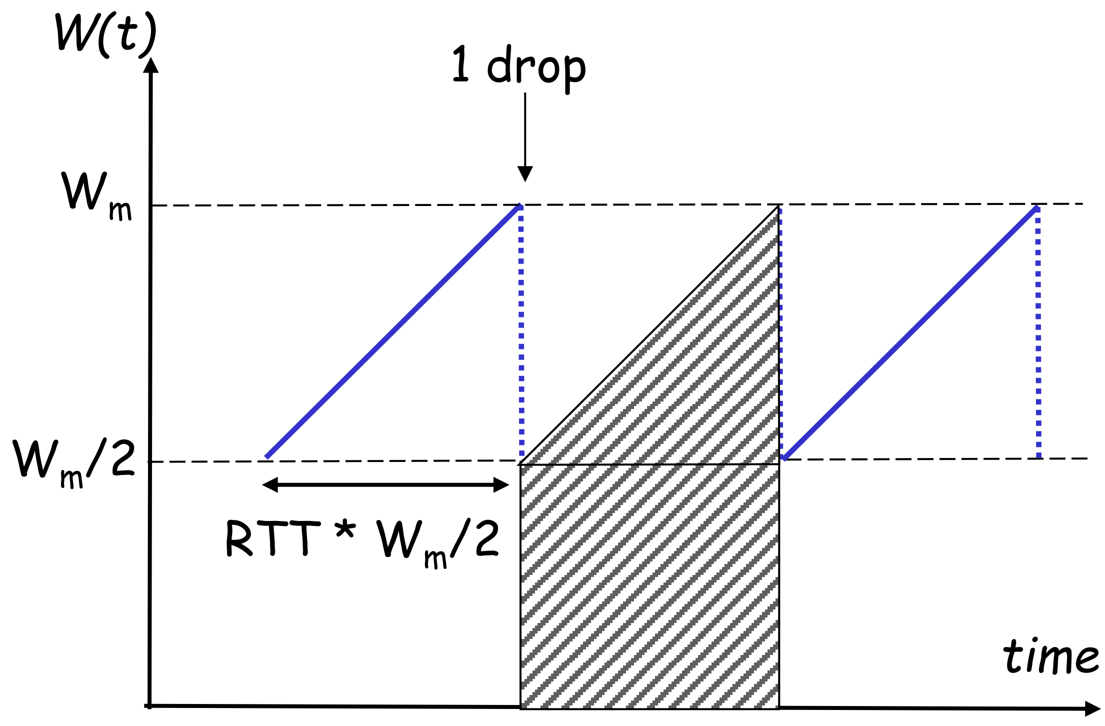
1 drop every $\frac{1}{2} \times \left(\frac{W_m}{2}\right)^2 + \left(\frac{W_m}{2}\right)^2 pkt$ so, drop rate is: $p = \frac{8}{3W_m^2}$

Throughput λ is the packets sent divided by the time it took to send them

$$\lambda = \frac{3W_m}{4RTT}$$

From the two eq.

$$\lambda \approx \frac{\sqrt{3/2}}{RTT\sqrt{p}}$$



Reflections on TCP

- The probing mechanism of TCP is based on causing congestion then detecting it
- Assumes that all sources cooperate
- Assumes flows are long enough
- Too bursty
- Vulnerable to non-congestion related loss (e.g. wireless errors)
- Unfair to long RTT flows