End-to-End Congestion Control TCP

Dina Katabi

nms.csail.mit.edu/~dina

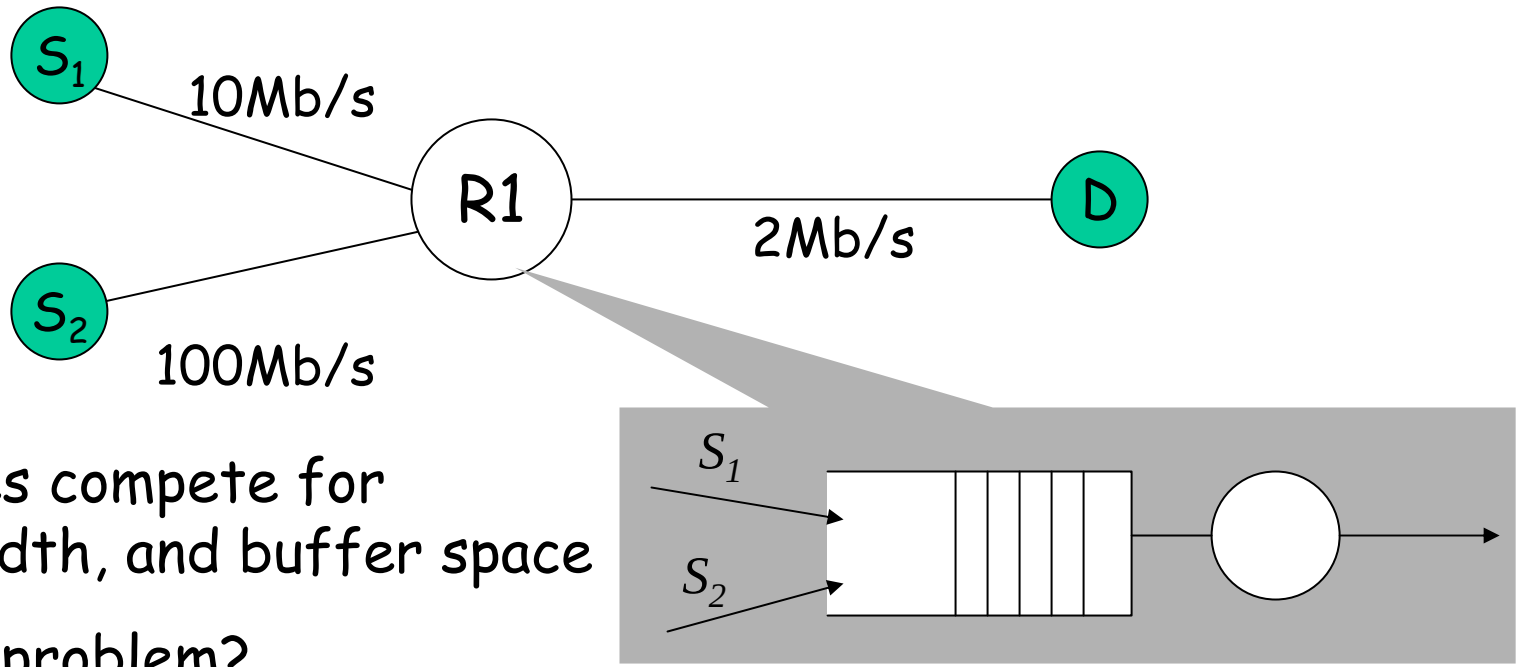
Sharing the Internet

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

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)

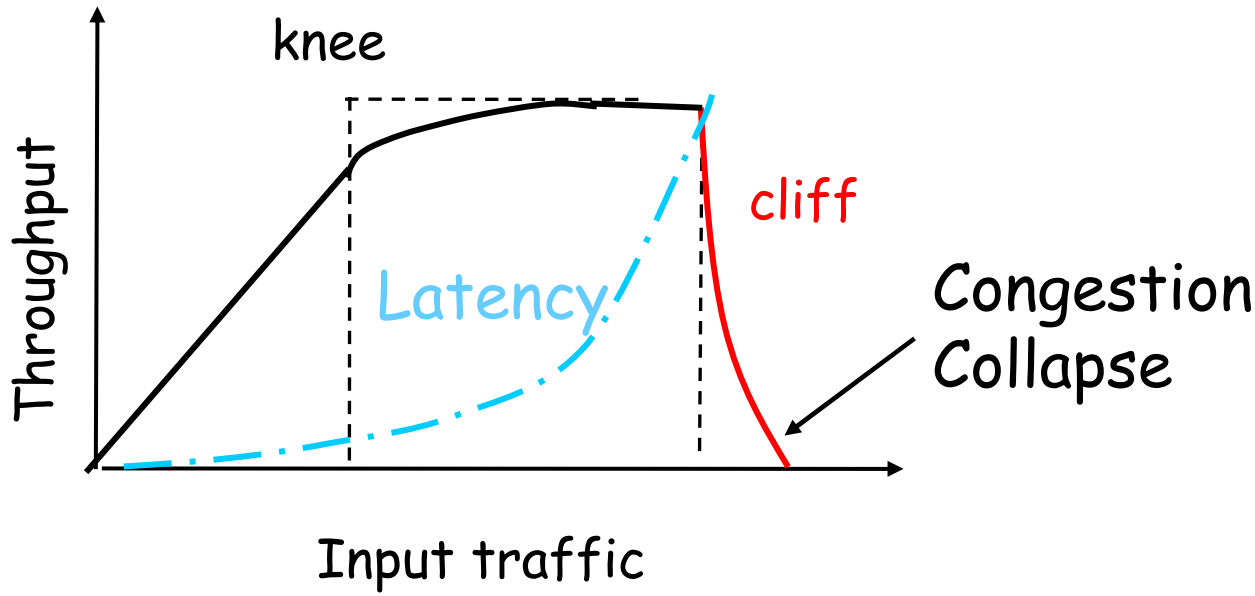
Congestion



- Sources compete for 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)

Congestion Collapse

Increase in input traffic leads to decrease in useful work



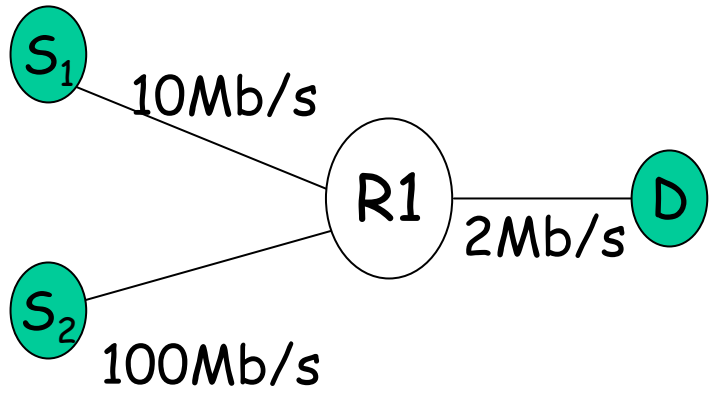
- Many possible causes
  - Spurious retransmissions of packets still in flight
  - Packet consume resources and then they are dropped downstream
  - Control Traffic

What can be done?

- Increase network resources
  - But demands will increase too!
- Admission Control
  - Used in telephone networks
  - Hard in the Internet because can't model traffic well
- Congestion control: ask the sources to slow down
  - Plausible because Internet applications are elastic
  - But how?
    - How do the sources learn of congestion?
    - What is the correct sending rate?
    - What's the role of the sender and what is the role of the router?

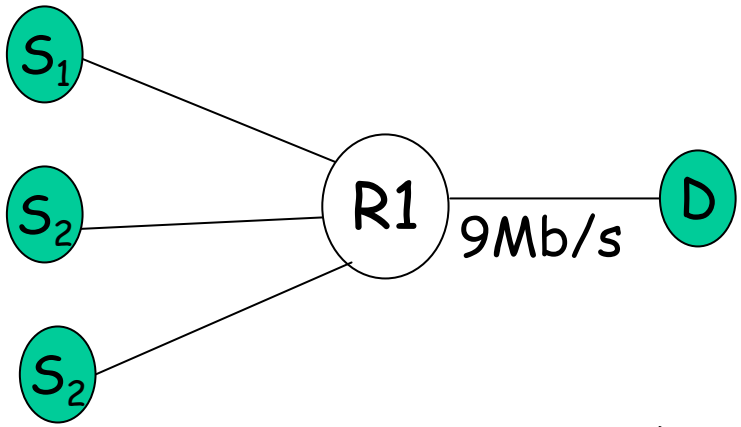
Objectives of Cong. Cont.

- 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
- ρ_1= 1Mb/s
- ρ_2= 7Mb/s
- ρ_3= ∞

- Other measures of fairness
- Want controlled unfairness

Max-min Fair Rates
- μ_1= 1Mb/s
- μ_2= 4Mb/s
- μ_3= 4Mb/s

2 Approaches to Congestion Control

- End-system congestion control
  - Router is dumb
  - Sender detects cong.
  - Sender reacts
  - E.g. TCP
- Network-centric cong. cont.
  - Router help in detecting cong and deciding what to do
  - E.g., XCP

Important Principles

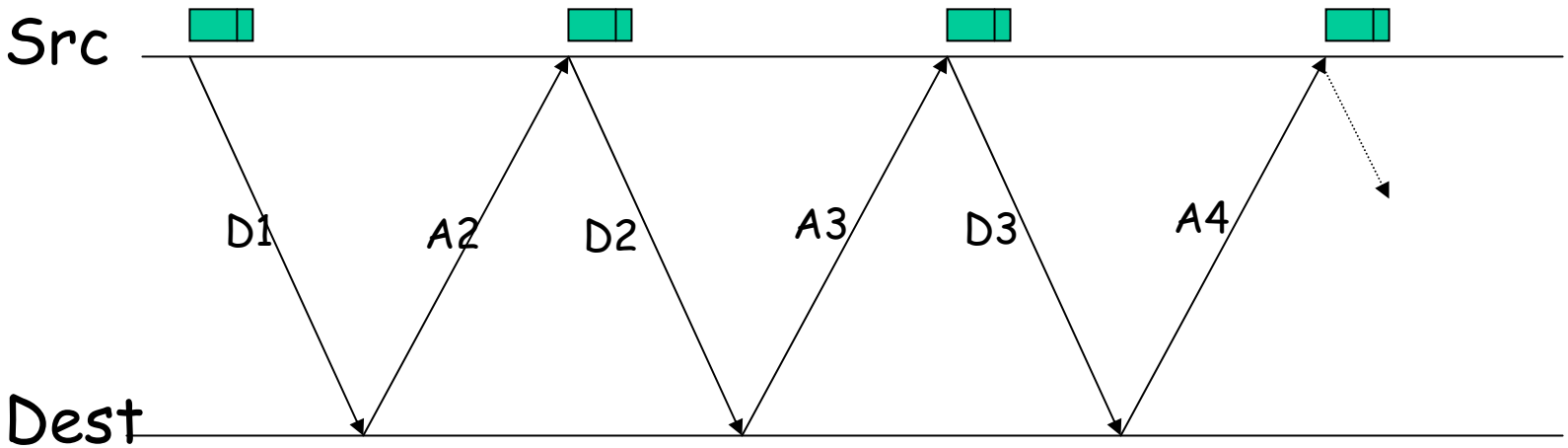
- Distributed
- Stable & Robust
- Keep routers simple

TCP

TCP

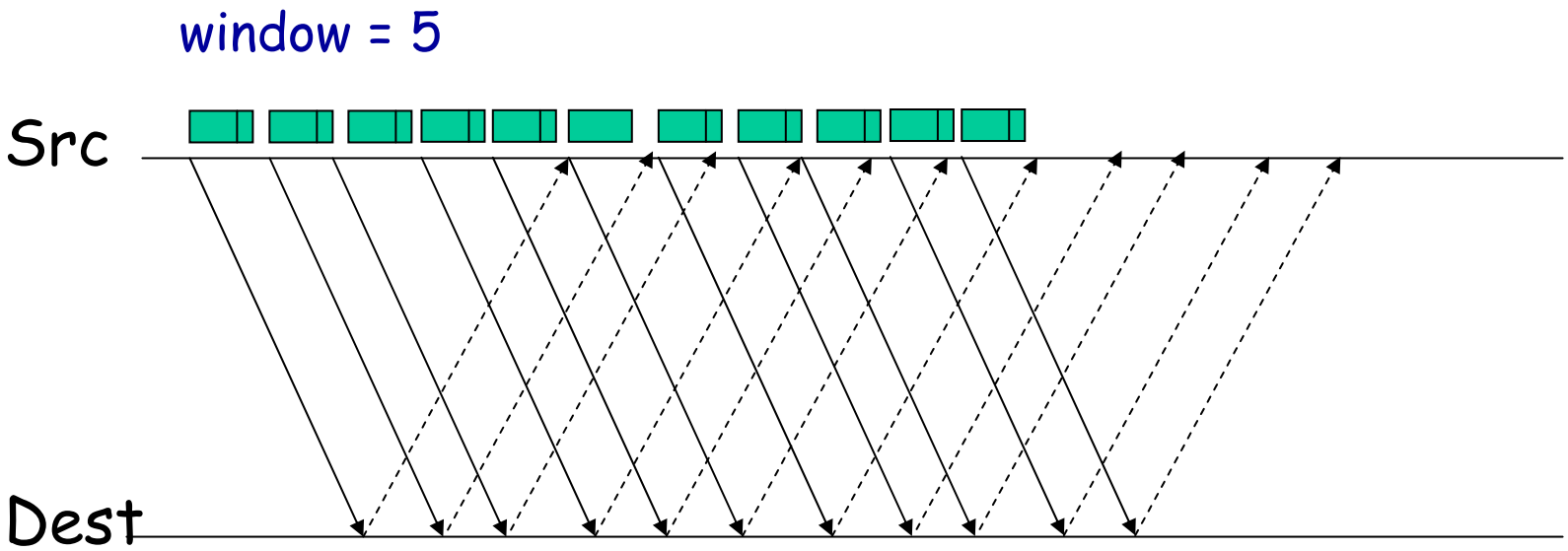
- TCP provides reliability & congestion control
- Reliable transmission ensures the receiver application receives the correct and complete data sent by the sender application
  - TCP recovers from errors and lost packets
- Congestion control
  - sender reacts to congestion and discovers the fair and efficient send rate

TCP Provides Reliability

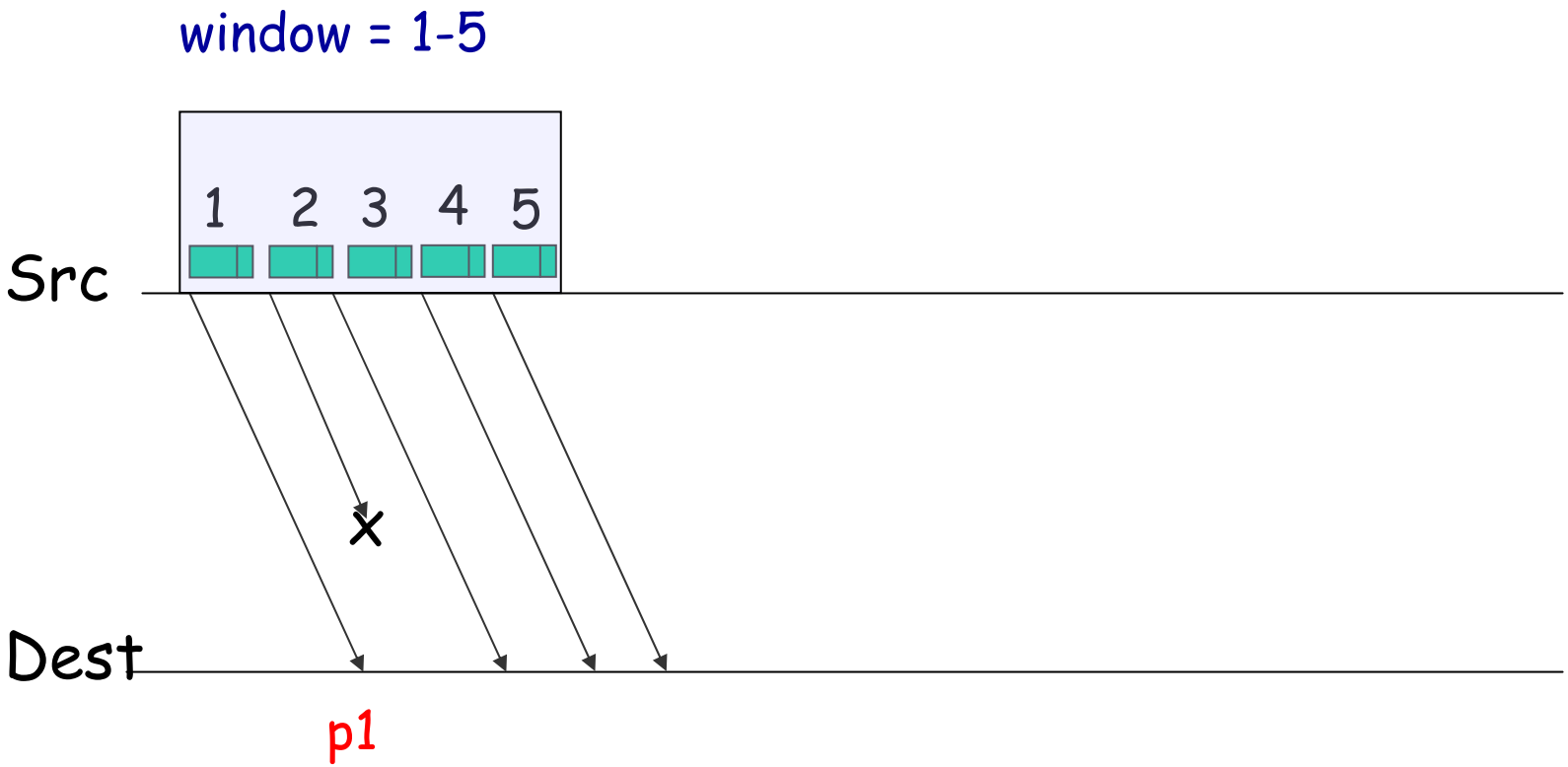


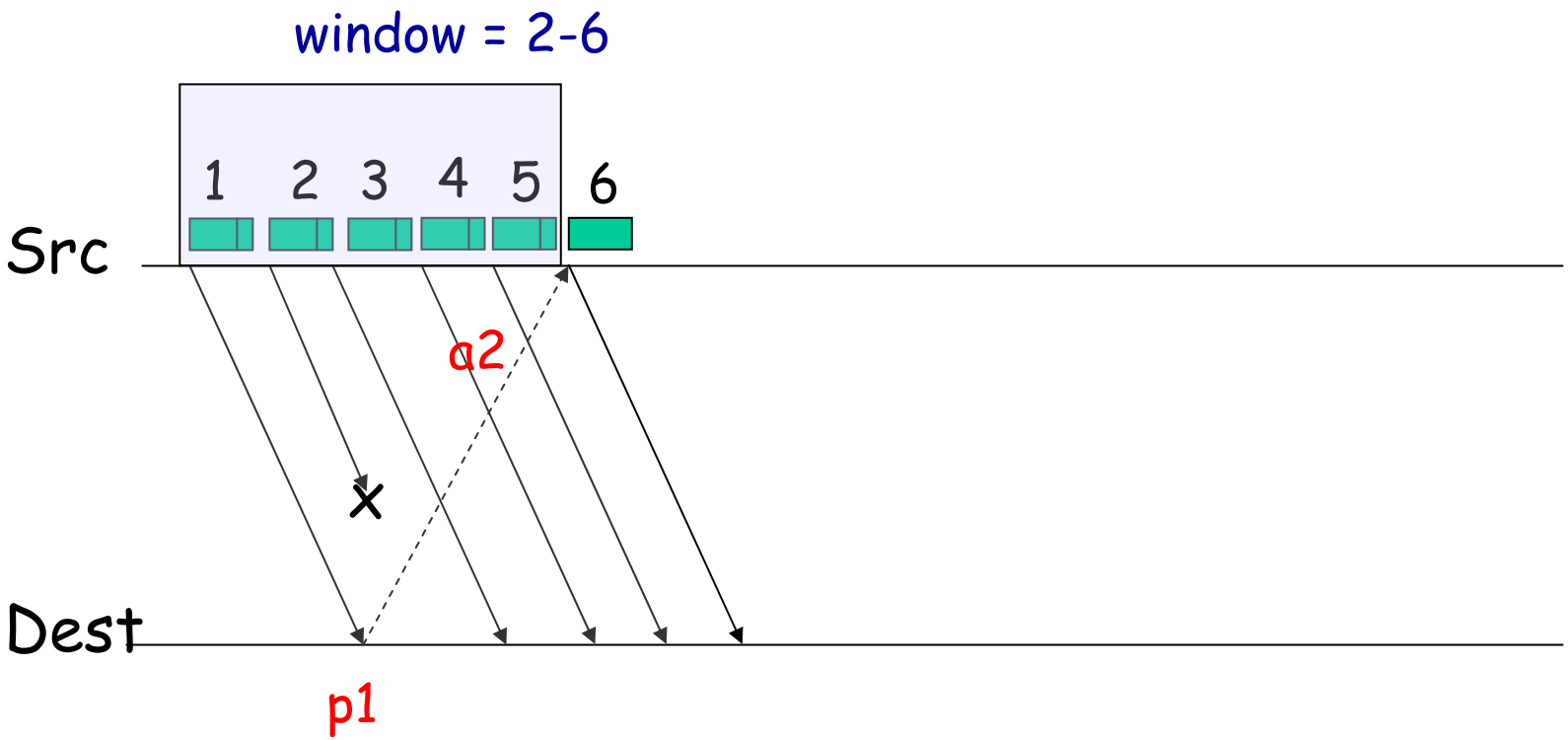
- Packets have sequence numbers
- Receiver acks the next sequence number it expects to receive

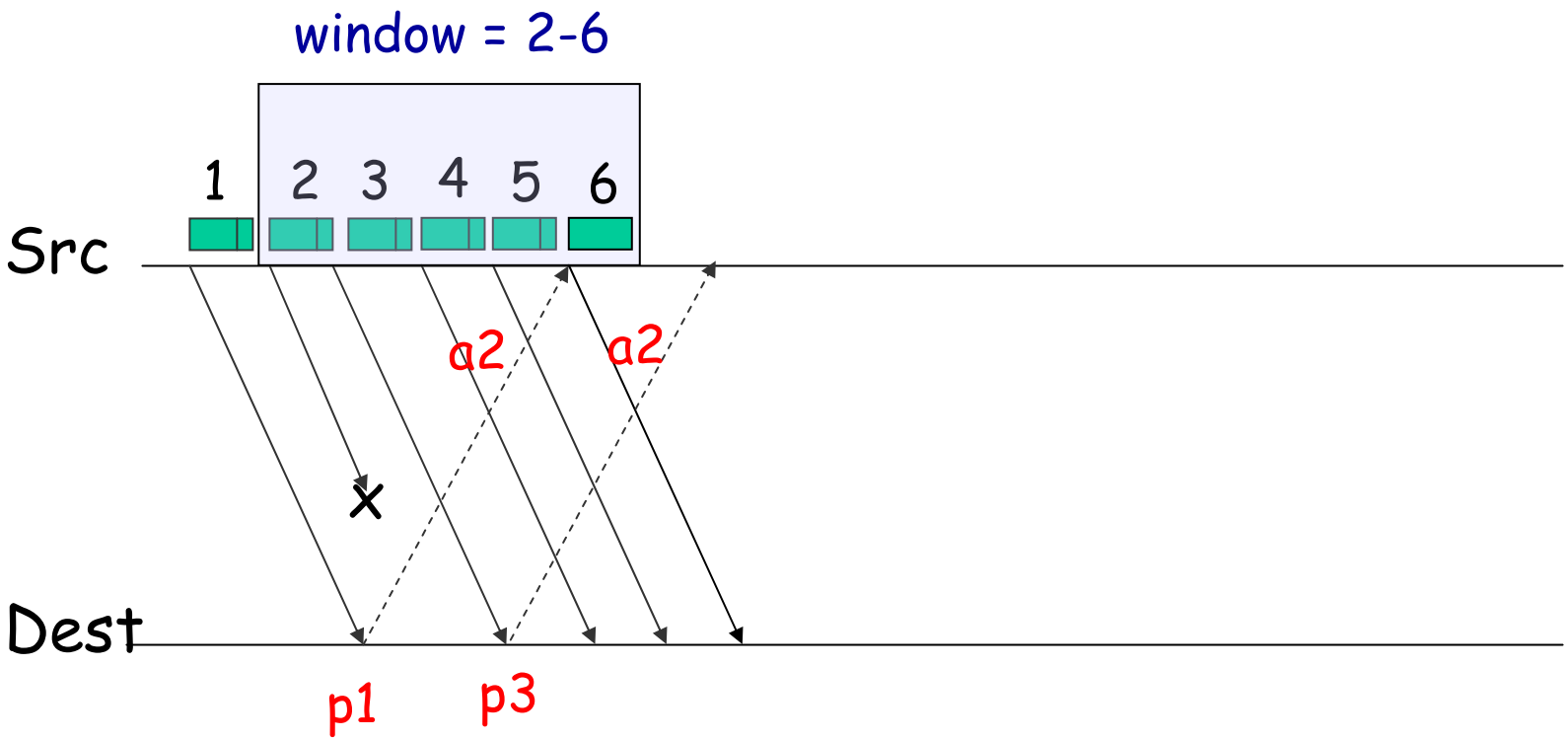
TCP Provides Reliability

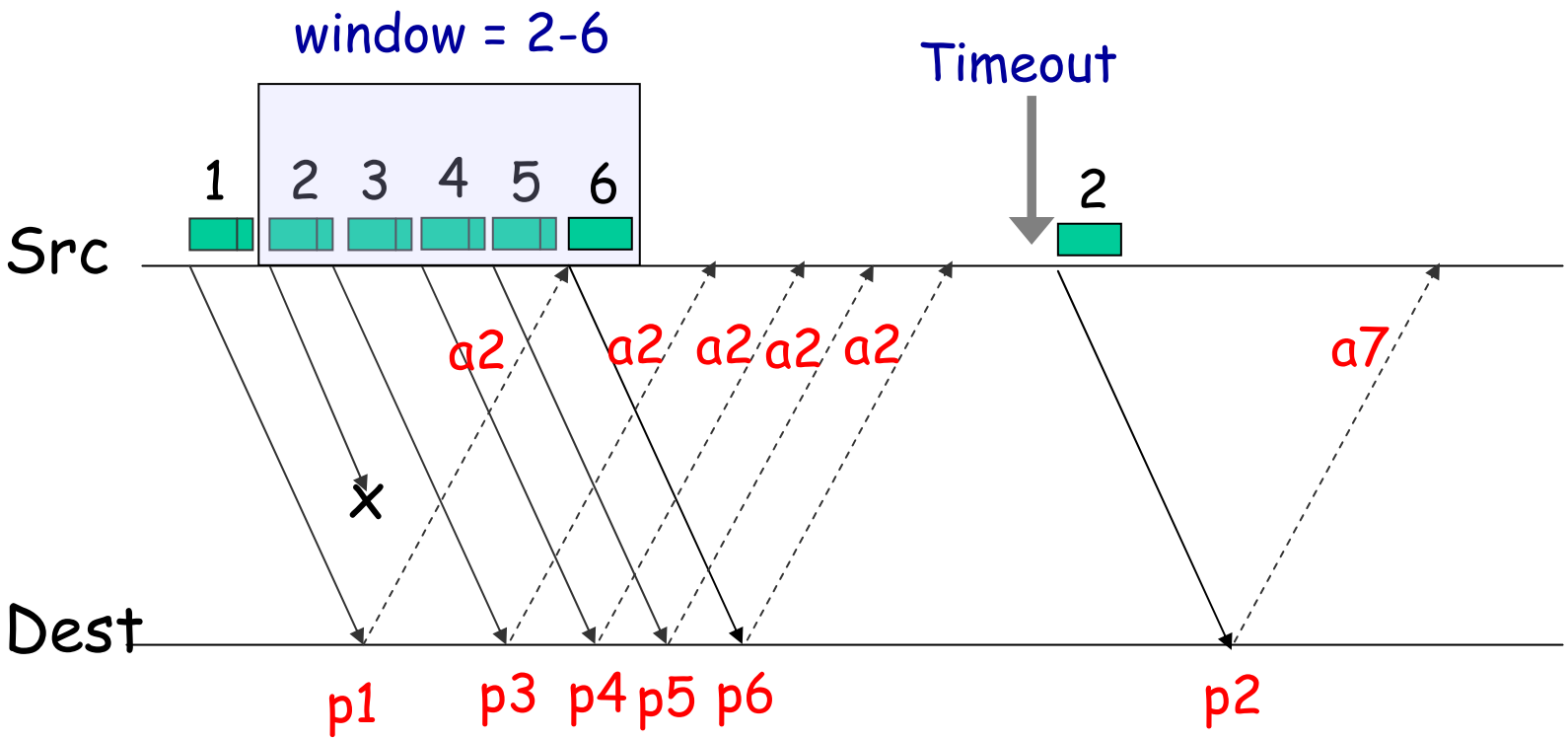


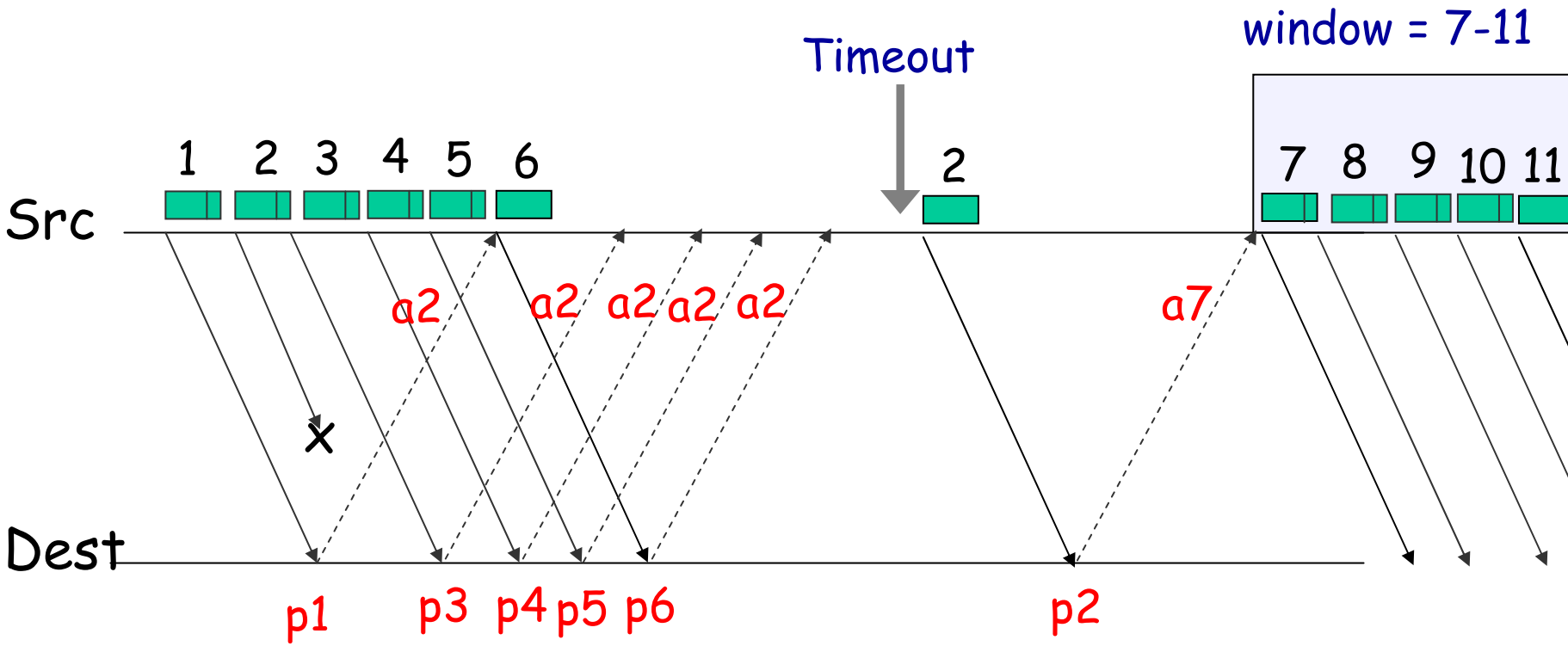
- Sender sends a window of packets (has a window of outstanding pkts at any time)
- Packets have sequence numbers → a loss creates a hole in the received seq. number
- Receiver uses cumulative acks, i.e., it acks the next seq. number it expects to receive
  - If ack K then received all packet with sequence number i< k
- Sender advances the window beyond packets that were acked
- Sender times out
- Sender retransmits dropped packets


window = 1-5
1
2
3
4
5
Src
X
Dest
p1


window = 2-6
1
2
3
4
5
6
Src
a2
X
Dest
p1


window = 2-6
1
2
3
4
5
6
Src
Dest
a2
a2
X
p1
p3


window = 2-6
Timeout
1
2
3
4
5
6
2
Src
a2
a2
a2
a2
a2
a7
X
Dest
p1
p3
p4
p5
p6
p2



Timeout-based recovery is slow

New versions of TCPs use Fast Retransmit

End-system Cong. Cont.

- TCP uses a host-based approach
  - Routers are DropTail and FIFO
  - End-systems detect congestion
  - End-systems react to congestion
- Idea:
  - Send a few packets. If a packet is dropped decrease rate. If no drops, increase rate
    - How does TCP detect drops?

TCP controls throughput via the congestion window

- Congestion window is the number of packets sender can send without an acknowledgement
- TCP is window-based: sources change their sending rate by modifying the window size: "cwnd"
  - Throughput = cwnd /RTT
- Why not changing rate directly?
  - Window provides conservation of packets at equilibrium
  - Window protocols are easy to implement
  - But, they are BURSTY

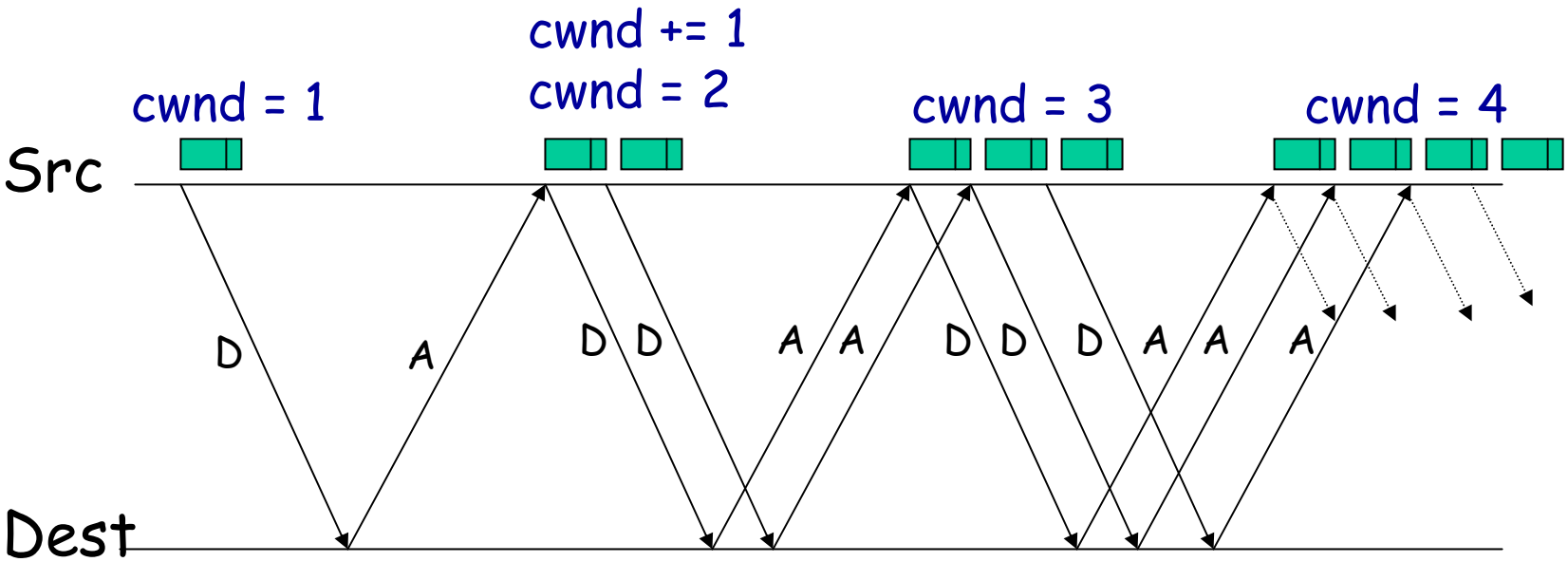
How much should TCP Increase/decrease?

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

    No loss: cwnd = cwnd + 1

    A loss: cwnd = cwnd /2

Additive Increase

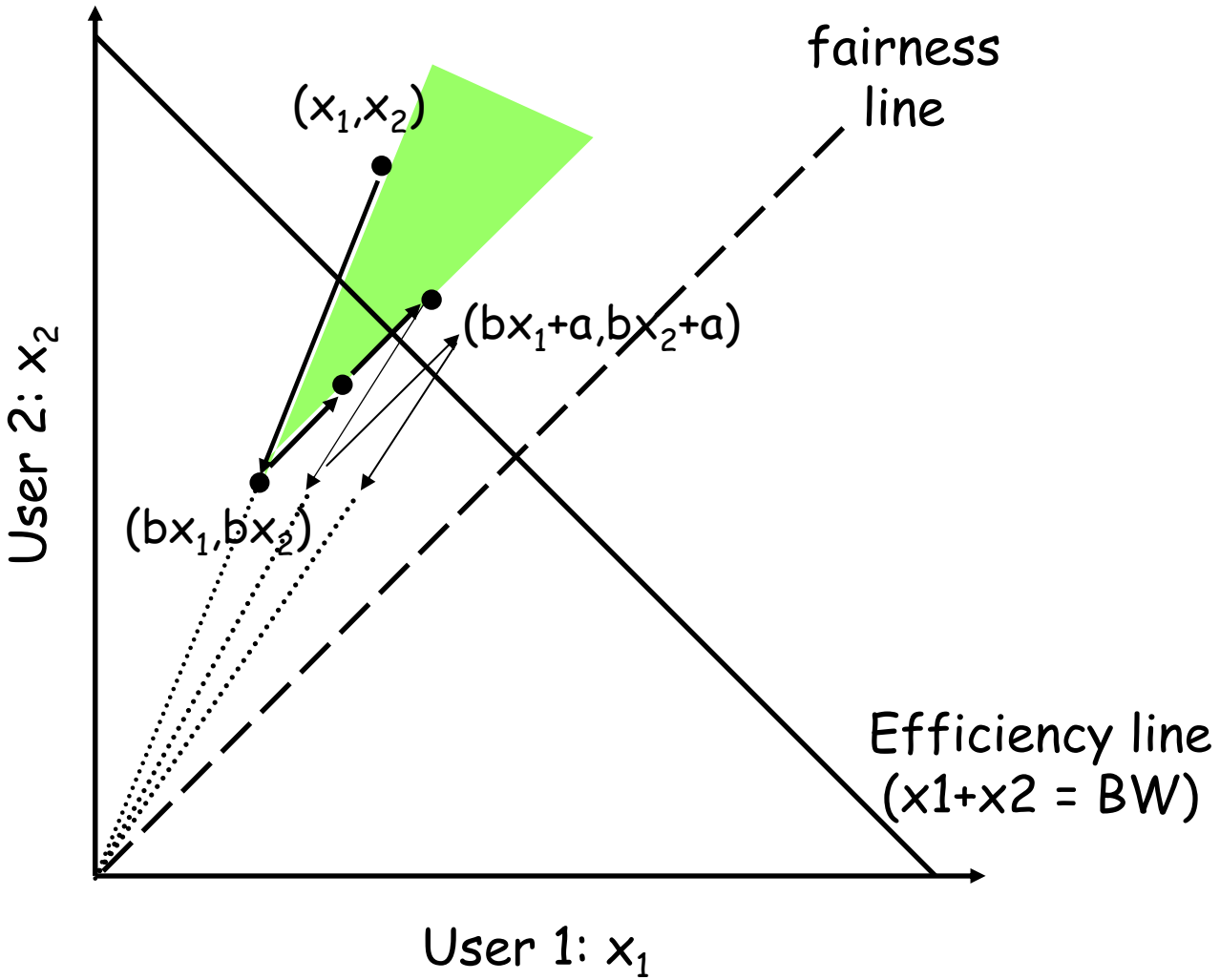


Actually,

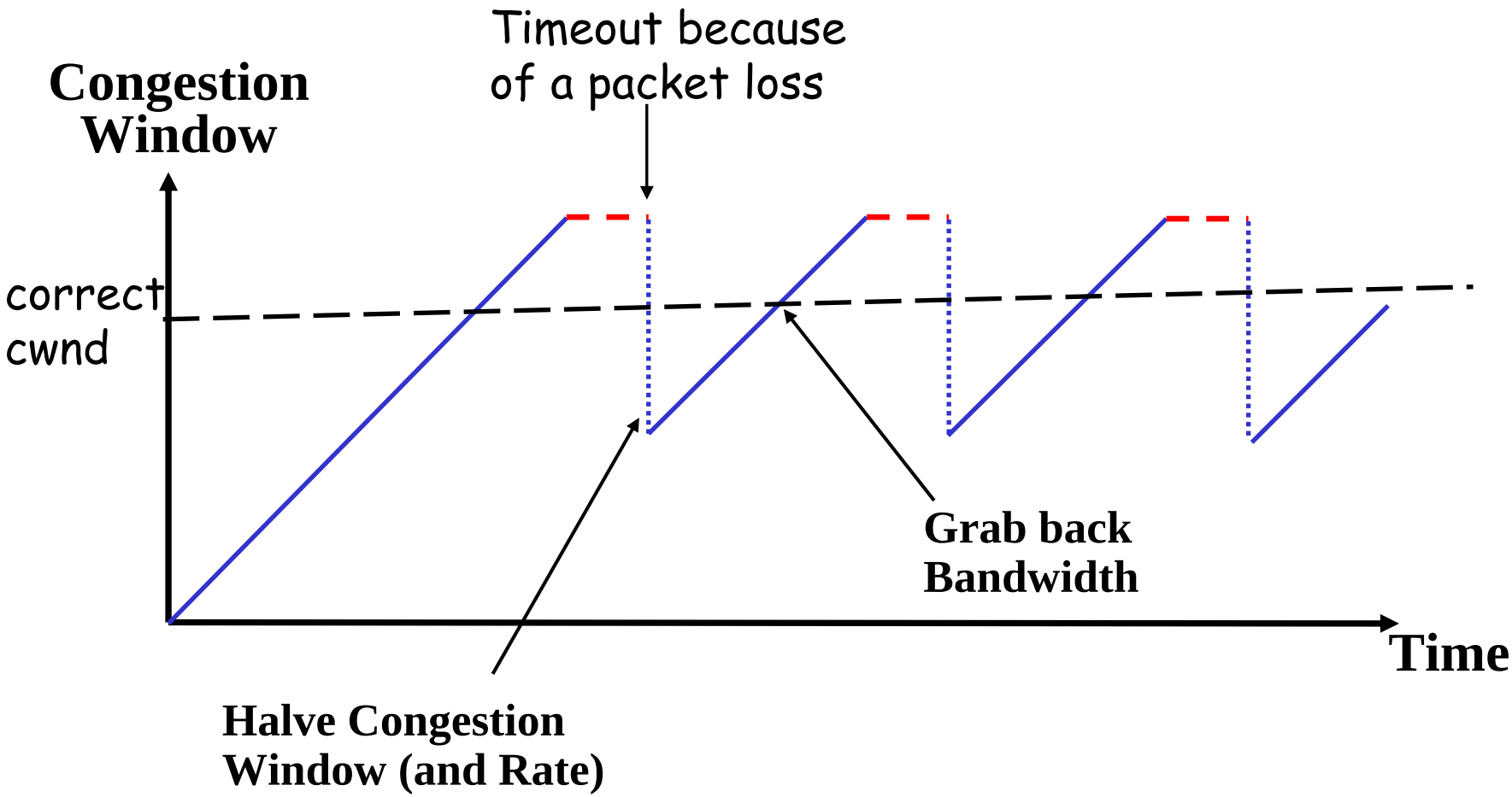
On ack arrival: cwnd = cwnd + 1/cwnd

On timeout: cwnd = cwnd /2

AIMD Leads to Efficiency and Fairness



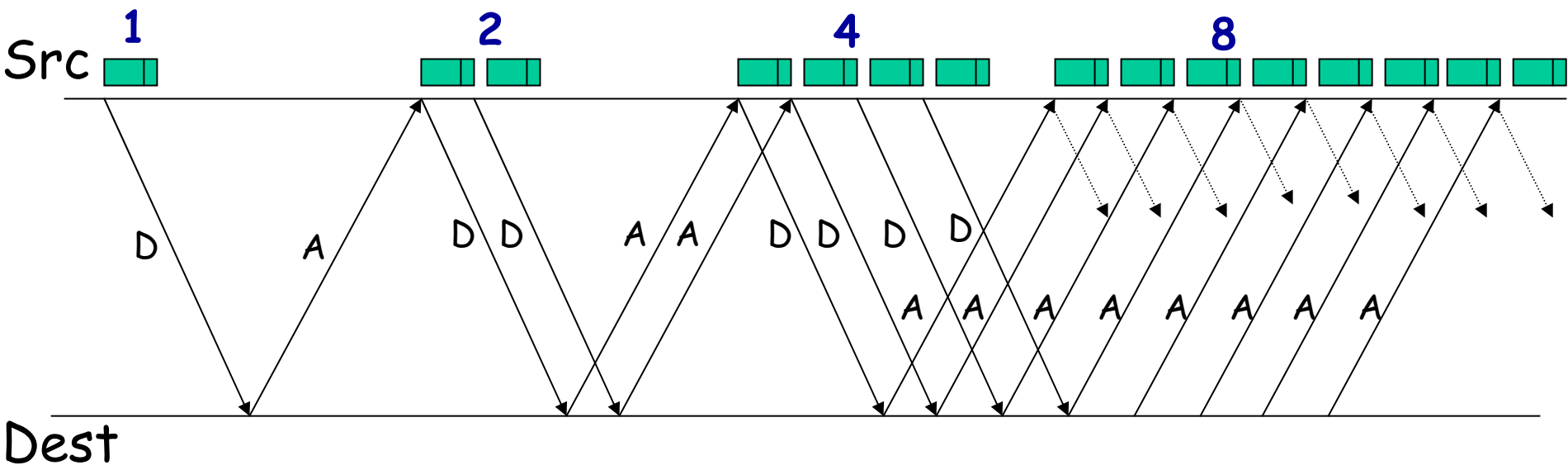
TCP AIMD



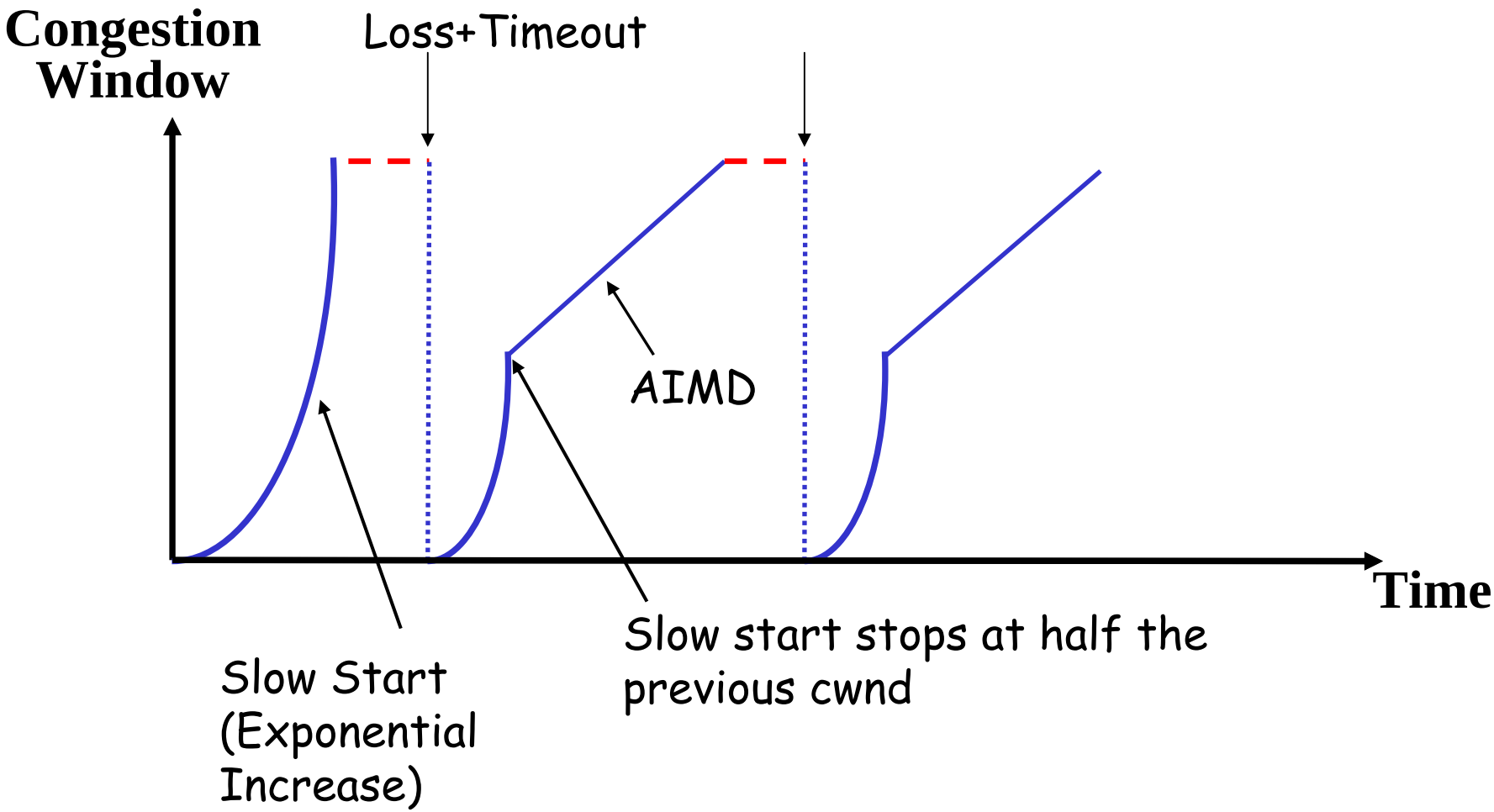
Need the queue to absorb these saw-tooth oscillations

"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

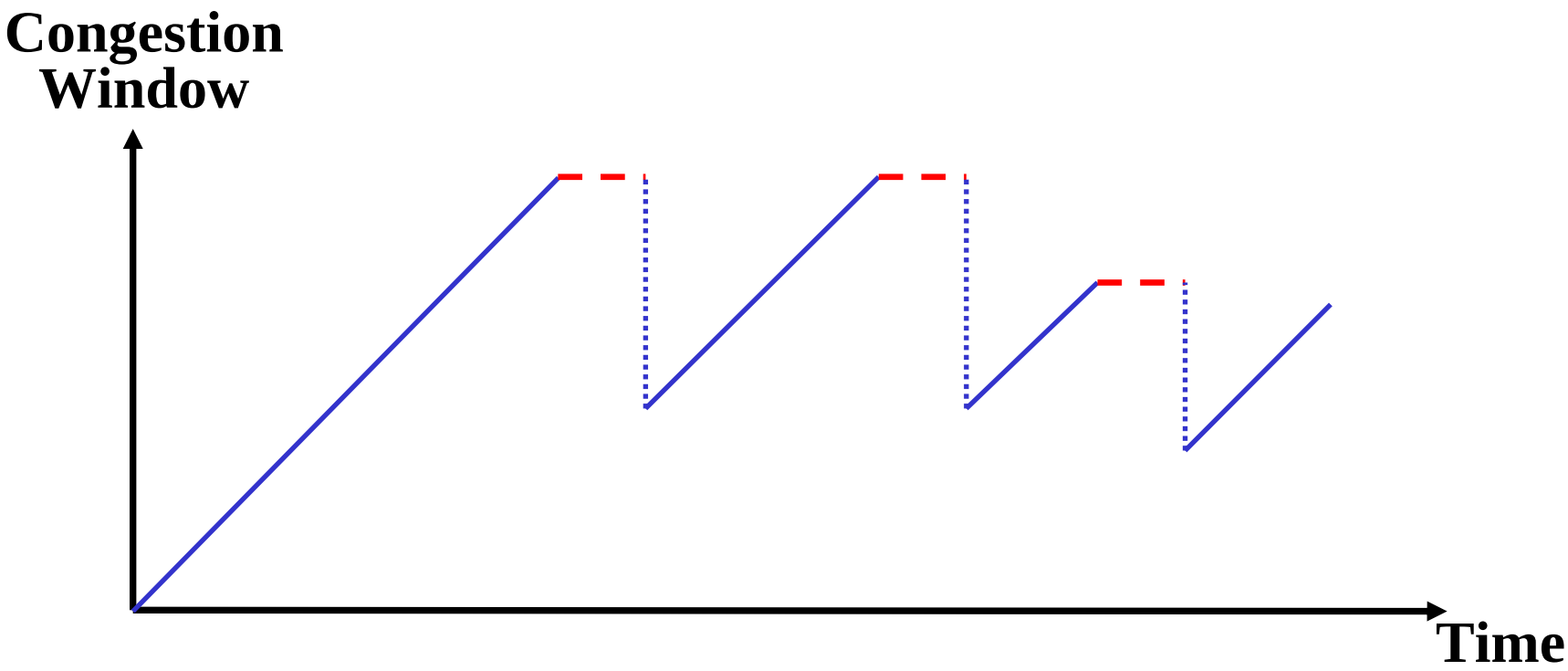
Fast Retransmit

- Timeouts are too slow
- When packet is dropped, receiver still acks the last in-order packet
- Sender interprets 3 duplicate ACKs as a sign of drop
  - Why 3? When this is not useful?

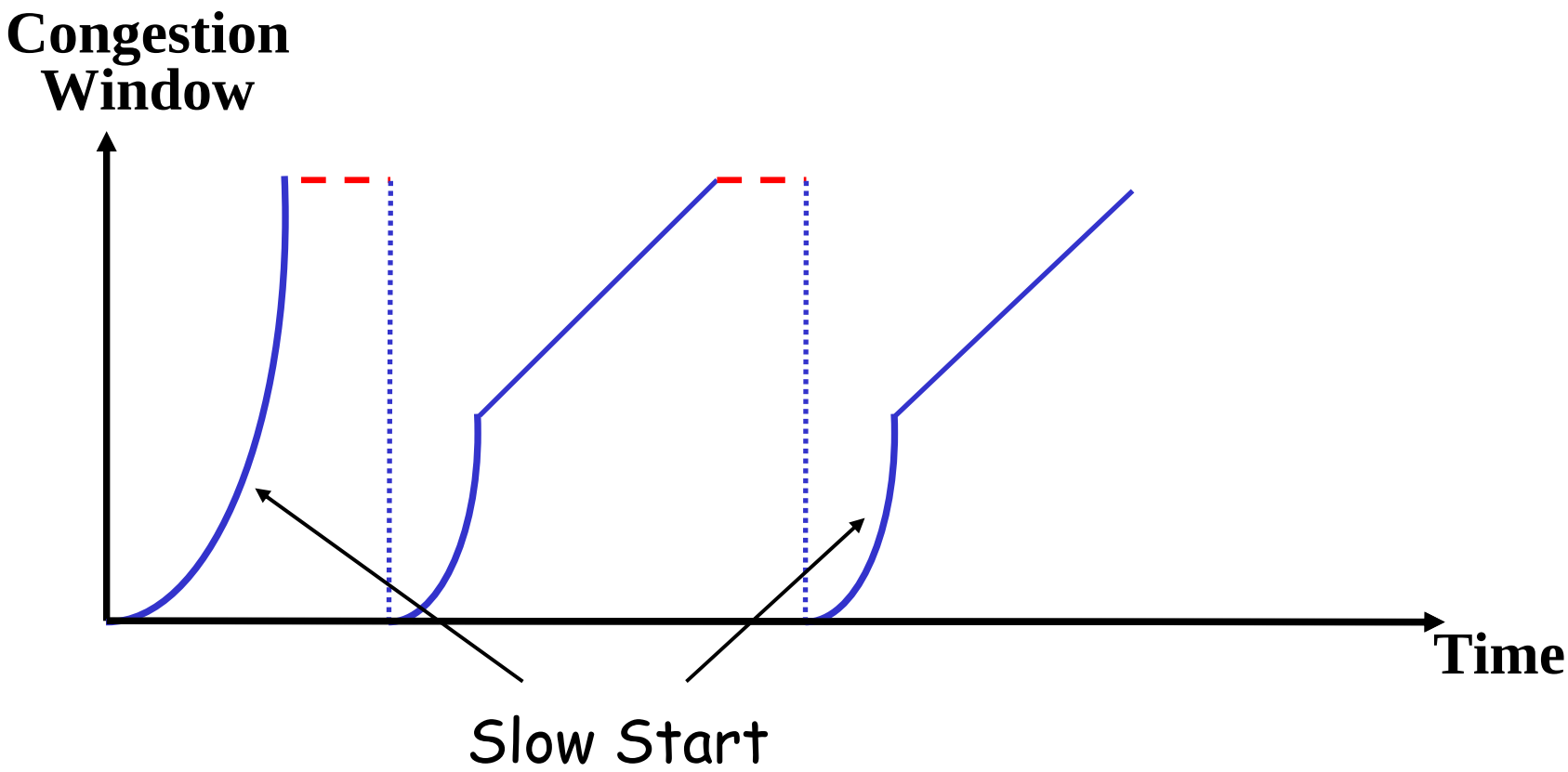
Fast Recovery

- If there are still ACKs coming in, then no to go back to cwnd=1 and do slow-start
- Divide cwnd by 2 after Fast Retransmit
- Increment cwnd by 1/cwnd for each dupack (i.e., do AIMD)

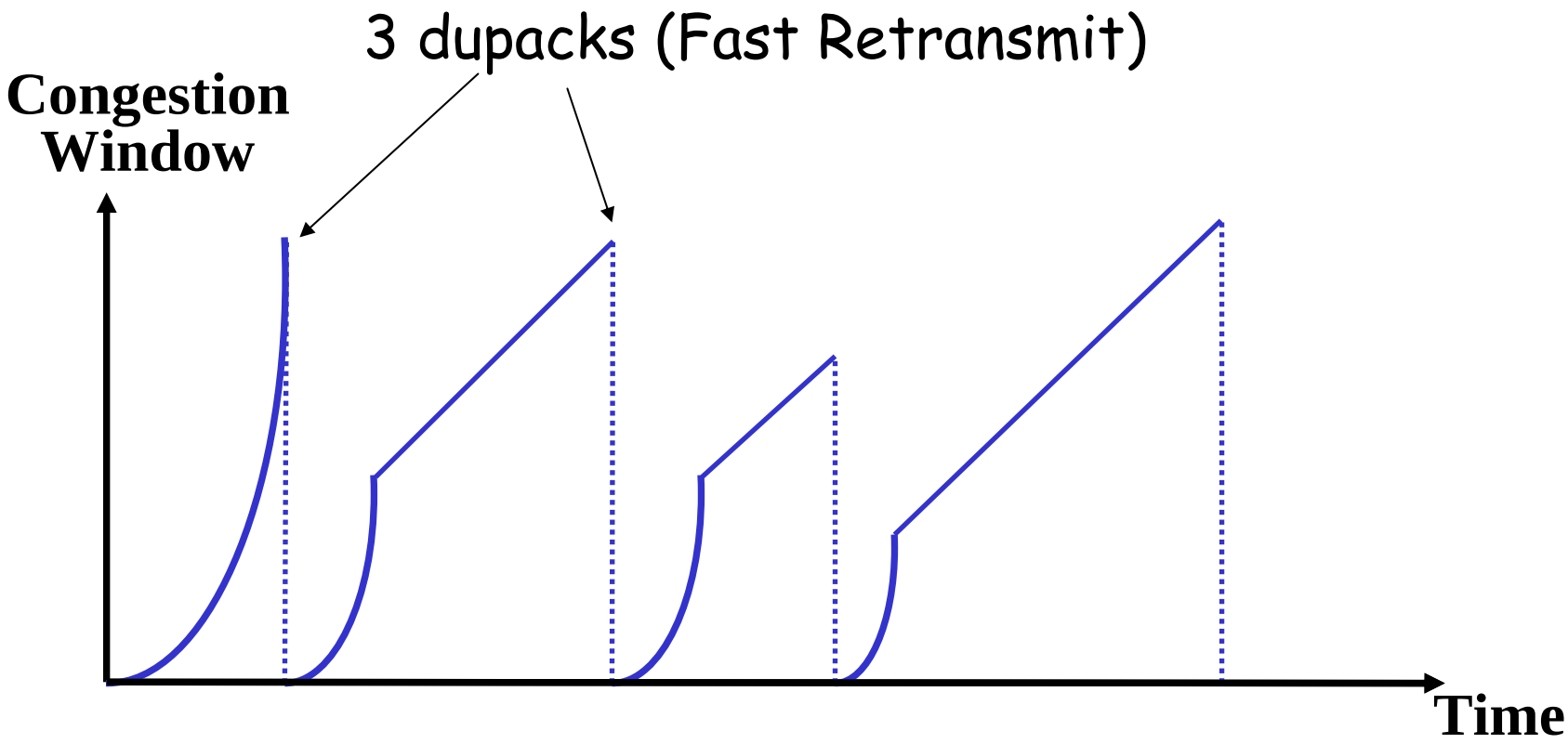
Putting It Together



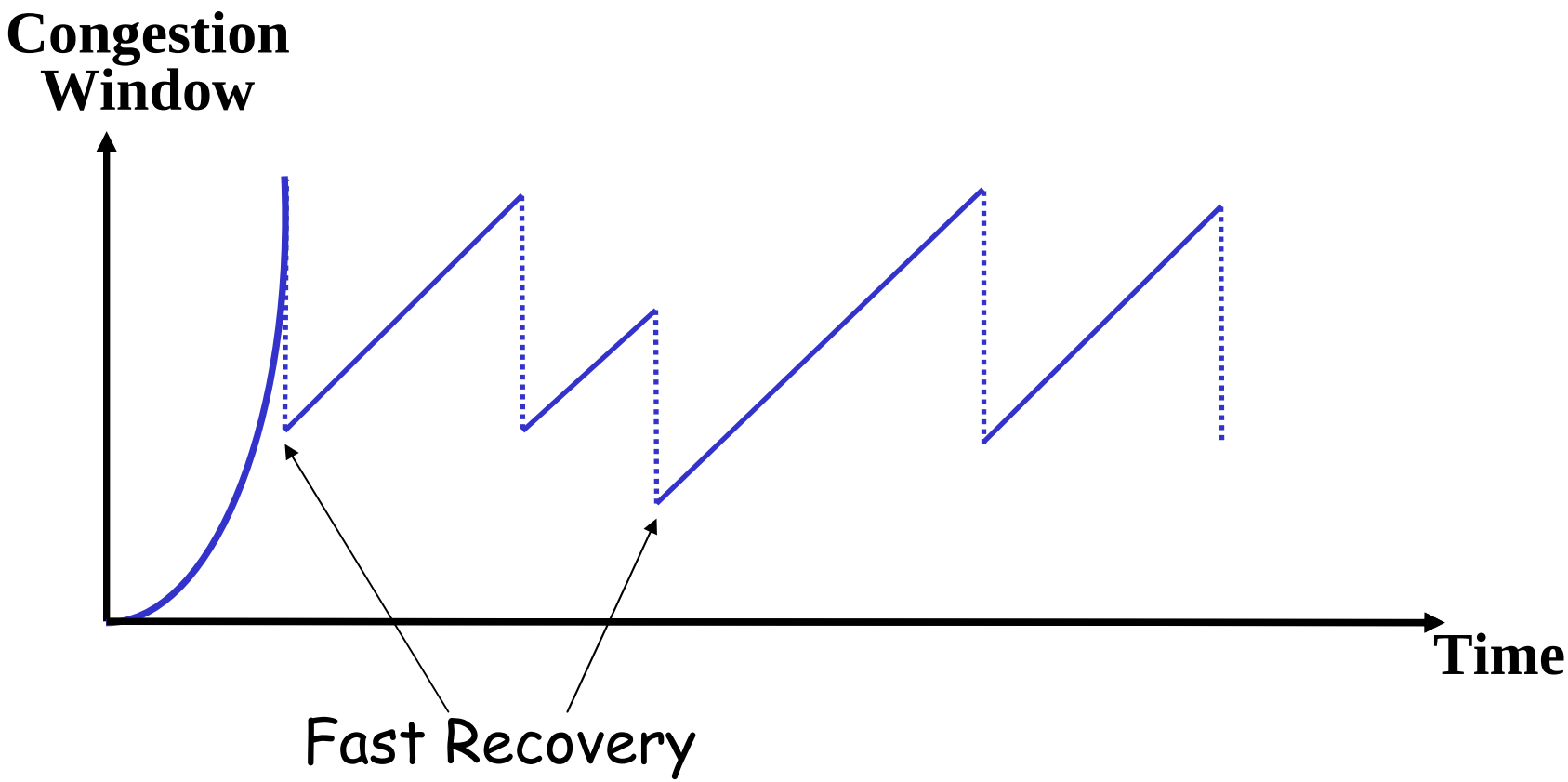
Putting It Together



Putting It Together



Putting It Together



Steady-State Throughput as Function of Loss Rate

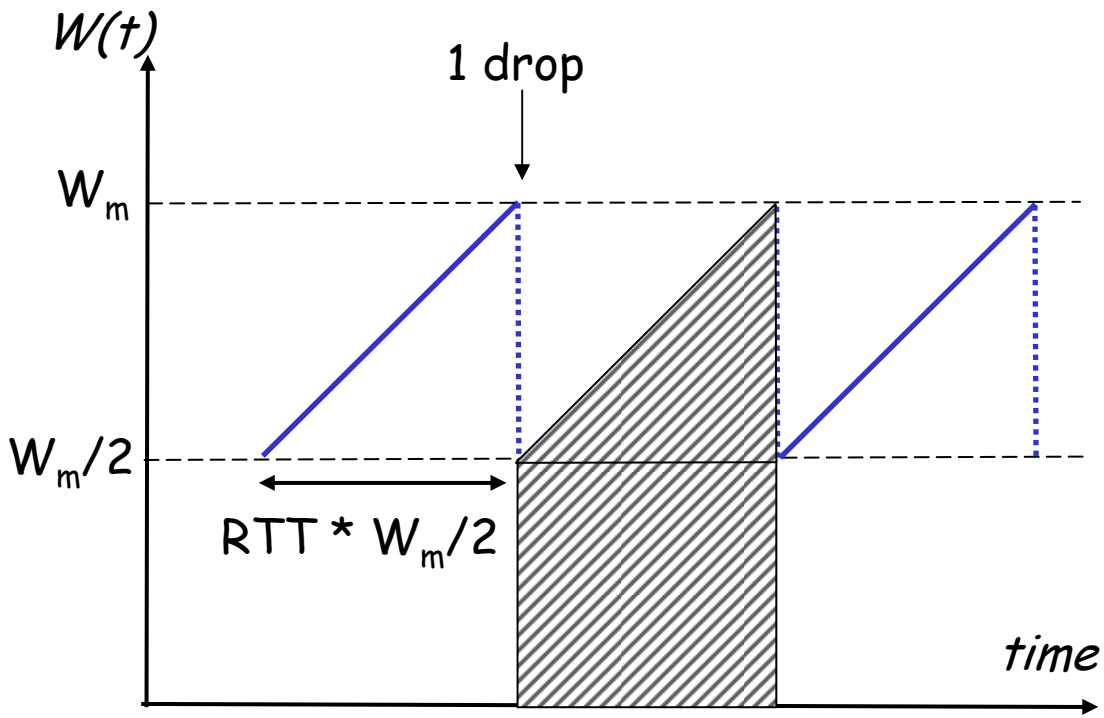
1 drop every $\frac{1}{2}\times\left(\frac{W_m}{2}\right)^2+\left(\frac{W_m}{2}\right)^2 pkt$ so, drop rate is: $p=\frac{8}{3W_m^2}$

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