

# Computer Networks

Jiaqi Zheng

*Material with thanks Mosharaf Chowdhury, and many other colleagues.*

# Agenda

---

- TCP congestion control wrap-up
- TCP throughput equation
- Problems with congestion control

# Recap

---

- Flow Control
  - Restrict window to RWND to make sure that the receiver isn't overwhelmed
- Congestion Control
  - Restrict window to CWND to make sure that the network isn't overwhelmed
- Together
  - Restrict window to  $\min\{\text{RWND}, \text{CWND}\}$  to make sure that neither the receiver nor the network are overwhelmed

# CC Implementation

---

- States at sender
  - **CWND** (initialized to a small constant)
  - **ssthresh** (initialized to a large constant)
  - **dupACKcount** and **timer**
- Events
  - **ACK** (new data)
  - **dupACK** (duplicate ACK for old data)
  - **Timeout**

# Event: ACK (new data)

---

- If  $CWND < ssthresh$ 
  - $CWND += 1$

- *$CWND$  packets per RTT*
- *Hence, after one RTT with no drops:  
 $CWND = 2 \times CWND$*

# Event: ACK (new data)

---

- If  $CWND < ssthresh$

- $CWND += 1$

***Slow start phase***

- Else

- $CWND = CWND + 1/CWND$

***Congestion avoidance phase***

- *CWND packets per RTT*
- *Hence, after one RTT with no drops:*  
 $CWND = CWND + 1$

# Event: TimeOut

---

- On Timeout
  - $\text{ssthresh} \leftarrow \text{CWND}/2$
  - $\text{CWND} \leftarrow 1$

# Event: dupACK

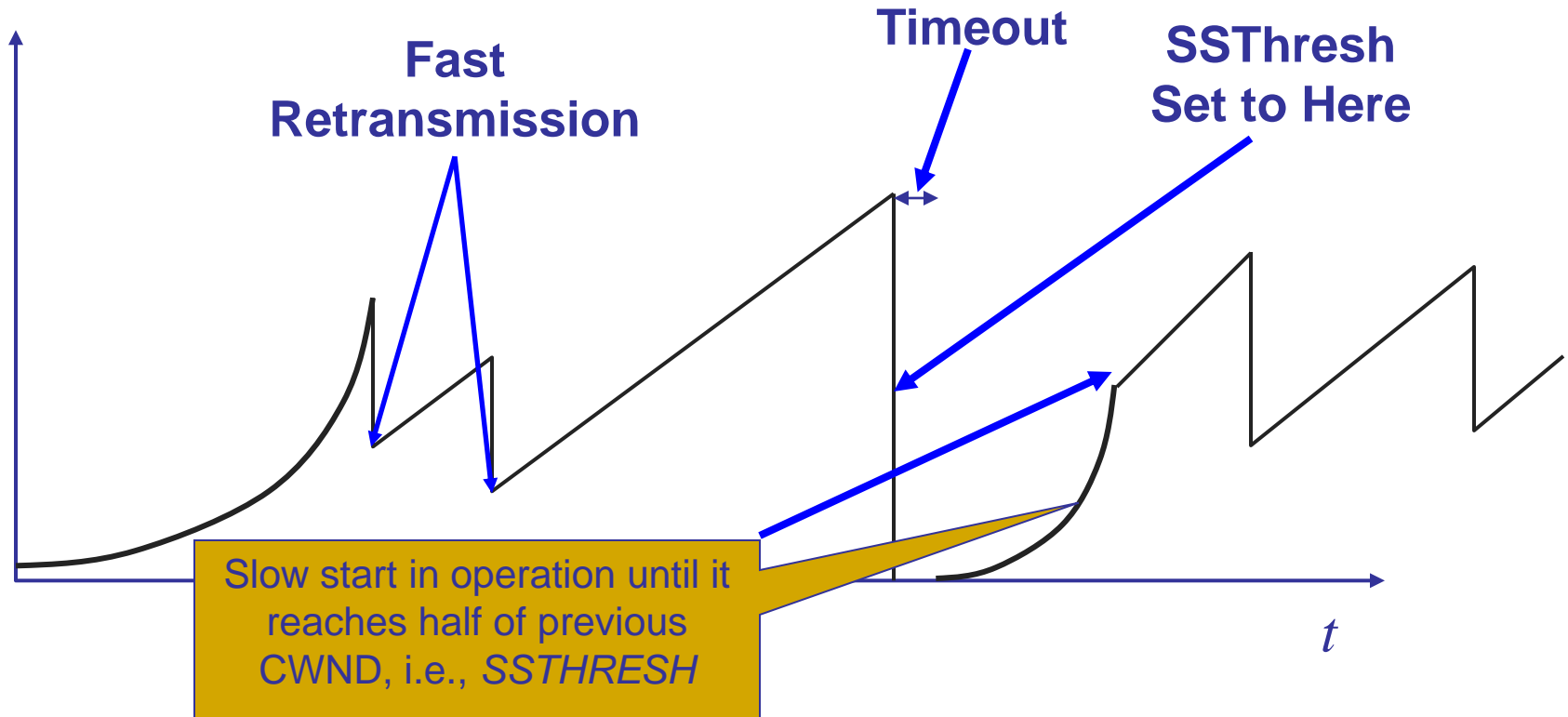
---

- dupACKcount ++
- If dupACKcount = 3 /\* fast retransmit \*/
  - ssthresh = CWND/2
  - CWND = CWND/2



# Example

*Window*



Slow-start restart: Go back to  $\text{CWND} = 1 \text{ MSS}$ , but take advantage of knowing the previous value of  $\text{CWND}$

# Not done yet!

---

- **Problem:** congestion avoidance too slow in recovering from an isolated loss

# Example

---

- Consider a TCP connection with:
  - CWND=10 packets
  - Last ACK was for packet # 101
    - »i.e., receiver expecting next packet to have seq. no. 101
- 10 packets [101, 102, 103,..., 110] are in flight
  - Packet 101 is dropped

# Timeline: [~~101~~, 102, ..., 110]

---

- ACK 101 (due to 102)  $cwnd=10$  dupACK#1 (no xmit)
- ACK 101 (due to 103)  $cwnd=10$  dupACK#2 (no xmit)
- ACK 101 (due to 104)  $cwnd=10$  dupACK#3 (no xmit)
- RETRANSMIT 101  $ssthresh=5$   $cwnd=5$
- ACK 101 (due to 105)  $cwnd=5 + 1/5$  (no xmit)
- ACK 101 (due to 106)  $cwnd=5 + 2/5$  (no xmit)
- ACK 101 (due to 107)  $cwnd=5 + 3/5$  (no xmit)
- ACK 101 (due to 108)  $cwnd=5 + 4/5$  (no xmit)
- ACK 101 (due to 109)  $cwnd=5 + 5/5$  (no xmit)
- ACK 101 (due to 110)  $cwnd=6 + 1/6$  (no xmit)
- ACK 111 (due to 101) ← only now can we transmit new packets
- Plus no packets in flight so ACK “clocking” (to increase CWND) stalls for another RTT

# Solution: Fast recovery

---

- Idea: Grant the sender temporary “credit” for each dupACK so as to keep packets in flight
- If  $\text{dupACKcount} = 3$ 
  - $\text{ssthresh} = \text{CWND}/2$
  - $\text{CWND} = \text{ssthresh} + 3$
- While in fast recovery
  - $\text{CWND} = \text{CWND} + 1$  for each additional dupACK
- Exit fast recovery after receiving new ACK
  - set  $\text{CWND} = \text{ssthresh}$

# Example

---

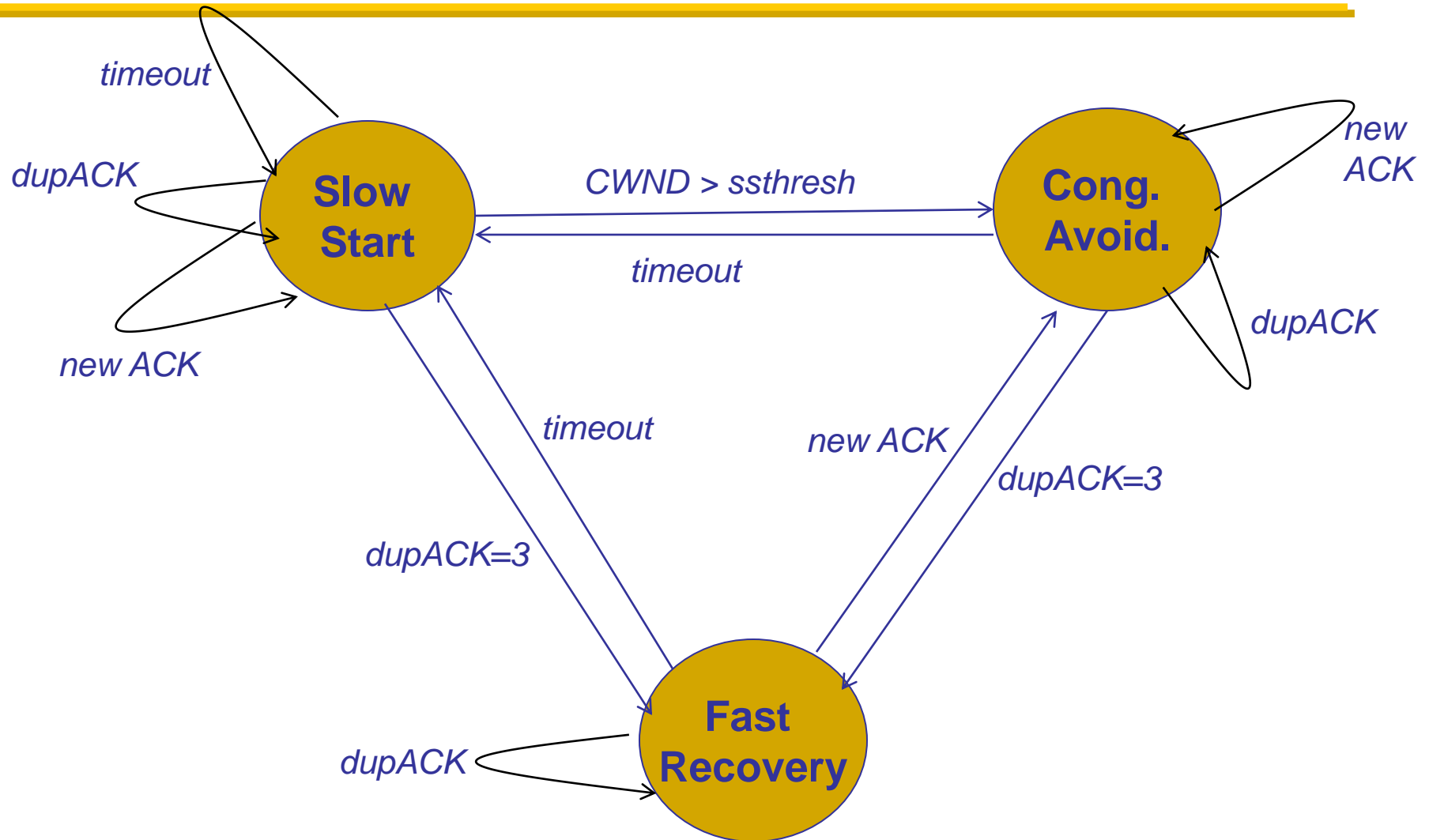
- Consider a TCP connection with:
  - CWND=10 packets
  - Last ACK was for packet # 101
    - »i.e., receiver expecting next packet to have seq. no. 101
- 10 packets [101, 102, 103,..., 110] are in flight
  - Packet 101 is dropped

# Timeline: [~~101~~, 102, ..., 110]

---

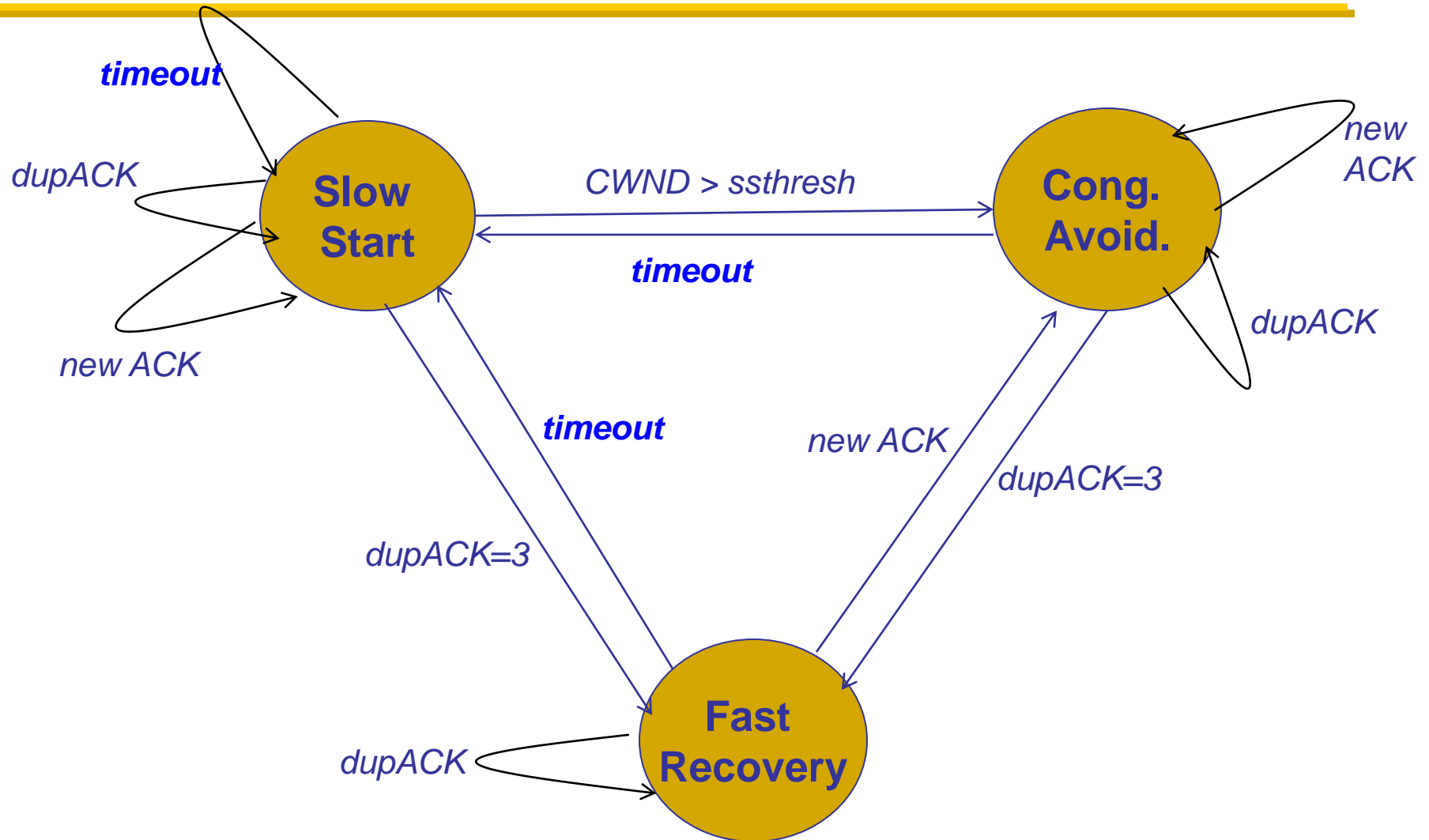
- ACK 101 (due to 102) cwnd=10 dup#1
- ACK 101 (due to 103) cwnd=10 dup#2
- ACK 101 (due to 104) cwnd=10 dup#3
- RETRANSMIT 101 ssthresh=5 cwnd= 8 (5+3)
- ACK 101 (due to 105) cwnd= 9 (no xmit)
- ACK 101 (due to 106) cwnd=10 (no xmit)
- ACK 101 (due to 107) cwnd=11 (xmit 111)
- ACK 101 (due to 108) cwnd=12 (xmit 112)
- ACK 101 (due to 109) cwnd=13 (xmit 113)
- ACK 101 (due to 110) cwnd=14 (xmit 114)
- ACK 111 (due to 101) cwnd = 5 (xmit 115) ← exiting fast recovery
- Packets 111-114 already in flight
- ACK 112 (due to 111) cwnd =  $5 + 1/5$  ← back in cong. avoidance

# TCP state machine

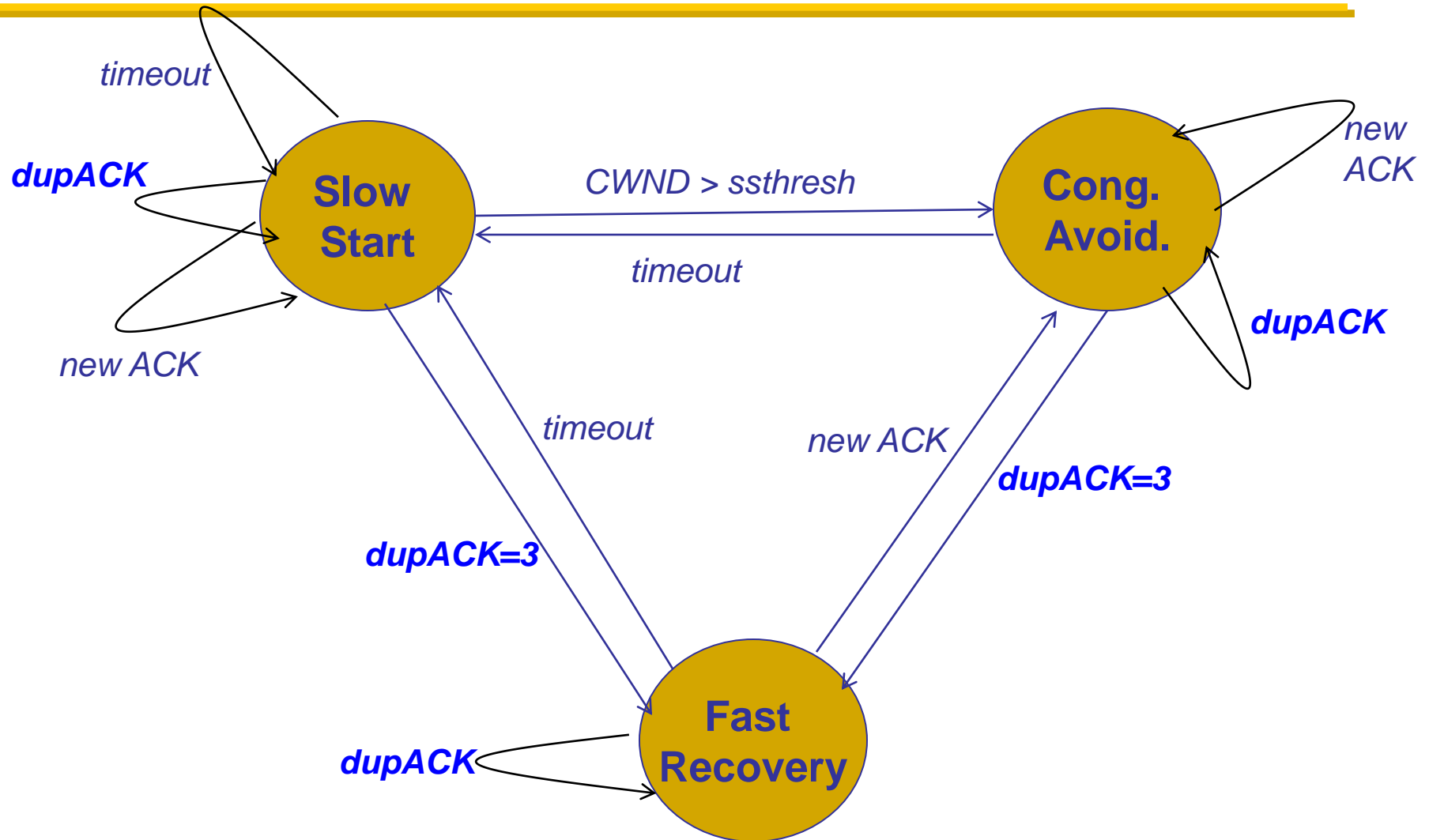




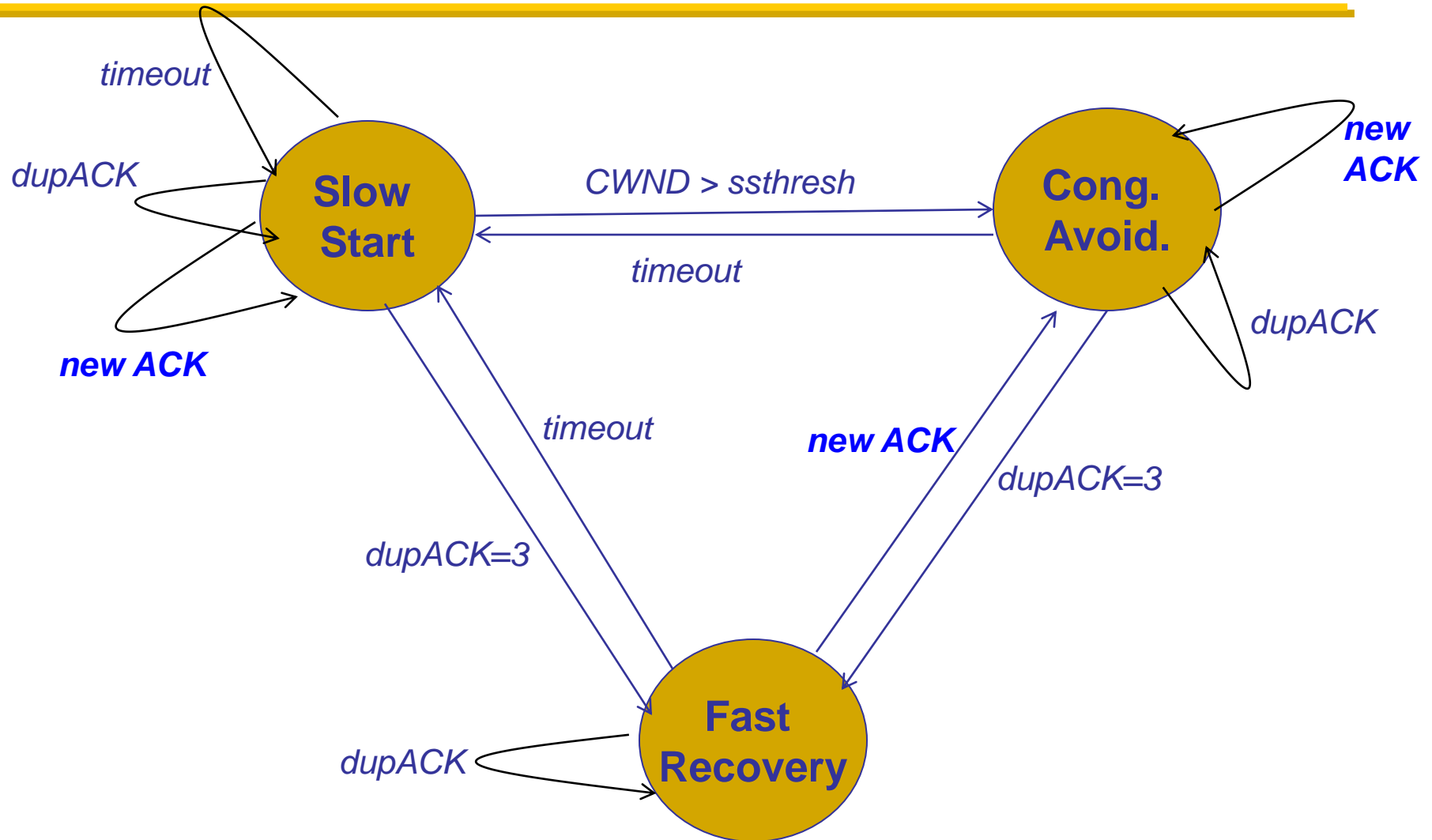
# Timeouts → Slow Start



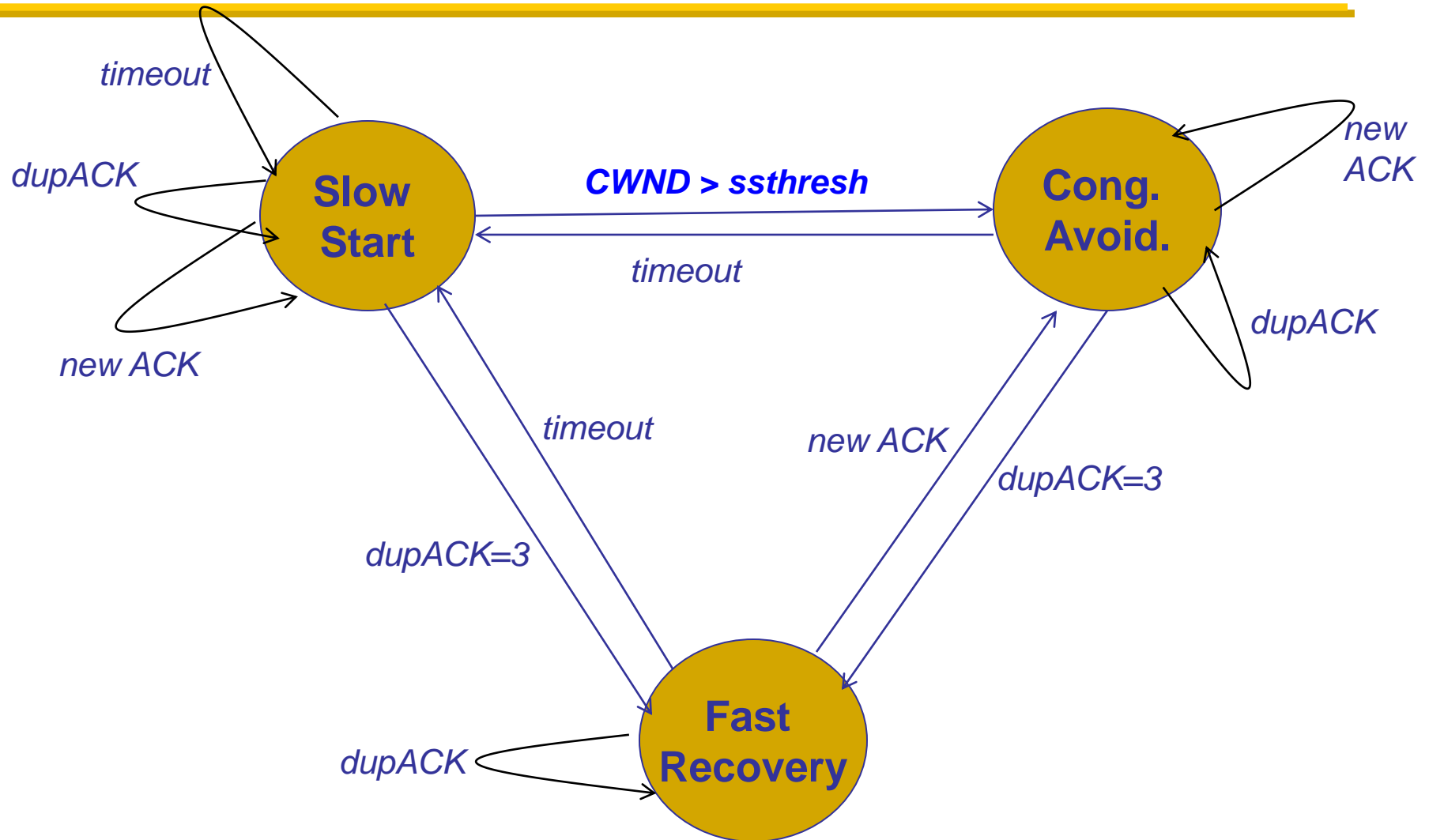
# dupACKs → Fast Recovery



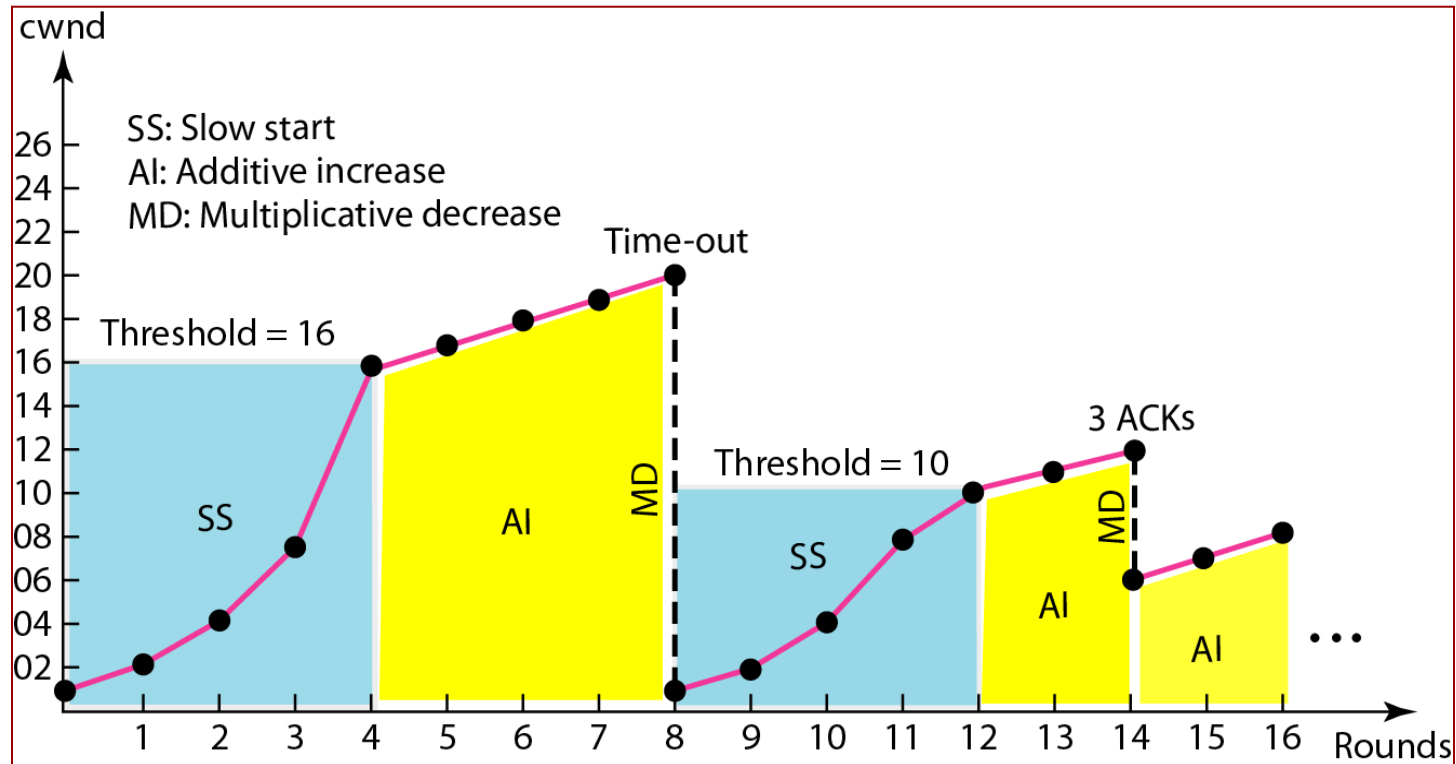
# New ACK changes state ONLY from Fast Recovery



# TCP state machine



# Timeout and Dup-ack



# TCP flavors

---

- TCP-Tahoe
  - $CWND = 1$  on 3 dupACKs
- TCP-Reno
  - $CWND = 1$  on timeout
  - $CWND = CWND/2$  on 3 dupACKs
- TCP-newReno
  - TCP-Reno + improved fast recovery
- TCP-SACK
  - Incorporates selective acknowledgements

**Our default  
assumption**

# How can they coexist?

---

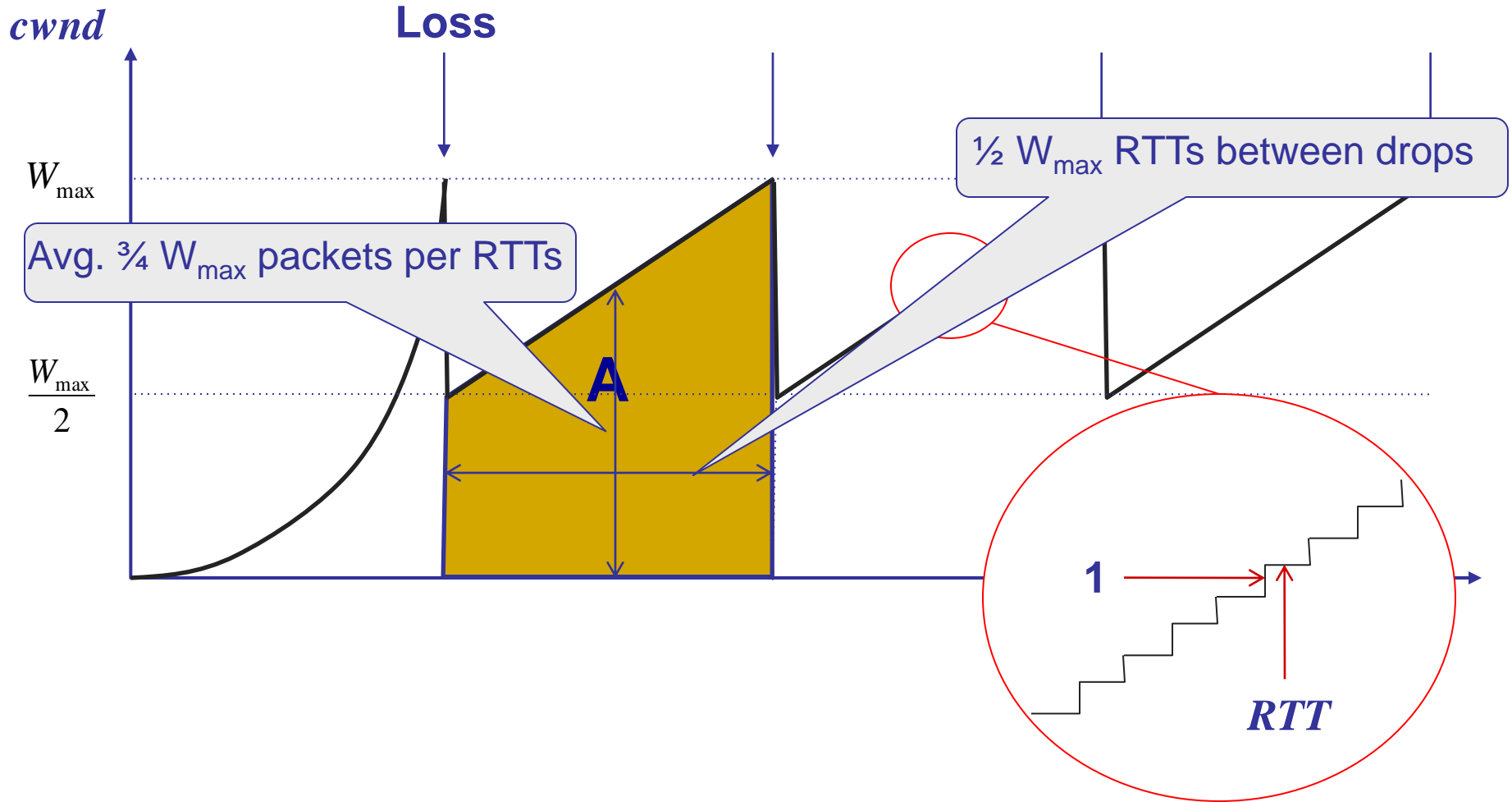
- All follow the same principle
  - Increase CWND on good news
  - Decrease CWND on bad news

---

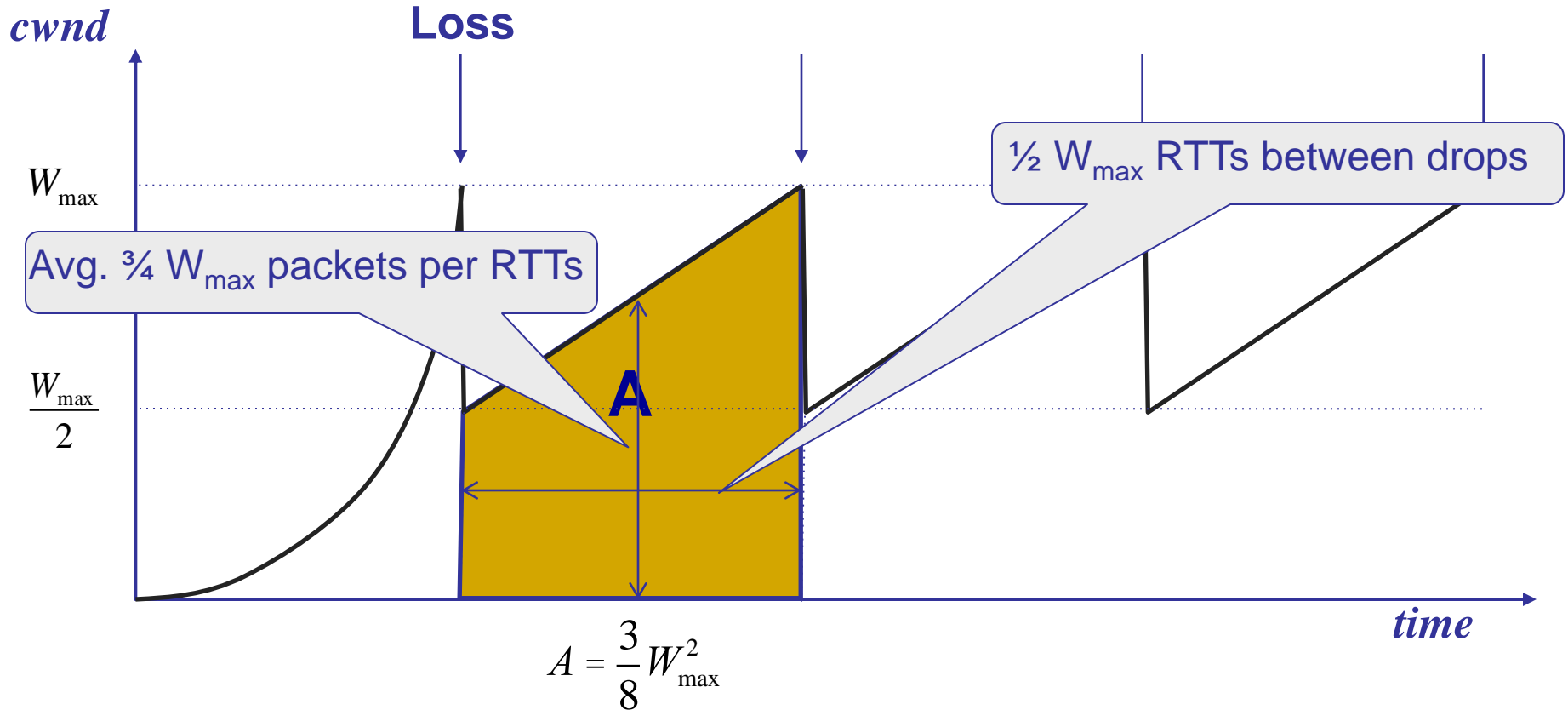
# **TCP THROUGHPUT EQUATION**



# A simple model for TCP throughput



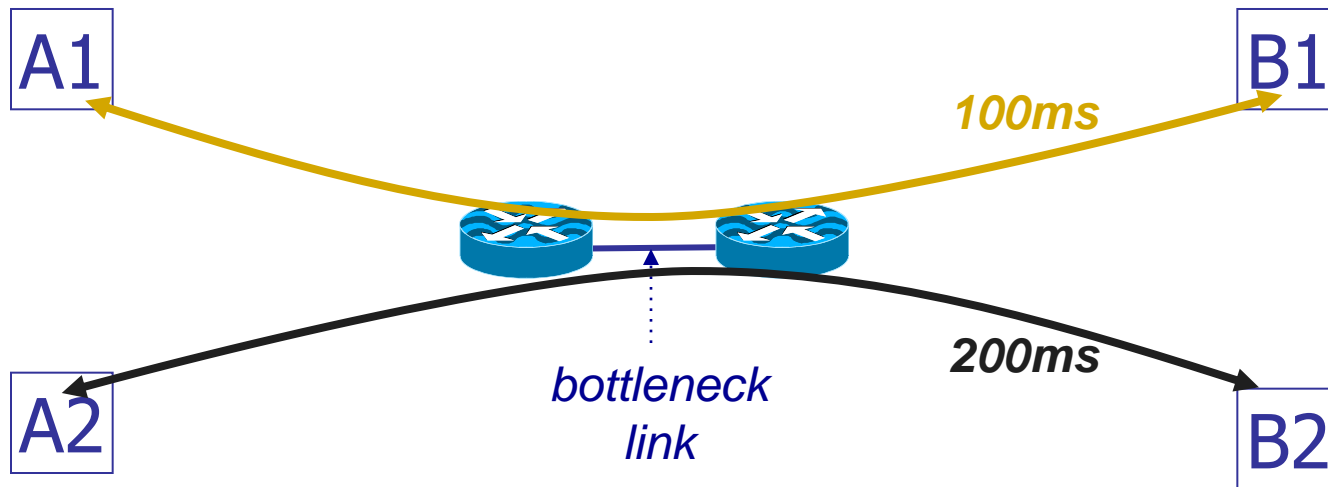
# A simple model for TCP throughput



# Implications (1): Different RTTs

$$\text{Throughput} = \sqrt{\frac{3}{2}} \frac{1}{RTT \sqrt{p}}$$

- Flows get throughput inversely proportional to RTT
- TCP unfair in the face of heterogeneous RTTs!



# Implications (2): High-speed TCP

---

$$\text{Throughput} = \sqrt{\frac{3}{2}} \frac{1}{RTT \sqrt{p}}$$

- Assume  $RTT = 100\text{ms}$ ,  $MSS=1500\text{bytes}$ ,  $BW=100\text{Gbps}$
- What value of  $p$  is required to reach  $100\text{Gbps}$  throughput?
  - $\sim 2 \times 10^{-12}$
- How long between drops?
  - $\sim 16.6$  hours
- How much data has been sent in this time?
  - $\sim 6$  petabits

# Adapting TCP to high speed

---

- Once past a threshold speed, increase CWND faster
  - A proposed standard [Floyd'03]: once speed is past some threshold, change equation to  $p-.8$  rather than  $p-.5$
  - Let the additive constant in AIMD depend on CWND
- Other approaches?
  - Multiple simultaneous connections ([hack but works today](#))
  - Router-assisted approaches

# Implications (3): Rate-based CC

$$\text{Throughput} = \sqrt{\frac{3}{2}} \frac{1}{RTT \sqrt{p}}$$

- TCP throughput swings between  $W/2$  to  $W$
- Apps may prefer steady rates (e.g., streaming)
- “Equation-Based Congestion Control”
  - Ignore TCP’s increase/decrease rules and just follow the equation
  - Measure drop percentage  $p$ , and set rate accordingly
- Following the TCP equation ensures “TCP friendliness”
  - i.e., use no more than TCP does in similar setting

# Implications (4):

## Loss not due to congestion?

---

- TCP will confuse corruption with congestion
- Flow will cut its rate
  - Throughput  $\sim 1/\sqrt{p}$  where  $p$  is loss prob.
  - Applies even for non-congestion losses!

# Implications (5):

## Short flows cannot ramp up

---

- 50% of flows have  $< 1500\text{B}$  to send; 80%  $< 100\text{KB}$
- Implications
  - Short flows never leave slow start!
    - » They never attain their fair share
  - Too few packets to trigger dupACKs
    - » Isolated loss may lead to timeouts
    - » At typical timeout values of  $\sim 500\text{ms}$ , might severely impact flow completion time



# Implications (6):

## Short flows share long delays

---

- A flow deliberately overshoots capacity, until it experiences a drop
- Means that delays are large, and are large for everyone
  - Consider a flow transferring a 10GB file sharing a bottleneck link with 10 flows transferring 100B
  - Larger flows dominate smaller ones

# Implications (7): Cheating

---

- Three easy ways to cheat
  - Increasing CWND faster than +1 MSS per RTT

# Implications (7): Cheating

---

- Three easy ways to cheat
  - Increasing CWND faster than +1 MSS per RTT
  - Using large initial CWND
    - » Common practice by many companies

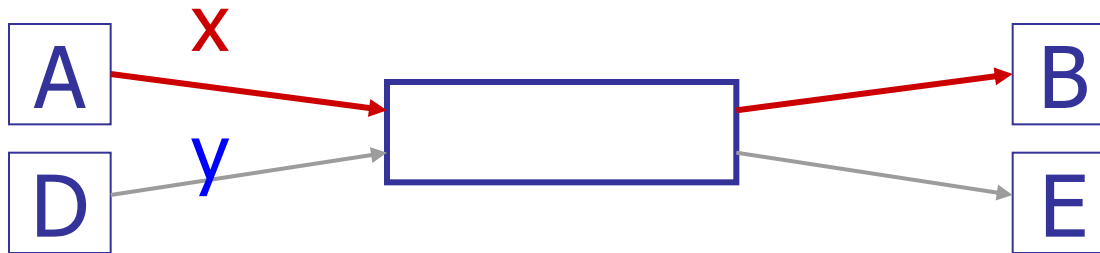
# Implications (7): Cheating

---

- Three easy ways to cheat
  - Increasing CWND faster than +1 MSS per RTT
  - Using large initial CWND
    - » Common practice by many companies
  - Opening many connections

# Open many connections

---



- Assume
  - A starts 10 connections to B
  - D starts 1 connection to E
  - Each connection gets about the same throughput
- Then A gets 10 times more throughput than D

# Implications (8): CC intertwined with reliability

---

- CWND adjusted based on ACKs and timeouts
- Cumulative ACKs and fast retransmit/recovery rules
- Complicates evolution
  - Changing from cumulative to selective ACKs is hard
- Sometimes we want CC but not reliability
  - e.g., real-time applications
- We may also want reliability without CC

# Summary

---

- TCP works even though it has many flaws
- Many of them can be fixed via assistance from the network