南京大學
NANJING UNIVERSITY

Computer Networks

Wenzhong Li, Chen Tian

Nanjing University

Material with thanks to James F. Kurose, Mosharaf Chowdhury, and other colleagues.

Outline

- TCP congestion control wrap-up
- TCP throughput equation
- Problems with congestion control
- Router assisted 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{RWND, 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 = 2xCWND

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

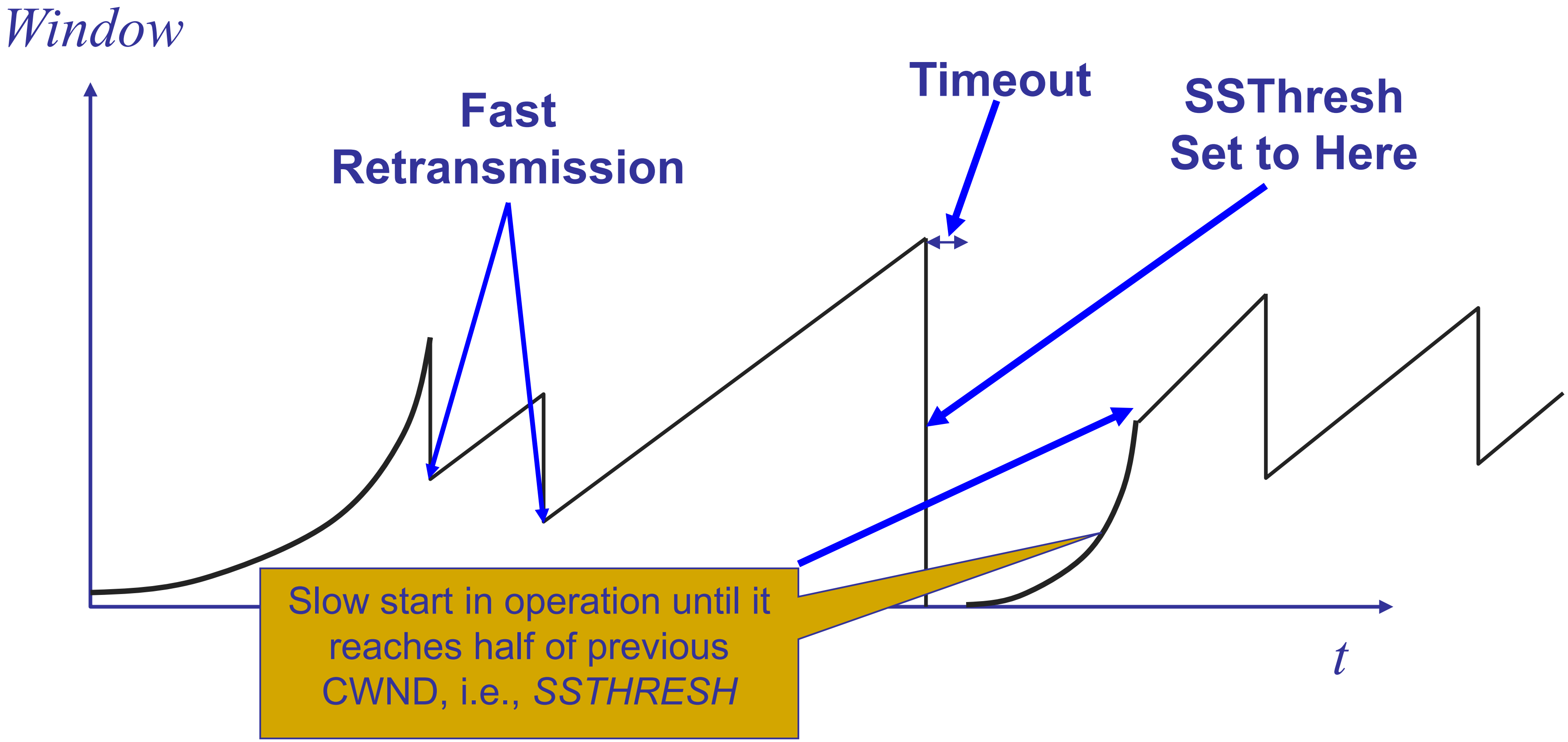
Event: TimeOut

- On Timeout
  - ssthresh ← CWND/2
  - CWND ← 1

Event: dupACK

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

Example



Slow-start restart: Go back to CWND = 1 MSS, but take advantage of knowing the previous value of 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 dupACKcount = 3
  - ssthresh = CWND/2
  - CWND = ssthresh + 3
- While in fast recovery
  - CWND = CWND + 1 for each additional dupACK
- Exit fast recovery after receiving new ACK
  - set CWND = ssthresh

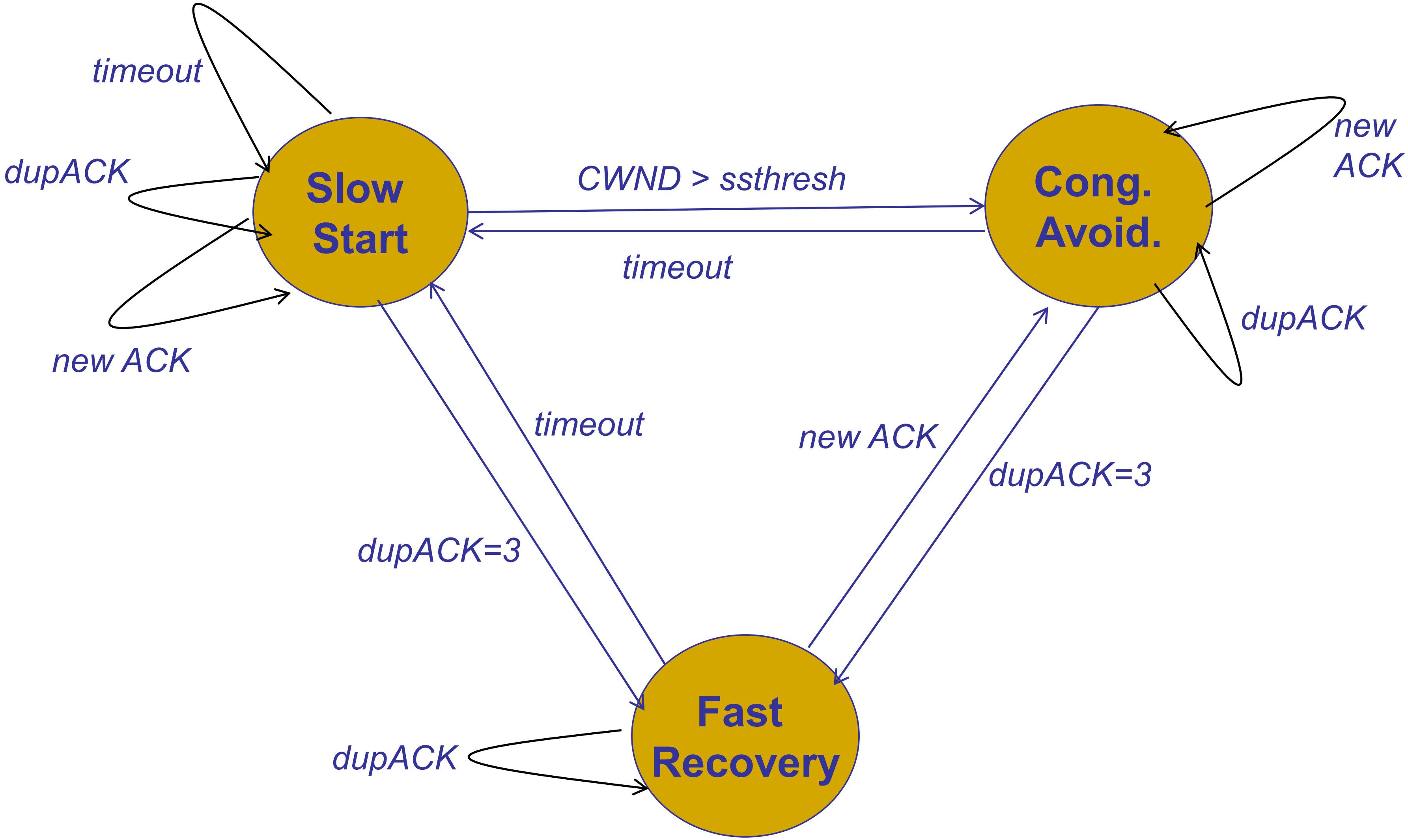
Example

- 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

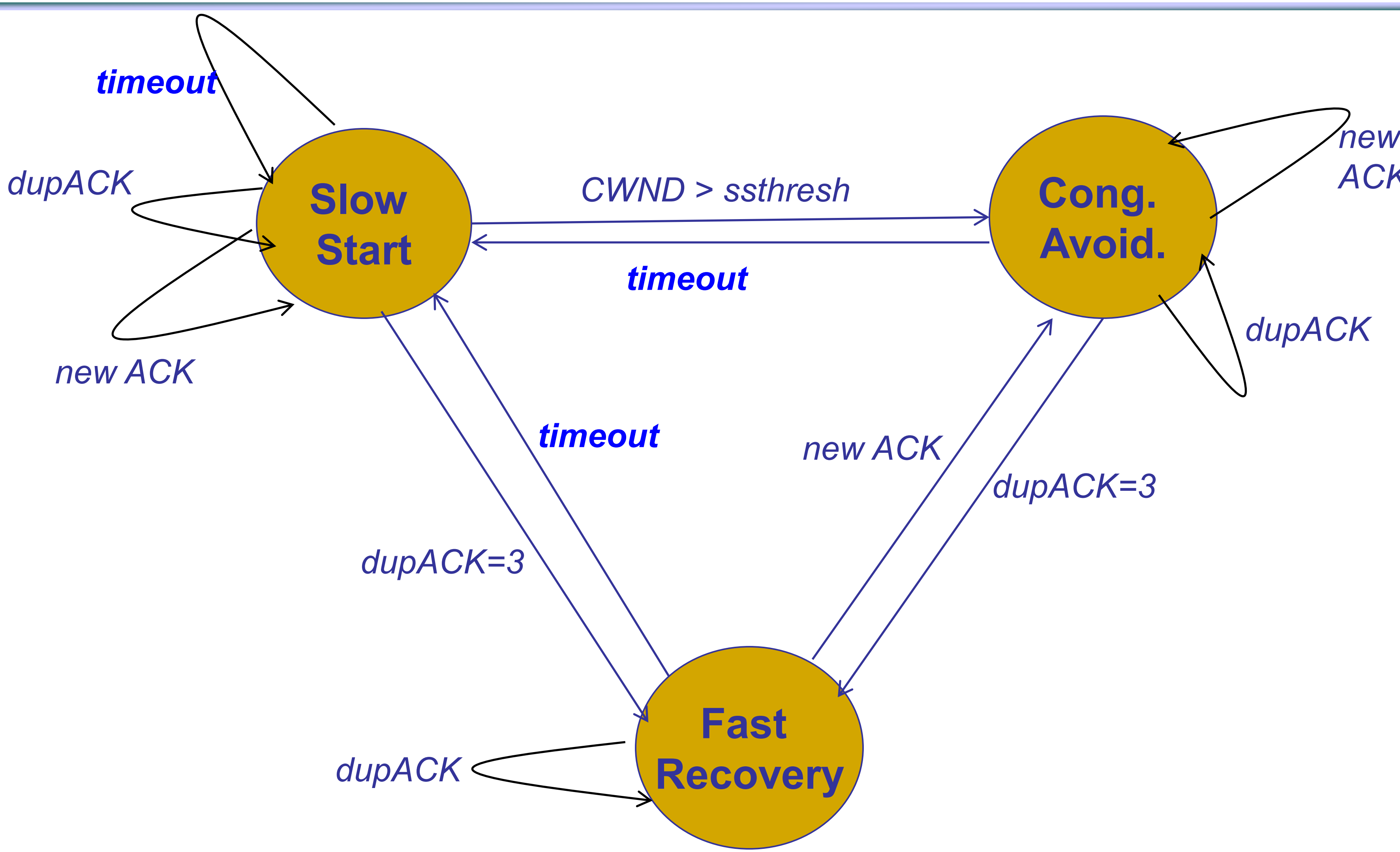
Timeline: [~~101~~, 102, …, 110]

- 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

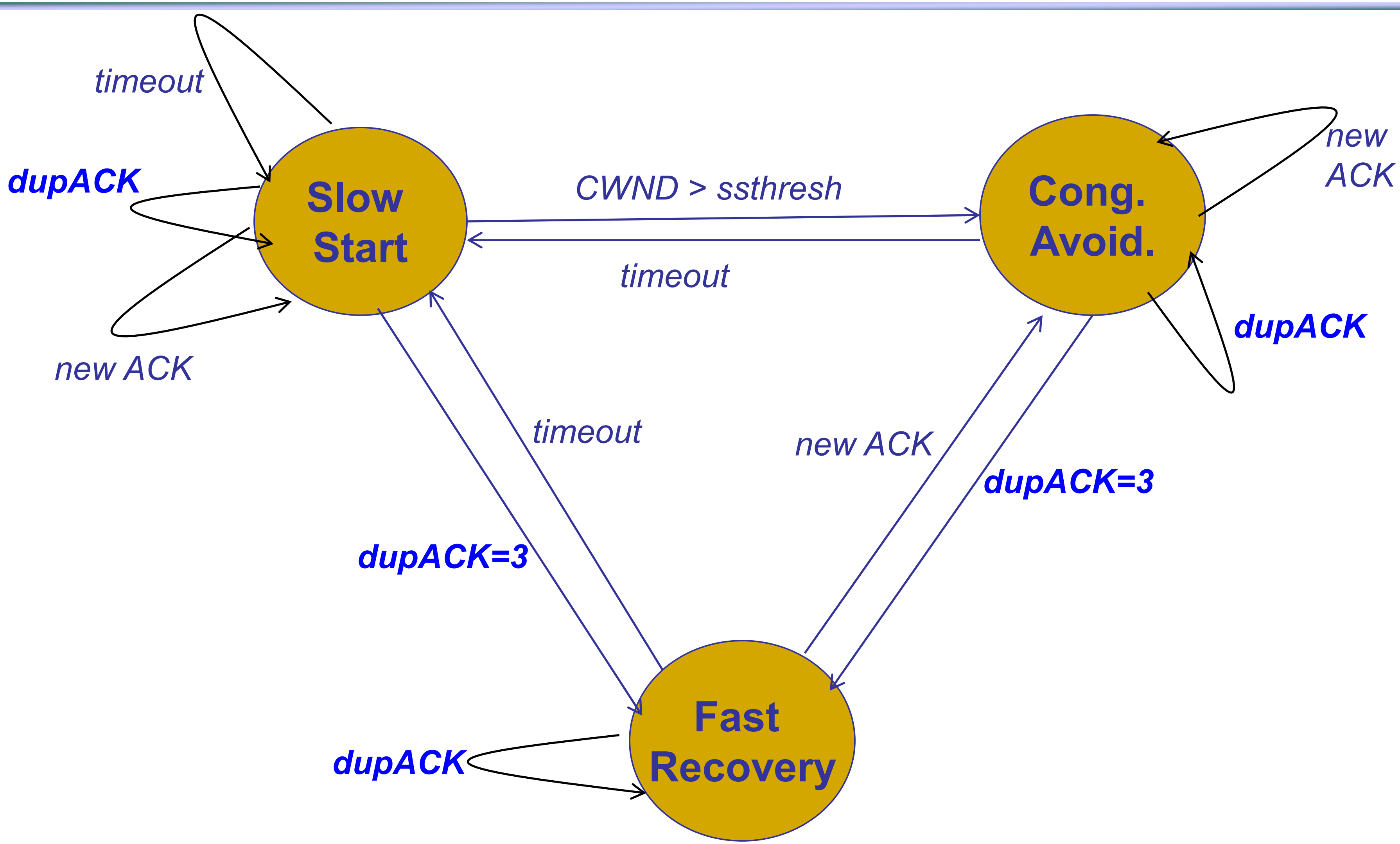
TCP state machine



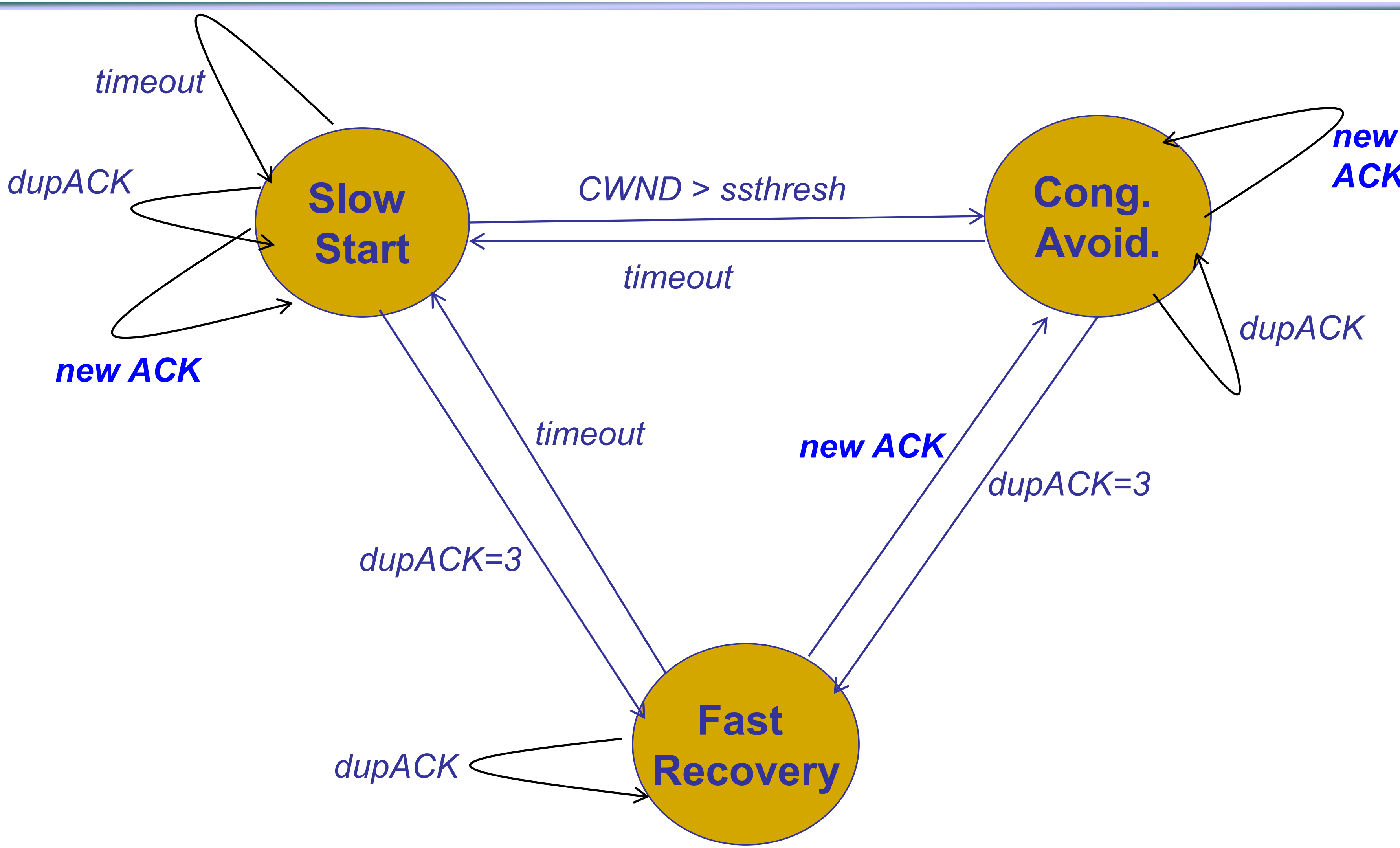
Timeouts → Slow Start



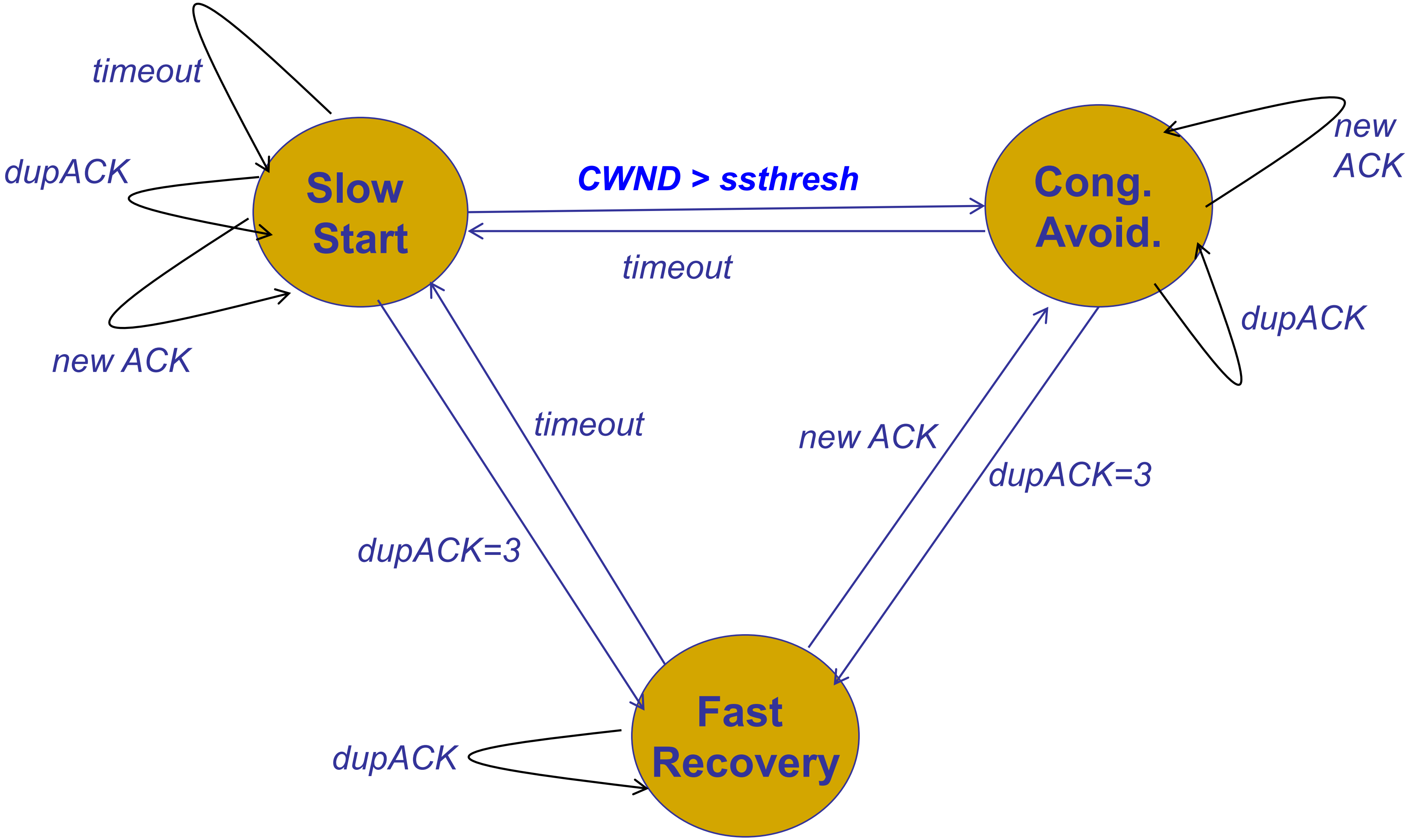
dupACKs → Fast Recovery



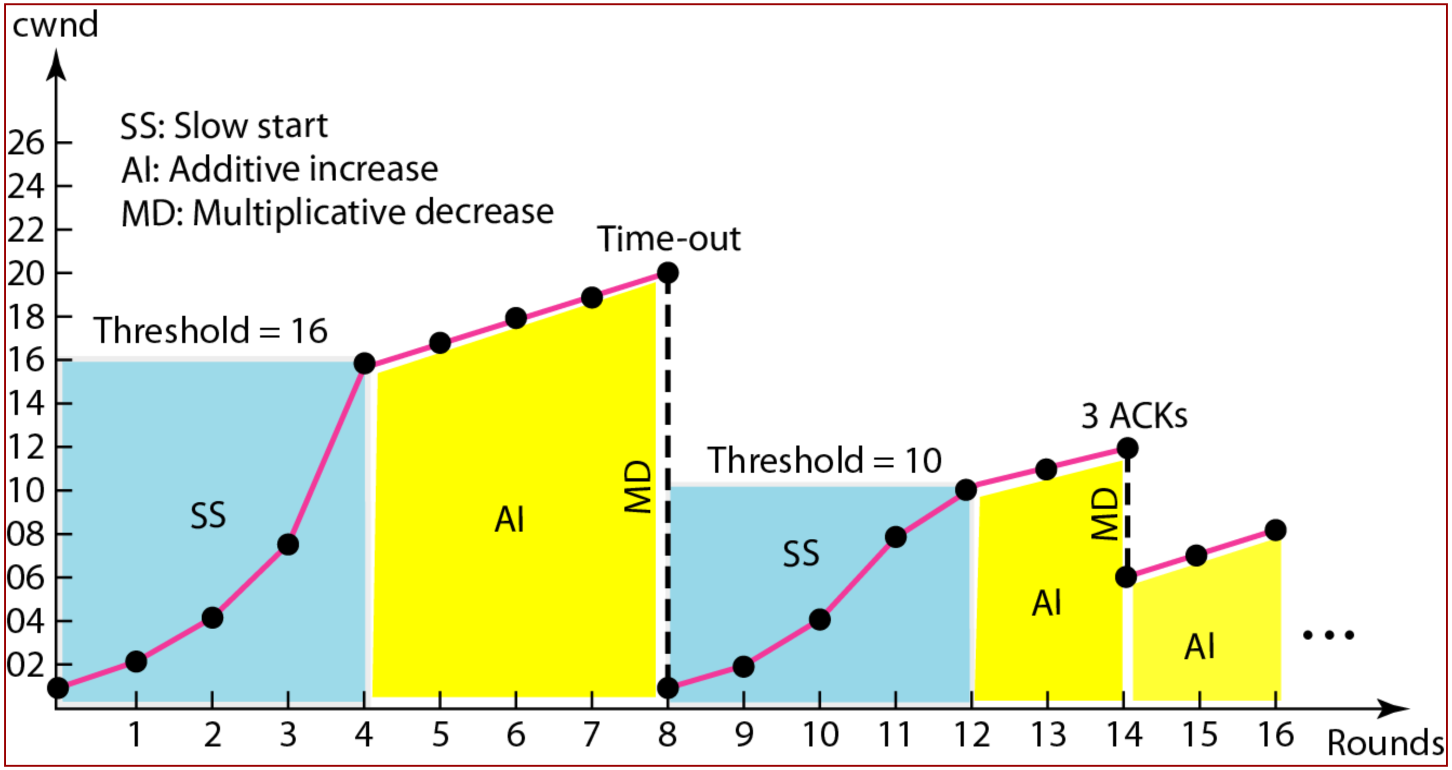
New ACK changes state ONLY from Fast Recovery



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

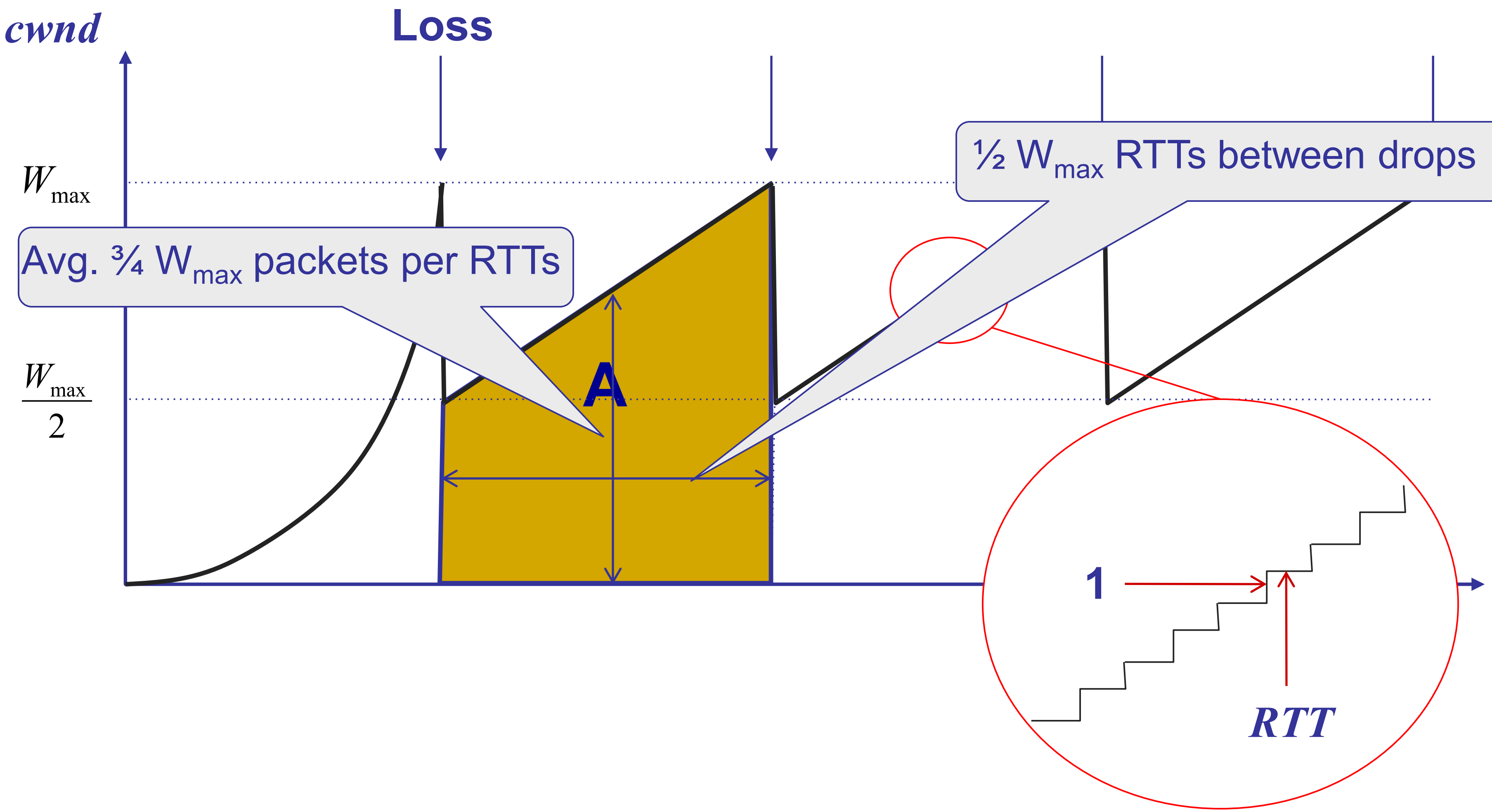
Our default assumption

How can they coexist?

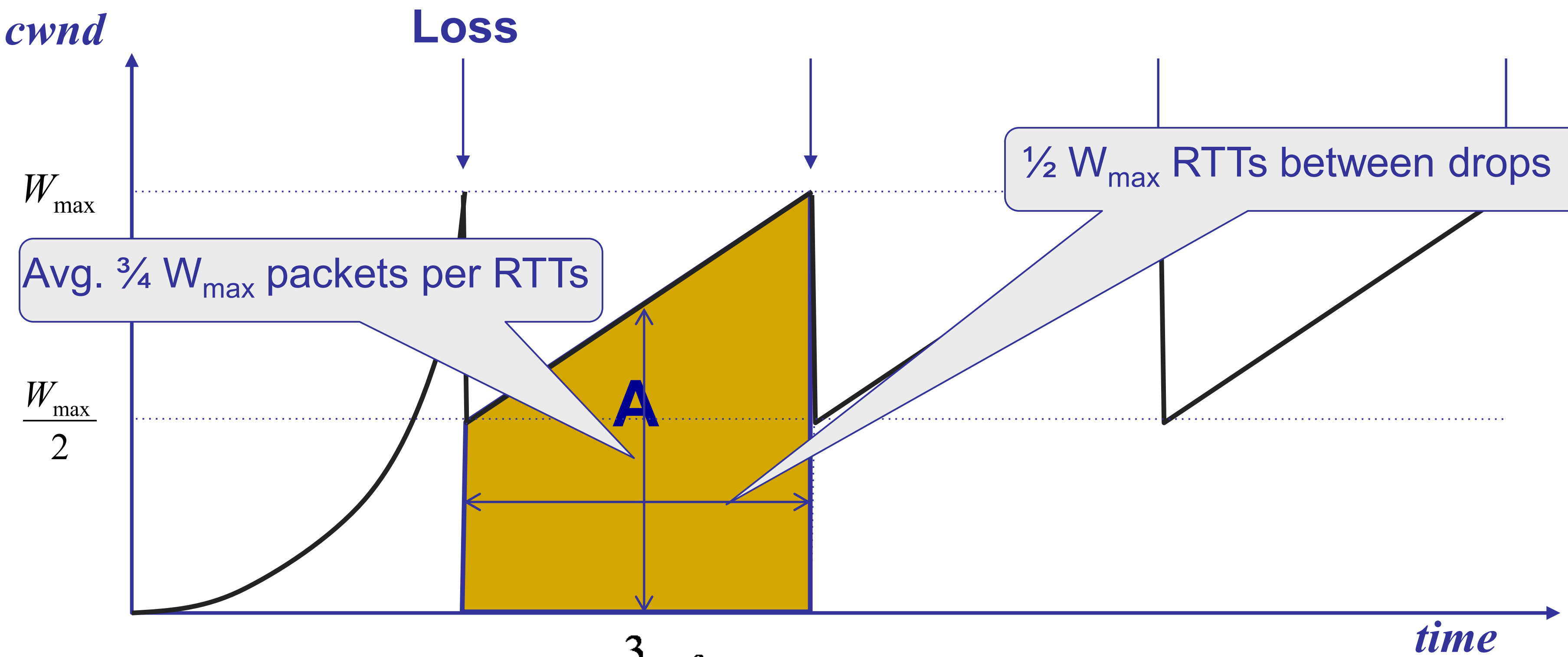
- All follow the same principle
  - Increase CWND on good news
  - Decrease CWND on bad news

TCP THROUGHPUT EQUATION

A simple model for TCP throughput



A simple model for TCP throughput



$$A = \frac{3}{8} W_{max}^2$$

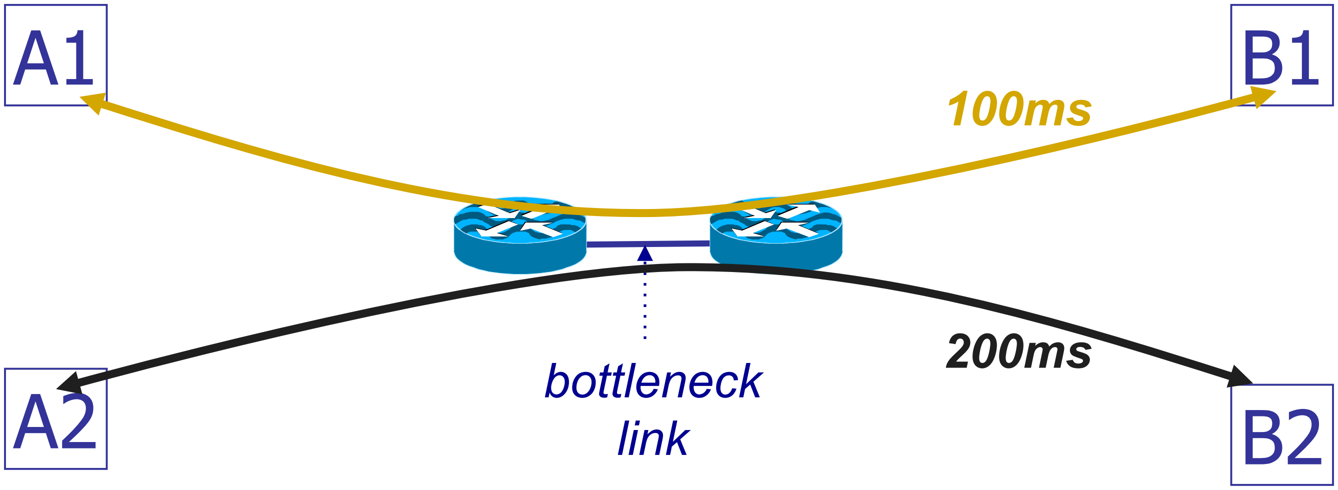
Packet drop rate, $p = 1/A$

Throughput, $B = \dfrac{A}{\left(\frac{W_{max}}{2}\right) RTT} = \sqrt{\frac{3}{2}} \dfrac{1}{RTT\sqrt{p}}$

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

Adapting TCP to high speed

- Once past a threshold speed, increase CWND faster
  - 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 is 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 ~ 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 < 1500B to send; 80% < 100KB
- 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 ~500ms, might severely impact flow completion time

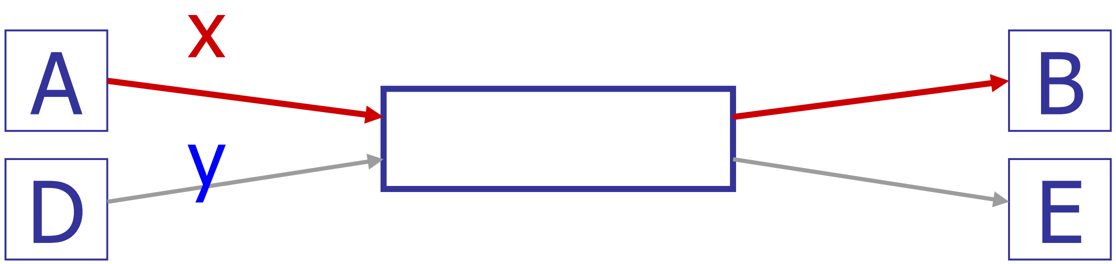
Implications (6): Short flows share long delays

- 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
  - 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

ROUTER-ASSISTED CONGESTION CONTROL

Recap: TCP problems

- Misled by non-congestion losses
- Fills up queues leading to high delays
- Short flows complete before discovering available capacity
- AIMD impractical for high speed links
- Saw tooth discovery too choppy for some apps
- Unfair under heterogeneous RTTs
- Tight coupling with reliability mechanisms
- End hosts can cheat

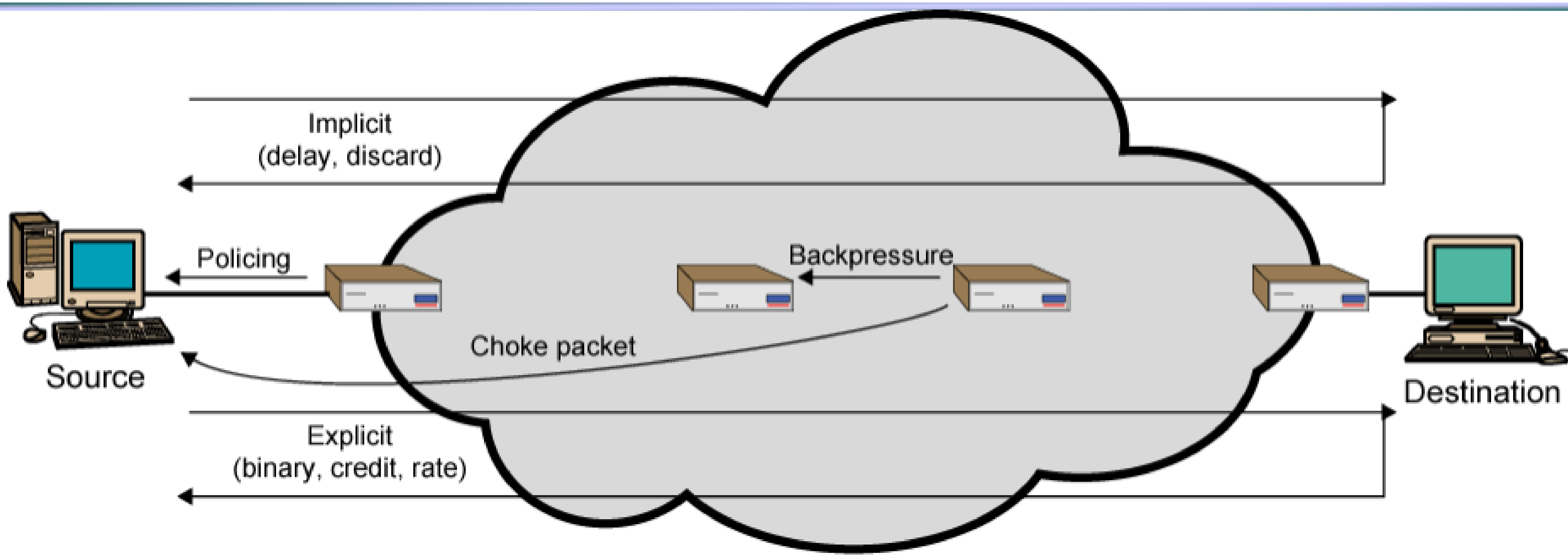
Routers tell endpoints if they're congested

Routers tell endpoints what rate to send at

Routers enforce fair sharing

Could fix many of these with some help from routers!

Mechanisms for Congestion Control

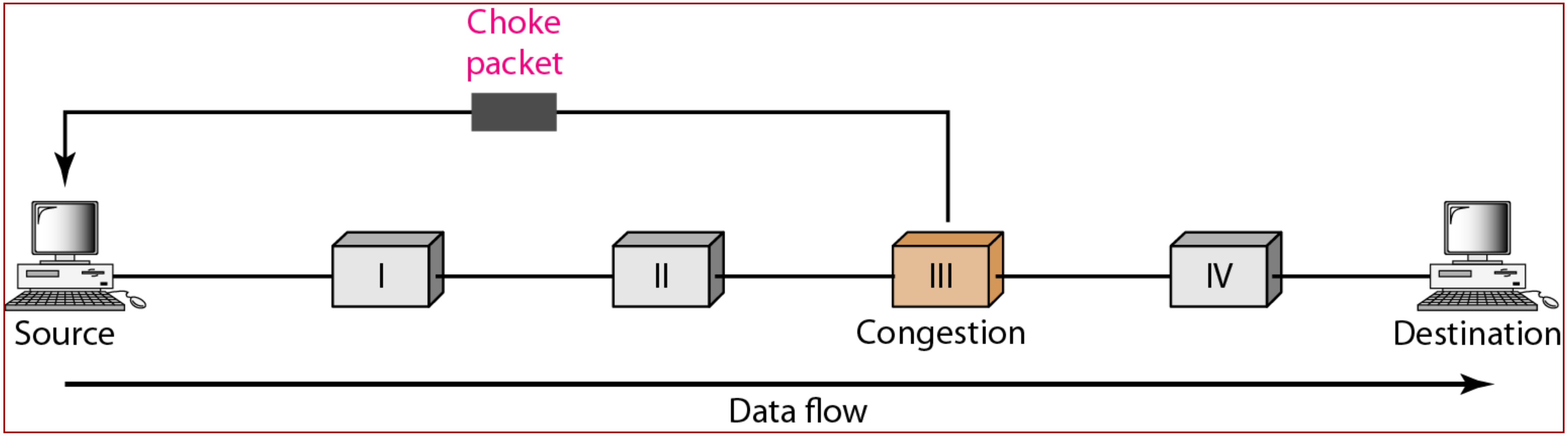


- Choke Packet
- Backpressure
- Warning bit
- Random early discard
- Fair Queuing (FQ)

- 抑制分组
- 反压
- 警告位
- 随机早期丢弃
- 公平队列

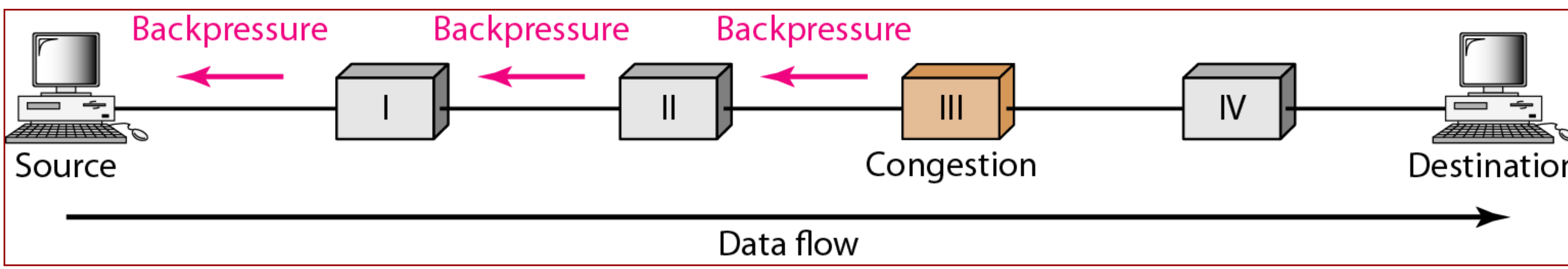
(1) Choke Packet

- Control packet
  - Generated at congested node
  - Sent to source node
- Source quench： using ICMP to notify source
  - From router or destination, sent for every discarded packet



(2) Backpressure

- Hop-by-Hop Choke Packets
  - Propagation time > transmission time (long distance or high speed link)
  - Choke packets from router to source are not effective
  - Require each hop to reduce its transmission



(3) Warning Bit

Explicit Congestion Notification (ECN)

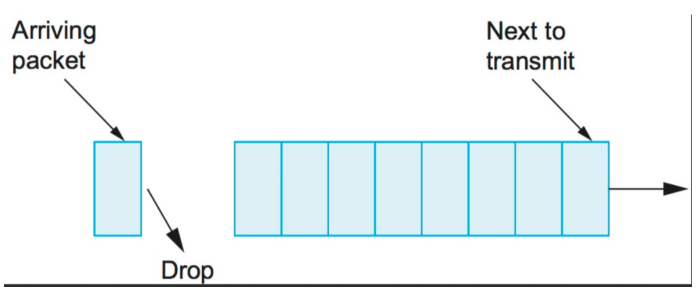
- Single bit in packet header; set by congested routers
  - If data packet has bit set, then ACK has ECN bit set
- Many options for when routers set the bit
  - Tradeoff between (link) utilization and (packet) delay
- Congestion semantics can be exactly like that of drop
  - i.e., end-host reacts as though it saw a drop

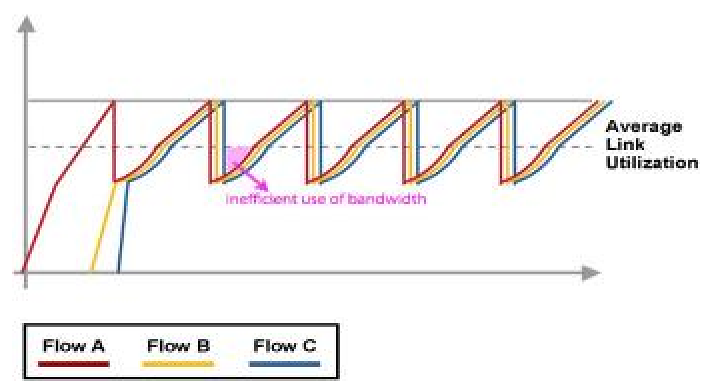
ECN

- Advantages:
  - Don’t confuse corruption with congestion; recovery w/ rate adjustment
  - Can serve as an early indicator of congestion to avoid delays
  - Easy (easier) to incrementally deploy
    - » Today: defined in RFC 3168 using ToS/DSCP bits in the IP header
    - » Common in datacenters
- Use ECN as congestion markers
  - Whenever I get an ECN bit set, I have to pay $$

(4) Random Early Discard

- Control congestion at routers (switches)
  - Combined with congestion window at hosts
- TCP global synchronization problem
  - Traffic burst fills queues so packets lost, TCP connections enter slow start
  - Traffic drops so network under utilized, connections leave slow start at same time causing burst again
- Handle the problem – RED
  - Router randomly discards packets before buffer becomes completely full





The RED Algorithm

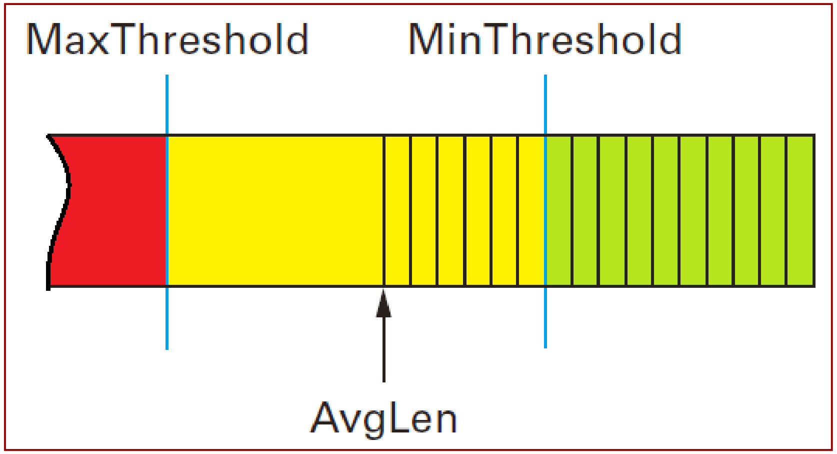
Compute average queue length

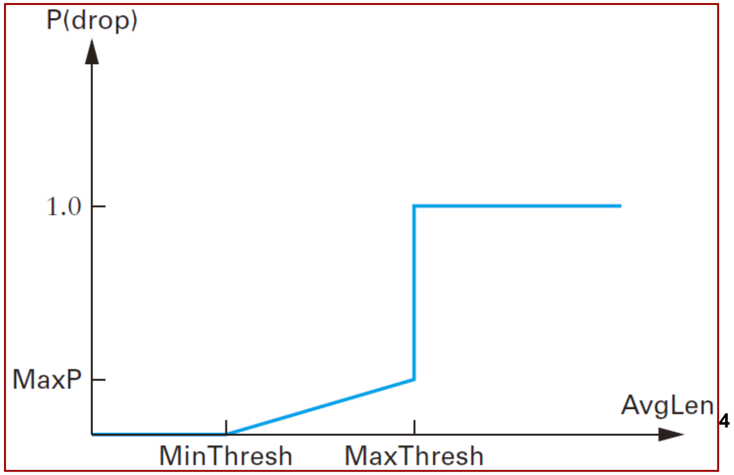
$$avgLen = (1-\omega)\times avgLen + \omega \times sampleLen$$

Calculate average queue size *avgLen*
if *avgLen* < TH_{min}
    queue packet

else if $TH_{min} \leq avgLen < TH_{max}$
    calculate probability p
    with probability p discard packet
    else with probability 1-p queue packet

else if *avg* $\geq TH_{max}$
    discard packet





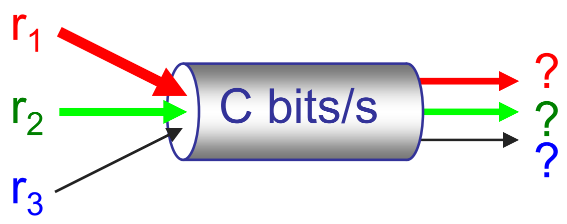
(5) Fairness: General approach

- What does “fair” mean exactly?

- Routers classify packets into “flows”
  - Let’s assume flows are TCP connections
- Each flow has its own FIFO queue in router
- Router services flows in a fair fashion
  - When line becomes free, take packet from next flow in a fair order

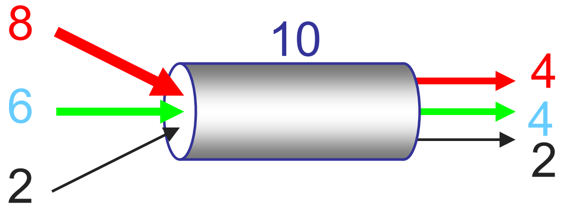
Max-Min fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:
  - $a_i = \min(f, r_i)$
  - where f is the unique value such that $\mathrm{Sum}(a_i) = C$



Example

- $C = 10$; $r_1 = 8$, $r_2 = 6$, $r_3 = 2$; $N = 3$
- $C/3 = 3.33 \rightarrow$
  - r_3's need is only 2
    - »Can service all of r_3
  - Remove r_3 from the accounting: $C = C - r_3 = 8$; $N = 2$
- $C/2 = 4 \rightarrow$
  - Can't service all of r_1 or r_2
  - So hold them to the remaining fair share: $f = 4$



f = 4:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

Max-Min fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:
  - $a_i = \min(f, r_i)$
  - where f is the unique value such that $\mathrm{Sum}(a_i) = C$
- If you don't get full demand, no one gets more than you
- This is what round-robin service gives if all packets are the same size

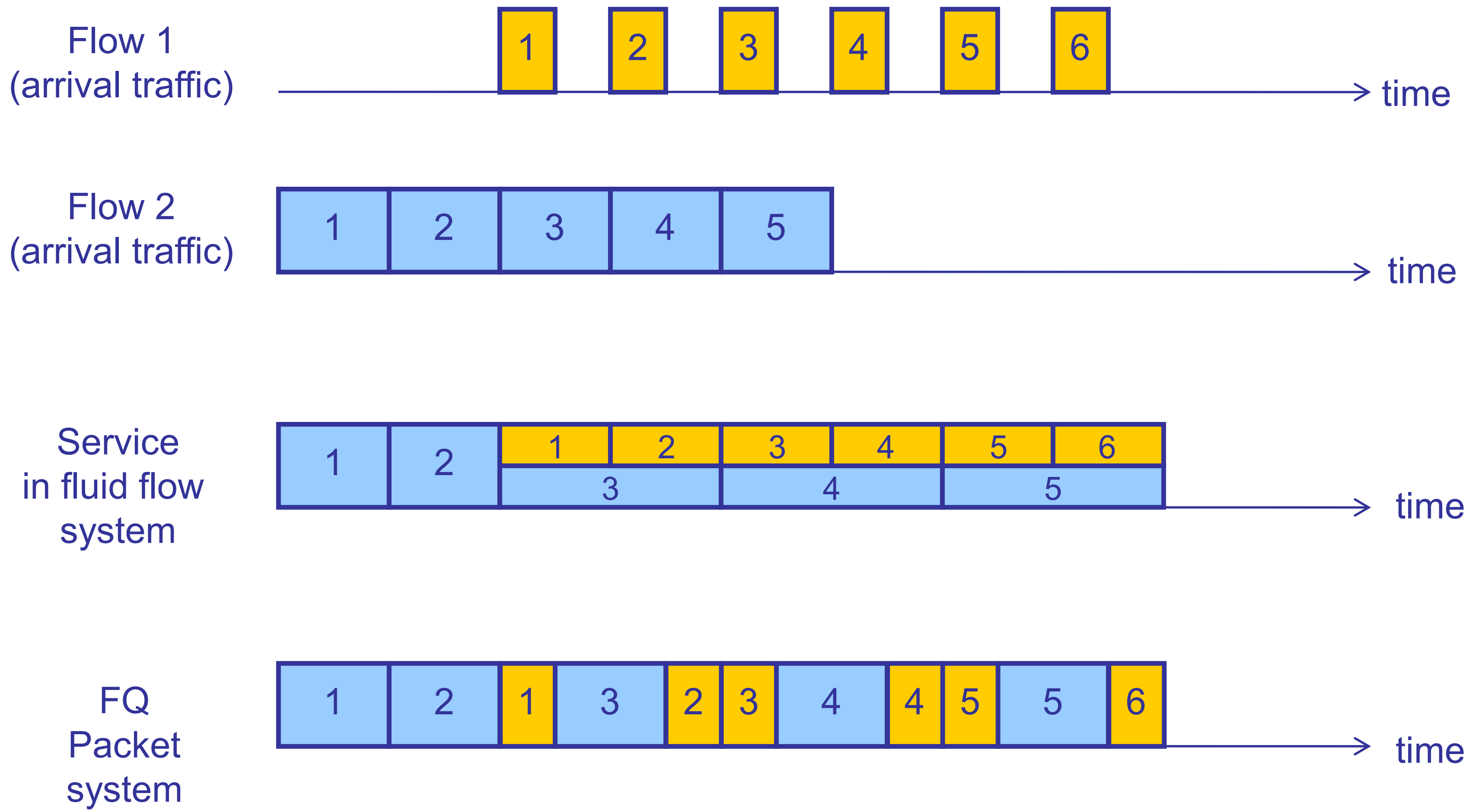
How do we deal with packets of different sizes?

- Mental model: Bit-by-bit round robin (“fluid flow”)
- Can you do this in practice?
  - No, packets cannot be preempted
- But we can approximate it
  - This is what “fair queuing” routers do

Fair Queuing (FQ)

- For each packet, compute the time at which the last bit of a packet would have left the router if flows are served bit-by-bit
- Then serve packets in the increasing order of their deadlines

Example


Flow 1
(arrival traffic)
1 2 3 4 5 6
time
Flow 2
(arrival traffic)
1 2 3 4 5
time
Service
in fluid flow
system
1 2
1 2 3 4 5 6
3 4 5
time
FQ
Packet
system
1 2 1 3 2 3 4 4 5 5 6
time

Fair Queuing (FQ)

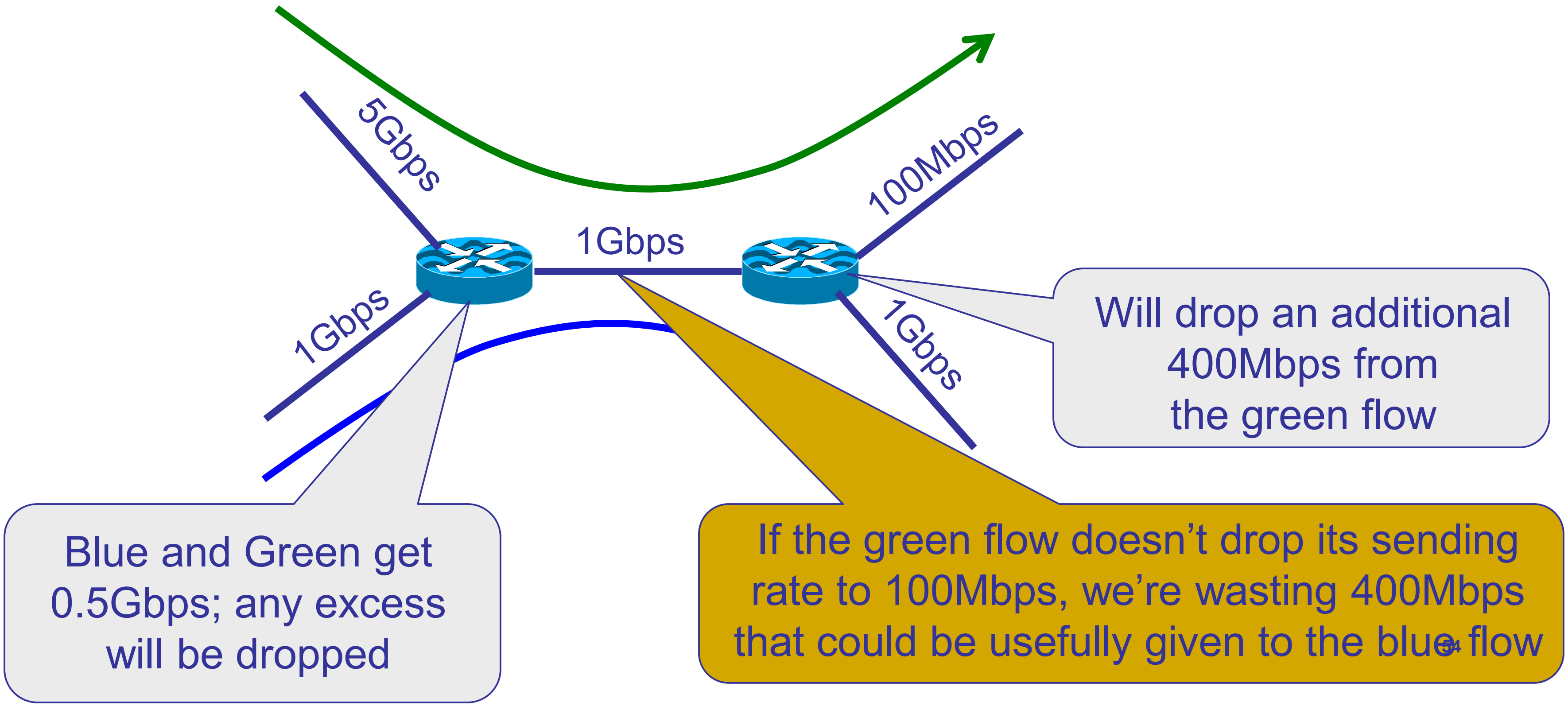
- Implementation of round-robin generalized to the case where not all packets are equal sized
- Weighted fair queuing (WFQ): assign different flows different shares
- Today, some form of WFQ implemented in almost all routers
  - Not the case in the 1980-90s, when CC was being developed
  - Mostly used to isolate traffic at larger granularities (e.g., per-prefix)

FQ vs. FIFO

- FQ advantages:
  - Isolation: cheating flows don't benefit
  - Bandwidth share does not depend on RTT
  - Flows can pick any rate adjustment scheme they want
- Disadvantages:
  - More complex than FIFO: per flow queue/state, additional per-packet book-keeping

FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion



FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion
  - Robust to cheating, variations in RTT, details of delay, reordering, retransmission, etc.
- But congestion (and packet drops) still occurs
- We still want end-hosts to discover/adapt to their fair share!

Fairness is a controversial goal

- What if you have 8 flows, and I have 4?
  - Why should you get twice the bandwidth?
- What if your flow goes over 4 congested hops, and mine only goes over 1?
  - Why shouldn't you be penalized for using more scarce bandwidth?
- What is a flow anyway?
  - TCP connection
  - Source-Destination pair?
  - Source?

Why not let routers tell what rate end hosts should use?

- Packets carry “rate field”
- Routers insert “fair share” f in packet header
- End-hosts set sending rate (or window size) to f
  - Hopefully (still need some policing of end hosts!)
- This is the basic idea behind the “Rate Control Protocol” (RCP) from Dukkipati et al. ’07
  - Flows react faster

Summary

- TCP congestion control wrap-up
- TCP throughput equation
- Problems with congestion control
- Router assisted congestion control
  - Choke packet
  - Backpressure
  - Warning bit
  - Random early discard
  - Fair Queuing (FQ)