Congestion Control

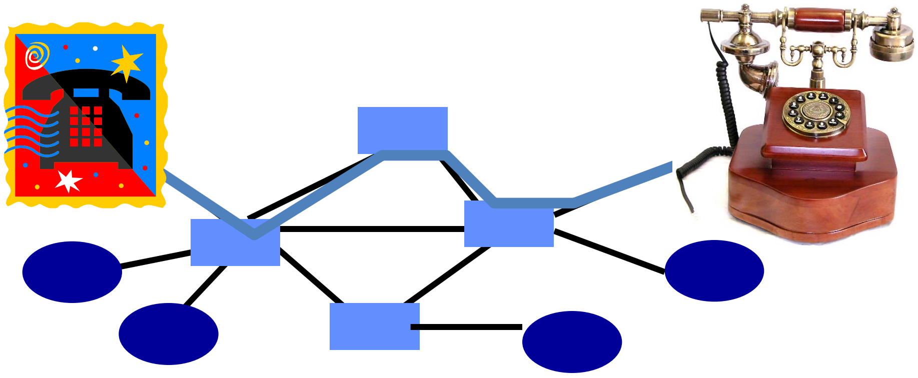
Note: The slides are adapted from the materials from Prof. Richard Han at CU Boulder and Profs. Jennifer Rexford and Mike Freedman at Princeton University, and the networking book (Computer Networking: A Top Down Approach) from Kurose and Ross.

Goals of Today’s Lecture

- Congestion in IP networks
  - Unavoidable due to best-effort service model
  - IP philosophy: decentralized control at end hosts
- Congestion control by the TCP senders
  - Infers congestion is occurring (e.g., from packet losses)
  - Slows down to alleviate congestion, for the greater good
- TCP congestion-control algorithm
  - Additive-increase, multiplicative-decrease
  - Slow start and slow-start restart

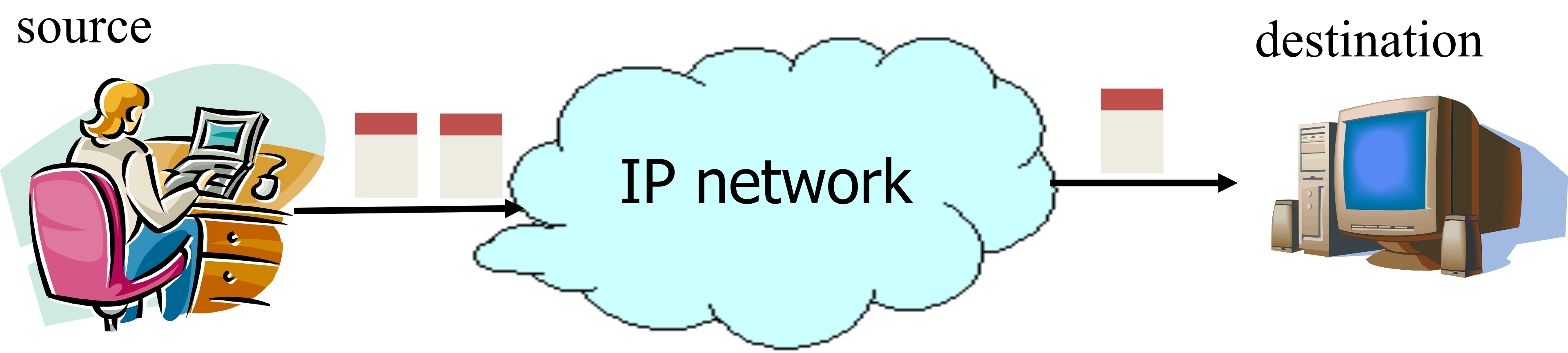
No Problem Under Circuit Switching

- Source establishes connection to destination
  - Nodes reserve resources for the connection
  - Circuit rejected if the resources aren't available
  - Cannot have more than the network can handle



IP Best-Effort Design Philosophy

- Best-effort delivery
  - Let everybody send
  - Network tries to deliver what it can
  - … and just drop the rest

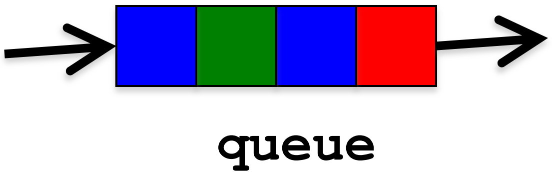


Congestion Control

Distributed Resource Sharing

Congestion

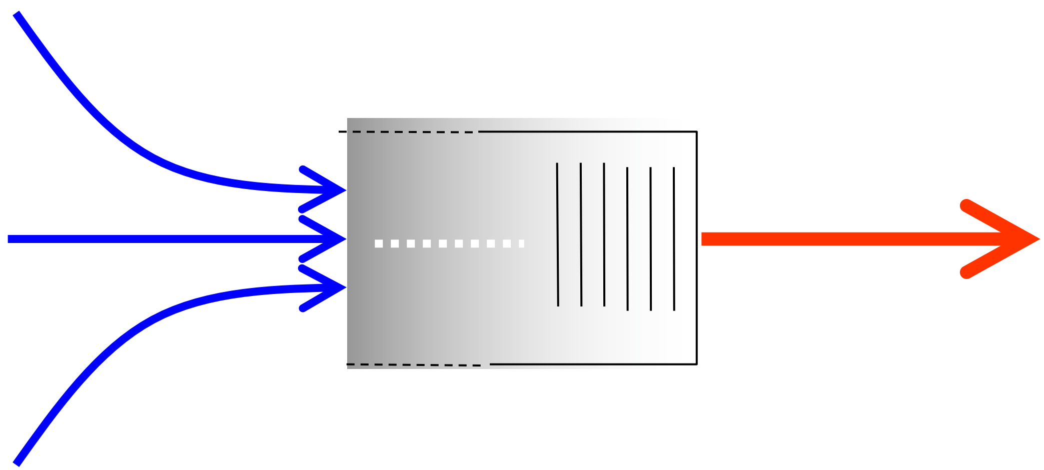
- Best-effort network does not "block" calls
  - So, they can easily become overloaded
  - Congestion == "Load higher than capacity"
- Examples of congestion
  - Link layer: Ethernet frame collisions
  - Network layer: full IP packet buffers



- Excess packets are simply dropped
  - And the sender can simply retransmit

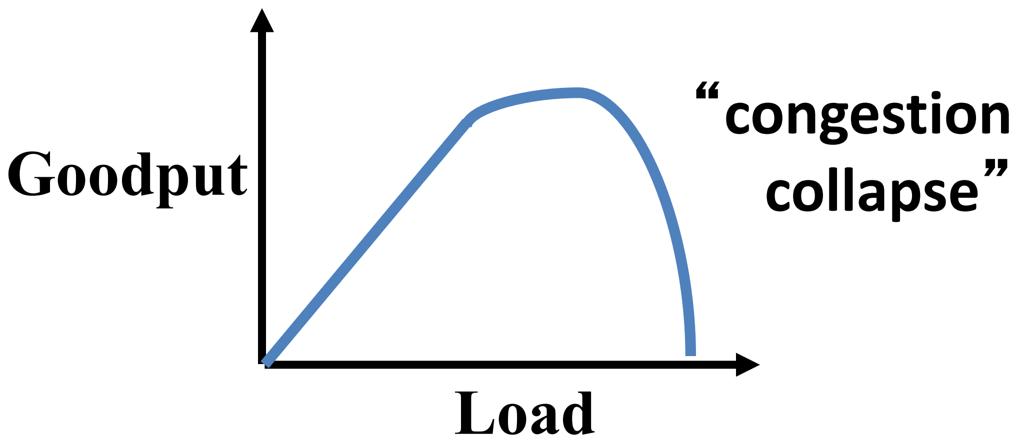
Congestion is Unavoidable

- Two packets arrive at same time
  - Router can only transmit one: must buffer or drop other
- If many packets arrive in short period of time
  - Router cannot keep up with the arriving traffic
  - Buffer may eventually overflow



Congestion Collapse

- Easily leads to *congestion collapse*
  - Senders retransmit the lost packets
  - Leading to even *greater* load
  - ... and even *more* packet loss

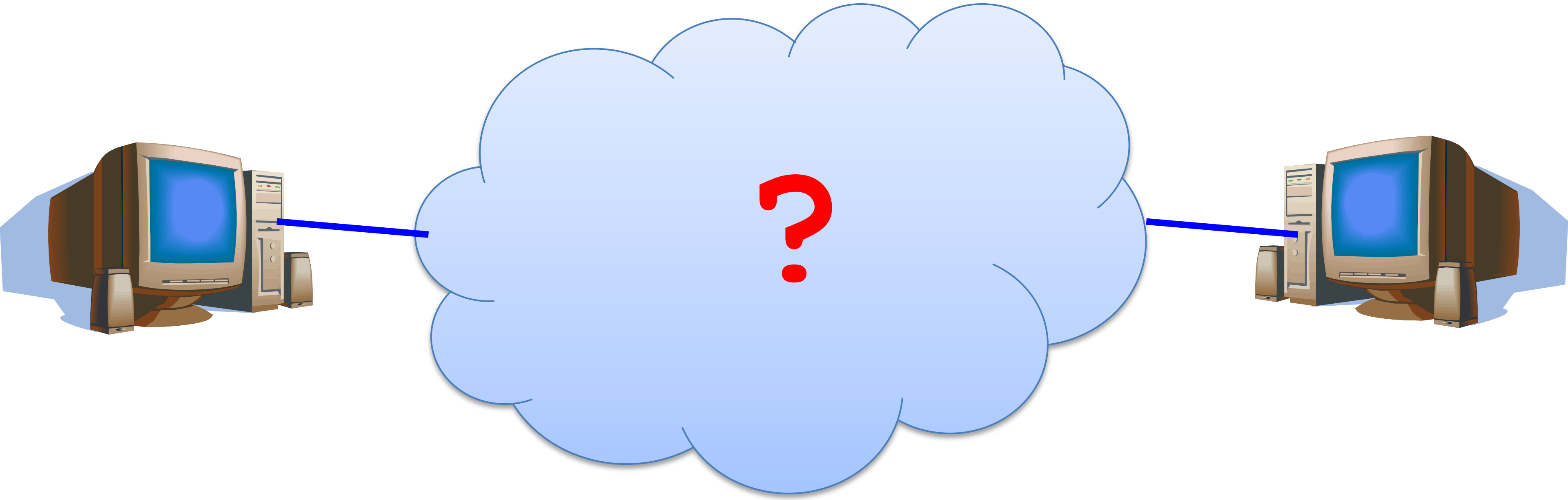


Increase in load that results in a *decrease* in useful work done.

Ways to Deal With Congestion

- Ignore the problem
  - Many dropped (and retransmitted) packets
  - Can cause congestion collapse
- Reservations, like in circuit switching
  - Pre-arrange bandwidth allocations
  - Requires negotiation before sending packets
- Pricing
  - Don't drop packets for the high-bidders
  - Requires a payment model, and low-bidders still dropped
- Dynamic adjustment (TCP)
  - Every sender infers the level of congestion
  - Each adapts its sending rate "for the greater good"

Detect and Respond to Congestion



- What does the end host see?
- What can the end host change?

Many Important Questions

- How does the sender know there is congestion?
  - Explicit feedback from the network?
  - Inference based on network performance?
- How should the sender adapt?
  - Explicit sending rate computed by the network?
  - End host coordinates with other hosts?
  - End host thinks globally but acts locally?
- What is the performance objective?
  - Maximizing goodput, even if some users suffer more?
  - Fairness? (Whatever *that* means!)
- How fast should new TCP senders send?

Detecting Congestion

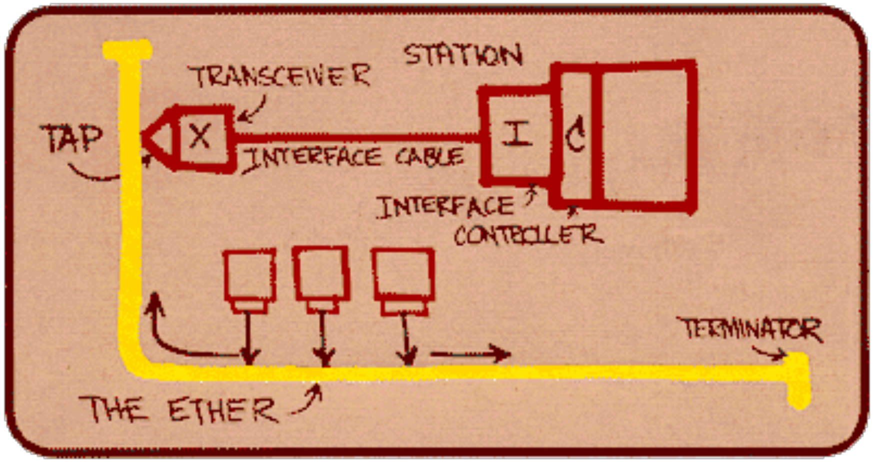
- Link layer
  - Carrier sense multiple access
  - Seeing your own frame collide with others
- Network layer
  - Observing end-to-end performance
  - Packet delay or loss over the path

Responding to Congestion

- Upon detecting congestion
  - Decrease the sending rate
- But, what if conditions change?
  - If more bandwidth becomes available,
  - … unfortunate to keep sending at a low rate
- Upon *not* detecting congestion
  - Increase sending rate, a little at a time
  - See if packets get through

Ethernet Back-off Mechanism

- **Carrier sense:**
  - Wait for link to be idle
  - If idle, start sending
  - If not, wait until idle

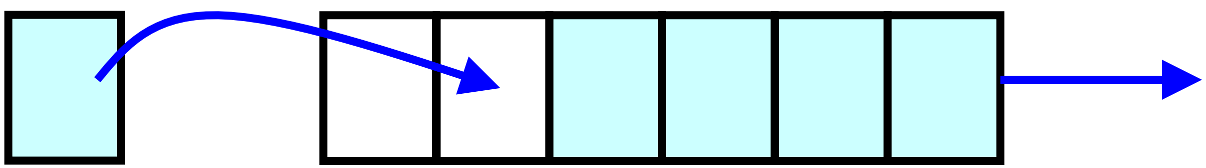


- **Collision detection:** listen while transmitting
  - If collision: abort transmission, and send jam signal
- **Exponential back-off:** wait before retransmitting
  - Wait random time, exponentially larger per retry

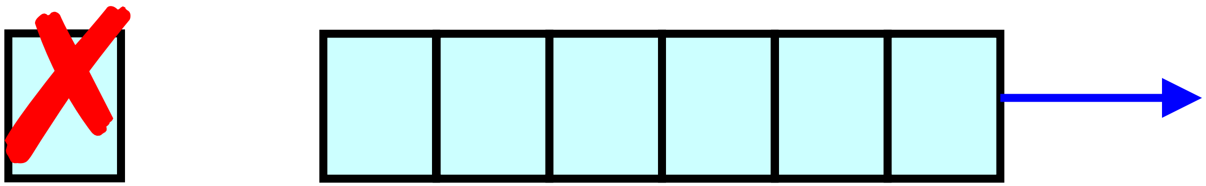
TCP Congestion Control

Where Congestion Happens: Links

- Simple resource allocation: FIFO queue & drop-tail
- Access to the bandwidth: first-in first-out queue
  - Packets transmitted in the order they arrive



- Access to the buffer space: drop-tail queuing
  - If the queue is full, drop the incoming packet



How it Looks to the End Host

- Delay: Packet experiences high delay
- Loss: Packet gets dropped along path

- How does TCP sender learn this?
  - Delay: Round-trip time estimate
  - Loss: Timeout and/or duplicate acknowledgments

Congestion Control Algorithms

Loss-based	Delay and rate-based TCPs	A hybrid delay and loss based TCPs	Forward Error Correction
* TCP-Reno/New Reno * TCP-SACK * BIC/CUBIC * HSTCP * STCP	* Vegas, FAST * Westwood * WTCP	* H-TCP * TCP-Illinois * TCP-Africa * Yeah-TCP * Compound-TCP	* LT-TCP * Maelstrom

What Can the End Host Do?

- Upon detecting congestion (well, packet loss)
  - Decrease the sending rate
  - End host does its part to alleviate the congestion
- But, what if conditions change?
  - If bandwidth becomes available, unfortunate if remains sending at low rate
- Upon *not* detecting congestion
  - Increase sending rate, a little at a time
  - And see if packets are successfully delivered

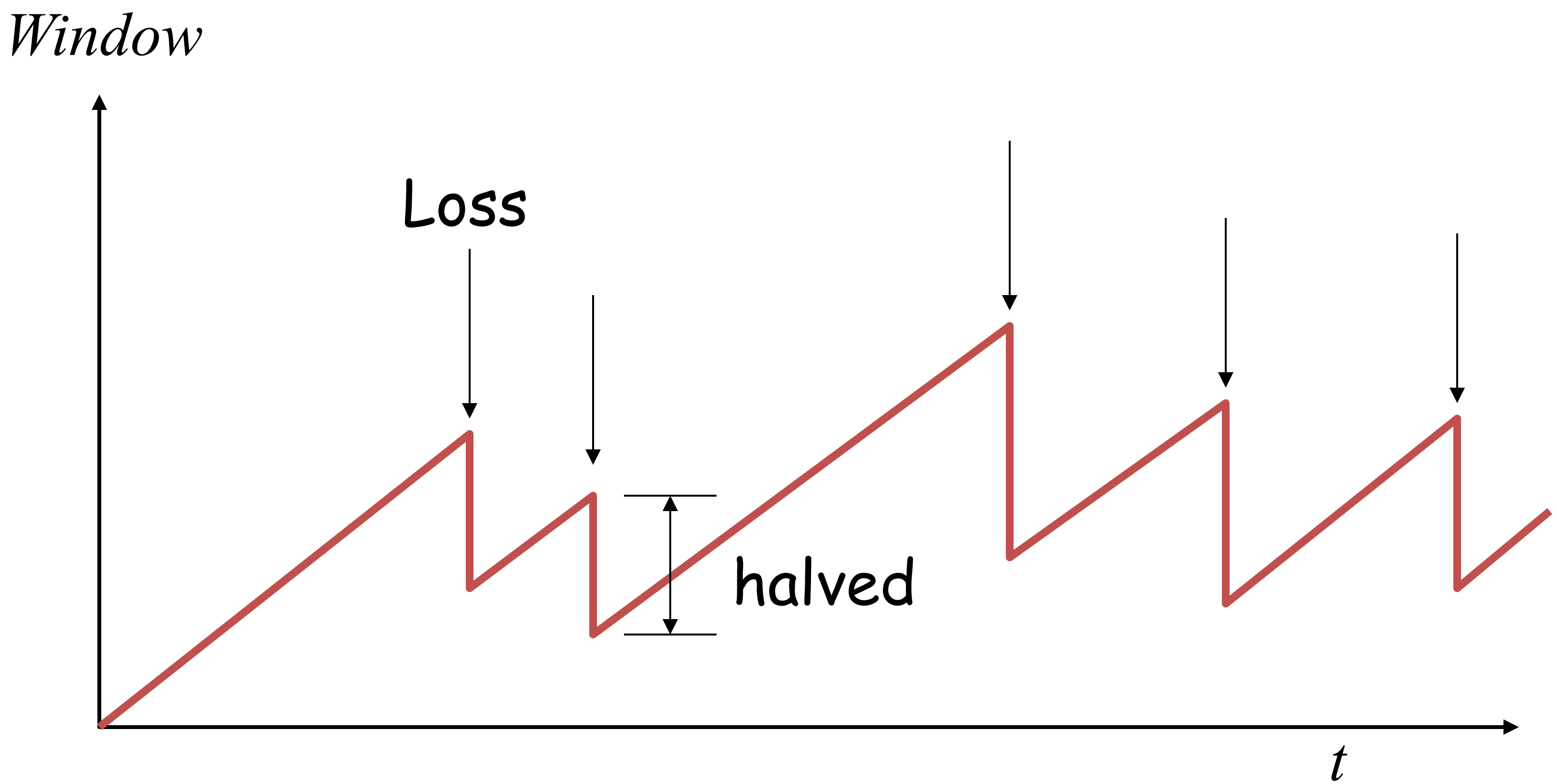
TCP Congestion Window

- Each TCP sender maintains a congestion window
  - Max number of bytes to have in transit (not yet ACK'd)
- Adapting the congestion window
  - Decrease upon losing a packet: backing off
  - Increase upon success: optimistically exploring
  - Always struggling to find right transfer rate
- Tradeoff
  - Pro: avoids needing explicit network feedback
  - Con: continually under- and over-shoots "right" rate

Additive Increase, Multiplicative Decrease (AIMD)

- How much to adapt?
  - Additive increase: On success of last window of data, increase window by 1 Max Segment Size (MSS)
  - Multiplicative decrease: On loss of packet, divide congestion window in half
- Much quicker to slow than speed up!
  - Over-sized windows (causing loss) are much worse than under-sized windows (causing lower thruput)
  - AIMD: A necessary condition for stability of TCP

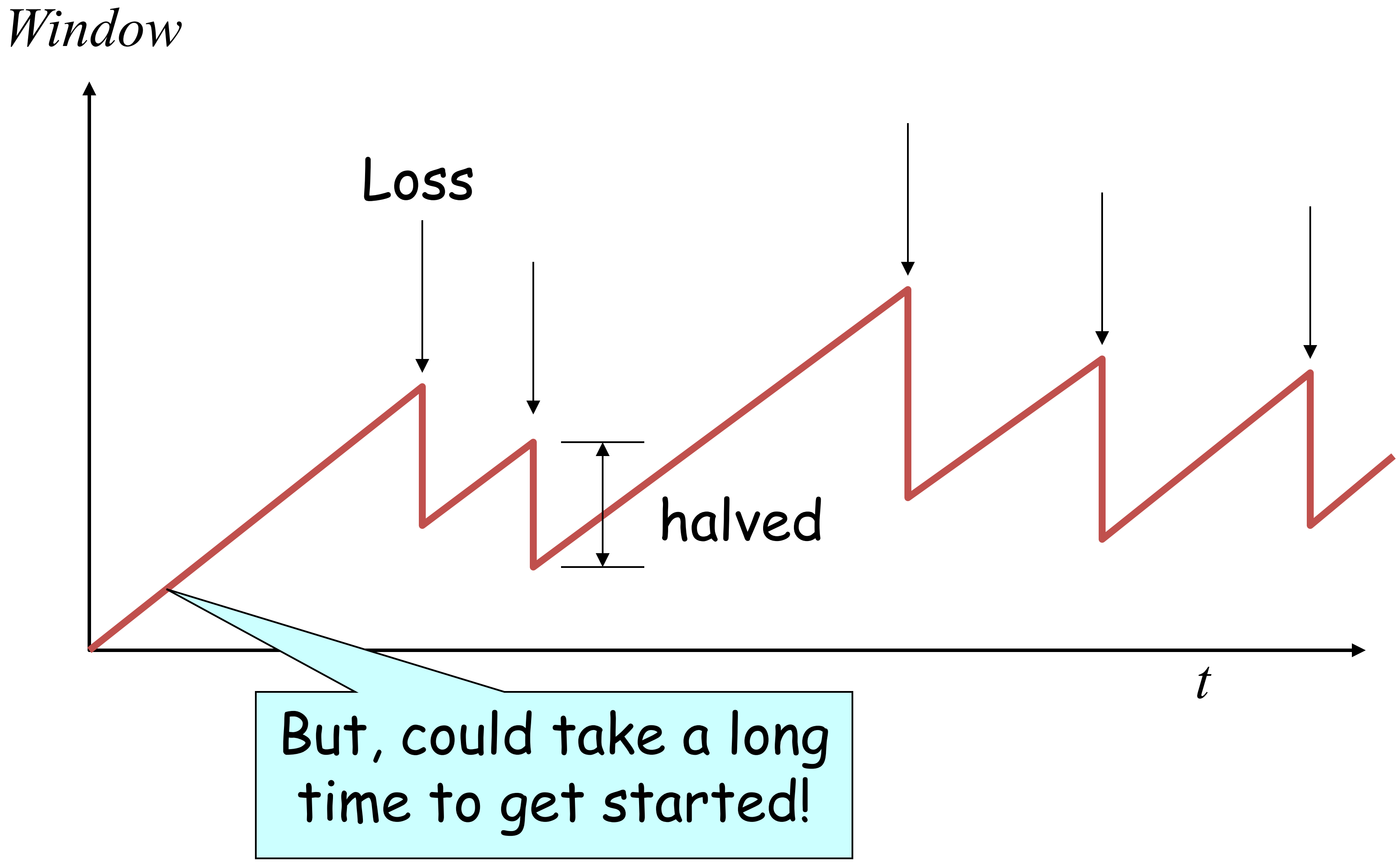
Leads to the TCP "Sawtooth"



Starting a New Flow

How Should a New Flow Start?

Start slow (a small CWND) to avoid overloading network

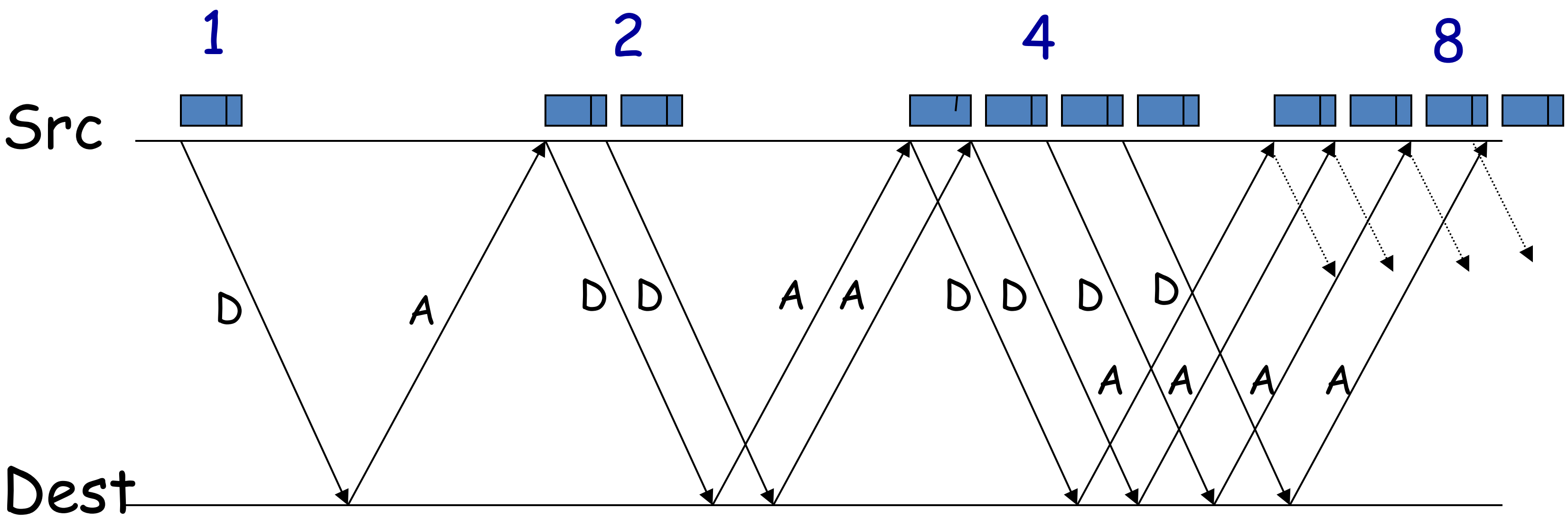


"Slow Start" Phase

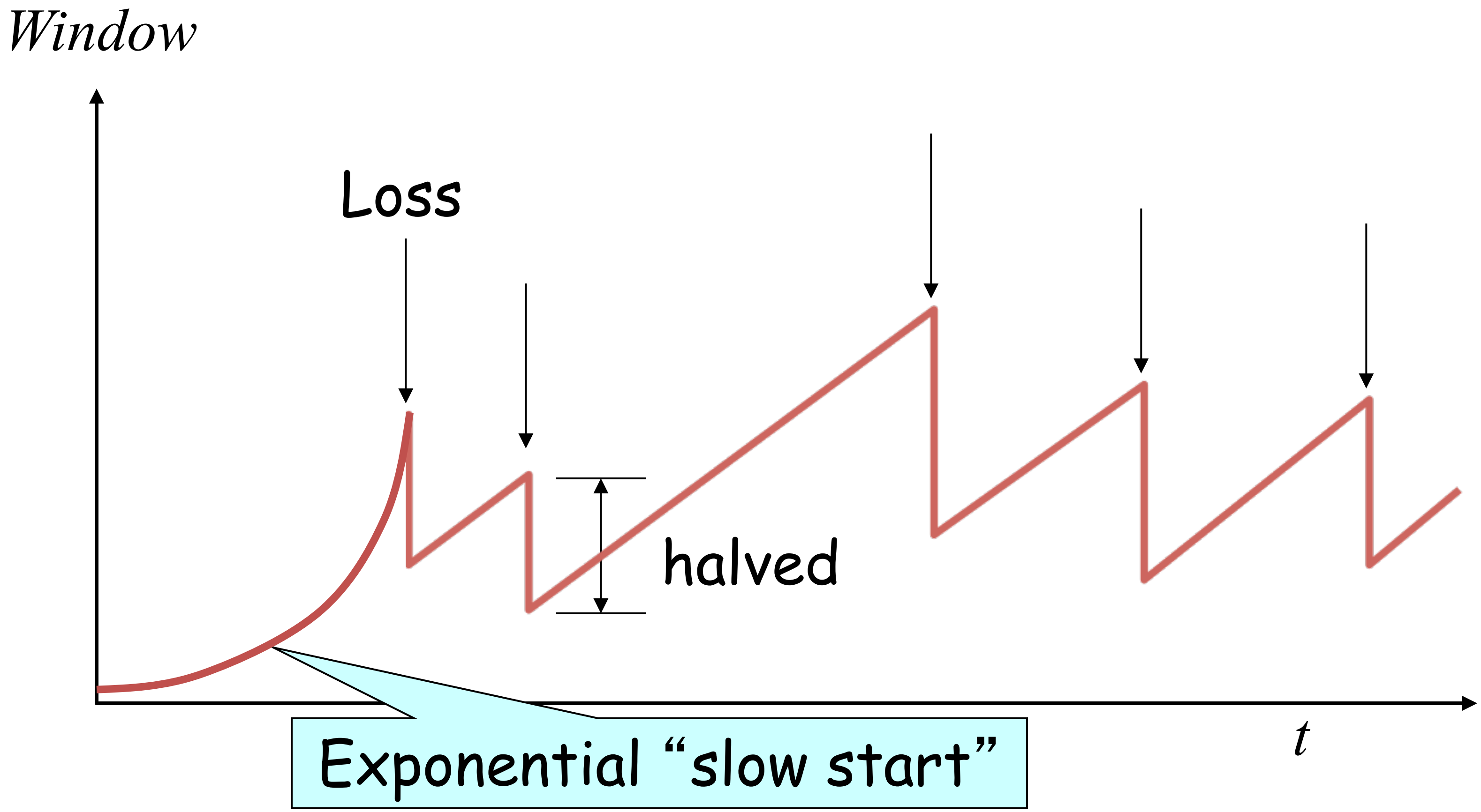
- Start with a small congestion window
  - Initially, CWND is 1 MSS
  - So, initial sending rate is MSS / RTT
- Could be pretty wasteful
  - Might be much less than actual bandwidth
  - Linear increase takes a long time to accelerate
- Slow-start phase (really "fast start")
  - Sender starts at a slow rate (hence the name)
  - … but increases rate exponentially until the first loss

Slow Start in Action

Double CWND per round-trip time



Slow Start and the TCP Sawtooth

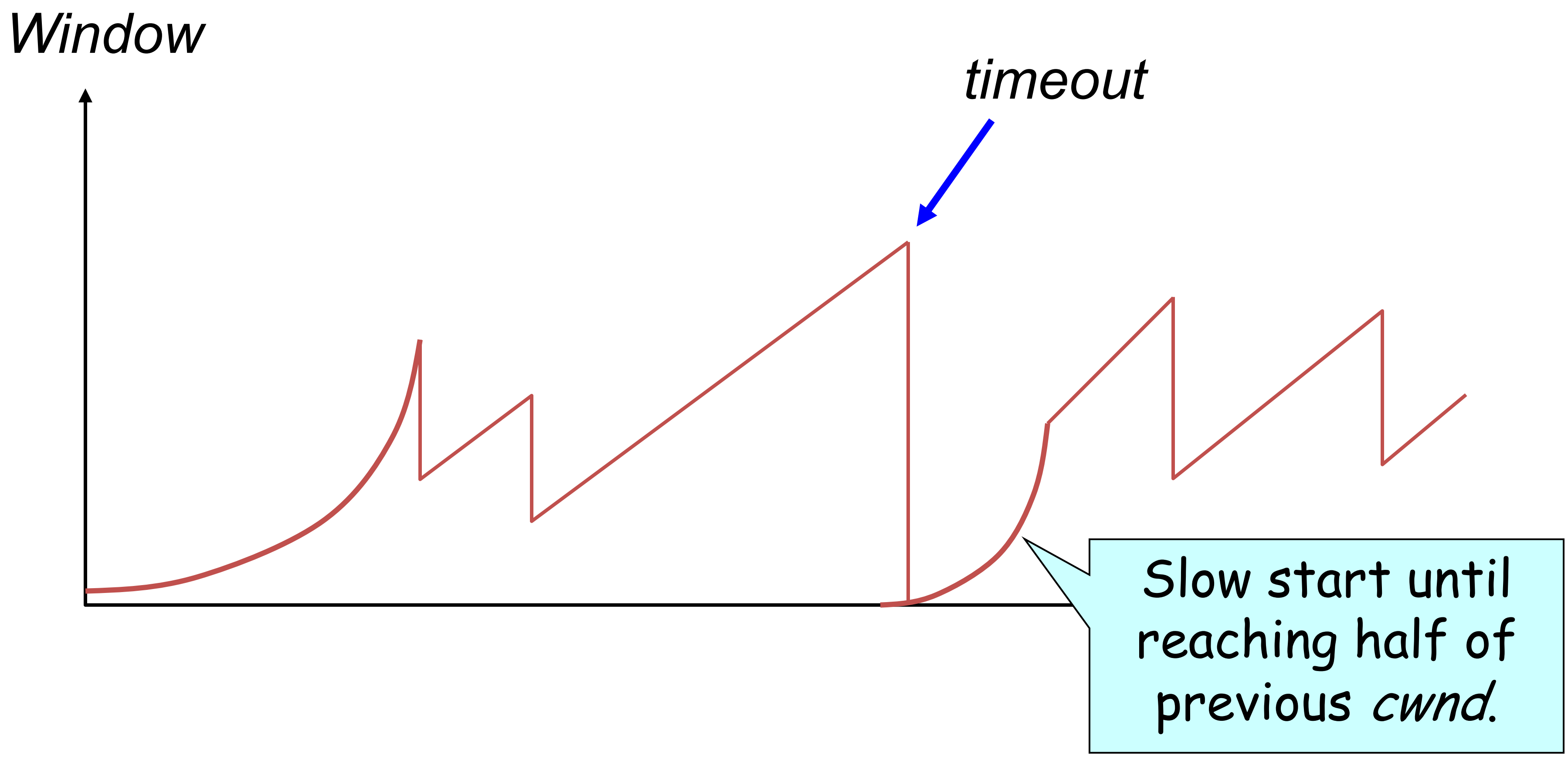


- So-called because TCP originally had no congestion control
  - Source would start by sending an entire receiver window
    - Led to congestion collapse!

Two Kinds of Loss in TCP

- Timeout vs. Triple Duplicate ACK
  - Which suggests network is in worse shape?
- Timeout
  - Packet n is lost and detected via a timeout
    - When? n is last packet in window, or all packets in flight lost
  - After timeout, blasting entire CWND would cause another burst
  - Better to start over with a low CWND
- Triple duplicate ACK
  - Packet n is lost, but packets n+1, n+2, etc. arrive
    - How detected? Multiple ACKs that receiver waiting for n
    - When? Later packets after n received
  - After triple duplicate ACK, sender quickly resends packet n
  - Do a multiplicative decrease and keep going

Repeating Slow Start After Timeout



Slow-start restart: Go back to CWND of 1, but take advantage of knowing the previous value of CWND.

Repeating Slow Start After Idle Period

- Suppose a TCP connection goes idle for a while
- Eventually, the network conditions change
  - Maybe many more flows are traversing the link
- Dangerous to start transmitting at the old rate
  - Previously-idle TCP sender might blast network
  - … causing excessive congestion and packet loss
- So, some TCP implementations repeat slow start
  - Slow-start restart after an idle period

Receiver Window vs. Congestion Window

- Flow control
  - Keep a *fast sender* from overwhelming *a slow receiver*
- Congestion control
  - Keep a *set of senders* from overloading the *network*

- Different concepts, but similar mechanisms
  - TCP flow control: receiver window
  - TCP congestion control: congestion window
  - Sender TCP window =

    min { congestion window, receiver window }

Sources of poor TCP performance

- The below conditions *may* primarily result in:

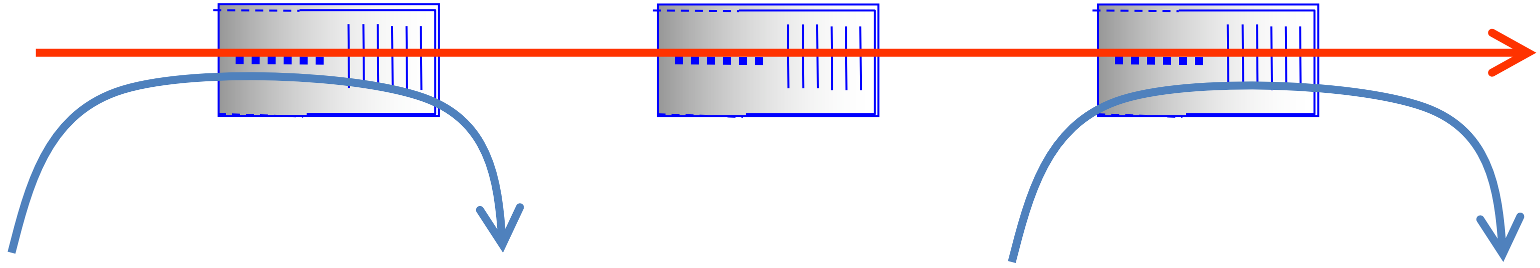
(A) Higher pkt latency (B) Greater loss (C) Lower throughput

1. Larger buffers in routers
2. Smaller buffers in routers
3. Smaller buffers on end-hosts
4. Slow application receivers

Fairness

TCP Achieves Some Notion of Fairness

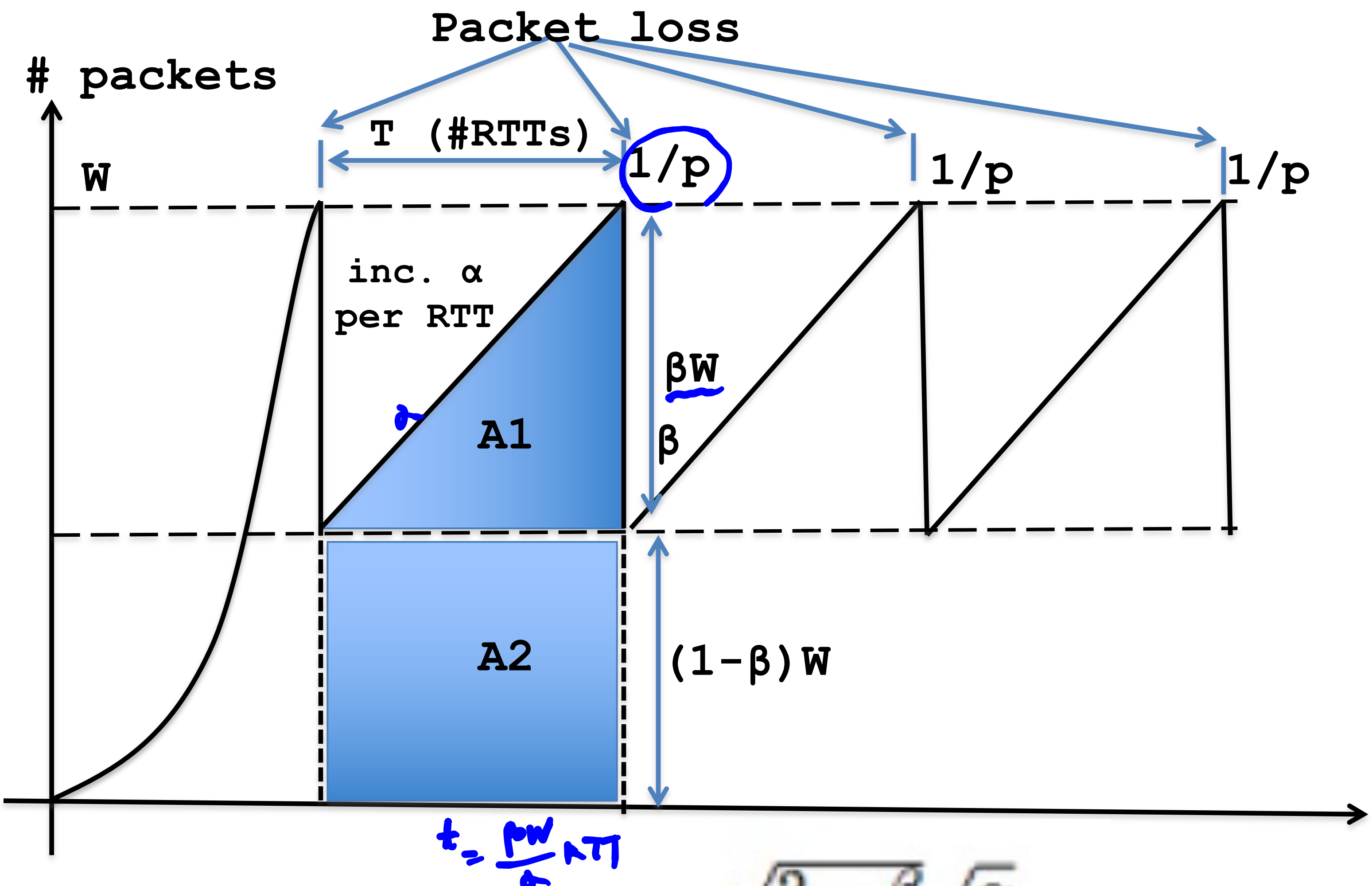
- Effective utilization is not only goal
  - We also want to be *fair* to various flows
  - … but what does *that* mean?
- Simple definition: equal shares of the bandwidth
  - N flows that each get 1/N of the bandwidth?
  - But, what if flows traverse different paths?
  - Result: bandwidth shared in proportion to RTT



What About Cheating?

- Some folks are more fair than others
  - Running multiple TCP connections in parallel (BitTorrent)
  - Modifying the TCP implementation in the OS
    - Some cloud services start TCP at > 1 MSS
  - Use the User Datagram Protocol
- What is the impact
  - Good guys slow down to make room for you
  - You get an unfair share of the bandwidth
- Possible solutions?
  - Routers detect cheating and drop excess packets?
  - Per user/customer failness?
  - Peer pressure?

AIMD (α, β) Throughput



$$E\{W_{aimd}\} = \frac{\sqrt{2-\beta}\sqrt{\alpha}}{\sqrt{2\beta}RTT\sqrt{p}}$$

AIMD (α, β) Throughput (Cont'd)

W: The window size immediately before a loss event
α: Addictive increase factor. The window size increases α every RTT round
β: The multiplicative decrease factor. After a loss event,
the window size is reduced to (1-β)W
RTT: The RTT of a flow
P: Loss event rate

The total number of packets sent in a loss epoch is A1+A2, then 1/p = A1+A2

$$\frac{1}{p} = A = \left(W\beta\frac{W\beta}{\alpha}\right)\frac{1}{2} + \left(W(1-\beta)\frac{W\beta}{\alpha}\right) = \frac{W^2(2-\beta)\beta}{2\alpha}$$

By solving the above equation for W will be $W = \frac{\sqrt{2\alpha}}{\sqrt{\beta(2-\beta)p}}$

The number of RTTs (T) during one loss epoch is $T = \frac{W\beta}{\alpha}RTT$

The average window size (throughput) is A/T

$$E\{W_{aimd}\} = \frac{A}{T} = \frac{\frac{W^2(2-\beta)\beta}{2\alpha}}{\frac{W\beta}{\alpha}RTT} = \frac{\sqrt{2-\beta}\sqrt{\alpha}}{\sqrt{2\beta}RTT\sqrt{p}} = AIMD(\alpha, \beta)$$

TCP Problem

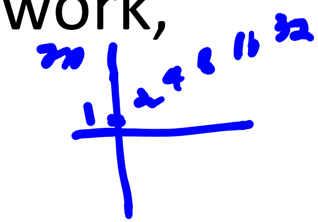
1. Suppose that there are two flows with 10ms and 100ms RTT respectively. What is the throughput ratio between these two flows?

   (A) 1:2 (B) 2:1 (C) 4:1 (D) 1:4 (E) 8:1 (F) 8:1 (G) 1:10 (H) 10:1

TCP Problem

- 1 MSS = 1KB
- Max capacity of link: 200 KBps
- RTT = 100ms
- New TCP flow starting, no other traffic in network, assume no queues in network

BDP = 200 KBps × 0.1
= 20KB



1. About what is cwnd at time of first packet loss?

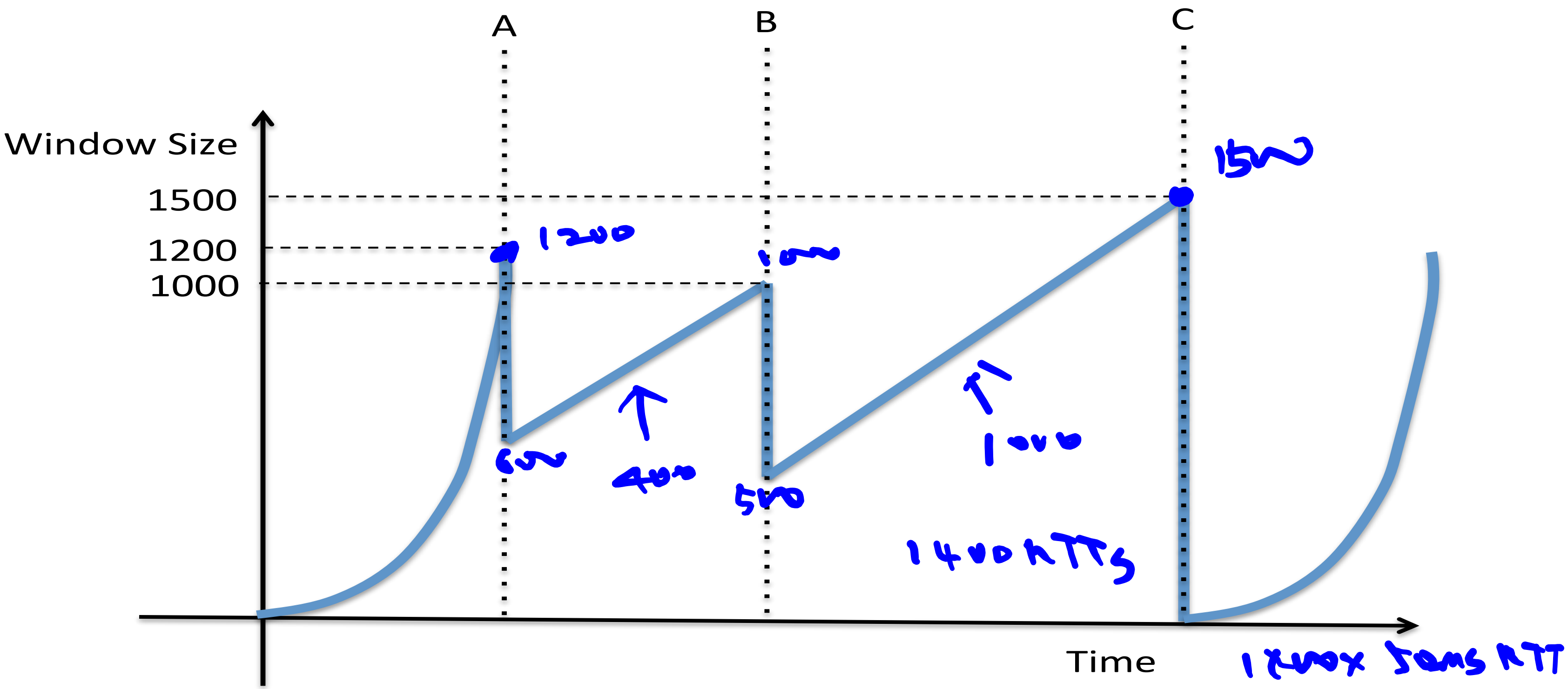
   (A) 16 pkts (B) 32 KB (C) 100 KB (D) 200 KB

2. About how long until sender discovers first loss?

   (A) 400 ms (B) 600 ms (C) 1s (D) 1.6s

TCP Problem (Cont.)

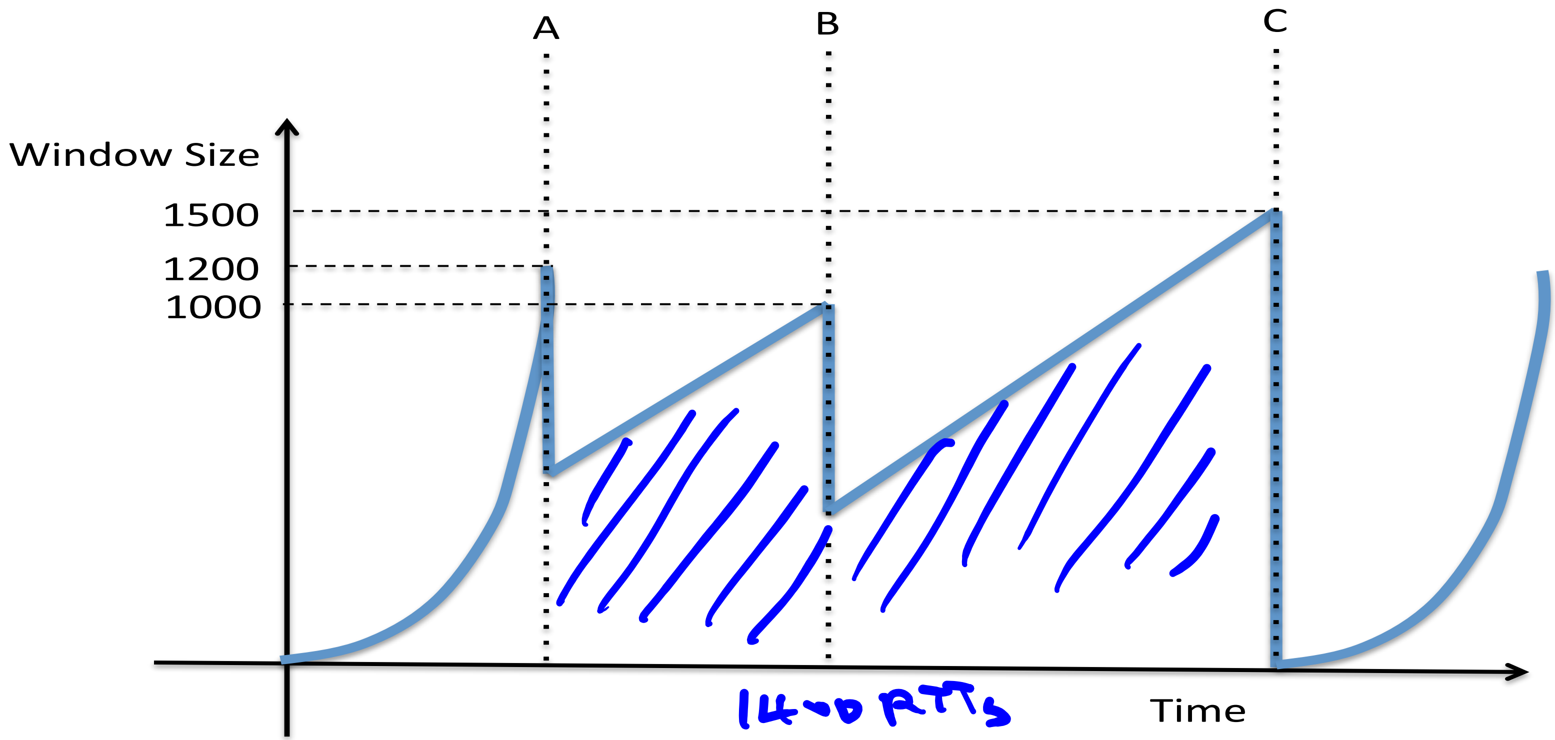
AIMD(1, 0.5)



1. Assume that the round-trip-time between the sender and the receiver is 50ms. How much time will it take for the sender to reach to event C from event A?

(A) 10 secs (B) 30 secs (C) 60 secs (D) 70 secs

TCP Problem (Cont.)



~~1.~~ Suppose that one RTT is 100ms and the MSS size is 1000. What is the throughput between events A and C?

(A) 17.4 Mbps (B) 24.2 Mbps (C) 35.4 Mbps (D) 52.8 Mbps

Conclusions

- Congestion is inevitable
  - Internet does not reserve resources in advance
  - TCP actively tries to push the envelope
- Congestion can be handled
  - Additive increase, multiplicative decrease
  - Slow start and slow-start restart
- Fundamental tensions
  - Feedback from the network?
  - Enforcement of "TCP friendly" behavior?