

CSC 458/2209 – Computer Networks

Handout # 13: Congestion Control



Professor Yashar Ganjali
Department of Computer Science
University of Toronto

yganjali@cs.toronto.edu

<http://www.cs.toronto.edu/~yganjali>

Announcements

- Problem Set 2
 - Will be posted early next week
 - **Due: Nov. 15th at 5pm**
 - Submit electronically as ps2.pdf

- Programming Assignment 2
 - Will be posted later next week
 - New assignment to avoid relying on first assignment
 - Simpler, and completed individually.
 - **Due: Nov. 22nd at 5pm**

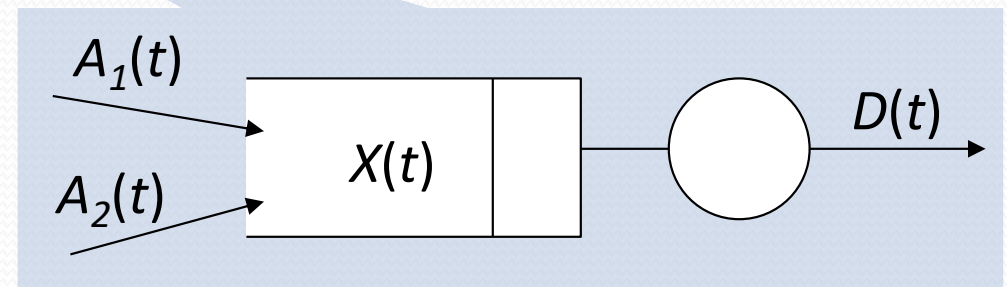
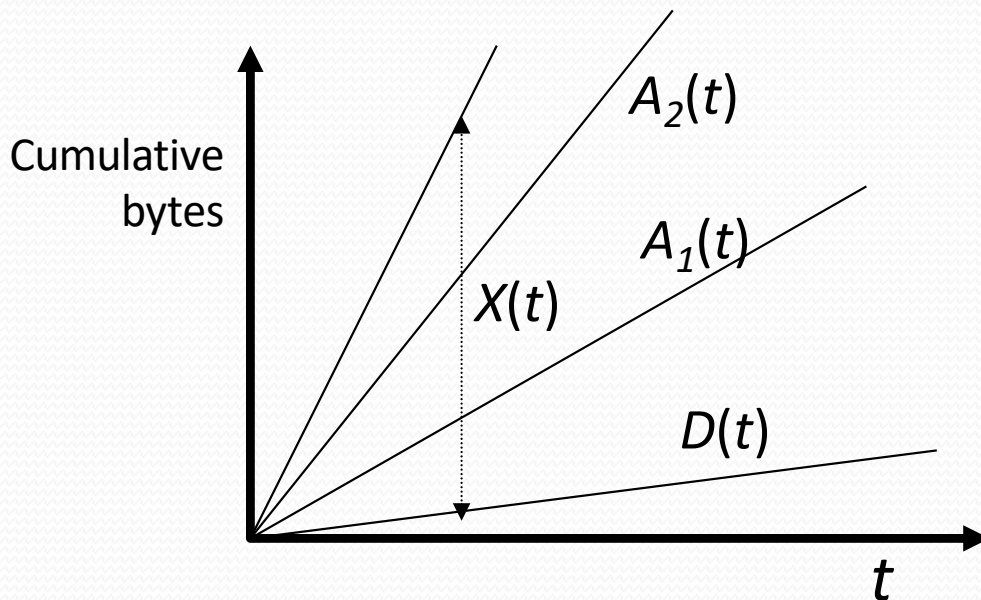
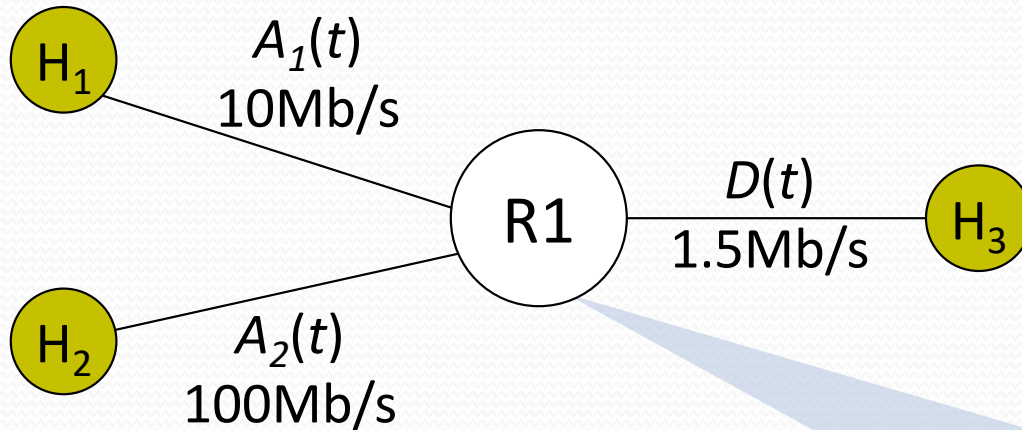
Announcements

- Problem Set 1
 - Marks will be posted by the end of the week.
 - Please contact Nafiseh and Kasra for remark requests.
- Programming Assignment 1
 - Marks will be posted next week
- Midterm
 - Marked next week
- Tutorials:
 - No tutorial this week
 - Next week: PS2 review and sample problems

Today's Lecture

- Principles of congestion control
 - Learning that congestion is occurring
 - Adapting to alleviate the congestion
- TCP congestion control
 - Additive-increase, multiplicative-decrease
 - Slow start and slow-start restart
- Related TCP mechanisms
 - Nagle's algorithm and delayed acknowledgments
- Active Queue Management (AQM)
 - Random Early Detection (RED)
 - Explicit Congestion Notification (ECN)

What is Congestion?



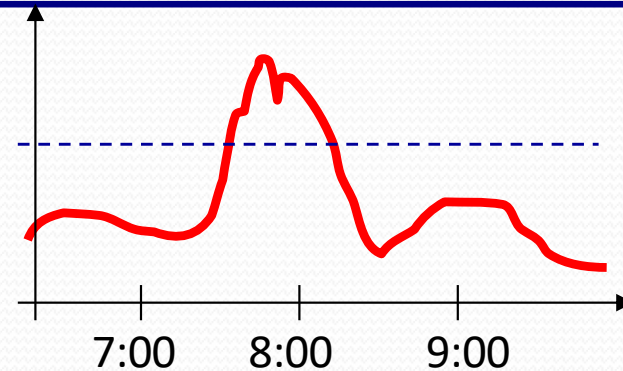
Flow Control vs. Congestion Control

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

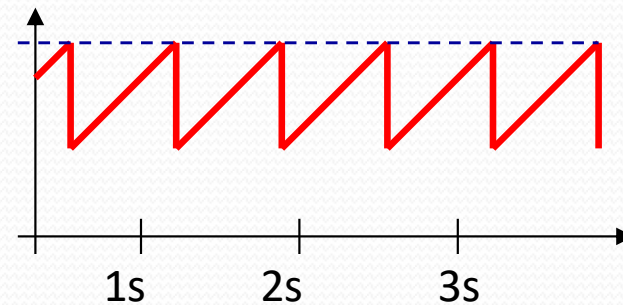
- Different concepts, but similar mechanisms
 - TCP flow control: receiver window
 - TCP congestion control: congestion window
 - TCP window: $\min\{\text{congestion window, receiver window}\}$

Time Scales of Congestion

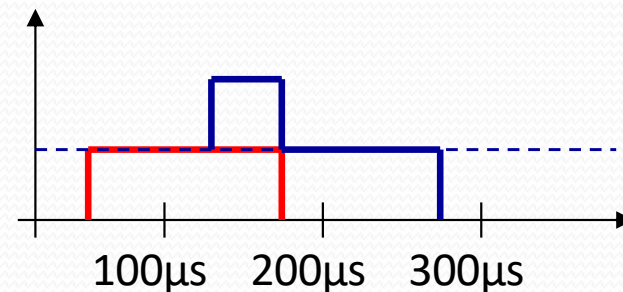
Too many users using a link during a peak hour



TCP flows filling up all available bandwidth

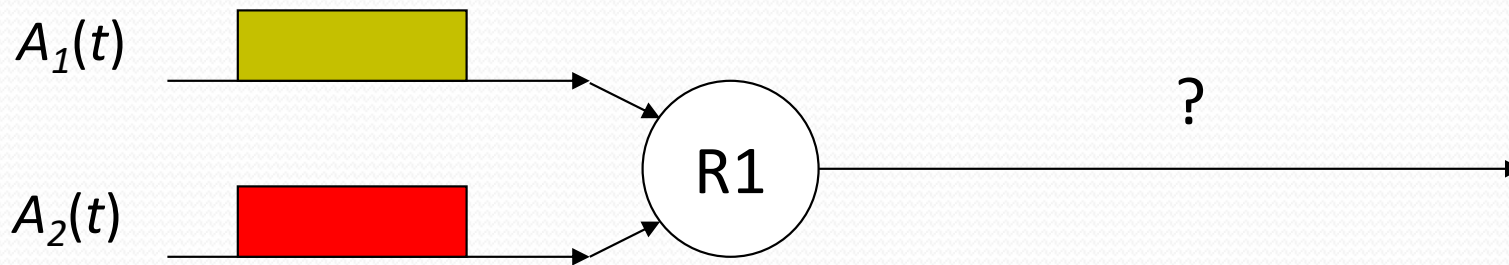


Two packets colliding at a router – also referred to as contention



Dealing with Congestion

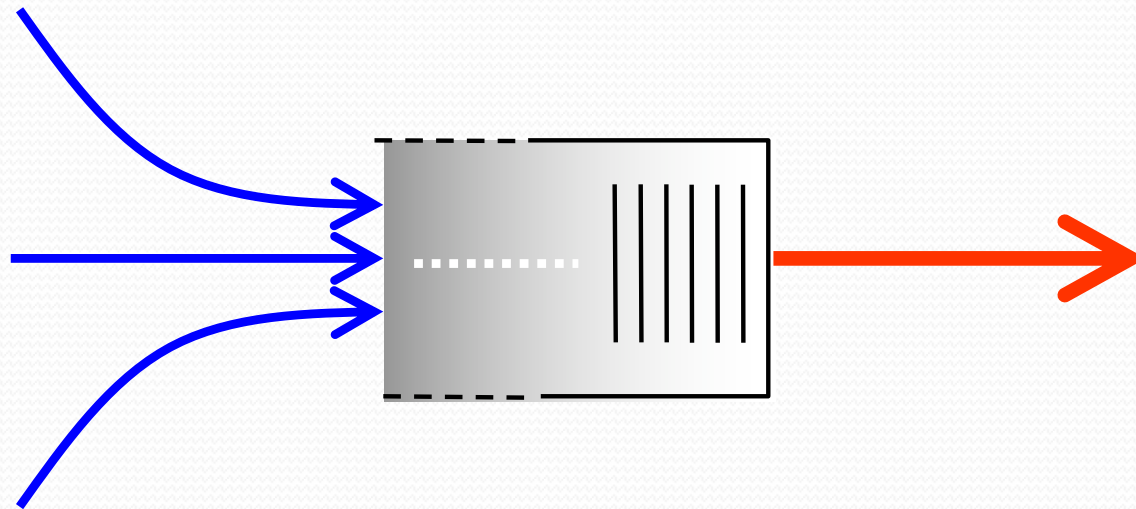
Example: two flows arriving at a router



Strategy	
Drop one of the flows	<p>The diagram shows a single horizontal line representing a transmission path. A yellow bar is positioned above the line, and an arrow points to the right from the end of the line, indicating that only the first flow is being transmitted.</p>
Buffer one flow until the other has departed, then send it	<p>The diagram shows a single horizontal line representing a transmission path. A yellow bar is positioned above the line, followed by a red bar. An arrow points to the right from the end of the line, indicating that the flows are being transmitted one after the other.</p>
Re-Schedule one of the two flows for a later time	<p>The diagram shows a single horizontal line representing a transmission path. A yellow bar is positioned above the line, and a red bar is positioned below the line, further to the right. An arrow points to the right from the end of the line, indicating that the flows are being transmitted at different times.</p>
Ask both flows to reduce their rates	<p>The diagram shows a single horizontal line representing a transmission path. A yellow bar and a red bar are positioned above and below the line, respectively, and are shorter than in the previous diagrams. An arrow points to the right from the end of the line, indicating that both flows are being transmitted at reduced rates.</p>

Congestion is Unavoidable

- Two packets arrive at the same time
 - The node can only transmit one
 - ... and either buffer or drop the other
- If many packets arrive in a short period of time
 - The node cannot keep up with the arriving traffic
 - ... and the buffer may eventually overflow



Arguably Congestion is Good!

- We use packet switching because it makes efficient use of the links. Therefore, buffers in the routers are frequently occupied.
- If buffers are always empty, delay is low, but our usage of the network is low.
- If buffers are always occupied, delay is high, but we are using the network more efficiently.
- So how much congestion is too much?

Congestion Collapse

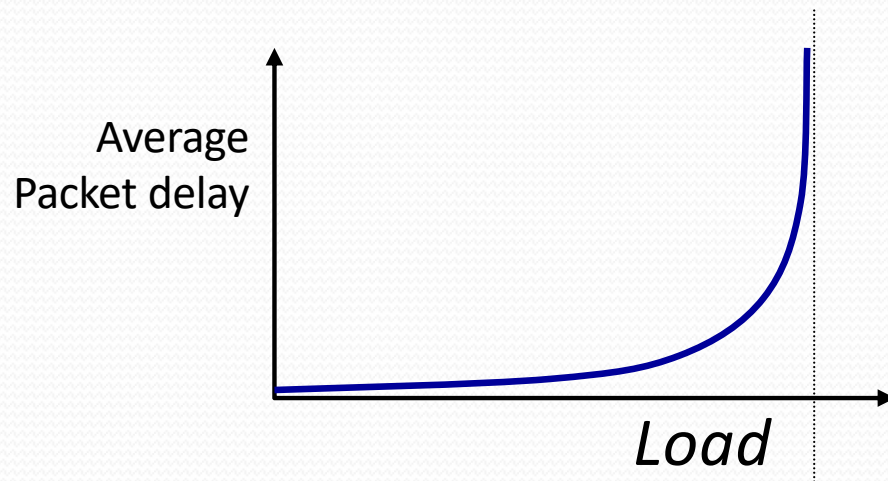
- **Definition:** Increase in network load results in a decrease of useful work done
- Many possible causes
 - Spurious retransmissions of packets still in flight
 - Classical congestion collapse
 - Solution: better timers and TCP congestion control
 - Undelivered packets
 - Packets consume resources and are dropped elsewhere in network
 - Solution: congestion control for ALL traffic

What Do We Want, Really?

- High throughput
 - Throughput: measured performance of a system
 - E.g., number of bits/second of data that get through
- Low delay
 - Delay: time required to deliver a packet or message
 - E.g., number of msec to deliver a packet
- These two metrics are sometimes at odds
 - E.g., suppose you drive a link as hard as possible
 - ... then, throughput will be high, but delay will be, too

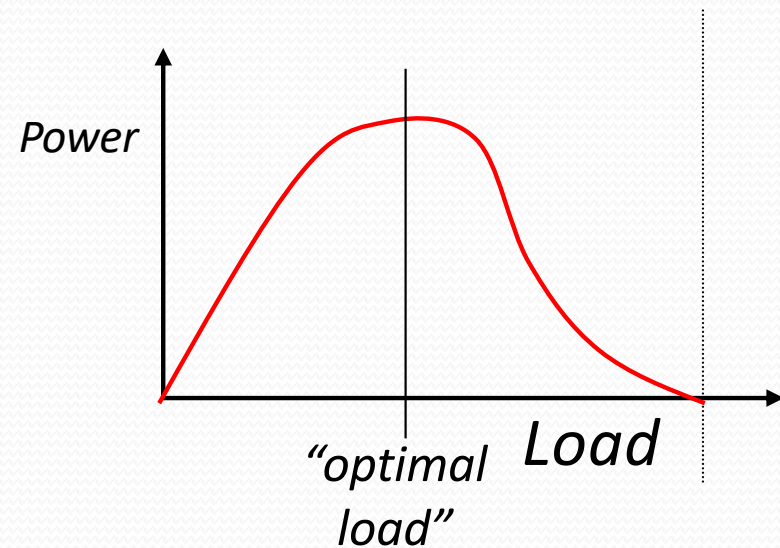
Load, Delay, and Power

Typical behavior of queuing systems with random arrivals:



A simple metric of how well the network is performing:

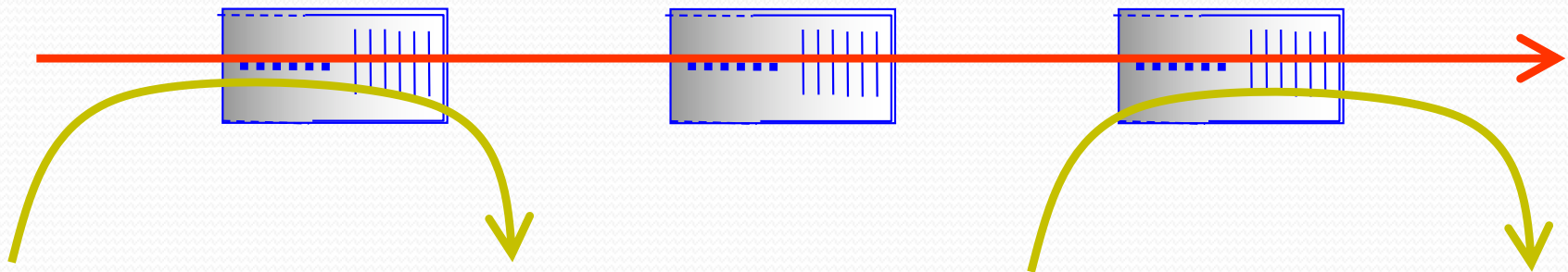
$$Power = \frac{Load}{Delay}$$



Goal: maximize power

Fairness

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

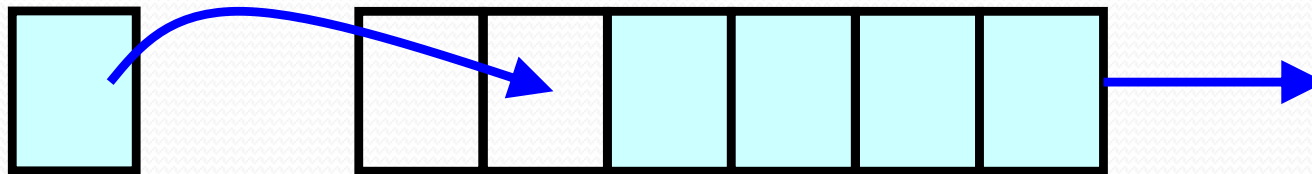


Resource Allocation vs. Congestion Control

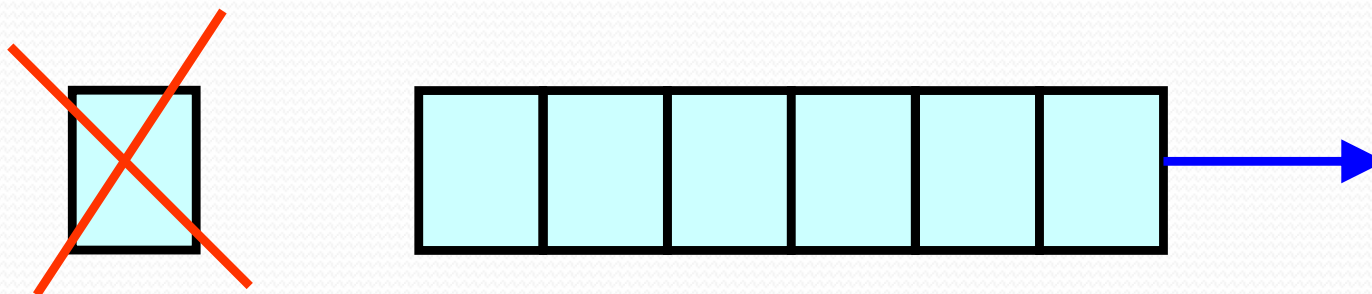
- Resource allocation
 - How nodes meet competing demands for resources
 - E.g., link bandwidth and buffer space
 - When to say no, and to whom
- Congestion control
 - How nodes prevent or respond to overload conditions
 - E.g., persuade hosts to stop sending, or slow down
 - Typically has notions of fairness (i.e., sharing the pain)

Simple Resource Allocation

- Simplest approach: FIFO queue and drop-tail
- Link bandwidth: first-in first-out queue
 - Packets transmitted in the order they arrive



- Buffer space: drop-tail queuing
 - If the queue is full, drop the incoming packet



Simple Congestion Detection

- Packet loss
 - Packet gets dropped along the way
- Packet delay
 - Packet experiences high delay
- How does TCP sender learn this?
 - Loss
 - Timeout
 - Triple-duplicate acknowledgment
 - Delay
 - Round-trip time estimate

Options for Congestion Control

- Implemented by host versus network
- Reservation-based, versus feedback-based
- Window-based versus rate-based.

TCP Congestion Control

- TCP implements host-based, feedback-based, window-based congestion control.
- TCP sources attempts to determine how much capacity is available
- TCP sends packets, then reacts to observable events (loss).

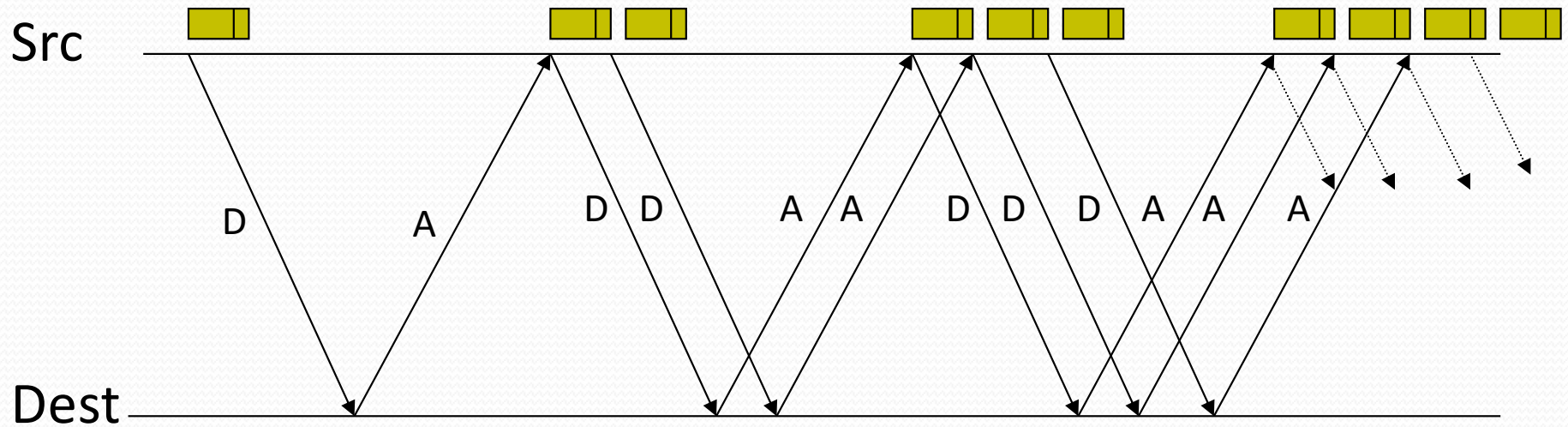
Idea of TCP Congestion Control

- Each source determines the available capacity
 - ... so it knows how many packets to have in transit
- Congestion window
 - Maximum # of unacknowledged bytes to have in transit
 - The congestion-control equivalent of receiver window
 - $\text{MaxWindow} = \min\{\text{congestion window}, \text{receiver window}\}$
 - Send at the rate of the slowest component: receiver or network
- Adapting the congestion window
 - Decrease upon losing a packet: backing off
 - Increase upon success: optimistically exploring

Additive Increase, Multiplicative Decrease

- How much to increase and decrease?
 - Increase linearly, decrease multiplicatively
 - A necessary condition for stability of TCP
 - Consequences of over-sized window are much worse than having an under-sized window
 - Over-sized window: packets dropped and retransmitted
 - Under-sized window: somewhat lower throughput
- Multiplicative decrease
 - On loss of packet, divide congestion window in half
- Additive increase
 - On success for last window of data, increase linearly

Additive Increase

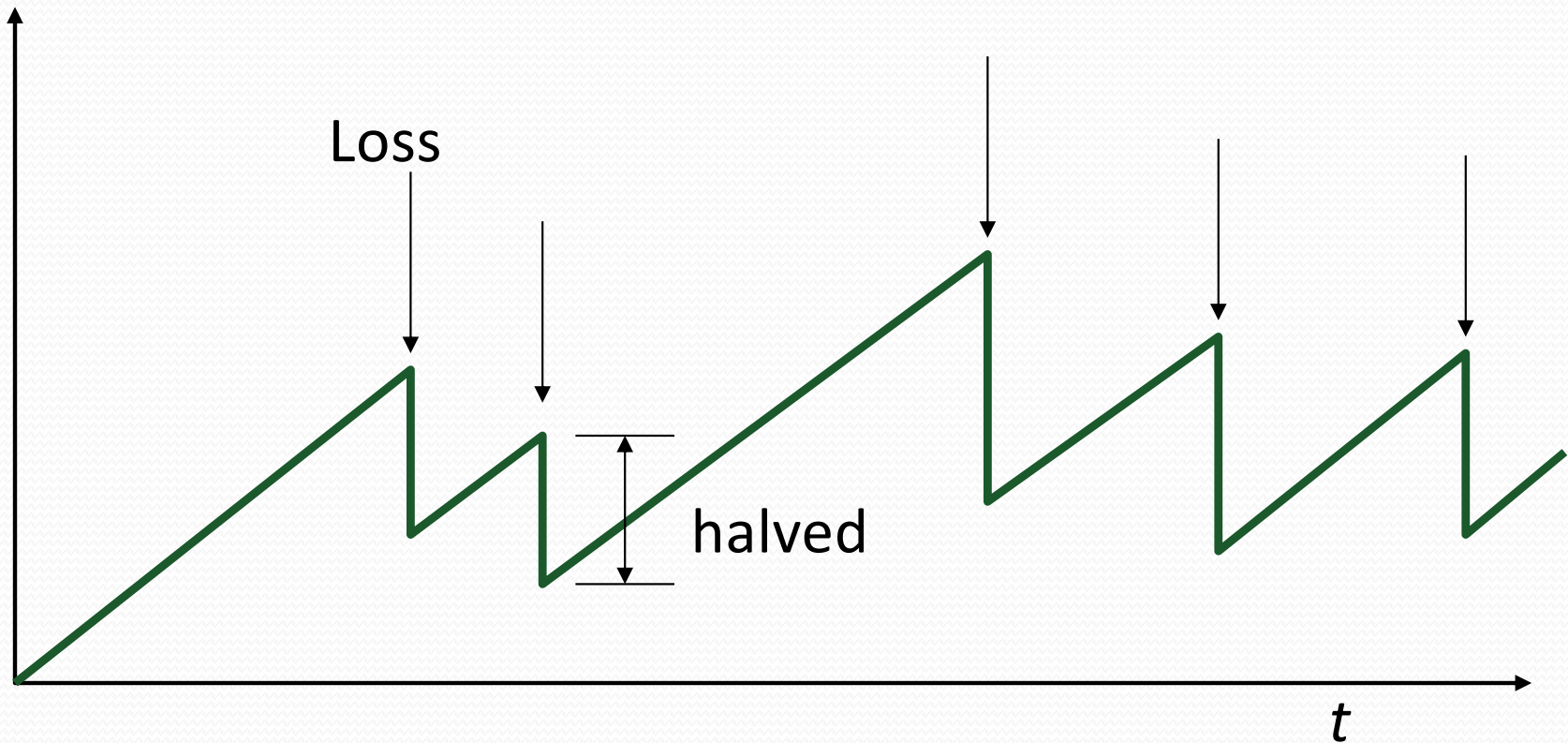


Actually, TCP uses bytes, not segments to count:
When ACK is received:

$$cwnd_{+} = MSS \left(\frac{MSS}{cwnd} \right)$$

Leads to the TCP “Sawtooth”

Window

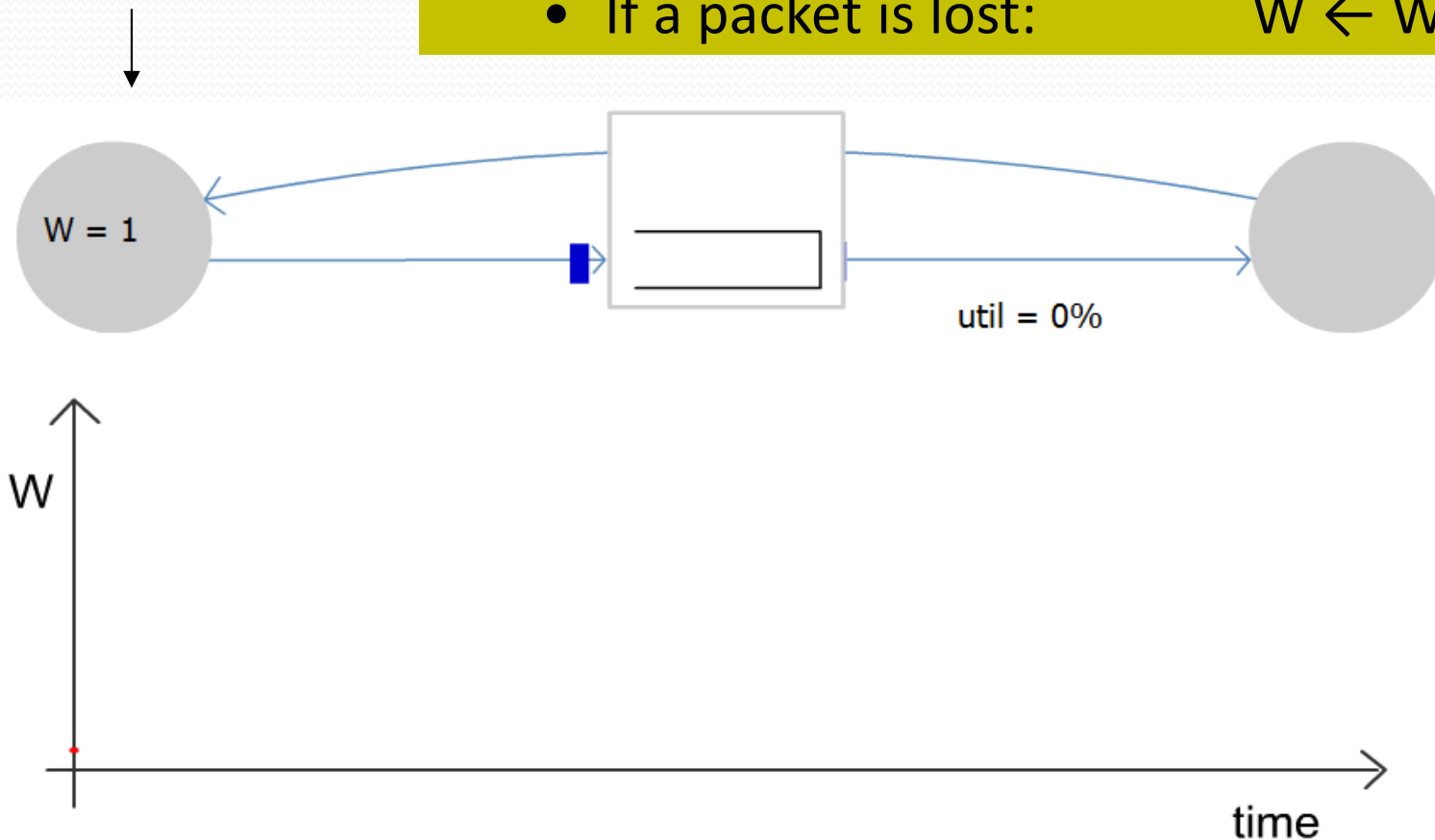


Congestion Window Evolution

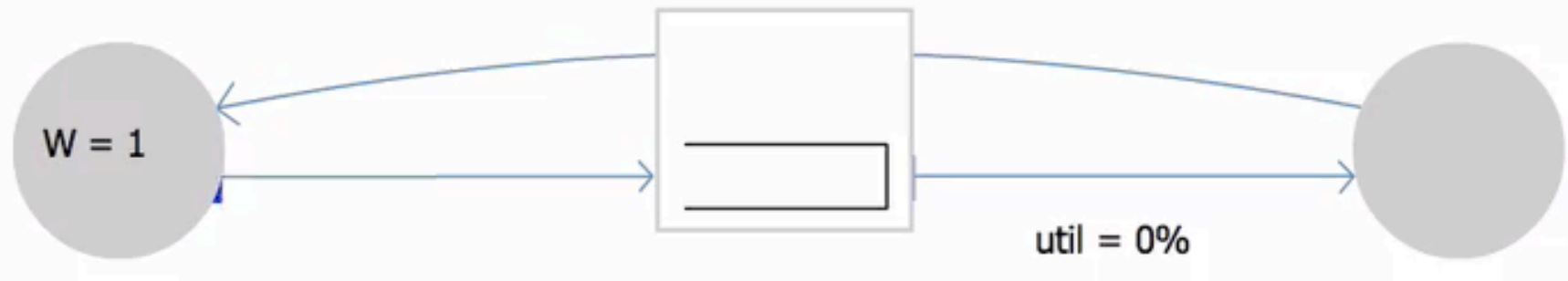
Only W packets
may be outstanding

Rule for adjusting W

- If an ACK is received: $W \leftarrow W+1/W$
- If a packet is lost: $W \leftarrow W/2$



Congestion Window Evolution



Practical Details

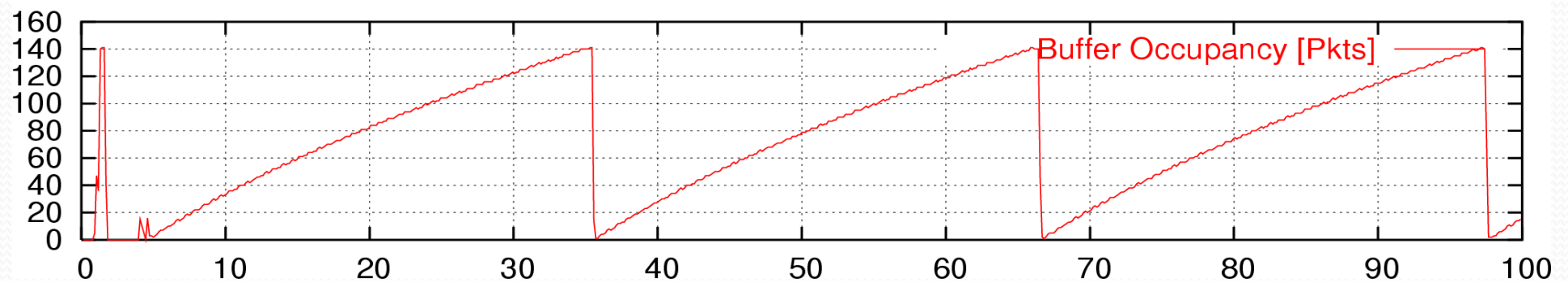
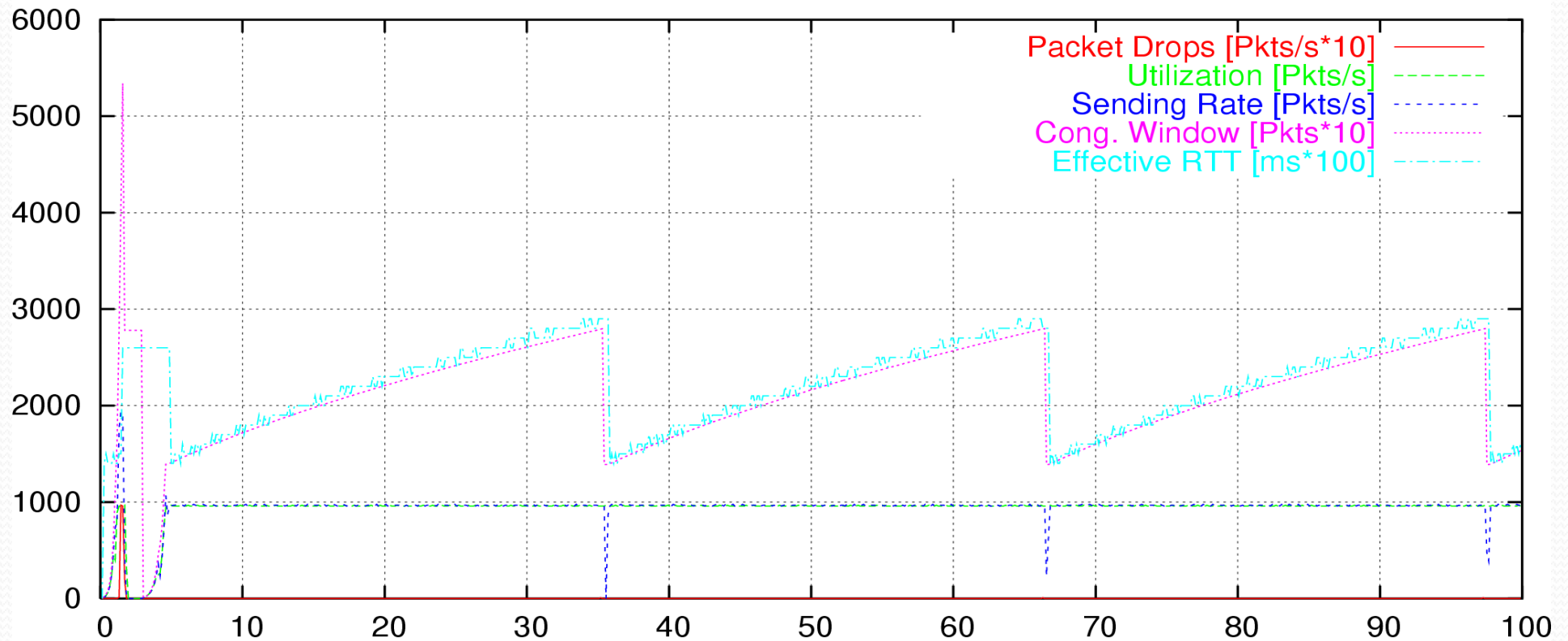
- Congestion window
 - Represented in bytes, not in packets (Why?)
 - Packets have MSS (Maximum Segment Size) bytes
- Increasing the congestion window
 - Increase by MSS on success for last window of data
 - In practice, increase a fraction of MSS per received ACK
 - # packets per window: $CWND / MSS$
 - Increment per ACK: $MSS * (MSS / CWND)$
- Decreasing the congestion window
 - Never drop congestion window below 1 MSS

TCP Sending Rate

- What is the sending rate of TCP?
- Acknowledgement for sent packet is received after one RTT
- Amount of data sent until ACK is received is the current window size W
- Therefore sending rate is $R = W/RTT$
- Is the TCP sending rate saw tooth shaped as well?

TCP Sending Rate and Buffers

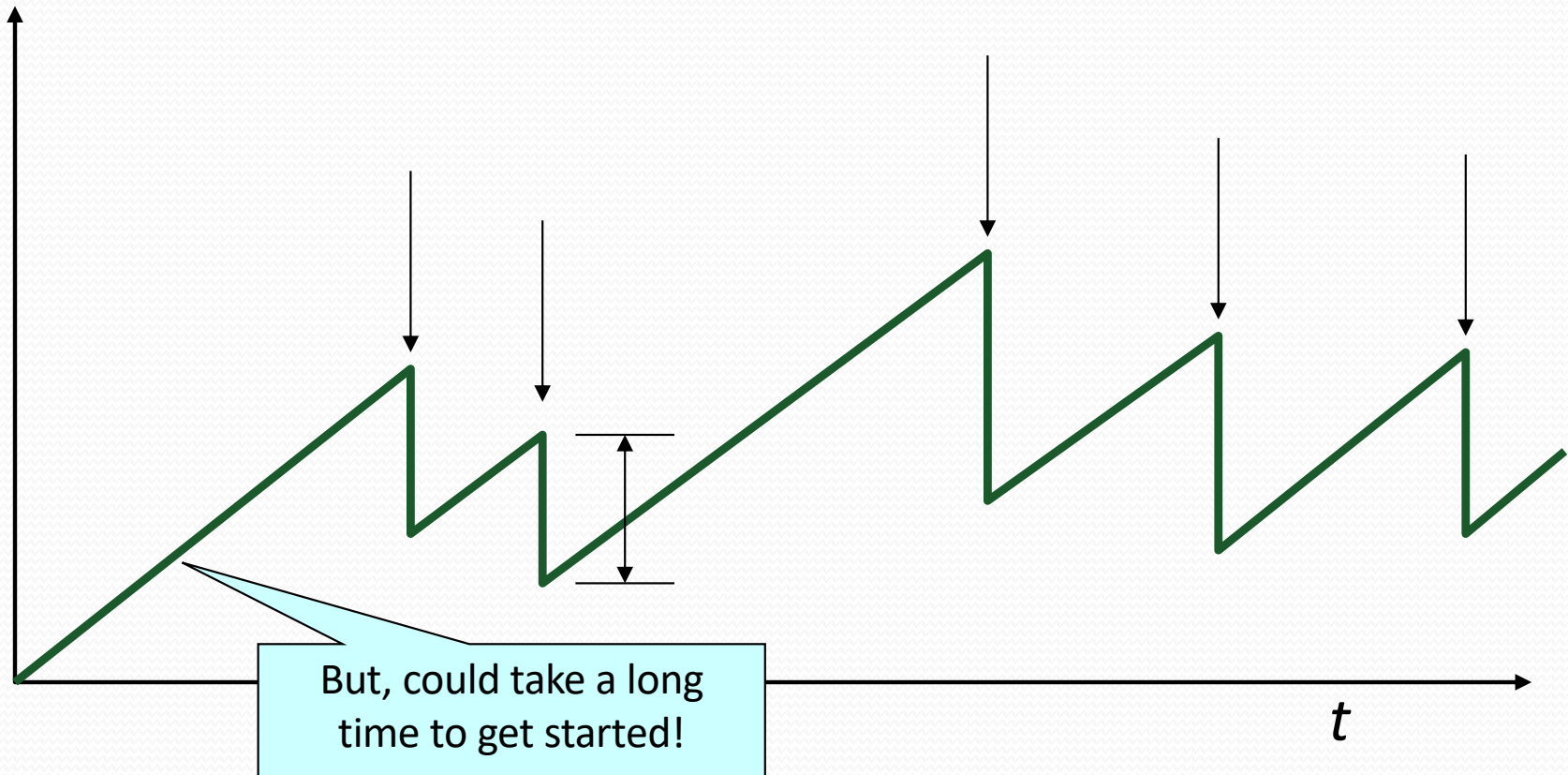
TCPSIM: Time evolution of a TCP flow#(RTT 142ms, BW 8000kb, buffer 142 pkts of 1000 bytes)



Getting Started

Need to start with a small CWND to avoid overloading the network.

Window



“Slow Start” Phase

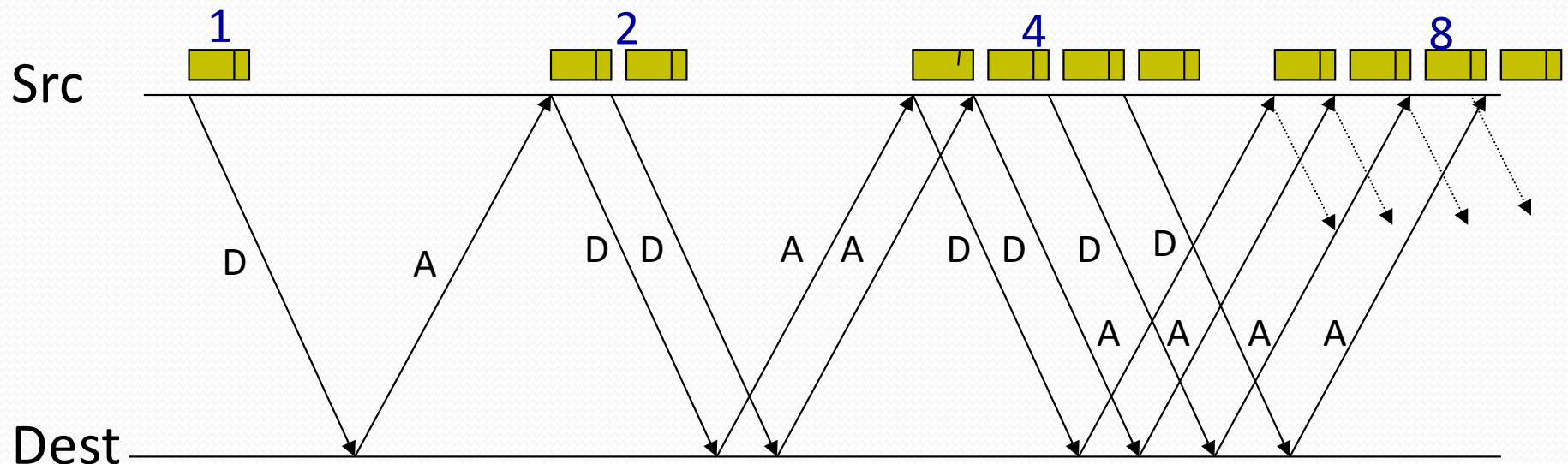
- Start with a small congestion window
 - Initially, CWND is 1 MSS
 - So, initial sending rate is MSS/RTT
- That could be pretty wasteful
 - Might be much less than the actual bandwidth
 - Linear increase takes a long time to accelerate
- Slow-start phase (really “fast start”)
 - Sender starts at a slow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

Slow Start in Action

Double CWND per round-trip time

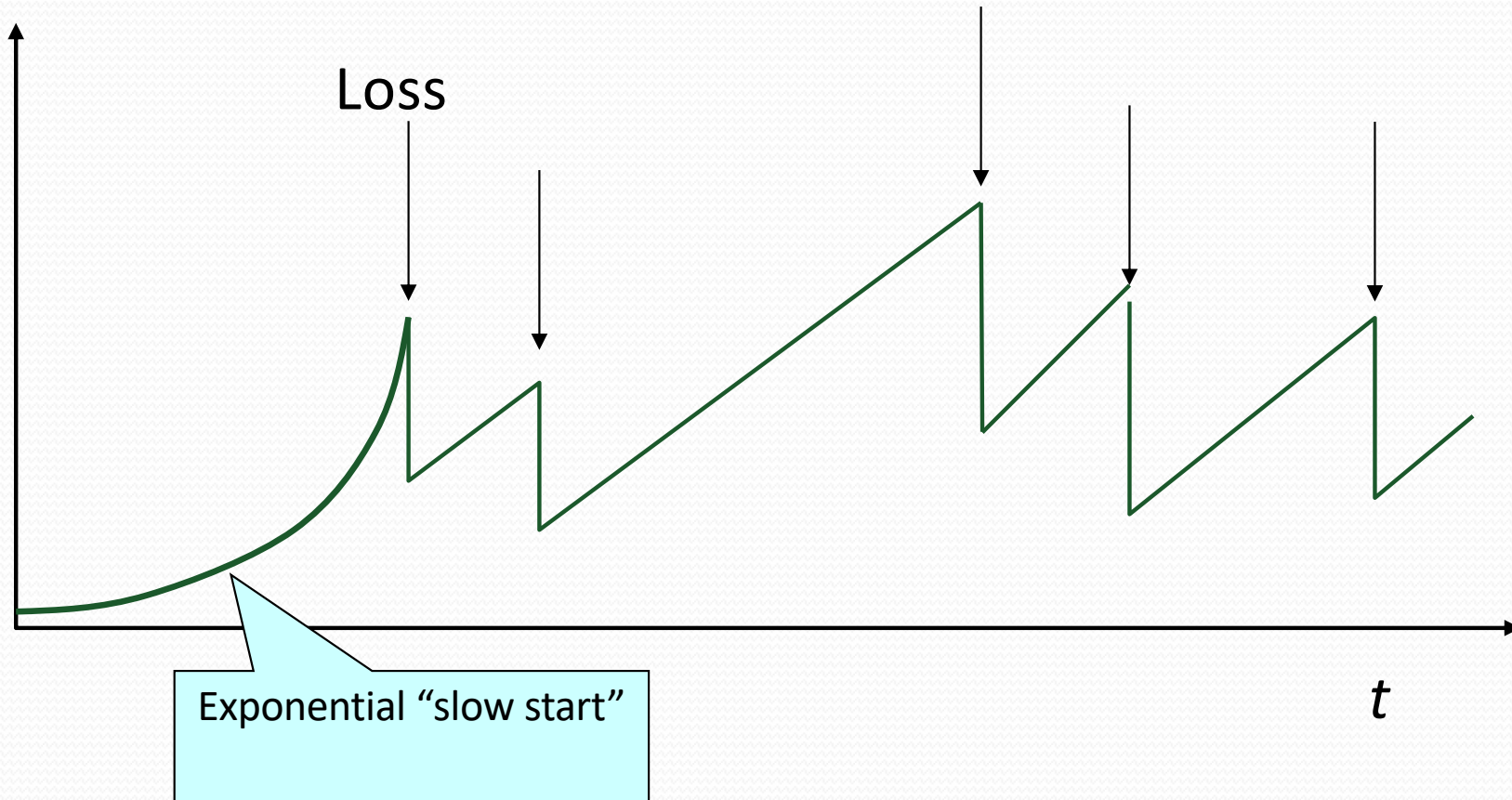
=

Increase CWND by 1 for each ACK received



Slow Start and the TCP Sawtooth

Window



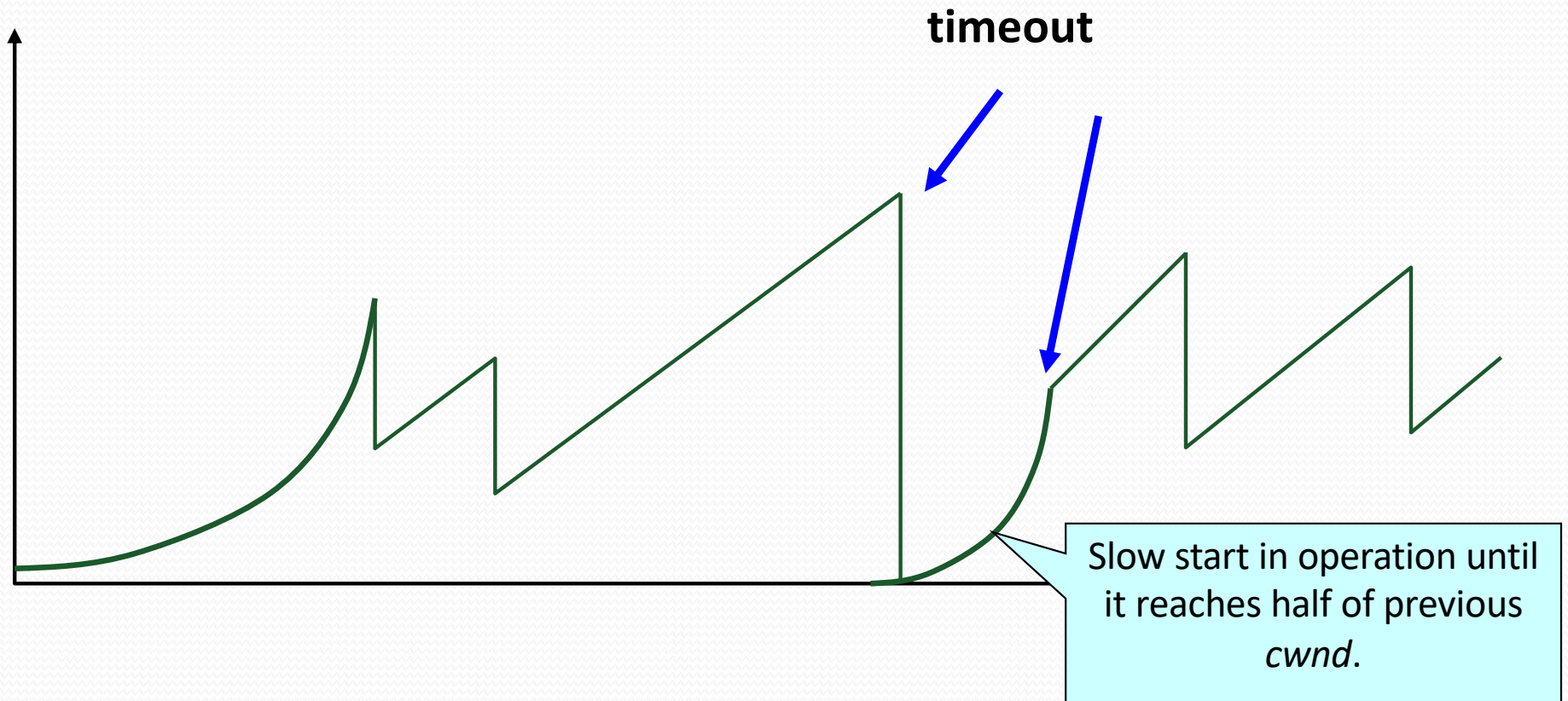
Why is it called slow-start? Because TCP originally had no congestion control mechanism. The source would just start by sending a whole window's worth of data.

Two Kinds of Loss in TCP

- Triple duplicate ACK
 - Packet n is lost, but packets $n+1$, $n+2$, etc. arrive
 - Receiver sends duplicate acknowledgments
 - ... and the sender retransmits packet n quickly
 - **Do a multiplicative decrease and keep going**
- Timeout
 - Packet n is lost and detected via a timeout
 - E.g., because all packets in flight were lost
 - After the timeout, blasting away for the entire CWND
 - ... would trigger a very large burst in traffic
 - **So, better to start over with a low CWND**

Repeating Slow Start After Timeout

Window



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
 - E.g., Telnet session where you don't type for an hour
- Eventually, the network conditions change
 - Maybe many more flows are traversing the link
 - E.g., maybe everybody has come back from lunch!
- Dangerous to start transmitting at the old rate
 - Previously-idle TCP sender might blast the network
 - ... causing excessive congestion and packet loss
- So, some TCP implementations repeat slow start
 - Slow-start restart after an idle period

Other TCP Mechanisms

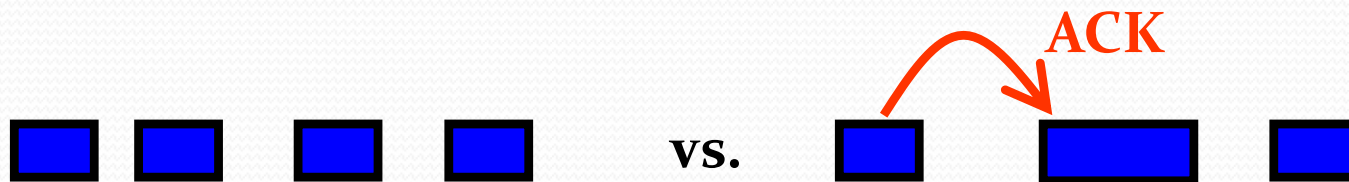
- Nagle's Algorithm and Delayed ACK

Motivation for Nagle's Algorithm

- Interactive applications
 - Telnet, ssh and rlogin
 - Generate many small packets (e.g., keystrokes)
- Small packets are wasteful
 - Mostly header (e.g., 40 bytes of header, 1 of data)
- Appealing to reduce the number of packets
 - Could force every packet to have some minimum size
 - ... but, what if the person doesn't type more characters?
- Need to balance competing trade-offs
 - Send larger packets
 - ... but don't introduce much delay by waiting

Nagle's Algorithm

- Wait if the amount of data is small
 - Smaller than Maximum Segment Size (MSS)
- And some other packet is already in flight
 - I.e., still awaiting the ACKs for previous packets
- That is, send at most one small packet per RTT
 - ... by waiting until all outstanding ACKs have arrived



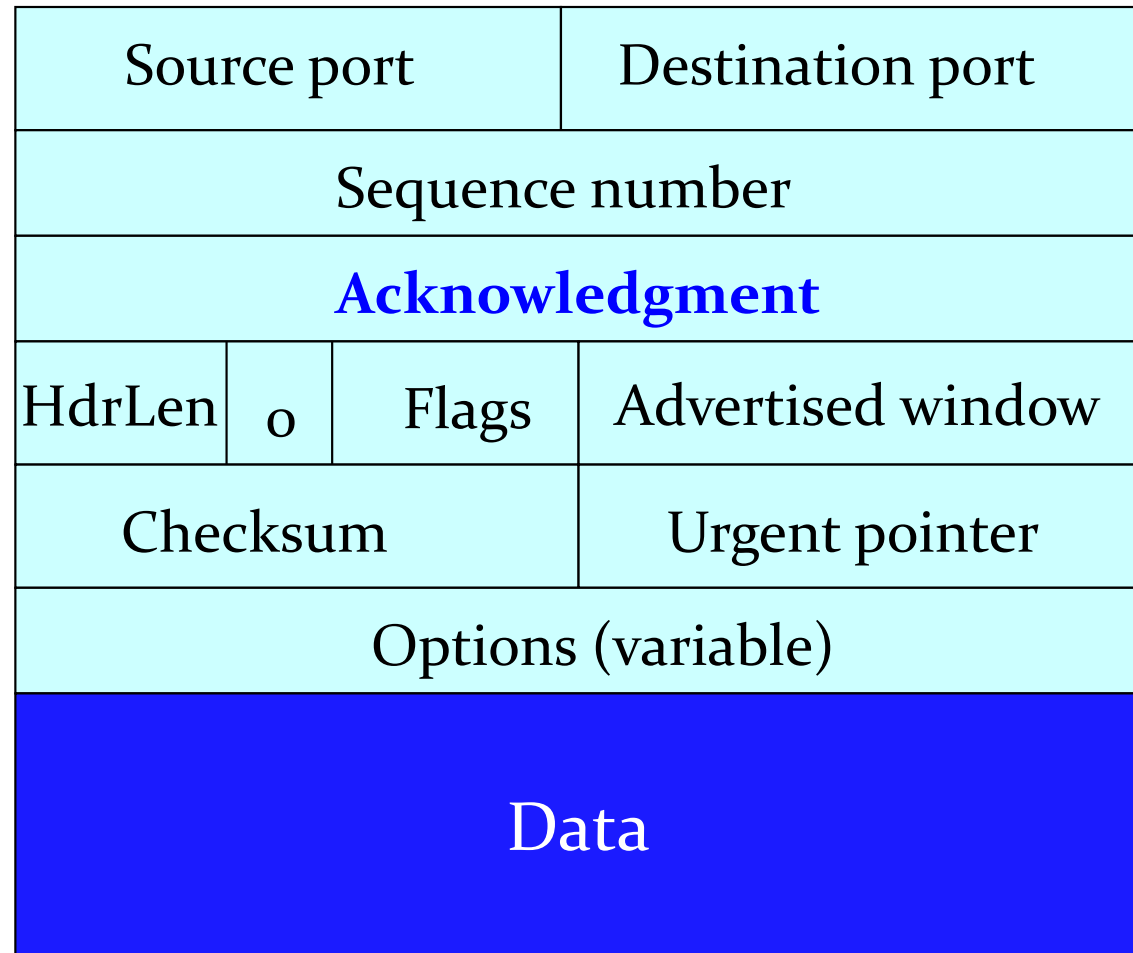
- Influence on performance
 - Interactive applications: enables batching of bytes
 - Bulk transfer: transmits in MSS-sized packets anyway

Motivation for Delayed ACK

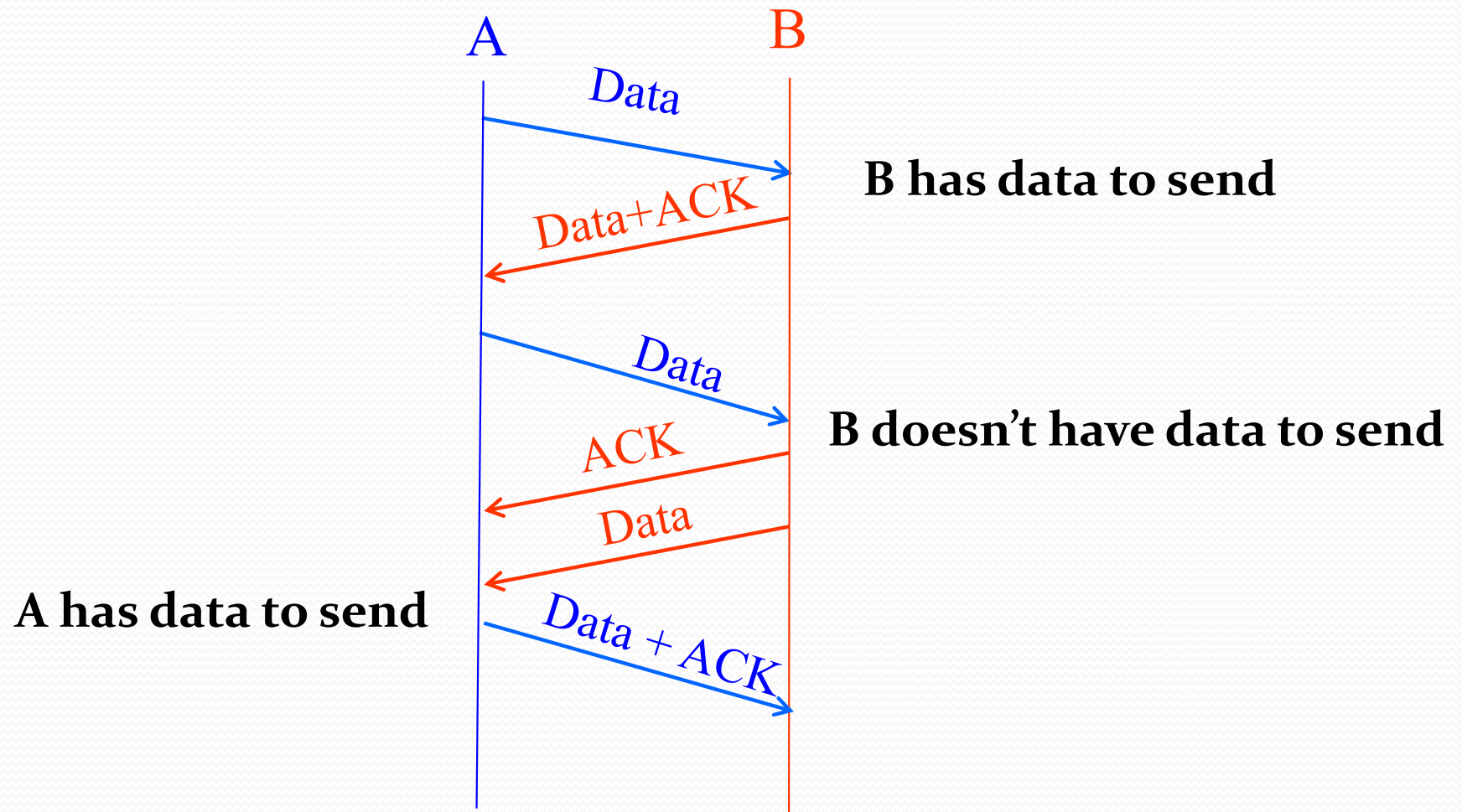
- TCP traffic is often bidirectional
 - Data traveling in both directions
 - ACKs traveling in both directions
- ACK packets have high overhead
 - 40 bytes for the IP header and TCP header
 - ... and zero data traffic
- Piggybacking is appealing
 - Host B can send an ACK to host A
 - ... as part of a data packet from B to A

TCP Header Allows Piggybacking

Flags: SYN
FIN
RST
PSH
URG
ACK

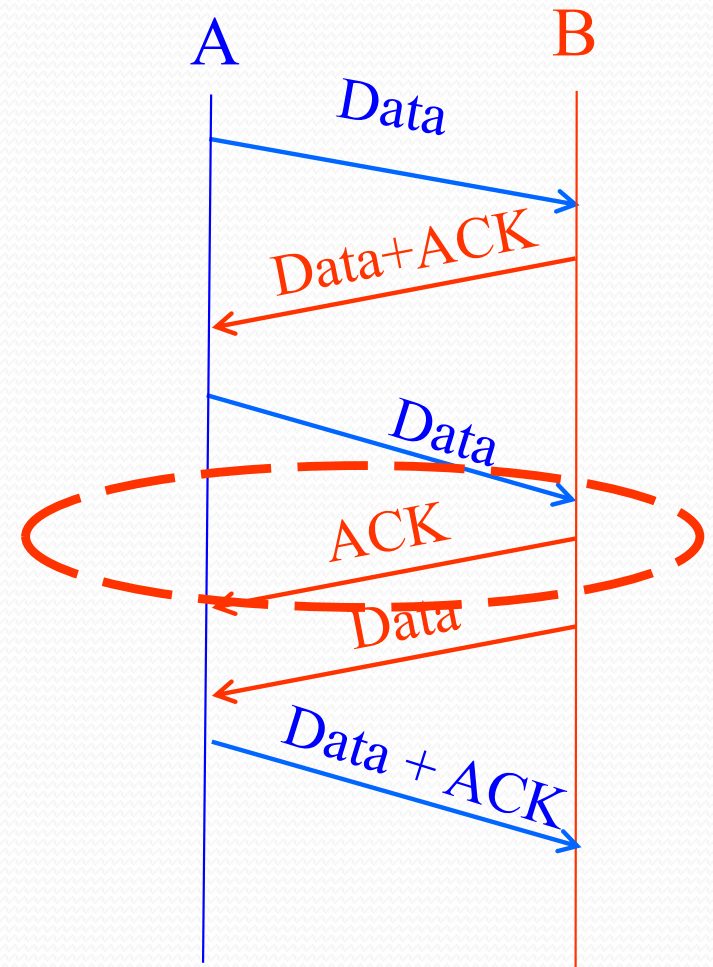


Example of Piggybacking



Increasing Likelihood of Piggybacking

- Increase piggybacking
 - TCP allows the receiver to wait to send the ACK
 - ... in the hope that the host will have data to send
- Example: rlogin or telnet
 - Host A types characters at a UNIX prompt
 - Host B receives the character and executes a command
 - ... and then data are generated
 - Would be nice if B could send the ACK with the new data



Delayed ACK

- Delay sending an ACK
 - Upon receiving a packet, the host B sets a timer
 - Typically, 200 msec or 500 msec
 - If B's application generates data, go ahead and send
 - And piggyback the ACK bit
 - If the timer expires, send a (non-piggybacked) ACK
- Limiting the wait
 - Timer of 200 msec or 500 msec
 - ACK every other full-sized packet

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
- Active Queue Management can help
 - Random Early Detection (RED)
 - Explicit Congestion Notification (ECN)