Congestion Control (TCP)

Arvind Krishnamurthy

University of Washington

Overview

- TCP reliable delivery
- TCP congestion control
- Discussion of congestion control

TCP: Reliable Delivery

- Detect missing data: sequence number
  - Used to detect a gap in the stream of bytes
- Detect bit errors: checksum
  - Used to detect corrupted data at the receiver
- Recover from lost data: retransmission
  - Sender retransmits lost or corrupted data
  - Two main ways to detect lost packets
    - Retransmission timeout and fast retransmission

How long should sender wait?

- Sender sets a timeout to wait for an ACK
  - Too short: wasted retransmissions
  - Too long: excessive delays when packet lost
- TCP sets timeout as a function of the RTT

- Questions:
  - what are the hurdles in calculating a good timeout?
  - how would you improve the scheme from the paper?

Flaw in this Approach

- An ACK doesn't really acknowledge a transmission
  - Rather, it acknowledges receipt of the data
- Consider a retransmission of a lost packet
  - If you assume the ACK goes with the 1st transmission, the SampleRTT comes out way too large
- Consider a duplicate packet
  - If you assume the ACK goes with the 2nd transmission, the Sample RTT comes out way too small
- Simple solution in the Karn/Partridge algorithm
  - Only collect samples for segments sent one single time

Fast Retransmission

- Better solution possible under sliding window
  - Packet n might have been lost, but packets n+1, n+2, and so on might get through

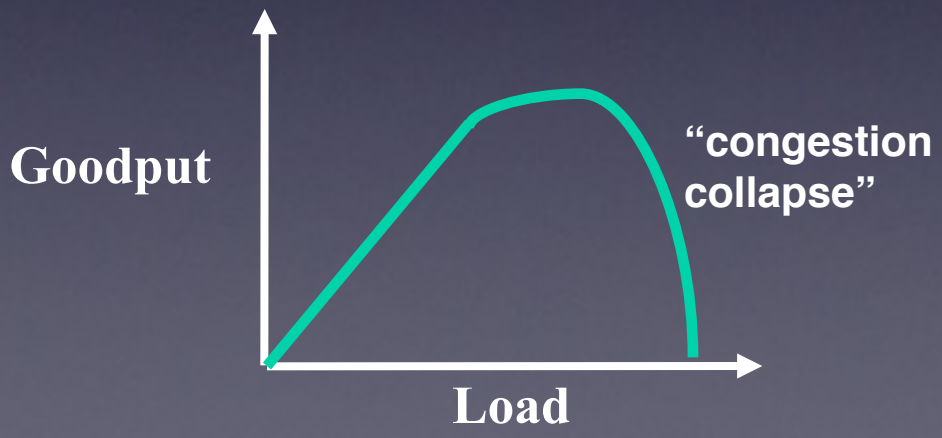
- Idea: have the receiver send ACK packets
  - ACK says that receiver is still awaiting nth packet
    - Repeated ACKs suggest later packets have arrived
  - Sender can view the "duplicate ACKs" as an early hint

- Fast retransmission
  - Sender retransmits data after the "triple duplicate ACK"

Congestion Control

Congestion

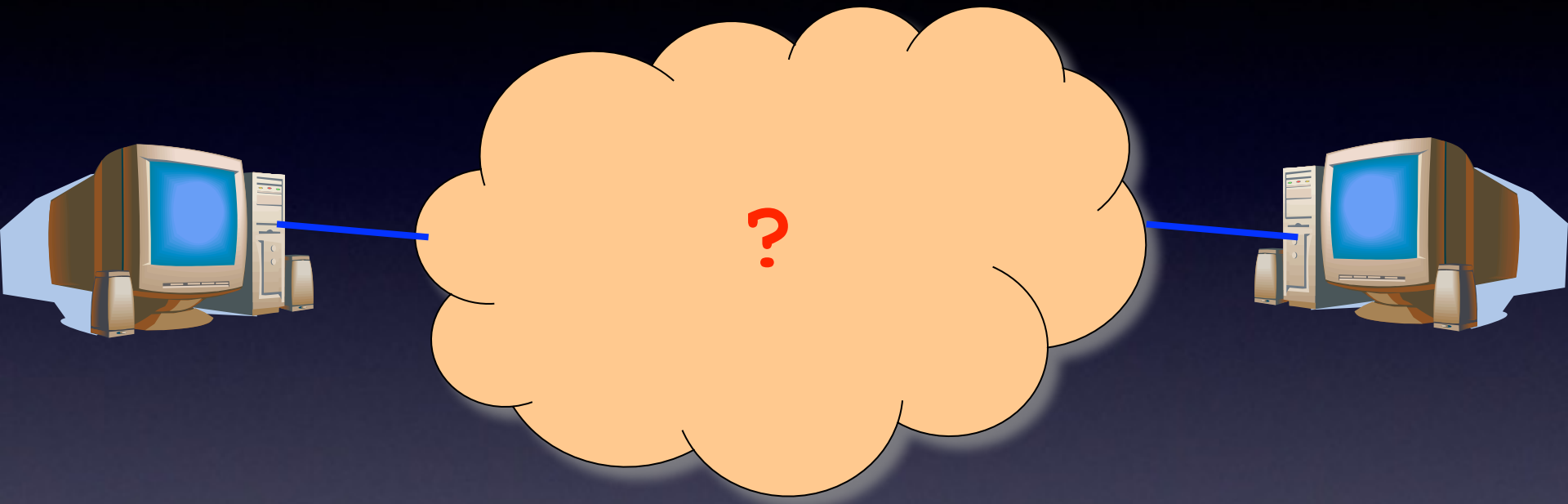
- What is congestion?
  - Load is higher than capacity
- What do IP routers do?
  - Drop the excess packets
- Why is this bad?
  - Wasted bandwidth for retransmissions



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

- Question: can we design a network that doesn't have any congestion at all?

Inferring at End-hosts



- What does the end host see?
  - Round-trip loss
  - Round-trip delay

Hosts adapt sending rate

- Congestion window
  - Maximum number of bytes to have in transit, i.e., # of bytes still awaiting acknowledgments
- Upon detecting congestion
  - Decrease the window size (e.g., divide in half)
  - End host does its part to alleviate the congestion
- Upon not detecting congestion
  - Increase the window size, a little at a time
  - End host learns whether conditions have changed

Leads to TCP Sawtooth

- TCP sawtooth is referred to as AIMD
  - "Additive increase, multiplicative decrease"

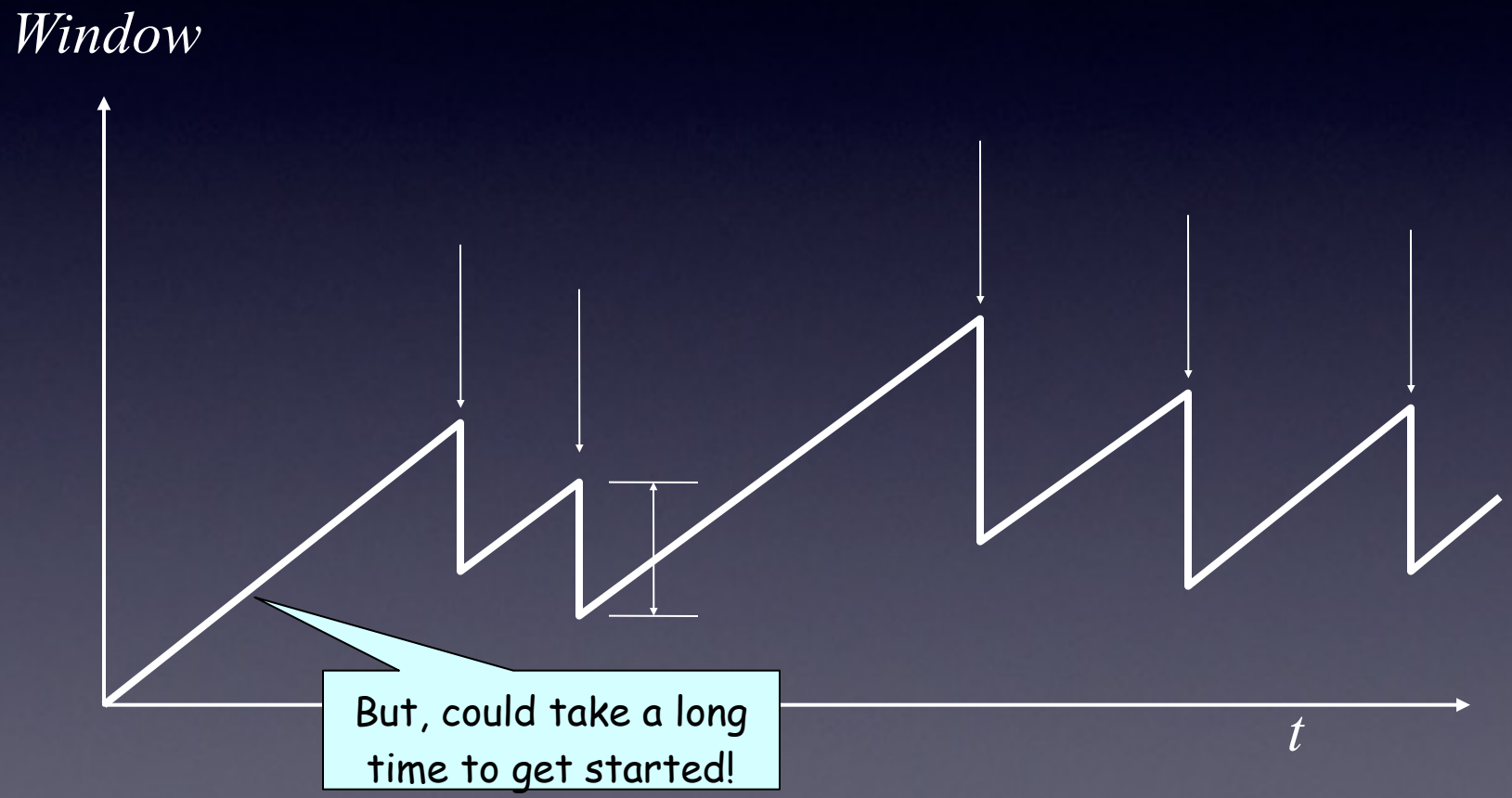
- Question:
  - what are other alternatives to AIMD?
  - is AIMD good? and if so, why?

Receive 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
  - TCP window: min{congestion window, receiver window}

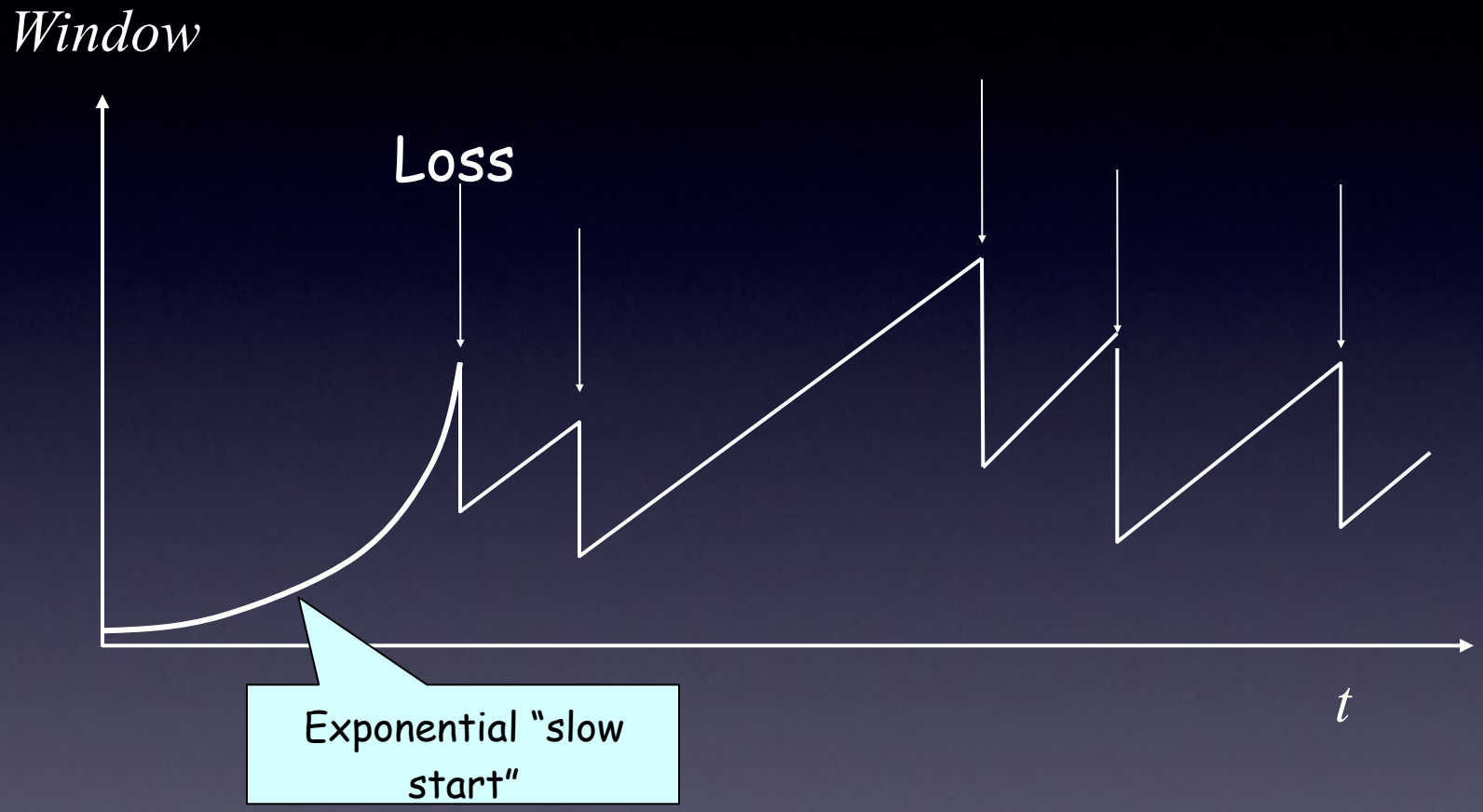
How should a new flow start?



“Slow Start” phase

- Start with a small congestion window
  - Initially, CWND is 1 Max Segment Size (MSS)
- That could be pretty wasteful
  - Might be much less than the actual bandwidth
  - Linear increase takes a long time to accelerate
- Slow-start phase
  - Sender starts at a slow rate (hence the name)
  - But increases the rate exponentially, until the first loss event

Slow Start



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

- Recall that we have fast retransmit

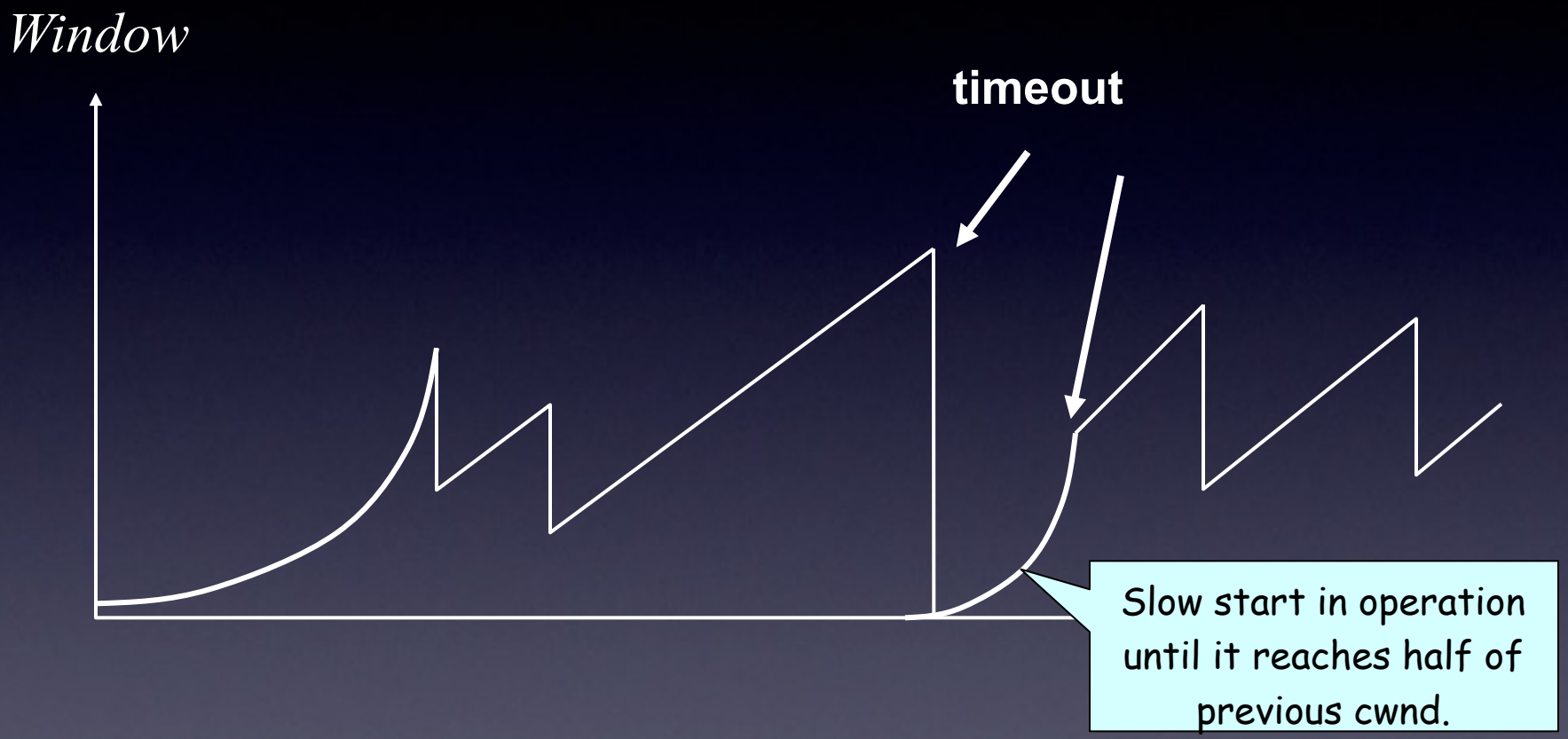
- Question:
  - when are timeouts triggered in TCP?
  - what should we do to cwnd upon timeouts?

Two kinds of loss

- Timeout
  - Packet n is lost and detected via a timeout
    - E.g., because all packets in flight were lost
  - After timeout, blasting away for the entire CWND would trigger a very large burst in traffic
  - So, better to start over with a low CWND

- Triple duplicate ACK
  - Packet n is lost, but packets n+1, n+2, etc. arrive
  - And the sender retransmits packet n quickly
  - Do a multiplicative decrease and keep going

Slow Start after Timeout



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

- Question:
  - what factors determine TCP performance?
  - alternately, what would be an analytical model for TCP performance?

Factors impacting TCP Performance

- Round-trip time
  - Throughput proportional to 1/RTT
- Receiver window
  - Throughput is limited by window/RTT
- Slow start and additive increase
  - Certain number of RTTs needed to send the data, even in the absence of any congestion
- Packet loss and congestion window decreases
  - Throughput proportional to 1/sqrt(loss)
  - Duplicate ACKs don't happen for short transfers and bursty loss, and timeout losses are expensive

- Question: suggestions for a "clean-slate" TCP?
  - either take advantage of more information on end-hosts
  - or try to rely on network/router support