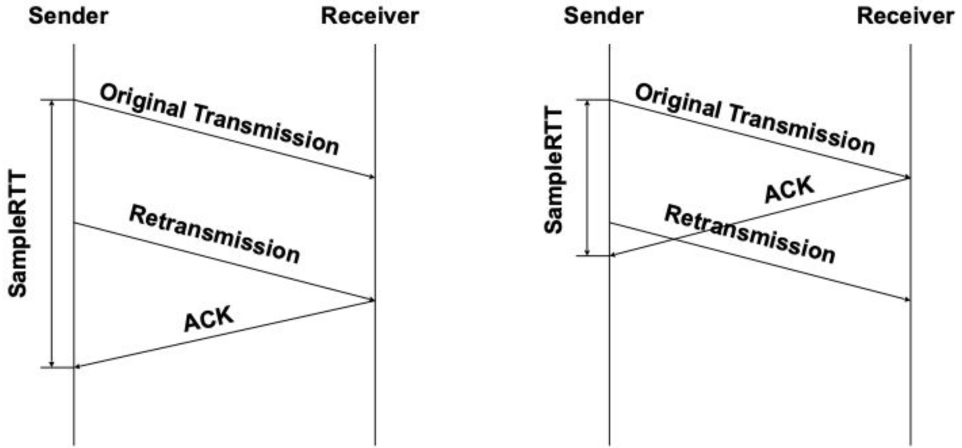
We Need RTT, but Problem: Ambiguous Measurements

How do we differentiate between the real ACK, and ACK of the retransmitted packet?



Karn/Partridge Algorithm

- Measure SampleRTT only for original transmissions
  - Once a segment has been retransmitted, do not use it for any further measurements
  - Computes EstimatedRTT using $\alpha = 0.875$
- Timeout value (RTO) = 2 × EstimatedRTT
- Use exponential backoff for repeated retransmissions
  - Every time RTO timer expires, set RTO ← 2·RTO
    - (Up to maximum ≥ 60 sec)
  - Every time new measurement comes in (= successful original transmission), collapse RTO back to 2 × EstimatedRTT

Reality

- Implementations often use a coarse-grained timer
    - 500 msec is typical

- So what?
    - Above algorithms are largely irrelevant
    - Incurring a timeout is expensive

- So we rely on duplicate ACKs

Loss with Cumulative ACKs

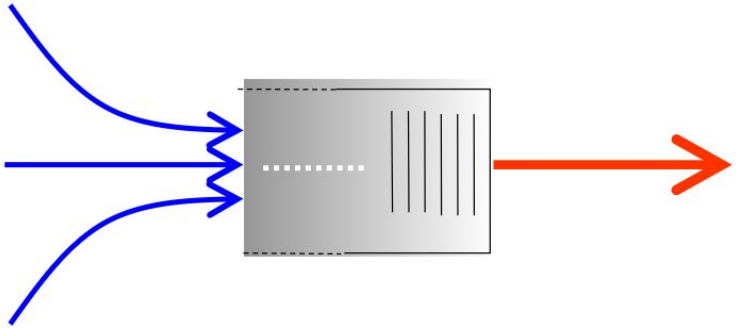
- Sender sends packets with 100B and seqnos.:
  - 100, 200, 300, 400, 500, 600, 700, 800, 900, …
- Assume the fifth packet (seqno 500) is lost, but no others
- Stream of ACKs will be:
  - 200, 300, 400, 500, 500, 500, 500,…

Loss with Cumulative ACKs

- "Duplicate ACKs" are a sign of an isolated loss
    - The lack of ACK progress means 500 hasn't been delivered
    - Stream of ACKs means *some* packets are being delivered
- Therefore, could trigger resend upon receiving k duplicate ACKs
    - TCP uses k=3

Congestion Control

Because of traffic burstiness and lack of BW reservation, congestion is inevitable



If many packets arrive within
a short period of time
the node cannot keep up anymore

Congestion is not a new problem

- The Internet almost died of congestion in 1986
  - throughput collapsed from 32 Kbps to... 40 bps
- Van Jacobson saved us with Congestion Control
  - his solution went immediately into BSD
- Recent resurgence of research interest after brief lag
  - new methods (ML), context (Data centers), requirements

Congestion is not a new problem

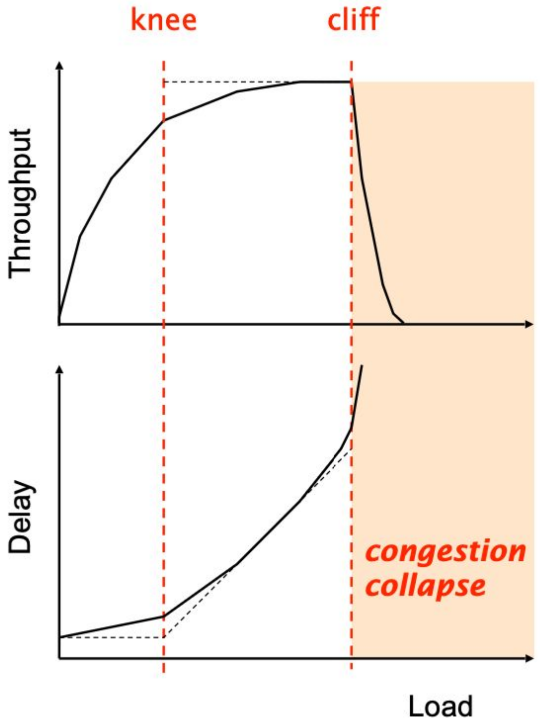
- The Internet almost died of congestion in 1986
  - throughput collapsed from 32 Kbps to... 40 bps
- Van Jacobson saved us with Congestion Control
  - his solution went right into BSD
- Recent resurgence of research interest after brief lag
  - new methods (ML), context (Data centers), requirements

Congestion is not a new problem

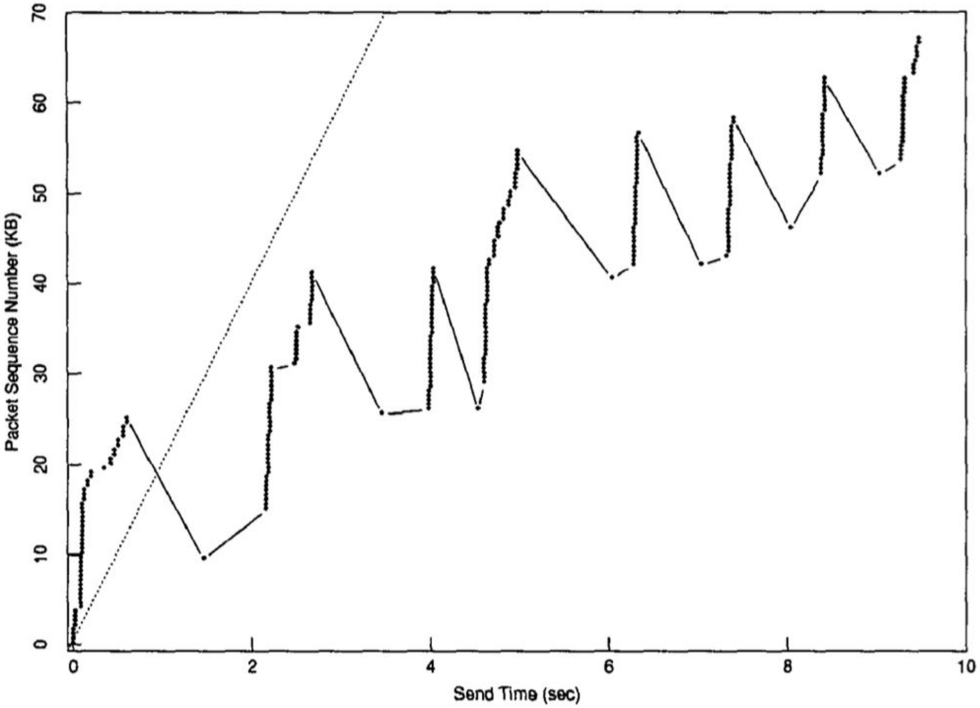
original behavior	On connection, nodes send full window of packets
	Upon timer expiration, retransmit packet immediately
meaning	sending rate only limited by flow control
net effect	window-sized burst of packets

Congestion collapse

Knee	point after which	
throughput	increases	slowly
delay	increases	quickly
Cliff	point after which	
throughput	decreases	quickly
delay	tends to	infinity



Congestion collapse



Congestion control aims to solve three problems

#1	bandwidth estimation	How to adjust the bandwidth of a single flow to the bottleneck bandwidth? could be 1 Mbps or 1 Gbps...
#2	bandwidth adaptation	How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
#3	fairness	How to share bandwidth “fairly" among flows, without overloading the network

Congestion control differs from flow control

Flow control	prevents **one fast sender** from overloading **a slow receiver**
Congestion control	prevents **a set of senders** from overloading **the network**

TCP solves both using two distinct windows

Flow control

prevents **one fast sender** from overloading a slow receiver

solved using a **receiving window**

Congestion control

prevents **a set of senders** from overloading the network

solved using a **“congestion” window**

The sender adapts its sending rate based on these two windows

Receiving Window RWND	How many bytes can be sent without overflowing the receiver buffer? based on the receiver input
Congestion Window CWND	How many bytes can be sent without overflowing the routers? based on network conditions
Sender Window	minimum(CWND, RWND)

The 2 key mechanisms of Congestion Control

detecting
congestion

reacting to
congestion

The 2 key mechanisms of Congestion Control

detecting congestion

reacting to congestion

There are essentially three ways to detect congestion

Approach #1	Network could tell the source but signal itself could be lost
Approach #2	Measure packet delay but signal is noisy delay often varies considerably
Approach #3	Measure packet loss fail-safe signal that TCP already has to detect

There are essentially three ways to detect congestion

Approach #1

Network could tell the source

Best solution - delay and signaling-based methods are hard & risky

but signal is noisy

delay often varies considerably

Approach #3

Measure packet loss

fail-safe signal that TCP already has to detect

Detecting losses can be done using ACKs or timeouts, the two signal differ in their degree of severity

duplicated ACKs	mild congestion signal packets are still making it
timeout	severe congestion signal multiple consequent losses

The 2 key mechanisms of Congestion Control

detecting congestion

reacting to congestion

Remember: congestion control aims to solve three problems

#1	bandwidth estimation	How to adjust the bandwidth of a single flow to the bottleneck bandwidth? could be 1 Mbps or 1 Gbps...
#2	bandwidth adaptation	How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
#3	fairness	How to share bandwidth “fairly" among flows, without overloading the network

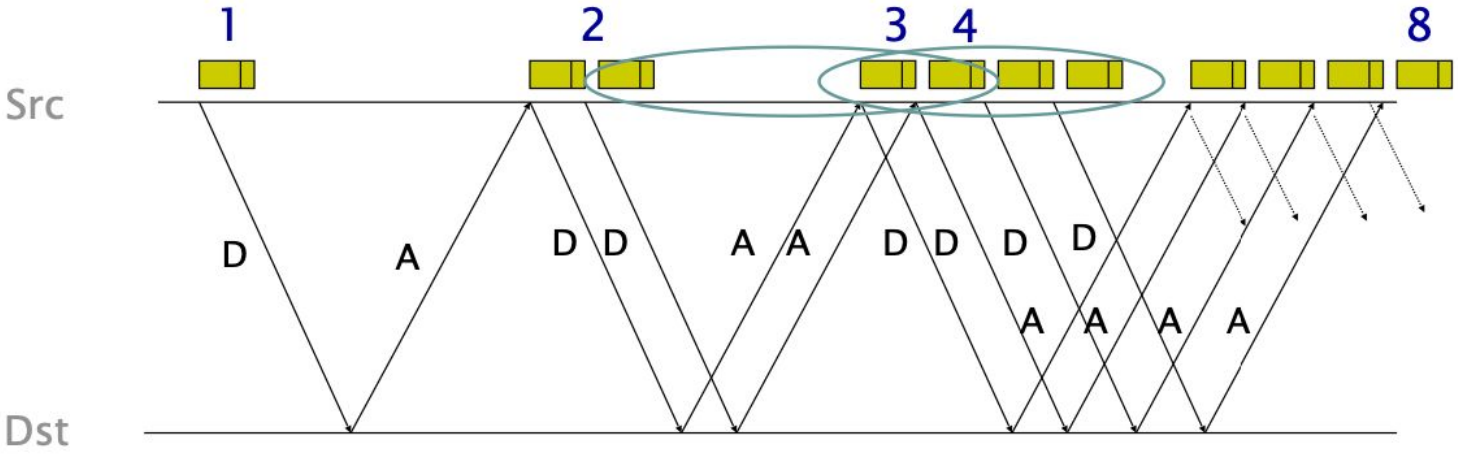
Remember: congestion control aims to solve three problems

#1	bandwidth estimation	How to adjust the bandwidth of a single flow to the bottleneck bandwidth? could be 1 Mbps or 1 Gbps...
#2	bandwidth adaptation	How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
#3	fairness	How to share bandwidth "fairly" among flows, without overloading the network

The goal here is to quickly get a first-order estimate of the available bandwidth

Intuition	Start slow but rapidly increase until a packet drop occurs	
Increase policy	cwnd = 1	initially
	cwnd += 1	upon receipt of an ACK

This increase phase, known as slow start, corresponds to an… exponential increase of CWND



slow start is called like this only because of starting point

The problem with slow start is that it can result in a full window of packet losses

Example	Assume that CWND is just enough to "fill the pipe" After one RTT, CWND has doubled All the excess packets are now dropped
Solution	We need a more gentle adjustment algorithm once we have a rough estimate of the bandwidth

#1	bandwidth **estimation**	How to adjust the bandwidth of a single flow to the bottleneck bandwidth? could be 1 Mbps or 1 Gbps…
#2	bandwidth **adaptation**	How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
#3	**fairness**	How to share bandwidth “fairly" among flows, without overloading the network

The goal here is to track the available bandwidth, and oscillate around its current value

Two possible variations

- Multiplicative Increase or Decrease

  cwnd = a * cwnd

- Additive Increase or Decrease

  cwnd = b + cwnd

… leading to four alternative design

The goal here is to track the available bandwidth, and oscillate around its current value

	increase behavior	decrease behavior
AIAD	gentle	gentle
AIMD	gentle	aggressive
MIAD	aggressive	gentle
MIMD	aggressive	aggressive

The goal here is to track the available bandwidth, and oscillate around its current value

How do we choose a scheme? Based on fairness

AIAD	gentle	gentle
AIMD	gentle	aggressive
MIAD	aggressive	gentle
MIMD	aggressive	aggressive