Effects of Retransmission on Congestion

- Ideal case
  - Every packet delivered successfully until capacity
  - Beyond capacity: deliver packets at capacity rate
- Realistically
  - As offered load increases, more packets lost
    - More retransmissions → more traffic → more losses …
  - In face of loss, or long end-end delay
    - Retransmissions can make things worse
    - In other words, no new packets get sent!
  - Decreasing rate of transmission in face of congestion
    - Increases overall throughput (or rather "goodput") !

Congestion: Moral of the Story

- When losses occur
  - Back off, don't aggressively retransmit
    i.e., be a nice guy!
- Issue of fairness
  - "Social" versus "individual" good
  - What about greedy senders who don't back off?

Approaches towards Congestion Control

Two broad approaches towards congestion control:

End-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

Network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate sender should send at

TCP Approach

- Basic Ideas:
  - Each source "determines" network capacity for itself
  - Uses implicit feedback, adaptive congestion window
  - ACKs pace transmission ("self-clocking")
- Challenges
  - Determining available capacity in the first place
  - Adjusting to changes in the available capacity

TCP Congestion Control

- "probing" for usable bandwidth:
  - ideally: transmit as fast as possible (`Congwin` as large as possible) without loss
  - *increase* `Congwin` until loss (congestion)
  - loss: *decrease* `Congwin`, then begin probing (increasing) again

- two "phases"
  - slow start
  - congestion avoidance
- important variables:
  - `Congwin`
  - `threshold`: defines threshold between slow start and congestion avoidance phases
- Q: how to adjust Congwin?

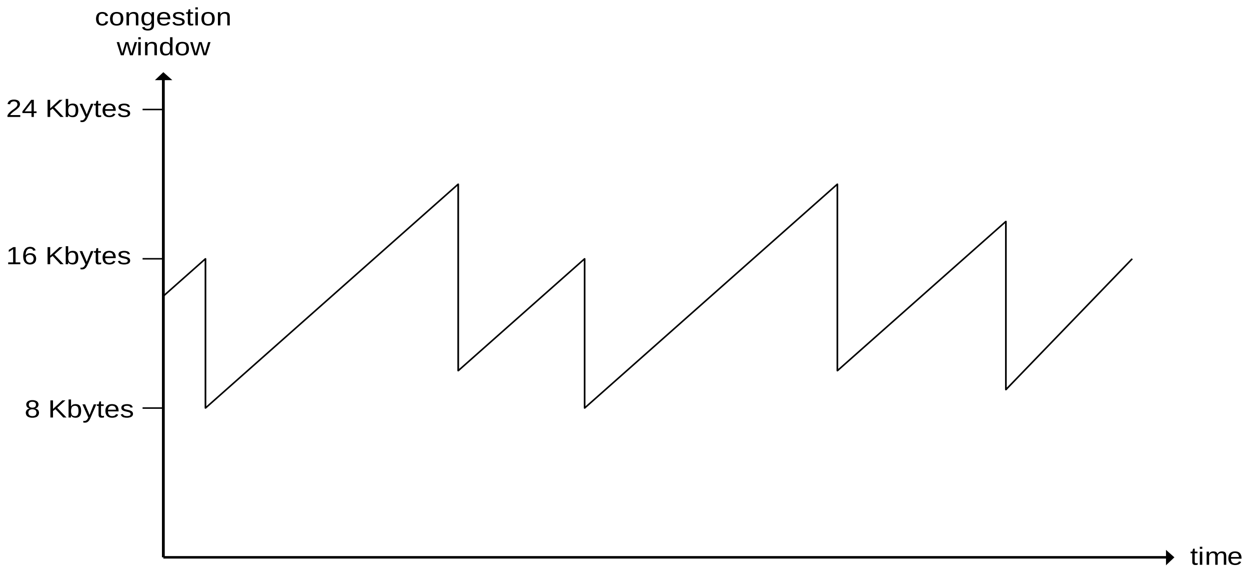
Additive Increase/Multiplicative Decrease (AIMD)

- Objective: Adjust to changes in available capacity
  - A state variable per connection: CongWin
    - Limit how much data source has is in transit
  - MaxWin = MIN(RcvWindow, CongWin)
- Algorithm:
  - Increase CongWin when congestion goes down (no losses)
    - Increment CongWin by 1 pkt per RTT (linear increase)
  - Decrease CongWin when congestion goes up (timeout)
    - Divide CongWin by 2 (multiplicative decrease)

TCP AIMD

<u>multiplicative decrease:</u>
cut `CongWin` in half after loss event

<u>additive increase:</u>
increase `CongWin` by 1 MSS (max. seg. size) every RTT in the absence of loss events



Long-lived TCP connection

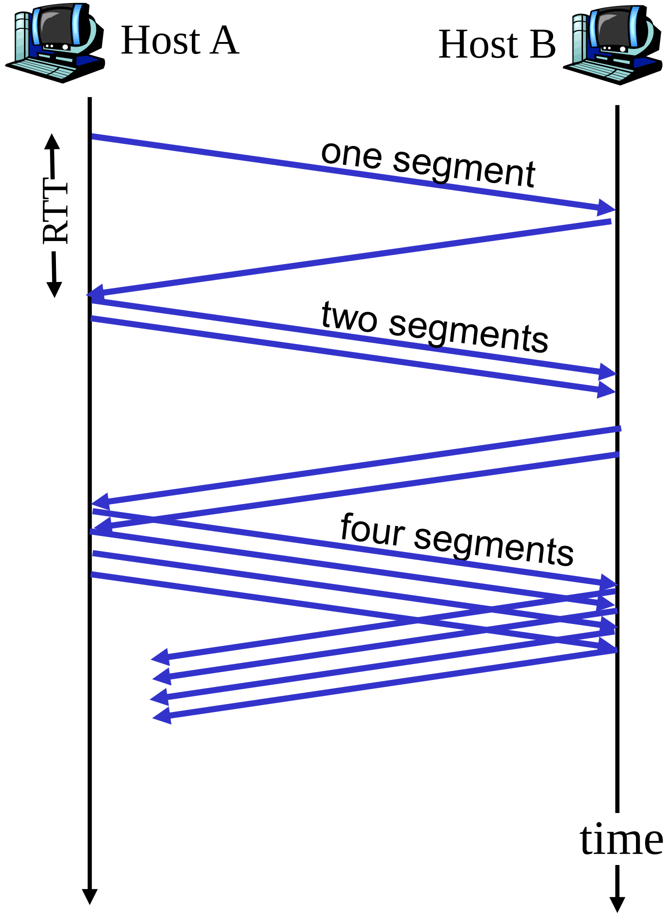
Why Slow Start?

- Objective
  - Determine the available capacity in the first place
- Idea:
  - Begin with congestion window = 1 pkt
  - Double congestion window each RTT
    - Increment by 1 packet for each ack
- Exponential growth, but slower than “one blast”
- Used when
  - first starting connection
  - connection goes dead waiting for a timeout

TCP Slowstart

Slowstart algorithm

```
initialize: CongWin = 1
for (each segment ACKed)
      CongWin++
until (loss event OR
        CongWin > threshold)
```

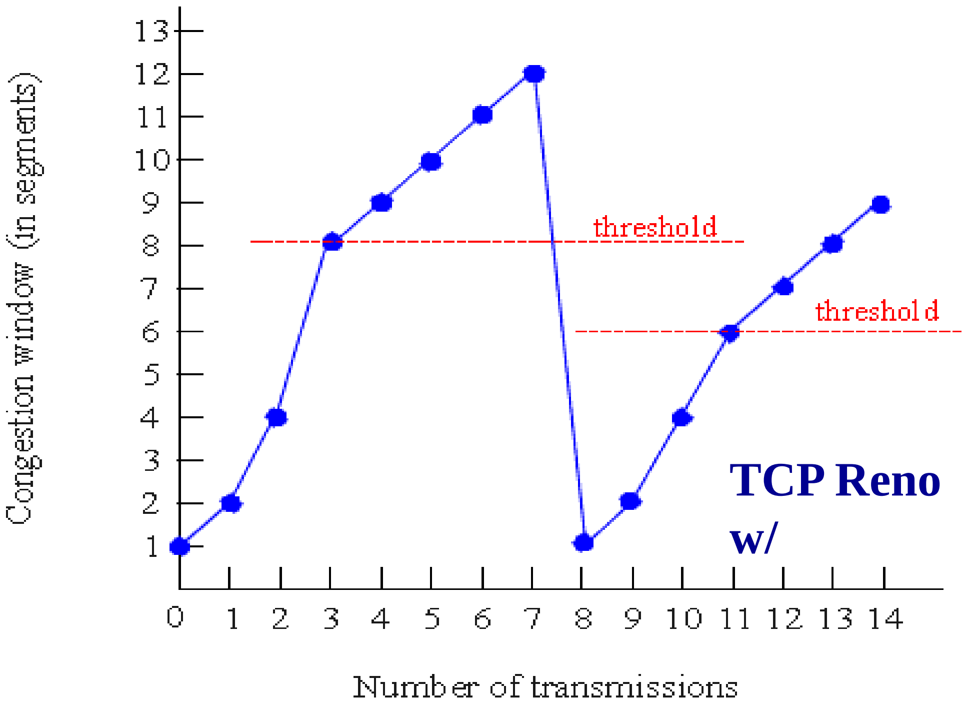


- exponential increase (per RTT) in window size (not so slow!)
- loss event: timeout (TCP Tahoe/Reno) and/or three duplicate ACKs (TCP Reno only)

TCP Congestion Avoidance

Congestion Avoidance

```
/* slowstart is over       */
/* Congwin > threshold */
Until (loss event) {
  every W segments ACKed:
      Congwin++
  }
Threshold: = Congwin/2
Congwin = 1
perform slowstart
```



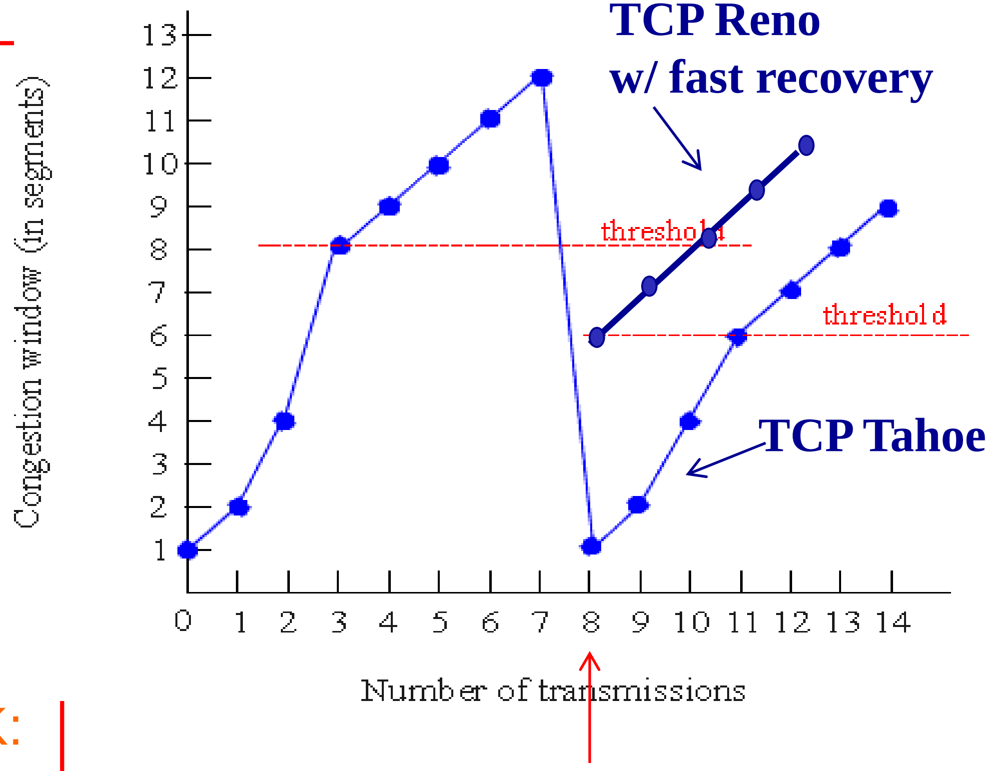
Fast Recovery/Fast Retransmit

- Coarse-grain TCP timeouts lead to idle periods
- Fast Retransmit
  - Use duplicate acks to trigger retransmission
  - Retransmit after three duplicate acks
- After "triple duplicate ACKs", Fast Recovery
  - Remove slow start phase
  - Go directly to half the last successful CongWin
  - Enter congestion avoid phase
- Implemented in TCP Reno (used by most of today's hosts)

TCP Congestion Avoidance Revisited

Congestion Avoidance

```
/* slowstart is over        */
/* Congwin > threshold */
until (loss event) {
  every W segments ACKed:
      CongWin++
  }
threshold: = Congwin/2
if loss event = time-out:
CongWin = 1;
perform slowstart;
if loss event = triple duplicate ACK:
 CongWin: = threshold;
  perform congestion avoidance;
```



loss event:
triple duplicate ACKs

TCP Congestion Control: Recap

- end-end control (no network assistance)
- sender limits transmission:
  `LastByteSent-LastByteAcked`
  `£CongWin`
- Roughly,

$$\boxed{\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}}$$

- **`CongWin`** is dynamic, function of perceived network congestion

How does sender perceive congestion?

- loss event = timeout *or* 3 duplicate ACKs
- TCP sender reduces rate (**`CongWin`**) after loss event

three mechanisms:

- AIMD
- slow start
- conservative after timeout events

TCP Congestion Control: Recap (cont'd)

- When CongWin is below threshold, sender in slow-start phase, window grows exponentially:
  - or commonly implemented using the following method:

    for each ACK received, CongWin: = CongWin + MSS;

- When CongWin is above **Threshold**, sender is in congestion-avoidance phase, window grows linearly
  - If current CongWin=W: every W segments ACKed: CongWin++
  - or commonly implemented using the following method:

    for each ACK received, CongWin: = CongWin + MSS/CongWin;

- When a triple duplicate ACKs occurs, threshold set to CongWin/2, and CongWin set to threshold.
- When timeout occurs, threshold set to CongWin/2, and CongWin is set to 1 MSS.

TCP Congestion Control: Sender Actions

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed