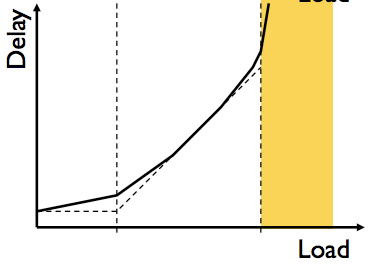
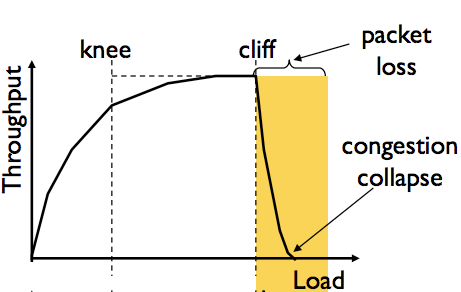
**Congestion Control**

Congestion = too many sources sending too much data too fast for the network to handle

Without congestion control:

* Increases delivery latency (delay in router buffers)
* Increases packet loss rate (router buffer overflows)
* Increase retransmissions, many unnecessary
* Wastes capacity of traffic that is never delivered
* Increases congestion 🡪 cycle continues. (eventual network congestion collapse)



**Knee Point**

Throughput increases slowly

Delay increases really fast

**Cliff Point**

Throughput begins to drop to zero  
(Congestion collapse)

Delay approaches infinity

General approaches to congestion control:

* **End-to-End Congestion Control**
  + Congestion calculated by end-system. Approach taken by TCP
  + No assistance from network
* **Network-assisted Congestion Control**
  + Routers provide feedback to end-systems.
  + Gives explicit rate for sender to send at.

**TCP Congestion Control**

TCP’s solution is to use a Congestion Window which controls the no. of packets in flight (un-acked)

* Send control window no. bytes 🡪 wait RTT for ACKS 🡪 send more bytes
* **Sending Rate =~ CWND / RTT** bytes per second
* Sender varies window size CWND to control the sending rate

Types of Congestion Windows

* Sender: **Congestion Window** **CWND**
  + How many bytes can be sent without overflowing routers? Computed by sender.
* Receiver: **Flow Control Window RWND**
  + How many bytes can be sent without overflowing receiver’s buffers? Computer by recv and reported to sender
* **Sender-Side Window = min{CWND , RWND}**

Basic structure of Sender Rate Adjustment:

* Upon receipt of ACK, increase sending rate.
* Upon detection of loss, decrease sending rate.

Rate adjustment depends on the phase of congestion control we’re in:

* Discovering available bottleneck bandwidth vs. Adjusting to bandwidth variations

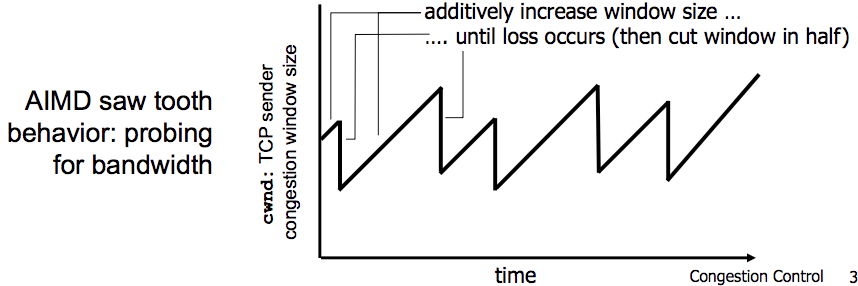
**Bandwidth Discovery with Slow Start (SS)**

* When connection begins, initial rate is slow then increase rate exponentially until the first packet loss event.
  + Initially CWND = 1 MSS (max size seg)
  + Double CWND every RTT or alternate implementation Incrementing CWND for every ACK received

**Adjusting to Bandwidth Variations / Congestion Avoidance**

* Slow start gave an estimate of available bandwidth
* Now, we want to track variations of this available bandwidth, fluctuating around its current value.
  + This involves repeatedly probing (rate increase) and backoff (rate decrease)
* TCP uses **Additive Increase Multiplicative Decrease (AIMD)** congestion avoidance method

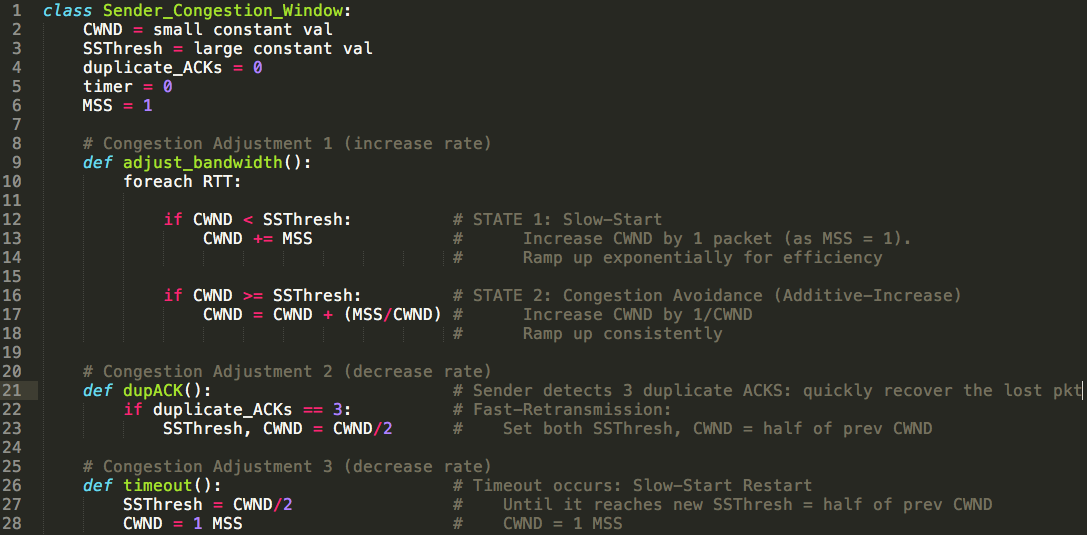
**AIMD**: Sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs

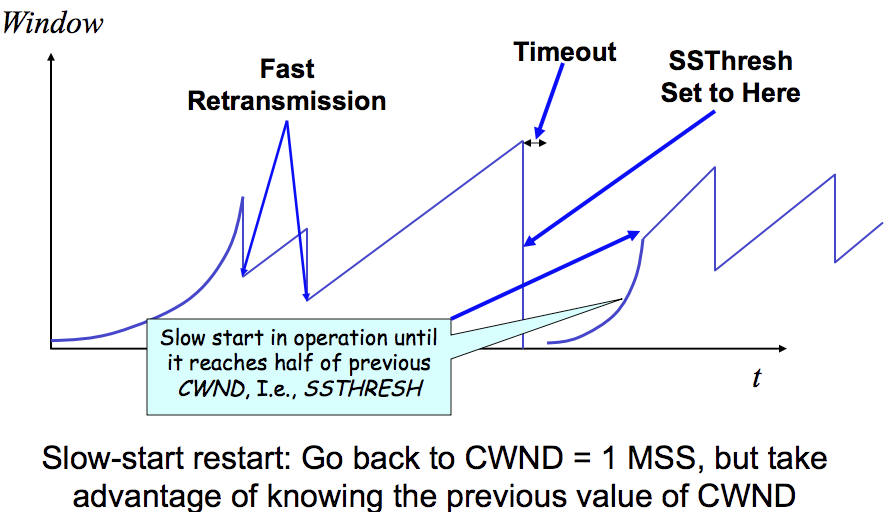
* **Additive Increase**: Increase CWND by 1 MSS every RTT until loss detected.
  + For each successful RTT, CWND = CWND + 1 or for each ACK, CWND = CWND + 1/CWND
* **Multiplicative Decrease**: Cut CWND in half after a loss occurs

This behaviour is called  
the “TCP Sawtooth”

Determine when the sender should do Slow-Start or Congestion Avoidance via. **Slow-Start-Threshold (SSTHRESH)**

* if CWND < SSThresh = Slow-Start
* if CWND >= SSThresh = Congestion Avoidance / Additive-Increase

****

Summary of Congestion Control:

* Rate Increases:
  + **Slow-Start**: CWND = CWND + MSS
  + **Congestion Avoidance**: CWND = CWND + MSS/CWND
* Rate Decreases:
  + **DupACKs**: SSThresh = CWND/2 | CWND = CWND/2
  + **Timeout**: SSThresh = CWND/2 | CWND = 1 MSS

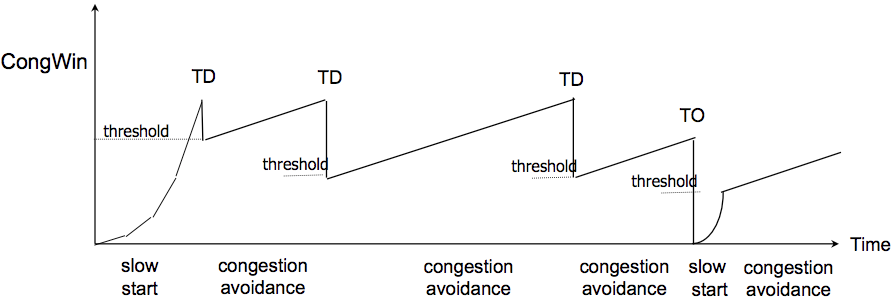
**TCP Flavours**

**TCP Tahoe**

* CWND = 1 on both DupACK / Timeout
* Restarts Slow-Start after both Fast-Retransmit and Timeout

**TCP Reno (same as code example above)**

* CWND = 1 on timeout
* CWND = CWND/2 on DupACK



TCP Reno Graph:

**TD =** Triple DupACK

**TO** = Timeout

**TCP New Reno**

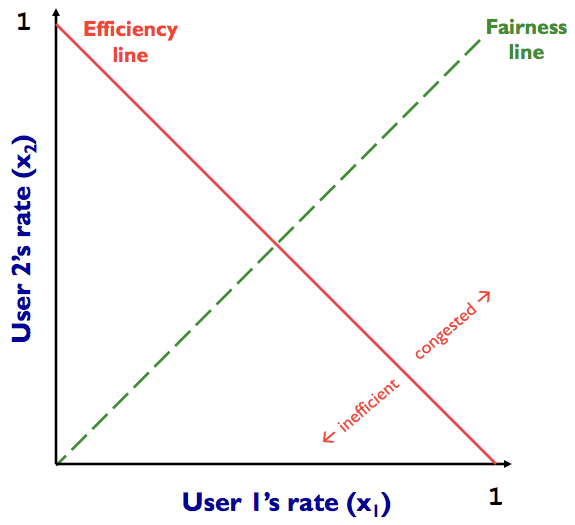
* TCP Reno + improved fast recovery

**PART 2: TCP Fairness**

**Fairness Goal**: If X number of TCP sessions share the same bottleneck link of Bandwidth R, each should have average R/X

**TCP** uses **Additive Increase Multiplicative Decrease (AIMD)**. Four alternative options:

1. AIAD: Gentle increase, gentle decrease
2. AIMD: Gentle increase, drastic decrease
3. MIAD: Drastic increase, gentle decrease
4. MIMD: Drastic increase, drastic decrease

Congestion control model

Two users: **X1 and X2**

**X1 + X2 > 1** = Congestion

**X1 + X2 < 1** = Unused Capacity

**X1 + X2 = 1** = Fair

**A** = Inefficient / Not fair (x1 + x2 = 0.7)

**B** = Efficient / Fair (x1 + x2 = 1)

**C** = Congested / Not fair (x1 + x2 = 1.2)

**D** = Efficient / Not Fair (x1 + x2 = 1)

**AIAD**: Add increase X1,2 | Add decrease X1,2

* Does NOT converge to fairness

**AIMD:** Add increase X1,2 | Mult decrease X1,2,

* Converges to fairness (rates will equalise eventually)

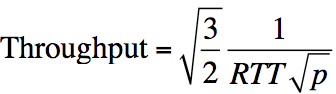
**D**

**C**

**A**

**B**

**Limitations of TCP Congestion Control**



**Limitation 1**: **Different RTTs**

* Max throughput that a TCP connection can reach is **proportional to MSS** and **inversely proportional to RTT**
* Connections with a longer RTT (further from each other) are at a disadvantage to connections with a shorter RTT (closer to each other)
* **When RTT’s are vastly different for each connection, TCP is unfair**

**Limitation 2: Loss NOT due to congestion**

* TCP may confuse packet corruption with congestion.
* The flow will cut its rate even when it is for non-congestion losses

**Throughput =~ 1 / sqrt(p)** where p = packet loss probability

**Limitation 3: Short flows**

* 50% of flows have < 1500B to send
* 80% < 100KB to send
* Short flows never leave the slow start phase = short flows will never attain fair share
* Too few packets to trigger dupACKS
  + Isolated loss may lead to timeouts
  + **A typical timeout value = ~500ms, which will severely impact latency**

**Limitation 4: TCP fills up queues 🡪 long delays**

* A flow deliberately overshoots capacity, until it experiences a drop
* This means that delays are large for everyone
  + Consider a flow transferring 10GB file, sharing a bottleneck link with 10 flows transferring 100B

**Limitation 5: Cheating (three easy ways to cheat)**

1. Increase CWND faster than +1MSS per RTT
2. Opening many connections:  
   A start 10 connections to B vs. D start 1 connection to E 🡪 A will have 10x throughput than D
3. Using large initial CWND