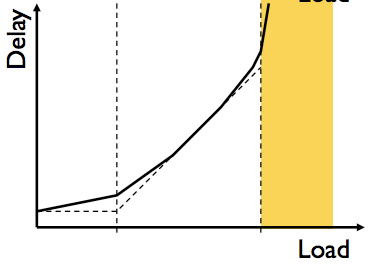
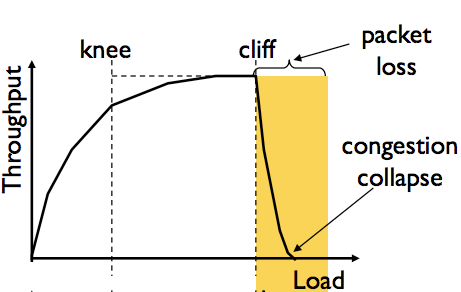
**Congestion Control**

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

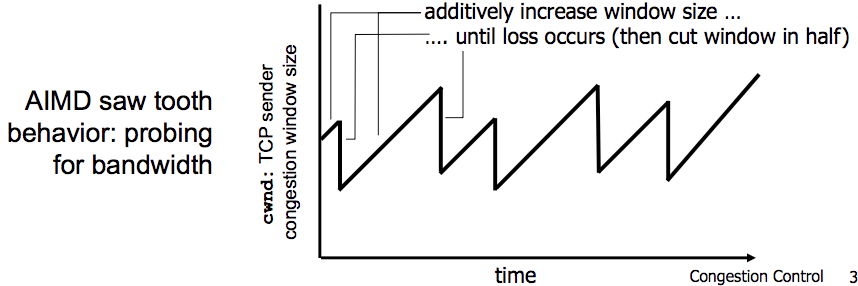
**Bandwidth Discovery with Slow Start (SS)**

* 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

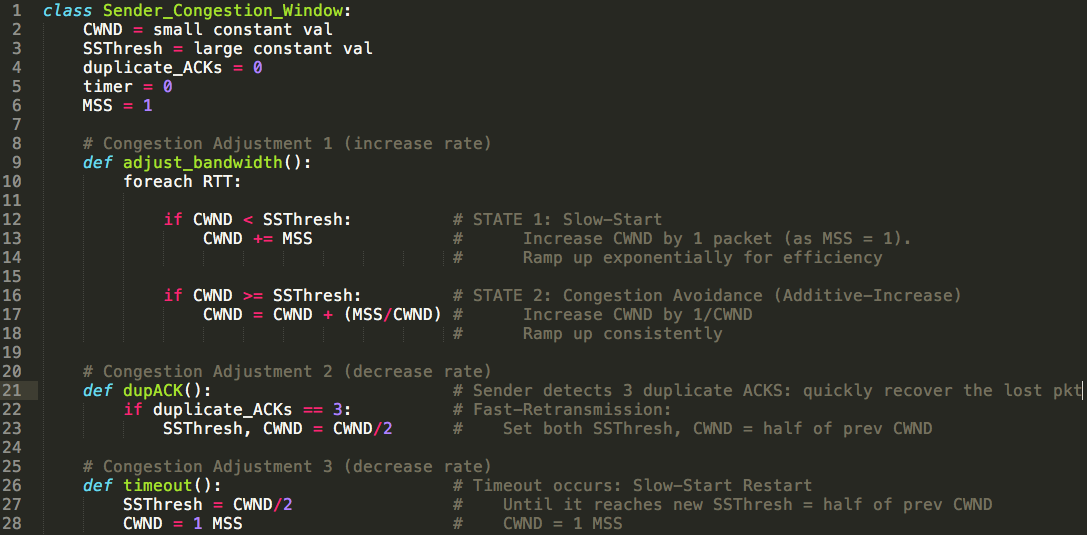
**AIMD**: Sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs

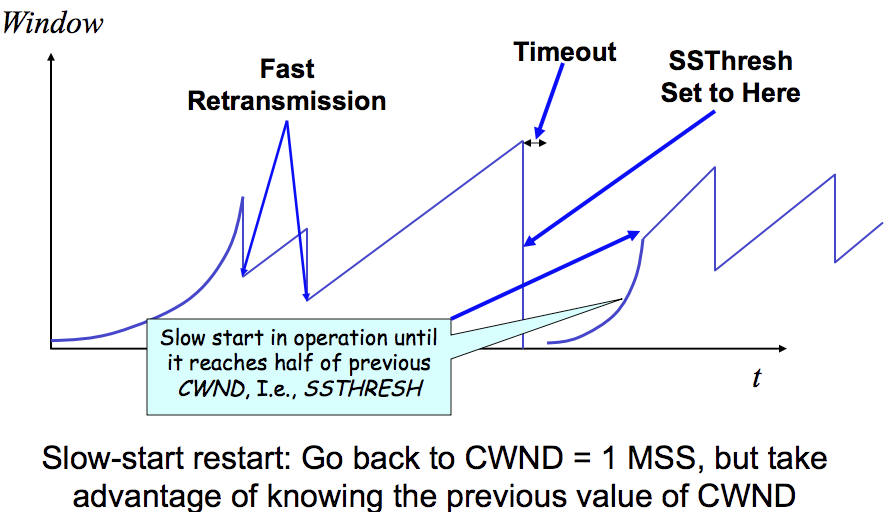
* **Additive Increase**: Increase CWND by 1 MSS every RTT until loss detected.
  + For each successful RTT, CWND = CWND + 1 or for each ACK, CWND = CWND + 1/CWND
* **Multiplicative Decrease**: Cut CWND in half after a loss occurs

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

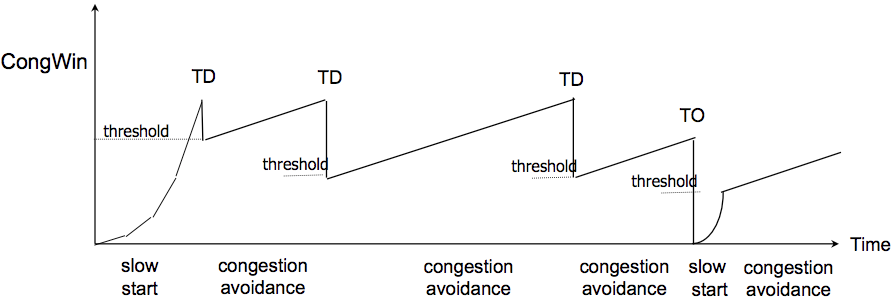
**TCP Flavours**

**TCP Tahoe**

* CWND = 1 on both DupACK / Timeout

**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