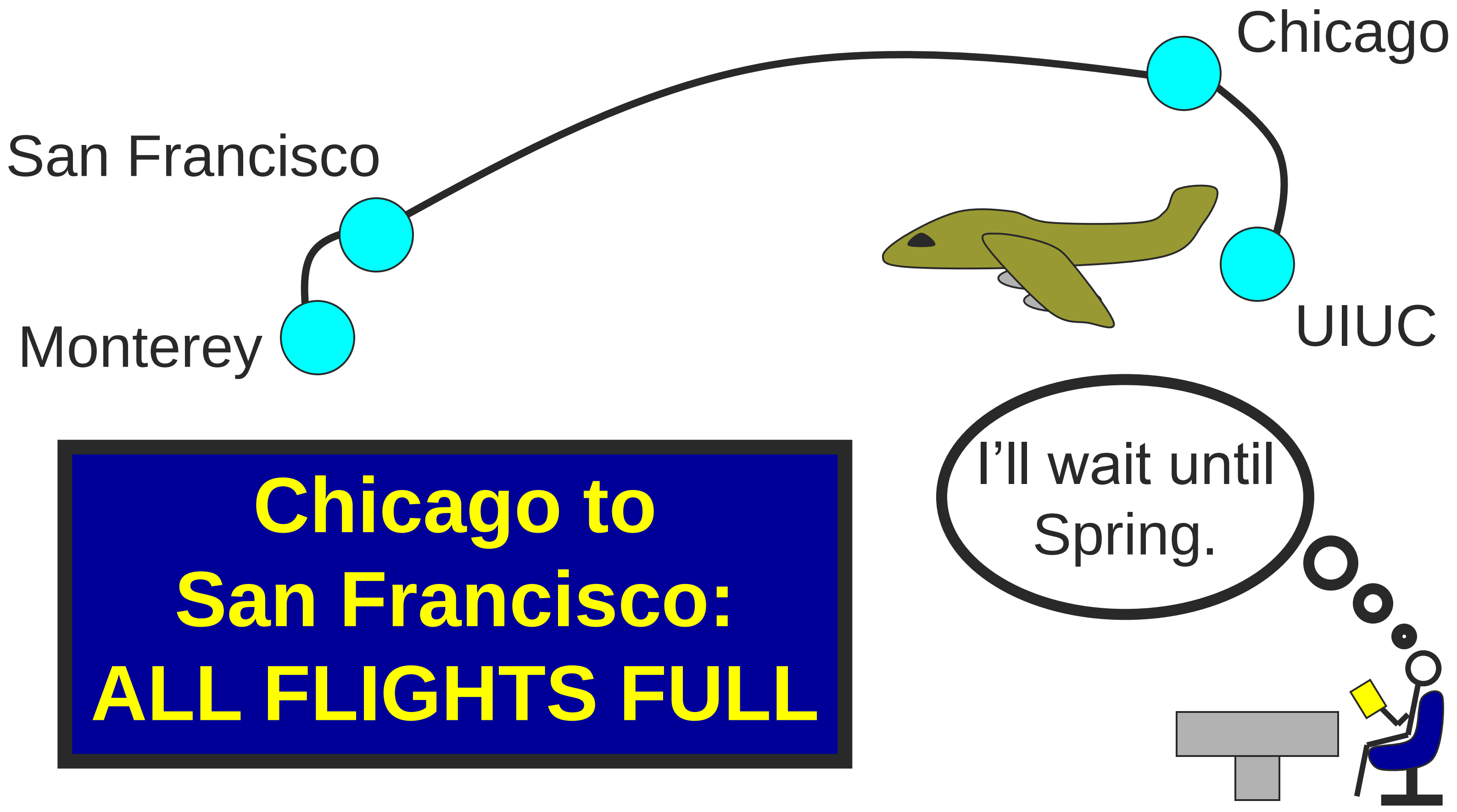
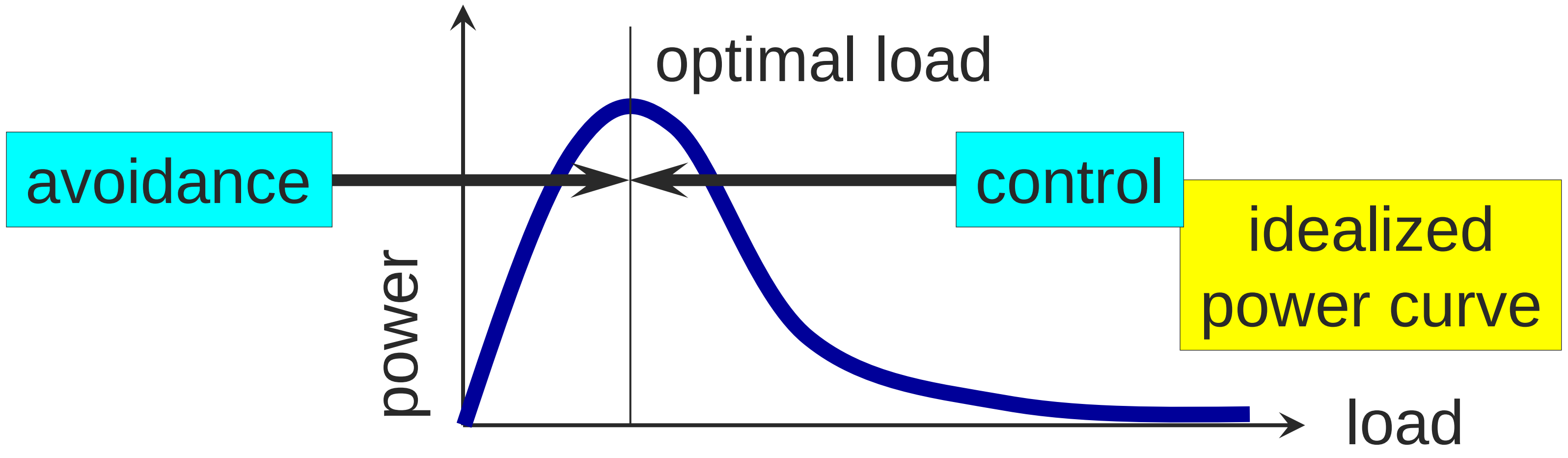
Next Topic: Vacation Planning



Congestion Avoidance

- Control vs. avoidance
  - Control: minimize impact of congestion when it occurs
  - Avoidance: avoid producing congestion
- In terms of operating point limits



Congestion Avoidance

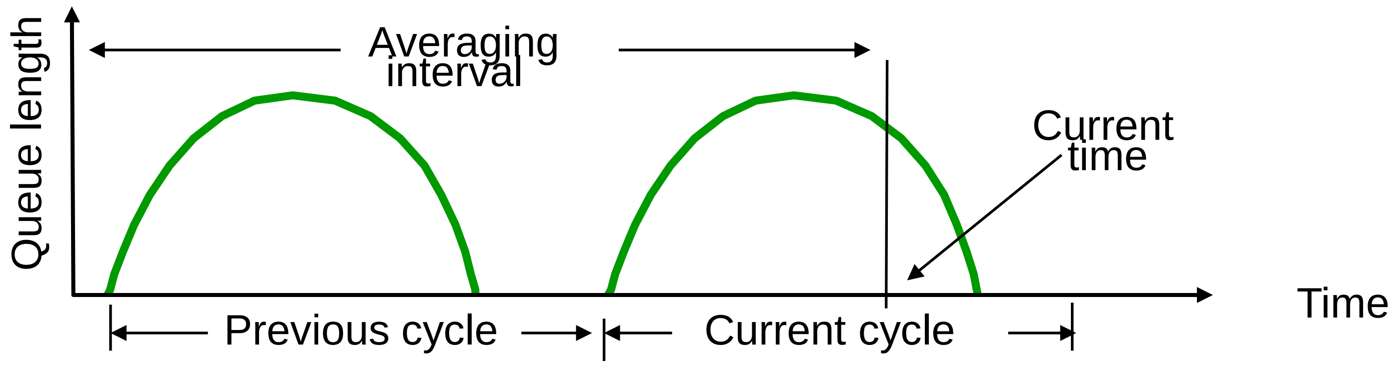
- TCP's strategy
  - Control congestion once it happens
  - Repeatedly increase load in an effort to find the point at which congestion occurs, then back off
- Alternative Strategy
  - Predict when congestion is about to happen and reduce the rate at which hosts send data just before packets start being discarded
  - Congestion avoidance, as compared to congestion control
- Two possibilities
  - Host-centric
    - TCP Vegas (may get some help from routers as in DECbit or via RED gateways)
  - Router-centric
    - Virtual circuits with reserved resources (ATM, RSVP)

DECbit (Destination Experiencing Congestion Bit)

- Developed for the Digital Network Architecture
- Basic idea
  - One bit allocated in packet header
  - Any router experiencing congestion sets bit
  - Destination returns bit to source
  - Source adjusts rate based on bits
- Note that responsibility is shared
  - Routers identify congestion
  - Hosts act to avoid congestion

DECbit

- Router
  - Monitors length over last busy + idle cycle
  - Sets congestion bit if average queue length is greater then 1 when packet arrives
  - Attempts to balance throughput against delay
    - smaller values result in more idle time
    - larger values result in more queueing delay



DECbit

- End Hosts
  - Destination echoes congestion bit back to source
  - Source records how many packets resulted in set bit
  - If less than 50% of last window had bit set
    - Increase **`CongestionWindow`** by 1 packet
  - If 50% or more of last window had bit set
    - Decrease **`CongestionWindow`** by 0.875 percent
- Note:
  - Techniques used in DECbit known as explicit congestion notification (ECN)
  - Proposal to add ECN bit to TCP in progress

Router-Based Congestion Avoidance

- Random Early Detection (RED) gateways
  - Developed for use with TCP
  - Basic idea
    - Implicit rather than explicit notification
    - When a router is “almost” congested
    - Drop packets randomly
  - Responsibility is again shared
    - Router identifies, host acts
    - Relies on TCP’s response to dropped packets

RED Overview

- Observation
  - Transient congestion
    - Should be accommodated for by having large enough queues
  - Longer-lived congestion
    - Reflected as an increase in the average queue size
- Approach
  - Detect incipient congestion from average queue size
    - Upper bound for average queue length

RED Overview

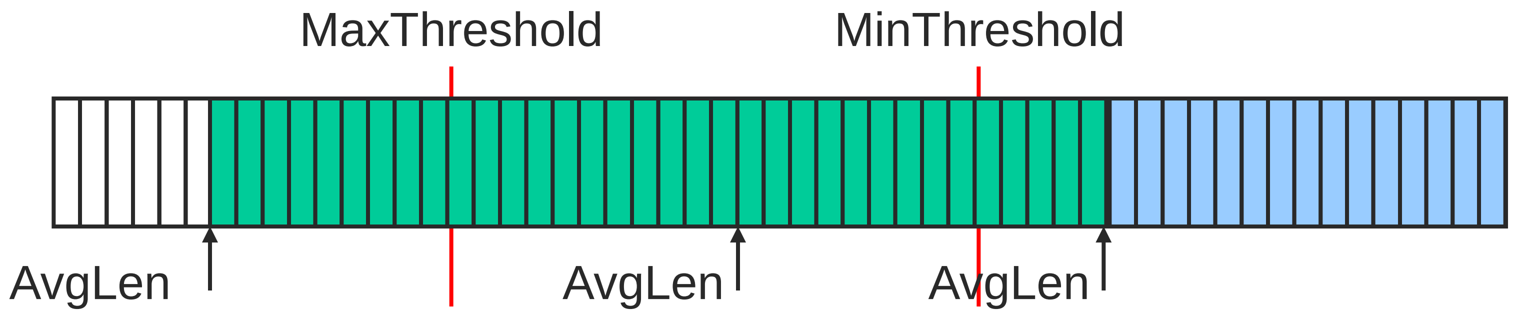
- Notify the end host of congestion
  - Dropping packet
  - Marking packet
- Select connections randomly
  - Avoid global synchronization
- Change dropping probability dynamically
- Avoid bias against bursty data
  - Use average queue length
  - Random marking

Random Early Detection (RED)

- Hosts
  - Implement TCP congestion control
  - Back off when a packet is dropped
- Routers
  - Compute average queue length (exponential moving average)
    - AvgLen = (1 – Weight) * AvgLen + Weight * SampleLen
    - 0 < Weight < 1 (usually 0.002)
    - SampleLen is queue length at packet arrival time

RED – Dropping Policy

- If AvgLen ≤ MinThreshold
  - Enqueue packet
- If MinThreshold < AvgLen < MaxThreshold
  - Calculate P and drop arriving packet with probability P
- If MaxThreshold ≤ AvgLen
  - Drop arriving packet

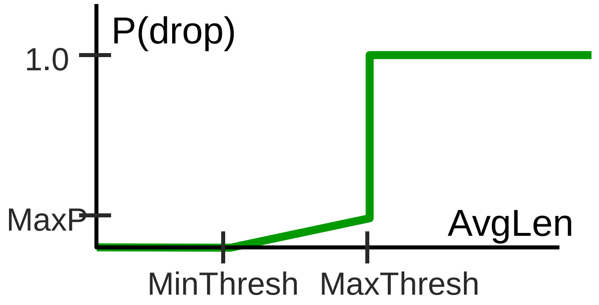


RED – Dropping Probability

- Computing P
  - P is a function of AvgLen and Count
  - Count is the number of packets that have arrived since last reset
  - Reset happens when either a packet is dropped or AvgLen is above MaxThreshold

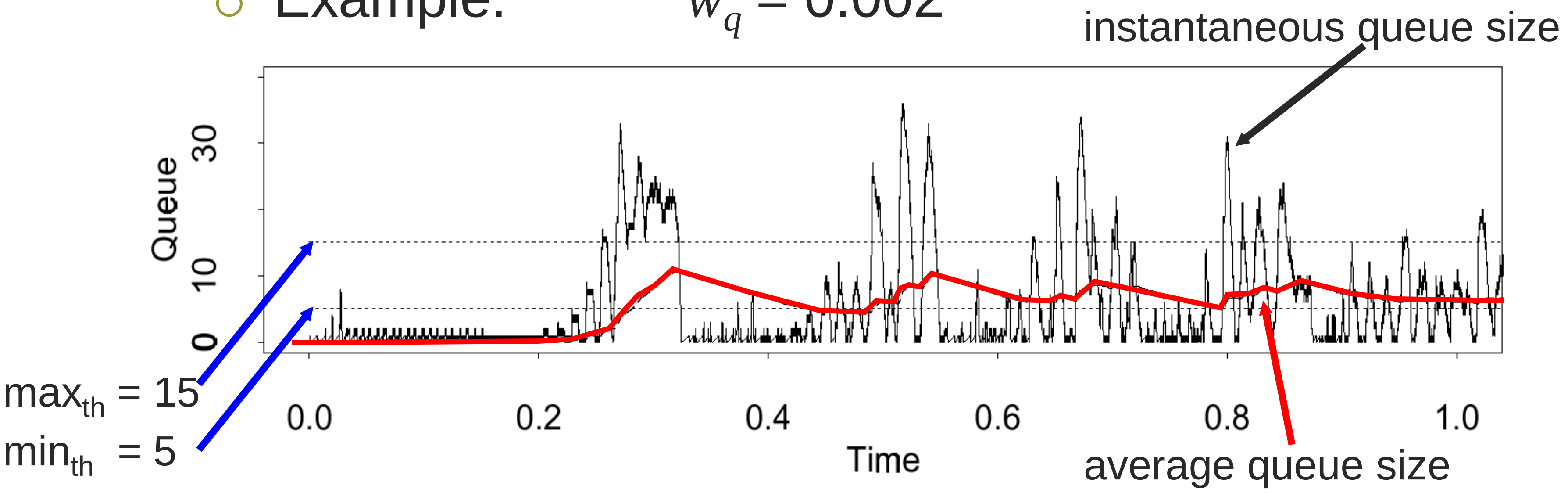
$$\text{TempP} = \frac{(\text{MaxP}) * (\text{AvgLen} - \text{MinThreshold})}{\text{MaxThreshold} - \text{MinThreshold}}$$

$$P = \frac{\text{TempP}}{(1 - \text{count} * \text{TempP})}$$



Calculate Average Queue Size

- Low pass filter $avg \leftarrow (1 - w_q) avg + w_q\, q$
  - If idle: $avg \leftarrow (1 - w_q)^m avg$
  - Example: $w_q = 0.002$



min_{th} and max_{th}

- Determined by the desired average queue size
  - Should be set sufficiently to maximize network utilization
- min_{th}
  - Controls the size of bursts
- max_{th}
  - Depends on the maximum average delay
- max_{th} - min_{th}
  - Should be larger than increase in average queue size in one round trip time
  - Avoid global synchronization

RED Parameters

- MaxP is typically set to 0.02
  - When the average queue size is halfway between the two thresholds, the gateway drops roughly 1 out of 50 packets.
- MinThreshold is typically max/2
- Choosing parameters
  - Carefully tuned to maximize power function
  - Confirmed through simulation
  - But answer depends on accuracy of traffic model

Tuning RED

- Probability of dropping a particular flow's packet(s)
  - Roughly proportional to the that flow's current share of the bandwidth
- If traffic is bursty
  - **`MinThreshold`** should be sufficiently large to allow link utilization to be maintained at an acceptably high level
  - If no buffer space is available, RED uses Tail Drop
- Difference between two thresholds
  - Should be larger than the typical increase in the calculated average queue length in one RTT
  - Setting **`MaxThreshold`** to twice **`MinThreshold`** is reasonable for traffic on today's Internet
- Penalty Box for Offenders

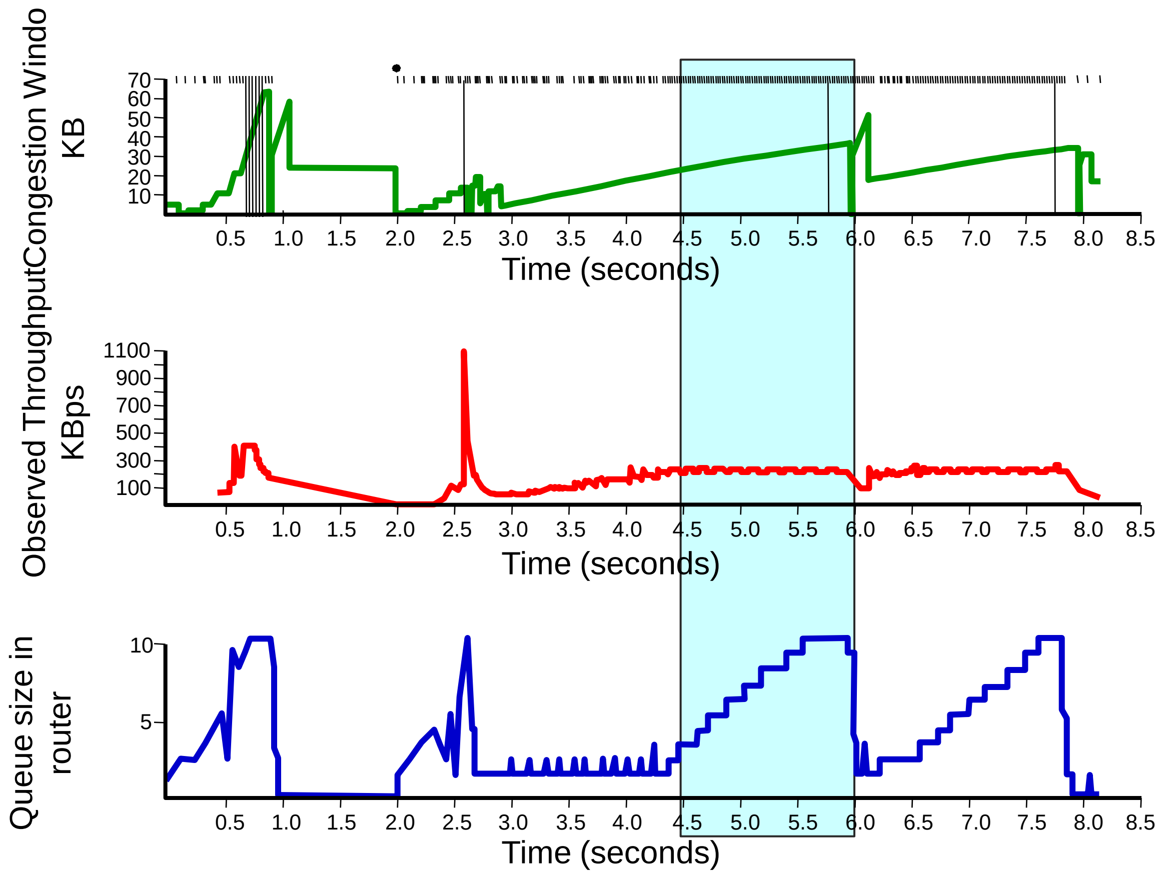
Source-Based Congestion Avoidance

- Idea
  - Source watches for some sign that some router's queue is building up and congestion will happen soon
- Examples
  - RTT is growing
  - Sending rate flattens

Source-Based Congestion Avoidance

- Observe RTT
    - If current RTT is greater than average of minRTT and maxRTT, decrease congestion window by one-eigth
- Observe RTT and Window Size
    - Adjust window once every two RTT
        - If (CurrWindow – OldWindow) * (CurrRTT – OldRTT) > 0, decrease window by one-eigth, otherwise increase window my one MSS
- Observe sending rate
    - Increase window and compare throughput to previous value
- Observe throughput
    - Compare measured throughput with observed throughput
    - TCP Vegas

TCP Vegas



Congestion Window KB
70
60
50
40
30
20
10
Observed Throughput KBps
1100
900
700
500
300
100
Queue size in router
10
5
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
Time (seconds)

TCP Vegas

- Basic idea
  - Watch for signs of queue growth
  - In particular, difference between
    - increasing congestion window
    - stable throughput (presumably at capacity)
  - Keep just enough "extra data" in the network
    - Time to react if bandwidth decreases
    - Data available if bandwidth increases

TCP Vegas

- Implementation
  - Estimate uncongested RTT
    - baseRTT = minimum measured RTT
    - Expected throughput = congestion window / baseRTT
  - Measure throughput each RTT
    - Mark time of sending distinguished packet
    - Calculate data sent between send time and receipt of ACK

TCP Vegas

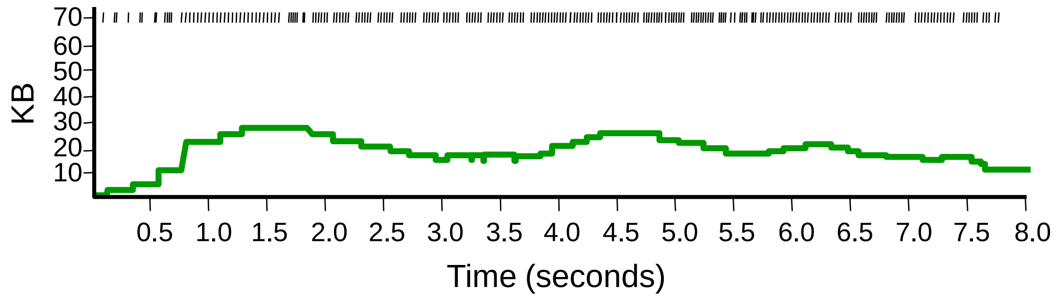
- Act to keep the difference between estimated and actual throughput in a specified range
  - Below minimum threshold
    - Increase congestion window
  - Above maximum threshold
    - Decrease congestion window
- Additive decrease used only to avoid congestion
- Want between 1 and 3 packets of extra data (used to pick min/max thresholds)

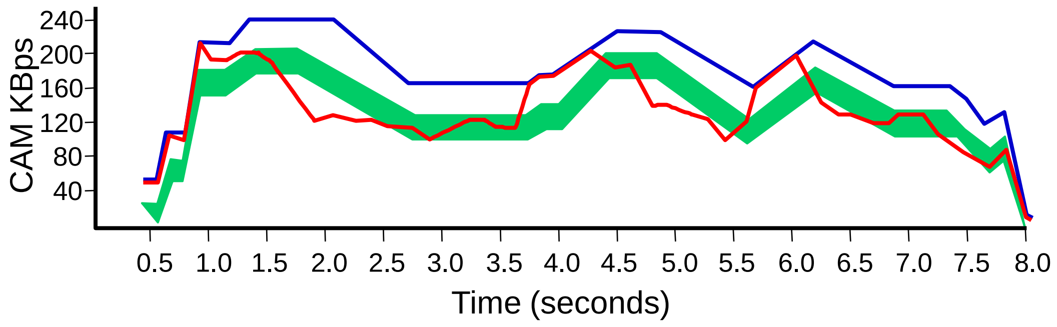
TCP Vegas Algorithm

- Let BaseRTT be minimum of all measured RTTs
  - Commonly the RTT of the first packet
- If not overflowing the connection, then
  - ExpectRate = CongestionWindow/BaseRTT
- Source calculates sending rate (ActualRate) once per RTT
- Source compares ActualRate with ExpectRate
  - Diff = ExpectedRate – ActualRate
  - if Diff < a
    - Increase CongestionWindow linearly
  - else if Diff > b
    - Decrease CongestionWindow linearly
  - else
    - Leave CongestionWindow unchanged

TCP Vegas Algorithm

- Parameters
  - α = 1 packet
  - β = 3 packets
- Even faster retransmit
  - Keep fine-grained timestamps for each packet
  - Check for timeout on first duplicate ACK


KB
Time (seconds)


CAM KBps
Time (seconds)