Congestion Control

Outline

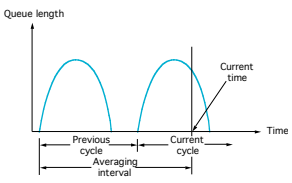
- Congestion Avoidance
- RED
- TCP Vegas

Congestion Avoidance

- TCP's strategy
  - control congestion once it happens
  - repeatedly increase load in an effort to find the point at which congestion occurs, and then back off
- Alternative strategy
  - predict when congestion is about to happen
  - reduce rate before packets start being discarded
  - call this congestion *avoidance*, instead of congestion *control*
- Two possibilities
  - router-centric: DECbit and RED Gateways
  - host-centric: TCP Vegas

DECbit

- Add binary congestion bit to each packet header
- Router
  - monitors average queue length over last busy+idle cycle



  - set congestion bit if average queue length >= 1
  - attempts to balance throughput against delay

End Hosts

- Destination echoes bit back to source
- Source records how many packets resulted in set bit
- If less than 50% of last window's worth had bit set
  - increase **`CongestionWindow`** by 1 packet
- If 50% or more of last window's worth had bit set
  - decrease **`CongestionWindow`** by 0.875 times

Random Early Detection (RED)

- Notification is implicit
  - just drop the packet (TCP will timeout)
  - could make explicit by marking the packet
- Early random drop
  - rather than wait for queue to become full, drop each arriving packet with some *drop probability* whenever the queue length exceeds some *drop level*

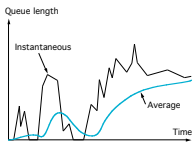
RED Details

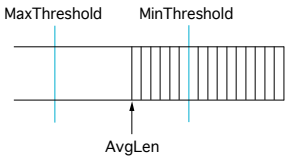
- Compute average queue length

```
AvgLen = (1 - Weight) * AvgLen +
         Weight * SampleLen
```

0 < **`Weight`** < 1 (usually 0.002)

`SampleLen` is queue length each time a packet arrives





RED Details (cont)

- Two queue length thresholds

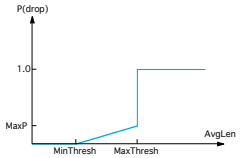
```
if AvgLen <= MinThreshold then
    enqueue the packet
if MinThreshold < AvgLen < MaxThreshold then
    calculate probability P
    drop arriving packet with probability P
if MaxThreshold <= AvgLen then
    drop arriving packet
```


RED Details (cont)

- Computing probability P

```
TempP = MaxP * (AvgLen - MinThreshold)/
   (MaxThreshold - MinThreshold)
P = TempP/(1 - count * TempP)
```

- Drop Probability Curve

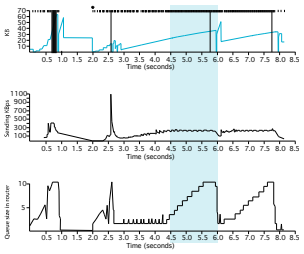


Tuning RED

- Probability of dropping a particular flow's packet(s) is roughly proportional to the share of the bandwidth that flow is currently getting
- **`MaxP`** is typically set to 0.02, meaning that when the average queue size is halfway between the two thresholds, the gateway drops roughly one out of 50 packets.
- If traffic is bursty, then **`MinThreshold`** should be sufficiently large to allow link utilization to be maintained at an acceptably high level
- 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

TCP Vegas

- Idea: source watches for some sign that router's queue is building up and congestion will happen too; e.g.,
  - RTT grows
  - sending rate flattens



Algorithm

- Let **`BaseRTT`** be the minimum of all measured RTTs (commonly the RTT of the first packet)
- If not overflowing the connection, then

  `ExpectedRate = CongestionWindow/BaseRTT`
- Source calculates sending rate (**`ActualRate`**) once per RTT
- Source compares **`ActualRate`** with **`ExpectedRate`**

```
Diff = ExpectedRate - ActualRate
if Diff < α
  increase CongestionWindow linearly
else if Diff > β
  decrease CongestionWindow linearly
else
  leave CongestionWindow unchanged
```


Algorithm (cont)

- Parameters
  - α = 1 packet
  - β = 3 packets