Chapter 6 Congestion Control and Resource Allocation

Congestion-Avoidance Mechanisms

- Congestion avoidance is to predict when congestion is about to happen and then to reduce sending rate of source host before packets start being discarded.
- Two approaches to congestion avoidance
  - With router assistance: DECbit and RED
  - Without router assistance: TCP Vegas
- DECbit
  - Developed for use on the Digital Network Architecture (DNA) created by DEC
  - DNA is a connectionless network with a connection-oriented transport protocol
  - Thus, DECbit can be applied to TCP and IP
  - A congestion bit is added to packet header
  - Router actions
    - Router computes average queue length over last busy + idle cycle, plus the current busy cycle
    - Router sets congestion bit in the packet if average queue length $\geq 1$ when the packet arrives
    - A large queue length threshold leads to high throughput, A small queue length threshold leads to low delay
      - A queue length threshold of 1 optimizes the power function
End host actions

- Destination host copies the congestion bit into the ACK it sends back to the source
- The source host maintains a congestion window, and watches to see what fraction of the last window’s worth of packets resulted in the congestion bit being set
  - If fraction < 50%, increase congestion window by 1 packet.
  - If fraction \( \geq 50\% \), decrease congestion window to 0.875 times previous value
  - A threshold of 50% maximizes the power function

Random Early Detection (RED)

- RED is designed to be used in conjunction with TCP
- Implicit congestion notification: router drops packets when congestion is imminent
  - The source is effectively notified by the subsequent timeout or duplicate ACKs
- Early drop: the router drops a few packets before it has exhausted its buffer space
- Random drop: a router drops each arriving packet with some drop probability when the queue length exceeds a threshold

Details of RED

- Average queue length computation
  - \( \text{AvgLen} = (1 - \text{Weight}) \times \text{AvgLen} + \text{Weight} \times \text{SampleLen} \)
  - \( 0 < \text{Weight} < 1 \), SampleLen is the length of the queue when a sample measurement is made
In software implementations, the queue length is measured every time a new packet arrives at the router. In hardware, it is calculated at some fixed sampling interval.

- The weighted running average calculation tries to detect long-lived congestion.
  - Two queue length thresholds: `MinThreshold` and `MaxThreshold`.
    - If `AvgLen \leq MinThreshold`, queue the packet.
    - If `AvgLen \geq MaxThreshold`, drop the arriving packet.
    - If `MinThreshold < AvgLen < MaxThreshold`, calculate probability `P` and drop the arriving packet with probability `P`.
  
- Computing probability `P`.
  - `P` is a function of both `AvgLen` and how long it has been since the last packet was dropped.
  - `TempP = MaxP \times \frac{(AvgLen - MinThreshold)}{(MaxThreshold - MinThreshold)}`.
  - `P = \frac{TempP}{1 - count \times Temp}`.
  
  - `count` = number of newly arriving packets that have been queued (not dropped), and `AvgLen` has been between the two thresholds.
    - Using `count` in the calculation makes closely spaced drops less likely than widely spaced drops. This ensures a roughly even distribution of drops over time.
- RED provides fair resource allocation because the probability of dropping a particular flow’s packet(s) is roughly proportional to the share of the bandwidth the flow is currently getting at the router

- TCP Vegas
  - TCP Vegas is a source-based congestion avoidance mechanism
    - No assistance from routers
    - Source host detects the incipient stages of congestion before losses occur
  - The goal of TCP Vegas is to maintain the right amount of extra data in the network
    - Extra data is data that the source would not have transmitted had it been trying to match exactly the available bandwidth of the network
    - Too much extra data → long delays and congestion
    - Too little extra data → source cannot respond rapidly enough to transient increases in the available network bandwidth
  - The algorithm
    - Define BaseRTT to be the RTT of a packet when the flow is not congested
      - BaseRTT is set to the minimum of all measured RTTs
    - Calculate expected throughput
      - ExpectedRate = CongestionWindow/BaseRTT
    - Calculate current sending rate, ActualRate, once per RTT
      - ActualRate = K/(t1-t2), t1 is the time when a packet is sent, t2 is the time when its ACK is received, K is the number of bytes sent between t1 and t2
Adjust congestion window based on ActualRate, ExpectedRate, and two thresholds $\alpha$, $\beta$ ($\alpha < \beta$)

- $\text{Diff} = \text{ExpectedRate} - \text{ActualRate}$ ($\text{Diff} \geq 0$)
- If $\text{Diff} < \alpha$, increase CongestionWindow linearly during next RTT (having too little extra data in the network)
- If $\text{Diff} > \beta$, decrease CongestionWindow linearly during next RTT (having too much extra data in the network)
- If $\alpha < \text{Diff} < \beta$, leave CongestionWindow unchanged (having the right amount of extra data in the network)

TCP Vegas uses multiplicative decrease when a timeout occurs

- The linear decrease is an *early* decrease in the congestion window that should happen before congestion occurs and packets start being dropped