CHAPTER 6
Congestion Control

P. Farrell and H. Peyravi
Department of Computer Science
Kent State University
Kent, Ohio 44242
farrell@mcs.kent.edu
http://www.cs.kent.edu/~farrell

Fall 2001
CHAPTER 6

Congestion Control

P. Farrell and H. Peyravi

Department of Mathematics and Computer Science
Kent State University
Kent, Ohio 44242
farrell@cs.kent.edu

Fall 2001
Contents

- Congestion Control Issues
- Queuing Disciplines
- TCP Congestion Control
- Congestion Avoidance Mechanisms
Congestion Control Issues

- Two sides of the same coin
  - pre-allocate resources so that to avoid congestion
  - send data and control congestion if (and when) it occurs

- Two points of implementation
  - hosts at the edges of the network (transport protocol)
  - routers inside the network (queuing discipline)

- Underlying service model
  - best-effort (assume for now)
  - multiple qualities of service (later)
**Connectionless flows**

- sequence of packets sent between source/destination pair
- maintain *soft state* at the routers

**Taxonomy**

- router-centric versus host-centric
- reservation-based versus Feedback-based
- window-based versus rate-based

**Evaluation**

- fairness
- power (ratio of throughput to delay)
Queuing Disciplines

- **First-In-First-Out (FIFO)**
  - does not discriminate between traffic sources

- **Fair Queuing (FQ)**
  - explicitly segregates traffic based on flows
  - ensures no flow captures more than its share of capacity
  - variation: weighted fair queuing (WFQ)

![Round-Robin Service Diagram]
Problem: packets not all the same length
  - really want bit-by-bit round robin
  - not feasible to interleave bits (schedule on packet basis)
  - simulate by determining when packet would finish

For a single flow
  - suppose clock ticks each time a bit is transmitted
  - let $P_i$ denote the length of packet $i$
  - let $S_i$ denote the time when start to transmit packet $i$
  - let $F_i$ denote the time when finish transmitting packet $i$
  - $F_i = S_i + P_i$
  - When does router start transmitting packet $i$?
    - If before router finished packet $i - 1$ from this flow, then immediately after last bit of $i - 1$ ($F_{i-1}$)
    - If no current packets for this flow, then start transmitting when arrives (call this $A_i$)
  - Thus: $F_i = MAX(F_{i-1}, A_i) + P_i$

For multiple flows
  - calculate $F_i$ for each packet that arrives on each flow
  - treat all $F_i$'s as timestamps
  - next packet to transmit is one with lowest timestamp

Not perfect: can’t preempt the packet currently being transmitted
TCP Congestion Control

Idea

- assumes best-effort network (FIFO or FQ routers)
- each source determines network capacity for itself
- uses implicit feedback
- ACKs pace transmission (*self-clocking*)

Challenge

- determining the available capacity in the first place
- adjusting to changes in the available capacity
Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
- New state variable per connection: CongestionWindow
  - limits how much data source has in transit
  \[ \text{MaxWin} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow}) \]
  \[ \text{EffWin} = \text{MaxWin} - (\text{LastByteSent} - \text{LastByteAcked}) \]
- Idea:
  - increase CongestionWindow when congestion goes down
  - decrease CongestionWindow when congestion goes up
- Question: how does the source determine whether or not the network is congested?
- Answer: a timeout occurs
  - timeout signals that a packet was lost
  - packets are seldom lost due to transmission error
  - lost packet implies congestion
**Algorithm:**

- Increment `CongestionWindow` by one packet per RTT (*linear increase*)
- Divide `CongestionWindow` by two whenever a timeout occurs (*multiplicative decrease*)

In practice: increment a little for each ACK

\[
\text{Increment} = \frac{(\text{MSS} \times \text{MSS})}{\text{CongestionWindow}}
\]

\[
\text{CongestionWindow} += \text{Increment}
\]

where MSS is maximum message size
Example trace: sawtooth behavior
**Slow Start**

- **Objective:** determine the available capacity in the first place
- **Idea:**
  - Begin with CongestionWindow = 1 packet
  - Double CongestionWindow each RTT

- Exponential growth, but slower than all in one blast
- Used...
  - When first starting connection
  - When connection goes dead waiting for a timeout
Fast Retransmit and Fast Recovery

- Problem: coarse-grain TCP timeouts lead to idle periods
- Fast retransmit: use duplicate ACKs to trigger retransmission
Congestion Avoidance Mechanisms

TCP's strategy
- to control congestion once it happens
- to 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, and reduce the rate at which hosts send data just before packets start being discarded
- we call this congestion *avoidance*, to distinguish it from congestion *control*

Two possibilities
- router-centric: DECbit and RED Gateways
- host-centric: TCP Vegas
Chapter 6: Congestion Control

**DECbit**

- Add binary congestion bit to each packet header
- Router
  - monitors average queue length over last busy+idle plus current busy cycle
  - set congestion bit if average queue length greater than 1 when packet arrives
  - attempts to balance throughput against delay
- End Hosts
  - destination echos 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, then increase CongestionWindow by 1 packet
  - if 50% or more of last window’s worth had bit set, then decrease CongestionWindow by 0.875 times
Random Early Detection (RED) Gateways

- 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: fills in the details
  - compute average queue length
    \[
    \text{AvgLen} = (1 - \text{Weight}) \times \text{AvgLen} + \text{Weight} \times \text{SampleLen}
    \]
  - \(0 < \text{Weight} < 1\) (usually 0.002)
  - SampleLen is queue length each time a packet arrives

![Graph showing instantaneous and average queue length over time](image-url)
two queue length thresholds

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

MaxThreshold  MinThreshold

AvgLength

probability P

- not fixed
- function of AvgLen and how long since last drop (count)
  keeps track of new packets that have been queued while
  AvgLen has been between the two thresholds

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

- 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 some router’s queue is building up and congestion will happen soon; e.g.,

- RTT is growing
- 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
  \[ \text{ExpectedRate} = \frac{\text{CongestionWindow}}{\text{BaseRTT}} \]

- source calculates current sending rate (ActualRate) once per RTT (read how)
- source compares ActualRate with ExpectedRate

\[ \text{Diff} = \text{ExpectedRate} - \text{ActualRate} \]
if Diff < $\alpha$
  \[ \text{increase CongestionWindow linearly} \]
else if Diff > $\beta$
  \[ \text{decrease CongestionWindow linearly} \]
else
  \[ \text{leave CongestionWindow unchanged} \]
- Parameters
  - $\alpha$: 1 packet
  - $\beta$: 3 packets
- Why not multiplicative decrease?
- Go to multiplicative if there is a timeout