Congestion Control

Outline
- Queuing Discipline
- Reacting to Congestion
- Avoiding Congestion

Issues
- Two sides of the same coin
  - pre-allocate resources so at to avoid congestion
  - control congestion if (and when) is 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)

Framework
- 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 Discipline

- 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)
- Problem?

FQ Algorithm

- 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: $F_i = \text{MAX}(F_{i-1}, A_i) + P_i$

FQ Algorithm (cont)

- 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 current packet
- Example

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: \texttt{CongestionWindow}
  - limits how much data source has in transit

\[
\text{MaxWin} = \min(\text{CongestionWindow}, \text{AdvertisedWindow})
\]
\[
\text{EffWin} = \text{MaxWin} - (\text{LastByteSent} - \text{LastByteAcked})
\]

- Idea:
  - increase \texttt{CongestionWindow} when congestion goes down
  - decrease \texttt{CongestionWindow} when congestion goes up

AIMD (cont)

- 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

AIMD (cont)

- Algorithm
  - increment \texttt{CongestionWindow} by one packet per RTT (linear increase)
  - divide \texttt{CongestionWindow} by two whenever a timeout occurs (multiplicative decrease)

- In practice: increment a little for each ACK
  \[
  \text{Increment} = \frac{\text{MSS} \times \text{MSS}}{\text{CongestionWindow}}
  \]
  \[
  \text{CongestionWindow} += \text{Increment}
  \]
Slow Start

• Objective: determine the available capacity in the first
• Idea:
  – begin with $\text{CongestionWindow} = 1$ packet
  – double $\text{CongestionWindow}$ each RTT (increment by 1 packet for each ACK)

Fast Retransmit and Fast Recovery

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

Results

• Fast recovery
  – skip the slow start phase
  – go directly to half the last successful $\text{CongestionWindow}$ ($ssthresh$)
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

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

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 throughout against delay

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

\[ \text{AvgLen} = (1 - \text{Weight}) \times \text{AvgLen} + \text{Weight} \times \text{SampleLen} \]

\[ 0 < \text{Weight} < 1 \text{ (usually 0.002)} \]

\( \text{SampleLen} \) is queue length each time a packet arrives

\[ \text{MaxThreshold} \]

\[ \text{MinThreshold} \]

\[ \text{AvgLen} \]

RED Details (cont)

• Two queue length thresholds

  if \( \text{AvgLen} \leq \text{MinThreshold} \) then
  enqueue the packet

  if \( \text{MinThreshold} < \text{AvgLen} < \text{MaxThreshold} \) then
  calculate probability \( P \)
  drop arriving packet with probability \( P \)

  if \( \text{MaxThreshold} \leq \text{AvgLen} \) then
  drop arriving packet

RED Details (cont)

• Computing probability \( P \)

\[ \text{TempP} = \frac{\text{MaxP} \times (\text{AvgLen} - \text{MinThreshold})}{(\text{MaxThreshold} - \text{MinThreshold})} \]

\[ P = \frac{\text{TempP}}{1 - \text{count} \times \text{TempP}} \]

• 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

• \( \text{MaxP} \) 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 \( \text{MinThreshold} \) 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 \( \text{MaxThreshold} \) to twice \( \text{MinThreshold} \) is reasonable for traffic on today’s Internet

• Penalty Box for Offenders
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 $\text{BaseRTT}$ 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 sending rate ($\text{ActualRate}$) once per RTT
- Source compares $\text{ActualRate}$ with $\text{ExpectedRate}$
  $$\text{Diff} = \text{ExpectedRate} - \text{ActualRate}$$
  - if $\text{Diff} < \alpha$
    - increase $\text{CongestionWindow}$ linearly
  - else if $\text{Diff} > \beta$
    - decrease $\text{CongestionWindow}$ linearly
  - else
    - leave $\text{CongestionWindow}$ unchanged

Algorithm (cont)

- Parameters
  - $\alpha = 1$ packet
  - $\beta = 3$ packets

- Even faster retransmit
  - keep fine-grained timestamps for each packet
  - check for timeout on first duplicate ACK