Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.892 Fall 1999 **Lecture 4: Congestion Management** September 20, 1999

Overview. Principles and practice in unicast congestion control. Theory of linear control algorithms; TCP and DECbit schemes.

1 Congestion management principles

1.1 The problem

- Resource contention: [what resource(s)?] Caused by (i) overload (ii) heterogeneity. [How's this different from flow control?]
- Throughput vs. load curve. See Figure 1 for what happens when no congestion control is done.
- Why is this important? Else low (zero) throughput for everyone. Congestion collapse (increasing load, diminishing throughputs). (E.g., congestion collapse in mid-1980s in ARPANET).
- Why is this hard? Distributed (incomplete knowledge), changing conditions and traffic, want stability/convergence, want fairness. Want high *power*, the ratio of throughput to delay.

1.2 Terminology

- **Bottleneck:** Draw picture showing a bottleneck link and delays.
- **Power** = Throughput/Delay. Should this be large or small? Ideally, if you had perfect knowledge of the network, what would you make this be for a connection?
- **Efficiency:** want to be as close to available bandwidth as possible.
- **Fairness:** *max-min* definition [Gafni & Bertsekas 1984]: an allocation of a resource is fair if:
    1. no user receives more than their request, ρ_i.
    2. no other allocations satisfying (1) has higher minimum allocation
    3. condition (2) recursively holds as we remove the minimal user and reduce the total resource accordingly.
- This condition reduces to $\mu_i = MIN(\mu_{fair}, \rho_i)$, where μ_i is the allocation for user i and ρ_i is user i's request. Assumes that all users have equal rights to resource.
- **Jain's fairness index:** $(\sum x_i(t))^2/(n \sum x_i(t)^2)$, assuming x_i is user i's allocation and all users are equally greedy for the resource.
- **Distributedness:** no centralized agent. Scales a lot better.

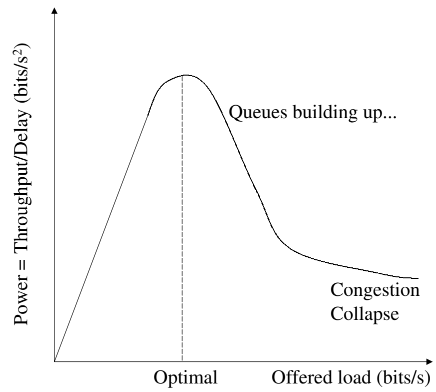

Figure 1: Schematic view of throughput vs. offered load, showing the optimal operating point and congestion collapse region.

- **Convergence:** minimize oscillations and move towards fairness.
- **Congestion control:** react to congestion (reactive control). Key questions: (1) How to decrease rate of transmission upon congestion? (2) How to carefully probe for spare bandwidth?
- **Congestion avoidance:** proactive control. Anticipate it and slow down before it happens (or at least before losses happen). E.g., delay-based (CARD, Vegas), marking-based (DECbit, ECN). Pros and cons? Sometimes need router support. Delay-based schemes often don't work well because of variable delays.
- **Window-based:** Use a (sliding) window; corresponding rate is roughly window/rtt.
- **Rate-based:** No window to modulate, at best an upper bound flow control window. Clock packets out using timers. Key point: need router support in some form to preserve rate. (E.g., packet-pair + fair queueing; explicit rate enforcement at router,...). Also, rate is really 2 parameters, not 1—burst size and burst interval!
- **Router support** for congestion control: FQ, reservations, RED/ECN, backpressure-based hop-by-hop schemes, incentives/disincentives for sources needs to be looked at closely.
- Will hardware advances (faster links, routers,...) solve congestion in the long run, so this is a non-problem?
- NO! Actually could make the problem worse! (diverse apps, more flows, increased heterogeneity). Lots of small flows make the problem a lot harder.
- Congestion management has many hard and open problems!

2 Linear controls

- What form of control? Two aspects: probe for more bandwidth, slow down if you think the network's overloaded.
- In theory, simple enough strategy. But there are some key questions.
  - Congestion signal?
  - How to probe?
  - How to slow down?
  - And remember, want to be efficient, fair, distributed and not oscillate wildly.
- Signal either from router notification or (in TCP today) from loss of packets.
- Literally an infinite solution space! For starters, look at *linear controls* (from control theory).
- $x_i(t+1) = a_I + b_I x_i(t)$ if no congestion signal, and $x_i(t+1) = a_D + b_D x_i(t)$ if congested. Obviously this is linear because the a's and b's are constants.
- Phase plot pictures. In general, n dimensions. $X_1 + X_2 = 1$ is the optimal utilization line. Equifairness line is $X_1 = X_2$. What is underutilization region? Overload? Optimal point?

  What happens to this picture when there are proxy hosts, differently priced flows etc.
- What happens pictorially for the different controls? Additive/multiplicative increase/decrease (AIMD) is what we want because it converges towards fairness (due to AI) and doesn't overload the link (due to MD).
- Efficiency: Means that $\sum x_i(t)$ increases with t as long as there's no congestion. This means that $b_I > 1 - na_I/\sum x_i(t), \forall n, \forall \sum x_i(t)$. Similarly, $b_D < 1 - na_D/\sum x_i(t), \forall n, \forall \sum x_i(t)$. On picture, this goes on 45-degree line through current allocation (assuming distributedness).
- Fairness: Must never decrease. Since $f(x(t)) \leq 1, f(x(t+1)) > f(x(t))$ implies convergence. This yields $a/b \geq 0$. I.e., they must be of the same sign. Show calculation on board. Leads to $a_I \geq 0, b_I \geq 0, a_D \geq 0, 0 \leq b_D < 1$. If pure multiplicative decrease, no change in fairness. Additive increase improves fairness. Additive decrease generally reduces it.
- Distributedness: Means $x_{i+1}(t) > x_i(t)$ when not loaded. Along with previous conditions, this leads to $a_I > 0, b_I \geq 1; a_D = 0, 0 \leq b_D < 1$. Remember, these are *sufficiency* conditions (not necessity). (Assumes no truncation of bandwidth.)
- As it turns out, when users truncate their x_i's, $a_I \geq 1$ and $b_I \geq 1$ (paper has details). What's the best value of b_I? Why? (This ensures quickest convergence to fairness.)
- Explain why only AIMD works for all values of $x_i(t)$ and n in terms of the phase plot. Reemphasize linear control space.
- Summary: upon congestion, reduce total load but don't reduce fairness (pure multiplicative decrease). Try to improve fairness while probing using additive increase. Neat!

3 TCP congestion avoidance and control

- History: series of ARPANET congestion collapses in the mid-1980s. UCB to LBL throughput dropped by 1000X!
- Key principle: *packet conservation.* New packet in network only when you're sure old one has left (received or dropped). In equilibrium, sender-receiver-sender loop has a constant number of packets in transit.
- Seems like TCP should always obey conservation, if it sends out new packets upon receiving ACKs. Yet, congestion collapses were happening. Why?
- Three possible reasons:
  1. Connection doesn't reach equilibrium.
  2. Premature injection of new packets by sender.
  3. Equilibrium can't be reached because of resource limits along path (or wild oscillations occur).
- Contributions of paper:
  1. Slow-start: reaching equilibrium
  2. Conservation at equilibrium: RTO based on rtt variance, Karn's algorithm, exponential timer backoff (we saw this last class).
  3. Adapting to resource limits: congestion avoidance and window reduction upon congestion.
  4. Better loss recovery using fast retransmissions.

3.1 Self clocking and slow-start

- Consider the picture of the "pipe" in Jacobson's paper. Self clocking using ACKs. Data transmissions are strobed by arriving ACKs. [When does self clocking break]?
- For smooth transmissions, need continuous flow of ACKs.
- Q: How to reach bottleneck bandwidth rate? Use slow-start. Draw picture. For each ACK, open up window by one. This increases exponentially with time (so it's not that slow after all!). Code is trivial (extra variable, *cwnd*). (Similar to Jain's CUTE scheme).

3.2 Window increase/decrease algorithms

- Assume losses imply congestion. (Not valid over many links today). Avoids need for gateway fixes.
- Additive-increase/multiplicative-decrease strategy very similar to RJ90. Somewhat different explanation from CJ89 based on stability (not as formal though).
- $L_i = N + \gamma L_{i-1}$. $\gamma > 1$ or γ close to 1 implies L_n goes as $\gamma^n L_0$ to infinity. So, need multiplicative decrease. Use 0.5 (instead of 7/8) because we know $W/2$ "worked" in slow-start phase. [If you knew the router queue size at the bottleneck, how would you reduce the window?]

- Code for this is trivial too (additional *ssthresh* variable). Explain continuous increase by $1/cwnd$ and contrast to DECbit's discontinuous decision process.

3.3 Summary

This paper has had *tremendous* impact and the corresponding implementation is the basis for TCP congestion management in most systems today.

4 DECbit scheme

- Series of papers on congestion avoidance in the mid-80s. This is the key paper. Schemes implemented in Digital Network Architecture (DNA).
- Q: difference between flow and congestion control?
- Q: What's the difference between congestion avoidance and control? This paper is about avoidance (operate around the knee).
- Draw picture of topology.
- At the highest level, scheme has two components: source function and gateway function.
- Overview of DECbit:
  - Sources modulate transmission using a window (not rate).
  - Gateways monitor average queue lengths.
  - When average queue length exceeds threshold, mark bit on packet. This is the congestion indicator.
  - Receivers echo this bit in ACK (simple).
  - Senders monitor frequency of bits; when many marked bits arrive, reduce window; else increase window.
- Why is it called the *binary* feedback scheme?
- Note: no explicit congestion indication messages (like source quench). Why not? (These add to congestion precisely when network resources are scarce!)
- What do routers do?
  1. Detect congestion: mark bits when average exceeds threshold (one).
  2. Averaging filter: Interesting scheme based on regeneration points.
- What do sources do?
  1. Examine feedback bits: look for majority decision in last window of marked bits. The threshold for number of set bits is closely related to queue algorithm for deciding when to mark bits.
  2. Make decision on updating window: when, how often, etc.

3. Increase/decrease algorithms. Use additive-increase/multiplicative-decrease ($\alpha = 1; \beta = 1/8$).

- Goals same as last time: efficiency, fairness, "distributedness," convergence.

4.1 Router mechanism

- Issue #1: How to signal feedback about congestion? Easy, just set congestion indication bit in header of packets.
- Issue #2: to use hysteresis or not in marking bits? I.e., could use two thresholds, T_1 and T_2. If q_a (average queue size) $> T_2$, mark bits, until $q_a < T_1$. Turns out, for power optimization, $T_1 = T_2$.
- Issue #3: How to average?

1. Instantaneous queue lengths? No, oscillates too much.
2. Fixed time averages? No, need to work across many rtts.
3. EWMA? No, has rtt biases (but works better than #2).
4. Time averages between regeneration points: more complex, but doesn't have rtt biases. Draw picture idea. Define busy and idle periods. Previous cycle alone not enough; use previous and current cycles.
5. Problem with scheme: under heavy load, goes as $1/(1-\rho)$ and variance goes as $1/(1-\rho)^3$, so it becomes sluggish and oscillates.

4.2 Source mechanism

- Suppose decision to update window made now (to W_c), at time t when window is W_p. Effect will start becoming visible only one rtt later, after W_p empties. And only an rtt *after* that will we know if we did the right thing, after W_c further packets empty, about 2 rtts from now. That's when the next decision is made. Note: lossy ack path messes this up badly (as q. about this).
- Corollary: at each decision point, use last W_c acks, not all acks since last decision.
- What fraction of marked bits in window signifies congestion? Why? Draw pic. of birth-death process of Markov chain assuming M/M/1 queue. $p_n = (\rho)^n p_0$. So $p_0 = 1 - \rho$. Expected queue length $= \rho/1 - \rho$ means that $\rho = 0.5$ if queue threshold is 1, etc. So look for 50% marked bits at source.
- Use multiplicative-decrease (7/8) additive-increase (1). 7/8 justified from simulation.

4.3 Summary

Interesting scheme with gateways helping in congestion avoidance. Not used any more, but this series of papers laid much of the foundation for TCP congestion management. Next class will look at gateway support in more detail; we'll see similarities (and key differences) between DECbit and RED+ECN.

© 1999 Hari Balakrishnan DO NOT REDISTRIBUTE

In the next two classes, we'll study the gateway side of congestion control in more detail, including RED and fair queueing.