6.896 Lecture 6: Congestion Control Practice

Hari Balakrishnan

September 28, 1998

1 Outline

- **Readings: RJ90, JK88**
- Summary of principles
- DECbit scheme
- TCP congestion control

2 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?
- 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.

2.1 Router mechanisms

- 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.

2.2 Source mechanisms

- 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.

2.3 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.

3 Announcements

- Project suggestions on web page (see class email). You can also pick your own topic if you like.
- Proposals due 10/5/98 (5pm) in room NE43-305 (Elliott).
- Any questions?

4 TCP

- 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
  1. Slow-start: reaching equilibrium
  2. Conservation at equilibrium: Karn's algorithmrtt variance, exponential timer backoff, Karn's algorithm.
  3. Adapting to resource limits: congestion avoidance and window reduction upon congestion.
  4. Better loss recovery using fast retransmissions.

4.1 Self clocking and slow-start

- Picture of pipe. Self clocking using ACKs. Data transmissions strobed by arriving ACKs. Q: when does self clocking break?
- For smooth transmissions, need continuous flow of ACKs. Changed old infrequent ACK policy to current delayed ACK policy.

- Q: How to reach bottleneck bandwidth rate? Use slow-start. Draw picture. For each ACK, open up window by one. Show exponential increase with time (so it's not that slow after all!). Code is trivial (extra variable, *cwnd*). (Similar to Jain's CUTE scheme).

- Q: How much time to reach "optimal" window W? What is the value of this window in terms of bottleneck bw B and rtt R?

4.2 Eliminating spurious retransmissions

- Fix round-trip timer estimation. Karn's algorithm to avoid retransmission ambiguity.

- Use linear deviation term instead of scaled rtt estimate for rto. Using variance term a good idea (cf. Edge's 1983 paper).

- Jacobson/Karels came up with a cheap way of estimating deviation and rtt.

- $rto = rtt + 4mdev$. Why 4? Explain in terms of slow start behavior.

- Exponential backoffs for stability.

4.3 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 differen explanation from CJ89 based on stabiloity (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.

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

4.4 Improving loss recovery

- *Fast retransmit:* Data-driven loss recovery using duplicate ACKs.

- *Fast recovery:* Rather than slow start after above, be in congestion avoidance (we know that a number of packets made it in the previous window).

- *Selective ACK, SACK:* In addition to cumulative ACKs, send info about which later segments reached. Good for large delay-bandwidth product links.

4.5 Summary

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