Computer Networks
ICS 651

- congestion collapse
- congestion control: TCP Reno
- congestion control: TCP Vegas
- congestion control: TCP cubic
- other ways of detecting congestion
- addressing congestion
- router intervention
- Internet Explicit Congestion Notification
- FIFO queueing
- fair queueing

TCP summary so far

- reliable transmission:
  - sequence and acknowledgement numbers
    - counting bytes (not packets)
  - retransmission if no ack is received by timeout
  - checksum for correctness
  - flow control window to regulate sending speed
- ports and demultiplexing
- optimizations: Nagle algorithm and delayed acks
- TCP options
- push and urgent
- next topic: congestion control window

Router Congestion

- assume a fast router
- two gigabit-ethernet links receiving lots of outgoing data
- one (relatively) slow internet link (10 Mb/s uplink speed) sending the outgoing data
- if the two links send more than a combined 10 Mb/s over an extended period, the router buffers will fill up
- eventually the router will have to discard data due to congestion: more data being sent than the line can carry

Congestion Collapse

- assume a fixed timeout
- if I have n bytes/second to send, I send them
- if they get dropped, I retransmit them (total 2n bytes/second, 3n bytes/second, ...)
- when there is congestion, packets get dropped
- if everybody retransmits with fixed timeout, the amount of data sent grows, increasing congestion
  - so some congestion (enough to drop packets) causes more congestion!!!!
- eventually, very little data gets through, most is discarded
- the network is (nearly) down

TCP Reno

- exponential backoff: if retransmit timer was t before retransmission, it becomes 2t after retransmission
- careful RTT measurements give retransmission as soon as possible, but no sooner
- keep a **congestion window**:
  - effective window is lesser of: (1) flow control window, and (2) congestion window
  - congestion window is kept only on the sender, and never communicated between the peers
  - congestion window (cwin) starts at 1 MSS, grows by 1 MSS for every MSS acked: this is the exponential growth phase of the congestion window, called **slow start**
  - on a retransmission, thresh = cwin / 2, and cwin = 1
  - then, use slow start while cwin < thresh
  - then (after cwin >= thresh) for each ack, add to the window the value MSS * newly-acked/window: this adds one MSS to the window for each whole window that is acked (typically, once every RTT) resulting in linear growth
  - fast retransmit is similar -- interesting details at RFC 2001.

RTT Estimate

- RFC 1122, section 4.2.3.1
- RTT estimate must be accurate, or TCP will incorrectly assume that the network is congested
- Karn/Partridge algorithm: don't use retransmitted segments for RTT estimation.
- for accurate RTT estimate, keep a running average of RTTs: $RTTaverage_x = (1 - alpha)\ RTTaverage_{x-1} * alpha\ RTT_x$
- For example, alpha = 0.125 (1/8).
- Could set the timeout to $RTT_x * 2$.
- but also keep track of the variance in RTT

Jacobson/Karels algorithm

- receive an ack with round-trip-time RTT_x
- New estimate: $RTTaverage_x = (1 - alpha)\ RTTaverage_{x-1} + alpha\ RTT_x$
- Deviation average:
- $DevAve_x = (1-beta)\ DevAve_{x-1} + beta\ |RTT_x - RTTAverage_{x-1}|$
- $Timeout_x = u\ RTTaverage_x + phi\ DevAve_x$

  $0 < delta < 1$ (typically, delta = 1/8 for RTTaverage, and 1/4 for Dev)

  u = 1

  phi = 4

TCP Vegas

- Reno detects congestion after it happens
- Reno also causes congestion by increasing the window until congestion occurs
- early congestion detection: as queues get filled in the router, packets take longer, so the RTT increases
- when RTT gets bigger, we can slow our sending
- when RTT gets back to minimum, we can increase our sending
- TCP Vegas had all the features of TCP Reno, plus congestion avoidance when the RTT increased above the minimum
- not standard, but tested to work well
  - works particularly well on networks that would otherwise see large queueing delays

TCP cubic

- window size grows with the cube of the time since the last congestion
  - prior congestion control algorithms define window growth in terms of round-trip times
  - growth based on real time is more fair to congestions with longer RTTs
- $W_{cubic} = C(t - K)^3 + W_{max}$
  - W_{max} is the window size at which congestion is detected
  - $K = \text{cuberoot}(BW_{max} / C)$
  - BW_{max} is the new, small window after congestion is detected
  - window increase from BW_{max} is initially fast, slows down around W_{max}, and as long as no congestion is detected, increases again more and more quickly
- fairness: flows with initially larger W_{max} have a larger K, so increase more slowly
- cubic is now the default for Linux, MacOS, Windows

CUBIC: A New TCP-Friendly High-Speed TCP Variant

Injong Rhee, and Lisong Xu

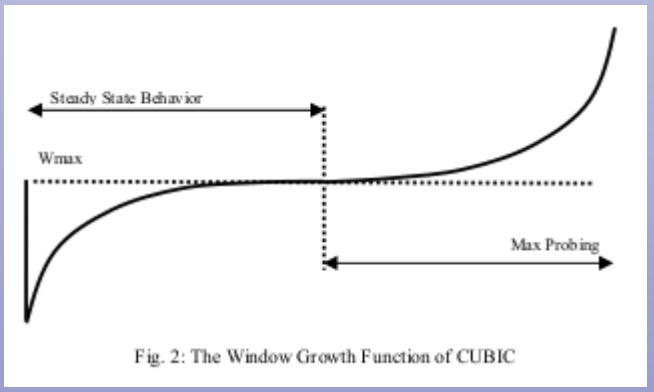

Fig. 2: The Window Growth Function of CUBIC

Detecting and Addressing Congestion

- detecting congestion:
  - queues get longer
  - RTT gets bigger
  - data / RTT (power) starts to drop as you try to send more
- addressing congestion:
  - additive increase/multiplicative decrease (needed for stability if congestion is occurring)
  - additive increase/additive decrease (TCP Vegas) -- works as long as congestion can be avoided
  - setting flow rate
  - bandwidth reservation

Router Intervention to Avoid or React to Congestion

- Random Early Discard (RED) -- causes TCP Reno to back off
- information feed-forward -- the receiver must then return congestion information to the sender (see Internet ECN, below)
- information feedback -- requires route back to sender, does not work in Internet (except source quench ICMP, which is deprecated)
- communication time from router to sender may be insufficient if sender is sending lots of stuff. Also, stability issues -- all senders could increase their sending rate at the same time
- credits: can only send as much as we have in the "bank", automatically (but not immediately) replenished
  - similar to a window

Internet Explicit Congestion Notification

- ECN, explicit congestion notification, RFC 3168.
- in ECN, two of the bits of the Type of Service (ToS) field are used to indicate (a) whether congestion notification is requested (ECT), and (b) whether the packet experienced congestion (CE).
- TCP uses two new bits: ECE (ECN-Echo, to report that a packet was received with the CE bit set -- bit before URG), and CWR (Congestion Window Reduced, bit before ECE), to indicate that the ECE bit was received.
- compatible with hosts and routers that don't do ECN

typical usage of ECN

- senders can set ECT
- routers can change ECT to CE to record that congestion was experienced, perhaps instead of dropping a packet
- transport layer is informed of CE, sends an ECE
- receiver of ECE reduces congestion window, sends CWR

FIFO queueing

- each packet is placed at the end of the queue
- packets (that take the same route) are never reordered
- delay is proportional to queue size
- works reasonably well in Internet, with TCP congestion control
- if all but one sender do congestion control, and one does not, the one that doesn't (IP telephony, multicasting) might grab much of the bandwidth

Fairness

- "everyone" gets the same treatment
- hard to do in a distributed system:
- local fairness (every flow gets the same treatment on this router) discriminates against flows that cross more routers (parking garage problem)
- global fairness requires global co-ordination, so local fairness is often the best we are willing to do

Fair Queueing

- one FIFO queue for each flow
- packets are taken in round-robin order from each queue that has them
- problem: large packets counted the same as small packets
- logically, we want to send one bit from each flow in round-robin order

Fair Queueing with different size packets

- the virtual clock ticks once for each bit sent from each of the queues
- so if there are more active queues, that means the virtual clock advances more slowly
- the virtual finish time for a packet is its start "time" plus the size of the packet
- the virtual start time of a packet is the largest of:
  - the finish time for the previous packet in the queue (a computed quantity), or
  - the actual virtual arrival time of the packet
- to be fair, select and transmit the packet with the lowest virtual finish time