Lecture 12: Congestion Control

12.0 Main Points

Motivation and concepts
TCP approach
Rates and credits

12.1 Motivation

Switch/router can receive more than it can handle: what if
rate of packets arriving for a particular output > bandwidth of outgoing link

Original Internet approach: keep routers simple. Just drop packets if too many arrive. Sender will notice and retransmit.

What can go wrong?

Van Jacobsen -- use dropped packets as signal that host is sending too fast

Drops => overflow => decrease rate by decreasing window size
No drops => no overflow => increase rate by increasing window size

12.2 TCP congestion control

Two parts:
slow start
additive increase/multiplicative decrease

Addititive Increase/Multiplicative decrease

Throughput = window size / RTT

Timeout => dropped packet => cut window size in half

Receive ack => no drop => increase window size by 1 pkt/(# of acks per window)

Oscillates around bottleneck rate
need to continually proble network to see if more bandwidth is available
need to quickly make room if another flow starts up
Slow start
Slow start = multiplicative increase , starting with window size of 1, doubling every round trip until drop signals that we've exceeded bandwidth.

Implications of multiplicative increase during slow start:

1. when reach point where sending faster than bottleneck bandwidth, will exceed by a lot -- often lose most of the packets in the window.

2. highly bursty traffic pattern, will cause drops on other flows sharing bottleneck

As a sliding window protocol, TCP is *self-clocking*. Returning acks key sends.

But

ack compression -- non-uniform gaps between acks because of variations in queueing delay at other links

What happens after a drop and retransmit?

Use slow start (exponential increase from 1) after a timeout to regain self-clocking. When slow start reaches half old window size (that caused drop), start linear increase (called "congestion avoidance").

Similarly, what if sender remembers bottleneck bandwidth to receiver, from the last time they communicated?

Use slow start to ramp up from 1 to half of old bottleneck bandwidth; then increase additively from there.

Do we need a timeout to detect a packet drop?

If see duplicate acks at the sender, implies either:

i) route change & packets out of order => reduce rate

ii) dropped packet => reduce rate and resend

TCP assumes (ii) -- called "fast retransmit". If see 3 duplicate acks, then assume drop, decrease window by 2, and resend "dropped" packet.
Fast recovery -- use duplicate acks to maintain self-clocking. Each duplicate ack means some packet successfully arrived. After retransmitting the missing packet, cut window in half, and start sending new data keyed by the arrival of dup-acks.

Most receivers "delay" acks

wait 200milliseconds, and then ack;
or until another data packet arrives;
or until there is data to be sent along with ack;
but don't wait if data arrived out of order

-- allows reply data to be bundled with ack.
-- BUT! slows down slow start/linear increase
What if two TCP flows share a link?

Regardless of initial bandwidths, will eventually stabilize at equal distribution.

What if one TCP shares a link with one UDP?

one misbehaving flow can cause drops on other correctly behaving flow
Partial solution: weighted fair queueing

Allocate bandwidth fairly among all flows at a router/switch
Drop packets for those flows that have longer queue lengths

With FIFO, throughput is proportional to sending rate
With WFQ, throughput is proportional to link bandwidth / # of flows

TCP Vegas -- use acks to measure delivered bandwidth

adjust window size to match offered to delivered bandwidth

If delivered rate >= offered, increase window size
If delivered rate <= offered, decrease

Alternatives to TCP-style congestion control -- have network explicitly tell host how much it can send.

Two approaches :

1. Rate control (ATM standard). Bottleneck link sends feedback messages upstream :
fair share = bandwidth/# of flows

At host, meters out packets at specified rate, via "leaky bucket"

2. Hop by hop credits (competing ATM standard, also found in Myrinet and other high performance networks): similar to hop by hop sliding window, without retransmission

Upstream switch throttles back if no space for packet at downstream switch.

As packets depart downstream switch, "credit" message flows back to inform upstream switch it can send more data

Unfortunately, can get "head of line blocking" -- traffic that could get through must wait for queued packets to get out of the way

Solutions:
1. Make network so fast, never any congestion (Myrinet solution)
2. per-flow hop-by-hop credits (AN2) -- queue per destination or per connection