CS 268: Lecture 4
(TCP Congestion Control)

Ion Stoica

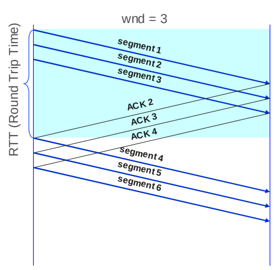
January 30, 2003

Problem

- How much traffic do you send?
- Two components
  - Flow control – make sure that the receiver can receive as fast as you send
  - Congestion control – make sure that the network delivers the packets to the receiver

Flow control: Window Size and Throughput

- Sliding-window based flow control:
  - Higher window → higher throughput
    - Throughput = wnd/RTT
  - Need to worry about sequence number wrapping
- Remember: window size control throughput



Why do You Care About Congestion Control?

- Otherwise you get to congestion collapse
- How might this happen?
  - Assume network is congested (a router drops packets)
  - You learn the receiver didn't get the packet
    - either by ACK, NACK, or Timeout
  - What do you do? retransmit packet
  - Still receiver didn't get the packet
  - Retransmit again
  - …. and so on …
  - And now assume that everyone is doing the same!
- Network will become more and more congested
  - And this with duplicate packets rather than new packets!

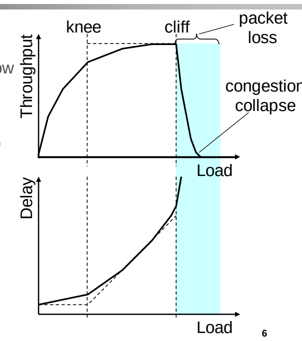
Solutions?

- Increase buffer size. Why not?
- Slow down
  - If you know that your packets are not delivered because network congestion, slow down
- Questions:
  - How do you detect network congestion?
  - By how much do you slow down?

5

What's Really Happening?

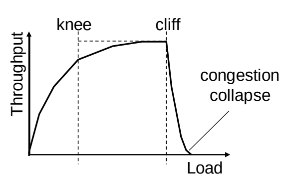
- Knee – point after which
  - Throughput increases very slow
  - Delay increases fast
- Cliff – point after which
  - Throughput starts to decrease very fast to zero (congestion collapse)
  - Delay approaches infinity
- Note (in an M/M/1 queue)
  - Delay = 1/(1 – utilization)



6

Congestion Control vs. Congestion Avoidance

- Congestion control goal
  - Stay left of cliff
- Congestion avoidance goal
  - Stay left of knee



7

Goals

- Operate near the knee point
- Remain in equilibrium
- How to maintain equilibrium?
  - Don't put a packet into network until another packet leaves. How do you do it?
  - Use ACK: send a new packet only after you receive and ACK. Why?
  - Maintain number of packets in network "constant"

8

How Do You Do It?

- Detect when network approaches/reaches knee point
- Stay there
- Questions
  - How do you get there?
  - What if you overshoot (i.e., go over knee point) ?
- Possible solution:
  - Increase window size until you notice congestion
  - Decrease window size if network congested

9

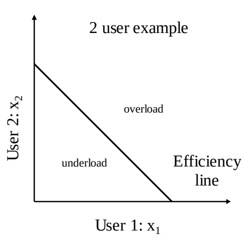
Detecting Congestion

- Explicit network signal
  - Send packet back to source (e.g. ICMP Source Quench)
    - Control traffic congestion collapse
  - Set bit in header (e.g. DEC DNA/OSI Layer 4[CJ89], ECN)
    - Can be subverted by selfish receiver [SEW01]
  - Unless on every router, still need end-to-end signal
  - Could be be robust, if deployed
- Implicit network signal
  - Loss (e.g. TCP Tahoe, Reno, New Reno, SACK)
    - +relatively robust, -no avoidance
  - Delay (e.g. TCP Vegas)
    - +avoidance, -difficult to make robust
  - Easily deployable
  - Robust enough? Wireless?

10

Efficient Allocation

- Too slow
  - Fail to take advantage of available bandwidth $\rightarrow$ underload
- Too fast
  - Overshoot knee $\rightarrow$ overload, high delay, loss
- Everyone's doing it
  - May all under/over shoot $\rightarrow$ large oscillations
- Optimal:
  - $\Sigma x_i = X_{goal}$
- Efficiency = 1 - distance from efficiency line



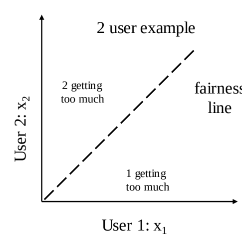
11

Fair Allocation

- Maxmin fairness
  - Flows which share the same bottleneck get the same amount of bandwidth

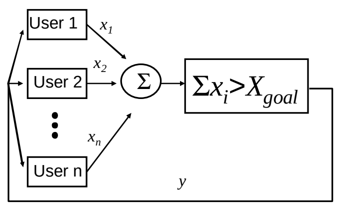
$$F(x) = \frac{\left(\sum x_i\right)^2}{n\left(\sum x_i^2\right)}$$

- Assumes no knowledge of priorities
- Fairness = 1 - distance from fairness line



12

Control System Model [CJ89]



- Simple, yet powerful model
- Explicit binary signal of congestion
  - Why explicit (TCP uses implicit)?
- Implicit allocation of bandwidth

13

Possible Choices

$$x_i(t+1) = \begin{Bmatrix} a_I + b_I x_i(t) \ \ increase \\ a_D + b_D x_i(t) \ decrease \end{Bmatrix}$$

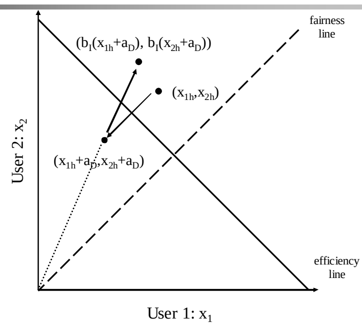
- Multiplicative increase, additive decrease
  - $a_I=0$, $b_I>1$, $a_D<0$, $b_D=1$
- Additive increase, additive decrease
  - $a_I>0$, $b_I=1$, $a_D<0$, $b_D=1$
- Multiplicative increase, multiplicative decrease
  - $a_I=0$, $b_I>1$, $a_D=0$, $0<b_D<1$
- Additive increase, multiplicative decrease
  - $a_I>0$, $b_I=1$, $a_D=0$, $0<b_D<1$
- Which one?

14

Multiplicative Increase, Additive Decrease

- Does not converge to fairness
  - Not stable at all
- Does not converges to efficiency
  - Stable iff

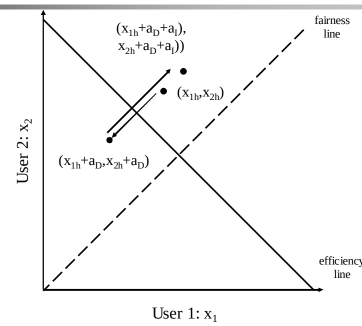
$$x_{1h} = x_{2h} = \frac{b_I a_D}{1 - b_I}$$



15

Additive Increase, Additive Decrease

- Does not converge to fairness
  - Stable
- Does not converge to efficiency
  - Stable iff $a_D = a_I$

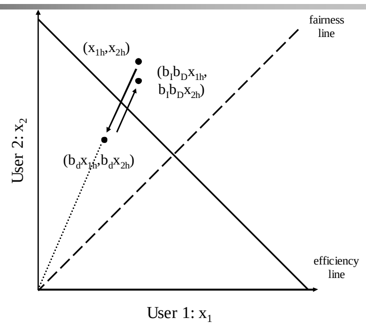


16

Multiplicative Increase, Multiplicative Decrease

- Does not converge to fairness
  - Stable
- Converges to efficiency iff

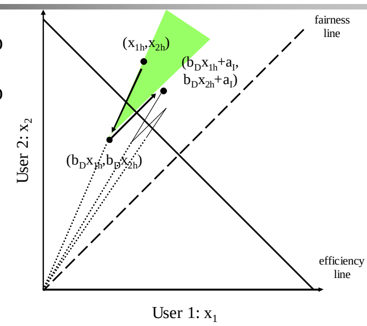
$$b_I \geq 1$$
$$0 \leq b_D < 1$$



17

Additive Increase, Multiplicative Decrease

- Converges to fairness
- Converges to efficiency
- Increments smaller as fairness increases
  - effect on metrics?



18

Significance

- Characteristics
  - Converges to efficiency, fairness
  - Easily deployable
  - Fully distributed
  - No need to know full state of system (e.g. number of users, bandwidth of links) (why good?)
- Theory that enabled the Internet to grow beyond 1989
  - Key milestone in Internet development
  - Fully distributed network architecture requires fully distributed congestion control
  - Basis for TCP

19

Modeling

- Critical to understanding complex systems
  - [CJ89] model relevant for 13 years, 10^6 increase of bandwidth, 1000x increase in number of users
- Criteria for good models
  - Realistic
  - Simple
    - Easy to work with
    - Easy for others to understand
  - Realistic, complex model → useless
  - Unrealistic, simple model → can teach something about best case, worst case, etc.

20

TCP Congestion Contol

- [CJ89] provides theoretical basis
  - Still many issues to be resolved
- How to start?
- Implicit congestion signal
  - Loss
  - Need to send packets to detect congestion
  - Must reconcile with AIMD
- How to maintain equilibrium?
  - Use ACK: send a new packet only after you receive and ACK. Why?
  - Maintain number of packets in network “constant”

21

TCP Congestion Control

- Maintains three variables:
  - cwnd – congestion window
  - flow_win – flow window; receiver advertised window
  - ssthresh – threshold size (used to update cwnd)
- For sending use: win = **min**(flow_win, cwnd)

22

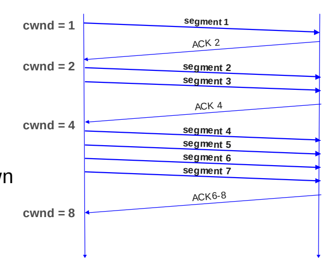
TCP: Slow Start

- Goal: discover congestion quickly
- How?
  - Quickly increase *cwnd* until network congested → get a rough estimate of the optimal of *cwnd*
  - Whenever starting traffic on a new connection, or whenever increasing traffic after congestion was experienced:
    - Set *cwnd* =1
    - Each time a segment is acknowledged increment *cwnd* by one (*cwnd*++).
- Slow Start is not actually slow
  - *cwnd* increases exponentially

23

Slow Start Example

- The congestion window size grows very rapidly
- TCP slows down the increase of *cwnd* when ***cwnd* >= *ssthresh***



24

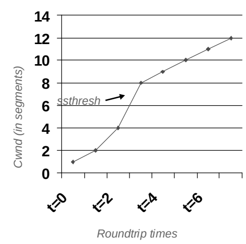
Congestion Avoidance

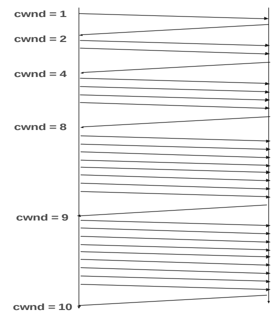
- Slow down "Slow Start"
- **If** *cwnd > ssthresh* **then**
  each time a segment is acknowledged
  increment *cwnd* by *1/cwnd* (*cwnd += 1/cwnd*).
- So *cwnd* is increased by one only if all segments have been acknowlegded.
- (more about *ssthresh* latter)

25

Slow Start/Congestion Avoidance Example

- Assume that *ssthresh = 8*





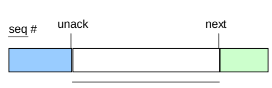
26

Putting Everything Together: TCP Pseudocode

```
Initially:
    cwnd = 1;
    ssthresh = infinite;
New ack received:
    if (cwnd < ssthresh)
        /* Slow Start*/
        cwnd = cwnd + 1;
    else
        /* Congestion Avoidance */
        cwnd = cwnd + 1/cwnd;
Timeout:
    /* Multiplicative decrease */
    ssthresh = cwnd/2;
    cwnd = 1;
```

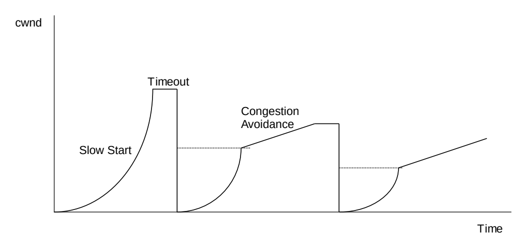
```
while (next < unack + win)
    transmit next packet;

where   win = min(cwnd,
                  flow_win);
```



27

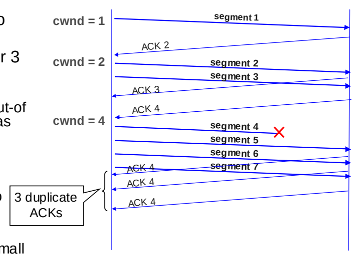
The big picture



28

Fast Retransmit

- Don't wait for window to drain
- Resend a segment after 3 duplicate ACKs
  - remember a duplicate ACK means that an out-of sequence segment was received
- Notes:
  - duplicate ACKs due to packet reordering
    - why reordering?
  - iwindow may be too small to get duplicate ACKs



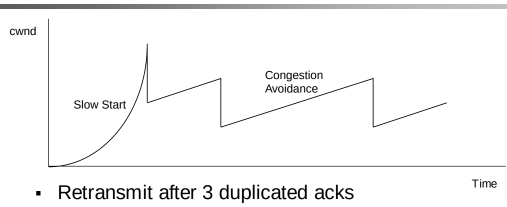
29

Fast Recovery

- After a fast-retransmit set *cwnd* to *ssthresh/2*
  - i.e., don't reset *cwnd* to 1
- But when RTO expires still do *cwnd* = 1
- Fast Retransmit and Fast Recovery → implemented by TCP Reno; most widely used version of TCP today

30

Fast Retransmit and Fast Recovery



- Retransmit after 3 duplicated acks
  - prevent expensive timeouts
- No need to slow start again
- At steady state, *cwnd* oscillates around the optimal window size.

31

Reflections on TCP

- Assumes that all sources cooperate
- Assumes that congestion occurs on time scales greater than 1 RTT
- Only useful for reliable, in order delivery, non-real time applications
- Vulnerable to non-congestion related loss (e.g. wireless)
- Can be unfair to long RTT flows

32