15-744: Computer Networking

L-4 TCP

This Lecture: Congestion Control

- Congestion Control
- Assigned Reading
  - [Chiu & Jain] Analysis of Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks
  - [Jacobson and Karels] Congestion Avoidance and Control

Introduction to TCP

- Communication abstraction:
  - Reliable
  - Ordered
  - Point-to-point
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled
- Protocol implemented entirely at the ends
  - Fate sharing
- Sliding window with cumulative acks
  - Ack field contains last in-order packet received
  - Duplicate acks sent when out-of-order packet received

Key Things You Should Know Already

- Port numbers
- TCP/UDP checksum
- Sliding window flow control
  - Sequence numbers
- TCP connection setup
- TCP reliability
  - Timeout
  - Data-driven

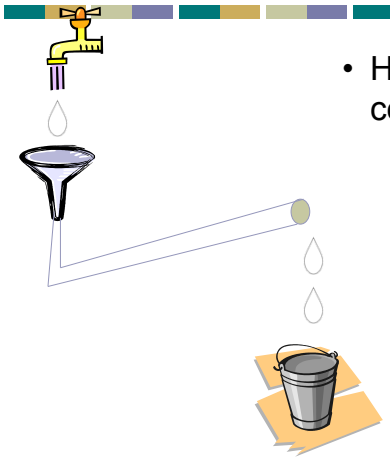
Overview

- Congestion sources and collapse
- Congestion control basics
- TCP congestion control
- TCP modeling

5

Internet Pipes?

- How should you control the faucet?



6

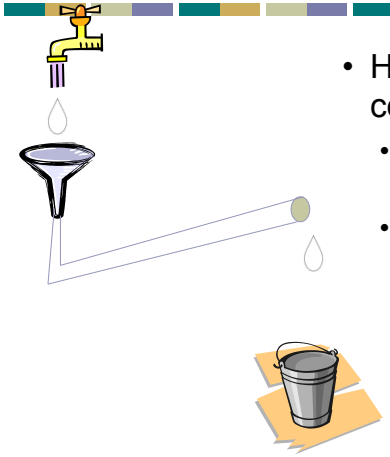
Internet Pipes?

- How should you control the faucet?
  - Too fast – sink overflows!

7

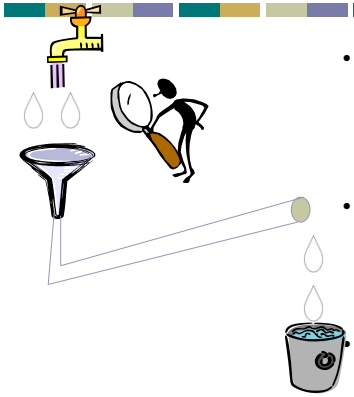
Internet Pipes?

- How should you control the faucet?
  - Too fast – sink overflows!
  - Too slow – what happens?



8

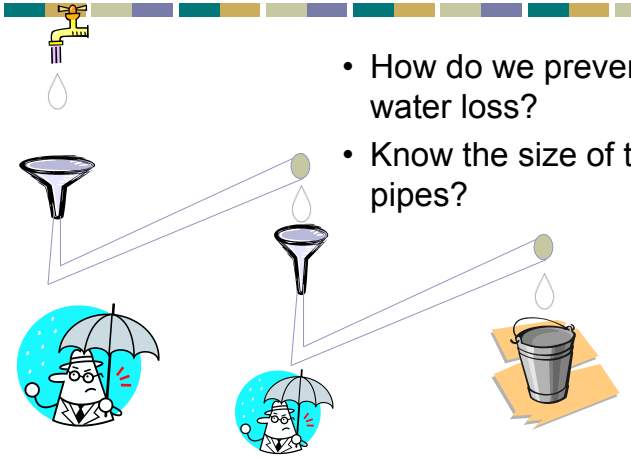
Internet Pipes?



- How should you control the faucet?
  - Too fast – sink overflows
  - Too slow – what happens?
- Goals
  - Fill the bucket as quickly as possible
  - Avoid overflowing the sink
- Solution – watch the sink

9

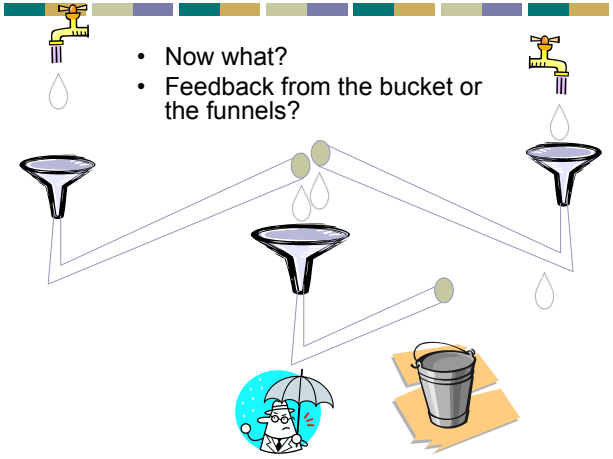
Plumbers Gone Wild!



- How do we prevent water loss?
- Know the size of the pipes?

10

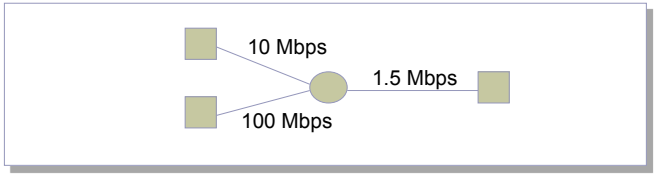
Plumbers Gone Wild 2!



- Now what?
- Feedback from the bucket or the funnels?

11

Congestion



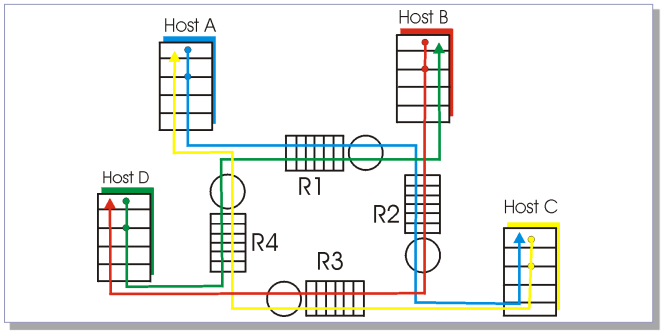
- Different sources compete for resources inside network
- Why is it a problem?
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
  - In many situations will result in < 1.5 Mbps of throughput (congestion collapse)

12

Causes & Costs of Congestion

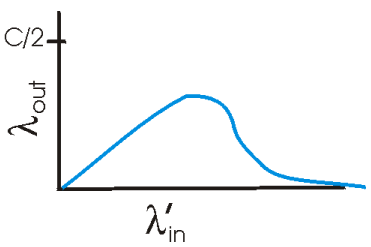
- Four senders – multihop paths
- Timeout/retransmit

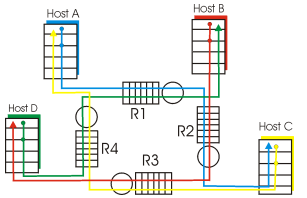
Q: What happens as rate increases?



13

Causes & Costs of Congestion





- When packet dropped, any "upstream transmission capacity used for that packet was wasted!

14

Congestion Collapse

- Definition: *Increase in network load results in decrease of useful work done*
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - Classical congestion collapse
    - How can this happen with packet conservation
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for ALL traffic

15

Other Congestion Collapse Causes

- Fragments
  - Mismatch of transmission and retransmission units
  - Solutions
    - Make network drop all fragments of a packet (early packet discard in ATM)
    - Do path MTU discovery
- Control traffic
  - Large percentage of traffic is for control
    - Headers, routing messages, DNS, etc.
- Stale or unwanted packets
  - Packets that are delayed on long queues
  - "Push" data that is never used

16

Where to Prevent Collapse?

- Can end hosts prevent problem?
  - Yes, but must trust end hosts to do right thing
  - E.g., sending host must adjust amount of data it puts in the network based on detected congestion
- Can routers prevent collapse?
  - No, not all forms of collapse
  - Doesn't mean they can't help
    - Sending accurate congestion signals
    - Isolating well-behaved from ill-behaved sources

17

Congestion Control and Avoidance

- A mechanism which:
  - Uses network resources efficiently
  - Preserves fair network resource allocation
  - Prevents or avoids collapse
- Congestion collapse is not just a theory
  - Has been frequently observed in many networks

18

Approaches Towards Congestion Control

- Two broad approaches towards congestion control:

- End-end congestion control:
  - No explicit feedback from network
  - Congestion inferred from end-system observed loss, delay
  - Approach taken by TCP
- Network-assisted congestion control:
  - Routers provide feedback to end systems
    - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
    - Explicit rate sender should send at
  - Problem: makes routers complicated

19

Example: TCP Congestion Control

- Very simple mechanisms in network
  - FIFO scheduling with shared buffer pool
  - Feedback through packet drops
- TCP interprets packet drops as signs of congestion and slows down
  - This is an assumption: packet drops are not a sign of congestion in all networks
    - E.g. wireless networks
- Periodically probes the network to check whether more bandwidth has become available.

20

Overview

- Congestion sources and collapse
- Congestion control basics
- TCP congestion control
- TCP modeling

21

Objectives

- Simple router behavior
- Distributedness
- Efficiency: $X_{knee} = \Sigma x_i(t)$
- Fairness: $(\Sigma x_i)^2/n(\Sigma x_i^2)$
- Power: $(\text{throughput}^{\alpha}/\text{delay})$
- Convergence: control system must be stable

22

Basic Control Model

- Let's assume window-based control
- Reduce window when congestion is perceived
  - How is congestion signaled?
    - Either mark or drop packets
  - When is a router congested?
    - Drop tail queues – when queue is full
    - Average queue length – at some threshold
- Increase window otherwise
  - Probe for available bandwidth – how?

23

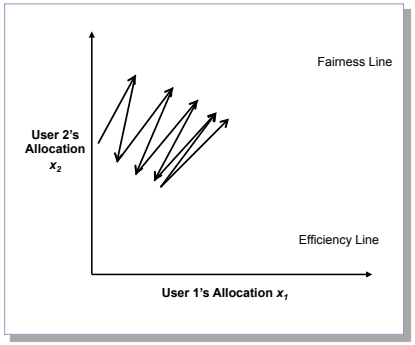
Linear Control

- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
  - Window(t + 1) = a + b Window(t)
  - Different a_i/b_i for increase and a_d/b_d for decrease
- Supports various reaction to signals
  - Increase/decrease additively
  - Increased/decrease multiplicatively
  - Which of the four combinations is optimal?

24

Phase plots

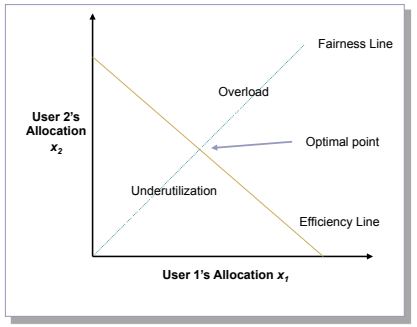
- Simple way to visualize behavior of competing connections over time



25

Phase plots

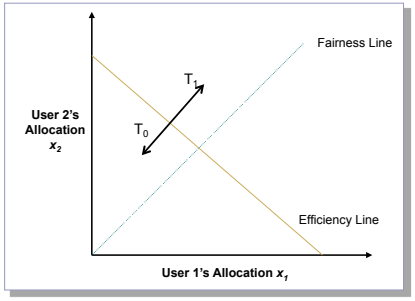
- What are desirable properties?
- What if flows are not equal?



26

Additive Increase/Decrease

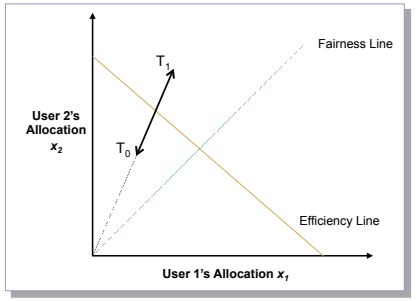
- Both X_1 and X_2 increase/decrease by the same amount over time
  - Additive increase improves fairness and additive decrease reduces fairness



27

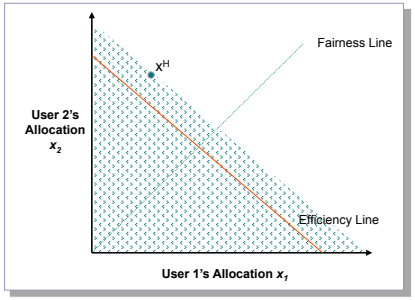
Multiplicative Increase/Decrease

- Both X_1 and X_2 increase by the same factor over time
  - Extension from origin – constant fairness



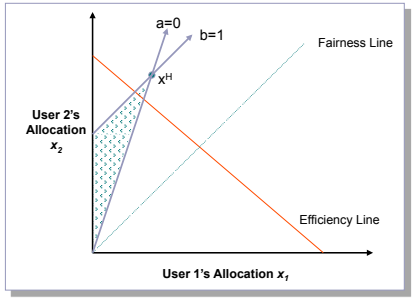
28

Convergence to Efficiency



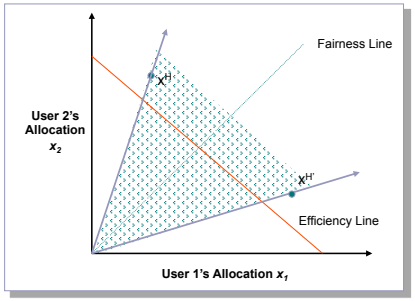
29

Distributed Convergence to Efficiency



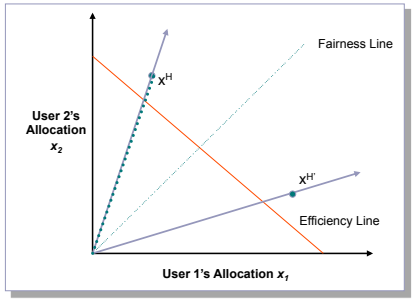
30

Convergence to Fairness



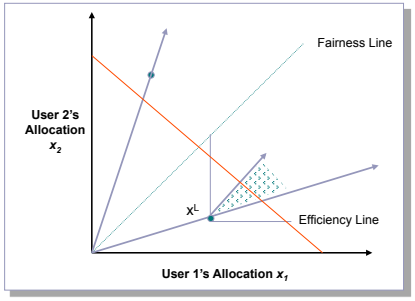
31

Convergence to Efficiency & Fairness



32

Increase



33

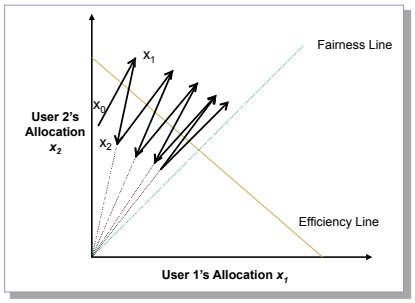
Constraints

- Distributed efficiency
  - I.e., Σ Window(t+1) > Σ Window(t) during increase
    - $a_i > 0$ & $b_i \geq 1$
    - Similarly, $a_d < 0$ & $b_d \leq 1$
- Must never decrease fairness
  - a & b's must be ≥ 0
  - $a_i/b_i > 0$ and $a_d/b_d \geq 0$
- Full constraints
  - $a_d = 0$, $0 \leq b_d < 1$, $a_i > 0$ and $b_i \geq 1$

34

What is the Right Choice?

- Constraints limit us to AIMD
  - Can have multiplicative term in increase (MAIMD)
  - AIMD moves towards optimal point



35

Questions

- Fairness – why not support skew → AIMD/GAIMD analysis
- Delayed feedback → ?
- More bits of feedback → DECbit, XCP, Vegas
- Guess # of users → hard in async system, look at loss rate?
- Stateless vs. stateful design
- Wired vs. wireless
- Non-linear controls → Bionomial

36

Overview

- Congestion sources and collapse
- Congestion control basics
- TCP congestion control
- TCP modeling

TCP Congestion Control

- Motivated by ARPANET congestion collapse
- Underlying design principle: packet conservation
  - At equilibrium, inject packet into network only when one is removed
  - Basis for stability of physical systems
- Why was this not working?
  - Connection doesn't reach equilibrium
  - Spurious retransmissions
  - Resource limitations prevent equilibrium

TCP Congestion Control - Solutions

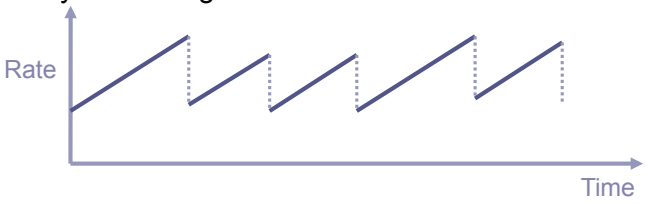
- Reaching equilibrium
  - Slow start
- Eliminates spurious retransmissions
  - Accurate RTO estimation
  - Fast retransmit
- Adapting to resource availability
  - Congestion avoidance

TCP Congestion Control

- Changes to TCP motivated by ARPANET congestion collapse
- Basic principles
  - AIMD
  - Packet conservation
  - Reaching steady state quickly
  - ACK clocking

AIMD

- Distributed, fair and efficient
- Packet loss is seen as sign of congestion and results in a multiplicative rate decrease
  - Factor of 2
- TCP periodically probes for available bandwidth by increasing its rate



41

Implementation Issue

- Operating system timers are very coarse – how to pace packets out smoothly?
- Implemented using a congestion window that limits how much data can be in the network.
  - TCP also keeps track of how much data is in transit
- Data can only be sent when the amount of outstanding data is less than the congestion window.
  - The amount of outstanding data is increased on a “send” and decreased on “ack”
  - (last sent – last acked) < congestion window
- Window limited by both congestion and buffering
  - Sender’s maximum window = Min (advertised window, cwnd)

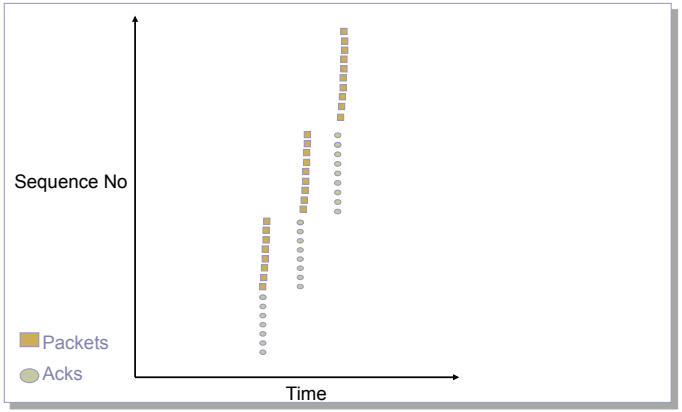
42

Congestion Avoidance

- If loss occurs when cwnd = W
  - Network can handle 0.5W ~ W segments
  - Set cwnd to 0.5W (multiplicative decrease)
- Upon receiving ACK
  - Increase cwnd by (1 packet)/cwnd
    - What is 1 packet? → 1 MSS worth of bytes
    - After cwnd packets have passed by → approximately increase of 1 MSS
- Implements AIMD

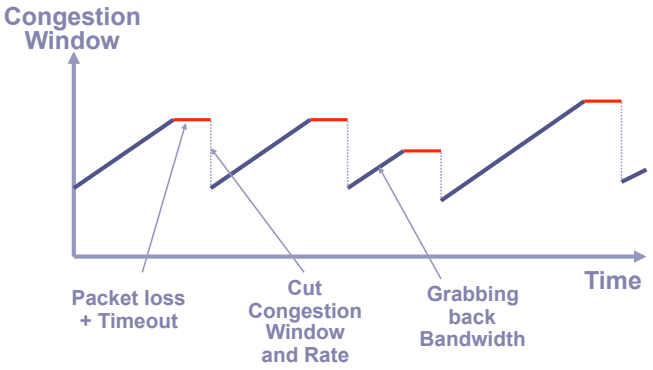
43

Congestion Avoidance Sequence Plot



44

Congestion Avoidance Behavior



45

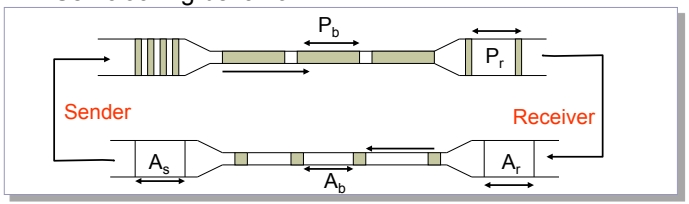
Packet Conservation

- At equilibrium, inject packet into network only when one is removed
  - Sliding window and not rate controlled
  - But still need to avoid sending burst of packets → would overflow links
    - Need to carefully pace out packets
    - Helps provide stability
- Need to eliminate spurious retransmissions
  - Accurate RTO estimation
  - Better loss recovery techniques (e.g. fast retransmit)

46

TCP Packet Pacing

- Congestion window helps to "pace" the transmission of data packets
- In steady state, a packet is sent when an ack is received
  - Data transmission remains smooth, once it is smooth
  - Self-clocking behavior



47

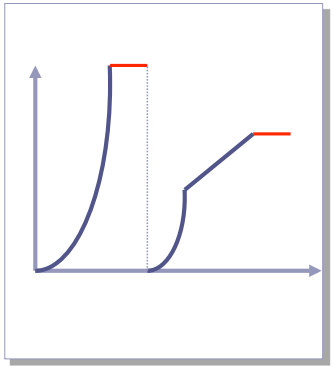
Reaching Steady State

- Doing AIMD is fine in steady state but slow…
- How does TCP know what is a good initial rate to start with?
  - Should work both for a CDPD (10s of Kbps or less) and for supercomputer links (10 Gbps and growing)
- Quick initial phase to help get up to speed (slow start)

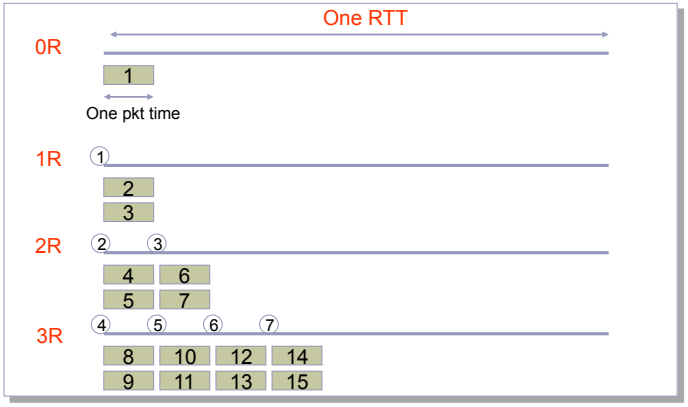
48

Slow Start Packet Pacing

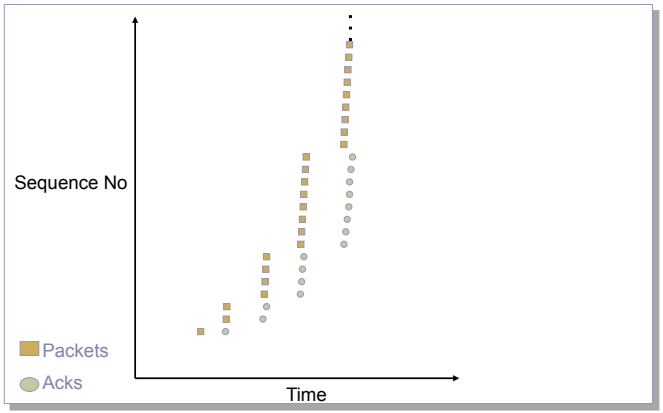
- How do we get this clocking behavior to start?
  - Initialize cwnd = 1
  - Upon receipt of every ack, cwnd = cwnd + 1
- Implications
  - Window actually increases to W in RTT * $\log_2(W)$
  - Can overshoot window and cause packet loss



Slow Start Example



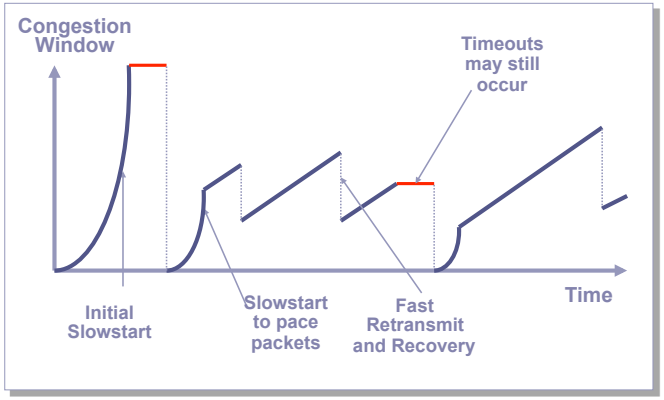
Slow Start Sequence Plot



Return to Slow Start

- If packet is lost we lose our self clocking as well
  - Need to implement slow-start and congestion avoidance together
- When timeout occurs set ssthresh to 0.5w
  - If cwnd < ssthresh, use slow start
  - Else use congestion avoidance

TCP Saw Tooth Behavior



53

How to Change Window

- When a loss occurs have W packets outstanding
- New cwnd = 0.5 * cwnd
  - How to get to new state?

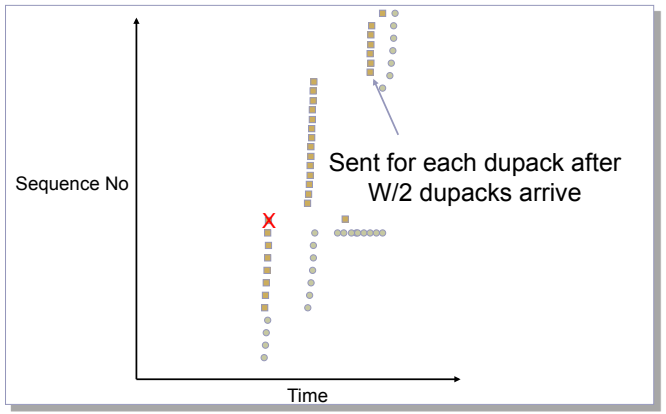
54

Fast Recovery

- Each duplicate ack notifies sender that single packet has cleared network
- When < cwnd packets are outstanding
  - Allow new packets out with each new duplicate acknowledgement
- Behavior
  - Sender is idle for some time – waiting for ½ cwnd worth of dupacks
  - Transmits at original rate after wait
    - Ack clocking rate is same as before loss

55

Fast Recovery



56

Questions

- Current loss rates – 10% in paper
- Uniform reaction to congestion

Next Lecture

- Fair-queueing
- Assigned reading
  - [Demers, Keshav, Shenker] Analysis and Simulation of a Fair Queueing Algorithm
  - [Stoica, Shenker, Zhang] Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks*

Class Project

- End goal → workshop quality paper
  - 6-8 pages
  - Imagine early versions of the paper you have read so far
  - Need not be experimental/system building
  - Must have some experimental/simulation/theoretical results
  - Must be practical/network oriented in nature ☺

Class Project

- Group size → preferably 2
- Project meetings (2 during semester)
  - 15 min meetings to discuss project ideas and get feedback
  - Project idea list posted --- will be updated
- Proposal (1-2pg)
  - Basic idea
  - Description of some related work
  - Rough timeline
  - Necessary/requested resources
- Checkpoint (date TBD – roughly 1month away)
  - Should have preliminary experiments done

Project Ideas

- Relation between RED and small buffers.
  - Recent work (McKweon) has suggested that routers don't need large buffers to support good TCP performance. However, earlier work on RED seems quite similar - are they really so different? Nick McKweon seems to think so. However, they look the same to me. Aren't the correct tuning parameters for RED just the same as the size of the right buffer for small buffer networks. Isn't the tradeoff of "fear of underutilization" vs. amount of buffer/delay the same?
- Relation between TCP and desync
  - All this small buffer stuff seems to rely on a collection of TCP flows becoming desynchronized. Earlier work assumed that this never happened. RED really made the assumption that this never happened and, thus, introduced randomized losses. McKweon's measurements suggest that it does happen but there seems little sound justification for when this happens.

Project Ideas

- Why not duplicate/encode early packets in a TCP connection?
  - Everyone seems to show how their TCP does better. But results are often dominated by timeouts on flows early on. Why not just duplicate the early part of the transfer multiple times or just be more aggressive early on? What would be the overall impact on Internet workload?

Project Ideas

- Congestion control for sensors
  - Is the Sigcomm paper from USC right? Do we really need to specialize congestion control for tree topologies or can we get something like TCP or XCP to work well in multihop wireless environments?

EXTRA SLIDES

The rest of the slides are FYI

TCP Vegas Slow Start

- ssthresh estimation via packet pair
- Only increase every other RTT
  - Tests new window size before increasing

Packet Pair

- What would happen if a source transmitted a pair of packets back-to-back?
- Spacing of these packets would be determined by bottleneck link
  - Basis for ack clocking in TCP
- What type of bottleneck router behavior would affect this spacing
  - Queuing scheduling

Packet Pair

- FIFO scheduling
  - Unlikely that another flows packet will get inserted in-between
  - Packets sent back-to-back are likely to be queued/forwarded back-to-back
  - Spacing will reflect link bandwidth
- Fair queuing
  - Router alternates between different flows
  - Bottleneck router will separate packet pair at exactly fair share rate

Packet Pair in Practice

- Most Internet routers are FIFO/Drop-Tail
- Easy to measure link bandwidths
  - Bprobe, pathchar, pchar, nettimer, etc.
- How can this be used?
  - NewReno and Vegas use it to initialize ssthresh
  - Prevents large overshoot of available bandwidth
  - Want a high estimate – otherwise will take a long time in linear growth to reach desired bandwidth

TCP Vegas Congestion Avoidance

- Only reduce cwnd if packet sent after last such action
  - Reaction per congestion episode not per loss
- Congestion avoidance vs. control
- Use change in observed end-to-end delay to detect onset of congestion
  - Compare expected to actual throughput
  - Expected = window size / round trip time
  - Actual = acks / round trip time

TCP Vegas

- If actual < expected < actual + α
  - Queues decreasing → increase rate
- If actual + α < expected < actual + β
  - Don't do anything
- If expected > actual + β
  - Queues increasing → decrease rate before packet drop
- Thresholds of α and β correspond to how many packets Vegas is willing to have in queues

TCP Vegas

- Fine grain timers
  - Check RTO every time a dupack is received or for "partial ack"
  - If RTO expired, then re-xmit packet
  - Standard Reno only checks at 500ms
- Allows packets to be retransmitted earlier
  - Not the real source of performance gain
- Allows retransmission of packet that would have timed-out
  - Small windows/loss of most of window
  - Real source of performance gain
  - Shouldn't comparison be against NewReno/SACK

TCP Vegas

- Flaws
  - Sensitivity to delay variation
  - Paper did not do great job of explaining where performance gains came from
- Some ideas have been incorporated into more recent implementations
- Overall
  - Some very intriguing ideas
  - Controversies killed it

Changing Workloads

- New applications are changing the way TCP is used
- 1980's Internet
  - Telnet & FTP → long lived flows
  - Well behaved end hosts
  - Homogenous end host capabilities
  - Simple symmetric routing
- 2000's Internet
  - Web & more Web → large number of short xfers
  - Wild west – everyone is playing games to get bandwidth
  - Cell phones and toasters on the Internet
  - Policy routing
- How to accommodate new applications?

Binomial Congestion Control

- In AIMD
  - Increase: $W_{n+1} = W_n + \alpha$
  - Decrease: $W_{n+1} = (1- \beta) W_n$
- In Binomial
  - Increase: $W_{n+1} = W_n + \alpha/W_n^k$
  - Decrease: $W_{n+1} = W_n - \beta W_n^l$
  - k=0 & l=1 → AIMD
  - l < 1 results in less than multiplicative decrease
    - Good for multimedia applications

Binomial Congestion Control

- Rate ~ 1/ (loss rate)$^{1/(k+l+1)}$
- If k+l=1 → rate ~ $1/p^{0.5}$
  - TCP friendly if $l \leq 1$
- AIMD (k=0, l=1) is the most aggressive of this class
  - Good for applications that want to probe quickly and can use any available bandwidth

TCP Friendly Rate Control (TFRC)

- Equation 1 – real TCP response
  - 1st term corresponds to simple derivation
  - 2nd term corresponds to more complicated timeout behavior
    - Is critical in situations with > 5% loss rates → where timeouts occur frequently
- Key parameters
  - RTO
  - RTT
  - Loss rate

RTO/RTT Estimation

- Not used to actually determine retransmissions
  - Used to model TCP's extremely slow transmission rate in this mode
  - Only important when loss rate is high
  - Accuracy is not as critical
- Different TCP's have different RTO calculation
  - Clock granularity critical →500ms typical, 100ms, 200ms, 1s also common
  - RTO = 4 * RTT is close enough for reasonable operation
- EWMA RTT
  - $RTT_{n+1} = (1-\alpha)RTT_n + \alpha RTTSAMP$

Loss Estimation

- Loss event rate vs. loss rate
- Characteristics
  - Should work well in steady loss rate
  - Should weight recent samples more
  - Should increase only with a new loss
  - Should decrease only with long period without loss
- Possible choices
  - Dynamic window – loss rate over last X packets
  - EWMA of interval between losses
  - Weighted average of last n intervals
    - Last n/2 have equal weight

Loss Estimation

- Dynamic windows has many flaws
- Difficult to chose weight for EWMA
- Solution WMA
  - Choose simple linear decrease in weight for last n/2 samples in weighted average
  - What about the last interval?
    - Include it when it actually increases WMA value
    - What if there is a long period of no losses?
    - Special case (history discounting) when current interval > 2 * avg

Slow Start

- Used in TCP to get rough estimate of network and establish ack clock
  - Don't need it for ack clock
  - TCP ensures that overshoot is not > 2x
  - Rate based protocols have no such limitation – why?
- TFRC slow start
  - New rate set to min(2 * sent, 2 * recvd)
  - Ends with first loss report → rate set to ½ current rate

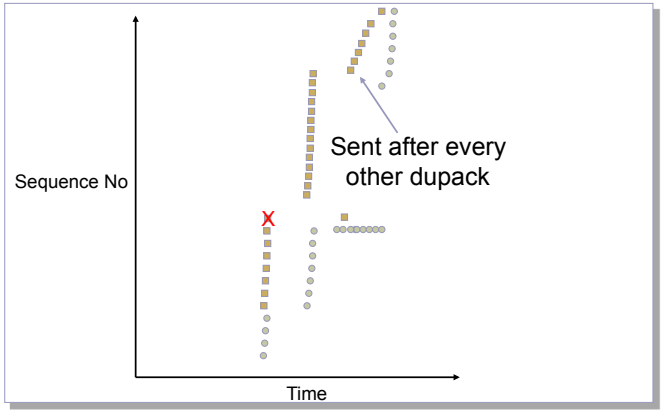
Congestion Avoidance

- Loss interval increases in order to increase rate
  - Primarily due to the transmission of new packets in current interval
  - History discounting increases interval by removing old intervals
  - .14 packets per RTT without history discounting
  - .22 packets per RTT with discounting
- Much slower increase than TCP
- Decrease is also slower
  - 4 – 8 RTTs to halve speed

NewReno Changes

- Send a new packet out for each pair of dupacks
  - Adapt more gradually to new window
- Will not halve congestion window again until recovery is completed
  - Identifies congestion events vs. congestion signals
- Initial estimation for ssthresh

Rate Halving Recovery



Delayed Ack Impact

- TCP congestion control triggered by acks
  - If receive half as many acks → window grows half as fast
- Slow start with window = 1
  - Will trigger delayed ack timer
  - First exchange will take at least 200ms
  - Start with > 1 initial window
    - Bug in BSD, now a “feature”/standard