15-744: Computer Networking

L-4 TCP

TCP Congestion Control

- Congestion Control
- RED

- Assigned Reading
  - [FJ93] Random Early Detection Gateways for Congestion Avoidance
  - [TFRC] Equation-Based Congestion Control for Unicast Applications

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
- Chiu&Jain analysis of linear congestion control

Overview

- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

5

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

6

TCP Congestion Control - Solutions

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

7

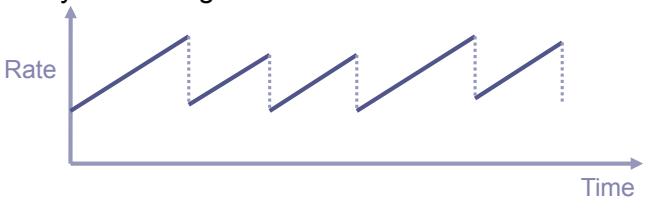
TCP Congestion Control

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

8

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



9

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)

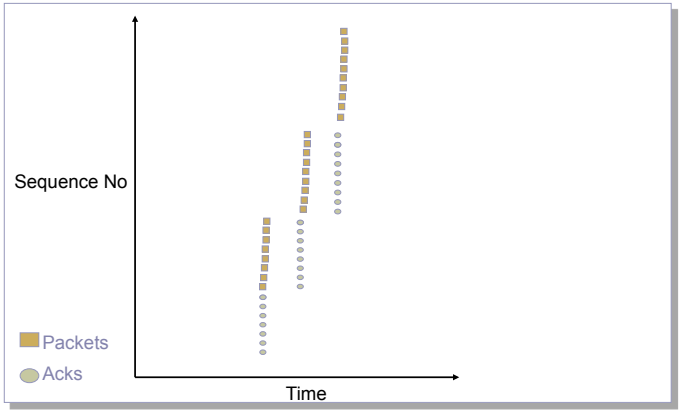
10

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

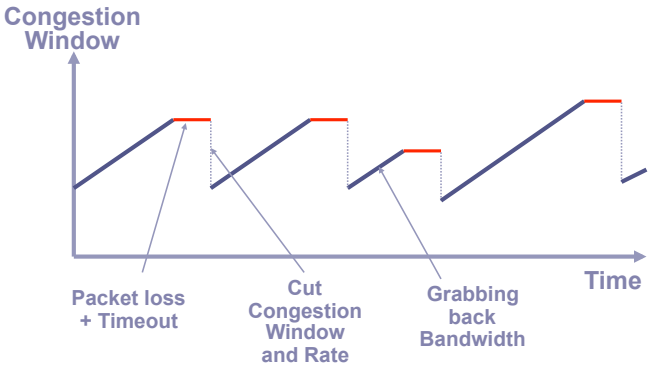
11

Congestion Avoidance Sequence Plot



12

Congestion Avoidance Behavior



13

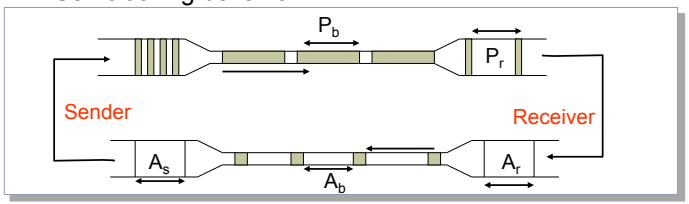
Packet Conservation

- 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)

14

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



15

Aside: Packet Pair

- What would happen if a source transmitted a pair of packets back-to-back?
- 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
- Basis for many measurement techniques

16

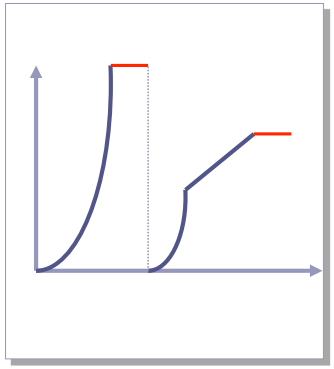
Reaching Steady State

- 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)

17

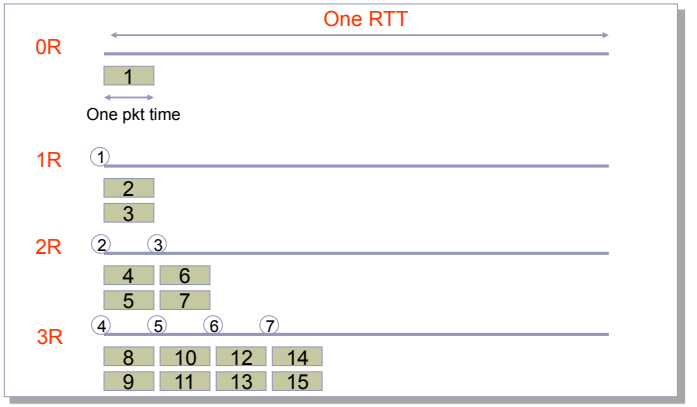
Slow Start Packet Pacing

- 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



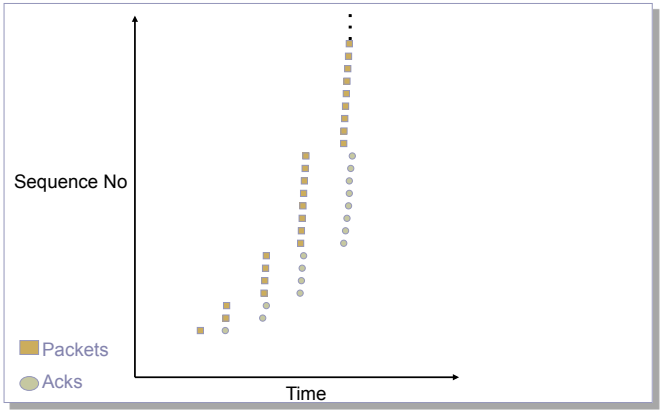
18

Slow Start Example



19

Slow Start Sequence Plot



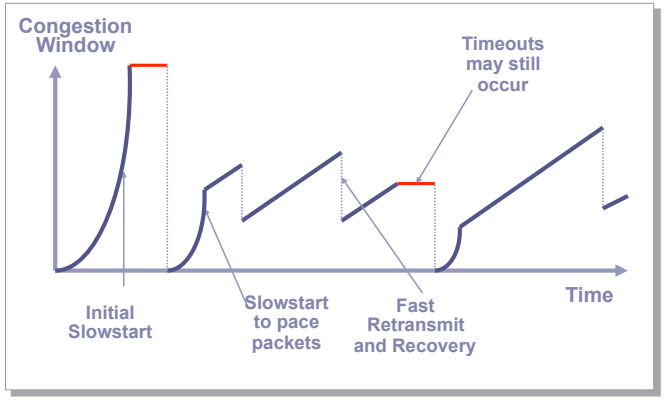
20

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

21

TCP Saw Tooth Behavior



22

Questions

- Current loss rates – 10% in paper
- Uniform reaction to congestion – can different nodes do different things?
  - TCP friendliness, GAIMD, etc.
- Can we use queuing delay as an indicator?
  - TCP Vegas
- What about non-linear controls?
  - Binomial congestion control

23

Overview

- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

24

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?

TCP Friendliness

- What does it mean to be TCP friendly?
  - TCP is not going away
  - Any new congestion control must compete with TCP flows
    - Should not clobber TCP flows and grab bulk of link
    - Should also be able to hold its own, i.e. grab its fair share, or it will never become popular
- How is this quantified/shown?
  - Has evolved into evaluating loss/throughput behavior
  - If it shows 1/sqrt(p) behavior it is ok
  - But is this really true?

TCP Friendly Rate Control (TFRC)

- Equation 1 – real TCP response

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)}$$

  - 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

- RTO not used to perform 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

29

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

30

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

31

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

32

Overview

- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

33

TCP Performance

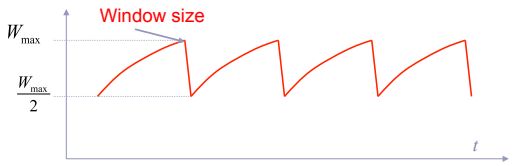
- Can TCP saturate a link?
- Congestion control
  - Increase utilization until… link becomes congested
  - React by decreasing window by 50%
  - Window is proportional to rate * RTT
- Doesn't this mean that the network oscillates between 50 and 100% utilization?
  - Average utilization = 75%??
  - No…this is *not* right!

34

TCP Congestion Control

Rule for adjusting W
- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2

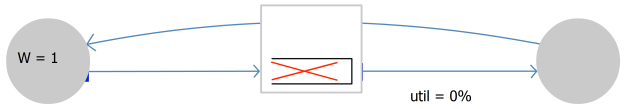
Only W packets may be outstanding



35

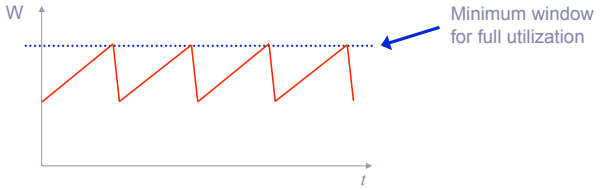
Single TCP Flow

Router without buffers



36

Summary Unbuffered Link



- The router can't fully utilize the link
  - If the window is too small, link is not full
  - If the link is full, next window increase causes drop
  - With no buffer it still achieves 75% utilization

37

TCP Performance

- In the real world, router queues play important role
  - Window is proportional to rate * RTT
    - But, RTT changes as well the window
  - Window to fill links = propagation RTT * bottleneck bandwidth
    - If window is larger, packets sit in queue on bottleneck link

38

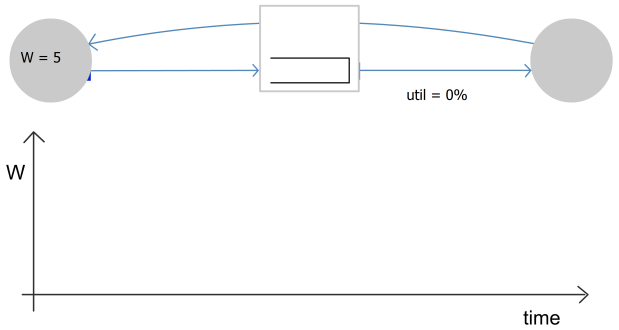
TCP Performance

- If we have a large router queue → can get 100% utilization
  - But, router queues can cause large delays
- How big does the queue need to be?
  - Windows vary from W → W/2
    - Must make sure that link is always full
    - W/2 > RTT * BW
    - W = RTT * BW + Qsize
    - Therefore, Qsize > RTT * BW
  - Ensures 100% utilization
  - Delay?
    - Varies between RTT and 2 * RTT

39

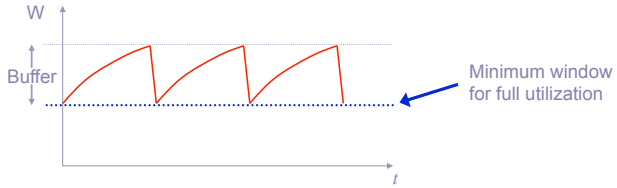
Single TCP Flow

Router with large enough buffers for full link utilization



40

Summary Buffered Link



- With sufficient buffering we achieve full link utilization
  - The window is always above the critical threshold
  - Buffer absorbs changes in window size
    - Buffer Size = Height of TCP Sawtooth
    - Minimum buffer size needed is 2T*C
  - This is the origin of the rule-of-thumb

41

Overview

- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

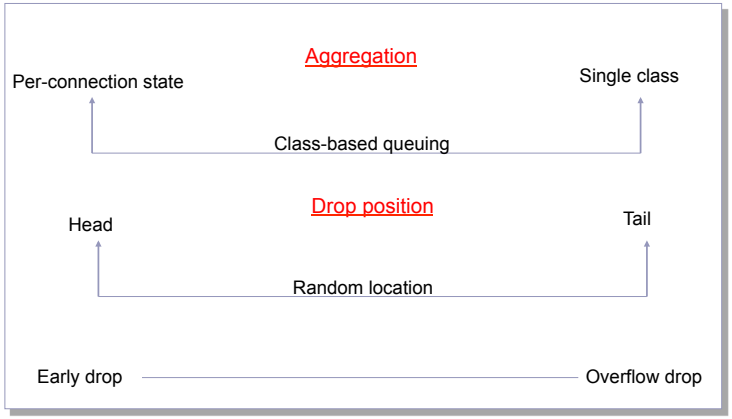
42

Queuing Disciplines

- Each router must implement some queuing discipline
- Queuing allocates both bandwidth and buffer space:
  - Bandwidth: which packet to serve (transmit) next
  - Buffer space: which packet to drop next (when required)
- Queuing also affects latency

43

Packet Drop Dimensions



44

Typical Internet Queuing

- FIFO + drop-tail
  - Simplest choice
  - Used widely in the Internet
- FIFO (first-in-first-out)
  - Implies single class of traffic
- Drop-tail
  - Arriving packets get dropped when queue is full regardless of flow or importance
- Important distinction:
  - FIFO: scheduling discipline
  - Drop-tail: drop policy

45

FIFO + Drop-tail Problems

- Leaves responsibility of congestion control to edges (e.g., TCP)
- Does not separate between different flows
- No policing: send more packets → get more service
- Synchronization: end hosts react to same events

46

Active Queue Management

- Design active router queue management to aid congestion control
- Why?
  - Routers can distinguish between propagation and persistent queuing delays
  - Routers can decide on transient congestion, based on workload

47

Active Queue Designs

- Modify both router and hosts
  - DECbit – congestion bit in packet header
- Modify router, hosts use TCP
  - Fair queuing
    - Per-connection buffer allocation
  - RED (Random Early Detection)
    - Drop packet or set bit in packet header as soon as congestion is starting

48

Overview

- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

49

Internet Problems

- Full queues
  - Routers are forced to have have large queues to maintain high utilizations
  - TCP detects congestion from loss
    - Forces network to have long standing queues in steady-state
- Lock-out problem
  - Drop-tail routers treat bursty traffic poorly
  - Traffic gets synchronized easily → allows a few flows to monopolize the queue space

50

Design Objectives

- Keep throughput high and delay low
- Accommodate bursts
- Queue size should reflect ability to accept bursts rather than steady-state queuing
- Improve TCP performance with minimal hardware changes

51

Lock-out Problem

- Random drop
  - Packet arriving when queue is full causes some random packet to be dropped
- Drop front
  - On full queue, drop packet at head of queue
- Random drop and drop front solve the lock-out problem but not the full-queues problem

52

Full Queues Problem

- Drop packets before queue becomes full (early drop)
- Intuition: notify senders of incipient congestion
  - Example: early random drop (ERD):
    - If qlen > drop level, drop each new packet with fixed probability *p*
    - Does not control misbehaving users

53

Random Early Detection (RED)

- Detect incipient congestion, allow bursts
- Keep power (throughput/delay) high
  - Keep average queue size low
  - Assume hosts respond to lost packets
- Avoid window synchronization
  - Randomly mark packets
- Avoid bias against bursty traffic
- Some protection against ill-behaved users

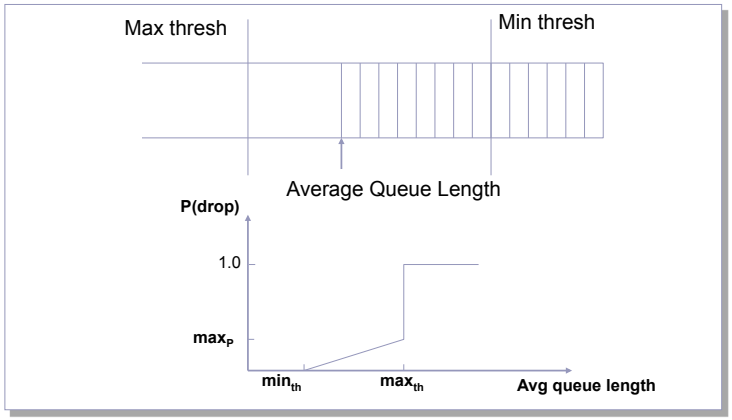
54

RED Algorithm

- Maintain running average of queue length
- If avgq < min_{th} do nothing
  - Low queuing, send packets through
- If avgq > max_{th}, drop packet
  - Protection from misbehaving sources
- Else mark packet in a manner proportional to queue length
  - Notify sources of incipient congestion

55

RED Operation



56

RED Algorithm

- Maintain running average of queue length
  - Byte mode vs. packet mode – why?
- For each packet arrival
  - Calculate average queue size (avg)
  - If $min_{th} \leq avgq < max_{th}$
    - Calculate probability P_a
    - With probability P_a
      - Mark the arriving packet
    - Else if $max_{th} \leq avg$
      - Mark the arriving packet

57

Queue Estimation

- Standard EWMA: $avgq = (1-w_q)\ avgq + w_q qlen$
  - Special fix for idle periods – why?
- Upper bound on w_q depends on min_{th}
  - Want to ignore transient congestion
  - Can calculate the queue average if a burst arrives
    - Set w_q such that certain burst size does not exceed min_{th}
- Lower bound on w_q to detect congestion relatively quickly
- Typical $w_q = 0.002$

58

Thresholds

- min_{th} determined by the utilization requirement
  - Tradeoff between queuing delay and utilization
- Relationship between max_{th} and min_{th}
  - Want to ensure that feedback has enough time to make difference in load
  - Depends on average queue increase in one RTT
  - Paper suggest ratio of 2
    - Current rule of thumb is factor of 3

59

Packet Marking

- max_p is reflective of typical loss rates
- Paper uses 0.02
  - 0.1 is more realistic value
- If network needs marking of 20-30% then need to buy a better link!
- Gentle variant of RED (recommended)
  - Vary drop rate from max_p to 1 as the avgq varies from max_{th} to 2* max_{th}
  - More robust to setting of max_{th} and max_p

60

Talks

- Radia Perlman – TRILL: Soul of a New Protocol
  - CIC 1201 – Noon Monday 9/27
- Alberto Toledo – Exploiting WLAN Deployment Density: Fair WLAN Backhaul Aggregation
  - Gates 8102 – 1:30 Monday 9/27
- Nina Taft – ANTIDOTE: Understanding and Defending against the Poisoning of Anomaly Detectors
  - Gates 8102 – Noon Wednesday 9/29
- Oct 14th – noon Google talk on M-lab
- Nov 4th – networking for the 3rd world

Next Week

- Attend one of the talks
- Monday lecture: fair queuing
- Wed no lecture
- Fri

EXTRA SLIDES

The rest of the slides are FYI

Extending RED for Flow Isolation

- Problem: what to do with non-cooperative flows?
- Fair queuing achieves isolation using per-flow state – expensive at backbone routers
  - How can we isolate unresponsive flows without per-flow state?
- RED penalty box
  - Monitor history for packet drops, identify flows that use disproportionate bandwidth
  - Isolate and punish those flows

Stochastic Fair Blue

- Same objective as RED Penalty Box
  - Identify and penalize misbehaving flows
- Create L hashes with N bins each
  - Each bin keeps track of separate marking rate (p_m)
  - Rate is updated using standard technique and a bin size
  - Flow uses minimum p_m of all L bins it belongs to
  - Non-misbehaving flows hopefully belong to at least one bin without a bad flow
    - Large numbers of bad flows may cause false positives

65

Stochastic Fair Blue

- False positives can continuously penalize same flow
- Solution: moving hash function over time
  - Bad flow no longer shares bin with same flows
  - Is history reset →does bad flow get to make trouble until detected again?
    - No, can perform hash warmup in background

66

How to Change Window

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

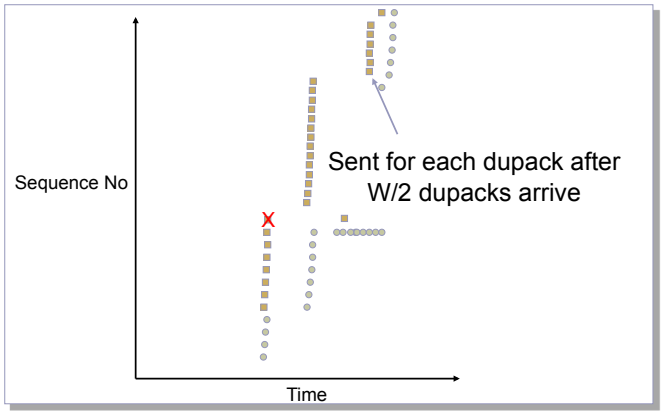
67

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

68

Fast Recovery



69

Packet Marking in RED

- Marking probability based on queue length
  - $P_b = max_p(avgq - min_{th}) / (max_{th} - min_{th})$
- Just marking based on P_b can lead to clustered marking
  - Could result in synchronization
  - Better to bias P_b by history of unmarked packets
  - $P_a = P_b/(1 - count*P_b)$

70

CHOKe

- CHOse and Keep/Kill (Infocom 2000)
  - Existing schemes to penalize unresponsive flows (FRED/penalty box) introduce additional complexity
  - Simple, stateless scheme
- During congested periods
  - Compare new packet with random pkt in queue
  - If from same flow, drop both
  - If not, use RED to decide fate of new packet

71

CHOKe

- Can improve behavior by selecting more than one comparison packet
  - Needed when more than one misbehaving flow
- Does not completely solve problem
  - Aggressive flows are punished but not limited to fair share
  - Not good for low degree of multiplexing → why?

72

FRED

- Fair Random Early Drop (Sigcomm, 1997)
- Maintain per flow state only for active flows (ones having packets in the buffer)
- min_q and max_q → min and max number of buffers a flow is allowed occupy
- avgcq = average buffers per flow
- Strike count of number of times flow has exceeded max_q

73

FRED – Fragile Flows

- Flows that send little data and want to avoid loss
- min_q is meant to protect these
- What should min_q be?
  - When large number of flows → 2-4 packets
    - Needed for TCP behavior
  - When small number of flows → increase to avgcq

74

FRED

- Non-adaptive flows
  - Flows with high strike count are not allowed more than avgcq buffers
  - Allows adaptive flows to occasionally burst to max_q but repeated attempts incur penalty

75

TCP Vegas Slow Start

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

76

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

77

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

78

TCP Vegas

- 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
- 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

79

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

80

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

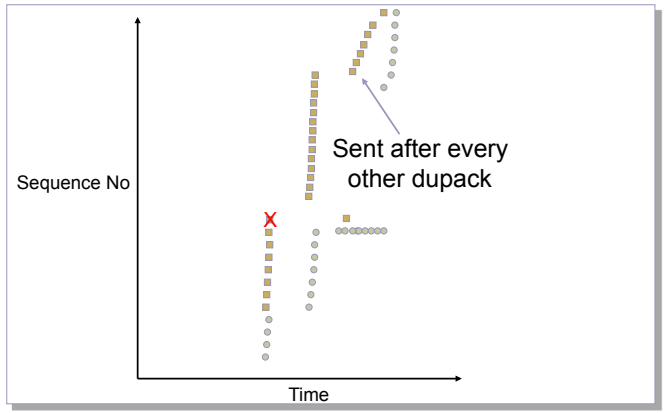
Binomial Congestion Control

- 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

Rate Halving Recovery



85