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 (2 sections)

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

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

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)
**Congestion Avoidance**

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

**Congestion Avoidance Sequence Plot**

**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 \( \Rightarrow \) 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)
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

![TCP Packet Pacing Diagram](image)

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 (called slow start)

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 * log₂(W)
  - Can overshoot window and cause packet loss

Slow Start Example

<table>
<thead>
<tr>
<th>0R</th>
<th>One RTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1R</td>
<td></td>
</tr>
<tr>
<td>2R</td>
<td></td>
</tr>
<tr>
<td>3R</td>
<td></td>
</tr>
</tbody>
</table>

![Slow Start Example Diagram](image)
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

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
Overview

• TCP congestion control

• TFRC
  • See Hugo Pinto’s slides (slides here are optional)
  • TCP and queues

• Queuing disciplines

• RED

Changing Workloads

• New applications are changing the way TCP is used

  • 1980’s Internet
    • Telnet & FTP $\rightarrow$ long lived flows
    • Well behaved end hosts
    • Homogenous end host capabilities
    • Simple symmetric routing

  • 2000’s Internet
    • Web & more Web $\rightarrow$ 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 $\rightarrow$ 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 \(\Rightarrow\) 500ms typical, 100ms, 200ms, 1s also common
  - \(\text{RTO} = 4 \times \text{RTT}\) is close enough for reasonable operation
- EWMA RTT
  - \(\text{RTT}_{n+1} = (1-\alpha)\text{RTT}_n + \alpha \text{RTTSAMP}\)

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 \(\min(2 \times \text{sent}, 2 \times \text{recv})\)
  - Ends with first loss report \(\Rightarrow\) rate set to \(\frac{1}{2}\) 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

### 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!

### TCP Congestion Control

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

**Only $W$ packets may be outstanding**

**Window size**

- $W_{max}$
- $W_{min}/2$

### Overview

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

### 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

### TCP Performance

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

### 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 \( 2\times C \)
  - This is the origin of the rule-of-thumb
Overview

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

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

Packet Drop Dimensions

- Per-connection state: Aggregation, Single class
- Class-based queuing: Early drop, Drop position, Random location, Tail
- Head

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
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

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

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

Overview

• TCP congestion control
• TFRC
• TCP and queues
• Queuing disciplines
• RED
Internet Problems

- Full queues
  - Routers are forced to 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

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

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

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
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

RED Algorithm

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

RED Operation

![Diagram showing RED operation with thresholds and queue length](image.png)

RED Algorithm

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

- Standard EWMA: \( \text{avgq} = (1-w_q) \text{avgq} + w_q \text{qlen} \)
  - Special fix for idle periods – why?
- Upper bound on \( w_q \) depends on \( \text{min}_{th} \)
  - 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 \( \text{min}_{th} \)
- Lower bound on \( w_q \) to detect congestion relatively quickly
- Typical \( w_q = 0.002 \)

Thresholds

- \( \text{min}_{th} \) determined by the utilization requirement
  - Tradeoff between queuing delay and utilization
- Relationship between \( \text{max}_{th} \) and \( \text{min}_{th} \)
  - 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

Packet Marking

- \( \text{max}_p \) 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 \( \text{max}_p \) to 1 as the avgq varies from \( \text{max}_{th} \) to \( 2 \times \text{max}_{th} \)
  - More robust to setting of \( \text{max}_{th} \) and \( \text{max}_p \)

Coming Up

- Friday lecture: Fair Queuing
  - Read WFQ paper
  - First two sections of XCP paper
- Next week: taking a look at routers
  - Will fix readings