15-441 Computer Networking

Lecture 10 – TCP & Routers

Overview

- TCP & router queuing
- TCP details
- Workloads

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 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 → 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

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

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 completely to the 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

FIFO + Drop-tail 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

Active Queue Management

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

Design Objectives

- Keep throughput high and delay low
  - High power (throughput/delay)
- 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

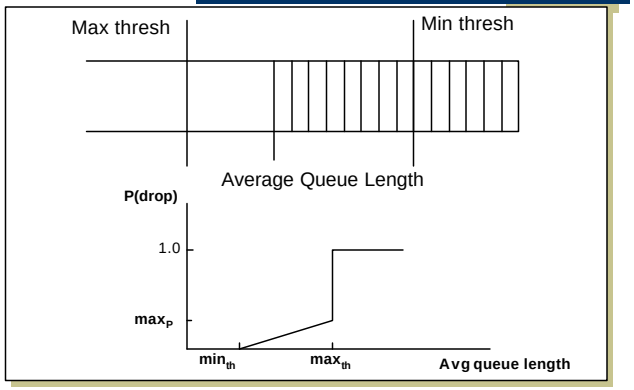
Random Early Detection (RED)

- Detect incipient congestion
- Assume hosts respond to lost packets
- Avoid window synchronization
  - Randomly mark packets
- Avoid bias against bursty traffic

RED Algorithm

- Maintain running average of queue length
- If avg < min_{th} do nothing
  - Low queuing, send packets through
- If avg > 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



Overview

- TCP & router queuing
- TCP details
- Workloads

Observed TCP Problems

- Too many small packets
  - Delayed acks
  - Silly window syndrome
  - Nagel's algorithm

Delayed ACKS

- Problem:
  - In request/response programs, you send separate ACK and Data packets for each transaction
- Solution:
  - Don't ACK data immediately
  - Wait 200ms (must be less than 500ms – why?)
  - Must ACK every other packet
  - Must not delay duplicate ACKs

TCP ACK Generation [RFC 1122, RFC 2581]

Event	TCP Receiver action
In-order segment arrival, No gaps, Everything else already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
In-order segment arrival, No gaps, One delayed ACK pending	Immediately send single cumulative ACK
Out-of-order segment arrival Higher-than-expect seq. # Gap detected	Send duplicate ACK, indicating seq. # of next expected byte
Arrival of segment that Partially or completely fills gap	Immediate ACK if segment starts at lower end of gap

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

Silly Window Syndrome

- Problem: (Clark, 1982)
  - If receiver advertises small increases in the receive window then the sender may waste time sending lots of small packets
- Solution
  - Receiver must not advertise small window increases
  - Increase window by min(MSS,RecvBuffer/2)

Nagel’s Algorithm

- Small packet problem:
  - Don’t want to send a 41 byte packet for each keystroke
  - How long to wait for more data?
- Solution:
  - Allow only one outstanding small (not full sized) segment that has not yet been acknowledged
  - Can be disabled for interactive applications

TCP Extensions

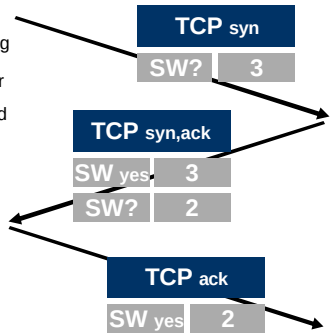
- Implemented using TCP options
  - Timestamp
  - Protection from sequence number wraparound
  - Large windows
  - Maximum segment size

Large Windows

- Delay-bandwidth product for 100ms delay
  - 1.5Mbps: 18KB
  - 10Mbps: 122KB
  - 45Mbps: 549KB
  - 100Mbps: 1.2MB
  - 622Mbps: 7.4MB
  - 1.2Gbps: 14.8MB
- 10Mbps > max 16bit window
- Scaling factor on advertised window
  - Specifies how many bits window must be shifted to the left
  - Scaling factor exchanged during connection setup

Window Scaling: Example Use of Options

- "Large window" option (RFC 1323)
  - Negotiated by the hosts during connection establishment
  - Option 3 specifies the number of bits by which to shift the value in the 16 bit window field
  - Independently set for the two transmit directions
- The scaling factor specifies bit shift of the window field in the TCP header
  - Scaling value of 2 translates into a factor of 4
- Old TCP implementations will simply ignore the option
  - Definition of an option



Maximum Segment Size (MSS)

- Exchanged at connection setup
  - Typically pick MTU of local link
- What all does this effect?
  - Efficiency
  - Congestion control
  - Retransmission
- Path MTU discovery
  - Why should MTU match MSS?

Protection From Wraparound

- Wraparound time vs. Link speed
  - 1.5Mbps: 6.4 hours
  - 10Mbps: 57 minutes
  - 45Mbps: 13 minutes
  - 100Mbps: 6 minutes
  - 622Mbps: 55 seconds
  - 1.2Gbps: 28 seconds
- Why is this a problem?
  - 55seconds < MSL!
- Use timestamp to distinguish sequence number wraparound

Overview

- TCP & router queuing
- TCP details
- Workloads

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

Short Transfers

- Fast retransmission needs at least a window of 4 packets
  - To detect reordering
- Short transfer performance is limited by slow start → RTT

Short Transfers

- Start with a larger initial window
- What is a safe value?
  - TCP already burst 3 packets into network during slow start
  - Large initial window = min (4*MSS, max (2*MSS, 4380 bytes)) [rfc2414]
    - Not a standard yet
  - Enables fast retransmission
  - Only used in initial slow start not in any subsequent slow start

Well Behaved vs. Wild West

- How to ensure hosts/applications do proper congestion control?
- Who can we trust?
  - Only routers that we control
  - Can we ask routers to keep track of each flow
    - Per flow information at routers tends to be expensive
    - Fair-queuing later in the semester

TCP Fairness Issues

- Multiple TCP flows sharing the same bottleneck link do not necessarily get the same bandwidth.
  - Factors such as roundtrip time, small differences in timeouts, and start time, … affect how bandwidth is shared
  - The bandwidth ratio typically does stabilize
- Users can grab more bandwidth by using parallel flows.
  - Each flow gets a share of the bandwidth to the user gets more bandwidth than users who use only a single flow

TCP (Summary)

- General loss recovery
  - Stop and wait
  - Selective repeat
- TCP sliding window flow control
- TCP state machine
- TCP loss recovery
  - Timeout-based
    - RTT estimation
  - Fast retransmit
  - Selective acknowledgements

TCP (Summary)

- Congestion collapse
  - Definition & causes
- Congestion control
  - Why AIMD?
  - Slow start & congestion avoidance modes
  - ACK clocking
  - Packet conservation
- TCP performance modeling
- TCP interaction with routers/queuing