Outline

- Congestion control fundamentals
  - Challenges
  - Basic mechanisms
- TCP congestion control
- TCP slow start

Congestion

- Many sources "share" resources inside network
- Problem: demand can exceed capacity of the network
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delays (queuing in router buffers)
- Challenge:
  - How do we coordinate all nodes in the Internet?

Causes & Costs of Congestion

- Four senders – multihop paths
- Timeout/retransmit

Q: What happens as rate increases?
Causes & Costs of Congestion

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

Congestion Collapse

- Definition: Increase in network load results in decrease of useful work done
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - 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

Plan for Today

- So far we considered two networks
  - Network 1: 1 router, 3 links
  - Network 2: 4 routers, 8 links
- Next step: how do we deal with congestion in the Internet
  - Millions of routers
  - Even more links
  - 100s of millions of senders

Outline

- Congestion control fundamentals
  - Challenges
  - Basic mechanisms
- TCP congestion control
- TCP slow start
Congestion Control Goals

- A mechanism that:
- Uses network resources efficiently:
  \[ \text{High } X = \sum_i x_i(t) \]
- Prevents collapse
  - Congestion collapse is not just a theory
  - Has been frequently observed in many networks
- Preserves fair network resource allocation
  
  For example: \((\sum x_i)^2/n(\sum x_i)^2\)

Two Approaches Towards Congestion Control

End-to-end congestion control:

- No explicit feedback from network
- End-systems infer congestion status from observed loss, delay, …
- Approach taken by TCP
- Problem: making it work
  - Avoid significant packet loss
  - Maintain high utilization

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 (ATM)
- Problem: makes routers more complicated
  - Per-flow state → poor scalability
  - Can sometimes be avoided

Congestion Control with Binary Feedback (TCP)

- Very simple mechanisms in network
  - FIFO scheduling with shared buffer pool
  - Feedback through packet drops (or binary feedback)
- TCP interprets packet drops as signs of congestion and sender slows down
  - This is an assumption: packet drops are not a sign of congestion in all networks, e.g., wireless networks
  - Sender periodically probes the network to check whether more bandwidth has become available
  - Key questions: how much to reduce (after a drop) and increase (when probing) rate

Linear Control

- Many different possibilities for reaction to congestion and probing for bandwidth
  - Examine simple linear controls
    - Window\((t+1) = a + b\) Window\((t)\)
    - Different \(a/b\) for increase and \(a/b\) for decrease
  - Supports various reaction to signals
    - Increase/decrease additively
    - Increased/decrease multiplicatively
    - Which of the four combinations is optimal?
  - Example of closed loop control: system must converge!
  - In addition to efficiency, fairness, … goals
Phase Plots

- Simple way to visualize behavior of competing connections over time
- Sequence of steps with 2 synchronized senders

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

Multiplicative Increase/Decrease

- Both $X_1$ and $X_2$ increase by the same factor over time
- Extension along line through origin
- Constant fairness
Achieving Fairness AND Efficiency

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

What is the Right Choice?

Outline

- Congestion control fundamentals

- TCP congestion control
  - Implementing AIMD
  - Packet pacing
  - Fast recovery
  - TCP slow start

TCP Congestion Control: Implicit Feedback and 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: by one packet per RTT
Implementation Issue: How to Implement AIMD efficiently

- Operating systems have coarse grain timers – how do control the transmit rate?
  - 100 Mbs → 1500 Byte packet every ~120 µsec
- Solution: uses a congestion window to implement AIMD
  - This is the same strategy that is used for flow control
  - Rate = window / RTT, with RTT more or less constant
- If loss occurs, cut congestion window W in half
  - 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 → increased cwnd by 1 MSS
  - Corresponds to an increase of 1 MSS every roundtrip time

Implementation Issue: Putting the Pieces Together

- Both congestion and flow control want to control when packets can be transmitted – who is really in charge?
- Solution: using a single window to control transmission
  - Sender’s maximum window = Min (advertised window, cwnd)
  - In English: can send packets if it does not flood receiver AND it does not congest the network
- The two windows are updated independently
  - Both windows are decreased when a packet is send
  - Advertised window: increased when the receiver sends window update, meaning it freed up a buffer
  - Cwnd: increased when the receiver ACKs the reception of data, meaning data left the network
  - Either event can trigger a send

Implementation Issue: How to Send Packets Smoothly

- Networks do not like very bursty traffic
  - Leads to queue overflow and increases packet loss
- Solution: congestion window helps to “pace” the transmission of data packets – “packet pacing”
  - In steady state, a packet is sent when an ack is received
  - Self-clocking behavior: flow remains smooth, once it is smooth
Congestion Avoidance Sequence Plot
Pacing and “AI”

Remember Fast Retransmit?

Outline

- TCP connection setup/data transfer
- TCP congestion avoidance
- TCP fast recovery and slow start
  - Almost there!

What Happens when we are Not in Steady State

- “Self-clocking behavior: flow remains smooth, once it is smooth”
  - How do you become smooth if you are not?
  - Is a issue where there is no data in transit
    - No data in transit → no ACKs → no self clocking
    - At the start of a connection, after an idle time, after a timeout
  - Fast retransmit can avoid timeout but still disrupts flow
    - Solution: fast recovery
  - If there is an idle time, for whatever reason, we need to (re)start packet pacing
    - Solution: slow start
Fast Recovery

- With fast retransmit, TCP can often avoid timeout, but loss signals congestion → cut window in half
- Challenge: how do we maintain ack clocking?
- Observation: each duplicate ack notifies sender that a single packet has cleared the network
- When < new cwnd packets are outstanding
  - Allow new packets to be sent for each new duplicate acknowledgement
- Behavior
  - Sender is idle for some time – waiting for ½ cwnd worth of dupacks
  - Transmits at original rate after wait with ack clocking

Reaching Steady State

- Doing AIMD is fine in steady state but how do we get started …
- 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)
  - Need quick initial phase to help TCP get up to speed
- Also, after a timeout, the “pipe has drained”
  - cwnd = 0.5 * cwnd
  - How do we restart ACK clocking?

Slow Start Packet Pacing

- How do we get this clocking behavior to start?
  - Initialize cwnd = 1
  - Upon receipt of every ack, cwnd = cwnd + 1
  - Packet loss means you are going too fast
  - Hopefully Fast Retransmit works!
- Allows TCP to quickly find a good window size
  - Exponential increase!
  - Reaches W in RTT * log₂(W)
  - Also starts packet pacing
- How is this slow?
Starting of Packet Pacing

- CC window increases by 1 packet/ACK
- Queuing separates packet pair
- This repeats each RTT
- Until pipe is full

Slow Start Sequence Plot

- Sequence No
- Time
- Packets
- Ack

TCP Sawtooth Behavior

- Timeouts may still occur

Important Lessons

- TCP state diagram → setup/teardown
- TCP timeout calculation → how is RTT estimated
- Modern TCP loss recovery
  - Why are timeouts bad?
  - How to avoid them? → e.g. fast retransmit