Go-Back-N Recovery

- Receiver sends cumulative ACKs
  - When out of order packet - send nothing (wait for source to timeout)
  - Otherwise sends cumulative ACK
- Sender implements Go-Back-N recovery
  - Set timer upon transmission of packet
  - Retransmit all unacknowledged packets on timeout
- Performance during loss recovery
  - No longer have an entire window in transit
  - Can have much more clever loss recovery
    - Receiver can send cumulative ACK even for out of order packets - Why?

Basic Go-Back-N in Action

Outline

- Transport introduction
- Error recovery and flow control
  - Connection establishment
  - Review stop-and-wait and friends
  - ACK and retransmission strategies
    - Making things work (well) in TCP
    - Timeouts
- Congestion control
- Transport optimization and futures
TCP = Go-Back-N Variant

- Sliding window with cumulative acks
  - Receiver can only return a single "ack" sequence number to the sender.
  - Acknowledges all bytes with a lower sequence number
  - Starting point for retransmission
  - Duplicate acks sent when out-of-order packet received
- But: sender only retransmits a single packet.
  - Reason???
    - Only one that it knows is lost
    - Network is congested → shouldn’t overload it
- Error control is based on byte sequences, not packets.
  - Retransmitted packet can be different from the original lost packet – Why?

Duplicate ACKs - Fast Retransmit

- What are duplicate acks (dupacks)?
  - Repeated acks for the same sequence
- When can duplicate acks occur?
  - Loss
  - Packet re-ordering
  - Window update – advertisement of new flow control window
- Assume re-ordering is infrequent and not of large magnitude
  - Receipt of 3 or more duplicate acks is indication of loss
  - Don’t wait for timeout to retransmit packet
  - When does this fail?
How about Multiple Losses?

Now what? - timeout

SACK

- Basic problem is that cumulative acks provide little information
- Selective acknowledgement (SACK) adds a bitmask of packets received
  - Implemented as a TCP option
  - Encoded as a set of received byte ranges (max of 4 ranges/often max of 3)
  - Sender can potentially fill multiple “gaps”
- When to retransmit?
  - Still need to deal with reordering → wait for out of order by 3pkts

Outline

- Transport introduction
- Error recovery and flow control
  - Connection establishment
  - Review stop-and-wait and friends
  - ACK and retransmission strategies
  - Making things work (well) in TCP
  - Timeouts
- Congestion control
- Transport optimization and futures
**Round-trip Time Estimation**

- Wait at least one RTT before retransmitting
- Importance of accurate RTT estimators:
  - Low RTT estimate
    - unneeded retransmissions
  - High RTT estimate
    - poor throughput
- RTT estimator must adapt to change in RTT
  - But not too fast, or too slow!
- Spurious timeouts
  - “Conservation of packets” principle – never more than a window worth of packets in flight
  - Most timeouts set using coarse clock, e.g., 500 msec

**Original TCP Round-trip Estimator**

- Round trip times exponentially averaged:
  - New RTT = $\alpha$ (old RTT) + $(1 - \alpha)$ (new sample)
  - Recommended value for $\alpha$: 0.8 - 0.9
    - 0.875 for most TCP's
- Retransmit timer set to $(b \times$ RTT), where $b = 2$
  - Every time timer expires, RTO exponentially backed-off
  - Not good at preventing spurious timeouts
    - Why?

**Jacobson’s Retransmission Timeout**

- Key observation:
  - At high loads, round trip variance is high
- Solution:
  - Base RTO on RTT and standard deviation
    - RTO = RTT + 4 * rttvar
  - new_rttvar = $\beta$ * dev + (1 - $\beta$) old_rttvar
    - Dev = linear deviation
    - Inappropriately named – actually smoothed linear deviation

**Important Lessons Basic TCP**

- Transport service
  - UDP $\rightarrow$ mostly just IP service
  - TCP $\rightarrow$ congestion controlled, reliable, byte stream
- Types of ARQ protocols
  - Sliding window for high throughput
  - Go-back-n $\rightarrow$ can keep link utilized (except w/ losses)
    - Selective repeat $\rightarrow$ efficient loss recovery
- TCP uses go-back-n variant
  - Avoid unnecessary retransmission ..
  - ... and gaps in the flow (fast retransmit/recovery, SACK)
Outline

- Transport introduction
- Error recovery and flow control
- Congestion control
  - Fundamentals
  - TCP congestion control
  - Slow start
- Transport optimization and futures

Internet Pipes?

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

Plumbers Gone Wild!
• How do we prevent water loss?
• Know the size of the pipes?

Plumbers Gone Wild 2!
• Now what?
  • Feedback from the bucket or the funnels?

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
  – Coordinate all nodes in the Internet?
• Manifestations:
  – Lost packets (buffer overflow at routers)
  – Long delays (queuing in router buffers)
  – Can result in throughput less than bottleneck link (1.5Mbps for the above topology) → a.k.a. congestion collapse
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
    • 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

Congestion Control and Avoidance

• A mechanism that:
  – 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
Approaches Towards Congestion Control

- Two broad approaches towards congestion control:
  - End-to-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 (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 (= 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

Objectives

- Simple router behavior
- Distributedness
- Efficiency: $X = \sum x_i(t)$
- Fairness: $(\sum x_i)^2/n(\sum x_i^2)$
  - What are the important properties of this function?
- Convergence: control system must be stable

Linear Control

- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
    - $\text{Window}(t + 1) = a + b \text{Window}(t)$
    - Different $a/b_i$ 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?
Phase Plots

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

User 1’s Allocation $x_1$
User 2’s Allocation $x_2$

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

Efficiency Line
Fairness Line

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

Efficiency Line
Fairness Line

Multiplicative Increase/Decrease

- Both $X_1$ and $X_2$ increase by the same factor over time
- Constant fairness
Achieving Fairness AND Efficiency

What is the Right Choice?

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

Outline

• Transport introduction
• Error recovery and flow control
• Congestion control
  – Fundamentals
  – TCP congestion control
  – Slow start
• Transport optimization and futures

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

- 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.
  - Similar to using a flow control window to avoid flooding receiver
  - 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”
  - \((\text{last sent} - \text{last acked}) < \text{congestion window}\)
- Window used is limited by both congestion and flow control
  - Sender’s maximum window = \(\text{Min (receiver flow ctl window, network congestion ctl window)}\)

### Packet Conservation

- At equilibrium, inject packet into network only when one is removed
  - Controlled by sliding window, not rate
  - 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)

### 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

### Congestion Avoidance

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

**Sequence No**

- 10
- 9
- 8

<table>
<thead>
<tr>
<th>Time</th>
<th>Sequence No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

**Packets**

**Acks**

**Congestion Avoidance Behavior**

- **Packet loss + retransmit**
- **Cut Congestion Window and Rate**
- **Grabbing back Bandwidth**

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 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 with ack clocking

Remember Fast Retransmit?

- Fast Retransmit
- Fast Retransmit Fails
- Much Faster!
Fast Recovery

Sequence No

Sent for each dupack after W/2 dupacks arrive

Outline

• Transport introduction
• Error recovery and flow control
• Congestion control
  – Fundamentals
  – TCP congestion control
  – Slow start
• Transport optimization and futures