L12: end to end layer

6.033 Spring 2007
http://web.mit.edu/6.033
Slides from many folks

Network layer provides best effort service

- Packets may be:
  - Lost
  - Delayed (jitter)
  - Duplicated
  - Reordered
  - ...
- Problem: Inconvenient service for applications
- Solution: Design protocols for E2E modules
  - Many protocols/modules possible, depending on requirements

This lecture: some E2E properties

- At most once
- At least once
  - Exactly once?
- Sliding window
- Case study: TCP
- Tomorrow: Network File System (NFS)

At Least Once

- Sender persistently sends until it receives an ack
- Challenges:
  - Duplicate ACKs
  - What value for timer

Duplicate ACK problem

- Problem: Request 2 is not delivered
  - violates at least once delivery
Solution: nonce

- Label request and ack with unique identifier that is never re-used

Engineering a nonce

- Use sequence numbers
- Challenges:
  - Wrap around?
  - Failures?

Time-out value

- Fixed is bad. RTT changes depending on congestion
  - Pick a value that's too big, wait too long to retransmit a packet
  - Pick a value too small, generates a duplicate (retransmitted packet).
  - Adapt the estimate of RTT adaptive timeout

Adaptive Timeout:

- Exponential weighted moving averages
  - Samples $S_1, S_2, S_3, \ldots$
  - Algorithm
    - $\text{EstimatedRTT} = T_0$
    - $\text{EstimatedRTT} = \alpha S + (1-\alpha) \text{EstimatedRTT}$
    - where $0 \leq \alpha \leq 1$
  - What values should one pick for $\alpha$ and $T_0$?
    - Adaptive timeout is also hard

At Most Once Challenges

- Server shouldn't process req 1
- Server should send result preferably
**Idea: remember sequence number**

- Server remembers also last few responses

**Problem: failures**

- Performed request 1 twice!
- How to maintain the last nonce per sender (tombstone)?
- Write to non-volatile storage?
- Move the problem? (e.g., different port number)
- Make probability of mistake small?
- How about exactly once? (Need transactions)

**How fast should the sender sends?**

- Waiting for acks is too slow
- Throughput is one packet/RTT
  - Say packet is 500 bytes
  - RTT 100ns
  - Throughput = 40Kb/s, Awful!
- Overlap pkt transmission

**Send a window of packets**

- Assume the receiver is the bottleneck
  - Maybe because the receiver is a slow machine
- Receiver needs to tell the sender when and how much it can send
- The window advances once all previous packets are acked too slow

**Sliding Window**

- Senders advances the window whenever it receives an ack sliding window
- But what is the right value for the window?

**The Right Window Size**

- Assume server is bottleneck
  - Goal: make idle time on server zero
  - Assume: server rate is 8 bytes/s
  - Window size = 8 x RTT
  - Danger: sequence number wrap around
- What if network is bottleneck?
  - Many senders?
  - Sharing?
  - Next lecture
**“Negative” ACK**

- Minimize reliance on timer
- Add sequence numbers to packets
- Send a Nack when the receiver finds a hole in the sequence numbers

**Difficulties**
- Reordering
- Cannot eliminate acks, because we need to ack the last packet

---

**E2E layer in Internet**

![Diagram showing the 4-layer Internet model](image)

**TCP header**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>tcp.sport</td>
</tr>
<tr>
<td>Destination Port</td>
<td>tcp.dport</td>
</tr>
<tr>
<td>Sequence Number</td>
<td>tcp.seq</td>
</tr>
<tr>
<td>Acknowledgment Number</td>
<td>tcp.ack</td>
</tr>
<tr>
<td>Offset</td>
<td>tcp.off</td>
</tr>
<tr>
<td>Flags</td>
<td>tcp.flags</td>
</tr>
<tr>
<td>Window</td>
<td>tcp.mss</td>
</tr>
<tr>
<td>Urgent Pointer</td>
<td>tcp.urgp</td>
</tr>
<tr>
<td>TCP options</td>
<td>tcp.options</td>
</tr>
</tbody>
</table>

---

**UDP**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>udp.sport</td>
</tr>
<tr>
<td>Destination Port</td>
<td>udp.dport</td>
</tr>
<tr>
<td>Length</td>
<td>udp.len</td>
</tr>
<tr>
<td>Checksum</td>
<td>udp.sum</td>
</tr>
</tbody>
</table>

---

**Transmission Control Protocol (TCP)**

- Connection-oriented
- Delivers bytes at-most-once
- Bidirectional
  - ACKs are piggybacked

---

**Closing a TCP connection**

![Diagram showing the closing of a TCP connection](image)