Reliability & Congestion Control

6.033 Lecture 13
Dina Katabi & Frans Kaashoek

The Internet Stack

<table>
<thead>
<tr>
<th>Protocol Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
</tr>
<tr>
<td>Transport</td>
</tr>
<tr>
<td>TCP / UDP</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>IP</td>
</tr>
<tr>
<td>Link</td>
</tr>
</tbody>
</table>

Internet: Best Effort

No Guarantees:
- Variable Delay (jitter)
- Variable rate
- Packet loss
- Duplicates
- Reordering
- Maximum length

E2E Transport

- Reliability: “At Least Once Delivery”
  - Lock-step
  - Sliding Window
- Congestion Control
  - Flow Control
  - Additive Increase Multiplicative Decrease

“At Least Once” (Take 1): Lock-Step

- Each data packet has a sequence number set by sender
- Receiver: upon receipt of packet $k$, sends acknowledgment (ack) for $k$ (“I got $k$”)
- Sender: Upon ack $k$, sends $k+1$. If no ack within timeout, then retransmit $k$ (untilacked)

How Long to Set Timeout?

- Fixed timeouts don’t work well
  - Too big → delay too long
  - Too small → unnecessary retransmission
- Solution
  - Timeout should depend on RTT
  - Sender measures the time between transmitting a packet and receiving its ack, which gives one sample of the RTT
But RTT Could Be Highly Variable

Example from a TCP connection over a wide-area wireless link
Mean RTT = 0.25 seconds; Std deviation = 0.11 seconds!

Calculating RTT and Timeout: (as in TCP)

Exponentially Weighted Moving Average (EWMA)

- Estimate both the average $rtt_{avg}$ and the deviation $rtt_{dev}$

- Procedure calc_rtti($rtt_{sample}$)
  $$rtt_{avg} \leftarrow a \cdot rtt_{sample} + (1-a) \cdot rtt_{avg}; ~/a = 1/8 ~/d$$
  $$rtt_{dev} \leftarrow b \cdot \text{abs}(rtt_{sample} - rtt_{avg}) + (1-b) \cdot rtt_{dev}; ~/b = 1/4 ~/d$$

- Procedure calc_timeout($rtt_{avg}$, $rtt_{dev}$)
  $$\text{Timeout} \leftarrow rtt_{avg} + 4 \cdot rtt_{dev}$$

Improving Performance

- Lock-step protocol is too slow: send, wait for ack, send, wait for ack, ...
- Throughput is just one packet per RTT
- Solution: Use a window
  - Keep multiple packets in the network at once
  - overlap data with acks

At Least Once (Take 2): Fixed Window

- Receiver tells the sender a window size
- Sender sends window
- Receiver acks each packet as before
- Window advances when all pkts in previous window are ack'd
- If a packet times out $\Rightarrow$ retransmit
- Still much idle time

At Least Once (Take 3): Sliding Window

- Sender advances the window by 1 for each in-sequence ack it receives
  - Reduces idle periods
  - Pipelining idea!
- But what's the correct value for the window?
  - We'll revisit this question
  - First, we need to understand windows

Sliding Window in Action
Example: $W = 5$; We show how the window slides with ack arrivals

 Sender: W = 1-3, W = 2-4, W = 3-5, W = 4-6
 Receiver: W = 2-6
### Sliding Window in Action

Example: $W = 5$; We show how the window slides with ack arrivals.

<table>
<thead>
<tr>
<th>Snr</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcvr</td>
<td>d1</td>
<td>d2</td>
<td>d3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Slide 1

<table>
<thead>
<tr>
<th>Snr</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcvr</td>
<td>d1</td>
<td>d2</td>
<td>d3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Slide 2

<table>
<thead>
<tr>
<th>Snr</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcvr</td>
<td>d1</td>
<td>d2</td>
<td>d3</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>a4</td>
</tr>
</tbody>
</table>

### Handling Packet Loss

Sender advances the window on arrivals of in-sequence acks.
- Can’t advance on $a3$’s arrival.

<table>
<thead>
<tr>
<th>Snr</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcvr</td>
<td>d1</td>
<td>d2</td>
<td>d3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Slide 1

<table>
<thead>
<tr>
<th>Snr</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcvr</td>
<td>d1</td>
<td>d2</td>
<td>d3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Slide 2

<table>
<thead>
<tr>
<th>Snr</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcvr</td>
<td>d1</td>
<td>d2</td>
<td>d3</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>a4</td>
</tr>
</tbody>
</table>

### What is the Right Window Size?

- **Window is too small**
  - Long idle time
  - Underutilized Network
- **Window too large**
  - Congestion
Case study: TCP

- TCP: reliable pipe to send bytes
- Uses acknowledgements to adopt to:
  - link capacity
  - rate at which server processes
  - congestion in the network
  - lost packets
- Explicit setup and tear-down

E2E Transport

- Reliability: “At Least Once Delivery”
  - Lock-step
  - Sliding Window
- Congestion Control
  - Flow Control
  - Additive Increase Multiplicative Decrease

Setting Window Size: Flow Control

Setting Window Size: Congestion

Setting Window Size: Congestion

Congestion Control

- Basic Idea:
  - Increase cwnd slowly; if no drops → no congestion yet
  - If a drop occurs → decrease cwnd quickly
- Use the idea in a distributed protocol that achieves
  - Efficiency, i.e., uses the bottleneck capacity efficiently
  - Fairness, i.e., senders sharing a bottleneck get equal throughput (if they have demands)
Additive Increase Multiplicative Decrease

- Every RTT:
  - No drop: $cwnd = cwnd + 1$
  - A drop: $cwnd = cwnd/2$

Additive Increase

![Additive Increase Diagram]

AIMD Leads to Efficiency and Fairness

Consider two users who have the same RTT

- MD $\rightarrow$ move on lines through origin
- AI $\rightarrow$ move on lines parallel to fairness line

Efficiency line $cwnd1 + cwnd2 = RTT \times bottleneck_{cap}$

User 1: $cwnd_1$

User 2: $cwnd_2$

Fairness line $cwnd1 = cwnd2$

Summary of E2E Transport

- Reliability Using Sliding Window
  - Tx Rate = $W/RTT$
- Congestion Control
  - $W = \min(Receiver_{buffer}, cwnd)$
  - $cwnd$ is adapted by the congestion control protocol to ensure efficiency and fairness
  - TCP congestion control uses AIMD which provides fairness and efficiency in a distributed way