Reliability & Sliding Window

Jasleen Kaur

Fall 2016

ARQ: Recovering Lost Data

- Segment loss may occur due to:
  - non-recoverable bit errors
  - buffer overflow
  - multiple attempts at shared medium access

- Need mechanisms to recover lost frames

- Basic idea: use Automatic Repeat Request (ARQ)
  - Acknowledgements (sent by receiver)
    - Indicate successful delivery
  - Timeouts (maintained by sender)
    - Indicate when packet is likely to be lost
ARQ Example: The Stop-and-Wait Protocol

- Sender sends next packet only after it receives ACK for transmitted packet
  - At most one unacknowledged packet in flight
  - Uses sequence numbers to distinguish retransmissions from new packets
    - How many distinct sequence numbers needed?
- Expected throughput
  - No more than 1 frame/RTT

Sliding Window: Efficient ARQ Example

- Allows multiple unacknowledged packets
  - Number of unacknowledged packets upper bounded by window
Reliable, In-order, Byte-stream Delivery

- Sender assigns seq no. to every byte
- Sender maintains:
  - Last Byte Written (LBWritten)
  - Last Byte Sent (LBSent)
  - Last Byte Acked (LBAcked)
- Invariant:
  - LBAcked ≤ LBSent ≤ LBWritten
- Sender buffers only bytes between:
  - (LBAcked, LBWritten]
  - Retransmits packets that time-out waiting for ACKs

- Receiver maintains:
  - Last Byte Read (LBRead)
  - Next Byte Expected (NBExp)
  - Last Byte Received (LBRcvd)
- Receiver (cumulative) ACKs send NBExp
- Invariant:
  - LBRead ≤ NBExp ≤ LBRcvd + 1
- Receiver buffers only bytes between:
  - [LBRead, LBRcvd]

Sliding Window: Flow Control

- New invariants:
  - LBSent – LBAcked ≤ AdvWin
  - EffectiveWin = AdvWin – (LBSent – LBAcked)
- Sender buffer limits (MaxSendBuf)
  - add invariant:
    - LBWritten – LBAcked ≤ MaxSendBuf
- Receiver buffer limits (MaxRcvBuf)
  - add new invariant
    - LBRead – LBRcvd ≤ MaxRcvBuf
- Receiver advertises to sender:
  - AdvWin = MaxRcvBuf – [(NBExp – 1) – LBRead]

Is it possible that an ACK arrival will not allow further transmissions?

Is this guaranteed to prevent buffer overflow at either ends?
**Sequence Number Space Required**

- How large should the sequence number space be?
  - As large as the maximum window size?
  - MaxSeqNum > 2*Window

**Adaptive Retransmissions**

- Central question: How to set retransmission timeout?
  - Internet paths can vary significantly in their propagation length
  - End-to-end RTTs can vary significantly over time
    - RTO should be an adaptive function of current RTT conditions

- Associated question: How to sample path RTTs?
  - Given retransmitted packets? And given delayed ACKs?
    - Workable solution: stop sampling during retransmission phases
Adaptive RTO: Compute as a Function of RTT

- **Original algo:**
  - Maintain running average of RTT samples
    - Exponentially weighted average
      - EstimatedRTT = $a \times$ EstimatedRTT + $(1-a) \times$ SampledRTT
    - How does $a$ impact computation?
  - Use RTT estimate to compute timeout:
    - RTO = $2 \times$ EstimatedRTT
    - Just a conservative value

- **Problem:** does not take RTT variance into account

---

**RTT Variability In Practice**

- UNC study measured per-segment RTTs for 1 million TCP transfers
  - 70% transfers see a standard deviation of 10 ms or more
Median RTTs can be more than twice of min RTTs for 30% of transfers

Use variance in RTTs to estimate timeout
- If variance is small, why set RTO to twice the value of EstimatedRTT?

Algo:
- Calculate running averages of both RTT and its variation
  - Diff = SampledRTT - EstimatedRTT
  - EstimatedRTT = EstimatedRTT + (1-a)*Diff
  - Deviation = Deviation + (1-b)*(|Diff| - Deviation)

- RTO = u*EstimatedRTT + p*Deviation
  - Typically, u = 1, p = 4, a = b = 0.875
  - Large variance causes Deviation to dominate calculation of RTO
RTO Timers In Practice

- Most Unix TCP implementations check to see if a timeout should occur, only once every clock tick
  - Ultimately, algorithm only as good as granularity of system clocks
    - 10-100 ms for current OSes

- Most implementations take only one RTT sample per RTT

- Most implementations use a minRTO value of 200 ms – 1000 ms

- TCP extensions:
  - Instead of relying on coarse-grained timers for measuring RTTs, use per-packet timestamps to measure RTT

Timeouts may happen no earlier than 1 second after segment was transmitted!

Course Projects

- Study a network protocol or Internet-wide distributed system
  - Either study it through real-world measurements
  - Or study it through simulation / analysis
  - Or implement your version as an application-layer overlay

- Groups of 2 are fine (with appropriately scaled project)

- Email project proposal by Thursday of next week
  - Project topic
  - Brief summary of proposed methodology (and timeline)

- Milestones (presentations):
  - Project proposal
  - Mid-project progress
  - Final project presentation