TCP

Overview

- Connection-oriented:
  - handshaking (exchange of control msgs) to initialize sender, receiver state before data exchange
- Reliable data transfer:
  - guaranteed arrival, no error, in order
- Flow controlled:
  - sender does not overwhelm receiver
- Congestion controlled:
  - sender does not overwhelm the network
- No delay or bandwidth guarantee.

TCP Segment Structure

- source port #
- dest port #
- sequence number
- acknowledgement number
- receive window
- Urg data ptr
- Options (variable length)
- application data (variable length)
- URG: urgent data (generally not used)
- ACK: ACK # valid
- PSH: push data now (generally not used)
- RST, SYN, FIN: connection estab (setup, teardown commands)
- Internet checksum

Maximum Segment Size (MSS)

- MSS is the maximum TCP segment that’d fit into the link layer frame
- Local MSS
- Path MSS
  - Try and error probing
TCP Reliable Data Transfer

- TCP provides reliable data transfer service on top of IP's unreliable service
  - Pipelined transmissions
  - Cumulative ACKs
  - When the receiver receives out-of-order segments, it buffers them and re-ACKs the last in-order data
  - Retransmit a single segment at each timeout
  - The sender retransmits at timeout or when receiving duplicate ACKs
- Somewhere between Go-back-N and Selective Repeat, closer to which?

TCP Timeout

Q: principles for setting transmission timeout value?

- too short: premature timeout and unnecessary retransmissions
- too long: slow reaction to segment loss
- longer than normal RTT (round trip time)
  - but RTT varies

Timeout

Expected Round Trip Time

Derive expected RTT from past RTT measurement.

- Basic measurement: measured time from segment transmission until ACK receipt
- Stability: RTT fluctuates, we want to avoid instability (pre-mature reaction to short-term spikes)
  - average several recent measurements, not just current RTT
- Agility: in case things do change, we want to adjust quickly
  - give more recent measurements higher weight

EWMA – Exponentially Weighted Moving Average

- influence of past samples decreases exponentially fast

\[
\text{ExpectedRTT} = \frac{\text{SampleRTT}_1 + \alpha \text{SampleRTT}_2 + \alpha^2 \text{SampleRTT}_3 + \ldots}{1 + \alpha + \alpha^2 + \ldots}
\]

\[
\text{SampleRTT}_1 \text{ is RTT for the most recent data segment,}
\text{SampleRTT}_2 \text{ is RTT for the next recent data segment, etc.}
\]

\[
\Rightarrow \text{ExpectedRTT} = \alpha \text{ExpectedRTT}_{\text{last}} + (1-\alpha) \text{SampleRTT}_1
\]

- typical value in TCP: \( \alpha = 0.875 \)
**Example RTT Estimation**

![RTT Data Diagram](image)

**TCP Timeout**

Setting the timeout:
- ExpectedRTT plus “safety margin”
- Larger variation in RTT → larger safety margin
- TimeoutInterval = ExpectedRTT + 4*DevRTT
- DevRTT also estimated using EWMA of past measurements

**TCP Sender Events and Processing**

Data ready to send:
- Create segment with seq #
- Seq # is byte-stream number of first data byte in segment
- Start timer
- Timeout value: we just decided it!!

Timeout:
- Retransmit segment that caused timeout
- Restart timer

ACK recv:
- Slide sender window if acknowledges previously unacked segments
- Retransmit if 3 duplicate ACKs

**TCP byte-oriented seq. #’s and ACKs**

Seq. #’s:
- Byte stream “number” of first byte in segment’s data

ACKs:
- Seq # of next byte expected from other side
- Cumulative ACK
- Piggybacked in data segments in the reverse direction
TCP in Action: Cumulative ACK

- Host A
- Host B
- Time-out period often relatively long:
  - Long delay before resending lost packet
- When receiver receives out-of-order segments, it re-ACKs the last in-order byte
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmission: resend segment before timer expires, restart timer

TCP in Action: Duplicate ACKs and Fast Retransmission

- Host A
- Host B
- 3 duplicate ACKs resend 92
- Cumulative ACK scenario

Fast Retransmission

Outline

- Segment structure
- Reliable data transfer
- Flow control
- Connection management
TCP Flow Control

- Receive side of TCP connection has a receive buffer.
- App process may be slow at reading from buffer.

Flow Control:
- Receiver advertises spare room by including value of RcvWindow in segments
- Sender limits unACKed data to RcvWindow, therefore guarantees receive buffer doesn't overflow

TCP Connection Management

- **Establishment:**
  - TCP sender, receiver establish "connection" before exchanging data segments
  - Initialize TCP variables: starting seq. #s, MSS, buffers, flow control info (e.g. RcvWindow)
- **Teardown:**
  - Free up resources after mutually close

TCP Connection Establishment

**Three-way handshake:**

**Step 1:** client (active open) sends TCP SYN segment to server
- specifies initial seq #
- no data

**Step 2:** server (passive open) host receives SYN, replies with SYNACK segment
- server allocates buffers
- specifies server initial seq. #

**Step 3:** client receives SYNACK, replies with ACK segment, which may contain data

TCP Connection Teardown

**Closing a connection:**
- close socket: close(sockfd);

**Step 1:** A (active closing host) sends TCP FIN control segment.
- Step 2: B (passive closing host) receives FIN, replies with ACK. Closes connection, sends FIN.
- Step 3: A receives FIN, replies with ACK. Enters "timed wait" – resend ACK in case it is lost.
- Step 4: B receives ACK. Connection closed.
TCP State Transition Diagram

- CLOSED
- LISTEN
- SYN_RCVD
- SYN_SENT
- ESTABLISHED
- FIN_WAIT_1
- FIN_WAIT_2
- CLOSING
- LAST_ACK
- TIME_WAIT
- CLOSING

- Passive open
- Active open/SYN
- Send/SYN
- SYN + ACK
- SYN + ACK/ACK
- SYN/SYN + ACK
- ACK
- Close/FIN
- FIN/FINACK
- FIN/FINACK
- FIN/ACK
- ACK
- Close
- Close/FIN
- Close/FIN

- Timeout after two segment lifetimes

Disclaimer

- Parts of the lecture slides contain original work of James Kurose, Larry Peterson, and Keith Ross. The slides are intended for the sole purpose of instruction of computer networks at the University of Rochester. All copyrighted materials belong to their original owner(s).