CS 456 Hao Pan
Salahuddin, Mohammad
Spring 2018

Chapter 3.5 Connection-Oriented Transport (TCP)

3.5.1 Overview

- TCP is **point-to-point**. There is *one sender* and *one receiver*.
- It is a reliable, in-order byte stream. There are no message boundaries.
- TCP utilizes *pipelining* to control congestion by setting a window size for flow control.
- Data flows in the same connection both ways (**full duplex data**).
- TCP is **connection-oriented**. A *handshaking procedure* must be done before data is exchanged.
- By utilizing *flow control*, a the receiver will not be overwhelmed by packets sent by the sender.
- See a visual of the TCP segment structure on **slide 3-58**.

3.5.2 TCP Sequence Numbers and ACKs

- **Sequence numbers** represent the byte stream number of the first byte in a segment's data.
- The sequence number of the next byte is expected from the other side. Once it is received, an ACK is sent.
- TCP does not handle out-of-order segments in a specific way. It is up to the implementor to decide how it is handled.

3.5.3 TCP Round Trip Time and Timeout

- The TCP timeout value should be set in proportion to the RTT.
- The estimated RTT is calculated by: $EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT$.
- Ideally, the timeout interval should be set to the $EstimatedRTT$ plus a *safety margin*.
- Deviation from $EstimatedRTT$ can be calculated using $SampleRTT$ by:
  - $DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$
  - Note that β is typically set to 0.25.
- Thus, the timeout interval should be: $TimeoutInterval = EstimatedRTT + 4 * DevRTT$.

3.5.4 TCP and Reliable Data Transfer

- TCP creates rdt (reliable data transfer) service on top of an IP's unreliable service.
- It does this by *pipelining segments, using cumulative acks, and a single retransmission timer*.
- Retransmissions are triggered by *timeout events* and *duplicate acks*.
- See example transmissions on **slides 3-68 and 3-69**.
- See table showing actions in different scenarios on **slide 3-70**.

3.5.4.1 TCP Fast Retransmit

- The time-out period is usually long, meaning if a packet is lost, there may be a long delay.
- If a sender receives many duplicate ACKs back-to-back (3), it is a sign that a segment may have been lost.
- When that happens, do not wait for a timeout, resend unacked segment with the smallest seq # immediately.

3.5.5 TCP Flow Control

- Since the receiver controls the sender, the sender shouldn't overflow the receiver's buffer by transmitting packets too quickly.
- The receiver places a **rwnd** value in the TCP header of a *receiver-to-header* segment to represent the amount of free buffer space they have.
- Typically, the receiver's buffer, **RcvBuffer** is set to 4096 bytes, though different operating systems will adjust it according to their standards.
- The sender limits the amount of unacked data to the receiver's *rwnd* value. Doing so guarantees that the receiver's buffer will not overflow.

3.5.6 TCP Connection Management

3.5.6.1 Starting a Connection

- Before exchanging data, the sender and receiver does a handshake.
- They should each agree to establish a connection by agreeing on the connection parameters.
- A **3-way handshake** is commonly used, where:
  - The client sends an init packet to the server, which contains the *init sequence number*, x, and a *SYN message* (synchronize).
  - The server receives the init packet, they choose another *init sequence number*, y, and sends the init back to the client which contains y as well as an emphACK for the SYN message, SYNACK.
  - The client receives the server's init, which indicates that the server is live. They send an *ACK for the server's init*.
  - Once the server receives that ACK, they know the client is live and the connection is established.
  - See a visual example of this on **slide 3-80**.

3.5.6.2 Closing a Connection

- The client and server must each close their own respective side of the TCP connection.
- They send a TCP segment with a *FIN bit = 1*. Once that is sent, they may no longer send messages, though they can still receive them.
- The other side responds with an *ACK to the FIN*.
- This process is repeated for the other side. Then the connection is closed.
- See a visual example on **slide 3-83**.