Project II (Task 2): Transmission Control Protocol (TCP)

Assigned: April 13, 2019
Task II due: May 5 @11:59am, 2019 (70% of the grade)

1 Overview

In this task, you will implement a congestion control protocol (similar to TCP) for the simple file transfer application (done in task 1). Please remember to read the complete assignment handout more than once so that you know exactly what is being provided and what functionality you are expected to add.

This is a group project and you must find exactly one partner to work with.

2 Task II Outline

During the course of this project, you will do the following:

1. implement congestion control
   (a) slow start
   (b) congestion avoidance
   (c) fast retransmit

3 Tasks

This section details the requirements of the assignment. This high-level outline roughly mirrors the order in which you should implement functionality.

3.1 100% Reliability and Sliding Window

In task 1 we use a fixed-size window of 10 packets. However, the implementation for task 1 did not dictate that you buffer out of sequence packets at the receiver. In this task we should add the out of sequence functionality at the receiver.

The sender should not send packets that fall out of the window. The Figure above shows the sliding windows for both sides. The sender slides the window forward when it gets an ACK for a higher packet number. There is a sequence number associated with each packet and the following constraints are valid for the sender:

Sending side:

1. LastPacketAcked, LastPacket Sent

2. LastPacketSent, LastPacketAvailable

3. LastPacketAvailable - LastPacketAcked, WindowSize

4. packet between LastPAcketAcked and LastPacketAvailable must be "buffered". You can either implement this by buffering the packets or by being able to regenerate them from the datafile

When the sender sends a data packet it starts a timer for it. It then waits for a fixed amount of time to get the acknowledgment for the packet. Whenever the receiver receives a packet, it sends an ACK for NextPacketExpected.

For example, upon receiving a packet with sequence number = 8, the reply would be "ACK 9", but only if all packets with sequence numbers less than 8 have already been received. These are called cumulative ACKs. The sender has two ways to know if the packets it sent did not reach the receiver: either a time-out occurred, or the sender received "duplicate ACKs".

1. if the sender sent a packet and did not receive an ACK for it before the timer for the packet expired, it resends the packet.

2. if the sender sent a packet and received duplicate ACKs, it knows that the next expected packet (at least) was lost. To avoid confusion from re-ordering, a sender counts a packet lost only after 3 duplicate ACKs in a row.

Receiver side: The receiver needs to buffer out of sequence packets. Upon receiving an out of sequence packet, the receiver buffers that packet in case there is still a space in the buffer. The receiver then sends a duplicate ACKs to the NextPacketExpected packet.

3.2 Congestion Control

Broadly speaking, the idea of TCP congestion control is for each source to determine how much capacity is available in the network, so it knows how many packets it can safely have "in transit" at the same time. Once a given source has this many packets in transit, it uses the arrival of an ACK as a signal that one of its packets has left the network, and it is therefore safe to insert a new packet into the network without adding to the level of congestion.

By using ACKs to pace the transmission of packets, TCP is said to be "self-clocking". TCP Congestion Control mechanism consists of the algorithms of Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery. You can read more about these mechanisms in our textbook Section 3.7.

In the first part of the project, your window size was fixed to 10 packets. The task of this second part is to dynamically determine the ideal window size. When a new connection is established with a host on another network, the window is initialized to one packet. Each time an ACK is received, the window is increased by one packet. This process is called Slow Start. The sender keeps increasing the window size until the first loss is detected or

until the window size reaches the value ssthresh (Slow-Start threshold), after which it enters Congestion Avoidance mode (see below). For a new connection the ssthresh is set to a very big value, we'll use 64 packets. If a packet is lost in Slow Start, the sender sets ssthresh to max(currentwindowsize/2, 2), in case the client returns to Slow Start again.

Congestion Avoidance slowly increases the congestion window and backs off at the first sign of trouble. In this mode when new data is acknowledged by the other end, the window size increases, but the increase is slower than in Slow Start. The increase in window size should be at most one packet each round-trip time (regardless how many ACKs are received in that RTT). That is, when the sender receives an ACK it usually increases the window by a fraction equal to 1/CWND. You may notice here that you need to use a float variable for the CWND, however when you send data you are always going to take the floor of the CWND. As soon as the whole window is acknowledged, only then these fractions would sum to a whole 1.0 and as a result the CWND would then have increased by 1. This is in contrast to Slow Start where the window size is incremented for each ACK. Recall that when the sender receives 3 duplicate ACK packets, you should assume that the packet with sequence number = acknowledgment number was lost, even if a time out has not occurred yet. This process is called **Fast Retransmit**.

Fast Retransmit Similar to Slow Start, in Congestion Avoidance if there is a loss in the network (resulting from either a time out, or duplicate acks), ssthresh is set to max(windowsize/2, 2). The window size is then set to 1 and the Slow Start process starts again.

3.3 Graphing Window Size

Your program must generate a simple output file (named CWND.csv) showing how your window size varies over time. This will help you to debug and test your code, and it will also help us to grade your code. The output format is simple, it should record the time when the window changed and the new window size. You should also amend the plotting script to plot the CWND and how it evolves over time.

Final Submission
The final submission should include a working reliable data transfer protocol.

Late submissions
Late submissions will be handled according to the policy given in the course syllabus.

GOOD LUCK (and get started) !!!