Lab 2: UDP and Reliable Data Transfer 3.0

CSE 3461
Tuesday, Thursday 12:45-2:05 Section
November 7, 2014

Derek DiCillo (Dicillo.5, 200171097)
Brett Koenig (Koenig.337, 200150712)

Overview

The group created a client and server interaction modeling Reliable Data Transfer 3.0 protocol. The client connects to the server and tells the server which file it would like retrieved. The server then finds the desired file on the file system and splits it into packets with a payload of 1K bytes. The server inserts two numbers into the header of each packet.The first is the packet number, and the second is the total number of packets that should be received for the complete file. These two numbers are all that the group has added to the payload, constituting the header of each packet. In the first step the group implemented the process of the client requesting a file and the server sending that packet back without loss. The group based their implementation of the first part off of the sample code that was emailed to the class by the instructor, with credit given to John Schultz and his UDP echo server. For part B of the program, the group integrated a random number generator. This random number generator enables the user to enter a loss frequency at runtime. The server then “loses” packets with roughly the frequency that was entered by the user. In part C of the application, the group essentially fixed this lossy version of the server. RDT 3.0 was implemented which delivers ACKs from the client to the server when a packet is received. Furthermore, if there is loss detected either by a timeout or the wrong ACK being returned, the server sends the lost packet once more.

Difficulties

The most difficult part of the application for the group was part C. Furthermore, the toughest part of part C was detecting a timeout event on the server. This event was to be triggered by the amount of time it took for the sender, or server, to receive the ACK from the sender. The difficulties were due to the recvfrom() function being a blocking function. This meant that if the packet was lost, and there was no ACK being sent back to the server, it would still wait for the ACK that was not being sent and therefore nothing would happen. The group formed a solution to this problem that worked with child processes. The group spun off a child process that contained the recvfrom() function and then timed the amount of time that had passed since that process was created. If that process had been running longer than the group’s desired timeout length, the process was killed and the packet was sent once again. After implementing the child process solution, the group was challenged by variable values that were being accessed in both the child and parent processes. However, this was able to be fixed by a shared memory solution.

Instructions

To compile and run the source code of the client and server UDP implementation please download the directory containing the source code to a Linux machine. Next, using a terminal, navigate to the directory containing the source code. Then, run “make.” Then in the terminal, please execute the following command, “./server <portNumber> <P_l>” (portNumber should be replaced by an integer value and is solely to name a port that the server will connect to, and P_l should be replaced by a float value between 0 and 1 and represents the frequency of loss). Next, in another terminal window please execute the following command, “./client <sender_hostname> <sender_portnumber> <filename>” (please replace the values that are surrounded by brackets by literals). The observed output should be different messages displayed on both terminal windows, stating which packet is being sent, which packet is received by the sender, and different statements about the ACKs that have been sent, received, and which are expected.