**Computer Networks Final Project, Fall 2017**

Tyler Matthews, Navin Ramkishun

For our project, we have implemented a UDP-File Transfer Protocol using the Java Programming Language. We chose Java over C for two reasons – easier readability and more comfortability. Our UDP Protocol consists of two separate files: *Server.java* and *Client.java*, and the design is as follows:

*Server.java*:

This program starts by taking in the port number as an argument. Once the proper argument is given, the program will be in a waiting state until a file is sent from *Client.java*; therefore, this program must be running before *Client.java* is run. Assuming that a file of an arbitrary size is sent over, Server.java will:

* Receive an incoming datagram using the *DatagramSocket.receive()* function.
* Match the “sequence number” of the datagram received to the sequence number that was sent with the datagram itself, which will ensure that the packets will arrive in order.
* If a packet arrives out of order, it will be discarded.
* An ACK will be sent using the *Server.sendAck()* function to let the sender know that a packet was successfully received after it is checked to be within sequential order.
* Repeat previous steps until all packets are received.
* After all packets are received, useful statistics will be printed which will let the user know the name of the new file received, total size in MB, throughput, and the total transfer time.

*Client.java*:

Unlike Server.java, this program will need more than one argument in order to successfully run. In order to send a file, *Server.java* must already be running and in a waiting state. Then, the packet loss rate, destination IP address, destination port number, file name to transfer, and the file name to create on the receiver’s end will need to be passed in. Once the correct arguments are passed in, *Client.java* will:

* Print the message “Sending the file” onto the screen.
* Send a packet using the *DatagramSocket.send()* function with a sequence number so that the receiver can validate that a packet is being received in the correct order.
* Wait for an ACK from the receiver with a timeout value of 50 milliseconds before sending the next packet in sequence.
* Re-send a packet if a corresponding ACK wasn’t received.
* Repeat until all packets have been sent and an ACK was received for each.
* Print useful statistics to the screen; a “File [filename] has been sent successfully” notification, followed by the file size, transfer time, throughput, and number of retransmissions.