SYSC3303 Assignment 1

Brandon Schurman – 100857068

February 6, 2015

Objective:

The objective of the system is to simulate the TCP protocol, which guarantees packet delivery, using the connectionless UDP protocol.

Architecture:

The system achieves this behaviour by utilizing two servers. Namely, `Server1` and `Server2`, which each listen for incoming packets on a java DatagramSocket. Clients are able to transfer files of any format towards Server1 using the java DatagramPacket class.

Since the UDP – IPv4 protocol specifies a maximum data size of 65,507 bytes, the `Client` application sends smaller fragments of the file at a time. The use case Currently, these fragments are defined to be 80 bytes in size, however this can be changed in the `Client.java` file. Before sending a fragment of the file, the Client calculates a hashed checksum on the fragment, which it stores as metadata in an ad-hoc `Message` class. The Message contains the fragment of the file being transferred, the calculated checksum, and the name of the file. Lastly, the client converts this Message into a stream of bytes, which it encapsulates and sends in a java DatagramPacket towards Server1. The Client then waits for a response, which may be an error code or a notification of successful delivery, before continuing to send more fragments of the file. If no response is received after 30 milliseconds, the Client will resend the same DatagramPacket, and continue to wait.

UDP inherently does not guarntee that packets will arrive in-tact (ie they may be corrupted) or that packets will arive at the destination at all. Server1 acts to exagurate this property by randomly dropping some of the packets received by the Client, or by manually corrupting the data contained in some packets. However, if a packet is not dropped, it is sent on to Server2. The role of Server2 is to analyse the data for errors. It does so by calculating a checksum using the same procedure that was used by the Client. If this checksum is different from the checksum contained in the Message sent by the Client, then Server2 responds with an error message, which is sent to Server1, and then back to the Client. If however, Server2 determines that the data is not corrupted, it collects the data and stores it to a buffer, then

sends a notification of success to Server1, which is then sent back to the Client. Server2 can distinguish which Client is associated with which file buffer by mapping it to the `senderID` attribute sent in the `Message`.

This process is repeated until the Client finishes sending all fragments of the file to Server2, through Server1. When the Client has finished, Server2 writes the buffer that was used to collect the fragments to a file of the same name that the Client entered. The file written by Server2 is an exact copy of the the Client's file. The Client then disconnects, but both Server1 and Server2 continue to listen for more incoming files from other instances of the Client application.

See figure 1 for a UML illustration of the described system.

Testing and Running:

To run the system, start both servers on their defualt ports by invoking `java Server1` and `java Server2`. Then run `java Client` to start a Client session. You will be prompted to enter a file path in the Client program. This must be the absolute path to an existing file of any format. On unix-based systems (at least), this file must also have read and write permissions for the user. Be sure that the file you choose to send with the Client is not contained in the same directory as the `Server2.class` executable. Server2 writes the file within the same directory as it is contained. And since Server2 writes to the file using the same name, you would therefore overwrite the original copy of your file. All output showing the progress of Server1 Server2 and the Client during their operations are printed to the default console using `System.out` and `System.err`.

Conclusion:

We conclude that sending arbitrarily sized files over UDP is indeed feasible by simulating the TCP protocol to guarantee packet delivery. This simulation is evidentally much slower than the standard TCP protocol, but this may be in part due to the frequency packets are dropped or manually corrupted by Server1, thus requiring the Client to resend duplicate data often and sometimes repeatedly.

Figure 1: UML diagram