Eric Craaybeek and Benjamin Tozier  
Network DesignProject; Phase 4 Documentation

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

The function of this program is to create an implementation of reliable data transfer 3.0 (rdt3.0) that is built on top of the already existing UDP. It will be able to send a file from client to server and recover from any bit errors it may encounter as well as any packet loss.

This process begins by running the NetDesignServer.py module. A Socket is created for the server, of address family “AF\_INET” (IPv4) and socket type “SOCK\_DGRAM”, or a UDP socket.This socket is then bound using its own address as the hostname and 12000 as the port number. This means that any message addressed to that port will then pass through to the socket of this server.

The server then enters a state machine where it waits for packets and recovers from bit errors. Each state waits for a packet with a specific sequence number. If the sequence number is wrong, or if the file is determined to be corrupt by verifying with the checksum, then the state restarts and sends an ack for the previous state to inform the client to resend the data. If the sequence number is right and the data is determined to be not corrupted then the server extracts the data, appends it to the destination file, and goes to the next state. The next state just waits for the next sequence number. In this way it does the exact same thing as the previous state except with the new sequence number.

In the Client program, a similar setup occurs for the socket. The servers address is specified at the beginning of the program. The address will be ‘localhost’ if both modules are being run on the same machine. If each module is being run on separate machines this address will be the IP address of the server machine. The UI is then initialized using the tkinter toolkit. This UI takes user input for a filename. This filename will be the source file to be transferred. A file to be read is opened as ‘rb’ (read; bytes) to remove any encoding or decoding of strings. If the file is invalid, the UI and the application close.

The client then enters its own state machine. It begins by reading the first 1024 bytes and calls PackageHeader(). This packages the data with a sequence number and a checksum for the data. This packet is then sent to the server. Values specified in the gui will determine the rate at which the packet will be dropped or corrupted. After this it waits for an acknowledgement for the specific sequence number that it just sent notifying the client that the file was received. The ack also has a chance of being lost or corrupted specified by the GUI. The client then repeats this with a new sequence number and the next chunk of data. This loops until a packet with an EOF (b’’) is reached. The files and socket are then closed.

**Classes and Data Types  
socket**

A socket in this regard, is a class that takes in an addressing protocol (in this case IPv4) and a messaging protocol (in this case UDP.) The resulting structure then can be used as an endpoint for the server to access the port (12000) being used for the transactions.

The class has the member functions bind, recvfrom, sendto, and close used in this project. Bind associates a port with the socket, allowing the socket to access the traffic through that port. Recvfrom allows a socket to receive data of a size in bytes through its assigned port. Sendto is the inverse of the Recvfrom function, writing to the port instead of reading. Close simply closes the socket.

**App**

Tkinter is the standard and most widely used UI toolkit for python. The App class initializes two variables inside itself, instructions and entryPath.

Instructions is simply a 2-by-15 text box telling the user to enter the name of a local file. EntryPath is a tkinter item of type Entry(), setup to take in a string text value. On pressing the Enter key while inside the entryPath input, it runs the send\_file member function of App.

Send\_file carries out the protocol to send the file to the server.

Timer is a function which creates a thread that counts down from a specified value. Once this timer reaches zero it will execute a function specified in its arguments. In this case it will execute the timeout function. The timeout function executes just resends the packet and restarts the timer.

Cancel is a function that will stop the timer function. It is used to reset the timeout timer, and is contained within endtimeout. Endtimeout just resets the timer and recalculates what the timer should be set to using the previous estimated RTT and a freshly measured sample RTT.

**DataFunctions**

This contains various functions that we devised to deal with packet management and other data manipulation. A list of functions is below.

ChecksumAddition – does the math involved in creating the checksum.

MakeChecksum – data is passed in and it uses the ChecksumAddition function to generate a checksum for the inputted set of data.

CheckChecksum – a packet is passed in and this function compares the data in the packet with the checksum also contained within the packet and returns whether or not there is an error.

InsertChecksum – adds checksum to a packet

RemoveChecksum – inputs a packet and returns that packet minus the checksum

AddSequenceNum – inputs packet, returns packet with sequence number added

RemoveSequenceNum – inputs packet, returns packet with sequence number removed

CheckSequenceNum – checks a packets sequence num against a passed in sequence num

PackageHeader – packages the sequence number and data, as well as creates and adds the checksum

UnpackageHeader – extracts data from packet

IsAck – verifies that an Ack message corresponds to the correct sequence number

CorruptPacket – corrupts a packet, used to test bit error resilience

LossCheck – checks whether or not a packet should be lost by using a pseudo random number generator.

**SocketFunctions**

Udt\_send – sends a packet using unreliable data transfer. Also has checks if the packet should be dropped and if it should, it does not send.

Rdt\_rcv – receives a packet from unreliable data transfer, but assumes that elements to make data reliable are present in the received packet

**Results**

The fully implemented program was able to successfully correct both packet loss and bit errors on the outgoing data or the incoming acknowledgements. The chance of these happening was specified in the GUI and impacted the transfer time of the file. The graph below depict the chance that each type of error vs the transfer time. I is apparent that they all follow the same trend. Had more samples been taken the chart likely would have normalized even more, and the lines would likely be even less distinguishable.