

Implementing Reliability in UDP Application Layer

Computer Networking Project

By Sanjay Dookhoo, Sachin Rajkumar and Shanea Lewis

Group Name: Client/Server Hackers

Table of Contents

[Introduction 3](#_Toc468170423)

[Reliability 4](#_Toc468170424)

[Implementing UDP Reliability 7](#_Toc468170425)

[Testing UDP Implementation 8](#_Toc468170426)

[Single Computer 8](#_Toc468170427)

[Single Computer (Simulated Network) 9](#_Toc468170428)

[Two Remote Computers 10](#_Toc468170429)

[Conclusion 11](#_Toc468170430)

[Bibliography 12](#_Toc468170431)

# Introduction

The User Datagram Protocol (UDP) is a communication protocol that creates connections which allow a measure of loss between applications. While UDP provides a minimal message-passing transport to applications, it lacks reliability, as there is no guarantee that messages sent will be received.

Reliability, in network systems, refers to the ability of a network to carry out a desired operation, most often communication. This project seeks to introduce and implement reliability, into a UDP client/server program at the application layer.

# Reliability

UDP can be thought of as a lower layer protocol which provides communication services used by the Application layer protocols.

To implement reliability within a UDP program emphasis is placed on error detection and handling. This is added at the application layer, using specific error recovery protocols.

The UDP segment format is as follows:

|  |  |
| --- | --- |
| Source port # | Dest port # |
| Length | Checksum |
| Application data | |

The checksum detects errors, such as flipped bits, in the transmitted segment. As UDP is a network communication, it is expected that it facilitates communication between a sender and a receiver. The checksum, at the sender adds the segment contents, treating it as a sequence of integers, and then places this value into the UDP checksum field. The receiver computes the checksum of the segment received and compares the calculated checksum with the value in the checksum field. If these values are not equal, then an error has been detected.

There are several ways to build reliable data transfer protocols:

1. Use of a perfectly reliable underlying channel, in which the protocol is deemed trivial. As Figure 1 shows there is only one state with a transition back to itself in both sending and receiving side. At Figure 1 (a) the sending side receives and accepts data from the upper layer through the rdt\_send(data) event. The sending side then put the data into a packet and sends the packet into the channel. The receiving side receives a packet from the underlying channel and then removes the data from the packet. This data is then sent up to the upper layer. As seen in Figure 1 there is no feedback as nothing can go wrong.

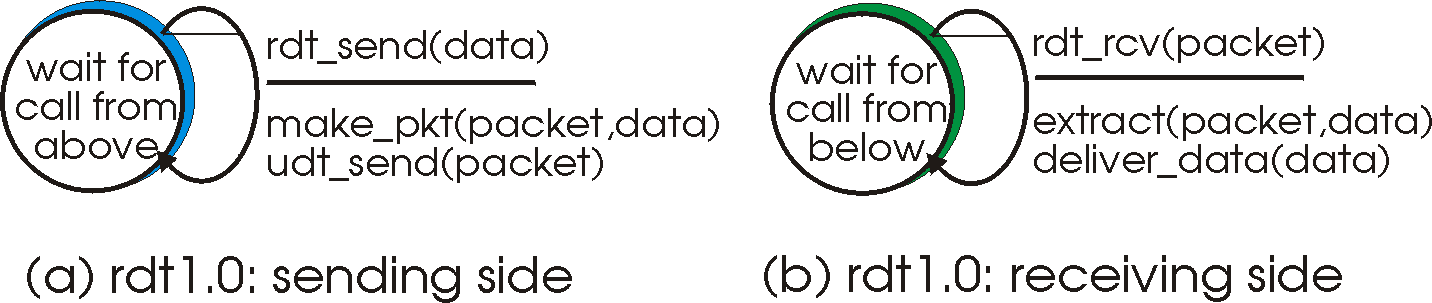


Figure 1: Reliable Data transfer over a perfectly reliable channel rdt1.0

1. Use of a channel with bit errors, a more realistic model. These bit errors occur as the packet is transmitted, propagated or buffered. This method utilizes Automatic Repeat reQuest (ARQ) protocols. ARQ requires error detection, receiver feedback and retransmission protocols to handle the presence of bit errors. Figure 2 illustrates this model.

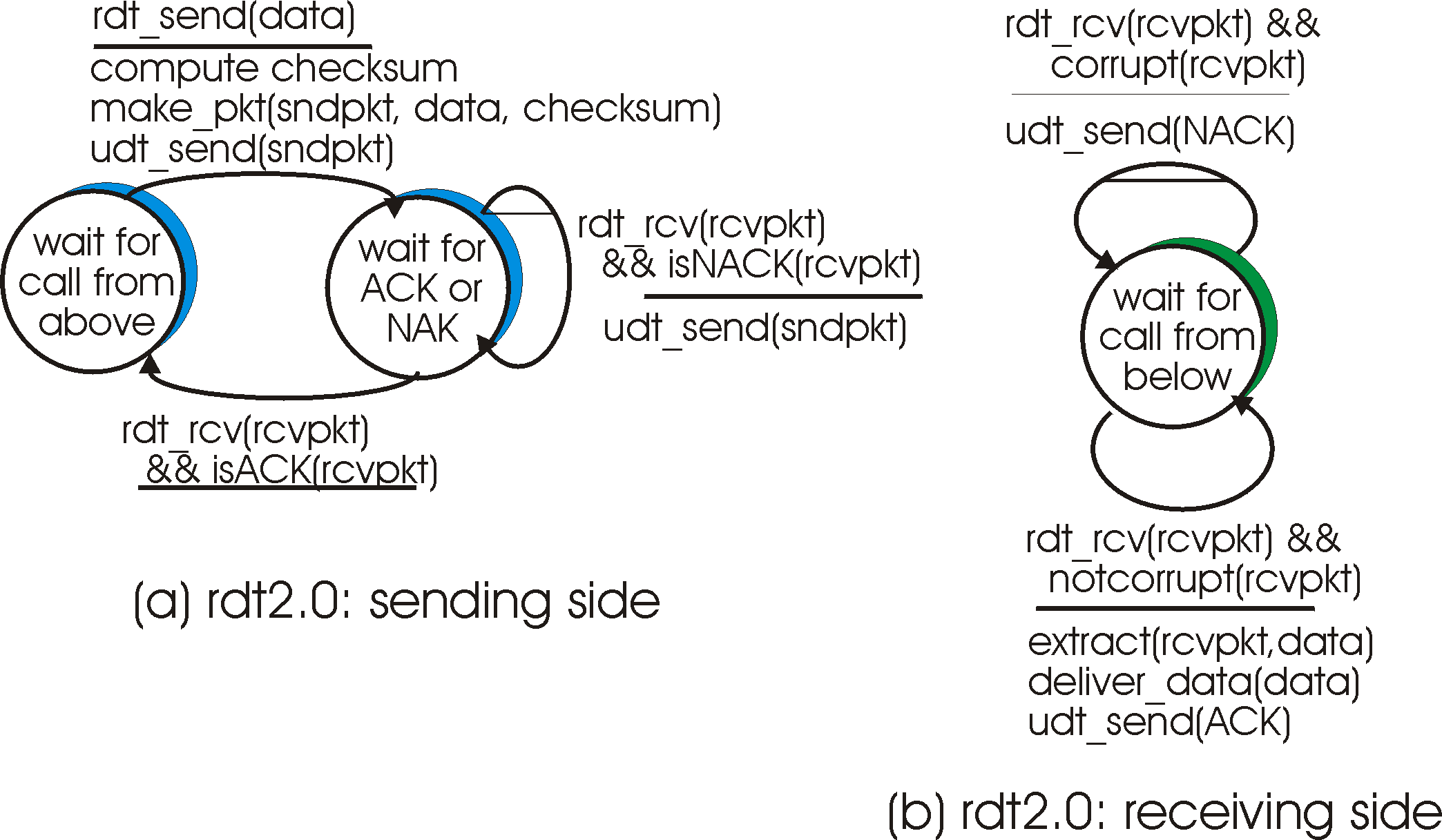


Figure 2: RELIABLE DATA TRANSFER OvER A CHANNEL WITH BIT ERRORS rdt2.0

This project uses the implementation of rdt2.2, negative acknowledgements (NAK)-free reliable data transfer over a channel with bit errors, to implement reliability in the designed client/server program. Instead of using both positive acknowledgments (ACK) and negative acknowledgments, only ACKs are used when an out of order packet is received. The receiver sends an ACK for the packet it received. When a corrupted packet is received an ACK is sent for the last correctly received packet, causing the sender to receive duplicate ACKs for the same packet. This indicates that there was an error with the receiver in getting the last packet.

The sequence number of the packet being acknowledged must be included, by the receiver. This number is checked by the sender, in the received ACK message. Figure 3.1 and 3.2 illustrates this model.

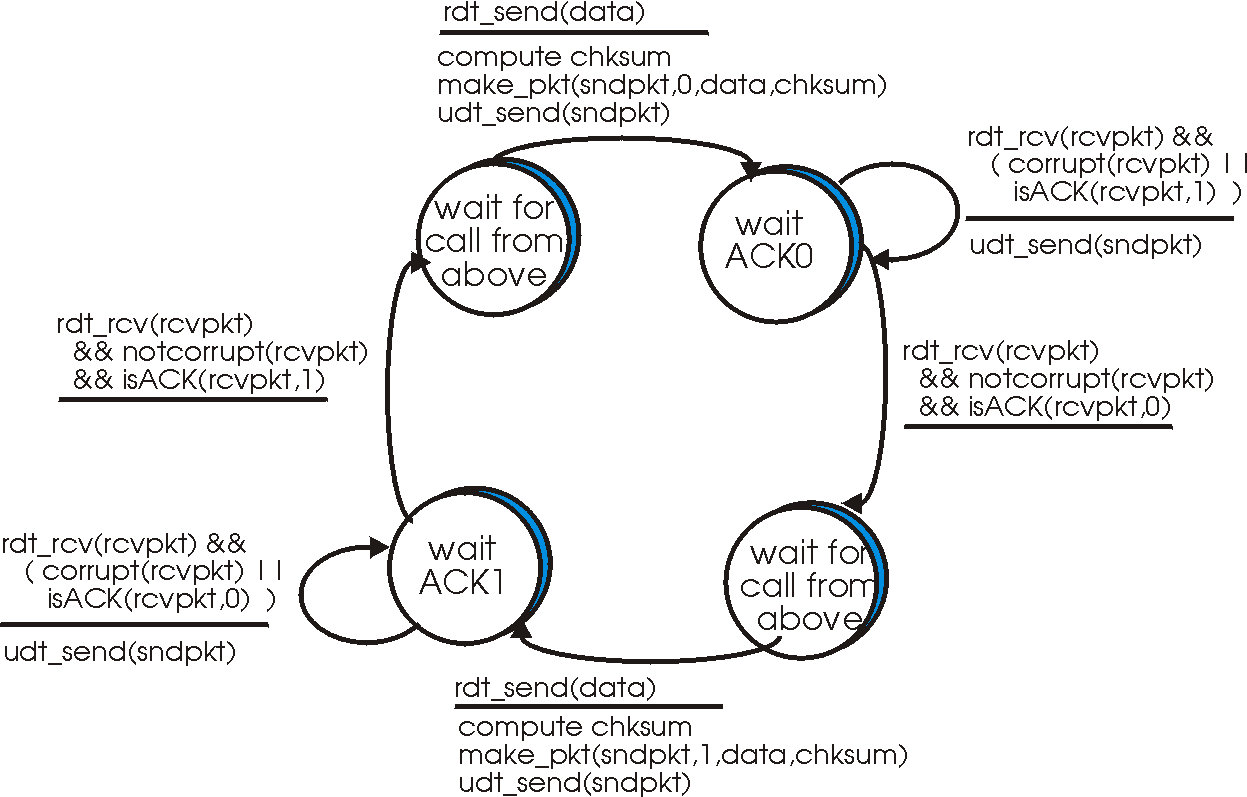


Figure 3.1: RDT 2.2 Sender Side

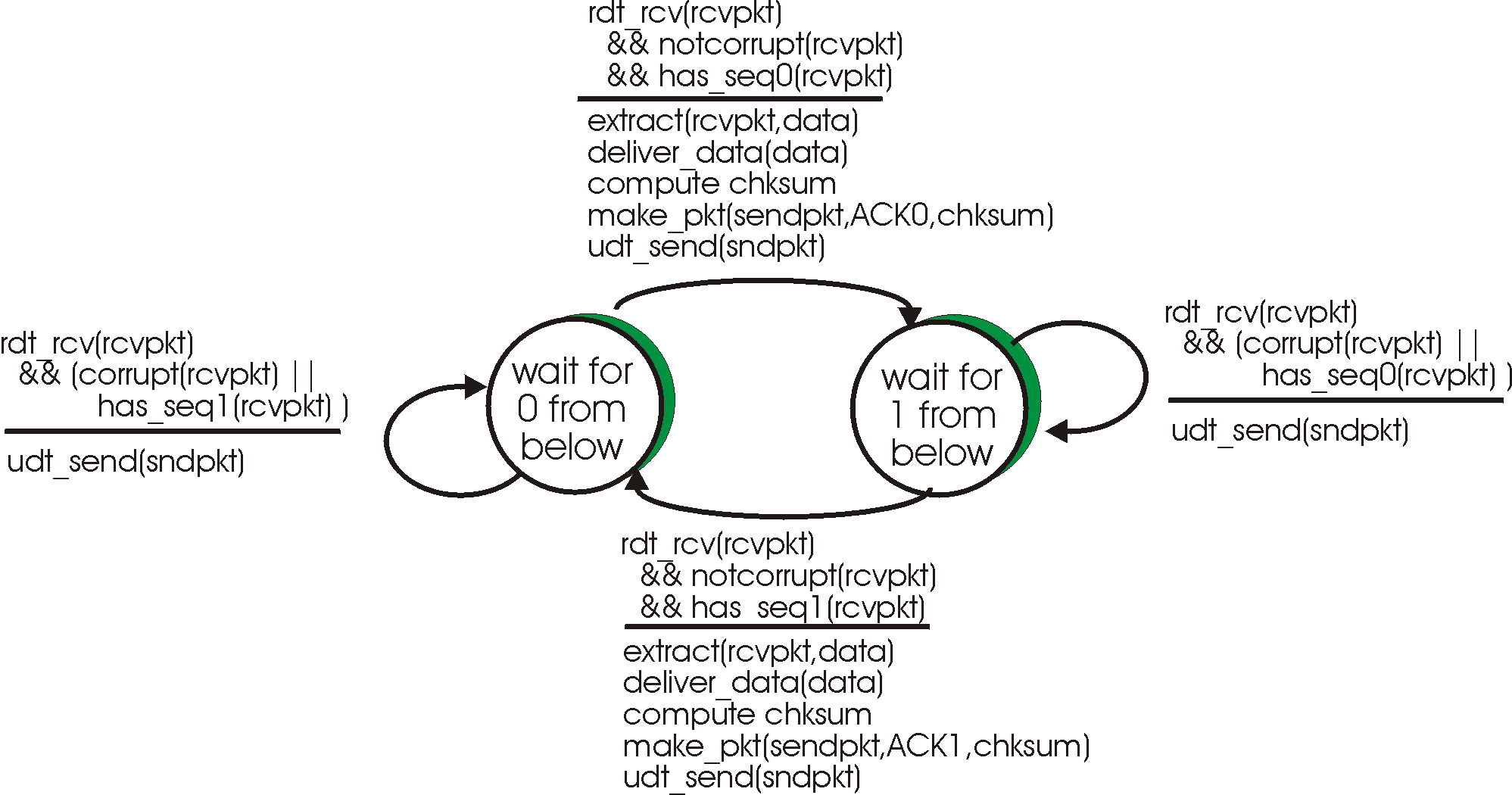


Figure 3.2: RDT 2.2 Receiver side

# Implementing UDP Reliability

The project we worked on was concerned with creating an API. The functions necessary for server and client communication are encapsulated in two classes with its associated functions. The server and client files that are included was used for testing of the API created.

User Manual for the demo (Single Computer):

1. Download file
2. Extract zip file (Project3150.zip) to folder Project3150
3. Enter the folder Project3150
4. Shift + Right Click an empty space then click “Open command window here”, repeat two more times
5. First command window:
   1. Type “javac Rdt.java” and hit enter key
6. Second command zwindow:
   1. Type “javac Server.java” and hit enter key
   2. Type “java Server” and hit enter key
7. Third command window:
   1. Type “javac Client.java” and hit enter key
   2. Type “java Client” and hit enter key
8. You will observe that Strings are transferred over the network from the server to the client. Should any errors be encountered (as seen in RDT 2.2) it will be shown on the related command window

User Manual for the demo (Two Computers):

It depends on the computer that acts as a client or server. For a client, perform all steps as shown above, **but for step 4 (Open only 2 windows) and exclude step 6.**

For a server, perform all steps as shown above, **but for step 4 (Open only 2 windows) and exclude step 7.**

## Testing UDP Implementation

Testing UDP requires either one computer or two computers. The unreliability of a network determines the unreliability of data transfer over UDP. Testing UDP utilizing only one computer will not be as effective of a UDP test between two computers, because the reliability of the network will be much greater compared to two computers at remote locations. Because of this, for testing purposes, a simulation of a network will be created, by altering the chances of a packet being corrupted or transferred in the wrong order before it is actually sent. In the three examples to follow, it will be shown how RDT (Reliable Data Transfer) using UDP can be set up on a single computer, or on a single computer (with a simulated unreliable network) or on two separate remote computers.

### Single Computer

On line 18 it can be seen that “localhost” was used. Localhost is used to refer to the computer that is currently being used. The address that was supposed to be placed on line 18 refers to the computer the client connects to, to receive data. Since a single computer will be used, the server will be the current computer also. The port in line 16 can be any port that is available. From line 21 to 24, an initial data transfer occurs for the server to establish which client it is connected to and its port being used.

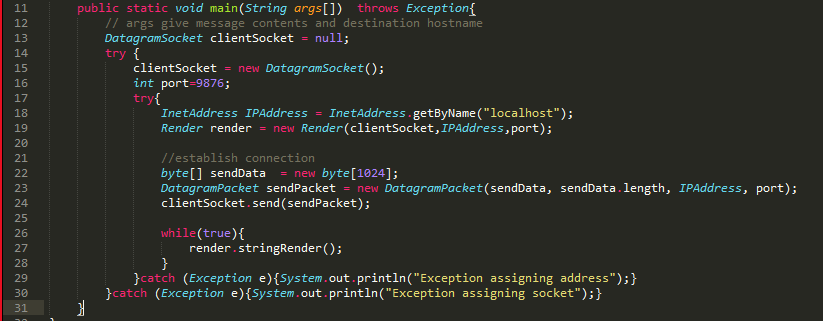


Figure 1: Client

Line 36 stores the IPAddress of the computer that it is connected to. Line 37 stores the port of the computer that it is also connected to.

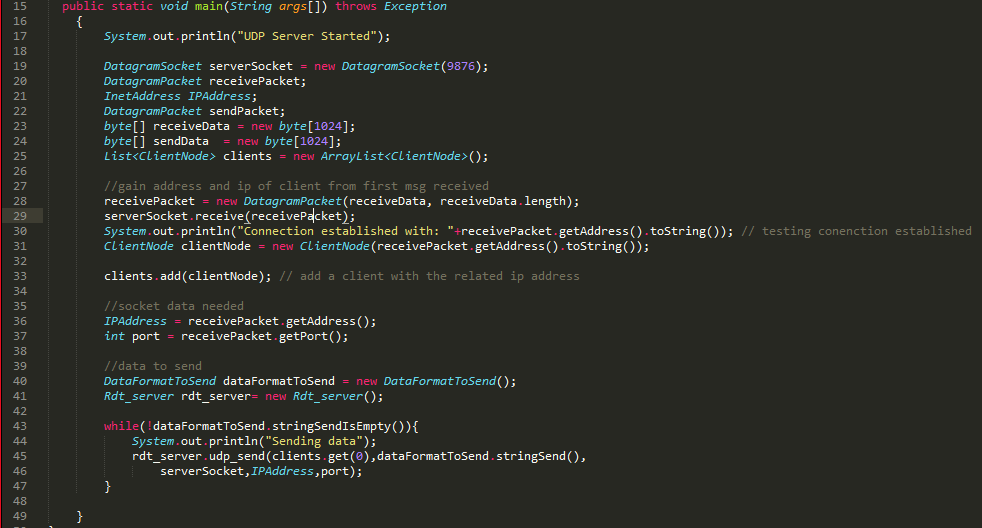


Figure 2: Server

### Single Computer (Simulated Network)

On a single computer the corruption of packets can be simulated by line 137-138. The concept that is utilized here is, once the packet to be sent is constructed, a RNG(Random Number Generator) determines which packet of the set of packets made by this package is to be altered. This altering is akin to the data corruption that can and will occur across a UDP network that does not implement a version of RDT.

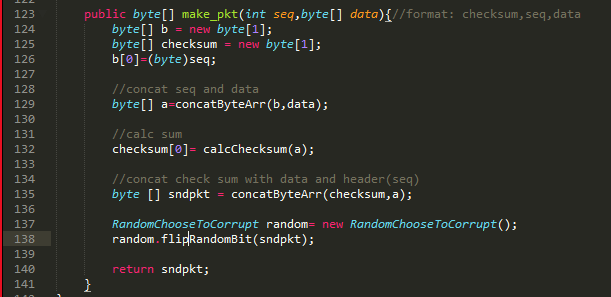


Figure 3: Packet altering

Naturally, if an unreliable network is to be simulated, the “degree” of unreliability should be alterable. On line 150, the ‘int’ in the function call ‘rand.nextInt’(15) can be changed. The degree of unreliability can be viewed as a 1 in 15 chance of the packet being manually corrupted.

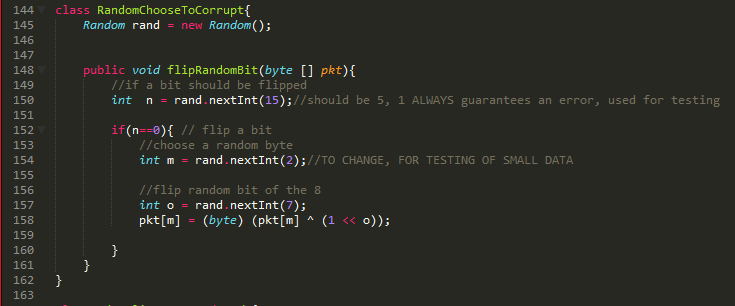


Figure 4: Altering degree of unreliability to be simulated

### Two Remote Computers

For two remote computers, the server side does not change from what is seen in Figure 2. On the other hand, in the client side, the only change from what was seen with “Single Computer”, is the IP address of the server to be connected to, that is, line 18 as shown below.

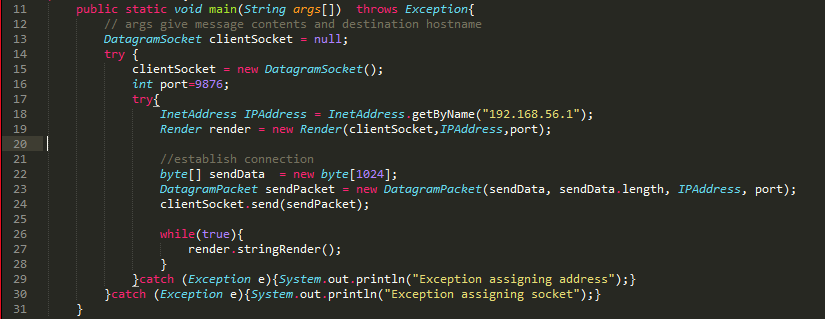


Figure 5: Client side for remote computers

# Conclusion

The team was able to introduce reliability (specifically RDT 2.2) into the application layer of UDP. From this project, we were able to view why UDP may be preferred over TCP despite lacking in established protocols for reliable data transfer. We were able to conclude that in the application layer, the degree of reliability can be coded for supporting the exact needs of what needs to be transferred, how fast and the distance etc.

# Bibliography

Ross, K. W. (2013). *Computer Networking: A Top-Down Approach (6th Edition).*