Instructions

Due: 10/3/16 11:59PM

Complete the following assignment in pairs, or groups of three. Submit your work into the Dropbox on D2L into the "Programming Assignment 2" folder. Both partners will submit the same solution and we will only grade one solution for each group.

Learning Objectives

In this lab you will:

- Work with a layered network architecture
- Understand and implement the Stop-and-Wait Protocol with ACK (Acknowledgments), NACK (Negative Acknowledgments), and re-transmissions

Assignment

During this project, you will implement the Reliable Data Transmission protocols RDT 2.1, and RDT 3.0 discussed in class and the textbook by extending an RDT 1.0 implementation.

Starting Code

The starting code for this project (`prog2.zip` in the D2L content area) provides you with the implementation several network layers that cooperate to provide end-to-end communication.

```
APPLICATION LAYER (client.py, server.py)
TRANSPORT LAYER (rdt.py)
NETWORK LAYER (network.py)
```

The client sends messages to the server, which converts them to pig latin, and transmits them back. The client and the server send messages to each other through the transport layer provided by an RDT implementation using the `rdt_1_0_send` and `rdt_1_0_receive` functions. The starting `rdt.py` provides only the RDT 1.0 version of the protocol, which does not tolerate packet corruption, or loss. The RDT protocol uses `udt_send` and `udt_receive` provided by `network.py` to transfer bytes between the client and server machines. The network layer may corrupt packets or lose packets altogether. Your job will be to extend `rdt.py` to tolerate packet corruption and loss. The provided implementation of `network.py` is reliable, but we will test your code with non-zero probability for packet corruption and loss by changing the values of `prob_pkt_loss` and `prob_byte_corr` of the `NetworkLayer` class. You may change those variables yourself to test your code.

Program Invocation

To run the starting code you may run:

```
python server.py 5000
```

and

```
python client.py localhost 5000
```

in separate terminal windows. Be sure to start the server first, to allow it to start listening on a socket, and start the client soon after, before the server times out.

BONUS 1

The network layer may also reorder packets. If you set prob_pkt_reorder to a non-zero probability you will start seeing packet that are reordered. I will award **one bonus point** to any group that implements RDT 3.1, which delivers packets in the correct order.

BONUS 2

The RDT 3.1 is a stop and wait protocol. I will award **one bonus point** to any group that implements RDT 4.0 – a pipelined reliable data transmission protocol based on either Go-back-N (GBN), or Selective Repeat (SR) mechanisms.

What to Submit

You will submit you version of `rdt.py`, which implements the send and receive functions for RDT 2.1, and RDT 3.0. RDT 2.1 tolerates corrupted packets through retransmission. RDT 3.0 tolerates corrupted and lost packets through retransmission. The necessary functions prototypes are already included in `rdt.py`. For the purposes of testing you may modify `client.py` and `server.py` to use these functions instead of those of RDT 1.0.

We will grade the this assignment as follows:

1. [2 points] `partners.txt` with your partner's, or partners' first and last name.
2. [10 points] `rdt.py` that correctly implements RDT 2.1
3. [13 points] `rdt.py` that correctly implements RDT 3.0
4. [1 point] (BONUS 1) `rdt.py` that correctly implements RDT 3.1
5. [1 point] (BONUS 2) `rdt.py` that correctly implements RDT 4.0

Acknowledgements

This project was adapted from Kishore Ramachandran version of this assignment.