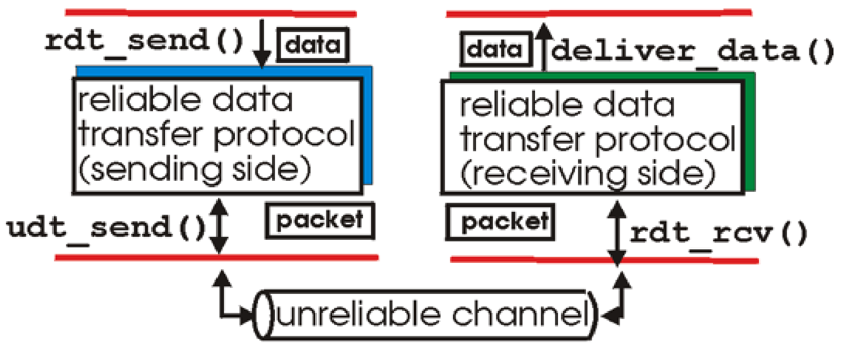
Programming Project 3 – Reliable Data Transmission (RDT)

Introduction

In this project you will code part of a simulation of reliable data transmission over an unreliable underlying system. You will complete the RDTLayer class that allows the transfer of string data through an unreliable channel in a simulated send-receive environment.


rdt_send()
data
reliable data transfer protocol (sending side)
udt_send()
packet
data
deliver_data()
reliable data transfer protocol (receiving side)
packet
rdt_rcv()
unreliable channel

Code for the UnreliableChannel class, the Segment class (data or ack packets**), and also the calling main function, is provided. You will need to complete the implementation for the RDTLayer class. If you read the rdt_main.py file, you'll notice that the RDT_Layer class is used for both the client and server instance. Sending data is done in processSend() and receiving data is done in processReceiveAndSendRespond(). Both methods will have to identify if this object instance is the client or server, and act accordingly. So, the RDTLayer must therefore be able to send and receive data and ack segments.

Your code should be able to simulate many of the features discussed in your textbook for RDT and TCP. Note that we will not be using actual TCP headers or bytes. This is more high-level than that, but the principles are the same.

RDTLayer

The RDTLayer class needs to provide reliable transport no matter how bad the unreliable channel is at delivering packets. And it is bad! The UnreliableChannel class may do any of the following:

- Deliver packets out-of-order (data or ack packets)
- Delay packets by several iterations (data or ack packets)
- Drop packets (data or ack packets)

- Experience errors in transmission (failed checksum - data packets only)
- Various combinations of the above

***The words 'segment' and 'packet' are frequently used interchangeably with TCP communications.*

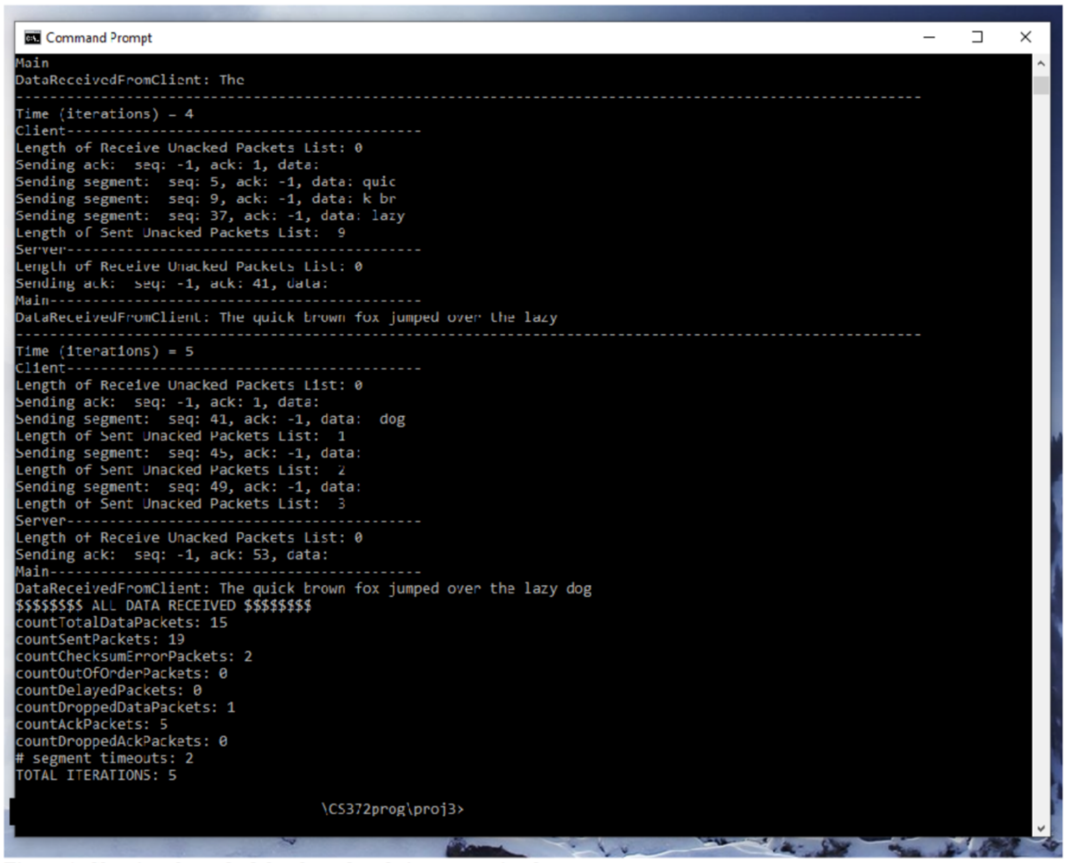
The RDTLayer class must handle all of these and deliver the data correctly and as efficiently as possible. In this project, you must finish the implementation of the RDTLayer class. To help build your RDTLayer implementation, the UnreliableChannel class takes flags that can turn on/off the different types of unreliability. It is recommended that you use those flags, starting with completely reliable channels, get that working, and go from there, enabling each of the unreliable features in turn.

Note that the Segment class already has methods for calculating and checking the checksum. It also has methods for saving a 'timestamp' (iteration count). We will be using iteration count instead of time, because the simulation executes much too quickly to use actual time.

Your final RDTLayer implementation must:

- Run successfully with the original provided UnreliableChannel and Segment classes
- Deliver all of the data with no errors
- Succeed even with all of the unreliable features enabled
- Send multiple segments at a time (pipeline) (This project program is operating in discrete time slices/iterations rather than continuous time. If you are sending the maximum amount of data you can at each iteration (based on the state of the available flow control window) instead of strictly a single segment at each iteration, you are likely meeting the pipeline requirement.)
- Utilize a flow-control 'window'
- Utilize cumulative ack
- Utilize Go-Back-N or selective retransmit
- Include segment 'timeouts' (use iteration count, not actual time)
- Abide by the payload size constant provided in the RDTLayer class
- Abide by the flow-control window size constant provided in the RDTLayer class
- Efficiently run with the fewest packets sent/received. Who can transmit all of the data with the fewest iterations?

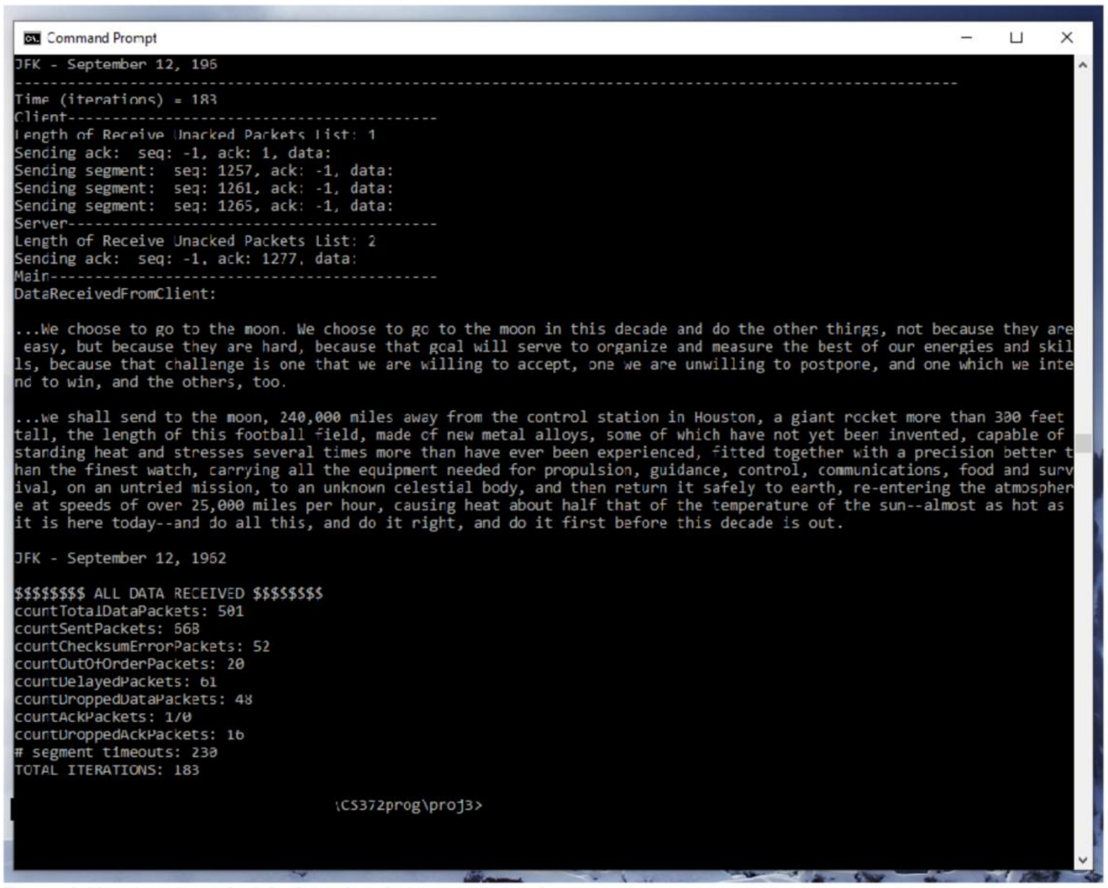
Example screenshots:



```
Main
DataReceivedFromClient: The
----------------------------------------------------------------------------------------------------------
Time (iterations) = 4
Client------------------------------------------
Length of Receive Unacked Packets List: 0
Sending ack:  seq: -1, ack: 1, data:
Sending segment:  seq: 5, ack: -1, data: quic
Sending segment:  seq: 9, ack: -1, data: k br
Sending segment:  seq: 37, ack: -1, data: lazy
Length of Sent Unacked Packets List:  9
Server------------------------------------------
Length of Receive Unacked Packets List: 0
Sending ack:  seq: -1, ack: 41, data:
Main--------------------------------------------
DataReceivedFromClient: The quick brown fox jumped over the lazy
----------------------------------------------------------------------------------------------------------
Time (iterations) = 5
Client------------------------------------------
Length of Receive Unacked Packets List: 0
Sending ack:  seq: -1, ack: 1, data:
Sending segment:  seq: 41, ack: -1, data:  dog
Length of Sent Unacked Packets List:  1
Sending segment:  seq: 45, ack: -1, data:
Length of Sent Unacked Packets List:  2
Sending segment:  seq: 49, ack: -1, data:
Length of Sent Unacked Packets List:  3
Server------------------------------------------
Length of Receive Unacked Packets List: 0
Sending ack:  seq: -1, ack: 53, data:
Main--------------------------------------------
DataReceivedFromClient: The quick brown fox jumped over the lazy dog
$$$$$$$$ ALL DATA RECEIVED $$$$$$$$
countTotalDataPackets: 15
countSentPackets: 19
countChecksumErrorPackets: 2
countOutOfOrderPackets: 0
countDelayedPackets: 0
countDroppedDataPackets: 1
countAckPackets: 5
countDroppedAckPackets: 0
# segment timeouts: 2
TOTAL ITERATIONS: 5

[redacted]\CS372prog\proj3>
```

Figure 1: Showing the end of the short data being transmitted.



```
JFK - September 12, 196
----------------------------------------------------------------------------------------------------------
Time (iterations) = 183
Client------------------------------------------
Length of Receive Unacked Packets List: 1
Sending ack:  seq: -1, ack: 1, data:
Sending segment:  seq: 1257, ack: -1, data:
Sending segment:  seq: 1261, ack: -1, data:
Sending segment:  seq: 1265, ack: -1, data:
Server------------------------------------------
Length of Receive Unacked Packets List: 2
Sending ack:  seq: -1, ack: 1277, data:
Main--------------------------------------------
DataReceivedFromClient:

...We choose to go to the moon. We choose to go to the moon in this decade and do the other things, not because they are
 easy, but because they are hard, because that goal will serve to organize and measure the best of our energies and skil
ls, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we inte
nd to win, and the others, too.

...we shall send to the moon, 240,000 miles away from the control station in Houston, a giant rocket more than 300 feet
tall, the length of this football field, made of new metal alloys, some of which have not yet been invented, capable of
standing heat and stresses several times more than have ever been experienced, fitted together with a precision better t
han the finest watch, carrying all the equipment needed for propulsion, guidance, control, communications, food and surv
ival, on an untried mission, to an unknown celestial body, and then return it safely to earth, re-entering the atmospher
e at speeds of over 25,000 miles per hour, causing heat about half that of the temperature of the sun--almost as hot as
it is here today--and do all this, and do it right, and do it first before this decade is out.

JFK - September 12, 1962

$$$$$$$$ ALL DATA RECEIVED $$$$$$$$
countTotalDataPackets: 501
countSentPackets: 568
countChecksumErrorPackets: 52
countOutOfOrderPackets: 20
countDelayedPackets: 61
countDroppedDataPackets: 48
countAckPackets: 170
countDroppedAckPackets: 16
# segment timeouts: 230
TOTAL ITERATIONS: 183

[redacted]\CS372prog\proj3>
```

Figure 2 Showing the end of the long data being transmitted

What to turn in

1. Include a copy of all of your code.
2. In rdt_main.py, you will see two different length texts to send through the RDTLayer. Start with the smaller one, then try the larger one.
3. Take screenshots of your running code for both the larger and smaller texts and include it in your submission doc.
4. Add any comments/questions to your submission doc. You can include your assumptions that helped narrow the requirements for you.