Reliable Transport Assignment

# CSCI 4760

**Due date**: End of Spring Break. Finish beforehand to enjoy a relaxing holiday!

**What is this**? In this assignment you will implement the details of reliably transmitting data across an unreliable network. The design principles are covered in the Transport Layer chapter of our text. We will use cumulative acknowledgement with NAKs.

In order to simulate an unreliable network, I will set up a relay station on my VM. UDP datagrams sent to my relay station will sometimes be forwarded correctly to their destination, sometimes corrupted and forwarded, and sometimes dropped entirely. For this assignment, datagrams will not be delivered out of order.

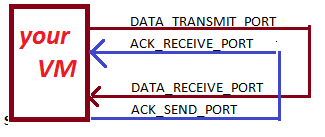
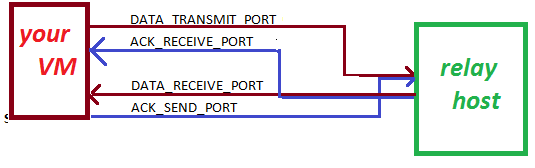
**Standard UDP ports to use**: Unlike TCP, UDP allows only one socket to be bound to the same port on the same host. I suggest the following set of four port numbers:

* 2015 = DATA\_TRANSMIT\_PORT, used by sender to transmit data segments;
* 2016 = ACK\_RECEIVE\_PORT, used by sender to receive ACK and NAK segments;
* 2017 = DATA\_RECEIVE\_PORT, used by receiver to receive data segments;
* 2018 = ACK\_SEND\_PORT, used by receiver to send ACK and NAK segments.

**Steps in completing this assignment**:

1. Implement the ReliableTransportMessage class, using the provided JUnit test class. This will involve encoding and reconstituting a message, just as in the first assignment. Hopefully our experience will help this run a bit more quickly this time!
2. Implement the ReliableSender and ReliableReceiver classes. These classes wrap ReliableTransportMessage objects inside UDP datagrams, which makes this assignment an example of ***tunneling***: encapsulating messages at one layer inside messages of another protocol at the same layer.

Here are the iterations to get to a successful conclusion. For each iteration, you should be able to transmit a text file of 3000 bytes or more and display it correctly on the receiver’s standard output (which could be redirected into a file). This size is chosen because it will force your sequence numbers to “wrap around” to zero. I suggest using a fragment of one of your favorite books or short stories as the text file. It helps to use a pure text file, rather than a word processing document.

* + **Iteration 1**: directly communicate with your own receiver on your own host. In this case the underlying UDP delivery is reliable, but you should still acknowledge packets .
  + **Iteration 2**: communicate through my reliable relaying service on port **2019** of host 172.17.152.122. Both data and ACKS should go through the relay. Because the encapsulated ReliableTransportMessage objects contain the destination IP address and UDP port number, your messages should be forwarded to their intended destination.
  + **Iteration 3**: communicate through my “semi-reliable” relaying service on port **2020** of host 172.17.152.122. This service mangles about 20% of the DATA packets, but correctly transmits all ACK and NAK packets.
  + **Iteration 4**: communicate through my unreliable relaying service on port **2021** of host 172.17.152.122. This service mangles about 20% of both ACK and DATA packets, and drops another 10% of both ACK and DATA packets.
  + **Iteration 5**: communicate via the unreliable relayer with two of your classmates.

# Grading Rubric

|  |  |
| --- | --- |
| ***Accomplishment*** | ***Points*** |
| ReliableTransportMessage passes JUnit tests | 20 |
| Iteration 1: Communicate with self on local VM | 30 |
| Iteration 2: Communicate with self via reliable relayer | 20 |
| Iteration 3: Communicate with self via semi-reliable relayer | 10 |
| Iteration 4: Communicate with self via unreliable relayer | 10 |
| Iteration 2: Communicate with two others via unreliable relayer | 10 |