Project 2: Reliable UDP
Assigned: 9/26/2015, Due: 10/10/2015

Project Goal

Today, applications have two choices when it comes to the transport channel: TCP or UDP. While UDP is an unreliable protocol, TCP provides reliability, flow control, and congestion control. TCP has an explicit connection establishment phase, which some application may not find desirable. Your job in this project is to design and implement a file transfer application which has all the good features of TCP without the connection establishment phase. Your application, **reliable UDP**, should build its transport functionality on top of UDP. The goal of this project is to make you familiar with the implementation of TCP and to solidify your understanding of the fundamental networking issues related to reliable data transfer.

Project Specification

You must implement both server and client sides for **reliable UDP**. Your programs must be written in C/C++ and should work on the Burrow or Shark Linux machines. The steps involved in finishing this project are the following:

1. **Design header:** Design a header that both the server and the client will use. This header will be a simplified version of the actual header TCP uses. The simplifying assumptions you should make are: 1) Your packet size should not exceed 1500 bytes, including IP, UDP, reliable UDP header and data , 2) no *options* exist, implying that the header length is fixed, 3) the only *flag* to be supported is the **ACK** flag, which is used to indicate whether the packet contains data or acknowledgment, 4) the *AdvertisedWindow* is a fixed parameter, negotiated offline and not part of the header (your server and client programs should accept this as a command line parameter), and 5) there cannot be any urgent data, implying that there is no need to provision for *UrgPtr*. Make sure to discuss the header you design in your project write-up.
2. **Implement sliding window algorithm for reliability:** The receiver should use *duplicate acknowledgments* to indicate a lost or out-of-order packet. Cumulative acknowledgments are allowed.
3. **Implement adaptive retransmission:** Implement Jacobson/Karels algorithm to estimate round trip time (RTT).
4. **Implement congestion control:** TCP has two distinct phases of data transmission: *slow start* and *congestion avoidance (additive increase/multiplicative decrease).* Your program should implement both of these. The output of your program should indicate when the switch from slow start to congestion avoidance occurs, along with the number and percentage of packets that are transferred in each phase.

5. **Simulate loss and delay:** Your client program should simulate various network conditions. Toward that goal, you should enable and disable each of these features as needed during your demonstration:
    - **Variable packet loss rate:** You can simulate different rates of packet loss by probabilistically dropping packets. You perform probabilistic dropping by using the *rand()* function and a drop percentage.
    - **High latency communication:** By adding delays at the receiver (before processing a received packet), you can simulate increased latency. You may use the *sleep()* system call to add delays.

    Experiment with these parameters even before the demonstration and document their effect on the number of packets transmitted in *slow start* and *congestion avoidance* phases.

Resources

- Section 3.7 of your textbook discusses TCP's congestion control. See section 3.5 for other aspects of TCP.
- Pseudo code for the Jacobson/Karels algorithm
    - Estimated_RTT = (1-α) Estimated RTT + (α) Sample_RTT
    - In the original TCP Specification, α=.0125
    - Jacobson/Karels included a variation component to the calculation for the Estimated_RTT
    - Estimated_RTT = Estimated_RTT + δ (Sample_RTT-Estimated_RTT)
    - Deviation = Deviation + δ (|Sample_RTT- Estimated_RTT|- Deviation)
    - Timeout = μ * Estimated_RTT + φ * Deviation
    - Typically φ=4, μ = 1, δ is between 0 and 1

Extra Credit (10% bonus points)

- **Implement selective acknowledgments:** Your program should be able to switch between duplicate and selective acknowledgments. You should run your program under varying circumstances to determine if this approach provides any benefit. Describe the results in your write-up.
- **Provide sequence number wrap-around protection:** You must implement an approach to prevent ambiguity when the sequence number wraps around. You may choose the approach the book suggests or you may use your own approach. It is NOT acceptable to simply increase the number of bits in the sequence number, since this only delays wrap-around, rather than solving it. If you choose this option, you must make it possible to set the initial sequence number such that it will quickly wrap-around.

Deliverables and Grading

The grading will be based on the following three deliverables, all of which are due by 11.59pm on the due date:

- **Code submission and demo:** Submit your code as a single file archive (.tar or .tar.gz file formats only) via Canvas.
- **Project report:** Prepare a 2-page report detailing your implementation. The report should use a 10-point font, with margins on any of the four sides not exceeding 1 inch. Include the report in your code submission.