Project 3 : Reliable UDP

CSCI P538 Computer Networks

Group Member: Shengyu Chen, Xiner Zhang

Project Report

UDP is known as connectionless prototype. It has been widely used by numbers of web application. Because it avoid building connection with three way handshake. It is relatively faster than other web application using TCP. However, traditional UDP has a serious potential problem, it does not have congestion control. When network is busy, some data may be lost because of the buffer overflow on routers. In other work, UDP is unreliable. Therefore, in this project we implement a reliable UDP with flow control, and congestion control. The project is able to perform file transfer with all the good feature included in TCP without three way handshake.

Our project is written in C with both the client side and the server side. Below are some major features we implemented in this project.

Design Header: In tradition UDP prototype, UDP segment structure contains the following elements: source port, destination port, length, and checksum. These information are necessary for each packet being transfer. In this project we design a UDP header with four different variables. They are sequence number, ACK flag, ACK number and packet size. Each data packet transferring with this reliable UDP prototype will contain these three variables and a data buffer. Both the client and the server will adapt this same pattern of packet header. The sequence number of the packet being acknowledged by an ACK flag. The ACK flag has the value of either -1 or 1. This will acknowledge if data is received by the server. ACK number and packet size will be used to determine if a packet lost has appeared or not while sending back information to client.

Implement sliding window algorithm for reliability: As the most important feature in this project, our group implemented a sliding window algorithm to ensure reliability. At the server side, the receiver window is set to a constant of 20 maximum segment size. After transfer each packet, we update the starting point of the sliding. If the current sequence number and the new starting point of the sliding window is the same, than we can conclude there is no packet loss and pursue for the next packet. Otherwize, we will transmit this packet again. At the client side, the sender window may vary. If the congestion window is smaller, than sender window will set to the same size of the congestion window. If the receiver window is smaller, than the sender window will set to the same size of the

receiver window. The sliding window will limit how much data source in transit. The sender window is able to tell us if the current packet has finish transmitting by comparing the current byte confirmed with the finishing point of the window.

Implement adaptive retransmission: In this project we also maintain an estimate round-trip time (RTT) to detect timeout. When UDP starts a transfer, it will also start a timer. When it receive an ACK for this packet, it will stop the timer. In this project we use Jacobson/Karels algorithm to calculate this estimate round-trip time. We set the timeout base on this esmited RRT. Because we need to ensure that the timeout is larger than the estimated RRT. This is also done by the Jacobson/Karels algorithm. When a timeout do occurs, it will send the timeout packet again. In the meanwhile, the value of timeout will be doubled to avoid a premature timeout occurring for a subsequent segment that will soon be acknowledged.

Implement congestion control: In TCP congestion-control algorithm, it has three major features. They are slow start, congestion avoidance and fast recovery. Especially slow start and congestion avoidance. These are mandatory features for TCP congestion control. We adopted these features in our reliable UDP program as well. When we first start transmitting data from client, the congestion window is set as 1 maximum segment size. However, the size will congestion window will grow every time it recieve a acknowledged segment. Therefore, our UDP transfer starts slow and grow exponentially during the transmission. But when congestion is recognized by a timeout, the congestion window will set back to the initial value and start slow again. It also sets the value of a second state variable, ssthresh (slow start threshold) to half of the value of the congestion window value. If the value of cwnd is the same as the ssthresh, the program will end the slow start and start the congestion avoidance mode. In congestion avoidance mode, instead of increasing congestion window exponentially, we will increase it linearly by adding 1 maximum segment size each round trip.

We have test the project with different size of files transfer. The program is able to transmit the files without lost content while we simulate loss and delay. We included a makefile for easier compiling and testing. In conclusion, we are able to design and implement a reliable UDP file transmitting with a lot of the congestion control feature in TCP. We have a sliding window for each of the client and the server. We calculate a more accurate estimate timeout time with Jacobson/Karels algorithm. We also includes congestion control with slow start and

congestion avoidance. In the meanwhile, we are able to avoid setting up a connection with the three way handshake.