Project 2: Simple Window-based Reliable Data Transfer

Eric Chiang, 204 290 453
June Lee, 604 500 920

In this project we are using UDP protocol with our own layer that would replicate Go-Back-N protocol. The requirement was to keep the packet size under 1KB, but we opted to make the packet sizes to be much less (the data portion itself is 256B).

Implementation Description

Compile

To compile the project a user can simply use “make” at shell to create the two programs “sender” and “receiver.”

Sender

To run this program a user can simply type in ./sender <portnumber> <CWnd> <Pl> <Pc>. The <portnumber> is a mandatory field, while the other options are not. The <CWnd> field is the user determined size of window, if none specified then it is defaulted to 5. Pl and Pc should range between 0 through 1, for they are the probabilities of packet loss and corruption.

Receiver

To run this program a user can simply type in ./receiver <portnumber> <filename> <Pl> <Pc>. The <portnumber> and <filename> are mandatory fields, while the other options are not. Pl and Pc should range between 0 through 1, for they are the probabilities of packet loss and corruption.

Header Format

For the header we decided to include the signal type (an ack, request, etc), sequence number, source port number, destination port number, and the total size of the data. The header file is included in struct pack (which is our packet data structure) which also includes a portion for a data up to size 256B.

Messages

The only real messages that sender and receiver would transmit is the request and the ack.

Timeouts

Our standard timeout time is 2s, upon the timer exceeding this amount then the program will behave like Go-Back-N and sender will retransmit everything in it’s window.

Window-Based Protocol

We decided to use the Go-Back-N protocol where the receiver will only accept packets sequentially and will only ack the ones received sequentially. The sender tries to send all of it's window content, but upon timeout will "go back" and resubmit from last good sequence.

Loss and Corruption

For losses we use rand() at the given probability of 'Pl.' If the packet were to be lossed both receiver and sender do not send a packet or an ack for that instance. For a corrupted file, we send the signal 'COR' and the side that receives this packet or ack simply ignores it.

Difficulties

We had many difficulties, usually due to how we were handling the memory. We tried to smoothen out all issues by extensively managing the way we use memory.