COMP312-09A - Connectionless network communications with error detection

March 26, 2009

1 Introduction

The aim of this assignment is to explore connectionless network data services and error detection.

You will implement Java class providing a reliable transport protocol for sending data across an unreliable network connection. The reliable transport protocol will run over the provided DodgyDatagramSocket class, which replaces DatagramSocket. The DodgyDatagramSocket will randomly delay, corrupt, discard, and reorder packets you send through it, so you will need to do a good job of your ReliableDatagramSocket to get the file through.

To make use of your reliable transport class you should implement a simple client and server for uploading files. Your client and server applications will not have any features implemented at the application layer to provide reliability, like TFTP has. Instead, this will be done with your ReliableDatagramSocket. Your client does not have to tell the server the name of the file being transmitted; your server can just store the contents in `file001`. The focus of this assignment is in the ReliableDatagramSocket implementation; there are no marks provided for your file transfer application, so you should not spend much time on it, beyond what is required to get it working and demonstrate your ReliableDatagramSocket.

You can get 6/10 marks for this assignment by successfully implementing a stop-and-wait protocol in ReliableDatagramSocket. To get full marks, you need to successfully implement a sliding window protocol and do some elementary performance evaluation.

Your assignment will be verified in person in R-block Lab 6. You should aim to complete your assignment by Monday April 27th. I will be in the lab to verify this assignment on Monday at 10am before the lecture, and then again from 12. It is worth 10% of your final grade.

2 Advice

The DodgyDatagramSocket class provides the following methods.

```
public class DodgyDatagramSocket extends DatagramSocket {
    public DodgyDatagramSocket(int i);
    public DodgyDatagramSocket();
    public void send(DatagramPacket pkt);
    public void setDropChance(double d);
    public void setCorruptChance(double d);
    public void setDelayChance(double d);
    public void setDelayLen(int i);
    public void setVerbose(boolean flag);
}
```

When I verify your assignment, you should not be using any of the set functions. However, if it aids you in debugging your ReliableDatagramSocket implementation, you can use them.

Java provides a `java.util.zip.CRC32` class which is useful for calculating a checksum over a byte array. I suggest you use that.

Use the Linux command line tool `md5sum` to check that the file you received is identical to the file sent.

Suggested method of send() operation for stop and wait:

1. Read a packet-sized chunk from the file into a byte buffer.
2. Form a DatagramPacket containing the data and send this to your ReliableDatagramSocket.
3. Inside ReliableDatagramSocket, allocate a new DatagramPacket that is slightly larger to contain your header.
4. Transmit this to the server.
5. Wait for acknowledgement. Retransmit after a timeout if you do not receive one.
6. Once acknowledged, return back to the client.

To implement a sliding window protocol, you will need to extend your send routine to accept new packets to transmit until the window is full. The receive routine will need to keep any data received out of order until all previous packets have arrived. Once packets are in order they should be delivered to the application.

In the sender, as you receive datagrams from the application and have to deliver these datagrams in the correct order to the remote application, it makes sense for your sequence numbers to relate to datagrams, not bytes. Your sender will need to keep track of:

- `sendSeqNext` - The next sequence number to be sent
- `sendSeqUnAck` - The oldest sequence number not acknowledged.
- `sendSeqMax` - The maximum number of packets that can be unacknowledged.
- `sendData[]` - An array to store all packets that have not been acknowledged.

Your receiver will need to keep track of:

- `recvSeqCur` - The sequence number of the last segment received correctly and in sequence.
- `recvSeqMax` The maximum number of segments that can be buffered.
- `recvData[]` An array to store all packets received out of order.

The easiest way to implement ReliableDatagramSocket sliding window is to split it into threads.

- one thread to listen on the DodgyDatagramSocket for incoming packets, check, re-order and then deliver these packets to the application.
- one thread waiting to be given packets to be sent.

Remember to use `synchronize` on any variables shared amongst the threads!

3 Questions

To complete the assignment, please write answers in the space provided. This forms part of the verification process.

Question 1: UDP does not have the ability to signal end of file, as there is no disconnection process. How do you signal end-of-file?

Question 2: Draw the header you prepend in your ReliableDatagramSocket.

Question 3: Briefly describe your ReliableDatagramSocket implementation. If you implement a sliding window, say how you decide a packet has been lost, and what action you take when this occurs.

Question 4: If you implemented a sliding window solution, please report how throughput varies as a function of window size. Choose a suitable large file to send, and window size of 1, 5, 10, and 20 packets.

Assignment 2 Mark sheet

Name: __________

Id: ___________

Date: ___________

First three questions answered well.

(2 marks)

ReliableDatagramSocket implementation. ReliableDatagramSocket works, is well written, does not have serious flaws, and is well commented.

(4 marks)

Sliding window. Implemented a sliding window solution. Answered question four.

(4 marks)