CSC 469 Program #2 – Reliable Data Transfer

Build reliable data transfer on top of an unreliable channel.

1. Overview

Program #2 is worth 8 points or 8% of your final grade.
It's due in on Thursday February 15, 2007. We'll set some intermediate due dates in class.

The goals of program #2 are:

1. Building reliable data transfer on top of an unreliable channel, ala TCP
2. Multi-threaded programming
3. Using UDP and some code to simulate an unreliable channel
4. Keep learning Java

Program #2 can be done individually or in power programming pairs (P-cubed). Good luck!

2. Description

Implement a progression of reliable data transfer (RDT) algorithms described in Chapter 3 of our text and in lecture.

The algorithms

The RDT algorithms are:

- rdt 1.0… no errors (um, just to get your infrastructure debugged)
- rdt 2.2… one-bit errors in both directions (I think we can skip rdt 2.1)
- rdt 3.0… one-bit errors, plus lost packets and out of order packets

Save each of these "rdt" checkpoints somewhere. Then, with your infrastructure and rdt 3.0 in hand, crank out one of these big boy algorithms:

- Go-Back-N (GBN)… all errors with pipelining of segments
- Selective Repeat (SR)… similar to above, different algorithm

If you're working alone, implement either one of these algorithms. If you're working in pairs, then split up an each of you implement either GBM or SR.

Testing
Let's test our algorithms by sending and receiving a file using your fancy algorithm. The receiver and sender programs should run as shown below. The default value for IP should be "localhost" and you can pick default send/receive UDP port numbers.

```
java Sender filename [port] [IP]

java Receiver filename [port] [IP]
```

To test our algorithms, we're going to simulate an unreliable channel using UDP and some code that will play havoc with packets.

Implementation decisions and challenges
There's a lot of scaffolding to build here to get things going. Kurose assigns a variation of our program on page 295. I think that I will copy his goodies and hand them out to you.

Some of the big issues I see:

- Threads… you need to add Java threads to your toolbox because you don't want to block waiting to send/receive segments.
- UDP… I would advise doing UDP *after* you can fake data transfer without them. This is what Kurose does in his "similar" C implementation.
- Corrupting… you'll want to write a class that corrupts packets, either as they are sent or received.
- TCP… remember, we're not implementing TCP, just reliable data transfer, so you don't need all the TCP bells and whistles. Be small.
- Sender/Rcvr interface… exactly how do the sender and receiver send/get messages to/from your reliable transfer guy?
- OOP… being object-oriented will really pay off on a problem like this one. What objects are present in this problem domain? Look at Kurose's C spaghetti and do better.

We'll spend some class time to hash out some of these issues en masse. I'll also try to help by publishing a number of Java interfaces that may help you.

3. Grading

Create a `program02` folder on your k: drive for your deliverables…

- **`README.txt`** – this is your conduit to communicate to me the state of your program, where everything is, how it works, etc.
- **Your design notes** – on the k: drive
- **Your code** – on the k: drive
- **Your documentation** – on the k: drive and w: drive (see below)… the latest, greatest Javadoc magic is:
    - In jGrasp, create a project with all your source files included, then try the `Project/Generate Documentation` menu… that'll give you the cool index for all your source
    - Use the `–author` option to have your `@author` tag displayed
    - Use the `–linksource` option to have a formatted copy of your source code linked into your Javadoc web pages
- **Your tests** – on the k: drive
- **Your web page** – Post your source-linked Javadoc web pages on your noctrl. Web site

Please follow our class coding guidelines at: DTN #3 - Java Coding Guidelines.

4. Hints

There are some nice code examples in our textbook:

- Starting on page 156, of course, you can see his example using UDP
- Starting on page 165, the author shows us his tiny web server. The interesting thing for us may be how he reads and writes files as arrays of bytes in Java. This may be of use to you as you segment your file data into UDP packets that limited in their byte size. Yes?

The `Timer` class in the Java API looks interesting and may be of use to us. I have to research this one myself though.

Oh more pedestrian note, I like the `ArrayList` of all the List collections. It's an auto-resizing array, like `vector` in C++. Also, I saw some "old" list treatment in Program #1. The new "foreach" loop is the way to play:

```
ArrayList<String> names;

// fill up names list

for( String name: names) {
   // name will iterate through each String  in names
}
```