National University of Singapore
School of Computing

CS2105

Assignment 2

Semester 2 13/14

Deadline

30 March, 2014 (Sunday), 11:59pm.

Objective

In this assignment, you will implement a reliable transfer protocol on top of a unreliable channel that can drop packets or corrupt packets randomly (but always deliver packets in order).

Pre-requisite

You are expected to be familiar with the alternating bit protocol (rdt 3.0).

The assignment will be done under a controlled UNIX environment. Familiarity with UNIX environment (how to copy/move/delete/edit files, how to compile and run programs, etc.) is assumed.

Administrative Matters

This is an *individual* assignment.

An account has been setup for you on host cs2105-z.comp.nus.edu.sg. To access your account, ssh to the host and login using your SoC UNIX username and password *from a SoC host or through SoC VPN*.

If you have any questions or encounter any problems with the steps discussed in the assignment, please contact the teaching staff through CS2105's blog.

Introduction

For this assignment, you are given the code for a receiver and sender for three network layers: application, reliable transport (RDT), and unreliable transport (UDT).

The `UDTSender` and `UDTReceiver` classes simulate a networking layer that delivers packets unreliably. A packet can get loss or get corrupted randomly. However, for simplicity, you can assume that this layer delivers packets in order.

The `FileSender` and `FileReceiver` classes implement a file transfer application for this assignment. `FileReceiver` basically waits for a connection, receives a file, and saves the file onto the disk with a given name. To run `FileReceiver`, use

```
java FileReceiver <port> <filename>
```

For example,

```
java FileReceiver 9000 foo.zip
```

listens on port 9000 for connection and dumps the bytes received into a file named `foo.zip`.

The `FileSender` program is basically a file uploader that connects to the `FileReceiver` and sends the data read from a given file to `FileReceiver`. To run `FileSender`,

```
java FileSender <filename> <hostname> <port>
```

For example,

```
java FileSender foo.zip sunfire.comp.nus.edu.sg 9000
```

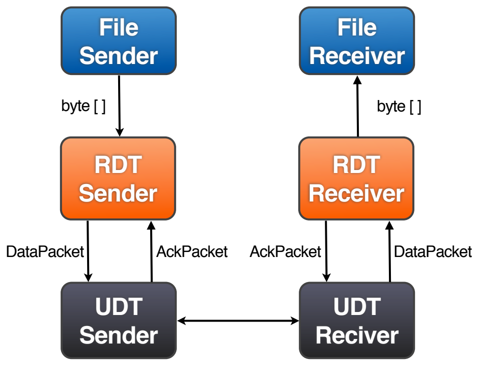

Figure 1: Interaction between the Classes

connects to a `FileReceiver` running on sunfire.comp.nus.edu.sg at port 9000 and uploads `foo.zip` to the server.

Obviously, `FileSender` and `FileReceiver` require data delivery to be reliable. This is where you come in.

Your Tasks

You are also provided with two classes, `RDTSender` and `RDTReceiver`, that interface between the application (`FileSender` and `FileReceiver`) and the unreliable data transport (`UDTSender` and `UDTReceiver`). `RDTSender` and `RDTReceiver` should employ techniques from Lecture 3, including acknowledgement, retransmission, timeout, and sequence number, to make sure that data to be delivered from "above" is sent correctly to the other side. You need not, however, implement windowing nor checksum. You can assume that a corrupted packet can be magically detected through checking the `isCorrupted` flag of a packet.

So, your task in this assignment is to modify the given `RDTSender.java` and `RDTReceiver.java` such that `FileSender` and `FileReceiver` will work correctly – the received file is exactly the same as the file uploaded.

You should read the given set of source code carefully to understand the interaction between the different classes. The code is documented and should be pretty self-explanatory. There are only about 300 lines of code in total.

Figure 1 shows the overall relationship between the classes in this assignment.

How to Do It in Java

Timer

You will need two java.util.* classes to implement timeout. The `Timer` class schedules a task to be executed periodically after a certain delay, while the `TimerTask` class implements the code that you want to execute periodically. For example, to print a given number after 5 seconds, and subsequently every 5 seconds, you create the following task,

```
class NumberPrinter extends TimerTask {
  int x;
  NumberPrinter(int toPrint) {
    x = toPrint;
  }
  public void run() {
    System.out.println(x);
  }
}
```

and schedule the timer elsewhere like this.

```
  Timer timer = new Timer();
  timer.schedule(new NumberPrinter(10), 5000, 5000);
```

When you want to stop printing,

```
  timer.cancel();
```

You can read the Java document on these two classes for more details.

Timers and Timeout Value

Your implementation should work for any retransmission timeout values. You should at least test your code with a timeout value of less than or equal to 5ms (to test the case with premature timeout) and 100ms.

Sending and Receiving Packets

(This operation has been done for you, so this section is for your information only)

To send and receive packets, the given code uses object serialization. Two types of packets are defined, `DataPacket` and `AckPacket`. They are implemented in `DataPacket.java` and `AckPacket.java` respectively. Both classes implement the Java object serialization interface, which allows an object to be "serialized", i.e., to be converted into a byte stream that can be written somewhere (for instance, to disk, to a socket). The byte stream can be read and transformed back into an object.

To send and receive data packets and ack packets, the given code wraps an object I/O stream around a socket, and calls `readObject()` and `writeObject()` to read and write `DataPacket` objects and `AckPacket` objects, effectively sending and receiving packet objects between the sender and the receiver.

Dropping and Corrupting Probability

The UDT layer randomly corrupts or drops packets according to the probability `P_DROP` and `P_CORRUPT` respectively. You can set these value to anything you want during testing. If you have trouble getting the code to work, it might be useful to set one or more of these variables to 0 for debugging purposes.

Detecting Closed Connection

One practical issue that needs to be dealt with for rdt 3.0 is the closing of the connection. In this assignment, the sender sends a packet with no data (0 byte) to signal that there is no more packet. The receiver closes the connection when it receives all the data correctly. This action may lead to an `EOFException` being thrown at the sender.

You may handle this exception at the sender side by gracefully closing the connection (inside the `catch` block). You need not implement a TCP-like four-way handshake (FIN/ACK) to close the connection.

Verifying The Received File

One way to check if the file received is the same as the file sent is to use cryptographic hash functions (Lecture 8).

On cs2105-z, the command `digest` can be used to compute a hash value for a file. Different algorithms can be used, including MD5 and SHA-1, two popular MAC hashing function. Here is what you can do:

1. Check the hash value of the file you want to send. Suppose the file is called `foo.zip`:

   ```
   bash$ digest -a md5 foo.zip
   6e47011c85bf4532141b726105e11c0a
   ```

   You will see a hex string printed after running the command. That is the hash of your file computing using the MD5 algorithm.

2. Repeat the command above for the received file and compared its hash to the hash computed at the sender. They must be identical if the two files are identical. The chances that the file is corrupted during transmission but still produce the same hash value is small, due to the property of cryptographic hash function.

Your cs2105-z Account

An account on the server, cs2105-z.comp.nus.edu.sg, has been setup for you. From within SoC (or through SoC-VPN), ssh to cs2105-z using your SoC UNIX id and password.

Copy the files prepared for you to your home directory, by executing:

```
cp -r ~sadm/a2 .
```

Note that you must put your files inside the directory `a2` directly under your home directory.

You will be responsible for the security of your own source code. Please be careful and set the correct permission for your files. They should not be readable by anyone else except the owner (chmod 600 *.java will ensure that).

Note that many of you will be running `FileReceiver` on the same host, and therefore must use a different port number. To prevent collision, you should avoid ”nice” port numbers such as 8000 or 8080.

Submission and Grading

There is no need to submit the program by email or IVLE workbin. We will collect your assignment from your home directory on cs2105-z.comp.nus.edu.sg when the deadline is over.

We will test your assignment automatically using a grading program. For this to work, you must not modify other java files (except `RDTSender` and `RDTReceiver`) in any signficant way (changing loss/corruption probability and adding printing debugging statement is OK). If you suspect that there is a bug in these code, please contact us by posting on the blog.

You MUST name your java program `RDTSender.java` and `RDTReceiver.java`. We will only compile these two files when we grade. You MUST not implement additional classes in other *.java files.

Using Another Platform

If you like to work on your assignment on other platforms (Windows, Mac) that you are more familiar with, you are free to do so. But when you submit your assignment, you should ensure that your program runs properly under cs2105-z.comp.nus.edu.sg and your java code is located under $HOME/a2 on cs2105-z.comp.nus.edu.sg.

Plagiarism Warning

You are free to discuss the assignment with your peers. But, ultimately, you should write your own code. We employ zero-tolerance policy against plagiarism. If you are caught copying from other student, or let others copy your code, you will receive zero for this assignment. Further disciplinary action may be taken by the school.

Grading

- 3 marks: Correctly handle corrupted packets.
- 3 marks: Correctly handle duplicate packets.
- 4 marks: Correctly handle lost packets.

We will deduct one marks for every failure to following instructions (wrong directory name, wrong filename, etc.)

THE END