Progamming Assignment 3: Reliable UDP

Submission

- **Due**: Nov 6 2013, 23:59pm
- **Late Penalty**: 10% per day
- **Submission Format**: Create a folder P3 under your home directory on cs2105-z.comp.nus.edu.sg, and upload your Java source code to this folder. (Submission time is your last modification time.) Within the folder P3, the compile command for your source code should be javac *.java, and the command to run your source code should be "java Sender portNum1 portNum2 inputFilePath outputFileName" and "java Receiver portNum3 portNum4 outputFolderPath". (All without quotation marks.)
- If you need to clarify anything regarding this assignment please only write emails to Luo Chengwen (chluo@comp.nus.edu.sg). You are also encouraged to post questions regarding the programming assignment in the IVLE forum.

**** Important ****: Please strictly follow the submission rules since the grading will be done automatically by a grading script, your submission may not be graded if the format is not correct. If your are using IDEs such as Eclipse, please try to compile using command line before you submit to make sure that your programs can be compiled successfully on the server. Finally, test your program on cs2105-z.comp.nus.edu.sg.

1 Overview

In this assignment, you are required to write a sender program and a receiver program which can reliably communicate with each other over UDP. As mentioned in class, UDP itself is unreliable. Therefore, your task is to design and implement an application level protocol which can achieve reliable file transfer on top of UDP.

2 Unreliable Network Simulator

Internet itself is not reliable, packets may suffer from lost, delay, re-ordering and corruption through Internet. To simulate these uncertainties in this assignment,

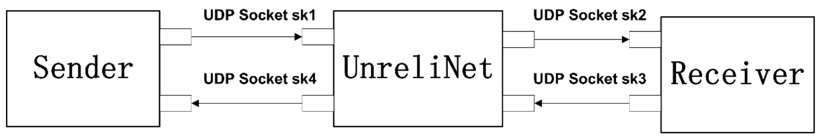

Figure 1: One-to-one Reliable File Transfer

a simple simulation tool *UnreliNet* is used. UnreliNet is an independent program running between sender program and receiver program to introduce disruptions for the communication. It takes all sender or receiver's packet and forwards them to the other party. Packets passing through the UnreliNet may be dropped, delayed, reordered or corrupted by certain probability. The challenge of this assignment is to design a reliable protocol over UDP for the sender and receiver so that files can be successfully transferred regardless of the existence of these uncertainties.

3 Basic Task: One-to-one File Transfer (140 Points)

3.1 Task Description

In this task, you will need to implement a sender program and a receiver program and design an application-layer protocol for reliable file transfer on top of UDP. As shown in Figure 1, three entities (Sender, UnreliNet and Receiver) and four UDP sockets are involved (sk1, sk2, sk3 and sk4). Each packet between Sender and Receiver passes through the UnreliNet, so that the UnreliNet can drop, corrupt, reorder and delay the packet with certain probability, or simply forwards your packet to the other side. You goal here is to ensure that your file can be successfully transferred between Sender and Receiver using your protocol. The file can be either txt file or binary file. Recall that you have learned some reliable protocols in the class, i.e., Stop-and-wait (SW), Go-Back-N (GBN), Selective Repeat (SR), etc. You can implement any one of them or design your own protocol, but you will need to ensure that your protocol is efficient enough since efficiency is also one of the grading metric in the assignment (As will be discussed later).

Before sending any data, four UDP sockets should be established. The 1st one is from the sender program to UnreliNet, which we call sk1. The 2nd one is from UnreliNet to the receiver program, which we call sk2. The 3rd one is from the receiver program to UnreliNet, which we call sk3. The last one is from UnreliNet to the sender program, which we call sk4. sk1 and sk2 are used to transfer data packets and sk3 and sk4 are mainly for acknowledgements. You can then send any data you want to send through sk1 (e.g., real data plus checksum). UnreliNet may corrupt, drop or delay your data arbitrarily, and then send the remaining (possibly corrupted and out-of-order) data to receiver program through sk2. You may want to send ACK or NAK from receiver program to sender program. You can achieve this with sk3 and sk4. However, again, your ACK or NAK may suffer corruption, delay or drop when it passes through UnreliNet.

3.2 UnreliNet Program

The UnreliNet program is used to test your program and you don't need to implement it. As mentioned above, the UnreliNet will perform four actions on the packet: *drop, corrupt, delay* and *reorder.* To be more realistic, UnreliNet will simulate network error on a per packet basis. This implies if UnreliNet decides to drop the 2nd packet, then it will drop all data carried by this packet. This rule also applies to the packet delay scenario. We also note here that UnreliNet will delay each of your packet for at most 500ms. Note, however, that UnreliNet may only corrupt part of a packet. To handle packet corruption, you may need to add checksum to your data to ensure that the correct file is received.

UnreliNet takes three parameters, the 1st one is the port number of sk1, the 2nd one is the port number of sk3, the 3rd one is the destination port of sk4. However, to make the system more extensible to multiple sender scenario later, the UnreliNet will not be aware of the Sender's receiving port number (port number of sk4) when starting. Sender is responsible to register at the UnreliNet in order to receive the acknowledgements. If the sk4's port number is 10000, the sender then will need to send a packet containing only the following string to the UnreliNet: *"REG:10000"*. To avoid ambiguity we assume that there will be no such strings in the test files and we assume this packet can always be successfully sent to the UnreliNet. The UnreliNet will parse the string received and add this sender to the system. After registration, the UnreliNet will be able to forward the data received from sk3 back to sk4. The registration is already done in the sample program provided for you. To start the UnreliNet, use the following command:

>$ java UnreliNet sk1_port sk2_port sk3_port

3.3 Receiver Program

Your receiver program should take three parameters. The 1st one is the destination port of sk2, and the 2nd one is the destination port of sk3. The third one is the output folder path. The receiver program should write the file that is transferred from the sender program to the folder path. You program is considered to work correctly if and only if the content of the input file and output file are identical. On the server, you can use the command "cmp" to compare the content of two different files. In particular, if you type "cmp input.txt output.txt", and the contents of these two files are identical, then nothing will be outputted. Otherwise, cmp will tell you the byte and line numbers at which the first difference occurred. To start the receiver program:

>$ java Receiver sk2_port sk3_port outputFolderPath

3.4 Sender

Your sender program should have three parameters. The 1st one is the port number of sk1, and the 2nd one is the destination port of sk4. The sender program should read the file specified by the third parameter. The file can be either txt file or binary file, and the file size can be large. To start the sender:

>$ java Sender sk1_port sk4_port inputFilePath outputFileName

where outputFileName is the name of the file to be saved at the receiver side. FileName is relative to the output directory. You should encode the file name in you protocol so that the receiver knows what name to use.

3.5 Other Requirements

If you are using a sliding window protocol (e.g., go-back-n or selective-repeat), make sure the windows size is at most 10, and each packet should contain at most 1000 bytes data. We impose a restriction on maximum window size because we do not want your program to (blindly) send out too much data within a short period of time and exhaust all network bandwidth.

Secondly, your sender program and receiver program should correctly exit when the transmission is finished.

Thirdly, your sender program and receiver program should not ask for user input once it starts execution. That is, you only need to type the start commands. Once your programs starts execution, they should finish without the need for further user input.

4 Bonus Task: Multi-to-One File Transfer (20 points)

In most realistic settings there are more than one senders. In this task, multiple senders are sending different files simultaneously. The only receiver needs to handle all concurrent inputs and decide which output file to write to and which sender to acknowledge. Your sender and receiver programs should be able to handle different transfer sessions. (e.g., one simple way may be to tag each packet for each file session, akin to adding a port number). As shown in Figure 2, each sender uses different socket to receive packets from the UnreliNet, therefore each sender needs to register at the UnreliNet first in order to receive the acknowledgements from receiver.(Since senders can dynamically join in the system and the UnreliNet does not know the existence of these senders beforehand.) Once UnreliNet has one packet to send back to senders, it will broadcast to all registered senders. Therefore your sender should be able to tell whether the received packet belong to it or not.

In the testing, the receiver program will only quit after all file transfers are completed. After all senders finishes their file transfer, all files should have been

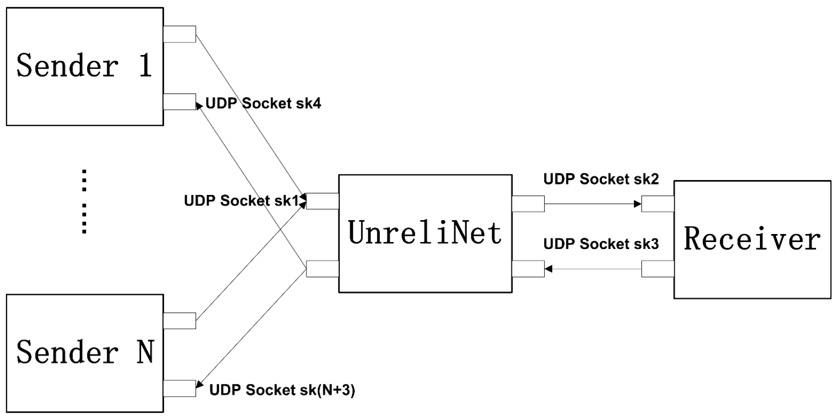

Figure 2: Multi-to-one Reliable File Transfer

saved correctly inside the destination folder by the receiver. In the testing you can assume that output file names used are different so that there will be no name conflicts when writing files.

5 Testing Your Program

Along with this assignment, we have provided you simple version of UnreliNet programs. We also provide a dummy sender which just sends a packet every some time, and a dummy receiver which just echos back whatever data it receives. These two programs should give you some idea on what a UDP sender/receiver looks like. Note, they are just simple demo programs, we do not implement any file I/O or reliable transmission functionality. It is your job to modify the interfaces and add any other necessary functionalities following the instructions.

6 Advices on This Assignment

You should first start the receiver program, in which you will listen to the destination port of sk2 and create sk3. Then, you should start UnreliNet. UnreliNet will listen to the destination port of sk1 and sk3; and create s2. Finally, you should start sender program, in which you will register at the UnreliNet and listens to the destination port of sk4 and create sk1. By now, you can start sending data. You should always finish the basic task first before proceeding with the bonus task.

7 Grading

You should strictly follow the instructions provided in this assignment to make sure that your submission can be successfully graded by the script. Make sure that it can compile on the server and also your program interfaces follow this assignment descriptions. In the test, test files are randomly chosen with different file sizes and can be either txt file or binary files. Your submission will be graded out of 100 points:

(1) Task 1 (140 points)

- (10 points) Following the submission instructions, programs can be compiled and executed without error on the server.
- (10 points) Your program can successfully transfer (exactly same file received on the receiver folder) test files when network is normal (i.e., when we test your program with the UnreliNETNormal network simulator).
- (20 points) Successfully transfer test files when the simulator starts dropping packets (i.e., when UnreliNETDrop simulator is used).
- (20 points) Successfully transfer test files with UnreliNETDelay.
- (20 points) Successfully transfer test files with UnreliNETReorder.
- (20 points) Successfully transfer test files with UnreliNETCorrupt.
- (20 points) Time Efficiency. You will get 4, 8, 12, 16, 20 points if your total execution time in the evaluation is faster than 0%, 20%, 40%, 60%, 80% of all submissions respectively. Therefore, you are encouraged to implement more efficient reliable protocols.
- (20 points) Hidden test case with combinations of loss/delay/error/reordering behavior.

(2) Bonus Task (20 points)

- (20 points) Successfully implement multiple files transfer simultaneously.

Note: Your final marks cannot exceed 140 marks, i.e., if you get 160 marks, you final mark will still be 140.

THIS IS THE END OF THE ASSIGNMENT