National University of Singapore
School of Computing

CS2105 **Assignment 2** Semester 2 AY16/17

Learning Objective

In this assignment, we will implement the Go-Back-N protocol over an unreliable communication channel. After solving this assignment you should know the Go-Back-N protocol by heart.

Introduction

Alice wants to send a file to Bob. Unfortunately, in Alice's network, all TCP traffic is blocked by a firewall, i.e. she can neither send nor receive any data using TCP. After some exploration, Alice realizes that the firewall does not block UDP traffic. Therefore she decides to send the file over UDP but wants to ensure Bob can receive the file intact.

Help Alice by implementing a reliable transport on top of UDP. Alice should be able to send a file successfully to Bob using UDP communication only. To achieve reliable transport you should implement Go-Back-N.

No Group Work

This assignment should be solved **individually**. **Group work is not permitted**.

Submission Deadline

20th March 2017 (Monday), 6pm sharp. Please start early and submit early. You can always upload a new version before the deadline. We strongly advice against submitting only 2 minutes before the deadline. **2 points penalty** will be imposed on late submission (Late submission refers to submission or re-submission after the deadline, even if it is only 5 minutes).

This programming assignment is worth **9** marks.

Program Submission

1. Make sure each and every Java file contains your student number and name at the top.
2. Make sure the classes have the correct name and match the file name (e.g. Alice.java should contain a class called Alice but not alice).
3. Ensure that the programs run correctly on sunfire using the given commands.
4. Please submit your files to `CodeCrunch` website: https://codecrunch.comp.nus.edu.sg. You should submit at least **Alice.java** and **Bob.java**. However, feel free to upload more Java files, e.g. in case you created extra Java classes.
5. You can submit multiple Java files to `CodeCrunch` by pressing the <Ctrl> key when choosing programs to upload. Thus, you can upload all your files directly. Do not compress or archive your files, i.e. **do not use zip, rar, tar or similar**.
6. Please note that **1 mark penalty** will be imposed on all submissions that do not follow the submission format.
7. Note that `CodeCrunch` is just used for program submission and no test case has been mounted on it. Hence you may ignore the feedback from `CodeCrunch` regarding the quality of your programs.

You are not allowed to post your solutions in any public domain in the Internet.

Plagiarism Warning

You are free to discuss this assignment with your friends. However, you should **design and write** your own code. We employ a zero-tolerance policy against plagiarism. Confirmed breach will result in zero mark for the assignment and further disciplinary action from the school. **Do not copy code from the Internet.**

Grading Rubric

1. **[2 points]** You followed the submission format, e.g. you submit at least Alice.java and Bob.java. Your program compiles without errors. You do not submit zips or rars or similar archives.
2. **[2 points]** Programs can successfully transfer a file from Alice to Bob in a perfectly reliable channel (i.e. no error at all). The received file is **binary equivalent** to the sent file. (Use cmp or diff to test binary equivalence)
3. **[2 points]** Programs can successfully transfer a file from Alice to Bob in the presence of both data packet corruption and ACK packet corruption using Go-Back-N.
4. **[2 points]** Programs can successfully transfer a file from Alice to Bob in the presence of both data packet loss and ACK packet loss using Go-Back-N.
5. **[1 point]** Your implementation works correctly for different sender windows (different Ns in Go-Back-N). N is passed to the program as a command-line argument.

During grading we do not care what messages you print on the screen. However, we do care about binary equivalence, i.e. the received file must the exactly the same as the sent one. Please use the `cmp` command to test for binary equivalence. Refer also to assignment 0 for details.

If you **struggle** to solve this assignment, please post your questions in IVLE. Focus on the low hanging fruits first, before you try the harder ones. If you still have issues after a while, please approach the teaching staff.

Overall Architecture

There are three programs in this assignment, **Alice**, **Bob**, and **UnreliNET**. Their relationship is illustrated in Figure 1 below. **Alice** and **Bob** implement a one-way file transfer application over UDP protocol. The **UnreliNET** program simulates the transmission channel that randomly corrupts or loses packets. However, for simplicity, you can assume that this channel always delivers packets in order.

Figure 1: UnreliNet Simulates Unreliable Network

The **UnreliNET** program acts as a proxy between the **Alice** and **Bob**. **UnreliNET** may introduce bit errors to packets or lose packets randomly. It then forwards packets (if not lost) to the destination.

The **UnreliNET** program is complete and given. Your task in this assignment is to develop the **Alice** and **Bob** programs so that both sides will receive chat messages successfully in the presence of packet corruption and packet loss.

UnreliNET Class

The **UnreliNET** program simulates an unreliable channel that may corrupt or lose packets with a certain probability. This program is given and should not be changed.

To run **UnreliNET** on **sunfire**, type then following command:

```
java UnreliNET <P_DATA_CORRUPT> <P_ACK_CORRUPT>
<P_DATA_LOSS> <P_ACK_LOSS> <MIN_PROPAGATION_DELAY>
<MAX_PROPAGATION_DELAY> <UnreliNetPort> <RcvHost> <RcvPort>
```

For example:

```
java UnreliNET 0.3 0.2 0.1 0.05 50 150 9000 localhost 9001
```

listens on port 9000 and forwards all received data packets to **Bob** running on the same host at port 9001. Packets send from **Alice** to **Bob** have a 30% chance of getting corrupted and 10% chance of being lost. The **UnreliNET** program also forwards ACK packets back to **Alice.** Acknowledgements send from **Bob** to **Alice** have a 20% chance of being corrupted and a 5% chance of being lost. Furthermore, the propagation delay is between 50 and 150 ms.

Packet Error Rate

The **UnreliNET** program randomly corrupts or loses data packets and ACK packets according to the specified parameters P_DATA_CORRUPT, P_DATA_LOSS, P_ACK_CORRUPT, and P_ACK_LOSS. You can set these values to anything in the range [0, 0.3] during testing (setting a too large corruption/loss rate may result in a very slow transmission).

If you have trouble getting your code to work, it might be advisable to set them to 0 first for debugging purposes.

Alice Class

The **Alice** program is a simple one-way file transfer program. It opens a given file and sends its content as a sequence of packets to **UnreliNet**. **UnreliNet** then forwards the packets to **Bob**.

To run **Alice** on **sunfire**, type command:

```
java Alice <N> <path/filename> <unreliNetPort> <rcvFileName>
```

For example:

```
java Alice 10 ../test/cs2105.zip 9000 notes.zip
```

sends the file **cs2105.zip** from directory (relative path) **../test** to **UnreliNet** running in the same host at port 9000. **UnreliNet** will then forward the file to **Bob** to be stored as **notes.zip**. The sender window size (the N in Go-Back-N) is 10.

Notes:

1. For this assignment, we will always run **UnreliNET**, **Alice** and **Bob** programs on the same host.
2. **Alice** should terminate after (1) reading all data from the file and (2) forwarding it successfully to the **Bob**.
3. Use a **Maximum Segment Size of 512 bytes** or your packets may be rejected by the **UnreliNET** program.
4. Do not hard code the target file name “notes.zip” in the Bob program. During testing we will use different file names.
5. Again, be very careful about binary equivalence.
6. During grading the supplied file will exist, the path is correct, and the file name is smaller than 256 bytes.

Bob Class

The **Bob** program receives packets from **Alice** (through **UnreliNET**) and store the received file in the current working directory, with a filename specified by **Alice**.

To run **Bob** on **sunfire**, type command:

```
java Bob <rcvPort>
```

For example,

```
java Bob 9001
```

listens on port 9001 and stores the bytes received into a file whose name is given by **Alice**. **Bob** does not have to terminate, i.e. it can run infinitely and does not need to detect end of transmission and terminate. (This simplifies the implementation. Otherwise you would have to implement a full TCP teardown which is quite hard) You can assume that after **Alice** is terminated, **Bob** will not be reused for another communication attempt.

However, even though Bob does not have to terminate, Bob has to make sure that all the received data is stored on disk. A good way to achieve this is to flush any buffers and close the file properly once the last data is successfully received.

Running All Three Programs

You should first launch **Bob**, followed by **UnreliNET** in the second window. Finally, launch **Alice** in a third window to start data transmission. Please note that Alice and Bob take the ports (**<unreliNetPort>** and **<rcvPort>**, respectively) as command-line argument as described above. Please always test your programs on **localhost** to avoid the interference of network traffic on your programs.

The **UnreliNET** program simulates unreliable communication network and runs infinitely. Once launched, you may reuse it in consecutive tests. To manually terminate it, press <Ctrl> + c.

The Alice and Bob programs should not communicate with each other directly – all traffic has to go through the **UnreliNET** program

Tips and Hints

Reading Data from File

Please refer to Assignment 0 Exercise 3 on how to read data in batch. You should not read the entire file into an array as **`java.lang.OutOfMemoryError`** may be thrown when a huge file is used in testing. Especially, do not use the code that was provided in the skeleton of assignment 1.

Self-defined Header/Trailer Fields at Application Layer

UDP transmission is unreliable. To detect packet corruption or packet loss, you may need to implement reliability checking and recovery mechanisms at application layer. The following header/trailer fields are suggested though you may have your own design:

- Sequence number
- Checksum

Note that each packet **`Alice`** sends should contain **at most 512 bytes** of application data (inclusive of user-defined header/trailer fields), or **`UnreliNET`** will reject it.

Computing Checksum

To detect bit errors, **`Alice`** should compute checksum for every outgoing packet and embed it in the packet. **`Bob`** needs to re-compute checksum to verify the integrity of a received packet.

Please refer to Assignment 0 Exercise 2 on how to compute checksum using Java **`CRC32`** class.

Timer and Timeout Value

Alice may have to maintain a timer for unacknowledged packet. You may want to use the **`setSoTimeout()`** method of Java **`Socket`** class.

You should use a **timeout value of 2000ms**.

Reading/Writing Values to Header/Trailer Fields

The number of application layer header/trailer fields and the sequence of their appearance in a packet have to be agreed by sender and receiver before data transmission (i.e. an application layer protocol designed by you).

As will be discussed in the week 8 tutorial, you may use **`ByteBuffer`** class from the **`java.nio`** package to form a packet containing various header/trailer and application

message.

Common Pitfalls

The received file has to be binary equivalent to the sent file. A good way to test this is to use the **cmp** command (please refer to assignment 0 for more details). A bad way to test is to compare file sizes (especially if the file sizes are in KB). Another bad way is to transfer a file and then test if it can be opened (many applications have built-in error correction).

A Word of Advice

This assignment is complex and time-consuming. You are suggested to write programs incrementally and modularly. For example, deal with error-free transmission first, then data packet corruption, ACK packet corruption, etc. Test your programs after every single major change. Take note that partial credit will be awarded even if your programs do not meet all listed requirements.

Question & Answer

If you have any doubts on this assignment, please post your questions on IVLE forum or consult the teaching team. We will NOT debug programs for you. However, we may help to clarify misconceptions or give necessary directions if applicable.