National University of Singapore
School of Computing

CS2105 **Assignment 2** Semester 2 AY18/19

Introduction and Submission Deadline

In this assignment, you will implement a reliable one-way chat program[1] that sends messages over the an unreliable UDP channel that may either corrupt or drop packets randomly (but will always deliver packets in order). This programming assignment is **worth 7 points.**

The deadline of submission is **22 April 2019 (Monday), 6pm** sharp. Do not leave your submission to the last minute in case of unforeseeable situation. You may submit many times and only the last submission will be graded.

2 points penalty will be imposed on late submissions (i.e. submissions or re-submissions made after the deadline). **No submission will be accepted after 28 April 2019 and 0 point will be awarded.**

Group Work

This assignment should be solved individually. However, if you struggle, feel free to form a team with another student (max. team size is 2). Group submissions are subject to **2 points penalty** for each student.

Under no circumstances should you solve it in a group and then submit it as an individual solution. This is considered plagiarism. Group work needs special care during submission (see below).

Writing Your Programs

You are free to write your programs using any editor/IDE. However, you can only write your program in Python 3, C/C++, or Java.

You are responsible to ensure that your programs compile and run properly on **sunfire** because **we will only test and grade your programs on sunfire.** The screenshot on the next page shows respective language/compiler version on sunfire.

[1] We implement a one-way chat to keep the assignment simple.

```
cs2105@sunfire0.comp.nus.edu.sg ~> python --version
Python 3.7.0
cs2105@sunfire0.comp.nus.edu.sg ~> java --version
java 9.0.4
Java(TM) SE Runtime Environment (build 9.0.4+11)
Java HotSpot(TM) 64-Bit Server VM (build 9.0.4+11, mixed mode)
cs2105@sunfire0.comp.nus.edu.sg ~> gcc --version
gcc (GCC) 4.8.4
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

cs2105@sunfire0.comp.nus.edu.sg ~> g++ --version
g++ (GCC) 4.8.4
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Program Submission

For this assignment, you are required to submit one zip file that contains source programs for the sender and the receiver. Please name your zip file as **<Student number>.zip**, where **<Student number>** refers to your matric number which starts with letter A. Note that the first and last letters of your student number should be capital letters. One example file name would be **A0112345X.zip**. Your zip file should only contain two source programs, **Alice.py** and **Bob.py**. If you use Java, C, or C++, please replace the ".py" extension name with ".java", ".c", or ".cpp", respectively. Submit this zip file to IVLE folder "Assignment2_submission". **2 points** will be deducted if you fail to follow the submission format, including wrong file naming format, folder structure, file names or extensions (for example, .rar is not acceptable).

In case of group work, please designate one person to submit the zip file, instead of submitting the same file twice by two different persons. The file name should contain the matric numbers of two members and be named as **<Student number 1>-<Student number 2>.zip**. An example would be **A0165432X-A0123456Y.zip**.

We strongly recommend that you use Python 3 for the assignment. If you use Java or C/C++, we will compile and run your program for grading, using the default compilers on sunfire (g++ 4.8.4 or java 9.0.4). For C/C++ users, we will link your program with zlib during compilation.

The grading script will infer the programming language from your file extension (e.g. .py, .java, etc). For a Java program, the class name should be consistent with the source file name, and please implement the **public static main()** method so that the program can be executed as a standalone process after compilation. Do not put your class under any package paths.

Grading

We will grade your programs based on their correctness only. A grading script will be used to test your programs and no manual grading will be provided.

- **[2 points]** Programs can be successfully compiled on **sunfire** without any error. Program execution follows the specified commands exactly (see sections below). Source files are correctly named and there are no additional subfolders inside the zip file.
- **[1 point]** Programs can successfully send chat messages from Alice to Bob in a perfectly reliable channel (i.e. no error at all).
- **[2 points]** Programs can successfully send messages from Alice to Bob in the presence of packet corruptions.
- **[2 points]** Programs can successfully send messages from Alice to Bob in the presence of both packet corruptions and packet losses.

The chat messages received by Bob must be **identical** to the ones sent by Alice, to claim a successful transmission. The total size of input messages to Alice is guaranteed to be no larger than **5,000** bytes (including newline characters). During grading, Alice and Bob processes will be forced to terminate **60 seconds** after Alice is started. The grading script will then compare the output of Bob against the input to Alice. Please ensure your programs are efficient enough to meet the above criteria.

CAUSION: The grading script **grades your program according to what Bob prints out on the screen**. Thus, make sure that you **remove all debug output** before you submit your solution. In each test case, no points will be awarded if your output does not conform with the expected output.

A Word of Advice

This assignment can be time-consuming. We suggest that you start writing programs early, incrementally and modularly. For example, deal with error-free transmission first, then data packet corruption, ACK packet corruption, and etc. Test your programs frequently and make backup copies before every major change. Frequent backups will allow you to submit at least a partially working solution that is worth some marks.

Do not post your solution in any public domain on the Internet or share it with friends, even after this semester.

Overall Architecture

There are three programs involved in this assignment, **Alice**, **UnreliNET** and **Bob** (see Figure 1 below). **Alice** will send chat messages to **Bob** over UDP (and Bob may provide feedback as necessary). The **UnreliNET** program is used to simulate the unreliable transmission channel (in both directions) that randomly corrupt or drop packets. For simplicity, you can assume that **UnreliNET** always delivers packets in order.

Figure 1: UnreliNet Simulates Unreliable Network

Instead of sending packets directly to **Bob**, **Alice** will send all packets to **UnreliNET**. **UnreliNET** may introduce bit errors or lose packets randomly. It then forwards packets (if not lost) to **Bob**. When receiving feedback packets from **Bob**, **UnreliNET** may also corrupt them or lose them with certain probability before relaying them to **Alice**.

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

UnreliNET Program

The **UnreliNET** program simulates an unreliable channel that may corrupt or lose packets with tunable probability. This program is complete and provided as a Java class file. There's no necessity to view and understand the source code of **UnreliNET** to complete this assignment.

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

```
java UnreliNET <P_DATA_CORRUPT> <P_DATA_LOSS>
<P_ACK_CORRUPT> <P_ACK_LOSS> <unreliNetPort> <rcvPort>
```

For example:

```
java UnreliNET 0.3 0.2 0.1 0.05 9000 9001
```

The above command makes **UnreliNET** listens on port 9000 of localhost and forwards all received data packets to **Bob** running on the same host port 9001, with 30% chance of packet corruption and 20% chance of packet loss. The **UnreliNET** program also forwards ACK/NAK packets back to **Alice**, with a 10% packet corruption rate and a 5% packet loss rate.

Note that **UnreliNET** only accepts packets with a maximum payload of **64** bytes (**inclusive of user-defined header/trailer fields**). It will drop any packet beyond this size limit and show an error message on the screen.

Packet Error Rate

The **UnreliNET** program randomly corrupts or drops data packets and ACK/NAK packets according to the specified parameters P_DATA_CORRUPT, P_DATA_LOSS, P_ACK_CORRUPT, and P_ACK_LOSS. Please choose these values in the range [0, 0.3] during testing (setting a too large corruption/loss rate may result in a very slow transmission). The grading script also chooses parameters in the range [0, 0.3].

If you have trouble getting your code to work, better set the parameters to 0 first for debugging purposes.

Alice Program

Alice will read chat messages from standard input and sends them to **UnreliNET** (which will then forward to Bob as appropriate). The chat messages contain ASCII characters only and there is no empty message (i.e. blank line).

To run **Alice** on **sunfire**, type the following command (suppose **python3** alias is set up):

```
python3 Alice.py <unreliNetPort>
```

<unreliNetPort> is the port number **UnreliNET** is listening to. An example command line would be,

```
python3 Alice.py 9000
```

Bob Program

Bob receives chat messages from **Alice** (via **UnreliNET**) and prints them to standard output. It may also send feedback packets to Alice (also via **UnreliNET**) as you deem appropriate. The command to run Python version of Bob is:

```
python3 Bob.py <rcvPort>
```

For example,

```
python3 Bob.py 9001
```

With above command, Bob listens to port 9001 of localhost and prints all received messages to the standard output.

Some tips are given below.

1. Bob should only print messages that are correctly received from Alice, no more, no less.
2. Make sure you do not print anything irrelevant to standard output. In particular, remember to **remove all debug output before submission**.

Running All Three Programs

For this assignment, you will always run **`UnreliNET`**, **`Alice`** and **`Bob`** programs on the same host. You should launch **`Bob`** first, followed by **`UnreliNET`** in the second window. Finally, launch **`Alice`** in a third window to start data transmission.

The **`UnreliNET`** program simulates a unreliable network and runs infinitely. Once launched, you may reuse it in consecutive tests if you do not want to change its parameters. 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. **`Alice`** should terminate once all input is read from user and properly forwarded (i.e. the input stream is closed and everything in the input stream is successfully received by **`Bob`**).

There is no need for Bob to detect end of transmission and terminate. If you need to manually terminate it, press <Ctrl> + c.

Testing your programs

For the convenience of testing, you can use the file redirection feature to let **Alice** read from a file (rather than from keyboard input).

For example, you can run **Alice** as follows:

```
python3 Alice.py 9000 < input.txt
```

You can replace input.txt with another file name if you wish.

In the same way, you can let **Bob** prints to a file (rather than print on screen - just for the convenience of testing):

```
python3 Bob.py 9001 > output.txt
```

Remember that both **Alice** and **Bob** should take port number as command-line argument. Also remember to flush output buffer so that data will be written to **output.txt**.

Finally, you can compare input.txt to output.txt by issuing the following command:

```
cmp input.txt output.txt
```

This line compares two files byte by byte and prints the differences if found. If the two files are binary equivalent, nothing will be printed on the screen. This method is useful to detect unexpected outputs such as hidden characters.

A sample input.txt is provided for your testing.

Self-defined Header/Trailer Fields at Application Layer

UDP transmission is unreliable. To detect packet corruption or packet loss, you need to implement reliability checking and recovery mechanisms by yourself. The following header/trailer fields are recommended though you may choose a different design:

- Sequence number
- Checksum

You are reminded again that each packet **Alice** or **Bob** sends should contain **at most 64 bytes** of payload data (**inclusive of user-defined header/trailer fields**), or **`UnreliNET`** will drop it.

Computing Checksum

To detect bit errors, **`Alice`** should compute checksum for every outgoing packet. Please refer to Assignment 0 exercise 2 on how to compute checksum.

Timer and Timeout Value

Alice may have to maintain a timer for unacknowledged packet. You should set a timeout value of **50ms**. Socket timeout can be set using Python function socket.settimeout(). Other languages provide similar feature which can be found online.

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 required.

Plagiarism Warning

You are free to discuss this assignment with your friends. However, ultimately, you should write your own code. We employ zero-tolerance policy against plagiarism. If a suspicious case is found, we will award zero points and may take further disciplinary action.