Introduction

In this assignment, you will implement a reliable one-way chat program[1] that sends messages over an unreliable UDP channel that mav either corrupt or drop packets randomly (but will always deliver packets in order).

Grading

We will test and grade your program on the `sunfire` server. Please make sure that your programs run properly on `sunfire`. Moreover, you are allowed to use libraries installed in public folders of `sunfire` (e.g. `/usr/lib`) only.

We accept submission of Python 3 (**3.7**), Java, or C/C++ program, and we recommend that you use **Python 3** for your assignments. Programming languages other than Python 3, Java, and C/C++ are not allowed. For Python 3, we use the `python3` program installed in folder `/usr/local/Python-3.7/bin` on `sunfire` for grading. If you use Java or C/C++, we will compile and run your program for grading using the default compilers on `sunfire` (`java 9.0.4` installed in `/usr/local/java/jdk/bin`, or `gcc 4.8.4` installed in `/usr/local/gcc-4.8/bin`). For C/C++ users, we will link your program with zlib during compilation. The grading script infers your programming language

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

from the file extension name (.py, .java, .c). Therefore, please ensure your files have the correct extension names. For a Java program, the class name should be consistent with the source file name, and please implement the static `main()` method so that the program can be executed as a standalone process after compilation.

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

Grading Rubric

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. The grading script infers your programming language from the file extension name (.py, .java, .c). Therefore, please ensure your files have the correct extension names. 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.

1. **[2 points]** Programs are free from syntax errors and can be successfully executed (or compiled, for Java/C/C++). Program execution follows the specified commands exactly. Source files are correctly named and there are no additional subfolders inside the zip file.
2. **[1 points]** Programs can successfully send chat messages from Alice to Bob in a perfectly reliable channel (i.e. no error at all).
3. **[2 points]** Programs can successfully send messages from Alice to Bob in the presence of **packet corruptions**.
4. **[2 points]** Programs can successfully send messages from Alice to Bob in the presence of **packet losses**.
5. **[1 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 **10 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.

CAUTION: 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.**

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.

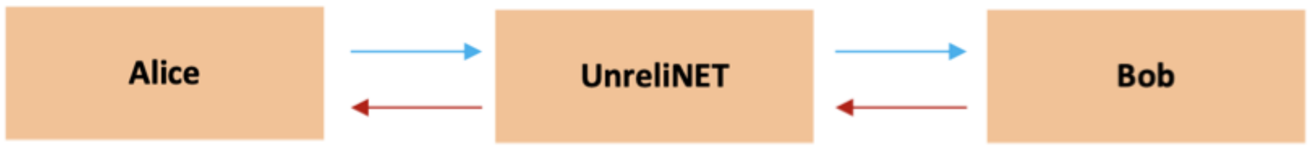

Figure 1: UnreliNET Simulates Unreliable Network

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` listen on port 9000 of localhost and forwards all received data packets to `Bob` running on the same host with 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 send 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). There's no delay regarding the input to Alice.

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**.
3. You may wish to use standard error (stderr in Python) for debugging/logging purposes.

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 an 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. Do terminate any unwanted running instance before starting next run, or exiting Sunfire.

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

To test your program, please use your SoC UNIX ID and password to log on to `sunfire` as instructed in Assignment 0 paper. To make the `python3` alias permanent, and thus avoid typing the `alias` command every time you log in, you can run the following command once to store the shortcut into the configuration file:

```
echo alias python3=/usr/local/Python-3.7/bin/python3 >> ~/.bash_profile
```

If you test your server manually, please select a port number greater than 1024, because the OS usually restricts usage of ports lower than 1024. If you get a `BindException: Address already in use` (or similar errors for other languages), please try a different port number.

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 <unreliNetPort> < input.txt
```

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

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

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

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.

We also release a comprehensive set of grading scripts to you. There are no hidden test cases during grading. However, passing all the test cases does not guarantee that you will get full marks. We will detect fraudulent cases such as hard-coding answers, or Alice communicates with Bob in any means other than via UnreliNET.

To use the grading script, please upload your programs along with the test folder given in the package to `sunfire`. Make sure that your programs and the test folder are in the same directory. Then, you can run the following command to test your programs:

```
bash test/RunTest.sh
```

By default, the script runs through all test cases. You can also choose to run a certain test case by specifying the case number in the command:

```
bash test/RunTest.sh 2
```

To stop a test, press <Ctrl> + c. If pressing the key combination once does not work, hold the keys until the script exits.

Note: If you receive "Failed" feedback indicating wrong output especially smaller output size, it can be due to timeout. In this case, you might also see some program exception followed by "KeyboardInterrupt" before the "Failed" line. This might be because that your programs are forced to be terminated by the grading program due to timeout.

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:

1. Sequence number
2. 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 socket timeout value of **50ms**. Socket timeout can be set using Python function `socket.settimeout()`. Other languages provide similar feature which can be found online.