Lab 4: Reliable Transport (10 points)

1 Download Lab

Download the Lab 4 files from Brightspace. The source code will be inside the directory "`Lab4/`". You must use the environment from Lab 0 to run and test your code. Next, open a terminal and "`cd`" into the "`Lab4/`" directory. Now you are ready to run the lab!

2 Simulated Network Configuration

For this lab, you will work with a simplified version of the network simulator used in Lab 3. The simulated network's configuration is specified in the file "`01.json`". The network comprises two clients A and B connected using the router "1" (Figure 1). Client A sends a file to client B via the router. The latency of each link is 1 second. The Maximum Segment Size (MSS) is set to 256 bytes, i.e., you cannot send a packet with payload larger than 256 bytes. The router can **drop packets (both data and acknowledgment packets)** randomly based on a loss probability parameter. Assuming a loss probability value of p, on receiving a packet, the router will drop that packet with probability p. The router will **never re-order the received packets**.

We will use the same network configuration as above throughout this lab and for the evaluation.

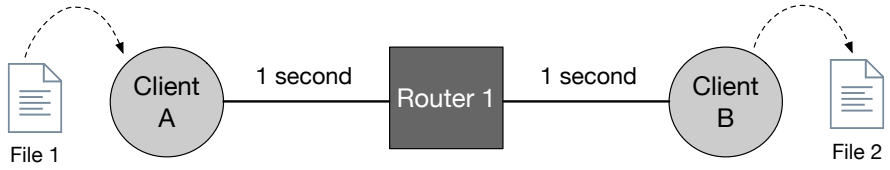

Figure 1: Simulated network topology. Client A sends the contents of File 1 to client B via router 1. Client B writes the received bytes into File 2.

3 Running the Code

You must run and test your code on `eceprog` using the environment from Lab 0. If your code does not run in that environment, you will not get any credit! Start the simulator using the command:

```
$ python3 network.py 01.json [send file path] [recv file path] [loss probability]
```

The "`01.json`" file argument specifies the configuration of the simulated network, as described in the above section. The "`send file path`" and "`recv file path`" arguments specify respectively the file to be sent over the network and the file to which the received bytes are to be written at the receiver. The "`loss probability`" argument specifies the probability of dropping a received packet at the router. **The loss probability value (p) must be an integer in the range $0 \leq p \leq 99$**.

4 Task

The task is to send a ***single*** ASCII text file ***reliably*** from client A (sender) to client B (receiver) via the router. The received file must be identical to the sent file, even under packet loss at the router. **Compressing the file/packet contents is <u>not</u> allowed. Multithreading is also <u>not</u> allowed**.

The provided code in the "`myClient.py`" file already implements the file transfer logic **assuming no packet loss.** The code first sets up a connection using `SYN, SYN-ACK, ACK` packets. After the connection is established, client *A* reads and sends the data from "`sendFile`" to the router by creating packets using the "`Packet`" class in "`packet.py`" file. The router forwards the received packets to client *B*. Once all the data in the "`sendFile`" has been sent, the code terminates the connection using `FIN, FIN-ACK, ACK` packets. **Connection set up and termination packets are never dropped** in this simulator. Client *B*, on receiving a data packet from the router, writes the payload of the packet to the "`recvFile`". You may run the provided code to check its behavior using a command such as,

```
$ python3 network.py 01.json sendfiles/file4.txt recvfiles/file4.txt 0
```

Your task is to augment the provided code in "`myClient.py`" with a **reliability protocol** that ensures reliable file transfer **even under packet loss.** You are free to implement ***any*** reliability protocol of your choice. We discussed several different options in Transport Layer lectures. Further, please refer to section 6 for grading considerations in choosing the right protocol. You will do all your implementation inside "`myClient.py`". **You must not modify any other file in the "`Lab4/`" directory. Use of any external libraries not already included in "`myClient.py`" must be approved by the instructors**.

5 Output

The simulator prints different characters on the terminal to track the progress of the file transfer.

`+` is printed every time the router receives a packet from client *A* and forwards it to client *B*.

`[+]` is printed every time the router receives a packet from client *A* and drops it.

`@` is printed every time the router receives a packet from client *B* and forwards it to client *A*.

`[@]` is printed every time the router receives a packet from client *B* and drops it.

Once the connection successfully terminates, the simulator prints the file transfer statistics on the terminal. This includes the total time of transfer and the total bytes sent (including the bytes that were dropped). At the end, the simulator matches the received file against the sent file, and prints either "SUCCESS" or "FAILURE" depending upon whether the sent and received files are identical or not. **If you print any custom / debug statements to the terminal, it will result in a 20% grade penalty.**

The simulator also generates `.dump` files inside the "`logs/`" directory logging the header and payload contents of each received packet. The log also indicates if a packet received at the router was dropped.

6 Grading

We will test your solution against several test cases using different files (chosen from the set of provided files in the "`sendfiles`" directory) and different loss probability values. Your solution will be evaluated for both correctness (i.e., sent and received files are identical) and performance. The two performance metrics that will be used for evaluation are the **total file transfer time** and the **number of bytes sent**.

For each test case, we will choose a "good" student solution as the **baseline** (e.g., a student solution that is in the 90+ percentile for both performance metrics amongst all correct solutions). We will also set a very generous **timeout** for each test case, equal to 100× the baseline file transfer time. For each test case, if your code terminates within the timeout period and the received file is identical to the sent file, you will receive **70% credit for correctness**. However, if your code times out for a test case or the received and sent files are not identical, you will receive 0 points for that test case.

Assuming your solution passes the correctness check, you will receive the remaining credits for that test case if, for your solution, the number of **bytes sent** is within 1.5× (for 15% credit) or within 1.75× (for 10% credit) or within 2× (for 5% credit) of the baseline, and the **file transfer time** is within 2× (for 15% credit) or within 3× (for 10% credit) or within 4× (for 5% credit) of the baseline. Your final grade will be the sum total of points obtained across all test cases.

7 Bonus Credits

You will receive 1 bonus point per test case, if your solution is within 1.25× for bytes sent **and** within 1.5× for file transfer time, compared to the baseline for that test case. The total bonus points are capped at 5 points per student.

8 Submission

You are required to submit **one** file "`myClient.py`" on Brightspace. **Do not submit a .zip file.**

9 Performance Baselines from Previous Year

Below are some perofrmance numbers for the baseline solutions from the previous year. Note that the baselines for this year may look very different from the previous year, depending upon the quality of submissions received this year. **Students are allowed to report the performance numbers for their solutions on Piazza to gauge the competition for this year**. Further, the test cases reported below are not representative of the test cases that will be used for evaluation, i.e., we may use different combinations of files and loss probability values for evaluation than the ones reported below.

File	10% loss		30% loss		50% loss		70% loss		90% loss	
	Time (s)	Bytes	Time (s)	Bytes	Time (s)	Bytes	Time (s)	Bytes	Time (s)	Bytes
file1.txt	15.221	468	25.134	722	35.146	1242	40.174	2428	160.247	12556
file2.txt	25.234	10044	40.266	11398	45.163	26198	70.203	41418	230.37	272704
file3.txt	50.28	57328	85.255	93008	135.33	155538	250.362	322406	935.948	1272066
file4.txt	110.278	149020	170.207	225914	295.361	382792	650.668	834342	2356.981	2992544
file5.txt	295.413	431617	455.601	651087	780.829	1107376	1656.525	2335082	6770.91	9365338