ECE/CS 438: Communication Networks **Fall 2024**

Machine Problem 2

Handed Out: Sep 23^{rd}*,* 2024 *Due: Oct* 23^{rd} *23:59,* 2024

Student Name:

1 Introduction

In this MP, you will implement a transport protocol with properties equivalent to TCP, with all **three states** discussed in class. You have been provided with a file called `sender_main.c`, which declares the function

`void reliablyTransfer(char* hostname, unsigned short int hostUDPport, char* filename, unsigned long long int bytesToTransfer)`.

This function should transfer the first `bytesToTransfer` bytes of filename to the receiver at `hostname: hostUDPport` correctly and efficiently, even if the network drops or reorders some of your packets. You also have `receiver_main.c`, which declares `void reliablyReceive(unsigned short int myUDPport, char* destinationFile)`. This function is `reliablyTransfer`'s counterpart, and should write what it receives to a file called `destinationFile`.

2 What is expected in this MP?

Your job is to implement `reliablyTransfer()` and `reliablyReceive()` functions, with the following requirements:

- The data written to disk by the receiver must be exactly what the sender was given, **including binary files**.
- Two instances of your protocol competing with each other must converge to roughly fairly sharing the link (same throughputs ±10%), within 100 RTTs. The two instances might not be started at the exact same time.
- Your protocol must be somewhat TCP friendly: an instance of TCP competing with you must get on average at least half as much throughput as your flow.
- An instance of your protocol competing with TCP must get on average at least half as much throughput as the TCP flow. (Your protocol must not be overly nice.)
- All of the above should hold in the presence of any amount of dropped packets. All flows, including the TCP flows, will see the same rate of drops. The network will not introduce bit errors.
- Your protocol must, in steady state (averaged over 10 seconds), utilize at least 70% of bandwidth when there is no competing traffic, and packets are not artificially dropped or reordered.
- You cannot use TCP in any way. Use SOCK_DGRAM (UDP), not SOCK_STREAM.

The test environment has a 20Mbps connection, and a 20ms RTT.

3 VM/Docker Setup - Replicating the Test Environment

You can use the same basic test environment used for MP1 with two containers/virtual machines. However, the network performance will be ridiculously good (same goes for testing on localhost), so you'll need to limit it. The autograder uses `tc` . If your network interface inside the container (or VM) is `eth0`, then run (from inside the container) the following command:

```
tc qdisc del dev eth0 root 2>/dev/null
```

to delete existing tc rules. Then use,

```
tc qdisc add dev eth0 root handle 1:0 netem delay 20ms loss 5%
```

followed by

```
tc qdisc add dev eth0 parent 1:1 handle 10:  tbf rate 20Mbit burst 10mb
latency 1ms
```

will give you a 20Mbit, 20ms RTT link where every packet sent has a 5% chance to get dropped. Simply omit the `loss n%` part to get a channel without artificial drops.

You can add '`sudo`' to run these commands in Virtual Machines.

(You can run these commands just on the sender; running them on the receiver as well won't make much of a difference, although you'll get a 40ms RTT if you don't adjust the delay to account for the fact that it gets applied twice.)

3.1 Debugging

PLEASE do not fall into the trap of "debugging on the autograder". If you submit a new version every time you make some change that might help pass an extra test, you are going to waste a lot of time waiting for results. Rather, only submit when you have made major progress or have definitively figured out what you were previously doing wrong. If you aren't genuinely surprised that your most recent submission didn't increase your score, you are submitting too often.

For testing your code locally, we recommend using `tcpdump` along with `wireshark` to test network statistics.

```
tcpdump -i eth0 host <host IP address> -w capture.pcap
```

allows you to capture all the packets transmitted from the local system to the remote host at the given IP address into the file `capture.pcap`. You can then analyze these files using Wireshark.

We didn't include `tcpdump` with the previous container. You can either install it yourself, or use this one.

```
https://courses.grainger.illinois.edu/cs438/fa2024/files/ece438_v2.
tar
```

Wireshark can be downloaded and used on your local machine.

4 Autograder and Submission

Similar to MP1, use the contents of the .release mp2 folder as a starting point and make the modifications required for this assignment.

The following lists the **additional notes for MP2**:

- The MTU on the test network is 1500, so up to a 1472 byte payload (IPv4 header is 20 bytes, UDP is 8) won't get fragmented. You can `sendto()` larger packets and the sockets library's UDP will handle fragmentation/reassembly for you. It's up to you to reason out the benefits and drawbacks of using large UDP packets in various settings.
- You can use the provided main files to be sure your program runs with the right interface, or write your own. **Executable names:** `reliable_sender` and `reliable_receiver`. As with mp1, a single run of "make" (with no arguments) inside your mp2 directory should build both binaries.
- Be sure you have a clean design for implementing the send/receive buffers. Trying to figure out where to get the data to resend an old packet won't be fun if your send window's buffer doesn't have a nice clean interface.
- Input files on the grader are READ-ONLY. Do not use the "rb+" mode to read them; the '+' adds write permission. (In general, you shouldn't use "rb+" unless you need it).

Modify the Makefile such that a simple `make` command in your mp2 folder creates the required executables. The autograder does just that. Be careful about the executable filenames and output filenames.

Submission instructions are same as for mp1. Tests generally take **10-15 minutes**, and there may be a queue of students. You can see where you are in the queue at `http://cs438fa22.csl.illinois.edu:8080/queue/queue_mp2.html`. This is a UIUC-private IP. Expect 2 hour+ waits near the deadline (but don't worry, you will be graded based on your git record).

Do refer to MP1 instructions for other notes.