Machine Problem 2

Abstract

This machine problem tests your understanding of reliable packet transfer. You will use UDP to implement your own version of TCP. Your implementation must be able to tolerate packet drops, allow other concurrent connections a fair chance, and must not be overly nice to other connections (should not give up the entire bandwidth to other connections).

1 Introduction

In this MP, you will implement a transport protocol with properties equivalent to TCP. 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.
- 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 Setup

You'll need 2 VMs to test your client and server together. Unfortunately, VirtualBox's default setup does not allow its VMs to talk to the host or each other. There is a simple fix, but then that prevents them from talking to the internet. So, be sure you have done all of your apt-get installs before doing the following! (To be sure, just run: **sudo apt-get install gcc make gdb valgrind iperf tcpdump**) Make sure the VMs are fully shut down. Go to each of their Settings menus, and go to the Network section. Switch the Adapter Type from NAT to "host-only", and click ok. When you start them, you should be able to ssh to them from the host, and it should be able to ping the other VM. You can use **ifconfig** to find out the VMs' IP addresses. If they both get the same address, **sudo ifconfig eth0 newipaddr** will change it. (If you make the 2nd VM by cloning the first + choosing reinitialize MAC address, that should give different addresses.)
New in MP2: You can use the same basic test environment described above. 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 VM is **eth0**, then run (from inside the VM) the following command:

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

to delete existing tc rules. Then use,

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

followed by

sudo 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 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 20ms RTT if you don't adjust the delay to account for the fact that it gets applied twice.)

4 Autograder and submission

We use the autograder to grade your MPs, the submission process is simple.
First, open the autograder web page:

http://10.105.100.204.

This is a ZJUI-private IP. If your device is not accessing it through the campus network, please use VPN to get a private IP

You will see two sections. **MP Submission** allows you to submit your assignment. You can do this by entering your Student ID (start with 320), selecting which MP you are submitting, selecting the file extension of your files and uploading your MP files. **Note**: only C/Cpp files are accepted. When uploading files, **add your files one by one, do not choose multiple files to add at one time**. **Submission History** section allows you to check your submission history and grade. You can do this by entering your student ID.

Caution: The queue can only handle 200 submissions at one time, so remember to check your submission status in **Submission History** after you submit your assignment. During the hours leading up to the submission, the queue could be long. So it is advisable to get your work done early.

5 Grade Breakdown

10%: You submitted your assignment correctly and it compiles correctly on the autograder
20%: Your receiver and sender can transfer small files correctly
20%: Your sender and receiver can transfer big files correctly
10%: Your sender and receiver can utilize empty link
20%: Your sender and receiver can transfer files correctly with package dropping
10%: Your sender and receiver are TCP friendly with no loss
10%: Your sender and receiver are TCP friendly with 1% loss
(We will use diff to compare the output file with the downloaded copy, and you should do the same. If diff produces any output, you aren't transferring the file correctly.)

6 Test Details

Your MP2 code will run entirely within Docker Containers and will be tested via Docker's private network.
The testing for MP2 is divided into 7 stages:

1. **Compiling**: We will compile the code for your sender and receiver. If the compilation is successful, you will receive the basic score for successful compilation; if it fails, you will receive a score of 0, and subsequent tests will not be conducted.

2. **Small file transfer test in a no-loss environment**: We will conduct a small file transfer test of your sender and receiver in a Docker network without any artificially induced packet loss. First, we'll launch your receiver code in one container and then your sender code in another container. The time limit for this task is 5 seconds. If your code can complete the transfer of the small file within 5 seconds and the output file successfully matches the source file when compared using diff, you will receive the score for this phase. If this phase fails, no further tests will be conducted.

3. **Large file transfer test in a no-loss environment**: We will test your code using a large file of 18.4MB. The testing method is the same as for the small file, but you have a time limit of 10 seconds. If the transfer is completed within 10 seconds and passes the diff comparison, you earn this phase's score. If this phase fails, subsequent tests will proceed unaffected.

4. **Channel bandwidth utilization test**: We will transfer a file in a no-loss environment and place a stricter time constraint on its transfer than the previous tests to ensure your code utilizes the channel bandwidth to its maximum potential. If the file transfer completes within the stipulated time and passes the diff comparison, you earn this phase's score. If this phase fails, subsequent tests will proceed unaffected.

5. **File transfer test with 5% packet loss and 20ms delay**: We will set the Docker network to have a 5% packet loss and a 20ms delay and test whether your code can accurately receive the file. We'll use a file of several KBs for the test. If the file is transferred correctly within 10 seconds and passes the diff comparison, you earn this phase's score. If this phase fails, subsequent tests will proceed unaffected.

6. **TCP-friendly test**: This test will be conducted within a 300Mbps channel. Using iperf3, we will create a TCP stream and measure its baseline rate when it's the only stream. Then, we'll run your code and iperf3 concurrently and test the rate the TCP stream can achieve when sharing the channel. If the TCP stream manages to get more than 40% of its baseline rate, the test is passed. If this phase fails, no further tests will be conducted.

7. **TCP-friendly test with 1% loss**: We will test within a 300Mbps channel with a

1% packet loss. Similar to the previous test, we will first record the baseline rate of only the TCP stream. We'll then test the rate the TCP stream can achieve when your code and the TCP stream share the channel. If the TCP stream gets more than 40% of its baseline rate, the test is passed.

Please note that due to the influence of the autograder's network environment, there might be a drop in speed during times of high submission volume. In such cases, you can resubmit after a while. Each test takes about a minute and a half. We will retain the highest score from all your submissions as your final score. The files you submit must be named either "receiver_main.cpp", "sender_main.cpp" or "receiver_main.c", "sender_main.c".