Project 1

Edit 2022-09-26: Removed loss_rate parameter from t_rcv/t_ncp interface. You do NOT need to emulate loss with your TCP benchmark program.

Overview

Goal: Construct a basic reliable point-to-point file transfer tool using UDP/IP, and evaluate its performance in the presence of packet loss.

As discussed in lecture, your protocol should use a selective repeat strategy employing cumulative acknowledgments and explicit negative acknowledgments.

You should work on this assignment in groups of exactly 2 students. Contact us if you cannot find a partner and we will arrange one for you.

Your programs should be written using C or C++.

The **initial design** submission date is **Monday, September 12, 2022, at the beginning of class**. The **final submission** date is **Monday, September 26, 2022, 11:59pm.**

File Transfer Tool Details

Using the unreliable UDP/IP protocol, you should write a tool that reliably copies a file from one machine to another machine.

The tool consists of two programs:

1. ncp - the sender process
2. rcv - the receiver process

Receiver Process

In order to perform a file transfer operation, a receiver process (rcv) should be run on the target machine using the following interface:

```
./rcv <loss_rate_percent> <port> <env>
```

`<loss_rate_percent>` is the loss percentage to emulate for sent packets, e.g. rcv 10 will randomly drop 10% of packets before sending.
`<port>` is the port on which the rcv process will receive incoming packets.
`<env>` should be either LAN or WAN to indicate whether the program should use parameters tuned for a local area network (LAN) or wide area network (WAN) environment.

A receiver process (rcv) should handle an UNLIMITED number of file-transfer operations, but it is allowed to handle one operation at a time (sequence them).
If a sender comes along while the receiver is busy, the sender should be blocked until the receiver completes the current transfer. The sender should know that it was blocked and report this.

Sender Process

A sender process (ncp) should be run on the source machine using the following interface:

```
./ncp <loss_rate_percent> <env> <source_file_name> <dest_file_name>@<ip_address>:<port>
```

Similarly to the rcv process, `<loss_rate_percent>` is the loss percentage to emulate for sent packets, and `<env>` indicates whether LAN or WAN parameters should be used.
`<source_file_name>` is the name of the local file on the sender machine to transfer to the receiver.
`<dest_file_name>` is the name to save the file as on the target (receiver) machine.
`<ip_address>` is the IP address of the target (receiver) machine.
`<port>` is the port on which the target rcv process receives messages (i.e. this should match the port specified for the rcv process)

A sender process (npc) handles one operation and then terminates.
You can assume that the source file name represents a specific single file.

Loss Emulation

To check the software, each process (both rcv and npc) must call a wrapper routine named `sendto_dbg` instead of the `sendto` system call. The wrapper routine will allow emulating a specific network loss percentage for the sent packets. **This MUST be done for every send operation in the code** (both for the sender and the receiver).

The wrapper routine `sendto_dbg` has a similar interface to sendto (see `man 2 sendto`).

The `sendto_dbg` routine will randomly decide to discard packets in order to create additional network message omissions based on the loss rate.
An initialization routine, `sendto_dbg_init`, should be called once, when the program is started, to set the loss rate for that program execution.
The source for the `sendto_dbg` and `sendto_dbg_init` routines is available in Canvas (**you do NOT need to write this yourself**).

Reporting

Both the sender (ncp) and the receiver (rcv) programs should report two statistics every 10 megabytes of data sent/received IN ORDER (all the data from the beginning of the file to that point was received with no gaps):

1. The total amount of data (in megabytes) successfully transferred so far.

2. The average transfer rate of the last 10 megabytes sent/received (in megabits/sec).

For our purposes, please use the base 10 definition: 1 megabyte = 1,000,000 bytes = 8,000,000 bits

At the end of the transfer, both sender and receiver programs should report:

1. the size of the file transferred (in **megabytes**)
2. the amount of time required for the transfer (in **seconds**)
3. the average transfer rate (in **megabits/sec**)

The **sender program** should additionally report:

1. the total amount of data sent (in megabytes), **including retransmissions**.

This metric will help you judge the efficiency of your protocol in terms of bandwidth usage. Ideally, you want your protocols to have a short total transfer time, high average transfer rate, and low bandwidth overhead (ratio of total data sent to actual file size).

Evaluating Performance

We have given you a GENI request rspec that you can use to set up your GENI evaluation environment. It includes 2 nodes in one site and a 100 megabit/sec link between them. The IP address of node-0 is set to 10.0.1.100, and the IP address for node-1 is set to 10.0.1.101. To simplify experiments and provide a consistent environment for all teams, we will emulate wide-area network latency for the evaluation instead of running real wide-area experiments. For extra credit, you can additionally test your protocol in a real wide-area environment.

Implementing a TCP benchmark

Using the reliable TCP/IP protocol, implement the corresponding t_ncp and t_rcv.

The interfaces should be:

```
./t_rcv <port>
```

and:

```
./t_ncp <source_file_name> <dest_file_name>@<ip_address>:<port>
```

These programs should report the amount of data transferred, the transfer rate, and the total transfer time, as described above for the ncp and rcv programs. The t_ncp program does not need to report the retransmission overhead.

The t_rcv is not required to handle multiple senders. It can handle a single sender and then exit.

Local area evaluation

Compare the performance of ncp/rcv to t_ncp/t_rcv by measuring the time it takes to copy a 100-Megabyte file from one GENI VM to another GENI VM located in the same site.

You should run this test for the ncp/rcv 5 times for each of the following loss rates: 0% 1% 5% 10% 20% 30%. You should average the results of the 5 tests and graph the results with the X-axis being the loss rate and the Y-axis being the time the copy took.

Thus, you should turn in results comparing eight experiments:
1. t_ncp/t_rcv
2. ncp/rcv with zero loss rate
3. ncp/rcv with 1% loss rate
4. ncp/rcv with 5% loss rate
5. ncp/rcv with 10% loss rate
6. ncp/rcv with 20% loss rate
7. ncp/rcv with 30% loss rate
8. ncp/rcv with zero loss rate, run AT THE SAME TIME as t_ncp/t_rcv
   (this will evaluate your protocol when competing with TCP)

You must turn in a table with the results of the experiments (for all 5 runs) and a graph with the averages as described above.

(Emulated) Wide area evaluation

Compare the performance of ncp/rcv to t_ncp/t_rcv by measuring the time it takes to copy a 100-Megabyte file with latency added to emulate a wide-area connection.

To achieve good performance in this evaluation, you will need to re-tune some of your parameters (i.e. adjust window size and timeouts). You should
experiment with different settings for these parameters and select the one that gives the best performance. Then, you should repeat the 8 experiments above in the wide area environment.

Your final report must include a table with the results, and a graph, as described above. You must also discuss how you performed your parameter tuning and why you selected the window size and timeouts that you did.

To tune your protocol for a WAN environment and run your WAN experiments, you should emulate latency of 40ms between the sites. You should use the Linux netem tool to do this.

Specifically, to add latency, you should run the following on **EACH** of your two VMs (node-0 and node-1):

```
sudo tc qdisc add dev eth1 root netem delay 20ms
```

This adds 20ms latency in each direction, for a round-trip delay of 40ms.

To test that it was successful, you should test pinging between the two nodes.
On node-0, run:

```
ping 10.0.1.101
```

You should see delay measurements of about 40ms.

To remove the added delay (e.g. to go back to testing the LAN environment), you should run on **EACH** of your two VMs (node-0 and node-1):

```
sudo tc qdisc del dev eth1 root
```

You can test pinging again to check that it was successful. You should now see delay less than 1ms.

If you are interested in learning more about what netem can do, see:
https://wiki.linuxfoundation.org/networking/netem
(https://wiki.linuxfoundation.org/networking/netem)

(Real) Wide area evaluation -- Extra Credit

Compare the performance of ncp/rcv to t_ncp/t_rcv by measuring the time it takes to copy a 100-Megabyte file from one GENI VM to another GENI VM located in a *different* site.

You should pick two different InstaGENI sites for this part of the evaluation. To make this interesting, you should select sites in different cities, such that the roundtrip time between your VMs is at least 10 milliseconds (measure using the "ping" utility). You should report the sites you picked and the roundtrip time you measured between them in your final report.

To achieve good performance in this evaluation, you will likely need to re-tune some of your parameters. As above, you should experiment with different settings for these parameters and select the one that gives the best performance. Then, you should repeat the 8 experiments above in your real wide area environment. Your final report must include a table with the results, and a graph, as described above. You should also discuss how you performed your parameter tuning and why you selected the window size and timeouts that you did.

If you do the extra credit evaluation, you should allow us to test it by using "WAN-EC" as the env option for ncp/rcv (instead of LAN/WAN).

Submission

Two separate submissions are required.

1. An electronic submission of your initial design. Due **Monday September 12 at the beginning of class**. You should also bring your initial design document to class on September 12 (either print it out or bring a laptop, as we will discuss your designs in class).

An initial design document should be a well-written, complete design of your ncp and rcv programs and the protocols used to make the transfer reliable. This includes the main ideas behind your protocol, the internal data structure in each program, the type and structure of the messages used, and the protocol (algorithm) description. It should be 1-2 pages long with diagrams as needed.

A final design document is required with your final submission. The final design document should be updated to reflect any changes made to your initial
design. Design makes up 20% of the exercise grade.

2. An electronic submission of all the documents/code described above. Due **Monday, September 26, 2022, 11:59pm.**. You should make a zip file named with both teammates' Pitt IDs with all your documents and code (DO NOT include the 100-Megabyte file transferred, core file, object files, or executables) and upload the zip file to Canvas.

The complete submission must include a report with your final design, performance results, graphs and discussion of the performance results and parameter tuning.

The programs should be complete, sound, and well documented. We should be able to easily run the local area and wide area variants of your programs WITHOUT MODIFYING YOUR CODE. We should be able to run your program with the "LAN" commandline option to reproduce your LAN results, and with the "WAN" commandline option to reproduce your WAN results.

A makefile that compiles and builds your complete program with clearly documented options (if needed) is required.

Example for creating the zip file on the commandline:

If your Pitt IDs are <pittid1> and <pittid2>, then create a directory named pittid1_pittid2 and copy all your source code, makefile, documentation and design in that directory. DO NOT copy the 100-Megabyte file you used for testing your programs or any intermediate other files.

Outside of the <pitt_id> directory run the following command:

```
zip -r pittid1_pittid2.zip pittid1_pittid2
```

To unzip and check the submission: `unzip pittid1_pittid2.zip`

Failure to comply with submission instructions will result in a lower grade. Your grade will include correctness, design, documentation, and, of course, efficiency.

Notes and suggestions

1. Your received file should be **identical** to the original file. Use the `diff` command to check this.

2. Your programs must work on the default GENI VM. You may use any development environment to work on your code, but all evaluation must be done on GENI. You should test your code on GENI

throughout your development process. In the past, teams that waited until the last minute to test on GENI were not able to complete the project successfully.

3. Do NOT rely on GENI to store your code. If your GENI resources expire, or there is a problem with the slice, you will lose any files stored there. I strongly recommend that you use a private Github repository to store your code and share it with your partner. You can clone the repo on your GENI VMs to allow you to easily pull your latest code to the VMs for testing.

4. Note that due to the shared nature of the GENI environment, bandwidth between VMs is limited to 100-Megabits/sec. For a LAN environment today, speeds are typically higher (e.g. 1 Gbps or 10 Gbps for academic/enterprise environments, or more for data center environments). If you want to experiment with a 1 Gbps cluster environment, you can also try running it in the CS elements cluster, but we are not requiring you to evaluate your protocol there to avoid issues with your programs conflicting with one another and overloading the network close to the deadline...

5. Your packet size should not exceed 1400 Bytes.

6. Experiment with different window sizes. Hint: you should be able to calculate the window size needed to obtain the maximum throughput *with no loss* based on roundtrip time, packet size, and network bandwidth. You should calculate and test this first. But, then think about how loss affects the ability to slide the window forward. How does your window size tuning change when you need to design the protocol to achieve high throughput under up to 30% loss?

7. In some cases, the file close operation may take a long time to complete. You may exclude this time from your final transfer rate (i.e. stop your timer just before calling fclose()) so that your measurements reflect the network latency rather than the latency to flush the file locally.