Attribution: This series of programming assignments, including codebase and documentation, is adopted from 'Sponge' of Stanford CS144 Introduction to Computer Networking by Prof. Keith Winstein https://cs144.github.io/.

Final Assignment: putting it all together

Due: May 31, 2024. 23:59
Late deadline: June 1, 2024. 23:59 (20% penalty)

1. Overview

By this point in the class, you've implemented a significant portion of the Internet's infrastructure. From Assignment 0 (a reliable byte stream) to Assignment 1–4 (the Transmission Control Protocol), Assignment 5 (an IP/Ethernet network interface) and Assignment 6 (an IP router), you have done a lot of coding!

In this assignment, **you won't need to do any coding** (assuming your previous assignments are in reasonable working shape). Instead, to cap off your accomplishment, you're going to *use* all of your previous assignments to create a real network that includes **your** network stack (host and router) talking to the network stack implemented by another student in the class.

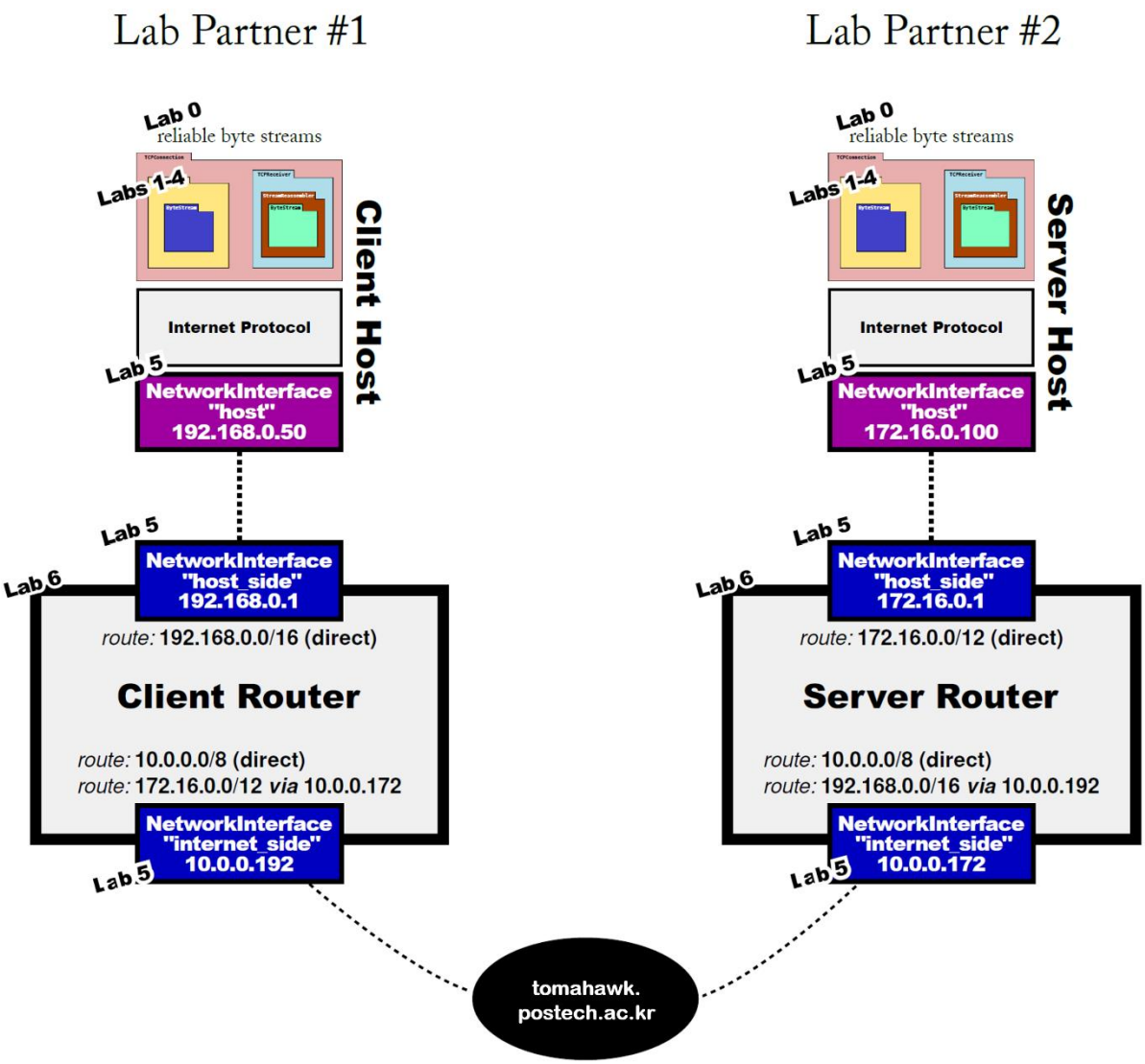
This assignment is done in pairs. You will need to work with your partner that you've designated on the team spreadsheet.

2. Getting started

1. Make sure you have committed all your solutions to Assignment 6. Please don't modify any files outside the top level of the `libsponge` directory, or `webget.cc`. (And please don't add extra files that your code relies upon.)
2. While inside the repository for assignments, run `git fetch` to retrieve the most recent version of assignments.
3. Download the code for Assignment 7 by running `git merge origin/lab7-startercode`.
4. Within your `build` directory, compile the source code: `make`.
   (you can run, e.g., `make -j4` to use four processors when compiling)
5. Outside the `build` directory, open and start editing the `writeups/assn7.md` file. This is the template for your assignment writeup and will be included in your submission.

3. The Network

In this assignment, you'll create a real network that combines your network stack with one implemented by another student in the class. Each of you will contribute one host (including your reliable `ByteStream`, your TCP implementation, and your `NetworkInterface`) and one router:



Because it's likely that you or your partner will be behind a Network Address Translator, the network connection between the two sides will flow through a relay server (tomahawk.postech.ac.kr).

We have glued your code together in a new application that can be found in `build/apps/lab7`. Here are the steps to run it:

1. Before doing these steps with a partner, **try them by yourself.** You can play both roles, client and server, by using two different windows or terminals on your VM. This way your network will include two copies of your code (host and router) talking to themselves. This is easier to debug than talking to a stranger!
   Once these steps work on your own, *then* try them with a partner. Decide which of the two of you will act as the "client" and who will be the "server". Once it works, you can always swap the roles and try again.

2. To use the relay, ***please check your team allocated port numbers***. This identifies your group and needs to be different from any other group working at the same time.

3. From the `build` directory, the "server" student runs:
   `./apps/lab7 server tomahawk.postech.ac.kr [server_port]`
   (replace "[server_port]" with your team server port).

   If all goes well, the "server" will print output like this:

```
$ ./apps/lab7 server tomahawk.postech.ac.kr 3000
DEBUG: Network interface has Ethernet address 02:00:00:5e:61:17 and IP address 172.16.0.1
DEBUG: Network interface has Ethernet address 02:00:00:cd:e7:e0 and IP address 10.0.0.172
DEBUG: adding route 172.16.0.0/12 => (direct) on interface 0
DEBUG: adding route 10.0.0.0/8 => (direct) on interface 1
DEBUG: adding route 192.168.0.0/16 => 10.0.0.192 on interface 1
DEBUG: Network interface has Ethernet address 5a:75:4e:8b:20:00 and IP address
172.16.0.100
DEBUG: Listening for incoming connection...
```

4. From the `build` directory, the "client" student runs:
   `./apps/lab7 client tomahawk.postech.ac.kr [client_port]`
   (replace "[client_port]" with your team client port).

   If all goes well, the "client" will print output like this:

```
$ ./apps/lab7 client tomahawk.postech.ac.kr 3001
DEBUG: Network interface has Ethernet address 02:00:00:41:c7:5b and IP address
192.168.0.1
DEBUG: Network interface has Ethernet address 02:00:00:e6:66:d9 and IP address
10.0.0.192
DEBUG: adding route 192.168.0.0/16 => (direct) on interface 0
DEBUG: adding route 10.0.0.0/8 => (direct) on interface 1 DEBUG: adding route
172.16.0.0/12 => 10.0.0.172 on interface 1
DEBUG: Network interface has Ethernet address 26:05:12:4a:8a:c9 and IP address
192.168.0.50
DEBUG: Connecting from 192.168.0.50:57005...
DEBUG: Connecting to 172.16.0.100:1234...
Successfully connected to 172.16.0.100:1234.
```

   and the "server" will print one more line:

```
New connection from 192.168.0.50:57005.
```

5. **If you see the expected output**, you're in really good shape—the two computers have successfully exchanged a TCP handshake!
   (a) Pat yourselves on the back (using appropriate social distancing protocols)—you've earned it!
   (b) Now it's time to exchange data. Type in one of the windows, and see the output appear in the other. Try typing in the reverse direction.
   (c) Try ending the connection. Type `Ctrl-D` when you are done. When each side does so, it will end input on the outbound `ByteStream` in that direction, while continuing to receive incoming data until the peer ends its own `ByteStream`. Verify this happens.

(d) When both sides have ended their `ByteStream`s, and one side has finished lingering for a few seconds, both programs should exit gracefully.

6. **If you don't see the expected output**, it may be time to turn on "debug mode". Run the "lab7" program with one additional argument: append a "debug" to the end of the command line. This will print out every Ethernet frame being exchanged, and you can see all the ARP and TCP/IP frames.

7. Once you have the network working between two windows on your own computer, it's time to try the same steps with a partner (and their own implementation).

4. Sending a file

Once it looks like you can have a basic conversation, try sending a file over the network. Again, you can try this yourself, and if all goes well, then try it with a partner. Here is how:

To **write** a one-megabyte random file to `/tmp/big.txt`:

```
dd if=/dev/urandom bs=1M count=1 of=/tmp/big.txt
```

To have the server send the file as soon as it accepts an incoming connection:

```
./apps/lab7 server tomahawk.postech.ac.kr [server_port] < /tmp/big.txt
```

To have the client close its outbound stream and download the file:

```
</dev/null ./apps/lab7 client tomahawk.postech.ac.kr [client_port] >
/tmp/big-received.txt
```

To compare two files and make sure they're the same:

`sha256sum /tmp/big.txt` or `sha256sum /tmp/big-received.txt`

If the SHA-256 hashes match, you can be almost certain the file was transmitted correctly.

5. Getting creative

For some (token) extra credit, if everything is working perfectly, we'd encourage you to do something creative and put something interesting in your writeup. Please feel free to modify the `lab7.cc` program as you see fit. You could create a more complicated network involving more students at the same time, or do something else we haven't anticipated. (To be clear: this is not at all required; but you may get an extra credit, +50% in maximum, for your creative work)

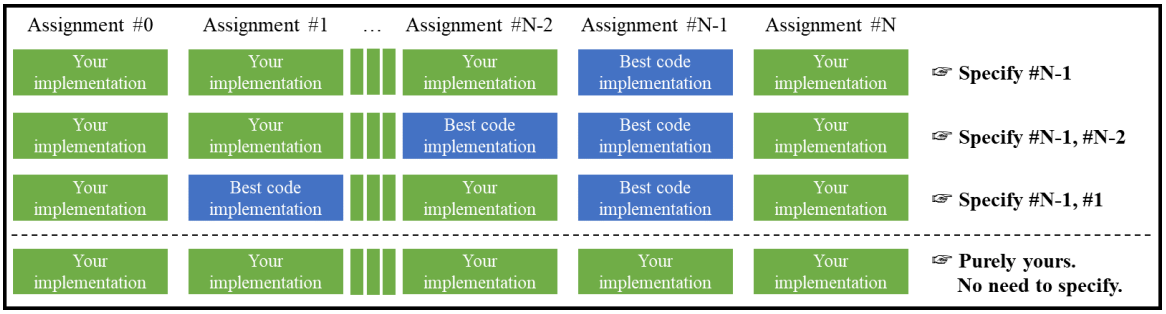
6. Submit

1. **Write a report in `writeups/assn7.md`.** This file should be a roughly 30-to-70-line document with no more than 80 characters per line to make it easier to read. **The report should contain the following sections:**

- Solo portion
  - ✓ Did your implementation successfully start and end a conversation with another copy of itself?
  - ✓ Did it successfully transfer a one-megabyte file, with contents identical upon receipt?
  - ✓ Please describe what code changes, if any, were necessary to pass these steps.
- Group portion
  - ✓ What is your team's name? Who is your partner (and what is their POVIS ID)?
  - ✓ Did your implementations successfully start and end a conversation with each other (with each implementation acting as "client" or as "server")?
  - ✓ Did you successfully transfer a one-megabyte file between your two implementations, with contents identical upon receipt?
  - ✓ Please describe what code changes, if any, were necessary to pass these steps, either by you or your partner.
- Creative portion
  - ✓ If you did anything for our "creative challenge", please boast about it! 😁

2. If you think the relay server is dead, please post it on the Q&A board.
3. If you did have to make changes to source code, please only make changes to the `.hh` and `.cc` files in the top level of `libsponge`. Within these files, please feel free to add private members as necessary, but please don't change the public interface.
4. Please don't add extra files—the automatic grader won't look at them and your code may fail to compile.
5. Before handing in any assignment, please run these in order:
   (a) `make format` (to normalize the coding style)
   (b) `git status` (to check for un-committed changes—if you have any, commit!)
   (c) `make` (to make sure the code compiles)
   (d) `make check` (to run the full test suite—please make sure you have not broken any tests)
6. In your writeup, please specify the usage of best-submission codes, if you have used **any**.

   Caution: When specifying your usage of best codes, you should include all of the best codes that exist in not only your current Assignment but **also in any part of your past Assignment that you are building upon**. The below figure shows examples of how to specify the usage of best codes.



If your submission is not purely yours, your score will be capped at 90% of the full score, and your code is not a candidate for the best code selection. This penalty persists unless you replace all the best codes you used **with your implementation**.

7. In your writeup, please also fill in the number of hours the assignment took you and any other comments.

8. When you have finished your implementation, do not forget to **commit all changes you made!** This **should be done before creating an archive** of your git repository. If you create an archive before committing changes, the archive will include only part or none of the changes you made. This may result in a lower score than you anticipated, or worse zero.

   If you are unsure whether your archive includes all changes you made, please refer to our Q&A item: "How do I make sure the created bundle includes all my commits?".

9. To archive your repository, use the following command to archive your git repository:

   ```
   git bundle create /tmp/<your_student_id>.git --all
   ```

   Important: this command will create an archive of your git history, which means that your solution must be committed to your git repository! In addition, we will be grading only the master branch of your repository, so please be sure that branch corresponds to the solution you want to turn in.

10. Please upload **your bundle file and the essay** to the proper entry on PLMS.

    (a) Each team is required to write an essay about the person (i.e., pioneers of computer networks) in their team's name. In the PLMS, you and your partner are set as a team, thus one of you is required to submit the essay to the homework entry (i.e., *[Homework] Essay about great minds in computer networks*).

    (b) The essay should be 20+ lines long, written in font size 10pt, and written in complete sentences and paragraphs. **Not bulleted format**. Please write it in MS word, and submit the **.docx** file.

    (c) Rough guidelines for essay content (can be flexibly adjusted to some context):

    i. About 20%: The person's career, e.g., where and when the person got the Ph.D., what position the person had worked in which company, university, etc.

    ii. About 50%: The person's contribution to computer networks, e.g., founding technology or principles, something striking developed on top of networks.

    iii. About 10%: The person's major awards, if applicable.

    iv. About 20%: *Your own opinions* about technological and/or social impacts that

the person’s contribution produced.