Checkpoint 7: putting it all together

Due: end of class (March 15, 11:59 p.m.)

0 Collaboration Policy

Collaboration Policy: Checkpoint 7 asks you to work with a groupmate—you'll need to connect together and perhaps debug together. In general, though, the collaboration rules are the same as checkpoint 0. Please do not look at other students' code or solutions to past versions of these assignments. Please fully disclose any collaborators or any gray areas in your writeup—disclosure is the best policy.

1 Overview

By this point in the class, you've implemented a significant portion of the Internet's infrastructure. From Checkpoint 0 (a reliable byte stream), to Checkpoints 1–3 (the Transmission Control Protocol), Checkpoint 5 (an IP/Ethernet network interface) and Checkpoint 6 (an IP router), you have done a lot of coding!

In this checkpoint, **you won't necessarily need to do any coding** (assuming your previous checkpoints are in good working shape). Instead, to cap off your accomplishment, you're going to *use* all of your previous labs 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 checkpoint is done in pairs. You will need to work with a lab partner (another student in the class). Please use the lab sessions to find lab partners, or EdStem if you cannot attend the lab session. If it's necessary, the same student can serve as "lab partner" more than once.

2 Getting started

1. Make sure you have committed all your solutions. Please don't modify any files outside the top level of the `src` directory, or `webget.cc`. You may have trouble merging the Checkpoint 7 starter code otherwise.

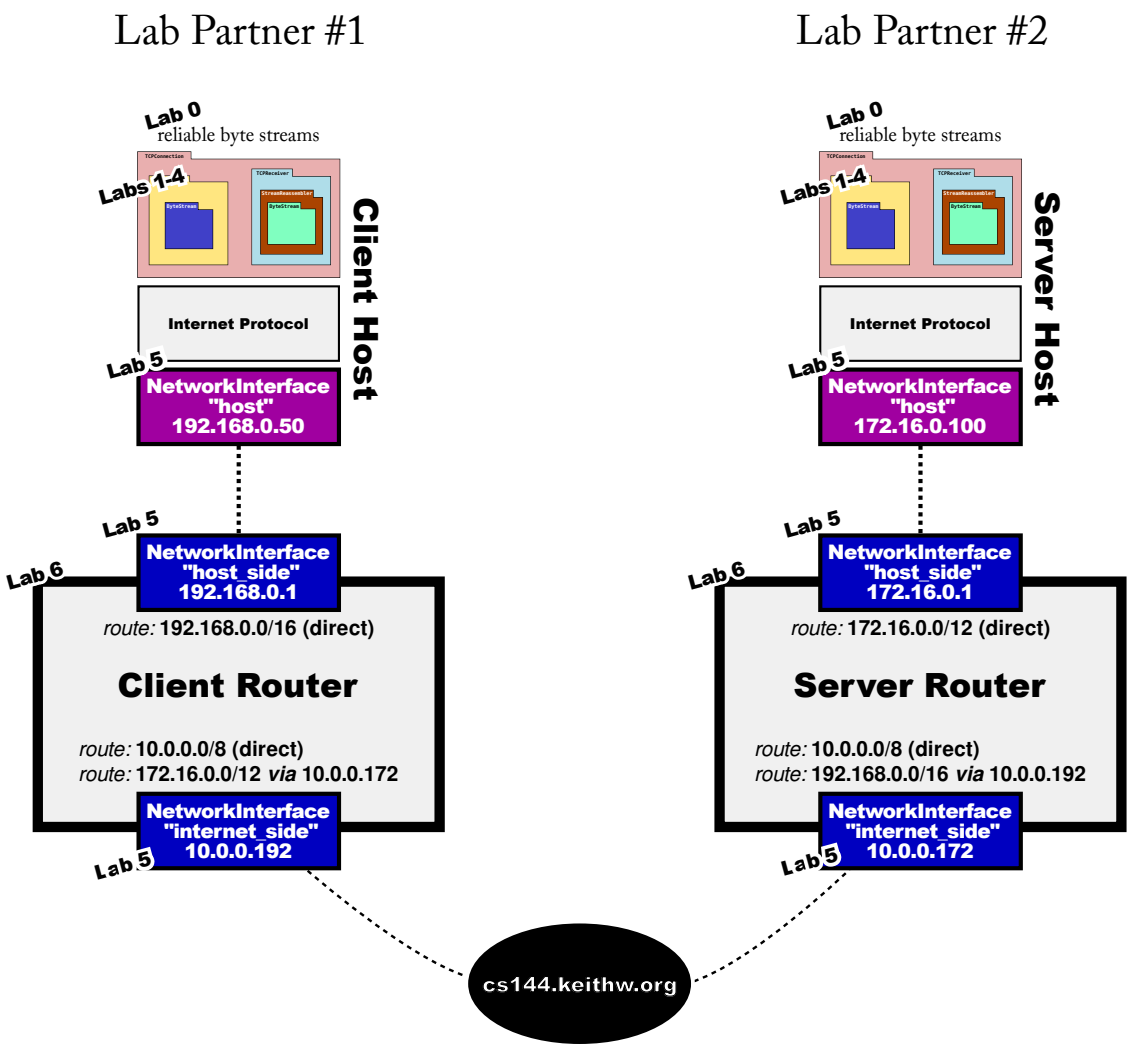
2. While inside the repository for the lab assignments, run `git fetch --all` to retrieve the most recent version of the lab assignment.

3. Download the starter code for Checkpoint 7 by running `git merge origin/check7-startercode`. (If you have renamed the "origin" remote to be something else, you might need to use a different name here, e.g. `git merge upstream/check7-startercode`.)

4. Make sure your build system is properly set up: `cmake -S . -B build`
5. Compile the source code: `cmake --build build`
6. Open and start editing the `writeups/check7.md` file. This is the template for your lab writeup and will be included in your submission.
7. Reminder: please make frequent **small commits** in your local Git repository as you work. If you need help to make sure you're doing this right, please ask a classmate or the teaching staff for help. You can use the `git log` command to see your Git history.

3 The Network

In this lab, 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 Byte Stream, your TCP implementation, and your NetworkInterface) and one router:



Because it's likely that you or your lab partner will be behind a Network Address Translator, the network connection between the two sides will flow through a relay server (cs144.keithw.org).

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

1. Before doing these steps with a lab 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 lab 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 pick a *random even number* between 1024 and 64000. This identifies your lab group and needs to be different from any other lab group working at the same time, so please do pick a random number. And it needs to be an even number. For the rest of these examples, we'll assume you picked "3000". But don't actually use "3000"—it needs to be a different number from everybody else.

3. From the "build" directory, the "server" student runs:
   `./apps/endtoend server cs144.keithw.org 3000`
   (replace "3000" with your actual number).

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

```
$ ./apps/endtoend server cs144.keithw.org 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/endtoend client cs144.keithw.org 3001`
   (replace "3001" with whatever your random number was, plus one).

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

```
$ ./apps/endtoend client cs144.keithw.org 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 ByteStreams, 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 "endtoend" 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 lab 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 lab 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/endtoend server cs144.keithw.org` *`even_number`* `< /tmp/big.txt`

To have the client close its outbound stream and download the file:
`</dev/null ./apps/endtoend client cs144.keithw.org` *`odd_number`* `> /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 Extra credit

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 endtoend.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.)

6 Submit

1. Write a report in `writeups/check7.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
     - Who is your lab partner (and what is their SUNet ID, e.g. `winstein`)?
     - 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 lab partner.
   - Creative portion
     - If you did anything for our "creative challenge," please boast about it!

2. 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 `src`. Within these files, please feel free to add private members as necessary, but please don't change the public interface.

3. Please don't add extra files—the automatic grader won't look at them and your code may fail to compile.

4. Please also fill in the number of hours the assignment took you and any other comments.

5. Please let the course staff know ASAP of any problems at the lab sessions, or by posting a question on EdStem.