Checkpoint 7 draft: making an Internet + something creative!

Due: end of class (Dec. 5, 11:59 p.m.)

0 Collaboration Policy

Collaboration Policy: Checkpoint 7 asks you to work with as many groupmates as possible—you'll need to connect together to build a class "Internet" and perhaps debug together. Talking about packets and ideas is strongly encouraged! As before, though, we're sticking to the clear line: please do not look at other students' *code* or ask for or receive code from anybody, including a chatbot or former student. As with previous checkpoints, please fully disclose collaborators, LLM prompts, or any gray areas in your writeup—this protects you.

1 Overview

By this point in the class, you've implemented a big 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 mastered much of the Internet's core infrastructure!

To cap off your accomplishment, you're going to *use* all of your previous labwork to design and run a **"Checkpoint 7 Capstone Internet"** that includes your network stack (host and router) talking to the network stacks implemented by other students in the class. Please use the lab sessions to coordinate and connect, or EdStem to coordinate if you cannot attend.

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` and `ip_raw.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`.
4. Make sure your build system is properly set up: `cmake -S . -B build`
5. Remember that if you have trouble, you can build "sanitizing" (bug-checking) versions of the applications with `cmake -S . -B build -DSANITIZED_APPS=True`
6. Compile the source code: `cmake -build build`
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 Read and Understand the Starter Code

As part of the checkpoint 7 starter code, we've given you two new "apps" that use your ByteStream, TCP implementation, NetworkInterface, and Router.

- `apps/tcp_eth_udp.cc` implements a **"full stack" endpoint**: your TCP implementation creates TCP segments, which our code encapsulates in Internet datagrams. These are given to your NetworkInterface, which encapsulates them in Ethernet frames. The `tcp_eth_udp` tool encapsulates these Etheret frames in user datagrams and sends them over the normal Internet.
- `apps/fun_router` implement a compatible **IP router** by giving your Router a command-line interface. Command-line flags can create interfaces and add routes to the router's routing table. The Router's NetworkInterfaces also send Ethernet frames, encapsulated in user datagrams, over the Internet.

1. **Read through the source code** of these two tools and work to understand how they work and what they're doing.
2. One key point: both of these tools send IP-in-Ethernet-in-UDP-in-IP. As a result, there are *two layers* of IP addresses at work: the "physical" addresses used in the outermost layer of Internet datagrams sent over the real Internet, and the "virtual" addresses used in the "Checkpoint 7 Capstone Internet" that you'll design in this checkpoint.
3. When you use these tools, the "physical" addresses will correspond to real IP addresses that can reach another student's computer. This can include the CS144 virtual private network you previously used in checkpoints 1 and 3; from these apps' point of view, that's also the "real" Internet.
4. By contrast, the "virtual" addresses will be your own invention—**we want you to design an Internet** with an interesting topology, as many routers as possible routing packets, and the appropriate routing tables so that many students can simultaneously communicate with each other over this "virtual" Internet.

4 Make a Tiny Internet on Your Own VM

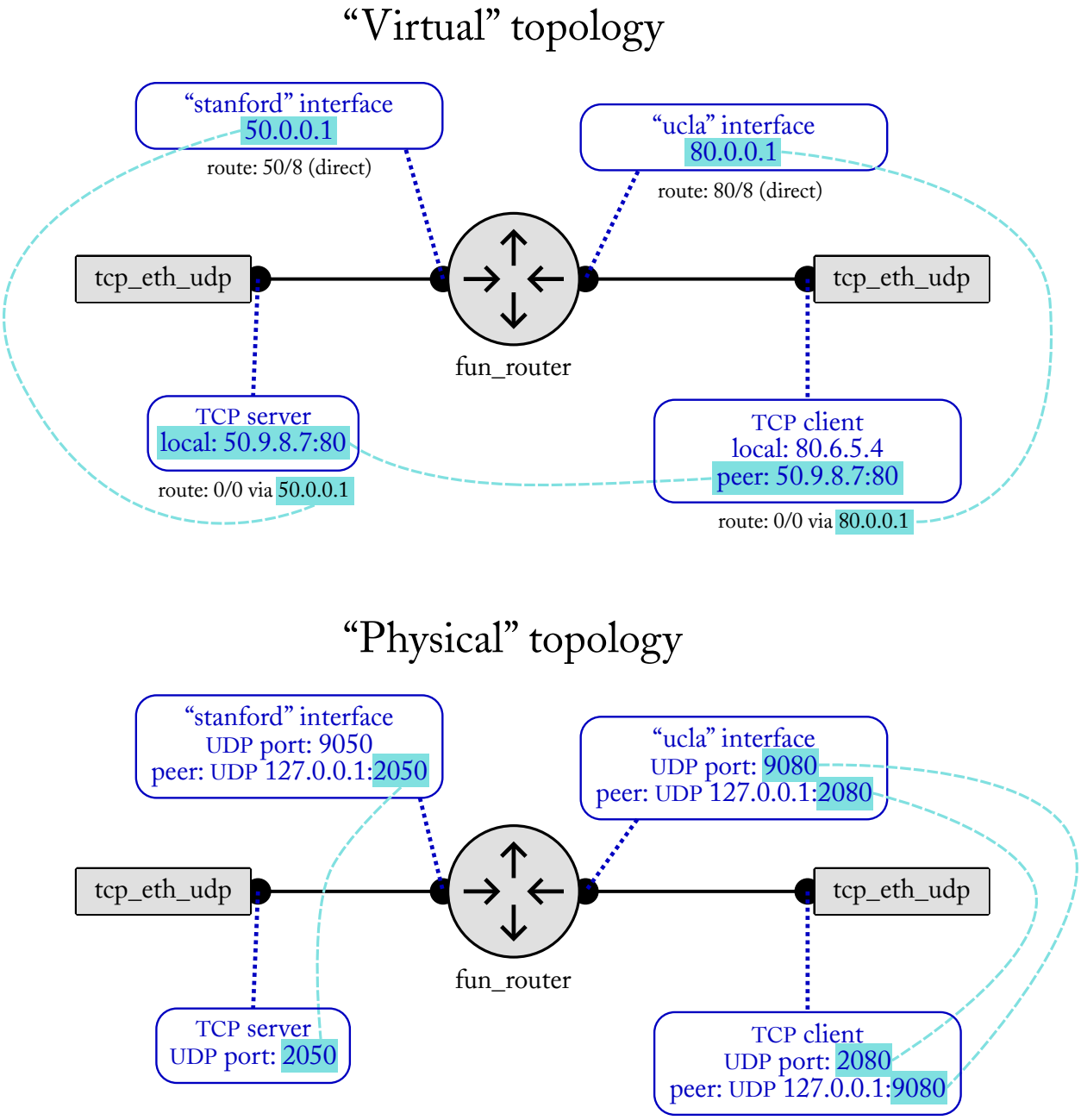
To get started, we're going to recreate the historic 1969 "first ARPANET connection" (the Internet's predecessor) between UCLA and Stanford, inside your computer. You will make a mini-Internet whose **virtual** topology looks like the below, by sending TCP segments between:

- a TCP server at "Stanford" with the IP address 50.9.8.7
- and a TCP client at "UCLA" with the IP address 80.6.5.4

- through a router with two interfaces: one for Stanford's network (with a route pointing to it for the range 50/8) and one for UCLA's network (80/8)

These IP datagrams will be encapsulated inside Ethernet frames that are inside user datagrams that are, in turn, inside "physical" IP datagrams. For now, these "physical" IP datagrams will all stay inside your computer, whose internal IP address is 127.0.0.1.

The complete "virtual" and "physical" addresses will look like the below. Take some time to study this diagram and understand it—it's okay if it seems like a lot at first blush.



To run this topology with the `fun_router` and `tcp_eth_udp` apps, you'll want to open **four** terminals on your VM.

- On the first terminal, run `sudo tshark -ni lo -d udp.port==2080,eth 'port 2080'` to **monitor** the packets in flight. The `-d udp.port==2080,eth` tells `tshark` to interpret the user-datagram payload as Ethernet, possibly with IP and TCP inside it.
- On the second terminal, run

```
 ./build/apps/fun_router \
interface:stanford:50.0.0.1:9050:127.0.0.1:2050 \
interface:ucla:80.0.0.1:9080:127.0.0.1:2080 \
route:50.0.0.0:8:stanford route:80.0.0.0:8:ucla
```

  . This command has all the information (virtual and physical) for the router in the above diagram, including the interface names. You can run `./build/apps/fun_router` by itself to see the help message, which describes the arguments.
- On the third terminal, run the TCP server (at "Stanford"): `./build/apps/tcp_eth_udp server 2050 50.0.0.1 50.9.8.7:80`. (You can run `./build/apps/tcp_eth_udp` by itself to see the help message.)
- On the fourth terminal, run the TCP client (from "UCLA"): `./build/apps/tcp_eth_udp client 2080 127.0.0.1:9080 80.0.0.1 80.6.5.4 50.9.8.7:80`

If everything works as intended, your "tshark" terminal will show an ARP exchange and then a TCP connection establishment. Your "router" terminal will show lots of packets in both directions. Your TCP client and server terminals will show a connection established (as you saw on checkpoint 3).

Try typing "LO" into the UCLA terminal and hitting Enter. Does it get to Stanford?

5 If you have trouble...

- Use the tshark terminal (or wireshark), and the debugging output from the router, to debug.
- Consider building the "sanitizing" (bug-checking) version of the `endtoend` program. It will find many instances of undefined behavior and use of invalid addresses in your code. (See above for directions.)
- Run the entire unit-test suite (including new tests your classmates have contributed this quarter) with `cmake -build build -target test`

6 An Internet of three people

Now that you have a working mini-Internet inside your own computer, try doing the same exercise with two partners, over the CS144 class network. One person should run the "router," one of you runs the TCP client, and the third person runs the TCP server. You may have to rejoin the CS144 class network (use the directions from checkpoint 1, and visit https://cs144.net).

Change the *physical* addresses to match your actual physical addresses (starting with `10.144`). You shouldn't have to change any of the *virtual* addresses.

7 Make a bigger Internet

During the next three weeks and two lab sessions, work to design the **biggest** virtual Internet you can (most routers and hosts). Each student will run a network, and we're hoping many of you will connect to create one big Inter-network. Assign IP blocks to one another, and subnetworks of IP blocks, and set up appropriate routing tables to connect several student-run routers in an "interesting" topology. Arrange for several TCP connections to be occurring at the same time over this network. In your lab report, include a diagram of the Internet's topology with all of the relevant addresses, routing tables, and connections—and the names of the students running each router or endpoint.

8 Something creative

Finally, please take the `webget`, `ip_raw`, `tcp_ipv4`, `tcp_udp_eth`, and `fun_router` programs as a jumping-off point to write a program that does **something creative** that involves computer networking. This could be a tiny game, a chat program, a tiny email client or Web browser, a traceroute/mtr program, a Web server, some kind of crazy router, or anything that inspires you that relates to any of the subject matter of this class.

9 Submit

Write a report in the form of a PDF. Please include:

1. A description of your three-person Internet. Who participated (please include SUnet IDs) in what roles? Were you able to exchange data in all directions? Can you send a 1-MB file and have it arrive exactly the same (using the same procedure as in checkpoint 3)? Did anything interesting happen (or did you find any bugs you had to fix)?
2. A description of your "bigger Internet." Please diagram the topology (hand-drawn and photographed is fine). Include the relevant address ranges, routing tables, and connections, and the names/SUNet IDs of the students running each router or endpoint. What workloads/flows did you run over this network? Were there any surprises?

3. Describe the creative thing you did! Show screenshots or describe what it does, as appropriate. Include an excerpt of the most salient or most interesting *piece of code* that you wrote for this purpose. And... why is this cool (in your view)?
4. Any reflections you have on the class or recommendations for future material to include in the CS144 lab in future years. Do you want to see a checkpoint on congestion control, or real-time audio or video?

The CAs and Keith are eager to read what you all come up with. Thank you for a great quarter, and happy holidays!