Lab 7: building an IP router

0 Collaboration Policy

The programming assignments must be your own work: You must write all the code you hand in for the programming assignments, except for the code that we give you as part of the assignment. Please do not copy-and-paste code from Stack Overflow, GitHub, or other sources. If you base your own code on examples you find on the Web or elsewhere, cite the URL in a comment in your submitted source code.

Working with others: You may not show your code to anyone else, look at anyone else's code, or look at solutions from previous years. You may discuss the assignments with other students, but do not copy anybody's code. If you discuss an assignment with another student,please name them in a comment in your submitted source code. Please refer to the course administrative handout for more details, and ask on Piazza if anything is unclear.

1 Overview

In this week's lab, you'll implement an IP router on top of your existing `NetworkInterface`. A router has *several* network interfaces, and can receive Internet datagrams on any of them. The router's job is to forward the datagrams it gets according to the **routing table**: a list of rules that tells the router, for any given datagram,

•What interface to send it out

•The IP address of the next hop

Your job is to implement a router that can figure out these two things for any given datagram. (You will not need to implement the algorithms that *make* the routing table, e.g. RIP, OSPF,BGP, or an SDN controller—just the algorithm that *follows* the routing table.)

Your implementation of the router will use the Sponge library with a new `Router` class, and tests that will check your router's functionality in a simulated network. Lab 7 builds on your implementation of `NetworkInterface` from Lab 6, but does *not* use the TCP stack you implemented in Labs 0–5. IP routers don't have to know anything about TCP, ARP, or Ethernet (only IP). We expect your implementation will require about **25–30 lines of code**.
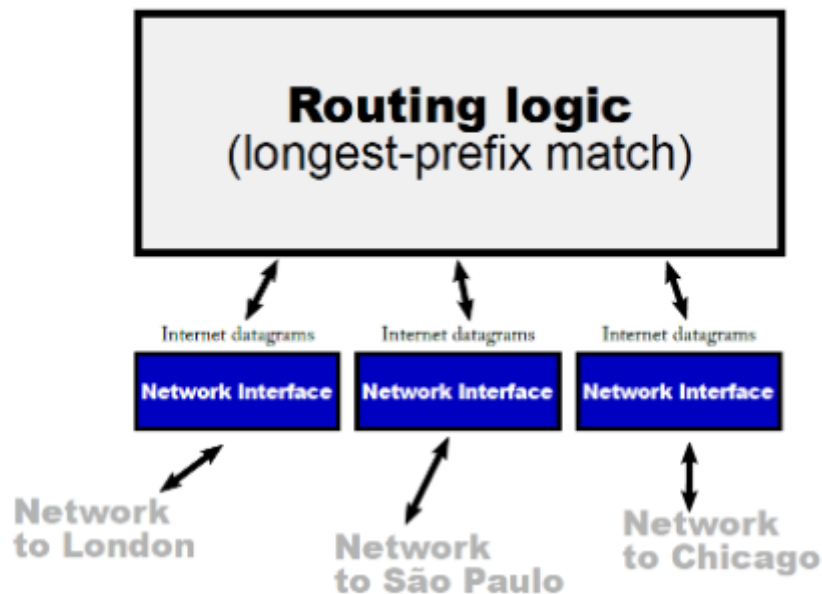
Figure 1: A router contains several network interfaces and can receive IP datagrams on any one of them. The router forwards any datagram it receives to the next hop, on the appropriate outbound interface. The routing table tells the router how to make this decision.

2 Getting started

1. Make sure you have committed all your solutions to Lab 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.) You may have trouble merging the Lab 7 starter code otherwise.
2. Just as Lab0, run `mkdir build`, `cd build`, `cmake ..` and `make` (you can run, e.g., `make -j4` to use four processors when compiling) one by one.
3. Outside the `build` directory, open and start editing the `writeups/lab7.md` file. This is the template for your lab writeup and will be included in your submission.

3 Implementing the Router

In this lab, you will implement a `Router` class that can:

•keep track of a routing table (the list of forwarding rules, or routes), and

•forward each datagram it receives:

  –to the correct next hop

  –on the correct outgoing `NetworkInterface`.

Your implementation will be added to the `router.hh` and `router.cc` skeleton files. Before you get to coding, please review the documentation for the new Router class.

Here are the two methods you'll implement, and what we're expecting in each:

```
   void add_route(const uint32_t route_prefix,

                            const uint8_t prefix_length,

                            const optional<Address> next_hop,

                            const size_t interface_num) ;
```

This method adds a route to the routing table. You'll want to add a data structure as a private member in the `Router` class to store this information. All this method needs to do is save the route for later use.

What do the parts of a route mean?

A route is a "match-action" rule: it tells the router that if a datagram is headed for a particular network (a range of IP addresses), and if the route is chosen as the most specific matching route, then the router should forward the datagram to a particular next hop on a particular interface.

The "match": is the datagram headed for this network? The route_prefix and prefix_length together specify a range of IP addresses (a network) that might include the datagram's destination. The route_prefix is a 32-bit numeric IP address. The prefix_length is a number between 0 and 32 (inclusive); it tells the router how many most-significant bits of the route_prefix are significant. For example, to express a route to the network "18.47.0.0/16" (this matchesany32-bit IP address where the first two bytes are 18 and 47), the route_prefix would be 305070080 ($18 \times 2^{24} + 47 \times 2^{16}$),and the prefix_length would be 16. Any datagram destined for "18.47.x.y" will match.

The "action": what to do if the route matches and is chosen. If the router is directly attached to the network in question, the next_hop will be an empty optional. In that case, the next_hop is the datagram's destination address. But if the router is connected to the network in question through some other router, the next_hop will contain the IP address of the next router along the path. The interface_num gives the index of the router's NetworkInterface that should use to send the datagram to the next hop. You can access this interface with the interface(interface_num) method.

```
   void route_one_datagram(InternetDatagram &dgram) ;
```

Here's where the rubber meets the road. This method needs to route one datagram to the next hop, out the appropriate interface. It needs to implement the "longest-prefix match"logic of an IP router to find the *best* route to follow. That means:

 • The Router searches the routing table to find the routes that match the datagram's destination address. By "match," we mean the most-significant `prefix_length` bits of the destination address are identical to the most-significant `prefix_length` bits of the `route_prefix`.

 • Among the matching routes, the router chooses the route with the *biggest* value of `prefix_length`. This is the **longest-prefix-match** route.

 • If no routes matched, the router drops the datagram.

 • The router decrements the datagram's TTL (time to live). If the TTL was zero already,or hits zero after the decrement, the router should drop the datagram.

 • Otherwise, the router sends the modified datagram on the appropriate interface(`interface(interface_num).send_datagram()`) to the appropriate next hop.

There's a beauty (or at least a successful abstraction) in the Internet's design here: the router never thinks about TCP, about ARP, or about Ethernet frames. The router doesn't even know what the link layer looks like. The router only thinks about Internet datagrams, and only interacts with the link layer through the `NetworkInterface` abstraction. When it comes to questions like, "How are link-layer addresses resolved?"or "Does the link layer even have its own addressing scheme distinct from IP?" or"What's the format of the link-layer frames?" or "What's the meaning of the datagram's payload?", the router just doesn't care.

4 Testing

You can test your implementation by running `make check_lab7`. This will test your router in a particular simulated network, shown in Figure 2.
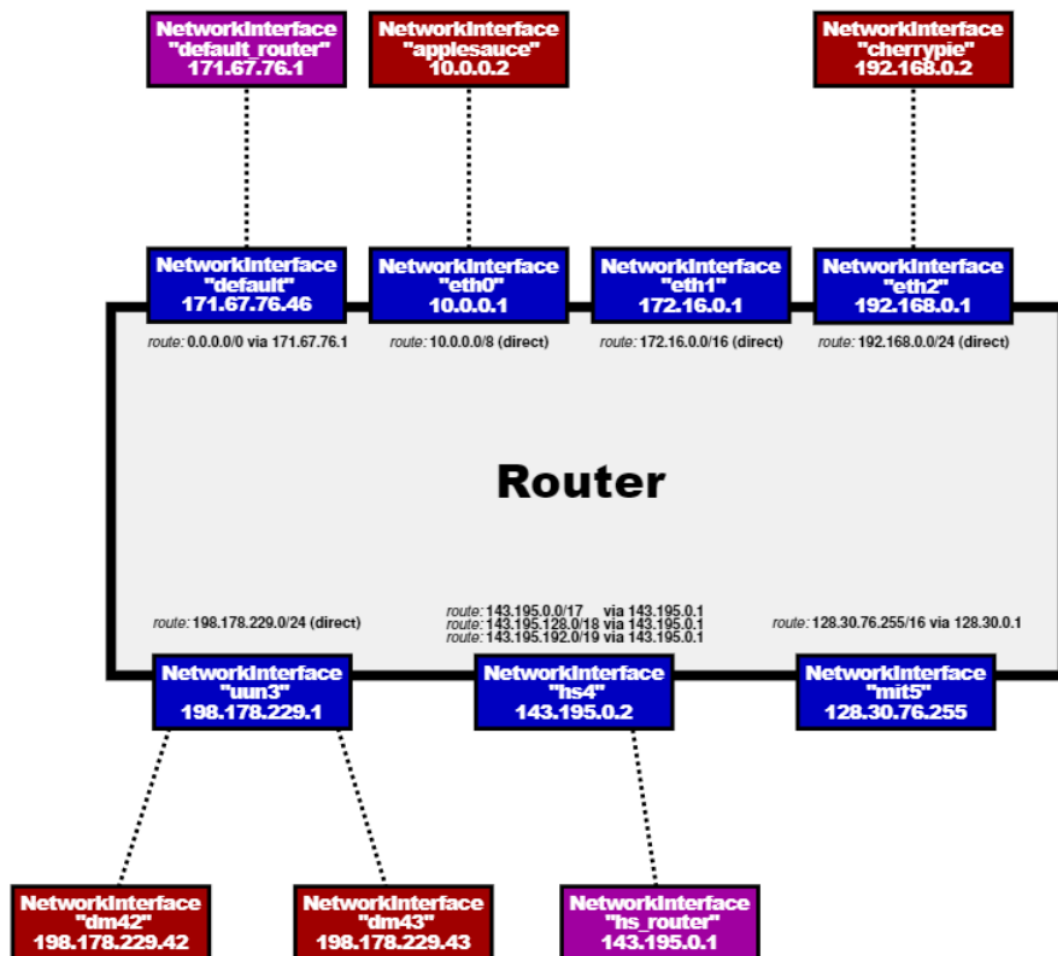

Figure 2: The simulated test network used in the apps/network_simulator tool, also run by `make check_lab7`. (Fun fact: the UUN network is David Mazieres's slice of the Internet, allocated in 1993. The whois tool, or the linked website, can be used to look up who controls each IP address allocation.)

5 Q & A

•*What data structure should I use to record the routing table?*

Up to you! But no need to get crazy. It's perfectly acceptable for each datagram to require *O(N)* work, where *N* is the number of entries in the routing table. If you'd like to do something more efficient, we'd encourage you to get a working implementation first before optimizing, and carefully document and comment whatever you choose to implement.

•How do I convert an IP address that comes in the form of an Address object, into a raw 32-bit integer that I can write into the ARP message?

Use the `Address::ipv4_numeric()` method.

•How do I convert an IP address that comes in the form of a raw 32-bit integer into an Address object?

Use the `Address::from_ipv4_numeric()` method.

•How do I compare the most-significant N bits (where 0≤N≤32) of one 32-bit IP address with the most-significant N bits of another 32-bit IP address?

This is probably the "trickiest" part of this assignment—getting that logic right. It may be worth writing a small test program in C++ (a short standalone program) or adding a test to Sponge to verify your understanding of the relevant C++ operators and double-check your logic.

Recall that in C and C++, it can produce undefined behavior to shift a 32-bit integer by 32 bits. It's a good idea to `make clean` and compile your code with the sanitizers on (`cmake -DCMAKE_BUILD_TYPE=RelASan ..`) to try to catch any undefined behavior in your code before you submit.

You can run the router test directly by running `./apps/network_simulator` from inside the build directory.

•If the router has no route to the destination, or if the TTL hits zero, shouldn't it send an ICMP error message back to the datagram's source?

In real life, yes, that would be helpful. But not necessary in this lab—dropping the datagram is sufficient. (Even in the real world, not every router will send an ICMP message back to the source in these situations.)

•How do I run the test suite for this lab?

`make check_lab7` (two tests). Or you can run the entire test suite with `make check` (161 tests).

6 Submit

1. In your submission, 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 of any of the classes.

2. 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 make sure the automated tests pass)

3. Write a report in `writeups/lab7.md`. This file should be a roughly 20-to-50-line document with no more than 80 characters per line to make it easier to read. The report should contain the following sections:

   a. **Program Structure and Design.** Describe the high-level structure and design choices embodied in your code. You do not need to discuss in detail what you inherited from the starter code. Use this as an opportunity to highlight important design aspects and provide greater detail on those areas for your grading TA to understand. You are strongly encouraged to make this writeup as readable as possible by using subheadings and outlines. Please do not simply translate your program into an paragraph of English.

b. **Implementation Challenges.** Describe the parts of code that you found most troublesome and explain why. Reflect on how you overcame those challenges and what helped you finally understand the concept that was giving you trouble. How did you attempt to ensure that your code maintained your assumptions, invariants, and preconditions, and in what ways did you find this easy or difficult? How did you debug and test your code?

c. **Remaining Bugs.** Point out and explain as best you can any bugs (or unhandled edge cases) that remain in the code.

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

5. When ready to submit, please follow the instructions in Our QQ group. Please make sure you have committed everything you intend before submitting.

6. Please also fill in the number of hours the assignment took you and any other comments. Good luck!

7. Please let the course staff know ASAP of any problems at the Tuesday-evening lab sessions, or by posting a question on Piazza. Good luck!