CSC458/2209 Programming Assignment 2

Winter 2025

1 Routing Packets in the Router

In PA1, you implemented a simplified network interface. Your implementation takes care of encapsulating the datagrams inside appropriate IP packets and Ethernet frames while handling ARP translations when needed. In PA2, you should implement a simplified router that performs one fundamental task: determining the next hop of a packet in its journey to the destination host. This includes finding the best route for the packet among the known routes, and forwarding the packet along the right direction. [1]

Background

The Internet is composed of a series of nodes that are connected to each other. A main design feature of this network is that there is no specific configuration defining how these different nodes are connected. Therefore, one of the main tasks in the network is to find a path for each packet to be delivered to the destination host. This problem is known as the routing problem, which was discussed in the class in detail. Within the network, specialized devices known as routers are used for this purpose.

In this assignment, you will implement a simplified router. A router is typically connected to multiple other routers and sub-networks. Its main job is to find the next hop for each packet that arrives at one of its ports. This decision is made based on a series of rules (known as route entries) that are installed in the router. For each incoming packet, the router goes through its installed route entries to find the best route for that packet, defined as the route that has the best match with the packet's destination. This is usually referred to as "longest prefix matching". Essentially, the router starts by comparing the destination address of the packet with the installed routes. Each route consists of an address and a prefix length. The address is a 32-bit IP address, and the prefix length is a number between 0 and 32. A destination IP address will match a route entry if it most significant bits match the IP address of the route, all the way to the prefix length.

For example, in the case of a prefix length of 8, the 8 most-significant bits of the destination IP address (i.e., the first octet of the IP address) should match the route IP address. Showing a route entry with the [address]/[length] format, the entry **`123.0.0.0/8`** will match destination IP addresses **`123.123.123.123`** and **`123.0.12.54`** but it will not match **`124.0.0.0`** or **`8.0.0.0`**.

The router will search through its routing table to find all matching routes for the destination IP address of the packet. If there is no such route, the packet is dropped. If there is more than one such route, it will select the route with the one with the longest prefix length (hence the name "longest prefix matching"). Therefore, you should be able to perform three tasks to route a packet in a router:

- **Installing a new route**: The router should have a routing table that allows installing new routes.

[1] This assignment is also based on one of Stanford's CS 144 lab assignments.

- **Finding the best matching routes**: The router should be able to search the routing table and find all routes that match the destination IP address of a packet. If more than one route matches the packet, it should select the one with the longest prefix.
- **Send the packet out of the router**: After finding the best route, the router should send the packet to the next hop. This involves examining the route rule to identify the appropriate interface and checking for a specific next hop. If the destination is not within a directly-connected network, the packet should be forwarded to another router, known as the "next hop," as specified in the routing table.

2 Getting Started

This assignment builds upon PA1, and you need a working solution for PA1 to be able to implement this assignment. We will run the submitted files inside the same VM that we provided for you to use. Please use the same VM to test your code before submitting if you are using another development environment, because we cannot support other environments (such as different versions of C++ compilers).

1. First, you should retrieve the latest version of the repository for PA2. You can either:
   - Merge the PA2 starter files into your current clone (from PA1). To do this, first:
     `git fetch --all`
     Then receive the updated files from the **origin/pa2** branch:
     `git merge origin/pa2 --no-edit`
   - If you have don't have access to your previous clone, you can start clean with cloning the starter code repository and switching to the **pa2** branch.
     `git clone https://github.com/yganjali/csc458-pa`
     `git switch pa2`
     Then you should manually copy the **network_interface.cc**, **network_interface.hh** files which you have implemented for your PA1 to the **src** folder. If your PA1 implementation is still incomplete, you should first focus on completing it, before moving on to the new assignment.
2. Either way, now you can verify that your build system is properly set up (don't skip this).
   `cmake -S . -B build`
3. Now you can try to build the starter code:
   `cmake --build build`
4. You are ready to start completing the assignment. Whenever you want to run tests on your code, you can run:
   `cmake --build build --target pa2`

TODO List!

There are 2 functions defined in the **Router** class (**router.cc** and **router.hh** files) that you need to complete:

1. `void Router::add_route( const uint32_t route_prefix, const uint8_t prefix_length, const optional<Address> next_hop, const size_t interface_num )`

   This function is called to install a new route rule in the routing table of the router. This method just adds the route to the routing table. The actual routing will happen in another function. Each route rule has four parts:

   - **route_prefix**: This is the prefix IP address of the rule, which is a 32-bit IP number.
   - **prefix_length**: This is a number between 0 and 32 that defined how many of the most significant bits of the packet's destination IP address should match the rule's IP address.
   - **interface_num**: If this route is the best matching route for a packet, then the packet should be sent on this specific interface number. You can get a reference to the network interface object that represents this interface by calling the `interface(interface_num)` method of the `Router` class. Refer to the class definition in `router.hh` file for more information.
   - **next_hop**: This is optional. If the destination host is in the network that is connected to the router's interface `interface_num`, then the packet can be directly sent to it on that interface. In this case, no `next_hop` value will be provided. If such value is provided, however, the packet should be sent to another router (as the next hop) that has IP address `next_hop`. Note that this does not mean changing the destination IP address of the packet.

2. `void Router::route()`

   This is the function that will process and handle all packets that arrived at the router (through various interfaces) and are waiting to be routed. It will process all such packets and send them appropriately. Here are the tasks that should be done in this function:

   - It should iterate through all network interfaces and process any packet that arrived at that interface and awaits processing.
   - For each such packet, it should find the best route. This means searching through the routing table to find the matching route with the longest prefix.
   - If there are no matching routes for a packet, the router should drop the packet. This means simply doing nothing more and moving on to the next packet.
   - Otherwise, the packet should be prepared for sending out. This means decrementing the TTL field. If the TTL was 0 before doing this or gets down to 0 as a result of this, the packet should be dropped.
   - Then the packet should be sent out on the interface that is specified in the rule. This will be done by calling the `send_datagram` of the corresponding network interface. (recall that this is the same function that you implemented in PA1)

In addition to completing these 2 functions, you can add any extra helper functions or member variables to the `Router` class (e.g., the routing table!) Therefore, you may modify both `router.cc` and `router.hh` files.

A few hints

Here are a few hints that may be useful for you in this assignment:

- You do not need to worry about optimizing your routing table lookup. Although it is a crucial aspect of designing routers (and includes designing special hardware to speed up such lookups), we will have relatively small routing tables in our tests. The goal of this assignment is for you to understand and implement the main idea of longest prefix routing and not optimizing it.
- You may need to convert the IP addresses between the **uint32_t** and **Address** format. You can use the defined helper functions inside the **Address** class for this (**Address::ipv4_numeric()** and **Address::from_ipv4_numeric()** methods).
- You can use arithmetic shift to remove least significant bits that should not be matched from the destination IP address and the prefix IP address before comparing them together. Also note the special case of prefix length 0 and the fact that you cannot shift a 32-bit integer by 32.
- When you drop a packet, there is no need to generate any ICMP message. Although this is done in real world to inform the sender that a packet was dropped, you are not responsible to do it in this simplified assignment.
- Note again that you should have a working solution for PA1 (i.e., **NetworkInterface** class) to be able to pass all tests. So you may want to run the tests for PA1 again before focusing on PA2 tests just to be sure.

3 Submission

We will use the MarkUs submission system for this assignment as well. You need to submit the four source code files for the **NetworkInterface** and **Router** classes (**network_interface.cc**, **network_interface.hh**, **router.cc**, and **router.hh**). You should also submit the **writeups/pa2.md** file that contains a few questions that you should answer about the assignment. In addition to checking your submission with automated tests (to test its functionality), TAs will read your source code and assess its coding style. The marking breakdown is similar to PA1:

- 90% of the assignment mark is for the automated tests (i.e., correct functionality). This 90% is divided into:
  - 50% that can be acquired by passing the publicly available tests (already part of the starter code, under **tests** directory),
  - Another 40%, that will be dedicated to private tests. This is to make sure your code doesn't run correctly only in the provided scenarios.
- 5% of the mark is for providing reasonable answers to the questions found in **pa2.md**.
- The remaining 5% is awarded for your coding style. The coding style includes factors such as inline comments, proper variable names, breaking complex functions into smaller functions, preventing duplicate code by defining helper functions, etc. You should be familiar with such marking from previous programming courses.

Some Notes

- **Important note**: Although it is advised to backup your work periodically using some version control method, **you must not make your solutions publicly available.** Make sure to keep your repositories private.
- This assignment is to be done individually. Our sample solution added around 100 lines of code to the starter code to pass all the tests.