Lab Checkpoint 5: down the stack (the network interface)

Due: Friday, Nov. 19, 5 p.m.
Late deadline: Nov. 21, 11:59 p.m.

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.

Piazza: Please feel free to ask questions on Piazza, but please don't post any source code.

1 Overview

也就是说我们前面是自底向上实现TCP协议，现在我们需要关注更底层的东西？

In this week's lab, you'll go down the stack and implement a network interface: the bridge between Internet datagrams that travel the world, and link-layer Ethernet frames that travel one hop. This component can fit "underneath" your TCP/IP implementation from the earlier labs, but it will also be used in a different setting: when you build a router in Lab 6, it will route datagrams *between* network interfaces. Figure 1 shows how the network interface fits into both settings.

In past labs, you wrote a TCP implementation that can successfully exchange **TCP segments** with any other computer that speaks TCP. How are these segments actually conveyed to the peer's TCP implementation? As we've discussed, there are a few options:

现在我们要深究底层了

- **TCP-in-UDP-in-IP.** The TCP segments can be carried in the payload of a user datagram. When working in a normal (user-space) setting, this is the easiest to implement: Linux provides an interface (a "datagram socket", `UDPSocket`) that lets applications supply *only the payload* of a user datagram and the target address, and the kernel takes care of constructing the UDP header, IP header, and Ethernet header, then sending the packet to the appropriate next hop. The kernel makes sure that each socket has an exclusive combination of local and remote addresses and port numbers, and since the kernel is the one writing these into the UDP and IP headers, it can guarantee isolation between different applications.

也许在web中，TCP carry的payload就是HTTP请求

UDP, User Datagram Protocol

这里简要介绍了下TCP-UDP-IP的实现：
TCP不由操作系统的内核实现，而是运行在用户态。用户会将数据封装在TCP segment中。操作系统会提供一个接口，用户只需传入TCP segment以及目的地址进入该接口就行。
在此接口中，操作系统会给用户传进来的数据报文增加各种协议头，添加端口号等等等。
所以说，其实操作系统只支持UDP-IP-ETH呀！新加的一层反倒是靠用户实现的！如果只需UDP，用户只需传入数据就行；如果还想要TCP，那就得由用户自己把数据封装成TCP报文。

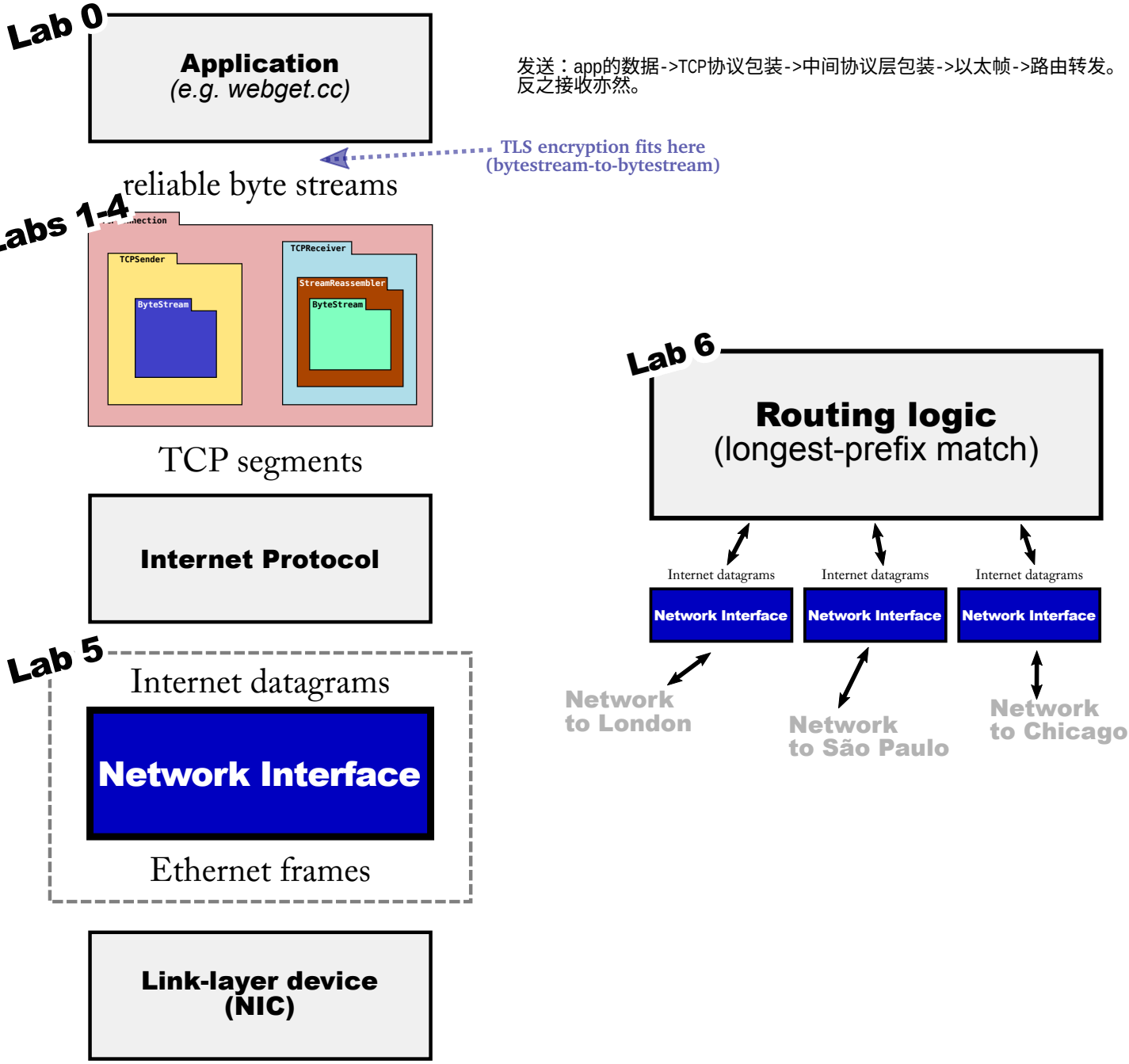

Figure 1: The network interface bridges the worlds of Internet datagrams and of link-layer frames. This component is useful as part of a host's TCP/IP stack (left side), and also as part of an IP router (right side).

TCP也可以直接与IP层相连。但这由于不用了操作系统提供的那一层UDP，我们要做的事情就会变多变复杂。

这个模式正是我们在lab0-4做的

TCP一般直接与IP相连而非经由UDP

- **TCP-in-IP.** In common usage, TCP segments are almost always placed directly inside an Internet datagram, without a UDP header between the IP and TCP headers. This is what people mean by "TCP/IP." This is a little more difficult to implement. Linux provides an interface, called a TUN device, that lets application supply an *entire* Internet datagram, and the kernel takes care of the rest (writing the Ethernet header, and actually sending via the physical Ethernet card, etc.). But now the application has to construct the full IP header itself, not just the payload.

这里也向我们说明了一下lab4的测试原理，也即采用TCP-IP模式，这些文件是用来把TCP封装为IP并且序列化的，然后之后传入到操作系统给的TUN接口就行。之后可以研究一下源码。

我怎么不知道。

You've done this already. In Lab 4, we gave you an object that represents an Internet datagram and knows how to parse and serialize itself (tcp_helpers/ipv4_datagram.{hh,cc}) and the logic to encapsulate TCP segments in IP (now found in tcp_helpers/tcp_over_ip.cc). The `CS144TCPSocket` uses these tools to connect your `TCPConnection` to a TUN device.

我曹，所以我们的目的是不靠内核了，全由我们自己来？

- **TCP-in-IP-in-Ethernet.** In the above approach, we're still relying on the Linux kernel for part of the networking stack. Each time your code writes an IP datagram to the TUN device, Linux has to construct an appropriate link-layer (Ethernet) frame with the IP datagram as its payload. This means Linux has to figure out the next hop's Ethernet destination address, given the IP address of the next hop. If it doesn't know this mapping already, Linux broadcasts a query that asks, "Who claims the following IP address? What's your Ethernet address?" and waits for a response.

ARP

这不挺好的？难道全部在用户态实现就会性能变强吗？我不大明白为什么要选择在用户态实现，只是为了学习吗？

也即路由地址转发、自学习，IP->ETH的这些函数，也即我们之后要实现的函数

These functions are performed by the *network interface*: a component that translates outbound IP datagrams into link-layer (e.g., Ethernet) frames and vice versa. (In a real system, network interfaces typically have names like `eth0`, `eth1`, `wlan0`, etc.) **In this week's lab**, you'll implement a network interface, and stick it at the very bottom of your TCP/IP stack. Your code will produce raw Ethernet frames, which will be handed over to Linux through an interface called a TAP device—similar to a TUN device, but more low-level, in that it exchanges raw link-layer frames instead of IP datagrams.

根据IP地址查找物理地址。有一个自学习过程

Most of the work will be in looking up (and caching) the Ethernet address for each next-hop IP address. The protocol for this is called the **Address Resolution Protocol**, or **ARP**.

We've given you unit tests that put your network interface through its paces. Then, at the end of this lab, you'll slightly modify your `webget` to use your TCP implementation, *running on your network interface*, so that the whole thing produces raw Ethernet frames and can still talk to a real webserver across the Internet. In Lab 6, you'll use the same network interface outside the context of TCP, as a part of an IP router.

所以我们只是做了由IP地址找到物理地址的过程，具体的路由实现还是由操作系统吗？

对于地址的这个传递过程，刚刚有点乱现在梳理一下。
我们通过约定俗成的域名访问某个主机，该域名会被DNS解析为IP地址，该IP地址会在网络层，也就是IP协议这边经过路由转发，填上下一站要去的地方的IP地址【如果是用户host的话会发给自己的default路由器】，然后在ETH层会根据这个下一站要去的地方的IP地址通过ARP协议得到下一站要去的物理地址。

2 Getting started

所以说，我们lab6就是要做的这种转发功能。它之所以说In Lab 6, you'll use the same network interface outside the context of TCP, as a part of an IP router.
是因为只有路由器起着真实的路由转发功能【host的路由转发只是转发给自己的海关路由器】。而路由器不需要什么TCP协议，所以lab6只基于lab5，跟lab0-4没有半毛钱关系，所以它才说"outside the context of TCP"。

1. Make sure you have committed all your solutions to Lab 4. 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 5 starter code otherwise.

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

总之，梳理了这么多，核心就是，lab5要实现以太网层，我们要用TCP-IP-ETH结构，TCP已经实现完了，IP已经给我们了，现在就只用写ETH就行。写ETH需要接触ARP。
lab6要写网络层的路由转发，由于可以看作只在路由器有用，因而只需要IP-ETH就行，TCP跟我们没关系了

3. Download the starter code for Lab 5 by running `git merge origin/lab5-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/lab5.md` file. This is the template for your lab writeup and will be included in your submission.

3 The Address Resolution Protocol ARP

Before you get to coding, please read:

- The public interface for the `NetworkInterface` object.
- Wikipedia's summary of ARP and the original ARP specification (RFC 826).
- The documentation/implementation of the EthernetFrame and EthernetHeader objects.
- The documentation and implementation of the IPv4Datagram, and IPv4Header objects (which can parse and serialize an Internet datagram, and when serialized, can be assigned to the payload of an Ethernet frame).
- The documentation and implementation of the ARPMessage object (which knows how to parse and serialize an ARP message, and can also serve as the payload of an Ethernet frame when serialized).

Your main task in this lab will be to implement the three main methods of `NetworkInterface` (in the `network_interface.cc` file), maintaining a mapping from IP addresses to Ethernet addresses. The mapping is a cache, or "soft state": the NetworkInterface keeps it around for efficiency's sake, but if it has to restart from scratch, the mapping will naturally be regenerated without causing a problem.

1. `void NetworkInterface::send_datagram(const InternetDatagram &dgram, const Address &next_hop)`

   也就是说是来自上层调用，这个会接收上层协议栈的报文

   This method is called when the caller (e.g., your `TCPConnection` or a router) wants to send an outbound Internet (IP) datagram to the next hop.[1] It's your interface's job to translate this datagram into an Ethernet frame and (eventually) send it.

   - *If the destination Ethernet address is already known*, send it right away. Create an Ethernet frame (with `type = EthernetHeader::TYPE_IPv4`), set the payload to be the serialized datagram, and set the source and destination addresses.

   所以需要调用序列化方法

[1] Please don't confuse the *ultimate* destination of the datagram, which is in the datagram's own header as the destination address, with the next hop. In this lab you're *only* going to care about the next hop's address.

- *If the destination Ethernet address is unknown*, broadcast an ARP request for the next hop's Ethernet address, and queue the IP datagram so it can be sent after the ARP reply is received.

也许我们可以放入一个以目标IP地址为key的map中

防止多播泛滥

Except: You don't want to flood the network with ARP requests. If the network interface already sent an ARP request about the same IP address in the last five seconds, don't send a second request—just wait for a reply to the first one. Again, queue the datagram until you learn the destination Ethernet address.

2. `optional<InternetDatagram> NetworkInterface::recv_frame(const EthernetFrame &frame)`

   This method is called when an Ethernet frame arrives from the network. The code should ignore any frames not destined for the network interface (meaning, the Ethernet destination is either the broadcast address or the interface's own Ethernet address stored in the `_ethernet_address` member variable).

   - *If the inbound frame is IPv4*, parse the payload as an `InternetDatagram` and, if successful (meaning the `parse()` method returned `ParseResult::NoError`), return the resulting `InternetDatagram` to the caller.
   - *If the inbound frame is ARP*, parse the payload as an `ARPMessage` and, if successful, remember the mapping between the sender's IP address and Ethernet address for 30 seconds. (Learn mappings from both requests and replies.) In addition, if it's an ARP request asking for our IP address, send an appropriate ARP reply.

所以我们的ARP结构应该是，IP地址为key，value为包含MAC地址和ticks的结构体

除了做这些事，如果是ARP response的话，也许还得从等待队列里发出去

3. `void NetworkInterface::tick(const size_t ms_since_last_tick)`

   This is called as time passes. Expire any IP-to-Ethernet mappings that have expired.

You can test your implementation by running `ctest -V -R "^arp"`**.** This test does not rely on your TCP implementation.

4 webget re^{2}visited

Remember your `webget.cc` that you wrote in Lab 0 (using the Linux-provided TCP implementation in `TCPSocket`)? And, remember how you modified it in Lab 4 to use your own TCP-in-IP implementation in `CS144TCPSocket`? As we discussed above, this still relies on the Linux kernel for part of the stack: the network interface that translates between IP and the link layer (Ethernet).

We'd like you to switch it to use your network interface without changing anything else. All you'll need to do is replace the `CS144TCPSocket` type with `FullStackSocket`.

This uses a TCP-in-IP-in-Ethernet stack as shown in a Figure 1 (left side): your `webget.cc` application, on top of your `TCPConnection` implementation of TCP, on top of the TCP-in-IP code in tcp_helpers/tcp_over_ip.cc, on top of your `NetworkInterface`.

Recompile, and run `make check_lab5` to confirm that you've gone full-stack: you've written a basic Web fetcher on top of *your own complete TCP implementation* and *your own network interface implementation*, and it still successfully talks to a real webserver.

If you have trouble, try running the program manually:
`./apps/webget cs144.keithw.org /hasher/xyzzy`, and try capturing what it's sending and receiving by using `wireshark`. You can save the packets it's sending and receiving by running `sudo tcpdump -i tap10 -w /tmp/packets.tap`. Then open the `/tmp/packets.tap` file in `wireshark`.

5 Q & A

- *How much code are you expecting?*

  Overall, we expect the implementation (in `network_interface.cc`) will require about 100–150 lines of code in total.

- *How does the NetworkInterface actually send an Ethernet frame?*

  Similar story to how the `TCPSender` and `TCPConnection` worked. For the `NetworkInterface`, push any outbound frame to the `_frames_out` queue and it will be popped and sent by the object's owner.

- *What data structure should I use to record the mapping between next-hop IP address and Ethernet addresses?*

  Up to you!

- *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.

- *What should I do if the NetworkInterface sends an ARP request but never gets a reply? Should I resend it after some timeout? Signal an error to the original sender using ICMP?* 问得好

  In real life, yes, both of those things, but don't worry about that in this lab. (In real life, an interface will eventually send an ICMP "host unreachable" back across the Internet to the original sender if it can't get a reply to its ARP requests.)

- *What should I do if an InternetDatagram is queued waiting to learn the Ethernet address of the next hop, and that information never comes? Should I drop the datagram after some timeout?*

  Again, definitely a "yes" in real life, but don't worry about that in this lab.

- *How do I run the test suite for this lab?*

  `make check_lab5` (two tests). Or you can run the entire test suite with `make check` (160 tests).

- *Where can I read if there are more FAQs after this PDF comes out?*

  Please check the website (https://cs144.github.io/lab_faq.html) and Piazza regularly.

6 Submit

1. In your submission, please only make changes to the `.hh` and `.cc` files in the top level of `libsponge`, and `apps/webget.cc`. 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. Please don't add extra files—the automatic grader won't look at them and your code may fail to compile.

3. **Please don't forget** to change your `webget` to use a `FullStackSocket` (instead of the `CS144TCPSocket` that you used in Lab 4, or the kernel's `TCPSocket` in Lab 0).

4. 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_lab5` (to make sure the automated tests pass)

5. Write a report in `writeups/lab5.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.

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

7. When ready to submit, please follow the instructions at https://cs144.github.io/submit. Please make sure you have committed everything you intend before submitting.

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