CS 4700 / CS 5700 - Network Fundamentals

Project 4: Raw Sockets

The final version of the project is due at 11:59pm on April 5, 2022.

Description

The goal of this assignment is to familiarize you with the low-level operations of the Internet protocol stack. Thus far, you have had experience using sockets and manipulating application level protocols. However, working above the operating system's networking stack hides all the underlying complexity of creating and managing IP and TCP headers.

Your task is to write a program called *rawhttpget* that takes a URL on the command line and downloads the associated file. You may use any HTTP code that you wrote for project 2 to aid in the process. However, **your program must use an SOCK_STREAM/IPPROTO_RAW socket**, which means that **you** are responsible for building the IP and TCP headers in each packet. In essence, you will be rebuilding the operating system's TCP/IP stack within your application.

WARNING: DO NOT TEST YOUR PROGRAM ON PUBLIC WEBSITES

As with project 2, you should not test your program against public web servers. **Only test your program against pages hosted by CCIS**. When you are testing your program, you will almost certainly send packets with invalid IP and TCP headers. These packets may trigger security warnings if you send them to public websites, e.g. the website adiinistrator may think someone is trying to hack their site with spoofed packets.

You may test your program against Fakebook, or against this assignment page. We have also created some large files full of random bytes that you can use to stress-test your implementation: 2 MB file (2MB.log), 10 MB file (10MB.log), 50 MB file (50MB.log).

High Level Requirements

You goal is to write a program called *rawhttpget* that takes one command line parameter (a URL), downloads the associated web page or file, and saves it to the current directory. The command line syntax for this program is:

./rawhttpget [URL]

An example invocation of the program might look like this:

./rawhttpget http://david.choffnes.com/classes/cs4700sp22/project4.php

This would create a file named *project4.php* in the current directory containing the downloaded HTML content. If the URL ends in a slash ('/') or does not include any path, then you may use the default filename *index.html*. For example, the program would generate a file called *index.html* if you ran the following command:

./rawhttpget http://www.ccs.neu.edu

The file created by your program should be exactly the same as the original file on the server. You can test whether your file and the original are identical by using *wget* or a web browser to download the file, and then comparing the file generated by your program and the original using *md5sum* or *diff*. Do not include extra info in your generated file, like HTTP headers or line breaks.

Since the point of this assignment is not to focus on HTTP, there are many things your program does not need to handle. Your program does not need to support HTTPS. Your program does not need to follow redirects, or handle HTTP status codes other than 200. In the case of a non-200 status code, print an error to the console and close the program. Your program does not need to follow links or otherwise parse downloaded HTML.

Low Level Requirements

The primary challenge of this assignment is that you **must** use raw sockets. A raw socket is a special type of socket that bypasses some (or all) of the operating system's network stack. For example, in C a socket of type SOCK_STREAM/IPPROTO_RAW bypasses the operating system's IP and TCP/UDP layers. In your program, you will need to create two raw sockets: one for receiving packets and one for sending packets. The receive socket must be of type SOCK_RAW/IPPROTO_TCP; the send socket must be of type SOCK_RAW/IPPROTO_RAW. The reason you need two sockets has to do with some quirks of the Linux kernel. The kernel will not deliver any packets to sockets of type SOCK_STREAM IPPROTO_RAW, thus your code will need to bind to the IPPROTO_IP interface to receive packets. However, since you are required to implement TCP and IP, you must send on a SOCK_RAW/IPPROTO_RAW socket.

There are many tutorials online for doing raw socket programming. I recommend Silver Moon's tutorial (http://www.binarytides.com/raw-sockets-c-code-linux/) as a place to get started. That tutorial is in C; Python also has native support for raw socket programming. However, **not all languages support raw socket programming**. Since many of you program in Java, I will allow the use of the RockSaw Library (http://www.savarese.com/software/rocksaw/), which enables raw socket programming in Java.

When you start to write your program, you will immediately notice that the SOCK_STREAM /IPPROTO_IP receive socket is *promiscuous*: it receives **all** packets that are being sent to your machine, regardless of whether they are TCP or UDP, their destination port number, etc. One of your tasks will be filtering the incoming packets to isolate the ones that belong to your program. All other packets can be ignored by your program.

Your program must implement all features of IP packets. This includes validating the checksums of incoming packets, and setting the correct version, header length and total length, protocol identifier, and checksum in each outgoing packet. Obviously, you will also need to correctly set the source and destination IP in each outgoing packet. You may use existing OS APIs to query for the IP of the remote HTTP server (i.e. handle DNS requests) as well as the IP of the source machine. **Be careful that you select the correct IP address of the local machine.** Do not bind to localhost (127.0.0.1)! Furthermore, your code must be defensive, i.e. you must check the validity of IP headers from the remote server. Is the remote IP correct? Is the checksum correct? Does the protocol identifier match the contents of the encapsulated header?

Your code must implement a subset of TCP's functionality. Your program must verify the checksums of incoming TCP packets, and generate correct checksums for outgoing packets. Your code must select a valid local port to send traffic on, perform the three-way handshake, and correctly handle connection teardown. Your code must correctly handle sequence and acknowledgement numbers. Your code may manage the advertised window as you see fit. Your code must include basic timeout functionality: if a packet is not ACKed within 1 minute, assume the packet is lost and retransmit it. Your code must be able to receive out-of-order incoming packets and put them back into the correct order before delivering them to the higher-level, HTTP handling code. Your code should identify and discard duplicate packets. Finally, your code must implement a basic congestion window: your code should start with cwnd=1, and increment the cwnd after each succesful ACK, up to a fixed maximum of 1000 (e.g. cwnd must be <=1000 at all times). If your program observes a packet drop or a timeout, reset the cwnd to 1.

As with IP, your code must be defensive: check to ensure that all incoming packets have valid checksums and in-order sequence numbers. If your program does not receive any data from the remote server for three minutes, your program can assume that the connection has failed. In this case, your program can simply print an error message and close.

Developing Your Program

Access to raw sockets requires root privileges on the operating system. Recall that raw sockets are promiscuous, i.e. they can observe all packets that arrive at a machine. It would be a security vulnerability if any program could open raw sockets, because that would enable you to spy on the network traffic of all other users using a shared machine (e.g. one of the CCIS machines).

Since we cannot give you root access to the CCIS machines, you will need to develop your program on your own Linux machine, or in a VM. We will be grading your code on a stock Ubuntu Linux 20.04 machine, so keep that in mind when developing your code and setting up your VM. **Do not develop your program on Windows or OSX**: the APIs for raw sockets on those systems are incompatible with Linux, and thus your code will not work when we grade it.

For most of you, the VM option will probably be easiest. There are many tutorials (https://ubuntu.tutorials24x7.com/blog/how-to-install-ubuntu-20-04-lts-on-windows-using-vmware-workstation-player) on how to do this. If you use Windows, you will need a (free) copy of VMWare Player, as well as an ISO of Ubuntu. Once you have your VM set up, you will need to install development tools. Exactly what you need will depend on what language you want to program in. There are ample instructions online explaining how to install gcc, Java, and Python-dev onto Ubuntu.

Modifying IP Tables

Regardless of whether you are developing on your own copy of Linux or in a VM, you will need to make one change to *iptables* in order to complete this assignment. You must set a rule in *iptables* that drops outgoing TCP RST packets, using the following command:

```
% iptables -A OUTPUT -p tcp --tcp-flags RST RST -j DROP
```

To understand why you need this rule, think about how the kernel behaves when it receives unsolicited TCP packets. If your computer receives a TCP packet, and there are no open ports waiting to receive that packet, the kernel generates a TCP RST packet to let the sender know that the packet is invalid. However, in your case, your program is using a raw socket, and thus the kernel has no idea what TCP port you are using. So, the kernel will erroneously respond to packets destined for your program with TCP RSTs. You don't want the kernel to kill your remote connections, and thus you need to instruct the kernel to drop outgoing TCP RST packets. You will need to recreate this rule each time your reboot your machine/VM.

Debugging

Debugging raw socket code can be very challenging. You will need to get comfortable with Wireshark (http://www.wireshark.org/) in order to debug your code. Wireshark is a packet sniffer, and can parse all of the relevent fields from TCP/IP headers. Using Wireshark, you should be able to tell if you are formatting outgoing packets correctly, and if you are correctly parsing incoming packets.

Language

You can write your code in whatever language you choose, as long as your code compiles and runs on a **stock** copy of Ubuntu 20.04 **on the command line**.

Be aware that many languages do not support development using raw sockets. I am making an explicit exception for Java, allowing the use of the RockSaw library. If you wish to program in a language (other than Java) that requires third party library support for raw socket programming, **ask me for permission** before you start development.

As usual, do not use libraries that are not installed by default on Ubuntu 20.04 (with the exception of RockSaw). Similarly, your code must compile and run on the command line. You may use IDEs (e.g. Eclipse) during development, but do not turn in your IDE project without a Makefile. Make sure you code has **no dependencies** on your IDE.

Submitting Your Project

Before turning in your project, you and your partner(s) must register your group. To register yourself in a group, execute the following script:

```
$ /course/cs5700sp22/bin/register project4 [team name]
```

This will either report back success or will give you an error message. If you have trouble registering, please contact the course staff. **You and your partner(s) must all run this script with the same [team name]**. This is how we know you are part of the same group.

To turn-in your project, you should submit your (thoroughly documented) code along with three other files:

- A Makefile that compiles your code.
- A plain-text (no Word or PDF) README file. In this file, you should briefly describe your high-level approach, what TCP/IP features you implemented, and any challenges you faced. **You must also include a detailed description of which student worked on which part of the code.**
- If your code is in Java, you must include a copy of the RockSaw library.

Your README, Makefile, source code, external libraries, etc. should all be placed in a directory. You submit your project by running the turn-in script as follows:

```
$ /course/cs5700sp22/bin/turnin project4 [project directory]
```

[project directory] is the name of the directory with your submission. The script will print out every file that you are submitting, so make sure that it prints out all of the files you wish to submit! **Only one group member needs to submit your project.** Your group may submit as many times as you wish; only the last submission will be graded, and the time of the last submission will determine whether your assignment is late.

Grading

This project is worth 24 points (for CS 4700 students, scale points up to 30). You will receive full credit if 1) your code compiles, runs, and correctly downloads files over HTTP, 2) you have not used any illegal libraries, and 3) you use the correct type of raw socket. All student code will be scanned by plagarism detection software to ensure that students are not copying code from the Internet or each other.

8 points will be awarded for each of the three protocols you must implement, i.e. 8 points for HTTP, 8 ponts for TCP, and 8 points for IP. 1 point will be deducted for poor documentation. Essentially, 8 points should be easy to earn; the other 16 are the challenge.

Extra Credit

There is an opportunity to earn 4 extra credit points on this assignment. To earn these points, you must use and AF_PACKET raw socket in your program, instead of a SOCK_RAW/IPPROTO_RAW socket. An AF_PACKET raw socket bypasses the operating systems layer-2 stack as well at layers 3 and 4 (TCP/IP). This means that your program must build Ethernet frames for each packet, as well as IP and TCP headers. You can assume that we will only test your code on machines with Ethernet connections, i.e. you do not need to worry about alternative layer-2 protocols like Wifi or 3G. This extra credit will be quite challenging, since it will involve doing MAC resolution with ARP requests. We have not discussed ARP in class, and you will need to learn about and handle this protocol on your own. Essentially, the challenge is to figure out the MAC address of the gateway, since this information needs to be included in the Ethernet header.

If you complete the extra credit, make sure to mention this in your README. Explain how you implemented Ethernet functionality, and any additional challenges your faced (e.g. ARP).

David Choffnes, Associate Professor, Khoury College of Computer Sciences, Northeastern University.
Last updated February 28, 2022.