**CS331 Project 1 Write Up – Ben Webb**

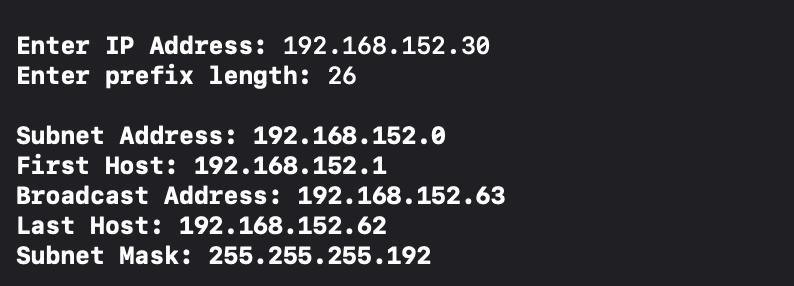
The purpose of this project was to create a program that could calculate IPv4 subnets by taking in the IPv4 address as well as the prefix length.

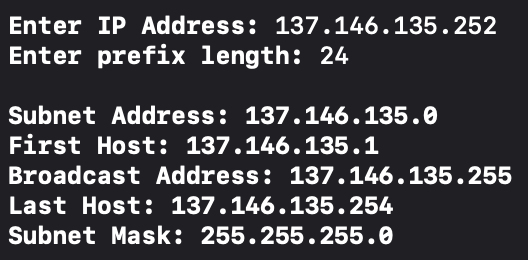
I decided to solve this problem in C++ by parsing an IPv4 char into bytes and using bitwise comparisons with the netmask that was created from the prefix length. The char was parsed using the command: sscanf(ip, “%hu.%hu.%hu.%hu”, &a, &b, &c, &d) which I found on the [StackOverflow](https://stackoverflow.com/a/10366772). ip is a char that is parsed into 4 unsigned shorts: a, b, c, and d.

To create the subnet mask, I also created 4 integer mask variables to pair with the 4 IPv4 bytes: ma, mb, mc, and md. Each byte of the mask was calculated to find the number of left raised bits using the general formula: mx = 256 – pow(2, 8 – std::min(prefix – x \* 8,8)). I first found how many bits the prefix raised of the IPv4byte (prefix - x \* 8). With x representing the successive bytes. I also created a ceiling of 8 for the maximum number of bits that can be raised in one byte. In order to left shift the bits, I then found the total number of bits raised (8 – std::min(prefix – x \* 8,8)). Using this, I found the decimal number corresponding to the number of bits using the pow() method. I finally, left shifted the byte by subtracting the decimal number from 256 resulting in a byte wise left shifted netmask.

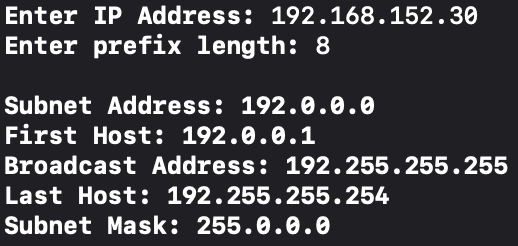
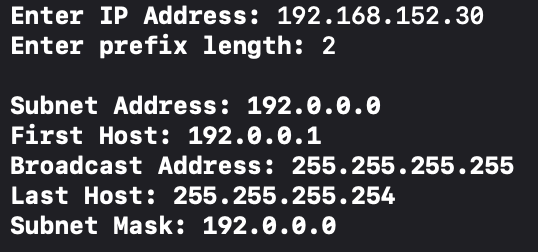
The subnet address was found by doing a bitwise comparison for each IPv4 and mask byte (a & ma, etc.). The first host was found by finding the address one larger than the subnet address in the final byte ((d & md) + 1). The broadcast address was found by finding the largest possible IPv4 address from the subnet address. This was done by inverting the last byte mask against a completely raised byte (255 ^ md). The last host was found by finding the address one smaller than the broadcast address ((255 ^ md) - 1).

The program prompts for an IPv4 address and a prefix length from which it will produce a fully calculated IPv4 subnet. Below is a demonstration of finding the IPv4 subnet of 192.168.152.30/26 as well as my IP address from the problem set of 137.146.135.252/24.

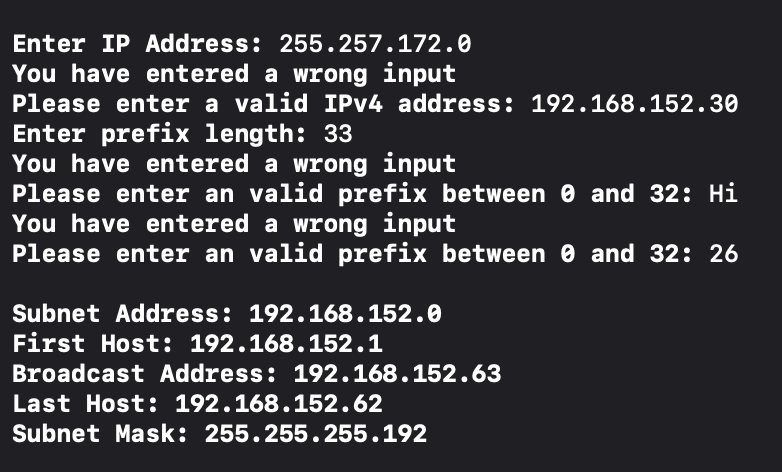




The screenshot below also demonstrates the compatibility to find the subnet when the prefix results in multiple bytes existing within the subnet.

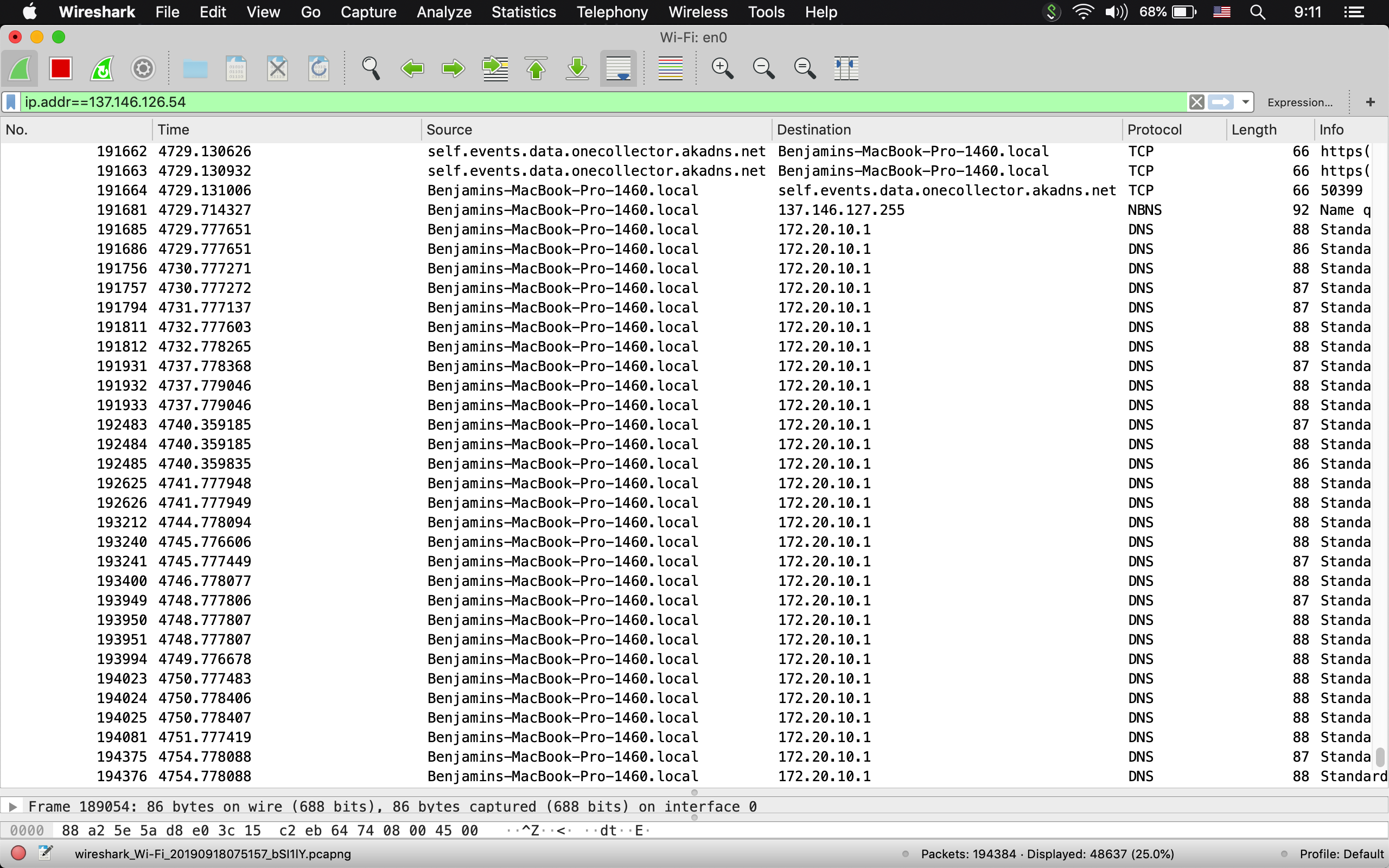
The command std::cin >> prefix at the end of the body of code was used to allow for readable printing into the terminal by preventing the exit message from appearing. Error checking for valid IPv4 and prefixes was derived from [hackerearth](https://www.hackerearth.com/practice/notes/validating-user-input-in-c/). Below is a demonstration of an invalid IP address as well as an invalid prefix, eventually producing the IPv4 subnet of 192.168.152.30/26.

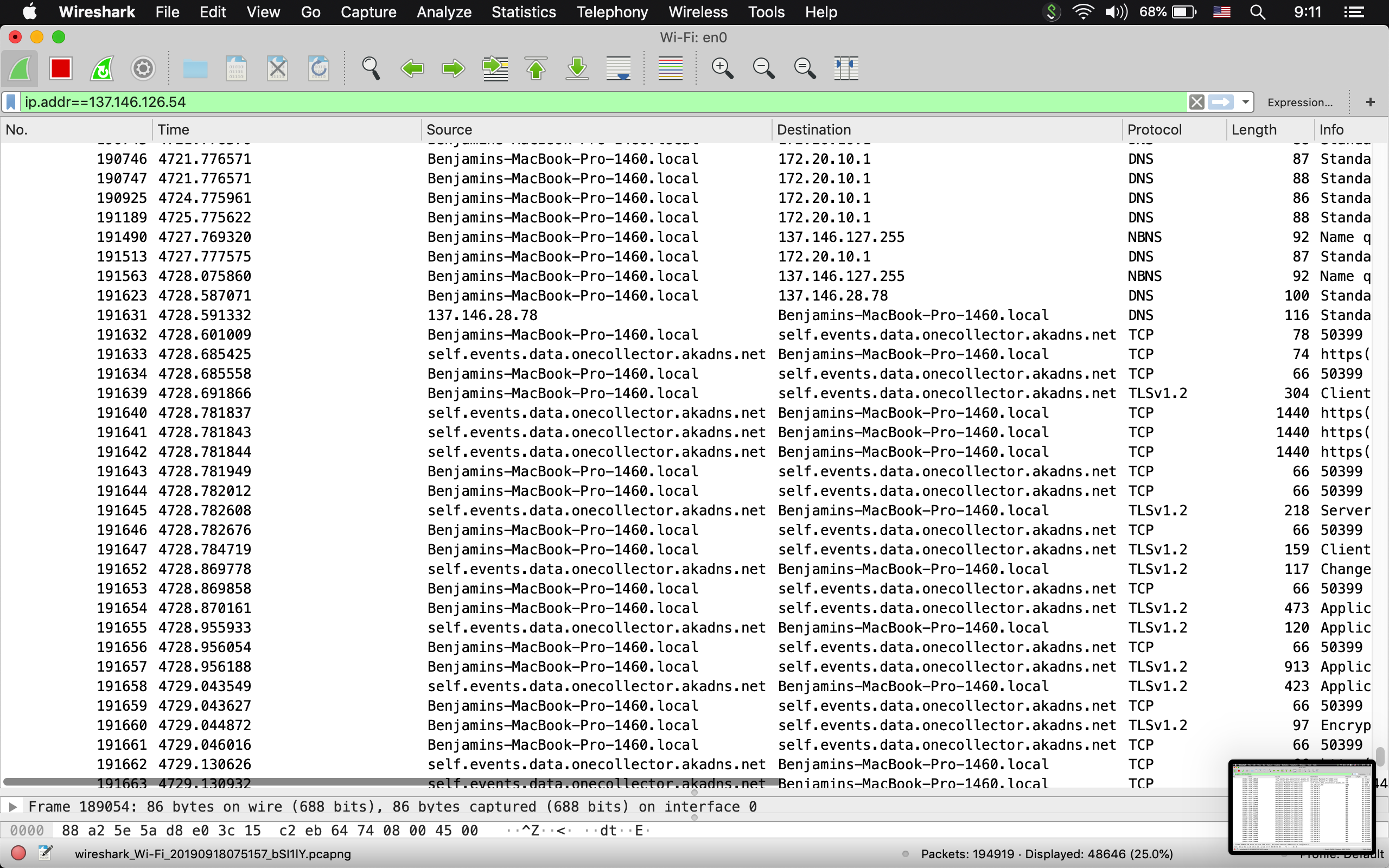


**Extensions**

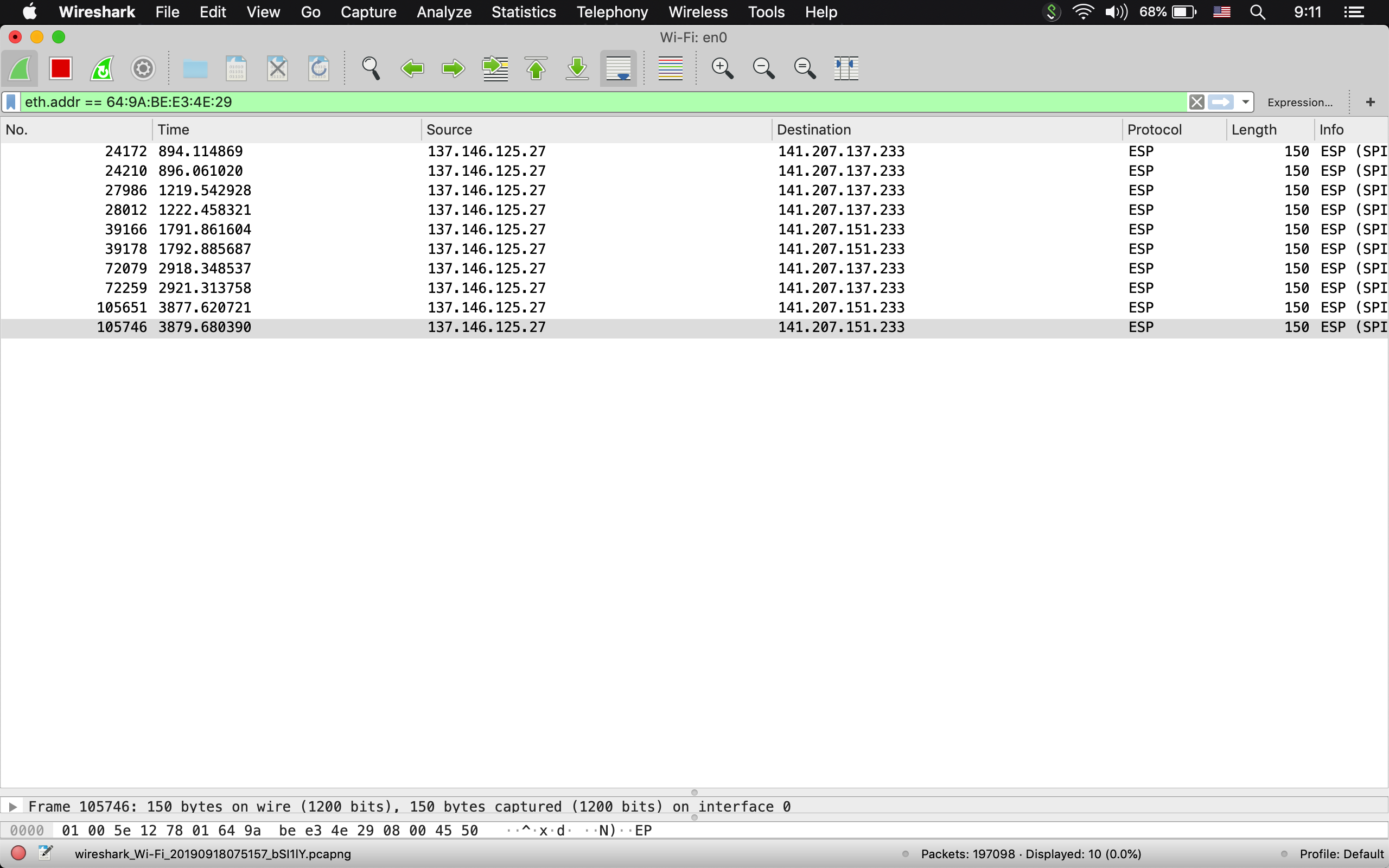
I decided to look at filters on wireshark as a part of my extensions.

I first decided to just view the traffic that is specific to my computer. To do this I typed ip.addr=137.146.126.54 into the display filter as I was receiving packets. I first closed all of the applications on my computer because I wanted to limit the amount and scope of broadcasts. The result can be seen in the first screenshot.

 I noticed that my computer still continually pings the ip address 172.20.10.1. I tried to look it up but there is no information about it. The filter was specifically created to find traces where my computer was the source and the destination, but 172.20.10.1 never sent any responses. My computer however send two pings back to back and then would rest for 2-3 seconds.



I then opened google chrome and googled colby college. As you can see, there is a significant spike in TCP requests. I did turn on hostname resolution so the IP of self.events.data.onecollector.akandns.net is not present. Here, you can see that there is a lot more activity with the intercepted packets being sent much more frequently.

I also wanted to see if I could track my phone. To do this I used the filter eth.addr== which filters for when the MAC address is either the source or the destination. I am not sure if this is because of the Colby network or something else, but I left the filter on for over an hour and this was the only traffic that was captured despite very actively using my phone to create internet traffic. 

I looked up this IP and found that it was Verizon wireless which makes sense because they are my carrier. I have no clue why this was the only traffic that was filtered, but I find it interesting that the only packets were for my mobile service provider.

