# Assignment 1 (100 points)

Anirudh Sivaraman

2020/09/11

## General instructions

For questions where we ask you to run a command and report on the results of the command, you can either (1) attach a screenshot of your terminal or (2) paste output from your terminal into your solution document.

## 1 (6 points) The five layers of the Internet

1. What are the five layers of the Internet protocol stack?
2. What are each of the five layers responsible for?
3. Where are each of the five layers implemented (end host or router)? Why?
4. Give an example of a protocol, application, or a link technology at each layer.

## 2 (5 points) The difference between capacity and propagation delay

I have listed four different kinds of communication links below:

1. A Bluetooth link between your wireless headphones and your laptop.
2. A transatlantic cable between London and NY connecting two large offices of a large company.
3. A satellite link from a ship or an in-flight WiFi link from an airplane.
4. The high-speed network inside a single building at a large company, colloquially called a datacenter.

Classify each of the above mentioned communication links into the following link characteristics:

1. High propagation delay and high capacity
2. High propagation delay and low capacity
3. Low propagation delay and high capacity
4. Low propagation delay and low capacity

Give reason for your answers. Give three more examples of other networks you can think of and what you think their propagation delay and capacity characteristics are.

# 3 (4 points) Finding your public IP address

The command line utility to view network configuration for Unix-based systems (e.g., Mac or Linux) is **ifconfig** (interface configuration). For a Windows-based system, it is **ipconfig** (Internet protocol configuration). If you run the above commands in a terminal (or command prompt), it will display network information such as IP address, MAC address, subnet, and more. For this assignment, we are only interested in the IP address.

A machine can be connected to a network in different ways, the most common ways being over WiFi or Ethernet. Each of these different ways is called an *interface*. When you run **ifconfig**/ **ipconfig**, it shows a list of active network interfaces, namely wlan0, eth0 or en0 depending on the system that you are using and how you are connected to the network. These strings are short names for the network interfaces, wlan0 being wireless LAN and eth0 being ethernet. Different operating systems have different short names.

Depending on how you are connected to the Internet, run the following commands to determine your IP address.

- "ifconfig en0" (Mac laptop)
- "ifconfig wlan0" (Linux laptop)
- "ifconfig eth0" (Linux desktop)
- "ipconfig /all" (Windows)

Is your IP address a private or a public IP address? For your reference, private IP addresses are IP addresses that belong to these IP address ranges: `https://en.wikipedia.org/wiki/Private_network#Private_IPv4_address_spaces`

Type IP address in the Google search toolbar on your laptop. This should give you the public IP address of your laptop. What is it? If you have a private address for your laptop, the public and private IPs should be different. Take out your cellphone and connect it to the same WiFi network as your laptop. Type IP address into the Google search toolbar on your cellphone. What is the public IP address of your cellphone? Is it the same as your laptop? Why?

Now, connect your cellphone to the cellular network if you have a cellular plan on your phone. Type IP address into the Google search toolbar on your phone. What is your phone's public IP address now? Is it the same as before or something different? Why?[1]

# 4 (7 points) Measuring throughput using wget

## 4.1 Background on network emulation

Network emulation is a technique that allows you to run real applications, but on networks with degraded capacity relative to what your physical link can offer. It's a good way to test out networking applications in challenged environments—especially if your own network is much faster than the networks you'll be deploying your applications in.

For this assignment, for two exercises (Exercise 4 and Exercise 6), we'll need to reduce the network's capacity to 1 Mbit/s or lower, which is typically lower than the speed of your laptop's WiFi link. This process is also called rate limiting to denote the fact that we are artificially limiting the rate at which packets are transmitted out of a link.

To rate limit your laptop's WiFi link, you can use Network Link Conditioner (NLC) (`http://nshipster.com/network-link-conditioner/`) for Mac OS X. For Linux, you can use Mahimahi's mm-link (`http://mahimahi.mit.edu/`). Both should take a short time to install, and NLC may already be on your Mac (check system preferences). If you have any trouble with the installation, please let us know.

Use the rate limiter (NLC or Mahimahi) to rate limit your network to a speed specified in each exercise in both the uplink and downlink directions, with no packet loss, and no added delay in each direction. NLC

[1] If you do not have a cellular phone plan, you can use a friend's phone for the experiment or answer the question as a thought experiment.

needs you to create a new profile to rate limit a network. Mahimahi needs you to specify the rate limit on the command line options for mm-link. For NLC, be sure to stop NLC after your experiment. Otherwise, you'll be left with a degraded link on your laptop. For Mahimahi, exiting mm-link will remove any rate limiting as well.

### 4.2 Background on wget

**wget** is a tool that uses HTTP (the same protocol your web browser uses) to retrieve web pages or other files on the Web. If wget is not installed on your machine, install it using MacPorts or Homebrew on Mac and apt-get or an equivalent package manager on Linux. For this assignment, we'll use wget to download a file available at a particular URL. wget also reports the current rate at which the file is being downloaded.

### 4.3 Measuring throughput

First, terminate any application that sends or receives networking traffic on your computer (e.g., browsers, chat clients, email clients, etc.) so that you can run a controlled experiment. Use wget to download a large file[2] using the command: "wget http://releases.ubuntu.com/16.04.3/ubuntu-16.04.3-desktop-amd64.iso -O /dev/null" Run wget for a few tens of seconds and answer the following:

1. What is the throughput that you see when you run this experiment on your laptop connected to a WiFi network? If it varies over time, what is the range of throughputs that you see? Why do you see this range?
2. Run the same experiment on a machine with a wired connection.[3] What throughput do you see?
3. Can you explain the difference in throughput between the first and second experiment?
4. Rate limit your network to 1 Mbit/s using NLC (or Mahimahi). What throughput do you see now? Why?

Attach wget's output on the terminal to subtantiate your answers above.

## 5 (8 points) Understanding protocol overhead using Wireshark

We'll continue with the wget example from before. Recall from lecture 3 how the application-layer throughput you see is less than the raw number of bits per second the underlying physical link can carry because of the overheads added by protocol headers such as TCP, IP, and WiFi/Ethernet. We'll now measure this overhead layer by layer. First, terminate any application that sends or receives networking traffic on your computer (e.g., browsers, chat clients, email clients, etc.) so that you can run a controlled experiment. Install Wireshark (https://www.wireshark.org/), which allows you to capture packets on your network.[4] Open Wireshark in sudo mode (run "sudo wireshark") and start a new capture session.

Now download the file from the previous exercise again using "wget http://releases.ubuntu.com/16.04.3/ubuntu-16.04.3-desktop-amd64.iso -O /dev/null". Filter out TCP traffic alone using Wireshark by typing tcp into the Wireshark searchbar (the place that says "Apply a display filter"). Now look at each of the packets carrying data corresponding to the file being downloaded. Answer the following questions:

1. What is the size of each packet that contains data for the file being downloaded (ubuntu-16.04.3-desktop-amd64.iso)? You can identify these packets in a Wireshark capture session by looking for packets that have the destination IP address set to the IP address of your laptop.

[2] Please ensure you're not running this experiment on a cellular network with a data plan.

[3] You can use a library machine for this. If you have trouble finding a machine with a wired connection, which has the requisite software installed, please let us know.

[4] Wireshark is a graphical version of the earlier command link packet capture program known as tcpdump, which dates back to 1988. As you can see, many of these tools, protocols, and interfaces (Wireshark, ping, Sockets, the Internet Protocol), go back a few decades, but are still widely used because they were so simple and well engineered that they were able to handle new use cases that were not anticipated when they were developed.

2. What is the number of bits in each header (WiFi/Ethernet depending on whether you are running this experiment on a laptop/desktop, IP, and TCP)? To do this exercise, when you click on a particular packet in Wireshark, you can select a particular protocol within that packet and look at the bytes corresponding to that protocol's headers in the bottomost pane. This should tell you how many bytes that protocol takes up.

3. What is the protocol overhead, i.e., the ratio of the number of bits taken up by protocol headers to the number of non-header bits?

Attach a screenshot of the Wireshark session showing your work. You can circle relevant parts of each packet to show how you identified the number of bits in each packet header.

# 6 (20 points) Measuring network latency using ping

### 6.0.1 Background on ping

Ping is probably the first tool a network admin uses to measure latency on a network. Like most networking tools, ping has been around for a long time (since 1983). Ping sends a special type of packet (called an ICMP packet) and measures the *round-trip latency* from host A to host B and back to host A. The round-trip latency is simply twice the standard latency from host A to host B, assuming the forward and return paths are symmetric.

You can run the ping program by typing this on the terminal.

```
ping www.google.com
```

Terminating ping will report the min, max, average, and standard deviation of round-trip latencies observed by ping. The report looks like this.

```
--- www.google.com ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 9.739/19.444/43.388/13.887 ms
```

The way ping works is as follows. Ping first sends an ICMP request packet from your computer to www.google.com, which www.google.com responds to with an ICMP response packet. Your computer measures the time elapsed between the time the request was sent out and the response comes back and reports this as the round-trip latency.

Recall from lecture that latency is made up of four components: application-level delay; transmission delay of individual packets, which depends on the packet size and the link's capacity; the propagation delay due to the physical distance between two points; and queueing delay because some other application ends up sharing the same link on a packet-switched network.

In our case, we'll ignore application-level and transmission delay because it takes very little time to generate a ping packet in the ping application and ping packets are very small, meaning that they have a small transmission delay. That leaves us with propagation delay and queueing delay. Let's try and understand each of these separately.

In the exercises below, when we ask you to measure the latency using ping, it's the average latency across 100 such pings. The default ping frequency is once per second. You can speed this up by using the -i argument to ping. But pinging more frequently than 10 times a second is not recommended because you risk overwhelming your network with ping packets alone.

### 6.0.2 Measuring propagation delay

First, let's look at propagation delay. For this, we'll measure the latency when the only network traffic coming out of your computer is because of ping, i.e., close your browser and any other networking applications. Ping a few sites in geographically different locations.

1. First start with your own computer (127.0.0.1 or localhost). What latencies do you observe?

2. Second, ping www.cs.nyu.edu from your laptop over WiFi. What latencies do you observe?

3. Third, ping www.cs.nyu.edu, but from a machine connected to the Internet using a wired connection, instead of WiFi. What latencies do you observe?

4. Fourth, ping 139.130.4.5, a domain name server in Australia. What latencies do you observe?

5. Can you explain the relative ordering of latencies across these four measurements?

### 6.0.3 Measuring queueing delay

Now, let's look at the queueing delay component of latency. For this, we need another application whose packets share the same queue/link as the ping packets. This other application is called cross traffic as far as the ping application is concerned because it's getting in the way of the pings. We'll also need to artificially limit the rate at which packets are sent out of your network to cause the queue to build up faster for this exercise. You can use the network emulation tools we described earlier (NLC and Mahimahi, §4.1) for this purpose. Use a rate limit of 1 Mbit/s to reduce your link's capacity to 1 Mbit/s. If you're using Mahimahi, make sure to run all the commands below within the same Mahimahi mm-link shell.

1. First, run "ping www.cs.nyu.edu" without any cross traffic. What is the latency?

2. To create cross traffic, download a large file in the background using the command: "wget http://releases.ubuntu.com/16.04.3/ubuntu-16.04.3-desktop-amd64.iso -O /dev/null &". Now run "ping www.cs.nyu.edu" and report the latency.

3. Where does this increased latency come from?

4. Kill the wget command. What is the latency now? You can toggle the wget command on and off a few times to see this effect.

5. Now rate limit the link to 500 kbit/s using NLC or Mahimahi and repeat the same pinging experiment with and without cross traffic. What latencies do you observe now? Are they better or worse than the 1 Mbit/s experiment? Why?

# 7 (10 points) Understanding DNS using Wireshark

We'll use Wireshark to analyze how a protocol works by looking at who is sending what to whom, what is in each packet, and essentially translating a log of packets in Wireshark (called a packet trace) into an English decription of what is going on. This is a useful skill when debugging network programs. For this exercise, let's look at how DNS works using Wireshark.

Open up Wireshark in sudo mode by running sudo wireshark. Add a filter to Wireshark using Wireshark's search toolbar that only shows you DNS packets, and then start capturing packets. Then open a terminal and run "dig +trace www.google.com @8.8.8.8" to carry out an iterative name lookup starting from the root servers. Stop the capture immediately after running dig so that you don't capture any more DNS packets (e.g., from some stray background application on your system) by accident. Answer these questions. For each of them, attach a screenshot of Wireshark and circle the relavent part of the screenshot corresponding to your answer.

1. What protocol does DNS use? TCP or UDP or something else?

2. What port does DNS use on the client and server side?

3. How many distinct IP addresses does the laptop contact during an iterative lookup?

4. What are these IP addresses?

5. How long does it take for the entire iterative lookup to complete? Use Wireshark to answer this question, i.e., don't report this time by running: "time dig +trace www.google.com".

6. Now, run "dig www.google.com" (without +trace). How long does this take?

7. Which IP address responds to the DNS lookup when +trace is not used?

# 8 (10 points) TCP string concatenation server

To start off with socket programming, we'll build a really simple networked application using sockets. Here, a client sends some data to the server, which concatenates another piece of data to that data and sends it back to the client. We have provided starter code for the client and server as a starting point for this assignment. Submit the final versions of your client and server programs.

# 9 (15 points) UDP RPC server

We'll now build a simple remote procedure call (RPC) system that allows a client to invoke methods remotely on a server machine. We'll support one RPC method where the client sends "prime(N)" and the server sends back yes or no depending on whether N is prime or not. You should use UDP to communicate between the client and the server. Use the provided starter code as a starting point, and submit the final versions of both your client and server programs.[5]

# 10 (15 points) TCP relay

We'll build a simple TCP relay in this exercise. A relay is a program that relays data between two other programs, often inspecting the data it is relaying for security and accounting purposes. A TCP relay works as follows. Let's assume there are two TCP end points ($A$ and $B$), connected as follows:

```
A <---> B
```

The double-headed arrow indicates that communication can happen in both directions between A and B: A can send to B and B can send to A.

Now, if we introduce a relay $R$ between $A$ and $B$, this picture is modified to:

```
A <----> R <----> B
```

Now $R$ takes bytes coming in from $A$ and sends them to $B$ and vice versa. This arrangement is at the heart of proxying, something you may have heard of. In such an arrangement, $R$ would be called a proxy server.[6] Proxy servers are used for several reasons. For instance, $R$ could require $A$ to authenticate with $R$ before communicating with $B$. $R$ could be used to account for how much traffic $A$ sends out of the network. It could also be used to inspect the traffic between $A$ and $B$ to ensure it doesn't contain anything malicious.

How do we build this relay? For this assignment, relay $R$'s work is to detect any bad words in the strings being transmitted and replace them with corresponsing good words. The bad words are: [$'virus'$, $'worm'$, $'malware'$] and their corresponding good words are: [$'groot'$, $'hulk'$, $'ironman'$]. Note that the length of the bad word and its corresponding good word is the same. This means after the relay replaces the bad words with the good words, the length of the string being transmitted does not change.

[5] UDP is unreliable. So, an application that calls recv() on a UDP socket may deadlock waiting for lost data that never arrives. Any real UDP application needs a timeout so that it doesn't wait indefinitely. This timeout can be implemented using select (see lecture 3 notes). We will ignore this complication here because losses are rare when communicating between two sockets on a single machine.

[6] In a real proxy server, a single $R$ might handle multiple $A$s and $B$s on each side, but we'll handle only a single $A$ and $B$ in our exercise.

For this exercise, you'll be required to turn in three programs, one each for $A$, $B$, and $R$. We'll also simplify the requirement to one directional communication from A to R to B, and not worry about communication from B to R to A. But, you cannot connect $A$ and $B$ directly; you have to connect them through the relay $R$.

We have provided starter code for each of the three components here: the sender (A), the receiver (B), and the relay (R). Along with the starter code we also provide you with a test file with 200 randomly generated strings that you can use to test your programs. These test strings can have zero or more bad words in them and their length is 200.

Please call the relay, sender, and receiver (in that order) as given below.

```
$: python3 relay.py <relay_port>
$: python3 sender.py <relay_port> <test_filename>
$: python3 receiver.py <relay_port>
```

Please note that the sender only reads test cases from a file. In case you want to have custom test cases you can create a new test file with the custom strings in them. Each line in the test file makes one test string.

```
Example:
Input string: gXPnDworm9s86Fq80v26TeCpCAvirusqsMU6m8Mt0SD
Relay receives the string and updates it to: gXPnDhulk9s86Fq80v26TeCpCAgrootqsMU6m8Mt0SD
Receiver should get: gXPnDhulk9s86Fq80v26TeCpCAgrootqsMU6m8Mt0SD
```

Submit the final version of each of the three programs.