Project

Due: 19 May 2022, 11:59pm

1 Introduction

We have reached the end of the course, congratulations! Take a deep breath. You've made it.

In this course, we have discussed some of the core protocols and concepts that power the Internet. Yet, there are many topics we have not had time to cover. Though some of the core protocols will be around forever, networking is a fast-moving field of CS, with new protocols, ways to build applications, and new performance and security concerns evolving every day. Thus, our goal is to give you the tools you need to tackle new networking challenges you encounter.

In this project, you will have an opportunity to use these skills by further exploring some component of networking that interests you—whether it's exploring something we've already covered more deeply, or by investigating something new. At the end of this document, we have provided a list of potential project ideas with resources to get started. You can either pick one of these ideas, or use your own.

This is an open-ended project, and is intended to be *much* lighter than IP or TCP. We have about two weeks left of the semester: in that time, you will propose a brief project idea, and spend about week working on it. After that, you will submit a brief writeup about what you did and what you learned (in addition to any code you have written). It's okay if you don't fully complete your idea in time, so long as you show progress and demonstrate what you've learned.

Meta-note This project concept has never been tested before in this class, and we don't have a lot of time left in the semester. My hope is to create something that's fun and interesting, but doesn't create a lot of stress. You all have put up with a lot this term—so I hope this project is a bit of a break.

2 Overview

In this project, you are asked to create something that demonstrates a networking concept of your choice, and to explain what is happening in a brief writeup. In general, your project might take one of two forms:

- **System-building**: Implement something that demonstrates a networking concept we have learned but not otherwise covered in a project. Examples could include building a networked application in an interesting way, or programmatically controlling network behavior with Software Defined Networking (SDN) applications. In this form, your deliverable would include your code for the application, and a writeup of what you have built and how it works

- **Network measurement**: Measure network traffic in a certain scenario to learn something about the network. This could involve writing scripts to send packets and do network measurements (ie, doing DNS queries from multiple vantage points to characterize a CDN), or parsing packet captures to gather information about certain traffic (eg. "How much of my traffic is encrypted?" or "How much bandwidth is my Zoom call using?"). In this case, your deliverable might include some results measurement results (including, eg, figures/tables, where applicable)

A list of sample topics is included in Section 4. You are welcome to use any of these, modify them, or suggest your own! Any ideas you suggest do not need to fit into these two categories—you can work on whatever you want, so long as we approve your idea.

2.1 Logistics

Groups You should work on your project in your IP/TCP group. If you want to alter your group, please contact the instructor and we will work something out.

Repository You can create a new repository for your project using the following link:
`https://classroom.github.com/a/mUVGTbKm`

This repository is completely blank. Since this is an open-ended project, there is no starter code or reference implementation—this repository is just a place to keep your work and collaborate.

Language requirements (or lack thereof) You can work on the project using any language(s) you want—whatever you think will help you accomplish the project most easily. For example, if you're doing network measurements, you could write a shell script or python program to make a bunch of DNS queries and store the results to a file, and/or a Python script to plot the results or parse a capture file. The sample projects have links to some software libraries for common languages that you may find useful—you are welcome to use these, or find other libraries for other languages you might prefer.

Implementation requirements Your implementation does not need to be particularly robust or complex, so long as you implement something to demonstrate progress toward your goal and show you understand the networking concepts involved. In general, this project is similar to what research is like: you can use any libraries, tools, resources, tutorials, or other materials that already exist to help you *so long as you understand how they work, and document where you found them*.

For system-building style projects, your implementation might start by following a tutorial to build X using some software framework, which you can then modify/extend to meet your goal. For measurement-style projects, your implementation might be some scripts that gather and/or plot results that you include in your writeup. Either way, you should expect to write some code on top of resources you find online, even if it's just code that "glues" components together.

2.2 Deadlines and deliverables

2.2.1 Project proposal (due by Monday, May 10, by 11:59pm EDT)

On or before Tuesday, May 10, you will write up a *brief* description of the project you want to implement and submit it on Gradescope. Your proposal writeup need not be more than a couple of paragraphs: just describe what you're thinking of doing, what you hope to achieve as a final deliverable, how you intend to get started, and any questions you have. **If you have enough ideas to write up your proposal before the deadline, we encourage you to submit early**—we will be monitoring Gradescope daily and are happy to provide feedback sooner!

2.2.2 Final writeup and submission (Thursday, May 19 by 11:59pm EDT)

When you are done, you will submit your work by pushing all code to your repository and submitting a short writeup. Your writeup should explain what you have built and what you learned. There is no length requirement, but a reasonable estimate is on the order of 3–4 pages of text/figures. In general, your writeup should contain at least the following components:

- **Introduction**: What were your overall project goals? What (briefly) did you achieve?
- **Design/Implementation**: What did you build, and how does it work? Explain the major design decisions behind what you implemented. For a system-building project, this would be an overview of the major components of your system design, similar what you might write in a readme. For a measurement-style project, this would be an explanation of what you intend to measure, how you intend to measure it, and how you would draw conclusions from the results.
- **Discussion/Results**: How far did you get toward your goal? In this section, describe any results you have, what you have learned, and any challenges you faced along the way. For a system-building project, you might include some logs or screenshots of your program operating. For a measurement-style project, this is the place for a concise summary of your results, perhaps in some figures or tables, and your interpretation of the data. If you don't meet your goal, or don't have a lot of results, that's okay—just describe what you have and what you learned along the way.
- **Conclusions/Future work**: Overall, what have you learned? How did you feel about this project overall? If you could keep working on this project, what would you do next? Are there any other directions of this work you find interesting? If you have any thoughts or feedback on this project model, please let us know!

Deadline Your final submission is due by **Thursday, May 19 by 11:59pm**. This deadline has been selected to give you maximum flexibility in this busy end of semester—it is the latest possible date I can choose while giving myself time to grade all of your work. Therefore, **no extensions or late days are possible**, other than for extenuating circumstances that would warrant a grade of Incomplete.

3 What "open-ended" means

Once again, this project is meant to be a chance for you to get started exploring some more network concepts in the very short time we have left in the semester. You are **not** being asked or expected to build a fully-fledged system or research project: just pick an idea, spend about a week on it—with a number of hours that you'd consider reasonable for a single class—and submit your work. If you envision your project as having steps 1–5, and step 1 takes you a week, **that's okay**: just submit what you have and tell me what you did and what you learned. So take this time to explore something that interests you, and have fun!

4 Sample project topics

The following pages contain some sample project ideas. Note that this project is new to the course, and that these are just ideas: you don't have to use them, or you can use a subset of one to help you reach an idea. For each project, we have listed a few resources that we think may be helpful, but these are just suggestions, and we have not necessarily tested them in this context.

Each project idea also lists a guess for a suitable development environment: due to the way in which certain network measurements may be performed, not all can fully run within the course container environment.

If you have questions about using any tools, development environments, etc, please feel free to ask us for help on Ed or during office hours. However, note that we have not tested these projects before, so we won't have all the answers—but we're happy to help you figure out how to approach the problem and point you in the right direction!

(Continued on the next page)

4.1 System-building: A better Snowcast

When we built Snowcast, you wrote code to manually compose messages in the Snowcast protocol format and send them along TCP sockets. This is a great exercise in implementing a protocol. However, modern applications often leverage frameworks to help build network APIs more quickly. One such framework is gRPC: users can define their API and message formats, and the gRPC framework automatically generates code for establishing connections, authentication, serializing messages and more, in your language of choice.

To explore these tools, you could implement part of Snowcast (or some other protocol of your choice) in gRPC, or some other framework that provides similar functionality. A good starting point might be to build a client and server that connects and exchanges Snowcast's `Hello/Welcome` messages, and then continue with selecting stations and streaming data. As you do this, reflect on the differences between using a framework like gRPC and building Snowcast on your own from a TCP or UDP socket. What components are automatically provided for you? What parts of the spec do you still need to implement? Is there anything that you would need to change about the protocol to use a framework like gRPC?

Environment You should be able to implement this from our course's container environment.

Notes/Resources

- `https://grpc.io/` contains getting started guides for various languages
- gRPC is based on Protocol Buffers (protobuf), a data serialization framework. You can read more about Protocol Buffers here

4.2 System-building: Implement a webserver

In class so far, we have had a taste of implementing server applications. Knowing what you know now, implement a webserver (starting from sockets, like Snowcast) that speaks enough of HTTP 1.0 or 1.1 to serve files from a configurable directory. You could add to your webserver by considering any of the following in your implementation:

- How could you make your server scale to large numbers of requests? Use a benchmarking tool like `wrk` to measure your server's performance (eg. number of requests that can be handled per second) and see how you could improve it
- Modern webservers don't just serve files, they often run server-side application code to make pages dynamic. How might you modify your server to do this in an extensible way? What are some security and performance implications?

Environment You should be able to implement this using our course's container environment.

Notes/Resources

- Wikipedia's page on HTTP has a good overview of the basic elements of the protocol, and a list of RFCs. For a quick experiment to measure performance, you can probably get away with just implementing the `GET` command (similar to what was shown in lecture)
- wrk, an HTTP benchmarking tool

4.3 System-building: SDN Application

We have discussed how routers and switches implement various protocols to control how packets are forwarded, such as OSPF, Spanning Tree, etc. We can think of these protocols as the network's *control plane*. Traditional switches and routers are shipped with pre-built firmware that implement these control plane protocols, which can only be tweaked to a limited degree by network administrators. Software Defined Networking (SDN) is a departure from this paradigm in which switches do not contain pre-written control-plane software, and instead only forward packets based on a generic set of forwarding rules. SDN switches export an API to configure rules, as well as provide information about packet events, allowing the switch's entire behavior to be controlled by an external application, often by a centralized network controller with a global view of the network. This separation of the control and data plan permits more dynamic, flexible, and extensible network configurations. SDN is an area of active research, with new hardware, methods, applications being developed to work with more "programmable" network devices.

One way to explore SDNs for your project is to experiment with writing an SDN application that runs on a network controller. Example applications can implement switch features like mac learning, Spanning Tree Protocol, DHCP, network telemetry, and so on. To implement an SDN application, you would write a program for an SDN controller—when a switch receives a packet, it will ask the controller how to handle the packet, which will consult your program to tell the switch how to set up its forwarding table or respond to the packet.

This course once had a whole assignment for building SDN applications, using the well-known OpenFlow protocol and Ryu SDN controller. To get started working with SDN, we recommend looking here first. This assignment was about implementing shortest-path forwarding, but the tutorial and setup information is useful even if you want to build a different kind of application.

Environment The easiest way to start with SDN is to use Mininet, which should be run in a VM[1]. In the past, our course has used a VM environment using Vagrant, which should be straightforward to set up if your computer supports it. You can find instructions on the course VM environment here:
`https://cs.brown.edu/courses/csci1680/s22/content/vagrant.pdf`
M1 mac users: This VM will not run on your system. If want to do this project, it would be extra work to figure out how to use Mininet in a container or VM on an M1 Mac—in fact, if this interests you, this alone could be your project (and it would really help me learn what's possible)!

Notes/Resources

- The starter code for the old SDN assignment can be found at: `https://github.com/brown-csci1680/sdn-starter`, but beware this code has not been tested in four years.
- Mininet is a network emulator that can create arbitrary network topologies with switches that you can use to test SDN applications.

[1]For reasons I can describe if you're interested, Mininet won't run in our container. We do not recommend running mininet natively on your machine, either, as it requires running ancient Python 2 code as root.

- Ryu is an easy-to-use OpenFlow-based SDN controller written in Python. It has some good tutorials for getting started and a lot of example applications
- P4 is another interface for programming network hardware, separate from the OpenFlow model described in the assignment here. P4 has gained much more traction for its flexibility in recent years compared to OpenFlow. You are welcome to experiment with P4 as well, though the setup cost may be higher

4.4 Measurement: Investigating Zoom traffic

As an old person, I don't understand how Zoom works—or, at least, in terms of how it uses the network. I want you to explain it to me. Capture some traffic while you use Zoom and report on what happens while using various features (hosting calls, joining calls, screen sharing, etc.). Can you tell what protocols are used? How much bandwidth does the call use? Does everything happen over one connection, or multiple? What IPs are involved, and where are they located? Does the video data get sent to Zoom's servers, or directly to other users on the connection? How does this differ from other videoconferencing applications (Hangouts, Messenger, FaceTime, etc.)?

You can start this by using Wireshark and looking at the traffic. For a more detailed analysis, one option is to write a script that parses a capture file and outputs some useful statistics. Note: I don't expect you to understand everything about how Zoom works (and indeed, much of the traffic may be encrypted), but I'm quite curious what can be learned from a surface-level analysis!

Environment To capture traffic from Zoom calls, you would need to install Wireshark on your own machine, rather than inside the container. For more detailed analysis, you can save the data captured by Wireshark to a file (ie, called a "capture file" or "pcap file") and process the data in your container environment (or anywhere else). Capture files are a standardized format that can be processed by various tools and libraries.

Notes/Resources

- Zoom has a whitepaper about its connection process. You might consider comparing what you observe against this (or potentially other resources you find online) to see if you can replicate their results
- `tshark`, Wireshark's terminal-based counterpart, has some good options for generating summaries of capture files that might be useful
- scapy and PyPCAPKit are Python libraries for parsing packet capture (PCAP) files

4.5 Measurement: Investigating CDNs

In class, we discussed how CDNs use DNS to direct users to nearby servers. We this by querying a domain name from multiple DNS servers at different geographic locations. One way to explore this further is to query a domain from many vantage points and examine the IPs that are returned. For example, let's say you ask 100 DNS servers around the world to resolve `randomsite.com`. How many different IPs do you learn? Where are they located? Do they all belong to the same content provider?

To investigate this, you could write a script that takes in a domain name, queries it against a list of DNS servers around the world, and examines the results. From here, you could potentially ask other questions of different domains: Do all the DNS responses use similar TTL values? Are the records signed with DNSSEC?

Environment You should be able to implement this from our course's container environment.

Notes/Resources

- `https://public-dns.info/` curates a list of public DNS servers around the world you can query.
- When sending DNS queries, don't send a huge number of queries to the same server in rapid succession—otherwise you might get blocked! Instead, wait >100ms in between queries.
- You can map IP addresses to coarse physical locations using a GeoIP database. For example, you can do this in Python using python-geoip, which reads an IP-to-location database stored on your system. You may need to install the database separately—this link should lead you to instructions.
- You can make DNS queries from a script by using a DNS library (a good Python one is `dnspython`, or you can simply run shell commands from a script and parse the output (fast, but can get ugly)

4.6 Measurement: Investigating network conditions with Mahimahi

Mahimahi is a set of tools to emulate various network conditions. For example, you can create links with certain levels of packet loss and delay in order to explore how protocols may behave under various conditions.

One way to explore this in your project could be to explore how modern TCP implementations behave differently under various network load conditions. How much is web page or video performance impacted? How might different congestion control algorithms behave differently (eg. BBR, vs. CUBIC, vs. Reno)? To report on your results, you could measure throughput observed for transferring files of a known size, monitor time to load web pages, play videos, etc.

Environment You may able to implement this from our course's container environment. Testing different congestion control mechanism may require support from your OS kernel, which may not be possible on all platforms—your mileage may vary, let us know if you have questions and we may be able to help.

Notes/Resources

- Mahimahi's documentation has some example usages
- `iperf` is a common tool for measuring network throughput. When supported by the OS, `iperf` supports testing TCP using different congestion control mechanisms