CPE 400 Final Project: Tracking Network Data to/from a Nintendo Switch

Name: Emerson Fleming

Goal of the project: The goal of this project is to analyze network data being sent between a home network and a Nintendo Switch. An idea that interested me as we moved through this course was how UDP is the best protocol for online gaming, as there needs to be no perceivable latency in an online match. Thus, I wanted to measure how the Nintendo Switch sends and receives network packets both when in normal console surfing and when in an online gaming setting. Specifically, I wanted to measure the difference between TCP and UDP protocols, as well as measure how often the Switch was the source or destination of these packets.

Details of Code:

- **How to run:**
    - *Step 1: Run Wireshark-to-CSV conversion*. For the Python scripts I wrote to interact with the data, I wrote a Python script that converts a Wireshark (.pcapng) file into a CSV file. To generate the CSV file, run: **python3 script-to-csv.py switch_data.pcapng switchdata.csv 192.168.137.58**
    - *Step 2: Generate a Bar Graph Based on the Data.* To generate a bar graph comparison of each protocol's source and destination packets, run: **python3 bar_graph.py**
    - *Step 3: Generate a Line Graph Based on the Data.* To generate a line graph comparison of each protocol's source and destination packets, run: **python3 line_graph.py**
- **Environment:** Be sure to download **python3**, as that is what I used to test the provided code. As far as Python libraries, be sure to download **pandas, matplotlib, and pyshark. (pip install pandas matplotlib) (pip install pyshark) (sudo apt-get install python3-pip if you don't have pip)**
- **Input/Output:** The inputs are all files, with the exception of the IP address in the first step's command line argument. This is the IP address of my Nintendo Switch, allowing for the CSV file to entirely be Nintendo Switch data. The outputs will be a CSV file as

seen in step 1, as well as two graphs created with matplotlib. To save these outputs, be sure to press the save button upon them popping up.

- **ADDITIONAL COMMENTS. If you encounter any issues, email me at efleming@nevada.unr.edu.**

Details of data:

- **Pointer to data:** I have attached a .zip file to this submission that houses all of the data. In it are all of the python scripts and the Wireshark capture file. I also have a folder in there with the precompiled data in the event that there is an issue with compilation (which there shouldn't be if the instructions are followed correctly), that includes pictures of the charts and the CSV file from the Wireshark data.
- **Cleaning and processing:** The purpose of the script-to-csv.py script is to better organize and clean the data so it's more readable than a typical Wireshark output file. Additionally, in the bar and line graph Python scripts, the CSV data that is pulled in is reorganized into an array that is much easier to code from.
- **Any extra comments:** While this will be explained further in the evaluation, I expected there to be more UDP protocols than TCP due to the nature of online gaming needing low latency between users.

Methodology: The data consists of a 30-minute play session on a Nintendo Switch console. In it, a game with asynchronous multiplayer (*Super Mario Bros. Wonder*) and a game with live multiplayer (*Splatoon 3)* were played, with the data being tracked for both. While I initially tried to collect the data for the Switch via a simple Wi-Fi packet capture, Nintendo blocked all TCP/UDP data from being picked up wirelessly, presumably for security reasons. Thus, I connected the ethernet port on my Switch dock to my PC, then shared the network through my PC, which was a method that allowed me to capture (presumably) all network traffic coming in and out of my Switch. I consulted the Wireshark forum pages consistently to figure out how to gather this data, but ultimately did not find a solution until I put it to the test via ethernet. While I initially planned to cover data that I collected in this manner from a PS5 and Xbox Series X as well, the sheer amount of data collected from those consoles overloaded any and all scripts that I wrote to filter and analyze the data, so I opted to stick with just the Switch.

Evaluation results:

As expected, the UDP source and destination protocols reigned supreme due to the prevalence of online gaming during this session. However, I was surprised by the volume of the TCP protocol alongside it, making up roughly 1/3 of the protocols detected. There were a few unrecognized protocols, but only 31 total out of nearly 200,000 data points, so it was pretty negligible.

My best guess as to why the amount of TCP protocols was pretty high is that these come from the idle/lobby parts of the online gaming experience. I imagine that since UDP is relatively unreliable in many ways, the initial connections must be done through TCP to connect the users all to Nintendo's online servers, and then once all players are in and verified the match begins and latency becomes much more of a concern. As seen in the line graph, there is a pretty clear delineation of TCP and UDP calls, with very little back-and-forth going on with the two protocols. The spikes in UDP can be assumed to be the two online play sessions, with the TCP being when I was waiting in game lobbies or online but not actively playing with other players. Since TCP requests often result in more data coming back to the user, it makes sense why the TCP destination stats are so much higher than the source, especially when compared to the nearly-identical UDP exchanges.

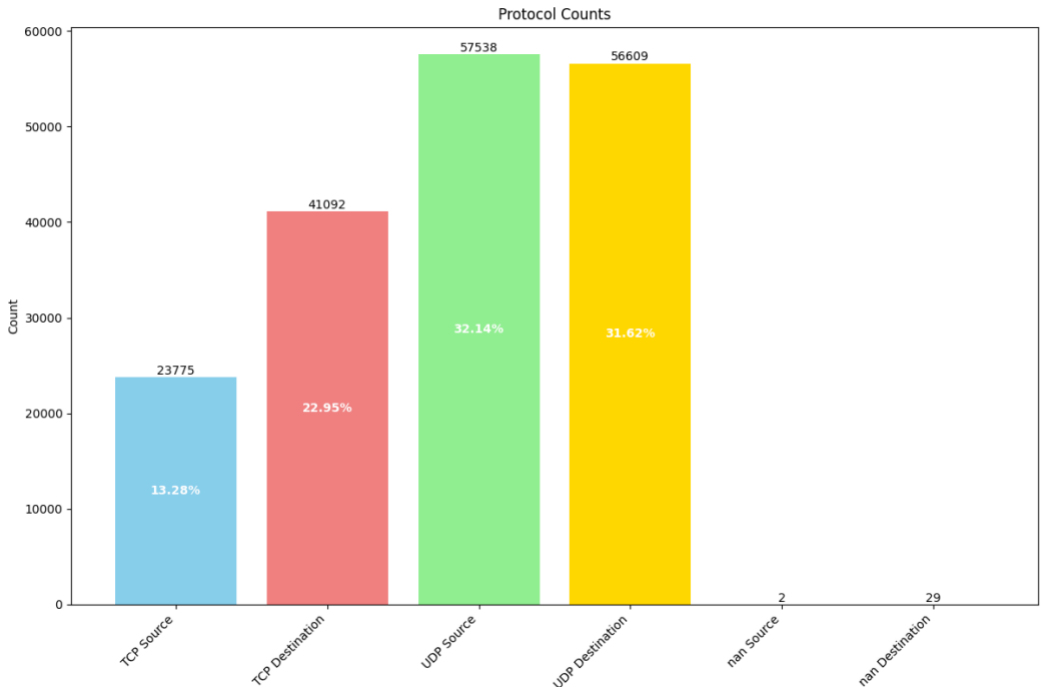

Figure 1: Bar graph denoting ratios of TCP to UDP source/destination protocols

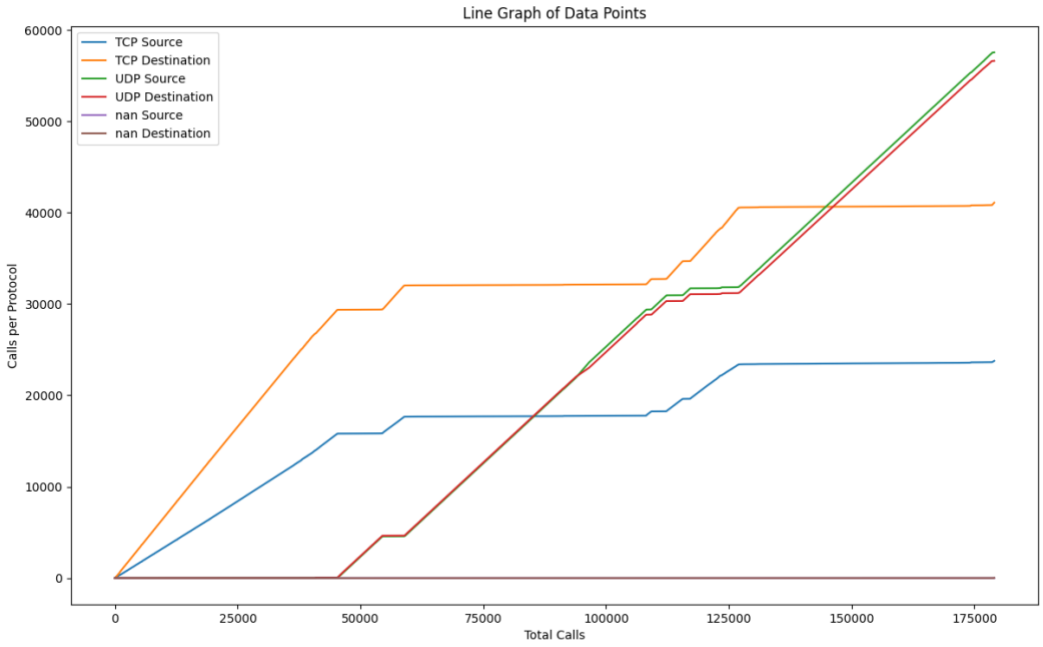

Figure 2: Line graph showing trends over play session in UDP and TCP protocols

Workload: I am the only member in my team.

Tools that I used: Wireshark, Nintendo Switch (w/ ethernet-capable dock), ethernet cable, PC

Challenges: The biggest challenge that I faced was trying to include PS5 and Xbox Series X data in this study. While I was able to capture the data, in the same time span (~30 minutes) of collection for these consoles as the Switch, nearly 4 million packet transfers occurred for each console, leading to .pcapng files over 10GB and unable to be worked with much. Thus, I shifted gears to only analyze the Nintendo Switch, as it had a data sample large enough to analyze but small enough to play around with in code.

Future directions: To better improve this project, a comparison between several consoles—possibly with the same game across all three—would help paint a better picture of how much packet sharing takes place in modern game consoles. While the Switch still had plenty of data, it is no secret that it is behind the curve in hardware and network capabilities in comparison to its contemporaries, the PS5 and Xbox Series X. If I had more time, I would have figured out a better method of condensing my collected data on the other consoles, so I could analyze how both the volume and ratios of TCP/UDP data would compare amongst one another.