Southeast Technological University

High Level Packet Sniffer

Final Report

Thassanai McCabe – C00250439
Supervisor: Paul Barry

Contents

Table of Figures

1. Introduction

This document describes CableOrca, the high-level packet sniffer and how it conformed to brief requirements. I highlight aspects of the development process such as successes and failures along with descriptions of both personal and technical learning experiences gained from completing the project. Furthermore, a description of my technology choices and the approach I would take if I could start again are found within this document.

2. Description of Submitted Project

The developed application is a packet sniffing tool that aims to be as accessible as possible. It is designed to be used by both technical and non-technical users, meaning knowledge of networking protocols and techniques is not required. It is not designed to replace existing powerful tools such as Wireshark for example, but instead, complement them. CableOrca acts as a strong baseline-creating tool when diagnosing network issues.

CableOrca can create and read packet capture files also known as .pcap files. These pcap files can be created using different tools and still be read by CableOrca. Furthermore, files created by CableOrca can be opened and read using other tools for analysis. CableOrca supports new generation packet capture files (.pcapng) as well as IPV6. The supported operating system is Windows 10.

Core features of the final application

- Network Speed Test
- Device Discovery
- Network Analysis
- Packet Capture File Parsing + Analysis
- Network Diagnostic Guidance
- Graph Generating

Screenshots

Home

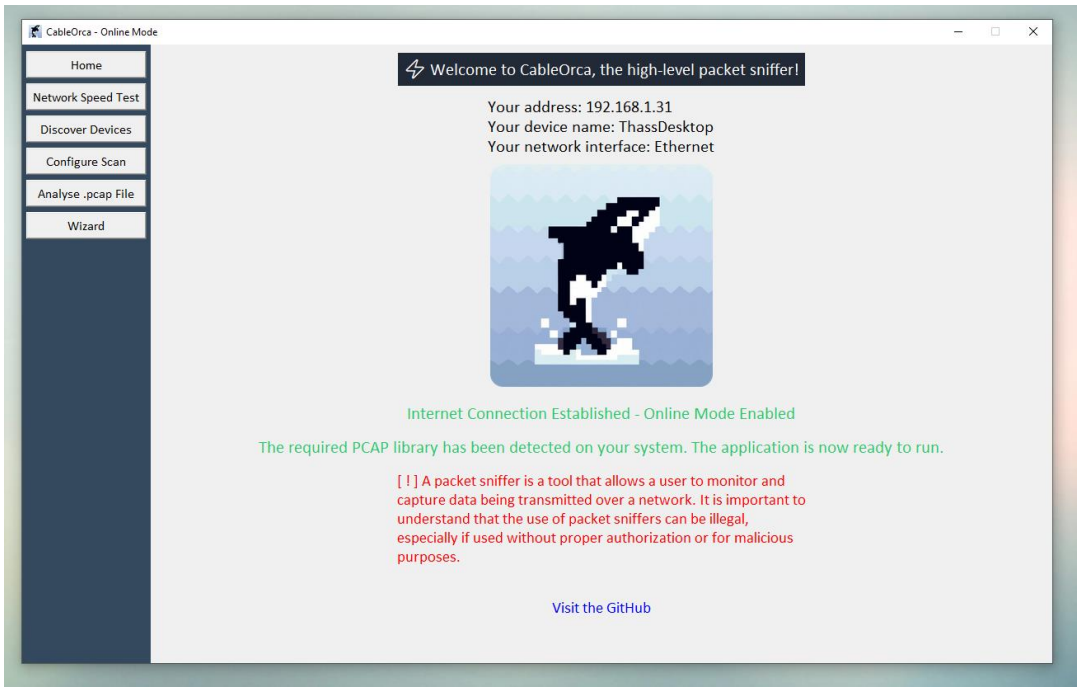

Figure 1 Home Menu

The home menu is the first thing a user sees upon opening the application. It highlights key information about the application and current device.

1. Device telemetry
2. Internet Connection Status
3. PCAP Library Status
4. Disclaimer

CableOrca relies on a packet capture library for Microsoft Windows. The one used during development was Npcap, which is a free library created by Nmap's developers. A message for the user is displayed if the required library is found on their system, informing the user that the application is ready to use. If this library is not found on the system, a red error message is displayed, informing the user what the issue is and providing them with a link to npcap's download page. Furthermore, if no internet connection is established when starting the application, a message is displayed to the user and the title of the application is changed to "CableOrca – Offline Mode".

To the left is a menu bar which allows users to jump to various parts of the application. These are:

1. Home (Current Window)
2. Network Speed Test
3. Discover Devices
4. Configure Scan
5. Analyse .pcap File
6. Wizard

Clicking on each item on the menu bar will change the current window to the chosen screen. The menu bar will always be available for users to access throughout the application.

Network Speed Test

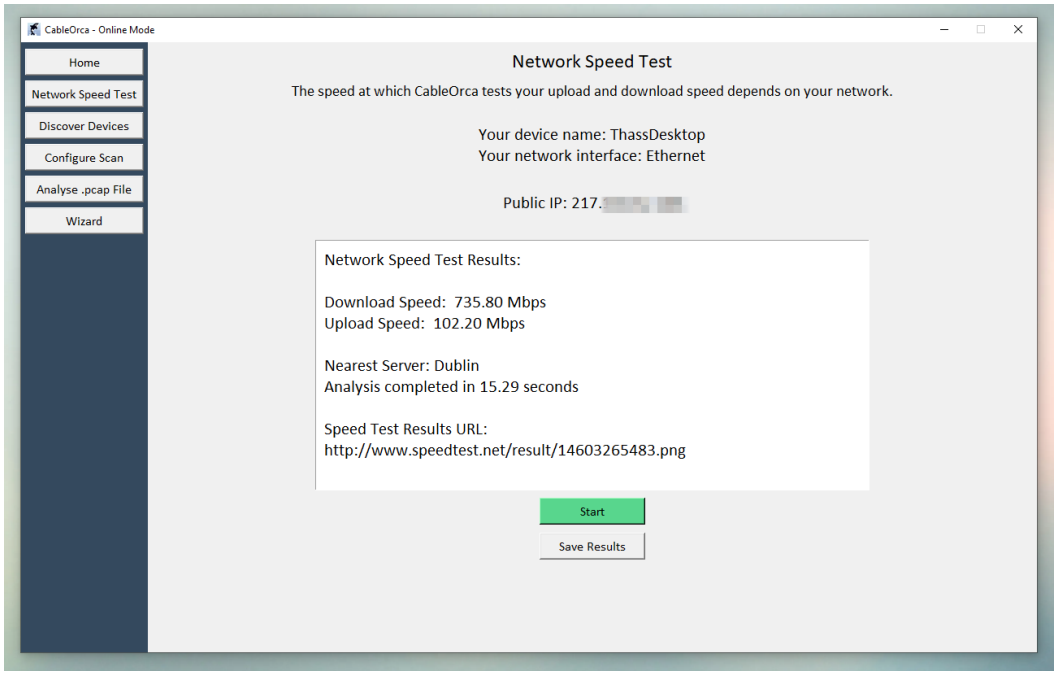

Figure 2 Network Speed Test

This window uses the Ookla speed test to test the user's internet speed. I display information such as the current device, current interface, and current public IP address for the user. The public IP address is fetched using Amazon web services, with a backup resolver called "api.ipify".

Users have the option to save their results if they need to refer to them later. To save results, Ookla provides a file link to accommodate this. This is shown in Figure 3. Internet speed tests cannot be run unless an internet connection is established. If no internet connection is available a message appears for the user on the application. If no internet connection is established, users can refresh the window and the application will re-check the status for internet connection automatically.

Information on the graphical user interface, such as the public IP address may not always be successfully determined by the application. Consequently, error handling and alternate messages are in place for these scenarios.

Speed Test Result

An example of a saved result through CableOrca is shown below:

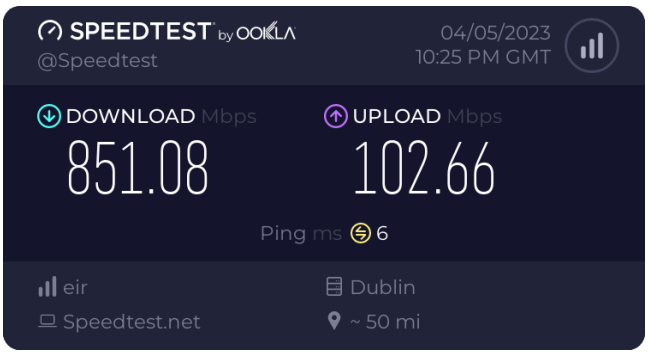

Figure 3 Saved Speed Test Result

Device Discovery

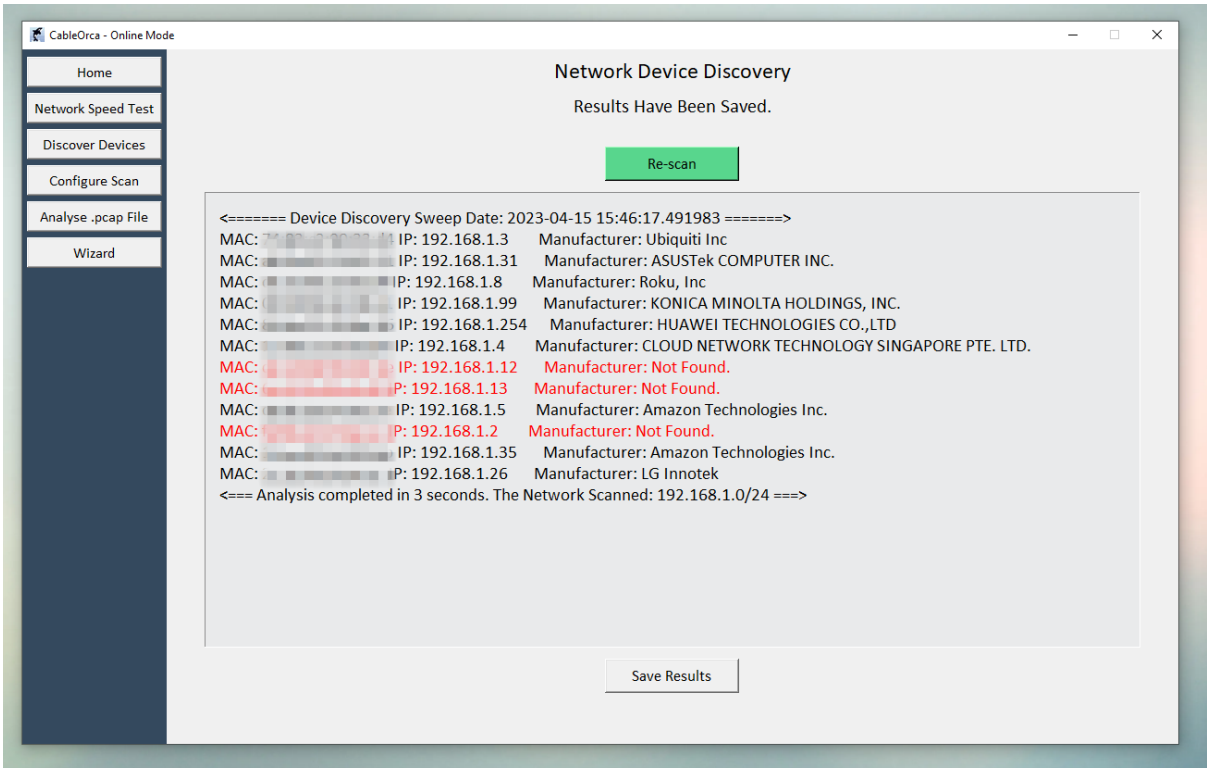

Figure 4 Network Device Discovery

This window allows users to discover devices on their network using an ARP scan. As described by Bonderud, 2020, ARP stands for Address Resolution Protocol and allows for the mapping of IP addresses to physical machine addresses.

The application will set a specified network address to scan. It determines this network address using the IP address of the host running the scan. In the screenshot I have provided, the bottom of the input shows “The Network Scanned: 192.168.1.0/24”. This is the automatically determined address that was scanned in this use case.

The list of devices and manufacturers has been censored for security reasons. Text entries highlighted in red represent devices that were found, but their manufacturer name could not be resolved. My application uses a Python module called "Mac vendor lookup" to resolve these manufacturer names. The MAC addresses of all devices are also displayed.

Users have the option to save this output as a file if they need to refer to it later. Information about the scan such as network address, time and date of scan, and analysis duration is added to this file for the user.

Scan Configuration

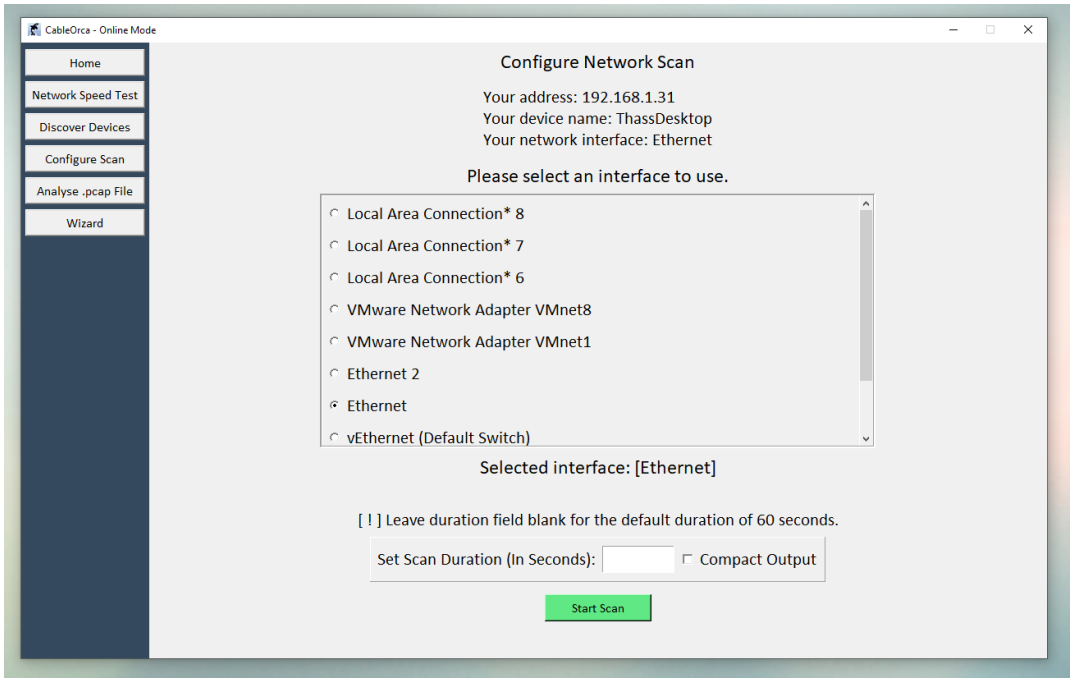

Figure 5 Network Analysis Configuration

This window allows users to configure a network scan. Users are presented with a dynamically created scrollable list of all the interfaces available on their devices. A suggested default interface, along with the IP and hostname of the current device is displayed above this list. Furthermore, for each list generated, there is an option to select "all interfaces" which uses all possible interfaces during network sniffing.

Users can set the duration of the scan in seconds using a text field, it can be left blank for a default duration of one minute. Users can also select an alternate output mode called "Compact Output". This changes how packets are represented to the user as they are captured in real-time after they start the scan.

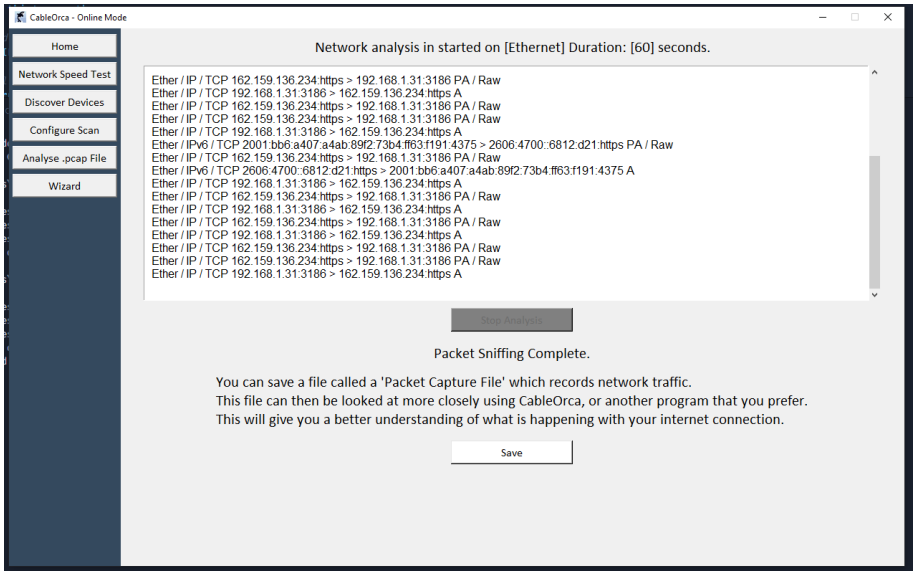

Figure 6 Network Analysis Example

Figure 6 shows what a user would see during network analysis. I have clicked the “Stop Analysis” button. This has caused a prompt to appear which informs users that they can now save the network traffic they have captured.

The scrollable text pane displays captured packets in real time. The heading of the page display information about the current scan like the interface used, and duration.

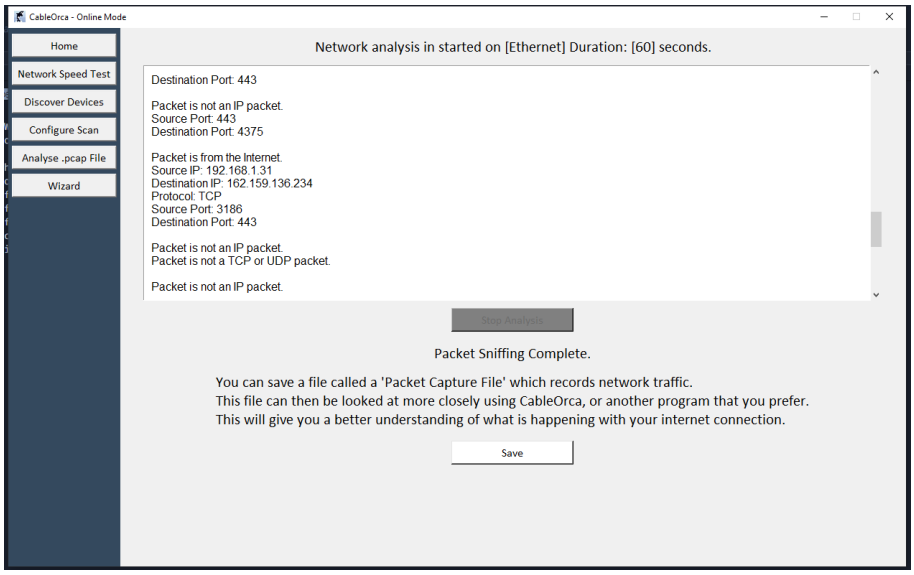

Figure 7 Network Analysis Example - Compact Output

Figure 7 displays what a user would see if they selected “Compact Output”. Here users can gather a consolidated view of information about packets captured such as protocol and whether a packet is involved in an internet connection or not.

After a scan duration has elapsed, or a user has clicked “stop”, the option to save a packet capture (.pcap) file is granted. Here users can select a file name and location for the .pcap file to be saved. This button is only available once packet sniffing has been stopped by the timer or user interaction.

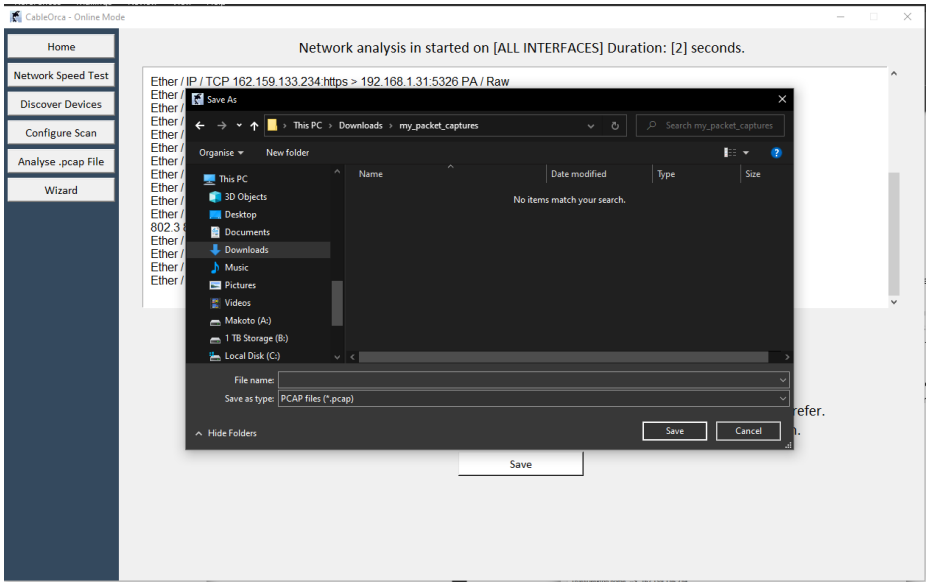

Figure 8 Saving a File with Windows File Dialog

Note that files can only be saved using the .pcap format. This file type can be examined in my application or another application that is .pcap file compatible.

PCAP File Analysis

Figure 9 PCAP File Analysis Window

This window allows users to analyse packet capture (.pcap) files. This file could be generated using my application, or other existing applications such as Wire Shark. After a file is selected. The "Tier" buttons get enabled. Tiers 1, 2, and 3 are different analysis modes available in my application for analysing a .pcap file. When a user selects a particular tier for analysis, they can still choose to switch to a different tier after the analysis has been completed, to gain additional insights into the same .pcap file.

Tier 1

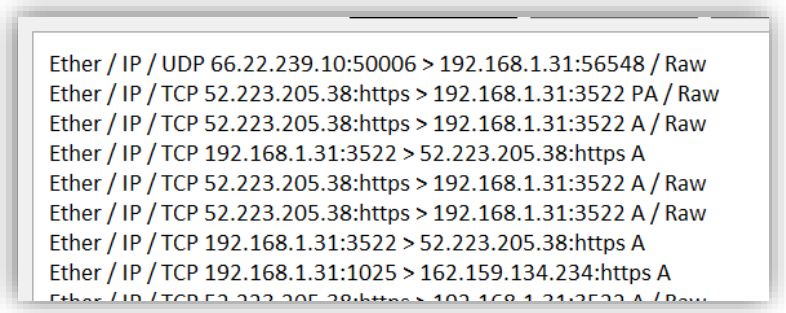

Figure 10 Raw File Reading

When users perform a "raw" read of a .pcap file, it means that the application reads packets at a low level, with little interpretation or processing of the data. This type of low-level analysis is the fastest way to read a file, but it may not be in a format that is easily understandable by all users.

Tier 2

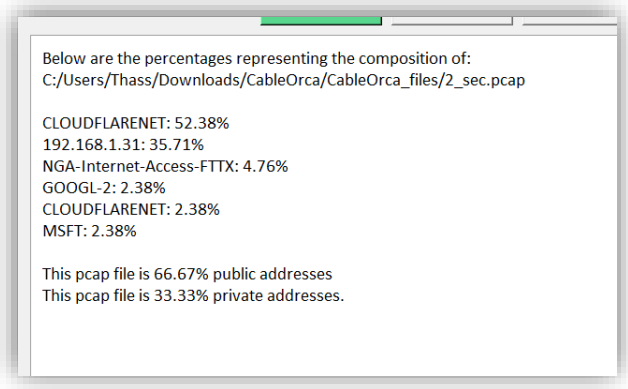

Figure 11 Tier 2 File Analysis

Figure 10 is what the output for a file called "2_sec.pcap" would look like if analysed using Tier 2. This gives users an alternative way of viewing their packet capture file. It takes each address in a file and attempts to translate it. It will then calculate what percentage of that packet capture file that specific address makes up. Using this tier, a user can quickly determine what may be using the most bandwidth on their device. In Figure 10, CLOUDFARENET appears to have been the most active address during the network scan.

Furthermore, a calculation of public to private addresses is shown below the output, which can allow users to quickly know if their traffic is mainly on their private network, or with the internet.

Tier 3

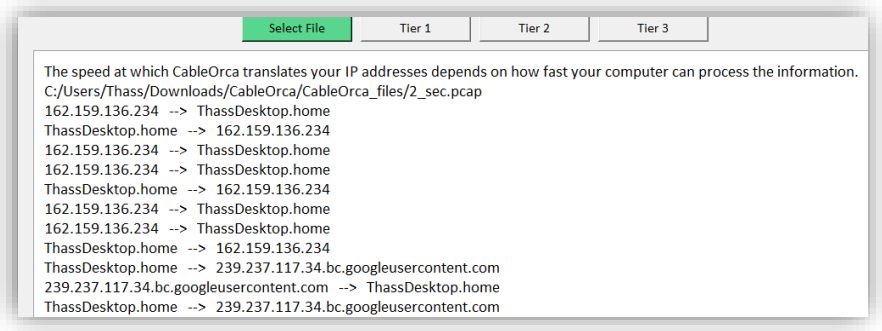

Figure 12 Tier 3, Socket Translation Reading

Tier 3 reading allows users to view network communications using socket translations. Data is represented using “source - > destination".

As socket translates addresses from the chosen .pcap file, a Python dictionary is constructed in the background. The purpose of this dictionary is to speed up the translation process for the user. Values in this dictionary may look like the following: "{200.394.298.394: google.com}".

By holding these values, the application can avoid translating addresses that have already been translated. Furthermore, the application will not attempt to translate addresses that it has already tried and failed to. An example of an address that would not be translatable would be an IP address that is in a private address range.

Network Diagnostics Wizard

As stated by (Rouse, 2017), a “wizard” is a piece of software that simplifies a task for the user. One might even call it a “digital tutorial”, or guide.

The goal of the network diagnostics wizard is to allow all users, including non-technical users, to quickly gather a baseline set of information to use when analysing their network. The wizard aims to act as a "first step" in a process to diagnose network issues.

The wizard asks a series of questions to the users which they can answer “yes” or “no” to. As users go through questions, they can ask the wizard to perform tasks for them or guide them in the right direction.

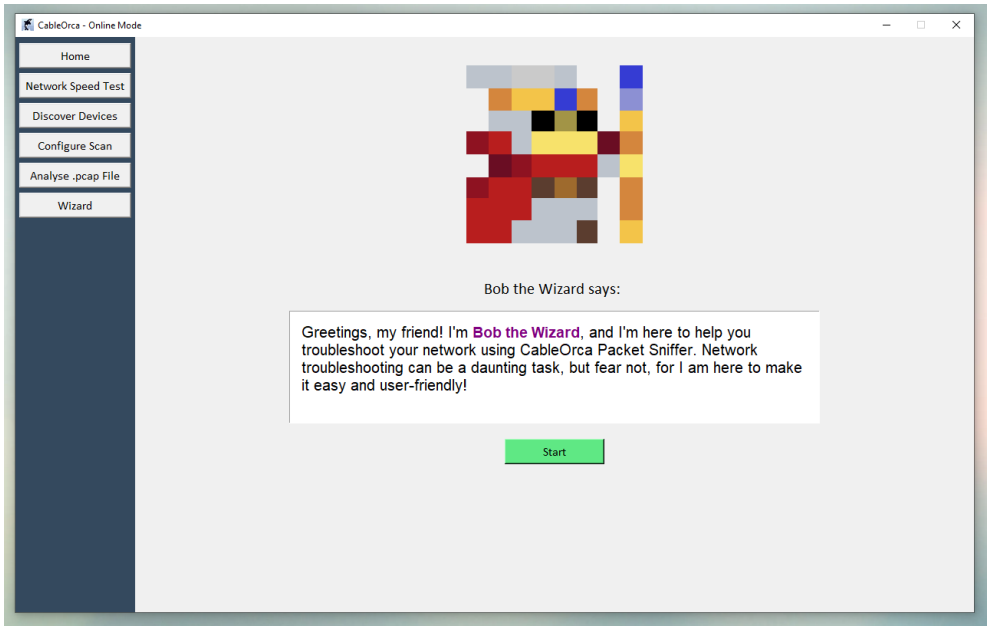

Figure 13 Network Diagnostic Wizard Introduction Screen

The Wizard, who I named Bob, is a friendly and approachable mascot in place to give the user something besides text to interact with when using the program.

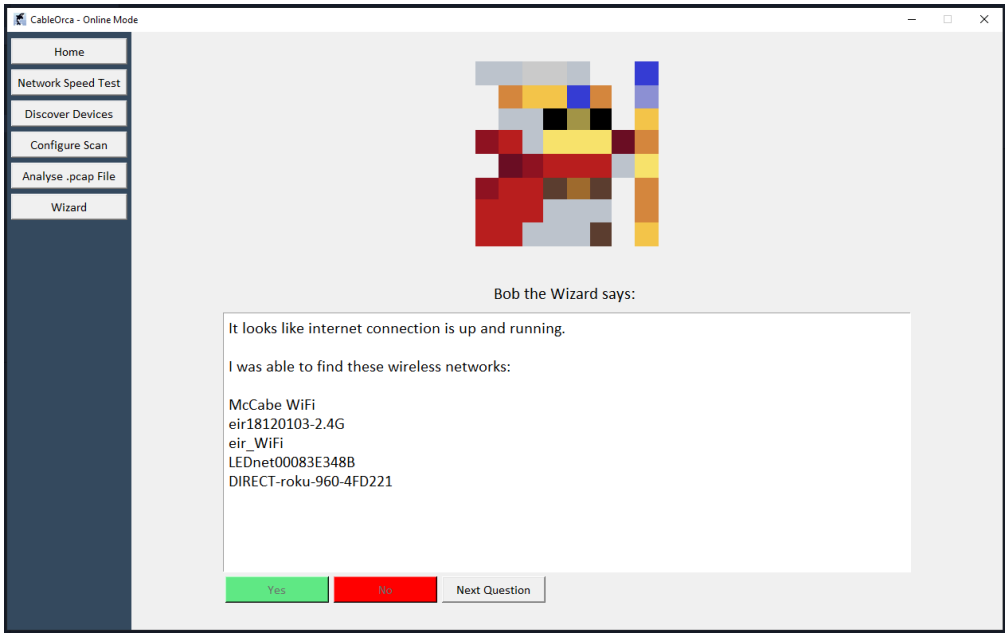

Figure 14 Wizard Checking Internet Connection

Figure 14 shows the wizard testing the internet connection. For this example, I added a TP-Link Wireless USB Adapter to my machine, allowing the application to discover wireless connections. The wizard pings various servers to test for internet connection. Multiple servers are used in case certain servers are temporarily unreachable. Users can navigate the wizard using the buttons below the next field which read “yes”, “no”, and “next question”.

Figure 15 Wizard Analysing a .PCAP File

Here is an example of the wizard analysing a file for a user. From here, a graph can be generated using the file provided to the wizard. This graph takes the top 10 most common addresses and created a bar chart. This chart can be saved as a file for future reference by the user. I have provided an example of a generated chart below using five seconds of network capture.

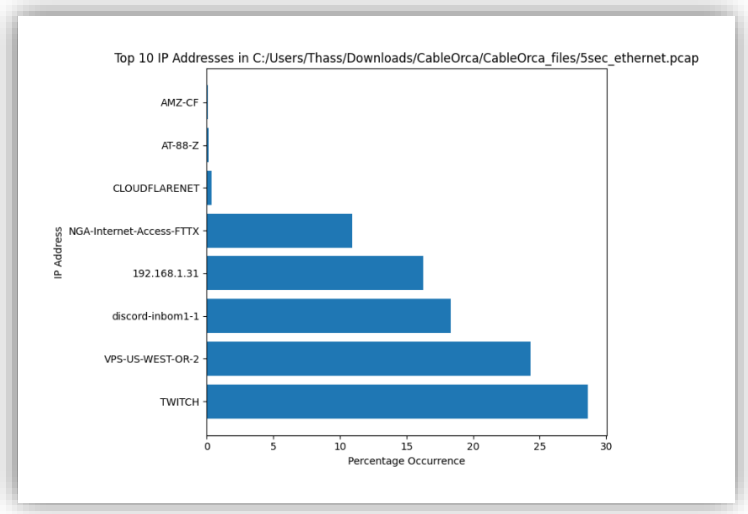

Figure 16 A Graph of "5sec_ethernet.pcap"

In this example, TWITCH makes up most of the traffic with just under 30% of the packet capture file being made up of TWITCH packets. The graph is automatically assigned a heading.

Overall, the wizard guides the user through each feature available in the application. These are internet speed tests, device discovery, packet capture, and file reading.

3. Description of Conformance to Specification and Design

The concept of the network diagnostics wizard was not something I had initially planned before starting development. However, implementing this feature allowed me to provide users with a way to make full use of the application regardless of technical knowledge. Furthermore, the implementation of a form of a digital tutorial expands on my initial goal of making the packet sniffer as “high-level” as possible.

Functional Requirements

These are the functional requirements I described in section 4 of the functional specification document. Here I highlight whether a certain requirement was added or not and I explain why a design may have had to change during development.

Functional Requirement 1.1

Title: Allow users to apply filters before and during packet capture.

Status: Not added.

One of the reasons I did not add filter configuration during and before network analysis was that I wanted to minimise parsing and processing done during live packet capture. I wanted to minimise this processing to keep the information that was being displayed to the user accurate to what was being captured in real-time by the application. By increasing the amount of parsing and processing during live packet capture, the slower the application would be when displaying live packets to the user. Instead, I decided that specific traffic should be analysed after a scan using an already saved packet capture file, instead of during a live capture.

Functional Requirement 1.2

Title: Allow users to generate graphs using analysis statistics

Status: Added

Users can generate and save a graph using packet capture data.

Functional Requirement 1.3

Title: Allow users to start and stop packet capture

Status: Added

While network analysis is configured using a timer, a manual start and stop button is implemented in the final application.

Functional Requirement 1.4

Title: Allow users to open existing .pcap files

Status: Added

Users can open .pcap files generated by CableOrca, or other existing packet capture applications.

Functional Requirement 1.5

Title: Allow users to save capture data to .pcap files

Status: Added

Users can save data to a .pcap file whether the timer for the scan duration has ended, or they have manually stopped analysis using the “stop” button on the application.

Functional Requirement 1.6

Title: Allow users to select a network interface to begin capturing packets on

Status: Added

Users can select what interface they wish to use for network traffic analysis by selecting an option from the generated list.

Functional Requirement 1.7

Title: Allow users to select all interfaces to begin capturing packets on

Status: Added

For all lists generated by the application, it will always include the ability to select all interfaces.

Functional Requirement 1.8

Title: Allow users to set a specific duration for packet capture

Status: Added

Before network analysis begins, users can enter the duration they wish to scan for in a text field.

Functional Requirement 1.9

Title: Automatically detect devices on a network and label them accordingly for the user

Status: Added

Using an ARP sweep, the application can discover and attempt to resolve the manufacturer names of devices found on the network. A network address is automatically determined.

Functional Requirement 1.10

Title: Automatically detect protocols on a network and label them accordingly for the user

Status: Added

Protocols are identified and displayed for the user when scanning live traffic and reading existing .pcap files.

Functional Requirement 1.11

Title: Automatically identify potentially malicious traffic on the network

Status: Not Added

My initial plan for this involved traffic pattern matching, like an intrusion detection system. I found this to be very challenging to implement in a way that would conform to my time restraints. Due to the wide range of variations malicious traffic can come in, I had to prioritise other core functions over it.

Functional Requirement 1.12

Title: DNS Resolution

Status: Added

Using the application's "socket translator" IP addresses can be resolved to respective names.

Functional Requirement 1.13

Title: Identify all network interfaces on the current device

Status: Added

The application can identify and resolve network interfaces to meaningful names for the user to select.

Functional Requirement 1.14

Title: Identify incompatible packet capture files

Status: Added

The application can identify incompatible packet capture files, such as those which do not contain any network capture data.

Functional Requirement 1.15

Title: Measure network speed using upload and download data

Status: Added

Ookla's speed test is used to determine the network speed.

Functional Requirement 1.16

Title: Provide the user with a graphical user interface (GUI)

Status: Added

The application utilizes Python's tkinter library to provide a graphical user interface for the user.

Functional Requirement 1.17

Title: Generate reports for the user

Status: Not Added

It took me a long time to learn matploitlib, consequently leaving me little time to implement a way to consolidate data and display it in a meaningful way to a user. A huge amount of data can be found on packet capture files as small as one second of network capture. This meant that I had to find the optimal way to consolidate this data while not failing to accurately portray it to a user.

Other Requirements

As indicated in my specification document, the requirements in this section are of a lower priority compared to the "core" features that have been previously identified.

Title: UNIX Support

Status: Not Added

UNIX support was not added because several features in my application rely on Python libraries which are only compatible with Microsoft Windows 10. One such example would be the library responsible for file dialogue when saving and opening files., which utilises Windows File Explorer. I would not have been able to implement UNIX support in the given time frame.

I felt it was acceptable to build the application with Microsoft Windows 10 support because this would be the most common operating system used by non-technical users at the time.

Title: Automatically save data upon a loss of connection to the network

Status: Not Added

This feature is not in the final application as it would add to the amount of processing and parsing done during network analysis. In this report, I have highlighted that I aimed to keep the amount of processing as low as possible during network analysis. Furthermore, I felt that the loss of an internet connection does not necessarily mean the loss of a connection to the network. Traffic will still be present on a local network if the internet connection is terminated.

Title: Allow for the customization of colours used to represent packets, devices, and protocols.

Status: Not Added

I found that how tkinter handles coloured output makes what I was trying to achieve very difficult. Tkinter uses "tagging" to colour output when it is added to a text field. There were many different colours I would need to tie to many different protocols. Furthermore, a large amount of processing would go into identifying and colouring output based on the contents of a packet. Due to the huge number of packets that can be analysed by the sniffer in just one second, I had to lower the priority of this feature, since a poor implementation of it would be detrimental to the performance of the application.

Title: Allow users to save and load packet capture filters

Status: Not Added

This feature was not added due to time constraints. I found it difficult to engineer an efficient way to filter traffic during live analysis. I had planned to save filters as a text file, which could be parsed by my application. An issue with this is the potential errors that could arise when stringing a wide range of different filters together.

Overall Conformance

While several features that were planned did not make it into the final application, these were mainly option or quality-of-life features that were never fundamental to the core functionality of the application. I was able to complete the main aspects that I planned during the planning phase of the project.

I underestimated the amount of work that would go into a system involving network traffic filtering. Consequently, I engineered the .pcap file parsing elements of the application to highlight and extract protocol and IP data during file reading. This meant that protocols and traffic types, such as private and internet traffic, were inspected without the filtering system initially planned.

Additionally, I would have liked to have expanded on the application's ability to generate graphs using packet capture file data. This proved to be challenging with one cause being the sheer amount of data that can appear in packet capture files. Ideas I had involved calculating averages and frequencies of IP addresses, protocols, and other various aspects of network packet capture. Just three seconds of network capture however can have hundreds if not thousands of packets in it, making it challenging to portray this data to a user in a meaningful way. An example of a graphing feature I could not implement was the use of pie charts. This was because network traffic typically comprised many packets from multiple devices, rather than numerous instances of a small number of IP addresses. This made graphs generated by matplotlib very difficult to read and create.

4. Learning

Technical

My two main choices for programming languages were C++ and Python. These programming languages are extremely different with C++ being very low-level, and Python being a high-level language. At its core, I felt the decision came down to which language would be most appropriate for building a graphical user interface in the given amount of time because a key part of my application was de-mystifying technical information for non-technical users. This is where Python stood out with its GUI library called Tkinter. In addition to this, Python had a very powerful packet capture and manipulation library called Scapy.

In my research document, I described my findings when comparing Scapy to C libraries such as libpcap, and I found that Scapy would be perfectly adequate for the tool being built. I had never worked with Scapy before, but I was comfortable with Python due to past projects and scripting experience.

Before starting the project, not only was Scapy new to me, but so was Tkinter. As a result, part of my planning phase involved deciding if I was going to use a GUI drag-and-drop builder or learn how to manually write code with Tkinter myself. I decided to manually write the code with Tkinter. One reason for this decision was the fact that in my experience, GUI builders would generate a lot of code that I did not understand. Furthermore, I felt that writing the GUI without a builder would allow me to tune it to exactly what I was looking for, rather than shaping the application around a starter template.

As a result of completing the project, I am extremely comfortable with both Scapy and Tkinter. Using Scapy I was faced with tasks such as analysing and creating network packets. With Tkinter, I learned different ways to display information to users using labels text fields and file dialogue popups. Additionally, I learned how Tkinter works at a high level with its packing and widget system. Both Python libraries were very well documented with code examples of basic usage available.

Matplotlib is a Python library that allows developers to visualise data. I used this to generate graphs for users who analyse packet capture files. I had never used this library before and found it difficult to learn. However great documentation and tutorial were available on for example matplotlib.org. I thought the library was very impressive and I quickly learned why it was popular among data analysts despite only learning the basics of the library at a very high level.

A technical issue I faced a lot of difficulty with was threading. When using threads, I found it difficult to learn how they work and what the best way to manage, terminate, and join them etc. was. Errors that would arise from threading would be detrimental to development, as threading errors would completely seize the application or cause it to behave in unexpected ways. While threading was difficult to learn, it was important to implement to achieve an adequate graphical user interface. I did not expect threading to be as much of an issue as it was. Finishing the project has given me a high-level understanding of how threads work and what the best practices are regarding using them.

Development

During development I researched different technologies I could use to improve my code quality. I used Pylint and Autoflakes. Pylint is a static code analyser that would inspect my code and point out issues and tweaks I should make to it using a command line output. Autoflakes was used to find unused imports and variables and general code improvements. I had never used or heard of these technologies despite having worked with Python before.

My GIT version control skills greatly improved as I was tasked with making various branches and restoring points during development, additionally, I used GitHub to back up and pull the latest versions of the code from the cloud when switching development environments for example my laptop on campus, and my desktop at home.

A technology I had not used when developing with Python before was Jupyter Notebook. I ran instances of Jupyter Notebook in Visual Studio Code, and it allowed me to run snippets of Python code when testing libraries and functions.

During development, I encountered countless situations where I would need to handle errors. An example of this would be attempting to display information that requires fetching data from the internet. I had to make sure I accommodate situations where that data could not be fetched, or the data fetching process is interrupted for example. Thankfully I had experience in error handling with Java because of my college courses, and applying this knowledge to Python did not present an issue. I had to ensure that the application stayed running despite non-fatal errors. Additionally, I had to accommodate user miss input such as cancelling the file-saving process or incorrect input. I managed this by for example disabling buttons and adding Boolean checks for the users' activity.

As the amount of code grew, I found it difficult to document and maintain it. I created several files to hold their dedicated functions for easier development. I had trouble deciding on a set way to build and destroy windows during application runtime. Furthermore, I found it hard to settle on a set "flow" for each window on the application. Some tasks would require little to no input from the user whereas a network scan configuration, for example, would require several. In the project, I create classes with windows and spawn them on the main window in a select few scenarios. I feel that this would have been the optimal solution to implement all features of the application and not just a select few.

A general technical achievement I earned was improved Python skills with new capabilities for building graphical user interfaces. I would be comfortable making more GUIs for Python applications, as well as writing better formatted and more efficient object-oriented Python. I also know a lot more about packet capture files and libraries. My only prior experience with network packet capture was the occasional need for Wire Shark during college work. This project has caused me to research deeper into the concept of packet sniffing and various formats of packet capture files.

Personal

A personal learning aspect from this project was the importance of time management. I had to dedicate a set number of hours to work on the project each week while balancing all the other modules during the final year. It was key for me to know the weighting of different assessments compared to aspects of the final year project and adjust the amount of time dedicated to certain tasks accordingly. Additionally, I learned that it's remarkably helpful to prototype all forms of work before implementing it, whether that be code or documentation. By deciding on aspects like code structure before development, better code consistency can be maintained. Applications I develop in the future will be planned out beforehand with code structure clearly defined and strictly followed. When writing the code for the final year project I was not as consistent as I would have liked to have been when writing aspects of the code.

A key aspect of my project was de-mystifying technical information for non-technical users. Consequently, I regularly had to approach the output of the program as someone with little networking and computing concept knowledge. I gained an understanding of how to recognise the primary audience for the output of my program, and how best to communicate program output to that target audience. Problem-solving was key to taking technical information and shaping it into non-technical terms without invalidating it.

During the project, I had to read a lot of documentation and I had to run a lot of sample pieces of code to try and understand them. I feel confident in my ability to approach new programming libraries and implement them in future projects.

5. Review

This section delves deeper into analysing the successes and challenges encountered during the project and outlines my revised approach if I were to start the project again.

Difficulties

Out of Date Documentation

In the early stages of planning, I was experimenting with different languages and libraries. When testing libpcap with C++, I found that the main documentation I had for the library was out of date at the time. I was unsure of what was causing my programming error before this.

Threading

I found that the development of a reporting system for users' packet capture files was hindered due to the requirement of threading. Matplot lib presented incompatibility with tkinter and I found it difficult to find a solution for this. Eventually, I discovered that Matplot lib can be imported into a project in various "modes". My solution was to implement a version of Matplotlib that was designed to write to files. My current implementation of Matplotlib was incompatible with GUIs.

Secondly, a key part of my application was analysing, and viewing traffic in real-time. To display information on a graphical user interface, you must run the graphical user interface in one thread while building and displaying the information in another thread. With this, I wanted to allow the user to stop network analysis at any time using a "stop" button. A challenge presented was stopping a thread while it was running. I found joining the thread and checking for a "stop" flag during runtime difficult to do in the way my threading was implemented.

Handling Errors

A challenge I found when developing a graphical user interface with many widgets was handling possible errors that could be caused by the user or application itself. I found it hard to keep track of all the widgets I had to disable if certain conditions were met or text fields that should be enabled if their output needs to be modified. I found that during runtime, the state of widgets on the GUI would often need to change depending on what is happening or what the user is doing. Documenting the code and highlighting state changes for different widgets assisted in this implementation.

Standardised Code

During development, I realised that it would have been best to take an object-oriented approach to each feature of my application. This would have made both code structure and code maintenance easier. This would have required me to gain a strong understanding of object-oriented Python before development.

Performance Issues

Certain tasks I wanted to carry out during the runtime of the application, for example fetching the user's public IP address would take too long with certain solutions. Issues arose with this because the information that was to be displayed on the GUI would have to be set before it is displayed. I fetch information from various parts of the internet such as Amazon Web Services for public IP addresses, and Ooklas speed test servers for testing upload and download speed. It took several iterations before adequate solutions were found for information gathering.

Blank Analysis Termination

An issue I encountered was with the network analysis window when used on interfaces with no network activity. In my environment, an example of such an interface would have been a virtual box interface on my desktop. I found that when starting analysis on interfaces that were not detecting any network activity, the stop button for ending the thread would no longer function.

I observed that this was because for each packet detected, a flag was checked to determine if the analysis should continue or not, however with no packets being detected, this flag was never checked. Despite many attempts, I could not find a solution to solve this problem as the thread failed to terminate when I needed it to. This was not detrimental to the overall functionality of the application.

Although I considered implementing timeouts to address this issue, I realized that the timeout flag had already been utilized in Scapy's sniff() function. This flag is either set by the user or set by the application at runtime as default.

What Went Well

Scapy Library

One of the primary factors that contributed to the success of CableOrca, the high-level packet sniffer was the power of the Scapy library. Not only did Scapy excel at packet capture and manipulation, but I found it was the optimal solution for identifying network interfaces, IP addresses, and hostnames of devices. There exist many Python libraries that do these tasks, but with Scapy providing adequate capabilities, it meant I could keep the number of library imports into my project to a minimum.

Language Choice- Python

I found Python to be a brilliant language to work with during the project. Libraries existed for different tasks I needed to carry out, additionally, many questions about Python issues were already discussed online on sources such as Stack Overflow. Python being a high-level language means it can almost be read like basic English.

Working at a low level with a device’s hardware was out of scope for my project, therefore, the ability to quickly prototype and experiment with different Python functions was very useful during development. Access to resources such as Jupyter Notebook encouraged this experimental and trialling approach to code.

I had previously stated that I worked with Python before, making personal projects and tools which is what further swayed my decisions when choosing a language during the project planning phase.

Use of GIT and Visual Studio Code

GIT and GitHub were key to the development process. Additionally, I modified my Visual Studio Code to allow for Python development and Jupyter Notebook files. This allowed me to efficiently run and test code as well as run Pylint and Autoflakes on my code to test its quality.

GIT version control is what allowed me to make various branches and commits when adding new features. It essentially allowed me to place "checkpoints" in my code that I could revert to if I ever needed. Furthermore, this made switching from development at home on my desktop to on-campus using my laptop very easy, using Git pushing and pulling. Git also prevents data loss, with code existing on the cloud as well as other devices besides the main development environment.

Outstanding & Missing Features

These are the main features I feel I would have liked to implement but could not due to various reasons.

Filters

An expanded packet filtering system could not be implemented. I found that aspects of the code which I did not expect to take a long time to learn, such as threading, ended up delaying my schedule.

Better Reporting System

I described difficulties with Matplotlib previously in section five. I found this library to be the optimal way to create reports for the user and if given more time to understand it, I could have created a new menu for reports and data visualisation.

.pcap files are very complex with huge amounts of data, meaning parsing different information out of them is a very time-consuming process.

UNIX Support

I found that Linux is slowly but surely becoming more adopted by non-technical computer users and hoped to bring my application to a point where it’s compatible with this operating system. A primary factor that prevented this was the use of the “OS” library during development, as well as the method I used to save and download files to the user’s system. My implementation largely relied on Microsoft Windows 10.

Conceptually, the high-level packet sniffer would be used by Microsoft Windows 10 users, however, making this a non-detrimental factor to the overall application.

If Starting Again

Here, I provide a retrospective analysis of the project and offer recommendations for someone embarking on a similar project in the future, outlining the key areas that I would approach differently if allowed to start this project again.

Object-Oriented Approach & Better Structure

If starting again, I would better structure my code and decide on a best practice while strictly sticking to it. Not only would this improve code readability, but it would greatly improve code maintainability and consequently, code efficiency.

In Figure 17, each box would be its own individual file, with files containing classes and functions related to their implemented feature. This approach would make the code very modular. File names and function naming would be key to organising a structure like this. Additionally, error handling for each implemented function for different features would be important to keep the application from crashing during runtime.

The creation and destruction of objects in my code could have been done a lot better. For example, when collecting information, I want to retain during runtime, I should have had dedicated classes to hold this information. Similarly, dedicated classes or files should be dedicated to the information I want to use globally. An example of this type of information would be the users' default network interface. This information would only need to be discovered once, perhaps at runtime, but would be used in many places throughout the application.

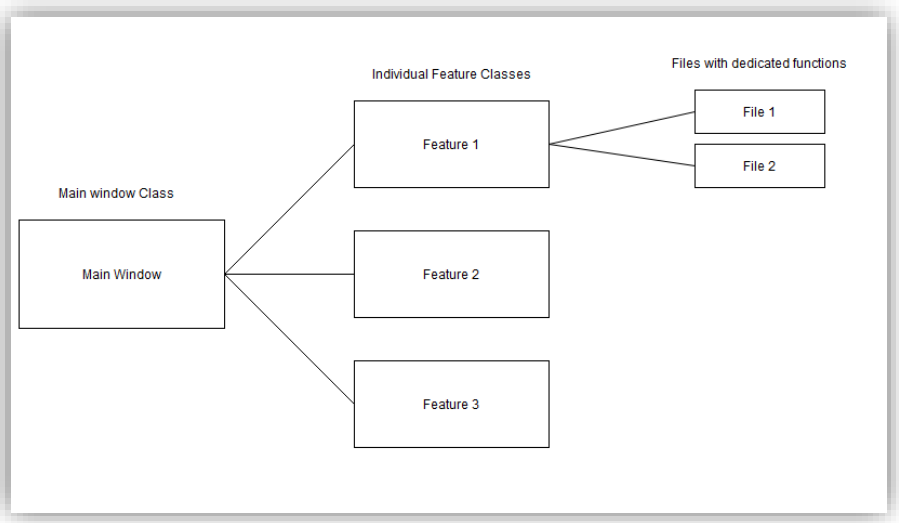

Figure 17 An Approach I Would Take if Starting Again - Diagram

Better Error Handling

I would have liked to have gained a better understanding of how to manage threads in Python. This would have allowed me to avoid errors relating to mishandled threads during testing. I found that I ended up needing threads for more tasks than I expected throughout the development process.

Better Performance

My implementation of a network speed test was very slow at times, with tests taking between 20 seconds to one minute. I think it would have been better to engineer a faster way of testing a user's internet speed.

6. The Result

Although there were some aspects that I could not delve into as much as I would have liked, and certain features that I was unable to implement, I believe that the project can be deemed a success. Considering the limited timeframe and the need to juggle other final-year modules alongside this project, I am satisfied with the overall outcome.

As per the specification of my project, the goal was to make a packet sniffer usable by anyone regardless of technical skills. I feel that with the implementation of features such as the simple GUI, high-level analysis, and diagnostics wizard which can output a high-level report, I have achieved this.

7. Acknowledgements

I would like to give a special thanks to Paul Barry, my project supervisor who gave me continuous guidance and answered my questions throughout the entire project. Paul's advice was fundamental to the development of CableOrca and keeping its development on track and schedule.

I want to thank my friends for supporting me throughout the entire duration of the college course.

I want to thank all my lecturers who have equipped me with the knowledge required to develop a project like this throughout my final year, as well as all previous years. It is thanks to them that I can apply what I have learnt in a practical setting. I found that all lecturers were more than happy to answer my questions no matter what it was or when I asked them. For that, I am truly grateful.

8. Declaration

PLAGARISM DECLARATION

- I declare that all material in this submission e.g., thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams, or other material, including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: Thassanai McCabe

Student Number: C00250439

Date: 17/04/2023

9. References

Rouse, M. (2017). *Wizard*. [online] Techopedia. Available at: https://www.techopedia.com/definition/32108/wizard-software [Accessed 10 Apr. 2023].

Bonderud, D. (2020). *Why Use ARP Scan*. [online] Cybrary. Available at: https://www.cybrary.it/blog/why-use-arp-scan/.