NETWORK TRACKING USING WIRESHARK AND PYTHON

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE 302: COMPUTER NETWORKS

Submitted by

THEIVARAYAN IYYAPPAN S

(Reg. No.: 124006080, B.Tech. Electronics and Instrumentation)

DECEMBER 2022

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

THANJAVUR, TAMIL NADU, INDIA - 613 401

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

Bonafide Certificate

This is to certify that the report titled **"Network Tracking using Wireshark and Python"** submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B.Tech. is a bonafide record of the work done by **Shri.Theivarayan Iyyappan S (Reg. No.: 124006080, B.Tech. Electronics and Instrumentation)** during the academic year 2022-23, in the School of Electrical and Electronics Engineering.

Project Based Work *Viva voce* held on _____

Examiner 1 **Examiner 2**

Table of Contents

LIST OF FIGURES

ABBREVIATIONS

KML	Keyhole Markup Language
Pcap	Packet Capture
Dst	Destination
Src	Source
Dpkt	Dynamic Public Key Technology
Venv	Virtual Environment

ACKNOWLEDGEMENTS

Firstly I express my gratitude to Prof. Dr S Vaidhyasubramaniam, Vice Chancellor, SASTRA Deemed University, who provided all facilities and constant encouragement during my study. I would specially thank and express my gratitude to our mentor Dr. V Elamaran, Assitant Professor, School of Electrical and Communication Engineering for providing mean opportunity to do this project and for his valuable input, guidance and support to successfully complete the project. I also thank all the Teaching and Non-teaching faculty, and all other people who have directly or indirectly helped me through their support, encouragement and all other assistance extended for completion of my project. Finally, I thank my parents and all others who helped me acquire interest in the project and aided me in completing it.

ABSTRACT

Data that has travelled around the globe in a specific amount of time is referred to as network traffic. So, in order to visualise the network traffic, we will now use Google Maps. To make it simpler for the user to recall or mentally associate the location, we have re-visualized the data in a map rather than just reading it. We will first examine the tools that will enable us to create the path before moving on to the process. So, instead of reading a number, text or data we will visualize it as a path in this project

INTRODUCTION

Network traffic is the amount of data that travelled across the world in a given time. So now we are going to visualize the network traffic in Google maps instead of reading the data we are visualizing it in a map so that it will be easy for the user to remember or get the location in mind before going to the working part we will look at the tools which will help us to produce the path.

1.1. Geolitecity Database

We need to download the GeoLiteCity database as this will be used to translate a IP address into a Geo location(longitude & latitude).

1.2. Wireshark

The wireshark tool is needed to capture the network traffic. The captured traffic will act as input to our Python script and will be the data displayed at the end using Google Maps.

1.3. Python

We need python to have the final code. We use the latest version of python along with pycharm ide to add libraries and to locate files easily.

1.4. Notepad++

We use notepad++ to convert the python code to .kml file.

WORKING

In this part we'll see how the captured traffic is converted into a .kml file and fed into Google map.

2.1. Capturing the network traffic

In this part we should create a input data which will consist of a captured pcapfile. At first we have to open wire shark and select a interface which has traffic going through it.

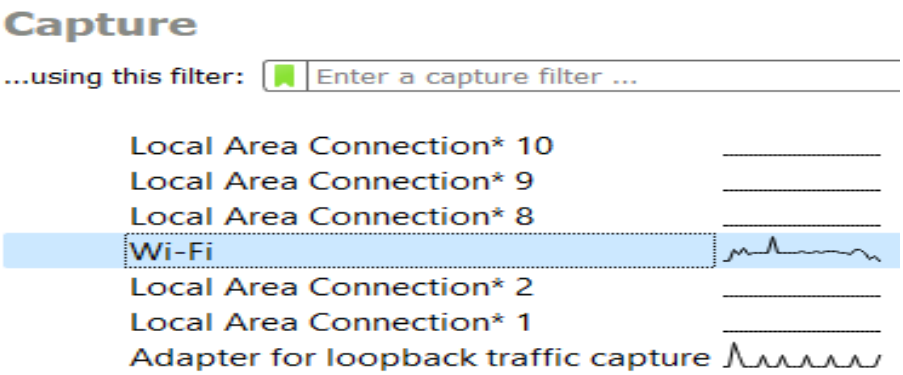

Fig 2.1. Captured pcap file

You are immediately prompted with network traffic because Wireshark automatically starts a fresh capture when you choose an interface. You must take the following actions to halt data collecting and save the information:

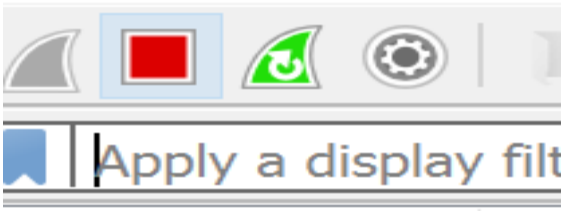

Fig 2.2. (contd.)

After stopping the capture, you must export the data in pcap format. To accomplish this, select File -> Export Specified Packets, and then choose the following option:

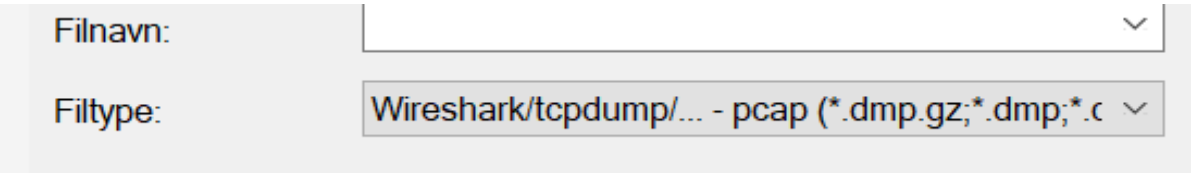

Fig 2.3. (contd.)

We may now proceed with the Python Implementation because the recorded data has been saved in the proper format.

2.2. Python Implementation

To begin, we must import the necessary libraries, which are as follows:

```
import dpkt
import socket
import pygeoip

gi = pygeoip.GeoIP('GeoLiteCity.dat')
```

Fig 2.4. Python Libraries

With the GeoLiteCity database that we downloaded before, as variable is declared. It must be placed in the Python project's root folder.

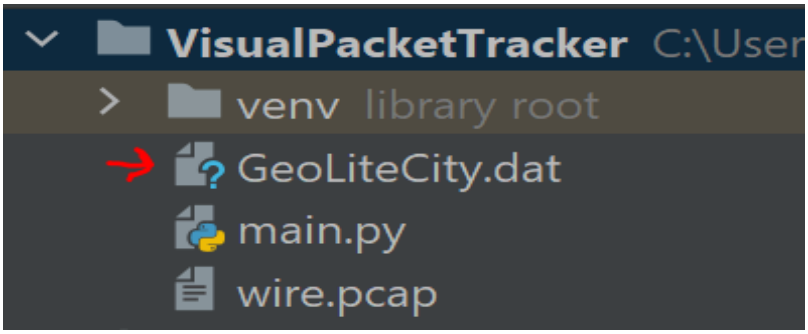

Fig 2.5. GeoLiteCity Database

The main method will open our captured data after we've finished, and it will also create the header and footer of the KML file—the output file that we'll upload to Google Maps—as well as open it.

The code below contains the tags <style> and</style> that allow us to alter the styling of the lines that Google Maps will create.

```
def main():
    f = open('wire.pcap', 'rb')
    pcap = dpkt.pcap.Reader(f)
    kmlheader = '<?xml version="1.0" encoding="UTF-8"?> \n<kml
xmlns="http://www.opengis.net/kml/2.2">\n<Document>\n'\
    '<Style id="transBluePoly">' \
                '<LineStyle>' \
                '<width>1.5</width>' \
                '<color>501400E6</color>' \
                '</LineStyle>' \
                '</Style>'
    kmlfooter = '</Document>\n</kml>\n'
    kmldoc=kmlheader+plotIPs(pcap)+kmlfooter
    print(kmldoc)
```

Fig 2.6. Styling of Lines

The application will loop through our pcap data in the code below plot IPs(pcap), extracting the source and destination IP addresses of each network packet that was captured.

```
def plotIPs(pcap):
    kmlPts = ''
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            dst = socket.inet_ntoa(ip.dst)
            KML = retKML(dst, src)
            kmlPts = kmlPts + KML
        except:
            pass
    return kmlPts
```

Fig 2.7. Plot ip

We must first connect a Geo location to our IP addresses in order to use them as input for Google Maps. The approach described below first locates the geolocation of each IP address before converting the data into KML format.

```
def retKML(dstip, srcip):
    dst = gi.record_by_name(dstip)
    src = gi.record_by_name('x.xxx.xxx.xxx')
    try:
        dstlongitude = dst['longitude']
        dstlatitude = dst['latitude']
        srclongitude = src['longitude']
        srclatitude = src['latitude']
        kml = (
            '<Placemark>\n'
            '<name>%s</name>\n'
            '<extrude>1</extrude>\n'
            '<tessellate>1</tessellate>\n'
            '<styleUrl>#transBluePoly</styleUrl>\n'
            '<LineString>\n'
            '<coordinates>%6f,%6f\n%6f,%6f</coordinates>\n'
            '</LineString>\n'
            '</Placemark>\n'
        )%(dstip, dstlongitude, dstlatitude, srclongitude,
srclatitude)
        return kml
    except:
        return ''
```

Fig 2.8. geolocation

We will get a complete KML file printed to the terminal. What is left to do is to copy this data into a file and give it a *.kml* extension using notepad++.

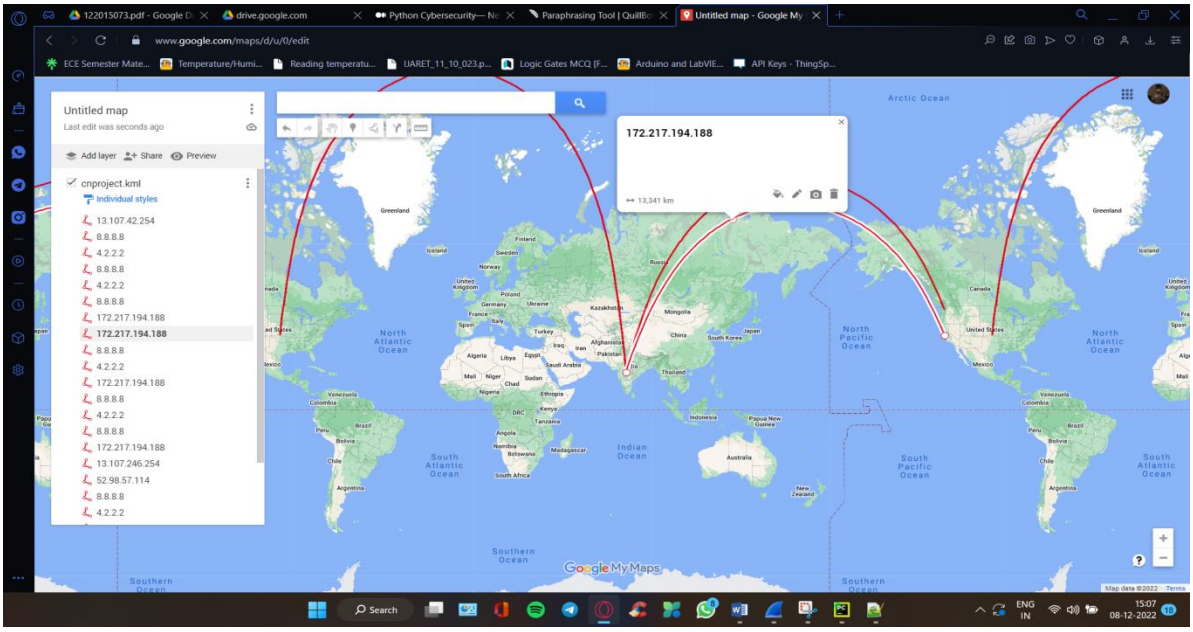

Fig 2.9. Visual path of the network traffic

After getting kml file we should upload it into Google my maps and finally we can see the visual path of our network traffic.

MERITS OF THE PROJECT

1) Instead of reading the data we are visualizing it as a map.

2) Availability of addresses in left side of the map so we can locate the address easily.

3) Flexibility to change the design of the path produced.

DEMERITS OF THE PROJECT

1) We can’t upload a file with huge data which exceeds the limit of data accepted in Google maps.

SOURCE CODE

```
import dpkt

import socket

import pygeoip


gi = pygeoip.GeoIP('GeoLiteCity.dat')


def retKML(dstip, srcip):

dst = gi.record_by_name(dstip)

src = gi.record_by_name('14.139.181.234')
   try:
dstlongitude = dst['longitude']

dstlatitude = dst['latitude']

srclongitude = src['longitude']

srclatitude = src['latitude']

kml = (
        '<Placemark>\n'

        '<name>%s</name>\n'

        '<extrude>1</extrude>\n'

        '<tessellate>1</tessellate>\n'

        '<styleUrl>#transBluePoly</styleUrl>\n'

        '<LineString>\n'

        '<coordinates>%6f,%6f\n%6f,%6f</coordinates>\n'

        '</LineString>\n'

        '</Placemark>\n'
```

```
        )%(dstip, dstlongitude, dstlatitude, srclongitude, srclatitude)
        return kml

    except:
        return ''

def plotIPs(pcap):

kmlPts = ''

for (ts, buf) in pcap:
            try:
eth = dpkt.ethernet.Ethernet(buf)

ip = eth.data

src = socket.inet_ntoa(ip.src)

dst = socket.inet_ntoa(ip.dst)

                KML = retKML(dst, src)

kmlPts = kmlPts + KML
            except:
                pass
        return kmlPts

def main():

 f = open('cnproject.pcap','rb')

pcap = dpkt.pcap.Reader(f)

kmlheader = '<?xml version="1.0" encoding="UTF-8"?>

\n<kmlxmlns="http://www.opengis.net/kml/2.2">\n<Document>\n' \


                '<Style id="transBluePoly">' \

                '<LineStyle>' \

                '<width>1.5</width>' \

                '<color>501400E6</color>' \

                '</LineStyle>' \

                '</Style>'
```

```
kmlfooter = '</Document>\n</kml>\n'

kmldoc = kmlheader + plotIPs(pcap) + kmlfooter


print(kmldoc)

if __name__ == '__main__':
main()
```

SNAPSHOTS

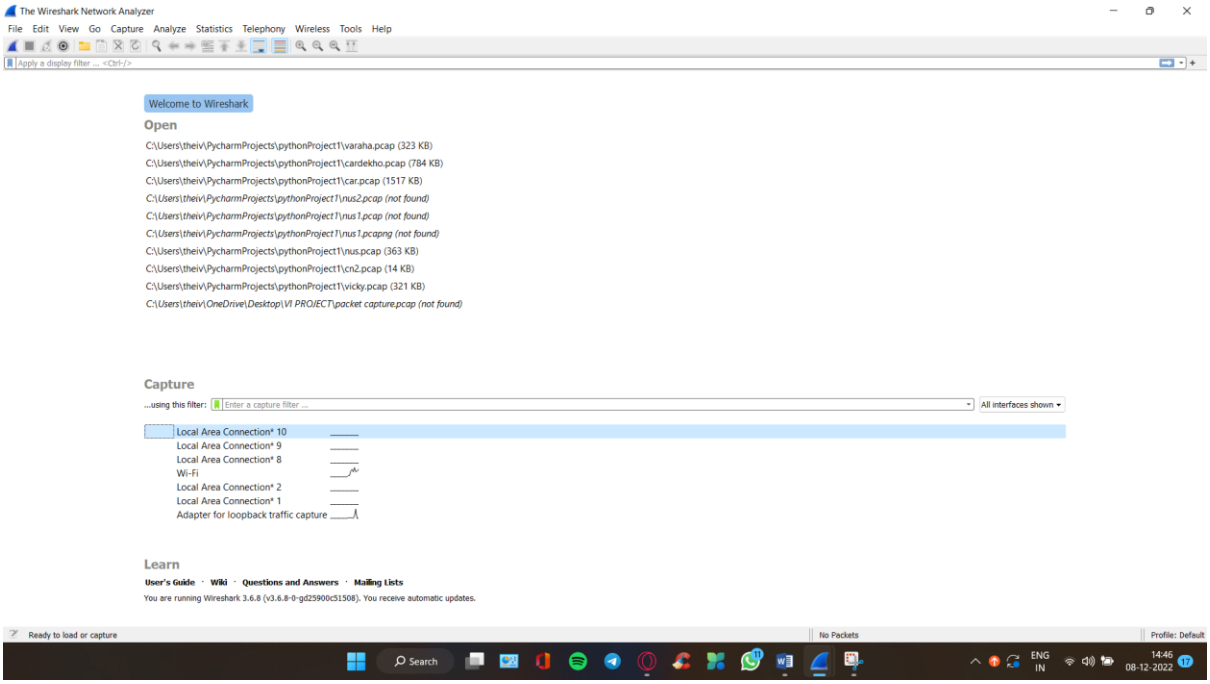

Fig 5.1. Selecting the interface which has traffic going through it.

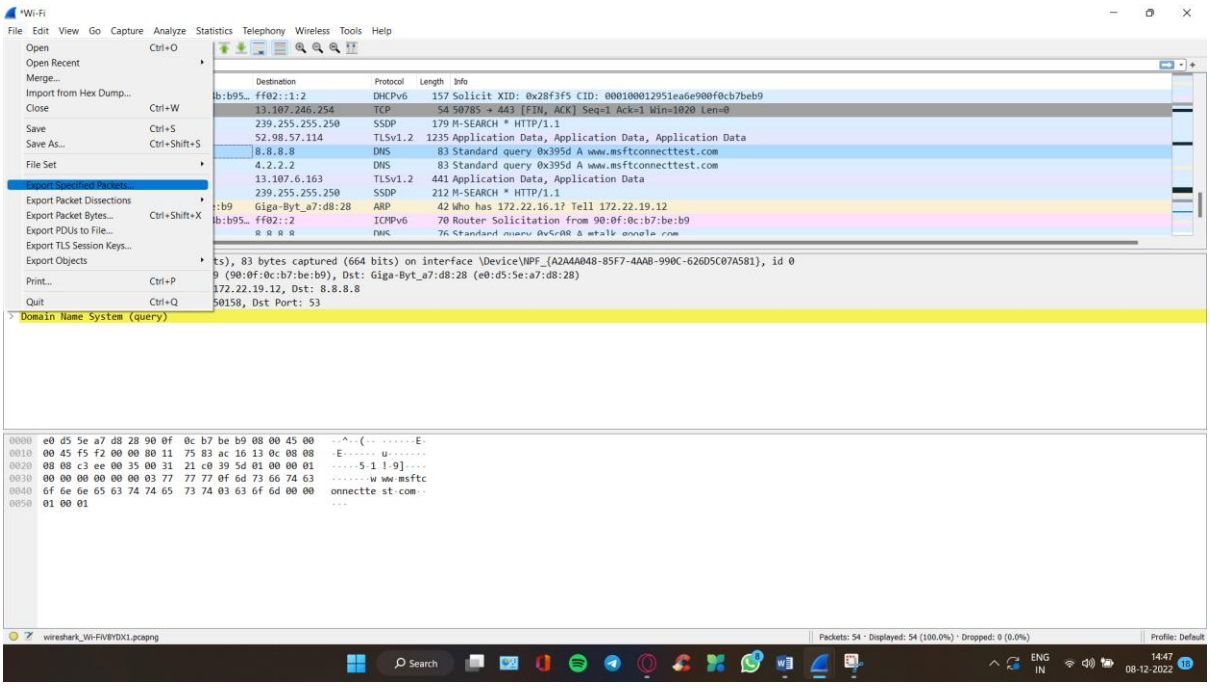

Fig 5.2. Stopping the traffic**.**

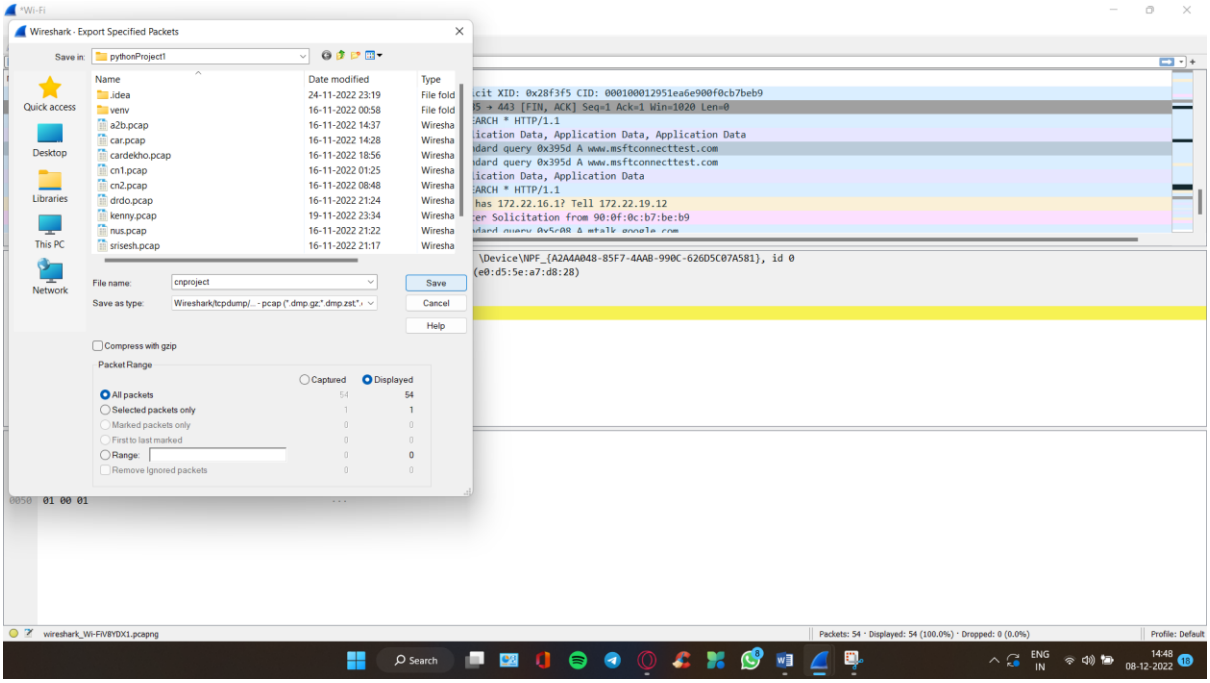

Fig 5.3. Saving it as a pcap file**.**

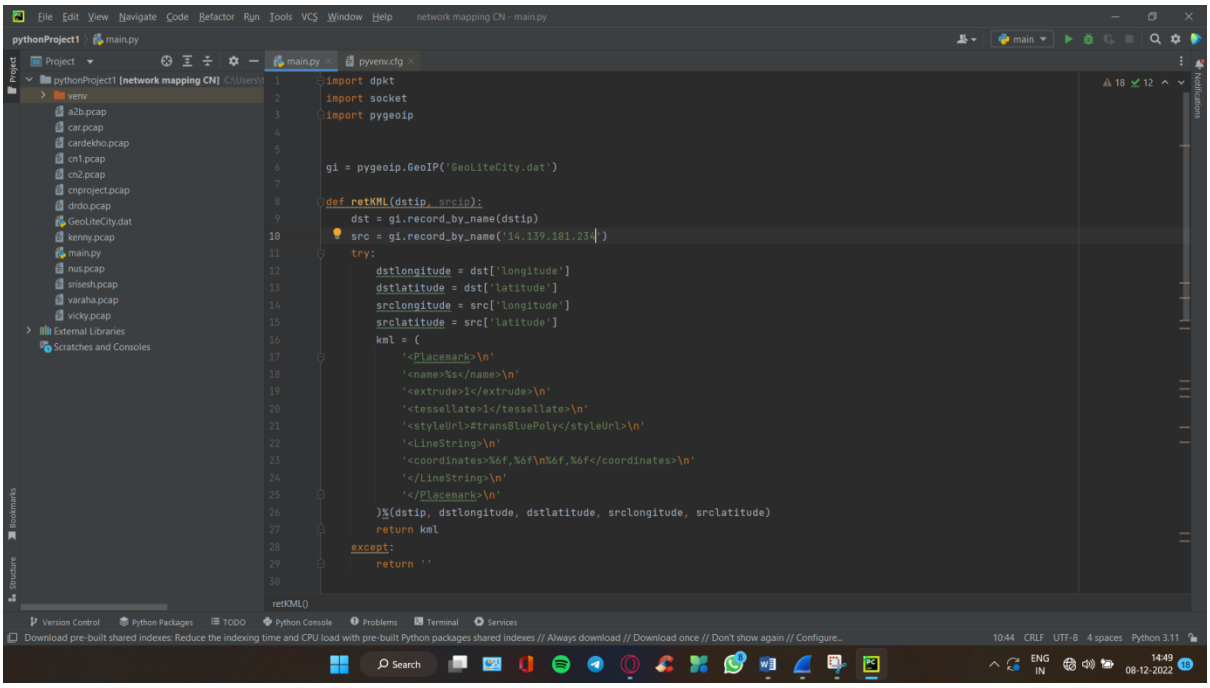

Fig 5.4. Python Code that we use to generate the address in .kml format.

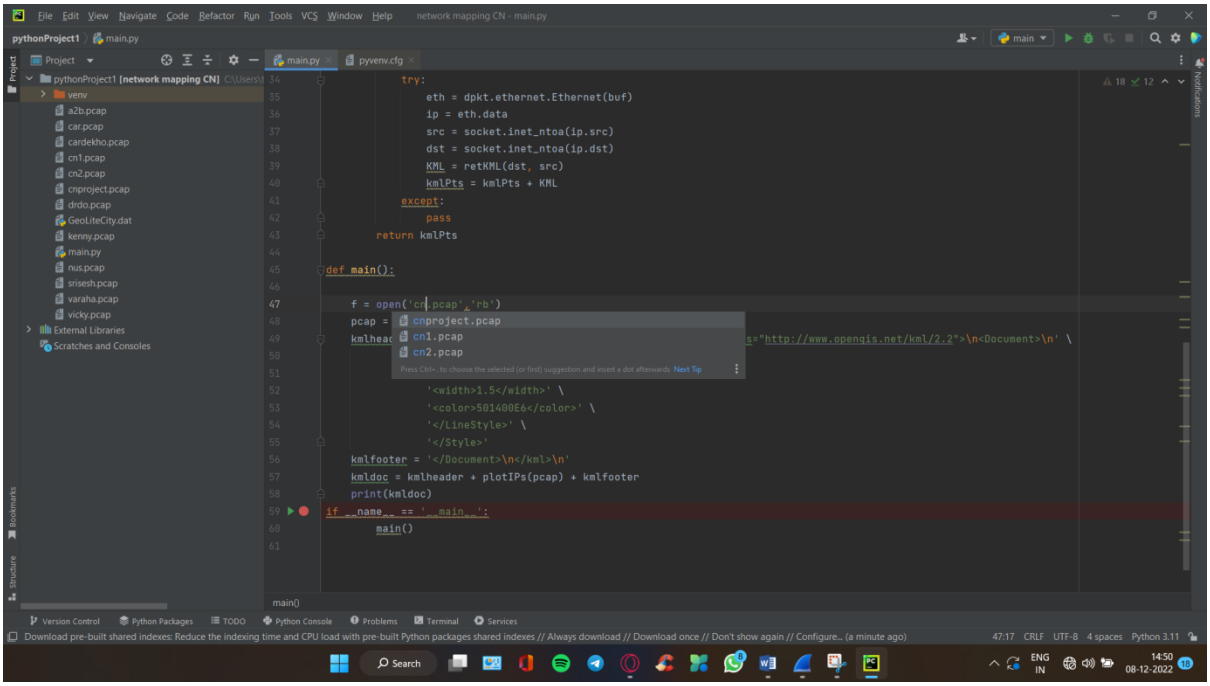

Fig 5.5. Typing the required pcap file.

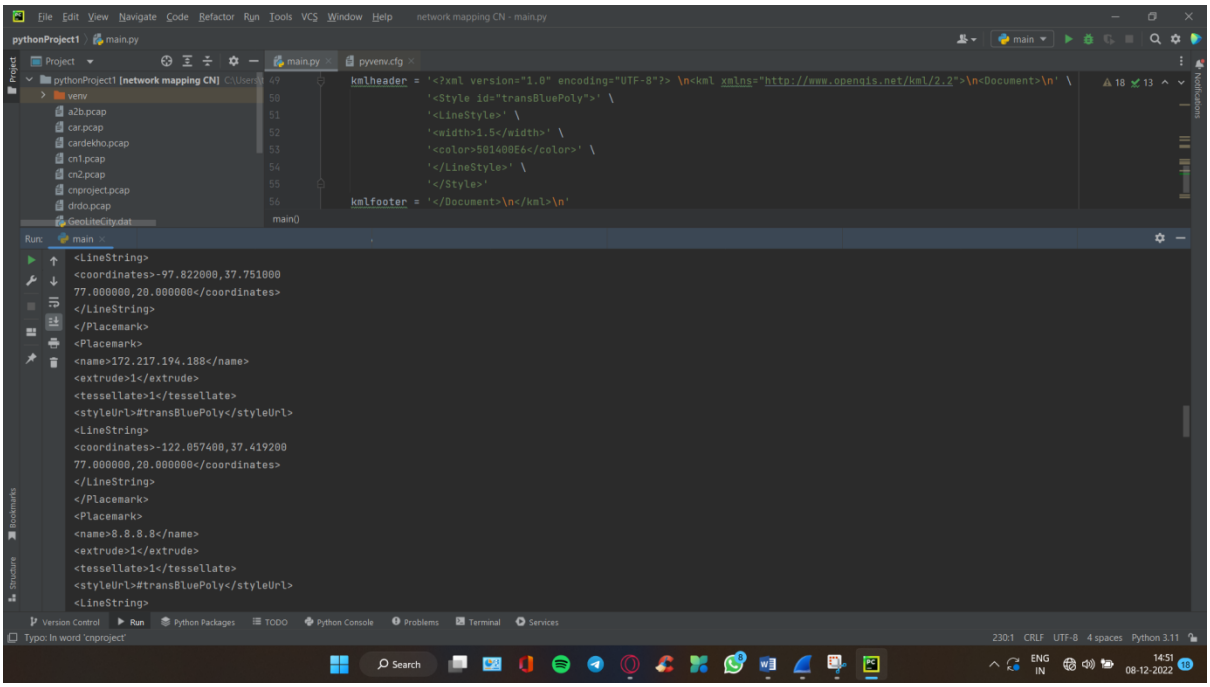

Fig 5.6. Output of the python code.

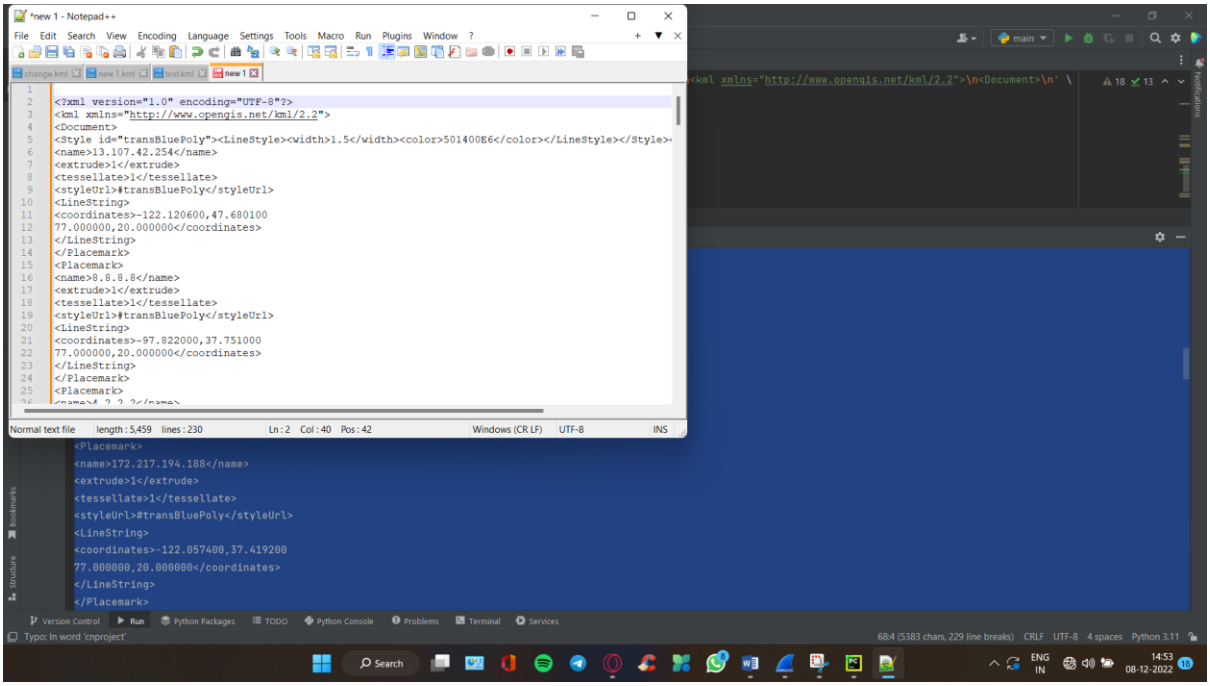

Fig 5.7. Pasting the output in notepad++.

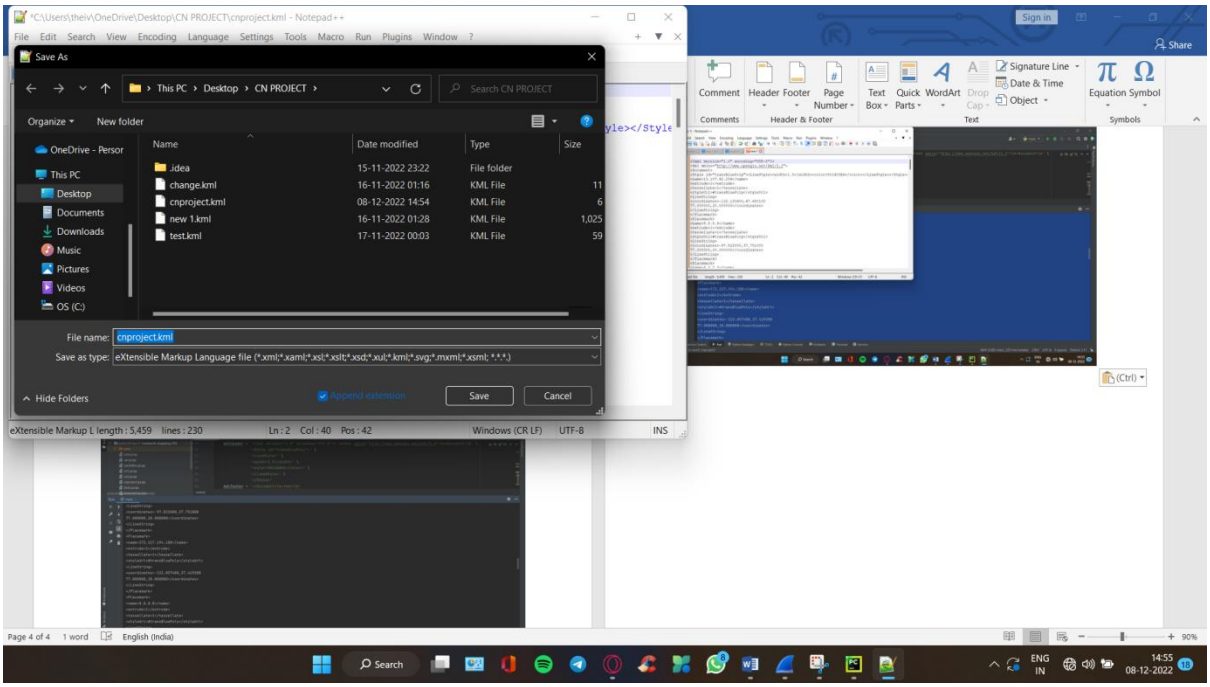

Fig 5.8. Saving it as a .kml file.

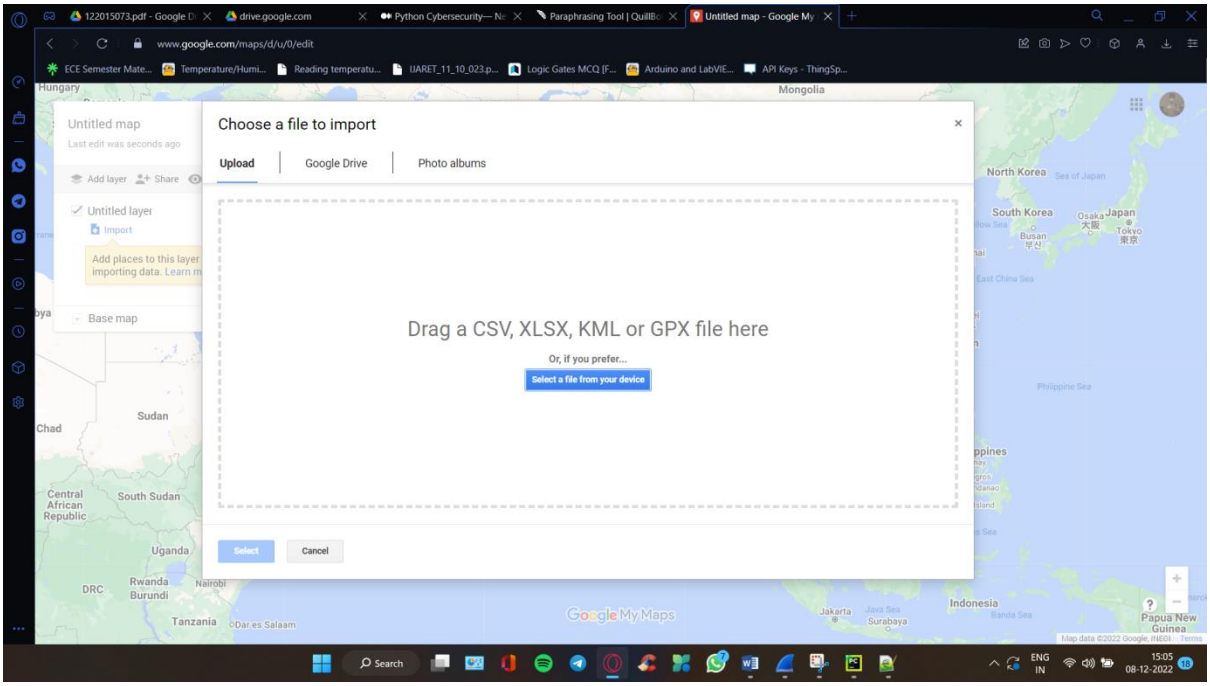

Fig 5.9. Uploading .kml file in google maps.

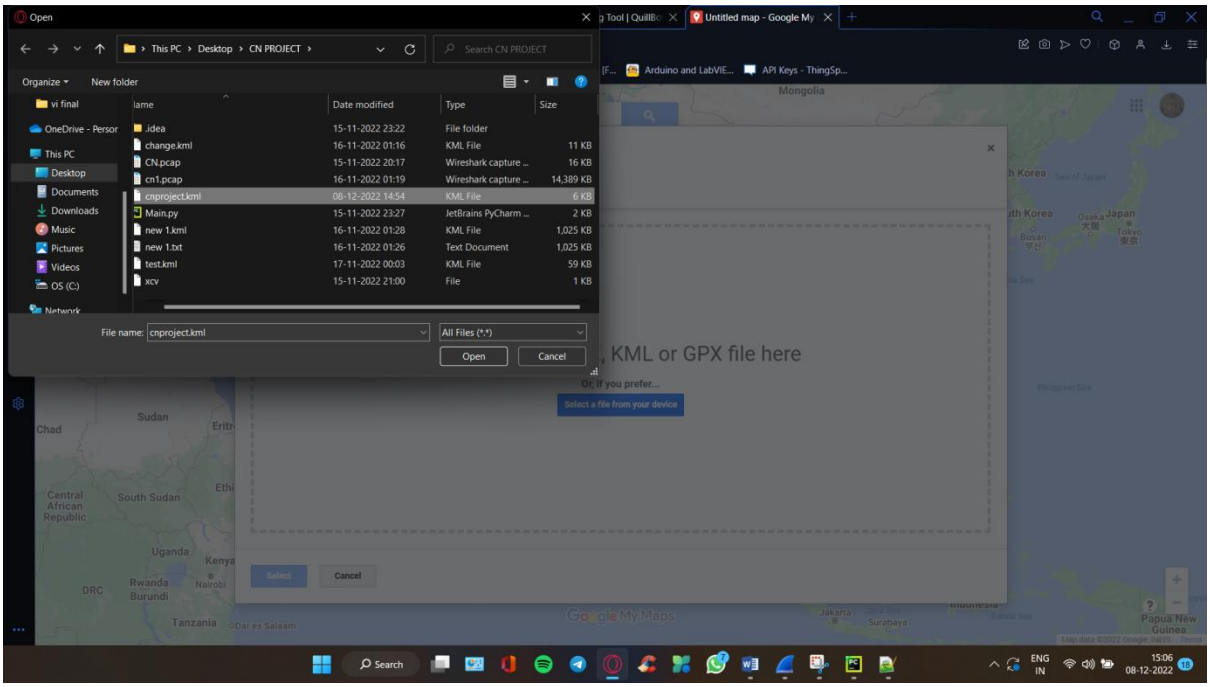

Fig 5.10. Picking out the file.

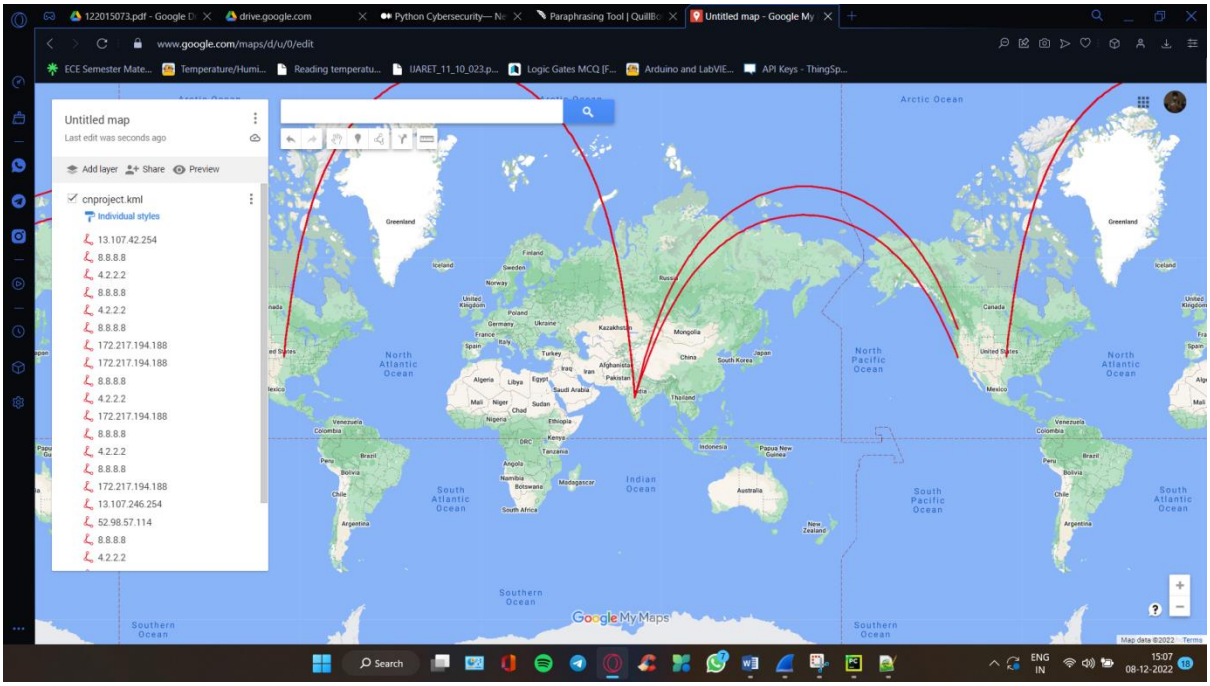

Fig 5.11. Google map showing the results.

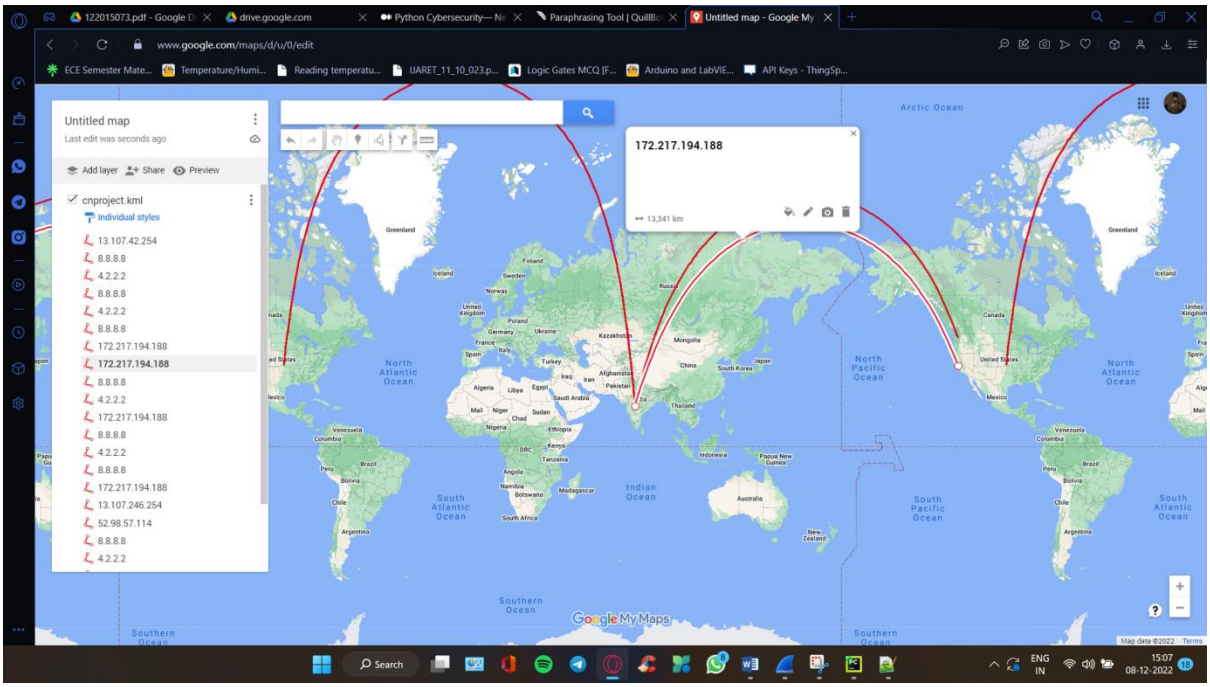

Fig 5.12. Choosing a particular address and seeing the highlighted path.

CONCLUSION

With the aid of Google Maps, Wireshark, and Python code, this application enables us to visualise the path, assisting the user in better comprehending the idea of global network traffic. In light of the foregoing, I would want to draw the conclusion that this project will aid in the user's understanding of network traffic and the route that data has taken.

FUTURE PLANS

We are unable to see the path of some packets recorded using Wireshark because of the limited file size that Google Maps would accept. So we can code for other websites that can accept the file size and visualise more paths in the future.

REFERENCES

https://medium.com/vinsloev-academy/python-cybersecurity-network-tracking-using-wireshark-and-google-maps-2adf3e497a93