Análisis inicial de Redes usando protocolo ICMP

Juan Javier Arosemena
jjarosemena@estud.usfq.edu.ec

Mateo Ayala
mayala@estud.usfq.edu.ec

February 7, 2020

1 Introducción

Se desea conocer los caminos más comunes que recorren los paquetes de datos entre la casa ecuatoriana promedio y ciertos sitios web de alto tráfico y uso común. Esta información, geolocalizada, permite una mejor comprensión de cómo la información cruza grandes distancias a través de la capa física del modelo de conexión actual. Como es claro, estos paquetes tienen que atravesar una serie de canales físicos que los llevan a su destino final, siendo los más notorios los cables submarinos a través de los cuales se envía la mayor parte de información del mundo. Por supuesto, este experimento sólo es tán válido como lo son la calidad de sus datos, por lo cual su funcionalidad se ve, de cierta manera, limitada.

Los datos que se usan en este experimento fueron recaudados en una etapa previa del proyecto. Se recolectaron tres datos (idealmente, en tiempos distintos del día) por participante del proyecto, con un total de 22 participantes. En total, se recolectaron 66 documentos por sitio web que describieran el camino que se empleó para llegar desde sus domicilios hasta las páginas correspondientes. A pesar de que los participantes tienen situaciones de alojamiento y conexiones variadas, es también importante aclarar que todos los estudiantes viven en Quito, se encuentran entre los 20 y los 25 años, y todos son estudiantes a tiempo completo de una universidad privada de Ecuador, por lo que los datos pueden estar sujetos a situaciones similares. Sin embargo, la variación general que existe entre los estudiantes es suficiente para, aunque sea *a grosso modo*, crear una imagen general del tráfico de internet de Quito a los sitios web investigados.

2 Metodología

Para la implementación de este experimento, se trabajó en lenguaje Python, haciendo uso de las bibliotecas ya disponibles para graficar las redes creadas, además de determinar el camino más común y averiguar más información de este. Se basó el código, además, en la implementación ya existente de Daniel Riofrío, PhD, para graficar la red general de todos los documentos de la investigación previa. Se continuó con los siguientes pasos:

1. Se determinó los sitios web a analizar para el proyecto.

- Las direcciones web a analizar son:
  - Google.com
  - Facebook.com
- A pesar de que esta elección fue al azar, estos sitios son ideales ya que por su alto tráfico y posición pública, los paquetes suelen llegar de manera confiable y rápida

2. Se separaron los archivos de interés del repertorio de archivos disponibles

- El repertorio cuenta con 66 archivos para 22 sitios web de uso cotidiano distintos
- Para los ejemplos en el informe, se utiliza la separación de Google.com

3. Se almacenó la información de cada una de las direcciones IP exploradas en estos caminos, además de las veces que cada conexión entre dos IP específicas fue utilizada, en una estructura de grafo

- Se filtran los datos que se consideran de una red LAN, y se los reemplaza con la dirección general "127.0.0.1"
- Se invierte los pesos del grafo, de tal manera que los caminos más transitados sean los más baratos

4. Se encuentra la ruta más usada entre el punto de inicio y el punto de llegada

- Debido a que existen varios puntos de llegada, se analiza la ruta más usada a cada uno
- Se grafica, además, las rutas más usadas de cada punto

5. Del punto de llegada más común, se analiza la información de cada nodo

- Se analiza la información de ANS y la información de Geolocalización
- En caso de que un IP sea privado, se informe de tal

Una vez obtenida toda esta información, se procede a analizar los resultados del experimento.

Se declaran los módulos a utilizar.

[1]:
```
import networkx as nx
import matplotlib.pyplot as plt
import os
import ip2geotools.databases.noncommercial as dbs
import pandas as pd
import descartes
import geopandas as gpd
from shapely.geometry import Point, Polygon
from ipwhois import IPWhois
from pprint import pprint

%matplotlib inline
```

Se declaran las variables y contenedores de datos iniciales.

[2]:
```
site_name = 'Google'
start_ip = '127.0.0.1'

#Path de lectura de archivos generados con traceroute.py
path = './All'

#lista todos los archivos dentro de la carpeta definida en path
files = os.listdir(path)
```

```
#diccionario que guarda los edges y sus frecuencias/pesos de los caminos mas␣
 ↪comunes.
edge_dict = {}

#Grafo dirigido que tendra todas las rutas que siguieron los paquetes.
G = nx.DiGraph()

#contado de ips filtradas
filtered_counter = 0
```

Se leen los archivos dentro del directorio `\All`. Cada archivo está estructurado de la forma: * IP destino * 1 IP * 2 IP . . . * n IP/Sitio Destino

y se arma el diccionario de edges con todas las secuencias de IPs en los archivos.

[3]:
```
#forloop usado para leer todos los archivos listados en el path.
for file_name in files:

    file_name_pieces = file_name.split('.')

    #Saltamos este archivo si no corresponde al IP estudiado
    if site_name not in file_name_pieces[1]:
        continue

    fd =  open(path+'/'+file_name, 'r')
    file = fd.read()
    file_split = file.split('\n')
    len_file_split = len(file_split)

    destination = ''
    prev = ''

    #ips que vamos a filtrar de los tres primeros registros de cada archivo que␣
 ↪vamos a analizar.
    #¿qué son estos ips?
    filter_ip = ['192.168.100.1', '192.168.10.1', '192.168.1.1',
                 '10.0.2.2', '192.168.0.1','172.21.140.1','192.168.4.1',
                 '192.168.0.254', '192.168.10.1', '192.168.1.10']

    #ip de destino que se encuentra en la primera linea del archivo generado por␣
 ↪traceroute.py
    destination = file_split[0].rstrip('\r')

    #forloop para iterar todas las entradas del archivo generado por traceroute.
 ↪py desde el primer registro del tipo 1-IP.
    for i in range(1, len_file_split-1):
        split_line = file_split[i].split('\t')
```

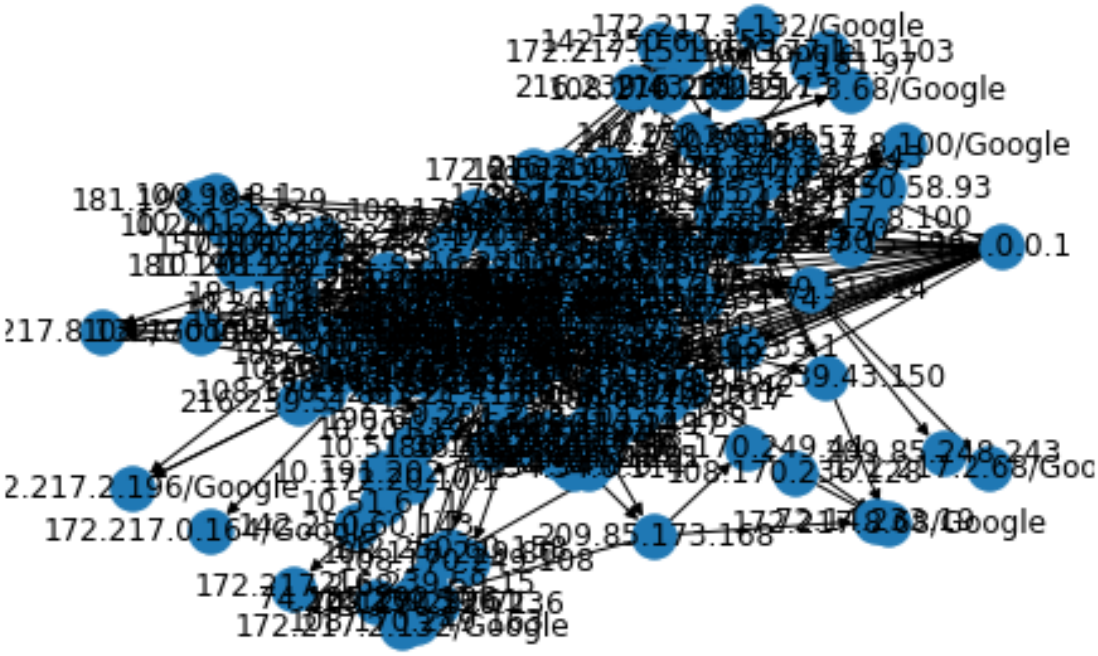
```
        curr = split_line[1]
        curr = curr.rstrip('\r')

        #filtrado de cualquier current ip que no sea una ip valida.
        if curr != '*':

            #filtrado de aquellas ips en filter_ip que esten en las primeras␣
↪tres lineas del archivo.
            if (i == 1 or i == 2 or i == 3) and (curr in filter_ip):
                filtered_counter = filtered_counter + 1
            else:
                #agrega los edges al diccionario que llevan al destino
                if curr == destination and prev != '':
                    edge = (prev, destination.rstrip('\r') + '/' +␣
↪file_name_pieces[1])
                    if edge in edge_dict:
                        edge_dict[edge] = edge_dict[edge] + 1
                    else:
                        edge_dict[edge] = 1
                else:
                    #agrega los edges de aquellos edges que no llevan a destino
                    #en el if / positivo -> agrega toda ip seguida de otra ip en␣
↪el camino del traceroute.
                    if prev != '':
                        edge = (prev, curr)
                        if edge in edge_dict:
                            edge_dict[edge] = edge_dict[edge] + 1
                        else:
                            edge_dict[edge] = 1
                    else:
                        #agrega el nodo especial que vamos a llamar 127.0.0.1␣
↪desde el cual vamos a buscar el camino mas corto.
                        if curr != destination and prev == '':
                            edge = (start_ip, curr)
                            if not edge in edge_dict:
                                edge_dict[edge] = 0
                    prev = curr

# Código por Daniel Riofrío (2020)
```

Se estructuran los edges de `edge_dict` en un grafo `G`. Luego, se grafica el grafo completo.

[4]:
```
#Se agregan todos los edges y nodos al grafo
for key, value in edge_dict.items():
    if value == 0:
        G.add_edge(key[0], key[1], weight=0)
    else:
```

```
        G.add_edge(key[0], key[1], weight=(1.0/float(value)))

#se imprime el total de nodos del grafo.
print("Nodes of graph: ")
print(len(G.nodes()))
#se imprime el total de edges del grafo.
print("Edges of graph: ")
print(len(G.edges()))
#se imprime el total de nodos filtrados.
print ("Filtered IPs: " + str(filtered_counter))
#se imprimen todos los nodos que estan conectados al nodo especial 127.0.0.1
print (G['127.0.0.1'])

#se dibuja el grafo.
nx.draw(G, with_labels=True)
#nx.draw_circular(G, with_labels=True)
plt.show()

# Código por Daniel Riofrío (2020)
```

```
Nodes of graph:
198
Edges of graph:
361
Filtered IPs: 65
{'172.21.18.126': {'weight': 0}, '181.198.150.1': {'weight': 0}, '172.20.10.1':
{'weight': 0}, '192.168.86.1': {'weight': 0}, '200.63.218.101': {'weight': 0},
'100.98.8.1': {'weight': 0}, '181.198.130.1': {'weight': 0}, '74.125.202.106':
{'weight': 0}, '172.253.66.57': {'weight': 0}, '172.253.50.89': {'weight': 0},
'186.4.244.1': {'weight': 0}, '100.97.32.1': {'weight': 0}, '157.100.244.1':
{'weight': 0}, '10.0.0.1': {'weight': 0}, '186.46.4.225': {'weight': 0},
'186.46.4.173': {'weight': 0}, '186.101.152.1': {'weight': 0}, '186.4.145.1':
{'weight': 0}, '100.96.233.1': {'weight': 0}, '200.63.218.185': {'weight': 0},
'100.98.68.1': {'weight': 0}, '186.101.137.1': {'weight': 0}, '181.198.191.129':
{'weight': 0}, '186.46.4.226': {'weight': 0}, '186.46.4.174': {'weight': 0}}
```



Se identifican todos los nodos finales del grafo (todos los nodos que terminan en `/Google`)

[5]:
```
endpoints = {}

for e, w in edge_dict.items():
    if site_name in e[1]:
        try:
            endpoints[e[1]] += w
        except:
            endpoints[e[1]] = w

most_used_endpoint = max(endpoints.items())

print("Existen "+ str(len(endpoints)) + " IPs destino para " + site_name + "\n")
print("El endpoint más usado es :\n", most_used_endpoint)
```

```
Existen 10 IPs destino para Google

El endpoint más usado es :
 ('172.217.8.68/Google', 4)
```

Se busca en todos los edges el nodo de inicio `127.0.0.1` y se identifican todos los nodos que lo suceden. Se guardan estos IPs como `startpoints`.

[6]:
```
startpoints = []
_startpoints = {}

for e, w in edge_dict.items():
    if start_ip in e[0]:
        startpoints.append(e[1])

for sp in startpoints:
    for e, w in edge_dict.items():
        if sp in e[0]:
            if e[0] not in _startpoints.keys():
                _startpoints[e[0]] = w
            else:
                _startpoints[e[0]] += w

print("Existen "+ str(len(_startpoints)) + " IPs que suceden al IP de inicio: "␣
 ↪+ start_ip + '\n')
total = 0
for sp, w in _startpoints.items():
    total += w

print(f"{str(total)} Total pesos de edges que proceden del edge con IP inicial␣
 ↪({start_ip}).\n")
print(f"Este número es mayor al número de archivos de {site_name} porque no␣
 ↪todos estos edges (_startpoints) proceden del edge con IP inicial {start_ip}")
```

```
Existen 25 IPs que suceden al IP de inicio: 127.0.0.1

117 Total pesos de edges que proceden del edge con IP inicial (127.0.0.1).

Este número es mayor al número de archivos de Google porque no todos estos edges
(_startpoints) proceden del edge con IP inicial 127.0.0.1
```

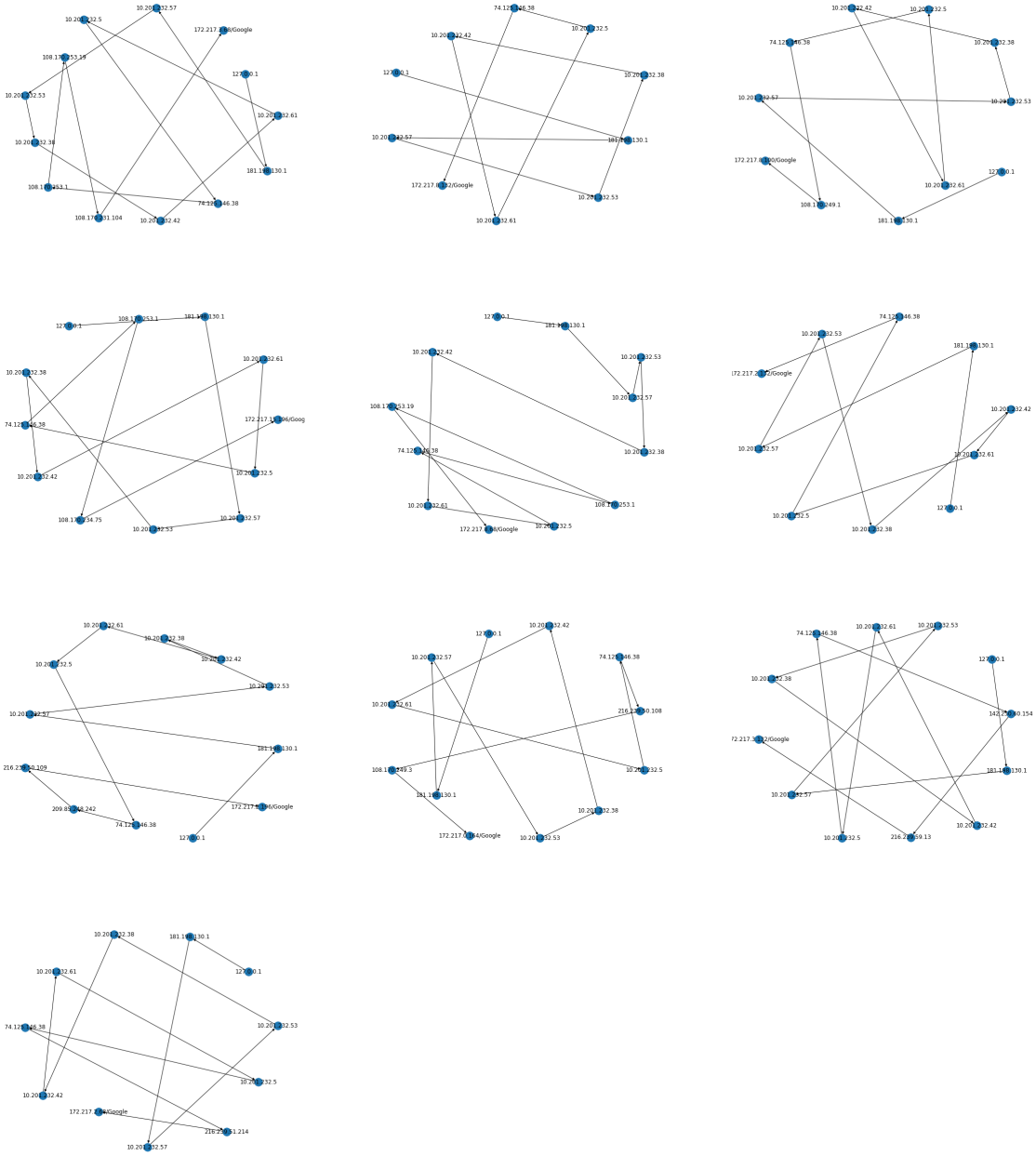
2.1 Shortest Path

Para encontrar el camino más usado en el grafo usamos el algoritmo de `shortest_path` que viene con `networkx`. Este algoritmo construye un camino de un nodo inicial a uno final siguiendo los edges con menores pesos posibles. Cabe recordar que en el código anterior, al crear el Grafo e insertar los edges, los pesos son inversamente proporcionales a las veces que un edge fue atravesado. Esto permite considerar la ruta que devuelve este algoritmo como el camino más usado para llegar al nodo destino.

[7]:
```
shortest_paths = []
plt.figure(figsize=(40, 120))
j = 0

# Para cada endpoint se busca su camino más corto en el grafo y se grafica.
```

```
for ep, times_used in endpoints.items():
    j += 1
    sp = nx.algorithms.shortest_path(G, source=start_ip , target=ep,␣
 ↪weight='weight')
    shortest_paths.append(sp)
    sp_G = nx.DiGraph()

    for i in range(len(sp)):
        try:
            sp_G.add_edge(sp[i], sp[i+1], weight=0)
        except:

            plt.subplot(10,3, j)

            #se dibuja el grafo.
            nx.draw(sp_G, with_labels=True)
            plt.xlabel(f"Endpoint:\t{ep}\nPath length:\t{len(sp)}\nTimes used:
 ↪\t{times_used}")

plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\matplotlib\backends\backend_agg.py:211: RuntimeWarning: Glyph 9 missing
from current font.
  font.set_text(s, 0.0, flags=flags)
```



2.2 WHOIS

Para obtener la información de cada nodo del camino más frecuente (determinado por el "shortest path" del peso inverso y el nodo más frecuentado), se utiliza la librería de Python IPWhois. Esta librería permite crear un objeto con el IP de un sitio web, y realizar búsquedas RDAP (recomendado) o WHOIS (descontinuado). Se vuelve a calcular el shortest path del nodo más usado y se lo guarda en una lista. A continuación, se procede a iterar sobre esta lista, que contiene direcciones IP. Para cada nodo IP se crea un objeto IPWhois y se realiza una búsqueda RDAP, la cual luego es guardada en un diccionario. Se accede este diccionario para imprimir la información del ASN

(anonnymous naming system), el código del país al que pertence, la descripción si la tiene, y la base de datos con la que se encuentra registrada. En caso de que se reporte un error de privacidad, la dirección simplemente indica que el nodo que se está intentando revisar es privado.

[8]:
```
print("Informacion WHOIS de cada nodo del camino\n")

most_used_path = nx.algorithms.shortest_path(G, source=start_ip ,␣
 ↪target=max(endpoints), weight='weight')
country_codes = []

for i in range(len(most_used_path)-1):
    ip = most_used_path[i]
    print(ip+'\n')
    try:
        obj = IPWhois(ip)
        result = obj.lookup_rdap(asn_methods=['dns', 'whois', 'http'])
        #Save country codes for later plotting
        country_codes.append(result['asn_country_code'])
        print(f"\tASN: {result['asn']}",f"\tCIDR: {result['asn_cidr']}",
              f"\tCodigo de pais: {result['asn_country_code']}",
              f"\tDescripcion: {result['asn_description']}",
              f"\tRegistrado en: {result['asn_registry']}\n",sep='\n')
    except:
        print("\tINFORMACION PRIVADA\n")

ep = most_used_path[-1]
ep_ip = ep.split('/')[0]
print(ep_ip+'\n')
try:
    obj = IPWhois(ep_ip)
    result = obj.lookup_rdap(asn_methods=['dns', 'whois', 'http'])
    print(f"\tASN: {result['asn']}",f"\tCIDR: {result['asn_cidr']}",
              f"\tCodigo de pais: {result['asn_country_code']}",
              f"\tDescripcion: {result['asn_description']}",
              f"\tRegistrado en: {result['asn_registry']}\n",sep='\n')
except:
    print("\tINFORMACION PRIVADA\n")
```

```
Informacion WHOIS de cada nodo del camino

127.0.0.1

        INFORMACION PRIVADA

181.198.130.1

        ASN: 27947
        CIDR: 181.198.128.0/19
```

```
        Codigo de pais: EC
        Descripcion: Telconet S.A, EC
        Registrado en: lacnic

10.201.232.57

        INFORMACION PRIVADA

10.201.232.53

        INFORMACION PRIVADA

10.201.232.38

        INFORMACION PRIVADA

10.201.232.42

        INFORMACION PRIVADA

10.201.232.61

        INFORMACION PRIVADA

10.201.232.5

        INFORMACION PRIVADA

74.125.146.38

        ASN: 15169
        CIDR: 74.125.0.0/16
        Codigo de pais: US
        Descripcion: GOOGLE, US
        Registrado en: arin

108.170.253.1

        ASN: 15169
        CIDR: 108.170.192.0/18
        Codigo de pais: US
        Descripcion: GOOGLE, US
        Registrado en: arin

108.170.253.19

        ASN: 15169
        CIDR: 108.170.192.0/18
```

```
        Codigo de pais: US
        Descripcion: GOOGLE, US
        Registrado en: arin

172.217.8.68

        ASN: 15169
        CIDR: 172.217.8.0/24
        Codigo de pais: US
        Descripcion: GOOGLE, US
        Registrado en: arin
```

[9]:
```
# Chequea cuantos edges conectan al IP de Telconet S.A.
ips = []

for edge, w in edge_dict.items():
    if edge[0] == '181.198.130.1':
        ips.append((edge[1], w))

print(f"Al menos {ips[0][1]} archivos incluyen una salida desde el IP de␣
 ↪Telconet S.A.: 181.198.130.1")
```

```
Al menos 6 archivos incluyen una salida desde el IP de Telconet S.A.:
181.198.130.1
```

2.3 Geolocalización y mapeo

Se utiliza la librería `ip2geotools` para consultar una base de datos que contiene información de geolocalización de IPs públicas. La base de datos usada es `HostIP` porque permite consultar de forma gratuita y sin necesidad de API keys.

[10]:
```
# Se obtiene el camino más usado en el grafo
most_used_path = nx.algorithms.shortest_path(G, source=start_ip ,␣
 ↪target=max(endpoints), weight='weight')

coordinates = []
just_countries = []


for ip in most_used_path:
    ip = ip.split('/')[0]
    # Se consulta la base de datos
    r = dbs.HostIP.get(ip)

    # Si no devuelve información, es una IP privada
    if not r.country and not r.region and not r.city and not r.longitude and not␣
 ↪r.latitude:
```

```
        print(ip, "Private", sep='\t')
    else:
        if r.longitude and r.latitude:
            coordinates.append((r.longitude, r.latitude))
        elif r.country:
            just_countries.append(r.country)

        print(f"\n{ip}\nCtry:\t{r.country if r.country else ''}\nCity:\t{r.city␣
↪if r.city else ''}\nRegn:\t{r.region if r.region else ''}\nLong:\t{r.longitude␣
↪if r.longitude else ''}\nLat:\t{r.latitude if r.latitude else ''}\n")

# Se guardan todos los países que se conocen que atraviesa el camino.
just_countries.extend(set(country_codes))
```

```
127.0.0.1       Private
181.198.130.1   Private
10.201.232.57   Private
10.201.232.53   Private
10.201.232.38   Private
10.201.232.42   Private
10.201.232.61   Private
10.201.232.5    Private
74.125.146.38   Private
108.170.253.1   Private
108.170.253.19  Private
172.217.8.68    Private
```

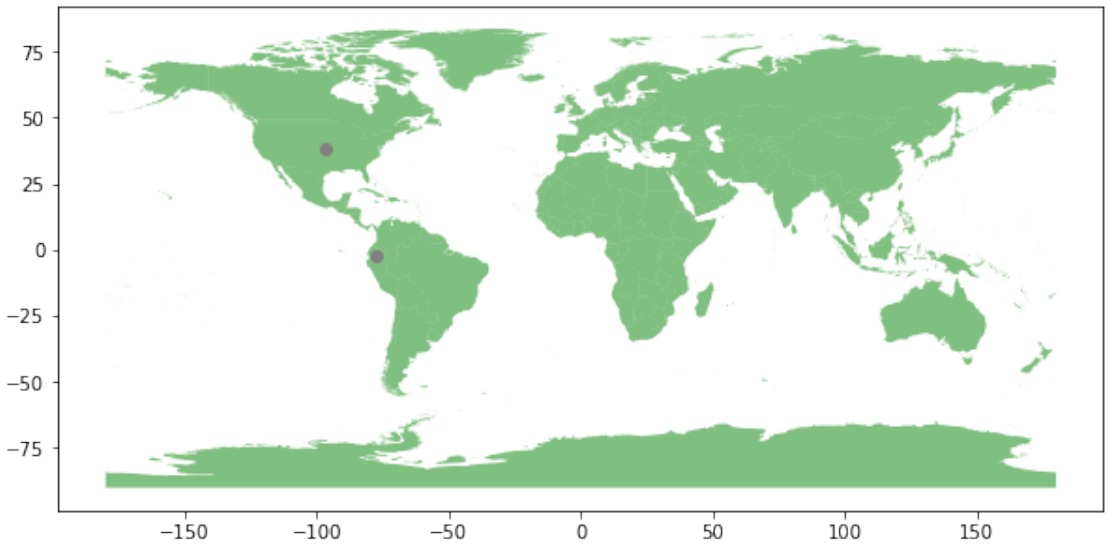
Para graficar las coordenadas de los IPs se utiliza la librería geopandas junto con un archivo `.shp` que contiene el mapa a graficar. Se grafican dos colores de puntos: * Rojo: IP con ubicación exacta conocida. * Gris: IP cuya única información es su país.

Para ubicar en el mapa los puntos grises se utiliza un archivo `.csv` que contienen información de cada país y sus coordenadas centrales respectivas. Estas coordenadas son usadas en el mapa.

[11]:
```
# Se genera el mapa del mundo
map = gpd.read_file('TM_WORLD_BORDERS-0.3.shp')

# Datos exactos
geometry = [Point(xy) for xy in coordinates]
geo_df_exact = gpd.GeoDataFrame(geometry= geometry)

# Se lee el .csv y se extraen los datos de coordenadas de países
country_df = pd.read_csv('countries_codes_and_coordinates.csv')
country_df = country_df.set_index('alpha-2')
_df = country_df.loc[just_countries]
just_c_coords = [xy for xy in zip([x[1] for x in _df['longitude'].items()],␣
↪[x[1] for x in _df['latitude'].items()])]
```

```
fig, ax = plt.subplots(figsize=(10, 10))

# Plot de puntos rojos
map.plot(ax=ax, alpha=0.5, color='green');
geo_df_exact.plot(ax=ax, color='red');

# Plot de puntos grises
geometry = [Point(xy) for xy in just_c_coords]
geo_df_just_c = gpd.GeoDataFrame(geometry= geometry)
geo_df_just_c.plot(ax=ax, color='gray');
```

```
C:\ProgramData\Anaconda3\lib\site-packages\geopandas\plotting.py:510:
UserWarning: The GeoDataFrame you are attempting to plot is empty. Nothing has
been displayed.
  UserWarning,
```



3 Conclusiones

El análisis de rutas de IP fue complicado desde el inicio. El mayor problema es obtener datos limpios, y desde la etapa de obtención de las rutas para cada sitio web ya faltan datos porque la mayoría archivos no muestran todos los IPs por los que pasaron los paquetes de datos. El análisis del grafo se pudo realizar correctamente y se pudieron encontrar las rutas más usadas omitiendo los IPs que no se hallaban en los archivos. Por último, el análisis de IPs mediante Whois resultó más informativo que la búsqueda de coordenadas globales con `ip2geotools` porque es muy escasa la información gratis que se puede obtener mediante este segundo método. Sin embargo, se pudo unificar la data obtenida de ambos métodos y se pudo graficar en un mapa tanto los IPs con coordenadas conocidas como los IPs cuya única información es su país.

- ¿Por dónde pasan las comunicaciones que salen del país?
  - Mediante Whois se pudo ver que en la ruta más usada existe un nodo de Telconet S.A. tanto para Google.com como Facebook.com
- ¿Cuántas personas usan un mismo servicio de internet en clase?
  - Analizando solamente la ruta más utilizada, existen 6 edges que conectan desde Telconet S.A. O sea que al menos 2 y máximo 6 estudiantes utilizan este servicio. Este caso es igual tanto para Google.com como Facebook.com
- ¿Es seguro que estemos haciendo este tipo de mediciones desde casa?
  - Seguridad: El nivel de seguridad de realizar las mediciones de traceroute es la misma que al ingresar a los sitios web mediante un navegador porque los paquetes enviados no contienen información más compremetedora que la que un navegador envía.
  - Fiabilidad: La fiabilidad de los datos es cuestionable porque el filtro de IPs LAN no es perfecto, y en este proceso se pueden haber filtrado IPs que no corresponden a una red LAN, o al reves, se pueden haber no filtrado IPs LAN.
- ¿Qué pasa si corremos este tipo de monitoreo todos los días, todos los minutos?
  - Las rutas varían a cada hora y minuto del día. La cantidad de tráfico en la web define en parte las decisiones de enrutamiento de los paquetes.

4 Anexo: Facebook.com

Este código es idéntico en funcionalidad al corrido para Google.com

[12]:
```
site_name = 'Facebook'
edge_dict = {}
G = nx.DiGraph()

#contado de ips filtradas
filtered_counter = 0

#forloop usado para leer todos los archivos listados en el path.
for file_name in files:

    file_name_pieces = file_name.split('.')

    #Saltamos este archivo si no corresponde al IP estudiado
    if site_name not in file_name_pieces[1]:
        continue

    fd =  open(path+'/'+file_name, 'r')
    file = fd.read()
    file_split = file.split('\n')
    len_file_split = len(file_split)

    destination = ''
    prev = ''
```

```
   #ips que vamos a filtrar de los tres primeros registros de cada archivo que␣
↪vamos a analizar.
   #¿qué son estos ips?
   filter_ip = ['192.168.100.1', '192.168.10.1', '192.168.1.1',
                '10.0.2.2', '192.168.0.1','172.21.140.1','192.168.4.1',
                '192.168.0.254', '192.168.10.1', '192.168.1.10']

   #ip de destino que se encuentra en la primera linea del archivo generado por␣
↪traceroute.py
   destination = file_split[0].rstrip('\r')

   #forloop para iterar todas las entradas del archivo generado por traceroute.
↪py desde el primer registro del tipo 1-IP.
   for i in range(1, len_file_split-1):
       split_line = file_split[i].split('\t')
       curr = split_line[1]
       curr = curr.rstrip('\r')

       #filtrado de cualquier current ip que no sea una ip valida.
       if curr != '*':

           #filtrado de aquellas ips en filter_ip que esten en las primeras␣
↪tres lineas del archivo.
           if (i == 1 or i == 2 or i == 3) and (curr in filter_ip):
               filtered_counter = filtered_counter + 1
           else:
               #agrega los edges al diccionario que llevan al destino
               if curr == destination and prev != '':
                   edge = (prev, destination.rstrip('\r') + '/' +␣
↪file_name_pieces[1])
                   if edge in edge_dict:
                       edge_dict[edge] = edge_dict[edge] + 1
                   else:
                       edge_dict[edge] = 1
               else:
                   #agrega los edges de aquellos edges que no llevan a destino
                   #en el if / positivo -> agrega toda ip seguida de otra ip en␣
↪el camino del traceroute.
                   if prev != '':
                       edge = (prev, curr)
                       if edge in edge_dict:
                           edge_dict[edge] = edge_dict[edge] + 1
                       else:
                           edge_dict[edge] = 1
                   else:
```

```
                        #agrega el nodo especial que vamos a llamar 127.0.0.1␣
 ↪desde el cual vamos a buscar el camino mas corto.
                        if curr != destination and prev == '':
                            edge = (start_ip, curr)
                            if not edge in edge_dict:
                                edge_dict[edge] = 0
                    prev = curr

# Código por Daniel Riofrío (2020)

#Se agregan todos los edges y nodos al grafo
for key, value in edge_dict.items():
    if value == 0:
        G.add_edge(key[0], key[1], weight=0)
    else:
        G.add_edge(key[0], key[1], weight=(1.0/float(value)))

#se imprime el total de nodos del grafo.
print("Nodes of graph: ")
print(len(G.nodes()))
#se imprime el total de edges del grafo.
print("Edges of graph: ")
print(len(G.edges()))
#se imprime el total de nodos filtrados.
print ("Filtered IPs: " + str(filtered_counter))
#se imprimen todos los nodos que estan conectados al nodo especial 127.0.0.1
print (G['127.0.0.1'])

#se dibuja el grafo.
nx.draw(G, with_labels=True)
#nx.draw_circular(G, with_labels=True)
plt.show()

# Código por Daniel Riofrío (2020)

endpoints = {}

for e, w in edge_dict.items():
    if site_name in e[1]:
        try:
            endpoints[e[1]] += w
        except:
            endpoints[e[1]] = w

most_used_endpoint = max(endpoints.items())

print("Existen "+ str(len(endpoints)) + " IPs destino para " + site_name + "\n")
```

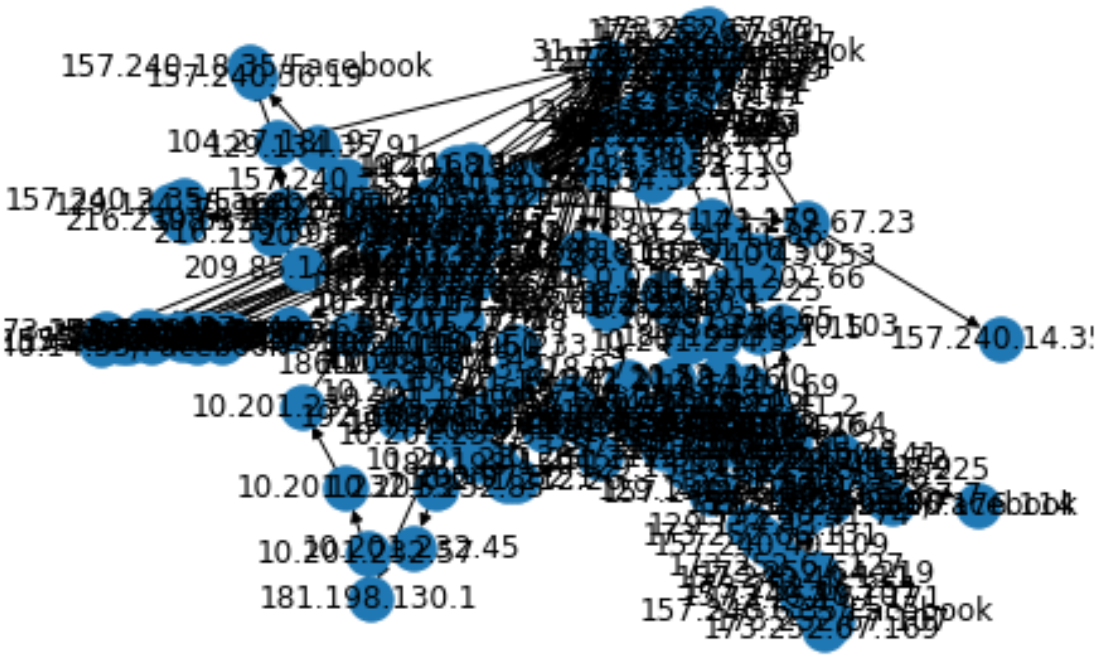
```
print("El endpoint más usado es :\n", most_used_endpoint)

startpoints = []
_startpoints = {}

for e, w in edge_dict.items():
    if start_ip in e[0]:
        startpoints.append(e[1])

for sp in startpoints:
    for e, w in edge_dict.items():
        if sp in e[0]:
            if e[0] not in _startpoints.keys():
                _startpoints[e[0]] = w
            else:
                _startpoints[e[0]] += w

print("Existen "+ str(len(_startpoints)) + " IPs que suceden al IP de inicio: "␣
 ↪+ start_ip + '\n')
total = 0
for sp, w in _startpoints.items():
    total += w

print(f"{str(total)} Total pesos de edges que proceden del edge con IP inicial␣
 ↪({start_ip}).\n")
print(f"Este número es mayor al número de archivos de {site_name} porque no␣
 ↪todos estos edges (_startpoints) proceden del edge con IP inicial {start_ip}")

shortest_paths = []
plt.figure(figsize=(40, 120))
j = 0

# Para cada endpoint se busca su camino más corto en el grafo y se grafica.
for ep, times_used in endpoints.items():
    j += 1
    sp = nx.algorithms.shortest_path(G, source=start_ip , target=ep,␣
 ↪weight='weight')
    shortest_paths.append(sp)
    sp_G = nx.DiGraph()

    for i in range(len(sp)):
        try:
            sp_G.add_edge(sp[i], sp[i+1], weight=0)
        except:

            plt.subplot(10,3, j)
```

```
            #se dibuja el grafo.
            nx.draw(sp_G, with_labels=True)
            plt.xlabel(f"Endpoint:\t{ep}\nPath length:\t{len(sp)}\nTimes used:
 ↪\t{times_used}")

plt.show()

print("Informacion WHOIS de cada nodo del camino\n")

most_used_path = nx.algorithms.shortest_path(G, source=start_ip ,␣
 ↪target=max(endpoints), weight='weight')
country_codes = []

for i in range(len(most_used_path)-1):
    ip = most_used_path[i]
    print(ip+'\n')
    try:
        obj = IPWhois(ip)
        result = obj.lookup_rdap(asn_methods=['dns', 'whois', 'http'])
        #Save country codes for later plotting
        country_codes.append(result['asn_country_code'])
        print(f"\tASN: {result['asn']}",f"\tCIDR: {result['asn_cidr']}",
              f"\tCodigo de pais: {result['asn_country_code']}",
              f"\tDescripcion: {result['asn_description']}",
              f"\tRegistrado en: {result['asn_registry']}\n",sep='\n')
    except:
        print("\tINFORMACION PRIVADA\n")

ep = most_used_path[-1]
ep_ip = ep.split('/')[0]
print(ep_ip+'\n')
try:
    obj = IPWhois(ep_ip)
    result = obj.lookup_rdap(asn_methods=['dns', 'whois', 'http'])
    print(f"\tASN: {result['asn']}",f"\tCIDR: {result['asn_cidr']}",
              f"\tCodigo de pais: {result['asn_country_code']}",
              f"\tDescripcion: {result['asn_description']}",
              f"\tRegistrado en: {result['asn_registry']}\n",sep='\n')
except:
    print("\tINFORMACION PRIVADA\n")

most_used_path = nx.algorithms.shortest_path(G, source=start_ip ,␣
 ↪target=max(endpoints), weight='weight')

coordinates = []
just_countries = []
```

```
for ip in most_used_path:
    ip = ip.split('/')[0]
    r = dbs.HostIP.get(ip)
    if not r.country and not r.region and not r.city and not r.longitude and not␣
 ↪r.latitude:
        print(ip, "Private", sep='\t')
    else:
        if r.longitude and r.latitude: coordinates.append((r.longitude, r.
 ↪latitude))
        elif r.country: just_countries.append(r.country)

        print(f"\n{ip}\nCtry:\t{r.country if r.country else ''}\nCity:\t{r.city␣
 ↪if r.city else ''}\nRegn:\t{r.region if r.region else ''}\nLong:\t{r.longitude␣
 ↪if r.longitude else ''}\nLat:\t{r.latitude if r.latitude else ''}\n")

just_countries.extend(set(country_codes))

# Chequea cuantos edges conectan al IP de Telconet S.A.
ips = []

for edge, w in edge_dict.items():
    if edge[0] == '181.198.130.1':
        ips.append((edge[1], w))

print(f"Al menos {ips[0][1]} archivos incluyen una salida desde el IP de␣
 ↪Telconet S.A.: 181.198.130.1")

map = gpd.read_file('TM_WORLD_BORDERS-0.3.shp')

geometry = [Point(xy) for xy in coordinates]

geo_df_exact = gpd.GeoDataFrame(geometry= geometry)


country_df = pd.read_csv('countries_codes_and_coordinates.csv')

country_df = country_df.set_index('alpha-2')
_df = country_df.loc[just_countries]
just_c_coords = [xy for xy in zip([x[1] for x in _df['longitude'].items()],␣
 ↪[x[1] for x in _df['latitude'].items()])]


fig, ax = plt.subplots(figsize=(10, 10))

map.plot(ax=ax, alpha=0.5, color='green');
geo_df_exact.plot(ax=ax, color='red');
```

```
geometry = [Point(xy) for xy in just_c_coords]
geo_df_just_c = gpd.GeoDataFrame(geometry= geometry)
geo_df_just_c.plot(ax=ax, color='gray');
```

```
Nodes of graph:
213
Edges of graph:
314
Filtered IPs: 64
{'172.21.18.126': {'weight': 0}, '181.198.150.1': {'weight': 0}, '172.20.10.1':
{'weight': 0}, '192.168.86.1': {'weight': 0}, '200.63.218.97': {'weight': 0},
'100.98.8.1': {'weight': 0}, '181.198.130.1': {'weight': 0}, '72.14.234.8':
{'weight': 0}, '104.27.181.97': {'weight': 0}, '209.85.143.102': {'weight': 0},
'186.4.244.1': {'weight': 0}, '100.97.32.1': {'weight': 0}, '157.100.244.1':
{'weight': 0}, '10.0.0.1': {'weight': 0}, '186.46.4.225': {'weight': 0},
'186.101.152.1': {'weight': 0}, '186.4.145.1': {'weight': 0}, '100.96.233.1':
{'weight': 0}, '200.63.218.177': {'weight': 0}, '100.98.68.1': {'weight': 0},
'186.101.137.1': {'weight': 0}, '181.198.191.129': {'weight': 0},
'186.46.4.174': {'weight': 0}, '186.46.4.226': {'weight': 0}}
```



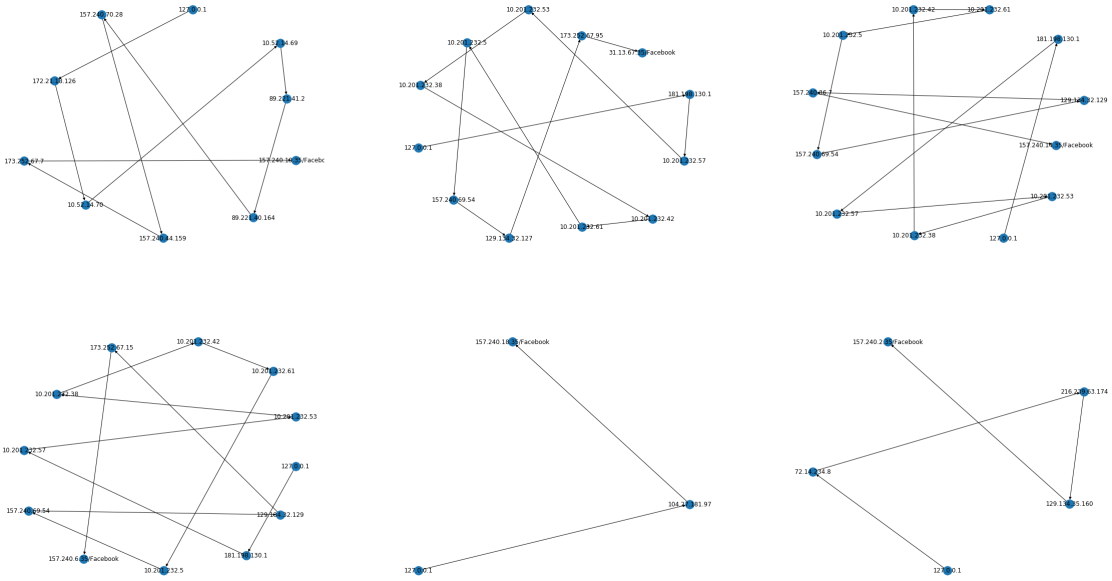

```
Existen 6 IPs destino para Facebook
```

El endpoint más usado es :
 ('31.13.67.35/Facebook', 33)
Existen 24 IPs que suceden al IP de inicio: 127.0.0.1

72 Total pesos de edges que proceden del edge con IP inicial (127.0.0.1).

Este número es mayor al número de archivos de Facebook porque no todos estos edges (_startpoints) proceden del edge con IP inicial 127.0.0.1

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:211: RuntimeWarning: Glyph 9 missing from current font.
  font.set_text(s, 0.0, flags=flags)



Informacion WHOIS de cada nodo del camino

127.0.0.1

        INFORMACION PRIVADA

181.198.130.1

        ASN: 27947
        CIDR: 181.198.128.0/19
        Codigo de pais: EC
        Descripcion: Telconet S.A, EC
        Registrado en: lacnic

10.201.232.57

        INFORMACION PRIVADA

10.201.232.53

        INFORMACION PRIVADA

10.201.232.38

        INFORMACION PRIVADA

10.201.232.42

        INFORMACION PRIVADA

10.201.232.61

        INFORMACION PRIVADA

10.201.232.5

        INFORMACION PRIVADA

157.240.69.54

        ASN: 32934
        CIDR: 157.240.0.0/17
        Codigo de pais: US
        Descripcion: FACEBOOK, US
        Registrado en: arin

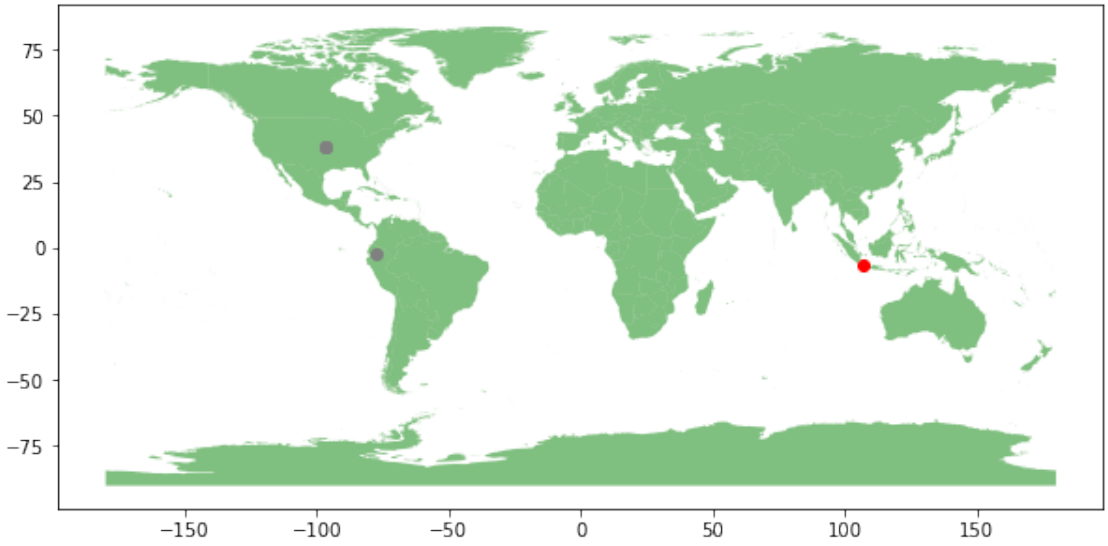
129.134.32.127

        ASN: 32934
        CIDR: 129.134.0.0/17
        Codigo de pais: US
        Descripcion: FACEBOOK, US
        Registrado en: arin

173.252.67.95

        ASN: 32934
        CIDR: 173.252.64.0/19
        Codigo de pais: US
        Descripcion: FACEBOOK, US
        Registrado en: arin

```
31.13.67.35

        ASN: 32934
        CIDR: 31.13.67.0/24
        Codigo de pais: IE
        Descripcion: FACEBOOK, US
        Registrado en: ripencc

127.0.0.1       Private
181.198.130.1   Private
10.201.232.57   Private
10.201.232.53   Private
10.201.232.38   Private
10.201.232.42   Private
10.201.232.61   Private
10.201.232.5    Private

157.240.69.54
Ctry:   US
City:
Regn:
Long:
Lat:


129.134.32.127
Ctry:   US
City:
Regn:
Long:
Lat:

173.252.67.95   Private

31.13.67.35
Ctry:   ID
City:   Jakarta
Regn:
Long:   106.8
Lat:    -6.26667

Al menos 6 archivos incluyen una salida desde el IP de Telconet S.A.:
181.198.130.1
```


75
50
25
0
−25
−50
−75
−150
−100
−50
0
50
100
150