Instituto Tecnológico Autónomo de México

Departamento Académico de Sistemas Digitales

*Nuevas Tecnologías Inalámbricas*

***Práctica 4***

***Redes Ad-hoc Autoconfigurables***

Integrantes:

Alfonso Venancio- 149211

Efraín Aguilar- 149643

Leandro Pantoja- 150883

Ulises Alejandre - 159235

12 de febrero de 2019

**Introducción**

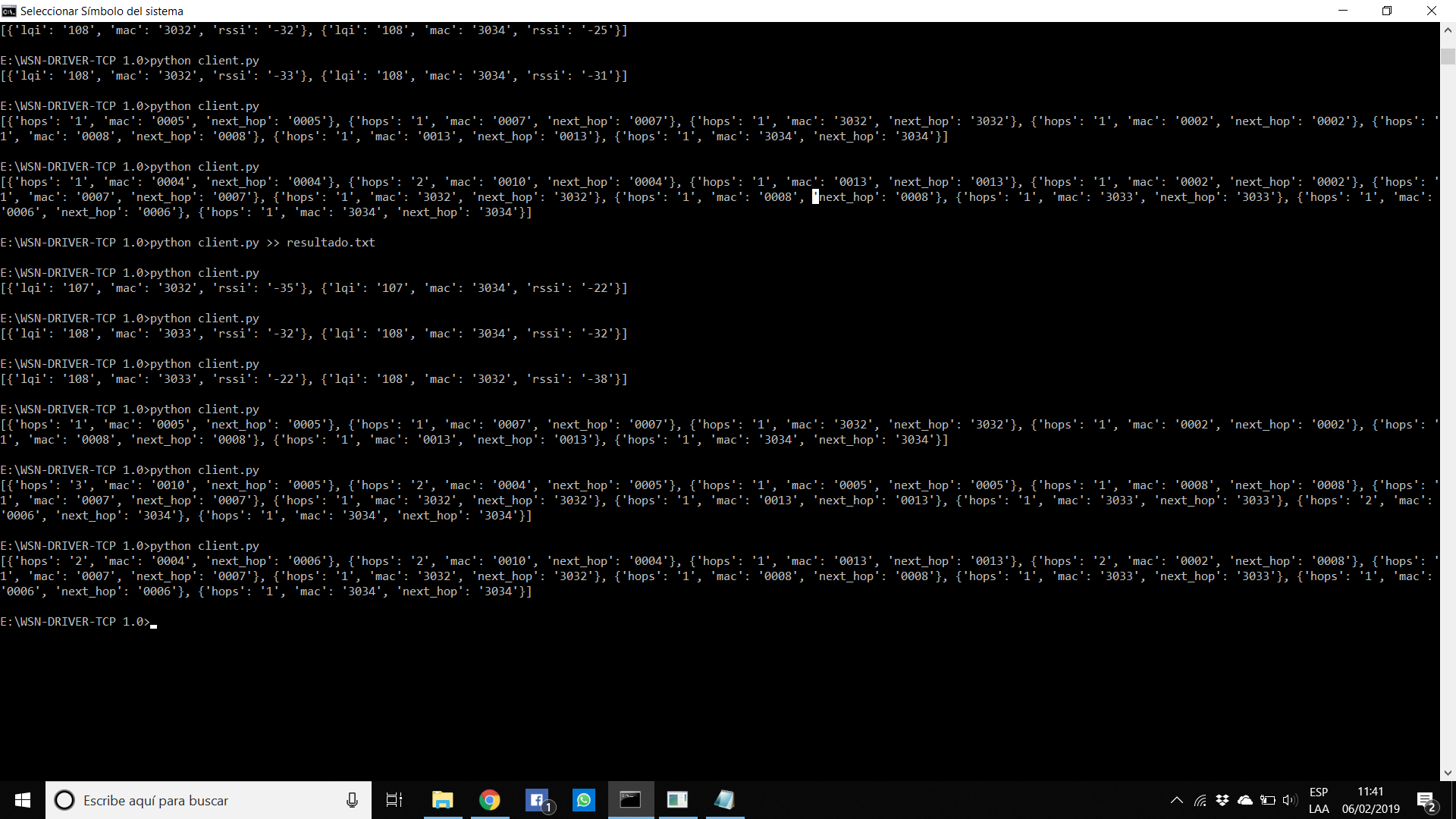
Las redes multisalto ad-hoc son una estructura de red que está compuesta por nodos que se comunican entre sí, pero no hay una inalámbrica fija. La comunicación entre dos nodos se realiza usando nodos intermedios, “saltando” de uno a otro hasta llegar al nodo destino; de allí el nombre de esta estructura de red. Cada nodo es un emisor y transmisor de radio.

**Desarrollo**

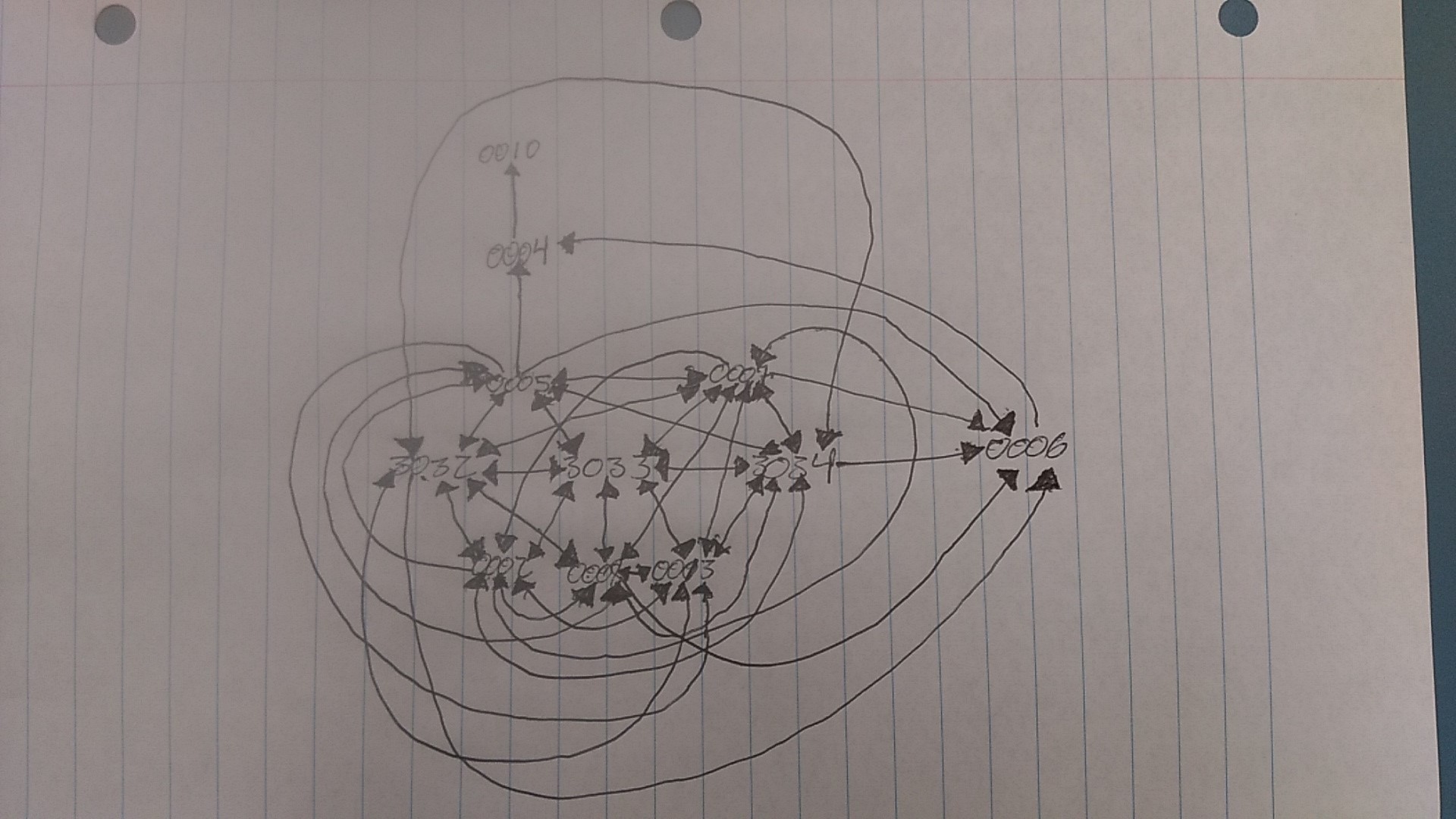
Esta práctica está formada por dos partes: la primera consistía en hacer un sondeo a partir de un radio que se nos fue asignado para poder dibujar un diagrama de la red generada por esos nodos; la segunda, programar un script que pudiera determinar la dirección MAC de los nodos y de la computadora a la que estaban conectados, además de poder mostrar un grafo de las conexiones entre todos los nodos de la red y obtener todas las rutas tomando un nodo inicial.

Para poder obtener de los radios se usó el driver “CP210x USB to UART Bridge VCP Driver” de Silicon Labs y la aplicación WSN-DRIVER-TCP. La aplicación se usa para que en la ventana de comandos de Windows podamos ver la información que proporcione el radio al mandar *queries* desde un script de python proporcionado por el profesor. Los comandos usados son los siguientes:

* {"id": "neighbours", "mac": "XXXX"} : muestra la información de los nodos adyacentes al de la dirección MAC proporcionada
* {"id": "all\_neighbours", "mac": "XXXX"}: muestra la información de todos los nodos a los que se puede llegar desde el de la dirección MAC proporcionada
* {"id": "attached\_devices", "mac": "XXXX"}: muestra la dirección MAC de la computadora a la que el radio especificado está conectado
* {"id": "routes", "mac": "XXXX"}: muestra todos los nodos destino accesibles desde el nodo origen especificado

*Figura 1: resultados de la terminal tras ejecutar el archivo de python con el comando neighbours.*

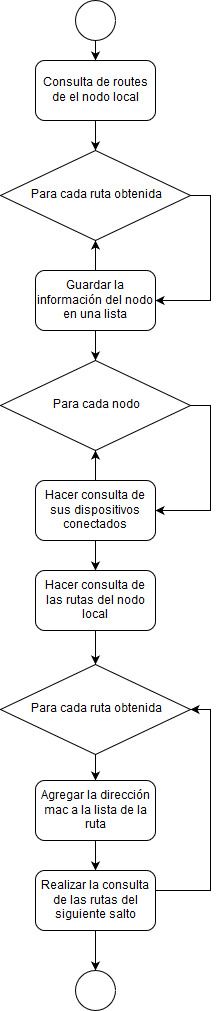
Para poder dibujar el grafo se tuvo que ejecutar el comando de *routes* y dibujar a mano las flechas que representan la dirección de la arista para cada nodo de la red. Después de repetir este proceso para los 11 nodos que teníamos disponibles en la red se obtuvo lo siguiente:



*Figura 2: Grafo direccional de la red*

Para la segunda parte de la práctica, primero se hizo un programa para encontrar todas las MAC disponibles de todos los radios, haciendo una consulta *routes* a un nodo y guardando la información en una lista. Para obtener la lista de las PCs en la red, se realizó una consulta *attached\_devices* para cada nodo y se agregaron solamente los nodos que sí tenían algún dispositivo conectado.

Para conocer las rutas completas de un nodo, primero se realiza un solicitud de *routes* al nodo que se quiere explorar. Después para cada una de esas rutas, se crea un espacio en un diccionario, tomando como llave la dirección mac destino. Después con un *do-while*, se solicitan las rutas de todos los siguientes saltos de cada una de las rutas del primer nodo y se comparan estas rutas para encontrar la que tiene la misma dirección de destino que la ruta que inicia desde el nodo local. Esto se repite hasta que el número de saltos hasta el destino sea 1, esto quiere decir, que el enlace es directo y que se encontró la ruta completa entre el nodo local y alguno de los nodos en la red.

****

*Figura 3: Diagrama de flujo para encontrar la lista de MACs de radios y computadoras y las rutas de un nodo en particular*

**Resultados**

Todos los códigos se integraron en uno solo, y los resultados se guardaron en archivos de texto. Un ejemplo de lo que el programa arroja si consideramos al nodo central 0004.

==================Todas las mac de los radios==================

0010

0005

0006

0002

0008

0004

==================Todas las mac de las PC==================

8500

F111

==================Todaslas rutas del radio local==================

[{'hops': '1', 'mac': '0010', 'next\_hop': '0010'}, {'hops': '1', 'mac': '0005', 'next\_hop': '0005'}, {'hops': '1', 'mac': '0006', 'next\_hop': '0006'}, {'hops': '2', 'mac': '0002', 'next\_hop': '0006'}, {'hops': '3', 'mac': '0008', 'next\_hop': '0005'}]

['0010']

['0005']

['0006']

['0006']

['0005', '0002']

==================Dibujo del grafo==================

El número total de nodos en la red es de: 6

El número total de enlaces entre los nodos es de: 14

Como actividad extra se realizó un programa para poder dibujar el grafo de conexiones de los radios apoyándonos con las funciones de **DiGraph(), add\_node()** y **add\_edge().**

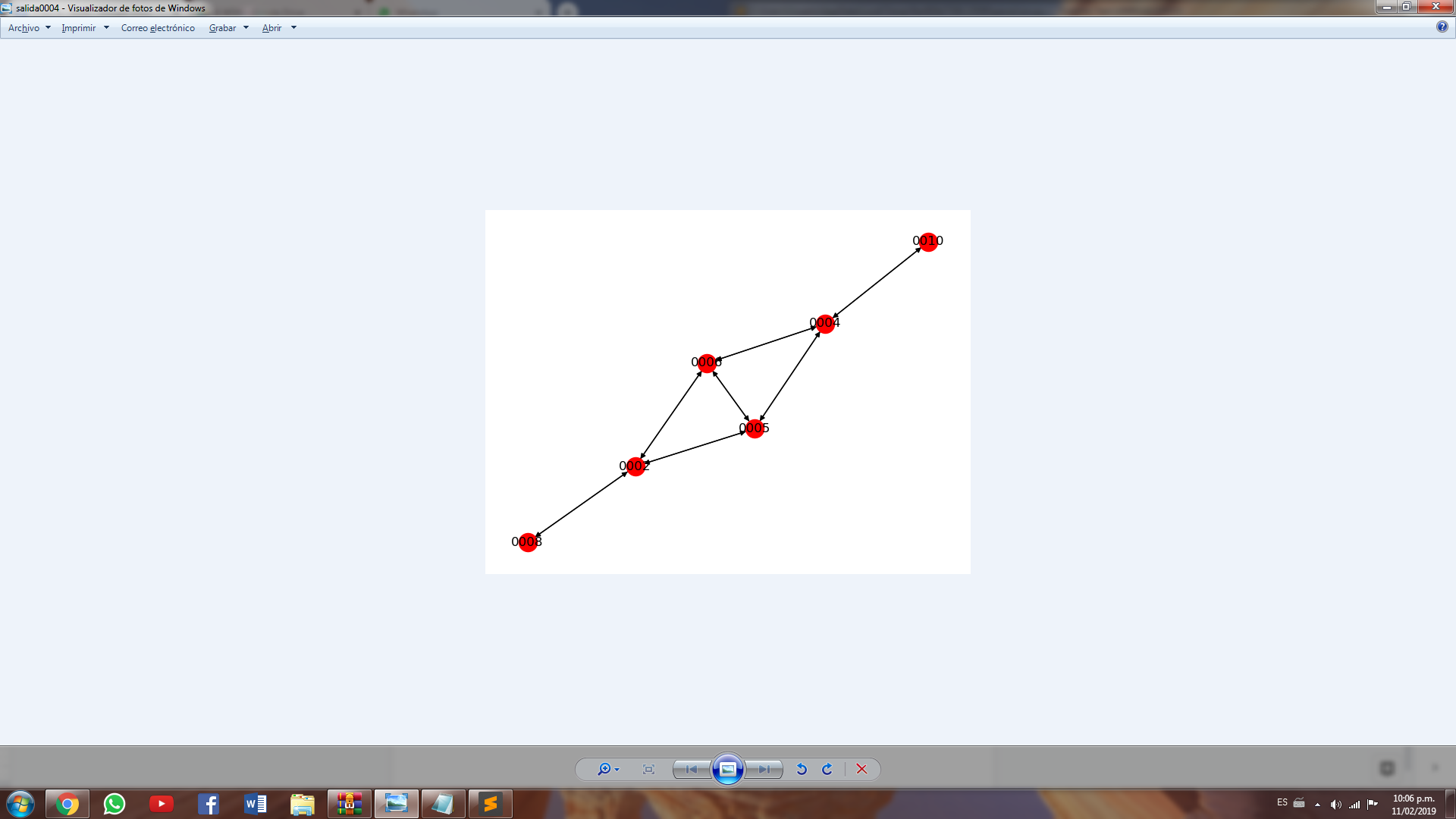


Figura 5: Dibujo de grafo direccional en python

Este es el código que se utilizó:

*#Python 3.7.2*

*import* json

*import* socket

*import* networkx *as* nx

*import* matplotlib.pyplot *as* plt

*# Metodos*

*# Recibe un query para mandarlo y regresa el JSON de respuesta*

def enviaSolicitud(*query*):

*try*:

sock.sendall(wsnQuery.encode())

received = sock.recv(1024)

*except*:

print("Excepcion de comunicacion")

*return* received

*# Configuraciones y variables*

HOST, PORT = "148.205.37.122", 6000

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

macLocal = "0010"

sock.connect((HOST, PORT))

*#Obtener las direcciones mac de todos los radios cercanos*

print("==================Todas las mac de los radios==================")

wsnQuery ='{"id": "routes", "mac": \"'+macLocal+'\"}'

respuesta = enviaSolicitud(wsnQuery)

arregloTodosRadios = json.loads(respuesta)

arregloTodosRadios.append({"hops":"-1","mac":macLocal,"next\_hop":"null"})

*for* radio in arregloTodosRadios:

print(radio['mac'])

*#Direcciones mac de las PC de la red*

arregloTodasPC = []

print("==================Todas las mac de las PC==================")

*for* radio in arregloTodosRadios:

wsnQuery ='{"id": "attached\_devices", "mac": \"'+radio['mac']+'\"}'

respuesta = enviaSolicitud(wsnQuery)

*if* str(json.loads(respuesta)) != "[]" :

arregloTodasPC.append(json.loads(respuesta))

*for* pcEnRed in arregloTodasPC:

print(pcEnRed[0])

*#Todas las rutas de mi nodo local*

print("==================Todas las rutas del radio local==================")

wsnQuery ='{"id": "routes", "mac": \"'+macLocal+'\"}'

respuesta = enviaSolicitud(wsnQuery)

diccionarioRutas = {}

arregloRutas = []

arregloRutas = json.loads(respuesta)

print(arregloRutas)

*for* ruta in arregloRutas:

diccionarioRutas[ruta['mac']] = []

rutaAuxiliar = ruta

i = 0

*while* True:

diccionarioRutas[ruta['mac']].append(rutaAuxiliar['next\_hop'])

wsnQuery = wsnQuery ='{"id": "routes", "mac": \"'+rutaAuxiliar['next\_hop']+'\"}'

respuestaActual = enviaSolicitud(wsnQuery)

i+=1

*if* i >= int(ruta['hops'])-1:

*break*

*for* subruta in json.loads(respuestaActual):

*if* subruta['mac'] == ruta['mac']:

rutaAuxiliar = subruta

*break*

print(diccionarioRutas[ruta['mac']])

*#Dibujar grafo de conexiones*

print("==================Dibujo del grafo==================")

topologia = nx.DiGraph()

*for* radio in arregloTodosRadios:

topologia.add\_node(radio['mac'])

wsnQuery ='{"id": "routes", "mac": \"'+radio['mac']+'\"}'

rutasIndiv = enviaSolicitud(wsnQuery)

*for* ruta in json.loads(rutasIndiv):

*if* ruta['hops'] == '1':

topologia.add\_edge(radio['mac'],ruta['mac'])

print("El número total de nodos en la red es de: "+str(topologia.number\_of\_nodes()))

print("El número total de enlaces entre los nodos es de: "+str(topologia.number\_of\_edges()))

nx.draw\_spectral(topologia,*with\_labels*=True)

plt.show()

*#Cerrado del socket de conexion*

sock.close()

**Conclusión**

En esta práctica pudimos trabajar con una red multisalto simple como una forma de introducción a esta topología de red. El requerimiento de dibujar la red a mano fue una forma de mostrar que una red con esta topología puede ser más complicada de lo que pareciera a simple vista y que la necesidad de programar un script que permita automatizar el proceso es una necesidad para redes más grandes.

De igual manera, pudimos ver la versatilidad de *Python* y lo útil que puede ser en el ámbito de ingeniería debido a la gran cantidad de usos que se le puede dar, por ejemplo, la actividad extra que realizamos en donde fue posible dibujar el grafo de manera automática mediante el uso de un script.

**Referencias**

* http://www.profheath.org/research/multi-hop-networking/