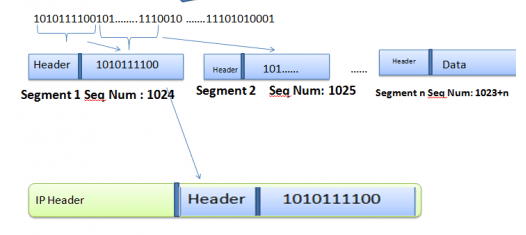
**Construyendo un sniffer de red con Python**

|  |  |
| --- | --- |
| **Nombre:** Univ. Mamani Chavez Carla Vanesa | **CI:** 9124602 LP  **Paralelo:** Martes |
| **Docente:** Lic. Gallardo Portanda Franz Ramiro | **Fecha :** 27/06/2020 |

1. Escriba un programa **Python** que interprete todos los campos de la **cabecera IP** y los despliegue en la pantalla. Pruebe su programa, ya sea **generando trafico local en su máquina o conectada a la red.** Analice la información que arroja su programa en el contexto del trafico capturado.
   1. Cabecera Ip

Es la parte que el protocolo IP añade al Segmento de datos proveniente de la capa de transporte para formar el datagrama IP. Contiene distintos campos con la información que le permitirá navegar por Internet en condiciones.

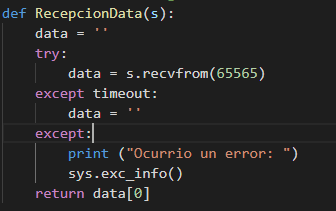
Así es como TCP ve los datos de la aplicación



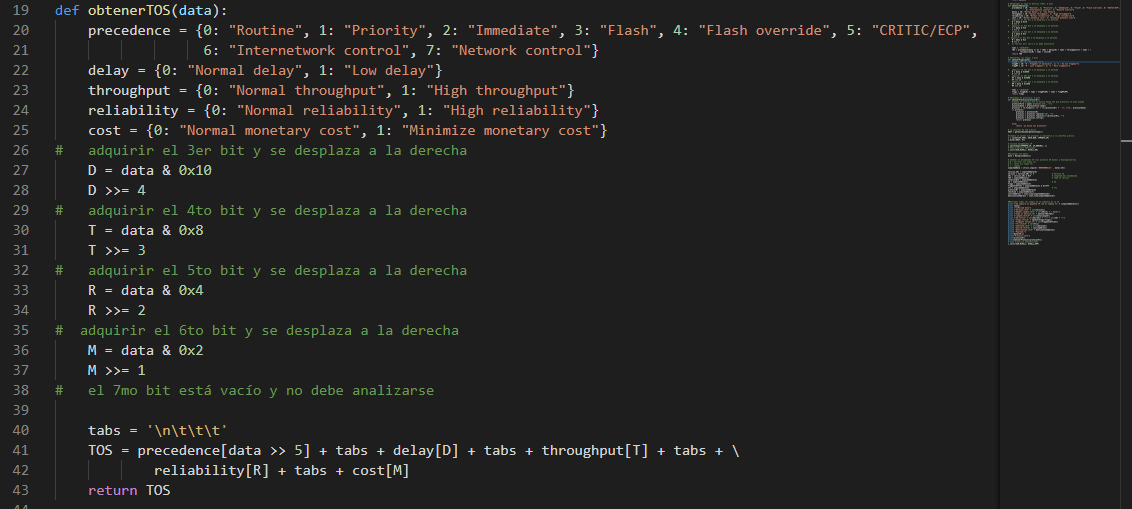
Los campos de la cabera Ip son los siguientes y trabajaremos con los siguientes:



Y primero recibimos los datos del socket, el método **recvfrom** en el módulo de socket nos ayuda a recibir todos los datos del socket. El parámetro pasado es el tamaño del búffer 65565 es el tamaño máximo del búffer.



Despues obtenemos el tipo de Servicio (Tos) del tamaño del campo: 8 bits. El cual indica la forma en que se trata un datagrama. Eso se traduce en si el datagrama tiene preferencia de encaminamiento o no. Realmente eso no se usa en Internet y todos los datagramas tienen el mismo valor ya que si se pudiese modificar piratas informáticos podrían aprovecharse de esa vulnerabilidad haciendo que sus paquetes se trataran con mayor urgencia.



**Donde:**

precedence permite indicar una procedencia.

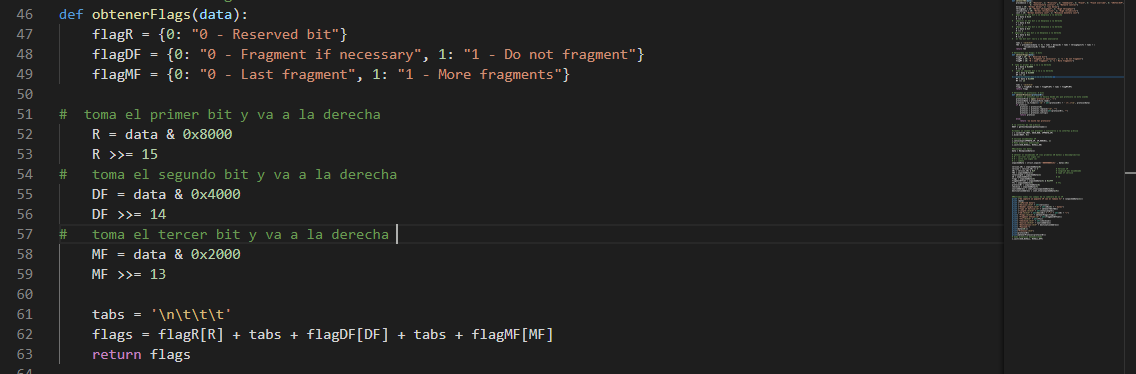
d delay, permite optimizar el retardo.

t throughput, permite optimizar la velocidad eficaz.

r reliability, permite optimizar la fiabilidad.

c cost, permite optimizar el costo (económico).

Despues pasamos a la captura de los Flags



**Donde:**

Tenemos los bit de reserva como flagR

Fragmento si es necesario o No fragmentar como flagDF

Último fragmento o Más fragmentos como flagMF

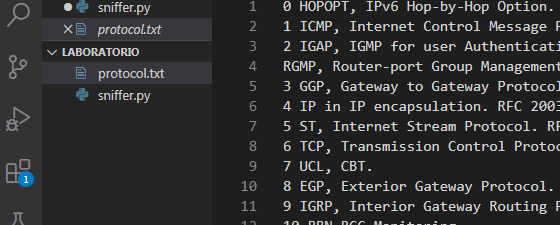
Posteriomente capturamos los protocolos donde leemos el archivo protocolos.txt con la siguiente instrucción

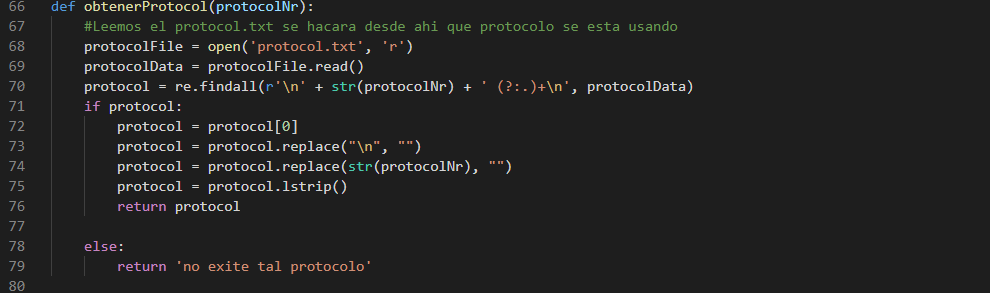
protocolFile = open('protocol.txt', 'r')

protocolData = protocolFile.read()

protocol = re.findall(r'\n' + str(protocolNr) + ' (?:.)+\n', protocolData)

Algo muy importante que el archivo protocolos.txt tiene que estar en el mismo directorio que el archivo sniffer.py

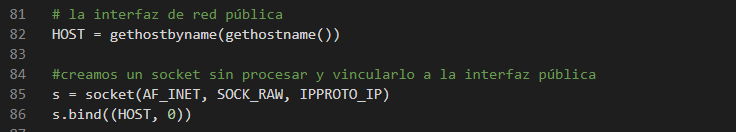




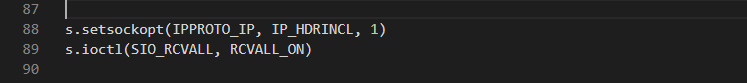
Tambien el tamaño del campo: 8 bits. Multiplexación de protocolos a nivel superior. Identifica mediante un número natural qué protocolo se ha encapsulado en este datagrama: TCP = 6, UDP = 17, ICMP = 1, IP = 0

**Continuamos con la función gethostbyname recupera la información del host correspondiente a un nombre de host de una base de datos de host.**

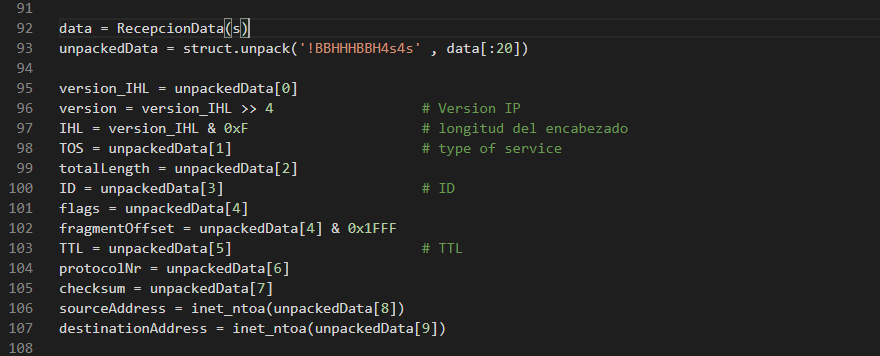
**Y tambien creamos un socket sin procesar y vincularlo a la interfaz pública.**



**Incluimos encabezados IP y el modo promiscuo habilitado**



**Recibimos los datos con data = RecepcionData(s) y adquirimos el encabezado IP (los primeros 20 bytes) y descomprimirlos.**



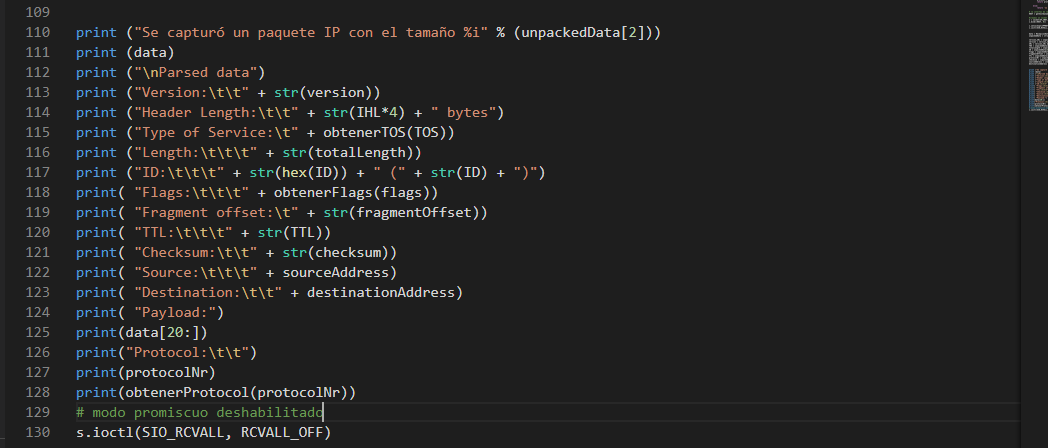
**Donde:**

**B - carácter sin signo (1)**

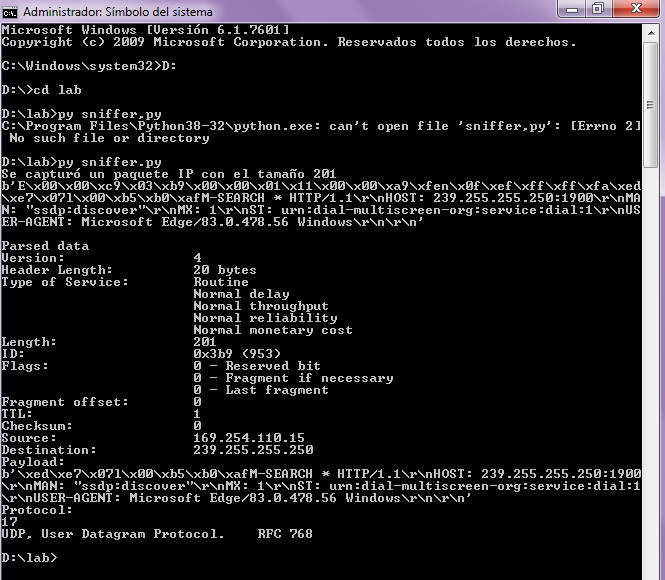
**H - corto sin signo (2)**

**s - cadena**

**Para finalizar mostramos todos los campos de la cabecera de la IP**



Lo ejecutamos el modo administrador en Simbolos de sistema (Porque estmos trabajando en el sistema operativo Windows)



En el recuerdo verde me genera error, porque cometi un error al poner el comando py sniffer.py en vez de eso puse py sniffer,py. Y genero el erro la coma en vez del punto.