PRÁCTICA 2 – MI PRIMER SOCKET

FECHA: 11/10/21

NOMBRE DEL EQUIPO: EL SIUU TEAM

PARTICIPANTES: -FISCHER SALAZAR CÉSAR EDUARDO
-LÓPEZ GARCÍA JOSÉ EDUARDO
-MEZA VARGAS BRANDON DAVID

UNIDAD ACADÉMICA: REDES DE COMPUTADORAS PANORÁMICA

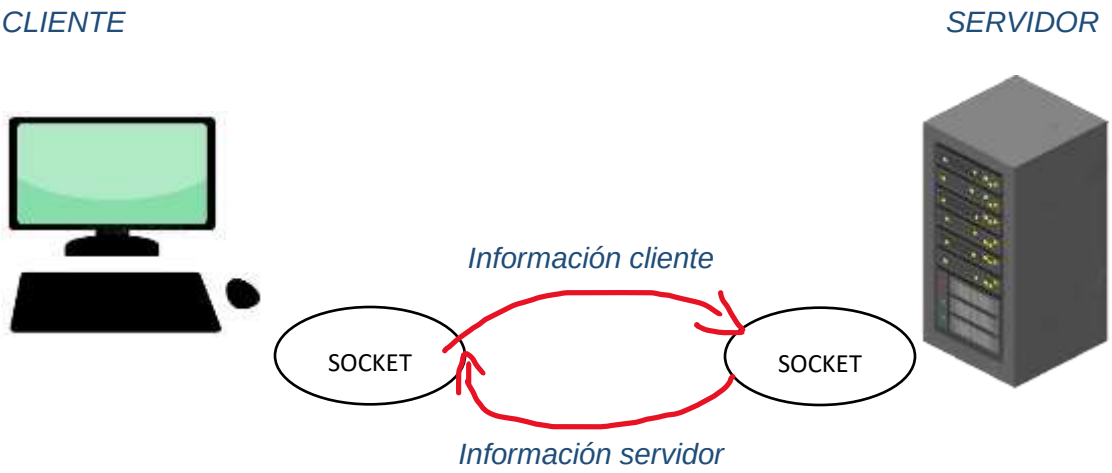


Figura 01. Analogía sockets.

OBJETIVOS

OBJETIVO PRINCIPAL: PROGRAMAR SOCKETS, RELACIONAR LA FUNCIÓN BIND CON EL SOCKET, PROGRAMAR UN CLIENTE Y ENVIAR UN MENSAJE POR LA RED. APOYÁNDONOS CON EL USO DEL ANALIZADOR DE RED WIRESHARK CAPTURAR TRAMA, DESCRIBIR LOS ENCABEZADOS DE CADA CAPA DEL MODELO TCP/IP Y CAPTURAR EN IMAGEN LA TRAMA CAPTURADA E INDICAR CADA PARTE DE LA TRAMA

OBJETIVO SECUNDARIO. VERIFICAR ENVIÓ DE UN MENSAJE A TRAVÉS DE LA RED.

ESCENARIO

LA FORMA MÁS BÁSICA DE CONSTRUIR UN SISTEMA DE COMUNICACIÓN ES IMPLEMENTANDO LA INTERFAZ DE PROGRAMACIÓN DE APLICACIONES DE SOCKET (API DE SOCKET - SOCKETS APLICACIÓN PROGRAMMING INTERFACE).

EN ESTA PRÁCTICA EL PARTICIPANTE DEBERÁ REALIZAR UN PROGRAMA QUE ENVÍE UN MENSAJE A TRAVÉS DE SU RED Y PODER SER CAPTURADO POR MEDIO DEL SOFTWARE WIRESHARK.

RECURSOS NECESARIOS PARA REALIZAR LA PRÁCTICA

- Compilador Linux y editor de Linux
- Manuales de man socket , man inet_atom, man 2 bind, man 7 ip
- Software wireshark

PARTE 1: DIAGRAMA DE FLUJO

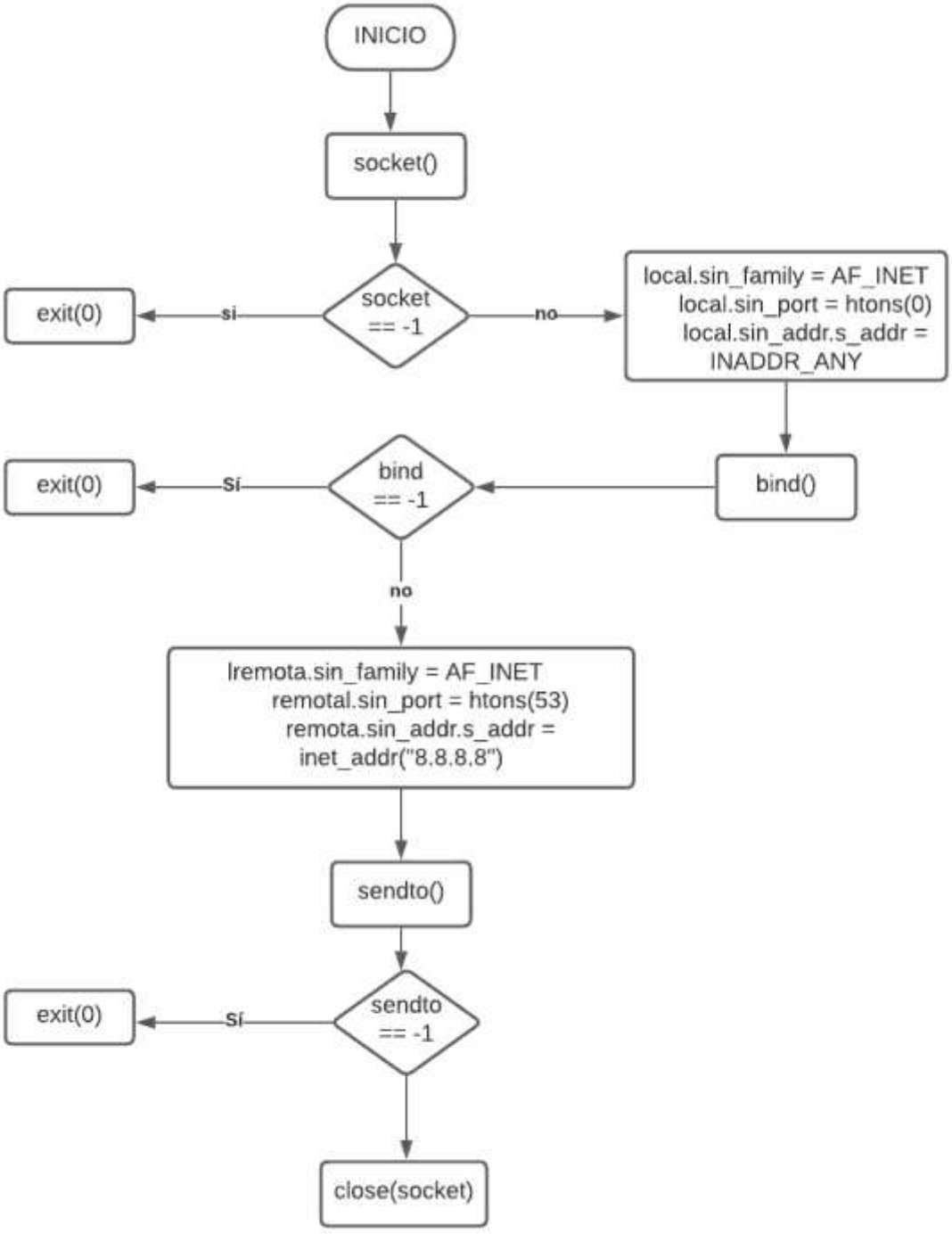


Figura 1. Diagrama de flujo del programa.

PARTE 2: CÓDIGOS, COMANDOS Y EJECUCIÓN Y EXPLICACIÓN.

2.1 INCLUYE CODIGO EXPLICANDO LÍNEA POR LÍNEA HASTA DONDE SE ABRE EL SOCKET, EL BIND.

```c
int main(){
    int udp_socket, bindValue, sendtoValue; //sendtoValue: para sendto y bindValue para bind
    struct sockaddr_in local, remota;//estructuras para el servidor y client
    unsigned char msg[100] = "Hola a todos soy un mensajito"; //Mensaje a enviar

    printf("\n\n\t\t\t*****MI PRIMER SOCKET*****\n\n");

    /*
        socket nos permite abrir un socket, AF_INET es la familia y usamos SOCK_DGRAM para abrir un socket
        udp, el 0 es para el protocolo IP en la cabecera IP a envar o recibir
    */
    udp_socket = socket(AF_INET, SOCK_DGRAM, 0);

   //Aqui indicamos si el socket se abrio correctamente o no: -1 = Error, de otra forma se abre correctamente
    if(udp_socket == -1){
        perror("\nError al abrir el socket");
        exit(0);//salimos del programa
    }else{
        perror("\nExito al abrir el socket");//se abrio el socket
        local.sin_family = AF_INET; //address family: AF_INET
        local.sin_port = htons(0); //port in network byte order, la api de socket asigna el puerto
        local.sin_addr.s_addr = INADDR_ANY;

        /*
            Con bind enlazamos un nombre a un socket, comop parametros esta el socket, la direccion local en este
            caso servidor y el tamaño de la direccion
        */
        bindValue = bind(udp_socket,(struct sockaddr *)&local,sizeof(local));

        //Si bindValue es -1 no se pudo hacer el bind
        if(bindValue == -1){
            perror("\nError en bind");
            exit(0);//salimos del programa
        }else{
            perror("\nExito en el bind");//se hizo el bind
```

Figura 2. Código hasta la parte de bind

2.2 INCLUYE LA CAPTURA DE PANTALLA DEL RESULTADO AL EJECUTAR LA PRIMER PARTE DEL PROGRAMA.

```
root@BDMV:/media/root/Mas espacio/Redes 1/Practicas/Practica 2# ./cliente


                        *****MI PRIMER SOCKET*****


Exito al abrir el socket: Success

Exito en el bind: Success
```

Figura 3. Ejecución de la primera parte

2.3 ANEXA EL CÓDIGO COMPLETO EXPLICANDO LÍNE A LÍNEA. YA CON LAS FUNCIONES LOCAL Y REMOTA, CAMBIAR EL NOMBRE DE SUS VARIABLES Y ESTRUCTURAS DE FORMA PERSONAL.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h> //L2 protocols
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <string.h>

int main(){
    int udp_socket, bindValue, sendtoValue; //sendtoValue: para sendto y bindValue para bind
    struct sockaddr_in local, remota;//estructuras para el servidor y client
    unsigned char msg[100] = "Hola a todos soy un mensajito"; //Mensaje a enviar

    printf("\n\n\t\t\t*****MI PRIMER SOCKET*****\n\n");

    /*
        socket nos permite abrir un socket, AF_INET es la familia y usamos SOCK_DGRAM para abrir un socket
        udp, el 0 es para el protocolo IP en la cabecera IP a envar o recibir
    */
    udp_socket = socket(AF_INET, SOCK_DGRAM, 0);

   //Aqui indicamos si el socket se abrio correctamente o no: -1 = Error, de otra forma se abre correctamente
    if(udp_socket == -1){
        perror("\nError al abrir el socket");
        exit(0);//salimos del programa
    }else{
        perror("\nExito al abrir el socket");//se abrio el socket
        local.sin_family = AF_INET; //address family: AF_INET
        local.sin_port = htons(0); //port in network byte order, la api de socket asigna el puerto
        local.sin_addr.s_addr = INADDR_ANY;

        /*
            Con bind enlazamos un nombre a un socket, comop parametros esta el socket, la direccion local en este
            caso servidor y el tamaño de la direccion
        */
        bindValue = bind(udp_socket,(struct sockaddr *)&local,sizeof(local));
```

```c
        //Si bindValue es -1 no se pudo hacer el bind
        if(bindValue == -1){
            perror("\nError en bind");
            exit(0);//salimos del programa
        }else{
            perror("\nExito en el bind");//se hizo el bind
            remota.sin_family = AF_INET; //address family: AF_INET
            remota.sin_port = htons(53); //port in network byte order, cambio puerto serv
            remota.sin_addr.s_addr = inet_addr("8.8.8.8"); //inet_addr: convertir cadena

            /*
                sendto la usamos para mandar mensajes a otro conector, aqui especificamos la direccion
                a donde transmitiremos el mensaje, en este caso al servidor
            */
            sendtoValue = sendto(udp_socket, msg, 100, 0, (struct sockaddr *)&remota, sizeof(remota));

            //Si devuelve -1 no se pudo enviar
            if(sendtoValue == -1){
                perror("\nError al enviar");
                exit(0);//salimos
            }else
                perror("\nExito al enviar"); //se envio el mensaje
        }
    }

    close(udp_socket); //cerramos el socket

    return 0;
}
```

Figura 4. Código completo de cliente.c

2.4 EJECUTA EN TU TERMINAL LOS COMANDOS IFCONFIG, ARP-A, PING, PING EXTENDIDO, NSLOOKUP E INCLUYE LA CAPTURA DE PANTALLA DE CADA UNO Y HAZ UN BREVE COMENTARIO

ANTES DE MOSTRAR LA CAPUTRA ES IMPORTANTE HACER LA SIGUIENTE INSTALACIÓN PARA TENER DISPONIBLES LOS COMANDOS ANTERIORES: APT-GET INSTALL NET-TOOLS

- **IFCONFIG:** Este comando muestra la información de las interfaces de red disponibles en la computadora:

```
root@BDMV:/media/root/Mas espacio/Redes 1/Practicas/Practica 2# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:69:7a:5f brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 78281sec preferred_lft 78281sec
    inet6 fe80::a00:27ff:fe69:7a5f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figura 5. Comando IFCONFIG

- **ARP-A:** ARP son las siglas en inglés de Address Resolution Protocol, este protocolo se encarga de encontrar la dirección hardware (Ethernet MAC) que corresponde a una dirección IP. El comando arp-a nos sirve para conocer nuestra IP, la IP y MAC de la puerta de enlace (gateway) y el tipo de direccionamiento:

```
root@BDMV:/media/root/Mas espacio/Redes 1/Practicas/Practica 2# arp -a
_gateway (10.0.2.2) at 52:54:00:12:35:02 [ether] on enp0s3
```

Figura 6. Comando arp -a

- **PING:** Este comando nos sirve para comprobar si una máquina esta en red o no. Se hace un diagnostico que permite hacer una verificación del estado de una determinada conexión o host local, va seguido de una dirección:

```
root@BDMV:/media/root/Mas espacio/Redes 1/Practicas/Practica 2# ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=0.280 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=64 time=0.252 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=64 time=0.370 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=64 time=0.213 ms
```

Figura 7. Comando ping

- **PING EXTENDIDO: El ping extendido realiza lo mismo que ping, con la particularidad de que este se hace de manera infinita, lo terminamos nosotros en Linux con ctrl+c:**

```
root@BDMV:/media/root/Mas espacio/Redes 1/Practicas/Practica 2# ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=0.280 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=64 time=0.252 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=64 time=0.370 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=64 time=0.213 ms
64 bytes from 10.0.2.2: icmp_seq=5 ttl=64 time=0.237 ms
64 bytes from 10.0.2.2: icmp_seq=6 ttl=64 time=0.171 ms
^C
--- 10.0.2.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 120ms
rtt min/avg/max/mdev = 0.171/0.253/0.370/0.065 ms
```

Figura 8. Comando ping extendido

- **NSLOOKUP:** Para este comando necesitamos instalar dnsutils: apt-get install dnsutils. Este comando nos sirve para saber si el DNS está resolviendo correctamente los nombres e IP's. Se usa para obtener la dirección ip conociendo el nombre y viceversa. Permite que consultemos los servidores de nombres para resolver un nombre de un host dado:

```
root@BDMV:/media/root/Mas espacio/Redes 1/Practicas/Practica 2# nslookup google.com
Server:         192.168.100.1
Address:        192.168.100.1#53

Non-authoritative answer:
Name:   google.com
Address: 216.58.195.238
Name:   google.com
Address: 2607:f8b0:4012:800::200e
```

Figura 9. Comando nslookup

2.5 INCLUYA LA CAPTURA DE PANTALLA DEL RESULTADO AL EJECUTAR EL PROGRAMA COMPLETO, CON AYUDA DEL PROGRAMA WIRE SHARK, Y EXPLIQUE LOS DATOS OBTENIDOS EN CADA CAPA DEL MODELO TCP/IP CAPTURA Y EXPLICACION PARA CADA CAPA.

```
root@BDMV:/media/root/Mas espacio/Redes 1/Practicas/Practica 2# ./cliente

                        *****MI PRIMER SOCKET*****


Exito al abrir el socket: Success

Exito en el bind: Success

Exito al enviar: Success
```

Figura 10. Ejecución completa del código.

CAPA DE ACCESO A LA RED

En la figura que se muestra a continuación podemos ver los datos obtenidos en la capa de acceso a la red del modelo TCP/IP.

```
▾ Ethernet II, Src: PcsCompu_69:7a:5f (08:00:27:69:7a:5f), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
  ▾ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
      Address: RealtekU_12:35:02 (52:54:00:12:35:02)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▾ Source: PcsCompu_69:7a:5f (08:00:27:69:7a:5f)
      Address: PcsCompu_69:7a:5f (08:00:27:69:7a:5f)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
```

Figura 11. Datos de la capa de accesos a la red

En esta parte se captura la dirección MAC destino a donde llegó nuestro paquete enviado, además de la dirección MAC fuente que fue desde donde se mandó la información, también nos muestra el tipo, en este caso IPv4.

CAPA DE INTERNET

A continuación, podemos ver los datos que arroja wireshark capturados pertenecientes a la capa de internet.

```
▾ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▾ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      0000 00.. = Differentiated Services Codepoint: Default (0)
      .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 128
    Identification: 0x1e51 (7761)
  ▾ Flags: 0x4000, Don't fragment
      0... .... .... .... = Reserved bit: Not set
      .1.. .... .... .... = Don't fragment: Set
      ..0. .... .... .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0xfffd [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 8.8.8.8
```

Figura 12. Datos de la capa de transporte

En esta capa podemos ver que nos manda la versión del IP, el tamaño del cabecero, además de mostrarnos algunas banderas. De igual forma en esta capa se indica el protocolo, en este caso fue UDP.

También, al final nos muestra la dirección IP destino y la dirección IP fuente.

CAPA DE TRANSPORTE

En la siguiente figura se muestra la información de la capa de transporte arrojada por wireshark.

```
▾ User Datagram Protocol, Src Port: 33533, Dst Port: 53
    Source Port: 33533
    Destination Port: 53
    Length: 108
    Checksum: 0x1c9c [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
```

Figura 13. Información de la capa de transporte

Los datos que nos arroja es el puerto fuente que es el 33533, el puerto destino que es el 53, mismo que se especifica en el programa, además de mostrarnos la longitud.

2.5 CON LA AYUDA DE UNA CAPTURA DE PANTALLA, IDENTIFICAR CADA PARTE DE LA TRAMA CAPTURADA.

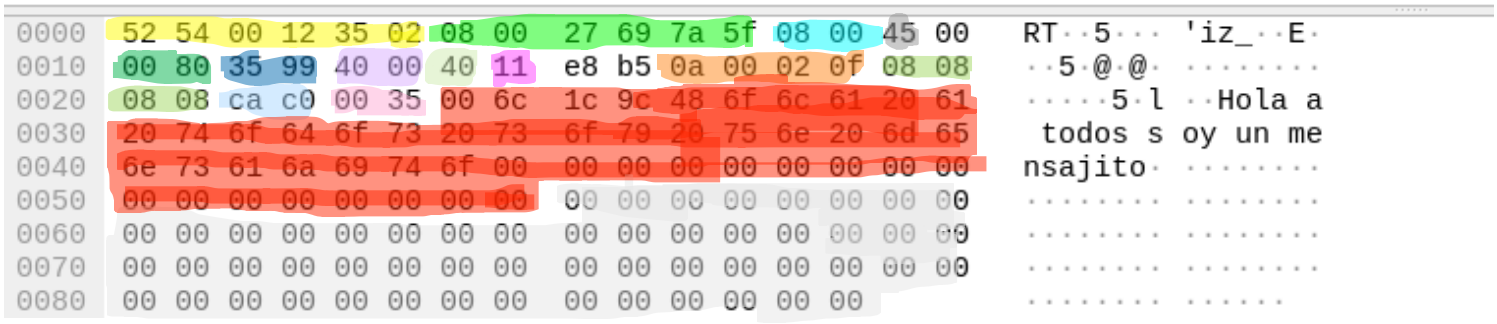
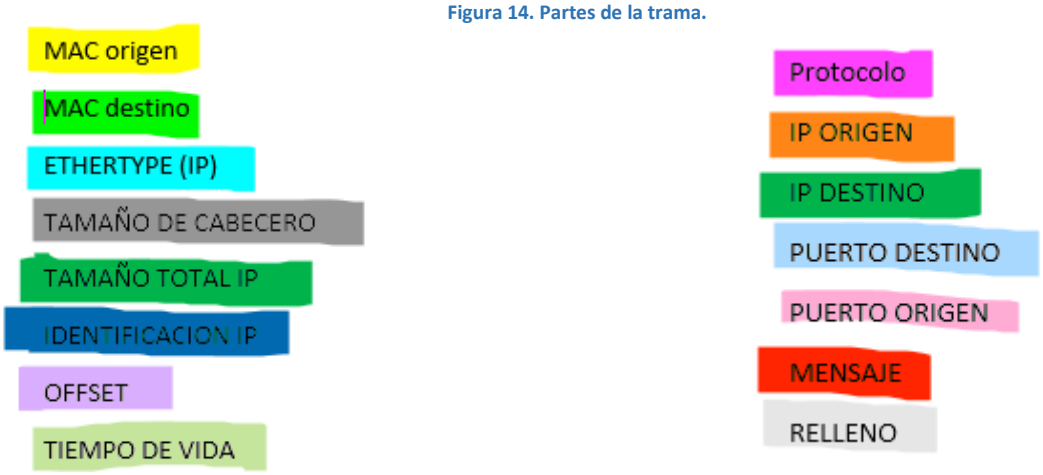


Figura 14. Partes de la trama.



2.6 MENCIONA GRAFICAMENTE Y POR ESCRITO COMO CONFIGURASTE EL WIRESHARK PARA RECIBIR LAS TRAMAS SOLO DE TU TARJETA DE RED Y HACIA EL SERVIDOR DNS.

En primer lugar, en la parte de capture escogemos la parte de option:

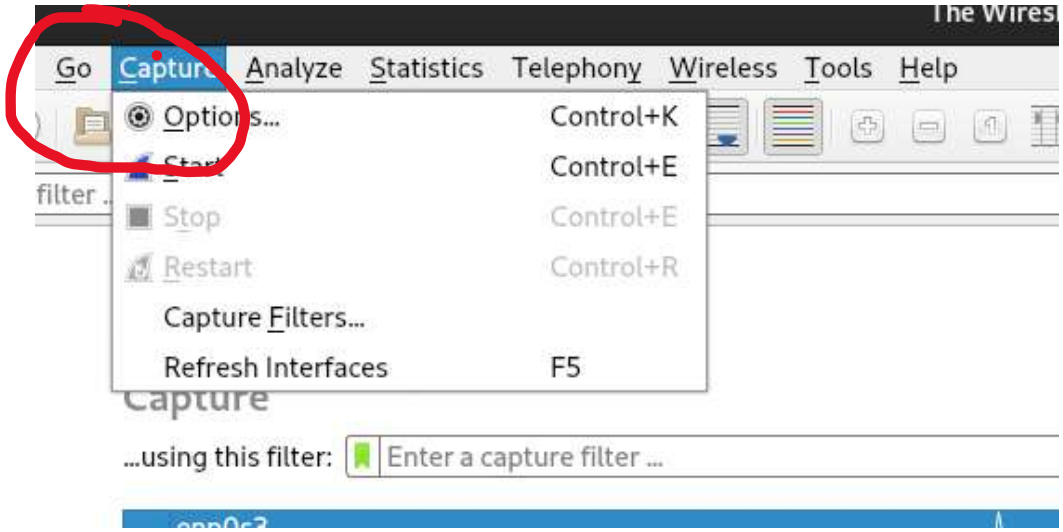


Figura 15. Primera configuración wireshark.

Posteriormente seleccionamos nuestra tarjeta de red y ponemos el filtro: ether host MAC como se ve en la siguiente imagen:

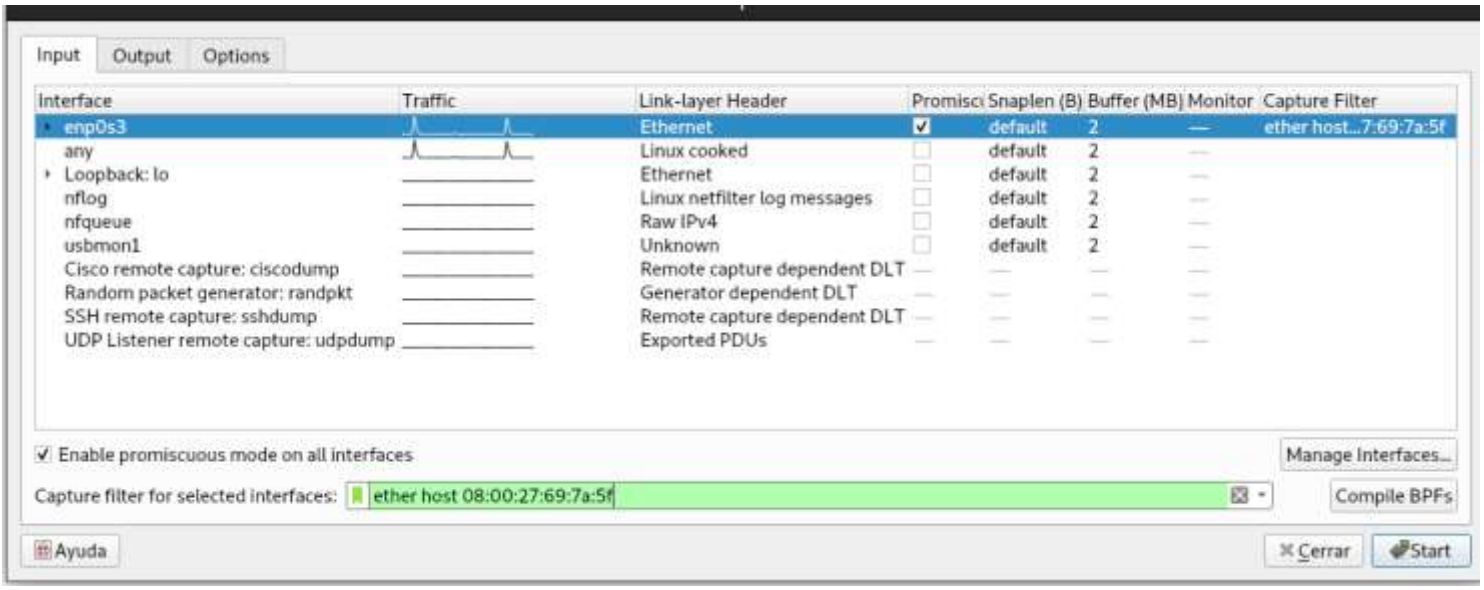


Figura 16. Aplicando filtro ether host.

Posteriormente se da clic en start para empezar con la captura de datos.

En la siguiente pantalla, tenemos que aplicar otro filtro, donde pondremos el puerto indicado en el programa, es el puerto 53:

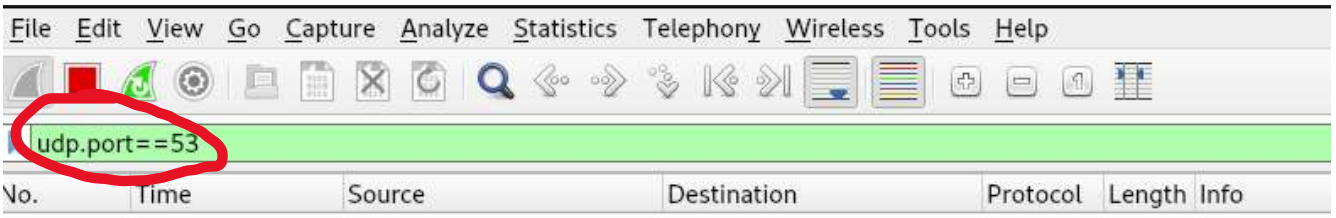
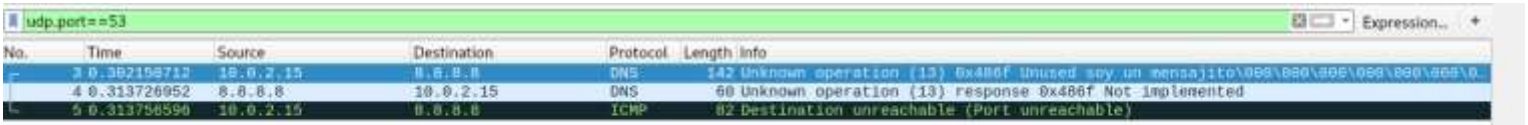


Figura 17. Aplicando el filtro del puerto.

Una vez configurado ya podemos ejecutar el programa cliente.c para comenzar a analizar los paquetes enviados:



CONTESTA LAS SIGUIENTES PREGUNTAS

¿Qué es un socket?

Un socket es conocido como un tipo de software que actúa como un punto final que funciona estableciendo un enlace de comunicación de red bidireccional entre el extremo del servidor y el programa receptor del cliente

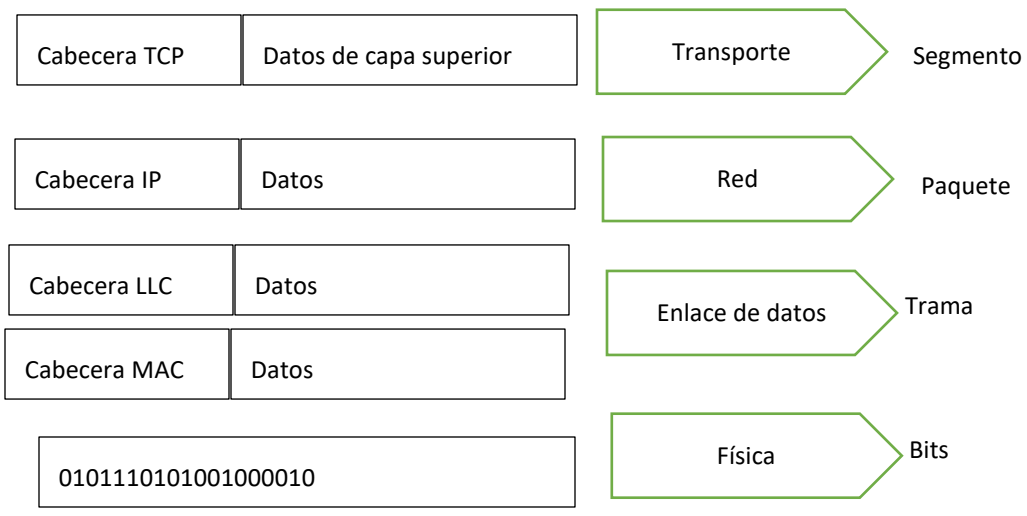
¿Qué es la API de protocolos?

Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados.

¿Menciona brevemente como es el flujo de la Información en la arquitectura TCP/IP?

La capa de aplicación proporciona a las aplicaciones la capacidad de acceder a los servicios de las demás capas y define los protocolos que utilizan las aplicaciones para intercambiar datos. La capa de transporte de esta arquitectura es responsable de proporcionar a la capa de aplicación, servicios de sesión y de comunicación de datagramas. La capa de Internet es responsable de las funciones de direccionamiento, empaquetado y enrutamiento. Y la capa de interfaz de red es responsable de la colocación de paquetes TCP/IP en la red y de la recepción de paquetes TCP/IP de fuera la red.

¿Menciona el nombre que se le va dando a la información en cada capa y sus encabezados?



¿Qué es un Socket?

Un socket es un método para la comunicación entre un programa del cliente y un programa del servidor en una red, se define, por tanto, como el punto final en una conexión.

¿Qué es un DNS?

El sistema de nombres de dominio (DNS) es el directorio telefónico de Internet. Las personas acceden a información en línea mediante nombres de dominio (como espn.com). El DNS traduce los nombres de dominio a direcciones IP para que los navegadores puedan cargar los recursos de Internet.

¿Qué es una dirección Lógica?

Es aquella dirección generada por la CPU mientras un programa se está ejecutando; tiene la característica de ser virtual, ya que no existe físicamente. Esta dirección se utiliza como referencia para acceder a la ubicación de la memoria física

¿Qué es una dirección Física?

La dirección física identifica una ubicación física en una memoria. MMU (Unidad de gestión de memoria) calcula la dirección física para la dirección lógica correspondiente. MMU también utiliza la dirección física de computación de direcciones lógicas.

¿Qué es La NIC?

Una NIC o tarjeta de interfaz de red es un dispositivo que conecta físicamente una computadora a una red. Esta conexión permite la comunicación de alta velocidad a las impresoras, enrutadores, computadoras u otros módems de banda ancha. Los tipos más comunes de tarjetas de red incluyen tarjetas Ethernet, inalámbricas y red en anillo.

¿Explique brevemente como se hace la consulta de un cliente a un servidor a través de la Red?

El cliente abre un socket para poder realizar esta conexión, posteriormente se realiza un bind, si el bind se realiza correctamente, el cliente procede a preguntar a una dirección remota si es posible mandarle datos, por su parte el servidor recibe esos datos. Todo esto usando el modelo OI, sabiendo que las capas superiores utilizan los servicios de las capas inferiores para ir generando el encapsulamiento de datos, esto lo vemos en nuestro programa con las distintas funciones que usamos.

3. CONCLUSIONES INDIVIDUALES DE CADA PARTICIPANTE DEL EQUIPO

FISCHER SALAZAR CÉSAR EDUARDO

Siendo la primera vez que tenía contacto con la programación de un socket esta práctica me ayudo a entender el cómo funcionan y en que capa se trabajan, ya que conforme fuimos avanzando pudimos construir que nos permitirá abrir un socket y modificarlo para que este nos permitiera enviar un paquete del cual logramos observar su contenido gracias a la captura de su trama que realizamos en wireshark.

LÓPEZ GARCÍA JOSÉ EDUARDO

Esta práctica nos ha ayudado a comprender y conocer mucho acerca del manual que ofrece Linux para la programación de sockets, ver cómo es su funcionamiento, qué elementos requiere para que operara de la mejor manera; y gracias al uso de la herramienta de wireshark, se logró comprender y visualizar toda la información que arrojó la apertura del socket dentro de la terminal del software.

MEZA VARGAS BRANDON DAVID

Gracias a esta práctica comprendí de mejor manera lo visto anteriormente en clase sobre sockets, las capas donde trabajan, así mismo como aprender a identificar los valores o información capturada en la trama al enviar un paquete, esto con la ayuda de wireshark, programa del cual no tenía conocimiento anteriormente, pero gracias a esta práctica logre hacer uso de este.

De igual forma, con esta práctica hacemos un acercamiento a lo que es la programación de sockets y nos damos la idea de cómo programas más grandes que involucran la red, se hacen.

Una práctica de suma importancia ya que es la base de todo lo que seguiremos viendo a lo largo del semestre.

BIBLIOGRAFÍA

1) Barcina, F. (2020). “Comandos de red de net-tools”. Obtenido de: https://delarioja.org/articulos/category-blog/134-comparativa-de-comandos-de-red-de-net-tools-y-iproute2
2) ¿Qué es un socket? (speedcheck.org)
3) ¿Qué es tarjeta de red o NIC? Definición, función y tipos de NIC | FS comunidad
4) Diferencias entre Dirección Física y Dirección Lógica (pc-solucion.es)