PROTOCOLOS DE INTERNET

1 CUATRIMESTRE DE 2024

PRÁCTICA

Trabajo Práctico N.º 1 Parte B

Comisión: FN

Profesor/a: Javier Adolfo Ouret

N°	Nombre y apellido	Carrera	DNI	Email
1	Christian Balderrama	Ing. Informática	42118900	christianbalderrama@uca.edu.ar
2	Fiorella Insfran Sanabria	Ing. Informática	96142187	fiorellainsfran@uca.edu.ar
3	Pablo Joaquin Cardozo	Ing. Informática	45178142	cardozopabloj@uca.edu.ar
4	Pablo Joaquin Margewka	Ing. Informática	37754332	pablomargewka@uca.edu.ar
5	Fabián Leonardo De Simone	Ing. Informática	39433563	fdesimone96@uca.edu.ar

El concepto básico de sockets a bajo nivel es enviar un solo paquete por vez con todos los encabezados de los protocolos completados dentro del programa en lugar de usar el kernel.

Unix proporciona dos tipos de sockets para el acceso directo a la red:

SOCK_PACKET, recibe y envía datos al dispositivo ubicado en la capa de enlace. Esto significa que el "encabezado" de la placa de red incluida en los datos puede ser escrito o leído. En general es el encabezado de Ethernet. Todos los encabezados de los protocolos subsiguientes también deben ser incluidos en los datos.

SOCK_RAW, es el que usaremos por ahora, que incluye los encabezados IP, protocolos y datos subsecuentes.

Desde el momento que lo creamos se puede enviar cualquier tipo de paquetes IP por este socket. También se pueden recibir cualquier tipo de paquetes que lleguen al host, después de que el socket fue creado, si se hace un "read()" desde él. Se puede observar que aunque el socket es una interfaz al encabezado IP, también es específico para una capa de transporte. Esto significa que para escuchar tráfico TCP, UDP, ICMP hay que crear 3 sockets sin formato por separado, usando IPPROTO_TCP, IPPROTO_UDP e IPPROTO_ICMP (los números de protocolo son 0 ó 6 para tcp, 17 para udp y 1 para icmp).

Con esta información es posible crear un simple "sniffer", que muestre todo el contenido de los paquetes TCP que se reciben. (En este ejemplo se evitan los encabezados IP y TCP, y se imprime solamente el "payload" con encabezados IP y TCP contenidos en el paquete).

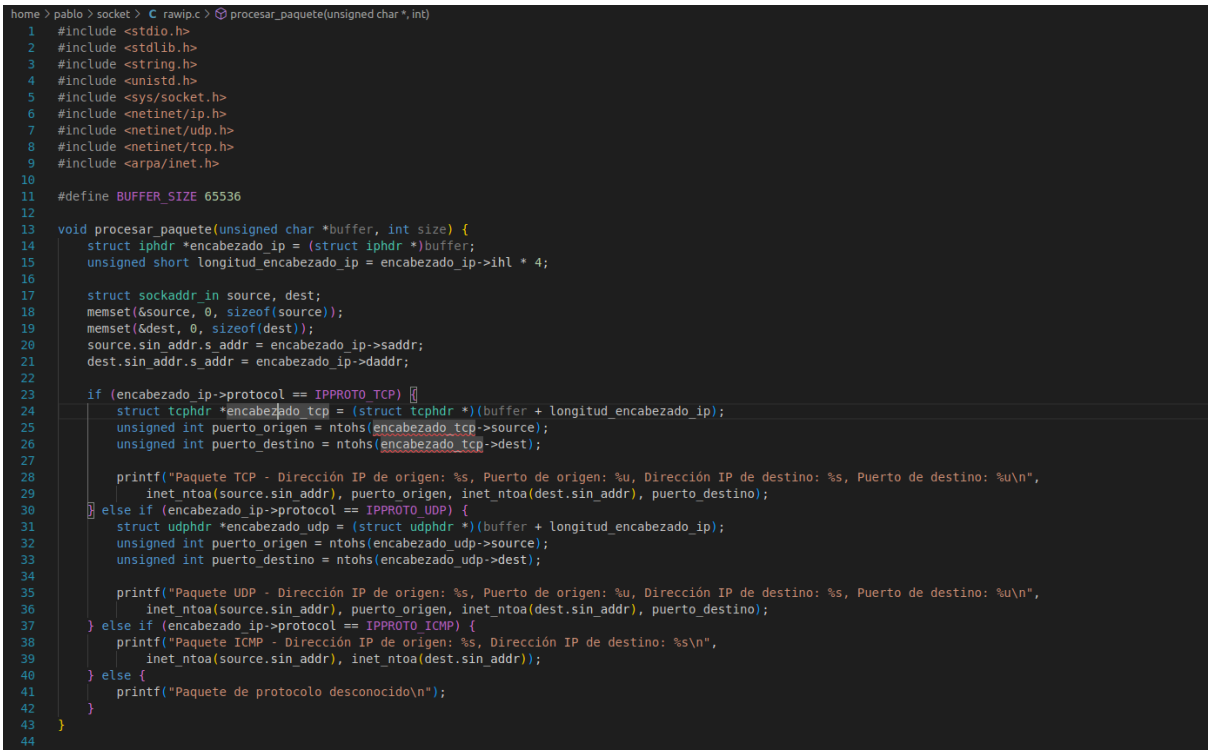
- Utilizando el código raw.c como base escribir un "sniffer" que es un programa que muestra el contenido del tráfico que llega.

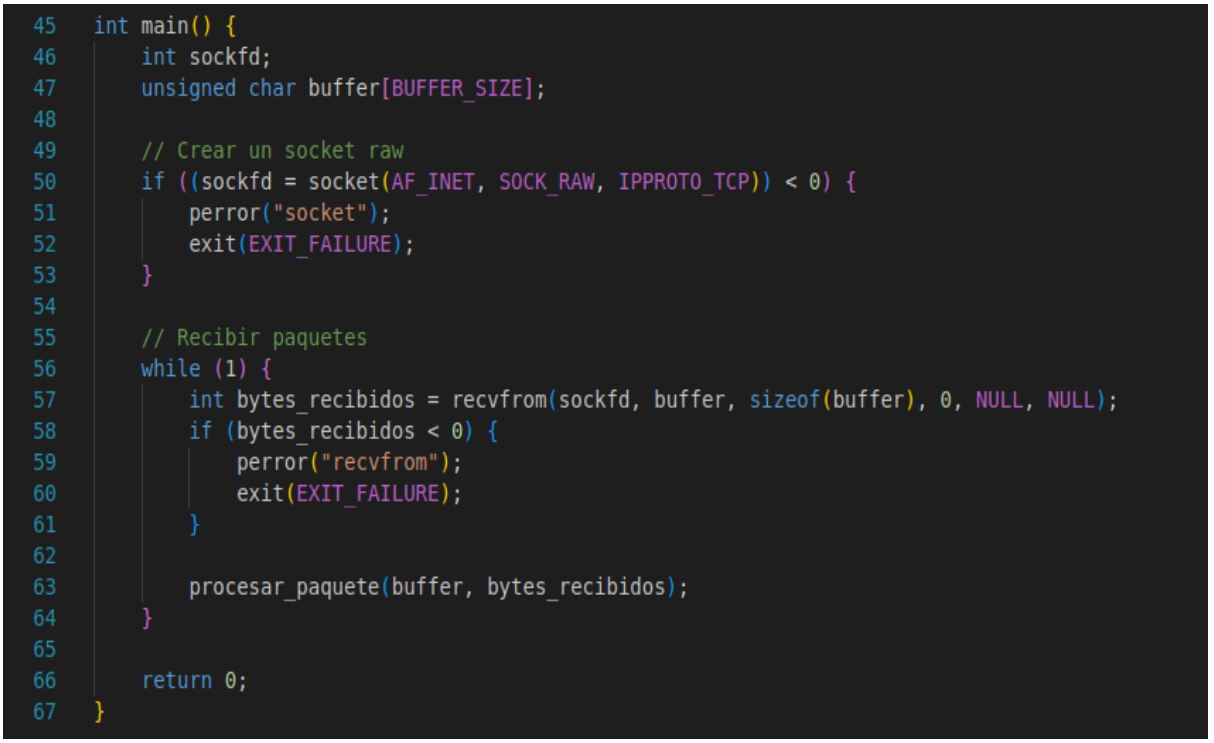
El resultado del archivo sniffer raw_socket_sniffer.c:

- Enviar al "sniffer" desde el tráfico cliente escrito en la parte A del TP1
- Enviar ICMP al "sniffer" y mostrar los resultados del LOG con comentarios.
- Mostrar resultados.

En primer lugar, para detectar los paquetes pasan a través de la red vamos a utilizar un sniffer. Este sniffer se encargará de mostrar el tipo de paquete

(TCP ,UDP o ICMP) , el puerto de entrada, puerto de salida y la dirección IP asociada. El código del sniffer utilizado es el siguiente:





Una vez que disponemos de nuestro sniffer, vamos a abrir una terminal de Linux, buscamos el directorio en donde se encuentra nuestro sniffer, compilamos y ejecutamos anteponiendo la palabra sudo para otorgarle privilegios de administrador.

```
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/socket$ cd /home/pablo/socket
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/socket$ gcc -o rawip rawip.c
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/socket$ sudo ./rawip
[sudo] password for pablo:
Paquete TCP - Dirección IP de origen: 31.13.94.52, Puerto de origen: 443, Dirección IP de destino: 31.13.94.52, Puerto de destino: 39102
Paquete TCP - Dirección IP de origen: 31.13.94.52, Puerto de origen: 443, Dirección IP de destino: 31.13.94.52, Puerto de destino: 39102
Paquete TCP - Dirección IP de origen: 140.82.114.25, Puerto de origen: 443, Dirección IP de destino: 140.82.114.25, Puerto de destino: 57422
Paquete TCP - Dirección IP de origen: 31.13.94.52, Puerto de origen: 443, Dirección IP de destino: 31.13.94.52, Puerto de destino: 39102
Paquete TCP - Dirección IP de origen: 140.82.114.25, Puerto de origen: 443, Dirección IP de destino: 140.82.114.25, Puerto de destino: 57422
Paquete TCP - Dirección IP de origen: 172.217.172.67, Puerto de origen: 443, Dirección IP de destino: 172.217.172.67, Puerto de destino: 51264
Paquete TCP - Dirección IP de origen: 172.217.172.67, Puerto de origen: 443, Dirección IP de destino: 172.217.172.67, Puerto de destino: 51264
Paquete TCP - Dirección IP de origen: 31.13.94.52, Puerto de origen: 443, Dirección IP de destino: 31.13.94.52, Puerto de destino: 39102
Paquete TCP - Dirección IP de origen: 142.251.133.35, Puerto de origen: 443, Dirección IP de destino: 142.251.133.35, Puerto de destino: 47486
Paquete TCP - Dirección IP de origen: 142.251.133.35, Puerto de origen: 443, Dirección IP de destino: 142.251.133.35, Puerto de destino: 47486
Paquete TCP - Dirección IP de origen: 142.251.134.46, Puerto de origen: 443, Dirección IP de destino: 142.251.134.46, Puerto de destino: 41684
Paquete TCP - Dirección IP de origen: 142.251.133.42, Puerto de origen: 443, Dirección IP de destino: 142.251.133.42, Puerto de destino: 56116
```

En este caso detectara solamente los paquetes TCP ya que en el socket de la función main del sniffer especificamos que solo aceptará paquetes TCP con la palabra IPPROTO_TCP

```
if ((sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_TCP)) < 0) {
```

Una vez que tengamos el sniffer corriendo vamos a poder detectar la cantidad de paquetes que circulan por la red.
En este caso lo que vamos a hacer es ejecutar el código del cliente y servidor utilizado en el punto A del trabajo práctico y veremos como el sniffer detecta la comunicación entre ambas aplicaciones.

Utilizaremos el siguiente servidor que se encarga de calcular el factorial y enviar el resultado al cliente:

```c
#include <arpa/inet.h> // inet_addr()
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#include <time.h> // clock_gettime()
#include <sys/types.h> // pid_t
```

```
// sudo ./EJ01_01_Socket_Servidor_Concurrente

#define MAX 1024
#define PORT 6667
#define SA struct sockaddr

double tiempo_transcurrido(struct timespec *inicio, struct timespec
*fin) {
    return (fin->tv_sec - inicio->tv_sec) * 1e9 + (fin->tv_nsec -
inicio->tv_nsec);
}

void func(int* sockfd) {
   struct timespec start, end;
   int connfd = *sockfd;
   int buff;
   for (;;) {
       int numBytes = read(connfd, &buff, sizeof(int));
       if (numBytes <= 0) {
           printf("El cliente ha cerrado la conexión.\n");
           break;
       }
       printf("Número recibido: %d\n", buff);

       // Calcular el factorial
       clock_gettime(CLOCK_MONOTONIC, &start);
int factorial = 1;
       for(int i = 1; i <= buff; ++i) {
           factorial *= i;
       }
       clock_gettime(CLOCK_MONOTONIC, &end);

       // Obtener el tiempo de ejecución
       double execution_time = tiempo_transcurrido(&start, &end);

       // Enviar la respuesta al cliente
       char response[MAX];
         sprintf(response, "Factorial de %d: %d\nTiempo de ejecución:
%.0f nanosegundos\nPID del proceso hijo: %d\n", buff, factorial,
execution_time, getpid());
       write(connfd, response, sizeof(response));

       if (strncmp("SALIR", response, 4) == 0) {
```

```
            printf("Salgo del servidor...\n");
            break;
        }
    }
    close(connfd);
    exit(0);
}

int main() {
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    else
        printf("Socket creado...\n");
    bzero(&servaddr, sizeof(servaddr));

    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
printf("Falla socket bind ...\n");
        exit(0);
    }
    else
        printf("Se hace el socket bind ..\n");

    if ((listen(sockfd, 5)) != 0) {
        printf("Falla el listen ...\n");
        exit(0);
    }
    else
        printf("Servidor en modo escucha ...\n");

    while (1) {
        unsigned int len = sizeof(cli);
        connfd = accept(sockfd, (SA*)&cli, &len);
        if (connfd < 0) {
```

```
            printf("Falla al aceptar datos en el servidor...\n");
            exit(0);
        }
        else
            printf("El servidor acepta al cliente ...\n");

        int pid = fork();
        if (pid < 0) {
            printf("Falla al crear el proceso...\n");
        }
        else if (pid == 0) { // Proceso hijo
                close(sockfd); // Cerrar el descriptor de archivo del
servidor en el proceso hijo
            func(&connfd);
        }
        else { // Proceso padre
            char response[MAX];
            sprintf(response, "PID del proceso padre: %d\n", getpid());
            write(connfd, response, sizeof(response));
            close(connfd); // Cerrar el socket en el proceso padre
        }
    }

    close(sockfd);
}
```

Por otra parte utilizaremos el siguiente cliente, que envía una serie de números y espera recibir el resultado por parte del servidor

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // read(), write(), close()
#include <string.h> // bzero()
#include <sys/socket.h>
#include <netinet/in.h> // struct sockaddr_in
#include <arpa/inet.h> // inet_addr()

// sudo ./EJ01_01_Socket_Cliente_B
```

```c
#define MAX 1024
#define PORT 6667
#define SA struct sockaddr

void func(int sockfd) {
   char buffer[MAX] = {0};
   int leovalor;

   for (int i = 1; i <= 5; i++) {
       // Enviar número al servidor
       write(sockfd, &i, sizeof(int));

       // Leer respuesta del servidor
       leovalor = read(sockfd, buffer, MAX);
       printf("%s\n", buffer);

       printf("Enviado: %d - ", i);
   }

   // Leer respuesta final del servidor
   leovalor = read(sockfd, buffer, MAX);
   printf("%s\n", buffer);
}

int main() {
   int sockfd;
   struct sockaddr_in servaddr;

   // socket: creo socket y lo verifico
   sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd == -1) {
       printf("Falla la creación del socket...\n");
       exit(0);
   }
   else
       printf("Socket creado ..\n");
   bzero(&servaddr, sizeof(servaddr));

   // asigno IP, PORT
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
   servaddr.sin_port = htons(PORT);
```

```c
    // Conecto los sockets entre cliente y servidor
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("Falla de conexión con servidor...\n");
        exit(0);
    }
    else
        printf("Conectado al servidor..\n");

    // Función para el chat
    func(sockfd);

    //Cierro el socket
    close(sockfd);
}
```

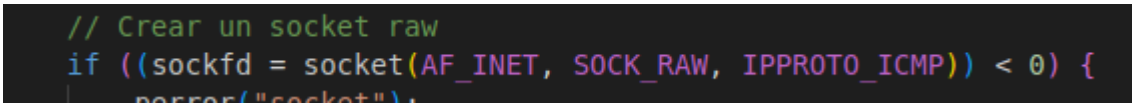
Para detectar el tráfico entre ambas aplicaciones primero abriremos el sniffer, luego el servidor y luego el cliente.
Como el cliente y el servidor utilizan el puerto 6667 para conectarse, y estamos ejecutando ambas aplicaciones desde un entorno local, entonces deberíamos poder encontrar con el sniffer, al menos un paquete, con el puerto 6667 y con la IP 127.0.0.1.

A continuación mostramos los resultados obtenidos:

```
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/socket$ cd /home/pablo/socket
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/socket$ gcc -o rawip rawip.c
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/socket$ sudo ./rawip
[sudo] password for pablo:
Paquete TCP - Dirección IP de origen: 31.13.94.52, Puerto de origen: 443, Dirección IP de destino: 31.13.94.52, Puerto de destino: 34332
Paquete TCP - Dirección IP de origen: 34.160.144.191, Puerto de origen: 443, Dirección IP de destino: 34.160.144.191, Puerto de destino: 37508
Paquete TCP - Dirección IP de origen: 34.160.144.191, Puerto de origen: 443, Dirección IP de destino: 34.160.144.191, Puerto de destino: 37508
Paquete TCP - Dirección IP de origen: 34.98.75.36, Puerto de origen: 443, Dirección IP de destino: 34.98.75.36, Puerto de destino: 41906
Paquete TCP - Dirección IP de origen: 34.98.75.36, Puerto de origen: 443, Dirección IP de destino: 34.98.75.36, Puerto de destino: 41906
Paquete TCP - Dirección IP de origen: 142.251.133.67, Puerto de origen: 443, Dirección IP de destino: 142.251.133.67, Puerto de destino: 35208
Paquete TCP - Dirección IP de origen: 142.251.133.67, Puerto de origen: 443, Dirección IP de destino: 142.251.133.67, Puerto de destino: 35208
Paquete TCP - Dirección IP de origen: 142.251.133.67, Puerto de origen: 443, Dirección IP de destino: 142.251.133.67, Puerto de destino: 35208
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 6667, Dirección IP de destino: 127.0.0.1, Puerto de destino: 56674
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 6667, Dirección IP de destino: 127.0.0.1, Puerto de destino: 56674
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 6667, Dirección IP de destino: 127.0.0.1, Puerto de destino: 56674
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 6667, Dirección IP de destino: 127.0.0.1, Puerto de destino: 56674
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 6667, Dirección IP de destino: 127.0.0.1, Puerto de destino: 56674
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 6667, Dirección IP de destino: 127.0.0.1, Puerto de destino: 56674
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 6667, Dirección IP de destino: 127.0.0.1, Puerto de destino: 56674
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 6667, Dirección IP de destino: 127.0.0.1, Puerto de destino: 56674
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 6667, Dirección IP de destino: 127.0.0.1, Puerto de destino: 56674
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 56674, Dirección IP de destino: 127.0.0.1, Puerto de destino: 6667
Paquete TCP - Dirección IP de origen: 142.251.133.35, Puerto de origen: 443, Dirección IP de destino: 142.251.133.35, Puerto de destino: 55084
```

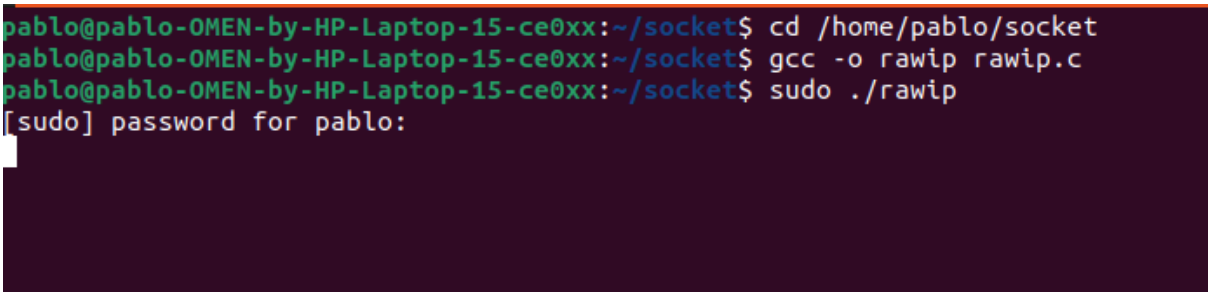
Efectivamente podemos ver que existen varios paquetes TCP, utilizando el puerto 6667 y con IP 127.0.0.1, con lo cual pudimos demostrar que el sniffer pudo detectar el tráfico entre estas dos aplicaciones.

A continuación vamos a hacer que nuestro sniffer pueda leer paquetes ICMP.
Para esto vamos a hacer una pequeña modificación en el código del sniffer.
En el socket de la función main vamos a cambiar IPPROTO_TCP por IPPROTO_ICMP tal como muestra la siguiente imagen



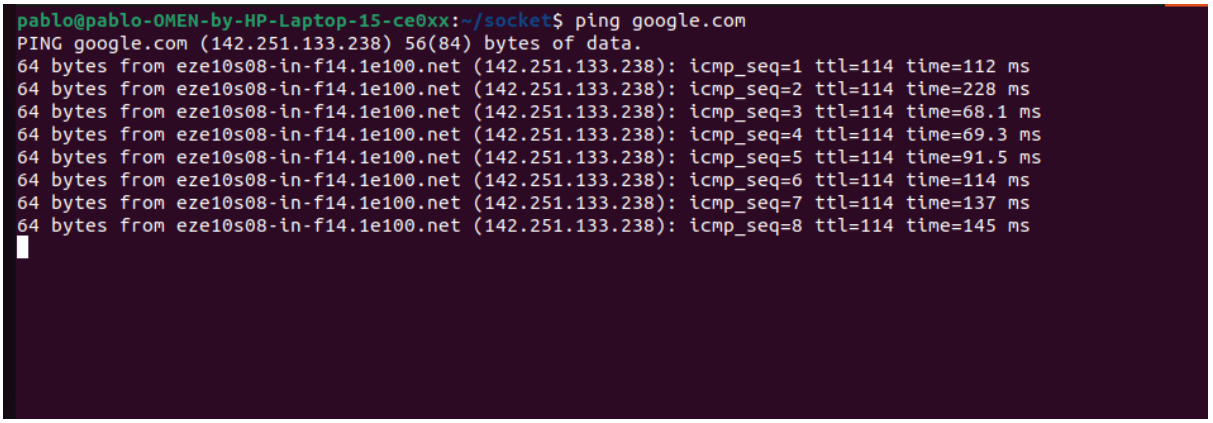
El objetivo ahora será leer paquetes ICMP. Para eso, vamos a utilizar un servicio que utiliza ICMP que es ping. Vamos a hacer un ping desde nuestra PC hacia cualquier otra ip como google.com y vamos a observar como el sniffer detecta este tipo de tráfico.

Ejecutamos nuevamente el sniffer con esta nueva modificación.



Podemos observar que a diferencia de antes, ahora no aparece ningún paquete, como si aparecía anteriormente

Posteriormente, abrimos una nueva terminal y ejecutamos el comando “ping google.com”, que nos mostrara la siguiente salida:



La salida del sniffer es:

```
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/socket$ cd /home/pablo/socket
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/socket$ gcc -o rawip rawip.c
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/socket$ sudo ./rawip
[sudo] password for pablo:
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
Paquete ICMP - Dirección IP de origen: 142.251.133.238, Dirección IP de destino: 142.251.133.238
```

Podemos observar que al utilizar el comando ping, nuestro sniffer detecto la presencia de paquetes ICMP a través de la red.