CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE SISTEMAS ELECTRÓNICOS

PROYECTO FINAL

PACKET SNIFFER

Redes de computadoras I

Ingeniería en Sistemas Computacionales

Profesor: Javier Santiago Cortes López

De Anda Medina Rene Rosendo ID: 260866

Gonzales Rivera Diana Esmeralda ID: 280109

Morquecho Canales Ernesto Alonso ID: 351214

Saucedo Díaz Valeria Michelle ID: 352288

Fecha de Entrega: 17 de Diciembre del 2024

ÍNDICE

INTRODUCCIÓN

El “Packet Sniffing” (rastreo de paquetes) es un método para interceptar cada paquete a medida que fluye a través de la red, es decir, una técnica en la cual el usuario “huele” los datos pertenecientes a otros usuarios de la red. (1)

Los programas “Packet Sniffer”, como el desarrollado en el presente proyecto, son programas que se utilizan para leer paquetes que viajan a través de la capa de red o Protocolo de control de transmisión/Protocolo de Internet (TCP/IP). Funcionan como una herramienta administrativa, ya que los administradores de red los utilizan para monitorear y validar el tráfico de la red. (1)

El siguiente reporte busca documentar y demostrar el desarrollo del proyecto final de la materia “Redes de computadoras I”, el cual consiste en generar un software que realice las funciones básicas de un “Packet Sniffer”, tomando como referencia el funcionamiento de “WireShark” (analizador de protocolos de red) que permita la captura y análisis de paquetes, dependiendo de las necesidades del usuario.

OBJETIVOS

El software deberá contar con

I. Funcionalidades básicas (parada/inicio, filtros de captura, exportación de resultados, áreas definidas para mostrar información en pantalla, entre otras).
II. Área en donde se muestre el tráfico capturado.
III. Área en donde se muestre la información de manera estructurada del paquete capturado que se seleccione en el área de tráfico capturado.
IV. Área en donde se muestre el contenido “raw” del paquete capturado que se seleccione en el área de tráfico capturado.
V. Cuatro tipos de filtrado (ip fuente, ip destino, puerto fuente, puerto destino, protocolo, entre otros.).
VI. Exportación a archivo “CSV” (requerido) o Excel (opcional), del tráfico capturado.
VII. Interfaz amigable con el usuario

CONCEPTOS NECESARIOS

Librerías

NPCAP

Es la biblioteca de captura y envío de paquetes del Proyecto Nmap para Microsoft Windows. Implementa la API Pcap abierta utilizando un controlador de kernel de

Windows personalizado junto con nuestra compilación de Windows de la excelente biblioteca libpcap. Esto permite que el software de Windows capture el tráfico de red sin procesar (incluidas redes inalámbricas, Ethernet por cable, tráfico de host local y muchas VPN) mediante una API simple y portátil. (2)

WINSOCK

También conocido como Windows Sockets 2, permite crear aplicaciones avanzadas de Internet, intranet y otras compatibles con red para transmitir datos de aplicación a través de la conexión, independientemente del protocolo de red que se use. Sigue el modelo de Windows Open System Architecture (WOSA), el cual define una interfaz de proveedor de servicios estándar (SPI) entre la interfaz de programación de aplicaciones (API), con sus funciones exportadas y las pilas de protocolos. (3)

- ***#include <winsock2.h>***: proporciona las herramientas básicas para trabajar con sockets.
- ***#include <ws2tcpip.h>***: encargada de las funcionalidades avanzadas relacionadas con TCP/IP.

Protocolos

Se tomaron en cuenta los protocolos TCP y UDP para la elaboración del filtrado más sencillo, de igual forma se tomaron en cuenta los siguientes protocolos no son totalmente dependientes de TCP o de UDP.

Protocolo	Capa OSI	Protocolo de Transporte	Descripción
ARP	Enlace de Datos (capa 2)	Ninguno	Mapea direcciones IP a direcciones MAC en una red local. Utilizado para la comunicación en redes Ethernet.
ICMP	Red (capa 3)	Ninguno	Proporciona mensajes de control y error en la comunicación de red, como los utilizados por ping y traceroute.
ICMPv6	Red (capa3)	Ninguno	Similar a ICMP, pero diseñado para IPv6. Incluye mensajes como Neighbor Solicitation y Neighbor Advertisement.
IGMP	Red (capa3)	Ninguno	Gestiona la membresía de grupos multicast en redes IPv4.
MLD	Red (capa3)	Ninguno	Similar a IGMP, pero diseñado para IPv6.
DHCP	Aplicación (capa 7)	Ninguno	Asigna dinámicamente direcciones IP y otros parámetros de configuración a dispositivos en una red.

SSDP	Aplicación (capa 7)	UDP (Puerto 1900)	Permite el descubrimiento de dispositivos en una red local mediante mensajes multicast UDP.
mDNS	Aplicación (capa 7)	UDP (Puerto 5353)	Realiza la resolución de nombres en redes locales sin la necesidad de un servidor DNS centralizado.
NDP	Red (capa 3)	Ninguno	Neighbor Discovery Protocol para IPv6, similar a ARP para IPv4.
SCTP	Aplicación (capa 7)	Ninguno (Independiente)	Proporciona características de transporte avanzadas como multi-streaming y multi-homing.

DESARROLLO

Para dar comienzo al proyecto, se tomó como base el Packet Sniffer proveniente del sitio oficial de NPCAP ya definido y compartido por el profesor (Figura 1), constando de tres scripts que daban como resultado una pequeña visualización del tráfico de paquetes en un sitio web (Figura 2). Dicho sitio proporcionado por NPCAP , se utilizó como una guía para la elaboración y comprensión de lo que es en esencia un analizador de grafico de red.

El equipo decidió llevarlo a cabo en Windows, por practicidad al trabajar en el software. Aunque tuvo un grado de dificultad, ya que se tuvo que configurar el entorno para el funcionamiento de la librería responsable de las funciones que permiten capturar el tráfico, aun asi el equipo en conjunto encontró la manera de que esto fuera posible. Dando pauta a implementar el ejemplo dado por el NPCAP, el cual se presenta en las siguientes imágenes.

```c
/*
 * nsniff.c
 */

#include <pcap/pcap.h>
#include <stdio.h>
#include <stdlib.h>

void call_me(u_char *user, const struct pcap_pkthdr *pkthdr,
             const u_char *packetd_ptr) {
  printf("You just recieved a packet!\n");
}

int main(int argc, char const *argv[]) {
  char *device = "enp0s3"; // remember to replace this with your device name
  char error_buffer[PCAP_ERRBUF_SIZE];
  int packets_count = 5;

  pcap_t *capdev = pcap_open_live(device, BUFSIZ, 0, -1, error_buffer);

  if (capdev == NULL) {
    printf("ERR: pcap_open_live() %s\n", error_buffer);
    exit(1);
  }

  if (pcap_loop(capdev, packets_count, call_me, (u_char *)NULL)) {
    printf("ERR: pcap_loop() failed!\n");
    exit(1);
  }

  return 0;
}
```

Figura 1. 1. Primera parte del Packet Sniffer.

```c
/*
 * nsniff.c
 */

#include <arpa/inet.h>
#include <netinet/ip.h>
#include <pcap/pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int link_hdr_length = 0;

void call_me(u_char *user, const struct pcap_pkthdr *pkthdr,
             const u_char *packetd_ptr) {
  packetd_ptr += link_hdr_length;
  struct ip *ip_hdr = (struct ip *)packetd_ptr;

  char packet_srcip[INET_ADDRSTRLEN];             // source ip address
  char packet_dstip[INET_ADDRSTRLEN];             // destination ip address
  strcpy(packet_srcip, inet_ntoa(ip_hdr->ip_src));
  strcpy(packet_dstip, inet_ntoa(ip_hdr->ip_dst));
  int packet_id = ntohs(ip_hdr->ip_id),           // identification
      packet_ttl = ip_hdr->ip_ttl,                // Time To Live
      packet_tos = ip_hdr->ip_tos,                // Type Of Service
      packet_len = ntohs(ip_hdr->ip_len),         // header length + data length
      packet_hlen = ip_hdr->ip_hl;                // header length

  printf("***********************************"
         "***********************************\n");
  printf("ID: %d | SRC: %s | DST: %s | TOS: 0x%x | TTL: %d\n", packet_id,
         packet_srcip, packet_dstip, packet_tos, packet_ttl);
}

int main(int argc, char const *argv[]) {
  char *device = "enp0s3";
  char error_buffer[PCAP_ERRBUF_SIZE];
  int packets_count = 5;

  pcap_t *capdev = pcap_open_live(device, BUFSIZ, 0, -1, error_buffer);

  if (capdev == NULL) {
    printf("ERR: pcap_open_live() %s\n", error_buffer);
    exit(1);
  }
```

Figura 1.2. Segunda parte del Sniffer

```
47    int link_hdr_type = pcap_datalink(capdev);
48
49    switch (link_hdr_type) {
50      case DLT_NULL:
51        link_hdr_length = 4;
52        break;
53      case DLT_EN10MB:
54        link_hdr_length = 14;
55        break;
56      default:
57        link_hdr_length = 0;
58    }
59
60    if (pcap_loop(capdev, packets_count, call_me, (u_char *)NULL)) {
61      printf("ERR: pcap_loop() failed!\n");
62      exit(1);
63    }
64
65    return 0;
66  }
```

Figura 1.2. Segunda parte del Packet Sniffer.

```
60     struct bpf_program bpf;
61       bpf_u_int32 netmask;
62
63       char *filters = "host www.duckduckgo.com";
64
65   if (pcap_compile(dev, &bpf, filters, 0, netmask) == PCAP_ERROR) {
66       printf("ERR: pcap_compile() %s", pcap_geterr(dev));
67   }
68
69   if (pcap_setfilter(dev, &bpf)) {
70       printf("ERR: pcap_setfilter() %s", pcap_geterr(dev));
71   }
```

Figura1.3. Bloque de código que conecta al host del sitio web (Parte 3).

```
$ cc -o nsniff nsniff.c -lpcap
$ sudo ./nsniff
********************************************************************************
ID: 35344 | SRC: 40.114.177.156 | DST: 40.114.177.156 | TOS: 0x0 | TTL: 64
PROTO: ICMP | TYPE: 8 | CODE: 0 |
********************************************************************************
ID: 0 | SRC: 192.168.8.4 | DST: 192.168.8.4 | TOS: 0x0 | TTL: 109
PROTO: ICMP | TYPE: 0 | CODE: 0 |
********************************************************************************
ID: 14704 | SRC: 40.114.177.156 | DST: 40.114.177.156 | TOS: 0x0 | TTL: 1
PROTO: TCP | FLAGS: S/-/- | SPORT: 41684 | DPORT: 60180 |
********************************************************************************
ID: 35616 | SRC: 40.114.177.156 | DST: 40.114.177.156 | TOS: 0x0 | TTL: 64
PROTO: ICMP | TYPE: 8 | CODE: 0 |
********************************************************************************
ID: 0 | SRC: 192.168.8.4 | DST: 192.168.8.4 | TOS: 0x0 | TTL: 109
PROTO: ICMP | TYPE: 0 | CODE: 0 |
********************************************************************************
ID: 19066 | SRC: 40.114.177.156 | DST: 40.114.177.156 | TOS: 0x0 | TTL: 1
PROTO: TCP | FLAGS: S/-/- | SPORT: 43220 | DPORT: 60180 |
********************************************************************************
ID: 35728 | SRC: 40.114.177.156 | DST: 40.114.177.156 | TOS: 0x0 | TTL: 64
PROTO: ICMP | TYPE: 8 | CODE: 0 |
********************************************************************************
ID: 0 | SRC: 192.168.8.4 | DST: 192.168.8.4 | TOS: 0x0 | TTL: 109
PROTO: ICMP | TYPE: 0 | CODE: 0 |
********************************************************************************
ID: 58529 | SRC: 40.114.177.156 | DST: 40.114.177.156 | TOS: 0x0 | TTL: 1
PROTO: TCP | FLAGS: S/-/- | SPORT: 43732 | DPORT: 60180 |
********************************************************************************
ID: 36282 | SRC: 40.114.177.156 | DST: 40.114.177.156 | TOS: 0x0 | TTL: 64
PROTO: ICMP | TYPE: 8 | CODE: 0 |
```

Figura 2. Representación del contenido final mostrado en la consola.

Una vez que se comprendió como es que funcionaba el programa, dado como ejemplo, para la implementación y desarrollo de nuestro propio Sniffer , se optó por modificar el ambiente dentro del IDE de Qt (framework multiplataforma orientado a objetos) para desarrollar una interfaz gráfica amigable con el usuario, basándose en el diseño de WireShark y en los requerimientos del proyecto . Gracias a esto se nos facilitó la programación de la libreria pcap en conjunto con la interfaz gráfica.

Después de realizar las configuraciones adecuadas en el programa, como lo fueron la manera en que se iban aplicar ciertos filtros de captura y el modo en que se iban a recibir los paquetes, se logró llegar a los requerimientos sobre mostrar el tráfico

de un dispositivo seleccionado, asi como la información de un paquete especifico dentro del tráfico del dispositivo.

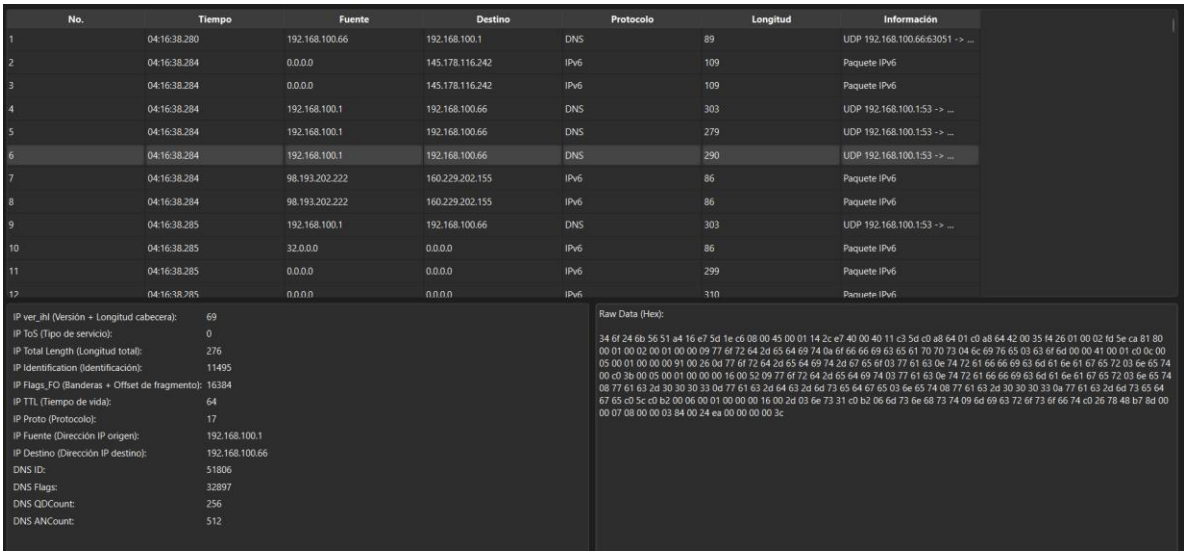

Figura3. Captura del tráfico de nuestro programa

Respecto a los protocolos principales que se configuraron en el programa además de los ya mencionados fueron los siguientes:

Protocolo	Tipo	Características	Usos
TCP	Capa de transporte	Confiable, orientado a la conexión, garantiza el orden y la integridad de los datos, control de flujo y congestión.	Navegación web, FTP, correo electrónico
UDP	Capa de transporte	No confiable, sin conexión, bajo overhead, adecuado para transmisión en tiempo real donde la velocidad es prioritaria sobre la confiabilidad.	Streaming, juegos en línea, DNS
IPv4	Capa de red	Direcciones de 32 bits, notación decimal punteada, escasez de direcciones, no incluye seguridad intrínseca.	Redes actuales, infraestructura de Internet
IPv6	Capa de red	Direcciones de 128 bits, notación hexadecimal, amplio espacio de direcciones, seguridad integrada (IPsec), autoconfiguración mejorada, elimina NAT.	Nuevas redes, dispositivos IoT, infraestructura futura de Internet

Para identificar cada protocolo, se realizó una estructura por protocolo, la cual contiene los datos de red de cada paquete que se encuentre con ese protocolo especifico.

```
/* IPv4 header */
typedef struct ip_header {
    u_char ver_ihl;        // Versión (4 bits) + Longitud de cabecera (4 bits)
    u_char tos;            // Tipo de servicio
    u_short tlen;          // Longitud total
    u_short identification; // Identificación
    u_short flags_fo;      // Banderas (3 bits) + Desplazamiento de fragmento (13 bits)
    u_char ttl;            // Tiempo de vida (TTL)
    u_char proto;          // Protocolo
    u_short crc;           // Suma de verificación del encabezado
    ip_address saddr;      // Dirección IP origen
    ip_address daddr;      // Dirección IP destino
    u_int op_pad;          // Opciones + Padding
} ip_header;

/* TCP header */
```

Figura 4. Encabezados de las estructuras de cada protocolo

Para la parte del uso de filtrados se llegó a la conclusión de que estos ayudarían a tener una eficiencia mejorada para la captura de los paquetes, una selección de información sobre el paquete más precisa y un uso de recursos reducido. Durante la elaboración del programa se tomaron en cuenta 5 tipos:

- **Por protocolo:** Especifica el protocolo que se va a usar (TCP, UDP, ARP, etc.).
- **Por dirección IP de destino como de origen asi como los puertos**

Para su implementación dentro del programa se siguió de forma estricta la sintaxis brindada por la librería pcap y sus funciones.

Comprendiendo el funcionamiento del programa, se procedió a mostrar la información relevante de los paquetes, como lo son

- ***Banderas:*** son indicadores de control presentes en los encabezados de protocolos como TCP e IPv4 y gestionan el estado de las conexiones, controlan fragmentaciones y otros comportamientos específicos.
- ***Tiempo de vida:*** es el campo en el encabezado IP que limita la vida útil del paquete en la red y previene que los paquetes queden atrapados en bucles de enrutamiento.
- ***Protocolo de red:*** son el conjunto de reglas que determinan como se transmiten y reciben los datos en la red.
- ***Encabezado:*** es la sección de metadatos en un paquete que contiene información esencial para la entrega y procesamiento de los datos, como lo son las direcciones, control de errores, gestión de conexiones, etc.

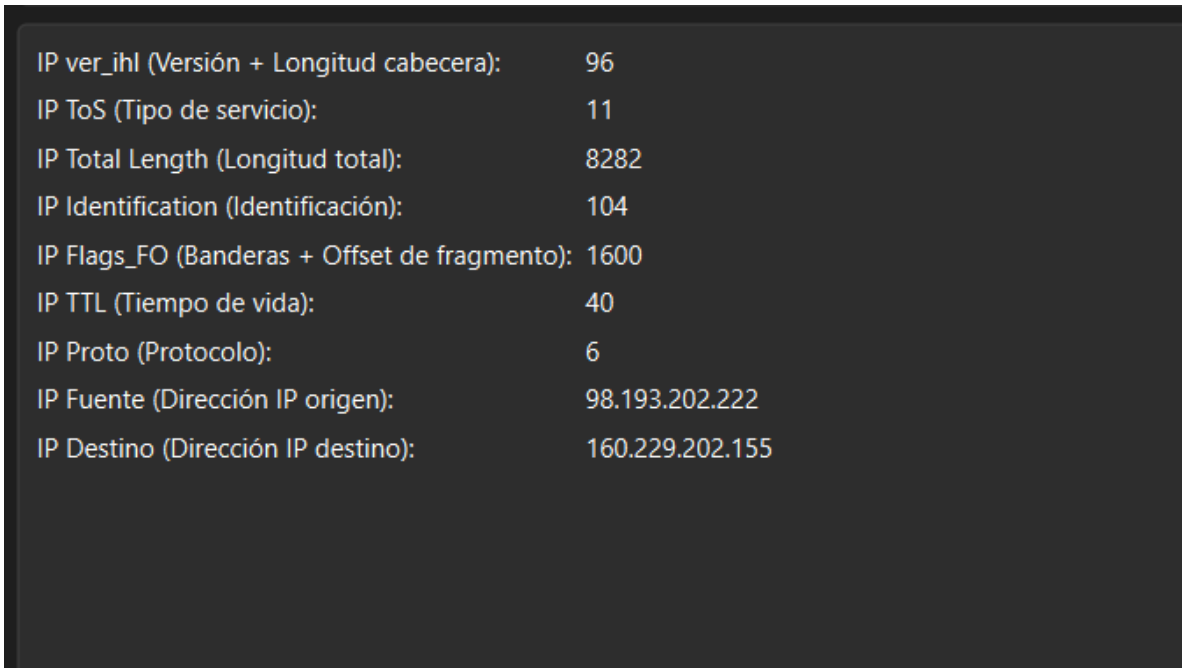

Figura 5. Información que es almacena en la estructura del protocolo especifico.

Asi mismo se realizó un apartado que mostrara el contenido en crudo de los paquetes.

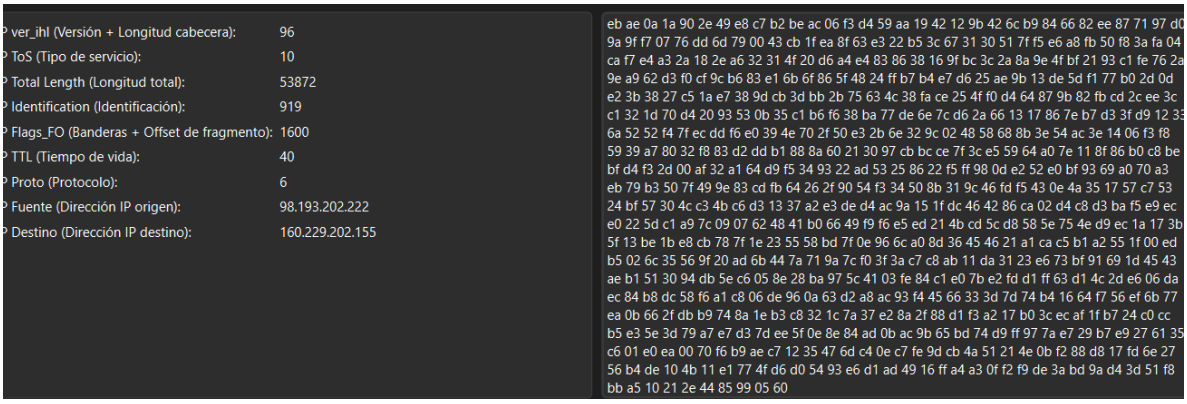

Figura 6. Formas en las que demostramos la información de los paquetes

COMPLICACIONES

Entre las complicaciones a las que se enfrentó el equipo para llevar a cabo el desarrollo del software, destaca la configuración del entorno en el cual se iba a trabajar, en este caso, Windows por medio del entorno de desarrollo de Visual Studio. Gracias a ciertas complicaciones que se tuvieron, se optó por una opción diferente con la realización de la interfaz. Provocando que el equipo tuviera que migrar a Qt, teniendo que familiarizarse con el mismo y con su forma de trabajar, siendo este de mucha ayuda gracias a su compatibilidad con el lenguaje de programación del proyecto, siendo C/C++.

Respecto al tema del filtrado, fue difícil la identificación de los filtros debido a problemas relacionados con la sintaxis, en relación con los protocolos, fue complicado aplicar las diferentes estructuras para mostrar la información pertinente a cada paquete respetando su protocolo y características.

VIDEO

https://eduuaa-my.sharepoint.com/:v:/g/personal/al260866_edu_uaa_mx1/EYboLSVN41tKgkIPQPrAdygB006sGouAqvDJR7oscRQJzA?referrer=Teams.TEAMS-ELECTRON&referrerScenario=MeetingChicletGetLink.view

REPOSITORIO EN GIT HUB

https://github.com/starcrash16/Sniffalant

CONCLUSIÓNES

1. Rene De anda Medina

A pesar de las dificultades con la biblioteca npcap, se logró dominar sus funciones. Este proyecto fue una oportunidad valiosa para aplicar los conocimientos de redes y programación de manera práctica. Personalmente, fue un gran aprendizaje a pesar de los desafíos, se alcanzó el resultado esperado, lo que demuestra la efectiva aplicación de los conceptos teóricos y prácticos adquiridos en el curso.

2. Diana Esmeralda González Rivera

Durante la elaboración del proyecto, se reforzaron los conocimientos teóricos sobre redes, específicamente el modelo OSCI, que permite entender las diferentes capas de red donde se aplican varios protocolos y filtros al tráfico. Esto facilitó una comprensión más profunda sobre la identificación de paquetes y el manejo de grandes volúmenes de información en un solo dispositivo. A pesar de las dificultades con la biblioteca npcap y su lógica, logramos comprender cada función. Personalmente, este proyecto fue un gran aprendizaje, ya que, aunque desafiante, se alcanzó el resultado esperado aplicando conocimientos de la materia y de programación.

3. Valeria Michelle Saucedo Díaz

El proyecto Packet Sniffer fue todo un reto, pero también una experiencia que me dejó mucho aprendizaje. Al principio, adaptarme al entorno Qt no fue fácil; entender cómo

funcionan sus componentes y cómo organizar la interfaz fue complicado. Pero logré comprenderla y pude sacarle provecho para que la interfaz funcionara y tuviera un buen diseño.

Otro desafío fue entender cómo están formados los paquetes de red, sus partes y qué significaba cada sección, como los encabezados y las banderas.

No creo ser la única de mis compañeros de equipo que ha quedado satisfecha con el resultado del proyecto. En sí, la materia se me dificultó un poco, hubo muchos conceptos que no conocía ni sabía cómo funcionaban, mis compañeros fueron de gran apoyo ante todas mis dudas sobre el desarrollo del proyecto, motivándome a yo también conocer a fondo los conceptos del proyecto

4. Ernesto Alonso Morquecho Canales

En este proyecto el reforzamiento de los conocimientos de la materia de Redes de Computadoras fue bastante importante ya que se utiliza gran parte de ellos para su elaboración. Aunque sin duda, la parte más complicada del proyecto para mi, fue lograr que el IDE utilizara correctamente las librerías de Npcap, ya que investigar cómo configurarlas fue un proceso bastante pesado, confuso y lleno de errores, pero a pesar de los desafíos, fue muy interesante ver cómo funcionan aplicaciones como Wireshark y lograr el objetivo propuesto sin duda fue algo bastante gratificante. Al final, toda esa investigación rindió frutos, ya que aprendí mucho sobre captura y análisis de tráfico de red, ver todo lo que circula en esta y que hay detrás de, también el manejo de librerías y uso de nuevas tecnologías hasta el momento desconocidas para mi.

REFERENCIAS

1. *Packet sniffing: a brief introduction*. (s. f.). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/1166620
2. *NPCAP: Windows Packet Capture Library & Driver*. (s. f.). Npcap. https://npcap.com/
3. Stevewhims. (2023, 13 junio). *Windows Sockets 2 - Win32 apps*. Microsoft Learn. https://learn.microsoft.com/es-es/windows/win32/winsock/windows-sockets-start-page-2