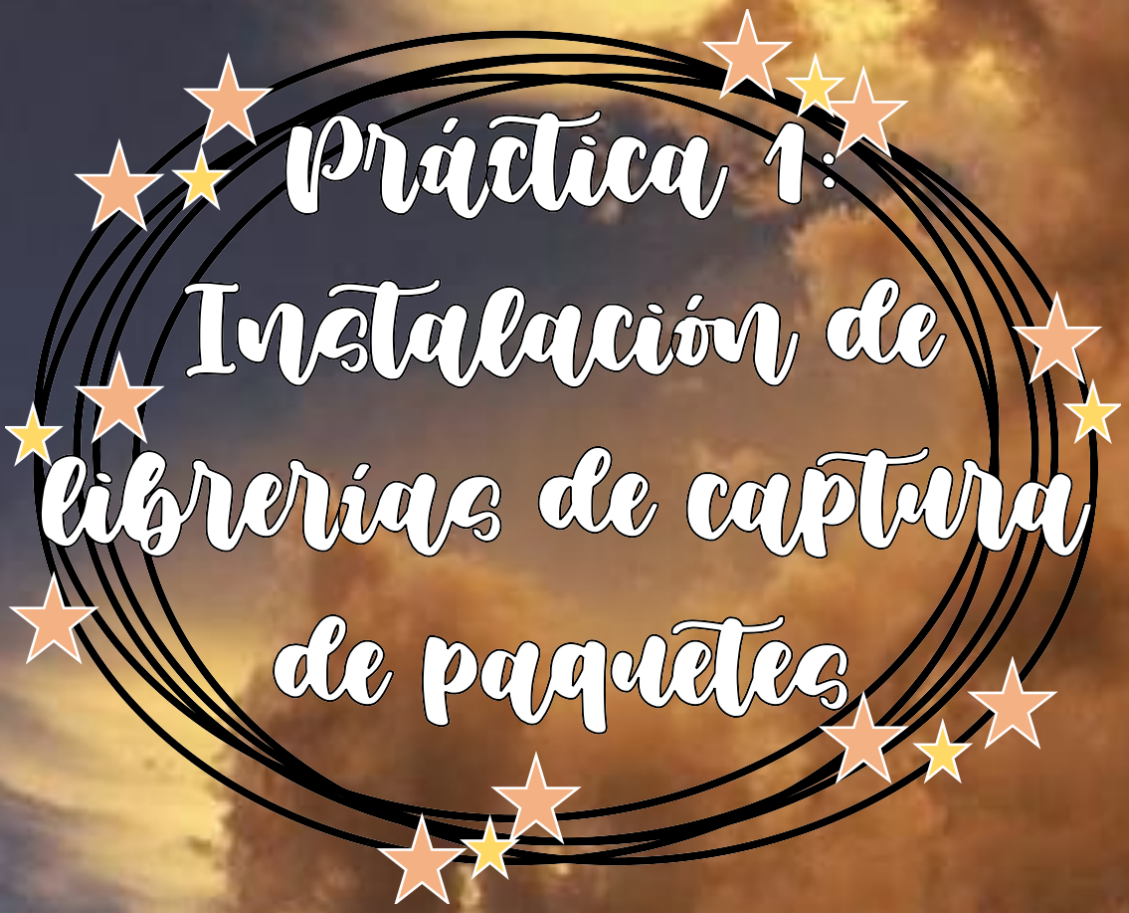
INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO



INTEGRANTES:

GONZÁLEZ MORA ERIKA GISELLE
OLIVARES MÉNEZ GLORIA OLIVA

PROFESOR: Moreno Cervantes Axel Ernesto

REDES DE COMPUTADORAS
2CV16
11/03/2021

Índice

1. Introducción

1.1. Instalación de librerías de captura de paquetes

Para realizar aplicaciones escritas en lenguaje C que necesiten realizar captura de paquetes o tramas ethernet, es necesaria la instalación de una librería esencial. En la práctica se hará uso del paquete de librerías de npcap.

Aclaremos que, durante 14 años, WinPcap fue el paquete libpcap estándar para Windows. Pero cuando este lanzó Windows 10 sin compatibilidad con NDIS 5, WinPcap no pudo mantenerse al día.

Windows 10 introdujo requisitos estrictos de firma de controladores que WinPcap no puede cumplir. Y npcap es totalmente compatible, con sus controladores probados y confirmados por Microsoft.

Construido sobre la base de código WinPcap probada y verdadera, con una serie de nuevas características interesantes y ampliamente probado con las versiones actualmente compatibles de Windows, Npcap es el futuro de WinPcap.

También, para realizar aplicaciones escritas en lenguaje Java que necesiten realizar captura de paquetes o tramas ethernet, es necesaria la instalación de esta librería importante. Además de realizar la configuración de la biblioteca PCAP4J en NetBeans. Aquí es esencial el registro de nuevas variables de entorno dado que npcap será instalado en Windows\System32\Npcap\ por default. En la siguiente práctica se mostrarán los pasos que se siguen para la realización de estas actividades en c y java.

2. Desarrollo

2.2. Encapsulación del paquete

La estructura de la trama de Ethernet agrega encabezados y tráilers a la PDU de Capa 3 para encapsular el mensaje que se envía.

Tanto el encabezado como el tráiler de Ethernet tienen varias secciones de información que el protocolo Ethernet utiliza. Cada sección de la trama se denomina campo. Hay dos estilos de tramas de Ethernet: el IEEE 802.3 (original) y el IEEE 802.3 revisado (Ethernet).

Las diferencias entre los estilos de tramas son mínimas. La diferencia más significativa entre el IEEE 802.3 (original) y el IEEE 802.3 revisado es el agregado de un delimitador de inicio de trama (SFD) y un pequeño cambio en el campo Tipo que incluye la Longitud, tal como se muestra en la siguiente figura.

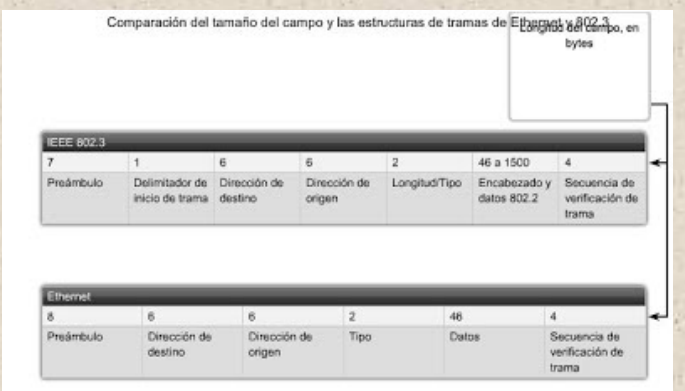


Fig. 1. IEEE 802.3 original vs IEEE 802.3 revisado.

2.3. Tamaño de la trama de Ethernet

El estándar Ethernet original definió el tamaño mínimo de trama en 64 bytes y el tamaño máximo de trama en 1518 bytes. Esto incluye todos los bytes del campo Dirección MAC de destino a través del campo Secuencia de verificación de trama (FCS). Los campos Preámbulo y Delimitador de inicio de trama no se incluyen en la descripción del tamaño de una trama. El estándar IEEE 802.3ac, publicado en 1998, amplió el tamaño de trama máximo permitido a 1522 bytes. Se aumentó el tamaño de la trama para que se adapte a una tecnología denominada Red de área local virtual (VLAN). Las VLAN se crean dentro de una red conmutada y se presentarán en otro curso.
Si el tamaño de una trama transmitida es menor que el mínimo o mayor que el máximo, el dispositivo receptor descarta la trama. Es posible que las tramas descartadas se originen en colisiones u otras señales no deseadas y, por lo tanto, se consideran no válidas.

Los campos Preámbulo (7 bytes) y Delimitador de inicio de trama (SFD) (1 byte) se utilizan para la sincronización entre los dispositivos de envío y de recepción. Estos ocho primeros bytes de la trama se utilizan para captar la atención de los nodos receptores. Básicamente, los primeros bytes le indican al receptor que se prepare para recibir una trama nueva.

2.4. Campo Dirección MAC de destino

El campo Dirección MAC de destino (6 bytes) es el identificador del receptor deseado. Como recordará, la Capa 2 utiliza esta dirección para ayudar a los dispositivos a determinar si la trama viene dirigida a ellos. La dirección de la trama se compara con la dirección MAC del dispositivo. Si coinciden, el dispositivo acepta la trama.

2.5. Campo Dirección MAC de origen

El campo Dirección MAC de origen (6 bytes) identifica la NIC o interfaz que origina la trama. Los switches también utilizan esta dirección para ampliar sus tablas de búsqueda. El rol de los switches se analizará más adelante en este capítulo.

2.6. Campo Longitud/Tipo

El campo Longitud/Tipo (2 bytes) define la longitud exacta del campo Datos de la trama. Esto se utiliza posteriormente como parte de la FCS para garantizar que el mensaje se reciba adecuadamente. En este campo debe ingresarse una longitud o un tipo. Sin embargo, sólo uno u otro podrá utilizarse en una determinada implementación. Si el objetivo del campo es designar un tipo, el campo Tipo describe qué protocolo se implementa.

El campo denominado Longitud/Tipo sólo aparecía como Longitud en las versiones anteriores del IEEE y sólo como Tipo en la versión DIX. Estos dos usos del campo se combinaron oficialmente en una versión posterior del IEEE, ya que ambos usos eran comunes. El campo Tipo de la Ethernet II se incorporó a la actual definición de trama del 802.3. La Ethernet II es el formato de trama de Ethernet que se utiliza en redes TCP/IP. Cuando un nodo recibe una trama, debe analizar el campo Longitud/Tipo para determinar qué protocolo de capa superior está presente. Si el valor de los dos octetos es equivalente a 0x0600 hexadecimal o 1536 decimal o mayor que éstos, los contenidos del campo Datos se codifican según el protocolo indicado.

2.7. Campos Datos y Relleno

Los campos Datos y Relleno (de 46 a 1500 bytes) contienen los datos encapsulados de una capa superior, que es una PDU de Capa 3 genérica o, con mayor frecuencia, un paquete IPv4. Todas las tramas deben

tener al menos 64 bytes de longitud. Si se encapsula un paquete pequeño, el Pad se utiliza para aumentar el tamaño de la trama hasta alcanzar este tamaño mínimo.

2.8. Campo Secuencia de verificación de trama

El campo Secuencia de verificación de trama (FCS) (4 bytes) se utiliza para detectar errores en la trama. Utiliza una comprobación cíclica de redundancia (CRC). El dispositivo emisor incluye los resultados de una CRC en el campo FCS de la trama.

El dispositivo receptor recibe la trama y genera una CRC para detectar errores. Si los cálculos coinciden, significa que no se produjo ningún error. Los cálculos que no coinciden indican que los datos cambiaron y, por consiguiente, se descarta la trama. Un cambio en los datos podría ser resultado de una interrupción de las señales eléctricas que representan los bits.

Fig. 2. Secuencia de verificación de trama Ethernet.

2.9. Compilación de ejemplos NPCAP en c

Lo primero que tenemos que hacer es instalar Npcap en Windows. Lo descargamos de la página oficial, ejecutamos el archivo .exe y nos aparecerá una ventana para aceptar los Términos y Condiciones.

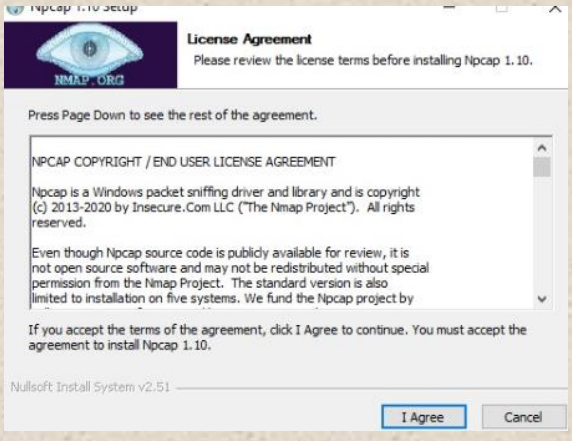


Fig. 3. Términos y Condiciones Npcap.

Aceptamos y en la siguiente ventana (opciones de instalación) habilitamos la opción de compatibilidad con WinPcap.

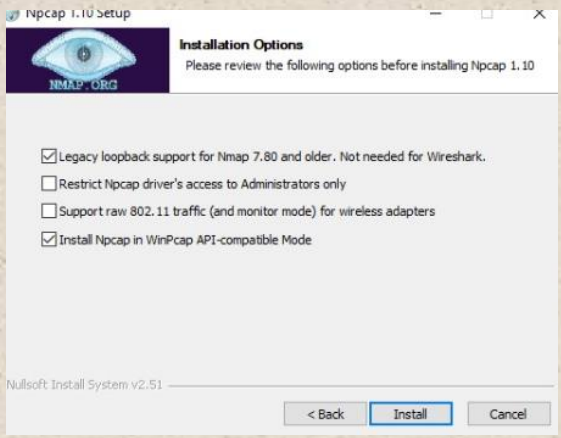
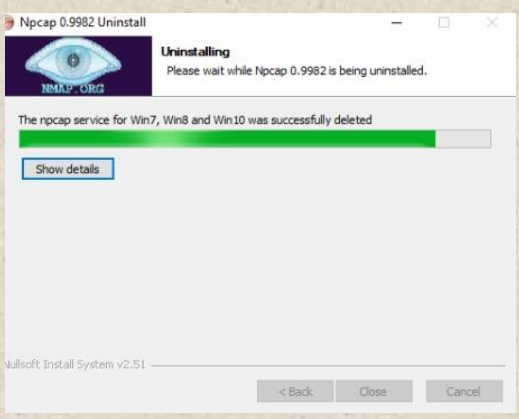
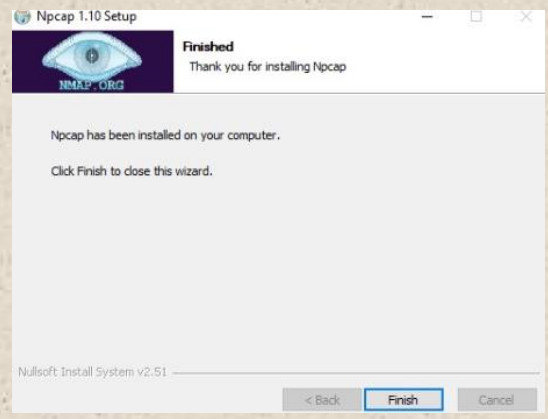


Fig. 4. Opciones de Instalación.

Posteriormente, comenzamos el proceso de instalación al presionar el botón “Install”.





(a) (b)

Fig. 5. Instalación. (a) Inicio, (b) Finalización.

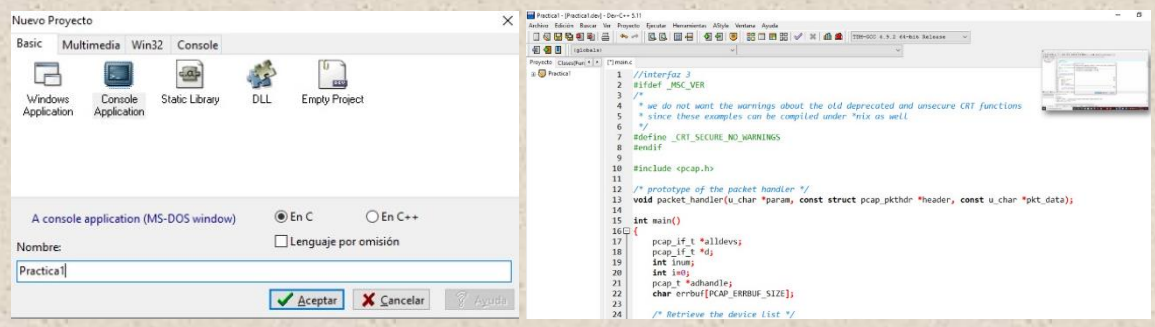
Lo siguiente que debemos hacer es descargar los archivos .zip que se encontrarán en la pantalla mostrada a continuación (npcap-1.10. zip y npcap-sdk-1.06.zip).

Index of /axel/redesnp/sniffer/NPCAP

Name	Last modified	Size	Description
Parent Directory		-	
Compilación de ejemp..>	2021-03-07 18:42	130K	
captura.c	2021-03-08 09:07	3.0K	
envia.c	2021-03-07 18:38	2.9K	
npcap-1.10.exe	2021-01-12 18:21	773K	
npcap-1.10.zip	2021-01-12 18:21	722K	
npcap-sdk-1.06.zip	2021-01-12 18:21	333K	

Fig. 6. Página donde se encuentran los archivos.

Extraemos los archivos de ambas carpetas en direcciones donde podamos acceder fácilmente a ellas. Ahora, abrimos el entorno de desarrollo Dev-C++ y generamos un proyecto, en este caso aplicación desde consola en el lenguaje C, y adicionamos el archivo captura.c (véase el programa en anexos) colocado en la pantalla de la Fig. 6.



(a) (b)

Fig. 7. Proyecto. (a) Generación, (b) Colocación de archivo.

A continuación, abrimos las opciones del proyecto y nos vamos a la pestaña de Compilador, se debe verificar que tenga el que se muestra en la siguiente pantalla.

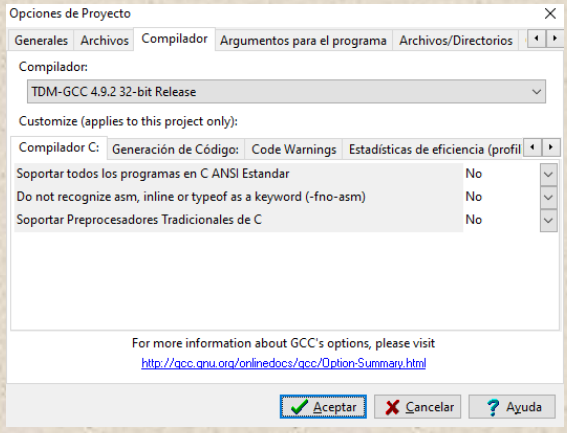


Fig. 8. Opciones de Proyecto – Compilador.

Ahora, tenemos que agregar los parámetros. Así que nos movemos a la pestaña de Argumentos para el programa y enlazar la biblioteca wpcap.lib ubicada en la ruta (npcap/Lib/wpcap.lib).

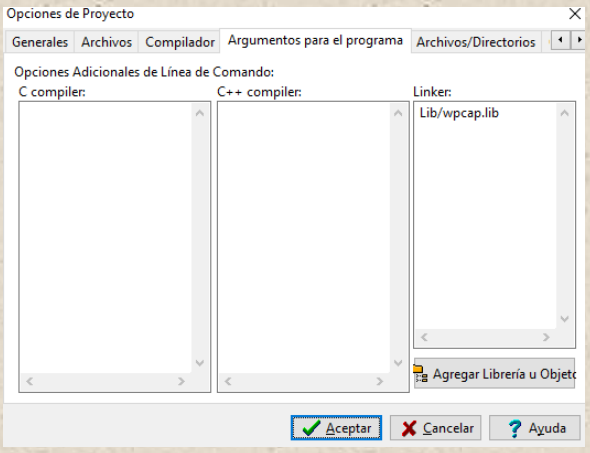


Fig. 9. Opciones de Proyecto – Argumentos para el Programa.

Luego establecemos los directorios de bibliotecas en la pestaña Archivos/Directorios, subpestaña Directorios de Librerías (.a .lib .dll) como se muestra en la Fig. 10.

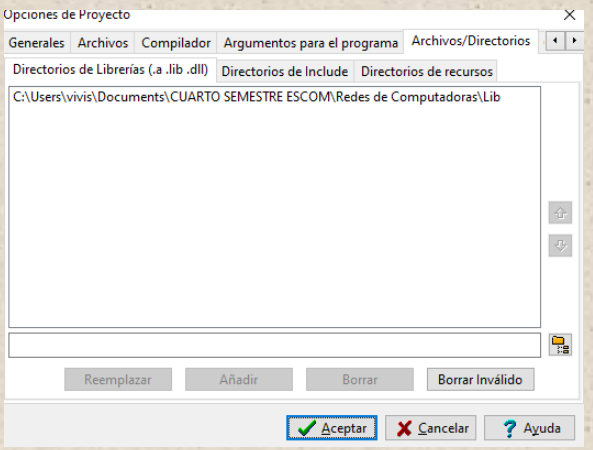


Fig. 10. Directorios de Bibliotecas.

También establecemos los directorios de los archivos include en la subpestaña Directorios de Include.

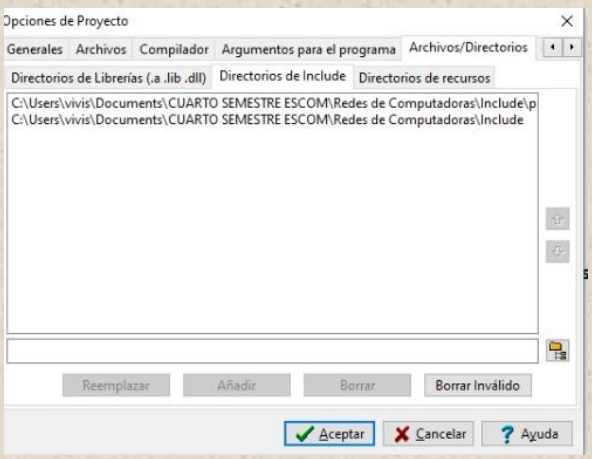


Fig. 11. Directorios de Include.

Seleccionamos la opción de Aceptar.

Ya de regreso en la hoja de trabajo, en la barra superior, podemos apreciar diversos íconos. Daremos al ícono con pequeños cuadros grises, es la opción de reconstruir todo. Después de seleccionar dicho ícono se iniciará el proceso de compilación con las nuevas modificaciones que hicimos en las opciones del proyecto.

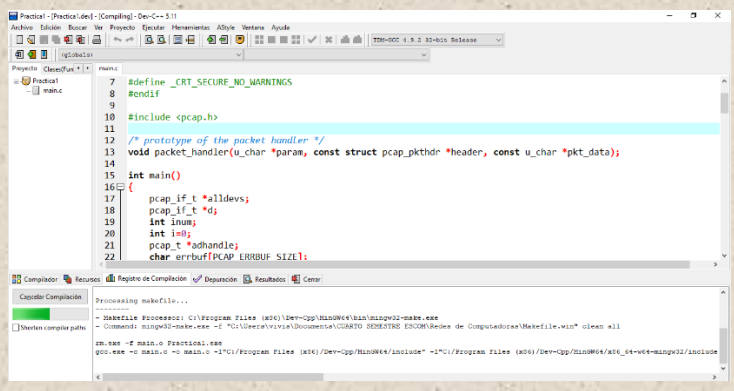


Fig. 12. Compilación.

Funcionamiento del programa

Luego de compilar el programa, si todo salió sin contratiempos nos mostrará una pantalla como la siguiente:

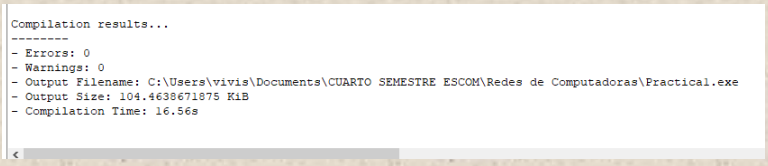


Fig. 13. Resultados de Compilación Exitosa.

Ahora, regresamos a los pequeños íconos de la parte superior, pero ahora seleccionaremos el que tiene 4 cuadros de colores pequeños y sin separación entre sí. El ícono se llama Compilar y Ejecutar. Esto es, como dice su nombre, para que ejecutemos el programa y veamos el resultado del código.

Posteriormente nos aparecerá la consola de Windows con el resultado del programa:

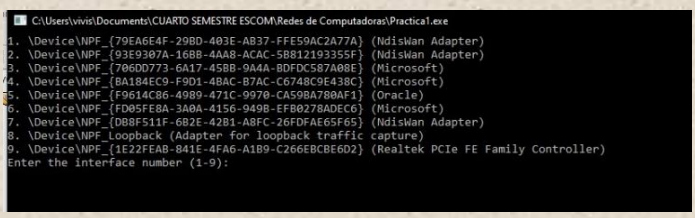
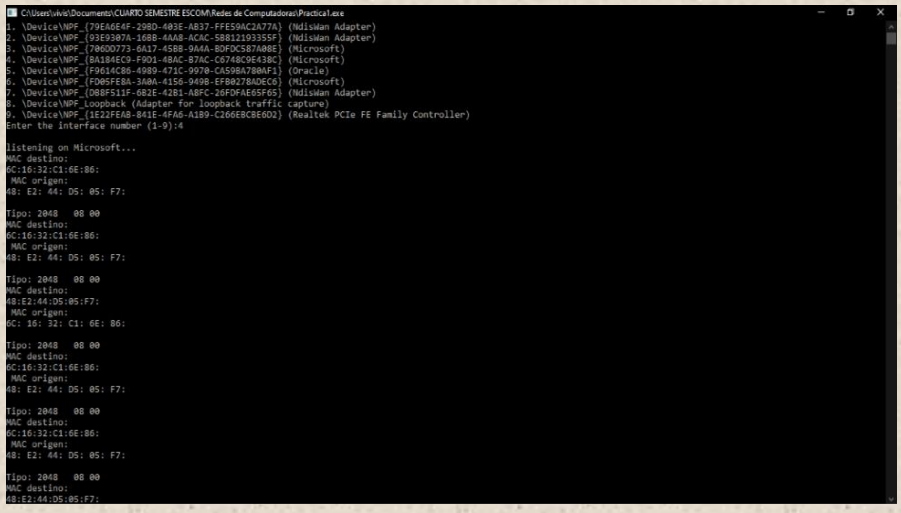


Fig. 14. CMD Windows.

Nos despliega la lista de las tarjetas de red instaladas en nuestro equipo, y se nos preguntará el número de la tarjeta que deseamos usar para realizar el envío de las tramas (véase la figura 14).

Una vez que digitamos el número de la tarjeta de red , el cual en este caso fue el 4 (véase la figura 15 a), que queremos usar se nos desplegará una lista de 15 tramas enviadas y se nos mostrará la trama, la dirección MAC de destino, la dirección MAC de origen, y el campo tipo con su equivalente en número decimal. Con este paso termina la ejecución de nuestro programa (véase la figura 15 b y 15 c).



(a)

(b)

(c)

Fig. 15. Capturas. (a) Parte 1, (b) Parte 2, (c) Parte 3.

2.10. Configuración de biblioteca PCAP4J en NetBeans

1. Instalación de NPCAP en Windows:

Dado que npcap será instalado en Windows\System32\Npcap\ por default, necesitas hacer lo siguiente para que NPCAP pueda ser cargado (véase la figura 21):

- Agrega la variable de entorno jna.library.path a Windows \System32\Npcap\ .
- Agrega la variable de entorno org.pcap4j.core.pcapLibName a Windows \System32\Npcap\wpcap.dll y org.pcap4j.core.packetLibName a Windows \System32\Npcap\Packet.dll .

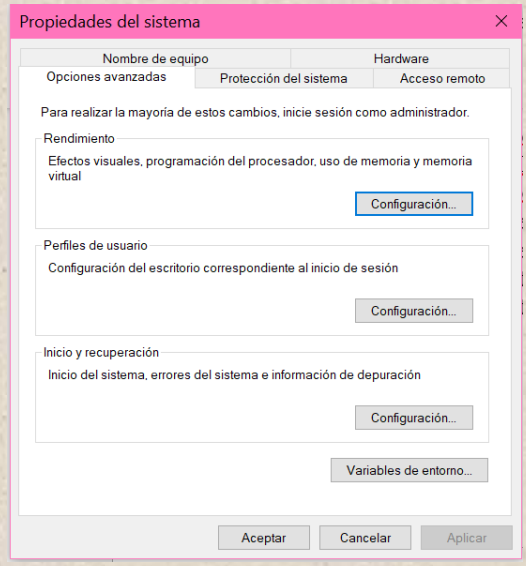


Fig. 16. Propiedades del sistema donde se agregan variables de entorno.

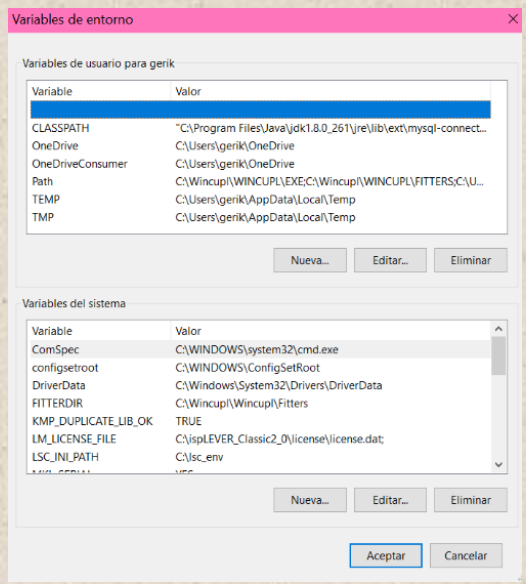


Fig. 17. Variables de entorno.

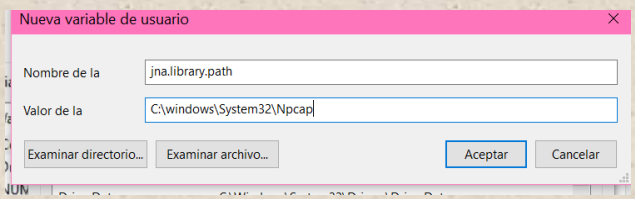


Fig. 18. Adición de la primer variable de entorno jna.library.path.

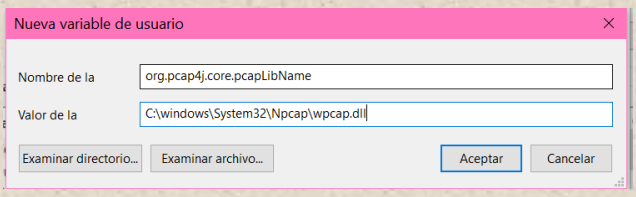


Fig. 19. Adición de la segunda variable de entorno org.pcap4j.core.pcapLibName.

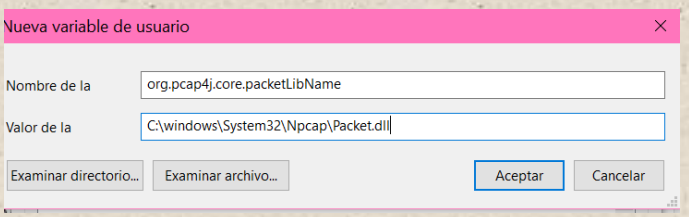


Fig. 20. Adición de la tercera variable de entorno org.pcap4j.core.packetLibName.

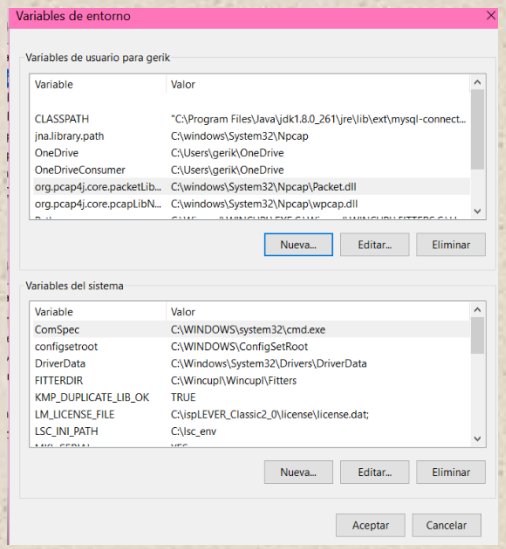


Fig. 21. Todas las variables de entorno ya agregadas.

2. Instala npcap en modo con compatibilidad con WinPcap (habilitar la compatibilidad durante el proceso de instalación) se puede observar en la figura 22.

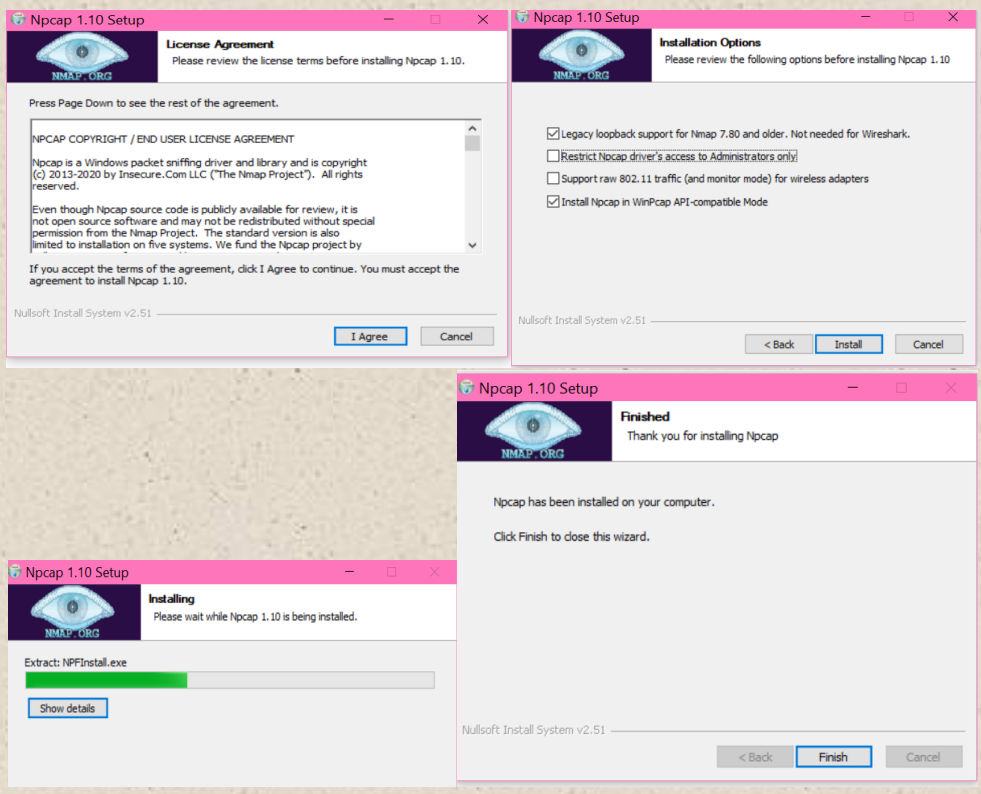


Fig. 22. Instalación de npcap.

3. Se abrió NetBeans IDE y se creó un proyecto nuevo de tipo JAVA WITH MAVEN -> Java Application. Se le puso nombre al proyecto como práctica 1, se dio clic derecho sobre la carpeta "Source Packages" del proyecto recién creado y se eligió la opción New->Java Class. Se creó un nuevo archivo de tipo Java->Java Class y se nombró GetNextRawPacket.java. Se puso el código de java del anexo en el archivo (véase la figura 28).

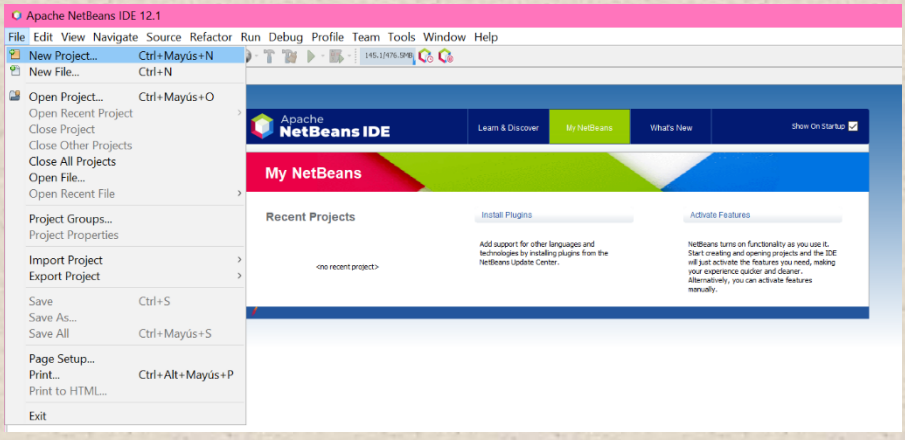


Fig. 23. Creación de nuevo proyecto en NetBeans.

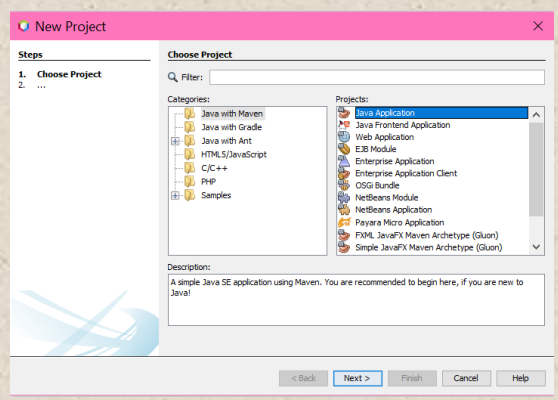


Fig. 24. Selección del archivo Java with Maven y Java Application.

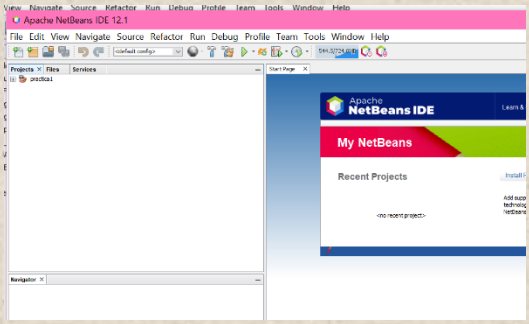


Fig. 25. Nombramiento del proyecto como practica 1.

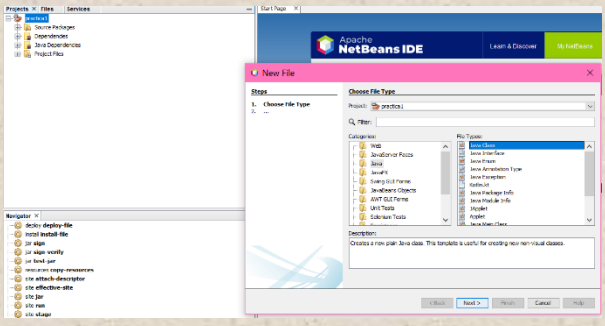


Fig. 26. Creación de la clase de Java.

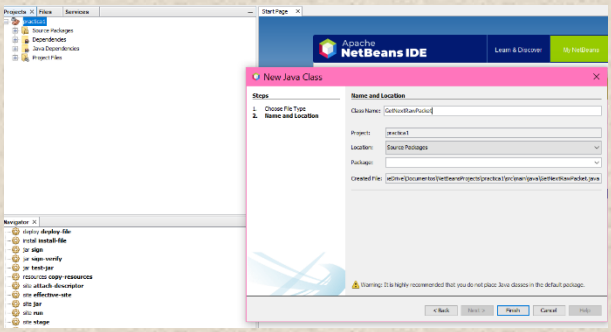


Fig. 27. Nombramiento de la clase como GetNextRawPacket.

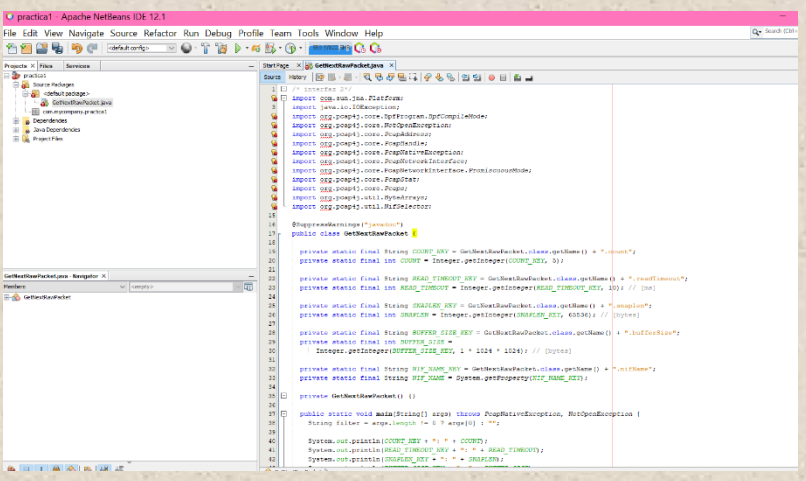


Fig. 28. Poner en el archivo el código.

4. Se añaden las dependencias de la biblioteca pcap4j dando clic derecho en dependencias y seleccionando agregar. Pues en la pestaña de búsqueda de la ventana de dependencias se busca la palabra pcap4j y se agregaron las dependencias encontradas: org.pcap4j-core(1.8.2[jar]central), JNA (net.java.dev.jna), Pcap4j-packetfactory-propertiesbased-1.8.2 y pcap4j-packetfactory-static-1.8.2. (véase la figura 31)

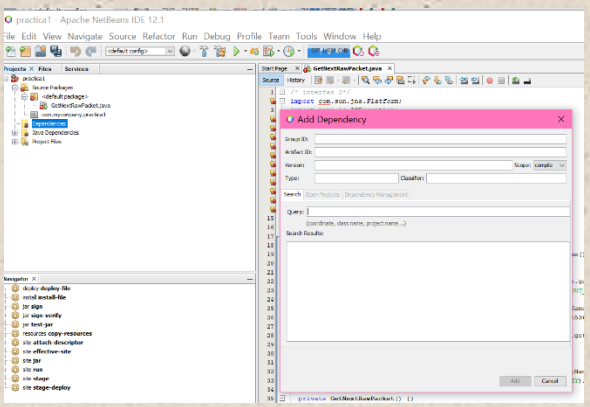


Fig. 29. Adición de dependencias al proyecto.

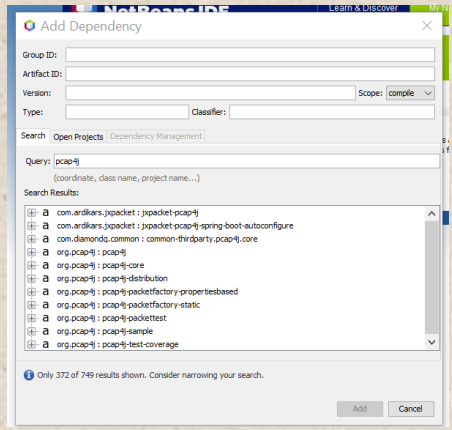


Fig. 30. Búsqueda de las dependencias de la biblioteca pcap4j.

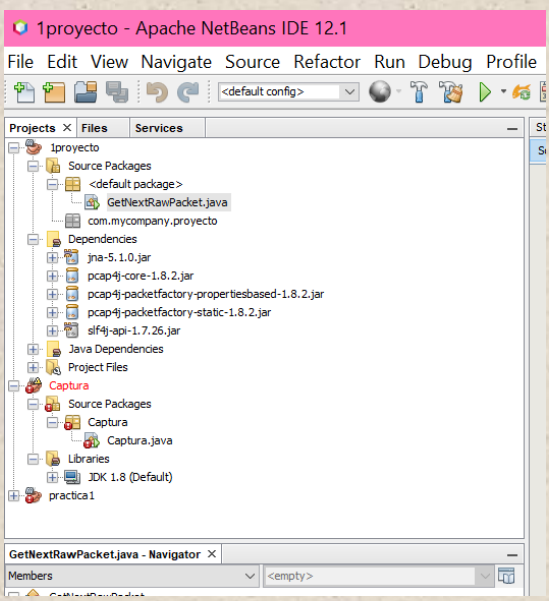


Fig. 31. Todas las dependencias agregadas.

5. Se dio clic derecho sobre el programa GetNextRawPacket.java y se dio clic a la opción del menú contextual “Run” para arrancar el programa y ver el resultado. En la ventana de Salida de NetBeans (Output) se enlistaron las interfaces de red encontradas (véase la figura 33 a) y se eligió la 1 (véase la figura 33 b) pues nos muestra una dirección IP válida (véase la figura 33 c).

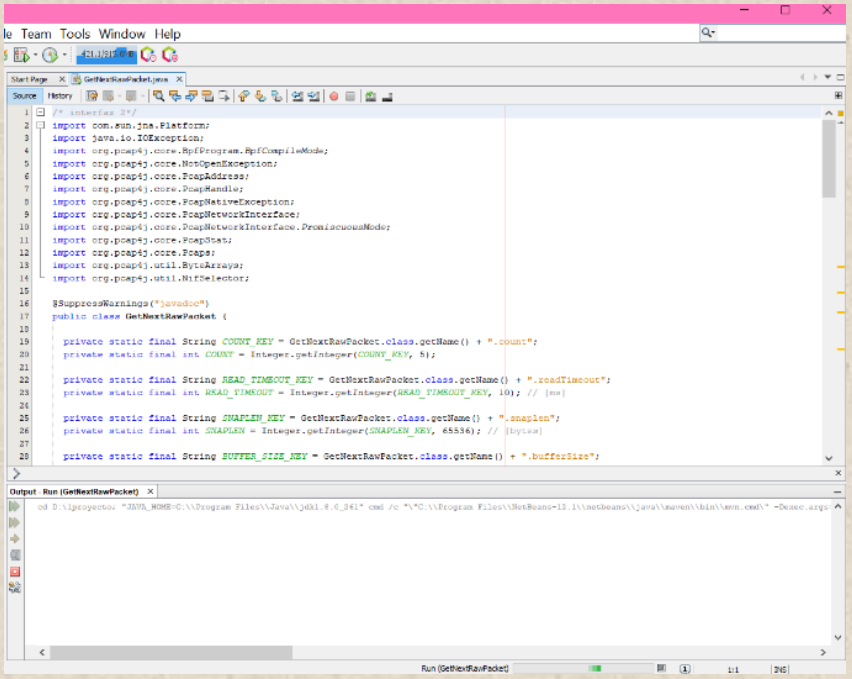
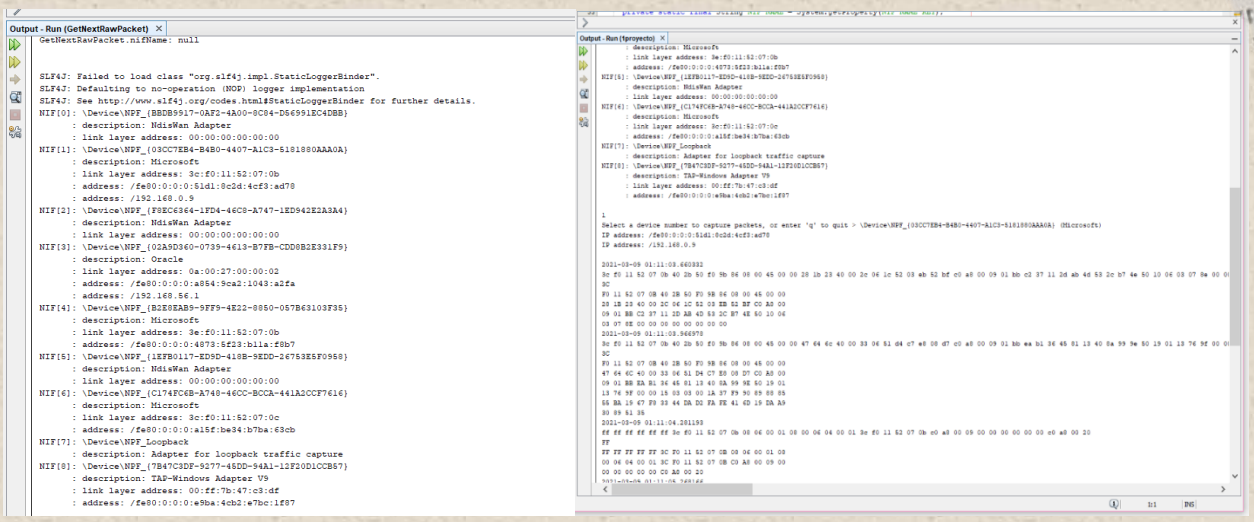
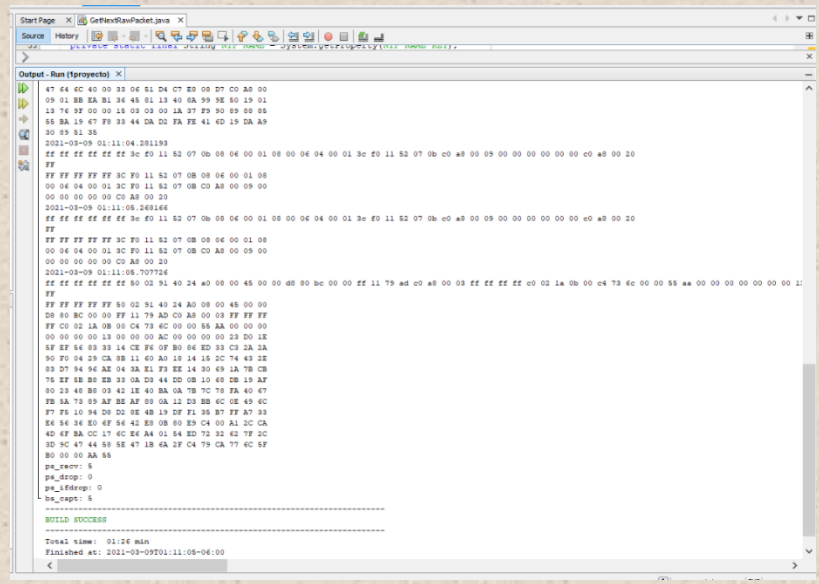


Fig. 32. Ejecución del proyecto.



a) b)



c)

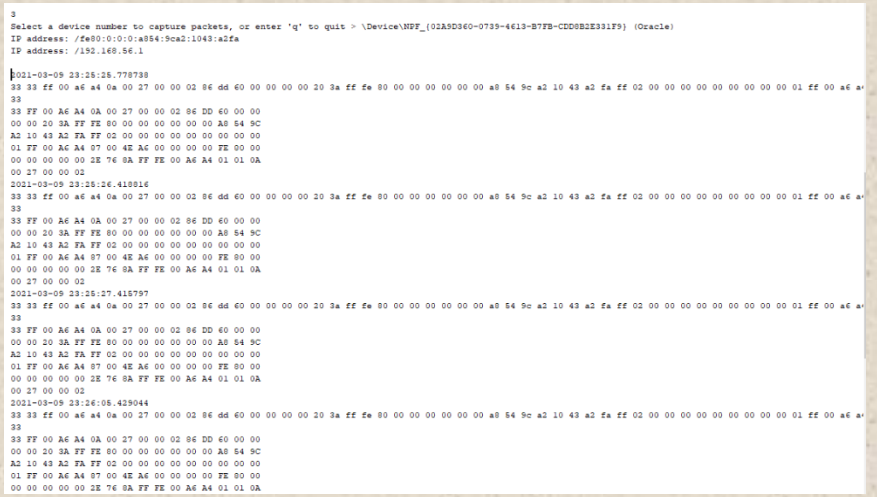
Fig. 33. Resultado de la trama a) y selección de la interfaz de red 1 b) y c) que nos muestra una dirección IP válida.

2.11. Funcionamiento del programa

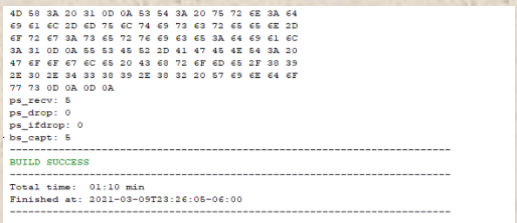
Al darle run al programa previamente escrito, este inmediatamente nos va a mostrar en la ventana de salida de Netbeans las interfaces de red encontradas en el sistema. Como se muestra en la siguiente figura:

Fig. 34. Selección de la interfaz de red 3 que nos muestra una dirección IP válida también.

Inmediatamente nos permitirá escoger una de las 8 enlistadas (en este caso). Se hizo la selección de la 1 ya que nos mostró una dirección IP válida. Aunque también se pudo haber elegido la numero 3. Se observa que son correctas porque tienen el link layer address y ambos address.



a)



b)

Fig. 35. Resultado del programa donde se muestra la trama a) y b).

3. Conclusiones

González Mora Erika Giselle

Se sabe que todos los dispositivos que se encuentran en una red Ethernet se encargan de intercambiar paquetes de datos, estos paquetes reciben el nombre de “tramas Ethernet”, un nombre bastante peculiar.

Gracias a esta práctica se comprendió que estas tramas contienen registros de datos muy valiosos para la comunicación de dispositivos a través de la red Ethernet y que estos registros consisten en un código binario que incluye las direcciones MAC destino, MAC origen, tipo de protocolo, información de control, datos, y sumas de comprobación.

También es importante recalcar que las tramas Ethernet tienen un tamaño que va desde los 64 bytes hasta los 1518 bytes y que, si una trama tiene un tamaño menor a los 64 bytes, o sobrepasa los 1518 bytes, el dispositivo receptor descarta esa trama.

Finalmente puedo decir que esta práctica más que difícil fe tediosa. Sin embargo, valió la pena hacer tantos pasos de instalación.

Olivares Ménez Gloria Oliva

Con esta práctica pude comprender mejor las tramas con los registros de datos para establecer la comunicación entre dispositivos a través de una red, reforzando así lo visto en clase.

El proceso de instalación fue bastante largo y hubo un poco de problemas durante el mismo.

Una vez que terminé de colocar los directorios de las bibliotecas y de los archivos include e intenté compilarlo, me salían ciertos errores en el registro de compilación.

Al principio no entendí lo que pasaba debido a que había seguido al pie de la letra las instrucciones. Sin embargo, me di cuenta después que el error había sido el compilador, no tenía seleccionado el que era correcto.

Con esto me doy cuenta de que absolutamente todos los pasos cuentan, incluidos los que podrían parecer que no tienen gran importancia, así que podré ser cuidadosa en ese aspecto.

En resumen, la práctica reforzó conocimientos y aunque fue un poco confuso el proceso de instalación al principio, al final todo salió bien.

4. Bibliografía

NMAP. (s. f.). *Npcap: WinPcap for Windows 10*. NMAP.org. Recuperado 9 de marzo de 2021, de https://nmap.org/npcap/windows-10.html

Trama de Ethernet - Redes de computadoras. (s. f.). Sites Google. Recuperado 9 de marzo de 2021, de https://sites.google.com/site/sabyrodriguezgamez/unidad-iv/4-2-trama-de-ethernet

5. Anexos

Código c

```
//interfaz 3
#ifdef _MSC_VER
/*
 * we do not want the warnings about the old deprecated and unsecure CRT functions
 * since these examples can be compiled under *nix as well
 */
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <pcap.h>

/* prototype of the packet handler */
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *pkt_data);

int main()
{
	pcap_if_t *alldevs;
	pcap_if_t *d;
	int inum;
	int i=0;
	pcap_t *adhandle;
	char errbuf[PCAP_ERRBUF_SIZE];

	/* Retrieve the device list */
	if(pcap_findalldevs(&alldevs, errbuf) == -1)
	{
		fprintf(stderr,"Error in pcap_findalldevs: %s\n", errbuf);
```

```
        exit(1);
    }

    /* Print the list */
    for(d=alldevs; d; d=d->next)
    {
        printf("%d. %s", ++i, d->name);
        if (d->description)
            printf(" (%s)\n", d->description);
        else
            printf(" (No description available)\n");
    }

    if(i==0)
    {
        printf("\nNo interfaces found! Make sure WinPcap is
installed.\n");
        return -1;
    }

    printf("Enter the interface number (1-%d):",i);
    scanf("%d", &inum);

    if(inum < 1 || inum > i)
    {
        printf("\nInterface number out of range.\n");
        /* Free the device list */
        pcap_freealldevs(alldevs);
        return -1;
    }
```

```
    /* Jump to the selected adapter */
    for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++);

    /* Open the device */
    /* Open the adapter */
    if ((adhandle= pcap_open_live(d->name,    // name of the device
                                  65536,                // portion
of the packet to capture.
                                                        // 65536
grants that the whole packet will be captured on all the MACs.
                                  1,                    //
promiscuous mode (nonzero means promiscuous)
                                  1000,                 // read
timeout
                                  errbuf                // error
buffer
                                  )) == NULL)
    {
        fprintf(stderr,"\nUnable to open the adapter. %s is not supported
by WinPcap\n", d->name);
        /* Free the device list */
        pcap_freealldevs(alldevs);
        return -1;
    }

    printf("\nlistening on %s...\n", d->description);

    /* At this point, we don't need any more the device list. Free it */
    pcap_freealldevs(alldevs);

    /* start the capture */
    pcap_loop(adhandle, 15, packet_handler, NULL);
```

```
	pcap_close(adhandle);

	return 0;

}



/* Callback function invoked by libpcap for every incoming packet */

void packet_handler(u_char *param, const struct pcap_pkthdr *header, const
u_char *pkt_data)

{

	struct tm *ltime;

	char timestr[16];

	time_t local_tv_sec;


	/*

	 * unused parameters

	 */

	(VOID)(param);

	(VOID)(pkt_data);


	/* convert the timestamp to readable format */

	local_tv_sec = header->ts.tv_sec;

	//ltime=localtime(&local_tv_sec);

	//strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);


	//printf("%s,%.6d len:%d\n", timestr, header->ts.tv_usec, header->len);

	int j=0,k=0;

	printf("MAC destino:\n");

	for(j=0;j<6;j++){

	   printf("%02X:",pkt_data[j]);

	}

	printf("\n MAC origen:\n");
```

```
      for(k=6;k<12;k++){
         printf("%02X: ",pkt_data[k]);
      }

      printf(" \n\n");
    unsigned short tipo = (pkt_data[12]*256)+pkt_data[13];
    printf("Tipo: %d   %02X %02X \n",tipo,pkt_data[12],pkt_data[13]);
}
```

Código Java

```
/* interfaz 2*/
import com.sun.jna.Platform;
import java.io.IOException;
import org.pcap4j.core.BpfProgram.BpfCompileMode;
import org.pcap4j.core.NotOpenException;
import org.pcap4j.core.PcapAddress;
import org.pcap4j.core.PcapHandle;
import org.pcap4j.core.PcapNativeException;
import org.pcap4j.core.PcapNetworkInterface;
import org.pcap4j.core.PcapNetworkInterface.PromiscuousMode;
import org.pcap4j.core.PcapStat;
import org.pcap4j.core.Pcaps;
import org.pcap4j.util.ByteArrays;
import org.pcap4j.util.NifSelector;

@SuppressWarnings("javadoc")
public class GetNextRawPacket {
  private static final String COUNT_KEY = GetNextRawPacket.class.getName() +
".count";
  private static final int COUNT = Integer.getInteger(COUNT_KEY, 5);
```

```
  private static final String READ_TIMEOUT_KEY =
GetNextRawPacket.class.getName() + ".readTimeout";

  private static final int READ_TIMEOUT =
Integer.getInteger(READ_TIMEOUT_KEY, 10); // [ms]

  private static final String SNAPLEN_KEY = GetNextRawPacket.class.getName()
+ ".snaplen";

  private static final int SNAPLEN = Integer.getInteger(SNAPLEN_KEY, 65536);
// [bytes]

  private static final String BUFFER_SIZE_KEY =
GetNextRawPacket.class.getName() + ".bufferSize";

  private static final int BUFFER_SIZE =

      Integer.getInteger(BUFFER_SIZE_KEY, 1 * 1024 * 1024); // [bytes]

  private static final String NIF_NAME_KEY = GetNextRawPacket.class.getName()
+ ".nifName";

  private static final String NIF_NAME = System.getProperty(NIF_NAME_KEY);

  private GetNextRawPacket() {}

  public static void main(String[] args) throws PcapNativeException,
NotOpenException {

    String filter = args.length != 0 ? args[0] : "";

    System.out.println(COUNT_KEY + ": " + COUNT);

    System.out.println(READ_TIMEOUT_KEY + ": " + READ_TIMEOUT);

    System.out.println(SNAPLEN_KEY + ": " + SNAPLEN);

    System.out.println(BUFFER_SIZE_KEY + ": " + BUFFER_SIZE);

    System.out.println(NIF_NAME_KEY + ": " + NIF_NAME);

    System.out.println("\n");


    PcapNetworkInterface nif;

    if (NIF_NAME != null) {

      nif = Pcaps.getDevByName(NIF_NAME);

    } else {

      try {

        nif = new NifSelector().selectNetworkInterface();

      } catch (IOException e) {

        e.printStackTrace();
```

```
    return;
  }

  if (nif == null) {
    return;
  }
}

System.out.println(nif.getName() + " (" + nif.getDescription() + ")");
for (PcapAddress addr : nif.getAddresses()) {
  if (addr.getAddress() != null) {
    System.out.println("IP address: " + addr.getAddress());
  }
}
System.out.println("");

PcapHandle handle =
    new PcapHandle.Builder(nif.getName())
        .snaplen(SNAPLEN)
        .promiscuousMode(PromiscuousMode.PROMISCUOUS)
        .timeoutMillis(READ_TIMEOUT)
        .bufferSize(BUFFER_SIZE)
        .build();

handle.setFilter(filter, BpfCompileMode.OPTIMIZE);

int num = 0;
while (true) {
  byte[] packet = handle.getNextRawPacket();
  if (packet == null) {
    continue;
```

```
      } else {
        System.out.println(handle.getTimestamp());
        System.out.println(ByteArrays.toHexString(packet, " "));
        for(int j=0;j<packet.length;j++){
            System.out.printf("%02X ",packet[j]);
            if(j%16==0)
                System.out.println("");
        }//for
          System.out.println("");
        num++;
        if (num >= COUNT) {
          break;
        }



      }
    }
    PcapStat ps = handle.getStats();
    System.out.println("ps_recv: " + ps.getNumPacketsReceived());
    System.out.println("ps_drop: " + ps.getNumPacketsDropped());
    System.out.println("ps_ifdrop: " + ps.getNumPacketsDroppedByIf());
    if (Platform.isWindows()) {
      System.out.println("bs_capt: " + ps.getNumPacketsCaptured());
    }
    handle.close();
  }
}
```