

INSTITUTO POLITÉCNICO NACIONAL
ESCOM

POLAROID

REDES

PRÁCTICA 9

"PROTOCOLO ICMP"

INTEGRANTES:

GONZÁLEZ MORA ERIKA GISELE

OLIVARES MENÉZ GLORIA OLIVA

GRUPO: ZCV16

...the
did before
overseeing
makers. I
w; but, like
ood, I have a
things together
ad been studying
extiles, draperies, hair arrange-
and I spent hours over the cos-
s, the wigs, and all the other details."
ou speak of studying these material
s," I said; "do you study women
in order to understand how they
act under certain circumstances?"
ever!" was the emphatic reply. "I
have to study them. I know. I
explain it, but if you should talk to
f some woman I have never
tell you what she is like a
ing she would do. It is
standing."
and you have the
ing of a v
h, there

...do you do when you are not
vamping?" I asked.
Oh—I read, and take the dogs for a
walk.

"I don't do anything very exciting,"
she added apologetically. "I don't go in
for sports. I'm not in the least athletic.
In fact, I'm afraid I am a physical coward.
I dread one of those struggles I told you
about. I don't like to be hurt. After
have been battered and bruised that
my doctor says to me:

"Well! if love meant
to the average woman
most unpopular

"And yet pe
gles when th
seats of
thrill, m
likes

...thing
where
ragedly that
vest." She w
name used
that she was l
her name or
an came from another
would not discuss the freak st
which she has been the subject.
"What difference does it make
said, with a shrug of her
may be what the press agents call
publicity" if people wonder what a
Frankenstein I am. If I under
contradict all the lies that are
about me, I shouldn't have
thing else."

MENTAL she
nating. She tal
gence of books. S
tive ideas about
Yet the play is
a dramatic
the
the
sh

ÍNDICE

1.	INTRODUCCIÓN.....	3
1.1.	¿QUÉ ES EL ICMP?.....	3
1.1.1.	ESTRUCTURA	3
1.1.2.	TIPOS DE PAQUETES.....	4
1.1.3.	UTILIZACIÓN	5
2.	DESARROLLO.....	6
3.	CONCLUSIONES.....	18
4.	REFERENCIAS.....	19

1. Introducción

1.1. ¿Qué es el ICMP?

Internet Control Message Protocol, por sus siglas en inglés, es utilizado para enviar mensajes de error e información operativa indicando, por ejemplo, que un host no puede ser localizado o que un servicio que se ha solicitado no está disponible.

Concretamente, los servidores de aplicaciones y las puertas de acceso como los routers, utilizan esta implementación del protocolo IP para devolver mensajes sobre problemas con datagramas al remitente del paquete, es decir, que estos mensajes de error se envían a la dirección IP de origen del paquete.

A pesar de que los mensajes están incluidos en paquetes IP tradicionales, el ICMP es un protocolo autónomo.

Los diversos servicios de red que se suelen utilizar hoy en día, como traceroute o ping, se basan en este protocolo.

1.1.1. Estructura

La cabecera del Internet Control Message Protocol está vinculada directamente con la del IP, aunque se muestra en el campo "Protocol" de IP mediante la inserción del número de protocolo, que puede ser 1 o 58 (ICMPv6).

Tiene la siguiente forma:

	Bit 0–7	Bit 8–15	Bit 16–23	Bit 24–31
0	Tipo	Código	Suma de verificación	
32	Datos sobre la cabecera			

El primer campo de 8 bits "Tipo" determina el tipo de mensaje al que hace referencia el paquete ICMP correspondiente.

El siguiente campo puede tomar los valores 0, 1 o 3. El primero representa la red de destino, el segundo el host deseado y el tercero el puerto esperado. Precisa cual de ellos tres no a respondido la solicitud.

Por otro lado, la suma de verificación de ICMP garantiza la exactitud del mensaje. Esta se forma de igual manera que la suma de verificación de otros protocolos estándar (IP, UDP, TCP).

Finalmente aparecen los datos del protocolo, que se crean y estructuran de manera muy diferente en función del tipo y de la instancia desencadenante. A menudo también se especifican aquí la

cabecera IP y los primeros 64 bits del paquete de datos que es responsable del mensaje de error o de la solicitud de estado.

El llamado tunneling de ICMP puede enviar datos de usuario bajo el radar de los cortafuegos o para establecer un canal de comunicación entre dos ordenadores.

1.1.2. Tipos de Paquetes

A continuación, se presentan los paquetes más importantes basados en el Internet Control Message Protocol:

<i>Tipo ICMP</i>	<i>Tipo ICMPv6</i>	<i>Nombre tipo</i>	<i>Código</i>	<i>Descripción</i>
	129	Echo Reply		Respuesta a un ping de red para comprobar la accesibilidad
3	1	Destination Unreachable	0-15	Mensaje ICMP que informa acerca de, por ejemplo, la accesibilidad de red de los componentes del campo "Código" (red, protocolo, puerto, host), sobre problemas de enrutamiento o sobre el bloqueo por parte de los cortafuegos
5	137	Redirect Message	0-3	Mensaje sobre el redireccionamiento de un paquete para la red indicada (0), para el host escogido (1), para el servicio especificado y para la red (2) o para el servicio y host especificados (3)
8	128	Echo Request		Ping de red
9	134	Router Advertisement		Lo utilizan los routers para informarse acerca de los diferentes clientes de red
11	3	Time Exceeded	0 o 1	Informe de estado que o bien indica que el tiempo de vida (Time to Live,

				TTL) de un paquete (0) o el tiempo de espera para el ensamblaje de paquetes IP (1) ha expirado
13	13	Timestamp		Dota al paquete IP de una marca de tiempo que se corresponde con el momento del envío y que es de utilidad para la sincronización de dos ordenadores
14	-	Timestamp Reply		Mensaje de respuesta a una petición de marca de tiempo enviado por el destinatario tras la recepción de esta
30	-	Traceroute		Tipo de mensaje ICMP obsoleto que se utilizaba para el seguimiento de la ruta de un paquete de datos en la red. Hoy en día se utilizan "Echo Request" y "Echo Reply" para estos fines

En general, podría decirse que en este protocolo hay 2 tipos de mensajes: mensajes informativos y mensaje destino inaccesible.

1.1.3. Utilización

El protocolo ICMP es esencial para la comunicación en redes IP. Es usado por los routers, los servidores y los clientes. Reciben información de red importante en forma de mensajes, los cuales están vinculados al protocolo de Internet.

Otro de los escenarios de aplicación más extendidos es el llamado ping de red, que puede ejecutarse con ayuda de aplicaciones del mismo nombre a través de las líneas de comandos del sistema operativo correspondiente.

Esta herramienta de diagnóstico es la solución más sencilla para comprobar la accesibilidad de un determinado host en la red. Para ello, el ping envía, por un lado, un paquete IP incluida una "Echo Request" ICMP(6) (tipo 8 o 128), al que, tras su recepción, el

receptor responderá con un paquete de datos que contiene la entrada ICMP "Echo Reply" (tipo 0 o 129).

Si no se localiza al sistema al que se ha enviado el ping, la última estación de red disponible enviará un paquete de respuesta, el cual se amplía con un componente ICMP, es decir, tipo 3 o 1 "Destination Unreachable" ("objetivo inalcanzable").

Los routers utilizan el protocolo ICMP para diversos fines. Por ejemplo, con el tipo "Router Advertisement" (tipo 9 de ICMP; tipo 134 de ICMPv6) pueden informar periódicamente a todos los participantes de red activos sobre su presencia y sobre diferentes datos de red.

Se guardan los datos recibidos en el caché y hacen que el router se convierta en la puerta de acceso estándar.

Asimismo, los routers intentan optimizar la ruta de los paquetes de datos en la red por medio del tipo "Redirect" de ICMP (tipo 5 o 137). Con ayuda de este tipo de mensajes, las interfaces de red son capaces de advertir a los hosts sobre la existencia de un hop (conexión intermedia) de mejor calidad para el envío de paquetes IP.

2. Desarrollo

Al ser el objetivo de esta práctica la mejor comprensión del protocolo ICMP, se realizó un código en Lenguaje C que, primero, generaría tramas IP y, posteriormente, analizaría cada una de ellas diciendo qué protocolo estaban siguiendo.

El programa principal se muestra a continuación.

```
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif
#include <pcap.h>
#include "IP.h"

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;
    int j = 0, k = 0;
    unsigned short tipo = (pkt_data[12] << 8) + pkt_data[13];
    ip_head *ip;

    switch (tipo)
    {
        case 2048:
            ip = (ip_head *) (pkt_data + 14);
            printf("-----T R A M A   I P   E N C O
N T R A D A-----\n");
            printf("-----\n");
            printf("-----\n");
            printf("-----\n");
            printf("-----\n");
    }
}
```



```

        imprimirTrama(header, pkt_data);
        printf("\n\n");
        analizarIp(ip, pkt_data);
        break;

    case 2054:
        break;

    default:
        break;
}

}

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

    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, 100, packet_handler, NULL);
pcap_close(adhandle);
return 0;
}

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

```

Como podemos observar en la cabecera del programa se utiliza una librería llamada "IP.h", la cual contiene las estructuras para la generación de tramas IP en el programa principal y el análisis de estas, haciendo énfasis en el ICMP.

```

#define LINE_LEN 50

/* 4 bytes IP address */
typedef struct ip_add
{
    u_char byte1;
    u_char byte2;
    u_char byte3;
    u_char byte4;
} ip_add;

/* IPv4 header */
typedef struct ip_head
{
    u_char ver_ihl; // Version (4 bits) + IP header length (4 bits)
    u_char tos; // Type of service
    u_short tlen; // Total length
    u_short identification; // Identification
    u_short flags_fo; // Flags (3 bits) + Fragment offset (13 bits)
    u_char ttl; // Time to live
    u_char proto; // Protocol
    u_short crc; // Header checksum
    ip_add saddr; // Source address
    ip_add daddr; // Destination address
    u_int op_pad; // Option + Padding
} ip_head;

/* UCP header*/
typedef struct udp_head
{
    u_short sport; // Source port
    u_short dport; // Destination port
    u_short len; // length
    u_short crc; // Checksum
} udp_head;

```



```

/* TCP header*/
typedef struct tcp_head
{
    u_short sport; // Source port
    u_short dport; // Destination port
    u_int sec_num; // sequence number
    u_int ack_num; // ack number
    u_char d_offset_rsv; // 4bit data offset +4bit reserved
    u_char flags; // TCP flags
    u_short window; // window
    u_short crc; // Checksum
    u_short upointer; // urgent pointer
} tcp_head;

/* ICMP header*/
typedef struct icmp_head
{
    u_char type; // ICMP type
    u_char code; // ICMP code
    u_short crc; // Checksum
} icmp_head;

void convertirNumeroBinario(u_char numero, u_char cantidad_bits)
{
    int corrimiento = 0;
    for(corrimiento = cantidad_bits - 1; corrimiento >= 0; corrimiento--)
        printf("%u", (numero >> corrimiento) & 0x01);
}

u_short invertir_bytes(u_short numero)
{
    return ((numero & 0xff) << 8) + (numero >> 8);
}

void imprimirSelectorClase(u_char tos)
{
    u_char tipo = (tos >> 5) & 0x3;
    convertirNumeroBinario(tipo, 3);

    switch(tipo)
    {
        case 0:
            printf(" Routine (default)\n");
            break;

        case 1:
            printf(" Priority (Trafico de datos)\n");
            break;

        case 2:
            printf(" Immediate (Trafico de datos)\n");
            break;

        case 3:
            printf(" Flash (Call Signaling)\n");
            break;

        case 4:
            printf(" Flash Override (Vcon., streaming)\n");
            break;

        case 5:
            printf(" CRITIC/ECP (Voz)\n");
            break;

        case 6:
    }
}

```

```

        printf(" Internetwork Control (Trafico de control)\n");
        break;

        case 7:
        printf(" Network Control (Trafico de control)\n");
        break;

        default:
        break;
    }
}

void imprimirECN(u_char tos)
{
    int tipo = tos & 0x3;
    convertirNumeroBinario(tipo, 2);

    switch(tipo)
    {
        case 0:
        printf(" Sin capacidad ECN\n");
        break;

        case 1:
        printf(" Capacidad de transporte ECN (0)\n");
        break;

        case 2:
        printf(" Capacidad de transporte ECN (1)\n");
        break;

        case 3:
        printf(" Congestion encontrada\n");
        break;
    }
}

void imprimirFlags(u_short flags)
{
    u_char tipo = (flags >> 13) & 0x3;
    convertirNumeroBinario(tipo, 3);

    switch(tipo)
    {
        case 0:
        printf(" Last fragment");

        case 1:
        printf(" More fragments");
        break;

        case 2:
        printf(" Don't fragment");
        break;
    }
    printf("\n");
}

void imprimirIP(ip_add ip)
{
    printf("%u.%u.%u.%u\n", ip.byte1, ip.byte2, ip.byte3, ip.byte4);
}

void analizarUDP(udp_head *udp)
{

```

```

        printf("-->Source port: %02X\n", invertir_bytes(udp -> sport));
        printf("-->Destination port: %02X\n", invertir_bytes(udp -> dport));
        printf("-->Length: %u\n", invertir_bytes(udp -> len));
        printf("-->Checksum: %02X\n", invertir_bytes(udp -> crc));
    }

void imprimirFlagTCP(u_char flag)
{
    if (flag & 1)
        printf(" FIN");
    if (flag & 2)
        printf(" SYN");
    if (flag & 4)
        printf(" RST");
    if (flag & 8)
        printf(" PSH");
    if (flag & 16)
        printf(" ACK");
    if (flag & 32)
        printf(" URG");
    if (flag & 64)
        printf(" ECE");
    if (flag & 128)
        printf(" CWR");
    printf("\n");
}

void analizarTCP(tcp_head *tcp)
{
    printf("-->Source port: %02X\n", invertir_bytes(tcp -> sport));
    printf("-->Destination port: %02X\n", invertir_bytes(tcp -> dport));
    printf("-->Secuence number: %u\n", tcp -> sec_num);
    printf("-->Ack number: %u\n", tcp -> ack_num);
    printf("-->Offset: %u\n", (tcp -> d_offset_rsv) >> 4);
    printf("-->Reserved: %u\n", (tcp -> d_offset_rsv) & 0x0f);
    printf("-->Flags: ");
    convertirNumeroBinario(tcp -> flags, 8);
    imprimirFlagTCP(tcp -> flags);
    printf("-->Windows size: %u\n", invertir_bytes(tcp -> window));
    printf("-->Checksum: %02X\n", invertir_bytes(tcp -> crc));
    printf("-->Urgent pointer: %02X\n", invertir_bytes(tcp -> upointer));
}

imprimirTipo(u_char tipo)
{
    switch(tipo)
    {
        case 0:
            printf("Echo Reply (0)\n");
            break;

        case 3:
            printf("Destination Unreachable (3)\n");
            break;

        case 5:
            printf("Redirect Message (5)\n");
            break;

        case 8:
            printf("Echo Request (8)\n");
            break;

        case 9:
            printf("Router Advertisement (9)\n");

```



```

        break;

    case 10:
        printf("Router Solicitation (10)\n");
        break;

    case 11:
        printf("Time Exceeded (11)\n");
        break;

    case 12:
        printf("Parameter Problem (12)\n");
        break;

    case 13:
        printf("Timestamp (13)\n");
        break;

    case 14:
        printf("Timestamp Reply (14)\n");
        break;
    }
}

void codigoEchoReply(u_char codigo)
{
    switch(codigo)
    {
        case 0:
            printf("Echo Reply (0)\n");
            break;
    }
}

void codigoEchoRequest(u_char codigo)
{
    switch(codigo)
    {
        case 0:
            printf("Echo request (0)\n");
            break;
    }
}

void codigoDestUn(u_char codigo)
{
    switch(codigo)
    {
        case 0:
            printf("Destination network unreachable (0)\n");
            break;

        case 1:
            printf("Destination host unreachable (1)\n");
            break;

        case 2:
            printf("Destination protocol unreachable (2)\n");
            break;

        case 3:
            printf("Destination protocol port (3)\n");
            break;

        case 4:
            printf("Fragmentation needed and DF flag set (4)\n");
            break;
    }
}

```

```

        case 5:
            printf("Source route failed (5)\n");
            break;
    }
}

void codigoRedMes(u_char codigo)
{
    switch(codigo)
    {
        case 0:
            printf("Redirect datagram for the network (0)\n");
            break;

        case 1:
            printf("Redirect datagram for the host (1)\n");
            break;

        case 2:
            printf("Redirect datagram for the type of service and network (2)\n");
            break;

        case 3:
            printf("Redirect datagram for the service and host (3)\n");
            break;
    }
}

void codigoRouterAdv_Sol(u_char codigo)
{
    switch(codigo)
    {
        case 0:
            printf("Use to discover the addresses of operational routers (0)\n");
            break;
    }
}

void codigoTimeEx(u_char codigo)
{
    switch(codigo)
    {
        case 0:
            printf("Time to live exceeded in transit (0)\n");
            break;

        case 1:
            printf("Fragment reassembly time exceeded (1)\n");
            break;
    }
}

void codigoParPro(u_char codigo)
{
    switch(codigo)
    {
        case 0:
            printf("Pointer indicates error (0)\n");
            break;

        case 1:
            printf("Missing required option (1)\n");
            break;

        case 2:

```

```

        printf("Bad length (2)\n");
        break;
    }
}

void codigoTimestamp(u_char codigo)
{
    switch(codigo)
    {
        case 0:
            printf("Used for time synchronization (0)\n");
            break;
    }
}

void codigoTimestampRep(u_char codigo)
{
    switch(codigo)
    {
        case 0:
            printf("Reply to timestamp message (0)\n");
            break;
    }
}

void imprimirCodigo(u_char tipo, u_char codigo)
{
    switch(tipo)
    {
        case 0:
            codigoEchoReply(codigo);
            break;

        case 3:
            codigoDestUn(codigo);
            break;

        case 5:
            codigoRedMes(codigo);
            break;

        case 8:
            codigoEchoRequest(codigo);
            break;

        case 9:
            codigoRouterAdv_Sol(codigo);
            break;

        case 10:
            codigoRouterAdv_Sol(codigo);
            break;

        case 11:
            codigoTimeEx(codigo);
            break;

        case 12:
            codigoParPro(codigo);
            break;

        case 13:
            codigoTimestamp(codigo);
            break;

        case 14:

```



```

        codigoTimestampRep(codigo);
        break;
    }
}

void analizarICMP(icmp_head *icmp)
{
    printf("-->Tipo: ");
    imprimirTipo(icmp -> type);
    printf("-->Codigo: ");
    imprimirCodigo(icmp -> type, icmp -> code);
    printf("-->Cheksum: %04X\n", invertir_bytes(icmp -> crc));
}

void analizarProtocolo(u_char protocolo, const u_char *pkt_data, u_char ihl)
{
    switch (protocolo)
    {
        case 1:
            printf("---> ICMP (1)\n");
            analizarICMP((icmp_head *) (pkt_data + 14 + ihl));
            break;

        case 6:
            printf(" TCP (6)\n");
            analizarTCP((tcp_head *) (pkt_data + 14 + ihl));
            break;

        case 17:
            printf(" UDP (17)\n");
            analizarUDP((udp_head *) (pkt_data + 14 + ihl));
            break;
    }
}

void analizarIp(ip_head *ip, const u_char *pkt_data)
{
    u_short flas_flo_i = invertir_bytes(ip -> flags_fo);
    u_char version = (ip -> ver_ihl) >> 4;
    u_char ihl = ((ip -> ver_ihl) & 15) * 4;

    printf("Version: %u\n", version);
    printf("IP Header Length: %d bytes (%u)\n", ihl, ((ip -> ver_ihl) & 15));
    printf("Class selector: ");
    imprimirSelectorClase(ip -> tos);
    printf("ECN: ");
    imprimirECN(ip -> tos);
    printf("Total length: %u\n", invertir_bytes(ip -> tlen));
    printf("Identification: %u\n", invertir_bytes(ip -> identification));
    printf("Flags: ");
    imprimirFlags(flas_flo_i);
    printf("Fragment offset: %u\n", flas_flo_i & 8191);
    printf("TTL: %u\n", ip -> ttl);

    printf("Protocolo:");
    analizarProtocolo(ip -> proto, pkt_data, ihl);

    printf("Checksum: %02X\n", invertir_bytes(ip -> crc));
    printf("Source IP address: ");
    imprimirIP(ip -> saddr);
    printf("Destination IP address: ");
    imprimirIP(ip -> daddr);
    printf("Options: %u\n\n", ip -> op_pad);

    if(ip -> proto == 1)

```

```

        system("pause");
    }

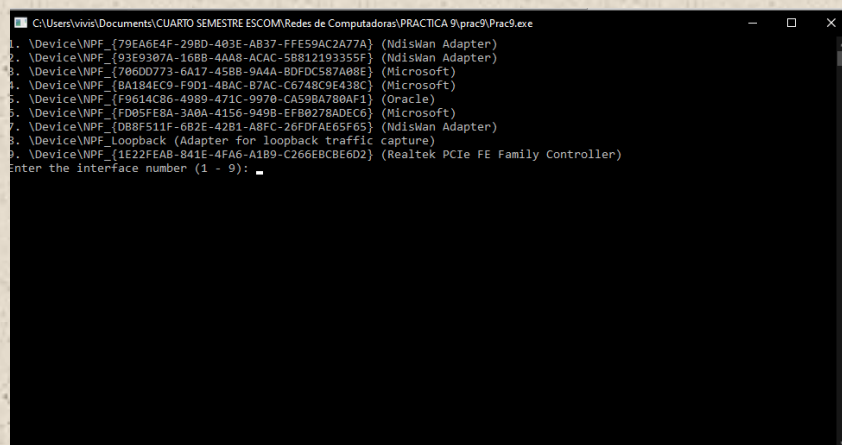
void imprimirTrama(const struct pcap_pkthdr *header, const u_char *pkt_data)
{
    int i;

    for (i = 1; (i < header->caplen + 1); i++)
    {
        printf("%.2x ", pkt_data[i - 1]);

        if ( (i % LINE_LEN) == 0)
            printf("\n");
    }
}

```

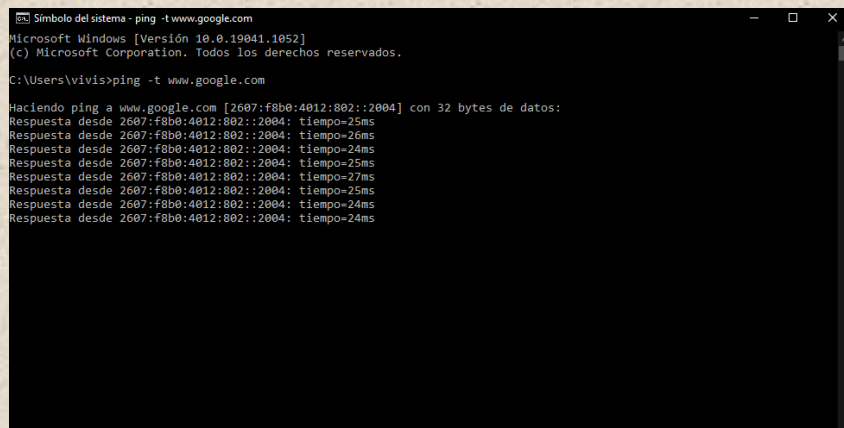
Al momento de compilarlo y ejecutarlo, aparece la siguiente pantalla:



Esto es para elegir la interfaz de donde se generarán las tramas.

Mientras el programa está en ejecución, debe abrirse otro Command Prompt (CMD) y colocar la siguiente sentencia: `ping -t www.google.com`

Aparecerá lo siguiente:



El ping se estará realizando infinitamente.

Ahora, en el CMD del programa se elegirá la interfaz número 4 para hacer las pruebas correspondientes.

Se mostrarán una serie de tramas IP, con distintos protocolos.

[illegible]

Al menos una debe manejar el protocolo ICMP, si eso no ocurre, cambiamos el ping y en lugar de ejecutarlo a Google, hacemos un ping a la IP de nuestra propia máquina.

Si no sabemos la IP, podemos sacarla a través del comando `ipconfig`.

```

C:\> Símbolo del sistema

Adaptador de Ethernet Ethernet 2:

    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . :

Adaptador de Ethernet Ethernet:

    Sufijo DNS específico para la conexión. . :
    Vínculo: dirección IPv6 local. . . : fe80::e1cc:b6e6:17e0:3314%9
    Dirección IPv6 de configuración automática: 109.254.33.20
    Máscara de subred. . . . . : 255.255.0.0
    Puerta de enlace predeterminada. . . . :

Adaptador de Ethernet Ethernet 4:

    Sufijo DNS específico para la conexión. . :
    Vínculo: dirección IPv6 local. . . : fe80::6c09:ef99:ea4c:baf5%2
    Dirección IPv6. . . . . : 192.168.56.1
    Máscara de subred. . . . . : 255.255.255.0
    Puerta de enlace predeterminada. . . . :

Adaptador de LAN inalámbrica Conexión de área local* 2:

    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . :

Adaptador de LAN inalámbrica Conexión de Área local* 4:

    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . :

Adaptador de LAN inalámbrica Wi-Fi:

    Sufijo DNS específico para la conexión. . :
    Dirección IPv6. . . . . : 2086:2f0:9161:4c39:74a6:6405:5d5e:6e86
    Dirección IPv6 temporal. . . . . : 2086:2f0:9161:4c39:9125:82fe:4d85:1827
    Vínculo: dirección IPv6 local. . . : fe80::74a6:b640:5d5e:b682%e
    Dirección IPv6. . . . . : 192.168.100.1
    Máscara de subred. . . . . : 255.255.255.0
    Puerta de enlace predeterminada. . . : fe80::1820
                                           192.168.100.1

```

La IP requerida es la que dice "Puerta de enlace predeterminada".

Haciendo el ping hacia la IP de nuestra computadora, se mostrará lo siguiente:

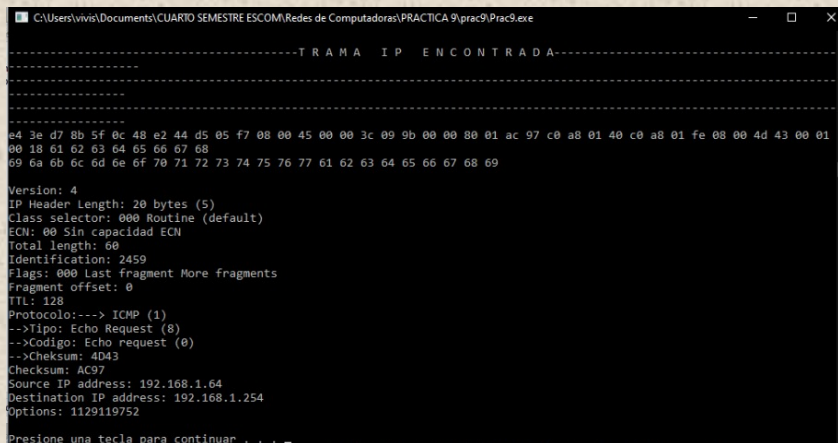
```
C:\Users\vivis>ping -t 192.168.100.1
```

Haciendo ping a 192.168.100.1 con 32 bytes de datos:

Respuesta desde	bytes=32	tiempo=1ms	TTL=64
192.168.100.1:	bytes=32	tiempo=2ms	TTL=64
192.168.100.1:	bytes=32	tiempo=1ms	TTL=64
192.168.100.1:	bytes=32	tiempo=2ms	TTL=64
192.168.100.1:	bytes=32	tiempo=630ms	TTL=64
192.168.100.1:	bytes=32	tiempo=1ms	TTL=64
192.168.100.1:	bytes=32	tiempo=1ms	TTL=64
192.168.100.1:	bytes=32	tiempo=1ms	TTL=64

Similar a lo que se ve cuando se realiza el ping a Google.

Ejecutando de nuevo el programa, podemos observar que ahora si aparece una trama que sigue el protocolo ICMP.



```
C:\Users\vivivi\Documents\CUARTO SEMESTRE ESCOM\Redes de Computadoras\PRACTICA 9\prac9\Prac9.exe
-----TRAMA IP ENCONTRADA-----
-----
e4 3e d7 8b 5f 0c 48 e2 44 d5 05 f7 08 00 45 00 00 3c 09 9b 00 00 80 01 ac 97 c0 a8 01 40 c0 a8 01 fe 08 00 4d 43 00 01
00 18 61 62 63 64 65 66 67 68
09 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69

Version: 4
IP Header Length: 20 bytes (5)
Class selector: 000 Routine (default)
ECN: 00 Sin capacidad ECN
Total length: 60
Identification: 2459
Flags: 000 Last fragment More fragments
Fragment offset: 0
TTL: 128
Protocol:----> ICMP (1)
-->Tipo: Echo Request (8)
-->Codigo: Echo request (0)
-->Checksum: 4043
Checksum: AC97
Source IP address: 192.168.1.64
Destination IP address: 192.168.1.254
Options: 1129119752
Presione una tecla para continuar . . .
```

3. Conclusiones

González Mora Erika Giselle

El Protocolo de Mensajes de Control Internet (ICMP) se usa para este el soporte básico de IP como si se tratara de un protocolo de nivel superior. Sin embargo, ICMP es realmente una parte integrante de IP, y debe ser implementado por todo módulo IP.

Los mensajes ICMP son enviados en varias situaciones: por ejemplo, cuando un datagrama no puede alcanzar su destino, cuando una pasarela no dispone de capacidad de almacenamiento temporal para reenviar el datagrama, y cuando la pasarela puede dirigir al "host" para enviar el tráfico por una ruta más corta.

Con esta práctica comprendí que el Protocolo Internet no está diseñado para ser absolutamente fiable. Pues el propósito de los mensajes de control no es hacer a IP fiable, sino suministrar información sobre los problemas en el entorno de comunicación.

Además de que considero que es algo que usamos en nuestra vida diaria y no lo sabemos.

Olivares Ménez Gloria Oliva

Con esta práctica comprendí mejor cómo funciona y la estructura del Internet Control Message Protocol. Considero que es un protocolo de gran importancia ya que es el encargado de mostrar los errores cuando una comunicación entre paquetes no logra concretarse.

De hecho, considero que es algo que usamos a diario sin saberlo. Por ejemplo, cuando tenemos problemas de conectividad con la red Wi-Fi y tratamos de entrar a alguna página o buscar algo en Google y nos aparece un mensaje de error diciendo que la página tardó demasiado tiempo en responder o que el tiempo de espera se agotó.

4. Referencias

Anónimo. (s.f.). Protocolo de control de mensajes de Internet. Junio 11, 2021, de Wikipedia Sitio web: [https://es.wikipedia.org/wiki/Protocolo de control de mensajes de Internet](https://es.wikipedia.org/wiki/Protocolo_de_control_de_mensajes_de_Internet)

Anónimo. (2019). ¿Qué es el ICMP? Aspectos destacados del protocolo de mensajes. Junio 11, 2021, de Digital Guide IONOS Sitio web: <https://www.ionos.mx/digitalguide/servidores/know-how/que-es-el-protocolo-icmp-y-como-funciona/>