Ping: Socket-RAW Implementation in C

Computer Networks Course - SSRI

Karim Bachir Kaddis Beshay

Registration number: 975410

Date: 04/01/2024

Index

Introduction

Motivation

The ping project was born from the proposal of the **Computer Networks Course** to carry out an optional project as an additional opportunity to acquire **practical skills**, as well as an **assessment**.

Objective

The goal of this project is to implement ping from 0 using:

- The C programming language
- Sockets of the “RAW” type
- The ICMP protocol

A ping program allows users to **send packages** to a specified host and **receive answers** from the same, providing useful information on the round-trip time (**RTT**) and packet loss percentage, as well as a reachability check.

Main Features

- **RAW Sockets and ICMP**: The program uses RAW sockets to allow direct manipulation of packets at the network level, in particular it exploits the ICMP protocol for communication.
- **Checksum Calculation**: The project includes a checksum calculation function, which is essential to ensure the integrity of data transmitted over the network.
- **ICMP Response Handling**: the various ICMP responses are managed and interpreted, providing the user with detailed information on errors or response times.
- **Ping Statistics**: The program collects and presents detailed statistics regarding sent and received packets and the percentage of loss.
- **Safe Exit Interrupt**: The implementation includes an interrupt handler that allows users to interrupt the program cleanly, ensuring that used resources are properly closed.

Start the program

1. Compile the .c file:
   gcc file.c -o pingRAW

2. Run the executable:
   sudo ./pingRAW <IP>

3. To stop the process:
   CTRL+C

```
labreti2019@LABRETI2019-VBOX:~/Scrivania$ gcc client\ socket\ RAW.c -o pingRAW
labreti2019@LABRETI2019-VBOX:~/Scrivania$ sudo ./pingRAW 8.8.8.8
[sudo] password for labreti2019:

Socket file descriptor 3

ICMP_FILTER socket option set...
SO_RCVTIMEO socket option set...

64 bytes from 8.8.8.8: icmp_seq=1 ttl=114 rtt=15.711636 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 rtt=15.499010 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 rtt=14.445957 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=114 rtt=14.214475 ms
^C
=== 8.8.8.8 ping statistics ===
4 packets sent, 4 packets received, 0% packet loss. time: 3470.507516 ms.

Exiting...
labreti2019@LABRETI2019-VBOX:~/Scrivania$
```

Figure 1:Use

Code Analysis

Data Structures and Definitions

- **ICMP_FILTER**: constant representing the value of the ICMP filter option.
- **PING_PCKT_S**: ping packet size, set to 64 bytes.
- **RCV_TIMEOUT**: receive timeout in seconds.
- **ping loop**: variable that controls the main loop of the program.
- **icmp_filter**: structure representing the ICMP filter.
- **ping_pckt** and **rcv_pckt**: structures of the sending and receiving packets, respectively.

Checksum Function

The function in figure[**2**] calculate the **data checksum** provided as argument.
The checksum is a control value that helps to **Verify data integrity** during transmission.

The calculation consists of the **sum of bytes** in pairs (16 bits at a time) and adding any carries.**complement to one** (NOT) of the sum corresponds to the checksum.

```c
uint16_t checksum(void *data, int len){
    uint16_t *buf = data;
    uint32_t sum = 0;
    // Somma dei byte 2 a 2
    while (len > 1){
        sum += *(buf++);
        len -= 2;
    }
    if (len == 1) sum += (uint8_t)*buf;
    // Da 32Byte a 16Byte
    while (sum >> 16){
        sum = (sum >> 16) + (sum & 0xFFFF);
    }
    return ~sum;
}
```

Figure 2:Checksum

send_ping function

This function implements the main loop of the ping program:

- **Setting socket options**: ICMP filter is set to accept only the desired packet types (**ECHOREPLY**,**DESTINATION_UNREACH**,**TIME_EXCELLENT**). Additionally, a receive timeout is set on the socket.

```
// Filtro per accettare solo reply, dst unreachable e time exceeded
struct icmp_filter filter;
filter.data = ~((1 << ICMP_TIME_EXCEEDED) | (1 << ICMP_DEST_UNREACH) | (1 << ICMP_ECHOREPLY));

// Imposta la socket option ICMP_FILTER
if (setsockopt(sd, SOL_RAW, ICMP_FILTER, &filter, sizeof(filter)) < 0){
    printf("\nSetting ICMP_FILTER socket option failed!\n");
    return;
}
printf("\nICMP_FILTER socket option set...\n");

// Imposta la socket option SO_RCVTIMEO
if (setsockopt(sd, SOL_SOCKET, SO_RCVTIMEO, (const char *)&rcv_timeout, sizeof rcv_timeout) < 0){
    printf("\nSetting SO_RCVTIMEO socket option failed!\n");
    return;
}
```

Figure 3:Setting socket options

send_ping function

- **Ping Send and Receive Loop**: the function enters a loop of sending and receiving pings as long as pingloop is equal to 1.
- **Construction of the sending package**: ICMP header, process ID, message sequence and payload are initialized. Then the checksum is calculated and the packet is sent on the socket.

```
// Setup dell'header ICMP
snd_pckt.hdr.type = ICMP_ECHO;
snd_pckt.hdr.un.echo.id = htons(pid);
snd_pckt.hdr.un.echo.sequence = htons(msg_count);
msg_count++;
// Payload ICMP
for (int i = 0; i < sizeof(snd_pckt.msg); i++)
    snd_pckt.msg[i] = i + '0';
// Calcolo della checksum e inserimento nell'header
snd_pckt.hdr.checksum = checksum(&snd_pckt, snd_pckt_len);
```

Figure 4:Construction of the sending package

send_ping function

- **Sending and receiving**: the packet is sent and the response is received.

```
// Invio del pacchetto
int sent = sendto(sd, &snd_pckt, snd_pckt_len, 0, (struct sockaddr *)dst_addr, dst_addr_len);
if (sent <= 0){
    printf("\nPacket send Failed!\n");
} else{
    // Ricezione del pacchetto
    int recvd = recvfrom(sd, &rcv_pckt, rcv_pckt_len, 0, (struct sockaddr *)&r_addr, &r_addr_len);
    if (recvd < 0){
        printf("\nPacket receive failed!\n");
    } else{
```

Figure 5:Sending and receiving

send_ping function

- **Response analysis**: the cases of ”Destination Unreachable”, ”Time Exceeded”, and ”Echo Reply” are handled.

  Round-trip time (RTT) and statistics are calculated.

```
// Dst unreachable
if (rcv_pckt.icmphdr.type == ICMP_DEST_UNREACH){
// Time exceeded
else if (rcv_pckt.icmphdr.type == ICMP_TIME_EXCEEDED){
// Reply
else if (rcv_pckt.icmphdr.type == ICMP_ECHOREPLY
         && rcv_pckt.icmphdr.un.echo.id == htons(pid)){
    printf("%d bytes from %s: icmp_seq=%d ttl=%d rtt=%Lf ms\n",
           PING_PCKT_S, ping_ip, msg_count,
           rcv_pckt.iphdr.ttl, rtt_msec);
    msg_received_count++;
}
else{
    printf("ERROR\n");
}
```

Figure 6:Response Management

send_ping function

- **Ping Statistics Calculation**: At the end of the loop, the total ping statistics are calculated, including packets sent, received and loss percentage.

```
double timeElapsed = ((double)(e_time.tv_nsec - s_time.tv_nsec)) / 1000000;
total_msec = (e_time.tv_sec - s_time.tv_sec) * 1000 + timeElapsed;
pcktLoss = ((msg_count - msg_received_count) / msg_count) * 100;

printf("\n=== %s ping statistics ===\n", ping_ip);
printf("%d packets sent, %d packets received, %d%% packet loss. time: %Lf ms.\n",
       msg_count, msg_received_count, pcktLoss, total_msec);
```

Figure 7:Ping Statistics

Main Function

This is the first function that is called when the process starts.

- The correct number of arguments passed to the program is checked
  1. program name
  2. the IP address

```
int sd, dst_addr_len;
struct sockaddr_in dst_addr;

if (argc != 2){
    printf("\nFormat %s <address>\n", argv[0]);
    return 0;
}
```

Figure 8:Topic Control

Main Function

- A RAW socket (SOCK_RAW) is created with the ICMP protocol (IPPROTO_ICMP).

```
// Setup destinatario
dst_addr.sin_family = AF_INET;
dst_addr.sin_addr.s_addr = inet_addr(argv[1]);
dst_addr_len = sizeof(dst_addr);
// Socket
sd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
if (sd < 0){
    printf("\nSocket file descriptor ERROR!!\nExiting...\n");
    return 0;
}
else printf("\nSocket file descriptor %d\n", sd);
```

Figure 9:Socket creation

Main Function

- **stop signal**(SIGINT) is captured to handle program exit when the user presses CTRL+C.
- The send_ping() function is called to start the send and receive loop.

```
// Cattura l'interrupt
signal(SIGINT, intHandler);

// Chiama il loop
send_ping(sd, &dst_addr, dst_addr_len, argv[1]);

printf("\nExiting...\n");
return 0;
```

Figure 10:Interrupt and call to send_ping()

Future developments

This project offers a solid starting point for further improvements. Here are some possible areas for future expansion and development:

- **IPv6 support**: currently, the project only supports IPv4. A possible future development could include support for IPv6, allowing the program to interact with hosts that use this addressing standard.
- Management of the **Configurable Timeout**: add the ability for users to configure the receive timeout dynamically during program execution.
- Options of **Advanced Configuration**: add advanced options to configure specific ping parameters, such as number of retries, packet size, and other ICMP options.
- **Graphical User Interface** (GUI): Create a graphical interface for the program, making it more accessible to users who prefer visual interaction.