

# CS542 – Computer Networking I - Project Report

**Atharva Tanaji Kadam - A20467229**  
**Yash Pradeep Gupte - A20472798**  
**Amrutham Lakshmi Himaja - A20474105**

## Introduction:

We have implemented ICMP 'ping command' on client and server with raw socket programming in C and python. We have sent two 256 byte messages from the sender to the receiver, and the receiver responds with two echo responses. Both request and response messages are stored in a text file which includes the headers and data.

## Review of technologies:

Each IP packet has a header (20 or 24 bytes) as well as data (variable length). The header contains the source and destination IP addresses, as well as other information that aid in packet routing. The content, such as a string of letters or a section of a webpage, is the data.

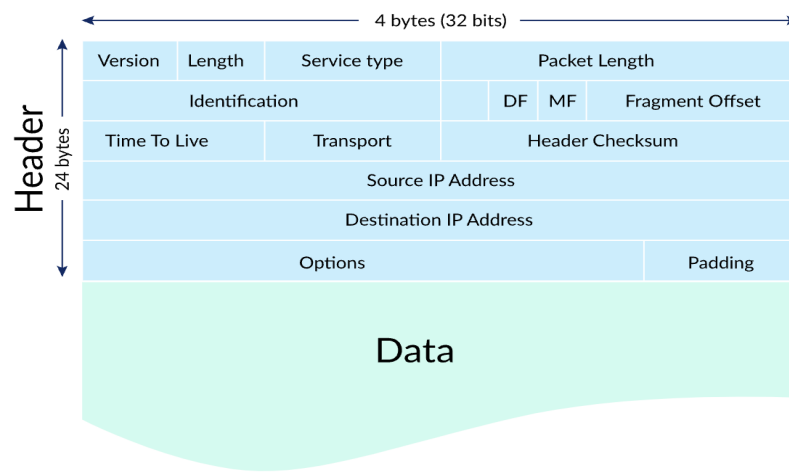


Figure 1: IP Packet

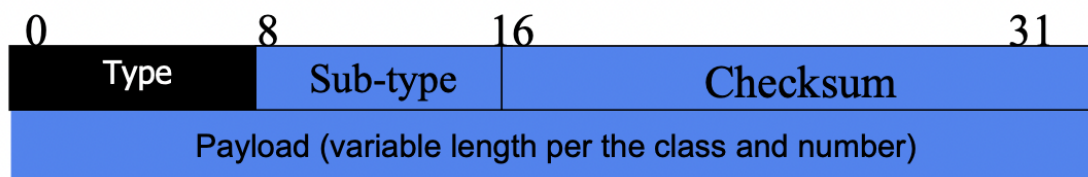


Figure 2. ICMP Packet

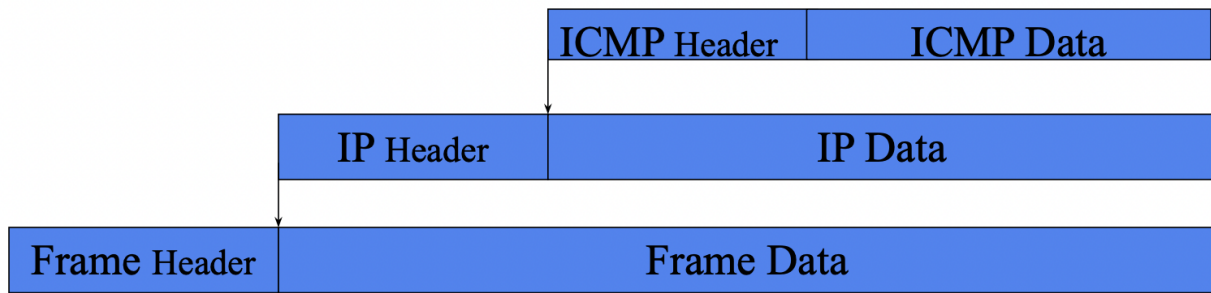


Figure 3. Representation of ICMP packet encapsulated in IP packet

The Internet Control Message Protocol (ICMP) is a network layer protocol that network devices use to identify communication problems in networks. The ICMP protocol is primarily used to determine whether data is reaching its intended destination on time. The ICMP protocol is commonly used on network devices like routers. ICMP is essential for error reporting and testing, but it can also be utilized in DDoS attacks. The main purpose of ICMP is to report errors. When two devices connect via the Internet, the ICMP sends out errors to the sending device if any of the data fails to reach its intended destination. If a packet of data is too large for a router to handle, the router will drop it and send an ICMP message back to the data's original source.

Ping can be implemented in a number of methods, both simple and complicated. Some implementations transmit a single ICMP packet and receive a single response. Some statistics and dynamic parameters are achieved, such as the size of optional data and the number of request messages.

**RAW Sockets:** Sockets are typically used to develop applications that run on top of a transport protocol like Datagram sockets, Stream sockets (TCP) (UDP). A lower layer protocol is required for some applications such as experimental transport protocols, such as ICMP and IGMP, which are built on IP rather than UDP or TCP. A raw socket provides direct IP access which is used to built applications on top of the network layer.

## Experiments:

We extend the application of ping command by utilizing raw sockets , ip packet and icmp packets. Primarily we will utilize the type 8 and type 0 which is ECHO REQUEST AND ECHO REPLY functions of ICMP to initiate our icmp packet. We then create an ICMP packet by initializing the data, checksum, type, code, ID and sequence number .

## 1. Python implementation

Ping is a computer network application that checks whether a specific host can be reached via an IP network. It operates by sending ICMP "echo reply" packets to the target host and listening for responses. Ping calculates the round-trip duration, logs packet loss, and generates a statistical summary of the received echo back packets and saves the messages that are sent and received at the receiver side and the sender side.

The steps we followed for the implementation is :

1. Socket creation
2. Adding the required header
3. Sending the socket from Server to Client
4. At Client, unpack the socket and write the message contents in a file
5. Send a response the server
6. At Server, unpack the socket and write the message contents in a file

```
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.getprotobyname("icmp"))
```

We have initially created a socket using socket which is imported from the socket library in python. The three parameters we have used for the socket creation are socket.AF\_INET, socket.SOCK\_RAW, socket.getprotobyname("icmp"). The AF\_INET is for the IPv4 and SOCK\_RAW refers to the socket creation using the raw\_socket. getprotobyname("icmp") refers to the prototype we are using is the icmp.

```
def message(seq):
    icmp_type = 8
    icmp_code = 0
    icmp_check_sum = 0
    icmpID = os.getpid() & 0xffff
```

```
send_packet = struct.pack('>BBHHH248s', icmp_type, icmp_code,
                           icmp_check_sum, icmpID, seq, data_part)
```

In the message function we have generated the message by keeping the first 8-bits are the message types if the message type is 8 then it is a Request and if it is 0 then it is Reply. The next 8-bits is the message type code which helps to provide the additional information about the type of error. The last 16 bits will be checksum which helps to check for the consistency of the ICMP message header for making sure the full data to be delivered. The header of the packet which is being generated in the message function is BBHHH248s – B is the unsigned character with the standard size of 1 and H is the unsigned short with the standard size of 2. The header will have a total size of 8 bytes.

```
def sending(icmpPacket, addr):
    s.sendto(icmpPacket, (addr, 1234))
    return s, addr
```

In the sending function we are sending the icmp packet to the host using the socket.sendto() with a dummy port number 1234.

```
# we are setting the bytes of the received packet to 1024 bytes
rcvd_packet, rcvdAddr = rawsocket.recvfrom(1024)
```

In the receive function we are receiving the packet that has been sent by the sending function using the socket.recvfrom function, the received packet have the length of 256+8 ICMP headers , you will notice that the lengths (in Bits) of the various fields are as follows: 8, 8, 16, 16, 16. If the sender is sending 256 Bytes length of data the receiver adds 8 Bytes and total to 264 Bytes.

```
def ping_command(host, noMsgs=2, timeout=2):
    address = socket.gethostbyname(host) # obtain ip Address
    print(" Ping {0} [{1}] have 256 Bytes of data :".format(host, address))
    timeLost = 0
    timeAccept = 0
    temptime = 0.0
    # Count the time of all packets send-receive
    count = []
```

In the ping\_command function we have passed the host as the local host (127.0.0.1), noMsgs is the number of messages that we wanted to send and receive as per the project we are sending the 2 messages with 256 lengths. The timeLost, timeAccept, temptime attributes are used to calculate the start time and end time for the packets to send and receive.

```
for i in range(noMsgs):
    i += 1
    icmp_pkt = message(i)

    rawsocket, destAddr = sending(icmp_pkt, address)
```

In this piece of code we are generating 2 messages and the messages are sent to the client or receiver using the sending function and returning values are saved in the rawsocket and destaddr variables. After we are getting the rawsocket and the destination address we are passing the rawsocket into the receiver function to get the replyTime, sequence and ttl. If the replyTime is less than 0 it means the request is timed out and if the response is greater than or equal to 0 the statistics(timeAccept, temptime) are calculated based on the replyTime.

```
ping_command("127.0.0.1")
```

We have called the ping\_command function by passing the localhost as an address to perform the above experiments. We have created two files "reply.txt" and "request.txt" to save the messages which are being sent and received. "Request.txt" consists of the data part which

is getting added into the packet, the length of the sending packet and the packet which is requested. "reply.txt" consists of length of the received file packet, address from which it received the packet, the packet which is received, The icmp header of the received packet and the icmp packet type.

## 2. C implementation

We implement raw socket in C by creating two files - client.c and sever.c. The execution outline is that the server pings to the client, then the client receives 2 ICMP requests from the server, the client will capture the data and save it in a text file along with some additional information. Next, the client will send ICMP REPLY back to the server which will print it on the screen as well as save the output in a text file. To achieve this, we first create an IP packet and an ICMP packet. The ICMP packet contains the type, sequence number, checksum, code and the payload/data.

We will be encapsulating the packets into a raw socket which will be sent and received between both the client and server programs. This ping implementation is a client server model sending and receiving ICMP messages (request / reply).

In both the programs we first define ip and icmp packet structures along with checksum. We allocate memory for these packets and the data section. We then initialize the raw sockets and bind them. We fill the icmp and ip packets with their required parameters.

The server programs first implements the sendto function of raw packets which will send an ICMP REQUEST to the client. Now in the client program we will implement the recvfrom function of raw socket which will receive the sent icmp request from the server. The client program will unpack the icmp header and data with the help of a decode\_response method which we have implemented and then create an ICMP REPLY which will be sent back to the server using the sendto method again. In the server part, it will receive the client's reply using the recvfrom method. In the end, both of these programs will save the ICMP REQUEST and REPLY messages and data into text files. The decode response method unpack all the values from the response which will be printed on the terminal and then saving these values in a text file.

For this implementation we have included winsock2.h and the standard libraries required for execution. Both of our programs client and server take one argument and that is the IP address to which we want to ping. In our case we have performed a ping query to the localhost that is 127.0.0.1. The following execution commands were performed on two separate terminals. We generate the object files of each of these c programs. Beginning with the server, execute the server file along with the IP address argument. On a separate terminal execute the client file along with the same IP address.

*Terminal one:*

```
gcc server.c -o client -lws2_32
Server 127.0.0.1
```

*Terminal two:*

```
gcc client.c -o client -lws2_32
client 127.0.0.1
```

### 1. Output for python program:

**reply.txt**

## Response.txt

[illegible]

## 2. Output for C program:

Terminal one

```
C:\Users\athar\OneDrive\Desktop\course work\CS 542 CN\final project>gcc server.c -o client -lws2_32
server.c: In function 'main':
server.c:124:9: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
    sleep(2);
    ^~~~~~

C:\Users\athar\OneDrive\Desktop\course work\CS 542 CN\final project>server 127.0.0.1
36 bytes from 127.0.0.1: icmp_seq = 1. time: 8031 ms icmp data : abcedfghijkiufbpaiuf
Replies received
36 bytes from 127.0.0.1: icmp_seq = 2. time: 8031 ms icmp data : abcedfghijkiufbpaiuf
Replies received

C:\Users\athar\OneDrive\Desktop\course work\CS 542 CN\final project>
```

Terminal two

```
C:\Users\athar\OneDrive\Desktop\course work\CS 542 CN\final project>gcc client.c -o client -lws2_32
client.c: In function 'main':
client.c:95:6: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
    sleep(3);
    ^~~~~~

C:\Users\athar\OneDrive\Desktop\course work\CS 542 CN\final project>client 127.0.0.1
36 bytes from 127.0.0.1: icmp_seq = 2. 0 time: 0 ms icmp data : abcedfghijkiufbpaiuf
Request received
36 bytes from 127.0.0.1: icmp_seq = 3. 0 time: 1000 ms icmp data : abcedfghijkiufbpaiuf
Request received

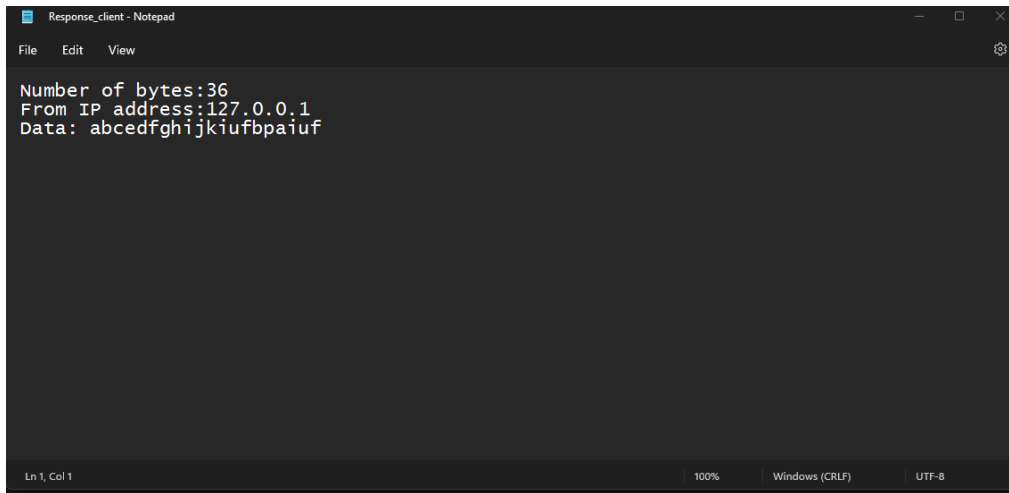
C:\Users\athar\OneDrive\Desktop\course work\CS 542 CN\final project>
```

Request file.txt

```
Request_server - Notepad
File Edit View
Number of bytes:36
From IP address:127.0.0.1
Data: abcedfghijkiufbpaiuf

Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Response file.txt



```
Response_client - Notepad
File Edit View
Number of bytes:36
From IP address:127.0.0.1
Data: abcedfghijklufbpauf
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Server sends the two packets to the client consisting of data and headers which the client receives and unpacks those messages. Client saves the messages to a text file and creates response packets to send it to the client. Client receives the response packets and saves it to a text file.

### Conclusion:

In conclusion, we have successfully implemented ICMP 'ping command' on client and server with raw socket programming in C and python. We were successfully able to write ECHO REQUEST and ECHO RESPONSE in 2 separate text files for both the messages. The python implementation executes a simple ping command to localhost and ICMP REQUEST and REPLY messages are communicated. We are saving the request and reply messages / data in two text files. We migrated from Python to C to get a better understanding of how the raw sockets and icmp works. The C implementation gave us better clarity when we constructed a client server model where the server sends a ping command to the client and the client replies with an ICMP reply. We learnt that raw sockets can be used to perform transmission of IP packets which can contain any type of data ( ICMP in our case) with manual configurations which can be modified according to the application requirements. We have uploaded our work and README file on [github](#).