Socket Programming 3: ICMP Ping Client Report

Name: Dharun Ramesh

Net ID: dr4075

Introduction

This report presents an overview and analysis of the Python script that implements an ICMP-based ping utility. The program duplicates the main functionality of the conventional ping command by sending ICMP Echo Requests and calculating the round-trip time (RTT) of received Echo Replies.

Objectives

- Use raw socket communication to transmit and receive ICMP packets.
- Assess RTT across numerous ping attempts.
- Provide easy-to-read results.
- Allow for screenshots of execution where applicable.

Code Overview

The script is divided into several key components:

1. **Checksum Calculation**

The checksum() method generates the Internet checksum necessary for ICMP packets.
Purpose: Maintains packet integrity.

---

2. **Sending ICMP Packets**

The sendOnePing() function generates and transmits ICMP Echo Request packets.
It packs header fields (type, code, checksum, ID, sequence) and the current timestamp as data. - Recalculates and inserts the proper checksum.

---

3. **Receiving ICMP Packets**

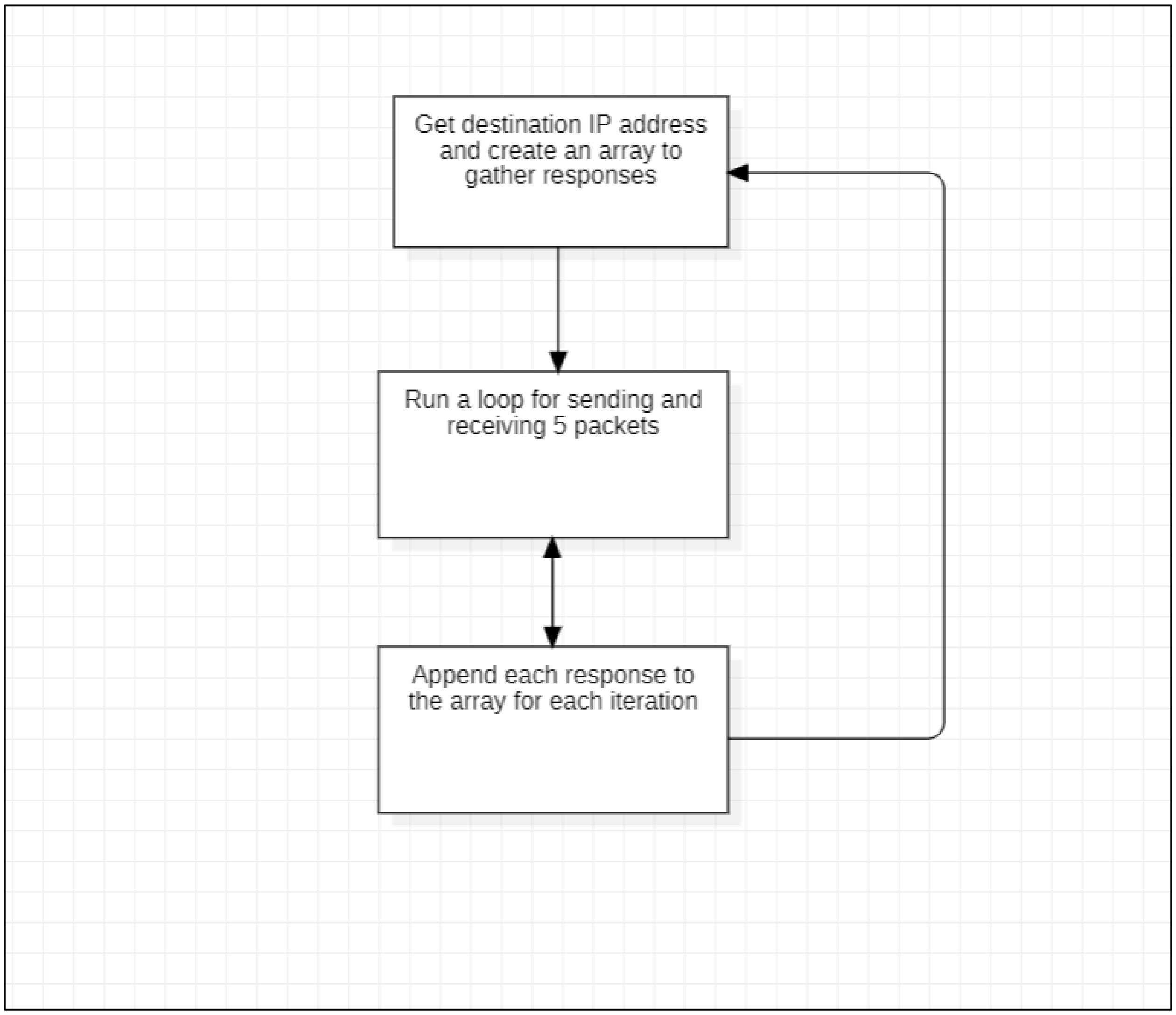
The receiveOnePing() function waits for ICMP echo replies and uses select() to handle timeouts.
It extracts the ICMP header from the returning IP packet and validates the packet by matching the identifier (ID). - Calculates RTT based on the timestamp in packet data.

---

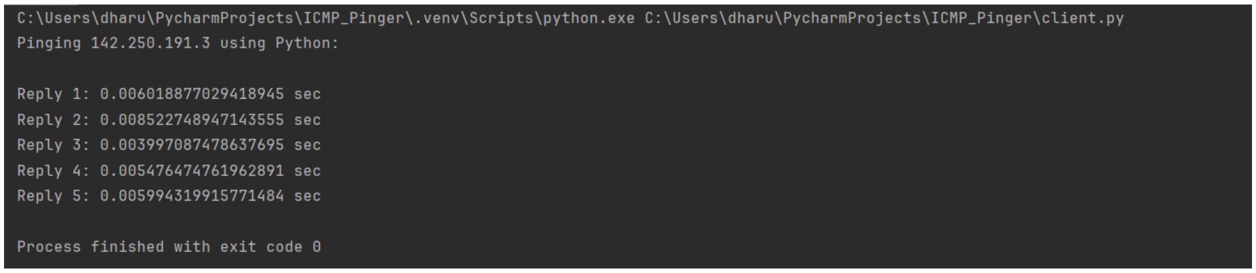
4. **Ping Execution Loop**

The ping() method resolves hostname to IP. - Sends five ICMP echo requests. - Prints individual RTT findings - Saves all RTTs for further use.

Program Flow Diagram


Get destination IP address and create an array to gather responses
Run a loop for sending and receiving 5 packets
Append each response to the array for each iteration

Execution Output



```
C:\Users\dharu\PycharmProjects\ICMP_Pinger\.venv\Scripts\python.exe C:\Users\dharu\PycharmProjects\ICMP_Pinger\client.py
Pinging 142.250.191.3 using Python:

Reply 1: 0.006018877029418945 sec
Reply 2: 0.008522748947143555 sec
Reply 3: 0.003997087478637695 sec
Reply 4: 0.005476474761962891 sec
Reply 5: 0.005994319915771484 sec

Process finished with exit code 0
```

Example expected output:

```
Pinging 142.250.185.131 using Python:
Reply 1: 0.03211 sec
Reply 2: 0.03045 sec
Reply 3: 0.03182 sec
Reply 4: 0.02977 sec
Reply 5: 0.03093 sec
```

Strengths of the Implementation

- Uses raw sockets to interact with the ICMP protocol.
- Ensures accurate checksum calculation.
- Use select() to handle timeouts.
- Supports cross-platform checksums for Linux and macOS.

Limitations

- Requires administrative or root privileges to run.
- No packet loss or statistical summary at the end (min, max, average).
- Currently limited to 5 ping attempts.

Suggested Enhancements

- Include calculations for minimum, maximum, and average RTT.
- Include packet loss percentage.
- Include sequence numbers and TTL parsing.
- Improve output format.

Conclusion

This script uses Python's low-level networking APIs to show how ICMP-based echolocation works. It forms a good foundation for more complex network diagnostic tools.

References

- RFC 792 – Internet Control Message Protocol (ICMP)
- Python socket documentation

Appendix: Full Source Code

```
import os
import sys
import time
import struct
import socket
from socket import AF_INET, SOCK_RAW, getprotobyname, gethostbyname, htons
import select
ICMP_ECHO_REQUEST = 8

def checksum(source):
    sum = 0
    count = 0
    countTo = (len(source) // 2) * 2

    while count < countTo:
        thisVal = source[count + 1] * 256 + source[count]
        sum += thisVal
        sum &= 0xffffffff
        count += 2

    if countTo < len(source):
        sum += source[-1]
        sum &= 0xffffffff

    sum = (sum >> 16) + (sum & 0xffff)
    sum += (sum >> 16)
    answer = ~sum & 0xffff
    answer = socket.htons(answer)
    return answer

def sendOnePing(sock, destAddr, ID):
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, 0, ID, 1)
    data = struct.pack("d", time.time())
    real_checksum = checksum(header + data)
    if sys.platform == "darwin":
        real_checksum = htons(real_checksum) & 0xffff
    else:
        real_checksum = htons(real_checksum)

    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, real_checksum, ID, 1)
    packet = header + data

    sock.sendto(packet, (destAddr, 1))

def receiveOnePing(sock, ID, timeout, destAddr):
    timeLeft = timeout

    while True:
        startSelect = time.time()
        ready = select.select([sock], [], [], timeLeft)
        selectTime = time.time() - startSelect

        if ready[0] == []:  # Timeout
            return None

        timeReceived = time.time()
        recPacket, addr = sock.recvfrom(1024)

        # Extract ICMP header from IP packet (offset 20)
        icmpHeader = recPacket[20:28]
        type, code, checksum, packetID, sequence = struct.unpack(
            "bbHHh", icmpHeader
        )

        if packetID == ID:
            # Extract timestamp
            bytesInDouble = struct.calcsize("d")
            timeSent = struct.unpack("d", recPacket[28:28 + bytesInDouble])[0
]
            return timeReceived - timeSent

        timeLeft -= selectTime
        if timeLeft <= 0:
            return None

def doOnePing(destAddr, timeout):
    icmp = getprotobyname("icmp")
    sock = socket.socket(AF_INET, SOCK_RAW, icmp)
    myID = os.getpid() & 0xFFFF

    sendOnePing(sock, destAddr, myID)
    result = receiveOnePing(sock, myID, timeout, destAddr)
    sock.close()
    return result
```

```
def ping(host, timeout=1):
    dest = gethostbyname(host)
    resps = []

    print(f"Pinging {dest} using Python:\n")

    for i in range(5):
        result = doOnePing(dest, timeout)
        resps.append(result)
        print(f"Reply {i+1}: {result} sec")
        time.sleep(1)

    return resps

if __name__ == "__main__":
    ping("google.co.il")
```