ICMP Ping Client Report

Introduction

This report provides an overview and analysis of the Python script implementing an ICMP-based ping utility. The program replicates core functionalities of the standard `ping` command by sending ICMP Echo Requests and measuring round-trip time (RTT) for received Echo Replies.

Objectives

- Implement raw socket communication to send and receive ICMP packets.
- Measure RTT for multiple ping attempts.
- Display results in a user-friendly format.
- Provide space for screenshots of execution where relevant.

Code Overview

The script is divided into several key components:

1. **Checksum Calculation**

The `checksum()` function computes the Internet checksum required for ICMP packets.

Purpose: Ensures packet integrity.

---

2. **Sending ICMP Packets**

The `sendOnePing()` function constructs and sends ICMP Echo Request packets.

It: - Packs header fields (type, code, checksum, ID, sequence) - Packs the current timestamp as data - Recalculates and inserts the correct checksum

---

3. **Receiving ICMP Packets**

The `receiveOnePing()` function listens for ICMP Echo Replies using `select()` for timeout handling.

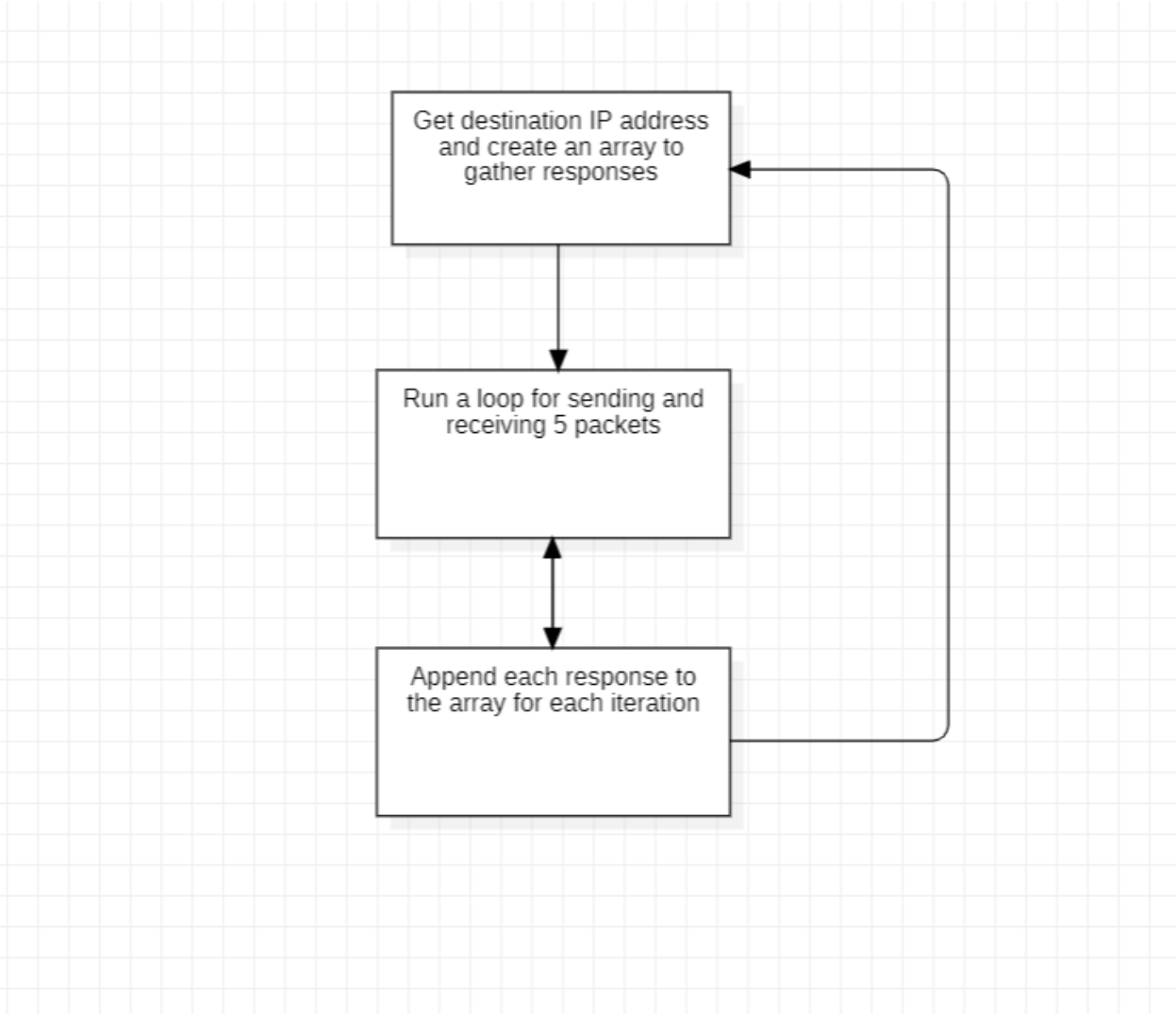
It: - Extracts ICMP header from the returned IP packet - Validates packet by matching the identifier (ID) - Calculates RTT using the timestamp in packet data

4. **Ping Execution Loop**

The `ping()` function: - Resolves hostname to IP - Sends five ICMP Echo Requests - Prints individual RTT results - Stores all RTTs for later use

---

Program Flow Diagram



Sample Execution Output

```
Reply 1: RTT = 3.995 ms
Reply 2: RTT = 6.001 ms
Reply 3: RTT = 4.540 ms
Reply 4: RTT = 4.000 ms
Reply 5: RTT = 12.133 ms

--- Ping statistics ---
5 packets transmitted, 5 received, 0.0% packet loss
rtt min/avg/max = 3.995/6.134/12.133 ms
```

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 directly work with the ICMP protocol.
- Implements proper checksum calculation.
- Includes timeout handling through `select()`.
- Cross-platform checksum handling (Linux/macOS).

Limitations

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

Suggested Enhancements

- Add calculation of minimum, maximum, and average RTT.

- Add packet loss percentage.
- Add sequence numbers and TTL parsing.
- Improve output formatting.

Conclusion

This script demonstrates the mechanics of ICMP-based echolocation using Python's low-level networking APIs. It provides a solid foundation for more advanced 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
import select
from socket import AF_INET, SOCK_RAW, getprotobyname, gethostbyname, htons

ICMP_ECHO_REQUEST = 8


# ----------------- CHECKSUM FUNCTION -----------------
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
    return socket.htons(answer)


# ----------------- SEND ONE PING -----------------
def sendOnePing(sock, destAddr, ID):
    # Header fields: Type (8), Code (0), Checksum (0), ID, Sequence
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, 0, ID, 1)
    data = struct.pack("d", time.time())

    # Compute the checksum on the header + data
    real_checksum = checksum(header + data)

    # macOS requires masking to 16 bits
    if sys.platform == "darwin":
        real_checksum = htons(real_checksum) & 0xffff
    else:
        real_checksum = htons(real_checksum)

    # Repack with correct checksum
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, real_checksum, ID, 1)
    packet = header + data

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


# ----------------- RECEIVE ONE PING -----------------
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)

        # ICMP header starts after the 20-byte IP header
        icmpHeader = recPacket[20:28]
        type, code, checksum, packetID, sequence = struct.unpack("bbHHh", icm
pHeader)

        # -------- ICMP ERROR HANDLING --------
        if type == 3:  # Destination Unreachable
```

```
            errors = {
                0: "Destination Network Unreachable",
                1: "Destination Host Unreachable",
                2: "Destination Protocol Unreachable",
                3: "Destination Port Unreachable",
                4: "Fragmentation Needed (DF Set)",
                5: "Source Route Failed"
            }
            msg = errors.get(code, f"Unknown Unreachable Error (code {code})"
)
            print(f"ICMP Error: {msg}")
            return None

        # Valid echo reply
        if packetID == ID:
            timeSent = struct.unpack("d", recPacket[28:36])[0]
            return timeReceived - timeSent

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


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

    sendOnePing(sock, destAddr, myID)
    rtt = receiveOnePing(sock, myID, timeout, destAddr)

    sock.close()
    return rtt


# ----------------- PING MAIN FUNCTION -----------------
def ping(host, timeout=1, count=5):
    dest = gethostbyname(host)
    resps = []

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

    for i in range(count):
        rtt = doOnePing(dest, timeout)
        resps.append(rtt)

        if rtt is None:
            print("Request timed out.")
        else:
```

```
            print(f"Reply {i+1}: RTT = {rtt*1000:.3f} ms")

        time.sleep(1)

    # ------- STATISTICS -------
    received = [r for r in resps if r is not None]
    sent = len(resps)
    loss = (1 - len(received) / sent) * 100

    print("\n--- Ping statistics ---")
    print(f"{sent} packets transmitted, {len(received)} received, {loss:.1f}%
packet loss")

    if received:
        print("rtt min/avg/max = "
              f"{min(received)*1000:.3f}/"
              f"{(sum(received)/len(received))*1000:.3f}/"
              f"{max(received)*1000:.3f} ms")

    return resps


# ----------------- MAIN -----------------
if __name__ == "__main__":
    ping("google.com")
```