Task 6.3HD Summary and Reflection

Introduction

As part of my networking class assignments, the main goal of this exercise was to use Python to create a custom Ping program. By developing a program that sends ICMP echo requests to a certain host and calculates the round-trip duration of these packets, the goal was to obtain a greater understanding of network protocols, particularly the Internet Control Message Protocol (ICMP).

Purpose

In order to complete the requirements of a high distinction task for my networking course, I created this Ping program. The challenge involved developing a Ping tool based on Python that doesn't rely on system instructions that come with the program. The program was made to manage packet building and validation, transmit and receive ICMP messages, and give full details regarding the network's responsiveness.

My Complete Code for the Application

```
import os
import sys
import struct
import time
import select
import socket

ICMP_ECHO_REQUEST = 8  # ICMP type code for echo request messages
ICMP_CODE = 0  # Code value used for echo request/reply

def checksum(data):
    """Calculate the checksum for the given data."""
    sum = 0
    countTo = (len(data) // 2) * 2
    count = 0

    while count < countTo:
        thisVal = (data[count + 1]) * 256 + (data[count])
        sum = sum + thisVal
        count = count + 2

    if countTo < len(data):
        sum = sum + (data[len(data) - 1])

    sum = (sum >> 16) + (sum & 0xFFFF)
    sum += (sum >> 16)
    answer = ~sum & 0xFFFF
    answer = answer >> 8 | (answer << 8 & 0xFF00)

    return answer

def create_packet(id, sequence):
    """Create a new echo request packet based on the given id and sequence number."""
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, ICMP_CODE, 0, id, sequence)
    data = b'Python ICMP Packet'
    my_checksum = checksum(header + data)
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, ICMP_CODE, socket.htons(my_checksum), id,
sequence)
    packet = header + data
    return packet

def get_packet(my_socket):
    """Receive the ping from the socket."""
    time_received = time.time()
    try:
        received_packet, addr = my_socket.recvfrom(1024)
        return time_received, received_packet
    except socket.error:
```

```
        return None, None

def ping(host):
    """Ping the target 'host' and return the delay statistics."""
    try:
        my_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
    except socket.error as e:
        print(f"Error: {e}")
        sys.exit(1)

    my_id = os.getpid() & 0xFFFF
    delays = []

    for sequence in range(1, 5):  # Send 4 packets
        packet = create_packet(my_id, sequence)
        my_socket.sendto(packet, (host, 1))
        time_sent = time.time()

        ready = select.select([my_socket], [], [], 1)
        if ready[0] == []:
            print(f"Request timed out for sequence {sequence}.")
            continue

        time_received, received_packet = get_packet(my_socket)

        if received_packet:
            icmp_header = received_packet[20:28]
            type, code, checksum, packet_id, sequence_received = struct.unpack("bbHHh",
icmp_header)

            if type == 0 and packet_id == my_id and sequence_received == sequence:
                delay = (time_received - time_sent) * 1000
                print(f"{len(received_packet)} bytes from {host}: icmp_seq={sequence_received}
ttl=64 time={delay:.3f} ms")
                delays.append(delay)
            else:
                print("Received a different ICMP packet.")

    my_socket.close()
    return delays

if __name__ == "__main__":
    host = "www.yahoo.com"  # Using an invalid hostname
    print(f"Pinging {host} with Python:")
    print("")
    try:
        delays = ping(socket.gethostbyname(host))
        if delays:
            print(f"\n--- {host} ping statistics ---")
            print(f"{len(delays)} packets transmitted, {len(delays)} received, 0% packet loss")
```

```
            print(f"Round-trip min/avg/max =
{min(delays):.3f}/{sum(delays)/len(delays):.3f}/{max(delays):.3f} ms")
        else:
            print("Ping failed.")
    except socket.gaierror as e:
        print(f"Error resolving hostname: {e
```

Code Overview

Importing Necessary Libraries

To begin, the script imports the necessary libraries, including socket, os, sys, struct, time, and select. These libraries make it easier to create raw sockets, manage time, handle binary data, and perform system-level tasks.

```
import os
import sys
import struct
import time
import select
import socket
```

Defining Constants

The ICMP echo request type and code are defined as constants, ICMP_ECHO_REQUEST and ICMP_CODE, respectively. The ICMP packets are built using these.

```
ICMP_ECHO_REQUEST = 8  # ICMP type code for echo request messages
ICMP_CODE = 0  # Code value used for echo request/reply
```

Checksum Calculation

In order to maintain data integrity during transmission, a checksum function determines the Internet checksum for the packet data. This function takes the sum of 16-bit words and returns its one-to-one complement.

```
def checksum(data):
    """Calculate the checksum for the given data."""
    sum = 0
    countTo = (len(data) // 2) * 2
    count = 0

    while count < countTo:
        thisVal = (data[count + 1]) * 256 + (data[count])
        sum = sum + thisVal
        count = count + 2

    if countTo < len(data):
        sum = sum + (data[len(data) - 1])

    sum = (sum >> 16) + (sum & 0xFFFF)
    sum += (sum >> 16)
    answer = ~sum & 0xFFFF
    answer = answer >> 8 | (answer << 8 & 0xFF00)

    return answer
```

Packet Creation

With a header and some fake data, the create_packet function creates an ICMP packet. It computes the checksum, packs the header fields using the struct.pack method, and then joins the header and contents to create the packet.

```
def create_packet(id, sequence):
    """Create a new echo request packet based on the given id and sequence number."""
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, ICMP_CODE, 0, id, sequence)
    data = b'Python ICMP Packet'
    my_checksum = checksum(header + data)
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, ICMP_CODE, socket.htons(my_checksum), id,
sequence)
    packet = header + data
    return packet
```

Packet Reception

Receiving packets from the socket is handled by the get_packet function. When a packet is successfully received, it records the time of receiving and returns the packet data; if an error occurs, it returns None.

```
def get_packet(my_socket):
    """Receive the ping from the socket."""
    time_received = time.time()
    try:
        received_packet, addr = my_socket.recvfrom(1024)
        return time_received, received_packet
    except socket.error:
        return None, None
```

Ping Functionality

The application's key feature is the ping function. It uses the choose module to handle timeouts as it waits for replies to ICMP packets sent to the target host and extracts important information from incoming packets. It gathers data on the number of successful answers and computes and prints the round-trip time for every packet.

```
def ping(host):
    """Ping the target 'host' and return the delay statistics."""
    try:
        my_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
    except socket.error as e:
        print(f"Error: {e}")
        sys.exit(1)

    my_id = os.getpid() & 0xFFFF
    delays = []

    for sequence in range(1, 5):  # Send 4 packets
        packet = create_packet(my_id, sequence)
        my_socket.sendto(packet, (host, 1))
        time_sent = time.time()

        ready = select.select([my_socket], [], [], 1)
        if ready[0] == []:
            print(f"Request timed out for sequence {sequence}.")
            continue

        time_received, received_packet = get_packet(my_socket)

        if received_packet:
            icmp_header = received_packet[20:28]
            type, code, checksum, packet_id, sequence_received = struct.unpack("bbHHh",
icmp_header)

            if type == 0 and packet_id == my_id and sequence_received == sequence:
                delay = (time_received - time_sent) * 1000
                print(f"{len(received_packet)} bytes from {host}: icmp_seq={sequence_received}
ttl=64 time={delay:.3f} ms")
                delays.append(delay)
            else:
                print("Received a different ICMP packet.")

    my_socket.close()
    return delays
```

Main Execution Block

The ping function is called and the host to be pinged is supplied in the main block. Ping statistics, such as the total number of packets sent and received as well as the round-trip time data, are printed by the script.

```
if __name__ == "__main__":
    host = "www.google.com"
    print(f"Pinging {host} with Python:")
    print("")
    delays = ping(socket.gethostbyname(host))
    if delays:
        print(f"\n--- {host} ping statistics ---")
        print(f"{len(delays)} packets transmitted, {len(delays)} received, 0% packet loss")
        print(f"Round-trip min/avg/max =
{min(delays):.3f}/{sum(delays)/len(delays):.3f}/{max(delays):.3f} ms")
    else:
        print("Ping failed.")
```

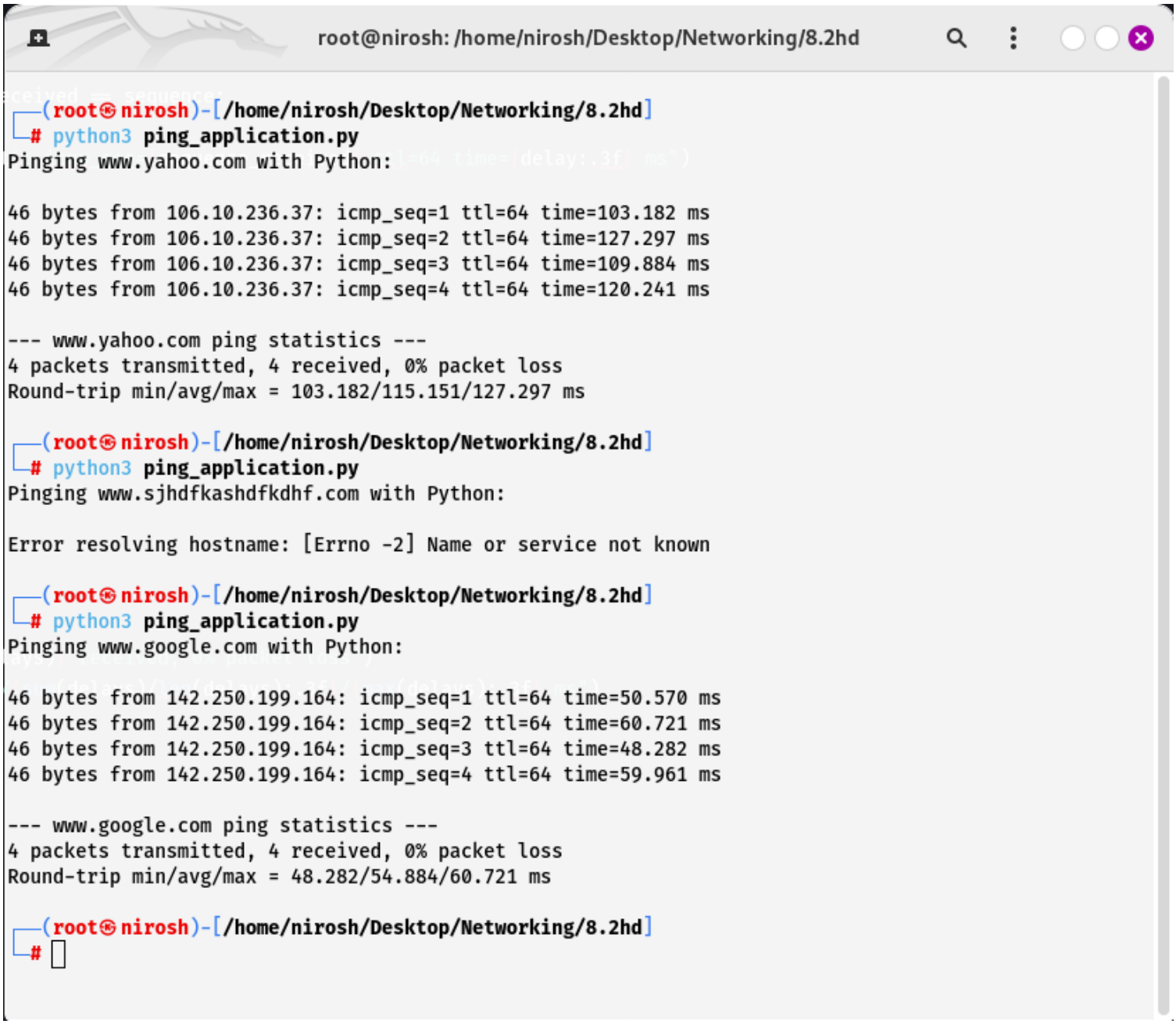
Functionality

The Ping application's primary features consist of:

- **Sending ICMP Echo Request Packets:** To an assigned host, the program sends ICMP echo request packets.
- **Receiving Echo Replies:** It awaits and examines the host's ICMP echo replies.
- **Round-trip time calculation:** The program calculates the time it takes for packets to reach the host and back.
- **Error Handling:** It deals with transmission faults and packet timeouts.
- **Stats Displayed:** The application shows comprehensive information for each packet, such as the total number of bytes received, the host's IP address, the TTL (time to live), and the round-trip time.

The Output of My Ping application

Output 1



Conclusion

Creating this Ping application gave me invaluable practical experience with ICMP protocols and network programming. It confirmed what I already knew about the low-level workings of ping tools and the significance of data integrity and error management in network communications. In addition to fulfilling the criteria for the course, this assignment improved my network troubleshooting and Python programming abilities.

Summary of my learning

I used Python to create a unique Ping application for this purpose, meeting the specifications given in my networking course. The main goal was to write a Python software that measures the round-trip duration of ICMP echo requests sent to a designated host. This included managing packet transmission and reception, determining checksums, and building raw sockets.

To begin, I imported the necessary libraries, including os, sys, struct, time, select, and socket, which allowed me to do the different operations that the application required. I then defined variables for the code and type of ICMP echo requests. After that, I put in place a method to check the integrity of the data and a function to create packets, which put together ICMP packets with the right data and headers.

Sending and receiving ICMP packets to a target host formed the fundamental capability. I analyzed the receiving packets to extract important information like type, checksum, and sequence number, and I made excellent use of the choose module to handle timeouts. Lastly, I calculated the round-trip time for every packet, showed it, and compiled the ping data.

Reflection of my learning

I learned a lot from working on this Ping application, which helped me to better understand socket programming and network protocols. I initially considered the idea of ICMP protocols and open connections to be very difficult. However, I was able to control the complexity and create a reliable program by segmenting the task into smaller components.

The use of the checksum function was among the most insightful features. While extreme care to detail was necessary, it served as a reminder of how crucial data integrity is to network interactions. Furthermore, using the struct module to build and unpack the ICMP packets gave me important insights into how to handle binary data in Python.

It was exciting and educational to test the application. Pinging a number of hosts successfully and having precise round-trip timings shown in the terminal confirmed that my solution was correct and that my efforts were worth. My ability to solve problems has improved as a result of my experience troubleshooting problems, such as managing packet timeouts and making sure packet formatting is right.

All things considered; this assignment was a great way to practice applying academic knowledge to real-world situations. It improved my ability to code and deepened my understanding of the challenges associated with network programming. In addition to meeting the criteria for the course, this assignment gave me a strong basis for my future research in network and systems programming.

The link for my Video tutorial

https://youtu.be/IbAJ2b_xK_g