UDP Ping and Heartbeat Implementation

Aditya Sharma (2022038) Abhay Dagar (2022014)

September 20, 2024

1 Introduction

This document outlines the implementation of two UDP-based network applications: a Ping system and a Heartbeat system. Both systems are designed to demonstrate basic principles of network programming, including handling unreliable connections and simulating real-world network conditions.

2 Part 1: UDP Ping System

2.1 Overview

The UDP Ping system consists of a client and server that communicate using the User Datagram Protocol (UDP). The client sends ping messages to the server, which echoes them back. This system simulates packet loss to mimic real network conditions.

2.2 Server Implementation

The server listens for incoming UDP packets, simulates a 30% packet loss, and responds to the client with the capitalized version of the received message.

2.3 Client Implementation

The client sends 10 ping messages to the server, calculates round-trip time (RTT) for each successful ping, and handles timeouts for lost packets.

2.4 Key Features

- Simulates 30% packet loss
- Calculates and displays RTT for each successful ping
- Provides summary statistics (min, max, average RTT, packet loss rate)

2.5 Sample Code Snippet

Here's a snippet of the client code:

```python
import socket
import time

# ... (rest of the code)

for i in range(10):
    start = time.time()
    message = 'Ping ' + str(i + 1) + " " + time.ctime(start)
    try:
        sent = mysocket.sendto(message.encode("utf-8"), server_address)
        data, server = mysocket.recvfrom(4096)
        end = time.time()
        elapsed = end - start
        print(f"RTT: {elapsed * 1000} milliseconds")
    except socket.timeout:
        print("#" + str(i) + " Request Timed out")
```

3 Part 2: UDP Heartbeat System

3.1 Overview

The UDP Heartbeat system is an extension of the Ping system, designed to monitor the availability of an application server by sending periodic UDP packets and analyzing the responses.

3.2 Server Modifications

The server now calculates the time difference between receiving the packet and the client's timestamp, sending this information back to the client.

3.3 Client Modifications

The client sends up to 1000 heartbeat messages, detects server downtime after 3 consecutive missed responses, and provides more detailed statistics.

3.4 Key Features

- Detects server downtime after 3 consecutive missed responses
- Simulates 30% packet loss
- Calculates time difference on the server side
- Provides detailed statistics including packet loss rate

3.5 Sample Code Snippet

Here's a snippet of the modified client code:

```python
import socket
import time

# ... (rest of the code)

consecutive_misses = 0
max_misses = 3

for i in range(1000):
    start = time.time()
    message = f"{i} {start}"
    try:
        sent = mysocket.sendto(message.encode("utf-8"), server_address)
        data, server = mysocket.recvfrom(4096)
        end = time.time()
        elapsed = end - start
        print(f"RTT: {elapsed * 1000} milliseconds")
        consecutive_misses = 0
    except socket.timeout:
        print(f"#{i} Request Timed out")
        consecutive_misses += 1
        if consecutive_misses >= max_misses:
            print("Server is assumed to be down after 3 consecutive misses.")
            break
```

3.6 Mathematical Analysis of Packet Loss

In our implementation, we simulated a 30% packet loss rate. In real-world scenarios with good network conditions and correct IP addresses, the actual packet loss rate would likely be much lower. However, for this analysis, we'll use our simulated 30% loss rate.

We can model each packet transmission as a Bernoulli trial, where:

- Success (packet received) probability: $p = 0.7$
- Failure (packet lost) probability: $q = 1 - p = 0.3$

3.6.1 Step-by-Step Approach to Recurrence Relation

Let E_n be the expected number of tries to get n consecutive failures.

After getting $n-1$ consecutive failures, the next attempt can either:

- Be a failure with probability p, which would complete the sequence of n failures, or
- Be a success with probability $1-p$, which would reset the sequence and require restarting the process to get n consecutive failures.

This gives the recurrence relation:

$$E_n = p \times (E_{n-1} + 1) + (1-p) \times (E_{n-1} + E_n + 1)$$

Simplifying:

$$E_n = \frac{2}{p} E_{n-1} + \frac{1}{p}$$

3.6.2 Expected Number of Tries to Get 3 Consecutive Failures

Using the recurrence relation, we can calculate the expected number of tries to get 3 consecutive failures. For $n = 3$ and $p = 0.3$, the expected number of tries is approximately:

$$E_3 \approx 174 \text{ tries}$$

This means, on average, it will take 174 tries to observe 3 consecutive failures.

3.6.3 Probability of 3 Consecutive Failures

The probability of getting exactly 3 consecutive failures (timed-out requests) is:

$$P(3 \text{ consecutive failures}) = q^3 = 0.3^3 \approx 0.027 = 2.7\% \tag{1}$$

3.6.4 Probability in Different Sample Sizes

Now, let's calculate the probability of observing at least one instance of 3 consecutive failures in different sample sizes:

For 10 requests: The probability of not seeing 3 consecutive failures in 10 requests is:

$$P(\text{no 3 consecutive failures in 10}) = 1 - (0.7 * 0.7^7 + 0.3 * 0.7^6) \approx 0.2097 \tag{2}$$

Therefore, the probability of seeing at least one instance of 3 consecutive failures is:

$$P(\text{at least one 3 consecutive failures in 10}) \approx 1 - 0.2097 = 0.7903 = 79.03\% \tag{3}$$

For 100 requests: Using the same logic, but with a larger sample size:

$$P(\text{at least one 3 consecutive failures in 100}) \approx 1 - (1 - 0.027)^{98} \approx 0.9328 = 93.28\% \tag{4}$$

For 1000 requests: And for an even larger sample size:

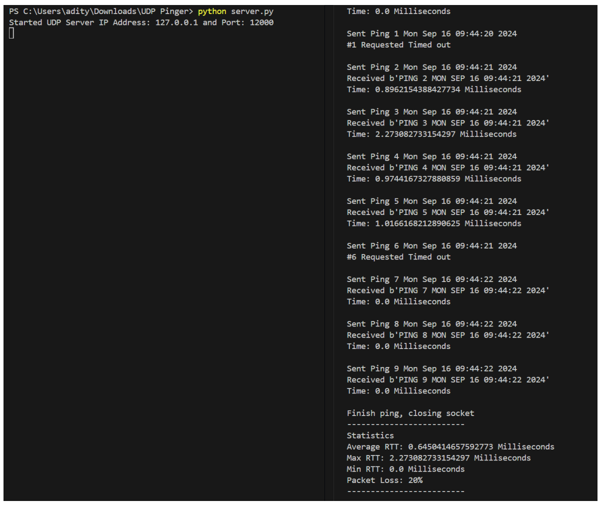
$$P(\text{at least one 3 consecutive failures in 1000}) \approx 1 - (1 - 0.027)^{998} \approx 0.9999 = 99.99\% \tag{5}$$

3.6.5 Conclusion

Based on this analysis:

- In 10 requests, we have a high chance (79.03%) of seeing at least one instance of 3 consecutive failures.
- In 100 requests, this probability increases to 93.28%.
- In 1000 requests, we're almost certain (99.99%) to observe at least one instance of 3 consecutive failures.
- On average, it takes around 174 tries to observe 3 consecutive failures.

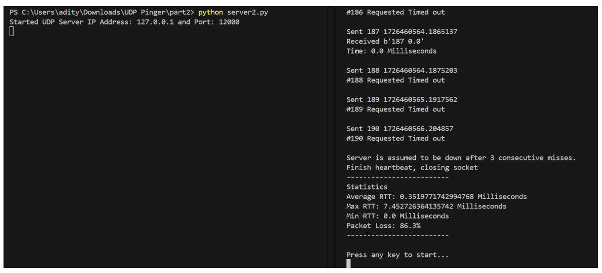
This mathematical model supports our choice of using 1000 heartbeats in our implementation, as it provides a very high probability of observing the server downtime scenario (3 consecutive missed responses).



```
PS C:\Users\adity\Downloads\UDP Pinger> python server.py
Started UDP Server IP Address: 127.0.0.1 and Port: 12000
```

```
Time: 0.0 Milliseconds

Sent Ping 1 Mon Sep 16 09:44:20 2024
#1 Requested Timed out

Sent Ping 2 Mon Sep 16 09:44:21 2024
Received b'PING 2 MON SEP 16 09:44:21 2024'
Time: 0.8962154388427734 Milliseconds

Sent Ping 3 Mon Sep 16 09:44:21 2024
Received b'PING 3 MON SEP 16 09:44:21 2024'
Time: 2.273082733154297 Milliseconds

Sent Ping 4 Mon Sep 16 09:44:21 2024
Received b'PING 4 MON SEP 16 09:44:21 2024'
Time: 0.9744167327880859 Milliseconds

Sent Ping 5 Mon Sep 16 09:44:21 2024
Received b'PING 5 MON SEP 16 09:44:21 2024'
Time: 1.0166168212890625 Milliseconds

Sent Ping 6 Mon Sep 16 09:44:21 2024
#6 Requested Timed out

Sent Ping 7 Mon Sep 16 09:44:22 2024
Received b'PING 7 MON SEP 16 09:44:22 2024'
Time: 0.0 Milliseconds

Sent Ping 8 Mon Sep 16 09:44:22 2024
Received b'PING 8 MON SEP 16 09:44:22 2024'
Time: 0.0 Milliseconds

Sent Ping 9 Mon Sep 16 09:44:22 2024
Received b'PING 9 MON SEP 16 09:44:22 2024'
Time: 0.0 Milliseconds

Finish ping, closing socket
-------------------------
Statistics
Average RTT: 0.6450414657592773 Milliseconds
Max RTT: 2.273082733154297 Milliseconds
Min RTT: 0.0 Milliseconds
Packet Loss: 20%
-------------------------
```

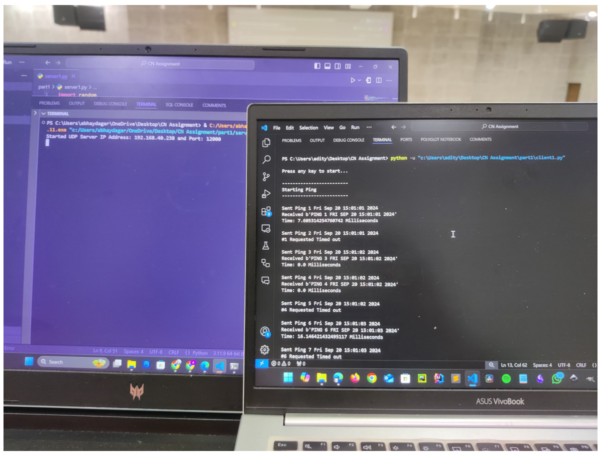
(a) Pinger running on the same system



```
PS C:\Users\adity\Downloads\UDP Pinger\part2> python server2.py
Started UDP Server IP Address: 127.0.0.1 and Port: 12000
```

```
#186 Requested Timed out

Sent 187 1726460564.1865137
Received b'187 0.0'
Time: 0.0 Milliseconds

Sent 188 1726460564.1875203
#188 Requested Timed out

Sent 189 1726460565.1917562
#189 Requested Timed out

Sent 190 1726460566.204857
#190 Requested Timed out

Server is assumed to be down after 3 consecutive misses.
Finish heartbeat, closing socket
-------------------------
Statistics
Average RTT: 0.3519771742994768 Milliseconds
Max RTT: 7.452726364135742 Milliseconds
Min RTT: 0.0 Milliseconds
Packet Loss: 86.3%
-------------------------

Press any key to start...
```

(b) HeartBeat running on the same system

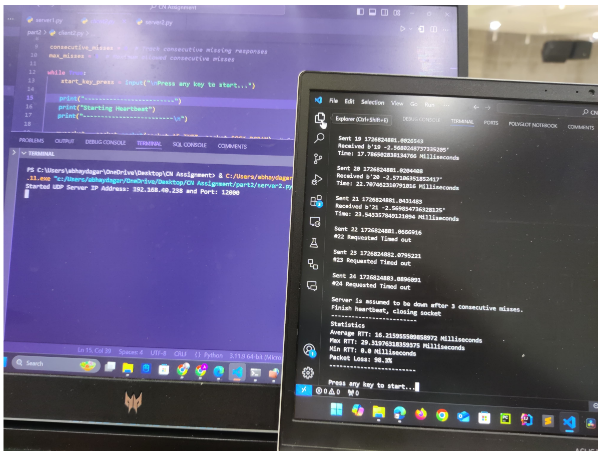


```
PS C:\Users\abhaydagar\OneDrive\Desktop\CN Assignment> & C:/Users/abha
.11.exe "c:/Users/abhaydagar/OneDrive/Desktop/CN Assignment/part1/serv
Started UDP Server IP Address: 192.168.40.238 and Port: 12000
```

```
PS C:\Users\adity\Desktop\CN Assignment> python -u "c:\Users\adity\Desktop\CN Assignment\part1\client1.py"

Press any key to start...

-------------------------
Starting Ping
-------------------------

Sent Ping 1 Fri Sep 20 15:01:01 2024
Received b'PING 1 FRI SEP 20 15:01:01 2024'
Time: 7.605314254760742 Milliseconds

Sent Ping 2 Fri Sep 20 15:01:01 2024
#1 Requested Timed out

Sent Ping 3 Fri Sep 20 15:01:02 2024
Received b'PING 3 FRI SEP 20 15:01:02 2024'
Time: 0.0 Milliseconds

Sent Ping 4 Fri Sep 20 15:01:02 2024
Received b'PING 4 FRI SEP 20 15:01:02 2024'
Time: 0.0 Milliseconds

Sent Ping 5 Fri Sep 20 15:01:02 2024
#4 Requested Timed out

Sent Ping 6 Fri Sep 20 15:01:03 2024
Received b'PING 6 FRI SEP 20 15:01:03 2024'
Time: 16.146421432495117 Milliseconds

Sent Ping 7 Fri Sep 20 15:01:03 2024
#6 Requested Timed out
```

CN Assignment
server1.py
part1 > server1.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE COMMENTS
File Edit Selection View Go Run
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POLYGLOT NOTEBOOK COMMENTS
Ln 13, Col 62 Spaces: 4 UTF-8 CRLF
ASUS VivoBook

(c) Pinger(Part1) running on different systems



```
consecutive_misses = 0  # Track consecutive missing responses
max_misses = 3  # Maximum allowed consecutive misses

while True:
    start_key_press = input("\nPress any key to start...")

    print("-------------------------")
    print("Starting Heartbeat")
    print("-------------------------\n")
```

```
PS C:\Users\abhaydagar\OneDrive\Desktop\CN Assignment> & C:/Users/abhaydagar
.11.exe "c:/Users/abhaydagar/OneDrive/Desktop/CN Assignment/part2/server2.py
Started UDP Server IP Address: 192.168.40.238 and Port: 12000
```

```
Sent 19 1726824881.0026543
Received b'19 -2.5680248737335205'
Time: 17.786502838134766 Milliseconds

Sent 20 1726824881.0204408
Received b'20 -2.57106351852417'
Time: 22.707462310791016 Milliseconds

Sent 21 1726824881.0431483
Received b'21 -2.569854736328125'
Time: 23.543357849121094 Milliseconds

Sent 22 1726824881.0666916
#22 Requested Timed out

Sent 23 1726824882.0795221
#23 Requested Timed out

Sent 24 1726824883.0896091
#24 Requested Timed out

Server is assumed to be down after 3 consecutive misses.
Finish heartbeat, closing socket
-------------------------
Statistics
Average RTT: 16.215955509858972 Milliseconds
Max RTT: 29.31976318359375 Milliseconds
Min RTT: 0.0 Milliseconds
Packet Loss: 98.3%
-------------------------

Press any key to start...
```

CN Assignment
server1.py
client2.py
server2.py
part2 > client2.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE COMMENTS
File Edit Selection View Go Run
Explorer (Ctrl+Shift+E)
DEBUG CONSOLE TERMINAL PORTS POLYGLOT NOTEBOOK COMMENTS

(d) HeartBeat(Part2) running on different systems

Figure 1: Screenshots of the UDP Ping and Heartbeat systems running on same and different systems

4 Screenshots

5 Conclusion

Both the UDP Ping and Heartbeat systems demonstrate important concepts in network programming, including handling unreliable connections, measuring network performance, and monitoring server availability. These implementations provide a foundation for understanding more complex network applications and protocols. The mathematical analysis of the Heartbeat system further illustrates the probabilistic nature of network reliability and justifies our design choices.