CS 536 Spring 2024

Lab 2: Basic Socket Programming and Lightweight Network Communication

Alireza Lotfi

February 14, 2024

Contents

1 Problem 2

1.1 Explanation of pingcheck.sh

The bash script is used to measure the round-trip time (RTT) of UDP packets. The script takes three arguments: *server's IP, server's port, and client's IP*. After initializing the server in another machine, the script executes the client code (`pingc.bin`) 10 times, each time sending UDP packets to the specified server IP and port. It then parses the output to extract the RTT values and saves them to a file named `rtt_results.txt`. Additionally, the script uses the legacy `/bin/ping` command to measure RTTs directly over IP, without involving application processes, and appends these results to the same file.

Note: do not forget to give proper permissions to the file (chmod +x pingcheck.sh)

```
#!/bin/bash
# Check if all required arguments are provided
if [ "$#" -ne 3 ]; then
    echo "Usage: $0 <server_ip> <server_port> <client_ip>"
    exit 1
fi
# Assign command-line arguments to variables
SERVER_IP=$1
SERVER_PORT=$2
CLIENT_IP=$3
# Run the client code 10 times and save the results
echo "# pingc.bin results" >> rtt_results.txt
for ((i=1; i<=10; i++)); do
    ../v1/pingc.bin "$SERVER_IP" "$SERVER_PORT" "$CLIENT_IP" | grep -oP '
   RTT: \K[0-9.]+ milliseconds' >> rtt_results.txt
    sleep 1
done
# Ping the server IP and save results in the same file
echo "# /bin/ping results" >> rtt_results.txt
/bin/ping -c 10 "$SERVER_IP" >> rtt_results.txt
echo "pingc.bin and /bin/ping results saved in rtt_results.txt"
```

1.2 Part 1 (linux PCs in HAAS G50)

Table 1 presents Round-Trip Time (RTT) results from pinging between two Linux PCs, amber10 (client) and amber05 (server). Two methods of pinging were used: `pingc/pings` and `/bin/ping`, each providing different RTT measurements.

- `pingc/pings` consistently yielded lower RTT values compared to `/bin/ping`.
- Both methods exhibited variability in RTT measurements, as indicated by the Mean Deviation (RTT mdev) values.

Table 1: RTT Results (client: amber10, server: amber05)

Iteration	**pingc/pings RTT (ms)**	**/bin/ping RTT (ms)**
1	0.497	0.323
2	0.565	0.597
3	0.540	0.591
4	0.525	0.735
5	0.755	0.641
6	0.749	0.644
7	0.545	0.840
8	0.740	0.841
9	0.450	0.842
10	0.402	0.662
RTT min (ms)	**0.402**	**0.323**
RTT avg (ms)	**0.561**	**0.671**
RTT max (ms)	**0.755**	**0.842**
RTT mdev (ms)	**0.074**	**0.150**

1.3 Part 2 - Bonus (one PC in G050 and one outside)

The table presents Round-Trip Time (RTT) results from pinging between two hosts, data.cs.purdue.edu (client) and amber05.cs.purdue.edu (server). Two methods of pinging were used: `pingc/pings` and `/bin/ping`, each providing different RTT measurements.

- `pingc/pings` consistently yielded higher RTT values compared to `/bin/ping`.
- Both methods exhibited variability in RTT measurements, as indicated by the Mean Deviation (RTT mdev) values.

Table 2: RTT Results (client: data.cs.purdue.edu, server: amber05.cs.purdue.edu)

Iteration	**pingc.bin RTT (ms)**	**/bin/ping RTT (ms)**
1	0.708	0.416
2	1.920	0.536
3	0.552	0.431
4	0.613	0.421
5	0.594	0.415
6	0.584	0.237
7	0.690	0.200
8	0.440	0.291
9	0.619	0.244
10	0.617	0.213
RTT min (ms)	**0.440**	**0.200**
RTT avg (ms)	**0.700**	**0.340**
RTT max (ms)	**1.920**	**0.536**
RTT mdev (ms)	**0.251**	**0.110**

1.4 Part 3 (different Locations)

1.4.1 Explanation of calculate_distance.sh

1. **calculate_distance_km():** Estimates network distance in km between Purdue University and a destination website using RTT, and converts it to miles.

```
calculate_distance_km() {
    speed_of_light=299792
    one_way_latency=$(echo "scale=6; $1 / 2.0" | bc)
    printf "Formula: \nDistance = Speed of Light * Time \nTime = RTT
   /2 = %s \nSpeed of Light = %s km/s\n" "$one_way_latency" "
   $speed_of_light"
    distance_km=$(echo "scale=3; $speed_of_light * $one_way_latency /
    1000.0" | bc)
    distance_miles=$(km_to_miles $distance_km)
    printf "Estimated distance: %.3f km or %s\n" "$distance_km" "
   $distance_miles"
}
```

2. **km_to_miles():** Converts distances from km to miles using the conversion factor (1 km $\approx$ 0.621371 miles).

3. **calculate_real_distance():** Computes actual geographic distance between Purdue University and a destination website using the Haversine formula.

4. **ping_and_calculate_distances():** Pings a destination website, calculates estimated network distance, and real geographic distance.

5. **Redirecting Output:** Redirects output to "ping_results.txt" using `tee` command.

6. **Main Execution:** Calls `ping_and_calculate_distances()` for each destination.

7. **Output:** Displays information about each destination, including ping results and distances.

1.4.2 Target locations

- **East Coast (United States):**
  - Domain Name: `www.mit.edu` (Massachusetts Institute of Technology)
  - Location: Cambridge, Massachusetts, United States (42.3601° N, 71.0942° W)
- **West Coast (United States):**
  - Domain Name: `www.stanford.edu` (Stanford University)
  - Location: Stanford, California, United States (37.4275° N, 122.1697° W)
- **Eastern Asia:**
  - Domain Name: `www.kyoto-u.ac.jp` (Kyoto University)
  - Location: Kyoto, Japan (35.0116° N, 135.7681° E)
- **Europe:**
  - Domain Name: `www.cam.ac.uk` (University of Cambridge)
  - Location: Cambridge, United Kingdom (52.2053° N, 0.1218° E)
- **Different Continent or Distant Place:**
  - Domain Name: `www.nus.edu.sg` (National University of Singapore)
  - Location: Singapore (1.2956° N, 103.7761° E)

1.4.3 Calculating Distances for Each Destination

Formula:

$$\text{Distance} = \text{Speed of Light} \times \text{Time}$$

$$\text{Time} = \frac{\text{RTT}}{2}$$

$$\text{Speed of Light} = 299792 \text{ km/s}$$

Note: we will use the average RTT out of 10 attemps for calculating the distance

- Pinging `www.mit.edu` ...
  - Round-trip min/avg/max/stddev = 38.227/42.239/45.906/2.689 ms
  - Estimated distance: 6331.457 km or 3934.184 miles
  - Real distance from Purdue to destination: 1327.294 km or 824.742 miles
- Pinging `www.stanford.edu` ...
  - Round-trip min/avg/max/stddev = 30.022/33.397/34.944/1.256 ms
  - Estimated distance: 5006.076 km or 3110.630 miles
  - Real distance from Purdue to destination: 2964.458 km or 1842.028 miles
- Pinging `www.kyoto-u.ac.jp` ...
  - Round-trip min/avg/max/stddev = 33.899/40.533/56.907/5.747 ms
  - Estimated distance: 6075.734 km or 3775.285 miles
  - Real distance from Purdue to destination: 8092.801 km or 5028.632 miles
- Pinging `www.cam.ac.uk` ...

  - Round-trip min/avg/max/stddev = 113.104/120.678/149.494/9.934 ms
  - Estimated distance: 18089.149 km or 11240.073 miles
  - Real distance from Purdue to destination: 5716.315 km or 3551.953 miles

- Pinging `www.nus.edu.sg` ...
  - Round-trip min/avg/max/stddev = 13.908/18.383/29.111/3.752 ms
  - Estimated distance: 2755.538 km or 1712.211 miles
  - Real distance from Purdue to destination: 9551.830 km or 5935.230 miles

2 Problem 3

2.1 Client termination issue

To address the issue of the client hanging indefinitely if the last data packet from the server is lost, a timeout mechanism will be implemented in the client code. This mechanism will allow the client to wait for a certain period for the expected data packet from the server and handle the situation appropriately if the packet does not arrive within the specified timeout duration.

2.1.1 Implementation Details

1. Define a timeout value in milliseconds.
2. Use `setsockopt` to set a receive timeout on the socket.
3. Utilize `select` or `poll` system calls to wait for data with a timeout.
4. Check for the timeout condition and handle it appropriately in the client code.

The client will set a timeout value for receiving data packets from the server. If the expected data packet is not received within the specified timeout duration, the client will consider the last data packet as lost and take appropriate action, preventing the client from hanging indefinitely.

2.2 Performance evaluation

The performance evaluation conducted with varying payload sizes (1000 bytes and 1400 bytes) across different file sizes (2560, 25600, and 256000 bytes) revealed notable differences in completion time and transfer speed, as summarized in Table 3. The evaluations were

Table 3: Performance Evaluation with Different Payload Sizes

File Size (bytes)	1000 Bytes Payload			1400 Bytes Payload		
	Completion Time (ms)	Transfer Speed (bps)	Payload Packets	Completion Time (ms)	Transfer Speed (bps)	Payload Packets
2560	1	20.48M	3	1	20.48M	2
25600	2	102.4M	26	2	102.4M	19
256000	6	341.33M	256	5	409.6M	183

performed using the client-server setup with the client being amber10 and the server being amber05.

When using a payload size of 1000 bytes, the completion time varied from 1 millisecond for the smallest file size of 2560 bytes to 6 milliseconds for the largest file size of 256000 bytes. Correspondingly, the transfer speed ranged from approximately 20.48 Mbps (megabits per second) to 341.33 Mbps across the same file sizes. Interestingly, the number of payload packets transmitted increased with the file size, indicating a higher overhead for larger files due to increased number of packets.

On the other hand, utilizing a larger payload size of 1400 bytes resulted in slightly improved performance metrics. The completion time decreased slightly compared to the 1000 bytes payload size, with completion times ranging from 1 millisecond to 5 milliseconds across the same file sizes. The transfer speed also exhibited an improvement, with speeds ranging from approximately 20.48 Mbps to 409.6 Mbps. Notably, the number of payload packets transmitted decreased for all file sizes compared to the 1000 bytes payload size, indicating reduced overhead and more efficient transmission with larger payload sizes.

Overall, increasing the payload size from 1000 bytes to 1400 bytes demonstrated enhancements in completion time and transfer speed across all file sizes tested. This improvement suggests that optimizing payload size can contribute to more efficient data transmission, particularly for larger files, by reducing the overhead associated with number of packets and transmission.