CS 342: Computer Networks Lab

Client-Server programming using both TCP and UDP sockets

Contents

Chapter 1

Client-Server Programming using TCP and UDP

1.1 Introduction

In this assignment, we implemented two C programs, a `server` and a `client`, to communicate using both TCP and UDP sockets. The protocol had two stages:

1. **TCP Handshake:** The client connects to the server and requests a UDP port. The server replies with a port number.
2. **UDP Communication:** The client sends a data message to the server using the negotiated UDP port. The server acknowledges with a response.

The server is concurrent and supports two scheduling policies: First-Come-First-Serve (FCFS) and Round-Robin (RR).

1.2 Working of source files

1.2.1 Client Program (client.c)

The client performs the following steps:

1. Creates a TCP socket and connects to the server using the IP address and port specified.
2. Sends a **Type-1** TCP message (Phase1Message1) to the server.
3. Receives a **Type-2** TCP message (Phase1Message2) from the server, which contains a dynamically assigned UDP port number.
4. Terminates the TCP connection.
5. Creates a UDP socket and sends a **Type-3** UDP message (Phase2Message1).
6. Waits for the server's **Type-4** UDP acknowledgement (Phase2Message2).

1.2.2 Server Program (server.c)

The server supports multi-threaded request handling and two scheduling modes:

- **FCFS:** Each request is processed completely in the order of arrival.
- **RR:** Each request is processed in slices (quantum), cycling through requests for fairness.

The server performs the following steps:

1. Listens for incoming TCP connections on a specified port.
2. Accepts client connections and enqueues them in a request queue.
3. Worker threads dequeue requests and process them based on the scheduling policy.
4. Generates a dynamic UDP port for each client and communicates it via TCP.
5. After simulated processing, performs UDP exchange with the client.

1.3 System Setup and Steps

1.3.1 Server

- Compiled with `gcc -pthread server.c -o server`.
- Run with command: `./server <Server TCP Port> <fcfs|rr>`
- Accepts TCP connections, negotiates UDP port, and queues requests.
- Worker threads process requests based on chosen scheduling policy.

1.3.2 Client

- Compiled with `gcc client.c -o client`.
- Run with command: `./client <Server IP Address> <Server TCP Port>`
- Connects to the server via TCP, requests UDP port, then switches to UDP.
- Sends data message and waits for server's acknowledgment.

1.4 Working of Scheduling Policies

1.4.1 Round Robin

- Each new client request is initialized with a fixed number of work units.
- Workers process exactly 1 unit per turn, then requeue the client.
- Clients are served cyclically until completion.
- Requests are finished fairly without starvation.

1.4.2 First Come First Server

- Clients are enqueued in arrival order.
- Workers process them in order without skipping.
- Clients are processed concurrently, but dequeued in arrival order.
- Requests stay with their thread until completion.

1.5 Screenshots of Execution

```
aeyaz-adil@aeyaz-adil-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/Downloads/A2$ ./clien
t 127.0.0.1 8080
Created Socket with fd : 3
Connection established with 127.0.0.1 at port 8080
 Sending Type-1 TCP msg : |1|36|'Request Message from Client over TCP'| (stage:
Phase1Message1)
Waiting to receive Phase1Message2...
Received Type-2 Message: |2|5|'43052'| (stage: Phase1Message2)
Terminating TCP connection 3 to server 127.0.0.1 port 8080
Creating UDP socket
Created UDP socket with fd : 3
Sending Type-3 UDP msg : |3|33|'Data Message from Client over UDP'| (stage: Phas
e2Message1)
Waiting to receive Phase2Message2...
Receieved UDP packet from 127.0.0.1  Client Port: 43052
Received Type-4 Message: |4|23|'ACK message from server'| (stage: Phase2Message2
)
Terminating UDP connection 3 to server 127.0.0.1 port 43052
Communication done. Closing client
```

(a) Client side two stage communication protocol

```
aeyaz-adil@aeyaz-adil-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/Downloads/A2$ ./serve
r 8080
Waiting for connection...
Connection established: Client IP: 127.0.0.1  Client Port: 39716
Waiting to receive Phase1Message1...
Received Type-1 Message: |1|36|'Request Message from Client over TCP'| (stage: P
hase1Message1)
Created UDP socket 3 (child-id 9068)
 Sending Type-2 TCP msg : |2|5|'43052'| (stage: Phase1Message2)
Closing TCP socket 4 at port 8080 with client (127.0.0.1)
Waiting to receive Phase2Message1...
Received Type-3 Message: |3|33|'Data Message from Client over UDP'| (stage: Phas
e2Message1)
Sending Type-4 UDP msg : |4|23|'ACK message from server'| (stage: Phase2Message2
)
Closed UDP socket 3 (child-id 9068)
Communication Done. Killed child connection process (9068).
```

(b) Server side two stage communication protocol

```
aeyaz-adil@aeyaz-adil-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/Downloads/A2 (3)$ ./server 8080 rr
Server listening on port 8080 (policy: RR)
Accepted connection: Client IP: 127.0.0.1  Client Port: 54084  fd=4
Waiting to receive Phase1Message1 (TCP) from client fd 4...
Received Type-1 Message: |1|36|'Request Message from Client over TCP'| (stage: Phase1Message1)
Created UDP socket 6 (for request handler)
 Sending Type-2 TCP msg : |2|5|'36895'| (stage: Phase1Message2)
Closing TCP socket 4 with client
RR: initialized request udp 36895 units_remaining=6
Worker 126564597495488: RR processing udp 36895, doing unit (before=6)
Worker 126564597495488: RR processing udp 36895, doing unit (before=5)
Accepted connection: Client IP: 127.0.0.1  Client Port: 54088  fd=5
Waiting to receive Phase1Message1 (TCP) from client fd 5...
Received Type-1 Message: |1|36|'Request Message from Client over TCP'| (stage: Phase1Message1)
Created UDP socket 7 (for request handler)
 Sending Type-2 TCP msg : |2|5|'48280'| (stage: Phase1Message2)
Closing TCP socket 5 with client
RR: initialized request udp 48280 units_remaining=6
Worker 126564589102784: RR processing udp 48280, doing unit (before=6)
Worker 126564597495488: RR processed udp 36895, units_remaining=4
RR: requeued udp 36895 for further processing
Worker 126564597495488: RR processing udp 36895, doing unit (before=4)
Worker 126564589102784: RR processing udp 48280, doing unit (before=5)
Worker 126564597495488: RR processing udp 36895, doing unit (before=3)
Worker 126564589102784: RR processed udp 48280, units_remaining=4
RR: requeued udp 48280 for further processing
Worker 126564589102784: RR processing udp 48280, doing unit (before=4)
Worker 126564597495488: RR processed udp 36895, units_remaining=2
RR: requeued udp 36895 for further processing
Worker 126564597495488: RR processing udp 36895, doing unit (before=2)
Worker 126564589102784: RR processing udp 48280, doing unit (before=3)
Worker 126564597495488: RR processing udp 36895, doing unit (before=1)
Worker 126564589102784: RR processed udp 48280, units_remaining=2
RR: requeued udp 48280 for further processing
Worker 126564589102784: RR processing udp 48280, doing unit (before=2)
Worker 126564597495488: RR processed udp 36895, units_remaining=0
Waiting to receive Phase2Message1 on udp port 36895...
Received Type-3 Message: |3|33|'Data Message from Client over UDP'| (stage: Phase2Message1)
Sending Type-4 UDP msg : |4|23|'ACK message from server'| (stage: Phase2Message2)
Closed UDP socket 6
Worker 126564597495488: RR finished udp 36895
Worker 126564589102784: RR processing udp 48280, doing unit (before=1)
Worker 126564589102784: RR processed udp 48280, units_remaining=0
Waiting to receive Phase2Message1 on udp port 48280...
Received Type-3 Message: |3|33|'Data Message from Client over UDP'| (stage: Phase2Message1)
Sending Type-4 UDP msg : |4|23|'ACK message from server'| (stage: Phase2Message2)
Closed UDP socket 7
```

(c) Round Robin (RR) policy handling by server for multiple clients

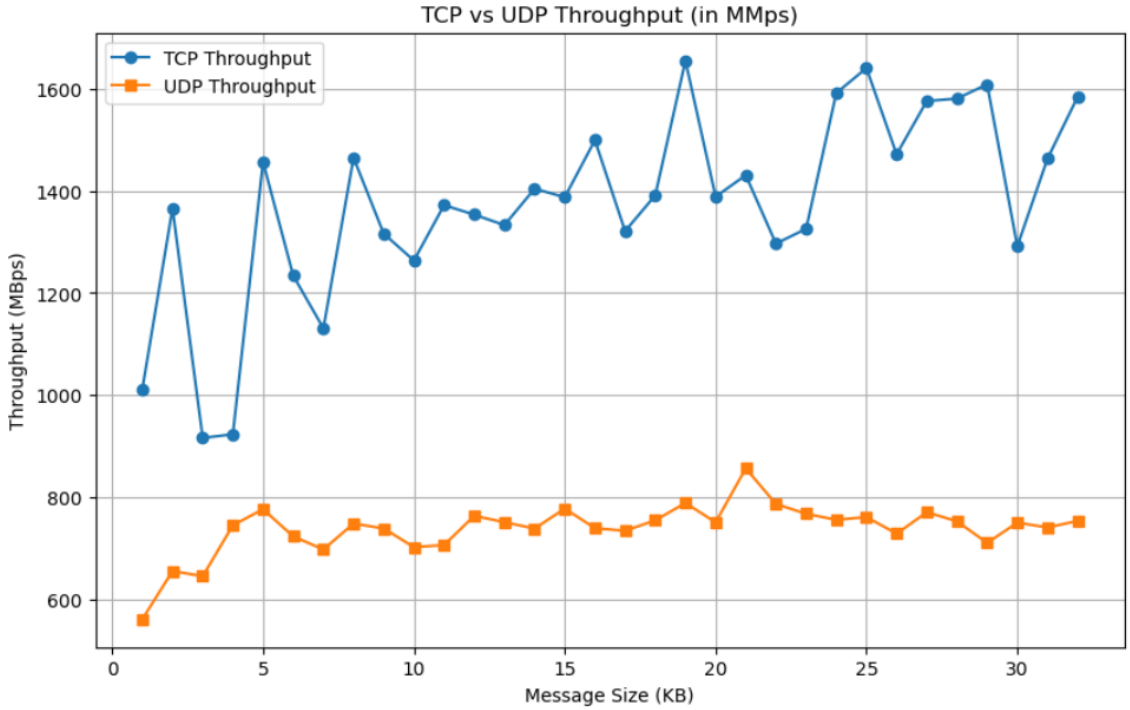
```
bash: export: `/home/aeyaz-adil/.bashrc': not a valid identifier
aeyaz-adil@aeyaz-adil-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/Downloads/A2 (3)$ gcc server.c -o server -pthread
aeyaz-adil@aeyaz-adil-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/Downloads/A2 (3)$ gcc client.c -o client
aeyaz-adil@aeyaz-adil-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/Downloads/A2 (3)$ ./server 8080 fcfs
Server listening on port 8080 (policy: FCFS)
Accepted connection: Client IP: 127.0.0.1  Client Port: 40444  fd=4
Worker 131608568919744: FCFS processing client fd 4
Waiting to receive Phase1Message1 (TCP) from client fd 4...
Received Type-1 Message: |1|36|'Request Message from Client over TCP'| (stage: Phase1Message1)
Created UDP socket 6 (for request handler)
 Sending Type-2 TCP msg : |2|5|'56082'| (stage: Phase1Message2)
Closing TCP socket 4 with client
Worker 131608568919744: FCFS working on client 56082 unit 1/6
Worker 131608568919744: FCFS working on client 56082 unit 2/6
Accepted connection: Client IP: 127.0.0.1  Client Port: 40454  fd=5
Worker 131608543741632: FCFS processing client fd 5
Waiting to receive Phase1Message1 (TCP) from client fd 5...
Received Type-1 Message: |1|36|'Request Message from Client over TCP'| (stage: Phase1Message1)
Created UDP socket 7 (for request handler)
 Sending Type-2 TCP msg : |2|5|'44135'| (stage: Phase1Message2)
Closing TCP socket 5 with client
Worker 131608543741632: FCFS working on client 44135 unit 1/6
Worker 131608568919744: FCFS working on client 56082 unit 3/6
Worker 131608543741632: FCFS working on client 44135 unit 2/6
Worker 131608568919744: FCFS working on client 56082 unit 4/6
Worker 131608543741632: FCFS working on client 44135 unit 3/6
Worker 131608568919744: FCFS working on client 56082 unit 5/6
Worker 131608543741632: FCFS working on client 44135 unit 4/6
Worker 131608568919744: FCFS working on client 56082 unit 6/6
Worker 131608543741632: FCFS working on client 44135 unit 5/6
Waiting to receive Phase2Message1 on udp port 56082...
Received Type-3 Message: |3|33|'Data Message from Client over UDP'| (stage: Phase2Message1)
Sending Type-4 UDP msg : |4|23|'ACK message from server'| (stage: Phase2Message2)
Closed UDP socket 6
Worker 131608568919744: Finished FCFS client (udp 56082)
Worker 131608543741632: FCFS working on client 44135 unit 6/6
Waiting to receive Phase2Message1 on udp port 44135...
Received Type-3 Message: |3|33|'Data Message from Client over UDP'| (stage: Phase2Message1)
Sending Type-4 UDP msg : |4|23|'ACK message from server'| (stage: Phase2Message2)
Closed UDP socket 7
Worker 131608543741632: Finished FCFS client (udp 44135)
```

(d) First-Come-First-Served (FCFS) policy handling by server for multiple clients

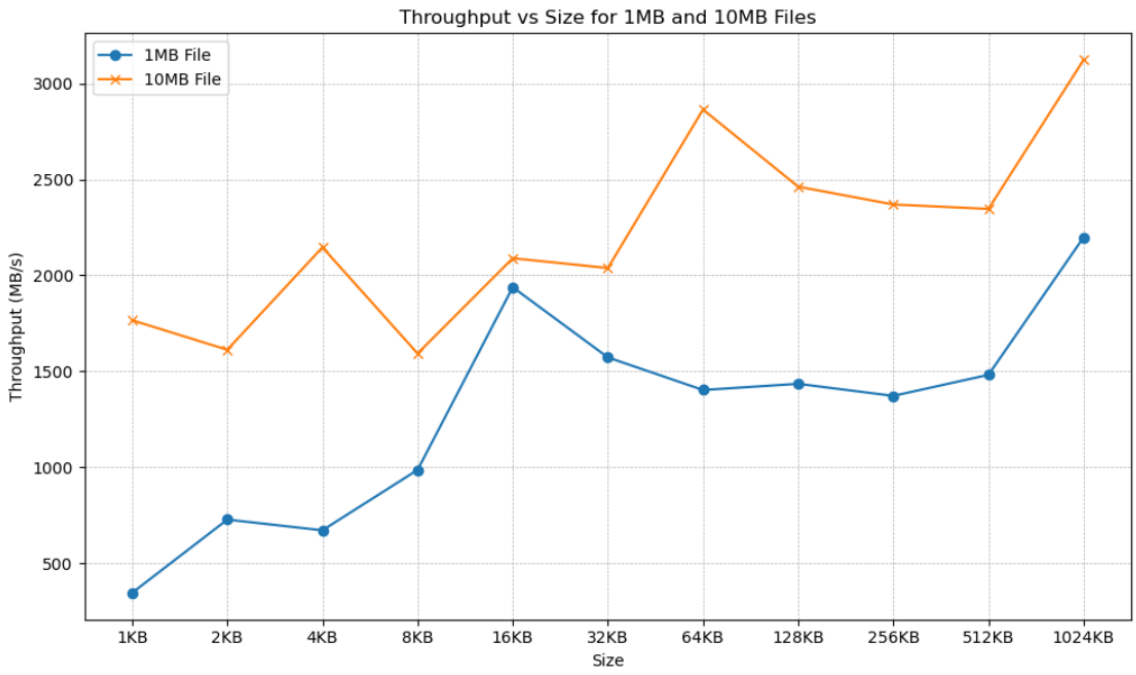
Figure 1.1: Working Client-Server Interactions

1.6 Performance Measurement

1.6.1 TCP vs UDP (1 KB – 32 KB)



1.6.2 TCP with Larger Transfers



1.7 Observations

TCP vs UDP (1 KB – 32 KB)

- TCP consistently achieves higher throughput compared to UDP across all message sizes.
- TCP throughput fluctuates but generally stays in the range of 1200–1600 Mbps.
- UDP throughput remains much lower, around 600–800 Mbps.
- Despite TCP's overhead due to reliability mechanisms, it sustains higher data transfer rates in this experimental setup.
- UDP, being connectionless and lacking flow/congestion control, shows lower and more stable throughput values.

TCP with Larger Transfers

- For the 1 MB file, throughput initially increases with message size and then stabilizes with minor fluctuations.
- For the 10 MB file, throughput consistently achieves significantly higher values, often exceeding 2000 Mbps and sometimes reaching above 3000 Mbps.
- Larger file transfers make better use of TCP's congestion window growth and pipeline efficiency.
- As a result, much higher throughput is achieved compared to smaller transfers.

1.8 Challenges Faced

During the implementation of the client–server communication system, several challenges were encountered:

- **Dual Protocol Communication:** The system required establishing a TCP connection for initial negotiation and then transitioning to UDP for data transfer. Ensuring correct port handoff between TCP and UDP was challenging, as the server dynamically generated UDP ports and the client had to correctly interpret and use them.
- **Concurrency and Scheduling:** The server supported multiple clients concurrently using worker threads. Implementing both First-Come-First-Serve (FCFS) and Round-Robin (RR) scheduling policies introduced complexity in maintaining fairness, preventing starvation, and ensuring thread-safe access to the request queue.
- **Synchronization and Thread Safety:** Shared data structures such as the request queue required synchronization mechanisms (mutex locks and condition variables). Incorrect handling could lead to race conditions, deadlocks, or missed signals, making careful design and testing necessary.

- **Graceful Shutdown:** Handling signals (e.g., `SIGINT`) and ensuring the server could shut down gracefully without leaving sockets open or worker threads hanging was non-trivial, especially when multiple clients were mid-communication.
- **Error Handling and Robustness:** Network programming is prone to partial transmissions, dropped connections, and client crashes. Proper handling of errors in socket creation, binding, sending, and receiving was essential to avoid resource leaks or inconsistent states.
- **Blocking vs Non-blocking Behavior:** Since functions like `recv()` are blocking by default, careful coordination was required to avoid deadlocks between clients and the server, particularly during the TCP handshake and UDP response phases.