Client-Server Implementation

In C++ with Boost.ASIO

Miguel Garcia
https://github.com/MiguelGarcia21/Client-Server-imp

Contents

1. Introduction

1.1 Purpose

This document describes the functional and non-functional requirements for a lightweight, modular client-server system using asynchronous I/O (ASIO). The system will support messaging, latency testing, client identification, and real-time broadcast.

1.2 Scope

The application will allow clients to:
- Ping the server to measure latency
- Send broadcast messages
- Be uniquely identified by the server
- Log activity and manage connections with thread-safe queues

2. Overall description

2.1 Domain Model – Concept

Message	**Ping**	**Broadcast**	**Connection**	**Identification**
Create	Timestamp register	Multicast	TCP connection	Assign Client ID
Send	Latency calculate	Broadcast Receive	Client validation	Log actions
Receive	Echo ping	Exclude Sender	Disconnect Handle	Map endpoints
Parse	Ping sender identification	Thread-safe distribution	Async IO	
Queue				

Message	**Description**
Create	Build packets
Send	Transamit data over network
Receive	Accept incoming data
Parse	Decode message content
Queue	Message buffer

Ping	**Description**
Timestamp register	Tag with send time
Latency calculate	Measure round trip
Echo ping	Server bounces ping back
Ping sender identification	Track sender ID

Broadcast	**Description**
Multicast	Sends to all clients

Connection	**Description**
TCP connection	Link client-server

Broadcast Receive	Clients get broadcasts
Exclude Sender	Skip original sender
Thread-safe distribution	Avoid data races

Client validation	Authenticate clients
Disconnect Handle	Correct termination
Async IO	Non-blocking operation

Identification	**Description**
Assign Client ID	Assign Client ID
Log actions	Log actions
Map endpoints	Map endpoints

2.2 User Roles & Activities

User	Activities
Client	Connect to server
	Send ping requests
	Broadcast messages
	Receive broadcast messages
Server	Accept and validate client connections
	Log client pings and IDs
	Manage broadcast and message routing

3. System Features

Feature	**Input**	**Expected Output**
Ping Server	Ping request	Echo ping + RTT Calculation
Broadcast Message	Message from client	All clients (except sender) receive message
Client Connect	IP/Port	Server assigns ID, logs, confirms connection
Error Handling	Client disconnect	Server waits for new connections gracefully

4. SWOT Analysis

5. Strengths	
Templating	Lightweight, modular design
TSQueue	Thread-safe message queue prevents data races
ASYNC	Cross-platform async networking efficiently

Weaknesses	
No encryption	Raw messages are vulnerable to interception
Minimal error recovery	Recovery plan
Scalability	Hardware limitation for testing

Opportunities	
SSL Security	Enhance security
Message compression	Memory management and efficiency
Database integration	Persistent event logging

Threats	
Network latency	Latency spikes distort ping measirements
Message validation	Wrong message structure could crash clients/servers
Competitive protocols	Websockets, gRPC

6. Interface Requirements

Function	Operational	Tecnical	Metric	Status
Ping Larency	Slow ping responses	TSQueues contentions	Round-trip time	Added to backlog
Client identificatio n	Duplicate Ids cause message routing error	ID Assignment logic	Duplicate Ids occurrance	Added to backlog
Message Reliability	Messages could fail to send or receive	async_write error handling	Message drop rate %	Not implemented
Server Performance	Server becomes unresponsive underload	ASIO thread workload	CPU usage spikes % over time	Not implemented

7. Backlog, Development phases and Testing

Story #	Story	Tasks	Test Status
Story 1	As a developer, I want to set up the core infrastructure for client-server communication so that data can be exchanged asynchronously.	Set up `asio::io_context` and use `asio::ip::tcp::acceptor` to listen for incoming connections (server). Implement `connection<T>` class to manage socket logic.	Test Result: `Story1_CoreNetworking`: Passed ☑

		Use `asio::ip::tcp::resolver` and `asio::async_connect` to connect client to server.	
		Run the ASIO context in dedicated threads on both server and client.	
		Define `message<T>` and `owned_message<T>` to structure messages.	
		Create a thread-safe queue `tsqueue<T>` to manage concurrent message processing.	
Story 2	As a server, I want to assign unique IDs to connected clients so they can be identified in the system.	Maintain an ID counter `nIDCounter` in the server class.	Test Result: `Story2_ClientIDManagement`: Passed ☑
		Assign IDs during the call to `ConnectToClient(uid)` in each new connection.	
		Expose the ID through `GetID()`.	
Story 3	As a client, I want to ping the server and receive a timestamp response so I can measure latency.	Create and send a message with ID `ServerPing` containing a timestamp (`std::chrono::system_clock::time_point`).	Test Result: `Story3_Ping`: Passed ☑
		Echo the same message back from the server to the originating client.	
		Compute round-trip time on the client and display the result.	
Story 4	As a client, I want to send a broadcast message so all connected clients (except me) receive it.	Send a `MessageAll` message from the client to the server.	Test Result: `Story4_Broadcast`: Passed ☑
		The server receives the message and creates a `ServerMessage` that includes the sender's client ID.	
		The server sends this message to all clients except the originator.	

		Clients print the received broadcast message with the sender’s ID.	
Story 5	As a system, I want to validate client connection states to avoid communication failures.	Use `IsConnected()` to validate client state before sending messages.	Test Result: `Story5_ConnectionValidation`: Passed ☑
		If a client is disconnected, remove it from the list (`m_deqConnections`).	
		Always restart the acceptor using `WaitForClientConnection()` to accept new connections.	
Story 6	As a developer, I want to log system activity so that I can monitor performance and debug issues.	Print relevant events to `std::cout`: connection established, disconnected, messages received, and errors.	Test Result: `Story6_Logging`: Passed ☑
		Include client IDs in logs to make debugging easier.	