TCP/IP Status Monitoring System

(Mindox Techno)

by: Charaka Gunasinghe | submitted on: December 17th, 2025

Introduction

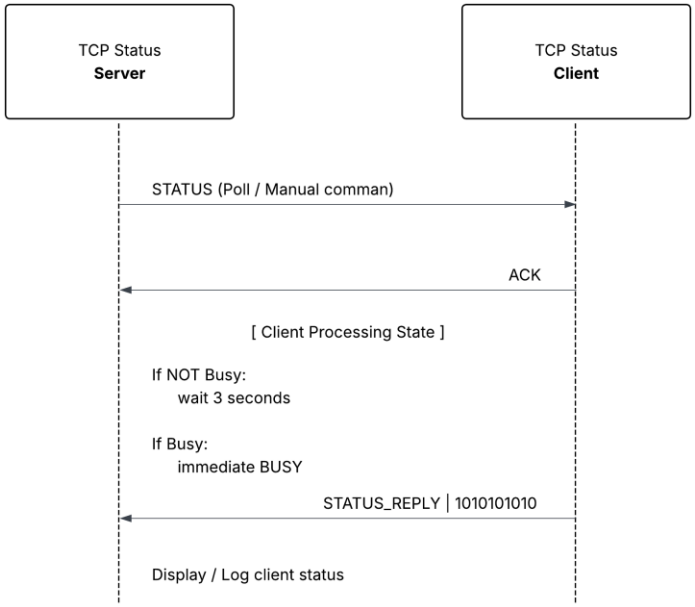
This assignment involves developing a TCP/IP server and client system in C#. The server can handle multiple clients concurrently, periodically polling them for status, and process commands. The client connects to the server, acknowledges received commands, simulates hardware processing with a delay, and responds with status or a busy message if already processing. The implementation demonstrates asynchronous programming, concurrency management, and robust error handling in a networked environment.

GitHub repositories

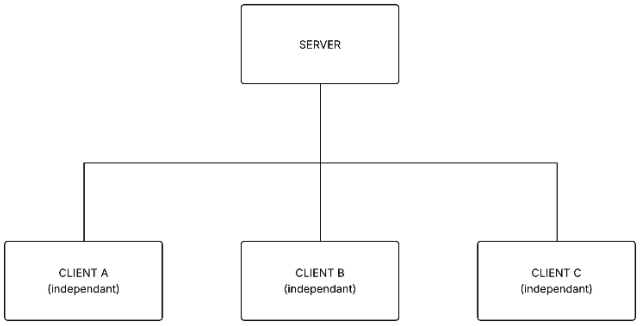
- **server:** https://github.com/CharakaJith/tcp-status-server
- **client:** https://github.com/CharakaJith/tcp-status-client

Design diagrams

1. **dataflow diagram**



2. **high-level architecture diagram**



TCP/IP client-server implementation

1. **Server application**
   - implemented functionalities
     - accepts multiple clients concurrently
     - continuously polls client for status (POLL Enable/Disable, configurable interval)
     - display received message in console

   - classes
     - **Program:** Entry point of the application; loads configuration and starts the StatusServer.
     - **TcpStatusServer:** TCP server that manages client connections, handles polling for status, and maintains client lifecycle.
     - **ClientConnection:** Represents and manages a single client connection, handling message reading, sending commands, and disconnection events.
     - **ProtocolMessages:** Defines all command and response message constants used in client-server communication.

   - concurrency and error handling
     - async/await ensures non-blocking connections
     - a lock is used to protect shared client list
     - exceptions handled gracefully in client communication

2. **Client application**
   - implemented functionalities
     - connects to server
     - sends acknowledgement (ACK) immediately upon receiving command (STATUS)
     - send status reply 3 seconds after receiving STATUS command (ACK / BUSY / STATUS_REPLY / ERROR / NAME)
     - responds with BUSY if new command arrives during the 3-second delay

    - simulates hardware delay and button status using ButtonStatusProvider

- classes
    - **Program:** Entry point of the application; loads configuration and starts the StatusServer.
    - **TcpStatusClient:** Represents a TCP client that connects to the server, handels incomoing commands, simulates hardware processing, and sends responses asynchronously.
    - **ButtonStatusProvide:** Provides simulated status for 10 buttons, generating a random 0 or 1 for each to mimic hardware input.
    - **ProtocolMessages:** Defines all command and response message constants used in client-server communication.

- error handling
    - exceptions on connect or send are logged
    - prevents crashing on server disconnect

3. Communication protocol

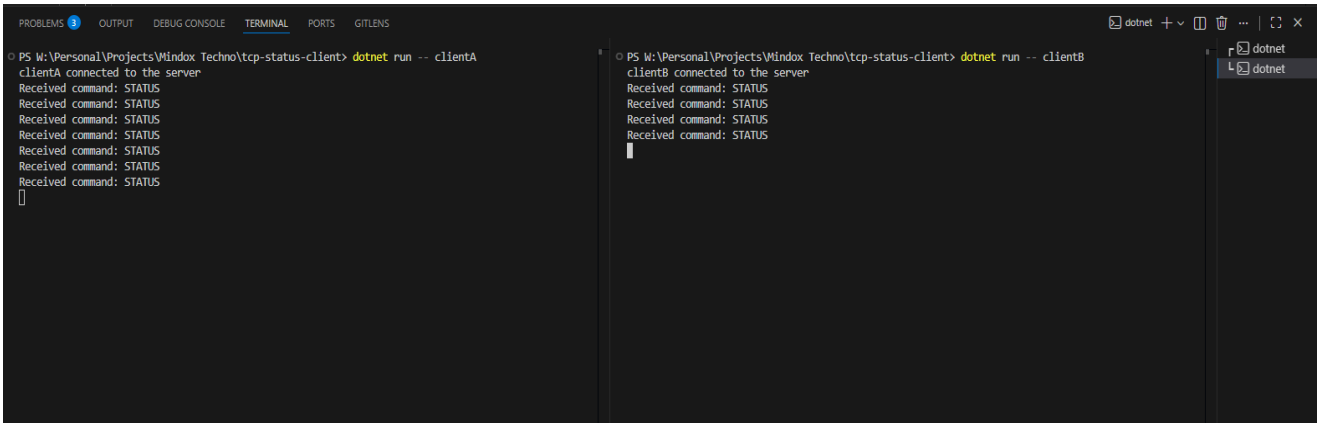
- messages are simple text-based
    - **STATUS:** server request
    - **ACK:** immediate client acknowledgement
    - **BUSY:** client busy response
    - **STATUS_REPLY|<status>:** client status after delay
    - **ERROR|<reason>:** unknown command

Demonstration

1. start the server

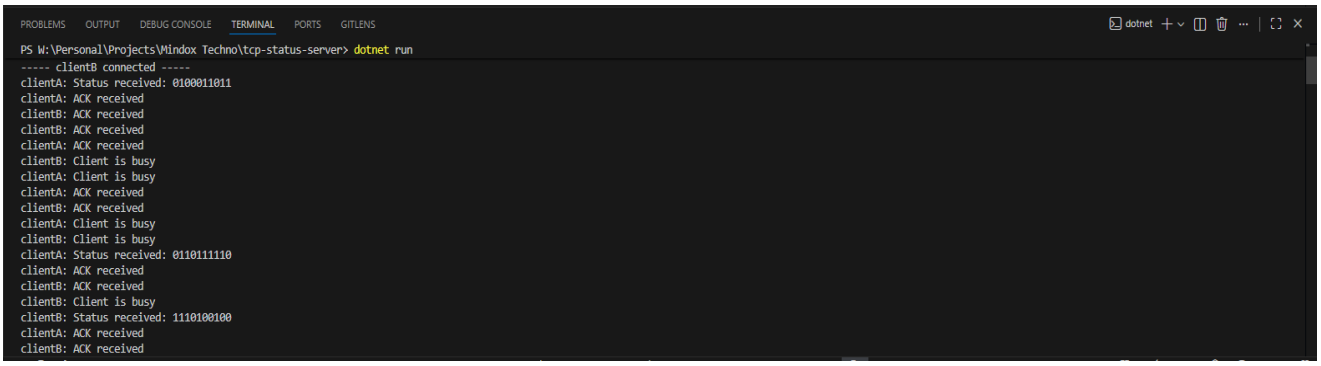
- Instructions on how to run the project are provided in the tcp-status-server GitHub README file.

2. **start one or multiple clients**



- Instructions on how to run the project are provided in the tcp-status-client GitHub README file.

3. **observe the console output**



References

- DotNET TCP/IP Networking Concepts
- Building a High-Performance TCP Server
- Network Programming in C#
- Simple TCP Server and Clients