

TCP/IP Status Monitoring System

(Mindox Techno)

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

Introduction

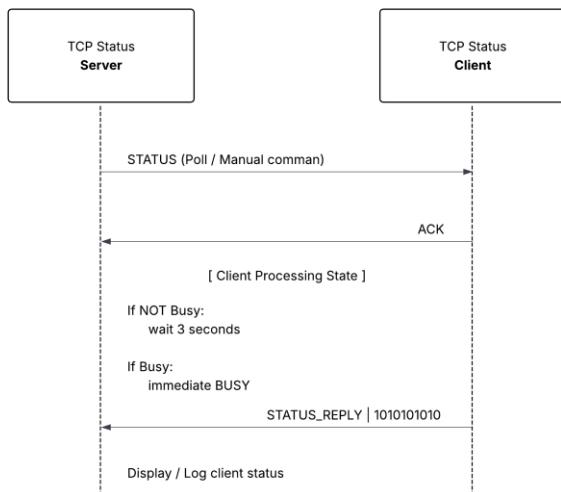
This assignment involves developing a TCP/IP server and client system in C#. The server can handle multiple clients concurrently, periodically polling them for status, and process commands. The client connects to the server, acknowledges received commands, simulates hardware processing with a delay, 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

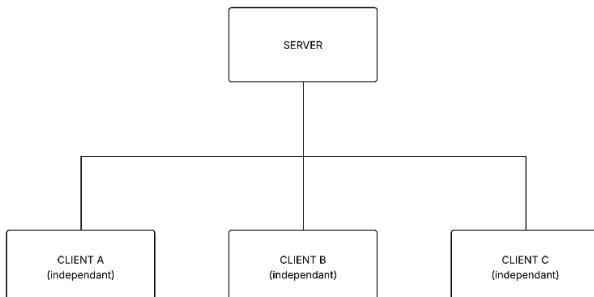
- server: <https://github.com/CharakaJith/tcp-status-server>
- client: <https://github.com/CharakaJith/tcp-status-client>

Design diagrams

1. dataflow diagram



2. high-level architecture diagram



TCP/IP client-server implementation

1. Server application

- implemented functionalities
 - o accepts multiple clients concurrently
 - o continuously polls client for status (POLL Enable/Disable, configurable interval)
 - o display received message in console
- classes
 - o **Program:** Entry point of the application; loads configuration and starts the StatusServer.
 - o **TcpStatusServer:** TCP server that manages client connections, handles polling for status, and maintains client lifecycle.
 - o **ClientConnection:** Represents and manages a single client connection, handling message reading, sending commands, and disconnection events.
 - o **ProtocolMessages:** Defines all command and response message constants used in client-server communication.
- concurrency and error handling
 - o async/await ensures non-blocking connections
 - o a lock is used to protect shared client list
 - o exceptions handled gracefully in client communication
- folder structure

```
tcp-status-server/
    |-- Program.cs
    |-- Server/
        |-- TcpStatusServer.cs
        |-- ClientConnection.cs
    |-- Protocol/
        |-- ProtocolMessages.cs
    |-- appsettings.json
    |-- README.md
```

2. Client application

- implemented functionalities
 - o connects to server
 - o sends acknowledgement (ACK) immediately upon receiving command (STATUS)
 - o send status reply 3 seconds after receiving STATUS command (ACK / BUSY / STATUS_REPLY / ERROR / NAME)
 - o responds with BUSY if new command arrives during the 3-second delay
 - o simulates hardware delay and button status using ButtonStatusProvider
- classes
 - o **Program:** Entry point of the application; loads configuration and starts the StatusServer.
 - o **TcpStatusClient:** Represents a TCP client that connects to the server, handles incoming commands, simulates hardware processing, and sends responses asynchronously.
 - o **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
- folder structure

```

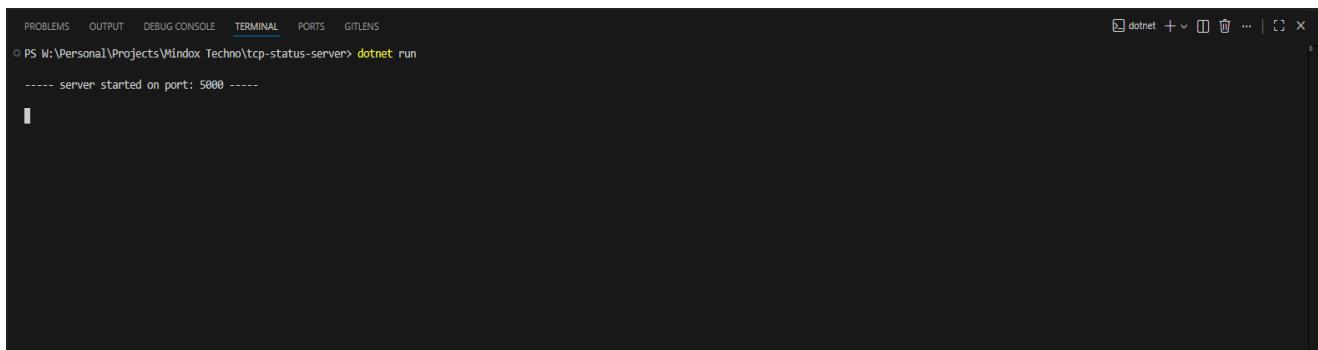
tcp-status-client/
    -- Program.cs
    -- Client/
        |   -- TcpStatusClient.cs
    -- Device/
        |   -- ButtonStatusProvider.cs
    -- Protocol/
        |   -- ProtocolMessages.cs
    -- README.md
    -- README.md
  
```

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

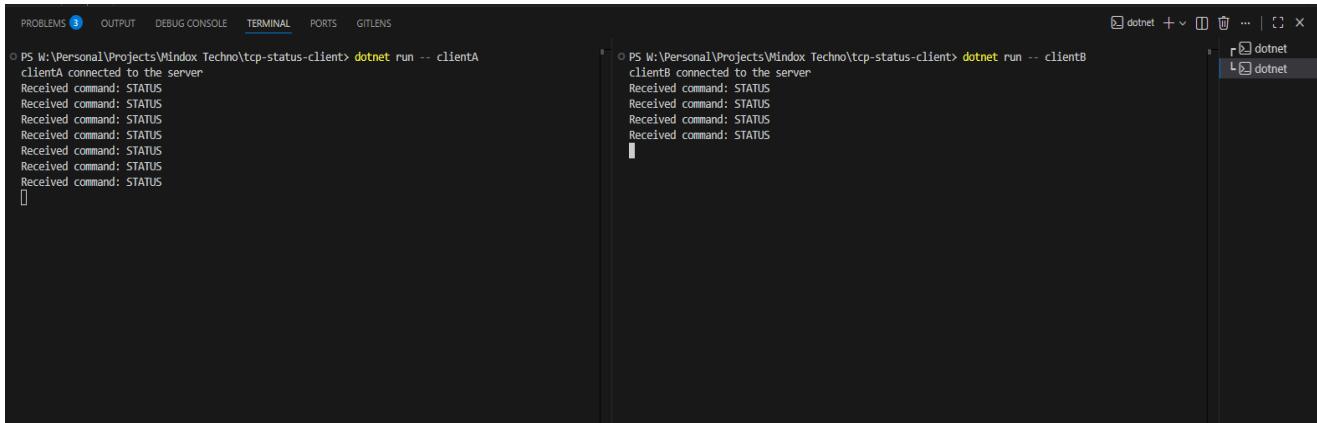
1. start the server



A screenshot of a terminal window titled "dotnet". The window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), PORTS, and GITLENS. The terminal shows the command "dotnet run" being executed in a directory "W:\Personal\Projects\Mindox Techno\tcp-status-server". The output of the command is: "server started on port: 5000".

- Instructions on how to run the project are provided in the [tcp-status-server GitHub README](#) file.

2. start one or multiple clients

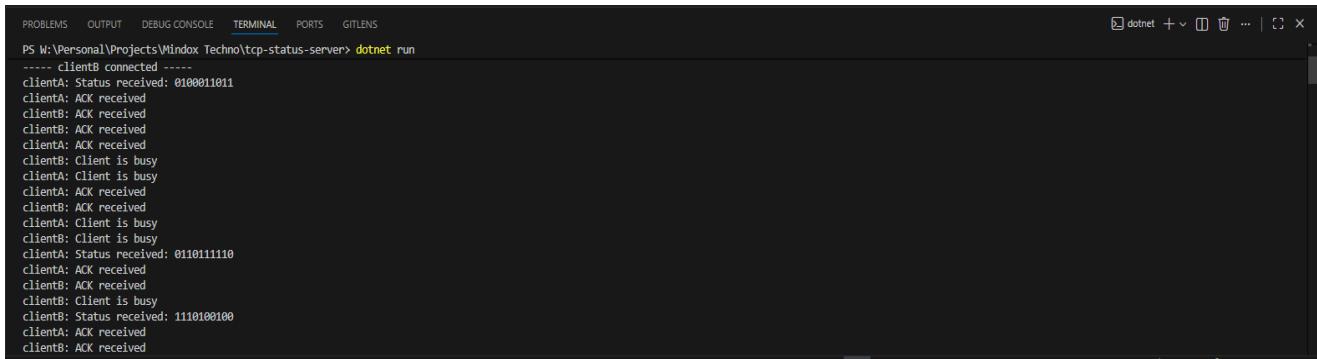


The screenshot shows two separate terminal windows side-by-side. Both windows have tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), PORTS, and GITLENS. The left terminal window displays the command PS W:\Personal\Projects\Windex Techno\tcp-status-client> dotnet run -- clientA followed by several lines of text indicating communication between clientA and clientB. The right terminal window displays the command PS W:\Personal\Projects\Windex Techno\tcp-status-client> dotnet run -- clientB, also showing communication between clientA and clientB.

```
PS W:\Personal\Projects\Windex Techno\tcp-status-client> dotnet run -- clientA
clientA connected to the server
Received command: STATUS
[...]
PS W:\Personal\Projects\Windex Techno\tcp-status-client> dotnet run -- clientB
clientB connected to the server
Received command: STATUS
Received command: STATUS
Received command: STATUS
Received command: STATUS
[...]
```

- Instructions on how to run the project are provided in the [tcp-status-client GitHub README](#) file.

3. observe the console output



The screenshot shows a single terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected), PORTS, and GITLENS. The terminal displays the command PS W:\Personal\Projects\Windex Techno\tcp-status-server> dotnet run. The output shows a continuous stream of text representing the interaction between clientA and clientB, including ACK and STATUS messages.

```
PS W:\Personal\Projects\Windex Techno\tcp-status-server> dotnet run
----- clientB connected -----
clientA: Status received: 0100011011
clientA: ACK received
clientB: ACK received
clientB: ACK received
clientA: ACK received
clientA: ACK received
clientB: Client is busy
clientA: Client is busy
clientA: ACK received
clientB: ACK received
clientA: Client is busy
clientB: Client is busy
clientA: Status received: 011011110
clientA: ACK received
clientB: ACK received
clientB: Client is busy
clientB: Status received: 1110100100
clientA: ACK received
clientB: ACK received
```

References

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