IoT Device Communication System Using C++ with Yocto for Raspberry Pi 4

Project Overview:

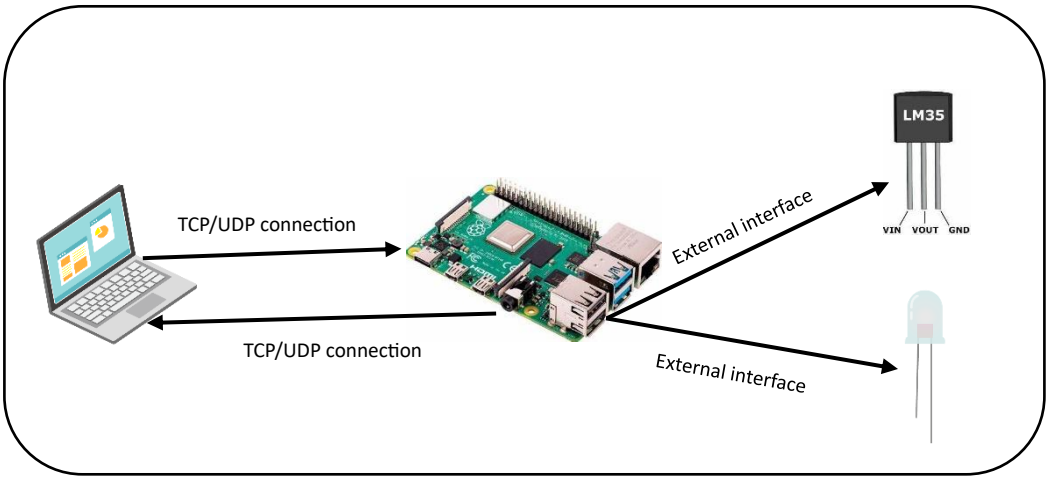
In this project, you will create an IoT device communication and control system that transmits temperature sensor data from a Raspberry Pi 4 client to a central server running on a laptop with a Qt6 GUI. The communication is handled over both TCP and UDP sockets and is implemented in C++ using object-oriented programming principles. The client application, integrated as a Yocto layer and burned into a Yocto-generated image, interfaces with a temperature sensor and controls an LED based on a critical temperature threshold sent from the server. The server application features a Qt6 GUI with three tabs—Real Time Monitor, Historical Analysis, and Configuration—to display the current temperature, analyze historical data, and calibrate the temperature threshold.

Project Structure:

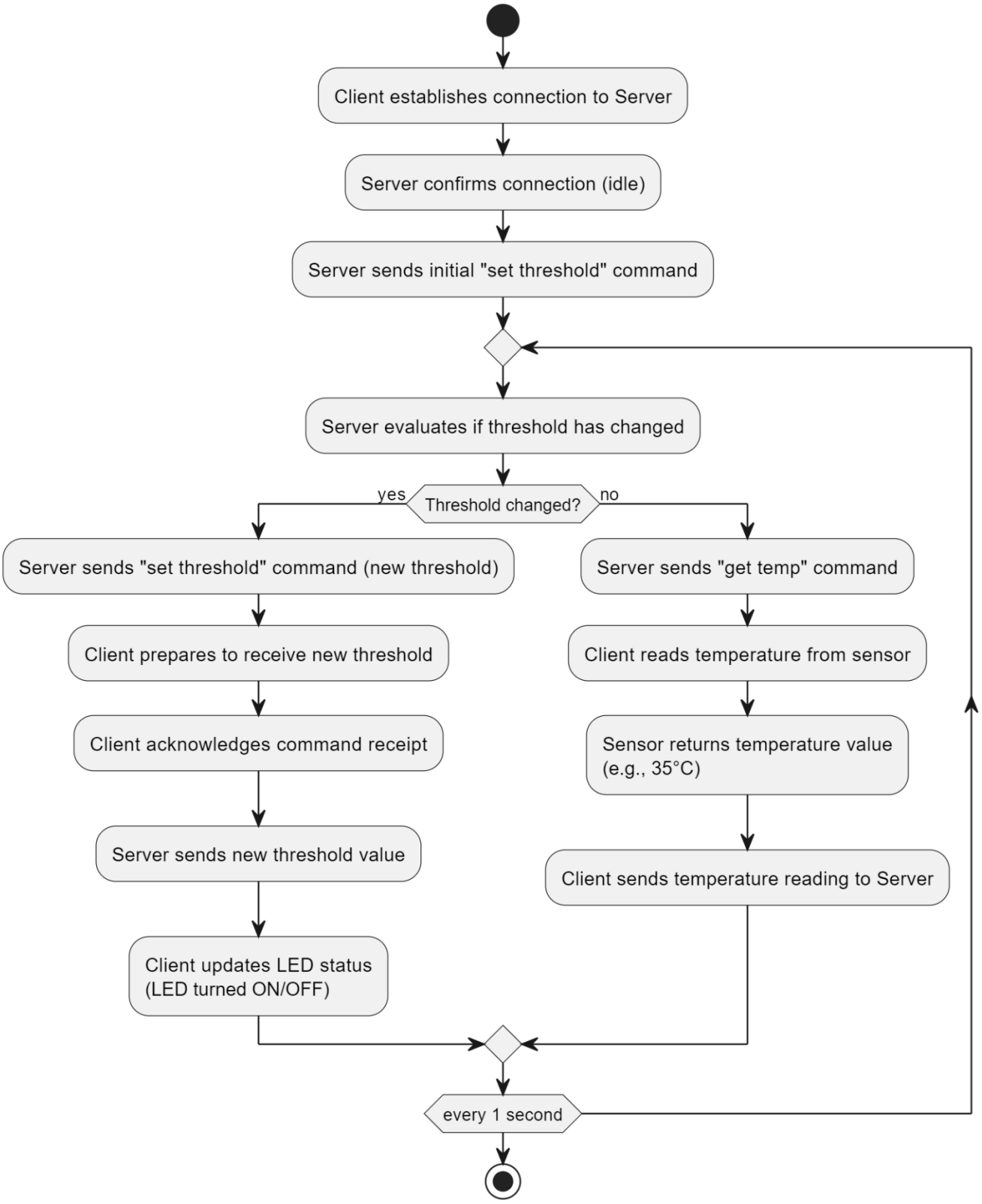
The project is divided into two main parts:

- Unicast TCP/UDP Server/Client Application
    - Server: Laptop with Qt6 GUI
    - Client: Raspberry Pi 4 with Temperature Sensor and LED Control

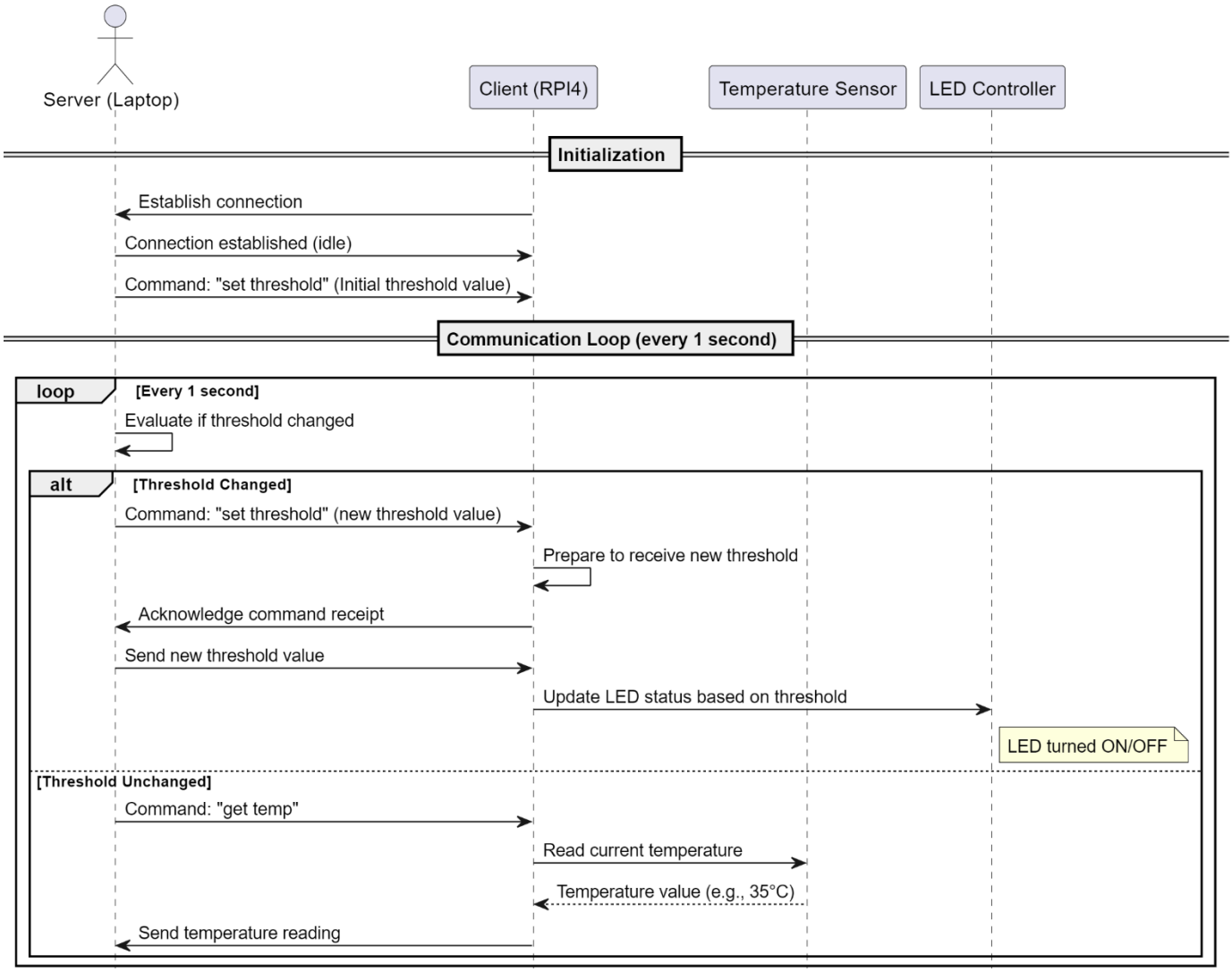
Project Block Diagram:



Project Flow Chart Diagram:



Project Sequence Diagram:



Deliverables:

- Full source code implementing the project.
- Yocto recipes for building the client application for Raspberry Pi 4.
- Qt6 project files and source code for the server application.
- Project demo **video (is a Must)**

Client Application (Raspberry Pi 4):

Task:

- Develop a C++ application to run on Raspberry Pi 4, integrated as a Yocto layer and burned into a Yocto-generated image.
- Interface with a temperature sensor to capture current temperature readings.
- Control an LED based on the critical temperature threshold received from the server.
- Communication Logic:
    - Initialization:
        - The client initiates a connection to the server.
        - Upon connection, the server confirms the connection (idle state) and immediately sends a "set threshold" command with the initial threshold value.
    - Communication Loop (Every 1 Second):
        - The server evaluates whether the critical temperature threshold has changed.
            - If the threshold has changed:
                - The server sends a "set threshold" command with the new threshold value.
                - The client prepares to receive the new threshold, acknowledges the command, and then receives the updated threshold.
                - The client updates the LED status accordingly (LED turned ON/OFF based on the new threshold).
            - If the threshold remains unchanged:
                - The server sends a "get temp" command.
                - The client reads the current temperature from the sensor.
                - The sensor value (e.g., 35°C) is sent back to the server.

Server Application (Laptop with Qt6 GUI):

Task:

- Develop a C++ server application with a Qt6 GUI that listens for client connections over both TCP and UDP.
- **Communication Logic:**
    - Initialization:
        - The client establishes a connection, and the server confirms it.
        - Immediately after, the server sends an initial "set threshold" command to provide the starting critical temperature.
    - Communication Loop (Every 1 Second):
        - The server automatically evaluates every second if the critical temperature threshold has changed:
            - Threshold Changed:
              The server sends a "set threshold" command to notify the client of the new threshold value.

        - Threshold Unchanged:
          The server sends a "get temp" command to request the current temperature reading.
- The GUI consists of Four tabs, **please refer to Resource.pdf file:**
    - **Tab 1: Real Time Monitor**
      Displays a simulated temperature meter showing the current temperature.
    - **Tab 2: Historical Analysis**
      Presents a 2D graph displaying historical temperature values received from the client.
    - **Tab 3: Configuration**
      Allows the user to set and calibrate the temperature threshold.
    - **Tab 4: Quick Access**
      This tab has 4 Buttons with ICON {Facebook, LinkedIn, Instagram}, upon clicking on it equivalent site will open up

Communication Protocol:

- Implement both TCP and UDP socket communication for the server/client connection.
- Server-Driven Interaction:
    - Initialization:
      The client establishes a connection, and the server sends a "set threshold" command with an initial threshold.
    - Loop:
      Every second, the server checks if the critical temperature threshold has changed:
        - If changed:
          The server instructs the client to prepare for receiving the new threshold, acknowledges the command receipt, sends the new threshold, and the client updates its LED status.
        - If unchanged:
          The server instructs the client to send the current temperature reading by issuing a "get temp" command.

OOP Design:

Enforce object-oriented programming (OOP) concepts such as Abstraction, Encapsulation, Inheritance, and Polymorphism throughout the project.

- **Abstract Class: Socket**
  Responsibilities:
  Define common socket behaviors (send, receive, connect, shutdown) with pure virtual functions.
  Functions:
    - virtual void waitForConnect() = 0;
    - virtual void connect() = 0;
    - virtual void send(const std::string& message) = 0;

    - virtual void receive() = 0;
    - virtual void shutdown() = 0;
- **Derived Classes: TCPSocket and UDPSocket**
  Inherit from *Socket* and implement protocol-specific behaviors for TCP and UDP communication.
- **Abstract Class: Channel**
  Data Member:
    - Socket* channelSocket; – A pointer to a *Socket* object (either *TCPSocket* or *UDPSocket*).
      Responsibilities:
      Manage socket interactions through the *channelSocket* member.
      Functions:
    - virtual void start() = 0;
    - virtual void stop() = 0;
    - virtual void send(const std::string& message) = 0;
    - virtual void receive() = 0;
- **Derived Classes: ServerChannel and ClientChannel**
  Implement specific server and client behaviors by utilizing the *channelSocket* to manage communication.

Development Environment:

- Yocto Setup:
  The client application should be cross-compiled for Raspberry Pi 4 using Yocto. Yocto recipes must be created for building and deploying the client application onto the Raspberry Pi 4.
- Qt6 Setup:
  Develop the server application using Qt6 to provide an intuitive and interactive GUI.
- Makefile:
  Provide a well-structured Makefile to automate the build process for both the server and client applications.

Tools and Technologies:

- C++ (OOP):
  For implementing socket communication and GUI logic while enforcing key OOP principles.
- Yocto:
  For cross-compiling and deploying the client application on Raspberry Pi 4.
- Qt6:
  For building the server GUI application.
- Socket Programming:
  For implementing TCP and UDP communication between the client and server.

Thank You
Edges For Training Team