

Lab 1: Individual Part Task 2

Ubiquitous Computing

Tiemor Amjad

Date: 06.11.2025

Task 2: Comprehensive Summary of Wi-Fi Capabilities

Task 2 required a comprehensive summary and analysis of the main points learned from the two specified Wi-Fi tutorials, focusing on the communication principles and their implications for Ubiquitous Computing. This summary is expected to cover approximately 2–3 pages.

2.1. Tutorial 1: Web Server via Existing Infrastructure (Client Mode Analysis)

This tutorial demonstrates setting up the Arduino Nano RP2040 Connect as a Web Server that operates within an existing Wi-Fi network (WLAN) by acting as a Client. This is the standard operational mode for Internet of Things (IoT) devices deployed in homes or offices.

A. Architectural Role and Principle

In this configuration, the Arduino's core architectural role is two-fold: it is a Client (connecting to the router) and simultaneously a Server (hosting the control interface).

- **Network Integration:** The board uses static credentials (`SSID`, `Password`) to join the network via `WiFi.begin(ssid, pass)`. This grants it a local IP address.
- **Request Handling:** The `WiFiServer server(80)` object is instantiated to listen for incoming connections on HTTP Port 80. Crucially, the web server is only accessible to clients (e.g., phones, PCs) already connected to the same WLAN.
- **Protocol Details (HTTP GET):** Control is achieved through simple HTTP GET `requests`. The HTML generated by the Arduino creates buttons that, when clicked, append specific strings (e.g., `/RH` for Red High) to the URL. The Arduino parses the incoming HTTP request string (`currentLine.endsWith("GET /RH")`) to determine which action to execute (e.g., `digitalWrite(LED_R, HIGH)`).

B. Core Functions and Implementation Details

The tutorial highlights several fundamental functions from the `WiFiNINA.h` library essential for client-side server operation:

Function/Class: `WiFi.begin(ssid, pass)` This function is responsible for **Connection Initialization**. It attempts to authenticate and connect the board to the external network. The use of a `while (status != WL_CONNECTED)` loop ensures the process is robust against transient failures by continuously retrying the connection until successful.

Function/Class: `WiFiServer server(80)` This object handles **Server Initialization**. It creates the listening socket on HTTP port 80. This is the mechanism that allows the microcontroller to host a web page and respond to client requests over the network.

Function/Class: `WiFiClient client = server.available()` This function acts as the **Request Listener**. It checks for a new incoming connection from a client (e.g., a web browser request). This must be constantly polled in the `loop()` function to ensure the server is responsive.

Function/Class: `client.print(...)` This method is used for **Response Generation**. It sends the HTTP response, including the mandatory headers (`HTTP/1.1 200 OK`, `Content-type:text/html`) and the raw HTML/CSS content back to the browser, which renders the control interface.

Function/Class: `currentLine.endsWith("GET /X")` This logic performs **Command Parsing**. It is the mechanism used to extract the user's intended action (actuator control) from the lengthy, raw HTTP request string sent by the browser when a button is clicked.

C. Architectural Implications for Ubiquitous Computing

This Client-Server mode is the dominant architecture for smart devices and offers significant advantages:

- **Centralized Access:** Control is possible from any device on the network, making it suitable for permanent installation (Smart Home).
- **Scalability for IoT:** By adding logic to send data out (`HTTP POST` to a remote server), this architecture easily extends to long-term **Data Aggregation**, allowing the posture device to upload performance statistics to a user profile in the cloud (as proposed in Task 1.3).
- **Firmware Management:** The connection to a wide-area network allows for background checks for **Over-the-Air (OTA) Firmware Updates**, ensuring the longevity and security of the deployed device.

2.2. Tutorial 2: Web Server in Access Point Mode (Server/AP Mode Analysis)

This tutorial explores a contrasting, **self-contained** communication model where the Arduino Nano RP2040 Connect creates its own isolated Wi-Fi network by functioning as both the **Server** and the **Access Point (AP)**.

A. Architectural Role and Principle

In this mode, the Arduino acts as the sole network provider, or the "network island."

- **Self-Contained Network:** The board initiates a private network using `WiFi.beginAP(ssid, pass)`. This network is typically isolated from the external internet. By default, the board assigns itself the IP address **192.168.4.1**.
- **Direct Connection:** External clients (phones, PCs) connect directly to the Arduino's generated SSID. Once connected, they can access the web server hosted by the Arduino at its default IP address.
- **Server Operation:** Similar to the Client Mode, the board uses `WiFiServer` to handle incoming HTTP requests from the directly connected clients. The RGB LED control mechanism (parsing `/RH`, `/GL`, etc.) remains identical.

B. Core Functions and Implementation Details

The AP mode introduces new setup commands distinct from the Client mode:

Function/Class: `WiFi.beginAP(ssid, pass)` This is the **AP Initialization** function. It is crucial because it switches the board from a Client role to an Access Point, creating a new local WLAN that other devices can see and join.

Function/Class: `WiFi.status()` This function is used for **State Monitoring**. It checks the current status of the Wi-Fi module, including `WL_AP_LISTENING` (confirming the AP is active) and detecting `WL_AP_CONNECTED` (a client successfully joined the network). This is key for managing the state of the AP.

Function/Class: `WiFi.config(...)` This is used for **Network Customization (Optional)**. It allows the developer to override the default local IP address (192.168.4.1), which can be useful in complex pervasive system deployments where specific IP ranges are required.

Function/Class: `server.available()` and `client.print(...)` These functions handle **Request Handling & Response**. They operate identically to the Client Mode, demonstrating that the underlying web server implementation logic is reusable regardless of whether the network is externally connected or self-generated.

C. Architectural Implications for Ubiquitous Computing

The AP/Server Mode is fundamental for scenarios requiring ad-hoc or local-only connectivity:

- **Initial Field Deployment (Configuration):** This is the ideal **First-Time Setup** method. A user can connect directly to the Arduino to input the permanent Wi-Fi credentials (SSID/Password) of the home network. This is often called a "captive portal" strategy.
- **Local Reliability:** The device's control interface remains accessible even if the main router or internet connection fails. This guarantees a local channel for diagnostic and manual control, making the device highly robust.
- **Pervasive System Interaction:** AP mode supports **peer-to-peer (P2P)** networking among multiple proximate devices. This can be used, for example, if five posture monitors needed to communicate quickly in a room without relying on the building's central network.

3. Conclusion

The analysis of Task 2 confirms the **architectural duality** of the Nano RP2040 Connect's Wi-Fi module.

1. **Client/Server Mode (Tutorial 1):** Essential for **remote control, IoT data flow, and large-scale analytical processing**, connecting the device seamlessly to the global internet infrastructure.
2. **AP/Server Mode (Tutorial 2):** Crucial for **local configuration, independent operation, and ad-hoc communication**, ensuring the device is deployable and reliable, even in environments without stable external Wi-Fi access.

These two distinct capabilities are indispensable for building true Ubiquitous Computing devices that can be locally deployed, globally managed, and robustly configured in any environment.