Drone Mesh Network - Point-to-Point Link Test Methodology

Objective

To evaluate and compare the performance of different wireless communication protocols (WiFi 4, WiFi 6, WiFi Long-Range, ESP-NOW) operating in the 2.4 GHz band for establishing a point-to-point communication link between two ESP32-C6 boards, simulating a basic drone-to-drone communication scenario. The evaluation will focus on key metrics relevant to drone telemetry exchange: communication range, data latency, packet loss rate, and received signal strength (RSSI), while adhering to Australian regulatory requirements.

Rationale for Protocol Selection

The goal of this project is to eventually implement a self-organizing and self-healing mesh network for drone communication. Several mesh networking solutions exist for the ESP32 platform, including Espressif's official ESP-WIFI-MESH and ESP-MESH-LITE, the popular community project PainlessMesh, and the potential use of standard routing protocols like OSLRv2 layered over protocols such as ESP-NOW.

- Espressif Mesh Comparison
- PainlessMesh
- OSLRv2 RFC

Before implementing and testing these complete mesh solutions, it is crucial to understand the performance characteristics of the underlying wireless protocols they utilize or could potentially utilize, specifically within the **2.4 GHz band supported by the ESP32-C6**. This point-to-point testing phase focuses on evaluating these foundational protocols:

- **WiFi 4 (802.11n) & WiFi 6 (802.11ax):** Selected because standard WiFi is the basis for ESP-WIFI-MESH, ESP-MESH-LITE, and PainlessMesh. Evaluating the raw link performance (range, latency, reliability) of WiFi 4 and the newer WiFi 6 on the ESP32-C6 (2.4 GHz only) provides essential baseline data relevant to these mesh implementations, especially considering the typical drone communication requirement for line-of-sight operation over distances exceeding 100 meters.

Feature	WiFi 4 (802.11n @ 2.4GHz)	WiFi 6 (802.11ax @ 2.4GHz)	Relevance to Drone Mesh (ESP32-C6)
Standard	IEEE 802.11n	IEEE 802.11ax	Foundation protocol for testing.

Frequency Band	2.4 GHz	2.4 GHz	Only band supported by ESP32-C6.
Max Data Rate	~150 Mbps (theoretical, 40MHz)	~574 Mbps (theoretical, 40MHz)	Higher potential throughput indicates efficiency gains.
Modulation	Up to 64-QAM	Up to 1024-QAM	More data per symbol, better spectral efficiency.
Key Technologies	MIMO, Channel Bonding (40MHz)	OFDMA, MU-MIMO (Up/Down), 1024-QAM, Target Wake Time (TWT), BSS Coloring	**OFDMA** crucial for efficiency in dense environments (many drones). **TWT** improves power saving. **MU-MIMO** improves simultaneous communication.
Efficiency	Lower, especially in dense environments	Higher, designed for dense device environments	WiFi 6 should handle multiple drone connections more gracefully.
Latency	Higher	Lower potential latency	Lower latency is critical for real-time control/telemetry.
Range	Generally good	Similar or slightly better range due to efficiency improvements.	Key metric; efficiency might improve practical range/reliability.
Power Consumption	Standard	Improved power efficiency via TWT	Important for drone battery life.
Device Compatibility	Widely compatible with nearly all existing WiFi devices	Backwards compatible with WiFi 4/5 devices; requires WiFi 6 devices for full features	ESP32-C6 supports both. Consider compatibility with other potential network nodes (e.g.,

			ground stations).

- **ESP-NOW:** Chosen as it is a proprietary, connectionless, low-power protocol from Espressif designed for quick transmission of small packets in the 2.4 GHz band. While documentation suggests it can achieve good range and efficiency, potentially exceeding standard WiFi range under certain conditions, it typically offers lower bandwidth. Its suitability for telemetry-sized packets and its potential as a base layer for lightweight mesh implementations (including those using OSLRv2) makes it a critical protocol to evaluate.
- **WiFi Long-Range (LR):** Included to assess the capabilities of Espressif's proprietary extension to standard 2.4 GHz WiFi protocols, which aims to increase communication range. While it achieves this by reducing bandwidth (refer to Espressif documentation), understanding its practical range and performance limits is valuable. Its potential integration with existing mesh solutions, if feasible, could be advantageous, although it's less commonly supported than standard WiFi modes.

By first characterizing these fundamental protocols in a controlled point-to-point scenario, we can make informed decisions about which protocols are most promising for the target application and provide a solid foundation for subsequent development and testing of full mesh network implementations.

Testbed Setup

- **Hardware:**
  - **Board 1 (Sender - Stationary):**
    - ESP32-C6 Development Board (with built-in antenna initially).
    - GPS Module (connected to ESP32-C6 via **UART**). Ensure GPS has a clear view of the sky.
    - Power source (e.g., USB power bank).
  - **Board 2 (Receiver - Mobile):**
    - ESP32-C6 Development Board (with built-in antenna initially).
    - GPS Module (connected to ESP32-C6 via **UART**). Ensure GPS has a clear view of the sky.
    - Laptop computer.
    - USB cable (connecting Board 2 to Laptop).
    - Power source for Board 2/GPS (can be laptop USB if sufficient).
- **Software:**
  - **ESP32 Firmware (Board 1 - Sender):**
    - Developed in C++ using PlatformIO.
    - Configured for the specific protocol under test (WiFi 4/6/LR, ESP-NOW).
    - Obtains absolute time from GPS NMEA data (UART).
    - Performs initial network time synchronization with Board 2 (see Section 5).
    - Periodically sends test packets containing:
      - Packet Sequence Number (incrementing).
      - Sender Timestamp (high-resolution esp_timer_get_time()).
      - Payload (simulating MAVLink telemetry, e.g., using a fixed, non-repeating payload pattern of 75 bytes).
    - Configurable packet sending rate (e.g., 10-50 Hz - test at a rate representative of MAVLink).
    - Configured Transmit Power level recorded (e.g., in dBm).
  - **ESP32 Firmware (Board 2 - Receiver):**
    - Developed in C++ using PlatformIO.
    - Configured for the specific protocol under test.
    - Obtains absolute time from GPS NMEA data (UART).
    - Participates in initial network time synchronization with Board 1 (see Section 5).
    - Listens for incoming packets.
    - Upon receiving a packet:
      - Records Receiver Timestamp (high-resolution esp_timer_get_time()).
      - Retrieves RSSI of the received packet.

        - Reads GPS data (Timestamp from NMEA, Latitude, Longitude, Altitude, Number of Satellites, Accuracy/HDOP).
        - Extracts Sequence Number and Sender Timestamp from the packet.
        - Sends all collected data along with the configured Tx Power of the Sender to the connected laptop via Serial USB.
    - **Laptop Software:**
        - PlatformIO Device Monitor (pio device monitor --raw) configured to log raw serial output directly to a timestamped file in CSV format with columns such as:
            - ReceivedTimestamp_ms
            - SequenceNumber
            - SenderTimestamp_us
            - ReceiverTimestamp_us
            - RawLatency_us
            - CorrectedLatency_ms
            - RSSI_dBm
            - ConfiguredTxPower_dBm
            - GPS_Timestamp_UTC
            - GPS_Lat, GPS_Lon
            - GPS_Alt, GPS_Sats
            - GPS_HDOP

        (Note: Timestamps in microseconds for calculation, final latency likely in ms).

        - Data processing script (e.g., Python with Pandas, NumPy, Haversine libraries) for analysis after the test. This script will apply the calculated initial offset to determine CorrectedLatency_ms.

- **Network Configuration (Australia Regulatory Compliance)**
  - **Frequency Band:** Testing will utilize the **2.4 GHz ISM band (Channels 1-13)**, as this is the only band supported by the ESP32-C6.
  - **Transmit Power:** It is a regulatory necessity to configure the ESP32's transmit power to comply with Australian Communications and Media Authority (ACMA) regulations. For the 2.4 GHz band used by these protocols, the maximum limit is **1 Watt (30 dBm) EIRP** (Effective Isotropic Radiated Power). *Firmware must explicitly set the power level to ensure this limit is not exceeded, and this configured level must be logged.*
  - **Channel Selection:** Select a *fixed*, relatively clear channel for each test run within the 2.4 GHz band (e.g., Channel 1, 6, or 11). Use a WiFi analyzer tool beforehand to identify channels with minimal existing interference at the test location. Use the *same* channel for all runs of a specific protocol test.
  - **Protocol Specifics:**
    - **WiFi 4/6:** Configure one ESP32 as an Access Point (AP) and the other as a Station (STA). Set the correct region code (e.g., "AU") if available in firmware settings.
    - **WiFi Long-Range:** Configure according to Espressif's documentation for LR mode, ensuring power limits are respected.
    - **ESP-NOW:** Pair the two devices using their MAC addresses. ESP-NOW operates on the 2.4 GHz band, typically using the same channel as the current WiFi configuration or a default channel if WiFi is disabled. Ensure this channel is consistent.

Test Parameters & Metrics

- **Independent Variables:**
  - **Protocol:** WiFi 4 (802.11n), WiFi 6 (802.11ax), WiFi Long-Range, ESP-NOW (all configured for Australian compliance on 2.4 GHz).
  - **Distance:** Variable, measured using GPS (0m up to the maximum achievable range).
  - **(Optional) Packet Rate:** Test different sending frequencies (e.g., 1Hz, 5Hz, 10Hz).
- **Dependent Variables (Metrics to Measure):**
  - **Distance (m):** Calculated using the Haversine formula based on the GPS coordinates recorded simultaneously by both boards.
  - **Received Signal Strength Indicator (RSSI, dBm):** As reported by the ESP32 receiver hardware.
  - **One-Way Latency (ms):** Calculated as Receiver Timestamp (esp_timer) - Sender Timestamp (esp_timer) - Initial_Offset. Requires initial network time synchronization (see Section 5).
  - **Packet Loss Rate (%):** Calculated over distance intervals (e.g., every 10m). Packet Loss = (Expected Packets - Received Packets) / Expected Packets * 100%.

Test Procedure

1. **Preparation:**
    - Select the protocol and 2.4 GHz channel to test (e.g., WiFi 4, Channel 6).
    - Configure firmware for Australian power limits (e.g., set Tx power to 20 dBm or the desired level below 30 dBm EIRP). Flash Sender firmware onto Board 1 and Receiver firmware onto Board 2. Ensure the configured Tx power is correctly set in the firmware and will be logged.
    - Place Board 1 at its designated fixed location. Record its precise GPS coordinates after achieving a stable satellite fix. Power it on.
    - Connect Board 2 (Receiver) and its GPS module to the laptop. Power it on and ensure its GPS has a satellite fix.
    - Start the PlatformIO device monitor logging on the laptop.
    - Position Board 2 near Board 1 (e.g., 1-2 meters apart).
    - **Initiate and verify the initial network time synchronization procedure (Section 5). Record the calculated initial offset.**
    - Verify initial communication: Check the log file for incoming packets with valid data, synchronized timestamps (considering the offset), low latency, and the correct configured Tx power value.
2. **Execution:**
    - Start moving Board 2 (and the connected laptop) slowly and directly away from Board 1, maintaining Line-of-Sight (LoS). Walk at a steady pace.
    - Ensure the antenna orientation of both boards remains relatively consistent (e.g., vertical).
    - Continuously monitor the live serial output (optional, as it's being logged) and GPS status.
    - Continue moving until the packet loss rate, calculated over a rolling 10-second window, consistently exceeds 80%, or no packets are received for 15 consecutive seconds.
    - Record the maximum approximate distance achieved (can be refined from GPS logs later).
    - Stop the device monitor logging and ensure the file is saved.

3. **Iteration:**
    - Repeat steps 1-2 for a minimum of 3 runs for each protocol/channel combination to ensure statistical significance.
    - Repeat the entire process (steps 1-3) for each of the other protocols (WiFi 6, WiFi LR, ESP-NOW).

Clock Synchronization Method (UART GPS + Network Sync)

Accurate one-way latency measurement without a GPS PPS signal requires compensating for the offset and potential drift between the internal high-resolution timers (esp_timer) of the two ESP32 boards.

1. **Absolute Time Reference:** Both boards obtain UTC time by parsing NMEA sentences ($GPRMC, $GPZDA, etc.) from their respective UART-connected GPS modules. This provides a common time reference but is typically not precise enough for microsecond-level synchronization due to parsing and UART jitter.
2. **High-Resolution Timestamps:** Packet send (Ts) and receive (Tr) events are timestamped using the local board's esp_timer_get_time() function, which provides microsecond resolution relative to the board's boot time.
3. **Initial Offset Calculation:** At the start of each test run, while the boards are stationary and close together (1-2m), perform a network time synchronization exchange (e.g., using ESP-NOW or a temporary WiFi link) to estimate the initial offset between their esp_timer clocks:
    - Board 1 sends multiple (e.g., 10) timestamped "ping" packets to Board 2. Each ping contains Board 1's esp_timer timestamp (T1).
    - Board 2 receives each ping at its esp_timer time (T2), records T1 and T2, and immediately sends an "ack" packet back containing T1 and T2.
    - Board 1 receives the ack at its esp_timer time (T3).
    - For each exchange, Board 1 calculates:
        - Round Trip Time (RTT) = T3 - T1
        - Estimated One-Way Delay (OWD) = RTT / 2
        - Estimated Offset = T2 - (T1 + OWD)
    - Average the calculated Estimated Offset values from the multiple exchanges to get a more robust Initial_Offset. This value represents the difference (esp_timer_B - esp_timer_A) at the start of the test.

4. **Applying the Offset:** This Initial_Offset is recorded and used during data analysis (Section 6) to correct the raw latency measurements.
5. **Accounting for Drift:** Be aware that the ESP32 internal oscillators will drift over time, causing the actual offset to change during the test. This method primarily corrects the *initial* offset. For highest accuracy on long tests, periodic re-synchronization (repeating step 3) would be necessary, but this may complicate the test procedure. The analysis should acknowledge potential drift as a source of error, especially for longer test durations.

Data Analysis Plan

1. **Data Parsing:** Load the logged raw text/CSV data into Python with Pandas. Clean and structure the data according to the defined format (see Section 2). Verify the ConfiguredTxPower_dBm column is consistent for a given test run. Extract the Initial_Offset calculated for the run.
2. **Distance Calculation:** For each logged receiver packet, calculate the distance between the known, fixed GPS coordinates of the stationary sender (Board 1) and the mobile receiver's (Board 2) logged GPS coordinates using the Haversine formula.
3. **Latency Calculation:** For each packet, calculate:
    - RawLatency_us = ReceiverTimestamp_us - SenderTimestamp_us
    - CorrectedLatency_us = RawLatency_us - Initial_Offset (using the offset calculated at the start of that specific test run)
    - Convert CorrectedLatency_us to milliseconds for analysis and plotting. Analyze the distribution of latencies (mean, median, 95th percentile).
4. **Packet Loss Calculation:** Analyze sequence numbers over distance intervals (e.g., 0-10m, 10-20m, ...). Within each interval, determine the number of received packets. Calculate the expected number of packets based on the difference between the highest and lowest sequence number received in that interval (+1), or based on the time duration of the interval and the known sending rate. Use this to calculate the loss rate: Loss = (Expected - Received) / Expected * 100%.
5. **Visualization:** Create plots for each protocol:
    - RSSI vs. Distance
    - Latency (mean, median, 95th percentile) vs. Distance
    - Packet Loss Rate (%) vs. Distance
6. **Comparison:** Overlay plots to compare protocols based on range, latency, reliability, and signal strength under Australian regulations (2.4 GHz, max 30 dBm EIRP).

Assumptions & Limitations

- **Line-of-Sight (LoS):** Test assumes clear LoS.
- **Point-to-Point:** Tests a single link, not a full mesh.
- **MAVLink Simulation:** Uses simulated payloads (fixed 75-byte pattern).
- **Antenna Performance:** Initial results are for built-in antennas. *Further tests may be conducted with specific external antennas if initial range/performance is insufficient, requiring documentation of the antenna type and gain.*
- **Environmental Factors:** Results specific to test conditions.
- **GPS Accuracy:** Distance calculation depends on GPS accuracy. Ensure good satellite fix (low HDOP, high satellite count).
- **Clock Synchronization Accuracy:** The accuracy of latency measurements depends on the initial network sync and is affected by ESP32 internal clock drift over the test duration. This method is less precise than GPS PPS synchronization.
- **Compliance:** Assumes firmware correctly implements ACMA power limits (1 W / 30 dBm EIRP for 2.4 GHz). It is a legal requirement to operate within these limits.
- **Hardware Limitation:** All tests are limited to the **2.4 GHz band** due to ESP32-C6 hardware constraints.

Expected Outcomes / Success Criteria

- Quantitative comparison of protocols operating in the 2.4 GHz band under Australian regulations.
- Data-driven insights for protocol selection for >100m LoS links using ESP32-C6.
- Identification of typical latency/reliability vs. distance for each protocol on 2.4 GHz, acknowledging the limitations of the clock synchronization method.
- Baseline performance data for built-in antennas, informing potential need for external antennas.