Comprehensive Analysis of Bluetooth Low Energy (BLE) Communication Methods using the ESP32 Platform

Author: S.A.C.A.Senanayaka
Affiliation: Unaffiliated
Email: achamath1@gmail.com

Abstract

The ESP32 microcontroller platform is pivotal in modern Internet of Things (IoT) development, owing to its integrated support for Bluetooth Low Energy (BLE). This paper systematically analyzes the four primary BLE communication paradigms applicable to the ESP32: Connected Attribute Exchange (GATT), Connectionless Data Dissemination (Advertising), Standardized Proximity Signaling (Beacons), and Multi-Hop Networking (BLE Mesh). We detail the architectural foundations, including the Generic Access Profile (GAP) and the Generic Attribute Profile (GATT) Client-Server duality, which dictate the structure and reliability of data flow. A comparative analysis is presented, assessing the trade-offs between throughput, latency, reliability, and scalability for each method. The findings provide critical guidance for system architects designing robust and energy-efficient wireless communication strategies using the ESP32 platform, ensuring optimal selection based on application requirements.

I. Introduction

The integration of robust wireless capabilities, specifically Wi-Fi and Bluetooth, within the ESP32 platform has cemented its role as a foundation for embedded systems.[1] Its support for Bluetooth Low Energy (BLE) facilitates efficient, low-power connectivity between diverse endpoints, including mobile devices, computers, and other embedded nodes.

A. The ESP32 Hardware and Software Architecture for BLE

The ESP32's architectural advantage lies in its dual-core Xtensa processor. This design allows for the concurrent execution of application logic and the latency-critical operations of the BLE stack, which is essential for maintaining timing requirements during advertising and connection events. This core separation enhances stability and responsiveness in demanding wireless applications.

Development platform selection significantly influences access to and control over BLE features:

- **Arduino Framework:** Offers high-level abstractions, ideal for rapid prototyping. It simplifies role establishment, such as creating a server instance using BLEDevice::createServer() [2] or initiating client connections.[2]
- **ESP-IDF (Espressif IoT Development Framework):** Provides access to advanced, low-level configuration required for enterprise-grade applications, particularly enabling features like Bluetooth Mesh [3], managed through utilities such as menuconfig.[5]

B. High-Level Communication Paradigms

BLE supports three fundamental architectural methodologies, each optimized for different network requirements:

1. **Connection-Oriented (GATT):** Utilizes a secure, bi-directional link between two devices (Point-to-Point).
2. **Connection-less (Advertising/Broadcast):** Implements unidirectional, stateless communication from a single device to multiple observers.
3. **Advanced Networking (Mesh):** Establishes a scalable, multi-hop network for many-to-many communication across extended physical areas.

II. Foundational Architecture (GAP and GATT)

Effective communication on the ESP32 relies on a thorough understanding of the two

mandatory protocol layers governing all BLE interactions.

A. Generic Access Profile (GAP): Roles and State Management

GAP manages the foundational aspects of device interaction, including discovery, connection establishment, and security.[6] It defines operational roles based on the device's function during the discovery and connection phases [7]:

- **Connection-Oriented Roles:**
  - **Peripheral:** Advertises and is connectable; waits for an initiator.[7]
  - **Central (Initiator):** Scans for peripherals and actively establishes connections.[6]
- **Connection-less Roles:**
  - **Broadcaster (Advertiser):** Transmits advertisement packets but cannot be connected to .
  - **Observer (Scanner):** Passively listens for advertisement packets without initiating a connection .

The selection of the GAP role (e.g., Central versus Broadcaster) is the initial architectural decision determining whether subsequent data exchange will be managed by the stateful GATT protocol or stateless advertising.

B. Generic Attribute Profile (GATT): Structured Data Exchange

GATT operates exclusively after a secure connection is established . It defines a hierarchical structure for organizing and exchanging data attributes, vital for bi-directional communication.

1. The Client-Server Duality

GATT enforces a strict Client-Server relationship post-connection.[8] The Peripheral typically functions as the GATT Server, and the Central as the GATT Client.

- **GATT Server:** Stores the data, organized into Services and Characteristics. The Server accepts requests (Read/Write) and sends asynchronous updates (Notify/Indicate).[10]

- **GATT Client:** Initiates commands to the Server to retrieve or modify data.[10]

2. The Attribute Hierarchy

Data is organized hierarchically within GATT [8]:

1. **Service:** A collection of data and behaviors fulfilling a specific function (e.g., a "Health Thermometer Service").[8]
2. **Characteristic:** The core data element, comprising a single value, access properties (READ, WRITE), and configuration information.[2]
3. **Descriptor:** Optional metadata, most notably the **Client Characteristic Configuration Descriptor (CCCD)**.[11]

3. Data Flow Control via CCCD

The CCCD is crucial for enabling Server-initiated data push operations (Notifications or Indications).[13] The Client must explicitly write an attribute to the CCCD to "subscribe" to the characteristic. This explicit action shifts control, allowing the Server (e.g., the ESP32) to send unsolicited data when needed, conserving power by ensuring data is only transmitted when a Client is actively listening.[13]

III. Connected Attribute Exchange (GATT)

This method involves establishing a direct, stateful Point-to-Point connection between an ESP32 (Server/Peripheral) and a host (Client/Central).[10]

A. Implementation Architecture

The ESP32 can fulfill both roles in a connected system:

- **As GATT Server (Peripheral):** Defines the service hierarchy, populates characteristic values (e.g., sensor readings) [2], and advertises its presence. Robust connection management is required to handle disconnections, as the Server is typically single-client until released.[16]
- **As GATT Client (Central):** Scans for target Servers, initiates the connection, and performs attribute discovery to map the Server's data structure.[10] Clients often manage data aggregation or control functions.[10]

B. Characteristic Properties and Data Flow Mechanisms

Characteristic properties govern data exchange and delivery guarantees [13]:

- **Read/Write Operations:** Always Client-initiated, suitable for on-demand data polling (Read) or configuration changes (Write).[13]
- **Notify (Unacknowledged Push):** Server-initiated push to a subscribed Client. Notifications are *unacknowledged* [13], maximizing throughput and minimizing latency for continuous data streaming, at the expense of guaranteed delivery.
- **Indicate (Acknowledged Push):** Server-initiated push requiring an explicit acknowledgement from the Client . Indications ensure guaranteed delivery, necessary for critical state changes, but introduce latency overhead that reduces maximum throughput.[13]

Operation Type	Initiator	Data Direction	Purpose	Reliability	Latency/Throughput
Read	Client	Server ⇒ Client	On-demand data polling	High	Medium (Request/Response Cycle)
Write	Client	Client ⇒ Server	Sending commands/Configuration	High	Medium (Request/Response Cycle)
Notify	Server	Server ⇒ Client	Streaming, continuous updates	Low (Unacknowledged)	Highest Throughput

Indicate	Server	Server ⇒ Client	Critical state updates	High (Acknowledged)	High Latency (ACK overhead)

C. Figures and Supplemental Multimedia Materials

The technical complexity of GATT's layered structure is significantly clarified through visual aids, which are recommended as accompanying figures or supplemental materials:

- **Figure 1. Hierarchical Data Structure:** A diagram illustrating the nested relationship between Services, Characteristics, and Descriptors, visually mapping the data organization on the ESP32 Server.[8]
- **Figure 2. State Flowchart for Connection:** A flowchart or sequence diagram demonstrating the transitions: Scanning $\rightarrow$ Advertising $\rightarrow$ Connection Establishment $\rightarrow$ Service Discovery, clarifying the responsibilities of the ESP32 in Central versus Server roles.[2]
- **Multimedia 1. Reliability Handshake Visualization:** A video or protocol sequence chart contrasting Notify (unacknowledged push) and Indicate (acknowledged push) [13], visually reinforcing the engineering trade-off between throughput and guaranteed delivery.[13]

IV. Connectionless Data Dissemination (Advertising)

This method leverages the fundamental discovery mechanism (GAP) to broadcast data, bypassing the connection overhead entirely .

A. Broadcaster and Observer Mechanics

The ESP32 acts as a **Broadcaster** (Advertiser), transmitting small data packets repeatedly.[6] Other devices act as **Observers** (Scanners), passively receiving these transmissions . The primary advantage is low latency and high scalability (unlimited passive observers), but the

payload is limited (typically $\leq$ 31 bytes), and reliability is not guaranteed.[16]

B. Structure of Advertising Data (AD Structures)

Advertising packets adhere to a strict structure [16]:

1. **Length Byte:** Specifies the length of the structure, excluding itself.[16]
2. **Data Type Byte (ADType):** Identifies the type of data (e.g., 0x09 for the device name).[16]
3. **Raw Data:** The variable data payload.[16] This structure enables the use of AD types, such as Manufacturer Specific Data (ADType 0xFF), for broadcasting proprietary, custom binary data in "micro-protocols".[16]

The connectionless approach results in a lower end-to-end delay compared to connected methods [18], making it preferred for sporadic, time-critical status bursts.

C. Figures and Supplemental Multimedia Materials

The efficiency and limitations of advertising are best conveyed through visualization:

- **Figure 3. Advertising Packet Structure:** A diagram illustrating the fixed component structure of the Advertising Data (AD) packet (Length Byte, Data Type Byte, Raw Data Payload), demonstrating how information is contained within the 31-byte limit.[16]
- **Multimedia 2. Unidirectional Flow Visualization:** An animation or flowchart demonstrating the one-to-many, stateless data flow: the ESP32 Broadcaster transmits continuously [6], and all Observers receive the data simultaneously, without a return path or acknowledgment.[6]

V. Standardized Proximity Signaling (Beacons)

Beacons are a specialized application of connectionless broadcasting that uses standardized payload formats (iBeacon and Eddystone) to enable system-level contextual interpretation by host operating systems (e.g., mobile devices).[20]

A. iBeacon Protocol

iBeacon utilizes a strictly defined 30-byte advertising packet, supported by the ESP32.[21] It includes a fixed preamble and critical identifier fields [8]:

- **Proximity UUID (16 bytes):** Unique identifier for the organization.[8]
- **Major/Minor Values (2 bytes each):** Identifiers for specific groups or individual devices.[8]
- **Measured Power (1 byte):** The Received Signal Strength Indication (RSSI) at 1 meter, used by the Client to calculate distance and infer proximity.[8]

B. Eddystone Protocol

Eddystone, also supported by the ESP32 [23], offers flexible frame types:

- **Eddystone-URL:** Transmits a compressed Uniform Resource Identifier (URI) for proximity-based web content.
- **Eddystone-UID:** Functions as a generic identifier, similar to iBeacon.
- **Eddystone-TLM (Telemetry):** Broadcasts metadata about the beacon itself, such as battery voltage and temperature.[23]

C. OS Integration and Functional Enablement

Adherence to standardized beacon payloads enables deep operating system integration.[15] For example, iOS utilizes its Core Location framework to monitor continuously for "beacon region crossing events" (entering or exiting proximity) based on the UUID, Major, and Minor fields, even when the application is backgrounded or closed.[20] This feature, critical for commercial tracking and proximity marketing, relies solely on the standardized payload structure.[15]

D. Figures and Supplemental Multimedia Materials

The practical, system-level function of beacons is best illustrated through dynamic content:

- **Figure 4. iBeacon Payload Mapping:** A detailed graphic mapping the 30-byte fixed structure of the iBeacon packet, highlighting the positions of the Proximity UUID, Major, Minor fields [8], and the Tx Power byte used for proximity calculation.[22]
- **Multimedia 3. Location-Based Monitoring Demonstration:** A video demonstrating how a host application utilizes the OS's native location services to trigger an event (e.g., a notification) upon entering a pre-defined beacon region, even when the app is closed.[15]

VI. Multi-Hop Networking and Control (BLE Mesh)

Bluetooth Mesh addresses the scalability and range limitations of P2P communication by allowing individual devices (nodes) to relay messages across multiple hops, creating a self-healing, many-to-many topology.[4]

A. Mesh Architecture and Node Roles

Mesh network coverage is dramatically extended through the use of intelligent nodes.[4] The ESP32 implements Mesh primarily through the ESP-IDF, incorporating the Zephyr Bluetooth Mesh stack [4]:

- **Relay Node:** Repeats mesh messages to extend physical coverage.[4]
- **Proxy Node:** Bridges the Mesh network (which uses broadcast flooding) to standard GATT-connected devices (e.g., smartphones).[4]
- **Low Power Node (LPN):** Battery-powered devices that maximize sleep time, communicating only periodically with a dedicated **Friend Node**.[4]
- **Friend Node:** A mains-powered device that buffers messages for its associated LPNs, enabling substantial power savings.[4]

B. Interoperability and Mesh Models

Interoperability is ensured by **Mesh Models**, standardized building blocks defining common device behaviors and states.[3]

- **Generic Models:** Describe fundamental state management functions. Examples include the Generic OnOff Client and Server Models [9], which enable remote control of boolean states (e.g., smart lighting control using the ESP32 as an OnOff Server [4]).

While providing vast scalability (hundreds of nodes), data transfer speed over Mesh is slower than a direct BLE P2P connection due to the overhead of multi-hop relaying and encryption. Mesh is best suited for control signals (where latency is acceptable) rather than high-bandwidth data streaming.[19]

C. Figures and Supplemental Multimedia Materials

The functionality of multi-hop networking and power-saving roles necessitates strong visualization:

- **Figure 5. Mesh Topology and Relay Path:** A diagram illustrating a message propagating across multiple ESP32 nodes (Relay Nodes), visually demonstrating how range is extended through hopping.[4]
- **Figure 6. LPN and Friend Node Interaction:** A graphic detailing the symbiotic relationship between a battery-powered Low Power Node (LPN) and its mains-powered Friend Node [4], explaining the mechanism for deep-sleep power optimization.[4]
- **Multimedia 4. Provisioning Workflow:** A step-by-step video tutorial demonstrating the process of "provisioning" an unprovisioned ESP32 node into the Mesh network, typically involving a mobile application (e.g., nRF Mesh) establishing a temporary GATT connection to exchange network credentials.[3]

VII. Advanced ESP32 Architectures and Interoperability

The ESP32's capabilities extend to more complex networking configurations.

A. Dual-Role Operation: Central and Peripheral

The ESP32 is capable of operating simultaneously as both a BLE Central and a BLE Peripheral.[23] This allows the creation of network bridges or gateways that function as a Server to a controlling host (e.g., smartphone) while concurrently acting as a Client to poll data from remote peripheral sensors. This requires robust management of the underlying Bluetooth stack resources.[23]

B. Interfacing with Diverse Host Platforms

Interoperability requires considering the native BLE stack of the host device:

- **Mobile Devices:** Typically function as BLE Clients, interacting with the ESP32 Server via specialized applications.[1]
- **Desktop Operating Systems:** Integration often requires familiarity with the specific native stacks (e.g., BlueZ on Linux). Developers using the ESP-IDF must configure the project environment via menuconfig to ensure required BLE functions are supported.[5]

VIII. Comparative Analysis and Selection Guidance

The architectural decision among the four methods is driven by the primary system constraint: throughput, guaranteed delivery, scalability, or latency. These criteria often represent mutually exclusive design trade-offs.

A. Synthesis of Performance and Reliability Trade-offs

Communication Method	Protocol Base	Primary Topology	Key Characteristic	Latency (Relative)	Effective Throughput	Scalability	Reliability Mechanism
GATT Connected (P2P)	GATT (L2CAP/ATT)	Star (P2P)	Bi-directional, secure	Medium	Highest (via Notify)	Low (Typically < 10)	Indications, Reconn

			data pipes.				ection Strategies
Raw Broadcast (GAP)	GAP (Advertising)	One-to-Many	Rapid, stateless data dissemination.	Very Low	Limited (31 bytes/packet)	Extremely High (Passive)	None (Unacknowledged)
Standard Beacons	GAP (Advertising)	One-to-Many	Contextual triggers, location services.	Low	Low (fixed packet structure)	Extremely High (Passive)	OS-level monitoring, high frequency
Bluetooth Mesh	Mesh Network Layer	Many-to-Many	Multi-hop, self-healing control network.	High (Relaying overhead)	Lowest	Extremely High (100s of nodes)	Acknowledged Models, TTL, Security

B. Architectural Selection Guidance

- **High Bandwidth / Bi-directional Control: Connected GATT** is mandatory. Prioritize Notify for speed or Indicate for data integrity.
- **Ultra-low Latency / Proximity Awareness: Raw Advertising** or **Beacons** are necessary. Use raw advertising for proprietary data, or beacons when required for OS-level proximity triggers.[20]
- **Wide-Area Control Networks: Bluetooth Mesh** is the only viable solution for large-scale, scalable deployments, accepting the corresponding trade-off in instantaneous data throughput .

IX. Conclusion

The ESP32 platform offers a versatile foundation for implementing complex BLE communication architectures. The choice between P2P GATT, Connectionless Advertising, or Mesh networking is a fundamental design decision that directly compromises between reliability, throughput, and system scalability. By systematically evaluating the operational mechanics of the four core methods—and leveraging the visual aids recommended herein to clarify their complex interactions—system architects can optimize ESP32 deployments for specific constraints, from maximizing data speed in simple P2P links to ensuring robust, long-range control over hundreds of nodes in a Mesh network.

Works cited

1. Using ESP32 BLE : 5 Steps - Instructables, accessed November 13, 2025, https://www.instructables.com/Using-ESP32-BLE/
2. ESP32 Bluetooth Low Energy (BLE) on Arduino IDE - Random Nerd Tutorials, accessed November 13, 2025, https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/
3. ESP-BLE-MESH - ESP32 - — ESP-IDF Programming Guide v5.0.4 documentation, accessed November 13, 2025, https://docs.espressif.com/projects/esp-idf/en/v5.0.4/esp32/api-guides/esp-ble-mesh/ble-mesh-index.html
4. ESP32 Bluetooth Mesh Projects: Building Next-Generation IoT Networks w - ThinkRobotics.com, accessed November 13, 2025, https://thinkrobotics.com/blogs/learn/esp32-bluetooth-mesh-projects-building-next-generation-iot-networks-with-self-healing-device-communication
5. Getting Started with BLE on the ESP32, accessed November 13, 2025, https://esp32.com/viewtopic.php?t=1204
6. GAP - impl Rust for ESP32, accessed November 13, 2025, https://esp32.implrust.com/bluetooth/ble/gap.html
7. Introduction - ESP32 - — ESP-IDF Programming Guide v5.5.1 documentation, accessed November 13, 2025, https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/ble/get-started/ble-introduction.html
8. Different Value Types of Characteristics | GATT Protocol | Bluetooth LE | v6.2.0 | Silicon Labs, accessed November 13, 2025, https://docs.silabs.com/bluetooth/6.2.0/bluetooth-gatt/characteristics-value-types
9. Generic OnOff models - Technical Documentation - Nordic Semiconductor, accessed November 13, 2025, https://docs.nordicsemi.com/bundle/ncs-2.9.1/page/nrf/libraries/bluetooth/mesh/gen_onoff.html
10. ESP32 BLE Server and Client (Bluetooth Low Energy) | Random ..., accessed

November 13, 2025, https://randomnerdtutorials.com/esp32-ble-server-client/
11. Services and characteristics - Nordic Developer Academy, accessed November 13, 2025, https://academy.nordicsemi.com/courses/bluetooth-low-energy-fundamentals/lessons/lesson-4-bluetooth-le-data-exchange/topic/services-and-characteristics/
12. data Throughput over BLE vs BLE Mesh - Nordic DevZone, accessed November 13, 2025, https://devzone.nordicsemi.com/f/nordic-q-a/98897/data-throughput-over-ble-vs-ble-mesh
13. [Deprecated] KBA_BT_0102: BLE Basics (master/slave, GATT client/server, data RX/, accessed November 13, 2025, https://community.silabs.com/s/article/x-deprecated-kba-bt-0102-ble-basics-master-slave-gatt-client-server-data-rx-x
14. How to Use Bluetooth(BLE) With ESP32 : 3 Steps - Instructables, accessed November 13, 2025, https://www.instructables.com/How-to-Use-BluetoothBLE-With-ESP32/
15. What is Eddystone Protocol and Specifications - MOKOSmart, accessed November 13, 2025, https://www.mokosmart.com/eddystone-protocol-and-specifications/
16. ESP32 (34) – BLE, raw advertising - lucadentella.it, accessed November 13, 2025, https://www.lucadentella.it/en/2018/03/29/esp32-34-ble-raw-advertising/
17. Looking for Information on the Bluetooth LE "Indicate" Behavior - Stack Overflow, accessed November 13, 2025, https://stackoverflow.com/questions/59430630/looking-for-information-on-the-bluetooth-le-indicate-behavior
18. BluFi BLE Service UUID in violation of BLE standard? - ESP32 Forum, accessed November 13, 2025, https://esp32.com/viewtopic.php?t=36947
19. The Bluetooth Mesh Standard: An Overview and Experimental Evaluation - PMC, accessed November 13, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC6111614/
20. ESP32 BLE Bluetooth Examples Confuse Me - Programming - Arduino Forum, accessed November 13, 2025, https://forum.arduino.cc/t/esp32-ble-bluetooth-examples-confuse-me/1344437
21. ESP32 #73 Arduino Eddystone Beacon Scanner - Hive.blog, accessed November 13, 2025, https://hive.blog/pcbreflux/@pcbreflux/5blo53em
22. iBeacon packets - Knowledge Base - Kontakt.io, accessed November 13, 2025, https://support.kontakt.io/hc/en-gb/articles/4413251561106-iBeacon-packets
23. ESP32 #72: Arduino Eddystone Beacon - YouTube, accessed November 13, 2025, https://www.youtube.com/watch?v=2hhy6houBcU
24. Bluetooth Mesh Models, accessed November 13, 2025, https://www.bluetooth.com/wp-content/uploads/2019/04/1903_Mesh-Models-Overview_FINAL.pdf
25. BLE dualRole Server/Client OnConnect() method overlapp. - Arduino · Issue #1153 · nkolban/esp32-snippets - GitHub, accessed November 13, 2025, https://github.com/nkolban/esp32-snippets/issues/1153