ACL Digital
an ALTEN group company

Understanding the Role of Bluetooth Low Energy (BLE) Mesh Network in Android

Table of contents

Introduction

Bluetooth has been vigorously pursued and developed since its launch in the year 2000 when it was initially aimed to act as a cable replacement technology. Bluetooth technology soon came to dominate computer peripherals and wireless audio products. However, in 2010, Bluetooth LE provided the next progressive step. Its impact has been significant and widely felt, most notably in tablets and smartphones, including wearable technologies, health and fitness, and Smart homes.

Bluetooth Low Energy (BLE) is a wireless communication technology developed to transmit and exchange data within a short-range. It needs little battery power to operate and can swiftly transfer small data packets to compatible devices. BLE is crucial to IoT, like wearables and broadcasting beacons, enabling data transfer through mesh networks in various settings.

Additionally, BLE has several notable features that differentiate it from other available wireless technologies, including robustness, interoperability, latency, ease of use, latency, range, and ultra-low power consumption.

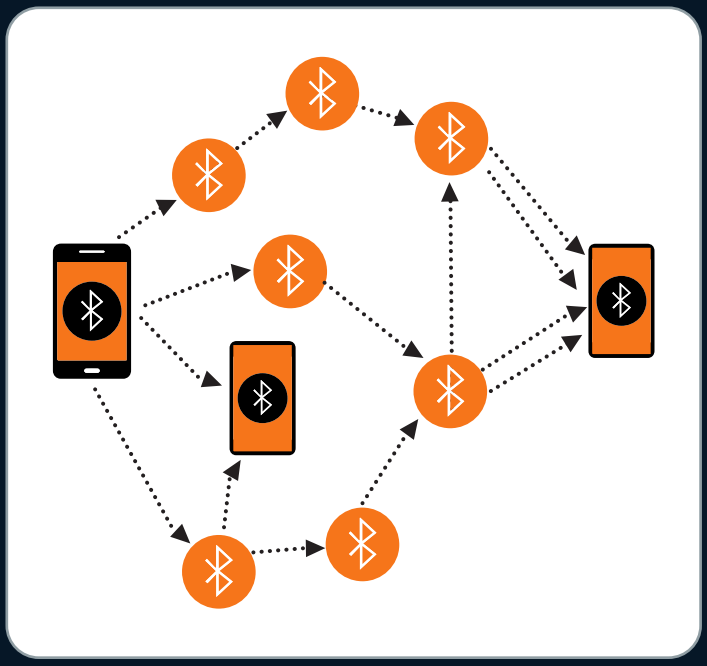
BLE IoT applications are ideal for fitness, healthcare, industrial and home automation, and indoor asset tracking scenarios, given the low-power requirements, infinite scalability, and self-healing reliability of emerging Bluetooth mesh networks.

Bluetooth is a prominent low-power wireless connectivity technology used to broadcast information between devices, transfer data, and stream audio. By introducing mesh networking capability, Bluetooth mesh networking is primed to catalyze further industrial automation, energy management, beacons, Smart city applications, robotics, and other industrial IoT and superior manufacturing solutions.

This white paper provides an overview of the Bluetooth Mesh Network and emphasizes some of its distinctive features. With pin-point configuration of appropriate parameters and proper deployment of the protocol stack, Bluetooth mesh would be able to strengthen the operation of complex networks with numerous devices.

Bluetooth Low Energy (BLE) Mesh Network

BLE Mesh network has many-to-many topology where devices (or nodes) can connect directly, dynamically, and non-hierarchically with every other device (or node) within the same network to efficiently route data from/to clients. Communication is achieved by utilizing messages, and devices can transmit messages to other devices so that the end-to-end communication range is spread far beyond the radio range of each node.

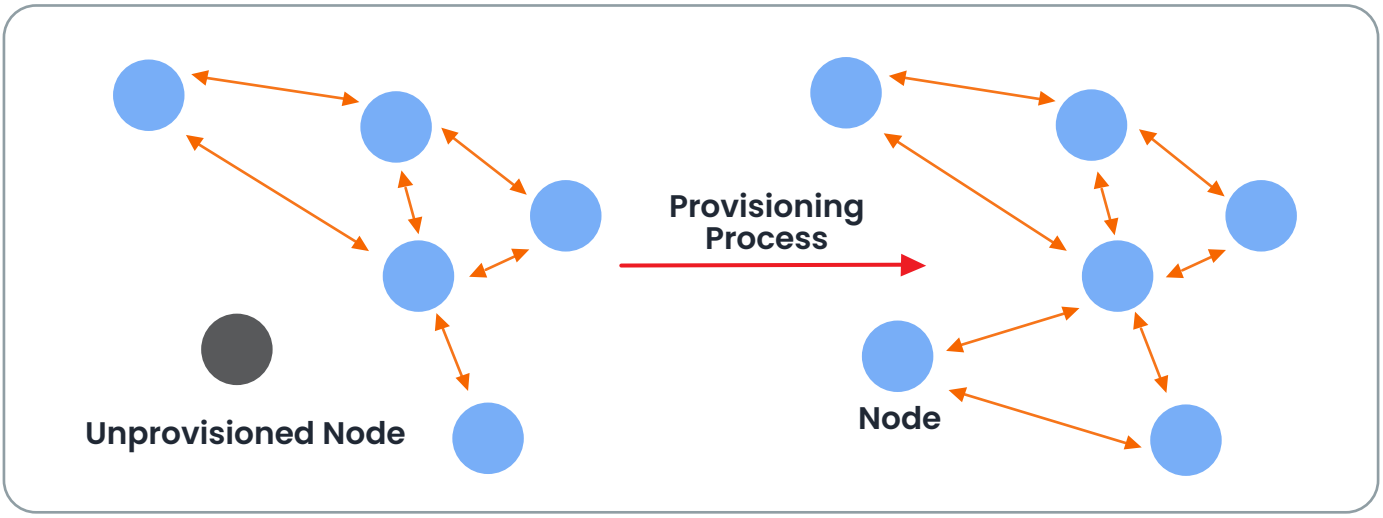


Mesh networks synchronously self-configure and self-organize by enabling dynamic distribution of workloads, particularly if few nodes fail. This contributes to fault-tolerance, self-healing capabilities, and lower maintenance costs.

Mesh Operation Concepts

Devices and Nodes

The devices within the mesh network are called Nodes, and devices outside the mesh are known as 'unprovisioned nodes.' The methodology of transforming an 'unprovisioned node' into a 'node' by adding to the mesh is called 'provisioning.'



BLE Mesh has diverse node types, and you can employ multiple features on these nodes.

Relay Node

This node relays messages for others as intermediate nodes that help mesh networks achieve extensive coverage & reliability.

Proxy Node

The Proxy feature is the ability of a node to relay messages between the GATT (General ATTribute) and advertising bearers. This feature allows devices, such as smartphones that support BLE but not BLE Mesh to communicate with a Mesh network.

Friend Node

This node has a friendly relationship with the Low Power node. A message for a low power node is sent to a related friend node that stores messages for the low power node while it's asleep. When the low power node becomes active, it receives messages from the corresponding friend node.

Low Power Node

This is a power-sensitive node that can skip being active all the time and wake itself as programmed and receive messages at an active time.

Elements

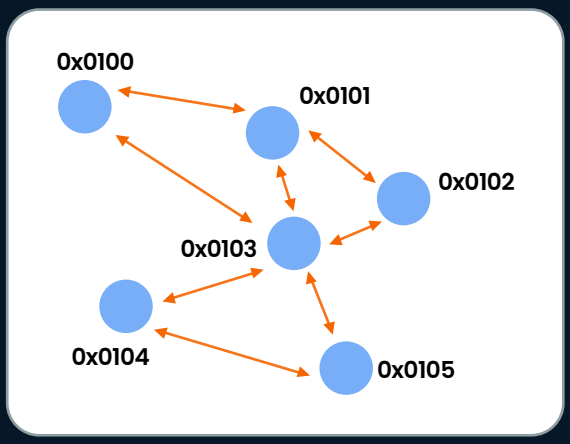
A node may include numerous parts that can be controlled independently. For example, a light fixture may possess multiple light bulbs that can be turned on/off independently. These additional parts of a single node are referred to as elements.

Address

It is essential to have an address to exchange messages in a Bluetooth mesh network. There are three types of addresses:

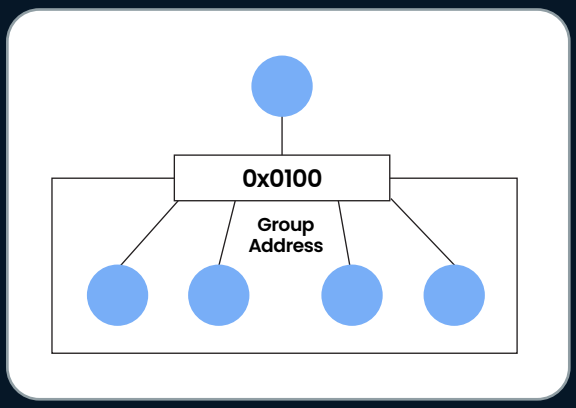
Unicast Address

An address uniquely identifies a single node assigned during the provisioning process.



Group Address

An address used to recognize a group of nodes. A group address usually mirrors a physical grouping of nodes, such as all nodes within a specific room.



Virtual Address

spread across one or more nodes and may be assigned to one or more elements. It acts as a label and takes the form of a 128-bit UUID with which any element can be associated. Virtual addresses are likely to be pre- configured at the time of manufacturing.

Models

A model defines the complete functionality of an element or some essential aspects. It brings together the concepts of states, transitions, bindings, and messages for an element component.

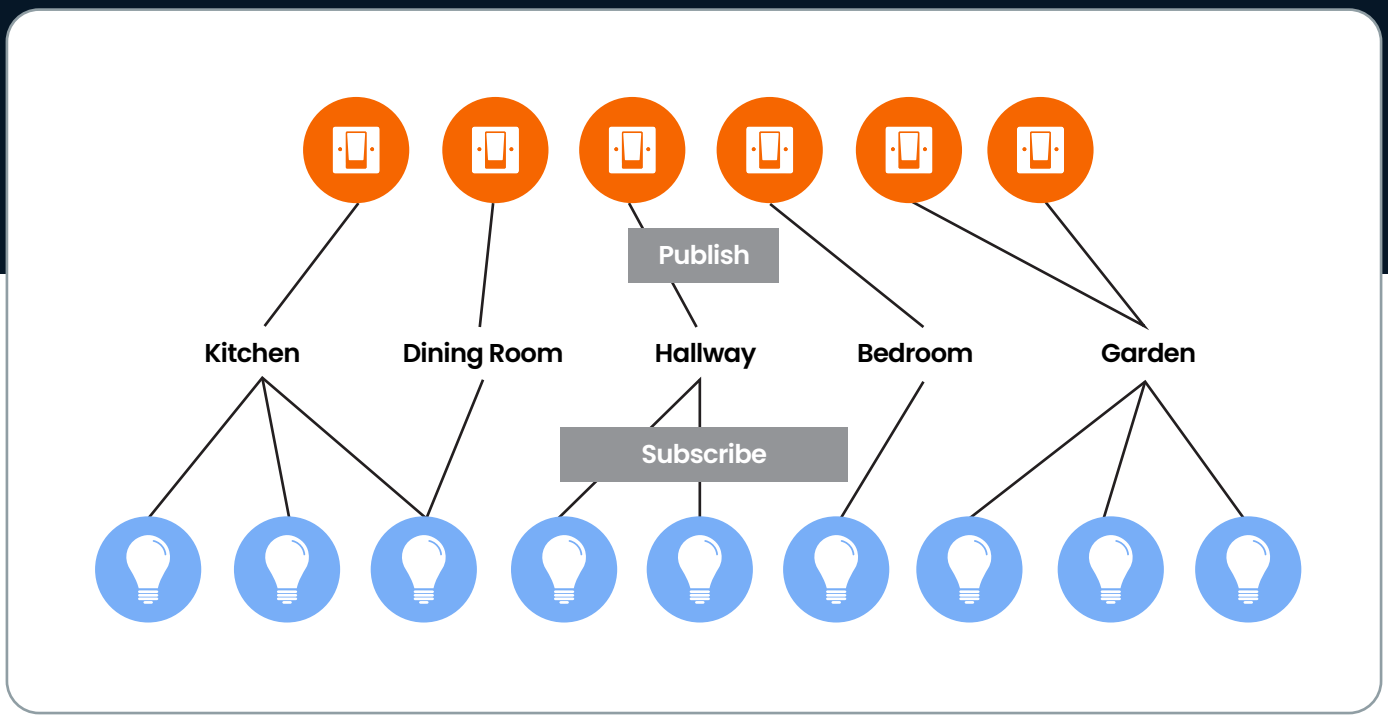
There are three types of Mesh Models – Server Model, Client Model, and Control Model.

Managed Flooding

BLE mesh uses a managed flood operation to transmit messages. A multi-path implementation inherently adds enough redundancy to ensure that a message reaches its destination. It prevents the devices from relaying previously received messages by adding all messages to a cached list. Every message contains a Time to Live (TTL) value that restricts the number of times a message can be relayed (up to a maximum of 126 times) by a device.

Publish/Subscribe Communication

Devices in a BLE mesh network communicate using a publish/subscribe model, where publishers can post a particular matter, and subscribers can subscribe to one or more topics of interest. A node in a Bluetooth Mesh network can subscribe to multiple addresses (kept in the subscriber list) and publish to one specific address (stored in the publish address).

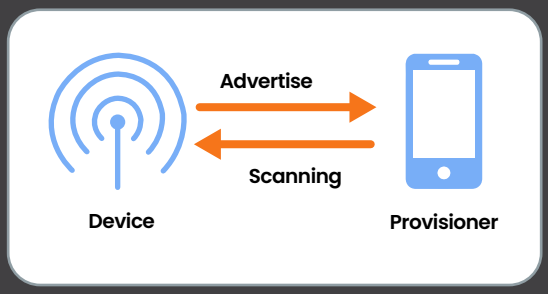


Provisioning

The process by which a device affixes the mesh network and turns it into a node is known as provisioning. The device used to push the provisioning process is known as the Provisioner. The provisioning process involves five steps,

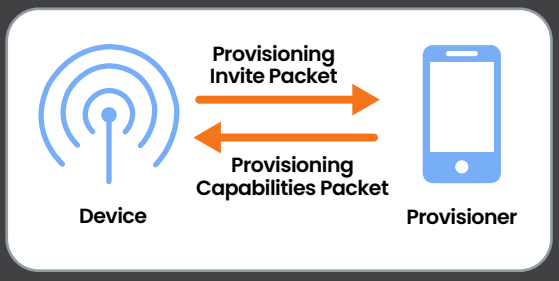
Beaconing

Unprovisioned device advertises its availability to be provisioned by using the advertising packets. A common way to trigger this process is by a defined sequence of button presses on the unprovisioned device.



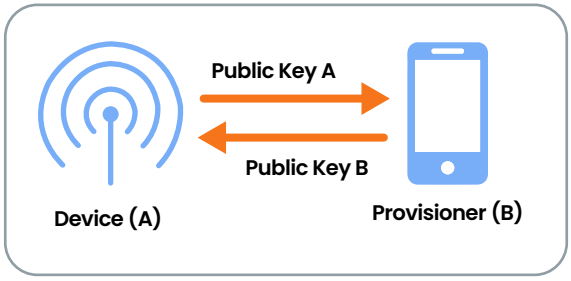
Invitation

The provisioner sends an invite to the device to be provisioned in the form of a Provisioning Invite PDU. The unprovisioned device responds with information about itself in a Provisioning Capabilities PDU.



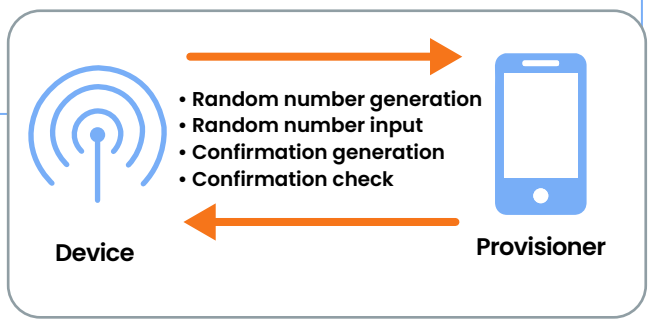
Public Key Exchange

Public keys are exchanged between the provisioner and the unprovisioned device. It is done directly over the BLE or via an out-of-band (OOB) channel.



Authentication

The unprovisioned device outputs a random, single or multi-digit number to the user in a particular form, using an action appropriate to its capabilities. E.g., it might flash an LED several times. The user enters the new device's digit(s) output into the Provisioner. A cryptographic exchange occurs between the two devices, involving the random number, to complete the authentication of each of the two devices to the other.



Distribution of the Provisioning Data - After authentication is complete, a session key is acquired by both the devices from their private keys and the interchanged peer public keys. The session key is then utilized to secure the connection to exchange additional provisioning data, including the network key (NetKey), a device key, a security parameter known as the IV index, and a unicast address assigned to the provisioned device by the provisioner. After this phase, the unprovisioned device will be known or called a node.

Mesh Security Keys

BLE Mesh has three types of security keys that offer security to different mesh elements and execute a critical capability in mesh security.

Dev Key

The device key authenticates and encrypts communications between the Provisioner & a node.

Net Key

The flexibility of the mesh network is offered by the actuality that any particular node can be put together in a specific form of configuration to use just single or multiple networks simultaneously. It facilitates us to utilize one or more subnets by specifying subnet NetKeys. Also, sub-networks are batches of nodes that can interact with one other at the network layer.

Alternatively, a node can be categorised to two or more separate mesh networks.

App Key

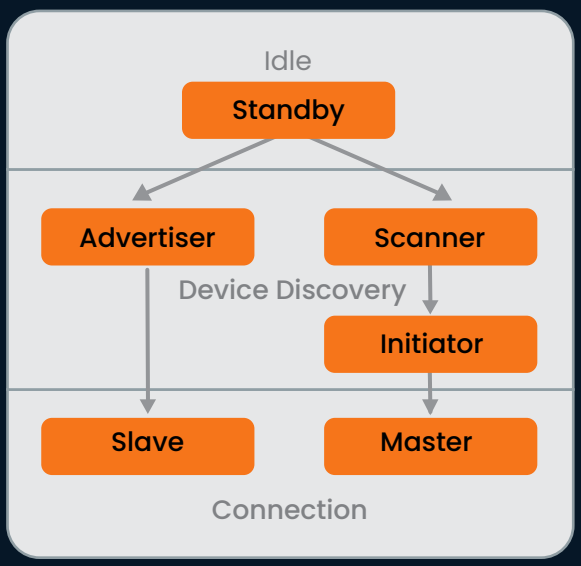
A mesh network can comprise more than one application, each with a distinctive AppKey.

BLE Communication in Android

For BLE-enabled devices to transmit data to each other, they must first form a communication channel. The device will scan for close by BLE devices. Once a device is found and a connection is made, data transfer will happen with the connected device based on the existing services and features.

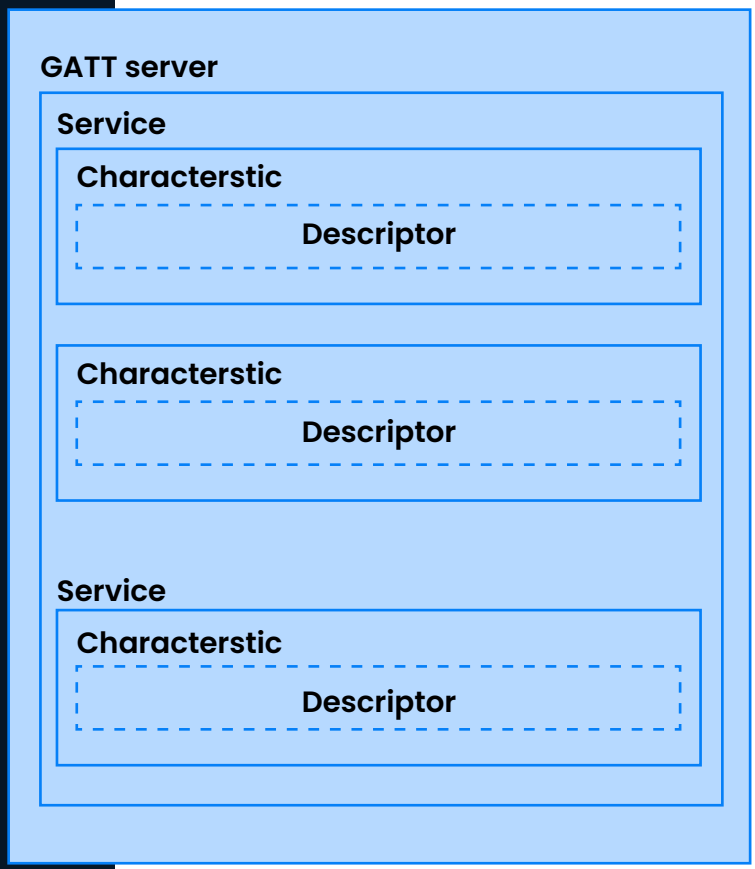
Generic Access Profile (GAP)

The GAP layer of the Bluetooth low-energy protocol stack is accountable for connection-related functionality. It describes how BLE-enabled devices can make themselves available and how two devices can communicate directly.



Generic Attribute Profile (GATT)

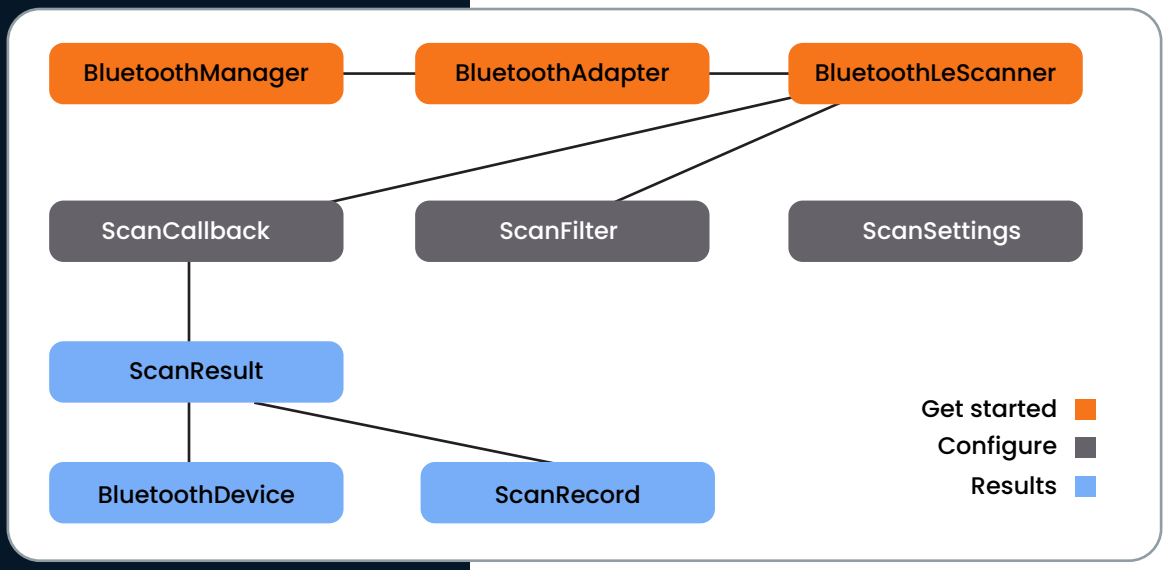
GATT layer of the BLE protocol stack defines how two BLE devices transfer data back and forth using concepts called services and characteristics. It uses a generic data protocol called the Attribute Protocol (ATT), which stores features, services, and corresponding data in a lookup table using 16-bit IDs for every entry in the table.



Android BLE API

Device Discovery

- **Bluetooth Manager –** High-level manager used to acquire an instance of a BluetoothAdapter and perform overall Bluetooth Management.
- **Bluetooth Adapter –** Deliver information on the on/off state of the Bluetooth hardware, authorizes to query for Bluetooth devices bonded to Android, & allows the ability to start BLE scans.
- **App Key –** A mesh network can comprise more than one application, each with a distinctive AppKey.
- **BluetoothLeScanner –** Provided by the BluetoothAdapter class, contains methods to perform scan-related operations for Bluetooth LE devices.
- **ScanFilter –** Allows constricting scan results to target specific devices during a BLE scan. A particular use case for applications is to filter BLE scan outcomes based on the BLE devices' advertised service UUIDs.
- **ScanSettings –** Specifies parameters about the scanning behavior such as scan mode, callback type, and threshold of advertising packets.
- **ScanResult –** Defines a BLE scan outcome acquired via BLE scan and includes the BLE device MAC address, RSSI (signal strength), & advertisement data. The getDevice() method reveals the Bluetooth device that may consist of the BLE device's name and allows the app to connect to it.
- **BluetoothDevice –** Represents a physical Bluetooth device that the app can connect to, bond (pair) with, or both. The essential information provided by this class includes name, address, category, & bonding state. The BluetoothAdapter.getRemoteDevice (String address) returns a Bluetooth Device for the given Bluetooth MAC address.



Scan for a BLE device with the BluetoothLeScanner object obtained from the BluetoothAdapter.

```
private val bluetoothAdapter: BluetoothAdapter by lazy {
    val bluetoothManager = getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
    bluetoothManager.adapter
}
private val bleScanner by lazy {
    bluetoothAdapter.bluetoothLeScanner
}

private fun scanLeDevice() {
    bleScanner.startScan(scanFilter, scanSettings, leScanCallback)
}

private val leScanCallback = object : ScanCallback() {
    override fun onScanResult(callbackType: Int, result: ScanResult) {
        val device: BluetoothDevice = result.device
    }
}
```

Call StartScan method equipped with the scanSettings, filters, and a scanCallback entity to acquire found devices.

The connectGatt() method on a BluetoothDevice will be utilized to form a connection with a BLE device. It returns the BluetoothGatt object, ideal for GATT-related operations like reading and writing.

```
private var bluetoothGatt: BluetoothGatt? = null

fun connectToDevice(device: BluetoothDevice){
    // connect to the GATT server on the device
    bluetoothGatt = device.connectGatt(this, false, bluetoothGattCallback)
}

private val bluetoothGattCallback = object : BluetoothGattCallback() {
    override fun onConnectionStateChange(gatt: BluetoothGatt?, status: Int, newState: Int) {
        if (newState == BluetoothProfile.STATE_CONNECTED) {
            // successfully connected to the GATT Server
        } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
            // disconnected from the GATT Server
        }
    }
}
```

Connected Device Interaction

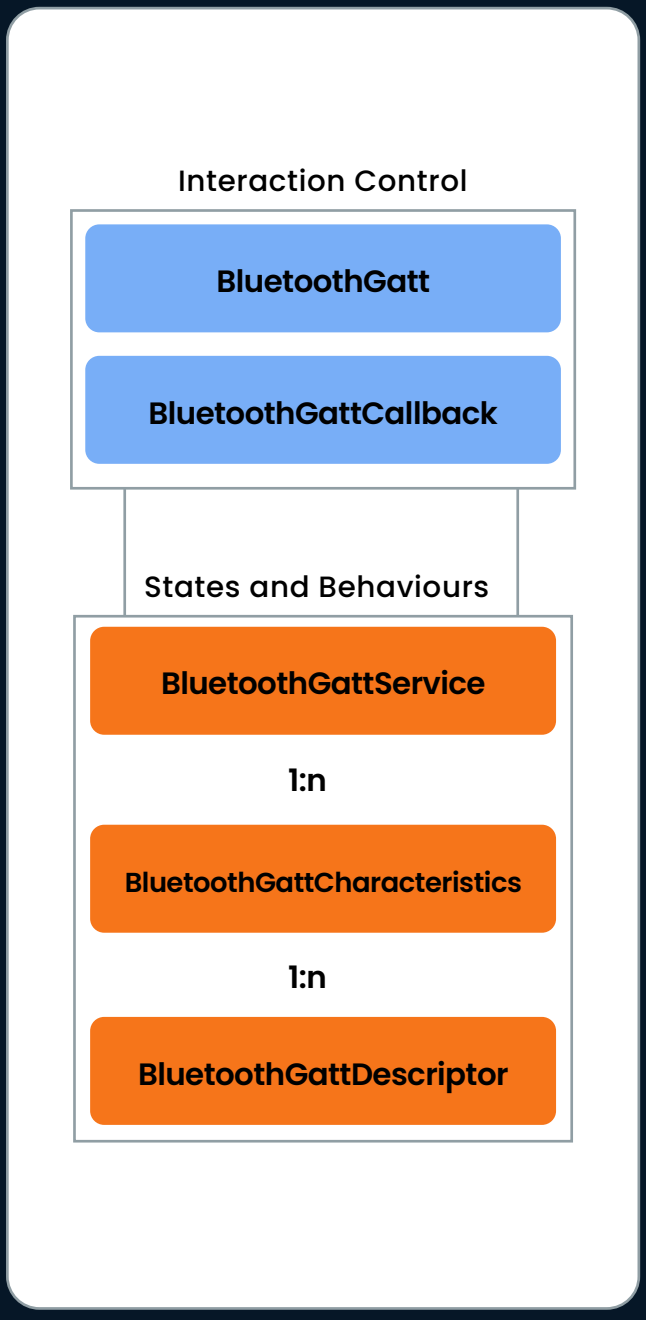
- **BluetoothGatt**
  This allows us to perform service discovery, connection teardown, request MTU updates, and access the services and characteristics present on the BLE device.

- **BluetoothGattCallback**
  The main interface the application has to implement to receive callbacks for most BluetoothGatt-related operations like writing, reading, or getting notified about incoming indications or notifications.

- **BluetoothGattService**
  Set of provided features & associated behaviors to interact with the device. Each service contains a collection of characteristics. E.g., you could have a "Heart Rate Monitor" service that includes "heart rate measurement" & other features.

- **BluetoothGattCharacteristics**
  An entity that includes meaningful data can typically be read or written. Uses permission properties (read, write, notify, indicate) to get a value. It contains a single value and 0-n descriptors that describe the characteristic's importance. E.g., the Serial Number String characteristic.

- **BluetoothGattDescriptor**
  Descriptors are defined attributes that describe a characteristic value. For example, a descriptor might specify a human-readable description, an acceptable range for a characteristic's value, or a unit of measure specific to a characteristic's value.



Once the device is connected, you will find the services and attributes available on the connected BLE device by calling the discoverServices() method using the BluetoothGatt object. The result of the service discovery will be delivered via BluetoothGattCallback's onServicesDiscovered() method.

```
fun discoverServices(){
    //Discover services via BluetoothGatt object saved from a successful connection attempt
    bluetoothGatt.discoverServices()
}

private val bluetoothGattCallback = object : BluetoothGattCallback() {
    override fun onConnectionStateChange(gatt: BluetoothGatt?, status: Int, newState: Int) {
       //Manage BLE device connection state changes
    }

    override fun onServicesDiscovered(gatt: BluetoothGatt, status: Int) {
        //Called when services are discovered
    }
}
```

Once the app has connected to a GATT server and located services, it can read and write attributes wherever it is supported. The GATT service delivers a list of characteristics read from the device.

```
// Iterate through the supported GATT Services/Characteristics.
private fun displayGattServices(gattServices: List<BluetoothGattService>?) {
    if (gattServices == null) return

    // Loops through available GATT Services.
    gattServices.forEach { gattService ->
        val serviceUuid = gattService.uuid.toString()
        val gattCharacteristics = gattService.characteristics

        // Loops through available Characteristics.
        gattCharacteristics.forEach { gattCharacteristic ->
            val characteristicUuid = gattCharacteristic.uuid.toString()
        }
    }
}
```

To query the data from the BLE device, call the readCharacteristic() method on the BluetoothGatt object, passing in the BluetoothGattCharacteristic, which is necessary. The results will be sent to the onCharacteristicRead() method in BluetoothGattCallback. If the read is successful, you can access the data by calling the getValue() method in BluetoothGattCharacteristic

```
fun readCharacteristic(characteristic: BluetoothGattCharacteristic) {
    bluetoothGatt.readCharacteristic(characteristic)
}

private val bluetoothGattCallback = object : BluetoothGattCallback() {
    override fun onCharacteristicRead(
        gatt: BluetoothGatt,
        characteristic: BluetoothGattCharacteristic,
        status: Int) {
            if (status == BluetoothGatt.GATT_SUCCESS) {
                val data: ByteArray? = characteristic.value
            }
        }
}
```

To send data to the BLE device, call the writeCharacteristic() method on the BluetoothGatt object, setting the fields - setWriteType() and setValue() and passing in the BluetoothGattCharacteristic. Depending on the write type, BluetoothGattCallback's onCharacteristicWrite() callback is guaranteed for WRITE_TYPE_DEFAULT (write with response), but not for WRITE_TYPE_NO_RESPONSE (write without response).

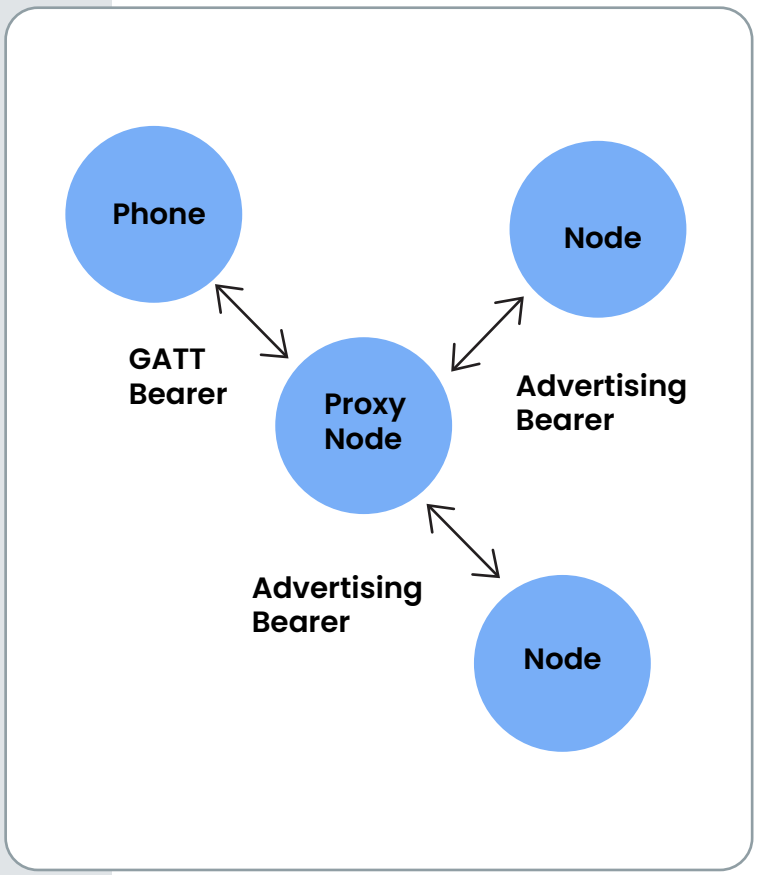
```
fun writeCharacteristic(characteristic: BluetoothGattCharacteristic, payload: ByteArray) {
    //Set the write type to WRITE_TYPE_DEFAULT or WRITE_TYPE_NO_RESPONSE
    characteristic.writeType = writeType
    characteristic.value = payload
    bluetoothGatt.writeCharacteristic(characteristic)
}

private val bluetoothGattCallback = object : BluetoothGattCallback() {

    override fun onCharacteristicWrite(
        gatt: BluetoothGatt,
        characteristic: BluetoothGattCharacteristic,
        status: Int) {
        //Called when writeType is set to WRITE_TYPE_DEFAULT
    }
}
```

Mesh Proxy node

Smartphones do not inherently reinforce Bluetooth mesh and cannot efficiently communicate with mesh-enabled end nodes. Proxy nodes permit BLE devices that do not include a Bluetooth mesh stack to interact with nodes in a mesh network. Any node that supports the Proxy feature can act as the interface for a smartphone over a GATT connection.

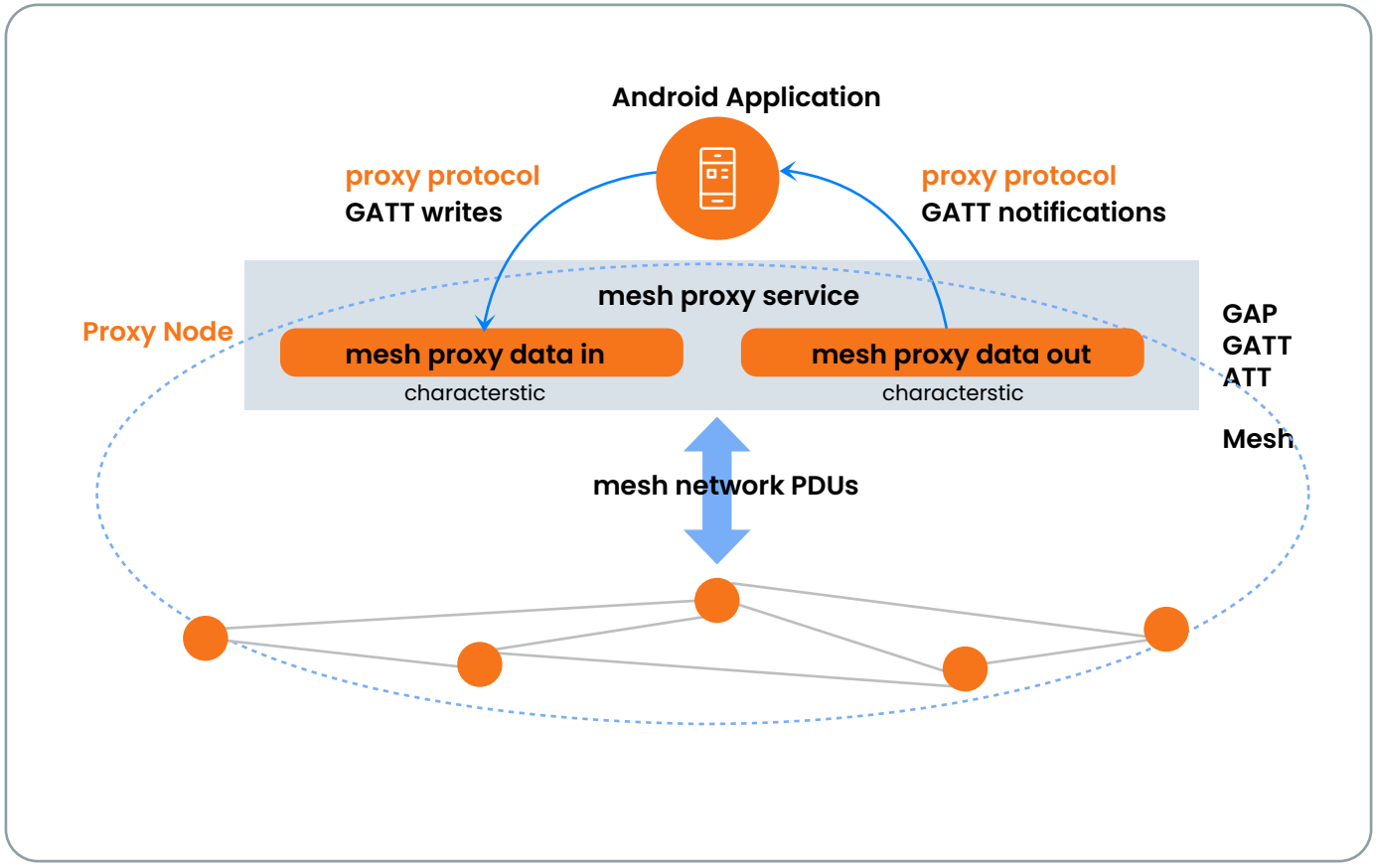


The proxy protocol is ideal for a connection-enabled device (GATT). The protocol is developed on top of GATT and authorizes a device to read and write proxy protocol PDUs from GATT attributes exposed by the proxy node. The proxy node executes the translation between proxy protocol PDUs and the mesh PDUs.

The proxy node is a mesh node that runs ATT, GATT, and GAP. The node includes a Bluetooth Low Energy service called Mesh Proxy Service. It has two elements,

Mesh proxy data in - writes data to the proxy node data in characteristic to inject a message into the mesh network.

Mesh proxy data out - acquires notification from proxy node data out characteristic, encapsulating mesh messages.



Conclusion

BLE mesh network assists in extending range, resilience, rapid adoption, and standardization, permitting more devices to be compatible. It offers default security, including distribution, encryption, and equipment authentication.

As a highly flexible, energy-efficient, and secure solution, BLE mesh can be used in conventional networking applications like industrial automation, home automation, Smart lighting, asset tracking, proximity detection, Smart healthcare, environmental monitoring, etc.

BLE mesh will improve the competence of Bluetooth technology in IoT applications facilitating the transfer of data through mesh networks in various settings.

References

- https://blog.csdn.net/qq_27114397/article/details/107127016
- https://blog.rtone.fr/en/bluetooth-mesh