Mục lục

[**2/ Bluetooth Low Energy – BLE:** 3](#_Toc535279109)

[**2.1/ Giới thiệu BLE:** 3](#_Toc535279110)

[**2.2/ Các phân lớp trong BLE:** 3](#_Toc535279111)

[*2.2.1/ Physical Layer:* 4](#_Toc535279112)

[*2.2.2/ Link Layer:* 4](#_Toc535279113)

[*2.2.3/ Giao thức ATT:* 6](#_Toc535279114)

[*2.2.4 / Generic Access Profile (GAP):* 7](#_Toc535279115)

[*2.2.5/ GATT – Generic Attribute Profile:* 10](#_Toc535279116)

[**2.3/ Tổng kết các hoạt động cơ bản của BLE:** 14](#_Toc535279117)

[**3/ Mạng Bluetooth Mesh:** 15](#_Toc535279118)

[**3.1/ Các khái niệm cơ bản về mạng Bluetooth mesh:** 15](#_Toc535279119)

[*3.1.1/ Giới thiệu mạng mesh:* 15](#_Toc535279120)

[*3.1.2/ Đặc điểm của mạng Bluetooth Mesh:* 15](#_Toc535279121)

[*3.1.3/ Node và thiết bị:* 16](#_Toc535279122)

[*3.1.4/ Elements:* 16](#_Toc535279123)

[*3.1.5/ Message:* 17](#_Toc535279124)

[*3.1.6/ Addresses:* 18](#_Toc535279125)

[*3.1.7/ Publish/Subscribe:* 18](#_Toc535279126)

[*3.1.8/ States:* 19](#_Toc535279127)

[*3.1.9/ Message và State:* 19](#_Toc535279128)

[*3.1.10/ Models:* 19](#_Toc535279129)

[*3.1.11/ Generics:* 20](#_Toc535279130)

[*3.1.12/ Scenes:* 21](#_Toc535279131)

[*3.1.13/ Provision:* 21](#_Toc535279132)

[*3.1.14/ Node:* 29](#_Toc535279133)

[*3.1.15/ Proxy Node:* 30](#_Toc535279134)

[*3.1.16/ Relay:* 30](#_Toc535279135)

[*3.1.17/ Kiến trúc Mesh System:* 31](#_Toc535279136)

[**3.2/ nRF SDK for Mesh** 33](#_Toc535279137)

[*3.2.1/ Cấu trúc của nRF SDK for Mesh:* 33](#_Toc535279138)

[*3.2.2/ Cấu trúc dữ liệu SET message:* 34](#_Toc535279139)

[*3.2.3/ Cấu trúc dữ liệu của message khi nhận tại Node server:* 35](#_Toc535279140)

[*3.2.4/ Khởi tạo:* 35](#_Toc535279141)

[*3.2.4/ Cách gửi một SET message phía Client:* 39](#_Toc535279142)

[*3.2.5/ Các sự kiện xảy ra khi nhận một message:* 39](#_Toc535279143)

[**4/ Hiện thực và kết quả:** 40](#_Toc535279144)

[**4.1/ Hiện thực:** 40](#_Toc535279145)

[**4.2/Kết quả:** 41](#_Toc535279146)

[**4.3/ Các mặt hạn chế:** 41](#_Toc535279147)

[**4.4/ Nhận xét về Bluetooth Mesh:** 42](#_Toc535279148)

[**4.5/ Những khó khăn gặp phải:** 42](#_Toc535279149)

[**4.6/ Hướng phát triển:** 43](#_Toc535279150)

[**Tài liệu tham khảo:** 43](#_Toc535279151)

# **2/ Bluetooth Low Energy – BLE:**

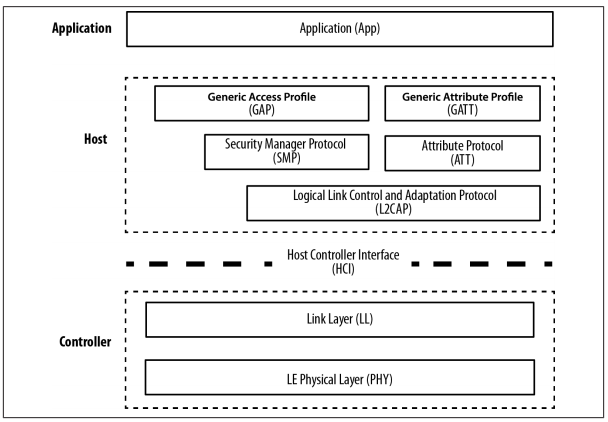
## **2.1/ Giới thiệu BLE:**

- BLE là chuẩn giao tiếp thừa hưởng những đặc điểm của Bluetooth thông thường nhưng công suất tiêu thụ lại thấp hơn rất nhiều. Với công suất tiêu thụ rất thấp, các board BLE thường có kích thước rất nhỏ nên có thể tích hợp vào trong nhiều đồ vật khác nhau.

- Một nhược điểm của BLE đó là tốc độ truyền dữ liệu khá thấp do công suất tiêu thụ thấp, vì thế BLE không thể ứng dụng cho streaming hoặc music/video. Tuy nhiên, yêu cầu về tốc độ truyền trong giao tiếp giữa các thiết bị không cần quá nhiều và liên tục, BLE hoàn toàn có thể đáp ứng được.

- Bluetooth Low Energy là lớp dưới cùng của Bluetooth Mesh. Các hoạt động về phát sóng và truyền dữ liệu của Bluetooth Mesh đều bằng BLE.

## **2.2/ Các phân lớp trong BLE:**

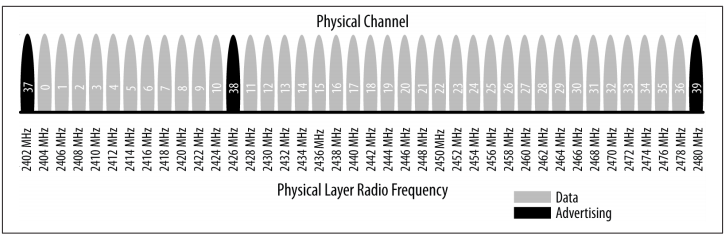


Hình 2.1: Các lớp giao thức trong BLE

### *2.2.1/ Physical Layer:*

- Lớp Physical bao gồm các module giao tiếp analog, chuyển tín hiệu analog thành tín hiệu digital.

- BLE giao tiếp bằng dải tần 2.4GHz và được chia thành 40 kênh từ 2400GHz đến 2480GHz. Kênh 37,38,39 dùng để advertising, thiết lập kết nối và gửi dữ liệu broadcast, các kênh còn lại dùng để gửi dữ liệu ứng dụng.



Hình 2.2: Các kênh giao tiếp trong BLE

### *2.2.2/ Link Layer:*

- Lớp Link tương tác trực tiếp với lớp Physical. Lớp Link có một ràng buộc hard real-time vì lớp Link phải tuân thủ các yêu cầu về thời gian định nghĩa trong BLE Specification.

- Một thiết bị BLE trong lớp Link có hai vai trò Master và Slave. Thiết bị khởi tạo kết nối sẽ là Master và thiết bị advertise và chấp nhận kết nối là Slave. Một master có thể kết nối với nhiều slave và slave có thể kết nối với nhiều master. Thông thường, smartphone và tablet là master, các thiết bị hạn chế về bộ nhỏ, đơn giản, hạn chế về bộ nhớ như cảm biến là slave.

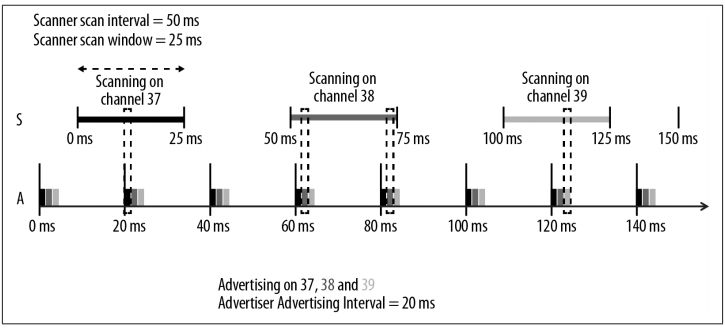
- Nhận dạng một thiết bị Bluetooth là một giá trị 48 bits duy nhất giữa các tihết bị và được gọi là Bluetooth device address, giống như địa chỉ MAC.

- BLE có hai loại packet là advertising và data packet. Advertising thực hiện hai mục đích:

* Broadcast dữ liệu mà không cần thiết lập kết nối đầy đủ.
* Phát hiện và kết nối với Slave.

- Mỗi advertising có 31 byte dữ liệu và được gửi theo một tần suất được định nghĩa bởi Advertising interval, trong khoảng 20 ms đến 10.24s. Giá trị Interval càng ngắn thì advertising packet sẽ được gửi nhiều hơn và tăng khả năng nhận packet của thiết bị khác, nhưng lại phải đánh đổi bằng việc gia tăng năng lượng tiêu thụ.

- Tại thiết bị nhận có một giá trị là Scanning interval quy định khoảng thời gian mà thiết bị sleep và thức dậy listen packet. Tương tự như Advertising interval, Scanning interval càng ngắn thì khả năng nhận packet sẽ cao đồng thời làm tăng điện năng.



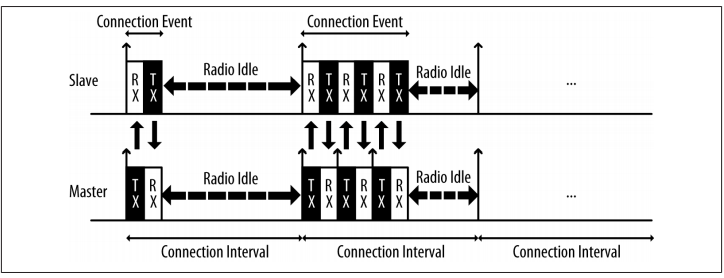
Hình 2.3: Advertising và Scanning interval

- Connection: kết nối là một chuỗi trao đổi dữ liệu giữa Slave và Master trong một khoảng thời gian định trước. Quá trình kết nối gồm có:

* Master bắt đầu scan để tìm các thiết bị slave đang advertising. Advertising packet có thể được lọc bằng Bluetooth device address.
* Khi phát hiện slave đang advertising phù hợp, master gửi một connection request packet đến slave.
* Slave phản hồi connection request packet và thiết lập kết nối.

- Một lần trao đổi dữ liệu là một connection event, Connection có 3 tham số:

* Connection Interval: thời giữa hai connection event có giá trị từ 7.5ms đến 4s. Interval càng thấp thì thông lượng càng cao nhưng điện năng tiêu thụ gia tăng
* Slave latency: số connection event mà slave có thể bỏ qua mà không ảnh hưởng đến kết nối.
* Connection supervision timeout: Thời gian tối đa giữa hai lần nhận packet trước khi kết nối bị ngắt.



Hình: Connection events

### *2.2.3/ Giao thức ATT:*

**- Attribute (thuộc tính) là đơn vị dữ liệu nhỏ nhất được định nghĩa trong ATT và GATT. Chúng chứa các user data về cấu trúc và nhóm của những Attribute khác chứa trong thiết bị Server.**

- ATT là một giao thức stateless client/server (dữ liệu client không lưu lại trên server) dựa trên Attribute (thuộc tính) thể hiện trên thiết bị. Giao thức có tính nghiêm ngặt khi một request chưa hoàn tất (không phản hồi hoặc request chưa được nhận) thì không gửi thêm bất kỳ request nào.

**- Giao thức ATT có hai vai trò là ATT Server, source của dữ liệu, và ATT Client, đọc ghi dữ liệu trên ATT Server.**

- Các thiết bị Server chứa dữ liệu được định dạng theo chuẩn thuộc tính, bao gồm:

* ***Giá trị tham chiếu (16bit-handle)*: tham chiếu đến các Attribute được lưu trong bộ nhớ của thiết bị Server.**
* ***UUID*: mã định dạng để xác định loại dữ liệu đại diện cho giá trị của thuộc tính (Type). Các giá trị UUID có thể được tùy chỉnh hoặc theo chuẩn của Bluetooth SIG. UUID theo chuẩn SIG thì dữ liệu đó có thể trao đổi với nhiều thiết bị BLE khác trên thị trường.**
* ***Permission*: cho phép các hoạt động ATT được thực thi với một Attribute và với những yêu cầu bảo mật.**
* ***Value*: giá trị của dữ liệu.**

**- *UUID*: là một giá trị 128 bits định danh duy nhất. Bluetooth SIG cung cấp các giá trị cho tất cả các type, service và profile để chuẩn hóa các dữ liệu truyền và nhận giữa các thiết bị Bluetooth từ nhiều hãng sản xuất khác nhau. Developer cũng có thể tự tạo một giá trị UUID cho ứng dụng của mình.**

**- Các thao tác dữ liệu trong ATT:**

* ***Read*: ATT Client đọc dữ liệu từ ATT Server.**
* ***Write without response*: ATT Client ghi giá trị dữ liệu trong ATT Server. ATT Client không đợi và yêu cầu xác nhận từ ATT Server.**
* ***Write with response*: ATT Client yêu cầu và đợi xác nhận sau khi ghi dữ liệu trong ATT Server.**
* ***Notification*: ATT Server sẽ tự truyền dữ liệu cho ATT Client và ATT Server sẽ không chờ xác nhận.**
* ***Indication*: tương tự như Notification nhưng ATT Server sẽ đợi xác nhận từ ATT Client.**

### ***2.2.4 / Generic Access Profile (GAP):***

- GAP (Generic Access Profile): là khối quản lý hầu hết việc phát sóng, kết nối, đồng bộ, bảo mật … cho đến việc thay đổi các thông số kết nối, truyền nhận dữ liệu từ radio module trong BLE chip.

- GAP có 4 vai trò:

* *Broadcaster*: áp dụng cho các thiết bị có ứng dụng chỉ truyền, phân phát dữ liệu thường xuyên (vd: cảm biến nhiệt độ). Broadcaster gửi dữ liệu ở dạng Advertising packet.
* *Observer*: áp dụng cho các thiết bị có ứng dụng chỉ nhận dữ liệu từ broadcaster (vd: màn hình hiển thị nhiệt độ). Observer listen dữ liệu đi cùng với các Advertising packet từ broadcaster.
* *Central*: Central tương ứng với Master trong lớp Link, áp dụng cho thiết bị có khả năng thiết lập kết nối với nhiều thiết bị khác. Vai trò Central yêu cầu cao về CPU và bộ nhớ để duy trì nhiều kết nối từ các thiết bị (Smartphone, tablet).
* *Peripheral*: Peripheral tương ứng với Slave trong lớp Link, là thiết bị phát các Advertising packet để Central nhận dạng và thiết lập kết nối. Các thiết bị Peripheral không yêu cầu cao về tài nguyên như Central.

*2.2.4.1/ Discovery:*

- *Discovery – Nhận dạng*: Discovery là quá trình nhận dạng sự hiện diện và các thông tin cơ bản của các thiết bị Peripheral của Central. Quá trình này không yêu cầu phải thiết lập kết nối hoặc trao đổi dữ liệu. Discovery có hai thủ tục là limited discovering và General discovering.

- Discoverability: là cách mà Peripheral advertise và Central có thể hoặc nên làm gì với thông tin advertising. Có các chế độ discoverability:

* Non-discoverable: các thiết khác sẽ không biết được sự hiện diện của Peripheral.
* Limited discoverable: chế độ này cho phép thiết bị được nhận dạng trong một chu kỳ thời gian và với độ ưu tiên thấp. Thiết bị Central hoạt động ở thủ tục limited discovery mới nhận dạng được.
* General discoverable: giúp thiết bị có thể nhận dạng khi được yêu cầu. Thiết bị trong chế độ này luôn mong muốn được phát hiện và thiết lập kết nối.

Thông thường, các thiết bị Peripheral luôn ở trong chế độ General discoverable. Khi thiết lập được kết nối với Central, Peripheral chuyển sang chế độ Non-discoverable.

*2.2.4.2/ Thiết lập kết nối:*

- Khi thiết lập kết nối, các thiết bị phải ở trong connectable mode. Tương tự như discovery, một số chế độ và hoạt động của thiết lập kết nối điều khiển các lựa chọn thiết bị để tương tác.

- Các chế độ thiết lập kết nối:

* *Non-connectable*: thiết bị Peripheral sẽ không phát Advertising packet và không Central nào có thiết lập kết nối được với chúng.
* *Directed connectable*: Peripheral gửi ADV\_DIRECT\_IND advertising packet với tần suất cao trong thời gian ngắn, không có user data và có một Bluetooth device address đích. Chế độ này cho phép reconnect nhanh khi Peripheral nghi nghờ Central đã kết nối với mình muốn thiết lập lại kết nối nhanh nhất có thể.
* *Undirected connectable*: Peripheral gửi ADV\_IND advertising packet. Đây là chế độ tiêu chuẩn, Peripheral muốn chúng có thể kết nối trong khoảng thời gian dài và mong muốn thiết lập kết nối với Central.

- Các thủ tục thiết lập kết nối:

* *Auto connection establishment*: Central duyệt White list (danh sách các Bluetooth device address, nếu advertising packet hoặc connection request được gửi từ địa chỉ không có trong danh sách thì chúng sẽ bị hủy) và kết nối với thiết bị đầu tiên được nhận dạng.
* *General connection establishment*: đây là thủ tục dùng kết nối với Peripheral chưa từng biết. Central bắt đầu scanning không dựa theo White list, thiết bị sẽ thông báo cho user hoặc truy cập dữ liệu trong advertising packet để quyết định kết nối. Kết nối được thực hiện bằng thủ tục Direct connection establishment.
* *Selective connection establishment*: Giống như thủ tục General nhưng thủ tục Selective sử dùng White list để lọc các advertising packet.
* *Direct connection establishment*: kết nối Central với một Peripheral cụ thể được nhận dạng bởi Bluetooth address bằng lớp Link.

*2.2.4.3/ Các thủ tục khác:*

- *Name discovery*: Device name nằm trong advertising packet là một chuỗi UTF-8 mô tả tên của thiết bị. Kích thước của advertising packet có giới hạn nên một số thiết bị không cho Device name vào advertising packet.

- *Connection parameter update*: mỗi kết nối được thiết lập cần những tham số được thiết lập bởi Central để cân bằng giữa thông lượng và điện năng tiêu thụ cho kết nối. Peripheral cũng có thể yêu cầu Central điều chỉnh thông số lại như mong muốn.

- *Terminate connection*: cả Peripheral và Central có thể chấm dứt kết nối vào bất kỳ lúc nào và sẽ cung cấp một mã thể hiện lý do chấm dứt kết nối.

*2.2.4.4/ Định dạng của Advertising packet:*

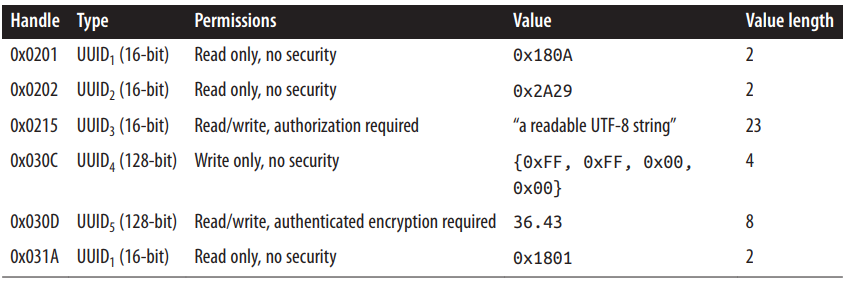
|  |  |  |
| --- | --- | --- |
| Tên | Data length (byte) | Mô tả |
| Flags | 1 | Thiết lập limited hoặc general discovery mode |
| Local name | variable | Tên thiết bị advertise |
| Appearance | 2 | Giá trị 16 bits mô tả loại thiết bị advertise |
| TX Power level | 1 | Năng lượng để truyền advertising packet, dùng để tín toán độ suy hao tín hiệu tại Central, Observer |
| Service UUID | variable | Danh sách các GATT service được Peripheral sử dụng để cho các thiết bị khác biết mà không cần lập kết nối |
| Slave Connection Interval Range | 4 | Đề xuất thiết bị Central về thời gian connection interval phù hợp với Peripheral |
| Service Solicitation | variable | Danh sách các service được hỗ trợ bởi thiết bị gửi packet. Cho phép Central nhận advertising packet để tìm ra các service mà Peripheral có thể truy cập |
| Service Data | variable | UUID định danh service và các dữ liệu của chúng |
| Manufacturer Specific Data | variable |  |

### ***2.2.5/ GATT – Generic Attribute Profile:***

**- Khối GATT lập ra chi tiết cách trao đổi dữ liệu thông qua kết nối BLE. Trái với GAP, GATT quản lý các quá trình và định dạng của dữ liệu truyền đi.**

**- GATT cung cấp framework cho tất cả GATT-based Profile đảm bảo khả năng tương tác giữa các thiết bị từ nhiều nhà sản xuất.**

**- GATT sử dụng giao thức ATT cho quá trình trao đổi dữ liệu. Các dự liệu trong GATT được tổ chức theo dạng attribute đã nêu trong phần giao thức ATT.**

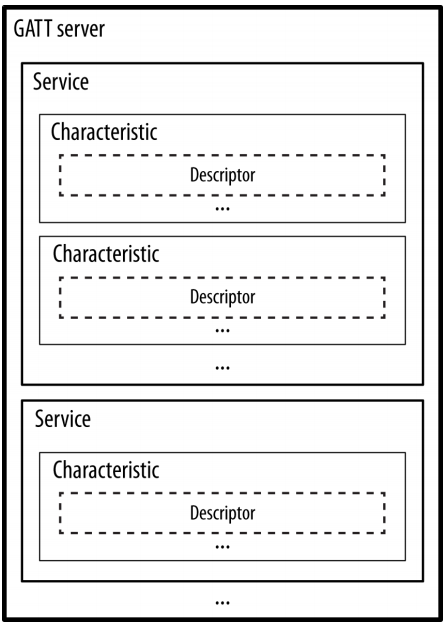


**Hình 2.4: ví dụ về các attribute của một server.**

**- Có hai vai trò trong GATT:**

* **GATT Client: tương ứng với ATT Client. GATT Client gửi request đến server và nhận phản hồi. GATT không biết gì về các attribute của server và phải thực hiện service discovery. Sau khi discover service, GATT Client có thể đọc, ghi các thuộc tính trên server.**
* **GATT Server: tương ứng với ATT Server. GATT Server nhận request từ Client và response. GATT server là nơi chứa dữ liệu cho Client tương tác theo dạng attribute.**

***2.2.5.1/ Tổ chức dữ liệu trong GATT:***



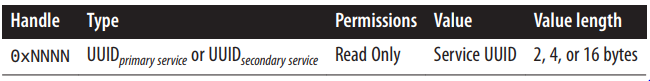
**Hình 2.5: Tổ chức dữ liệu trong GATT**

**- Các attribute trong server được nhóm vào các service, mỗi service có thể có 0 hoặc nhiều characteristic. Characteristic cũng có thể có 0 hoặc nhiều descriptor.**

**- Hầu hết các loại dữ liệu trong tổ chức trên đều có hai khái niệm là definition (các nhóm attribute tạo nên chúng) và declaration. Declaration là thuộc tính đầu tiên trong quá trình định nghĩa và đưa ra những metadata về dữ liệu đi kèm. Tất cả declaration đều có read-only permission và không yêu cầu bảo mật. Declaration cho phép Client discover các attribute trên server.**

***2.2.5.2/ Service***

**- Service là nhóm các thuộc tính có liên quan với nhau trong GATT server. Mỗi service đều được bắt đầu với thuộc tính service declaration**



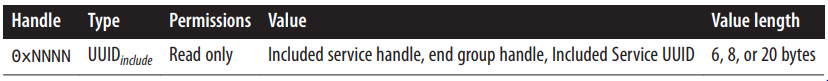
**Hình 2.6: Thuộc tính service declaration**

**- UUID cột Type được gán bởi SIG để khai báo một service attribute và mặc định là 0x2800.**

**- UUID cột Value là giá trị định danh loại chức mà service này thực hiện.**

**- Bên trong một service có thể có thêm những tham chiếu đến các service khác trên Server và được gọi là including definition. Including definition bao gồm một attribute chứa các yêu cầu để client tham chiếu đến các service được include.**

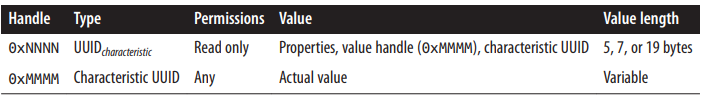
**- Các included service giúp tránh dữ liệu trùng lặp, tiết kiệm bộ nhớ và đơn giản tổ chức trong GATT server.**



**Hình 2.7: thuộc tính Include Declaration, UUID mặc định là 0x2802**

***2.2.5.3/ Characteristic:***

**- Characteristic là nơi chứa user data, bao gồm hai attribute là *characteristic declaration* (metadata về user data thực) và *characteristic value* (một attribute chứa user data trong trường value).**



**Hình 2.8: Định dạng một Characteristic**

- UUID characteristic khai báo thuộc tính là một characteristic mặc định là 0x2803.

- Các giá trị trong một Characteristic declaration:

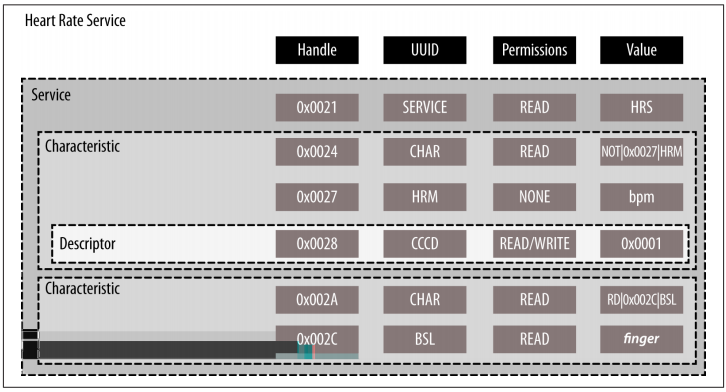
|  |  |  |
| --- | --- | --- |
| Name | Length (byte) | Description |
| Properties | 1 | Các hoạt động cho phép thực thi trên characteristic |
| Handle | 2 | Giá trị tham chiếu của attribute *Characteristic value* |
| UUID | 2, 4, 16 | UUID mô tả characteristic |

- *Characteristic value* chứa user data mà Client có thể đọc và ghi. Loại của attribute này là giá trị UUID trong trường value của Characteristic declaration. Trường value của attribute này là giá trị thực tế như nhiệt độ, nhịp tim.

- Characteristic descriptor cung cấp cho Client metadata về characteristic và value của chúng. Descriptor cũng là một attribute có declaration với UUID là loại descriptor.

- Các giá trị UUID của service và characteristic được SIG định nghĩa trên <https://www.bluetooth.com/specifications/gatt/services>

- Ví dụ heart rate service:



Hình 2.9: Heart rate service. UUID service là 0x180D

*2.2.5.4/ Service và Characteristic Discovery:*

- Client khi vừa kết nối sẽ không biết về những thuộc tính trên GATT server. Vì vậy, Client thực hiện chuỗi trao đổi packet để xác định các thuộc tính trên GATT server.

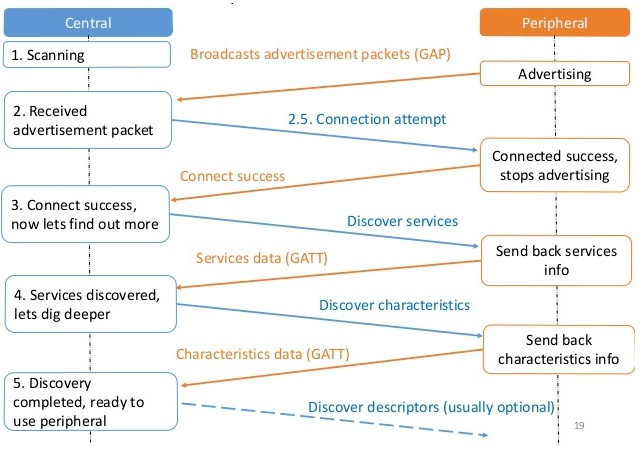
- Có hai dạng discovery chính là Discovery tất cả hoặc thông qua một UUID cụ thể:

* Client sẽ lấy toàn bộ danh sách các service và characteristic trên Server trong dạng Discovery thứ nhất. Dạng này được sử dụng khi Client hỗ trợ nhiều loại service.
* Discovery bằng UUID: Client chỉ lấy các service và characteristic theo UUID mà Client hướng tới

## **2.3/ Tổng kết các hoạt động cơ bản của BLE:**

- Có 4 hoạt động cơ bản:

* *Advertising*: là hoạt động phát dữ liệu nhận dạng của BLE Peripheral.
* *Scanning*: BLE Central thu thập dữ liệu nhận dạng của BLE Peripheral.
* *Connecting*: BLE Central sẽ yêu cầu gửi thêm các dữ liệu nhận dạng (Scan request) và BLE Peripheral sẽ gửi theo yêu cầu (Scan response). Sau đó BLE Central kiểm tra đầy đủ dữ liệu nhận dạng ( từ Advertising data và Scan response) và gửi yêu cầu kết nối (Connection request). Cuối cùng, BLE Peripheral sẽ chấp nhận hoặc từ chối kết nối (Connection response).
* *Discovering*: Thiết bị Client lấy Service và Characteristic mà thiết bị Server có thể cung cấp như nhiệt độ, độ ẩm, trạng thái … và Client sẽ có nhu cầu biết về các loại dữ liệu có thể nhận từ Server.

Hình 2.10: Các hoạt động cơ bản của BLE

# **3/ Mạng Bluetooth Mesh:**

## **3.1/ Các khái niệm cơ bản về mạng Bluetooth mesh:**

### *3.1.1/ Giới thiệu mạng mesh:*

- Khoảng giữa tháng 07/2017, tổ chức Bluetooth (SIG) đã đưa ra cấu hình của Bluetooth Mesh, một giao thức kết nối mới, cho phép nhiều thiết bị Bluetooth kết nối ngang hàng với nhau, tạo thành một mạng lưới rộng chứ không chỉ giới hạn giữa 2 máy như trước đây, tín hiệu sẽ đi từ máy này tới máy khác cho đến khi chạm được mục tiêu.

### *3.1.2/ Đặc điểm của mạng Bluetooth Mesh:*

- Hầu hết các thiết bị Bluetooth Low Energy giao tiếp với nhau thông qua mô hình mạng point-to-point, cho phép thiết bị giao tiếp 1-1. Smartphone có thể thiết lập kết nối point-to-point với nhiều thiết bị khác nhau và có thể giao tiếp trực tiếp với chúng. Tuy nhiên, các thiết bị kết nối với smartphone lại không thể giao tiếp được với nhau.

- Mạng Bluetooth Mesh xây dựng theo mô hình many-to-many cho phép các thiết bị có thể giao tiếp trực tiếp với các thiết bị khác trong mạng mesh. Giao tiếp được thực hiện thông qua messages, các thiết bị sẽ tiếp nhận và phát lại messages, hay còn gọi là relay, cho các thiết bị xung quanh cho tới khi message đến được đích. Việc tiếp nhận và phát lại messages giúp các thiết bị có thể giao tiếp với nhau ngoài vùng phủ sóng Bluetooth thông qua một thiết bị trung gian trong mạng Mesh.

- Ưu điểm của mạng Bluetooth Mesh là giúp gia tăng phạm vi, bao phủ sóng ở trong nhà nơi bị tường, các đồ vật cản sóng. Một ưu điểm nữa là việc relay tạo cho message nhiều đường dẫn tới đích và làm tăng xác suất truyền thành công nếu có đường dẫn bị lỗi so với chỉ truyền một hướng duy nhất.

- Bluetooth Mesh có một loại địa chỉ là group addresss, dùng để gom nhóm nhiều Node lại để điều khiển cùng một lúc ở mọi khoảng cách với kỹ thuật relay message.

A close up of a logo

Description generated with high confidence

Hình 3.1: Tín hiệu được truyền đi từ công tắc và được relay đến đèn.

### *3.1.3/ Node và thiết bị:*

- Những thiết bị chưa tham gia mạng Mesh được gọi là “unprovisioned”, quá trình thiết bị gia nhập mạng Mesh là “provisioning” và sau quá trình đó, thiết bị sẽ thành một Node của mạng Mesh.

- Thiết bị khi được provision vào mạng Mesh sẽ được cấp các key mã hóa và được quản lý bởi thiết bị provisioner, thông thường là smartphone hoặc tablet.

- Tất cả Node trong mạng Mesh đều sở hữu một NetKey để xác định Node đó thuộc về mạng Mesh nào.

### *3.1.4/ Elements:*

- Element là đại diện cho những thành phần hoạt động độc lập trong Node. Một Node có thể có nhiều element. Element đầu tiên của Node được gọi là primary. Ví dụ về một Node Switch:

Node Switch: On/Off

Primary Element #1: Công tắc 1 điều khiển đèn

Secondary Element #2: Công tắc 2 điều khiển quạt

- Ví dụ khác về Node Light, một element của Node Light có thể đại diện cho một bóng đèn hoặc đại diện cho nhiều bóng đèn khác nhau. Nếu element có nhiều đèn thì tất cả sẽ đều được điều khiển bởi một công tắc.

- Mỗi element trong node đều có một địa chỉ là unicast address. Element đầu tiên trong Node sẽ được gán địa chỉ trong quá trình provisioning và các element sau sẽ được gán địa chỉ theo địa chỉ primary Element.

- Địa chỉ element giúp Node xác định Element nào đang gửi và nhận message.

### *3.1.5/ Message:*

- Tất cả những giao tiếp trong mạng Mesh đều là hướng thông điệp, mỗi loại thông điệp đều có một opcode. Khi một node muốn truy xuất trạng thái hoặc điều khiển node khác, chúng gửi những thông điệp. Khi một node muốn cho node khác biết trạng thái của mình, chúng cũng gửi thông điệp.

- Thông điệp có hai dạng là acknowledged và unacknowledged:

* *Acknowledge*: thông điệp yêu cầu phản hồi từ node mà nó gửi đến. Việc phản hồi giúp xác nhận dữ liệu đã nhận thành công và trả dữ liệu của node nhận về cho node gửi.
* *Unacknowledge*: Không yêu cầu phản hồi.

- Một message có một trường opcode và một hành động (hàm callback) tương ứng. Opcode có thể có độ dài 1 octet (1 byte) cho những loại message đặc biệt yêu cầu tối đa kích thước dữ liệu gửi, độ dài 2 octet cho những các message bình thường và 3 octet cho các message định nghĩa bởi nhà sản xuất.

- Các message khi có kích thước quá lớn thì chúng sẽ được phân tách và gộp (segmentation and reassembly, SAR). Theo Mesh Specification, lớp Transport cung cấp 11 octet dữ liệu payload (bao gồm opcode, giá trị ,không gồm các trường như src, dest…) tối đa để vừa đủ truyền một unsegmented message packet.

- Lớp Transport có thể phân tách một message 32 lần, như vậy kích thước tối đa của một message là 384 octet.

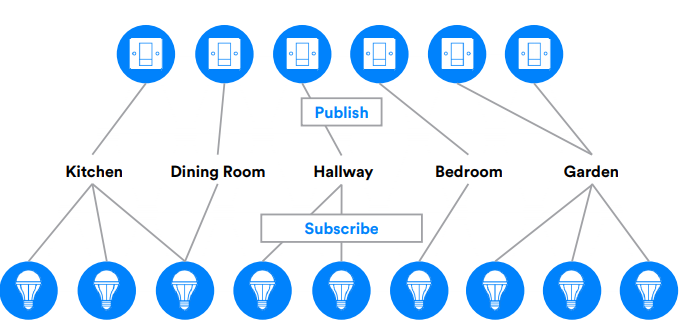
### *3.1.6/ Addresses:*

- Có 3 dạng địa chỉ:

* *Unicast address*: là địa chỉ của một element duy nhất và được cấp trong quá trình provision.
* *Group address*: là địa chỉ của một hoặc nhiều element.
* *Virtual address*: là địa chỉ được gán cho một hoặc nhiều element, spanning cho một hoặc nhiều node. Virtual address có dạng 128bit-UUID và thường được định nghĩa sẵn để gán cho các thiết bị trong mạng Mesh cùng hãng sản xuất.

### *3.1.7/ Publish/Subscribe:*

- Publish là việc xác định địa chỉ mà Node gửi message. Subscribe là việc thiết lập địa chỉ để nhận những thông điệp được gửi.



Hình 3.2: Publish/Subscribe

- Trong hình 2, công tắc 1 đang publish vào địa chỉ group Kitchen, đây là địa chỉ mà công tắc 1 sẽ gửi thông điệp. Đèn 1,2,3 subscribe tới địa chỉ group Kitchen để nhận thông điệp gửi đi từ công tắc 1, mà đã publish vào Kitchen. Cả ba bóng đèn sẽ được điều khiển bởi công tắc 1.

- Đèn thứ 3 đang subscribe tới Dining Room, như vậy đèn 3 có thể được điều khiển bởi công tắc 1 hoặc công tắc 2. Địa chỉ Garden có 2 công tắc publish vào, 2 công tắc này sẽ điều khiển 3 đèn đang subscribe tới Garden.

- Ưu điểm của việc publish/subscribe là khi một node mới tham gia hoặc rời khỏi mạng Mesh. Chúng chỉ việc publish/subscribe tới địa chỉ nào đó và không làm ảnh hưởng đến địa chỉ của những node khác đang publish/subscribe.

### *3.1.8/ States:*

- State thể hiện trạng thái của một element.

- Element thể hiện state ở bên ngoài được gọi là Server. Ví dụ như bóng đèn có hai trạng thái sáng và tắt.

- Element truy cập đến các state được gọi là Client. Ví dụ như công tắc điều khiển đèn bật và tắt.

### *3.1.9/ Message và State:*

- Message có 3 dạng:

* GET: lấy giá trị state của một hoặc nhiều node. Một message dạng STATUS sẽ trả về cho node gửi GET giá trị của state.
* SET: thay đổi giá trị state của các node. Nếu SET có dạng acknowledged, thì message STATUS sẽ trả về với giá trị state sau khi thay đổi.
* STATUS: dùng để gửi giá trị state của node khi nhận được GET, acknowledged SET message. STATUS cũng có thể dùng để thông báo state của mình cho các node xung quanh.

### *3.1.10/ Models:*

- Model định nghĩa các chức năng, các state của một element và message tương tác với những state đó. Một Element chỉ có thể mang một model. Có 3 dạng model:

* *Server* *model*: định nghĩa các loại message có thể truyền, nhận và các hành động sau khi truyền, nhận message (vd: bật tắt đèn).
* *Client model*: định nghĩa các message dùng để thay đổi và lấy các state của Server model. Client model không có state.
* *Control model*: bao gồm Server model, Client model và Control logic dùng để phối hợp với các model để kết nối và giúp chúng giao tiếp với nhau.

- Trong một Node có thể cài đặt cả Client và Server Model. Các Server và Client model tương tác với nhau thông qua cơ chế Subcribe (thường là Server) và Publish (thường là Client)

- Ví dụ về model của node Light Buld gồm:

* On/Off -> State: On/Off
* Độ sáng -> State: 0 – 10

- Thuật ngữ bound state nhắc đến việc một state trong Model thay đổi thì kéo theo các state khác chẳng hạn như nếu độ sáng 0 -> 1 thì state off -> on.

- Mỗi model trong mạng Mesh đều có một định danh duy nhất. Model theo chuẩn của Bluetooth SIG có 16bit trong khi Model của vendor là 32bit.

- Tất cả model đều được gán một Appkey được cấp trong quá trình provisioning. Appkey dùng để bảo mật các message tương tác với model. Mạng Mesh có nhiều Appkey, mỗi Appkey được sử dụng cho nhiều loại model có chức năng khác nhau.

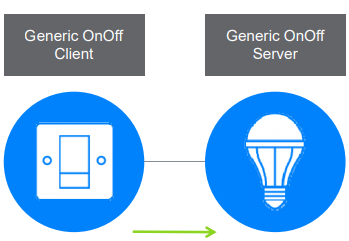
- Mạng Bluetooth Mesh giao tiếp theo mô hình client – server qua message. Nhiệm vụ của server là đưa ra các state của các element. Client có thể lấy, thay đổi và sử dụng state để xử lý.

### *3.1.11/ Generics:*

- Nhiều loại thiết bị có thể hoạt động với những state giống nhau như đèn, quạt, ổ điện… đều có state On và Off.

- Bluetooth Mesh specification định nghĩa một loại Model chung cho các thiết bị gọi là Generic. Với On, Off là Generic OnOff.

- Generics cho phép nhiều loại thiết bị có thể tương thích với mạng Mesh mà không cần phải tạo riêng cho chúng một model.



Hình 3.3: Hai loại generic model cho đèn và công tắc.

- Message SET trong Gerneric OnOff Model có thêm các tham số:

* OnOff: giá trị cần set
* TID – Transaction Identifier: message mới hoặc truyền lại lần nữa
* Transtion time: thời gian element thực hiện chuyển đổi state
* Delay: khoảng delay thực thi message

### *3.1.12/ Scenes:*

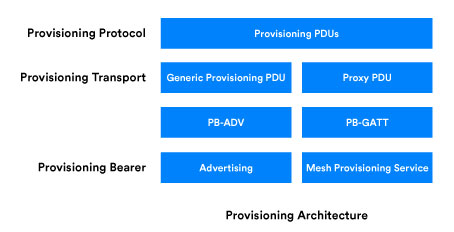
- Scene là tập các states được gọi lại khi node nhận được loại thông điệp đặc biệt hoặc vào một thời điểm nào đó. Scene được định danh bởi 16-bit Scene Number và là duy nhất trong mạng Mesh.

- Scene cho phép node được set bằng một tập các state, thể hiện cho một chức năng nào đó.

### *3.1.13/ Provision:*

- Provision là quá trình cho phép thiết bị gia nhập mạng Mesh.

- Bluetooth Mesh định nghĩa Provisioning PDU giao tiếp giữa provisioner và unprovisioned theo hình dưới:



Hình 3.4: Các Provisioning PDU của mạng Mesh

- *Provisioner Bearer*: Lớp này cho phép vận chuyển provisioning PDUs giữa provisioner và unprovisioned. Có hai loại provisioning bearer:

* PB-ADV: được dùng để provision, truyền Generic Provisioning PDUs thông qua các kênh phát Bluetooth (advertising). Thiết bị hỗ trợ PB-ADV phải scanning với duty-cycle gần như 100% để tránh mất mát PDUs.
* PB-GATT: được dùng để provision bằng Bluetooth mesh proxy PDUs từ giao thức proxy. Giao thức proxy cho phép node gửi và nhận các network PDUs, mesh beacon (tín hiệu mạng mesh), thông điệp thiết lập proxy, và provisioning PDUs thông qua kết nối có định hướng, BLE bearer. PB-GATT đóng gói các provisioning PDUs trong GATT provisioning service và được cung cấp khi provisioner không hỗ trợ PB-ADV. Loại provisioning bearer này được sử dụng trong đồ án để provision bằng smartphone.

- Giao thức Provisioning định nghĩa 10 loại provisioning PDU:

* Provisioning Invite
* Provisioning Capabilities
* Provisioning State
* Provisioning Public Key
* Provisioning Input Complete
* Provisioning Confirmation
* Provisioning Random
* Provisioning Data
* Provisioning Complete
* Provisioning Failed

- Tóm lược quá trình Provision thực hiện hai nhiệm vụ chính:

1. Xác nhận thiết bị unprovisioned để đảm bảo thiết bị đang tương tác với provisioner là thiết bị gia nhập mạng Mesh mà user mong muốn.
2. Xây dựng liên kết bảo mật với thiết bị unprovisioned và chia sẻ thông tin. Khi quá trình provision hoàn tất, thiết bị sẽ trở thành node trong mạng Mesh.

- Quá trình provisioning có 5 giai đoạn:

1. Beaconing
2. Invitation
3. Exchange public keys
4. Authentication
5. Distribution of provision data
6. *Beaconing*:

- Thiết bị GAP peripheral phát các advertising packet. Thiết bị GAP central tìm các advertising packet để phát hiện thiết bị đang phát.

- Nếu thiết bị hỗ trợ PB-ADV, nó sẽ phát packet theo format unprovisioned mà provisioner có thể phát hiện ra chúng.

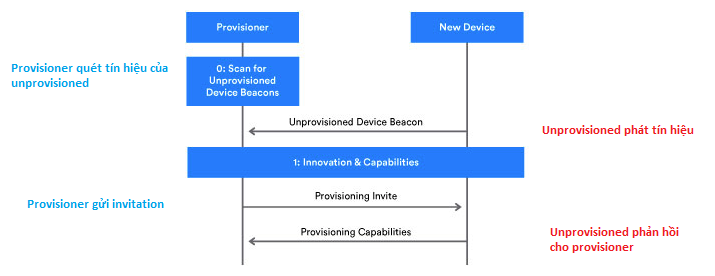
- Với PB-GATT, Mesh Proxy Provision service sẽ hỗ trợ quá trình provisioning và tương tác với provisioner. Trong giai đoạn beaconing, unprovisioned sẽ phát packet bao gồm UUID của Mesh Provisioning Service và được nhận dạng bởi provisioner.

1. *Invitation*:

- Sau quá trình, thiết bị provisioner và unprovisioned bắt đầu thiết lập provisioning bearer. Provisioner gửi một Provisioning PDU và unprovisioned phản hồi với một provisioning PDU tương ứng.

- Provisioning PDU tương thích mà unprovisioned phản hồi bao gồm:

* Số Element thiết bị hỗ trợ.
* Tập các giải thuật bảo mật.
* Những public key sử dụng công nghệ Out-Of-Band.
* Khả năng unprovisioned output giá trị tới user.
* Khả năng unprovisioned cho phép giá trị được nhập từ user.

- Tổng hợp giai đoạn 1,2 của quá trình provision:

Hình 3.5: Giai đoạn 1,2 của quá trình provision

1. *Trao đổi Public keys*:

- Có hai dạng mã hóa là symmetric (đối xứng, secret key) và asymmetric (bất đối xứng, public key):

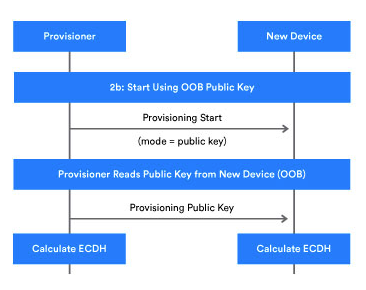
* *Mã hóa symmetric*: sử dụng cùng secret key để mã hóa – giải mã. Khi người gửi và người nhận biết được secret key, chúng có thể giải mã mọi message được mã hóa với key này. Tuy nhiên, rất khó trao đổi secret key qua một liên kết và ngăn chặn chúng rơi vào nhầm chỗ.
* *Mã hóa asymmetric*: sử dụng hai key public và private. Public key luôn sẵn sàng cho mọi người gửi tới mình và private chỉ có mình biết được. Mọi message được mã hóa với public key chỉ có thể được giải mã với cùng giải thuật và bằng cách sử dụng private key giống nhau. Như vậy không còn phải lo về việc truyền public key vì chỉ sử dụng cho việc mã hóa. Tuy nhiên, kỹ thuật này chậm hơn với symmetric và cần nhiều điện năng hơn cho việc xử lý.

- Trong mạng Bluetooth Mesh, kỹ thuật Symmetric thích hợp hơn vì đa số các thiết bị đều dựa trên chipset và các module nhúng, chúng khó xử lý được kỹ thuật phức tạp như Asymmetric và cũng yêu cầu nhiều điện năng hơn. Tuy nhiên Symmetric lại gặp vấn đề về bảo mật nên Bluetooth Mesh kết hợp cả hai kỹ thuật.

* *Mật mã Asymmetric*: Elliptic Curve Diffie–Hellman (ECDH) là một giao thức chấp nhận anonymous key cho phép hai phần, mỗi phần có một cặp elliptic curve public–private, thiết lập một shared secret thông qua một kênh không bảo mật. Mục dích của ECDH trong mạng Mesh là để thiết lập liên kết bảo mật giữa provisioner và unprovisioned sử dụng những public và private key để cấp phát symmetric secret key mà hai thiết bị có thể sử dụng để mã hóa – giải mã message.
* *Mật mã Symmetric*: mỗi thông điệp phát đi trong mạng Bluetooth Mesh được mã hóa theo mật mã AES-128

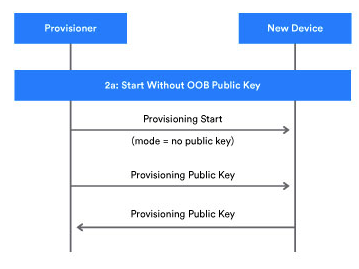
- Trong giai đoạn trao đổi public key có hai cách trao đổi ECDH key là thông qua Bluetooth hoặc OOB tunnel. Trong giai đoạn invation provisioning, thiết bị unprovisioned thông báo chúng có hỗ trợ gửi public key qua OOB tunnel hay không.

- Nếu có hỗ trợ OOB, một public key ngắn hạn được truyền từ provisioner tới thiết bị, và public key tĩnh được đọc bởi unprovisioned sử dụng kỹ thuật OOB.



Hình 3.6: Trao đổi public key nếu OOB được hỗ trợ

- Nếu không, quá trình được diễn ra theo hình dưới:



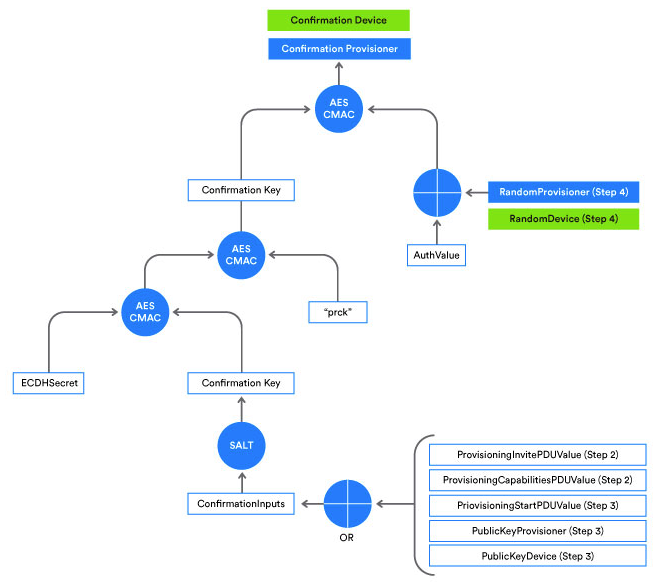
Hình 3.7: Trao đổi public key nếu OOB không được hỗ trợ

1. *Authentication: Static OOB hoặc No OOB*:

- Thiết bị provisioner và unprovisioned sử dụng giải thuật giống nhau để tạo ra hai giá trị là ConfirmationProvisioner và ConfirmationDevice.

- Hình mô tả giải thuật tạo các giá trị Confirmation:

* AES CMAC. SALT là các giải thuật tạo các giá trị authentication.

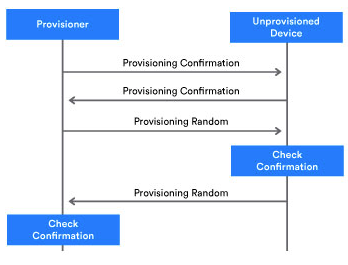


Hình 3.8: Giải thuật tạo Confirmation Value

- Khi hai giá trị Confirmation sẵn sàng, chúng được trao đổi giữa hai thiết bị.

- Quá trình xác nhận bắt đầu với việc Provisioner gửi RandomProvisioner. Unprovisioned sử dụng nó để tính lại giá trị Confirmation và so sánh với giá trị Confirmation ban đầu chúng nhận được.

* Nếu hai giá trị không giống nhau, quá trình provision sẽ bị hủy
* Nếu chúng giống nhau thì unprovisioned gửi lại cho Provisioner RandomDevice và provisioner thực hiện lại quá trình xác nhận như unprovisioned. Nếu xác nhận giống nhau thì quá trình authentication thành công.



Hình 3.9: Giai đoạn Authentication trong quá trình provision

1. *Data distribution*:

- Provisioner thực hiện cấp dữ liệu provisioning cho thiết bị mới vào mạng. Dữ liệu provisioning bao gồm:

|  |  |  |
| --- | --- | --- |
| Field | Size(octet-8bit) |  |
| Network key | 16 | - Còn gọi là NetKey. NetKey bảo mật cho quá trình giao tiếp ở lớp network và được chia sẻ cho toàn bộ các node. Node sở hữu NetKey được coi là đã gia nhập mạng Mesh.  - Provisioner sẽ tạo NetKey khi thiết bị đầu tiên gia nhập mạng. |
| Device key | 16 | Một key bảo mật duy nhất được provisioner và thiết bị đang gia nhập mạng sở hữu. |
| Key Index | 2 | Do kích thước NetKey quá lớn, không thể gửi trong một message. Để truyền message có hiệu quả, key được cấp 12bit global là key index, định danh ngắn cho key. Message truyền có key index sẽ tham chiếu trong key list. |
| Flags | 1 | Flag bitmask – Thông báo trạng thái của key. |
| IV index | 4 | Initialization Vector Index là giá trị 32bit được share giữa các node dùng để cung cấp entropy cho việc tính toán giá trị nonce |
| Unicast Address | 2 | Địa chỉ Unicast của Element đầu tiên của node mới. |

- Để cấp dữ liệu Provisioning được bảo mật, Provisioner sử dụng giải thuật AES-CCM để mã hóa dữ liệu với shared SessionKey.

- Khi dữ liệu provisioning đã được cấp, hai thiết bị ngang hàng sẽ biết được nhau qua DevKey và NetKey, thiết bị sẽ trở thành node trong mạng Mesh. Quá trình Provisioning kết thúc.

### *3.1.14/ Node:*

- *Relay node*:

* Những node có thể phát lại message đã nhận là relay node. Message có thể truyền đi khắp mạng Mesh, tạo thành nhiều điểm nhảy giữa các thiết bị bằng cách relay.
* Mesh network PDU có một trường là TTL – Time-to-live. Chúng là giá trị interger quy định số lần relay tối đa(hop). Ví dụ nếu TTL = 3, thì số lần relay tối đa của message là 3.

- *Low Power Node và Friend Node*:

* Có một số node có nguồn điện bị giới hạn và cần tiết kiệm năng lượng và những node chủ yếu gửi message nhưng đôi lúc vẫn cần nhận message.
* Ví dụ một cảm biến nhiệt độ hoạt động bằng thỏi pin nhỏ. Nó gửi nhiệt độ một phút một lần nếu nhiệt độ còn lớn hoặc nhỏ hơn mức đã thiết lập. Việc này có thể dễ dàng cài đặt mà không gây ra nhiều vấn đề về năng lượng.
* Tuy nhiên, người dùng có thể gửi message để thay đổi mức nhiệt độ. Trường hợp này ít xảy ra nhưng cảm biến vẫn phải hỗ trợ. Việc nhận message có những yêu cầu về duty cycle, điện năng tiêu thụ. Duty cycle 100% đảm bảo cảm biến sẽ nhận được mọi message thiết lập mức nhiệt độ nhưng lại làm gia tang điện năng tiêu thụ và Duty cycle thấp có thể khiến cho cảm biến bỏ qua một số message.
* *Low Power Node – LPN* hoạt động song song với những node không bị giới hạn về năng lượng (nguồn điện sạc được, AC…). Những thiết bị này là Friend Node. Chúng lưu trữ những message được gửi đến LPN và trả lại cho LPN. Khi có yêu cầu “waiting message”, tất cả message được trả về cho LPN theo thứ tự với cờ More Data(MD) để báo cho LPN vẫn còn message chưa được gửi. LPN có thể không yêu cầu Friend Node thường xuyên nên chúng có thể cân bằng việc tiết kiệm năng lượng với nhận và xử lý message vào những thời điểm thích hợp.

### *3.1.15/ Proxy Node:*

- Proxy Node cho phép thiết bị không có Bluetooth Mesh stack giao tiếp với những node trong mạng Mesh.

- Proxy Node đưa ra giao diện GATT mà thiết bị BLE như smartphone, tablet dùng giao tiếp với mạng Mesh. Giao thức Proxy Protocol được sử dụng với PB-GATT. Thiết bị GATT đọc và ghi các Proxy Protocol PDU từ trong các thuộc tính GATT được cài đặt bởi Proxy Node. Proxy Node chuyển đổi chúng thành hoặc từ Mesh PDU.

### *3.1.16/ Relay:*

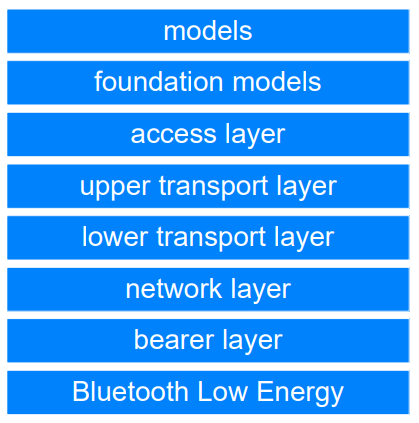
- TTL (Time-to-live) là thời gian sống của message trong mạng Mesh.

- Mạng Mesh có khả năng mở rộng phạm vi nhờ relay message. Các thiết bị khi nhận được một message có TTL lớn hơn hoặc bằng 2 sẽ relay message đó và giảm TTL đi 1. Message sẽ kết thúc tại Node mà giá trị TTL của message nhận được bằng 1. Việc relay không định hướng tạo nên hiệu ứng flooding và gia tăng khả năng truyền message vì có nhiều đường đi để message đến được Node đích.

- Các Node không cần bất cứ điều kiện gì cho việc relay ngoài giá trị TTL. ĐIều này có thể dẫn đến việc các một Node sẽ relay cùng một message nhiều lần, ví dụ như Node trung gian khi nhận message sẽ relay về cho Node phát và Node phát lại relay về Node trung gian. Để tránh việc relay trùng một message, các Node đều được cài đặt một message cache để nhận biết các message mà Node vừa nhận được, tránh relay trùng message.

- Việc relay nhiều lần sẽ ảnh hưởng đến thông lượng, traffic (lưu lượng message lớn có thể gây nghẽn mạng) và điện năng tiêu thụ. Vì thế cần sắp xếp ví trí các Node hợp lý, cân nhắc việc enable tính năng relay của Node, hiệu chỉnh giá trị TTL phù hợp.

### *3.1.17/ Kiến trúc Mesh System:*



Hình 3.10: Các phân lớp của mạng Bluetooth Mesh

- *Lớp Bearer*:

* Lớp Bearer định nghĩa cách Mesh PDU được vận chuyển giữa các Node sử dụng BLE stack ở dưới. Có hai loại Bearer được định nghĩa là Advertising Bearer, GATT Bearer.
  + *Advertising Bearer*: dựa trên BLE GAP advertising và scanning để tìm và nhận các Mesh PDU. Các message được truyền bằng loại bearer này.
  + GATT *Bearer*: cho phép một thiết bị không hỗ trợ Advertising Bearer (PB-ADV) giao tiếp gián tiếp với các Node sử dụng Proxy Protocol. Một Proxy Node có những thuộc tính GATT và hỗ trợ GATT Bearer cũng như PB-ADV để có thể convert và relay message giữa các loại Bearer.

- *Lớp network*:

* Lớp network định nghĩa các loại address của message và định dạng của message để cho phép PDU của lớp transport có thể truyền đi bởi lớp Bearer.
* Lớp network xác định các giao diện network nào để output message. Một bộ lọc input được cài đặt để xác định message nhận được sẽ đưa lên lớp network để xử lý hay không. Output message được đưa vào bộ lọc Output để bỏ qua hoặc đưa vào lớp Bearer.
* Phần Relay và Proxy được cài đặt trong lớp network.

- *Lớp Lower Transport*:

* Lớp Lower Transport: nhận các PDU từ lớp Upper Transport và gửi tới các thiết bị ngang hàng. Khi được yêu cầu, lớp này thực hiện tách và gộp các PDU. Với packet lớn, không đủ với một truyền, Lower Transport sẽ tách các PDU. Lớp Lower Transport của thiết bị nhận gộp các đoạn đã được tách.

- *Lớp Upper Transport*:

* Lớp Upper Transport thực hiện mã hóa – giải mã, authentication dữ liệu để gửi lên hoặc nhận từ lớp Access. Lớp này còn chịu trách nhiệm truyền control message giữa Lower Transport và Upper Transport giữa các node có Friendship (LPN – Friend).

- *Lớp Access*:

* Lớp Access định nghĩa cách sử dụng Lớp Upper Transport:
  + + Định dạng của application data
  + + Định nghĩa và kiểm soát hoạt động mã hóa – giải mã
  + + Xác nhận dữ liệu nhận từ Upper Transport tới đúng mạng và application trước khi chuyển cho lớp trên
* Truyền Access Message:
  + + Message được truyền từ địa chỉ unicast của Element tới địa chỉ đích có thể là địa chỉ unicast, group, virtual.
  + + Trường TTL thiết lập số lần message relay.
  + + Trường SRC: xác định địa chỉ Element của Node truyền message.
  + + Trường DST: xác định địa chỉ đích.
* Nhận Acess Message nếu thỏa các điều kiện:
  + + Opcode thuộc vào địa chỉ Model của Element.
  + + Địa chỉ đích bằng với địa chỉ mà Model của Element đã subcribe.
  + + Model được gán application, device key dùng cho việc bảo mật.

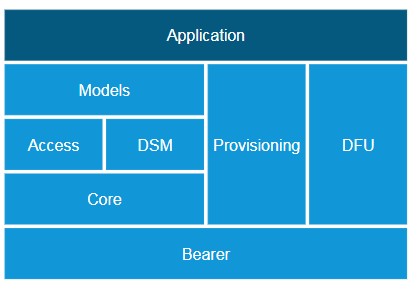
- *Lớp Foundation Model*: Cài đặt các model liên tới việc thiết lập và kiểm soát mạng Mesh.

- *Lớp Model*: Lớp này cài đặt các model cho ứng dụng như bật tắt, nhiệt độ, state…

## **3.2/ nRF SDK for Mesh**

- nRF SDK for Mesh là SDK của Nordic Semicondutor dành cho mạng Bluetooth Mesh. Phiên bản sử dụng trong đồ án là v2.2.0 và cần có nRF5 SDK 15.0.0 để complie.

### *3.2.1/ Cấu trúc của nRF SDK for Mesh*



Hình 3.11: Cấu trúc nRF SDK for Mesh

- *Models*: định nghĩa hành động và giao tiếp với dữ liệu truyền trong mạng Mesh. Mọi giao tiếp đều phải thông qua model.

- *Access*: lớp Access kiểm soát các model của thiết bị. Lớp này chứa các tham số của model, những message mà model xử lý, và thiết lập. Khi thiết bị nhận được message, lớp Access sẽ tìm model mà message đó hướng tới và gửi lên lớp Model.

- *Device State Manager – DSM*: chứa những key mã hóa và địa chỉ được sử dụng bởi - Mesh stack. Khi model được gán application key và publish, DSM lưu các giá trị và cung cấp cho model các handle để tham chiếu các giá trị này.

- *Mesh Core*:

* Bao gồm lớp network và transport, cung cấp những đặc tính cho message transport.
* Lớp Transport cung cấp cơ chế bảo mật bằng mã hóa mesh packet với application key và chia nhỏ chúng. Lớp này cũng có nhiệm vụ tái tạo các packet và đưa lên lớp Access
* Lớp Network mã hóa các segment của lớp Transport bằng network key và gán địa chỉ đích và nguồn. Khi nhận các Mesh packet, lớp Network thực hiện giải mã, xác định địa chỉ đích và nguồn kiểm tra packet có gửi tới thiết bị và có relay packet đó hay không.

- *Bearer*: là bộ điều khiển phát sóng và cung cấp giao diện để nhận gửi packet bất đồng - bộ cho những lớp trên.

- *Device Firmware Update – DFU*: cung cấp tính năng update firmware thông qua mạng Mesh bằng cách phối hợp với bootloader. DFU chỉ có trên các module, chip của Nordic.

- *Mesh stack*: đóng gói các Mesh module để thuận tiện cho việc sử dụng. Mesh stack thực hiện khởi tạo và enable, cung cấp hàm cho việc lưu và xóa dữ liệu liên quan tới provisioning và state.

### *3.2.2/ Cấu trúc dữ liệu SET message:*

- Cấu trúc dữ liệu của Generic On/Off client model: *generic\_onoff\_client\_t*

* *access\_model\_handle\_t model\_handle*: chỉ số của model thực hiện gửi message trong Node trong m\_model\_pool.
* *generic\_onoff\_client\_msg\_data\_t msg\_pkt*: Dữ liệu của message
* *uint8\_t on\_off*: Giá trị integer thực hiện set state trên Server, với model On/Off thì giá trị này là 1 hoặc 0.
* *uint8\_t tid*: Giá trị đánh dấu message là mới hoặc truyền lại.
* *uint8\_t transition\_time*: Thời gian chuyển state
* *uint8\_t delay*: Thời gian delay trước khi truyền

- Access message được gán các thông số từ Model message trước khi truyền. Cấu trúc dữ liệu của message ở lớp Access: *access\_message\_tx\_t*

* *access\_opcode\_t opcode*: opcode của message khi truyền.
* *const uint8\_t \* p\_buffer*: dữ liệu của message được gán bằng msg\_pkt của Model message.
* *uint16\_t length*: độ dài của message.
* *bool force\_segmented*: message sẽ được tách nhỏ khi truyền.
* Địa chỉ đích và địa chỉ nguồn sẽ được gán trong *packet\_tx()*.

- Tham số khi truyền message: *nrf\_mesh\_tx\_params\_t*

* *nrf\_mesh\_address\_t dst*: địa chỉ đích mà model publish, gồm có type và value
* *uint16\_t src*: địa chỉ unicast của Element mà model gửi.
* *uint8\_t ttl*: Thời gian sống của message.
* *bool force\_segmented*: buộc message phải phân mảnh khi truyền.
* *const uint8\_t \* p\_data*: con trỏ tới dữ liệu ứng dụng.
* *uint16\_t data\_len*: độ dài dữ liệu ứng dụng.
* *nrf\_mesh\_transmic\_size\_t transmic\_size*: kích thước message truyền ở lớp transport
* *nrf\_mesh\_secmat\_t security\_material*: các thành phần bảo mật gồm Appkey và Netkey.

### *3.2.3/ Cấu trúc dữ liệu của message khi nhận tại Node server:*

- Cấu trúc dữ liệu của message nhận được khi xử lý ở lớp Access: *access\_message\_rx\_t*

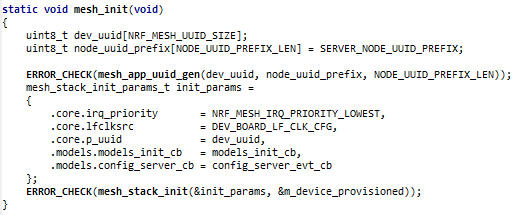
* *access\_opcode\_t opcode*: opcode của message.
* *const uint8\_t \* p\_data*: con trỏ tới byte đầu của dữ liệu message (msg\_pkt).
* *uint16\_t length*: độ dài message nhận được.
* *access\_message\_rx\_meta\_t meta\_data*: các dữ liệu của lớp access
* *nrf\_mesh\_address\_t src*: địa chỉ nguồn của message.
* *nrf\_mesh\_address\_t dst*: địa chỉ đích của message.
* *uint8\_t ttl*: time to live của message (sẽ giảm khi được Node trung gian relay).
* *dsm\_handle\_t appkey\_handle*: Appkey để giải mã message.
* *dsm\_handle\_t subnet\_handle*: Netkey để giải mã message.

### *3.2.4/ Khởi tạo:*

- Khởi tạo BLE:

* *gap\_params\_init()*: thiết lập chế độ sercurity open và tên của thiết bị advertise trong DEVICE\_NAME.
* *conn\_params\_init()*: khởi tạo các tham số BLE, gồm có connection interval, slave latency, supervision timeout.
* Enable softdevice, nơi chứa các hàm xử lý sự kiện BLE.

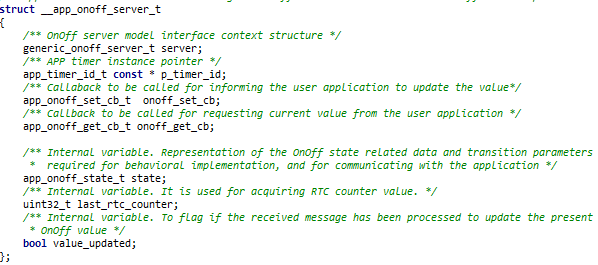
- Khởi tạo Mesh trong *mesh\_init()*:

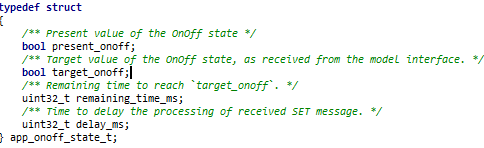


* Tạo một device UUID.
* irq\_priority: độ ưu tiên interrupt thấp nhất. Các sự kiện trong BLE nên được ưu tiên cao nhất.
* *models\_init\_cb()*: hàm dùng khởi tạo các model sau khi quá trình khởi tạo Mesh hoàn thành.
* *config\_server\_cb()*: sau khi khởi tạo thành công, gọi hàm reset trạng thái thiết bị về unprovisioned.
* *mesh\_stack\_init()* thực hiện khởi tạo các lớp trong mạng Mesh và gọi các hàm callback trên.
* Phương pháp PB-GATT được sử dụng để gia nhập mạng Mesh nên thực hiện khởi tạo *proxy\_init()*:
  + Service Mesh Proxy Provision có giá trị UUID là 0x1828, hai characteristic gồm Char\_TX có UUID 0x2ade và Char\_RX có UUID 0x2add.
  + Thiết lập advertising interval và hàm xử lý ngắt timer advertising.
* Bắt đầu advertise thông tin của thiết bị để App trên Smartphone tiến hành provision.

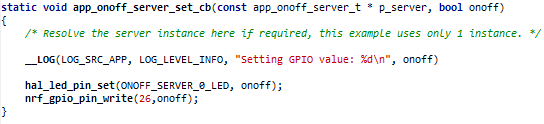
- Khởi tạo Model:

* *app\_onoff\_server\_t*: Cấu trúc dữ liệu lưu thông tin các hành động của model tại lớp Application:
  + *app\_onoff\_set\_cb\_t*: hàm callback được gọi sau khi cập nhật state mới từ SET message mà Server đã nhận, đây là hàm tập trung vào ứng dụng, sử dụng state từ SET message thực hiện bật tắt đèn hoặc gửi giá trị vừa nhận lên server.
  + *app\_onoff\_state\_t*: có thể thay thế hai giá trị state bằng một số integer cho các ứng dụng gửi dữ liệu.
  + *value\_updated*: biến thông báo state của Element đã được update.
  + *generic\_onoff\_server\_t server*: chứa thông tin về force\_segmented, transmic\_size và hàm callback khi nhận SET message. Hàm callback này khác với hàm trên, thực hiện update state của Element và gọi tới hàm *app\_onoff\_set\_cb\_t* ứng dụng ở trên.

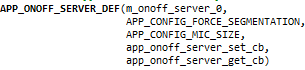




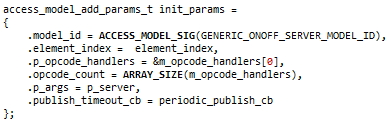
* Đầu tiên khởi tạo một *app\_onoff\_server\_t* instance của server model ở lớp application trong hàm main. *app\_onoff\_server\_set\_cb()* trong main.c được gán vào *app\_onoff\_set\_cb\_t* của struct trên. Hàm này sẽ thực hiện bật tắt đèn khi nhận được SET message.



* + Force segmentation là tùy chỉnh cho bắt buộc tách nhỏ message để gửi đi.
  + MIC size là kích thước message ở lớp transport.



* *models\_init\_cb()* được gọi sau khi quá trình provisioning hoàn tất, thực hiện khởi tạo và gán model instance với một Element (Element index), gán hàm thực hiện update state vào *generic\_onoff\_server\_t.server.p\_callback* (generic\_onoff\_state\_set\_cb(), đây là hàm thực hiện chuyển state, được gọi khi SET message nhận được xử lý xong ở lớp Access).
* Các tham số khởi tạo model: Mỗi model đều có một trường opcode và ID được định nghĩa bởi SIG. Trong phần opcode có thêm hàm xử lý sự kiện khi nhận được message trong phần khai báo *m\_opcode\_handlers*. Các hàm này cũng gọi tới hàm update state và application ở trên. Một model instance chỉ áp dụng cho một Element.



* access.c: Model khởi tạo bằng hàm *access\_model\_add()* được lưu vào một mảng m\_model\_pool và có danh sách các địa chỉ mà model subscript hoặc publish. Thông số TTL của Node có thể điều chỉnh tại biến gobal *m\_default\_ttl* trong file access.c .

- Khởi tạo model client: phía node Client có hai model ứng với hai Element đại diện cho hai cặp nút nhấn. Các bước khởi tạo cũng giống như model On/Off server nhưng cấu trúc dữ liệu của Client khác với Server và đã đưa ra ở phần các thông số của một SET message ở lớp Model.

### 

### *3.2.4/ Cách gửi một SET message phía Client:*

- Hàm *button\_event\_handler()* được gọi khi nhấn nút:

* Trước tiên khai báo các thông số của message: *generic\_onoff\_set\_params\_t* và *model\_transition\_t*
* Trong trường hợp nút nhấn là 0,2 thì *set\_params.on\_off* = 1 tương đương với bật đèn và 1,3 thì *set\_params.on\_off* = 0.
* Tăng biến tid để thông báo message này đã được gửi.
* Trường hợp nút nhấn 0,1 thì message sẽ được gửi bởi Client model instance 0 và 2,3 message được gửi bởi Client model instance 1.
* Hàm *generic\_onoff\_client\_set()* được gọi và tiến hành gán các tham số vào struct của message lớp Access bằng hàm *message\_set\_packet\_create()* và *message\_create().*

- Hàm *access\_model\_publish()* thực hiện gửi message đến địa chỉ mà model publish:

* Địa chỉ nguồn được gán bằng địa chỉ unicast của Element. Địa chỉ đích được gán bằng địa chỉ mà model đã publish.
* Gán các tham số của access message, appkey, TTL … vào struct của message để truyền.
* Message được đưa xuống lớp Upper transport và thực hiện mã hóa với Appkey rồi đưa xuống lớp lower transport.
* Nếu có yêu cầu phân tách message thì lower transport sẽ thực hiện. Sau đó chuyển message xuống lớp network.
* Tại lớp network, message tiếp tục được mã hóa bằng Netkey. Sau đó message được chuyển xuống bearer và phát bằng BLE.

### *3.2.5/ Các sự kiện xảy ra khi nhận một message:*

- Khi nhận một message, hàm *network\_packet\_in()* của lớp network sẽ được gọi để thực hiện bộ lọc message, giải mã với Netkey và kiểm tra có relay message hay không bằng hàm *should\_relay()*, với điều kiện TTL >= 2 và tính năng relay enable. Nếu message được thông qua bộ lọc, nó được gửi lên lớp lower transport, *transport\_packet\_in()*

- *transport\_packet\_in()* kiểm tra message đã có trong cache hay chưa. Nếu message được phân tách thì lower transport tiến hành gộp lại và đưa lên Upper transport.

- *upper\_transport\_packet\_in()* phân dữ liệu thành 2 loại là Control message và Access message. Các message điều khiển thuộc dạng Access message. Tại đây lớp Upper transport tiến hành giải mã message và gọi sự kiện trong lớp Access.

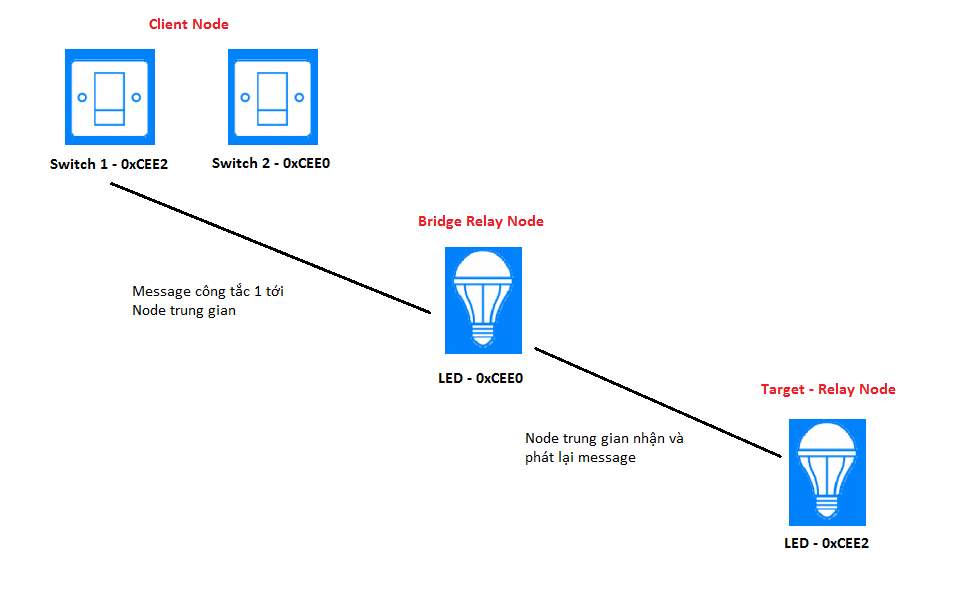
- *access\_incoming\_handle()* được gọi khi message được chuyển lên từ lớp Transport. Hàm kiểm tra địa chỉ đích với so với các địa chỉ mà các model trong m\_model\_pool đã subcribe và opcode của message với opcode của model. Nếu thỏa điều kiện thì gọi hàm xử lý callback trong phần khởi tạo model opcode, *handle\_set()*.

- *generic\_onoff\_state\_set\_cb()* trong app\_onoff.c được gọi để cập nhật state của Node và gọi hàm callback *app\_onoff\_server\_set\_cb()* trong file main.c sẽ sử dụng dữ liệu message bật tắt đèn.

# **4/ Hiện thực và kết quả:**

## **4.1/ Hiện thực:**

- Mô hình mạng Mesh:



Hình 3.12: Mô hình hiện thực

- Smartphone là thiết bị Provisioner, các Node thực hiện quá trình provisioning bằng phương pháp PB-GATT.

- Generic OnOff model Client/Server được sử dụng cho mạng Mesh này, Button 1 và Button 2 đại diện cho Switch On/Off 1, Button 3 và 4 đại diện cho switch On/Off 2.

- Trong mạng mesh chỉ có 1 Node trung gian nên chỉ cần một lần relay. Giá trị TTL = 2, được thiết lập trong biến m\_default\_ttl trong file access.c. Message khi được Node trung gian relay thì TTL giảm xuống còn 1, tránh hiện tượng vòng lặp message khi các Node relay qua lại lẫn nhau, làm giảm thông lượng của mạng.

- Phải đảm bảo việc Node trung gian đã nhận được tín hiệu ổn định thì mới có thể hiện thực tính năng relay.

- Khoảng cách giữa các node có thể điều khiển ổn định:

## **4.2/ Kết quả:**

- Với mô hình và khoảng cách như trên thì các message đều đến được Node đích một cách ổn định. Xác suất Node đích nhận được tín hiệu khoảng 90%.

- Khoảng cách giao tiếp giữa các Node trong không gian nhiều người qua lại, vật cản sẽ giảm. Hơn nữa các board được đặt ở trên bàn, làm giảm khả năng thu phát sóng. Khoảng cách thu phát sẽ tăng khu board cầm trên tay hoặc trong không gian hạn chế tiếp xúc mặt phẳng.

- Khoảng cách giao tiếp giữa hai Node còn phụ thuộc vào loại module phát sóng Bluetooth.

- Với 3 Node thì chỉ có một đường dữ liệu duy nhất được thực hiện, nếu có phát sinh lỗi thì message sẽ không đến được Node đích.

## **4.3/ Các mặt hạn chế:**

- Chưa hiểu sâu về quá trình Provisioning, các quá trình mã hóa – giải mã message.

- Khoảng cách giao tiếp ổn định giữa hai Node chưa tối ưu, khi áp dụng vào thực tế có thể phát sinh thêm nhiều Node khi ứng dụng những nơi không gian rộng, ngoài trời.

- Hiện chưa có công cụ để đo mức độ tiêu thụ điện.

## **4.4/ Nhận xét về Bluetooth Mesh:**

- Các SDK dành cho mạng Bluetooth Mesh luôn được cập nhật thường xuyên (hiện tại đã có phiên bản 3.1.0) cùng các hướng dẫn và diễn đàn về Bluetooth Mesh ngày càng phát triển giúp cho việc triển khai ứng dụng mạng Mesh trở nên dễ hơn.

- Các thiết bị Mobile hiện nay đều có chip Bluetooth nên có thể tương tác dễ dàng với mạng Mesh thông qua PB-GATT. Hiện tại đã có ứng dụng trên mobile có thể thực hiện provision và tương tác với các Node, tương lai có thể phát triển nhiều hơn các ứng dụng điều khiển và quản lý mạng Mesh.

- Với khoảng cách hiện thực như trên, Bluetooth Mesh có thể ứng dụng vào các ngôi nhà có kích thước phòng không quá lớn, đảm bảo một Node có thể điều khiển được toàn bộ một căn phòng. Đối với không gian lớn như nhà máy thì sẽ cần thêm nhiều Node để đảm bảo độ phủ sóng.

- Khả năng tiết kiệm năng lượng của Bluetooth Mesh sẽ không như Bluetooth Low Energy gốc vì các thao tác xử lý packet nhiều hơn nhưng năng lượng tiêu thụ vẫn ở mức thấp.

- Bluetooth Mesh sử dụng kỹ thuật managed flooding để truyền:

* Ưu điểm của kỹ thuật này là khả năng tạo ra nhiều đường đi cho message gia tăng độ tin cậy của mạng. Ngoài ra tốc độ truyền dữ liệu cũng nhanh hơn do không cần phải xác định đường đi.
* Một điểm quan trọng của kỹ thuật flooding là message có nhiều đường đi để đến được Node đích, tăng độ tin cậy của mạng.
* Khi một Node rời mạng, các Node khác sẽ không bị ảnh hưởng, dễ bảo trì.
* Các Node khi nhận message chỉ cần kiểm tra Time-to-live và thực hiện relay nên số lượng relay Node càng lớn thì thông lượng mạng càng giảm. Vì thế khi cài đặt mạng Mesh cần cân nhắc việc enable tính năng relay và điều chỉnh TTL phù hợp với vị trí của mỗi Node.

## **4.5/ Những khó khăn gặp phải:**

- Công nghệ Bluetooth Mesh mới được giới thiệu cách đây hơn năm các tài liệu tổng hợp giải thích kỹ về mạng Mesh chưa nhiều, mất thời gian thu thập và tổng hợp kiến thức.

- Khoảng cách giao tiếp được giữa hai Node khá xa nên khó khăn trong vấn đề đo khoảng cách vì không gian hạn chế. Vì vậy, đồ án ưu tiên thử nghiệm khoảng cách các Node giao tiếp ổn định nhất.

## **4.6/ Hướng phát triển:**

- Đặt các Node treo trong không gian để ít bị ảnh hưởng bởi vật cản.

- Nghiên cứu nhiều hơn về bản chất sóng Bluetooth và tìm giải pháp tối ưu khoảng cách giữa hai Node.

- Có thể thêm một Node trung gian nữa để tạo ra nhiều đường dữ liệu, tăng xác suất nhận được message.

- Tích hợp cảm biến và đèn vào chung một Node thông qua Element.

- Xây dựng một model dùng gửi dữ liệu cảm biến, một model điều chỉnh level của quạt, máy lạnh …

- Sử dụng một module kết nối Internet để gửi dữ liệu từ mạng Mesh lên server.

- Xây dựng công cụ quản lý các state của thiết bị trên server.

- Hiện thực gateway Node để các giao thức không dây khác có thể giao tiếp với mạng Mesh.

# 

# **Tài liệu tham khảo:**

\* Robert Davidson, Akiba, Carles Cufí, Kevin Townsend, 2014. *Getting Started with Bluetooth Low Energy*.

\* Nordic Documentation library:

Nrf SDK for Mesh:

[*https://www.nordicsemi.com/DocLib/Content/SDK\_Doc/Mesh\_SDK/v2-2-0/index*](https://www.nordicsemi.com/DocLib/Content/SDK_Doc/Mesh_SDK/v2-2-0/index)

nRF5 SDK v15.0.0:

[*https://www.nordicsemi.com/DocLib/Content/SDK\_Doc/nRF5\_SDK/v15-0-0/index*](https://www.nordicsemi.com/DocLib/Content/SDK_Doc/nRF5_SDK/v15-0-0/index)

\* Bluetooth SIG:

Mesh Technology Overview:

[*https://www.bluetooth.com/~/media/41f672fc201141a58c07e1ddc79bbe5c.ashx*](https://www.bluetooth.com/~/media/41f672fc201141a58c07e1ddc79bbe5c.ashx)

Mesh Profile Specification 1.0:

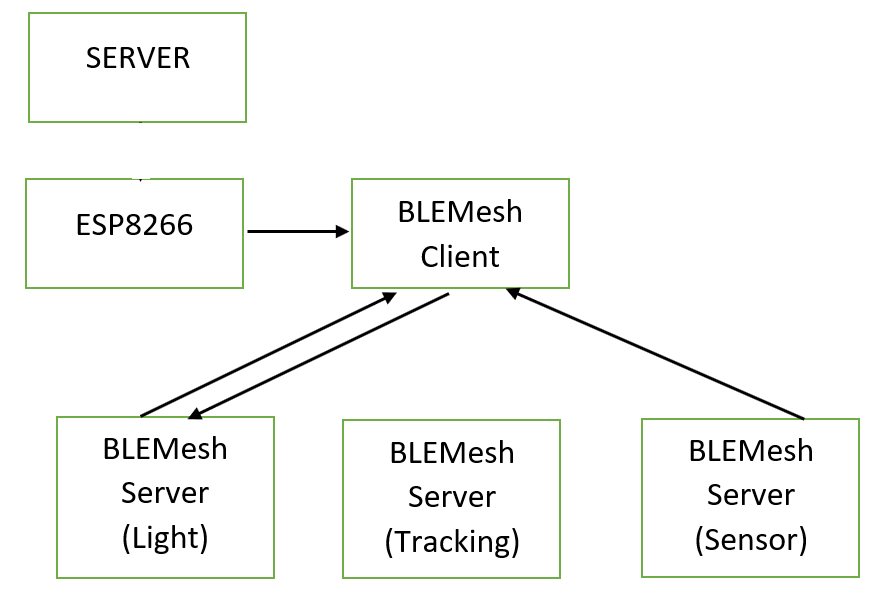
[*https://www.bluetooth.com/specifications/mesh-specifications*](https://www.bluetooth.com/specifications/mesh-specifications)

Provisioning:

<https://blog.bluetooth.com/provisioning-a-bluetooth-mesh-network-part-1>

<https://blog.bluetooth.com/provisioning-a-bluetooth-mesh-network-part-2>

**ANDROID**



**ESP8266**

**nRF52832**

Xác định các điểm tọa độ (x;y) trên bản đồ

Thu thập RSSI

Thu thập dữ liệu RSSI từ các Node Tracking tại các điểm tọa độ

Bộ lọc Kalman

Lưu các điểm tọa độ cùng với các giá trị RSSI

CSDL

Tìm điểm tọa độ gần giống nhất

**Tracking 2 + Flame**

**Tracking 1**

**Tracking 3**

0xC007

**E6**

0xC006

0xC008

**E7**

**E8**

0xC005

0xC004

**Tracking 4**

**E5**

**Tracking 5**

**E4**

0xC002

0xC001

0xC000

**Switch 2**

**E3**

**E1**

**Temp**

**Switch 1**

**E2**