RX23W グループ Bluetooth Mesh スタック

開発ガイド

要旨

Bluetooth® Mesh スタックは、Bluetooth Mesh Networking 仕様に準拠した Mesh ネットワークを構成して、多対多デバイス間で無線通信を実行するためのソフトウェアライブラリです。

本書は Bluetooth Mesh スタックパッケージのソフトウェア構成と各レイヤーの概要、Mesh アプリケーションの開発方法について示します。Bluetooth Mesh スタックパッケージの導入方法については「RX23W グループ Bluetooth Mesh スタック スタートアップガイド」(R01AN4874)を参照してください。

対象デバイス

RX23W グループ

関連文書

下記の文書がルネサスのウェブサイトにて公開されています。

文書名	文書番号
RX23W グループ ユーザーズマニュアル ハードウェア編	R01UH0823
CC-RX コンパイラ ユーザーズマニュアル	R20UT3248
Bluetooth Low Energy プロトコルスタック 基本パッケージ ユーザーズマニュアル	R01UW0205
RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド	R01AN5504
RX23W グループ Bluetooth Mesh スタック スタートアップガイド	R01AN4874
RX23W グループ Bluetooth Mesh スタック 開発ガイド	R01AN4875 本書

目次

1. Bluetooth Mesh 概要

本章は Bluetooth Mesh Networking 仕様で定義された Bluetooth Mesh の基本的な概念について示します。本仕様の詳細は、Bluetooth SIG ウェブサイトの Specifications List を参照してください。

図 1-1 に Bluetooth Mesh ネットワークの基本構成を示します。

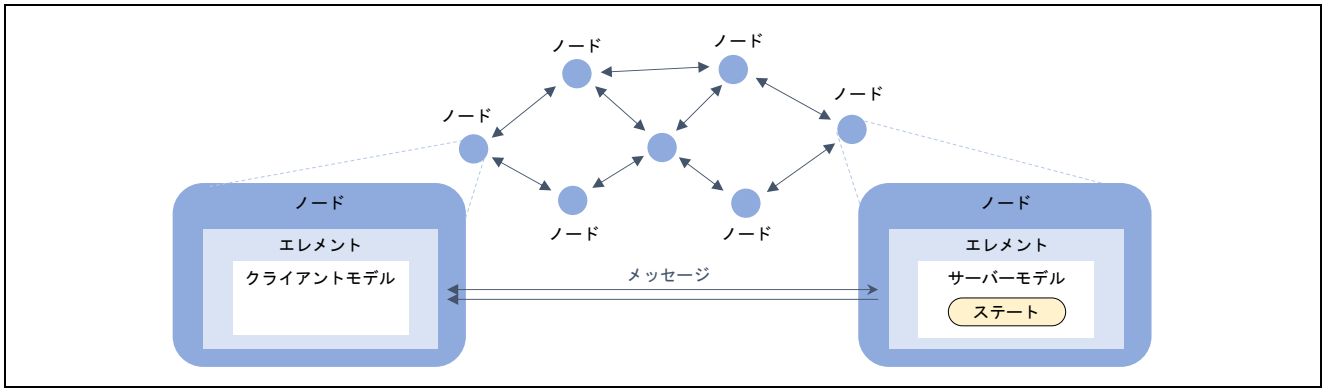

図 1-1　Bluetooth Mesh ネットワークの基本構成

1.1　ノード

ネットワークに参加しているデバイスはノード(Node)と呼ばれます。ネットワークは、アドレス空間と暗号鍵を共有するノードの集合です。ノード間の通信はネットワークキーによって暗号化されます。各ネットワークはネットワークキーから生成される Network ID で識別されます。デフォルトではプライマリサブネット(primary subnet)と呼ばれる 1 つのネットワークが構築されます。また通信対象を限定するために複数のサブネット(subnet)を構築することもできます。

1.2　エレメント

エレメント(Element)はノード内の論理的なオブジェクトです。ノードは少なくとも一つのエレメントを持ち、複数のエレメントを持つこともできます。最初のエレメントはプライマリエレメント(primary element)と呼ばれます。各エレメントはプロビジョニングによって割り当てられたユニキャストアドレスによって、ネットワーク上で一意に識別されます。

1.3　アドレス

ネットワークで使用されるアドレスは 16 ビット長です。アドレスタイプとして、未割り当てアドレス、ユニキャストアドレス、バーチャルアドレス、グループアドレスが定義されます。

表 1-1　アドレスタイプ

アドレスタイプ	値	値の範囲
未割り当てアドレス	0b0000000000000000	0x0000
ユニキャストアドレス	0b0xxxxxxxxxxxxxxx (0b0000000000000000を除く)	0x0001～0x7FFF
バーチャルアドレス	0b10xxxxxxxxxxxxx	0x8000～0xBFFF
グループアドレス	0b11xxxxxxxxxxxxx	0xC000～0xFFFF

- **未割り当てアドレス(Unassigned address)**

  未割り当てアドレスは、ユニキャストアドレスを割り当てられていないエレメントに設定されるアドレスです。メッセージの発信元アドレスや宛先アドレスとして使用することはできません。

- **ユニキャストアドレス(Unicast Address)**

  ユニキャストアドレスは、一つのエレメントを一意に識別するためのアドレスです。1 つのネットワークでは 32,767 個のユニキャストアドレスを使用できます。メッセージの発信元アドレスや宛先アドレスとして使用できます。

- **バーチャルアドレス(Virtual Address)**

  バーチャルアドレスは、ラベル UUID(Label UUID)から生成されるマルチキャストアドレスです。メッセージの宛先アドレスとして使用できます。

  ラベル UUID は複数のエレメントを分類するための 128 ビット長の値です。この値は任意に生成することができ、OOB(Out-Of-Band)によってデバイス間で共有されます。またラベル UUID およびバーチャルアドレスを一元的に管理する必要はありません。

- **グループアドレス(Group address)**

  グループアドレスは一元的に管理され、用途に応じて動的に割り当てられるマルチキャストアドレスです。メッセージの宛先アドレスとして使用できます。また全ノード(all-nodes)といった固定グループアドレス(Fixed Group Addresses)が定義されます。

表 1-2 固定グループアドレス

固定グループアドレス	値
all-proxies	0xFFFC
all-friends	0xFFFD
all-relays	0xFFFE
all-nodes	0xFFFF

1.4 ステート

ステート(State)は、エレメントの状態を示す値です。2 つ以上の値で構成されるステートはコンポジットステーツ(Composite States)と呼ばれます。さらに、他のステートに連動して変化する関係性のあるステートは、バウンドステーツ(Bound States)と呼ばれます。

ステートは瞬時に、または時間を掛けて変化します。イニシャルステート(Initial State)からターゲットステート(Target State)までの時間は、トランジションタイム(Transition Time)と呼ばれます。またカレントステート(Current State)からターゲットステートまでの時間は、リメイニングタイム(Remaining Time)と呼ばれます。

1.5 モデル

モデル(Model)は、各ノードがアプリケーションのシナリオに従って動作するための基本的な機能を標準化したものです。モデルはステート、ステートに作用するメッセージ、関連する動作を定義します。

1.5.1 クライアントとサーバー

モデルはクライアントーサーバー構成です。サーバーモデルはステートを持ちますが、クライアントモデルはステートを持ちません。

クライアントモデルは GET メッセージでサーバーモデルのステートを取得、または SET メッセージや SET Unacknowledged メッセージでサーバーモデルに新しいステートを設定することができます。

サーバーモデルはステートの変化を通知する場合や、GET メッセージや SET メッセージを受信した場合に、ACK として STATUS メッセージを送信します。SET Unacknowledged メッセージを受信した場合、サーバーモデルは ACK として STATUS メッセージを送信しません。

ノードは複数のエレメントを持つことができます。また各エレメントは重複しない複数のモデルを使用することができます。サーバーモデルはクライアントモデルからメッセージを受信することで、エレメントのステートを制御します。

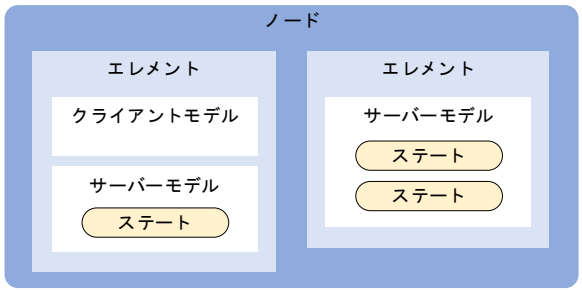

図 1-2　ノード構成

1.5.2　ファウンデーションモデル

ファウンデーションモデル(Foundation Models)は、各エレメントの動作を設定、管理するためのモデルです。各ノードのプライマリエレメントはコンフィグレーションサーバーモデル(Configuration Server Model)とヘルスサーバーモデル(Health Server Model)を持つ必要があります。

表 1-3　ファウンデーションモデル

モデル	SIG Model ID
Configuration Server	0x0000
Configuration Client	0x0001
Health Server	0x0002
Health Client	0x0003

1.5.3　コンフィグレーションモデル

コンフィグレーションモデル(Configuration Model)はノードの動作を設定するためのモデルです。ノードとエレメントの設定値はコンフィグレーションステートとして定義されます。

コンフィグレーションサーバーモデル(Configuration Server Model)はコンフィグレーションステート(Configuration State)を持つモデルです。一方、コンフィグレーションクライアントモデル(Configuration Client Model)はコンフィグレーションメッセージによって、コンフィグレーションサーバーの設定を管理するモデルです。コンフィグレーションメッセージはデバイスキー(Device Key)によって暗号化されます。デバイスキーは各ノードによって異なり、プロビジョニング時に生成されます。

1.5.4　ヘルスモデル

ヘルスモデル(Health Model)はノードの物理的な状態を監視するためのモデルです。

ヘルスサーバーモデル(Health Server Model)は物理的な障害情報を保持するためのフォールトステート(Fault State)を持つモデルです。一方、ヘルスクライアントモデル(Health Client Model)はヘルスメッセージによって、ヘルスサーバーの障害情報を監視するモデルです。ヘルスメッセージはアプリケーションキー(Application Key)によって暗号化されます。

1.5.5　パブリケーションとサブスクリプション

モデルのメッセージ送信動作はパブリケーション(Publication)、メッセージ受信動作はサブスクリプション(Subscription)と呼ばれます。モデルは、マルチキャストアドレスを宛先アドレスとして設定することで、複数エレメントに対してメッセージをパブリッシュ(Publish)することができます。またモデルはマルチキャストアドレス宛のメッセージを選択的にサブスクライブ(Subscribe)することができます。

図 1-3 にモデルによるメッセージのパブリッシュとサブスクライブを示します。各モデルはモデルパブリケーションステートのパブリッシュアドレスに従ってメッセージを送信します。パブリッシュアドレスがマルチキャストアドレスの場合、各メッセージはサブスクリプションリストステートのサブスクリプションアドレスに従って複数のモデルからサブスクライブされます。

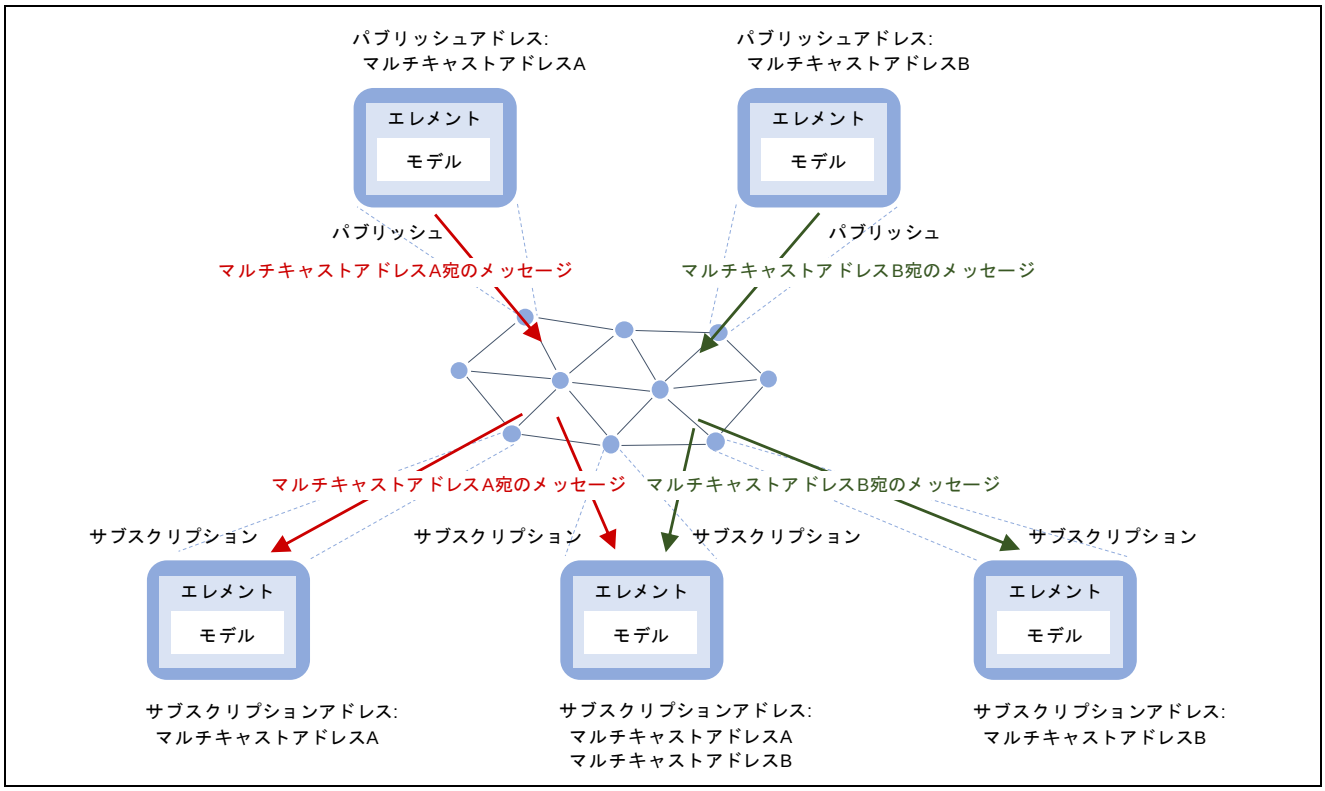

図 1-3　モデルによるメッセージのパブリッシュとサブスクライブ

1.6 メッセージ

ネットワークのノードによって送受信されるデータはメッセージ(Message)と呼ばれます。

メッセージフォーマットとして、Unsegmented メッセージと Segmented メッセージが定義されます。

- **Unsegmented メッセージ**

  Unsegmented メッセージは、セグメント化されていないデータを転送するためのメッセージです。11byte までの Access PDU を転送できます。

- **Segmented メッセージ**

  Segmented メッセージは、最大 32 セグメントまでセグメント化されたデータを転送するためのメッセージです。380byte までの Access PDU を転送できます。宛先ノードは全ての Segmented メッセージを受信後、データを再結合します。

図 1-4 に Access PDU のセグメント化と再結合を示します。各ノードは Network PDU を Mesh メッセージとして送受信します。

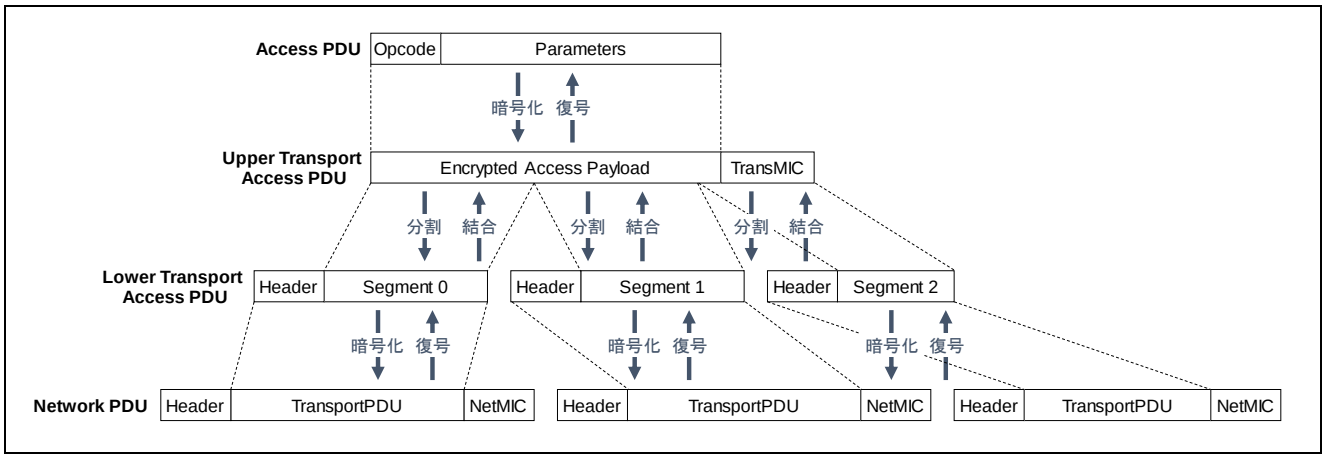

図 1-4 Access PDU のセグメント化と再結合

Network PDU のヘッダは発信元アドレス(SRC)や宛先アドレス(DST)、シーケンス番号(SEQ)といったフィールドを含みます。Network PDU はネットワークキー(Network Key)で暗号化されるため、同一のネットワークに参加しているノードだけがメッセージを復号化することができます。また、発信元アドレスや宛先アドレスは難読化(Obfuscation)されるため、ネットワークキーを持たないデバイスにそれらを特定されることはありません。

Lower Transport PDU のヘッダは Segmented か Unsegmented かを示す SEG やセグメント化されたデータを再結合するための SeqZero、SegO、SegN といったフィールドを含みます。

Access PDU はアプリケーションオペコード(Opcode)とアプリケーションパラメータ(Parameters)の 2 つのフィールドで構成されます。また Access PDU はアプリケーションキー(Application Key)またはデバイスキーで暗号化されます。これによってアプリケーションキーまたはデバイスキーを共有するノード間でのみ、データを共有することができます。アプリケーションキーはコンフィグレーションクライアントによって生成、配布されます。

1.7 Mesh ベアラー

Mesh ベアラー(Mesh Bearer)は、Mesh ネットワークにおいてメッセージを運搬する手段です。Bluetooth Low Energy 技術を利用した 2 つのベアラーが定義されます。

- **ADV ベアラー**

  本ベアラーは Non-Connectable and Non-Scannable Undirected Advertising でメッセージを送信します。ADV ベアラーが送信したメッセージは同時に複数のノードが受信できます。

  なおプロビジョニングでプロビジョニング PDU を運搬する場合、本ベアラーは PB-ADV と呼ばれます。

- **GATT ベアラー**

  本ベアラーは GATT サービスを通してメッセージを送信します。クライアント側のノードは Write Without Response によってメッセージを送信し、サーバー側のノードは Notification によってメッセージを送信します。GATT サービスを通して通信する前に、接続を確立する必要があります。GATT ベアラーが送信したメッセージは接続された対向ノードのみが受信できます。

  なおプロビジョニングでプロビジョニング PDU を運搬する場合、本ベアラーは PB-GATT と呼ばれます。

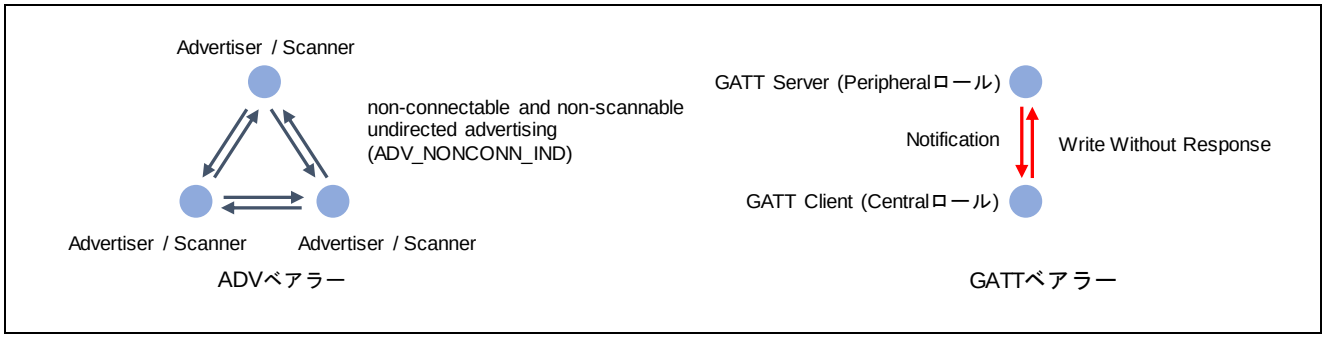

図 1-5 ADV ベアラーと GATT ベアラー

1.8　プロビジョニング

プロビジョニングはネットワークに参加するための処理です。プロビジョニングではプロビジョニングデータが配布されます。プロビジョニングデータは下記の情報を含みます。

- ネットワークキーとネットワークキーインデックス
- キーリフレッシュフラグと IV アップデートフラグ
- カレント IV インデックス
- プライマリエレメントのユニキャストアドレス

ネットワークに参加していないデバイスはアンプロビジョンドデバイス(Unprovisioned Device)と呼ばれます。各アンプロビジョンドデバイスは 128 ビットデバイス UUID(Device UUID)によって識別されます。

デバイスをネットワークに招待してプロビジョニングデータを配布するデバイスは、プロビジョニングクライアント(Provisioning Client)またはプロビジョナー(Provisioner)と呼ばれます。一般的に、プロビジョニングクライアントはスマートフォンまたはモバイル端末です。

プロビジョニングデータを受け取ってネットワークに参加するデバイスは、プロビジョニングサーバー(Provisioning Server)またはプロビジョニー(Provisionee)と呼ばれます。ネットワークへの参加が完了したデバイスはノードと呼ばれます。

1.9　コンフィグレーション

ノードがモデルを使用して他ノードと通信するためには、コンフィグレーション(Configuration)が必要です。コンフィグレーションによって、アプリケーションキーやパブリッシュアドレス、サブスクリプションアドレスといったモデル動作に必要な情報がノードに設定されます。

図 1-6 にノードの典型的なライフサイクルを示します。

新規導入されたデバイスはプロビジョナーによってプロビジョニングされることで、ネットワークに参加します。さらにコンフィグレーションクライアントによってコンフィグレーションされることで、他ノードとの Mesh モデルによる通信が可能となります。一般的に、コンフィグレーションクライアントはスマートフォンまたはモバイル端末です。

コンフィグレーションクライアントはノードに対して Config Node Reset メッセージを送信することで、ノードをネットワークから除外します。またコンフィグレーションクライアントはネットワークで使用される暗号鍵を更新するため、除外されたノードは他ノードとの通信が不能となります。

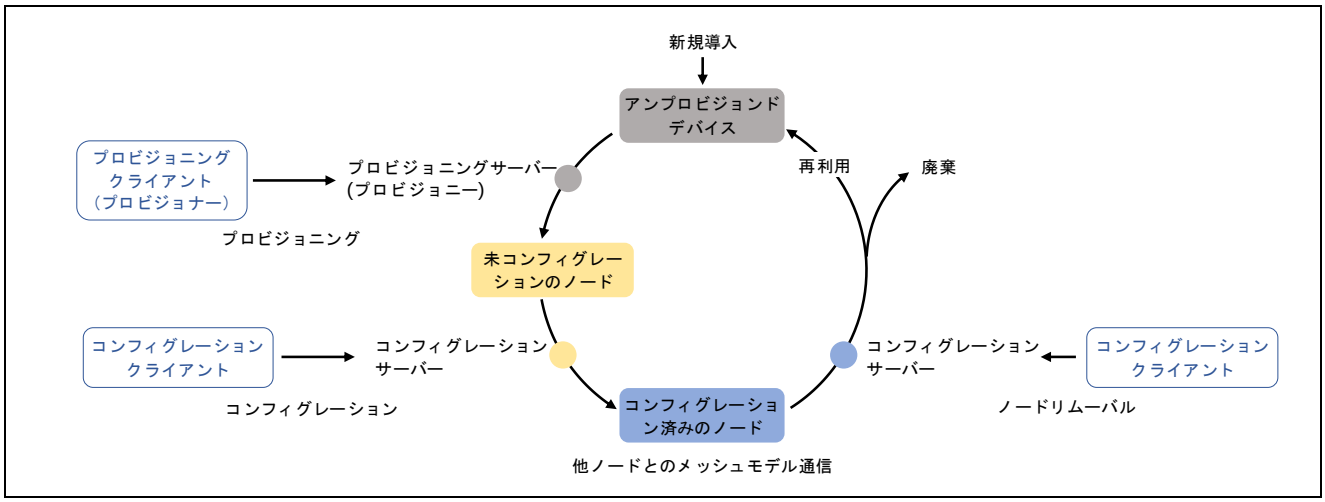

図 1-6　ノードのライフサイクル

1.10 オプション機能

下記の機能がオプション機能として定義されます。

- リレー機能
- プロキシ機能
- フレンド機能
- ローパワー機能

ノードが各オプション機能を有効化することで、多様な Mesh ネットワークを形成できます。

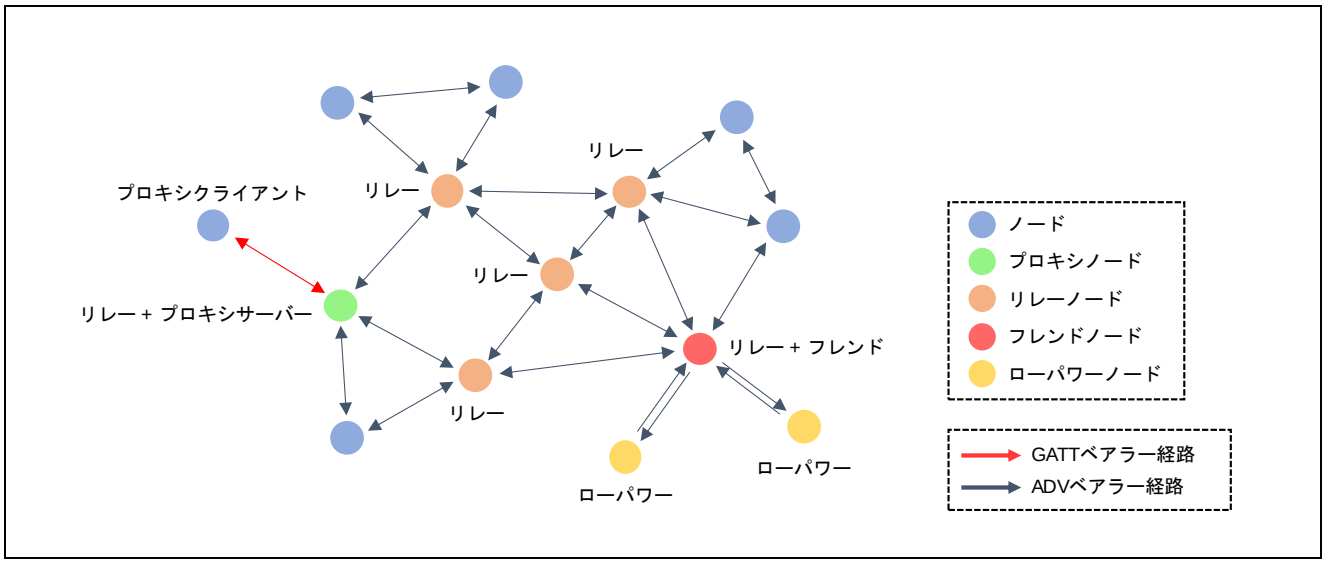

図 1-7　Mesh ネットワーク

1.10.1 リレー

リレー機能(Relay feature)は ADV ベアラーに対応するノードが、ADV ベアラーで受信したメッセージを再送信することで中継する機能です。これにより宛先ノードが発信ノードの無線到達範囲外であっても、メッセージを他のノードが次々と中継してネットワーク中に拡散することで、最終的にメッセージは宛先ノードに到達することができます。

メッセージを中継するノードは、リレーノード(Relay node)と呼ばれます。

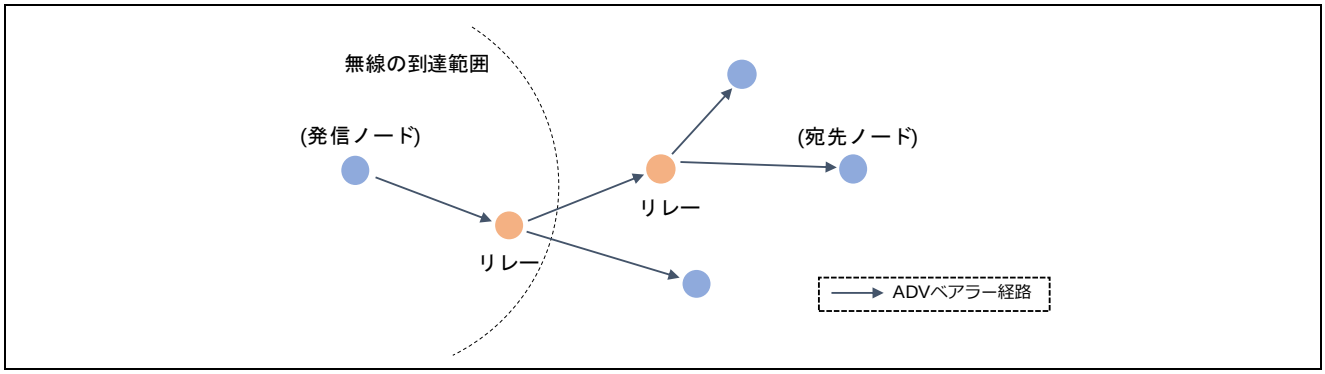

図 1-8　メッセージリレー

1.10.2 プロキシ

プロキシ機能(Proxy)は GATT ベアラーと ADV ベアラーの両方に対応するノードが、GATT ベアラーと ADV ベアラー間でメッセージを転送する機能です。

GATT ベアラーのみに対応するノードは、接続した対向ノードとしか通信できません。そこで本ノードはプロキシ機能に対応するノードと接続を確立します。これにより、本ノードが送信したメッセージは、プロキシノードが ADV ベアラーに転送し、最終的に宛先ノードに到達することができます。さらに他ノードが送信したメッセージは、プロキシノードが GATT ベアラーに転送し、本ノードに到達することができます。

GATT ベアラーと ADV ベアラー間でメッセージを転送するノードはプロキシサーバー(Proxy Server)と呼ばれます。またプロキシサーバーと接続して GATT ベアラーでメッセージを送受信するノードはプロキシクライアント(Proxy Client)と呼ばれます。

プロキシサーバーはプロキシクライアントのサブスクリプションアドレスを管理するためのリストを持ち、このリストはプロキシフィルタリストと呼ばれます。プロキシフィルタタイプとして、ホワイトリストフィルタまたはブラックリストフィルタを選択できます。プロキシフィルタタイプがホワイトリストフィルタの場合、プロキシサーバーはリストに登録されたアドレス宛のメッセージのみをプロキシクライアントに転送します。プロキシフィルタタイプがブラックリストフィルタの場合、プロキシサーバーはリストに登録されたアドレス宛のメッセージはプロキシクライアントに転送しません。

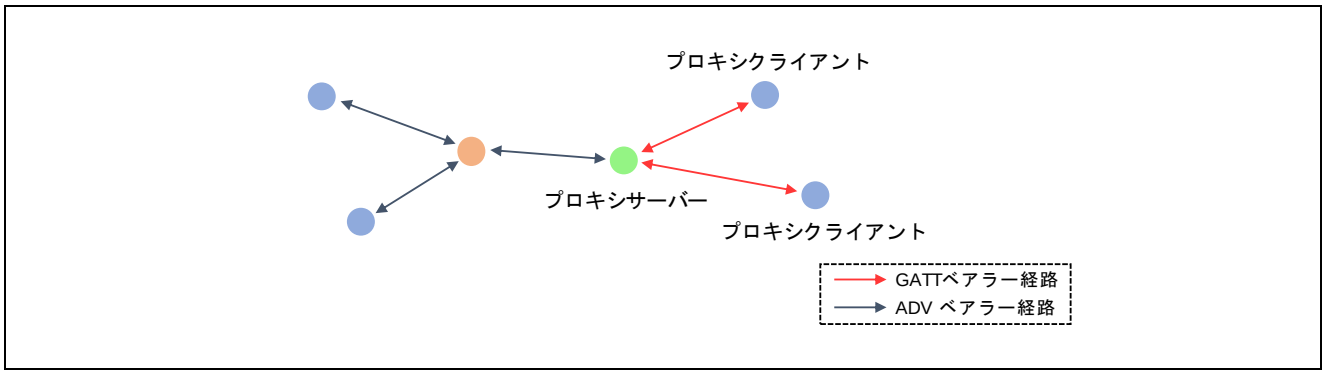

図 1-9　プロキシ

1.10.3 フレンドシップ

通常、ADV ベアラーに対応するノードは、メッセージを含む Advertising パケットを受信するため、常に Scan を実行します。ローパワー機能(Low Power)は、この Scan の実行時間を削減するための機能です。ローパワー機能に対応したノードは、Scan を休止することで消費電力を低減することができます。

ローパワー機能を実行するためには、フレンド機能(Friend feature)に対応したノードとフレンドシップ(Friendship)を確立する必要があります。フレンド機能はローパワーノード宛のメッセージを保持し、ローパワーノードが要求するタイミングで転送する機能です。

まずローパワーノードはフレンドノードに対して、フレンドとなることを要求します。フレンドノードがこれを承認することで、フレンドシップが確立します。フレンドシップの確立後、ローパワーノードは Scan を停止することができる一方、フレンドノードはローパワーノードが必要とするメッセージを保持しなければなりません。

フレンドノードは、ローパワーノードのサブスクリプションアドレスを管理するためのリストを持ち、このリストはフレンドサブスクリプションリストと呼ばれます。フレンドノードはフレンドシップの確立後、リストに登録されたマルチキャストアドレス宛のメッセージを保持します。

ローパワーノードは、フレンドノードにメッセージが保持されているかを間欠的に問い合わせ、問い合わせの期間だけ Scan を再開します。フレンドノードは保持しているローパワーノード宛のメッセージをこのタイミングで転送します。

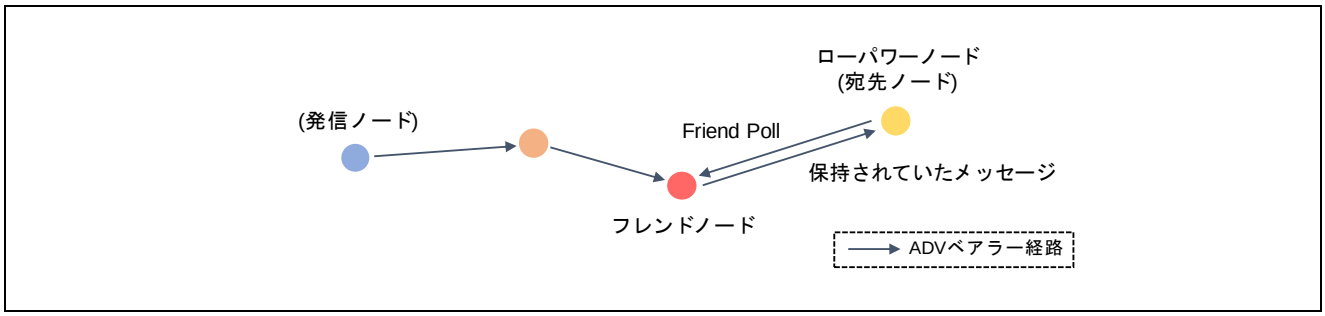

図 1-10 フレンドシップ

2. Bluetooth Mesh スタックパッケージ

本章は Bluetooth Mesh スタックパッケージに含まれるソフトウェアの概要を説明します。

Bluetooth Mesh スタックパッケージは Mesh アプリケーションのサンプルプログラムに加え、サンプルプログラムのビルドに必要な Bluetooth Mesh スタック、Bluetooth ベアラー、Bluetooth Low Energy プロトコルスタック、その他の FIT モジュールを含みます。さらに下記のデモプロジェクトを含みます。

tbrx23w_mesh_client:	Target Board for RX23W 向け Client モデルプロジェクト
tbrx23w_mesh_server:	Target Board for RX23W 向け Server モデルプロジェクト
tbrx23wmodule_mesh_client:	Target Board for RX23W module 向け Client モデルプロジェクト
tbrx23wmodule_mesh_server:	Target Board for RX23W module 向け Server モデルプロジェクト
rsskrx23w_mesh_client:	Renesas Solution Starter Kit for RX23W 向け Client モデルプロジェクト
rsskrx23w_mesh_server:	Renesas Solution Starter Kit for RX23W 向け Server モデルプロジェクト

デモプロジェクトの構成を以下に示します。本書では太字で示すソフトウェアについて解説します。各 FIT モジュールの詳細は各アプリケーションノートを参照してください。

```
{project}\                          :
    +---src\                        :
        |   main.c                  : Mesh サンプルプログラム
        |   mesh_appl.h             : Mesh サンプルヘッダ
        |   mesh_core.c             : Mesh コアモジュール
        |   mesh_model.c            : Mesh モデルモジュール
        |                           :
        +---vendor_model\           :
        |       vendor_api.h        : Vendor モデルヘッダ
        |       vendor_client.c     : Vendor Client モジュール
        |       vendor_server.c     : Vendor Server モジュール
        +---smc_gen\                :
            +---r_mesh_rx23w\       : Mesh FIT モジュール (R01AN4930)
            |   +---json\           : - QE for BLE 向け Mesh GATT サービス定義
            |   +---lib\            : - Mesh スタックライブラリ
            |   +---src\            :
            |       +---bearer\     : - Bluetooth ベアラー
            |       +---drivers\    : - Mesh ドライバ
            |       +---include\    : - Mesh スタックヘッダ
            +---r_ble_rx23w\        : BLE FIT モジュール (R01AN4860)
            |   +---lib\            : - Bluetooth Low Energy プロトコルスタックライブラリ
            +---r_bsp\              : BSP FIT モジュール (R01AN1685)
            +---r_byteq\            : BYTEQ FIT モジュール (R01AN1683)
            +---r_cmt_rx\           : CMT FIT モジュール (R01AN1856)
            +---r_flash_rx\         : FLASH FIT モジュール (R01AN2184)
            +---r_gpio_rx\          : GPIO FIT モジュール (R01AN1721)
            +---r_irq_rx\           : IRQ FIT モジュール (R01AN1668)
            +---r_lpc_rx\           : LPC FIT モジュール (R01AN2769)
            +---r_sci_rx\           : SCI FIT モジュール (R01AN1815)
            +---r_config\           : FIT モジュール設定
            +---r_pincfg\           : MCU 端子設定
```

サンプルプログラムのビルド環境構築については、「RX23W グループ Bluetooth Mesh スタック スタートアップガイド」(R01AN4874)の 6 章を参照してください。

2.1 システム構成

図 2-1 に Bluetooth Mesh スタックパッケージのシステム構成を示します。

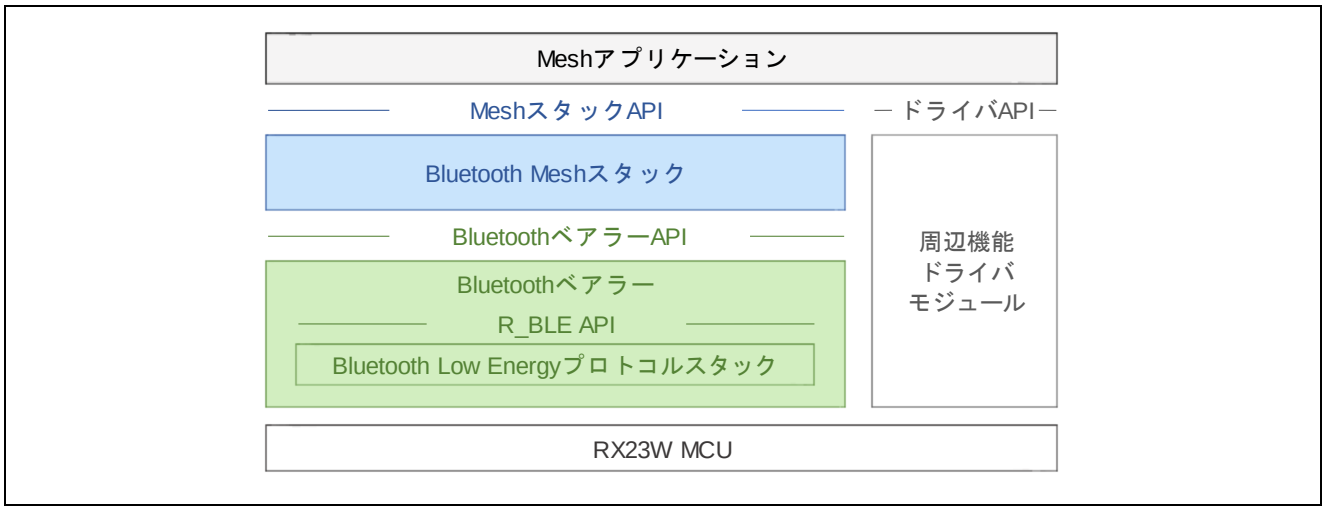

図 2-1 Bluetooth Mesh のシステム構成

Bluetooth Mesh スタックパッケージは下記のソフトウェアで構成されます。

- **Mesh アプリケーション**

  Mesh アプリケーションは、Bluetooth Mesh の無線通信機能を利用するアプリケーションプログラムです。ユーザは Mesh スタック API や Bluetooth ベアラーAPI の仕様を理解し、Mesh アプリケーションを開発する必要があります。なお Mesh アプリケーションのサンプルプログラムが Bluetooth Mesh スタックパッケージに含まれます。

- **Bluetooth Mesh スタック**

  Bluetooth Mesh スタック(以後、" Mesh スタック"と表記)は、Bluetooth Mesh Networking 仕様に準拠した多対多の無線通信機能をアプリケーションに提供するソフトウェアです。Mesh スタックは Mesh ネットワーク通信を利用するための Mesh スタック API を有します。なお Mesh スタックは RX23W グループ向け Mesh FIT モジュールに含まれます。

- **Bluetooth ベアラー**

  Bluetooth ベアラーは、Bluetooth Low Energy プロトコルスタックのラッパー関数を Mesh スタックとアプリケーションに提供する抽象化レイヤーです。Bluetooth ベアラーもまた RX23W グループ向け Mesh FIT モジュールに含まれます。

- **Bluetooth Low Energy プロトコルスタック**

  Bluetooth Low Energy プロトコルスタック(以後、"Bluetooth LE スタック"と表記)は、Bluetooth Low Energy 仕様に準拠した無線通信機能を上位レイヤーに提供するソフトウェアです。Bluetooth LE スタックは Bluetooth Low Energy 通信を利用するための R_BLE API を有します。なお Bluetooth LE スタックは RX23W グループ向け BLE FIT モジュールに含まれます。

- **周辺機能ドライバモジュール**

  アプリケーション、Mesh スタック、Bluetooth LE スタックはマイコンの周辺機能を利用します。RX マイコンのソフトウェア開発では Firmware Integration Technology(FIT)モジュールとして提供される周辺機能ドライバを利用できます。

2.2 Mesh アプリケーション

Bluetooth Mesh の無線通信機能を利用する Mesh アプリケーションは、ユーザが開発する必要があります。Bluetooth Mesh スタックパッケージは、Mesh アプリケーション開発のリファレンスとなるサンプルプログラムのソースコードを含みます。

Mesh アプリケーションのサンプルプログラム(以後、" Mesh サンプルプログラム"と表記)は、Mesh スタック API を利用して、プロビジョニングと Mesh ノードとしての基本的な動作を実行します。以降、Mesh サンプルプログラムについて解説します。

Mesh サンプルプログラムの主な特徴を示します。

- Unprovisioned Device 動作: PB-ADV ベアラーと PB-GATT ベアラーの両方に対応
- Configuration Server 動作: コンフィグレーション情報をデータフラッシュメモリに保存
- Generic OnOff Client 動作: RX23W 向け開発ボードのスイッチ押下で Generic OnOff Set メッセージを送信
- Generic OnOff Server 動作: Generic OnOff Set メッセージの受信で RX23W 向け開発ボードの LED を制御
- Vendor Client 動作: UART 経由で入力された文字列を Vendor Set メッセージで送信
- Vendor Server 動作: Vendor Set メッセージで受信した文字列を UART 経由で出力
- Low Power 動作: Friend ノードと Friendship を確立後、Friend Subscription List に Subscription アドレスを登録
- Proxy Server 動作: Proxy Client と接続を確立後、GATT ベアラー経由でメッセージを転送
- IV Update 開始機能: 送受信メッセージのシーケンス番号を監視し、閾値を超えた場合に IV Update を開始
- Mesh モニタ機能: 送受信メッセージをモニタリングして UART でログ出力

本サンプルプログラムは次の 2 つのモジュールを含みます。

- **Mesh コアモジュール**
  本モジュールはプロビジョニングサーバーとしてプロビジョニングを実行し、プロビジョニングの完了後はプロキシサーバーとして GATT ベアラーを有効化します。また Low Power ノードとして Friendship を制御します。

- **Mesh モデルモジュール**
  本モジュールはコンフィグレーションサーバーモデルとヘルスサーバーモデルに加え、Generic OnOff モデルと独自 Vendor モデルに関する動作を実行します。

2.2.1 Mesh コアモジュール

Mesh サンプルプログラムに含まれる Mesh コアモジュールは次の処理を実行します。本モジュールは mesh_core.c に実装されています。

- プロビジョニング処理
- プロキシ機能
- ローパワー機能
- IV Update 機能
- Mesh モニタ機能

2.2.2 Mesh モデルモジュール

Mesh サンプルプログラムに含まれる Mesh モデルモジュールは次の処理を実行します。本モジュールは mesh_model.c に実装されています。

- Mesh モデルの構成
- コンフィグレーションモデル
- Generic OnOff モデル
- Vendor モデル

2.2.3 Mesh モデル構成

本サンプルプログラムでは次のモデルを使用します。

- Configuration Server モデル
- Health Server モデル
- Generic OnOff Server モデル
- Generic OnOff Client モデル
- Vendor Server モデル
- Vendor Client モデル

図 2-2 に Mesh サンプルプログラムの各プロジェクトのモデル構成を示します。プライマリエレメントには Configuration Server モデル、Health Server モデルに加え Generic OnOff Server モデル、Generic OnOff Client モデル、Vendor Server モデル、Vendor Client モデルが配置されます。

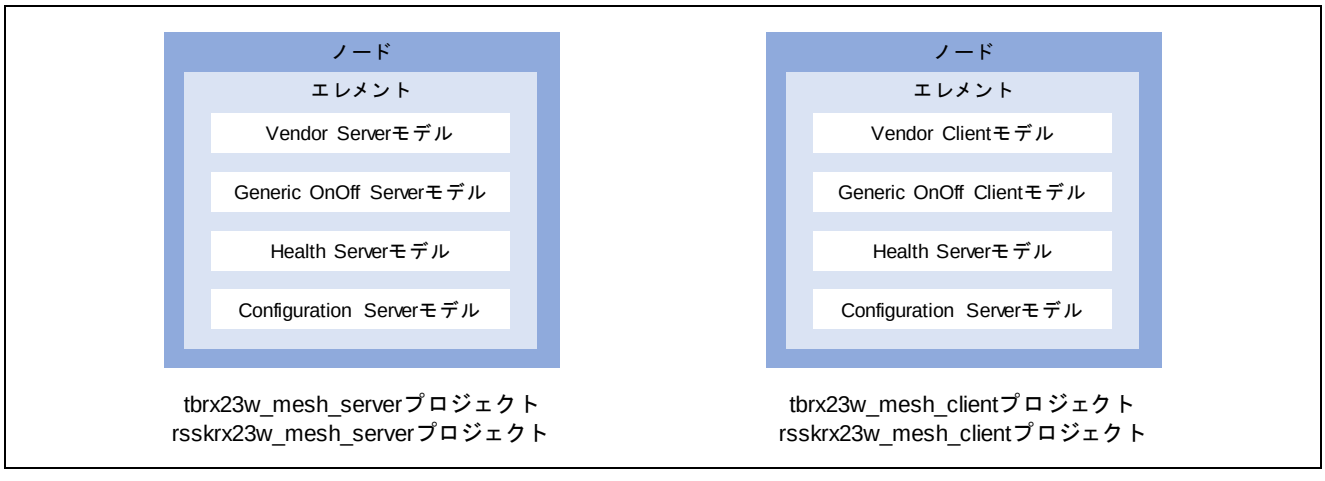

図 2-2 Mesh サンプルプログラムのモデル構成

2.2.3.1 コンフィグレーションモデル

コンフィグレーションモデルはノードを設定するためのモデルです。コンフィグレーションサーバーはノードやエレメント、モデルの動作設定を保持する複数のコンフィグレーションステートを持ちます。これらのステートはコンフィグレーションクライアントからのメッセージによって操作されます。

コンフィグレーションモデルが定義するステートやメッセージの詳細は Mesh Profile 仕様書の Chapter 4 "Foundation models"を参照してください。

表 2-1 コンフィグレーションモデルのステート

モデル名	SIG モデル ID (16 ビット)	ステート
Configuration Server	0x0000	Secure Network Beacon Composition Data Default TTL GATT Proxy Friend Relay Model Publication Subscription List NetKey List AppKey List Model to AppKey List Node Identity Key Refresh Phase Heartbeat Publish Heartbeat Subscription Network Transmit Relay Retransmit PollTimeout List
Configuration Client	0x0001	-

コンフィグレーションステートを格納するメモリ領域は Mesh スタックに確保されます。Mesh スタックはコンフィグレーションメッセージを受信するとコンフィグレーションステートを自動的に更新するため、アプリケーションがこれらを管理する必要はありません。またアプリケーションは Mesh スタック API を使用することで各コンフィグレーションステートにアクセスできます。

2.2.3.2 ヘルスモデル

ヘルスモデルはノードの物理的な状態を監視するためのモデルです。ヘルスサーバーはノードの物理的な障害状態を保持するためのフォールトステートを持ちます。このステートは障害発生時に更新されます。またヘルスクライアントからのメッセージによって、ノードのセルフテストを実行することができます。

またヘルスサーバーは人の注意を引き付ける仕組み(LED の点滅やノイズの生成など)を動作させるためのアテンションタイマステートも持ちます。

ヘルスモデルが定義するステートやメッセージの詳細は Mesh Profile 仕様書の Chapter 4 "Foundation models"を参照してください。

表 2-2 ヘルスモデルのステート

モデル名	SIG モデル ID (16 ビット)	ステート
Health Server	0x0002	Current Fault Registered Fault Health Period Attention Timer
Health Client	0x0003	-

ヘルスステートを格納するメモリ領域は Mesh スタックに確保されます。

2.2.3.3 Generic OnOff モデル

Generic OnOff モデルは Bluetooth SIG が定義するモデルです。Generic OnOff Server は On または Off を保持する Generic OnOff ステートを持ちます。本ステートは Generic OnOff Client からのメッセージによって操作されます。

Generic OnOff モデルが定義するステートやメッセージの詳細は Mesh Model 仕様書の Chapter 3 "Generics"を参照してください。

表 2-3 Generic OnOff モデルのステート

モデル名	SIG モデル ID (16 ビット)	ステート
Generic OnOff Server	0x1000	Generic OnOff (0x00: Off, 0x01: On)
Generic OnOff Client	0x1001	-

Generic OnOff ステートを保持するためのメモリ領域はアプリケーションで確保する必要があります。Mesh スタックは受信した Generic OnOff メッセージをコールバック関数で通知します。アプリケーションはコールバック関数で通知されたメッセージに従って Generic OnOff ステートを更新してください。

2.2.3.4 Vendor モデル

ユーザは Mesh スタックが提供する Access レイヤーの API を使用して、独自のモデルを定義することもできます。

ここでは Mesh サンプルプログラムに実装された Vendor モデルについて解説します。Vendor Server は任意の可変長データを保持する Vendor ステートを持ちます。本ステートは Vendor Client からのメッセージによって操作されます。

表 2-4 Vendor モデルのステート

モデル名	Vendor モデル ID (32 ビット)	ステート
Vendor Server	0x00010036 (デフォルト値)	Vendor state (任意の可変長データ)
Vendor Client	0x00020036 (デフォルト値)	-

表 2-5 Vendor メッセージ

ステート	メッセージ名	オペコード	方向
Vendor	Vendor Get	0xC10036 (デフォルト値)	Client → Server
	Vendor Set	0xC20036 (デフォルト値)	Client → Server
	Vendor Set Unacknowledged	0xC30036 (デフォルト値)	Client → Server
	Vendor OnOff Status	0xC40036 (デフォルト値)	Server → Client

2.3 Bluetooth Mesh スタック

Bluetooth Mesh スタックは Bluetooth Mesh Networking 仕様に準拠した多対多デバイス間で無線通信機能をアプリケーションに提供します。Mesh スタックのライブラリはパッケージに同梱されており、Mesh スタック API によって Mesh 機能を利用することができます。

図 2-3 に Bluetooth Mesh スタックの内部構成を示します。

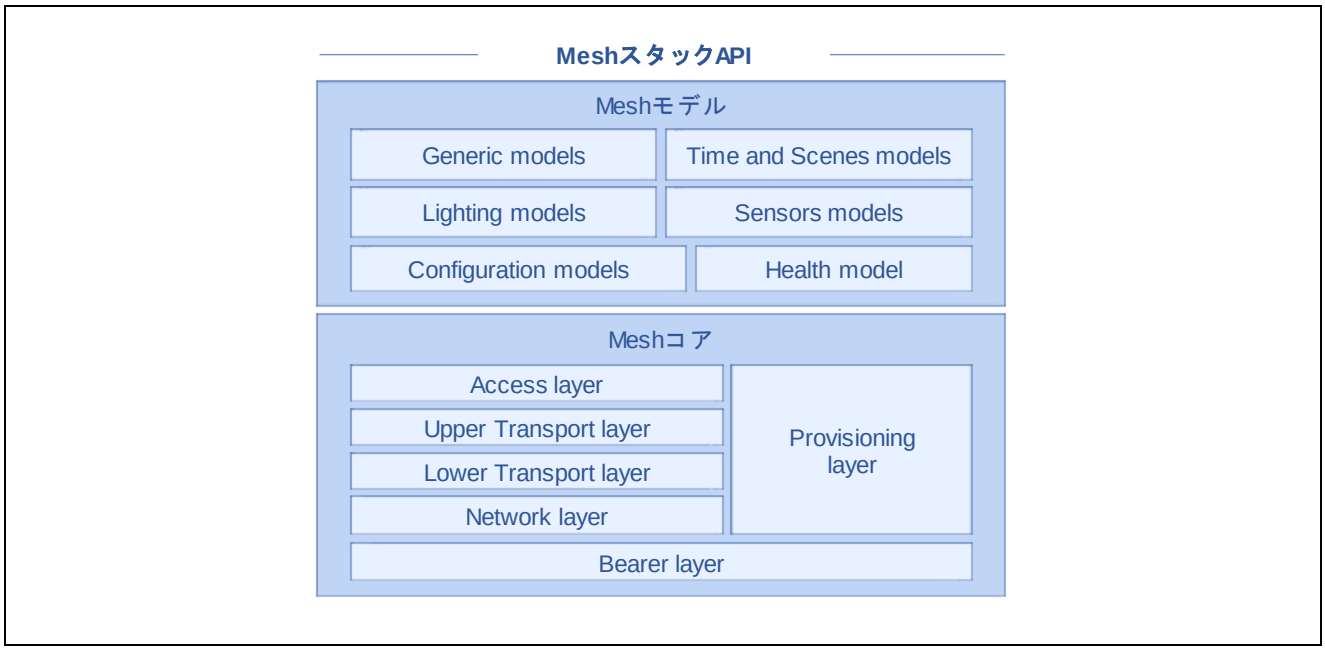

図 2-3 Bluetooth Mesh スタックの内部構成

Bluetooth Mesh スタックは次のブロックで構成されます。

- **Mesh コア**

  Mesh コアブロックは Mesh Profile 仕様で定義された各レイヤーに対応するモジュールで構成され、プロビジョニングと Mesh ネットワーク動作を実行するための機能をアプリケーションに提供します。Mesh Profile 仕様は、Bluetooth SIG の Specifications List にて Mesh Profile 仕様書を参照してください。

- **Mesh モデル**

  Mesh モデルブロックは Mesh Model 仕様で定義された各モデルに対応するモジュールで構成され、Mesh ネットワークでの基本動作を定義した Mesh モデルをサポートするための機能をアプリケーションに提供します。Mesh Model 仕様は、Bluetooth SIG の Specifications List にて Mesh Model 仕様書を参照してください。

Mesh スタック API の仕様は、Mesh FIT モジュールに同梱された Bluetooth Mesh スタック API マニュアル (blemesh_api.chm)を参照してください。

Mesh スタックは Bluetooth Mesh Networking 仕様が定義するプロトコルを実現するためのモジュールで構成されます。Mesh スタック API は各モジュールに対応した下記の関数プレフィックスを持ちます。

Mesh アプリケーションはアプリケーションのシナリオに応じて、Mesh スタック API を実行する必要があります。

表 2-6 Mesh スタック関数

モジュール		関数プレフィックス
Meshモデル		
	Generic OnOff	MS_generic_onoff_*()
	Generic Level	MS_generic_level_*()
	Generic Default Transition Time	MS_generic_default_transition_time_*(),
	Generic Power OnOff	MS_generic_power_onoff_*()
	Generic Power Level	MS_generic_power_level_*()
	Generic Battery	MS_generic_battery_*()
	Generic Location	MS_generic_location_*()
	Generic Property	MS_generic_property_*()
	Sensor	MS_sensor_*()
	Time	MS_time_*()
	Scene	MS_scene_*()
	Scheduler	MS_scheduler_*()
	Light Lightness	MS_light_lightness_*()
	Light CTL	MS_light_ctl_*()
	Light HSL	MS_light_hsl_*()
	Light xyL	MS_light_xyl_*()
	Light LC	MS_light_lc_*()
	Configuration	MS_config_*()
	Health	MS_health_*()
Meshコア		
	Access Layer	MS_access_*()
	Transport Layer	MS_trn_*()
	Lower Transport Layer	MS_ltrn_*()
	Network Layer	MS_net_*()
	Bearer Layer	MS_brr_*()
	Provisioning Layer	MS_prov_*()

2.4 Bluetooth ベアラー

Bluetooth ベアラーは、Bluetooth LE スタックのラッパー関数を Mesh スタックとアプリケーションに提供します。Bluetooth ベアラーのソースコードはパッケージに同梱されています。Bluetooth ベアラーAPI の仕様は、Mesh FIT モジュールに同梱された Bluetooth Mesh スタック API マニュアル (blemesh_api.chm)を参照してください。

Bluetooth LE スタックは、Bluetooth Low Energy 仕様に準拠した無線通信機能を上位レイヤーに提供します。ライブラリファイルはパッケージに同梱されています。R_BLE API の仕様は、BLE FIT モジュールに同梱された R_BLE API 仕様書(r_ble_api_spec.chm)を参照してください。

図 2-4 に Bluetooth ベアラーの内部構成を示します。メッセージ送受信のためのベアラー関数は、Mesh スタックが利用します。接続制御のためのベアラー関数は、Mesh アプリケーションが必要に応じて利用する必要があります。

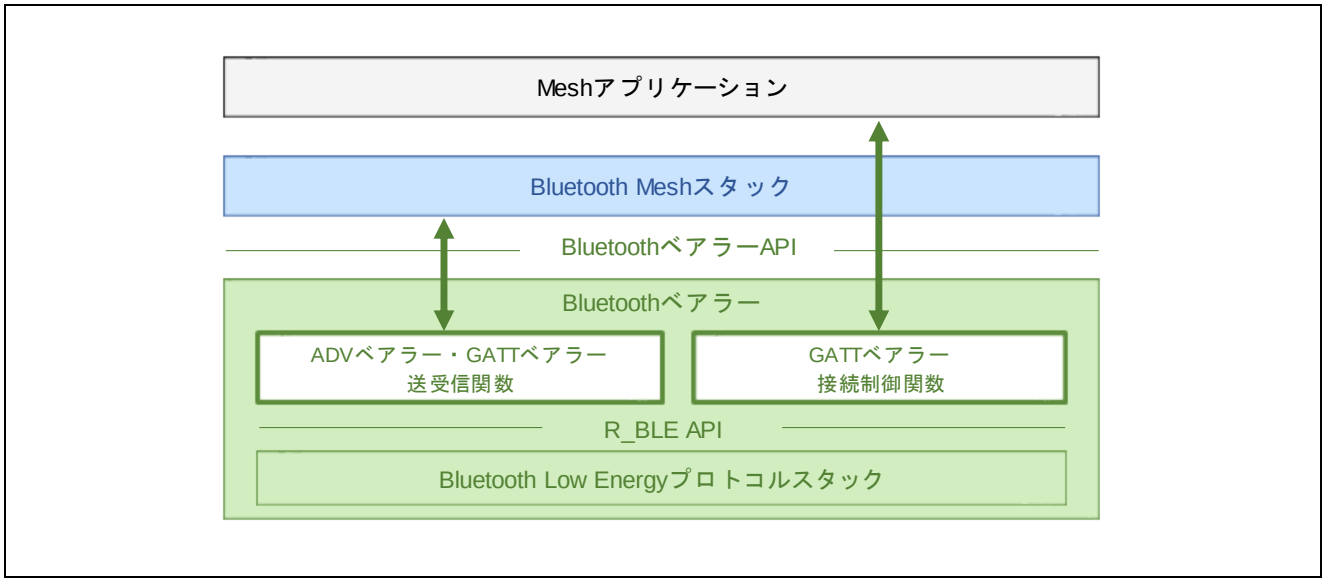

図 2-4 Bluetooth ベアラーの動作

2.4.1 メッセージ送受信のためのベアラー関数 (blebrr.c)

表 2-7 にメッセージ送受信のためのベアラー関数を示します。ベアラー関数はメッセージの送受信に加え、ADV ベアラーの動作状態を制御する機能を提供します。ベアラー関数は R_MS_BRR_Setup()によって Mesh スタックに登録されます。Mesh スタックはこれらの関数を使用して、プロビジョニング PDU および Mesh メッセージを送受信し、ADV ベアラー状態を制御します。

表 2-7 メッセージ送受信のためのベアラー関数

関数	処理
blebrr_adv_send()	メッセージの送信
登録不要 注	受信メッセージの処理
blebrr_adv_sleep()	ベアラーの休止
blebrr_adv_wakeup()	ベアラーの再開

注: 受信データ処理関数は R_MS_BRR_Setup()内でコールされる MS_brr_add_bearer()によって Mesh スタックが自動的に登録します。

2.4.2 接続制御のためのベアラー関数 (blebrr_pl.c, blebrr_gatt.c)

Mesh スタックは GATT ベアラーのための接続状態や GATT サービスは管理しません。このため GATT ベアラーを利用する場合、Mesh アプリケーションは接続管理のためのベアラー関数を直接実行して、接続状態や GATT サービスを管理する必要があります。

表 2-8 に接続制御のためのベアラー関数を示します。これらの関数は接続の確立と切断に加え、サービス探索、Notification の許可といった機能を提供します。

表 2-8 接続制御のためのベアラー関数

関数	処理	GATT Server (Peripheral)	GATT Client (Central)
R_MS_BRR_Register_GattIfaceCallback()	GATT コールバックの登録	使用	使用
R_MS_BRR_Set_GattMode()	GATT ベアラーモードの設定 注 1	使用	使用
R_MS_BRR_Get_GattMode()	GATT ベアラーモードの取得 注 1	使用	使用
R_MS_BRR_Disconnect()	接続の切断	使用	使用
R_MS_BRR_Set_ScanRspData()	Scan Response データの設定	使用	不使用
R_MS_BRR_Scan_GattBearer()	接続可能なデバイスのスキャン	不使用	使用
R_MS_BRR_Create_Connection()	接続の要求	不使用	使用
R_MS_BRR_Cancel_CreateConnection()	接続要求のキャンセル	不使用	使用
R_MS_BRR_Discover_Service()	サービスディスカバリの実行	不使用	使用
R_MS_BRR_Config_Notification()	Mesh GATT サービスの Notification 許可 注 2	不使用	使用
R_MS_BRR_Config_ServChanged()	GATT Service Changed の Indication 許可	不使用	使用

注 1: GATT ベアラーモードとはプロビジョニングモードまたはプロキシモード

注 2: GATT Server は Notification が許可された時点で MTU サイズを Mesh スタックに設定します。GATT Client から MTU サイズを変更する場合、GATT Client は Notification を有効化する前に MTU Exchange プロシージャを実行してください。

MTU の変更については、「RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド」(R01AN5504)の 8.4 節を参照してください。

2.4.3 Mesh GATT サービス (gatt_db.c)

GATT ベアラーによる Mesh メッセージの送受信には Mesh GATT サービスが使用されます。表 2-9 に Mesh GATT サービスの構成を示します。Mesh プロビジョニングサービスは GATT ベアラーによるプロビジョニング時に使用され、Mesh プロキシサービスはプロビジョニング完了後のプロキシ接続に使用されます。Mesh GATT サービスのどちらのサービスを公開するかは `R_MS_BRR_Set_GattMode()`で切り替えます。

表 2-9 Mesh GATT サービスの構成

サービス (UUID)	キャラクタリスティック (UUID)	プロパティ	値
Mesh Provisioning Service (0x1827)	Mesh Provisioning Data In Characteristic (0x2ADB)	Write Without Response	プロビジョニングクライアントからプロビジョニングサーバーへのプロビジョニングPDU
	Mesh Provisioning Data Out Characteristic (0x2ADC)	Notify	プロビジョニングサーバーからプロビジョニングクライアントへのプロビジョニングPDU
Mesh Proxy Service (0x1828)	Mesh Proxy Data In Characteristic (0x2ADD)	Write Without Response	プロキシクライアントからプロキシサーバーへのネットワークPDU、Meshビーコンまたはプロキシ設定を含むプロキシPDU
	Mesh Proxy Data Out Characteristic (0x2ADE)	Notify	プロキシサーバーからプロキシクライアントへのネットワークPDU、Meshビーコンまたはプロキシ設定を含むプロキシPDU

Mesh GATT サービスおよび他のサービスが定義された GATT データベースは Bluetooth ベアラーの gatt_db.c に実装されています。

2.4.4 ADV ベアラー動作

Mesh アプリケーションが R_MS_BRR_Setup()を実行すると、Bluetooth ベアラーは ADV ベアラーのメッセージ送受信関数を Mesh スタックに登録し、Scan を開始します。

Bluetooth LE スタックが受信した Advertising パケットが Mesh スタックに通知されます。また Mesh スタックがメッセージ送信関数を実行することで、Bluetooth LE スタックは Advertising パケットを送信します。

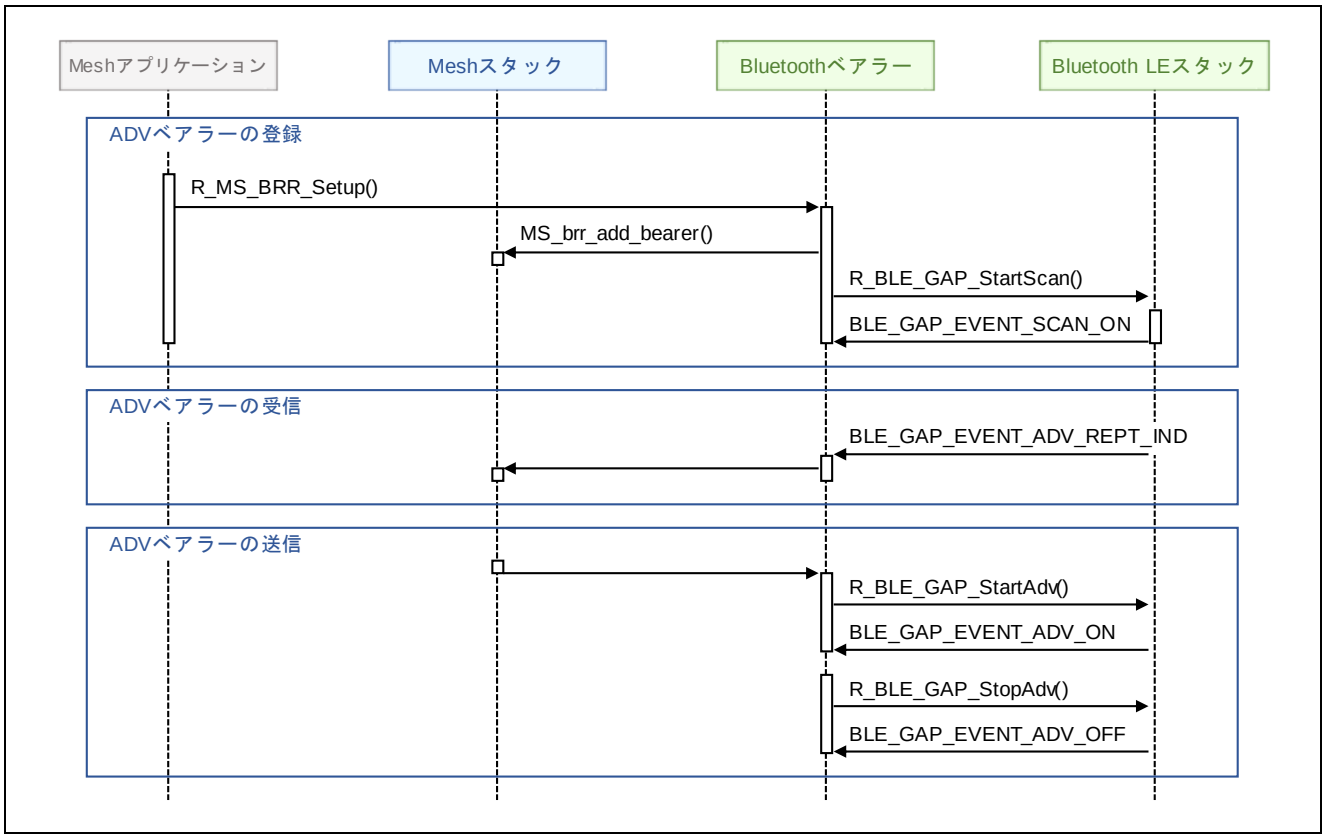

図 2-5 ADV ベアラー動作

2.4.5 GATT ベアラー動作

接続の確立後、Notification の有効化が完了すると、Bluetooth ベアラーは GATT ベアラーのメッセージ送受信関数を Mesh スタックに登録します。

GATT Server として動作する場合、Mesh スタックがメッセージ送信関数を実行すると、Bluetooth LE スタックは Notification でメッセージを送信します。また Write Without Response で受信したメッセージが Mesh スタックに通知されます。

GATT Client として動作する場合、Mesh スタックがメッセージ送信関数を実行すると、Bluetooth LE スタックは Write Without Response でメッセージを送信します。また Notification で受信したメッセージが Mesh スタックに通知されます。

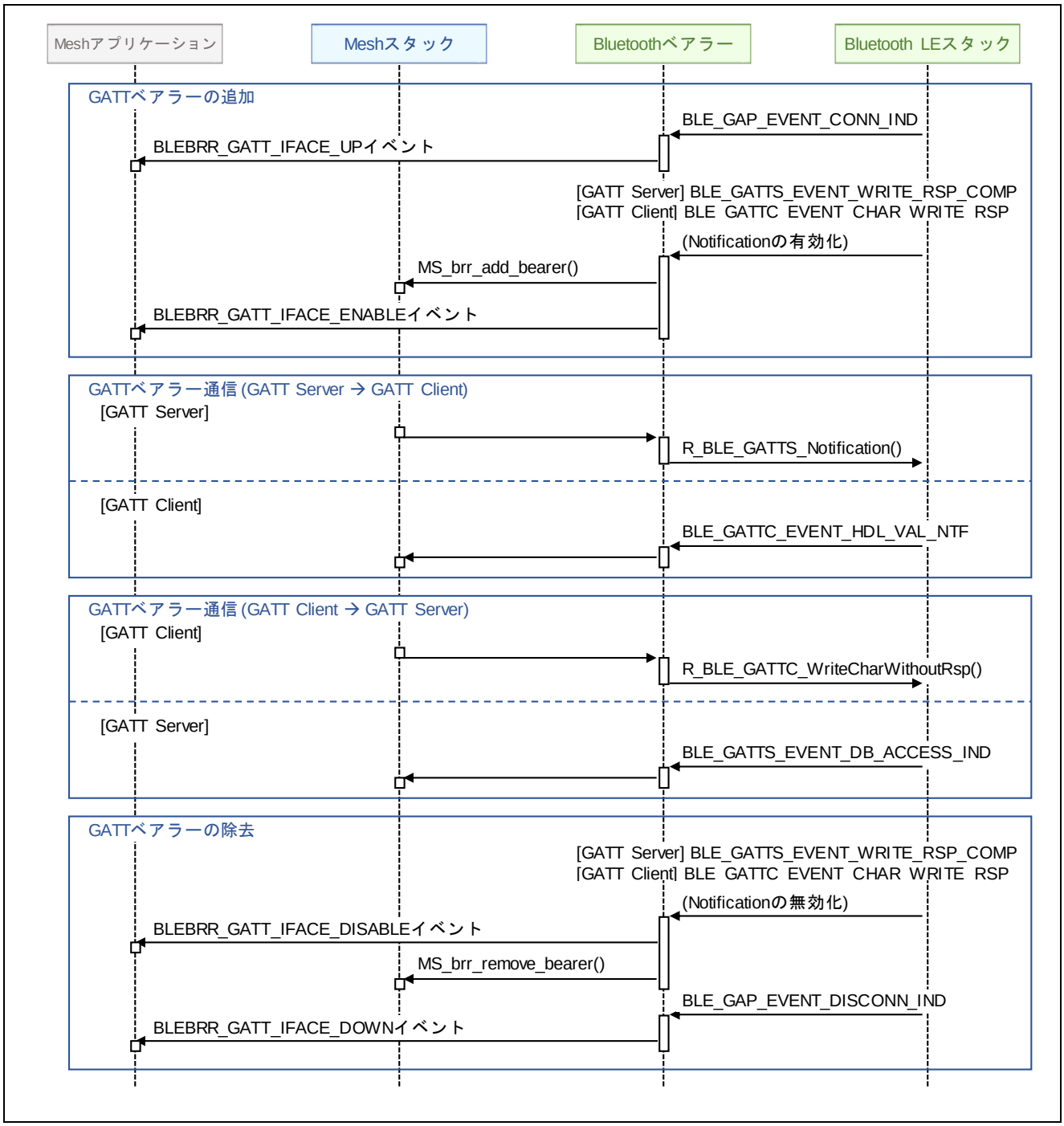

図 2-6 GATT ベアラー動作

2.5 MCU 周辺機能

Bluetooth Mesh スタックパッケージは表 2-10 に示す RX23W 周辺機能を利用します。

表 2-10 使用する RX23W 周辺機能

RX23W 周辺機能	周辺機能ドライバ	周辺機能を利用するソフトウェア
汎用 I/O ポート (GPIO) – P15、PB0 および PC0: Target Board 使用時 – P30、P31、P42 および P43: RSSK 使用時	GPIO FIT モジュール (R01AN1721)	Mesh サンプルプログラム
シリアルコミュニケーションインタフェース(SCI) – SCI8	SCI FIT モジュール (R01AN1815)	Mesh サンプルプログラム
コンペアマッチタイマ (CMT) – CMT2 および CMT3: Bluetooth LE スタックが占有 – CMT0 または CMT1: Mesh サンプルプログラムと Bluetooth ベアラーが共有	CMT FIT モジュール (R01AN1856)	Mesh サンプルプログラム Mesh スタック Bluetooth ベアラー Bluetooth LE スタック
消費電力低減機能 (LPC)	LPC FIT モジュール (R01AN2769)	Mesh サンプルプログラム Bluetooth LE スタック
E2 データフラッシュメモリ (FLASH) – ブロック 1～5	FLASH FIT モジュール(R01AN2184)	Mesh スタック
8 ビットタイマ (TMR) – TMR2 および TMR3	Mesh FIT モジュールに同梱(R01AN4930)	Mesh スタック

- **汎用 I/O ポート (GPIO)**

  Mesh サンプルプログラムは次の処理に GPIO を使用するために GPIO FIT モジュールを使用します。

  - 開発ボード上の LED 制御
  - 開発ボード上のスイッチの押下検知

- **シリアルコミュニケーションインタフェース (SCI)**

  Mesh サンプルプログラムは UART を経由したコンソールの入出力に SCI FIT モジュールを使用します。

- **コンペアマッチタイマ (CMT)**

  BLE FIT モジュールには CMT の 1 チャネルを複数の処理で共有できるソフトウェアタイマ (R_BLE_TIMER)が同梱されています。R_BLE_TIMER は CMT FIT モジュールを使用して CMT の 1 チャネルを占有します。また BLE FIT モジュールの Bluetooth LE スタックは CMT2 と CMT3 を占有します。

  Bluetooth ベアラーは下記の処理に R_BLE_TIMER を使用します。

  - ADV ベアラーによる Advertising の送信制御

  Mesh サンプルプログラムは下記の処理に R_BLE_TIMER を使用します。

  - 開発ボード上の LED 点滅
  - 開発ボード上のスイッチのチャタリング回避
  - Config Node Reset 受信時の MCU リセット遅延
  - IV Update プロシージャの終了

- **消費電力低減機能**

Mesh サンプルプログラムは MCU の消費電力低減機能を有効化するために LPC FIT モジュールを使用します。

- **フラッシュメモリ (FLASH)**

データフラッシュメモリを使用するためのデータフラッシュドライバは Mesh FIT モジュールの mesh_dataflash.c に実装されています。本ドライバは FLASH FIT モジュールを使用してデータフラッシュにアクセスします。ドライバが使用するフラッシュメモリの領域は MESH_CFG_DATA_FLASH_BLOCK_ID マクロと MESH_CFG_DATA_FLASH_BLOCK_NUM マクロにより設定することができます。

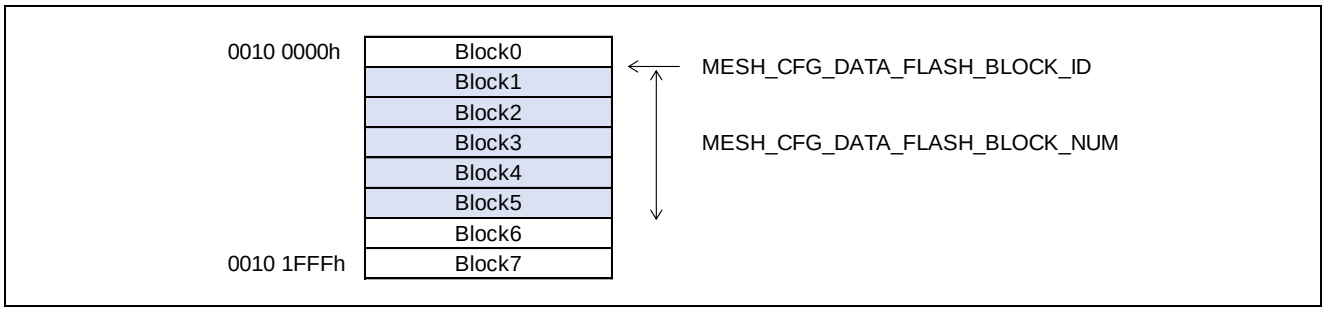

図 2-7 データフラッシュメモリの使用領域

Mesh スタックはデータフラッシュメモリに下記の情報を保持します。

- プロビジョニングデータ
  - ユニキャストアドレス
  - ネットワークキー
- コンフィグレーション情報
  - モデル構成
  - モデル設定
- IV インデックスとアップデート状態
- シーケンス番号

シーケンス番号を除く上記の情報はほとんど変更されることはありません。シーケンス番号はメッセージを送信する毎にインクリメントされます。インクリメント毎にシーケンス番号を書き込むと、フラッシュメモリは短期間で書き込み回数の限界に到達します。

そこでフラッシュメモリの書き込み頻度を低減するため、シーケンス番号はブロックとして扱われ、シーケンス番号が次のブロックに到達したときのみフラッシュに書き込まれます。シーケンス番号の長さであるブロックサイズは MESH_CFG_NET_SEQ_NUMBER_BLOCK_SIZE マクロにより設定でき、そのデフォルトサイズは 2048 です。

- **8 ビットタイマ (TMR)**

8 ビットタイマを使用するシステムタイムドライバは mesh_systemtime.c に実装されています。ドライバは 8 ビットタイマの 2 チャネルを使用して、1msec 単位 32 ビット幅のシステムタイムを生成します。

Mesh スタックはシステムタイムを使用して IV Update プロシージャの最小継続時間である 96 時間を監視します。

2.6 Mesh サンプルプログラムの設定

2.6.1 基本動作設定

Mesh サンプルプログラムには基本動作を設定するためのコンパイルスイッチがあります。コンパイルスイッチは mesh_appl.h に実装されています。

mesh_appl.h

```
/**
 * Monitor SEQ of Incoming and Outgoing message.
 * If SEQ is greater than or equal to threshold, initiating IV Update procedure.
 */
#define IV_UPDATE_INITIATION_EN         (1)

/**
 * Low Power Feature
 */
#define LOW_POWER_FEATURE_EN            (0)

/**
 * Monitoring Mesh Layer
 * Combination of the following macros can be set like "(MS_MONITOR_ACCESS_PDU |
MS_MONITOR_NET_PDU)".
 *    - MS_MONITOR_ACCESS_PDU
 *    - MS_MONITOR_TRANS_PDU
 *    - MS_MONITOR_LTRANS_PDU
 *    - MS_MONITOR_NET_PDU
 *    - MS_MONITOR_GENERIC_LOG
 * To specify all layer, MS_MONITOR_ALL macro can be set.
 * To disable monitoring, set MS_MONITOR_NONE.
 */
#define CONSOLE_MONITOR_CFG             (MS_MONITOR_NONE)

/** Logging Console using SCI (Serial Communication Interface) */
#define CONSOLE_OUT_EN                  (1)

/** ANSI escape sequence - CSI (Control Sequence Introducer) */
#define ANSI_CSI_EN                     (1)

/** SCI String Reception */
#define SCI_RCV_STRING_EN               (1)
#define SCI_RCV_STRING_BUFFER_LEN       (0x100)

/** Monitoring CPU Usage */
#define CPU_USAGE_EN                    (0)
```

- **IV アップデート開始処理の有効化**

IV_UPDATE_INITIATION_EN マクロを(1)に設定することで、IV Update 開始処理が有効となります。本処理は送受信するメッセージのシーケンス番号を監視し、シーケンス番号が閾値を超えると IV Update プロシージャを実行します。これにより自ノードまたは他ノードのシーケンス番号の枯渇を防止します。

設定マクロ	設定値	内容
IV_UPDATE_INITIATION_EN	0	IV Update 開始処理を無効化
	1	IV Update 開始処理を有効化

- **ローパワー機能の有効化**

LOW_POWER_FEATURE_EN マクロを(1)に設定することで、ローパワー機能が有効となります。プロビジョニングの完了後、フレンドノードとフレンドシップを確立し、ローパワーノードとして動作します。

設定マクロ	設定値	内容
LOW_POWER_FEATURE_EN	0	ローパワーノードへの遷移を無効化
	1	ローパワーノードへの遷移を有効化

- **Mesh モニタ設定**

CONSOLE_MONITOR_CFG マクロにモニタ設定マクロを設定することで、Mesh モニタ機能が有効となります。Mesh スタックの各レイヤーが送受信したメッセージとビーコンのログが出力されるため、Mesh アプリケーションの開発において Mesh ネットワークの通信を解析できます。

設定マクロ	設定値	内容
CONSOLE_MONITOR_LOG	MS_MONITOR_ACCESS_PDU	Access PDU
	MS_MONITOR_TRANS_PDU	Transport PDU
	MS_MONITOR_LTRANS_PDU	Lower Transport PDU
	MS_MONITOR_NET_PDU	Network PDU と Secure Network Beacon
	MS_MONITOR_GENERIC_LOG	Mesh スタックの内部イベント

- **コンソール出力設定**

CONSOLE_OUT_EN マクロを(1)に設定することで、コンソールへのログ出力が有効となります。Mesh サンプルプログラムの実行する API や Mesh スタックの返却するイベントをトレースできます。

設定マクロ	設定値	内容
CONSOLE_OUT_EN	0	コンソールログ出力を無効化
	1	コンソールログ出力を有効化

- **コンソールへの ANSI CSI 出力設定**

ANSI_CSI_EN マクロを(1)に設定することで、コンソールへの ANSI CSI (Control Sequence Introducer)が有効となります。Mesh サンプルプログラムは CSI によるログの色付けを行っています。使用するシリアルターミナルソフトが ANSI CSI に対応していない場合、ANSI_CSI_EN マクロは(0)に設定してください。

設定マクロ	設定値	内容
ANSI_CSI_EN	0	コンソールログへの ANSI CSI 出力を無効化
	1	コンソールログへの ANSI CSI 出力を有効化

- **コンソール文字列受信設定**

SCI_RCV_STRING_EN マクロを(1)に設定することで、コンソールからの文字列受信が有効となります。受信した文字列はコールバック関数で通知されます。

設定マクロ	設定値	内容
SCI_RCV_STRING_EN	0	コンソールからの文字列受信を無効化
	1	コンソールからの文字列受信を有効化
SCI_RCV_STRING_BUFFER_LEN	0x0001～0xFFFF	文字列受信バッファサイズ

- **CPU 使用率測定の有効化**

CPU_USAGE_EN マクロを(1)に設定することで、CPU 使用率測定が有効となります。Mesh サンプルプログラムはメインループにおいて CPU が RUN 状態と SLEEP 状態のそれぞれの時間を測定し、CPU 使用率ログをコンソールに出力します。

設定マクロ	設定値	内容
CPU_USAGE_EN	0	CPU 使用率測定を無効化
	1	CPU 使用率測定を有効化

2.6.2 Provisioning 動作設定

Mesh サンプルプログラムには Provisioning 動作を設定するための設定マクロがあります。設定マクロは mesh_core.c に実装されています。

mesh_core.c

```
/** Public Key OOB Flag (0:unavailable, 1:available) */
#define CORE_PROV_PUBKEY_OOBINFO            (0)
/** Static OOB Flag (0:unavailable, 1:available) */
#define CORE_PROV_STATIC_OOBINFO            (0)
/** Output OOB Actions Supported (bit0-bit4 specified with PROV_MASK_OOOB_ACTION_*) */
#define CORE_PROV_OUTPUT_OOB_ACTIONS        (0)
/** Output OOB Maximum size supported (0:not supported, 1-8 digits supported) */
#define CORE_PROV_OUTPUT_OOB_SIZE           (0)
/** Input OOB Actions supported (bit0-bit4 specified with PROV_MASK_IOOB_ACTION_*) */
#define CORE_PROV_INPUT_OOB_ACTIONS         (0)
/** Input OOB Maximum size supported (0:not supported, 1-8 digits supported) */
#define CORE_PROV_INPUT_OOB_SIZE            (0)

/** OOB Information (bit0-bit6 and bit11-bit15 specified with PROV_OOB_TYPE_*) */
#define CORE_PROV_BEACON_OOB_INFO           (0)
/**
 *  Encoded URI Information (payload length is up to 29 octets)
 */
#define CORE_PROV_BEACON_URI_INFO           {.payload = "\x17//www.example.com", .length = 18}
```

- **OOB Public Key**

CORE_PROV_PUBKEY_OOBINFO マクロを(1)に設定することで OOB Public Key が有効となります。OOB で配布する Public Key はコンソールに表示されます。

設定マクロ	設定値	内容
CORE_PROV_PUBKEY_OOBINFO	0	OOB Public Key を無効化
	1	OOB Public Key を有効化

- **Static OOB Authentication**

CORE_PROV_STATIC_OOBINFO マクロを(1)に設定することで Static OOB Authentication が有効となります。OOB で配布する AuthValue はプロビジョニングの初期化時にコンソールに表示されます。

設定マクロ	設定値	内容
CORE_PROV_STATIC_OOBINFO	0	Static OOB Authentication を無効化
	1	Static OOB Authentication を有効化

- **Output OOB Authentication**

CORE_PROV_OUTPUT_OOB_SIZE マクロに AuthValue の桁数(1～8)を設定することで Output OOB Authentication が有効となります。OOB で配布する AuthValue は Provisioning 中にコンソールに表示されます。

CORE_PROV_OUTPUT_OOB_ACTIONS マクロには AuthValue を出力する OOB アクションを設定します。複数の Output OOB アクションを設定することができます。例えば Blink 動作と Beep 動作に対応する場合、(PROV_MASK_OOOB_ACTION_BLINK | PROV_MASK_OOOB_ACTION_BEEP) を設定します。

設定マクロ	設定値	内容
CORE_PROV_OUTPUT_OOB_SIZE	0	Output OOB Authentication を無効化
	1～8	Output OOB Authentication を有効化 AuthValue の対応桁数を設定
CORE_PROV_OUTPUT_OOB_ACTIONS	PROV_MASK_OOOB_ACTION_BLINK	Blink 動作 (数値)
	PROV_MASK_OOOB_ACTION_BEEP	Beep 動作 (数値)
	PROV_MASK_OOOB_ACTION_VIBRATE	Vibrate 動作 (数値)
	PROV_MASK_OOOB_ACTION_NUMERIC	Numeric 表示 (数値)
	PROV_MASK_OOOB_ACTION_ALPHANUMERIC	Alphanumeric 表示 (英数字)

- **Input OOB Authentication**

CORE_PROV_INPUT_OOB_SIZE マクロに AuthValue の桁数(1～8)を設定することで Input OOB Authentication が有効となります。OOB で配布された AuthValue はコンソールに入力してください。

CORE_PROV_INPUT_OOB_ACTIONS マクロには AuthValue を入力する OOB アクションを設定します。複数の Input OOB アクションを設定することができます。例えば Push 動作と Twist 動作に対応する場合、(PROV_MASK_IOOB_ACTION_PUSH | PROV_MASK_IOOB_ACTION_TWIST) を設定します。

設定マクロ	設定値	内容
CORE_PROV_INPUT_OOB_SIZE	0	Input OOB Authentication を無効化
	1～8	Input OOB Authentication を有効化 AuthValue の桁数を設定
CORE_PROV_INPUT_OOB_ACTIONS	PROV_MASK_IOOB_ACTION_PUSH	Push 動作 (数値)
	PROV_MASK_IOOB_ACTION_TWIST	Twist 動作 (数値)
	PROV_MASK_IOOB_ACTION_NUMERIC	Numeric 入力 (数値)
	PROV_MASK_IOOB_ACTION_ALPHANUMERIC	Alphanumeric 入力 (英数字)

- **OOB Information**

CORE_PROV_STATIC_OOBINFO マクロには OOB Information を設定します。設定した OOB Information は Unprovisioned Device ビーコンで配布されます。複数の OOB Information を設定することができます。例えば URI とバーコードを設定する場合、(PROV_OOB_TYPE_URI | PROV_OOB_TYPE_BARCODE) を設定します。

設定マクロ	設定値	内容
CORE_PROV_BEACON_OOB_INFO	PROV_OOB_TYPE_OTHER	その他
	PROV_OOB_TYPE_URI	URI
	PROV_OOB_TYPE_2DMRC	機械判別可能な 2D コード
	PROV_OOB_TYPE_BARCODE	バーコード
	PROV_OOB_TYPE_NFC	Near Field Communication (NFC)
	PROV_OOB_TYPE_NUMBER	数値
	PROV_OOB_TYPE_STRING	文字列
	PROV_OOB_TYPE_ONBOX	ボックス上部
	PROV_OOB_TYPE_INSIDEBOX	ボックス内部

	PROV_OOB_TYPE_ONPIECEOFPAPER	紙面上
	PROV_OOB_TYPE_INSIDEMANUAL	マニュアル
	PROV_OOB_TYPE_ONDEVICE	デバイス内部

- **Encoded URI Information**

上述の CORE_PROV_BEACON_OOB_INFO マクロに PROV_OOB_TYPE_URI を設定した場合、CORE_PROV_BEACON_URI_INFO マクロに Encoded URI Information を設定する必要があります。設定した Encoded URI Information は AD タイプの<<URI>>で配布され、Encoded URI Information の Hash 値は Unprovisioned Device ビーコンで配布されます。

設定マクロ	設定値	内容
CORE_PROV_BEACON_URI_INFO	最大 29 octets	Encoded URI URI スキームは Bluetooth SIG の Assigned Numbers で定義された"URI Scheme Name String Mapping"でエンコード

2.7 Bluetooth ベアラーの設定

Mesh FIT モジュールに含まれる Bluetooth ベアラーの設定について示します。

blebrr.h

```
/** Enable GATT Bearer Client Role */
/*  ROM/RAM used can be reduced by disabling GATT Client functionalities */
#define BLEBRR_GATT_CLIENT                      (1)

/** Specify Device Address Type
  * Either Public Address or Static Random Address can be set by the macro below.
  *  - BLE_GAP_ADDR_PUBLIC
  *  - BLE_GAP_ADDR_RAND
  * Device Address is obtained from BLE Protocol Stack via Vendor Specific API.
  */
#define BLEBRR_VS_ADDR_TYPE                     (BLE_GAP_ADDR_RAND)
```

- **GATT クライアントの有効化**

BLEBRR_GATT_CLIENT マクロを(1)に設定することで、GATT ベアラーのクライアント機能が有効となります。

設定マクロ	設定値	内容
BLEBRR_GATT_CLIENT	0	GATT ベアラーの GATT クライアント動作を無効化
	1	GATT ベアラーの GATT クライアント動作を有効化

- **デバイスアドレスタイプ設定**

BLEBRR_VS_ADDR_TYPE マクロにデバイスアドレスタイプマクロを設定することで、Bluetooth ベアラーが使用するデバイスアドレスタイプを指定できます。

設定マクロ	設定値	内容
BLEBRR_VS_ADDR_TYPE	BLE_GAP_ADDR_PUBLIC	Public Device Address
	BLE_GAP_ADDR_RAND	Random Device Address

blebrr.c

```
#define BLEBRR_QUEUE_SIZE                   64
#define BLEBRR_ADVREPEAT_RAND_DELAY         10
```

- **ADV ベアラー送信設定**

ADV ベアラーの送信設定として下記の設定マクロが定義されます。

設定マクロ	設定値	内容
BLEBRR_QUEUE_SIZE	4 以上	送信キューサイズ
BLEBRR_ADVREPEAT_RAND_DELAY	1 以上	送信ランダムディレイ幅 (1msec 単位)

blebrr_pl.c

```
#define BLEBRR_CON_ADVINTMIN                0xA0
#define BLEBRR_CON_ADVINTMAX                0xA0
#define BLEBRR_CON_ADVTYPE                  BLE_GAP_LEGACY_PROP_ADV_IND
#define BLEBRR_CON_ADVCHMAP                 BLE_GAP_ADV_CH_ALL
#define BLEBRR_CON_ADVFILTERPOLICY          BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY

#define BLEBRR_SCAN_INTERVAL                0x0060
#define BLEBRR_SCAN_WINDOW                  0x0060

#define BLEBRR_INIT_SCANINTERVAL            0x0060
#define BLEBRR_INIT_SCANWINDOW              0x0060

#define BLEBRR_CONN_INTERVAL_MIN            0x0040
#define BLEBRR_CONN_INTERVAL_MAX            0x0040
#define BLEBRR_CONN_LATENCY                 0x0000
#define BLEBRR_CONN_SUPERVISION_TO          0x03BB
```

- **GATT ベアラー接続のためのコネクタブルアドバタイジング設定**

GATT ベアラー接続のためのコネクタブルアドバタイジング設定として下記の設定マクロが定義されます。

設定マクロ	設定値	内容
BLEBRR_CON_ADVINTMIN	0x20～0xFFFFFF	最小アドバタイジングインターバル (0.625msec 単位)
BLEBRR_CON_ADVINTMAX	0x20～0xFFFFFF	最大アドバタイジングインターバル (0.625msec 単位)
BLEBRR_CON_ADVTYPE	BLE_GAP_LEGACY_PROP_ADV_IND	アドバタイジングタイプ : Connectable and Scannable Undirected Legacy Advertising
BLEBRR_CON_ADVCHMAP	BLE_GAP_ADV_CH_37	アドバタイジングチャネル 37ch
	BLE_GAP_ADV_CH_38	アドバタイジングチャネル 38ch
	BLE_GAP_ADV_CH_39	アドバタイジングチャネル 39ch
	BLE_GAP_ADV_CH_ALL	全アドバタイジングチャネル
BLEBRR_CON_ADVFILTERPOLICY	BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY	アドバタイジングフィルタポリシー : Process Scan Requests and Connection Requests from All Devices

- **ADV ベアラーのスキャン設定**

ADV ベアラーのスキャン設定として下記の設定マクロが定義されます。

注: Advertising パケットの受信失敗を低減するため、インターバルとウィンドサイズは同じ値を設定してください。

設定マクロ	設定値	内容
BLEBRR_SCAN_INTERVAL	0x0004～0xFFFF	スキャンインターバル (0.625msec 単位)
BLEBRR_SCAN_WINDOW	0x0004～0xFFFF	スキャンウィンドウサイズ (0.625msec 単位)

- **GATT クライアント向けの GATT ベアラー接続設定**

GATT クライアント向けの GATT ベアラー接続設定として下記の設定マクロが定義されます。

設定マクロ	設定値	内容
BLEBRR_INIT_SCANINTERVAL	0x0004～0xFFFF	イニシエーティングのためのスキャンインターバル (0.625msec 単位)
BLEBRR_INIT_SCANWINDOW	0x0004～0xFFFF	イニシエーティングのためのスキャンウィンドウサイズ(0.625msec 単位)
BLEBRR_CONN_INTERVAL_MIN	0x0006～0x0C80	最小接続インターバル (1.25msec 単位)
BLEBRR_CONN_INTERVAL_MAX	0x0006～0x0C80	最大接続インターバル (1.25msec 単位)
BLEBRR_CONN_LATENCY	0x0000～0x01F3	ペリフェラルレイテンシー (コネクションイベント数)
BLEBRR_CONN_SUPERVISION_TO	0x000A～0x0C80	スーパービジョンタイムアウト (10msec 単位)

2.8 Mesh ドライバの設定

Mesh FIT モジュールに含まれる Mesh ドライバの設定について示します。

mesh_dataflash.h

```
#define DATAFLASH_EN    (1)
```

- **データフラッシュドライバの有効化**

DATAFLASH_EN マクロを(1)に設定することで、データフラッシュドライバが有効となり、ドライバ関数が Mesh スタックに登録されます。

設定マクロ	設定値	内容
DATAFLASH_EN	0	データフラッシュアクセスを無効化
	1	データフラッシュアクセスを有効化

mesh_systemtime.h

```
#define SYSTEMTIME_EN                   (1)
#define SYSTEMTIME_STRING_EN            (1)
```

- **システムタイムドライバの有効化**

SYSTEMTIME_EN マクロを(1)に設定することで、32 ビットシステムタイムを生成するシステムタイムドライバが有効となり、ドライバ関数が Mesh スタックに登録されます。

設定マクロ	設定値	内容
SYSTEMTIME_EN	0	システムタイム生成を無効化
	1	システムタイム生成を有効化

- **システムタイム文字列設定**

SYSTEMTIME_STRING_EN マクロを(1)に設定することで、32 ビットシステムタイムを文字列で生成する関数が有効となります。

設定マクロ	設定値	内容
SYSTEMTIME_STRING_EN	0	システムタイムの文字列出力を無効化
	1	システムタイムの文字列出力を有効化

3. アプリケーション開発

本章は Mesh サンプルプログラムの実装を参照しながら、Bluetooth Mesh スタックを利用したアプリケーションの開発方法を示します。図 3-1 に Mesh サンプルプログラムの動作フローを示します。

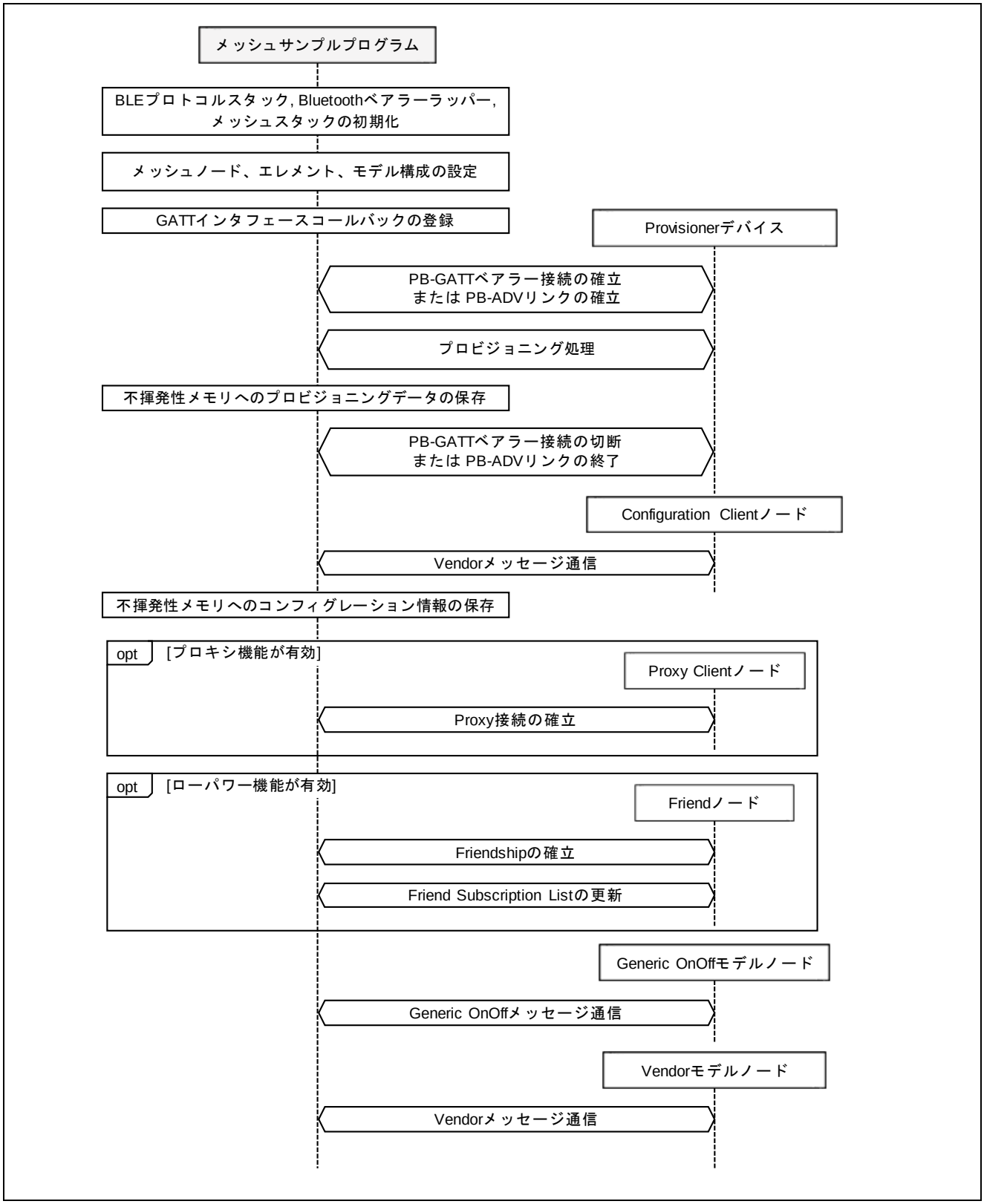

図 3-1 Mesh サンプルプログラムの動作フロー

3.1 メインルーチン

Mesh スタックは Bluetooth LE スタック上で動作します。このため、アプリケーションは Mesh スタックを初期化する前に、Bluetooth LE スタックと Bluetooth ベアラーを初期化する必要があります。

Bluetooth LE スタックの処理は、Bluetooth LE スタックスケジューラによって実行されます。このため、アプリケーションは Bluetooth LE スタックの初期化後、スケジューラ関数である R_BLE_Execute()をメインループで繰り返し実行する必要があります。

Mesh サンプルプログラムのメインルーチンを以下に示します。

- **メインルーチン (main.c)**

R_BLE_Open()と R_MS_BRR_Init()で Bluetooth LE スタックと Bluetooth ベアラーを初期化してください。必要に応じてその他の FIT モジュールも初期化してください。メインループでは Bluetooth LE スタックのスケジューラである R_BLE_Execute()を反復実行してください。

Bluetooth ベアラーの初期化処理は Bluetooth LE スタックのスケジューラで実行され、初期化の完了はコールバック関数で通知されます。

```
int main(void)
{
    /* Initialize Bluetooth LE Protocol Stack */
    R_BLE_Open();

    /* Initialize the Low Power Control function */
    R_BLE_LPC_Init();

    /* Initialize Timer */
    R_BLE_TIMER_Init();

    /* Initialize underlying BLE Protocol Stack to use as a Mesh Bearer */
    R_MS_BRR_Init(blebrr_init_cb);

    /* main loop */
    while (1)
    {
        /* Process Event */
        R_BLE_Execute();
    }
}
```

- Bluetooth **ベアラーの初期化完了コールバック (main.c)**

Bluetooth ベアラーの初期化完了通知を受け取るコールバック関数を実装してください。本コールバック関数で Mesh スタックが使用するリソースを初期化し、MS_init_ext()で Mesh スタックを初期化し、R_MS_BRR_Setup()で Bluetooth ベアラーを Mesh スタックに登録してください。これらの初期化後、Mesh アプリケーションを開始してください。

```
static void blebrr_init_cb(st_ble_dev_addr_t * own_addr)
{
    API_RESULT retval;
    MS_CONFIG config;

    /* Initialize Mesh Resources */
```

```
    mesh_section_init();
    mesh_mempool_init();
    mesh_storage_init();
    #if SYSTEMTIME_EN
    mesh_systemtime_init();
    #endif /* SYSTEMTIME_EN */

    /* Initialize Mesh Stack */
    MESH_MS_CONFIG(config);
    retval = MS_init_ext(&config);

    if (API_SUCCESS == retval)
    {
        /* Registers ADV Bearer with Mesh Stack and Start Scan */
        R_MS_BRR_Setup();

        /* Start Mesh Application */
        mesh_model_config(&gs_mesh_model_callbacks);
        mesh_core_setup();
    }
}
```

- **Mesh スタックの終了処理**

Mesh ネットワークでの通信が不要となった場合は MS_shutdown()で Mesh スタックを終了します。

Light LC Server Model を使用した場合、MS_light_lc_server_deinit()で Light LC Server Model を終了します。MS_health_server_deinit()で Health Server Model を終了後、MS_shutdown()で Mesh スタックを終了します。R_MS_BRR_Close()で Bluetooth Bearer が使用したリソースを解放します。

Bluetooth LE スタックが不要となった場合は R_BLE_Close()で終了します。

```
    /* Deinitialize Light LC Server Model, if it was initialized */
    MS_light_lc_server_deinit();

    /* Deinitialize Health Server Model */
    MS_health_server_deinit();

    /* Terminate Mesh Stack */
    MS_shutdown();

    /* Free the resources allocated by Bluetooth Bearer */
    R_MS_BRR_Close();

    /* Terminate Software Timer */
    R_BLE_TIMER_Terminate();

    /* Terminate Bluetooth LE Protocol Stack */
    R_BLE_Close();
```

3.2 ノード構成

アプリケーションは、エレメントやモデルといったノード構成を設定する必要があります。この構成は、何をどのように制御したいのかといった、アプリケーションが実行すべきシナリオによって異なります。

Mesh サンプルプログラムのノード構成の設定を以下に示します。

- **ノードとエレメント (mesh_model.c)**

MS_access_create_node()でノードを生成し、MS_access_register_element_ext()で任意の数のエレメントを登録してください。必要となるエレメントの数は、アプリケーションのシナリオによって異なります。プライマリエレメントのエレメントハンドルには MS_ACCESS_DEFAULT_ELEMENT_HANDLE を設定し、後続エレメントには MS_ACCESS_DEFAULT_ELEMENT_HANDLE に 1 ずつ加算した値を設定してください。例えば 3 つのエレメントを登録する場合、各エレメントハンドルは MS_ACCESS_DEFAULT_ELEMENT_HANDLE、(MS_ACCESS_DEFAULT_ELEMENT_HANDLE+1)、(MS_ACCESS_DEFAULT_ELEMENT_HANDLE+2)です。各エレメントハンドルは Mesh モデルの追加に使用します。

```
API_RESULT mesh_model_config(const mesh_model_callbacks_t * callbacks)
{
    API_RESULT retval;
    MS_ACCESS_NODE_ID      node_id;
    MS_ACCESS_ELEMENT_DESC element_desc;

    /* Create Node */
    retval = MS_access_create_node(&node_id);

    /* Register Element */
    if (API_SUCCESS == retval)
    {
        gs_element_handle = MS_ACCESS_DEFAULT_ELEMENT_HANDLE;
        element_desc.loc = ELEMENT_DESC_LOCATION;
        retval = MS_access_register_element_ext(node_id, &element_desc, gs_element_handle);
    }

    return retval;
}
```

3.3 プロビジョニング

3.3.1 プロビジョニングサーバー

ネットワークに参加して他のノードと通信するため、アプリケーションはプロビジョニングサーバーとしてプロビジョニングを実行し、プロビジョニングクライアントからプロビジョニングデータを受け取る必要があります。

Mesh サンプルプログラムはプロビジョニングサーバーとして動作します。プロビジョニングサーバーの処理を以下に示します。

- **プロビジョニング特性とプロビジョニングコールバック関数の登録 (mesh_core.c)**

認証方法などのプロビジョニング特性とプロビジョニングコールバック関数を MS_prov_register()で Mesh スタックに登録してください。

MS_prov_register()の使用例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_prov_config()の実装を参照してください。

- **プロビジョニングの開始 (mesh_core.c)**

MS_prov_setup()と MS_prov_bind()で Unprovisioned Device Beacon とコネクタブルアドバタイジングの送信を開始してください。

MS_prov_setup()と MS_prov_bind()の使用例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_prov_setup()と mesh_core_prov_bind()をそれぞれ参照してください。

- **プロビジョニングコールバック関数 (mesh_core.c)**

プロビジョニングイベントを受け取るためのコールバック関数を実装してください。プロビジョニングクライアントから提供されたプロビジョニングデータは MS_access_cm_set_prov_data()で Mesh スタックに登録する必要があります。

プロビジョニングコールバック関数の実装例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_prov_cb()の実装を参照してください。

プロビジョニングサーバーの Mesh スタック API シーケンスを次頁以降で示します。

3.3.2　プロビジョニングサーバーのシーケンス

(1)　プロビジョニングのセットアップ

本サンプルプログラムは PB-ADV ベアラーと PB-GATT ベアラーの両方に対応しており、PB-ADV ベアラーによる Unprovisioned Device ビーコンと、PB-GATT ベアラーのための Connectable Undirected Advertising を交互に送信します。

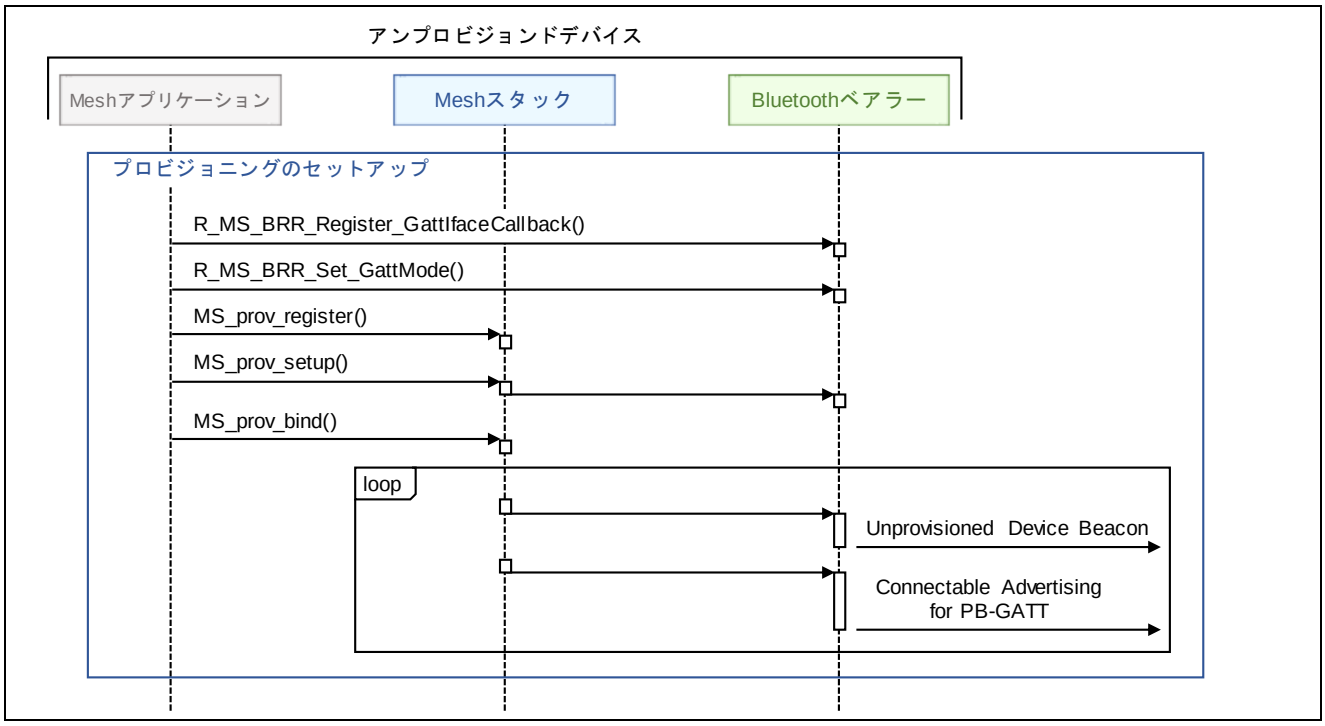

図 3-2　プロビジョニングのセットアップ

(2)　PB-ADV ベアラーによるセッションの確立

PB-ADV でプロビジョニング処理を実行する場合、プロビジョニングサーバーはプロビジョニングクライアントとセッションを確立します。またプロビジョニング処理の終了後、セッションを終了します。

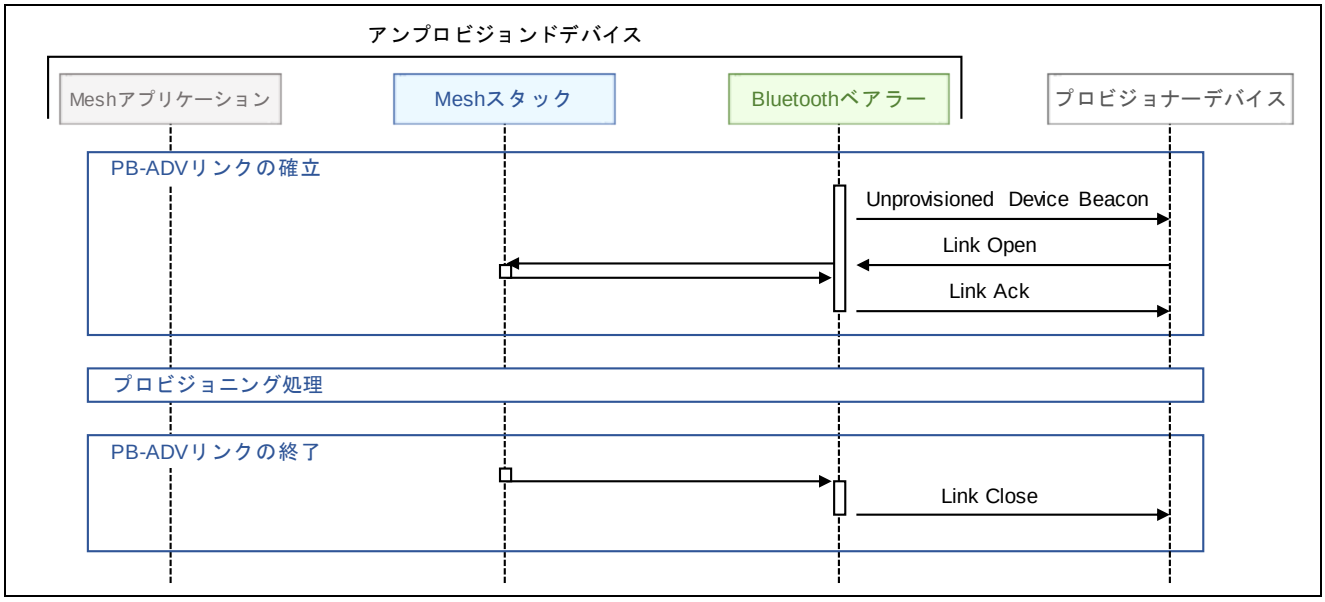

図 3-3　PB-ADV によるセッションの確立

(3) PB-GATT ベアラーによる接続の確立

PB-GATT でプロビジョニングを実行する場合、プロビジョニングクライアントはプロビジョニングサーバーと接続を確立し、Mesh プロビジョニングサービスの Notification を有効化します。またプロビジョニング処理の終了後、Notification を無効化し、接続を切断します。

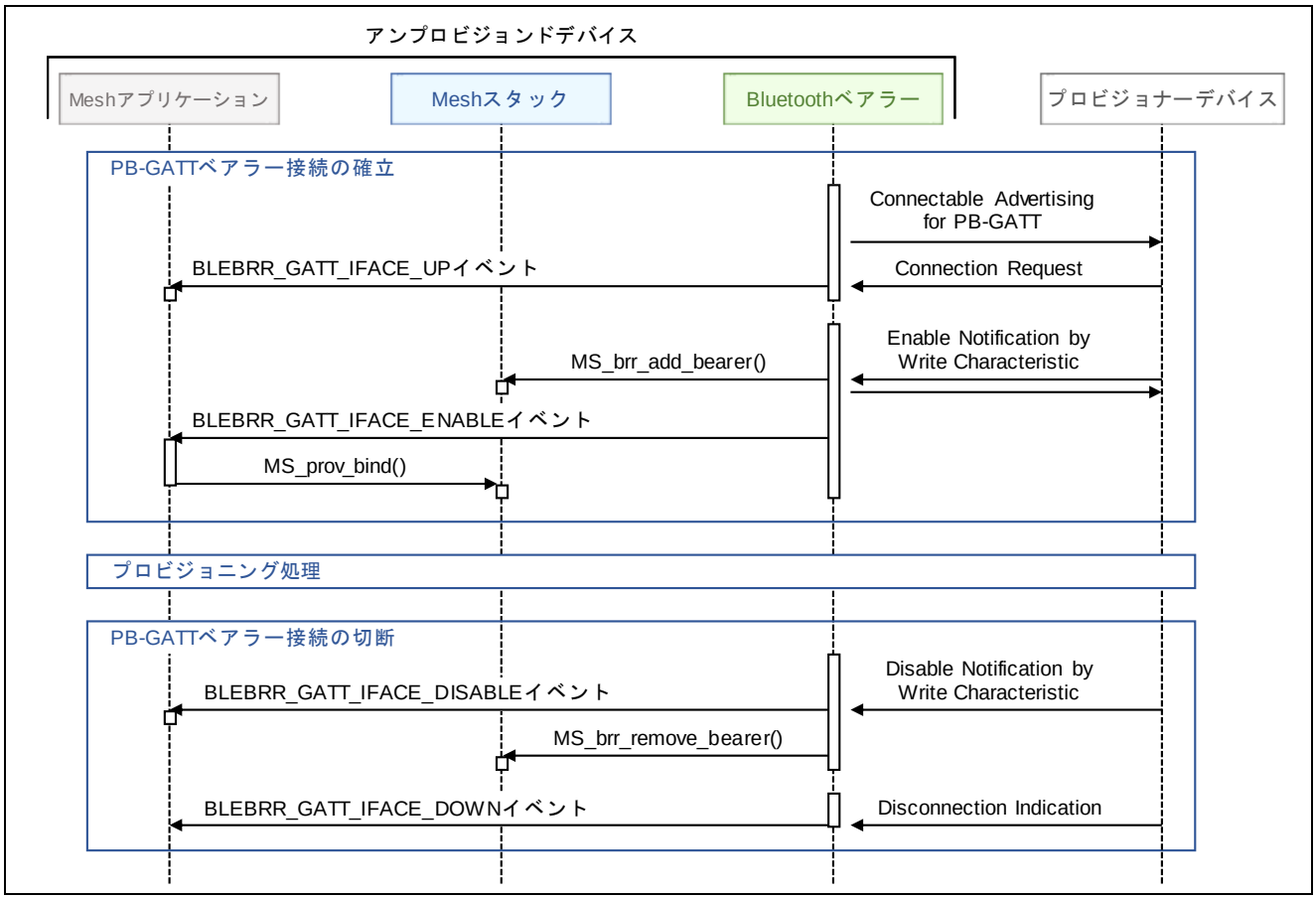

図 3-4 PB-GATT による接続の確立

(4) プロビジョニング処理

プロビジョニングベアラーによるセッションまたは接続の確立後、インビテーションからプロビジョニングデータの配布までのプロビジョニング処理が実行され、プロビジョニング PDU が交換されます。プロビジョニング処理では PB-ADV であっても PB-GATT であっても同一の処理が実行されます。

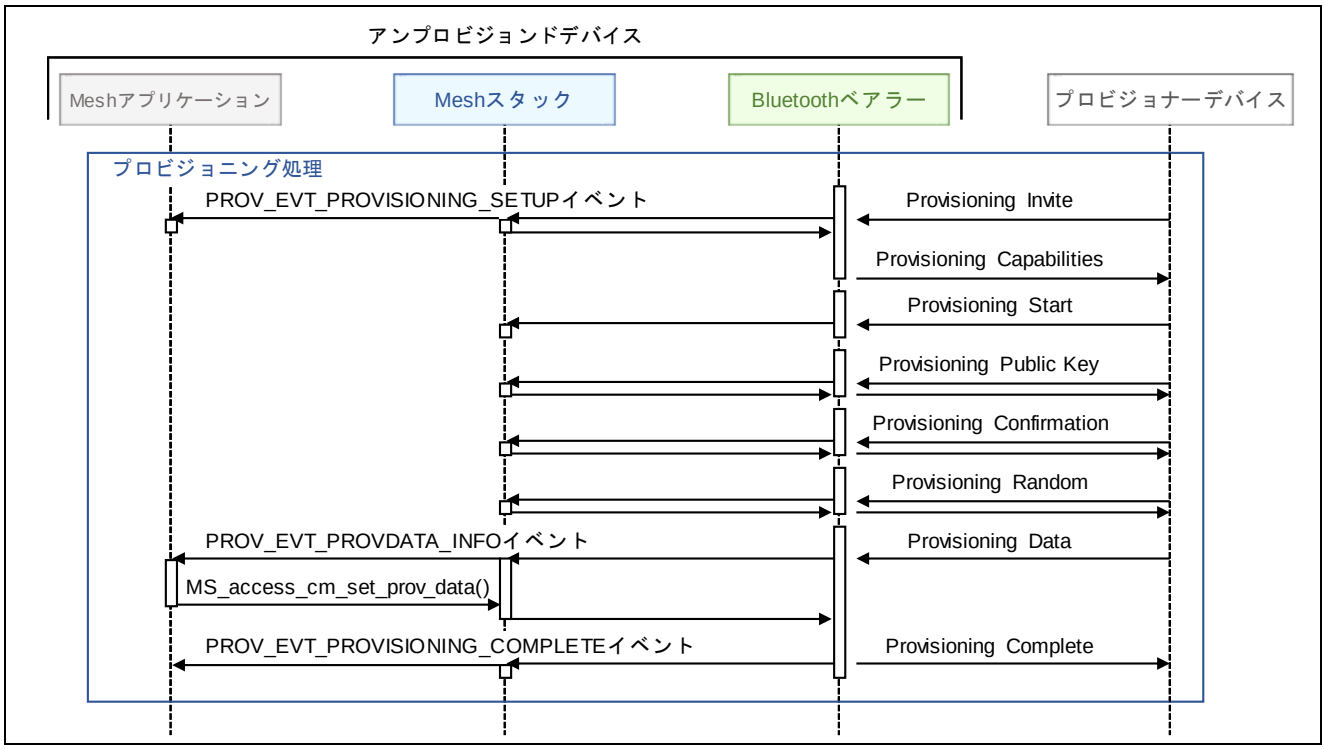

図 3-5 プロビジョニング処理

プロビジョニング処理でのセキュリティリスクの低減のため、下記が推奨されます。

- Public Key 交換ステップにおいて、OOB(Out Of Band)を使用すること
- Authentication ステップにおいて、128 ビットで取り得る最大のエントロピーを持つ乱数値、または暗号学的にセキュアな乱数値を AuthValue として選択すること

OOB で Public Key を配布するには、Public Key と Private Key を `MS_prov_generate_ecdh_key_pl()`で生成し、生成された Public Key を `MS_prov_set_local_public_key()`で Mesh スタックに設定します。対向のプロビジョナーデバイスへの Public Key の配布方法は、`PROV_OOB_TYPE_*`マクロで `MS_prov_setup()`の引数 `pdevice->oob` に設定してください。

OOB Authentication の AuthValue として使用可能な 128 ビット乱数は `R_BLE_VS_GetRand()`で生成することができます。Static OOB Authentication を利用する場合、生成した AuthValue は `MS_prov_set_static_oob_auth_pl()`で Mesh スタックに設定してください。Output OOB Authentication を利用する場合、生成した AuthValue は `MS_prov_set_authval()`で Mesh スタックに設定してください。

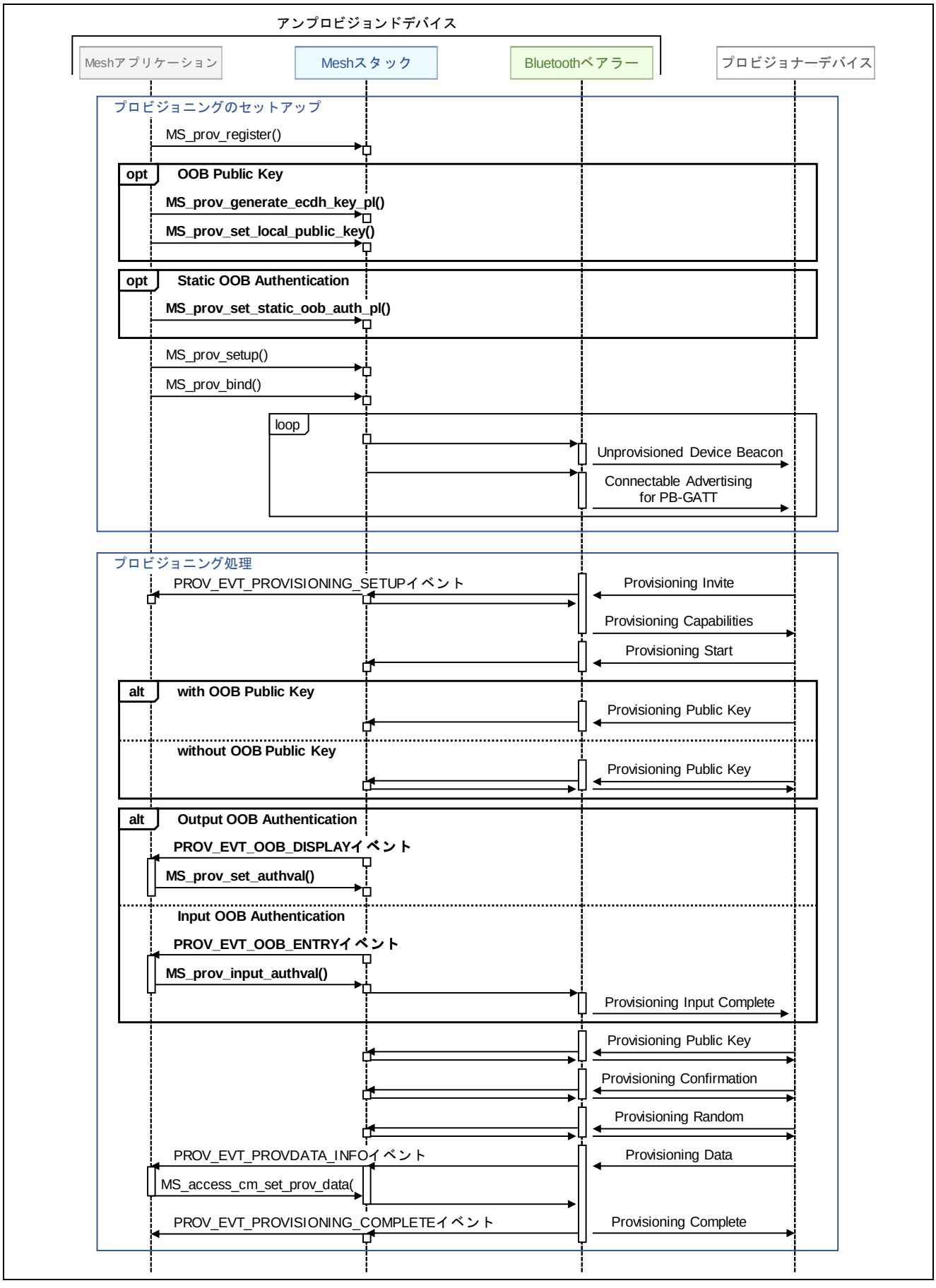

図 3-6 OOB を使用するプロビジョニング処理

3.4 プロキシ

本節ではプロキシサーバーまたはプロキシクライアントとして動作するための実装を示します。

3.4.1 プロキシサーバー

Mesh サンプルプログラムはプロキシサーバーとして動作することができます。Mesh サンプルプログラムのプロキシサーバー処理を以下に示します。

- **プロキシコールバック関数の登録 (mesh_core.c)**

R_MS_BRR_Set_GattMode()で Bluetooth ベアラーモードを BLEBRR_GATT_PROXY_MODE に設定してください。また、プロキシコールバック関数を MS_proxy_register()で Mesh スタックに登録してください。

R_MS_BRR_Set_GattMode()と MS_proxy_register()の使用例は、Mesh サンプルプログラム (mesh_core.c)の mesh_core_proxy_setup()の実装を参照してください。

- **コネクタブルアドバタイジングの開始 (mesh_core.c)**

プロキシクライアントとのプロキシ接続を確立するため、MS_proxy_server_adv_start()でコネクタブルアドバタイジングを開始してください。

MS_proxy_server_adv_start()の使用例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_proxy_start()の実装を参照してください。

- **プロキシコールバック関数 (mesh_core.c)**

プロキシイベントを受け取るためのコールバック関数を実装してください。

プロキシクライアントと接続を確立し、GATT プロキシサービスが有効化されると、MS_PROXY_UP_EVENT が通知されます。プロキシクライアントにキーリフレッシュフラグ、IV アップデートフラグ、カレント IV インデックスを配布するため、MS_net_broadcast_secure_beacon()で Secure Network Beacon を送信してください。

プロキシコールバック関数の実装例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_proxy_cb()の実装を参照してください。

- **プロキシ接続の切断**

プロキシクライアントとの接続を切断するには、R_MS_BRR_Disconnect()をコールしてください。

```
retval = R_MS_BRR_Disconnect(gs_proxy_client_conn_hdl);
```

3.4.2 プロキシクライアント

Mesh サンプルプログラムはプロキシクライアントとして動作することができます。Mesh サンプルプログラムのプロキシクライアント処理を以下に示します。

- **プロキシコールバック関数の登録 (mesh_core.c)**

R_MS_BRR_Set_GattMode()で Bluetoth ベアラーをプロキシモードに変更してください。また、プロキシコールバック関数を MS_proxy_register()で Mesh スタックに登録してください。

R_MS_BRR_Set_GattMode()と MS_proxy_register()の使用例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_proxy_setup()の実装を参照してください。

- **プロキシ接続の確立**

プロキシサーバーとのプロキシ接続を確立するため、R_BRR_Create_Connection()をコールしてください。プロキシサーバーをスキャンするには R_MS_BRR_Scan_GattBearer()をコールしてください。プロキシサーバーのデバイスアドレスは BLEBRR_GATT_IFACE_SCAN で通知されます。

```
st_ble_dev_addr_t remote_addr;
retval = R_MS_BRR_Create_Connection(&remote_addr, BLEBRR_GATT_PROXY_MODE);
```

- **プロキシコールバック関数 (mesh_core.c)**

プロキシイベントを受け取るためのコールバック関数を実装してください。

プロキシサーバーと接続を確立し GATT プロキシサービスが有効化されると、MS_PROXY_UP_EVENT が通知されます。MS_proxy_set_whitelist_filter()または MS_proxy_set_blacklist_filter()でプロキシサーバーのプロキシフィルタタイプを設定し、MS_access_cm_get_all_model_subscription_list()でプロキシサーバーのプロキシフィルタリストにサブスクリプションアドレスを追加してください。

プロキシコールバック関数の実装例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_proxy_cb()の実装を参照してください。

- **プロキシ接続の切断**

プロキシサーバーとの接続を切断するには、R_MS_BRR_Disconnect()をコールしてください。

```
retval = (API_SUCCESS == R_MS_BRR_Disconnect(gs_proxy_server_conn_hdl);
```

プロキシサーバーの Mesh スタック API シーケンスを次頁以降で示します。

3.4.2.1 プロキシのシーケンス

(1) プロキシ機能のセットアップ

本サンプルプログラムはプロキシ機能に対応しており、GATT ベアラーにのみ対応するコンフィグレーションクライアントは、GATT ベアラーで本サンプルプログラムの動作を設定することができます。さらに本サンプルプログラムは、GATT ベアラーにのみ対応するノードに対して、GATT ベアラーと ADV ベアラー間でメッセージを転送することができます。

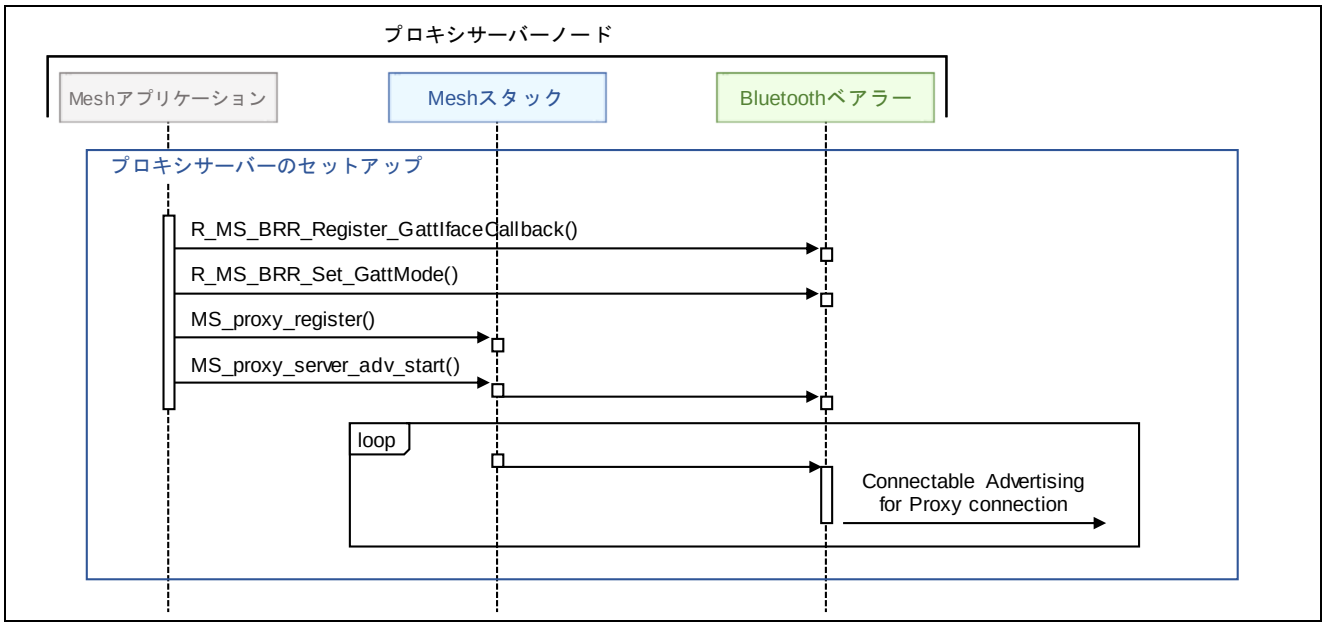

図 3-7 プロキシ機能のセットアップ

(2) プロキシ接続の確立

プロキシクライアントはプロキシサーバーと接続を確立し、Mesh プロキシサービスの Notification を有効化します。これにより、プロキシクライアントは GATT ベアラーによる Mesh メッセージの通信ができます。

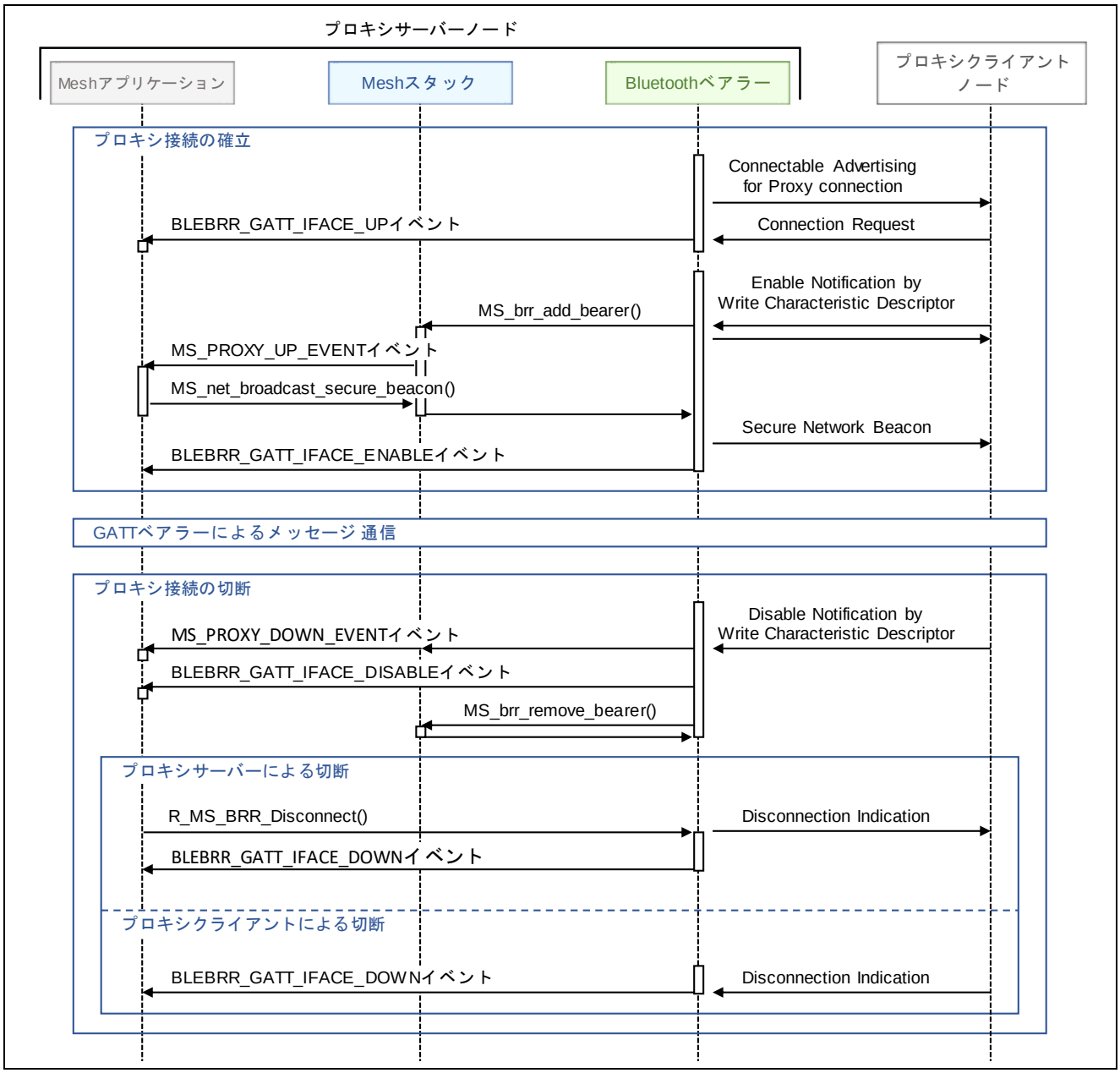

図 3-8 プロキシ接続の確立と切断

3.5 フレンドシップ

本節ではフレンドノードまたはローパワーノードとして動作するための実装を示します。

3.5.1 フレンドノード

フレンドノードとして動作するには、フレンド機能が有効化される必要があります。フレンド機能は下記のいずれかの手段によって有効化されます。

- コンフィグレーションクライアントが Config Friend Set メッセージを送信
- アプリケーションが MS_ENABLE_FRIEND_FEATURE()を実行

フレンド機能の有効化後、フレンドシップの確立やメッセージの保持、ローパワーノードへの応答といったフレンド機能に関連する処理は Mesh スタックが自動で実行するため、アプリケーションが対処する必要はありません。

3.5.2 ローパワーノード

ローパワーノードとして動作するには、アプリケーションがローパワー機能を有効化し、フレンドノードに対してフレンドシップ確立を要求する必要があります。フレンドシップの確立後、Mesh スタックはフレンドノードへの問い合わせと Scan の休止と再開を自動で実行します。

Mesh サンプルプログラムはローパワーノードとして動作することができます。Mesh サンプルプログラムのローパワーノードとしてフレンドシップを確立するための実装を以下に示します。

注: 本機能はデフォルトでは無効となっています。本機能を有効化するには、2.6 節を参照して LOW_POWER_FEATURE_EN マクロの値を変更してください。

- **ローパワー機能の有効化 (mesh_core.c)**

MS_ENABLE_LPN_FEATURE()でローパワー機能を有効化してください。

```
#if LOW_POWER_FEATURE_EN
MS_ENABLE_LPN_FEATURE();
#endif /* LOW_POWER_FEATURE_EN */
```

- **フレンドシップ確立の要求 (mesh_core.c)**

ローパワーノードとしてフレンドシップを確立するため、MS_trn_lpn_setup_friendship()で Friend Request メッセージを送信してください。本関数の引数にはフレンドシップコールバック関数に加え、フレンドノードに対する問い合わせのタイミングに関するパラメータを設定してください。

MS_trn_lpn_setup_friendship()の使用例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_lpn_setup()の実装を参照してください。

- **フレンドシップコールバック関数 (mesh_core.c)**

Mesh スタックが通知するフレンドシップイベントを受け取るコールバック関数を実装してください。フレンドシップが確立すると MS_TRN_FRIEND_SETUP_CNF が通知されます。またフレンドシップが終了すると MS_TRN_FRIEND_TERMINATE_IND が通知されます。

ローパワーノードはフレンドノードのフレンドサブスクリプションリストにサブスクリプションアドレスを追加または除去することができます。Mesh サンプルプログラムはフレンドシップの確立後、

MS_access_cm_get_all_model_subscription_list()でサブスクリプションリストから全てのサブスクリプションアドレスを取得し、MS_trn_lpn_subscrn_list_add()でフレンドノードのフレンドサブスクリプションリストに追加します。

フレンドシップコールバック関数の実装例は、Mesh サンプルプログラム(mesh_core.c)のmesh_core_lpn_cb()の実装を参照してください。

3.5.3　ローパワーノードのシーケンス

(1)　ローパワー機能の有効化とフレンドシップの要求

本サンプルプログラムはローパワー機能に対応しており、フレンドノードとフレンドシップを確立するための Friend Request 送信を開始します。

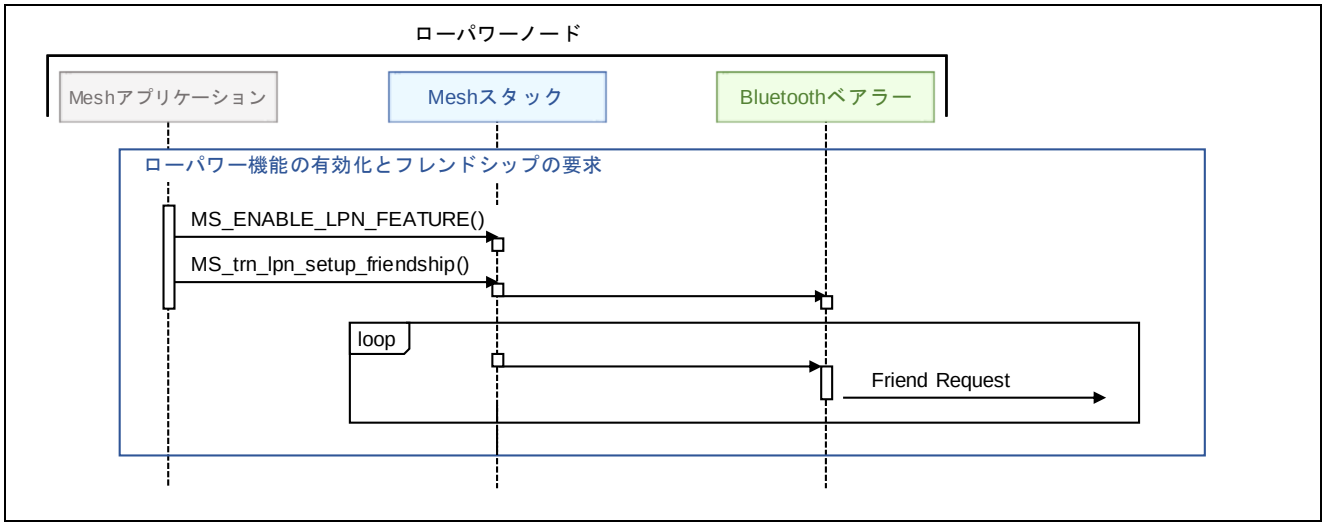

図 3-9　ローパワー機能の有効化とフレンドシップの要求

(2) フレンドシップの確立と切断

フレンドノードからの Friend Offer を受信することでフレンドシップが確立されます。フレンドシップの確立後、本サンプルプログラムはフレンドノードの Friend Subscription List に全てのサブスクリプションアドレスを登録します。これにより、これらのサブスクリプションアドレス宛のメッセージをフレンドノードが保持します。一方、Mesh スタックは定期的にメッセージポーリングを実行し、フレンドノードからメッセージを受信します。

フレンドノードへのメッセージポーリングの失敗が連続し、Friend Poll Timeout の発生によってフレンドシップが切断されると、本サンプルプログラムはフレンドシップを確立するための Friend Request 送信を再開します。

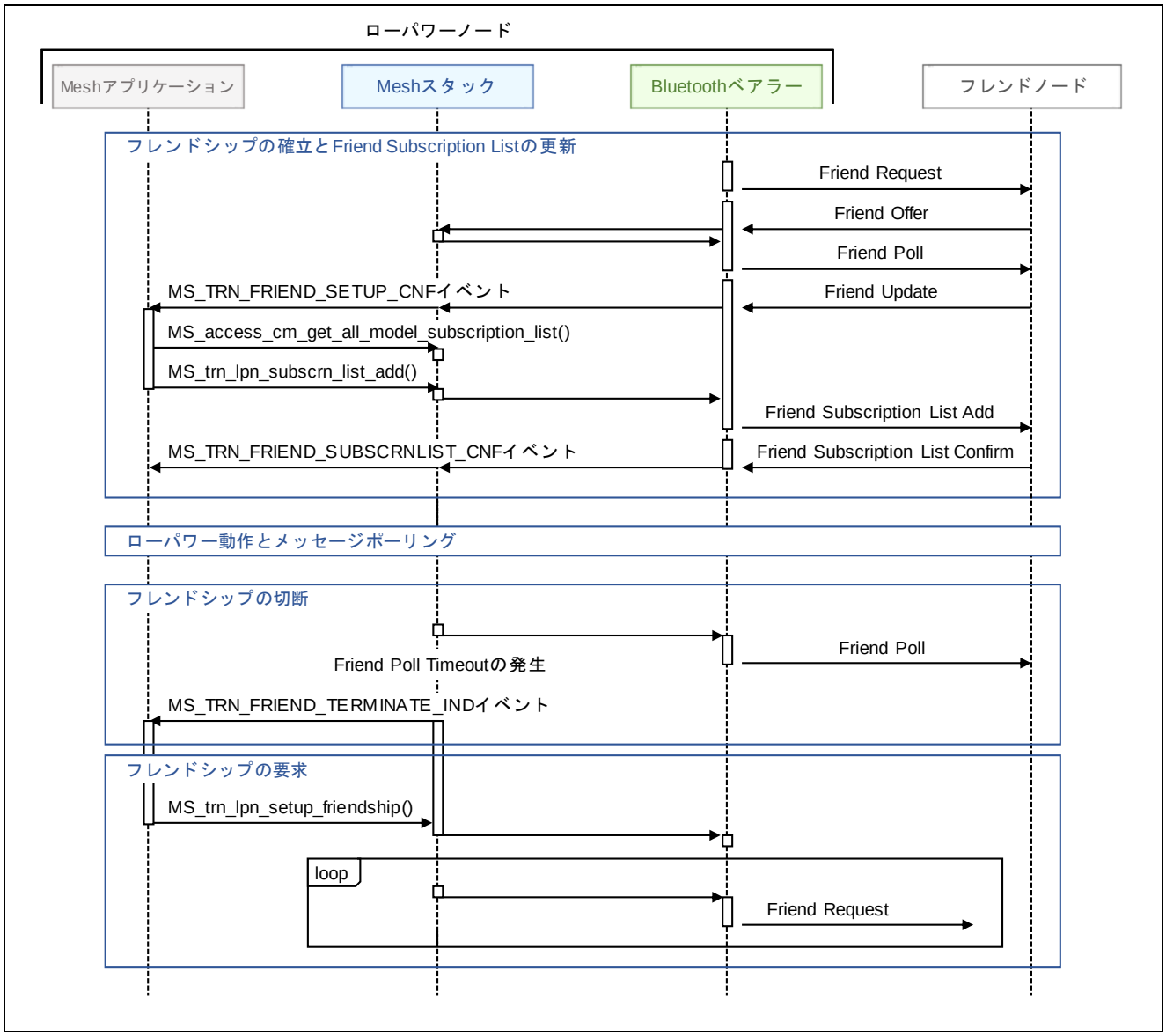

図 3-10 フレンドシップの確立と切断

ローパワーノードからフレンドシップを自発的に切断する場合は、MS_trn_lpn_clear_friendship()で Friend Clear メッセージを送信してください。切断の完了は MS_TRN_FRIEND_CLEAR_CNF イベントで通知されます。

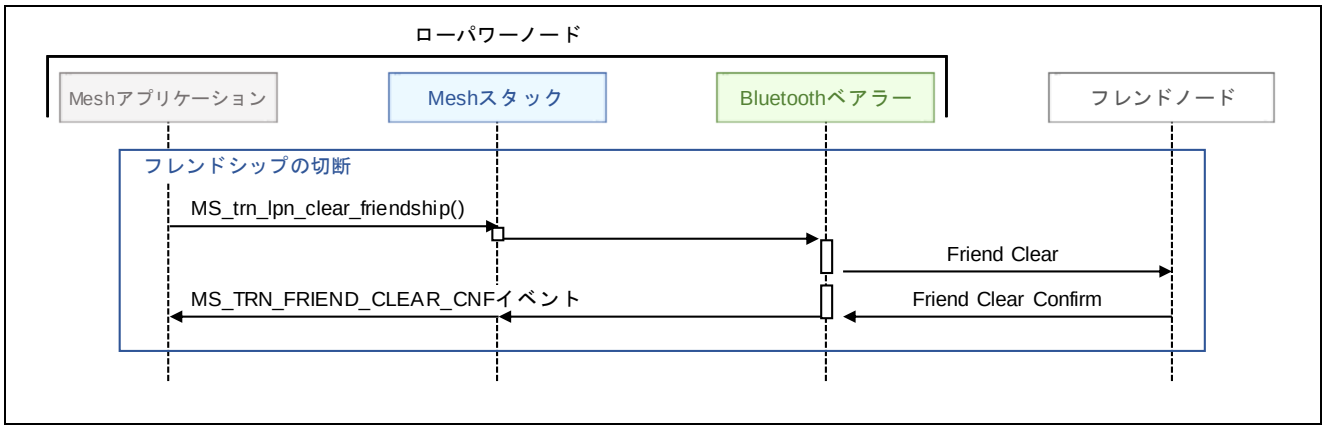

図 3-11　フレンドシップの切断

3.5.4　フレンドノードのシーケンス

(1)　フレンド機能の有効化

コンフィグレーションクライアントからフレンド機能が有効化されることで、本サンプルプログラムはフレンドノードとして動作することができます。Mesh スタックはフレンドシップの確立や切断などのフレンドノード動作に関するイベントを通知しません。

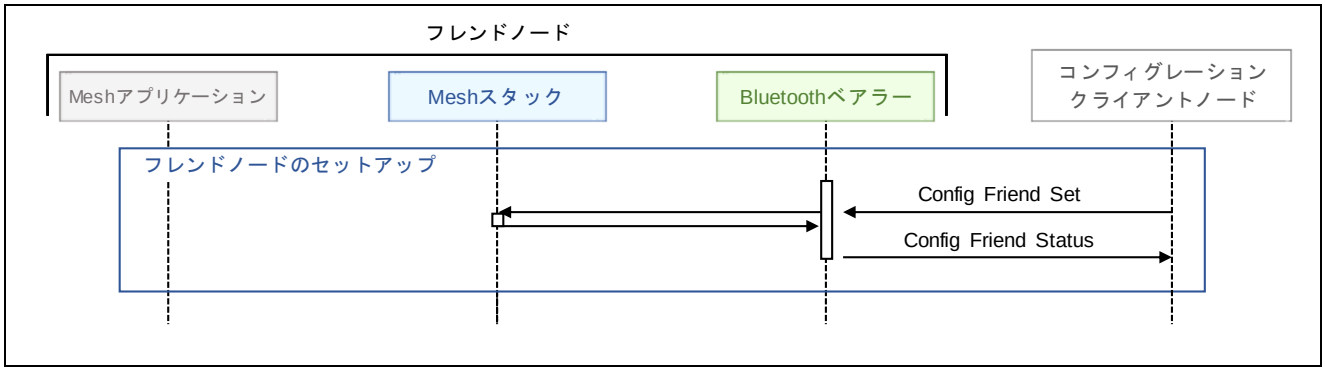

図 3-12　フレンド機能の有効化

(2) フレンドシップの確立と切断

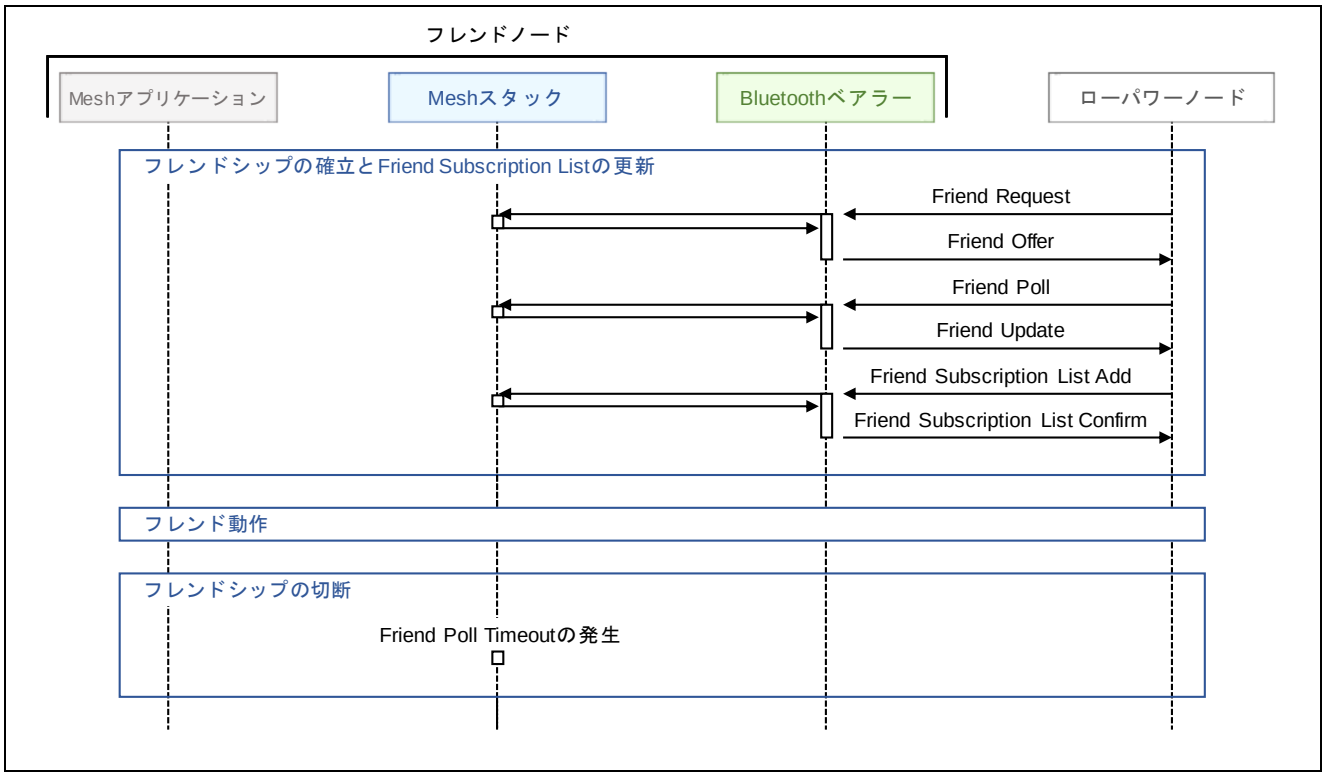

図 3-13 フレンドシップの確立と切断

3.6 コンフィグレーション

本節ではコンフィグレーションサーバーの実装方法を示します。

3.6.1 コンフィグレーションサーバー

プロビジョニングの完了後、ノードが各モデルによって通信するには、コンフィグレーションクライアントからアプリケーションキーやパブリッシュアドレス、サブスクリプションアドレスといった設定情報を受け取る必要があります。これらの設定情報は、コンフィグレーションモデルによってコンフィグレーションステートとして扱われます。

アプリケーションがコンフィグレーションサーバーモデルを登録すると、コンフィグレーションステートのメモリ領域が Mesh スタックに確保されます。Mesh スタックは、コンフィグレーションクライアントからコンフィグレーションメッセージを受信すると、コンフィグレーションステートを自動的に更新します。このため、アプリケーションがコンフィグレーションステートを管理する必要はありません。

Mesh スタックは、ローカルのコンフィグレーションステートにアクセスするための API を提供します。アプリケーションはこれらの API を使用することで、コンフィグレーションステートに直接アクセスすることもできます。コンフィグレーションステートにアクセスするための API は、Bluetooth Mesh スタック API マニュアル (blemesh_api.chm)の[Modules]→[Mesh Core]→[ACCESS (Access Layer)]→[API Definitions]に記載された `MS_access_cm_get_*()`を参照してください。

Mesh サンプルプログラムはコンフィグレーションサーバーとして動作します。Mesh サンプルプログラムのコンフィグレーションサーバーモデルの実装を示します。

- **コンフィグレーションサーバーモデルの登録 (mesh_model.c)**

  `MS_config_server_init()`でコンフィグレーションサーバーモデルをエレメントに登録します。

  `MS_config_server_init()`の使用例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_foundation_model_register()`の実装を参照してください。

- **コンフィグレーションサーバーコールバック関数 (mesh_model.c)**

  コンフィグレーションクライアントから受信したメッセージを受け取るコールバック関数を実装します。

  コンフィグレーションサーバーコールバック関数の実装例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_model_config_server_cb()`の実装を参照してください。

コンフィグレーションサーバーの Mesh スタック API シーケンスを次頁に示します。

3.6.2 コンフィグレーションサーバーのシーケンス

Config Node Reset メッセージを受信すると、Mesh スタックは全てのコンフィグレーションステートを消去します。また本サンプルプログラムは MCU をリセットし、プロビジョニングを再度実行します。

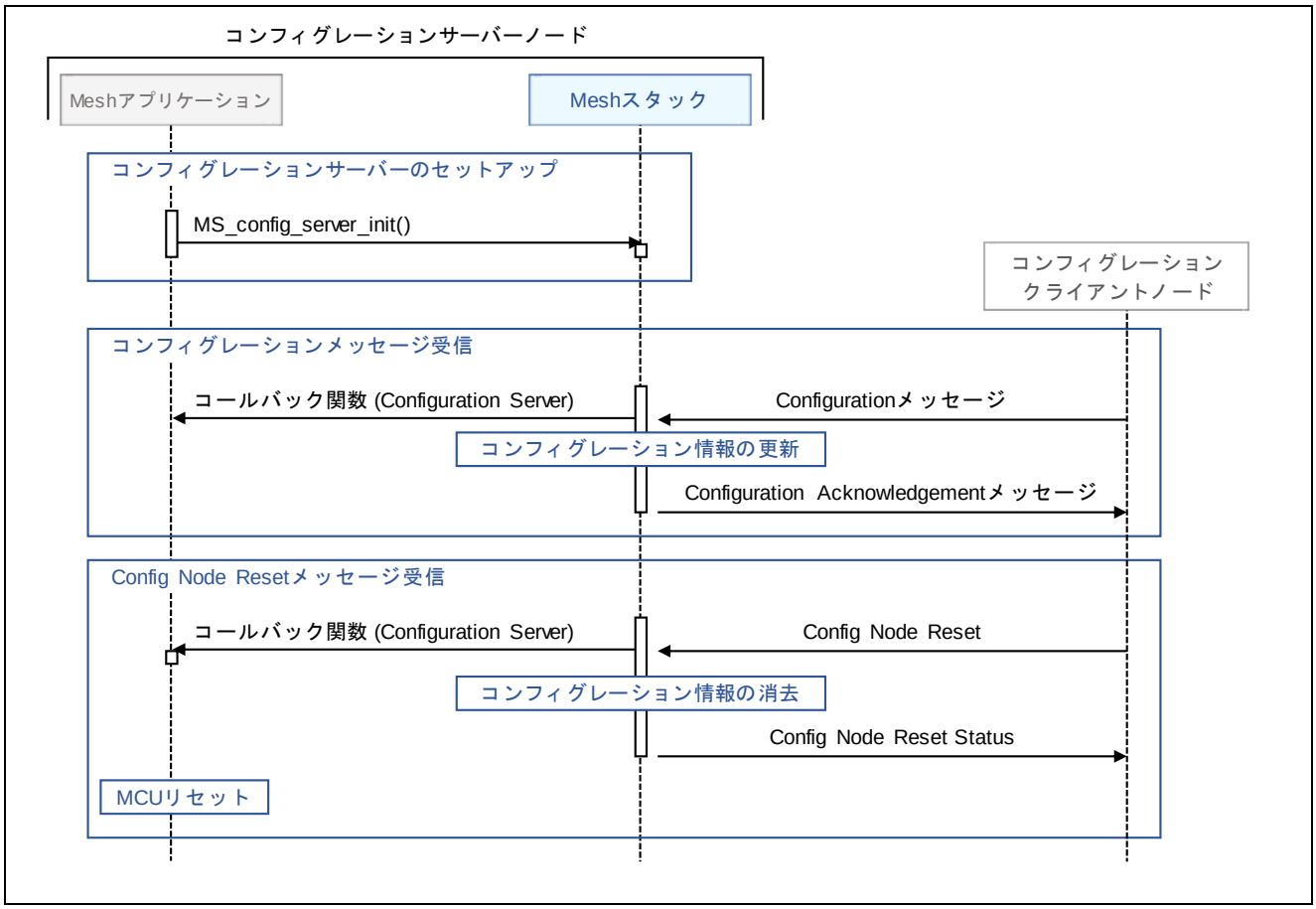

図 3-14 Mesh サンプルプログラムの Configuration Server 動作

3.7 ヘルスモデル

本節ではヘルスサーバーの実装方法を示します。

3.7.1 ヘルスサーバー

ヘルスサーバーはヘルスクライアントからの Health Fault Test メッセージによりセルフテストを実行します。またヘルスクライアントからの Health Attention Set メッセージによりアテンションタイマを起動します。

Mesh サンプルプログラムはヘルスサーバーとして動作します。Mesh サンプルプログラムのセルフテストとアテンションタイマ通知の実装を示します。

- **ヘルスサーバーモデルの登録 (mesh_model.c)**

  `MS_health_server_init()`でヘルスサーバーモデルをエレメントに登録します。

  `MS_health_server_init()`の使用例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_foundation_model_register()`の実装を参照してください。

- **テスト ID の定義 (main.c)**

  Health Fault Test メッセージの受信により実行するセルフテストのテスト ID を定義してください。

  テスト ID の例は、Mesh サンプルプログラム(main.c)の `e_mesh_health_test_id_t` の宣言を参照してください。

- **テスト関数の定義 (main.c)**

  Health Fault Test メッセージの受信により実行するセルフテストのテスト関数を定義してください。

  テスト関数の実装例は、Mesh サンプルプログラム(main.c)の `mesh_health_self_test_00()`、`mesh_health_self_test_01()`の実装を参照してください。

- **テスト ID とテスト関数の登録 (main.c)**

  定義したテスト ID とテスト関数の組合せを登録してください。

  テスト ID とテスト関数の登録例は、Mesh サンプルプログラム(main.c)の `gs_health_server_self_tests` の宣言を参照してください。

- **セルフテスト結果の追加と通知 (mesh_model.c)**

  フォールトステートにヘルステスト結果を追加して Health Fault Status メッセージを送信するため、`MS_health_server_report_fault()`を実行してください。

  `MS_health_server_report_fault()`の使用例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_model_health_server_fault_status()`の実装を参照してください。

- **アテンションタイマコールバック関数 (main.c)**

  Health Attention Set メッセージにより実行されるアテンションタイマコールバック関数を実装します。

  アテンションタイマの開始時は `MS_HEALTH_SERVER_ATTENTION_START` イベント、再開時は `MS_HEALTH_SERVER_ATTENTION_RESTART` イベントが通知されます。これらのイベントにより LED 点滅などのアテンション動作を開始してください。

  アテンションタイマの停止時は `MS_HEALTH_SERVER_ATTENTION_STOP` イベントが通知されます。このイベントによりアテンション動作を停止してください。

アテンションタイマコールバック関数の実装例は、Mesh サンプルプログラム(main.c)の mesh_health_server_cb()の実装を参照してください。

3.7.2　ヘルスサーバーのシーケンス

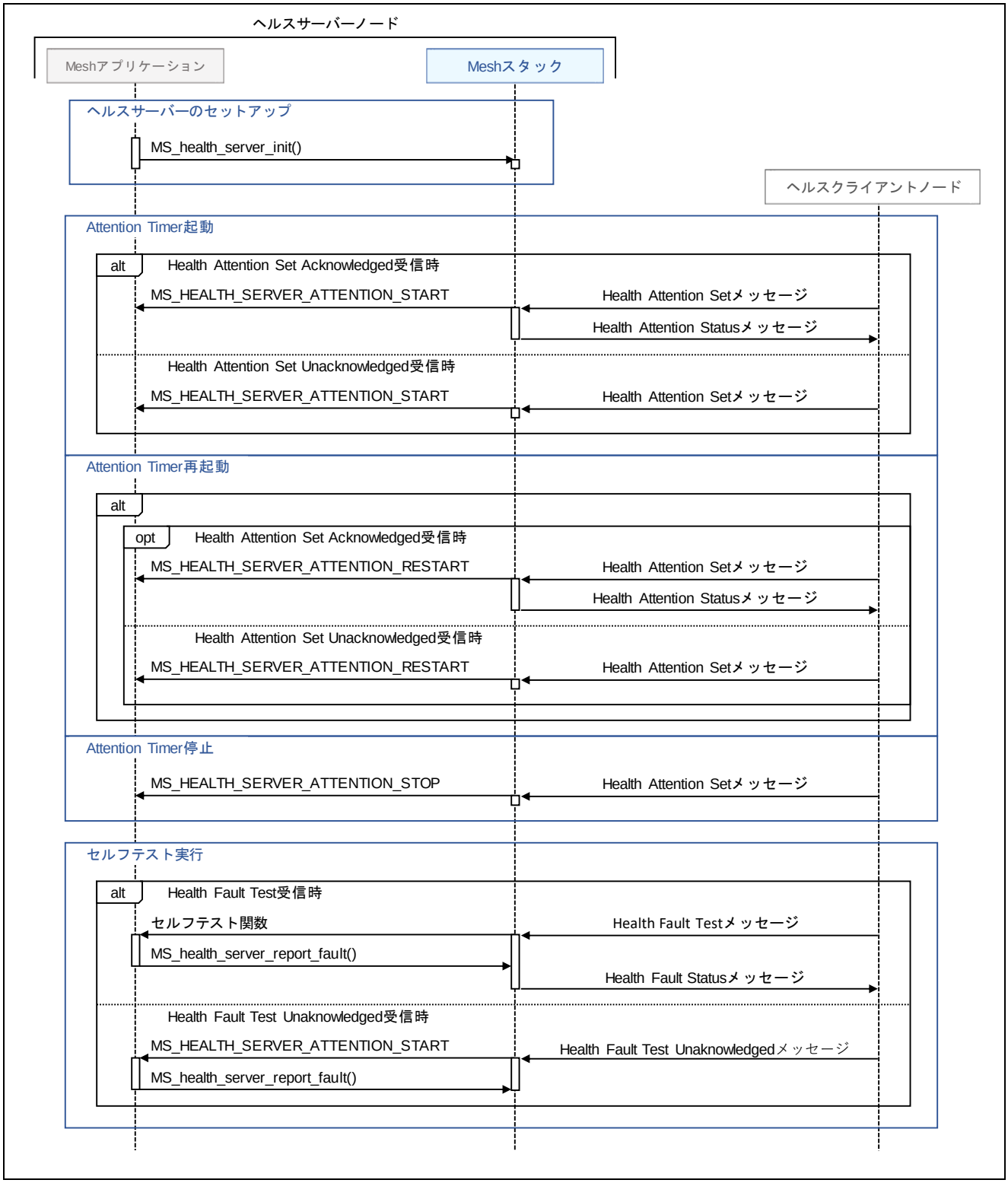

図 3-15　Mesh サンプルプログラムの Health Server 動作

3.8 アプリケーションモデル

アプリケーションが使用すべきモデルは、アプリケーションのシナリオによって異なります。アプリケーションは、一つのモデルまたは複数のモデルを使用することができます。Mesh スタックは、Bluetooth Mesh Model 仕様で定義されたモデルを使用するための API をアプリケーションに提供します。

本節では、Mesh サンプルプログラムの Generic OnOff モデルの実装を参照しながら、アプリケーションモデルの実装方法を示します。

Mesh サンプルプログラムは Generic OnOff クライアントまたは Generic OnOff サーバーとして動作します。Generic OnOff クライアントは、Generic OnOff サーバーの Generic OnOff ステートを ON または OFF に変更することができます。Mesh サンプルプログラムの Generic OnOff モデルの実装を以下に示します。

3.8.1 サーバーモデル

- **ステート変数の宣言と初期化 (mesh_model.c)**

ステートのインスタンスをグローバル変数として宣言し、ステート変数を初期化してください。

Generic OnOff ステートの宣言例は、Mesh サンプルプログラム(mesh_model.c)の `gs_onoff_state` 変数を参照してください。

Generic OnOff ステートの初期化例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_application_model_states_init()`を参照してください。

- **サーバーモデルの登録 (mesh_model.c)**

サーバーモデルとそのエレメントハンドル、コールバック関数の登録を実装します。

Generic OnOff サーバーモデル登録の例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_application_model_register()`の実装を参照してください。

- **サーバーモデルコールバック関数 (mesh_model.c)**

クライアントからのメッセージを受け取り、グローバル変数として定義されたステートを操作するためのコールバック関数を実装してください。

Generic OnOff サーバーモデルコールバック関数の実装例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_model_onoff_server_cb()`の実装を参照してください。

3.8.2 クライアントモデル

- **クライアントモデルの登録 (mesh_model.c)**

  クライアントモデルとそのエレメントハンドル、コールバック関数の登録を実装します。

  Generic OnOff クライアントモデル登録の例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_application_model_register()`の実装を参照してください。

- **メッセージを受け取るコールバック関数 (mesh_model.c)**

  サーバーからのメッセージを受け取るコールバック関数を実装します。

  Generic OnOff クライアントモデルコールバック関数の実装例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_model_onoff_client_cb()`の実装を参照してください。

- **メッセージの送信関数 (mesh_model.c)**

  GET や SET といったメッセージを送信する関数を実装します。さらに、アプリケーションはプッシュスイッチといったトリガに従って、これらの関数を実行する必要があります。

  Generic OnOff メッセージの送信例は、Mesh サンプルプログラム(mesh_model.c)の `mesh_model_onoff_client_get()`、`mesh_model_onoff_client_set()`、`mesh_model_onoff_client_set_unack()`の実装を参照してください。

Generic OnOff モデルの Mesh スタック API シーケンスを次頁に示します。

3.8.3 Generic OnOff モデルのシーケンス

Generic OnOff Client ノードとして動作する Mesh サンプルプログラムは、ボード上のスイッチが押下されると Generic OnOff Set Unacknowledged メッセージを送信します。一方、Generic OnOff Server ノードとして動作する Mesh サンプルプログラムは、Generic OnOff Set メッセージまたは Generic OnOff Set Unacknowledged メッセージを受信すると、ボード上の LED をオンまたはオフに制御します。

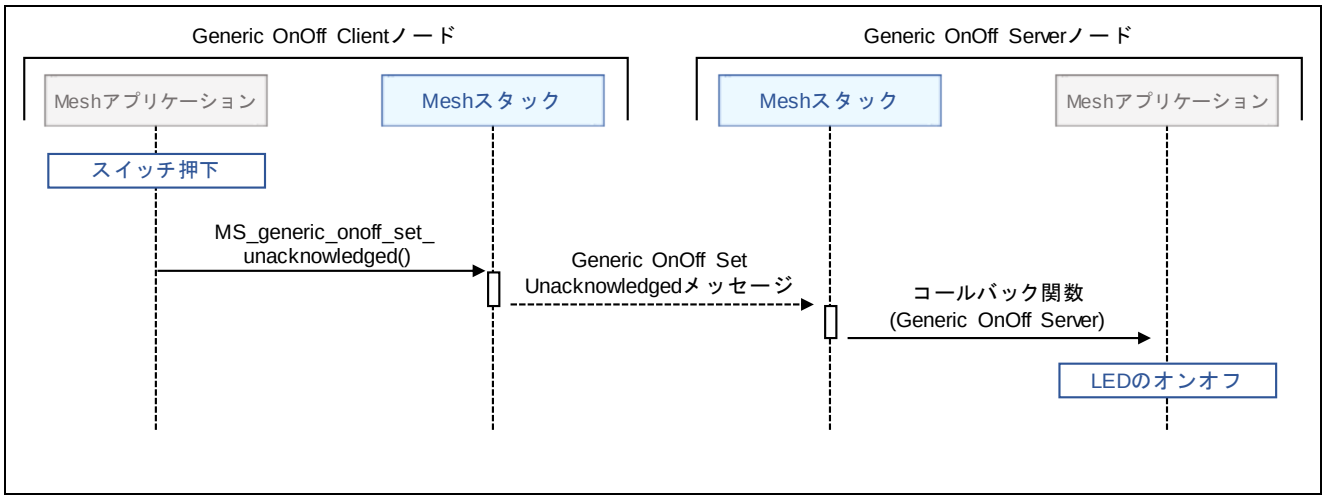

図 3-16 Mesh サンプルプログラムの Generic OnOff モデル動作

3.8.4 Vendor Model のシーケンス

Vendor Client ノードとして動作する Mesh サンプルプログラムは、コンソールから入力された文字列を Vendor Set Unacknowledged メッセージで送信します。一方、Vendor Server ノードとして動作するサンプルプログラムは、Vendor Set メッセージまたは Vendor Set Unacknowledged メッセージを受信すると、コンソールに文字列を出力します。

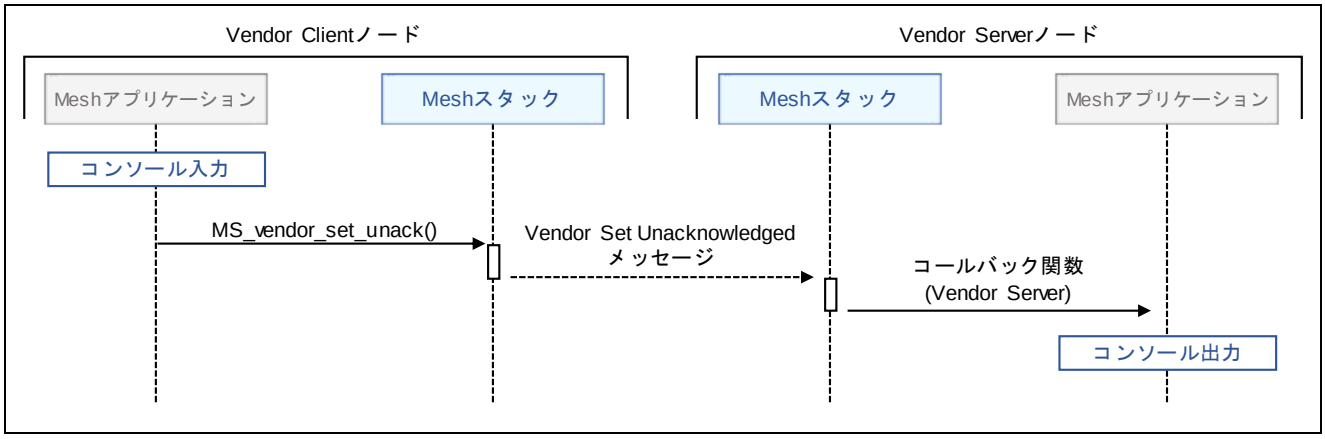

図 3-17 Mesh サンプルプログラムの Vendor モデル動作

3.8.5 Mesh モニタ機能

Mesh サンプルプログラムは Mesh スタックの各レイヤーが送受信した Protocol Data Unit(PDU)と Secure Network Beacon(SNB)をコンソール上にログ出力することができます。これにより Mesh アプリケーションの開発時、通信状況を容易に理解することができます。

Mesh サンプルプログラムの Mesh モニタ設定処理を以下に示します。

注: 本機能はデフォルトでは無効となっています。本機能を有効化するには、2.6 節を参照して CONSOLE_MONITOR_CFG マクロの値を変更してください。

- **Mesh モニタ機能の有効化 (mesh_core.c)**

MS_monitor_register_pl()で Mesh モニタ機能を有効化してください。

MS_monitor_register_pl()の使用例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_monitor_setup()の実装を参照してください。

- **Mesh モニタコールバック関数 (mesh_core.c)**

Mesh モニタコールバック関数にコンソールへの出力処理を実装してください。

Mesh モニタコールバック関数の実装例は、Mesh サンプルプログラム(mesh_core.c)の mesh_core_monitor_access_pdu()、mesh_core_monitor_trans_pdu()、mesh_core_monitor_ltrans_pdu()、mesh_core_monitor_net_pdu()、mesh_core_monitor_generic_log()を参照してください。

Mesh モニタ機能の Mesh スタック API シーケンスを次頁に示します。

3.8.5.1 Mesh モニタ機能のシーケンス

本サンプルプログラムは Mesh スタックの Mesh モニタ機能を有効化します。Mesh スタックの各レイヤーが送受信した Protocol Data Unit (PDU)と Secure Network Beacon (SNB)はコールバック関数で通知されます。

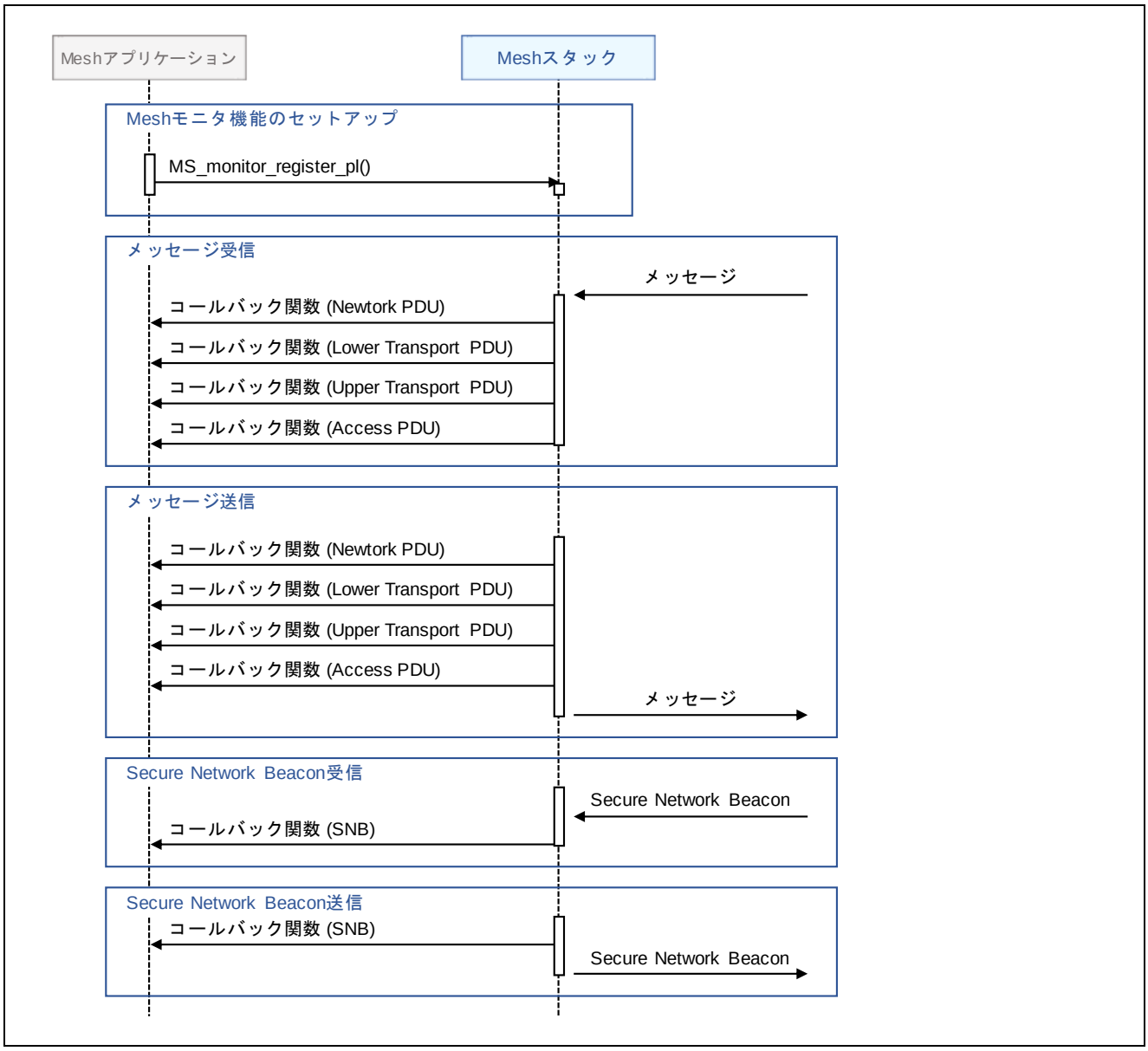

図 3-18 Mesh モニタ機能

4. Appendix

4.1 Command Line Interface プログラム

Command Line Interface (CLI)は、PC からシリアルインタフェース経由で Mesh スタック API を実行するためのインタフェースです。Bluetooth Mesh スタックパッケージには Command Line Interface プログラム (mesh_cli)が含まれます。

本プログラムを使用することで、Mesh スタックの無線通信動作を確認できます。さらに Mesh スタック API を使用例として、本プログラムのソースコードを参照することができます。

図 4-1 に Command Line Interface プログラムの使用例を示します。

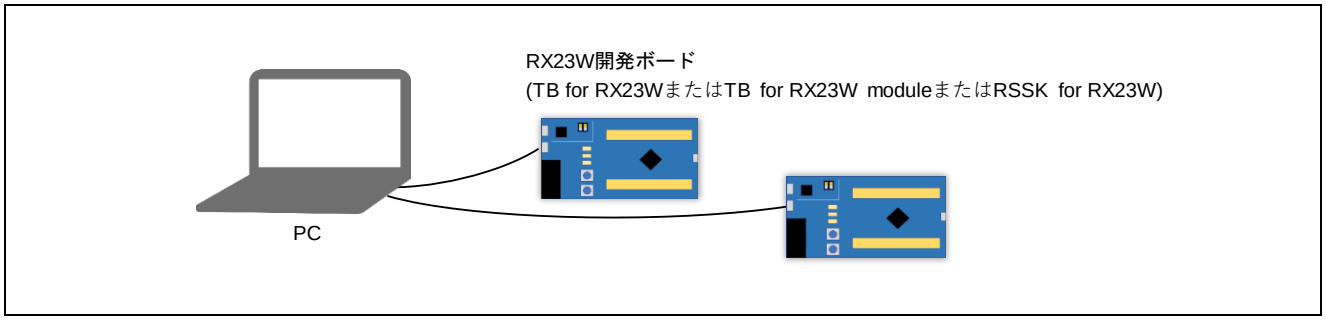

図 4-1 Command Line Interface プログラムの使用例

図 4-2 に Command Line Interface のシーケンス例を示します。本プログラムは Provisioning Client と Provisioning Server、Configuration Client と Configuration Server といった両方のロールを実行できます。

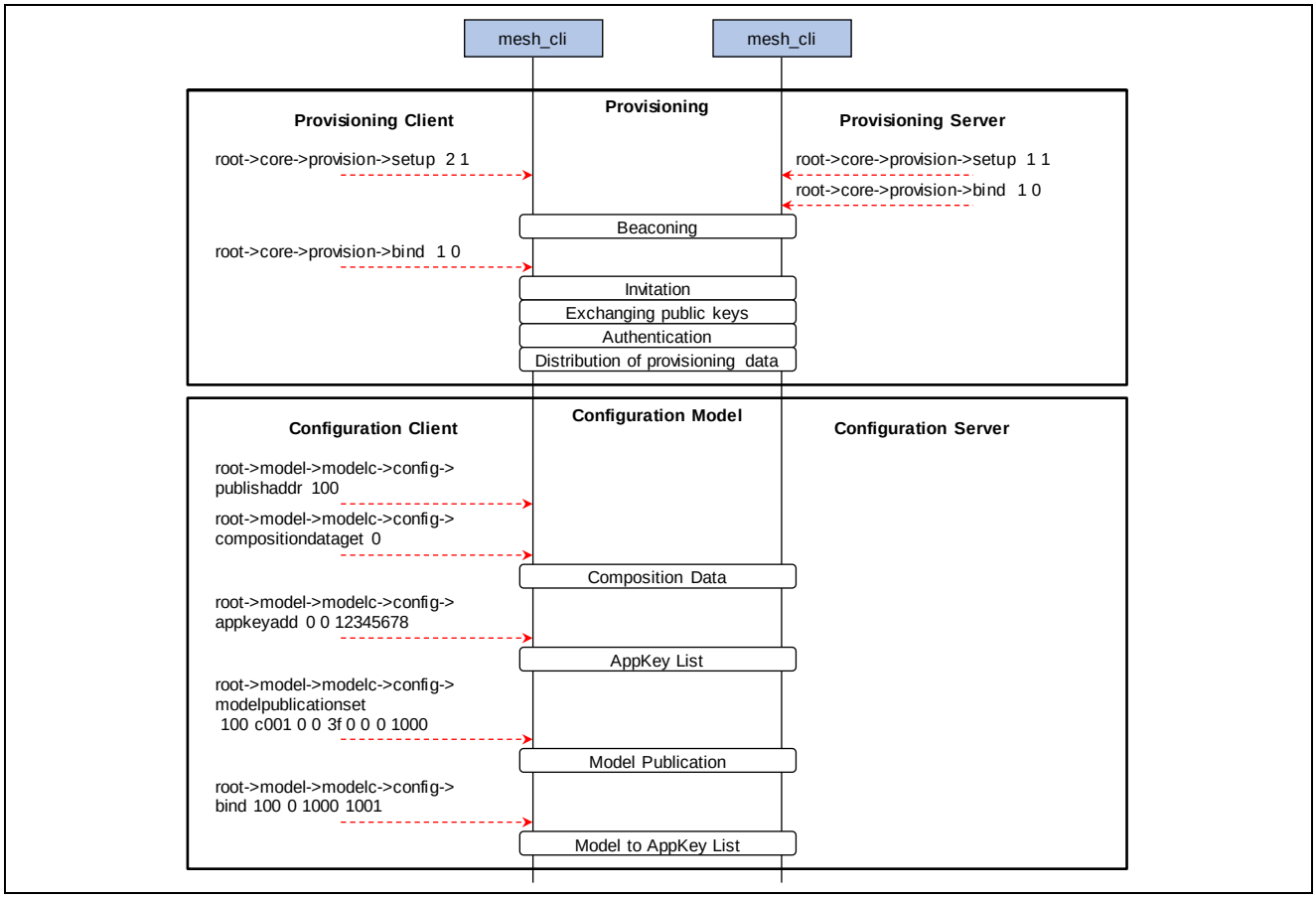

図 4-2 Command Line Interface のシーケンス例

Command Line Interface プログラムのビルド環境構築については、「RX23W グループ Bluetooth Mesh スタック スタートアップガイド」(R01AN4874)の 6 章を参照し、ワークスペースディレクトリに生成された mesh_cli プロジェクトを使用してください。

Target Board for RX23W と RSSK for RX23W は、PC と通信するための USB シリアル変換を搭載しています。Command Line Interface を操作するには、PC 上でシリアルターミナルツールを使用してください。(例：Tera Term)

表 4-1 に Command Line Interface プログラムと通信するためのシリアルポート設定を示します。

表 4-1　シリアルポート設定

項目	設定
ボーレート	115200 bps
データ	8 bits
パリティ	なし
ストップ	1 bit
フロー制御	なし

Command Line Interface の仕様については、Bluetooth Mesh スタックパッケージに同梱された FITDemos\mesh_cli\mesh_cli_guide.pdf を参照してください。

商標権および著作権

Bluetooth® のワードマークおよびロゴは Bluetooth SIG, Inc が所有する登録商標であり、ルネサスエレクトロニクス株式会社はこれらのマークをライセンスに基づいて使用しています。その他の商標および登録商標はそれぞれの所有者に帰属します。

RX23W グループ Bluetooth Mesh スタックは次のオープンソースソフトウェアを使用します。

crackle; AES-CCM, AES-128bit 機能

BSD 2-Clause License

Copyright (c) 2013-2018, Mike Ryan
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

改訂記録

Rev.	発行日	改定内容	
1.00	2019.09.30	-	初版発行
1.01	2019.11.29	P.4	図 1-1「Bluetooth Mesh ネットワークの基本構成」を追加
		P.5	1.4 節「ステート」を追加
		P.8	1.6 節「メッセージ」を追加
		P.10	図 1-6「ノードのライフサイクル」を追加
		P.40	2.5.1 項「スケジューラ」を追加
		P.57	3.6 節「コンフィグレーション」を追加
		P.52	3.5 節「フレンドシップ」を追加
		P.63	3.8.3 項「Generic OnOff モデルのシーケンス」を追加
		P.63	3.8.4 項「Vendor Model のシーケンス」を追加
		P.66	4 章「Appendix」を追加
1.10	2020.9.30	P.4	1.1 節「ノード」に「ネットワーク」節を統合
		P.4	1.2 節「エレメント」に「アドレス」節を統合
		P.8	1.5 節「メッセージ」に図 1-4 を追加
		P.10	「メッシュデバイスのライフサイクル」節を削除して図 1-6 を追加
		P.11	1.10 節「オプション機能」を追加
		P.15	2.1 節「システム構成」にデモプロジェクトの構成を追加
		P.16	2.2 節「Mesh アプリケーション」にメッシュサンプルプログラムの主な特徴を追加
		P.39	2.2 節「Mesh アプリケーション」に図 3-1 を追加
		P.53	3.5.3 項「ローパワーノードのシーケンス」を追加
		P.65	3.8.5.1 項「Mesh モニタ機能のシーケンス」を追加
		P.58	2.2.2.2 項「コンフィグレーションモデル」に表 5、表 6 と図 24 を追加
		P.63	2.2.2.3 項「Generic OnOff モデルに表 7、表 8 と図 25 を追加
		P.63	2.2.2.4 項「Vendor モデル」を追加
		P.22	2.3 節「Bluetooth Mesh スタック」に表 2-6 を追加
		P.26	2.4.4 項「ADV ベアラー動作」を追加
		P.27	2.4.5 項「GATT ベアラー動作」を追加
		-	「Bluetooth プロトコルスタック」節を削除
		P.30	2.6 節「Mesh サンプルプログラムの設定」を追加
		P.35	2.7 節「Bluetooth ベアラーの設定」を追加
		P.38	2.8 節「Mesh ドライバの設定」を追加
		-	「ベンダー独自 Mesh モデル」節を削除
		全体	一部の節順序、記述、図、表、ソースコードを更新
1.20	2021.9.30	P.6	1.5.2 項「ファウンデーションモデル」を追加
		P.6	1.5.4 項「ヘルスモデル」を追加
		P.39	2 章「Bluetooth Mesh スタックパッケージ」に記載されていたシーケンス図を 3 章「アプリケーション開発」に移動
		P.19	2.2.3.2 項「ヘルスモデル」を追加
		P.25	2.4.3 項「Mesh GATT サービス (gatt_db.c)」を追加
		P.48	3.4 節「プロキシ」を追加
		P.59	3.7 節「ヘルスモデル」を追加
		P.64	3.8.5 項「Mesh モニタ機能」を追加
		全体	一部の節順序、記述、図、表、ソースコードを更新
1.30	2022.12.22	P.14	Target Board for RX23W module 向けデモプロジェクトを追加
		P.14	デモプロジェクト構成に json フォルダを追加
		P.18～P.19	SIG モデルのメッセージ一覧を削除して、参照すべき Mesh 仕様書を追加

		P.31	「コンソール文字列受信設定」の解説を追加
		P.32	2.6.2 項「Provisioning 動作設定」を追加
		P.40	「Bluetooth ベアラーの初期化完了コールバック」で示す Mesh スタック初期化 API を `MS_init()`から `MS_init_ext()`に変更
		P.41	「Mesh スタックの終了処理」の解説を追加
		P.42	「ノードとエレメント」で示すエレメント登録 API を `MS_access_register_element()`から `MS_access_register_element_ext()`に変更
		P.46	OOB Public Key と OOB Authentication の使用方法の解説を更新
		P.43～P.64	Mesh サンプルプログラムから引用したコードを削除して、参照すべき Mesh サンプルプログラムの場所を追加
		全体	文書内のドキュメントリンクを更新
		全体	文書内の一部シーケンス図の誤記を訂正

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1 2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

© 2022 Renesas Electronics Corporation. All rights reserved.