Mesh OTA Application Note

V1.1
2024/01/25

Realtek Semiconductor Corp.
No.2, Innovation Road II, Hsinchu Science Park, Hsinchu 300, Taiwan
Tel.: +886-3-578-0211. Fax: +886-3-577-6047
www.realtek.com

COPYRIGHT

DISCLAIMER

Realtek provides this document 'as is', without warranty of any kind. Realtek may make improvements and/or changes in this document or in the product described in this document at any time. This document could include technical inaccuracies or typographical errors.

TRADEMARKS

Realtek is a trademark of Realtek Semiconductor Corporation. Other names mentioned in this document are trademarks/registered trademarks of their respective owners.

USING THIS DOCUMENT

This document is intended for the software engineer's reference and provides detailed programming information. Though every effort has been made to ensure that this document is current and accurate, more information may have become available subsequent to the production of this guide.

ELECTROSTATIC DISCHARGE (ESD) WARNING

This product can be damaged by Electrostatic Discharge (ESD). When handling, care must be taken. Damage due to inappropriate handling is not covered by warranty.

Do not open the protective conductive packaging until you have read the following, and are at an approved anti-static workstation.

- Use an approved anti-static mat to cover your work surface
- Use a conductive wrist strap attached to a good earth ground
- Always discharge yourself by touching a grounded bare metal surface or approved anti-static mat before picking up an ESD-sensitive electronic component
- If working on a prototyping board, use a soldering iron or station that is marked as ESD-safe
- Always disconnect the microcontroller from the prototyping board when it is being worked on

修订历史

日期	版本	修改	作者	审阅
2018/10/12	Draft v0.1	初稿	bill	
2018/11/16	Draft v0.2	增加裂变升级中产品 ID 使用说明	bill	
2018/12/11	Draft v0.3	增加 mesh ota 部分	bill	
2019/01/24	Draft v0.4	增加 ais ota 部分	bill	
2021/12/09	Draft v0.5	完善 mesh ota 部分	sterling	bill
2022/01/24	Draft v0.6	Mesh OTA 示例部分新增 Initiator 角色	sterling	bill
2023/06/29	V1.0	正式版本	sterling	bill
2024/01/25	V1.1	更新 blob transfer, firmware update, firmware distribution models 的 model ID	sterling	bill

目 录

表目录

图目录

1 简介

Mesh 设备 OTA 过程可以通过 BLE Link 上的 GATT 实现，也可以直接利用 Mesh Networking 实现。基于 GATT 的 OTA 可以利用 LE 链路的高速率，快速对设备进行升级。而基于 Mesh 的 OTA 则可以利用 Mesh 网络的组播特性，同时对多个设备进行升级。

- 基于 GATT 的 OTA，可以采用 Realtek 私有 OTA 技术，方便快捷的升级设备。
- 基于 Mesh 的 OTA，SIG 制定了相关的 Model 支持 OTA，后续进行介绍。

2 OTA over GATT

2.1 逐一 OTA

提供 IOS/Android APK，通过手机、平板等对设备进行逐一升级。

- RTK 私有 OTA
- 阿里 AIS OTA

2.2 裂变 OTA

RTK 私有 OTA 支持 Mesh 设备间互相升级，裂变方式传递。

- 高版本设备自动升级周围低版本设备。
- 低版本设备被升级成高版本设备后，也自动升级周围低版本设备。

裂变 OTA 使用要求。

- 设备需要支持 Master role
- 设备共用一份 Image
- Image 版本必须递增

设备间 OTA 需要配置 config file 中支持的链路数为 2，一条 Master 链路和一条 Slave 链路。

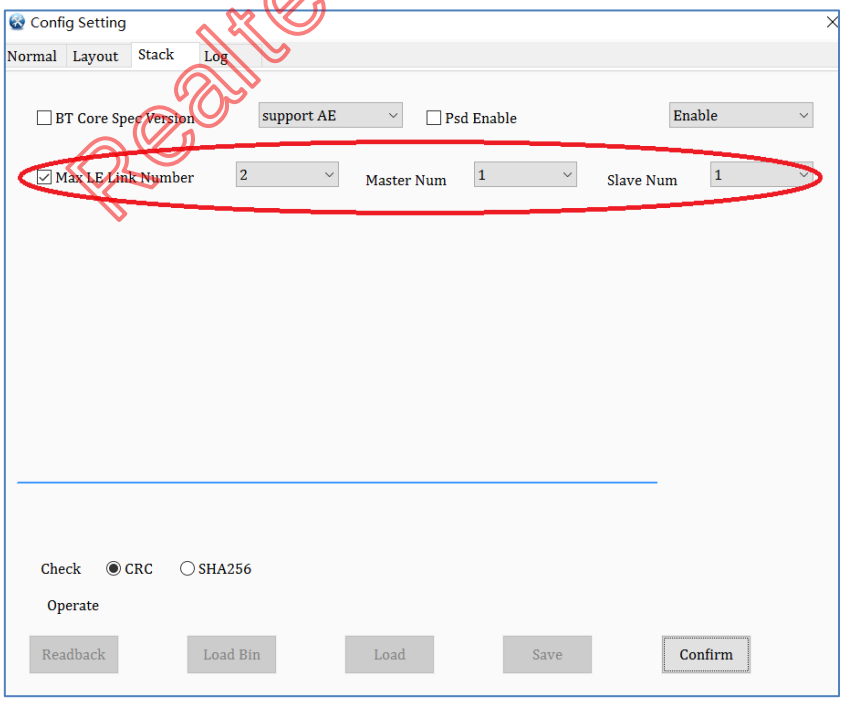

图 2-1 Config file 设置支持链路数

只有 Image 兼容的设备才能互相升级，为此定义 Image 兼容性标志。

- 公司 ID：宏 MANUFACTURE_ADV_DATA_COMPANY_ID
- 产品 ID：宏 DFU_PRODUCT_ID

注意：如果用户没有 SIG 分发的公司 ID，需要使用 Realtek 的公司 ID 0x005d，需要与 Realtek 协商产品 ID 的定义，以免发生产品 ID 冲突而造成不可估量的后果！！

为了保证升级的可控，必须递增 Image 版本。版本定义为宏 DFU_APP_VERSION。

3 OTA over Mesh

SIG 定义了相关 Model 可以利用 Mesh 网络进行 OTA，包含 BLOB (Binary Large Objects) Transfer Server/Client model、Firmware Update Server/Client model 和 Firmware Distribution Server/Client model。BLOB Transfer model 主要用于传输数据，Firmware Update model 和 Firmware Distribution model 主要负责协商和监控固件更新的过程。

SIG OTA 只允许版本递增，修改 DFU_UPDATER_FW_ID 宏递增版本号。mesh ali light 工程中，版本号由 ALI_VERSION_ID 决定。

3.1 固件更新介绍

Mesh 网络中定义了四个角色参与固件更新。

表 3-1 Mesh 固件更新角色

角色	说明
Updating Node（更新节点）	固件需要更新的节点
Initiator（发起者）	检查可用的固件更新，发送给 Distributor
Distributor（分发者）	提供固件给 Updating Node
Stand-alone Updater（独立更新者）	可以直接提供固件给 Updating Node

其中每个角色需要支持的 model 如下。

表 3-2 角色需要支持的 model

角色	Firmware Distribution Client	Firmware Distribution Server	Firmware Update Client	Firmware Update Server	BLOB Transfer Client	BLOB Transfer Server
Updating Node	-	-	-	必需	-	必需
Initiator	必需	-	必需	-	必需	-
Distributor	-	必需	必需	-	必需	必需
Stand-alone Updater	非必需	非必需	必需	-	必需	非必需

其中，Updating Node 角色是需要接收新的固件的节点；Distributor 角色将新的固件传输到需更新的 Updating Node 并监视其传输进度；Initiator 角色主要负责检查是否有可用的更新，将新的固件和必要的参数以及此次 Updating Node 的列表上传给 Distributor；Stand-alone Updater 角色可以直接获取新的固件并传输给 Updating Node。可以看出，Initiator 和 Stand-alone Updater 需要能检查并获取新的固件，一般为 mesh 网关设备或智能手机。在较为简单的应用场景中，Initiator 不是必需的，只需要将新的固件烧录到 Distributor 并设定好相应的参数以及需要更新的节点列表，然后传输该固件到 Updating Node 就可以完成 OTA 过程。

下面简单介绍每个 model 相关的内容，具体协议请参考 SIG 相关 Spec。

3.2 BLOB Transfer Model

BLOB Transfer Model 能传输大于最大访问层 PDU 大小的数据对象，并且能同时多播到多个节点。利用 Mesh 进行 OTA 的过程使用 BLOB Transfer Model 将新固件传输到能够接收它的所有节点。

3.2.1 状态

BLOB Transfer Model 使用的状态分为以下三类，第一类是 Server/Client 共用的状态，第二类是 Server 单独使用的状态，第三类是 Client 单独使用的状态。

3.2.1.1 Server/Client 共用的状态

本小节描述 BLOB Transfer Server/Client 共用的状态。

3.2.1.1.1 BLOB

BLOB 状态是一种复合状态，用于标识和表征正在传输的 BLOB，包括以下状态。

表 3-3 BLOB 状态

BLOB 状态	说明
BLOB ID 状态	用于标识网络中的 BLOB
BLOB Size 状态	表示 BLOB 的大小
Block Size Log 状态	计算传输中使用的 block 的大小
BLOB Data 状态	BLOB 数据
Total Blocks 状态	BLOB 被划分成 blocks 的总数

3.2.1.1.2 BLOB Block

BLOB 块状态是一种复合状态，用于标识正在传输的 block。BLOB block 状态包括以下状态。

表 3-4 BLOB block 状态

BLOB block 状态	说明
BLOB Number 状态	用于标识 BLOB 中的某一块 block
Current BLOB Size 状态	表示正在传输的当前 BLOB 的大小
Chunk Size 状态	表示某一块 block 中 chunk 的大小
Total Chunks 状态	表示某一块 block 中 chunk 的数量
BLOB Block Data 状态	表示某一块 block 的数据

3.2.1.1.3 Transfer TTL

Transfer TTL 状态表示从 BLOB Transfer Client 或 Server 发起网络 PDU 时使用的 TTL(Time to Live)值。

3.2.1.1.4 Transfer Mode

Transfer Mode 状态表示 BLOB 传输的模式，主要分为 Push 模式和 Pull 模式。

3.2.1.2 Server 状态

本小节描述 BLOB Transfer Server 单独使用的状态。

3.2.1.2.1 Transfer Phase

Transfer Phase 用于标识 BLOB Transfer Server 的状态。

3.2.1.2.2 Expected BLOB ID

Expected BLOB ID 状态用于标识 BLOB Transfer Server 期望接收到的 BLOB ID。

3.2.1.2.3 Blocks Not Received

Blocks Not Received 状态表示 Server 尚未接收到的 blocks。

3.2.1.2.4 Missing Chunks

Missing Chunks 状态表示 Server 尚未接收到并正在传输的 chunks。

3.2.1.2.5 Client MTU Size

Client MTU Size 状态表示发起 BLOB 传输过程的 Client 的 MTU(Maximum Transfer Unit)的大小。

3.2.1.2.6 Server Timeout Base

Server Timeout Base 状态用于计算 Server 在等待来自 Client 的消息的时间。

3.2.1.2.7 Capabilities

Capabilities 状态是一个复合状态，表示 BLOB Transfer Server model 实例的传输能力。Capabilities 状态包括以下状态。

表 3-5 BLOB Transfer Server model Capabilities 状态

Capabilities 状态	说明
Min Block Size Log 状态	表示 Server 可以接收的最小 block 大小
Max Block Size Log 状态	表示 Server 可以接收的最大 block 大小
Max Total Chunks 状态	表示一个 block 可以分成的最大的 chunk 数量
Max Chunk Size 状态	表示 Server 支持的最大 chunk 大小
Max BLOB Size 状态	表示 Server 可以接收的最大 BLOB 大小
Server MTU Size 状态	表示 Server 支持的 MTU 大小
Supported Transfer Mode 状态	表示 Server 支持的传输模式

3.2.1.3 Client 状态

本小节描述 BLOB Transfer Client 单独使用的状态。

3.2.1.3.1 BLOB Receivers

BLOB Receivers 状态用于存储参与传输的 BLOB Transfer Servers 列表，列表里的条目表示传输到 Server 的状态。

3.2.1.3.2 Active BLOB Receivers

Active BLOB Receivers 状态用于存储积极参与传输的 BLOB Transfer Servers 列表（例如，未超时），列表里的条目表示传输到 Server 的状态。

3.2.1.3.3 BLOB Receivers Capabilities

BLOB Receivers Capabilities 状态用于存储参与传输的 BLOB Transfer Servers 列表，列表里的条目表示 Server 的 Capabilities 状态。

3.2.1.3.4 BLOB Multicast Address

BLOB Multicast Address 状态包含组播地址、虚拟地址的 UUID 或者未分配地址。

3.2.1.3.5 AppKey Index

AppKey Index 状态表示 Client 与 BLOB Transfer Server 交换消息时使用的 AppKey 的 index。

3.2.1.3.6 Client Timeout Base

Client Timeout Base 状态用于计算 Client 等待来自 Server 消息的时间。

3.2.2 SIG 消息

3.2.2.1 BLOB Transfer Get

BLOB Transfer Get 消息用于获取 BLOB Transfer Server 上的 BLOB Transfer 状态，Server 回应 BLOB Transfer Status。

3.2.2.2 BLOB Transfer Start

BLOB Transfer Start 消息用于启动新的 BLOB 传输，Server 回应 BLOB Transfer Status。

3.2.2.3 BLOB Transfer Cancel

BLOB Transfer Cancel 消息用于取消正在进行的 BLOB 传输，Server 回应 BLOB Transfer Status。

3.2.2.4 BLOB Transfer Status

BLOB Transfer Status 消息用于报告 BLOB Transfer Server 的状态，无应答。

3.2.2.5 BLOB Block Get

BLOB Block Get 消息用于检索 Transfer 的阶段和找出正在传输的 Block，Server 回应 BLOB Block Status。

3.2.2.6 BLOB Block Start

BLOB Block Start 消息用于开始传输 block 到 Server 的过程，Server 回应 BLOB Block Status。

3.2.2.7 BLOB Block Status

BLOB Block Status 消息用于报告正在传输的 block 的状态，无应答。

3.2.2.8 BLOB Partial Block Report

BLOB Partial Block Report 消息被 Server 用于向 Client 请求 chunks，无应答。

3.2.2.9 BLOB Chunk Transfer

BLOB Chunk Transfer 消息用于将当前 block 的 chunk 传输到 BLOB Transfer Server，无应答。

3.2.2.10 BLOB Information Get

BLOB Information Get 消息用于获取 BLOB Transfer Server 的 Capabilities 状态，Server 回应 BLOB Information Status。

3.2.2.11 BLOB Information Status

BLOB Information Status 消息用于报告 BLOB Transfer Server 的 Capabilities 状态，无应答。

3.2.3 BLOB Transfer Server 接口

表 3-6 BLOB Transfer Server model 接口

函数	说明
blob_transfer_server_reg	注册 BLOB Transfer Server model
blob_transfer_server_set_data_cb	设置 BLOB Transfer Server model 回调函数
blob_transfer_server_caps_set	设置 BLOB Transfer Server 的 Capabilities
blob_transfer_server_caps_get	获取 BLOB Transfer Server 的 Capabilities
blob_transfer_server_init	初始化 BLOB Transfer Server
blob_transfer_server_load	加载 BLOB Transfer 程序
blob_transfer_server_clear	清除 BLOB Transfer Server
blob_transfer_server_phase_get	获取 BLOB Transfer Server 的当前阶段
blob_transfer_server_busy	查询 BLOB Transfer Server 是否繁忙
blob_transfer_handle_transfer_timeout	BLOB Transfer 传输超时函数
blob_transfer_handle_partial_report_timeout	BLOB Transfer 部分 Block 报告超时函数
blob_transfer_status	回应 BLOB Transfer 状态
blob_block_status	回应 BLOB Block 状态
blob_partial_block_report	发送 BLOB 部分 Block 报告
blob_info_status	回应 BLOB Information 状态

3.2.4 BLOB Transfer Client 接口

表 3-7 BLOB Transfer Client Model 接口和消息

函数	说明
blob_transfer_client_reg	注册 BLOB Transfer Client Model
blob_transfer_get	获取 Server 上的 BLOB Transfer 状态
blob_transfer_start	启动新的 BLOB 传输
blob_transfer_cancel	取消正在进行的 BLOB 传输
blob_block_get	检索 Transfer 的阶段和找出正在传输的 Block
blob_block_start	开始传输 Block
blob_chunk_transfer	传输当前 Block 的 Chunks
blob_info_get	获取 Server 的 Capabilities 状态

3.2.5 示例

使用 BLOB Transfer model 时主要有以下步骤。

1. 注册 Model
2. 实现 model_data_cb 函数

Initiator 与 Distributor 之间，Distributor 与 Updating Node 之间均需要实现该 model 来支持数据传输，示例介绍了 Distributor 与 Updating Node 之间的 BLOB Transfer model 实现框架，具体实现请参考 dfu_updater_app.c 和 dfu_distributor_app.c。

3.2.5.1 Server Model 示例

BLOB Transfer Server model model_data_cb 函数示例代码如下。

```
int32_t fw_update_handle_blob_server_data(const mesh_model_info_p pmodel_info,
                                          uint32_t type, void *pargs)
{
   switch (type)
   {
case BLOB_TRANSFER_SERVER_BLOCK_DATA:
{
/* Handle BLOB_TRANSFER_SERVER_BLOCK_DATA */
}
      break;
case BLOB_TRANSFER_SERVER_COMPLETE:
{
/* Handle BLOB_TRANSFER_SERVER_COMPLETE */
}
      break;
   case BLOB_TRANSFER_SERVER_CANCEL:
{
/* Handle BLOB_TRANSFER_SERVER_CANCEL */
}
      break;
   case BLOB_TRANSFER_SERVER_SUSPEND:
{
/* Handle BLOB_TRANSFER_SERVER_SUSPEND */
}
      break;
case BLOB_TRANSFER_SERVER_FAIL:
```

```
26.   {
27.   /* Handle BLOB_TRANSFER_SERVER_FAIL */
28.   }
29.         break;
30.   default:
31.         break;
32.     }
33.     return MODEL_SUCCESS;
34.   }
```

注册 BLOB Transfer Server model 代码如下。

```
1.    /* register blob transfer server model */
2.    blob_transfer_server_reg(0, fw_update_handle_blob_server_data);
```

3.2.5.2 Client Model 示例

BLOB Transfer Client model model_data_cb 函数示例代码如下。

```
1.    int32_t dfu_transfer_client_data(const mesh_model_info_p pmodel_info,
                                       uint32_t type, void *pargs)
2.    {
3.      switch (type)
4.      {
5.      case BLOB_TRANSFER_CLIENT_TRANSFER_STATUS:
6.        {
7.         /* Handle BLOB_TRANSFER_CLIENT_TRANSFER_STATUS */
8.        }
9.        break;
10.     case BLOB_TRANSFER_CLIENT_BLOCK_STATUS:
11.       {
12.        /* Handle BLOB_TRANSFER_CLIENT_BLOCK_STATUS */
13.       }
14.       break;
15.     case BLOB_TRANSFER_CLIENT_PARTIAL_BLOCK_REPORT:
16.       {
17.        /* Handle BLOB_TRANSFER_CLIENT_PARTIAL_BLOCK_REPORT */
18.       }
19.       break;
20.     case BLOB_TRANSFER_CLIENT_INFO_STATUS:
21.       {
22.        /* Handle BLOB_TRANSFER_CLIENT_INFO_STATUS */
```

```
        }
        break;
    default:
        break;
    }
    return MODEL_SUCCESS;
}
```

注册 BLOB Transfer Client model 代码如下。

```
/* register blob transfer client model */
blob_transfer_client_reg(0, dfu_transfer_client_data);
```

3.3 Firmware Update Model

Firmware Update Model 可以检索固件的信息，判断节点是否可以接收新的固件，开始固件传输以及应用固件。传输固件的过程则依赖 BLOB Transfer Model 实现。

3.3.1 状态

3.3.1.1 Server 状态

3.3.1.1.1 Firmware Information List

Firmware Information List 状态是 Updating Node 上安装的每个固件的 Update URIs 和 Firmware IDs 的列表。在节点成功应用新的固件更新后，此状态会发生变化。

表 3-8 Firmware Information List 状态格式

内容	描述
Current Firmware ID	标识安装在 Updating Node 上的固件
Update URI	新固件存档文件的位置

3.3.1.1.2 Update Phase

Update Phase 状态用于标识 Firmware Update Server 的更新阶段。

3.3.1.1.3 Firmware Update Additional Information

Firmware Update Additional Information 状态表示固件更新附加信息。

3.3.1.1.4 Update Firmware Image Index

Update Firmware Image Index 状态用于标识 Firmware Information List 中正在更新的固件。

3.3.1.1.5 New Firmware Image

New Firmware Image 状态为固件的二进制 image。

3.3.1.1.6 Update BLOB ID

Update BLOB ID 状态表示 BLOB Transfer Server 在固件传输中使用的 BLOB ID 值。

3.3.1.1.7 Update Firmware Metadata

Update Firmware Metadata 状态表示传入固件的元数据。

3.3.1.1.8 Update TTL

Update TTL 状态表示固件更新期间 BLOB Transfer Server 使用的 TTL 值。

3.3.1.1.9 Update Server Timeout Base

Update Server Timeout Base 状态表示 Firmware Update Server 暂停固件传输接收后的超时时间。

3.3.1.2 Client 状态

3.3.1.2.1 Update Receivers

Update Receivers 状态如下。

表 3-9 Update Receivers 状态格式

内容	描述
Address	Updating Node 的单播地址
Firmware Image Index	Updating Node 的固件的 index
Retrieved Update Phase	检索到的 Updating Node 的更新阶段状态
Status	Updating Node 上次操作后的状态
Update Additional Information	检索到的 Updating Node 的固件更新附加信息

3.3.1.2.2 Active Update Receivers

Active Update Receivers 状态用于识别积极参与更新的 Firmware Update Server。

3.3.1.2.3 Update Multicast Address

Update Multicast Address 状态包含组播地址、虚拟地址的 UUID 和未分配的地址。

3.3.1.2.4 Update AppKey Index

Update AppKey Index 状态表示 AppKey 的 index。

3.3.1.2.5 Update TTL

Update TTL 状态表示固件更新期间 BLOB Transfer Client 使用的 TTL 值。

3.3.1.2.6 Update Client Timeout Base

Update Client Timeout Base 状态用于计算发送到 Firmware Update Server 的响应超时。

3.3.2 SIG 消息

3.3.2.1 Firmware Update Information Get

Firmware Update Information Get 消息用于获取安装在节点上的固件的信息，Server 回应 Firmware Update Information Status。

3.3.2.2 Firmware Update Information Status

Firmware Update Information Status 消息用于报告有关安装在节点上的固件的信息，无应答。

3.3.2.3 Firmware Update Firmware Metadata Check

Firmware Update Firmware Metadata Check 消息用于检查节点能否接收固件更新，Server 回应 Firmware Update Firmware Metadata Status。

3.3.2.4 Firmware Update Firmware Metadata Status

Firmware Update Firmware Metadata Status 消息用于报告是否可以接受固件更新，无应答。

3.3.2.5 Firmware Update Get

Firmware Update Get 消息用于获取 Server 的当前状态，Server 回应 Firmware Update Status。

3.3.2.6 Firmware Update Start

Firmware Update Start 消息用于启动固件更新，Server 回应 Firmware Update Status。

3.3.2.7 Firmware Update Cancel

Firmware Update Cancel 消息用于取消固件更新并删除有关固件更新的存储信息，Server 回应 Firmware Update Status。

3.3.2.8 Firmware Update Apply

Firmware Update Apply 消息用于应用已经传输完成的固件，Server 回应 Firmware Update Status。

3.3.2.9 Firmware Update Status

Firmware Update Status 用于报告固件更新状态，无应答。

3.3.3 Firmware Update Server 接口

表 3-10 Firmware Update Server model 接口

函数	说明
fw_update_server_reg	注册 Firmware Update Server Model
fw_update_server_load	加载 Firmware Update Server
fw_update_server_add_info	添加固件更新信息
fw_update_server_set_verify_result	设置固件验证结果
fw_update_server_clear	清除 Firmware Update Server
fw_update_handle_blob_server_data	BLOB Transfer Server 回调函数

3.3.4 Firmware Update Client 接口

表 3-11 Firmware Update Client model 接口

函数	说明
fw_update_client_reg	注册 Firmware Update Client Model
fw_update_info_get	获取安装在节点上的固件的信息
fw_update_fw_metadata_check	检查节点能否接收固件更新
fw_update_get	获取 Server 的当前状态
fw_update_start	启动固件更新
fw_update_cancel	取消固件更新
fw_update_apply	应用已经传输完成的固件

3.3.5 示例

使用 Firmware Update model 时主要有以下步骤。

1. 注册 Model
2. 实现 model_data_cb 函数

示例介绍了 Distributor 与 Updating Node 之间的 Firmware Update model 实现框架，具体实现请参考 dfu_updater_app.c 和 dfu_distributor_app.c。

3.3.5.1 Server Model 示例

Firmware Update Server model model_data_cb 函数示例代码如下。

```
1.  int32_t dfu_update_server_data(const mesh_model_info_p pmodel_info,
                                   uint32_t type, void *pargs)
2.  {
3.      switch (type)
4.      {
5.      case FW_UPDATE_SERVER_METADATA_CHECK:
6.          {
7.           /* Handle FW_UPDATE_SERVER_METADATA_CHECK */
8.          }
9.          break;
```

```
    case FW_UPDATE_SERVER_START:
      {
       /* Handle FW_UPDATE_SERVER_START */
      }
      break;
    case FW_UPDATE_SERVER_VERIFY:
      {
       /* Handle FW_UPDATE_SERVER_VERIFY */
      }
      break;
    case FW_UPDATE_SERVER_VERIFY_CANCEL:
      {
       /* Handle FW_UPDATE_SERVER_VERIFY_CANCEL */
      }
      break;
    case FW_UPDATE_SERVER_APPLY:
      {
       /* Handle FW_UPDATE_SERVER_APPLY */
      }
      break;
    case FW_UPDATE_SERVER_BLOCK_DATA:
      {
       /* Handle FW_UPDATE_SERVER_BLOCK_DATA */
      }
      break;
    case FW_UPDATE_SERVER_FAIL:
      {
       /* Handle FW_UPDATE_SERVER_FAIL */
      }
      break;
    default:
      break;
    }
    return MODEL_SUCCESS;
}
```

注册 Firmware Update Server model 代码如下。

```
/* register firmware update server model */
fw_update_server_reg(0, dfu_update_server_data);
```

3.3.5.2 Client Model 示例

Firmware Update Client model model_data_cb 函数示例代码如下。

```
1.  int32_t dfu_update_client_data(const mesh_model_info_p pmodel_info,
                                   uint32_t type, void *pargs)
2.  {
3.      switch (type)
4.      {
5.      case FW_UPDATE_CLIENT_INFO_STATUS:
6.          {
7.           /* Handle FW_UPDATE_CLIENT_INFO_STATUS */
8.          }
9.          break;
10.     case FW_UPDATE_CLIENT_FW_METADATA_STATUS:
11.         {
12.          /* Handle FW_UPDATE_CLIENT_FW_METADATA_STATUS */
13.         }
14.         break;
15.     case FW_UPDATE_CLIENT_STATUS:
16.         {
17.          /* Handle FW_UPDATE_CLIENT_STATUS */
18.         }
19.         break;
20.     default:
21.         break;
22.     }
23.     return MODEL_SUCCESS;
24. }
```

注册 Firmware Update Client model 代码如下。

```
1.  /* register firmware update client model */
2.  fw_update_client_reg(0, dfu_update_client_data);
```

3.4 Firmware Distribution Model

Firmware Distribution Model 主要负责检查更新、上传 Updating Nodes list 和固件、启动和应用固件分发过程。上传固件的过程则依赖 BLOB Transfer Model 实现。

3.4.1 状态

3.4.1.1 Server 状态

3.4.1.1.1 Distribution Receivers

Distribution Receivers 状态是一个复合状态，包含的状态如下。

表 3-12 Distribution Receivers 状态

Distribution Receivers 状态	说明
Distribution Receivers List 状态	标识 Updating Nodes 和目标固件的列表
Distribution Receivers List Count 状态	Distribution Receivers List 状态的数量

3.4.1.1.2 Distributor Capabilities

Distributor Capabilities 是一个复合状态，表示 Firmware Distribution Server model 支持的固件更新能力。

表 3-13 Firmware Distribution Server model Capabilities 状态

Capabilities 状态	说明
Max Distribution Receivers List Size 状态	表示 Firmware Distribution Server 在 Distribution Receivers List 状态中可以存储的最大地址数
Max Firmware Images List Size 状态	表示 Firmware Distribution Server 在 Firmware Images List 状态中可以存储的最大固件数
Max Firmware Image Size 状态	表示存储在 Firmware Distribution Server 的单个固件的最大大小
Max Upload Space Size 状态	表示 Firmware Distribution Server 存储固件的总大小
Remaining Upload Space Size 状态	表示 Firmware Distribution Server 存储固件的可用空间
Out-of-Band Retrieval Supported 状态	表示 Firmware Distribution Server 是否支持通过 OOB 检索固件
Supported URI Scheme Names 状态	表示支持的 URI 方案的列表

3.4.1.1.3 Distribution Parameters

Distribution Parameters 状态是包含 Firmware Distribution Server model 在分发固件期间使用的参数的复合状态。

表 3-14 Distribution Parameters 状态

Distribution Parameters 状态	说明
Distribution Phase 状态	表示 Firmware Distribution Server 正在执行固件分发的阶段
Update Policy 状态	表示何时应用新的固件
Distribution Multicast Address 状态	表示目标地址
Distribution AppKey Index 状态	表示使用的 AppKey 的 index
Distribution TTL 状态	表示 Firmware Distribution Server 使用的 TTL 值
Distribution Timeout Base 状态	用于计算在固件分发时 Firmware Update Client 和 BLOB Transfer Client 的超时值
Distribution Transfer Mode 状态	表示固件分发的传输模式
Distribution Firmware Image Index 状态	标识正在分发的固件

3.4.1.1.4 Upload Parameters

Upload Parameters 状态是一个复合状态，其中包含 Firmware Distribution Server model 在上传固件到 Distributor 期间使用的参数。

表 3-15 Upload Parameters 状态

Upload Parameters 状态	说明
Upload Firmware ID 状态	标识正在上传到 Distributor 的固件
Upload Phase 状态	标识固件上传阶段
Upload Firmware Size 状态	表示正在上传的固件的大小
Upload BLOB ID 状态	表示上传期间使用的 BLOB ID
Upload Firmware Metadata 状态	表示正在上传的固件的元数据
Upload URI 状态	用于计算上传期间的超时值

3.4.1.1.5 Firmware Images List

Firmware Images List 状态包含存储在 Firmware Distribution Server 上的每个固件的信息。

表 3-16 Firmware Images List 状态格式

内容	说明
Index	列表中的位置
Firmware ID	标识固件
Firmware Metadata	Firmware ID 标识的固件的元数据

3.4.1.2 Client 状态

3.4.1.2.1 Update Information From Updating Nodes

Update Information From Updating Nodes 状态是一个列表，用于标识 Firmware Distribution Client 提供的，安装在每个 Updating Nodes 的固件。

表 3-17 Update Information From Updating Nodes 状态格式

内容	说明
Address	Updating Node 的单播地址
Firmware Images List	Firmware Images Entry 的列表

每个 Firmwre Images Entry 状态的格式如下。

表 3-18 Firmware Images Entry 状态格式

内容	说明
Current Firmware ID Length	当前固件 ID 字段长度
Current Firmware ID	Updating Node 上最近报告的固件版本
New Firmware ID Length	新固件 ID 字段的长度
New Firmware ID	Updating Node 可用的新固件
Update URI Length	URI 字段的长度
Update URI	用于定位新固件的 URI
Firmware Image Length	固件字段的长度
Firmware Image	固件

3.4.2 SIG 消息

3.4.2.1 Firmware Distribution Receivers Add

Firmware Distribution Receivers Add 消息用于添加新的条目到 Firmware Distribution Server 的 Distribution Receivers List 状态，Server 回应 Firmware Distribution Receivers Status。

3.4.2.2 Firmware Distribution Receivers Delete All

Firmware Distribution Receivers Delete All 消息用于删除 Firmware Distribution Server 的 Distribution Receivers List 状态上的所有条目，无应答。

3.4.2.3 Firmware Distribution Receivers Status

Firmware Distribution Receivers Status 消息用于报告 Distribution Receivers List 状态的当前大小，无应答。

3.4.2.4 Firmware Distribution Receivers Get

Firmware Distribution Receivers Get 消息用于获取每个 Updating Node 的固件分发状态，Server 回应 Firmware Distribution Receivers List。

3.4.2.5 Firmware Distribution Receivers List

Firmware Distribution Receivers List 消息用于报告每个 receiver 的固件分发状态，无应答。

3.4.2.6 Firmware Distribution Capabilities Get

Firmware Distribution Capabilities Get 消息用于获取 Firmware Distribution Server 的 Capabilities，Server 回应 Firmware Distribution Capabilities Status。

3.4.2.7 Firmware Distribution Capabilities Status

Firmware Distribution Capabilities Status 用于报告 Distributor 的 Capabilities，无应答。

3.4.2.8 Firmware Distribution Get

Firmware Distribution Get 消息用于获取 Firmware Distribution Server 上当前固件分发的状态，Server 回应 Firmware Distribution Status。

3.4.2.9 Firmware Distribution Start

Firmware Distribution Start 消息用于启动固件分发，Server 回应 Firmware Distribution Status。

3.4.2.10 Firmware Distribution Cancel

Firmware Distribution Cancel 消息用于取消固件分发，Server 回应 Firmware Distribution Status。

3.4.2.11 Firmware Distribution Apply

Firmware Distribution Apply 消息用于令 Updating Nodes 应用新固件，Server 回应 Firmware Distribution Status。

3.4.2.12 Firmware Distribution Status

Firmware Distribution Status 消息用于报告固件分发状态，无应答。

3.4.2.13 Firmware Distribution Upload Get

Firmware Distribution Upload Get 消息用于检查固件上传到 Firmware Distribution Server 的状态，Server 回应 Firmware Distribution Upload Status。

3.4.2.14 Firmware Distribution Upload Start

Firmware Distribution Upload Start 消息用于启动固件上传，Server 回应 Firmware Distribution Upload Status。

3.4.2.15 Firmware Distribution Upload OOB Start

Firmware Distribution Upload OOB Start 消息用于启用 OOB 的方式进行固件上传，Server 回应 Firmware Distribution Upload Status。

3.4.2.16 Firmware Distribution Upload Cancel

Firmware Distribution Upload Cancel 消息用于停止固件上传，Server 回应 Firmware Distribution Upload Status。

3.4.2.17 Firmware Distribution Upload Status

Firmware Distribution Upload Status 消息用于报告固件上传的状态，无应答。

3.4.2.18 Firmware Distribution Firmware Get

Firmware Distribution Firmware Get 消息用于检查某个特定的固件是否存储在 Firmware Distribution Server 上，Server 回应 Firmware Distribution Firmware Status。

3.4.2.19 Firmware Distribution Firmware Get By Index

Firmware Distribution Firmware Get By Index 消息用于按照 Firmware Images List 状态上特定的 index 检查哪个固件存储在 Firmware Distribution Server 上，Server 回应 Firmware Distribution Firmware Status。

3.4.2.20 Firmware Distribution Firmware Delete

Firmware Distribution Firmware Delete 消息用于删除存储在 Firmware Distribution Server 上的某一个固件，Server 回应 Firmware Distribution Firmware Status。

3.4.2.21 Firmware Distribution Firmware Delete All

Firmware Distribution Firmware Delete All 消息用于删除存储在 Firmware Distribution Server 上的全部固件，Server 回应 Firmware Distribution Firmware Status。

3.4.2.22 Firmware Distribution Firmware Status

Firmware Distribution Firmware Status 用于报告对存储的固件的操作后的状态，无应答。

3.4.3 Firmware Distribution Server 接口

表 3-19 Firmware Distribution Server model 接口

函数	说明
fw_dist_server_reg	注册 Firmware Distribution Server Model
fw_dist_server_add_receiver	添加 receiver 到 Firmware Distribution Server
fw_dist_server_is_receiver_empty	检查 receiver 是否为空
fw_dist_server_delete_all_receiver	删除所有的 receiver
fw_dist_server_add_image	添加 image 到 Firmware Images List
fw_dist_server_delete_all_image	删除 Firmware Images List 的所有 image
fw_dist_server_delete_image	删除 Firmware Images List 的某个 image
fw_dist_server_cancel_done	取消分发过程完成
fw_dist_handle_blob_server_data	BLOB Transfer Server 回调函数
fw_dist_server_start	开始固件分发
fw_dist_server_dist_failed	通知分发失败
fw_dist_server_set_upload_recvd_size	设置可接收上传固件大小
fw_dist_server_upload_oob_done	使用 OOB 上传固件完成
fw_dist_server_set_updating_status	设置节点更新状态

3.4.4 Firmware Distribution Client 接口

表 3-20 Firmware Distribution Client model 接口

函数	说明
fw_dist_client_reg	注册 Firmware Update Client model
fw_dist_recvs_add	为 Firmware Distribution Server 添加 receivers
fw_dist_recvs_delete_all	删除 Server 所有的 receivers
fw_dist_recvs_get	获取每个 Updating Node 的固件分发状态
fw_dist_caps_get	获取 Server 的 Capabilities
fw_dist_get	获取 Server 上当前固件分发的状态
fw_dist_start	启动固件分发
fw_dist_cancel	停止固件分发
fw_dist_apply	令 Updating Nodes 应用新固件
fw_dist_upload_get	检查固件上传到 Server 的状态
fw_dist_upload_start	启动固件上传
fw_dist_upload_oob_start	启用 OOB 的方式进行固件上传

函数	说明
fw_dist_upload_cancel	停止固件上传
fw_dist_fw_get	检查某个特定的固件是否存储在 Server 上
fw_dist_fw_get_by_index	按照 Firmware Images List 状态上特定的 index 检查哪个固件存储在在 Server 上
fw_dist_fw_delete	删除存储在 Server 上的某一个固件
fw_dist_fw_delete_all	删除存储在 Server 上的全部固件

3.4.5 示例

使用 Firmware Distribution model 时主要有以下步骤。

1. 注册 Model
2. 实现 model_data_cb 函数

示例介绍了 Distributor 与 Initiator 之间的 Firmware Distribution model 实现框架。具体实现请参考 dfu_distributor_app.c 和 dfu_initiator_app.c。

3.4.5.1 Server Model 示例

Firmware Distribution Server model model_data_cb 函数示例代码如下。

```
int32_t fw_dist_server_data(const mesh_model_info_p pmodel_info,
                            uint32_t type, void *pargs)
{
    switch (type)
    {
    case FW_DIST_SERVER_START:
        {
         /* Handle FW_DIST_SERVER_START */
        }
        break;
    case FW_DIST_SERVER_CANCEL:
        {
         /* Handle FW_DIST_SERVER_CANCEL */
        }
        break;
    case FW_DIST_SERVER_APPLY:
```

```
16.         {
17.          /* Handle FW_DIST_SERVER_APPLY */
18.         }
19.         break;
20.       case FW_DIST_SERVER_UPLOAD_START:
21.         {
22.          /* Handle FW_DIST_SERVER_UPLOAD_START */
23.         }
24.         break;
25.       case FW_DIST_SERVER_UPLOAD_OOB_START:
26.         {
27.          /* Handle FW_DIST_SERVER_UPLOAD_OOB_START */
28.         }
29.         break;
30.       case FW_DIST_SERVER_UPLOAD_BLOCK_DATA:
31.         {
32.          /* Handle FW_DIST_SERVER_UPLOAD_BLOCK_DATA */
33.         }
34.         break;
35.       case FW_DIST_SERVER_UPLOAD_COMPLETE:
36.         {
37.          /* Handle FW_DIST_SERVER_UPLOAD_COMPLETE */
38.         }
39.         break;
40.       case FW_DIST_SERVER_UPLOAD_FAIL:
41.         {
42.          /* Handle FW_DIST_SERVER_UPLOAD_FAIL */
43.         }
44.         break;
45.       case FW_DIST_SERVER_FW_DELETE:
46.         {
47.          /* Handle FW_DIST_SERVER_FW_DELETE */
48.         }
49.         break;
50.       case FW_DIST_SERVER_FW_DELETE_ALL:
51.         {
52.          /* Handle FW_DIST_SERVER_FW_DELETE_ALL */
53.         }
54.         break;
55.       case FW_DIST_SERVER_URI_CHECK:
56.         {
57.          /* Handle FW_DIST_SERVER_URI_CHECK */
58.         }
59.         break;
60.       default:
```

```
61.         break;
62.       }
63.       return MODEL_SUCCESS;
64.   }
```

注册 Firmware Distribution Server model 代码如下。

```
1.    /* register firmware distribution server model */
2.    fw_dist_server_reg(0, dfu_update_dist_data);
```

3.4.5.2 Client Model 示例

Firmware Distribution Client model model_data_cb 函数示例代码如下。

```
1.    int32_t fw_dist_client_data (const mesh_model_info_p pmodel_info,
                                   uint32_t type, void *pargs)
2.    {
3.       switch (type)
4.    {
5.       case FW_DIST_CLIENT_RECVS_STATUS:
6.          {
7.          /* Handle FW_DIST_CLIENT_RECVS_STATUS */
8.          }
9.          break;
10.      case FW_DIST_CLIENT_RECVS_LIST:
11.         {
12.         /* Handle FW_DIST_CLIENT_RECVS_LIST */
13.         }
14.         break;
15.      case FW_DIST_CLIENT_CAPS_STATUS:
16.         {
17.         /* Handle FW_DIST_CLIENT_CAPS_STATUS */
18.         }
19.         break;
20.      case FW_DIST_CLIENT_STATUS:
21.         {
22.         /* Handle FW_DIST_CLIENT_STATUS */
23.         }
24.         break;
25.      case FW_DIST_CLIENT_UPLOAD_STATUS:
26.         {
27.         /* Handle FW_DIST_SERVER_UPLOAD_STATUS */
```

```
        }
        break;
    case FW_DIST_CLIENT_FW_STATUS:
        {
         /* Handle FW_DIST_CLIENT_FW_STATUS */
        }
        break;
    default:
        break;
    }
    return MODEL_SUCCESS;
}
```

注册 Firmware Distribution Client model 代码如下。

```
/* register firmware distribution client model */
fw_dist_client_reg(0, dfu_dist_client_data);
```

3.5 Mesh OTA 示例

使用 mesh_provisioner 工程和 mesh_device 工程可以实现 Mesh OTA 过程，其中 mesh_provisioner 可以同时承担 Initiator 和 Distributor 角色，mesh_device 可以同时承担 Distributor 和 Updating Node 角色。同一个节点，多角色可以共存，可以根据场景需求做角色分配来进行测试。

3.5.1 Distributor - Updating Node

在较为简单的应用场景中可以减少 Initiator 角色，Distributor 角色给 Updating Node 角色传输固件即可。Initiator 的部分功能通过串口 CLI 命令行或者调用部分 API 方式来实现，同时新固件也需要提前烧录到 Distributor 的 Flash 中。RTK 的带有 MP 的 bin 文件组成为 MP Header（512Bytes）+Image Header（1K）+Payload，其中 MP Header 仅用作烧录。以烧录 image 的方式将 bin 到对应的区域后只剩下 Image Header 和 Payload，以 user data 的形式将 bin 文件固化在 flash 中则还会保留 MP Header。Mesh OTA 过程只需要 Image Header 和 Payload，如果以 user data 形式固化新的固件，则传输时需要将 size-512，同时首地址向后偏移 512Bytes。Flash 布局和 Image 构成的具体内容请参考《RTL87x2x OTA User Manual》。

3.5.1.1 命令

下面介绍 Distributor 角色与 OTA 过程相关的 CLI 命令。

表 3-21 Distributor 角色 CLI 命令

Cmd	说明
dfua	在 Distributor 升级节点列表中新增一个节点
dfud	删除 Distributor 升级节点列表中所有节点
dfus	启动固件升级过程

下面对每一个 Cmd 的参数含义进行解释，dfua 参数介绍如下。

表 3-22 dfua 参数介绍

dfua	说明
address	节点的单播地址
image index	image 的 index

dfua 参数中的 image index，是根据 mesh_device 的 Firmware Information List 状态里的添加顺序进行检索。往该 List 添加固件信息的 api 为 fw_update_server_add_info，默认只包含了 RTK 的固件信息，image index 为 0 则标识为 RTK 固件。如果没有新增固件信息，则 image index 参数应设置为 0。

dfud 不包含任何参数。

dfus 所需参数如下。

表 3-23 dfus 参数介绍

dfus	说明
dist_dst	分发目标地址，多播地址
dist_app_key_index	app key 的 index
update_timeout_base	Updating Nodes 计算接收数据的超时时间
fw_image_size	Image 的大小，单位为 Bytes
fw_image_start_addr	指向 Image Header 的首地址
update policy	Update policy，该参数可选，默认为 verify_and_update
metadata_len	Metadata 长度，该参数可选
metadata	Metadata，该参数可选

- 参数 dist_dst 应为多播地址，Updating Nodes 的 BLOB Transfer Server 和 Firmware Update Server 均需要订阅该多播地址。如果只给一个目标节点升级（Distributor 升级节点列表中只有一个节点），dist_dst 可以为该节点的单播地址，此时 Updating Nodes 的 BLOB Transfer Server 和 Firmware Update Server 可以不需要订阅任何多播地址；
- 参数 update_timeout_base 用于计算 BLOB 传输的超时时间：
  *blob transfer timeout:(update_timeout_base+1)*10s*；
- 参数 fw_image_size 应包含 Image Header(1K)和 Payload；
- 参数 fw_image_start_addr 应指向 Image Header 的首地址。
- 参数 update_policy 主要有 verify only 和 verify and update 两种，区别是验证完成之后是否直接更新
- 参数 metadata_len 和 metadata 则代表该 image 的 metadata

下面介绍 Distributor 端在 Mesh OTA 过程中所需的流程，Updating Node 端不需要进行操作。其中，Distributor 会对 Updating Nodes List 中的节点进行升级，部分 message 需要单独与节点进行交互，需要知道每个设备的单播地址。同时，分发固件 message 一般是发往组播地址，每个节点相应的 model 需要订阅对应的组播地址才能完成接受固件的过程。

表 3-24 Mesh OTA Distributor 端流程示例

Step	Distributor	说明
1	pbadvcon 000102030405060708090a0b0c0d0e0f prov	对 Updating Node 进行配网
2	aka x100 0 0	给 Updating Node 0x100 添加 app key 0，并绑定到 net key 0 上
3	mab x100 0 x1400ffff 0 mab x100 0 x1402ffff 0	给 x100 节点第 0 个 element 上的 BLOB Transfer Server 和 Firmware Update Server 绑定 app key 0
4	msa x100 0 x1400ffff xc000 msa x100 0 x1402ffff xc000	给 x100 节点第 0 个 element 上的 BLOB Transfer Server 和 Firmware Update Server 订阅组播地址 0xc000
5	dfua x100 0	在 Updating Nodes List 中新增 x100
6	dfus xc000 0 0 2956 x847000	升级启动，对存在于 Updating Nodes List 中并相应 model 订阅了 xc000 的节点进行固件升级，新固件大小为 2956 Bytes，在 Distributor 中存储的首地址为 x847000

3.5.1.2 API

下面介绍 Mesh OTA 过程中，在 Distributor 和 Updating Node 端的部分 API，更多详细的 API 使用参考 Mesh_SDK。

Updating Node 端，向 Firmware Information List 中添加固件信息的 API 介绍如下，默认只添加了 RTK 的固件信息。

函数 **bool fw_update_server_add_info(fw_id_t *pfw_id,**
uint8_t fw_id_len,
uint8_t *pupdate_uri,
uint8_t update_uri_len)

功能	添加固件信息到 Firmware Information List
参数	pfw_id：固件 company_id 和 version 信息 fw_id_len：pfw_id 数据的长度 pupdate_uri：固件下载的 uri update_uri_len：pupdate_uri 长度

Distribuor 端，对应 dfua、dfud 以及 dfus 三个命令的 API 如下。

函数	bool dfu_dist_receiver_add(fw_dist_receiver_t *precv)
功能	添加目标节点信息到 Updating Nodes List
参数	precv：目标节点信息

函数	void dfu_dist_receiver_remove_all(void)
功能	删除 Updating Nodes List 中所有节点的信息
参数	无

函数	bool dfu_dist_start(uint16_t dst, uint16_t app_key_index, uint16_t update_timeout_base, uint8_t update_policy, uint8_t *pfw_metadata, uint8_t metadata_len, uint32_t fw_image_size, uint32_t fw_image_start_addr, fw_image_data_get_t fw_image_data_get)
功能	启动固件分发
参数	dst：分发目标地址，多播地址 app_key_index：app key 的 index update_timeout_base：Updating Nodes 计算接收数据的超时时间 update_policy：节点收到固件后是否直接应用 pfw_metadata：固件的元数据 metadata_len：元数据长度 fw_image_size：Image 的大小，单位为 Bytes fw_image_start_addr：指向 Image Header 的首地址 fw_image_data_get：读取 Distributor 设备 flash 的函数

3.5.2 Initiator - Distributor - Updating Node

Initiator 角色负责将固件和待升级节点列表上传给 Distributor，并控制 Distributor 分发固件的过程。新固件需要提前烧录到 Initiator 的 Flash 中。RTK 的带有 MP 的 bin 文件组成为 MP Header（512Bytes）+Image Header（1K）+Payload，其中 MP Header 仅用作烧录。以烧录 image 的方式将 bin 到对应的区域后只剩下 Image Header 和 Payload，以 user data 的形式将 bin 文件固化在 flash 中则还会保留 MP Header。Mesh OTA

过程只需要 Image Header 和 Payload，如果以 user data 形式固化新的固件，则传输时需要将 size-512，同时首地址向后偏移 512Bytes。Flash 布局和 Image 构成的具体内容请参考《RTL87x2x OTA User Manual》。

Initiator 将固件上传给 Distributor 时，为了保证 Distributor 不会给自身升级，采用了在 OTA temp 区偏移 4K 的方式，即 Distributor 接收到的固件存储在 OTA temp 首地址+4K 的位置，后面接受到的固件依次存储。

3.5.2.1 命令

下面介绍 Initiator 角色与 OTA 过程相关的 CLI 命令。

表 3-25 Initiator 角色相关 CLI 命令

Cmd	说明
dfuus	上传固件到 Distributor
fdra	上传可升级节点到 Distributor
dfuds	启动 Distributor 固件分发过程

下面对每一个 Cmd 的参数含义进行解释，dfuus 参数介绍如下。

表 3-26 dfuus 参数介绍

dfuus	说明
dist_dst	Distributor 单播地址
dist_app_key_index	上传固件使用的 appkey
upload_timeout_base	上传超时时间
fw_image_size	固件大小，bytes
fw_image_start_addr	固件在 Initiator 存储的首地址

fdra 参数介绍如下。

表 3-27 fdra 参数介绍

fdra	说明
dst	Distributor 单播地址
app_key_index	上传可升级节点使用的 appkey
address	可升级节点单播地址
update fw index	升级固件 index

fdra 参数中的 update fw index，是根据 mesh_device 的 Firmware Information List 状态里的添加顺序进行检索。往该 List 添加固件信息的 api 为 fw_update_server_add_info，默认只包含了 RTK 的固件信息，image index 为 0 则标识为 RTK 固件。如果没有新增固件信息，则 update fw index 参数应设置为 0。

dfuds 所需参数如下。

表 3-28 dfuds 参数介绍

dfuds	说明
dst	Distributor 单播地址
app_key_index	Distribution start 消息使用的 app key
dist_app_key_index	Update start 消息使用的 app key
dist_timeout_base	Distributor 计算接收数据的超时时间
dist_fw_image_idx	标识传输的 Image
dist_multicast_addr	分发固件的目标地址
dist_dst_len	分发地址的字节数，判断是否为组播地址或者虚拟地址

下面介绍 Initiator 在 Mesh OTA 过程中所需的流程，Distributor 和 Updating Node 端不需要进行操作。其中，Distributor 会对 Updating Nodes List 中的节点进行升级，部分 message 需要单独与节点进行交互，需要知道每个设备的单播地址。

同时，分发固件 message 一般是发往组播地址，每个节点相应的 model 需要订阅对应的组播地址才能完成接受固件的过程。

表 3-29 Mesh OTA Initiator 端流程示例

Step	Distributor	说明
1	pbadvcon 000102030405060708090a0b0c0d0e0f Prov pbadvcon 000202030405060708090a0b0c0d0e0f prov	对 Distributor 和 Updating Node 进行配网
2	aka x100 0 0	给 Distributor 0x100 添加 app key 0，并绑定到 net key 0 上
3	aka x101 0 0	给 Updating Node 0x100 添加 app key 0，并绑定到 net key 0 上
4	mab x100 0 x1400ffff 0 mab x100 0 x1401ffff 0 mab x100 0 x1403ffff 0 mab x100 0 x1404ffff 0	给 Distributor 0x100 节点第 0 个 element 上的 BLOB Transfer Client/Server 和 Firmware Update Client 和 Firmware Distribution Server 绑定 app key 0

Step	Distributor	说明
5	mab x101 0 x1400ffff 0 mab x101 0 x1402ffff 0	给 Updating Node 0x101 节点第 0 个 element 上的 BLOB Transfer Server 和 Firmware Update Server 绑定 app key 0
6	msa x101 0 x1400ffff xc000 msa x101 0 x1402ffff xc000	给 Updating Node 0x101 节点第 0 个 element 上的 BLOB Transfer Server 和 Firmware Update Server 订阅组播地址 0xc000
7	dfuus x100 0 0 2956 x847000	上传固件，传输到 Distributor 0x100 节点，新固件大小为 2956 Bytes，在 Initiator 中存储的首地址为 x847000
7	fdra x100 0 x101 0	上传 Updating Node 0x101 节点信息到 Distributor 0x100 的 Updating Nodes List 中
8	dfuds x100 0 0 0 0 xc000 2	分发过程启动，对存在于 Distributor 0x100 的 Updating Nodes List 中并相应 model 订阅了 xc000 的节点进行固件分发，新固件编号为 0，分发目标地址长度为 2bytes

3.5.2.2 API

下面介绍 Mesh OTA 过程中，在 Initiator 端的部分 API，Distributor 和 Updating Node 端的 API 参考 3.5.1.2 小节，更多详细的 API 使用参考 Mesh_SDK。

Initiator 端，上传 Updating Nodes List 和启动 Distributor 固件分发过程的 API 可以直接调用对应 Model 层的 message，上传固件的 API 如下。

函数 **bool dfu_init_upload_start(uint16_t dst,**
uint16_t app_key_index,
uint16_t update_ timeout_base,
uint8_t *pfw_metadata,
uint8_t metadata_len,
uint32_t fw_image_size,
uint32_t fw_image_start_addr,
fw_image_data_get_t fw_image_data_get)

功能	启动固件上传
参数	dst：分发目标地址，多播地址 app_key_index：app key 的 index update_timeout_base：Updating Nodes 计算接收数据的超时时间 pfw_metadata：固件的元数据 metadata_len：元数据长度 fw_image_size：Image 的大小，单位为 Bytes

函数	**bool dfu_init_upload_start(uint16_t dst,** **uint16_t app_key_index,** **uint16_t update_ timeout_base,** **uint8_t *pfw_metadata,** **uint8_t metadata_len,** **uint32_t fw_image_size,** **uint32_t fw_image_start_addr,** **fw_image_data_get_t fw_image_data_get)**
	fw_image_start_addr：指向 Image Header 的首地址 fw_image_data_get：读取 Initiator 设备 flash 的函数

与上节介绍的 API 相比少了一个参数 *update_policy*，因为 Distributor 收到固件之后不用验证和应用，只需要存储在本地即可。

参考文献

[1] Mesh Profile Specification

[2] Mesh Model Specification

[3] RTL87x2x Mesh SDK User Guide

[4] RTL87x2x OTA User Manual