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

Revision History

Date	Version	Comments	Author	Reviewer
2018/10/12	Draft v0.1	Draft version	bill	
2018/11/16	Draft v0.2	Add descriptions for using product ID in fission upgrade	bill	
2018/12/11	Draft v0.3	Add mesh OTA part	bill	
2019/01/24	Draft v0.4	Add AIS OTA part	bill	
2021/12/09	Draft v0.5	Update the mesh OTA part	sterling	bill
2022/01/24	Draft v0.6	Add Initiator role to the mesh OTA example section	sterling	bill
2023/06/29	V1.0	Draft version	sterling	bill
2024/01/25	V1.1	Update model ID for blob transfer, firmware update, firmware distribution model	sterling	bill

Contents

Table List

Figure List

1 Introduction

The OTA process of Mesh devices can be implemented through GATT on BLE Link, or directly through Mesh Networking. The GATT-based OTA can utilize the high rate of the LE link to upgrade the device quickly. Mesh-based OTA can use the multicast feature of Mesh network to upgrade multiple devices at the same time.

- Based on GATT OTA, Realtek's private OTA technology can be used to upgrade equipment conveniently and quickly.
- Based on Mesh OTA, SIG has formulated related Models to support OTA, which will be introduced later.

2 OTA over GATT

2.1 OTA one by one

Provide IOS/Android APK to upgrade devices one by one through mobile phones, tablets, etc.

- RTK private OTA
- Alibaba AIS OTA

2.2 Fission OTA

RTK private OTA supports mutual upgrade between Mesh devices, and fission transmission.

- High-version devices automatically upgrade surrounding low-version devices.
- A device of a lower version is upgraded to a device of a higher version, the surrounding devices of the lower version are also automatically upgraded.

Fission OTA usage requirements.

- Device needs to support the Master role
- Devices share an Image
- Image version must be incremented

For inter-device OTA, the number of supported links in the config file needs to be configured as 2, one Master link and one Slave link.

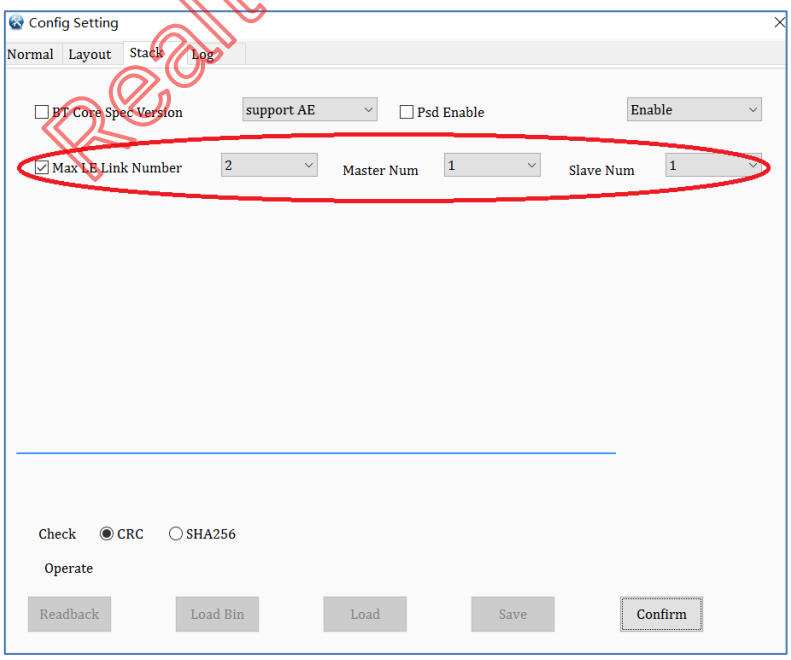

Figure 2-1 Config file setting the number of supported links

Only Image compatible devices can upgrade each other, define the Image compatibility flag for this.

- Company ID: macro MANUFACTURE_ADV_DATA_COMPANY_ID
- Product ID: macro DFU_PRODUCT_ID

Note: If the user does not have the company ID distributed by SIG, he needs to use Realtek's company ID 0x005d, and he needs to negotiate the definition of product ID with Realtek, so as to avoid product ID conflicts and cause immeasurable consequences!

In order to ensure a controllable upgrade, the Image version must be incremented. The version is defined as the macro DFU_APP_VERSION.

3 OTA over Mesh

SIG defines related models that can use Mesh network for OTA, including BLOB (Binary Large Objects) Transfer Server/Client model, Firmware Update Server/Client model and Firmware Distribution Server/Client model. The BLOB Transfer model is mainly used to transfer data, and the Firmware Update model and Firmware Distribution model are mainly responsible for negotiating and monitoring the firmware update process.

SIG OTA only allows the version to be incremented. Modify the DFU_UPDATER_FW_ID macro to increment the version number. In the mesh ali light project, the version number is determined by ALI_VERSION_ID.

3.1 Introduction for Firmware Update

Four roles are defined in the Mesh network to participate in firmware updates.

Table 3-1 Mesh firmware update roles

Role	Description
Updating Node	Nodes whose firmware needs to be updated
Initiator	Check for available firmware updates, send to Distributor
Distributor	Provide firmware to Updating Node
Stand-alone Updater	Firmware can be provided directly to Updating Node

The models that each role needs to support are as follows.

Table 3-2 Models that roles need to support

Role	Firmware Distribution Client	Firmware Distribution Server	Firmware Update Client	Firmware Update Server	BLOB Transfer Client	BLOB Transfer Server
Updating Node	-	-	-	required	-	required
Initiator	required	-	required	-	required	-
Distributor	-	required	required	-	required	required
Stand-alone Updater	not required	not required	required	-	required	not required

Among them, the role of Updating Node is the node that needs to receive new firmware. The Distributor role transfers the new firmware to the Updating Node to be updated and monitors its transfer progress. The Initiator role is mainly responsible for checking if there is an update available, uploading the new firmware and necessary

parameters and the list of Updating Nodes to the Distributor this time. The Stand-alone Updater role can directly acquire new firmware and transfer it to the Updating Node.It can be seen that the Initiator and Stand-alone Updater need to be able to check and obtain new firmware, usually mesh gateway devices or smartphones. In simpler application scenarios, the Initiator is not necessary. You only need to burn the new firmware to the Distributor, set the corresponding parameters and the list of nodes that need to be updated, and then transfer the firmware to the Updating Node to complete the OTA process.

The following briefly introduces the content related to each model. For specific protocols, please refer to the SIG related Spec.

3.2 BLOB Transfer Model

The BLOB Transfer Model can transfer data objects larger than the maximum access layer with PDU size and can multicast to multiple nodes simultaneously. The process of OTA with Mesh uses the BLOB Transfer Model to transfer the new firmware to all nodes capable of receiving it.

3.2.1 State

The states used by the BLOB Transfer Model are divided into the following three categories. The first category is the status shared by the Server/Client, the second category is the status used by the Server alone, and the third category is the status used by the Client alone.

3.2.1.1 Server/Client Shared States

This subsection describes the state shared by BLOB Transfer Server/Client.

3.2.1.1.1 BLOB

The BLOB state is a composite state that identifies and characterizes the BLOB being transmitted. BLOB states include the following states.

Table 3-3 BLOB State

BLOB state	Description
BLOB ID state	Used to identify the BLOB
BLOB Size state	Indicates the size of the BLOB
Block Size Log state	Calculate the size of the block
BLOB Data state	BLOB data
Total Blocks status	The total number of blocks of the BLOB

3.2.1.1.2 BLOB Block

The BLOB block state is a composite state that identifies the block being transferred. BLOB block states include the following states.

Table 3-4 BLOB block State

BLOB block state	Description
BLOB Number state	Used to identify a block in a BLOB
Current BLOB Size state	Indicates the size of the current BLOB being transferred
Chunk Size state	Indicates the size of the chunk in a block
Total Chunks state	Indicates the number of chunks in a block
BLOB Block Data state	Indicates the data of a block

3.2.1.1.3 Transfer TTL

The Transfer TTL state indicates the TTL (Time to Live) value used when the network PDU is initiated from the BLOB Transfer Client or Server.

3.2.1.1.4 Transfer Mode

The Transfer Mode state indicates the mode of BLOB transmission, which is mainly divided into Push mode and Pull mode.

3.2.1.2 Server States

This subsection describes the states used by BLOB Transfer Server alone.

3.2.1.2.1 Transfer Phase

Transfer Phase is used to identify the state of the BLOB Transfer Server.

3.2.1.2.2 Expected BLOB ID

Expected BLOB ID status for identifying BLOB Transfer Server expects to receive the BLOB ID.

3.2.1.2.3 Blocks Not Received

The Blocks Not Received state indicates that the Server has not received blocks yet.

3.2.1.2.4 Missing Chunks

The Missing Chunks status indicates that the server has not received and is transmitting chunks.

3.2.1.2.5 Client MTU Size

The Client MTU Size state indicates the size of the MTU (Maximum Transfer Unit) of the Client that initiates the BLOB transfer process.

3.2.1.2.6 Server Timeout Base

The Server Timeout Base state is used to calculate the time that the Server is waiting for a message from the Client.

3.2.1.2.7 Capabilities

The Capabilities state is a composite state that represents the transfer capabilities of an instance of the BLOB Transfer Server model. Capabilities states include the following states.

Table 3-5 BLOB Transfer Server model Capabilities States

Capabilities state	Description
Min Block Size Log state	Indicates the minimum block size that the server can receive
Max Block Size Log state	Indicates the maximum block size that the server can receive
Max Total Chunks state	Indicates the maximum number of chunks a block can be divided into
Max Chunk Size state	Indicates the maximum chunk size supported by the server
Max BLOB Size state	Indicates the maximum BLOB size that the server can receive
Server MTU Size state	Indicates the MTU size supported by the Server
Supported Transfer Mode state	Indicates the transmission mode supported by the server

3.2.1.3 Client States

This subsection describes the states used by BLOB Transfer Client alone.

3.2.1.3.1 BLOB Receivers

The BLOB Receivers state is used to store the list of BLOB Transfer Servers participating in the transfer. The entries in the list represent the state of the transfer to the Server.

3.2.1.3.2 Active BLOB Receivers

The Active BLOB Receivers state is used to store a list of BLOB Transfer Servers that are actively participating in transfers (eg, not timed out), and the entries in the list represent the state of the transfer to the Server.

3.2.1.3.3 BLOB Receivers Capabilities

The BLOB Receivers Capabilities state is used to store the list of BLOB Transfer Servers participating in the transfer. The entries in the list represent the Capabilities state of the Server.

3.2.1.3.4 BLOB Multicast Address

The BLOB Multicast Address state contains the multicast address, the UUID of the virtual address, or the unassigned address.

3.2.1.3.5 AppKey Index

The AppKey Index state indicates the index of the AppKey used by the client to exchange messages with the BLOB Transfer Server.

3.2.1.3.6 Client Timeout Base

The Client Timeout Base state is used to calculate the time that the Client waits for messages from the Server.

3.2.2 SIG Message

3.2.2.1 BLOB Transfer Get

The BLOB Transfer Get message is used to obtain the BLOB Transfer status on the BLOB Transfer Server, and the Server responds with the BLOB Transfer Status.

3.2.2.2 BLOB Transfer Start

The BLOB Transfer Start message is used to start a new BLOB transfer, and the Server responds with the BLOB Transfer Status.

3.2.2.3 BLOB Transfer Cancel

The BLOB Transfer Cancel message is used to cancel the ongoing BLOB transfer, and the Server responds with the BLOB Transfer Status.

3.2.2.4 BLOB Transfer Status

The BLOB Transfer Status message is used to report the status of the BLOB Transfer Server, with no ack.

3.2.2.5 BLOB Block Get

The BLOB Block Get message is used to retrieve the stage of Transfer and find out the Block being transferred, and the Server responds with the BLOB Block Status.

3.2.2.6 BLOB Block Start

The BLOB Block Start message is used to start the process of transferring blocks to the Server, and the Server responds with the BLOB Block Status.

3.2.2.7 BLOB Block Status

The BLOB Block Status message is used to report the status of the block being transmitted, with no ack.

3.2.2.8 BLOB Partial Block Report

The BLOB Partial Block Report message is used by the Server to request chunks from the Client, with no ack.

3.2.2.9 BLOB Chunk Transfer

The BLOB Chunk Transfer message is used to transfer the chunk of the current block to the BLOB Transfer Server without a response.

3.2.2.10 BLOB Information Get

The BLOB Information Get message is used to obtain the Capabilities status of the BLOB Transfer Server, and the Server responds with the BLOB Information Status.

3.2.2.11 BLOB Information Status

The BLOB Information Status message is used to report the Capabilities status of the BLOB Transfer Server, with no ack.

3.2.3 BLOB Transfer Server Interface

Table 3-6 BLOB Transfer Server Model Interface

Function	Instuction
blob_transfer_server_reg	Register BLOB Transfer Server model
blob_transfer_server_set_data_cb	Set BLOB Transfer Server model callback function
blob_transfer_server_caps_set	Set Capabilities of BLOB Transfer Server
blob_transfer_server_caps_get	Get Capabilities of BLOB Transfer Server
blob_transfer_server_init	Initialize BLOB Transfer Server
blob_transfer_server_load	Load the BLOB Transfer program
blob_transfer_server_clear	Clear BLOB Transfer Server
blob_transfer_server_phase_get	Get the current phase of BLOB Transfer Server
blob_transfer_server_busy	Query whether the BLOB Transfer Server is busy
blob_transfer_handle_transfer_timeout	BLOB Transfer transfer timeout function
blob_transfer_handle_partial_report_timeout	BLOB Transfer Block report timeout function
blob_transfer_status	Response the status of BLOB Transfer
blob_block_status	Response the status of BLOB Block
blob_partial_block_report	Send BLOB partial block report
blob_info_status	Response the status of BLOB Information

3.2.4 BLOB Transfer Client Interface

Table 3-7 BLOB Transfer Client Model Interfaces and Messages

Function	Description
blob_transfer_client_reg	Register BLOB Transfer Client Model
blob_transfer_get	Get the status of BLOB Transfer
blob_transfer_start	Start a new BLOB transfer
blob_transfer_cancel	Cancel an ongoing BLOB transfer
blob_block_get	Retrieve the stage of the Transfer and find out which Block is being transferred
blob_block_start	Start transmitting Block
blob_chunk_transfer	Transfer the Chunks of current Block
blob_info_get	Get the Capabilities of the Server

3.2.5 Examples

These are the main following steps when using the BLOB Transfer model.

1. Register Model
2. Implement the model_data_cb function

This model needs to be implemented between the Initiator and Distributor, and between Distributor and Updating Node to support data transfer. The example introduces the implementation framework of the BLOB Transfer model between Distributor and Updating Node. For the specific implementation, please refer to *dfu_updater_app.c* and *dfu_distributor_app.c*.

3.2.5.1 Server Model Example

BLOB Transfer Server model model_data_cb function example is as follows.

```
1.  int32_t fw_update_handle_blob_server_data(const mesh_model_info_p pmodel_info,
                                              uint32_t type, void *pargs)
2.  {
3.      switch (type)
4.      {
5.  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:
    {
    /* Handle BLOB_TRANSFER_SERVER_FAIL */
    }
        break;
    default:
        break;
      }
      return MODEL_SUCCESS;
    }
```


Register BLOB Transfer Server model is as follows.

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

3.2.5.2 Client Model Example

BLOB Transfer Client model model_data_cb function example is as follows.

```
int32_t dfu_transfer_client_data(const mesh_model_info_p pmodel_info,
                                 uint32_t type, void *pargs)
{
```

```
    switch (type)
    {
    case BLOB_TRANSFER_CLIENT_TRANSFER_STATUS:
      {
       /* Handle BLOB_TRANSFER_CLIENT_TRANSFER_STATUS */
      }
      break;
    case BLOB_TRANSFER_CLIENT_BLOCK_STATUS:
      {
       /* Handle BLOB_TRANSFER_CLIENT_BLOCK_STATUS */
      }
      break;
    case BLOB_TRANSFER_CLIENT_PARTIAL_BLOCK_REPORT:
      {
       /* Handle BLOB_TRANSFER_CLIENT_PARTIAL_BLOCK_REPORT */
      }
      break;
    case BLOB_TRANSFER_CLIENT_INFO_STATUS:
      {
       /* Handle BLOB_TRANSFER_CLIENT_INFO_STATUS */
      }
      break;
    default:
      break;
    }
    return MODEL_SUCCESS;
}
```

Register BLOB Transfer Client model is as follows.

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

3.3 Firmware Update Model

The Firmware Update Model can retrieve firmware information, determine whether a node can receive new firmware, start firmware transfer, and apply firmware. The process of transferring firmware depends on the BLOB Transfer Model implementation.

3.3.1 State

3.3.1.1 Server States

3.3.1.1.1 Firmware Information List

The Firmware Information List state is a list of Update URIs and Firmware IDs for each firmware installed on the Updating Node. This state changes after the node successfully applies a new firmware update.

Table 3-8 Firmware Information List Information List

Contents	Describe
Current Firmware ID	Identifies the firmware installed on the Updating Node
Update URI	Location of the new firmware archive file

3.3.1.1.2 Update Phase

The Update Phase status is used to identify the update phase of the Firmware Update Server.

3.3.1.1.3 Firmware Update Additional Information

Firmware Update Additional Information status indicates firmware update additional information.

3.3.1.1.4 Update Firmware Image Index

The Update Firmware Image Index status is used to identify the firmware being updated in the Firmware Information List.

3.3.1.1.5 New Firmware Image

The New Firmware Image status is the binary image of the firmware.

3.3.1.1.6 Update BLOB ID

The Update BLOB ID status represents the BLOB ID value used by the BLOB Transfer Server in firmware transfers.

3.3.1.1.7 Update Firmware Metadata

The Update Firmware Metadata state represents the metadata of the incoming firmware.

3.3.1.1.8 Update TTL

The Update TTL state represents the TTL value used by the BLOB Transfer Server during firmware updates.

3.3.1.1.9 Update Server Timeout Base

The Update Server Timeout Base state indicates the timeout period after Firmware Update Server suspends firmware transfer reception.

3.3.1.2 Client States

3.3.1.2.1 Update Receivers

Table 3-9 Update Receivers States Format

Contents	Describe
Address	The unicast address of Updating Node
Firmware Image Index	The firmware index of Updating Node
Retrieved Update Phase	Retrieved update phase states of Updating Node
Status	Updating Node state since last operation
Update Additional Information	Retrieved firmware update additional information for Updating Node

3.3.1.2.2 Active Update Receivers

Active Update Receivers status is used to identify Firmware Update Servers that are actively participating in updates.

3.3.1.2.3 Update Multicast Address

The Update Multicast Address state contains the multicast address, the UUID of the virtual address, and the unassigned address.

3.3.1.2.4 Update AppKey Index

The Update AppKey Index state indicates the index of the AppKey.

3.3.1.2.5 Update TTL

The Update TTL state represents the TTL value used by the BLOB Transfer Client during firmware updates.

3.3.1.2.6 Update Client Timeout Base

The Update Client Timeout Base state is used to calculate the timeout for responses sent to the Firmware Update Server.

3.3.2 SIG Message

3.3.2.1 Firmware Update Information Get

The Firmware Update Information Get message is used to obtain information about the firmware installed on the node, and the Server responds with Firmware Update Information Status.

3.3.2.2 Firmware Update Information Status

The Firmware Update Information Status message is used to report information about the firmware installed on the node, no reply.

3.3.2.3 Firmware Update Firmware Metadata Check

Firmware Update Firmware Metadata Check message is used to check whether the node can receive firmware updates, and the server responds with Firmware Update Firmware Metadata Status.

3.3.2.4 Firmware Update Firmware Metadata Status

Firmware Update The Firmware Metadata Status message is used to report whether the firmware update can be accepted, with no ack.

3.3.2.5 Firmware Update Get

The Firmware Update Get message is used to obtain the current status of the Server, and the Server responds to the Firmware Update Status.

3.3.2.6 Firmware Update Start

The Firmware Update Start message is used to start the firmware update, and the Server responds with the Firmware Update Status.

3.3.2.7 Firmware Update Cancel

The Firmware Update Cancel message is used to cancel the firmware update and delete the stored information about the firmware update, and the Server responds with Firmware Update Status.

3.3.2.8 Firmware Update Apply

The Firmware Update Apply message is used to apply the firmware that has been transferred, and the Server responds with Firmware Update Status.

3.3.2.9 Firmware Update Status

Firmware Update Status is used to report firmware update status, with no ack.

3.3.3 Firmware Update Server Interface

Table 3-10 Firmware Update Server Model Interface

Function	Description
fw_update_server_reg	Register Firmware Update Server Model
fw_update_server_load	Load Firmware Update Server
fw_update_server_add_info	Add firmware update information
fw_update_server_set_verify_result	Set Firmware Verification Results
fw_update_server_clear	Clear Firmware Update Server
fw_update_handle_blob_server_data	BLOB Transfer Server callback function

3.3.4 Firmware Update Client Interface

Table 3-11 Firmware Update Client model Interface

Function	Description
fw_update_client_reg	Register Firmware Update Client Model
fw_update_info_get	Get information about firmware installed on a node
fw_update_fw_metadata_check	Check if the node can receive firmware updates
fw_update_get	Get the current state of the Server
fw_update_start	Initiate firmware update
fw_update_cancel	Cancel firmware update
fw_update_apply	Apply the firmware that has been transferred

3.3.5 Eampple

These are the main following steps when using the Firmware Update model:

1. Register Model
2. Implement the model_data_cb function

The example introduces the implementation framework of Firmware Update model between Distributor and Updating Node. For specific implementation, please refer to *dfu_updater_app.c* and *dfu_distributor_app.c*.

3.3.5.1 Server Model Example

Firmware Update Server model model_data_cb function example is as follows.

```
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;
}
```

Register Firmware Update Server model is as follows.

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

3.3.5.2 Client Model Example

Firmware Update Client model model_data_cb function example is as follows.

```
int32_t dfu_update_client_data(const mesh_model_info_p pmodel_info,
                               uint32_t type, void *pargs)
{
  switch (type)
  {
  case FW_UPDATE_CLIENT_INFO_STATUS:
    {
     /* Handle FW_UPDATE_CLIENT_INFO_STATUS */
    }
    break;
  case FW_UPDATE_CLIENT_FW_METADATA_STATUS:
    {
     /* Handle FW_UPDATE_CLIENT_FW_METADATA_STATUS */
    }
    break;
  case FW_UPDATE_CLIENT_STATUS:
    {
     /* Handle FW_UPDATE_CLIENT_STATUS */
    }
    break;
  default:
    break;
  }
  return MODEL_SUCCESS;
}
```

Register Firmware Update Client model is as follows.

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

3.4 Firmware Distribution Model

The Firmware Distribution Model is mainly responsible for checking for updates, uploading the Updating Nodes list and firmware, and starting and applying the firmware distribution process. The process of uploading firmware depends on the BLOB Transfer Model implementation.

3.4.1 State

3.4.1.1 Server States

3.4.1.1.1 Distribution Receivers

The Distribution Receivers state is a composite state that contains the following states:

Table 3-12 Distribution Receivers States

Distribution Receivers States	Description
Distribution Receivers List States	List of Updating Nodes and target firmware
Distribution Receivers List Count States	Number of Distribution Receivers List States

3.4.1.1.2 Distributor Capabilities

The Distributor Capabilities state is a composite state that represents the firmware update capabilities supported by the Firmware Distribution Server model.

Table 3-13 Firmware Distribution Server model Capabilities States

Capabilities States	Description
Max Distribution Receivers List Size States	Indicates the maximum number of addresses that the Firmware Distribution Server can store in the Distribution Receivers List state
Max Firmware Images List Size States	Indicates the maximum number of firmware that the Firmware Distribution Server can store in the Firmware Images List state
Max Firmware Image Size States	Indicates the maximum size of a single firmware stored in the Firmware Distribution Server
Max Upload Space Size States	Indicates the total size of firmware stored by the Firmware Distribution Server
Remaining Upload Space Size States	Indicates the available space for Firmware Distribution Server to store firmware

Capabilities States	Description
Out-of-Band Retrieval Supported States	Indicates whether the Firmware Distribution Server supports retrieving firmware through OOB
Supported URI Scheme Names States	Indicates a list of supported URI schemes

3.4.1.1.3 Distribution Parameters

The Distribution Parameters state is a composite state containing parameters used by the Firmware Distribution Server model during distribution of firmware.

Table 3-14 Distribution Parameters States

Distribution Parameters States	Description
Distribution Phase States	Indicates the stage that the Firmware Distribution Server is performing firmware distribution
Update Policy States	Indicates when new firmware is applied
Distribution Multicast Address States	Indicates the destination address
Distribution AppKey Index States	Indicates the index of the AppKey used
Distribution TTL States	Indicates the TTL value used by the Firmware Distribution Server
Distribution Timeout Base States	Used to calculate the timeout value for Firmware Update Client and BLOB Transfer Client during firmware distribution
Distribution Transfer Mode States	Indicates the transfer mode for firmware distribution
Distribution Firmware Image Index States	Identify the firmware being distributed

3.4.1.1.4 Upload Parameters

The Upload Parameters state is a composite state that contains parameters used by the Firmware Distribution Server model during uploading firmware to the Distributor.

Table 3-15 Upload Parameters States

Upload Parameters States	Description
Upload Firmware ID States	Identifies the firmware being uploaded to Distributor
Upload Phase States	Identifies the firmware upload phase
Upload Firmware Size States	Indicates the size of the firmware being uploaded
Upload BLOB ID States	Indicates the BLOB ID used during upload
Upload Firmware Metadata States	Metadata representing the firmware being uploaded
Upload URI States	Timeout value used to calculate the upload period

3.4.1.1.5 Firmware Images List

The Firmware Images List state contains information about each firmware stored on the Firmware Distribution Server.

Table 3-16 Firmware Images List States format

Contents	Description
Index	Index of the list
Firmware ID	Identify firmware
Firmware Metadata	Metadata of the firmware identified by the Firmware ID

3.4.1.2 Client States

3.4.1.2.1 Update Information From Updating Nodes

The Update Information From Updating Nodes status is a list that identifies the firmware provided by the Firmware Distribution Client and installed on each Updating Nodes.

Table 3-17 Update Information From Updating Nodes States Format

Contents	Description
Address	The unicast address of Updating Node
Firmware Images List	List of Firmware Images Entry

The format of each Firmware Images Entry state is as follows.

Table 3-18 Firmware Images Entry States Format

Contents	Description
Current Firmware ID Length	Current Firmware ID Field Length
Current Firmware ID	Updating the most recently reported firmware version on Node
New Firmware ID Length	Length of new firmware ID field
New Firmware ID	New firmware available for Updating Node
Update URI Length	Length of the URI field
Update URI	URI for locating new firmware
Firmware Image Length	Length of firmware field
Firmware Image	Firmware

3.4.2 SIG Messages

3.4.2.1 Firmware Distribution Receivers Add

The Firmware Distribution Receivers Add message is used to add a new entry to the Status of the Distribution Receivers List of the Firmware Distribution Server, and the Server responds with the Firmware Distribution Receivers Status.

3.4.2.2 Firmware Distribution Receivers Delete All

The Firmware Distribution Receivers Delete All message is used to delete all entries in the Firmware Distribution Server's Distribution Receivers List status, with no ack.

3.4.2.3 Firmware Distribution Receivers Status

The Firmware Distribution Receivers Status message is used to report the current size of the Distribution Receivers List status, with no ack.

3.4.2.4 Firmware Distribution Receivers Get

The Firmware Distribution Receivers Get message is used to obtain the firmware distribution status of each Updating Node, and the Server responds to the Firmware Distribution Receivers List.

3.4.2.5 Firmware Distribution Receivers List

The Firmware Distribution Receivers List message is used to report the firmware distribution status of each receiver, with no ack.

3.4.2.6 Firmware Distribution Capabilities Get

The Firmware Distribution Capabilities Get message is used to obtain the Capabilities of the Firmware Distribution Server, and the Server responds with the Firmware Distribution Capabilities Status.

3.4.2.7 Firmware Distribution Capabilities Status

Firmware Distribution Capabilities Status is used to report the Capabilities of the Distributor, with no ack.

3.4.2.8 Firmware Distribution Get

The Firmware Distribution Get message is used to obtain the current firmware distribution status on the Firmware Distribution Server, and the Server responds with the Firmware Distribution Status.

3.4.2.9 Firmware Distribution Start

The Firmware Distribution Start message is used to start firmware distribution, and the Server responds with Firmware Distribution Status.

3.4.2.10 Firmware Distribution Cancel

The Firmware Distribution Cancel message is used to cancel firmware distribution, and the Server responds with Firmware Distribution Status.

3.4.2.11 Firmware Distribution Apply

The Firmware Distribution Apply message is used to make Updating Nodes apply the new firmware, and the Server responds with Firmware Distribution Status.

3.4.2.12 Firmware Distribution Status

The Firmware Distribution Status message is used to report the firmware distribution status, with no ack.

3.4.2.13 Firmware Distribution Upload Get

The Firmware Distribution Upload Get message is used to check the status of firmware upload to the Firmware Distribution Server, and the Server responds with Firmware Distribution Upload Status.

3.4.2.14 Firmware Distribution Upload Start

The Firmware Distribution Upload Start message is used to start firmware upload, and the Server responds with Firmware Distribution Upload Status.

3.4.2.15 Firmware Distribution Upload OOB Start

The Firmware Distribution Upload OOB Start message is used to enable OOB to upload firmware, and the Server responds with Firmware Distribution Upload Status.

3.4.2.16 Firmware Distribution Upload Cancel

The Firmware Distribution Upload Cancel message is used to stop the firmware upload, and the Server responds with the Firmware Distribution Upload Status.

3.4.2.17 Firmware Distribution Upload Status

The Firmware Distribution Upload Status message is used to report the status of the firmware upload, with no ack.

3.4.2.18 Firmware Distribution Firmware Get

The Firmware Distribution Firmware Get message is used to check whether a specific firmware is stored on the Firmware Distribution Server, and the Server responds with Firmware Distribution Firmware Status.

3.4.2.19 Firmware Distribution Firmware Get By Index

The Firmware Distribution Firmware Get By Index message is used to check which firmware is stored on the Firmware Distribution Server according to the specific index on the Firmware Images List status, and the Server responds to the Firmware Distribution Firmware Status.

3.4.2.20 Firmware Distribution Firmware Delete

The Firmware Distribution Firmware Delete message is used to delete a certain firmware stored on the Firmware Distribution Server, and the Server responds with the Firmware Distribution Firmware Status.

3.4.2.21 Firmware Distribution Firmware Delete All

The Firmware Distribution Firmware Delete All message is used to delete all firmware stored on the Firmware Distribution Server, and the Server responds with Firmware Distribution Firmware Status.

3.4.2.22 Firmware Distribution Firmware Status

Firmware Distribution Firmware Status is used to report the status of the stored firmware after operation, with no ack.

3.4.3 Firmware Distribution Server Interface

Table 3-19 Firmware Distribution Server Model Interface

Function	Description
fw_dist_server_reg	Register Firmware Distribution Server Model
fw_dist_server_add_receiver	Add receiver to Firmware Distribution Server
fw_dist_server_is_receiver_empty	Check if receiver is empty
fw_dist_server_delete_all_receiver	Delete all receivers
fw_dist_server_add_image	Add image to Firmware Images List
fw_dist_server_delete_all_image	Delete all images from Firmware Images List
fw_dist_server_delete_image	Delete an image from the Firmware Images List
fw_dist_server_cancel_done	Cancel distribution process complete
fw_dist_handle_blob_server_data	BLOB Transfer Server callback function
fw_dist_server_start	Start firmware distribution
fw_dist_server_dist_failed	Notification distribution failed
fw_dist_server_set_upload_recvd_size	Set the uploadable firmware size
fw_dist_server_upload_oob_done	Uploading firmware with OOB is done
fw_dist_server_set_updating_status	Set node update status

3.4.4 Firmware Distribution Client Interface

Table 3-20 Firmware Distribution Client Model Interface

Function	Description
fw_dist_client_reg	Register Firmware Update Client model
fw_dist_recvs_add	Add receivers for Firmware Distribution Server
fw_dist_recvs_delete_all	Delete all receivers of Server
fw_dist_recvs_get	Get firmware distribution status for each Updating Node
fw_dist_caps_get	Get Server Capabilities
fw_dist_get	Get the status of the current firmware distribution on the Server

Function	Description
fw_dist_start	Start firmware distribution
fw_dist_cancel	Stop firmware distribution
fw_dist_apply	Make Updating Nodes apply the new firmware
fw_dist_upload_get	Check the status of firmware upload to Server
fw_dist_upload_start	Start firmware upload
fw_dist_upload_oob_start	Firmware upload with OOB enabled
fw_dist_upload_cancel	Stop firmware upload
fw_dist_fw_get	Check if a particular firmware is stored on the Server
fw_dist_fw_get_by_index	Check which firmware is stored on the Server by a specific index on the Firmware Images List status
fw_dist_fw_delete	Delete a certain firmware stored on the Server
fw_dist_fw_delete_all	Delete all firmware stored on the Server

3.4.5 Example

These are the main following steps when using the Firmware Update model.

1. Register Model
2. Implement the model_data_cb function

The example introduces the implementation framework of the Firmware Distribution model between Distributor and Initiator. For the specific implementation, please refer to *dfu_initiator_app.c* and *dfu_distributor_app.c*.

3.4.5.1 Server Model Example

Firmware Distribution Server model model_data_cb function example is as follows.

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

```
11.         {
12.          /* Handle FW_DIST_SERVER_CANCEL */
13.         }
14.         break;
15.       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.   }
```

Register Firmware Distribution Server model is as follows.

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

3.4.5.2 Client Model Example

Firmware Distribution Client model model_data_cb function example is as follows.

```
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 */
```

```
        }
        break;
    case FW_DIST_CLIENT_UPLOAD_STATUS:
        {
        /* 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;
}
```

Register Firmware Distribution Client model is as follows.

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

3.5 Mesh OTA Example

The mesh OTA process can be realized by using the mesh_provisioner project and the mesh_device project. The mesh_provisioner can assume the roles of Initiator and Distributor at the same time, and mesh_device can assume the roles of Distributor and Updating Node at the same time. On the same node, multiple roles can coexist, and roles can be assigned according to the scene requirements for testing.

3.5.1 Distributor - Updating Node

In simpler application scenarios, the Initiator role can be reduced, and the Distributor role can transmit firmware to the Updating Node role. Some functions of the Initiator are implemented through the serial CLI command line or by calling part of the API. At the same time, the new firmware also needs to be burned into the Distributor's Flash in advance. RTK's bin file with MP consists of MP Header (512Bytes) + Image Header (1K) + Payload, where MP Header is only used for burning.

After burning the image to the corresponding area, only the Image Header and Payload are left. If the bin file is solidified in the flash in the form of user data, the MP Header will be retained. The Mesh OTA process only needs Image Header and Payload. If the new firmware is solidified in the form of user data, size-512 needs to be set during transmission, and the first address is offset by 512Bytes backwards. Please refer to *RTL87x2x OTA User Manual* for the specific content of Flash layout and Image composition.

3.5.1.1 CMD

The following describes the CLI commands related to the OTA process for the Distributor role.

Table 3-21 Distributor Role CLI cmd

Cmd	Description
dfua	Add a new node to the Distributor upgrade node list
dfud	Delete all nodes in the Distributor upgrade node list
dfus	Start the firmware upgrade process

The meaning of each Cmd parameter is explained below. The dfua parameter is introduced as follows.

Table 3-22 dfua parameters introduction

address	The unicast address of the node
image index	The index of Image
address	The unicast address of the node

The image index in the dfua parameter is retrieved according to the addition order in the Firmware Information List status of mesh_device. The api for adding firmware information to this List is *fw_update_server_add_info*, which only contains RTK firmware information by default. If the image index is 0, it is identified as RTK firmware. If there is no new firmware information, the image index parameter should be set to 0.

- dfud does not contain any parameters.
- The parameters required by dfus are as follows.

Table 3-23 dfus parameters introduction

dfus	Description
dist_dst	distribution destination address, multicast address
dist_app_key_index	index of app key
update_timeout_base	Updating Nodes calculate the timeout for receiving data
fw_image_size	The size of the Image, in Bytes
fw_image_start_addr	Points to the first address of the Image Header
update policy	Update policy, optional，verify_and_update by default
metadata_len	Length of metadata, optional
metadata	Metadata, optional

The parameter *dist_dst* should be a multicast address. Both the BLOB Transfer Server and Firmware Update Server of Updating Nodes need to subscribe to this multicast address. If only one target node is upgraded (there is only one node in the Distributor upgrade node list), *dist_dst* can be the unicast address of the node.

At this time, the BLOB Transfer Server and Firmware Update Server of Updating Nodes do not need to subscribe to any multicast address.

The parameter *update_timeout_base* is used to calculate the timeout time of BLOB transfer: blob transfer timeout:(*update_timeout_base* + 1)*10s.

The parameter *fw_image_size* should contain Image Header (1K) and Payload. The parameter *fw_image_start_addr* should point to the first address of the Image Header.

There are two main parameters *update_policy*: *verify_only* and *verify_and_update*. The difference is whether to update directly after verification is completed.

The parameters *metadata_len* and *metadata* represent the metadata of the image.

The following describes the process required by the Distributor side in the Mesh OTA process, and the Updating Node side does not need to be operated. Among them, the Distributor will upgrade the nodes in the Updating Nodes List, and some messages need to interact with the nodes separately and need to know the unicast address of each device.

At the same time, the distributed firmware message is generally sent to the multicast address, and the corresponding model of each node needs to subscribe to the corresponding multicast address to complete the process of accepting the firmware.

Table 3-24 Mesh OTA Distributor process example

Step	Distributor	Description
1	pbadvcon 000102030405060708090a0b0c0d0e0f prov	Provision the Updating Node
2	aka x100 0 0	Add app key 0 to Updating Node 0x100 and bind it to net key 0
3	mab x100 0 x1400ffff 0 mab x100 0 x1402ffff 0	Bind app key 0 to the BLOB Transfer Server and Firmware Update Server on the 0th element of the x100 node
4	msa x100 0 x1400ffff xc000 msa x100 0 x1402ffff xc000	Subscribe the multicast address 0xc000 to the BLOB Transfer Server and Firmware Update Server on the 0th element of the x100 node
5	dfua x100 0	Added x100 to Updating Nodes List
6	dfus xc000 0 0 2956 x847000	The upgrade is started, and the firmware of the node that exists in the Updating Nodes List and the corresponding model is subscribed to xc000 is upgraded. The size of the new firmware is 2956 Bytes, and the first address stored in the Distributor is x847000

3.5.1.2 API

The following introduces some APIs on the Distributor and Updating Nodes in the Mesh OTA process. For more detailed API usage, refer to Mesh_SDK.

Updating Node, the API for adding firmware information to the Firmware Information List is as follows. By default, only RTK firmware information is added.

Prototype	**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)**
Function	Add information to the Firmware Information List
Parameter	*pfw_id*: company_id and version of firmware *fw_id_len*: length of the pfw_id data *pupdate_uri*: uri for firmware download *update_uri_len*: length of the pupdate_uri

On the Distributor side, the APIs corresponding to the three commands dfua, dfud and dfus are as follows.

Prototype	**bool dfu_dist_receiver_add(fw_dist_receiver_t *precv)**
Function	Add information to the destination node Updating Nodes List
Parameter	*Precv*: target node information

Prototype	**void dfu_dist_receiver_remove_all(void)**
Function	Delete all firmware information in Updating Nodes List
Parameter	None

Prototype	**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)**
Function	Start firmware distribution
Parameter	*Dst*: distribution destination address, multicast address *app_key_index*: the index of app key *update_timeout_base*: Updating Nodes calculates the timeout for receiving data

Prototype	**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)**
	update_policy: whether to apply the firmware directly after the node receives the firmware *pfw_metadata*: firmware metadata *metadata_len*: metadata length *fw_image_size*: the size of the Image, in Bytes *fw_image_start_addr*: the address of the Image *fw_image_data_get*: function to read Distributor device flash

3.5.2 Initiator - Distributor - Updating Node

The Initiator role is responsible for uploading the firmware and the list of nodes to be upgraded to Distributor, and controlling the process of Distributor distributing firmware. The new firmware needs to be burned into the Initiator's Flash in advance. RTK's bin file with MP consists of MP Header (512Bytes) + Image Header (1K) + Payload, where MP Header is only used for burning. After burning the image to the corresponding area, only the Image Header and Payload are left. If the bin file is solidified in the flash in the form of user data, the MP Header will be retained.

The Mesh OTA process only needs Image Header and Payload. If the new firmware is solidified in the form of user data, size-512 needs to be set during transmission, and the first address is offset by 512Bytes backwards. Please refer to "RTL87x2x OTA User Manual EN" for the specific content of Flash layout and Image composition.

When the Initiator uploads the firmware to the Distributor, in order to ensure that the Distributor will not be upgraded by itself, the method of offsetting the OTA temp area by 4K is adopted. Store the firmware received by Distributor in the first address of OTA temp + 4K, and store the firmware received later in sequence.

3.5.2.1 CMD

The following describes the CLI commands related to the OTA process for the Initiator role.

Table 3-25 Initiator Role CLI cmd

Cmd	Description
dfuus	Upload firmware to Distributor
fdra	Upload upgradeable nodes to Distributor
dfuds	Start the Distributor firmware distribution process

The meaning of each Cmd parameter is explained below. The dfuus parameter is introduced as follows

Table 3-26 dfuus parameter introduction

dfuus	Description
dist_dst	The unicast address of Distributor
dist_app_key_index	The appkey used to upload the firmware
upload_timeout_base	Upload timeout
fw_image_size	Firmware size, bytes
fw_image_start_addr	The address of the firmware stored in the Initiator

The fdra parameters are described as follows

Table 3-27 fdra parameters introduction

fdra	Description
dst	The unicast address of Distributor
app_key_index	The appkey used to upload the Updating Node List
address	The unicast address of Updating Node List
update fw index	The index of image

The update_fw_index in the fdra parameter is retrieved according to the addition order in the Firmware Information List status of mesh_device. The api for adding firmware information to this List is *fw_update_server_add_info*, which only contains RTK firmware information by default. If the image index is 0, it is identified as RTK firmware. If there is no new firmware information, the update_fw_index parameter should be set to 0.

The parameters required by dfuds are as follows.

Table 3-28 dfuds parameter introduction

dfuds	Description
dst	The unicast address of Distributor
app_key_index	The app key used by the Distribution start message

dfuds	Description
dist_app_key_index	The app key used by the Update start message
dist_timeout_base	Distributor calculates the timeout for receiving data
dist_fw_image_idx	Image that identifies the transfer
dist_multicast_addr	Destination address to distribute firmware
dist_dst_len	The number of bytes of the distribution address to determine whether it is multicast address or virtual address

The following describes the process required by the Initiator in the Mesh OTA process. Distributor and Updating Node do not need to operate. Among them, the Distributor will upgrade the nodes in the Updating Nodes List, and some messages need to interact with the nodes separately and need to know the unicast address of each device. At the same time, the distributed firmware message is generally sent to the multicast address, and the corresponding model of each node needs to subscribe to the corresponding multicast address to complete the process of accepting the firmware.

Table 3-29 Mesh OTA Initiator process example

Step	Distributor	Description
1	pbadvcon 000102030405060708090a0b0c0d0e0f Prov pbadvcon 000202030405060708090a0b0c0d0e0f prov	Provision Distributor and Updating Node
2	aka x100 0 0	Add app key 0 to Distributor 0x100 and bind it to net key 0
3	aka x101 0 0	Add app key 0 to Updating Node 0x100 and bind it to net key 0
4	mab x100 0 x1400ffff 0 mab x100 0 x1401ffff 0 mab x100 0 x1403ffff 0 mab x100 0 x1404ffff 0	Bind app key 0 to the BLOB Transfer Client/Server, Firmware Update Client and Firmware Distribution Server on the 0th element of the Distributor 0x100 node
5	mab x101 0 x1400ffff 0 mab x101 0 x1402ffff 0	Bind app key 0 to the BLOB Transfer Server and Firmware Update Server on the 0th element of Updating Node 0x101
6	msa x101 0 x1400ffff xc000 msa x101 0 x1402ffff xc000	Subscribe the multicast address 0xc000 to the BLOB Transfer Server and Firmware Update Server on the 0th element of Updating Node 0x101
7	dfuus x100 0 0 2956 x847000	Upload the firmware and transfer it to the Distributor 0x100 node. The size of the new firmware is 2956 Bytes, and the first address stored in the Initiator is x847000

Step	Distributor	Description
7	fdra x100 0 x101 0	Upload Updating Node 0x101 node information to the Updating Nodes List of Distributor 0x100
8	dfuds x100 0 0 0 0 xc000 2	The distribution process starts, and the firmware is distributed to the nodes that exist in the Updating Nodes List of Distributor 0x100 and the corresponding model subscribes to xc000. The new firmware number is 0, and the distribution target address length is 2bytes

3.5.2.2 API

The following introduces some APIs on the Initiator side during the Mesh OTA process. For the APIs on the Distributor and Updating Node sides, refer to Section 3.5.1.2. For more detailed API usage, refer to Mesh_SDK.

On the Initiator side, the API for uploading the Updating Nodes List and starting the Distributor firmware distribution process can directly call the message of the corresponding Model layer. The API for uploading firmware is as follows

Prototype **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)

Function	start the firmware upload
Parameter	*Dst*: distribution destination address, multicast address *app_key_index*: the index of the app key *update_timeout_base*: Updating Nodes calculates the timeout for receiving data *pfw_metadata*: the metadata of firmware *metadata_len*: metadata length *fw_image_size:* the size of the Image , in Bytes *fw_image_start_addr*: the address of the Image *fw_image_data_get*: the function to read the flash of the Initiator device

Compared with the API described in 3.5.1.2, there is one less parameter update_policy, because the Distributor does not need to verify and apply the firmware after receiving the firmware, and only needs to store it locally.

References

[1] Mesh Profile Specification

[2] Mesh Model Specification

[3] RTL87x2x Mesh SDK User Guide

[4] RTL87x2x OTA User Manual