I-CUBE-LRWAN embedding FUOTA, application implementation

Introduction

This application note describes the FUOTA application embedded in the expansion software (I-CUBE-LRWAN) implementation on STM32L4 Series devices, and explains how to make use of the overall FUOTA process in order to provide the components needed for a FUOTA campaign.

This document applies within the framework of a FUOTA project, and is intended for teams or individuals, either internal or external to ST. It particularly targets FUOTA project integrators, or those integrating FUOTA modules in a wider system implementing end-device functions.

LoRa® is a type of wireless telecommunication network designed to allow long range communication at a very low bit rate, enabling long-life battery operated sensors. LoRaWAN® defines the communication and security protocol to ensure interoperability with LoRa networks.

AN5411 - Rev 1 - November 2019
For further information contact your local STMicroelectronics sales office.

www.st.com

1 General information

This document applies to STM32L476xx Series Arm®-based microcontrollers.

Note: *Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

arm

2 Overview

The FUOTA application in I-CUBE-LRWAN is compliant with the LoRa Alliance® specification protocol (LoRaWAN version V1.0.3) [1].

The FUOTA feature is implemented in the application layer, and is based on and compliant with the specific functionalities defined by the LoRA Alliance. These functionalities allow a multicast group (Remote Multicast Setup Spec V1.0.0 [4]) to be set up, to fragment and to send data packets (Fragmented Data Block Transport Specification v1.0.0 [3]) and finally to synchronize clocks (LoRaWAN Application Layer Clock Synchronization Specification v1.0.0 [5]) so that all devices can agree on the start of a FUOTA session.

Note: *Throughout this application note, the IAR™ EWARM IDE is used as an example to provide guidelines for project configuration.*

2.1 Application supports

The firmware update over-the-air (FUTOA) application supports:

- full firmware upgrade image (the entire firmware image is sent to the end-device)
- only applicable in Class-C mode
- only runs on STM32L476xx targets
- third-party middleware, mdedTLS (open source code) for the cryptographic services.

2.2 Terms and acronyms

Table 1. Acronyms used in this document

Acronym	Definition
ABP	Activation by personalization
APDU	Application protocol data unit
FUOTA	Firmware update over the air
FW	Firmware
HAL	Hardware abstraction layer
LoRa	Long-range radio technology
LoRaWAN	LoRa, wide-area network
MAC	Media access control
MCPS	MAC common part sublayer
MLME	MAC sublayer management entity
MPDU	Mac protocol data unit
MSC	Message sequence chart
OTA	Over-the-air
PLME	Physical sublayer management Entity
PPDU	Physical protocol data unit
SBSFU	Secure boot and secure firmware update
SFU	Secure firmware update
TPDU	Transport protocol data unit
SAP	Service access point

Table 2. Terms used in this document

Term	Definition
Firmware image	A binary image (executable0 run by end-device as user application
Firmware header	Meta-data describing the firmware image to be installed.
mbedTLS	mbedTLS implementation of the TLS and SSL protocols and the associated cryptographic algorithm.
".sfb" file	Binary file packing the firmware header and the firmware image.

2.3 References

Table 3. Document references

| Reference	Document
[1]	LoRa Alliance Specification Protocol (LoRaWAN version V1.0.3), March 2018
[2]	IEEE Std 802.15.4TM - 2011. Low-Rate Wireless Personal Area Networks (LR-WPANs)
[3]	LoRa Alliance Fragmented Data Block Transport over LoRaWAN Specification v1.0.0, September 2018 – [TS-004]
[4]	LoRa Alliance Remote Multicast Setup over LoRaWAN Specification v1.0.0, September 2018 – [TS-005]
[5]	LoRa Alliance Application layer clock synchronization over LoRaWAN Specification v1.0.0, September 2018 – [TS-003]
[6]	AN5056 Integration Guide for the X-CUBE-SBSFU STM32Cube Expansion Package – application note
[7]	UM2262 - Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package – user manual
[8]	UM2073 – STM32 LoRa Expansion Package for STM32 - user manual

3 LoRa standard and FUOTA application feature

This section provides a general overview of the LoRa and LoRaWAN recommendations. It deals in particular with the LoRa end-device and the FUOTA feature, which are the core subjects of this application note.

The LoRa software expansion is compliant with the LoRa Alliance LoRaWAN specification protocol [1].

3.1 Network architecture

Figure 1 shows the components and their protocol relationships, allowing the implementation of the firmware-over-the-air feature.

Figure 1. Network diagram
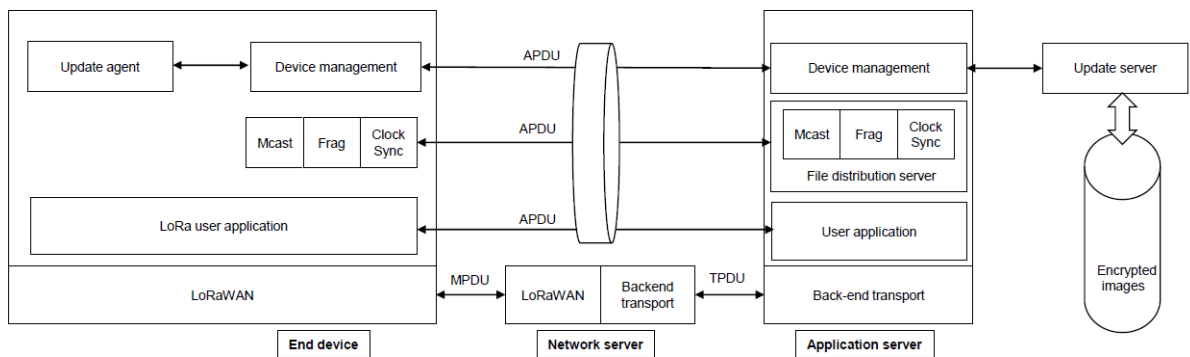
Note: *In the I-CUBE-LRWAN FUOTA package, the device management block is not implemented. The LoRa Alliance technical FUOTA working group is working on a LoRaWAN Firmware Management Protocol Specification, which documents and defines this block.*

3.1.1 Client/server architecture

The end-device where the software or firmware is to be updated is referred to as the *end node* or *client*. The other part of the system is referred to as the *cloud* or *server*, and provides the new software or firmware (Figure 2).

Figure 2. Client/server architecture example
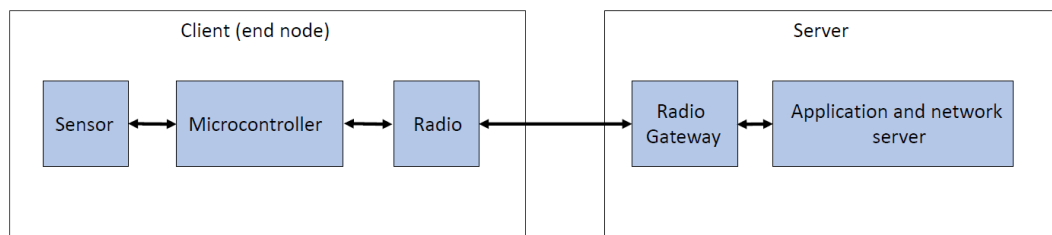
3.1.2 End-device architecture

An end-device consists of a host MCU that reads sensor data in order to transmit the sensor reading over the LoRa network by means of the LoRa radio module.

Data is encrypted by the host MCU and the radio packet is received by the gateway, which forwards it to the network server. The network server then sends data to the application server, which has the right key to decrypt the application data.

3.2 End-device classes

LoRaWAN [1] has several end-device classes to address the various needs of a wide range of applications.

The FUOTA application described in this application note is only 'Class-C enable'. In other words the FUOTA application is validated for network infrastructure supporting only Class-C mode.

Note: *The end-device supports Class-B mode. Nevertheless it is only 'Class B capable'. To be 'Class B enable' it is mandatory to proceed to a new integration and validation phase on a network infrastructure supporting Class-B mode for the FUOTA campaign.*

3.2.1 Class definition

- Bi-directional end devices - Class A - (all devices) see [1]
- Bi-directional end-devices with scheduled receive slots - Class B – (Beacon) see [1]
- Bi-directional end-devices with maximal receive slots - Class C – (Continuous)

Class-C mode is implemented to support FUOTA. Class-C end devices have almost continuously-open receive windows (RxC where the data blocks are received), and are only closed when transmitting (Tx) and receiving (Rx1, Rx2) in Class-A mode (Figure 3).

Figure 3. Tx/Rx timing diagram (Class C)
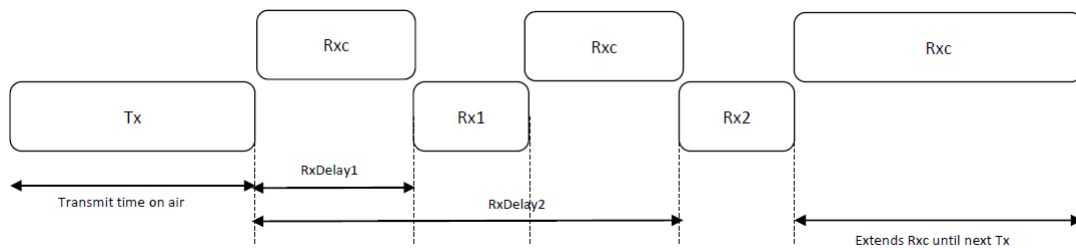
3.3 FUOTA - firmware update over the air

The FUOTA update process transfers a new software image (data file) from the server to the client, and updates the current software image (version N) running on the client with the new received software image (version N+1). Obstacles to successful completion of the FUOTA update process are:

- **Communication**. The new firmware image must be sent from the server to the client. This challenge is performed through the application-layer protocols running over LoRaWAN, which provide remote-multicast setup, fragmented data-block transport, and application-layer clock-synchronization services. The LoRaWAM Mac layer provides *Class-C mode* to transmit the data file in unicast or multicast mode.

- **Firmware update**. The client must migrate from the current to the new firmware image. This task is performed by the *Update_Agent* module. To succed, the *Update_Agent* module relies on the services provided by the SBSFU middleware

- **Memory**. The software architecture must be organized so that it can be executed when the update process completes. The solution must ensure the recovery of the new software version if there are installation issues. This task is handled by the SBSFU middleware.

- **Security**. When a new FW image is sent wirelessly from server to client, several security services must be assured, such as: authentication, confidentiality and integrity. This must be done either through the LoRaWAN protocol or by means of the SBSFU middleware security services.

3.4 Network protocol architectures

This section describes the end-to-end network protocol architecture (see Figure 4). The following protocol exchanges are used:

- MAC protocol data unit exchanges (MPDU)

- application protocol of an application data unit exchanges (APDU)

- LoRa protocol physical protocol data unit layer (PPDU).

Figure 4. LoRa network protocols
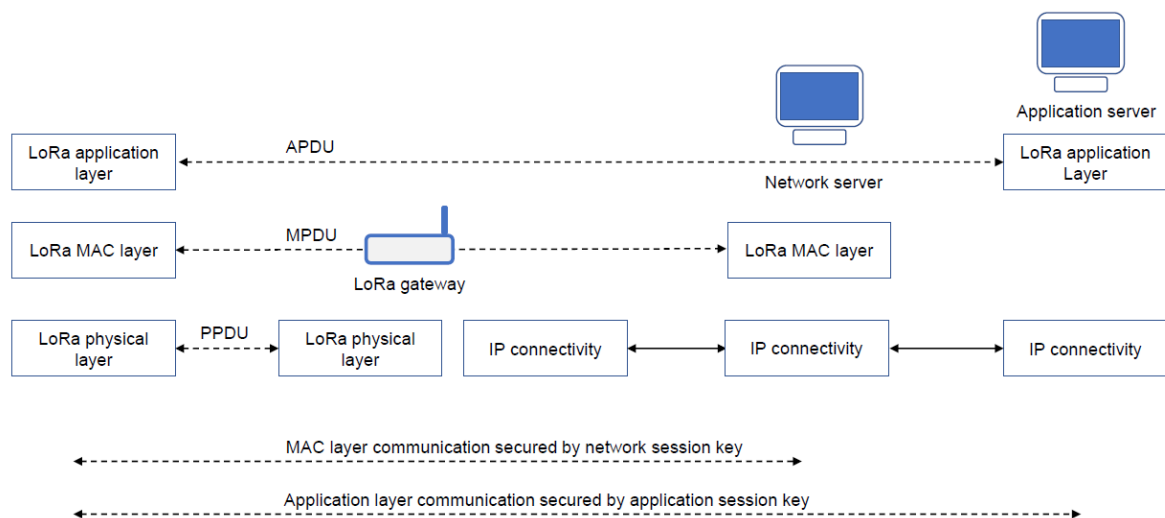

3.4.1 Network layer

The LoRaWAN architecture is defined in terms of blocks called layers. As shown in Figure 5, each layer is responsible for one part of the standard and offers services to higher layers. An end-device is made up of :

- a PHY, which embeds the radio frequency transceiver

- the MAC sublayer that provides access to the physical channel

- application layers that provide access to the LoRaWAN services protocol.

Figure 5. LoRaWAN layers
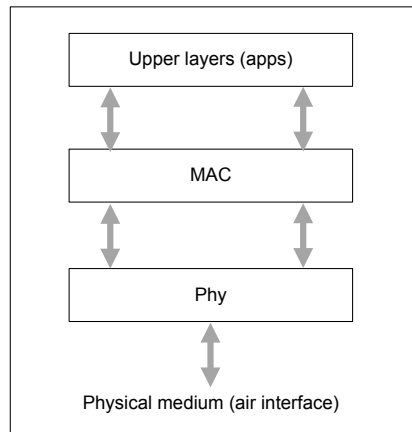
3.4.2 Physical layer (PHY)

The physical layer provides two services:

- the PHY data service enables the Tx/Rx of physical protocol data units (PPDUs)
- the PHY management service enables the personal-area network-information base (PIB) management.

3.4.3 MAC layer

The MAC layer provides two services:

- The MAC data service enables transmission and reception of MAC protocol data units across the physical layer (MPDU)
- The MAC sublayer management enables the PIB management.

3.4.4 Application layer

The application layer provides several messaging packages running over the LoRaWAN protocol. Here, FUOTA scopes the following:

A clock synchronization package (port Number 202)

- Synchronizes the end-device real-time clock to the network's GPS
- Makes all end devices of a multicast group to switch to Class C temporarily and synchronously.

A fragmented-data-block transport package (port Number201)

- Sets up / reports / deletes fragmentation transport sessions
- Several fragmentation sessions **may** be supported simultaneously by an end-device
- Fragmentation can be used either over multicast or unicast
- Authenticates a data block once reconstructed (only supported on V2 of the TS-004 – not supported on V1 – TS004))
- Reports the status of a fragmentation session.

A remote multicast setup package (port Number 200)

- Remotely creates a multicast group security context inside a group of end devices
- Reports the list of multicast contexts existing in the end-device
- Remotely deletes a multicast security context.
- Programs a Class-C multicast session
- Programs a Class-B multicast session.

Firmware management package (port Number 203) -- (not supported in the current version)

- Queries/manages the firmware version running on an end-device (including availability of the firmware-update version)
- Queries/manages the end-device hardware version
- Manages the end-device reboot at a given time.

Update agent module

Firmware module that interfaces a LoRaWAN stack block to an SBSFU block

- Get and transmit the complete file (after recombination) to the secure firmware update (SFU) process of the SBSFU.

User application

- Sensor/actuator processing – application use cases.

3.5 Network/end-device interworking

This section only shows the information flow between end-device and AS at the application-layer level during a FUOTA campaign. For a complete view and description of the end-device and network interactions, refer to the STM32 LoRa expansion package user manual [8].

3.5.1 Multicast and fragmentation set-up

Time synchronization

Before seting up a FUOTA session, the end-device must have synchronized its timing with the network using either *AppTimeReq* or *DeviceTimeReq* as shown in Figure 6.

Figure 6. Message sequence chart for device timing
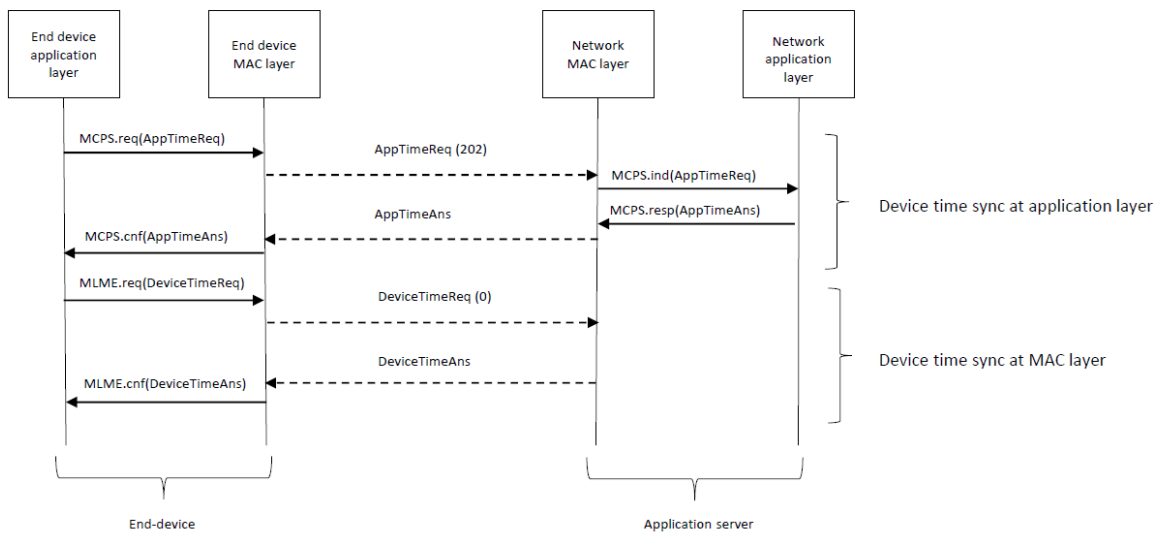
Note: *For the purposes of this presentation, the TimeReq sent by the MSC is divided into DeviceTimeReq and AppTimeReq parts. The LoRaWAN specification allows a MAC command to be piggybacked in an application payload. In the current implementation DeviceTimeReq is piggybacked in the AppTimeReq payload.*

Multicast, fragmentation setup and session creation (Class C only)

In order to receive a data block at the application level, it is necessary to have some exchanges between the network application layer and the end-device application layer. These exchanges are mainly to define: a Multicast Group ID, the fragmentation parameters (frag number and frag size), and to define and create the Class C session.

Figure 7. **Message sequence chart for Class-C creation**
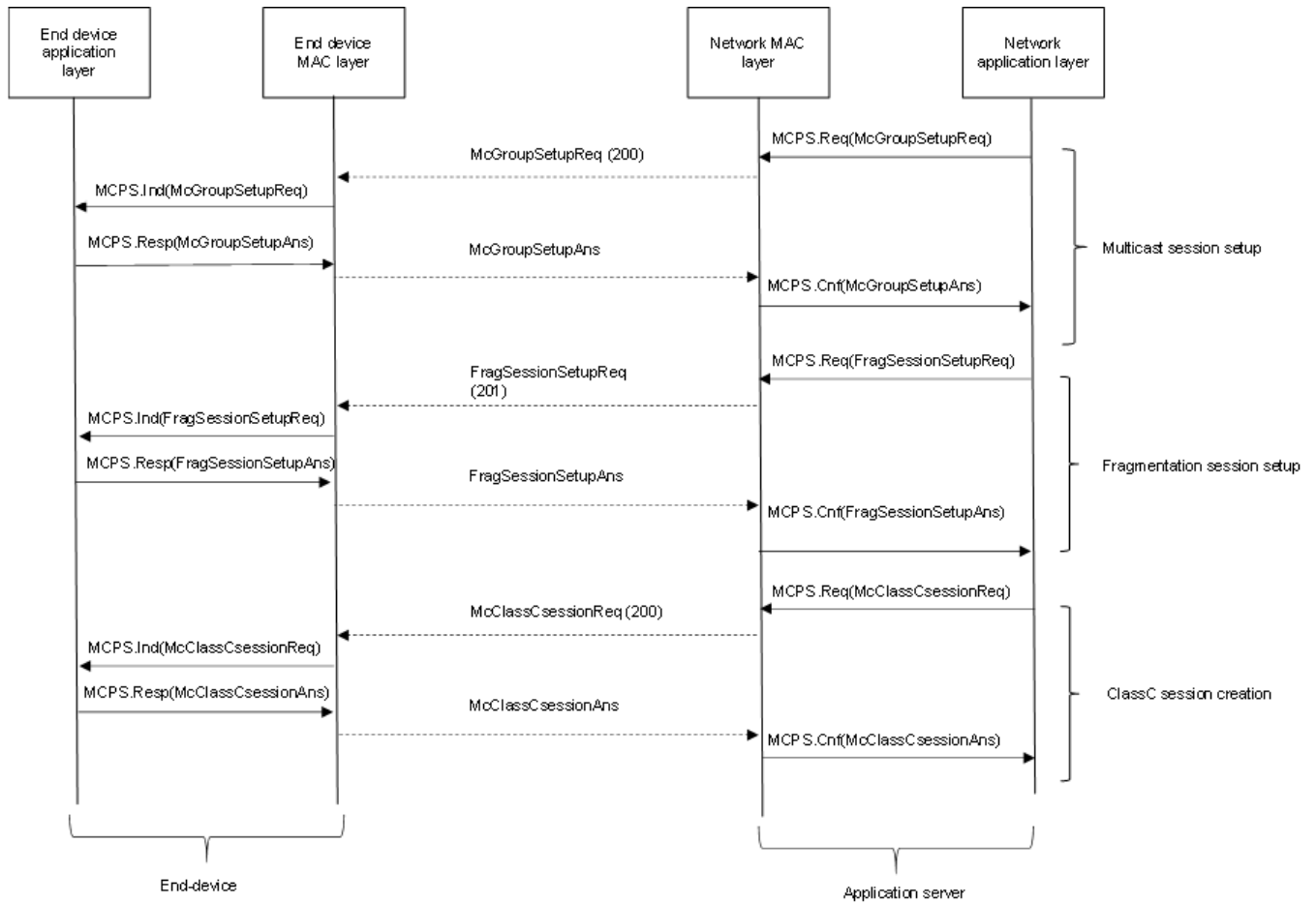
Fragment broadcasting and secure FW update process

As soon as the end-device is synchronized (Class C), it opens its Rx window in order to receive the data fragments [3]. It stays in this state until all the data fragments are recieved.

When the complete data block (see the note below) is received, the end-device closes its Rx windows, and if everything is OK from the 'data-block transfer' point-of-view, the end-device calls the *UpdateAgent* to start the secure firmware update (SFU) process.

Figure 8. **Message sequence chart for data block broadcasting**
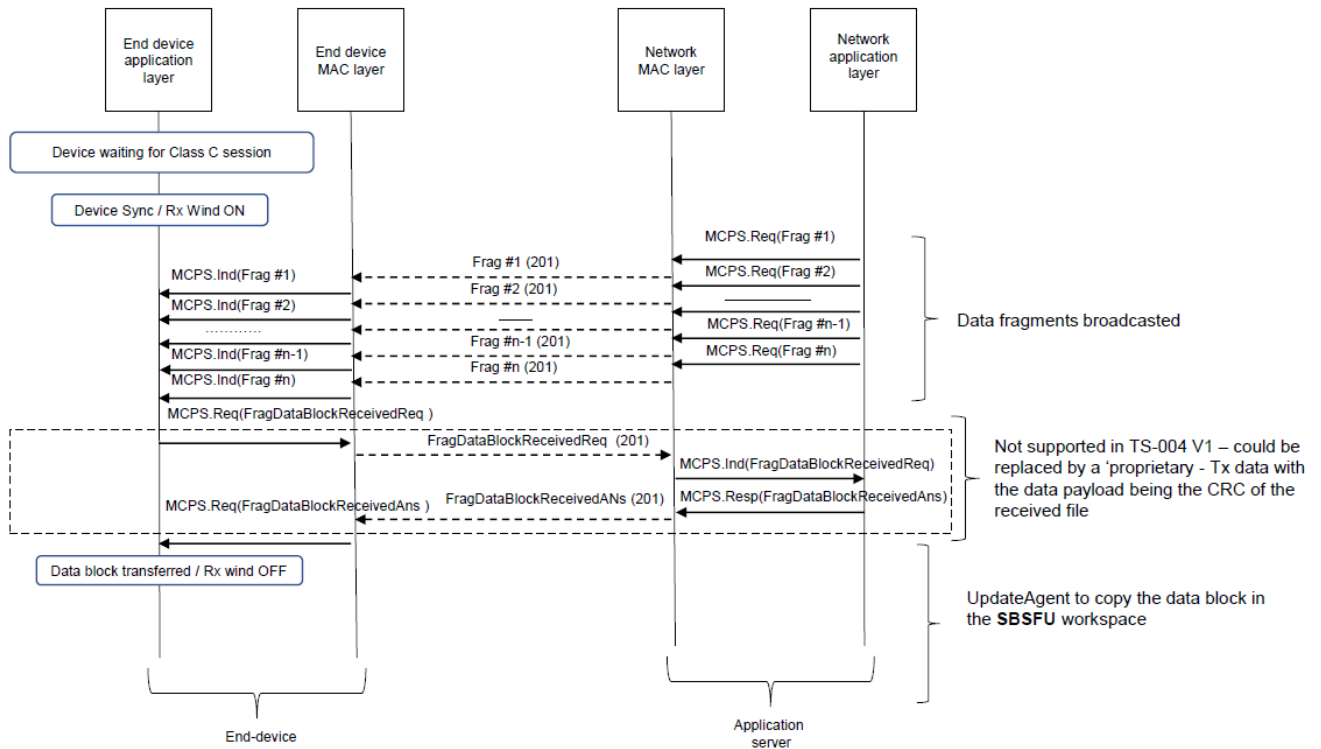
Note: A protocol enhancement is to be proposed in the next Fragmented Data Block Transport specification [TS-004] version. In this, after the complete data block is received, the end-device sends a FragDataBlockReceived.Req message to indicate the current status of the data transfer.

Note: Additional user 'proprietary' protocol statement:

The V1.0 package, and particularly Fragmented Data Transport TS-004, does not provide a way to inform the server that all data blocks have been properly received in order to rebuild the current download file. This is the case in the currently proposed implementation. The server always sends all the fragments (uncoded and coded), even if the current download file has been rebuilt before the end of the complete broadcast fragmentation transaction.

If needed, the user is responsible for implementing a 'proprietary' protocol to avoid such behavior.

For instance, when all the required fragments have been received and the current download file rebuilt, a simple crc32 can be computed and sent back to the server. The server should decide to stop broadcasting the remaining fragments.

This approach requires cooperation between the device maker and the network operator in order to define the 'proprietary' part of the protocol.

4 SBSFU/end-device manager relationship

4.1 Secure boot (SB)

The secure boot loader (SB) is software that permanently resides in the microcontroller's read-only memory. Secure boot checks the integrity and authenticity of the user application that is executed.

Secure boot executes every time a reset occurs and checks if there is a new OTA process to complete.

4.2 Secure firmware update (SFU)

Secure firmware update provides a secure implementation of in-field firmware updates, enabling secure download of a new firmware image either through UART or OTA to the end-device.

4.2.1 Device update scenario

When there is no new firmware to install

There is no OTA process to complete, and Secure Boot (SB) does some signature verifications and branches to the current active firmware - the (Slot#0) user application.

When there is a new FW to install

The OTA process has to be completed. Secure boot (SB) transfers control to secure firmware update (SFU), which contains the OTA update-related software. SFU does the firmware update in Slot#1 and does a firmware swap processing from Slot#1 to Slot#0, and transfers control to Secure Boot. Secure Boot checks if there is an OTA to complete, and branches to the new Firmware in Slot0#

Figure 9. Boot flow with SBSFU and SBSFU memory map
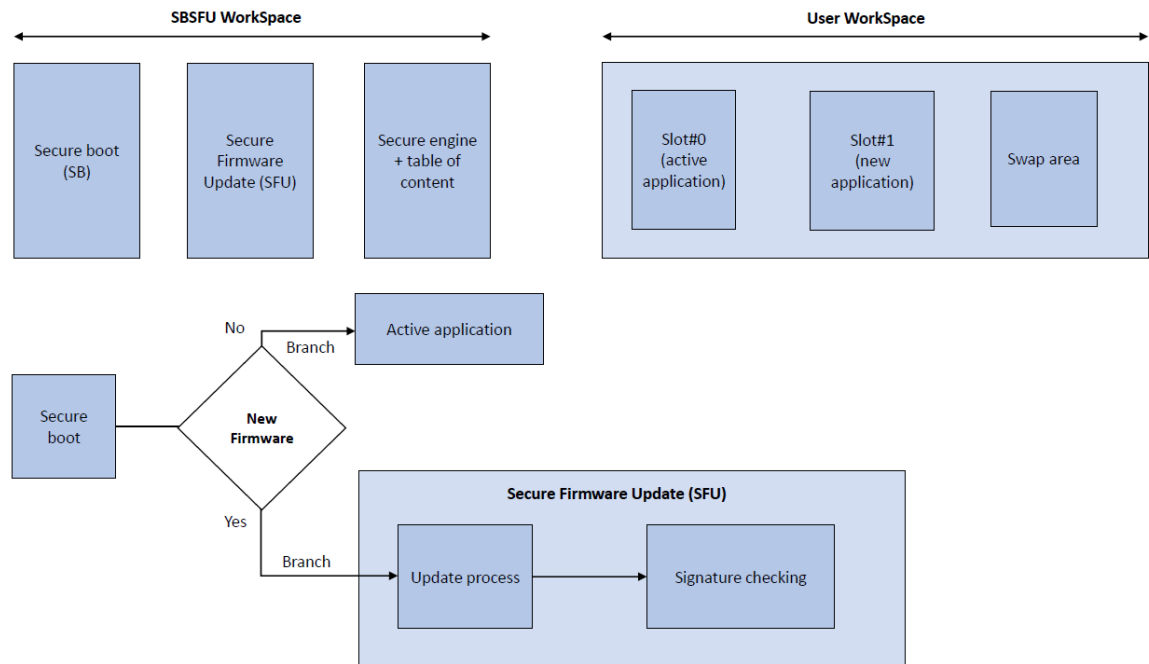
4.3 Security

Security is ensured either through the LoRaWAN protocol, or by means of SBSFU middleware security services.

During a Multicast session, all the end-devices share the same session keys. There is a potential risk when one of the devices becomes compromised, as an attacker can initiate a multicast session with rogue firmware. To counter this, the SBSFU can add a layer of security by using an asymmetric cryptography scheme. When a firmware update is generated, the update is signed (TAG) with a private key by means of the *PrepareImage* SBSFU tool [7]. When the end-device receives the firmware update, it verifies the signatures against the file received and the public key held in the 'secure core' part of the end-device.

Figure 10. Signed firmware image
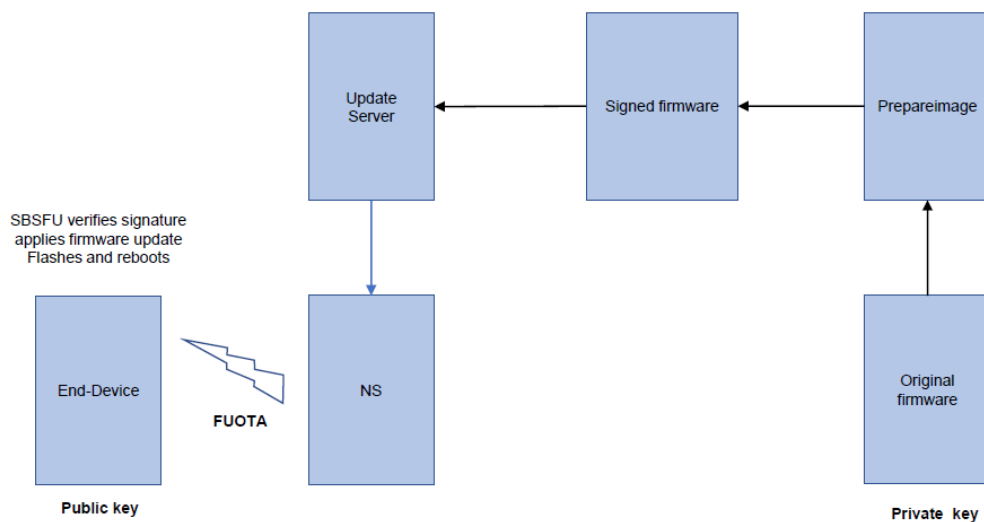
The next version of the Fragmented Data Block Transport TS-004 (V2) [3] introduces an end-device specific key (DataBlockintKey) that removes the risk when the McAppSKey or McNwkSKey is compromised.

4.4 end-device

The firmware update over the air feature has a direct impact on the memory size (FLASH/RAM) of the end-device. The amount of memory needed depends on the maximum size of the firmware image, the technology used to manage the new firmware-update image, and also of the bootloader requirements. The current version proposes a full firmware-update feature.

4.5 Update agent

During the first step of the FUOTA update process - the transfer of the data block (firmware image) from the server to end-device, every time the end-device receives a fragment [3] coming from the server, it is stored in SRAM. When the entire data block (new firmware image) is received from the server (recombination is done and all fragments received), it is transferred (written to Flash memory) by the *UpdateAgent* in the user workspace (Slot#1). After this, the *UpdateAgent* generates an *NVICReset* in order to transfer the control of the MCU to the secure boot loader.

Note: *The full-caching mechanism approach allows the number of writes to Flash memory to be reduced. However, this places a limit on the size of the new firmware image being downloaded. Both the caching method and the firmware image size to update depend on the end-device SRAM size.*

5   Design overview

This package offers a FUOTA LoRa project for STM32L476 microcontrollers. The FUOTA LoRa project is split into three sub-projects : *SecoreBin*, *SBSFU* and *UserApps*.

The middleware is provided in source-code format, and is compliant with the STM32Cube HAL driver.

Figure 11. Project file structure
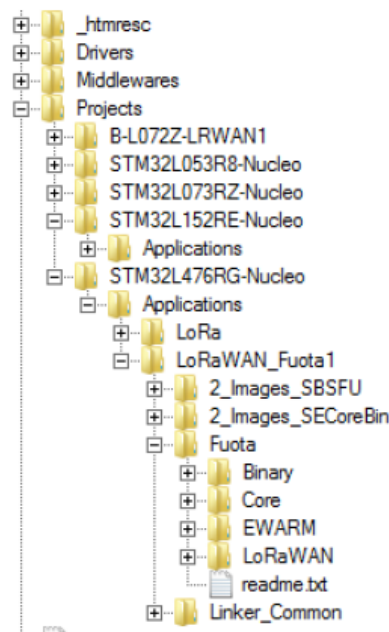
5.1   Features

5.1.1   SBSFU features

5.1.1.1   Secure boot (root of trust)

- Activation and checking of the necessary security mechanisms on the STM32L476 platform, to protect critical operations and secret data from attack.
- Checking the authentication and integrity of the user application before execution.

5.1.1.2   Secure firmware update (SFU)

- Detection of the new (encrypted) firmware version to install:
  - from a local download service (via the Ymodem)
  - pre-downloaded OTA via the user application (LoRaWAN)
- Firmware version management (check for unauthorized updates or unauthorized installation)
- Secure firmware update:
  - firmware authentication and integrity check
  - firmware decryption (if encryption is activated)
  - firmware installation
- On error occurrence during the new image installation, recovery of the new firmware image (rollback to the previous valid firmware version not supported)

• Execution of new installed firmware (once authenticated and integrity checked).

5.1.1.3 *Cryptography*

The official X-CUBE-SBSFU Expansion Package for STM32Cube [7] is delivered with two different cryptographic middleware libraries, either X-CUBE-CRYPTOLIB, which requires an export control agreement, or mbedTLS cryptographic services delivered as open-source code. The I-CUBE-LRWAN FUOTA application package only provides the mbedTLS option.

To use X-CUBE-CRYPTOLIB, the user must download the official X-CUBE-SBSFU Expansion Package for STM32Cube, and replace mbedTLS with X-CUBE-CRYPTOLIB in the project. These two libraries are equivalent in terms of cryptographic services.

The I-CUBE-LRWAN FUOTA package is by default configured as: **Asymmetric without encryption** .

Features	Asymmetric with AES encryption	Asymmetric without encryption	Symmetric (AES GCM)
Confidentiality	AES CBC encryption (firmware binary)	-	AES CBC encryption (firmware binary)
Integrity	SHA256 (firmware header + firmware binary)		-
Authentication	SHA256 of the firmware header is ECDSA signed SHA256 of the firmware binary stored in the firmware header		AES GCM tag (firmware header and firmware binary)
Cryptographic keys in end-device	Private AES CBS key (secret) public ECDSA	Public ECDSA key	Private AES GCM key (secret)

5.1.1.4 *SBSFU configuration*

Cryptographic libraries

As specified in Section 5.1.1.3 Cryptography, the I-Cube-LRWAN FUOTA project project integrates the third-party middleware mbedTLS (open-source code).

Figure 12. Cryptographic library file location

Cryptographic default scheme

By default, the project is configured without cryptography. In this case there is no firmware encryption, only authentication and integrity are ensured.
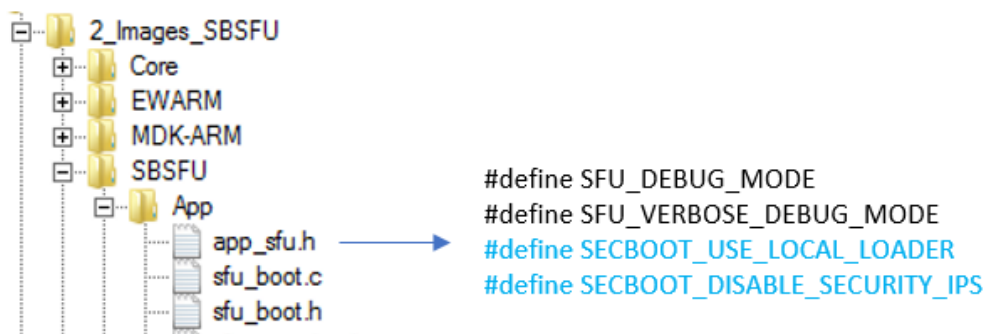
Figure 13. Default cryptographic file structure
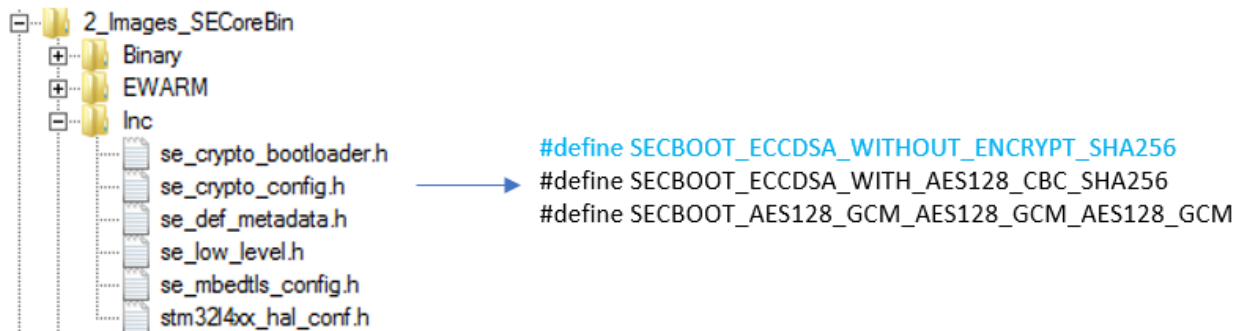
SBSFU application features

Configuration possibilities are offered through option compilation switches. By default the project supports local loader, and all the security IPS are turned off to make debug easier.

Figure 14. File structure for SBFU application features


5.1.2 LoRaWAN features

- **LoRaWAN L2 V1.0.3**
  – Class A (baseline) / Class C (continuous) and Class B (beacon)
- **Application Layer V1**
  – Clock synchronization, fragmentation data block, and remote multicast setup

5.2 Firmware architecture

Figure 15 summarizes the firmware design and the components involved in an end -device supporting the FUOTA features.

Figure 15. Top-level firmware design
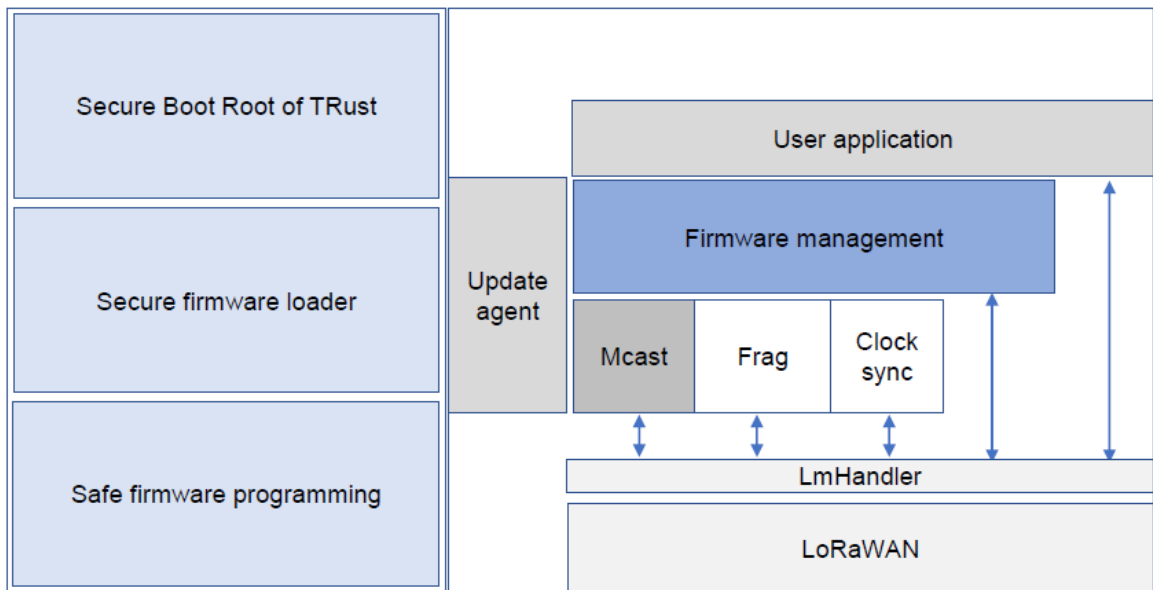
Note: *As stated in Section 3.1 Network architecture, the firmware management block is not implemented.*

6 FUOTA middleware programming guidelines

This section describes the LoRaMac handler APIs.

6.1 Middleware initialization

This function initializes the LoRaMac layer. It initializes the callback primitives of the MCPS and MLME services (see [8]) and the run-time initialization of the LoRaMac layers (active region, Tx parameters, and so on).

Table 4. *LmHandlerInit* description

Function	Description
LmHandlerErrorStatus_t LmHandlerInit (LmHandlerCallbacks_t *callbacks, LmHandlerParams_t *handlerParams);	LoRaMac handler initialization.

6.2 LoRaMac process

This function processes the LoRaMac events and radio events.

Table 5. *LmHandlerProcess* description

Function	Description
void LmHandlerProcess(void);	LoRaMac process – this function has to be called in the main loop. When no operation is pending this function requests to enter Low-power mode.

6.3 Package registering

This function registers the required packages (PACKAGE_ID_COMPLIANCE, PACKAGE_ID_CLOCK_SYNC, PACKAGE_ID_REMOTE_MCAST_SETUP, PACKAGE_ID_FRAGMENTATION).

Table 6. *LmHandlerPackageRegister* description

Function	Description
LmHandlerErrorStatus_t LmHandlerPackageRegister (uint8_t id, void *params);	Perform registration of all the packages needed by the application. These packages are dedicated for: synchronization, fragmentation, multicast setup, and compliant test.

6.4 Join a LoRa network

This function sends a Join request to a LoRa network in Class A.

Table 7. *LmHandlerJoin* description

Function	Description
void LmHandlerJoin(void);	Perfom a Join request. For ABP devices, this is a pass-through function.

6.5 Check the join status

This function checks whether the device is joined to the network.

Table 8. *LmHandlerJoinStatus* description

Function	Description
LmHandlerFlagStatus_t LmHandlerJoinStatus(void);	Check the Join device status. LORAMAC_HANDLER_SET if joined, otherwise LORAMAC_HANDLER_RESET.

6.6 LoRaWAN class change

This function requests the MAC layer to change the LoRaWAN class of the device.

Table 9. *LmHandlerRequestClass* description

Function	Description
LmHandlerErrorStatus_t LmHandlerRequestClass(DeviceClass_t newClass);	Switch Class Request. Only switching from class A to class B/C or from class B/C to class A is allowed

6.7 Check the LoRaMac handler state

This function checks the state of the LoRaMac handler.

Table 10. *LmHandlerIsBusy* description

Function	Description
bool LmHandlerIsBusy(void);	Indicates if the LoRaMac Handler is busy. Returns true or false.

6.8 Send an uplink frame

This function requests the MAC layer to send a Class A uplink frame.

Table 11. *LmHandlerSend* description

Function	Description
*LmHandlerErrorStatus_t LmHandlerSend(LmHandlerAppData_t *appData, LmHandlerMsgTypes_t isTxConfirmed);*	Request a data appData to be sent with an indication of whether the Tx is *TxConfirmed* or *TxUnConfirmed*.

7 Getting started

7.1 FUOTA programing guide

This section describes how to generate a FUOTA application (referred to as a UserApp in the SBSFU literature). The developer must follow this flow description step-by-step.

7.2 Folder structure

A top-level view of the file structure is shown in Figure 16.

Figure 16. Project file structure
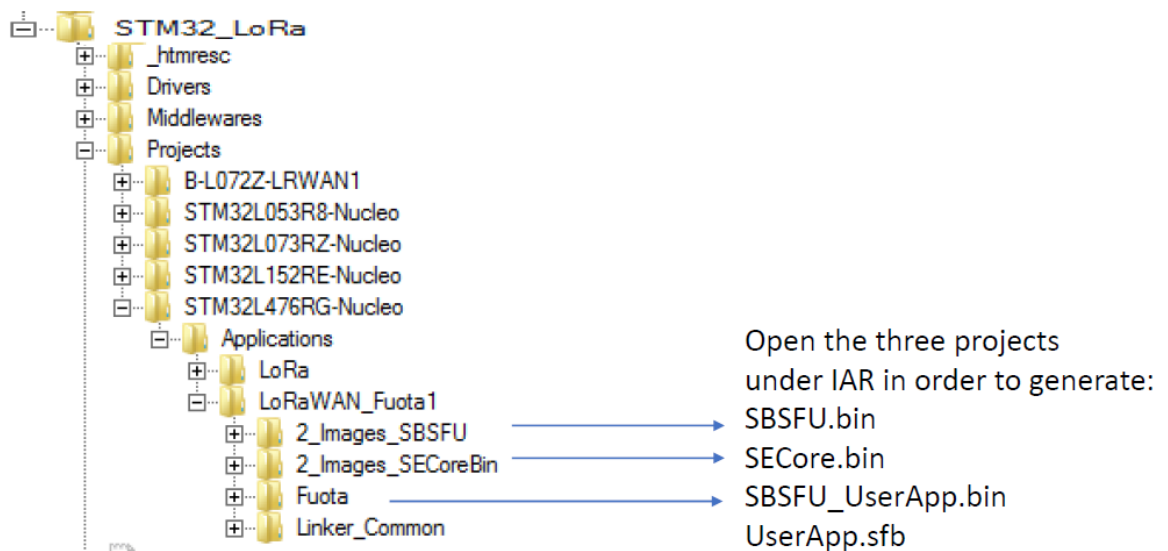
7.2.1 How to generate a FUOTA application

The following steps must be followed to generate a FUOTA application.

1. **SECore binaries**

   This step is needed to create the SECoreBin engine binary including all the required 'trusted' code and keys. The binary is linked to the SBSFU code in step 2.

2. **Secure boot and secure firmware update (SBSFU)**

   This step compiles the SBSFSU source code implementing the state machine and protection configurations. It links the code with the SECoreBin engine binary, including the 'trusted' code. It also generates a file that includes symbols used by the user application to call the SE interface functions.

3. **User Application (FUOTA)**

   This step generates the user application binary file (FUOTA) that is is uploaded to the device by the SFU process (UserApp.sfb).

   It generates a binary file concatening the SBSFU binary, the user application (FUOTA) in clear format with the corresponding FW header added (SBSFU_UserApp.bin).

Note: *For each step (1, 2 and 3), open the respective sub-project: 2_images_SECoreBin (step 1), 2_images_SBSFU (step 2), and end node (step 3) in the dedicated IDE folder. Then regenerate (make) the respective binary file. Each sub-project is configured (in the project options) in order to call the 'postbuild.bat' file when needed.*

See [7] for details on how to configure a complete SBSFU project.

Figure 17. **Application generations steps**
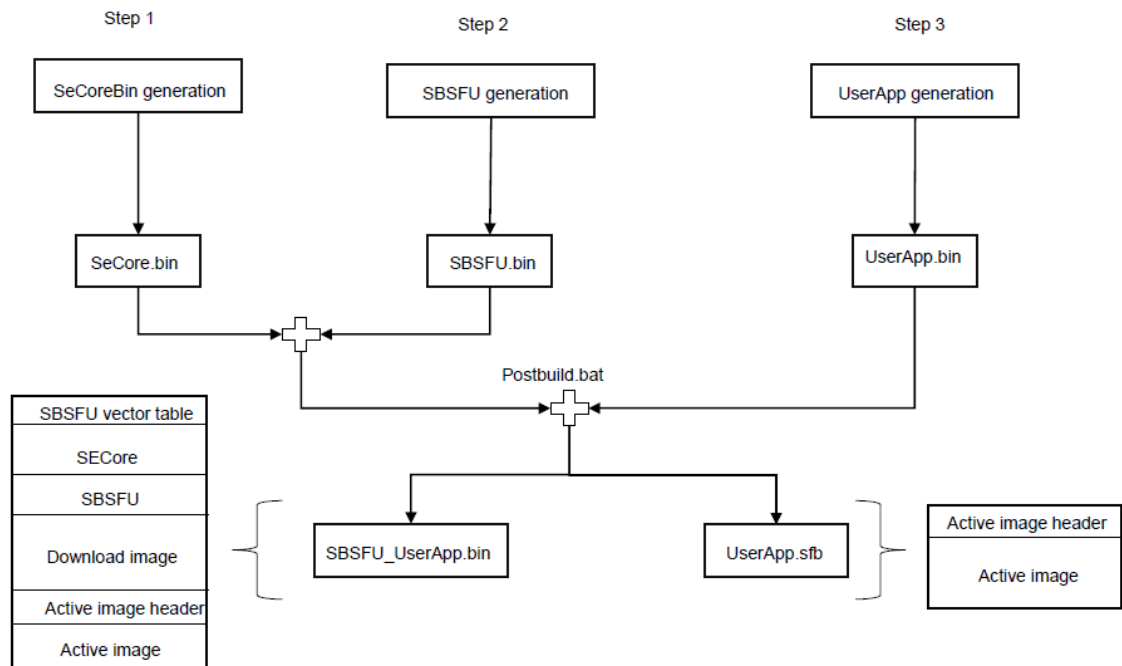
7.2.2 How to generate a data block (new firmware update)

To generate a new piece of firmware (N+1) to be downloaded over-the-air to allow the old current firmware (N) to be updated, the I-CUBE-LRWAN package is delivered with the 'prepareimage' firmware image tool.

This tool comes from the original X-CUBE-SBSFU STM32Cube Expansion Package. For a complete and detailed description of this tool, see [7] (Appendix E).

'Prepareimage' flow:

Even if only the full-firmware update feature is supported, the *diff* option of the prepareimage tool should nevertheless be systematically applied. In the best case, code modification is located in adjacent sections, and the *diff* may generate a useful *UserApp_To_Download.bin* file. In the worst case, the *diff* result is not relevant and the *UserApp_To_Download.bin* file is the same as *User_app.bin*.

1. Generate the 'To_Download' image

```
python prepareimage.py diff -1 UserApp_v1.bin -2 UserApp_v2.bin
UserApp_To_Download.bin -a 16 --poffset UserApp_To_Download.offset
```

2. Generate the clear complete firmware tag (SHA256 of the clear FW)

```
python prepareimage.py sha256 UserApp_v2.bin UserApp_v2.sign
```

3. Generate the clear 'To_Download' firmware tag (SHA256 of the clear FW)

```
python prepareimage.py sha256 UserApp_To_Download.bin UserApp_Download.sign
```

4. Generate the .sfb firmware metadata (header) and the 'To_Download' binary

```
python prepareimage.py pack -k ECCKEY.txt -r 44 -p 1 -v 2 -f UserApp_v2.bin -t
UserApp_v2.sign --pfw UserApp_To_Download.bin --ptag UserApp_To_Download.sign --
poffset UserApp_To_Download.offset UserApp_To_Download.sfb
```

Note: *X-CUBE-SBSFU firmware image tool preparation readme.txt is in <project>Middlewares\ST \STM32_Secure_Engine\Utilities\KeysAndImages folder.*

Figure 18. File generation flow
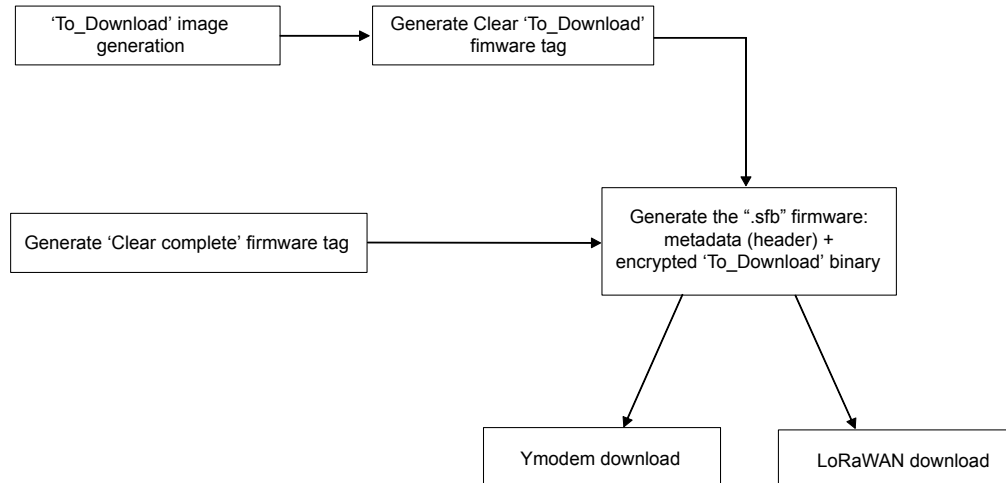
7.2.3 How to download the data block (full firmware) to the end-device

There are two ways to download the firmware:

- a local download via UART virtual COM using Ymodem protocol
- remote download via over-the-air mechanisms proposed by the LoRaWAN protocol.

The Ymodem protocol should be used during the development phase, whereas the LoRaWAn protocol is a main feature of the product. It is up to the user to choose the right approach.

For a complete description of Ymodem usage see [7].

7.2.4 How to create and manage a FUOTA campaign

This section does not aim to define, or show how to create, a FUOTA campaign on an application server. These aspects of a FUOTA campaign depend on the services provided by the network operator. Hence, here only the salient points relating to FUOTA campaign support are outlined.

The application server must:

- support the following packages:
  – the synchro package (TS-003) [5]
  – the fragmentation package (TS-004) [3]
  – the MulticastSetup package (TS-005) [4]
- support Class-C mode, as defined in the LoRaWAN specification V1.0.3 [1]
- be compliant with the 'interop test' proposed by the FUOTA working group of the LoRa Alliance
- have the capability to manage the data block (firmware image) to be downloaded.

'Interop test' is the minimum test proving that the end-device is able to receive a data block file from the server. This minimum test is shown in Section 3.5 Network/end-device interworking.

7.2.5 How to debug the end-node application

The complete system consists of a secure boot and an end-node application. When the target resets, the secure boot starts first. After some secure-boot checking the system jumps to the entry point of the end-node application. Since the end-node application is linked to the secure boot, the end-node application (*UserApp.bin*) cannot be downloaded directly with the debugger. In order to debug the end-node application, the following flow must be respected:

1. Flash the target with the complete system (*SBSBF_UserApp.bin*) using the ST Link tools.
2. Once the target is Flashed, the sub-project can be attached to the running target. Debug (with breakpoints, watch variables, and so on).

3. Modifications can now only be done on the end-node application. There are two ways to reload the target with the new binary:

    a. Reload the complete system (SBSFU_UserApp.bin) as described in (step 1), and attach to the target (step 2)

    b. Load the end-node application only (UserApp.sfb) via the YModem. When the target is running, attach as described in Step 2.

For further details on how to debug an application running on SBSFU, see [8].

7.3 Software description

When the I-CUBE-LRWAN delivery is unzipped, the package has the following structure.

Figure 19. I-CUBE-LRWAN project structure
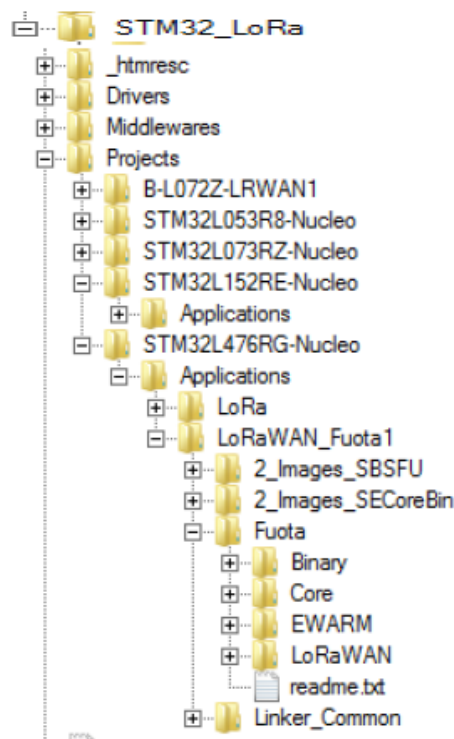
The I-CUBE-LRWAN package contains a FUOTA project. The FUOTA project is made up of three sub-projects: SECoreBin, SBSFU and End-node. Each sub-project has a dedicated IDE folder (IAR IDE environment) containing the *Lora.eww* file to activate in order to start the IAR IDE debugger.

7.3.1 Compilation switches

7.3.1.1 Crypto switches

SE_CoreBin instantiates the crypto scheme selected with *SECBOOT_CRYPTO_SCHEME*.

Table 12. Crypto switch descriptions

Symbols	Description	Default state
SECBOOT_ECCDSA_WITHOUT_ENCRYPT_SHA256	No FW encryption, only authentication and integrity are ensured.	Enabled
SECBOOT_ECCDSA_WITH_AES128_CBC_SHA25	Authentication, integrity, and confidentiality are ensured.	Disabled
SECBOOT_AES128_GCM_AES128_GCM_AES128_GCM	Authentication, integrity, and confidentiality are ensured.	Disabled

7.3.1.2 Security switch

SBSFU instantiates the security item selected through *SECBOOT_DISABLE_SECURITY_IPS*. When this symbol is defined, all IP security protections are disabled (WRP, RDP, IWDG, DAP, and so on). See [8]

Table 13. Security switch description

Symbols	Description	Default state
SECBOOT_DISABLE_SECURITY_IPS	Disables all security IPs simultaneously when activated.	Enabled.

7.3.1.3 Debug switch

In *\Projects\NUCLEO-L476RG\Applications\2_Images_OSC\End_Node\LoRaWAN\App\inc\hw_conf.h*

- Debug mode can be enabled by commenting out the *#define DEBUG* line
- Low-power mode can be enbled by uncommenting the *#define LOW_POWER_DISABLE* line.

7.3.1.4 Sensor switches

When no sensor expansion board is plugged into the setup, the #define SENSOR_ENABLED line must be commented out in *\Projects\NUCLEO-L476RG\Applications\2_Images_OSC\End_Node\LoRaWAN\App\inc\hw_conf.h*

7.3.1.5 Switch options

Table 14 summarizes the main application configuration options.

Table 14. Switch options

Functon	Switch option	Definition	Where
LoRa stack	OVER_THE_AIR_ACTIVATION	Application uses over-the-air activation procedure	Commissioning
	STATIC_DEVICE_EUI	Static or dynamic end-device identifying	Commissioning
	STATIC_DEVICE_ADDRESS	Static or dynamic end-device address	Commissioning
	REGION_868	Enable the EU band selection	Compiler option setting
	REGION_915	Enable the US band selection	Compiler option setting
Debug	DEBUG	Enable MCU debug mode in Sleep, Stop and Standby modes	*hw_conf.h*
Sensor	SENSOR_ENABLED	Enable the call to the sensor board. It is up to the user to implement the function call to the relevant sensor.	*hw_conf.h*

8   Memory footprint

8.1   End-node application

Values given in Table 15 are measured for the following IAR (ARM Compiler 5.05) configuration

- optimization: optimized for size level 3
- debug option: off
- trace option: off
- expansion board: / SX1276MB1MAS

Table 15. Memory footprint figures

Item	Flash (bytes)	RAM (bytes)	Description
Application (user)	TBD		
+LoRaWAN ClassA/C	14953		
+App Layer ClockSync	636		
+App Layer Fragmentation	780		Memory footprint for the overall application (App_user + LoRa ClassA + LoRa stack)
+App Layer RemoteMCast	1704	76908	
+ LmHandler	1836		The RAM caching memory to receive the data block is 67 Kbytes
+ FragDecoder	1377		
+ Remainder	8620		
Total	61257		
			Total:
			61257 + 7981 (Lib) = 69238 bytes

8.2   SBSFU

Item	Flash (bytes)	RAM (bytes)	Description
SBSFU	Code 22536		*SE_Core.bin*
	Data 27405	6709	Part of SFU
	Data 5776		MbedTLS imposes a 4 Kbyte stack
			Total:
			22 536 + 33 268 = 55804 bytes

Revision history

Table 16. Document revision history

Date	Revision	Changes
04-Nov-2019	1	Initial version.

Contents

List of tables

List of figures

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.