Download

« RETURN TO TECHNICAL DOCUMENTS

Introduction

Being able to perform remote firmware upgrades over-the-air (FUOTA) is becoming a requirement for most Internet of Things (IoT) devices. The details of any FUOTA procedure are tightly linked to the MCU architecture used by the device. However, any FUOTA system requires an efficient and secured file delivery system to reliably push the firmware patch file to (potentially) many devices simultaneously. LoRaWAN® provides such a secured and reliable file distribution system using multicast. This service is provided by a set of application layer packages. This paper presents two of those packages, which form the basis of the LoRaWAN file distribution service: the “Remote Multicast Setup” package and the “Fragmented Data Block Transport” package.

Updating Firmware Over-the-Air

Updating firmware over-the-air is a complex task. Generally, the following steps are involved:

1. Identifying the devices that need to be updated and putting them in multicast groups
2. Generating a binary firmware patch file specific to the device platform to be upgraded
3. Signing this binary file with a private key to guarantee integrity and authenticity
4. Pushing the binary file to the group of devices to be upgraded
5. Each device verifies the signature to authenticate the file and its origin
6. Each device then installs the firmware update
7. Each device reports the status of the upgrade operation (success or failure). This information is collected by a device management platform which is used to monitor the status of the devices in the field.

Some of these steps (2, 3, 5, and 6) are very device-specific, and depend on many design choices, including but not limited to:

- Device MCU architecture (ARM, X86, etc.)
- Device memory capacity
- Whether or not the device uses an operating system
- Whether or not the device uses a secure boot-loader
- Whether or not the device firmware was designed for partial updates
- Whether or not the device features a cryptographic hardware accelerator

Such constraints make it impossible to standardize a FUOTA process that will work for every device in every LoRaWAN network. An additional challenge is the low bit rate intrinsic to LoRaWAN networks.

However, efficient and reliable transport to push firmware patches to a large group of devices through a LoRaWAN network *is* possible. Furthermore, LoRaWAN has a radio multicast capability. This means that a given radio packet transmitted by the network (containing a fragment of the firmware patch) may be received by many devices. This way, the patch file does not need to be transmitted individually to each device. FUOTA over LoRaWAN overcomes the low-bit rate issue by transmitting the patch file to all devices at once.

Firmware Updates with LoRaWAN

With LoRaWAN, there are two primary application layer packages for FUOTA:

- Multicast remote setup
- Fragmented data block transport

These application-layer packages are defined by the FUOTA Working Group within the LoRa Alliance®. The specifications are publically available on the LoRa Alliance website.

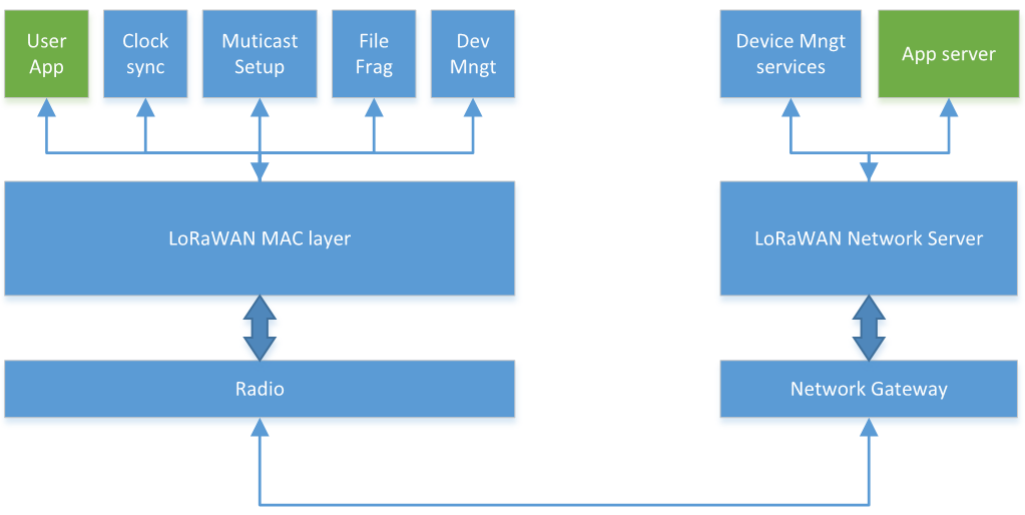

Figure 1: LORAWAN APPLICATION LAYER PACKAGES FOR FUOTA

On the device side, these packages live at the application layer and use the LoRaWAN MAC layer to transmit and receive packets to and from the network. Additionally, other packets may coexist at the device application layer, and generally some user application code will be present as well.

Each package on the device corresponds to a service implementation on the server side. For example, the “clock synchronization” *package* communicates with a clock synchronization *service* implemented on the back end. In Figure 1, all back end services related to FUOTA are grouped together in a single instance called the “Device Management Service.” Each of these packages uses a different port to communicate over the air. This way, the different data streams that correspond to each package can easily be separated in both the uplink and downlink directions.

LoRaWAN Multicast Group

The first step in updating firmware over-the-air in a LoRaWAN network is to identify and list the devices that need to be updated and place them into a LoRaWAN multicast group. This group can be created dynamically and be set up over the air. To do this, use the functions offered by the “Remote Multicast Setup” package.

Remote Multicast Setup Package

Every device in a LoRaWAN-based network is equipped with a unicast identity consisting of a device address (32 bits) and a unicast security context (a set of AES128 keys) that are used to uniquely identify and authenticate the device on the network.

Such a device may also belong to as many as four multicast groups. Multicast groups have the following characteristics:

- A multicast group address
- A multicast security context
  - *McAppSKey*: a multicast application key used to encrypt the payloads delivered to the group
  - *McNwkSKey*: a multicast network session key used to compute the Message Integrity Check (MIC) field of the packets that are sent to the group
- A multicast frame counter

If a device is part of more than one multicast group, the addresses of those groups must be distinct in order to differentiate among them.

The multicast security context is identical for all devices belonging to a given group; however, different multicast groups have different security contexts. They must also use different keys.

The “Remote Multicast Setup” package offers commands allowing the following operations:

1. Querying the package’s implementation version
2. Querying the list and status of the multicast groups to which a device has been assigned
3. Creating or modifying a multicast group for a given device
4. Deleting a multicast group definition from a device
5. Defining a Class B or Class C broadcast session and associating it with a multicast group

The **McGroupSetup** command allows you to create the complete context of a multicast group for a given device. This command embeds a mechanism to securely transport the multicast group keys over the air. This is achieved by transporting the multicast keys that are encrypted using a device-specific transport key. It also provides the minimum and maximum valid frame counter values and improves the security of the process by essentially limiting the lifetime of a multicast group.

Creating the context, however, is not sufficient for operating a multicast group.

LoRaWAN-based devices are low power and rely on a battery. These devices normally wake up only when they have data to transmit and then immediately go back to sleep. Therefore, to be able to transmit a frame to an entire group of devices, the system must first make sure that all of the devices within the multicast group have their receivers active at the same time and for the same duration.

To address the issue of ensuring that the receivers are all active at the same time, LoRaWAN networks use the GPS time (GPS epoch) as a time reference (This was chosen over UTC time because GPS time can be represented simply by an unsigned 32-bit counter, incremented by exactly 1 every second, independent of leap seconds. For an in-depth understanding of leap seconds, see https://en.wikipedia.org/wiki/Leap_second).

There are different ways to synchronize the clocks of LoRaWAN-based devices to the network’s time:

1. The device may periodically query the network time using the corresponding MAC layer command (The **DeviceTimeReq** command is available in the LoRaWAN specification version 1.0.3 and later)
2. The device may listen for the periodic Class B beacon, which contains the GPS time
3. The device may use a dedicated package called “Application Layer Clock Synchronization”

The decision of which techniques to use will be based on the trade-offs of power consumption, application requirements, and timing precision. A discussion of these trade-offs is beyond the scope of this document.

Once the device clocks are synchronized, the Device Management Service must program the devices to open their receivers simultaneously, and to use the same radio channel and data rate. For this purpose, the Remote Multicast Setup package allows you to define either a Class C or Class B multicast session.

Class C multicast sessions are defined by a start time, a duration, and a set of radio parameters (channel and data rate). When the devices’ clock time reaches the start time, the device programs its radios using the designated radio parameters and they turn on their receivers. The network can then broadcast packets using the same radio parameters at any moment during the Class C session.

This method is simple and does not require a very accurate timing synchronization. A few seconds of drift among devices will not create any issues. However, for the duration of the Class C session, the devices in the group leave their receivers on continuously. Therefore, this method is not power-efficient in regions where the network is obliged to respect a low transmission duty cycle (for example, less than ten percent in Europe). A ten percent duty cycle means that 90 percent of a device’s energy is wasted during a Class C multicast session.

This being the case, the package also supports using Class B multicast sessions. When a Class B multicast session is used, all the devices in a group must first synchronize with the Class B beacon (which broadcasts every 128 seconds) before the session starts. Once the session has begun, the devices in the group open periodic reception slots simultaneously, and they stay open for the duration of the session. This enables the devices to duty-cycle their receivers while still spending most of their time asleep.

Note: The network might not use every one of the available slots to transmit information. However, Class B mode is slightly more complicated to implement than Class C, and requires the network to broadcast the Class B beacon.

LoRaWAN is one of very few radio protocols that enables true multicast transmission over the air. For example, Cellular networks emulate multicast transmissions by setting up as many simultaneous unicast sessions as there are devices in a group. However, in this scenario, the data is actually sent as many times as there are devices in the group, rather than just once.

The Multicast Remote Setup package does not make any assumptions about the nature of the data frames being transmitted by the network during a Class C or Class B multicast session. The data frames, for instance, could be individual control frames (ones that turn an entire group of streetlights on and off, for example).

Therefore, during a multicast session, it is necessary to implement a means of efficiently fragmenting a file and ensuring complete delivery to every device in the group. To this end, the ”Fragmented Data Block Transport over LoRaWAN” package has been created. We will discuss this in the next section.

Transporting a File to a Multicast Group

The use of radio multicast transmissions improves the efficiency of the network tremendously. Each frame is transmitted only once, and can be received by all devices in the group simultaneously, rather than having to transmit the same frame individually to each device. However it comes with a challenge, namely, coping with missed receptions. Consider the following example:

A network broadcasts 100 fragments of a file to a group of 1,000 devices assembled in a single multicast group. The radio channel experiences interference, causing each device to randomly lose ten percent of the frames that are broadcast by the network. If the network were transmitting to a single device (unicast transmission) this would not be a big deal. In this case, while a fragment might need to be repeated when lost (which occurs about ten percent of the time) on average it would take 110 transmissions to successfully push 100 data fragments to a device.

However, this is more complicated in a multicast group. The first fragment broadcast by the network, on average, is received by 900 of the 1,000 devices. This means that 100 devices do not receive the first fragment. Therefore, the network transmits it again, this time (on average) ten of the remaining 100 devices will receive it. The network transmits the fragment for a third time, and one device will likely still miss it. So, on average, it takes approximately four transmissions of the same data to ensure that all devices receive the complete fragment. This number only grows if the number of devices in the group or the error rate are increased.

Thus, a different method must be used to guarantee that all devices receive the full file without having to repeat every fragment multiple times.

To overcome this problem, the "Fragmented Data Block Transport" package implements an erasure correction code. The principle behind this approach is this:

1. The file to be sent is fragmented into *N* equal-length fragments, such that each fragment can fit in a single LoRaWAN broadcast payload. In the case of our example, *N* = 100 fragments of equal length.
2. For those *N* uncoded fragments, the erasure code is used to generate *N + M* coded fragments, where *(N + M)/N* is the redundancy ratio. That redundancy ratio must be greater than the packet loss rate we want to compensate for, plus a margin. In our example, assuming a typical ten percent packet lost rate, a 20 percent redundancy ratio would be appropriate.
3. Those coded fragments are designed in such a way that any subset of *N* coded fragments out of the *(N + M)* generated fragments can be used to rebuild the *N* original file fragments, and therefore ensure transmission of the complete file.

Again, in the case of our example, the Device Management Service would divide the file into 100 fragments and generate 120 coded fragments. Those 120 coded fragments are then broadcast by the network during a Class C or Class B multicast session. Every device listening may lose a different subset of fragments during the session. However, as soon as a device receives at least 100 coded fragments out of the 120 that are broadcast by the network, the device is able to reconstruct the complete file.

Where the repetition method would lead to broadcasting 400 frames, using this coding scheme allows the network to achieve the same success rate while broadcasting only 120 frames.

In addition to the required coding/decoding methods just described, the "Fragmented Data Block Transport" package provides the following functionality:

- Querying the implementation version of the fragmentation package
- Creating a fragmentation session
- Requesting a device or group of devices to provide the status of a fragmentation session
- Deleting a fragmentation session

Prior to sending a file to a device or group of devices, a fragmentation session must be created, providing the device(s) with the details of the file that will be broadcast.

The **FragSessionSetupRequest** command is used with the following parameters:

- A fragmentation session ID. A device may have up to four active fragmentation sessions at the same time. This ID is used to differentiate them from one another.
- The fragment size expressed in bytes.
- The number of fragments required to reassemble the file being transported in the given session (ranging from 1 to 65535).
- The number of padding zeroes contained in the last fragment (The zeroes may be required if the file length is not an exact multiple of the fragment length. In this case, the zeroes would be added as padding for the last fragment length).
- A file descriptor. This is a freely-allocated four-byte field. If the file transported is a firmware patch image, this field might—for example—encode the version of the firmware being transported, allowing compatibility verification upon reception by the end device. The encoding of this field is application-specific.

Once the **FragSessionSetupRequest** command has been acknowledged by all devices within the multicast group, the coded fragments can be transmitted.

Although multicast transmissions are more efficient when transmitting the same file to a group of devices, the fragmentation package allows us to send fragments using a unicast downlink as well.

After all coded fragments have been broadcast, the server may query the status of each device using the **FragSessionStatusRequest** command. This command may be sent as either a unicast or multicast frame.

Devices respond to this command with the following information:

- The status of the fragmentation session: File Successfully Reconstructed, or an error code, if the file could not be reassembled
- The total number of fragments received during the fragmentation session
- The number of missing fragments received before the file could be reassembled

Additional coded fragments may be sent to devices that have not yet been able to entirely reassemble the file. Those fragments may be sent as unicast downlinks, or a new multicast session can be created if too many devices have been unable to reassemble the file.

Summary

In this paper we introduced the "Fragmented Data Block Transport" and "Remote Multicast Setup" application layer packages, and discussed how LoRaWAN provides an efficient and reliable fragmented-file delivery service. We also explained how using the fragmented-file delivery service can be used to push binary firmware updates over-the-air to large groups of devices. This is achieved by using the LoRaWAN protocol's unique over-the-air multicast capability in conjunction with an efficient fragment coding scheme, which significantly reduces the number of repeated transmissions required.