NEW RADIO COMMUNICATION FOR NEW ERA

NRC7394 Application Note

(Firmware OTA)

Ultra-low power & Long-range Wi-Fi

Ver 1.2
Oct. 24, 2025

NEWRACOM, Inc.

NRC7394 Application Note (Firmware OTA)
Ultra-low power & Long-range Wi-Fi

Office

Newracom, Inc.
505 Technology Drive, Irvine, CA 92618 USA
http://www.newracom.com

Contents

List of Figures

1 Overview

Firmware Over-the-Air (FOTA) enables manufacturers and developers to remotely deliver crucial firmware updates to devices post-deployment, eliminating the need for direct access to each device. Within the current application framework, two distinct firmware update procedures are supported. The initial approach entails periodic server polling to identify accessible firmware updates. The alternative method revolves around the reception of incoming firmware updates via the beacon frame.

This user guide plays a vital role as a supplementary resource accompanying both the sample FOTA update application and the broadcast FOTA capability. It aims to provide comprehensive insights and instructions for utilizing these features effectively.

1.1 Unicast FOTA

The reference 'sample_fota' application aims to showcase the functionality of FOTA using a basic Python-based HTTP server. However, it is also possible to employ any other custom HTTP server capable of communicating through the FOTA protocol as an alternative.

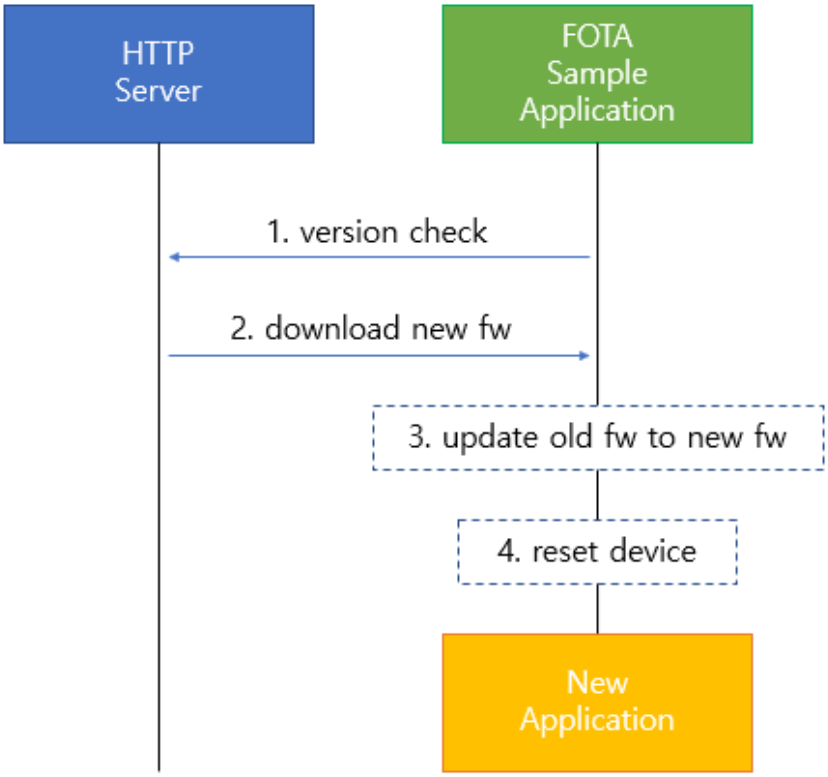

Figure 1.1 FOTA procedure

In the example application "sample_fota," the module follows a periodic fetching approach where it retrieves the new firmware version identifier every 10 seconds from the HTTP server. If a newer version is found on the server, the module proceeds to download and update the firmware binary accordingly. For a more comprehensive understanding of this procedure, you can refer to Chapter 2 and Chapter 3, which provide detailed information about the update process.

1.2 Broadcast FOTA

In contrast to the periodic polling approach used in "sample_fota," the broadcast FOTA is designed to command devices to update their firmware. This feature enables the simultaneous upgrade of a large number of devices by sending small chunks of firmware data in beacon frames from an access point. When the access point initiates the broadcast FOTA, all devices paired with the access point prepare to listen for incoming beacons containing firmware data chunks. The process to initiate the broadcast FOTA is detailed in Chapter 4, providing a step-by-step illustration of the procedure.

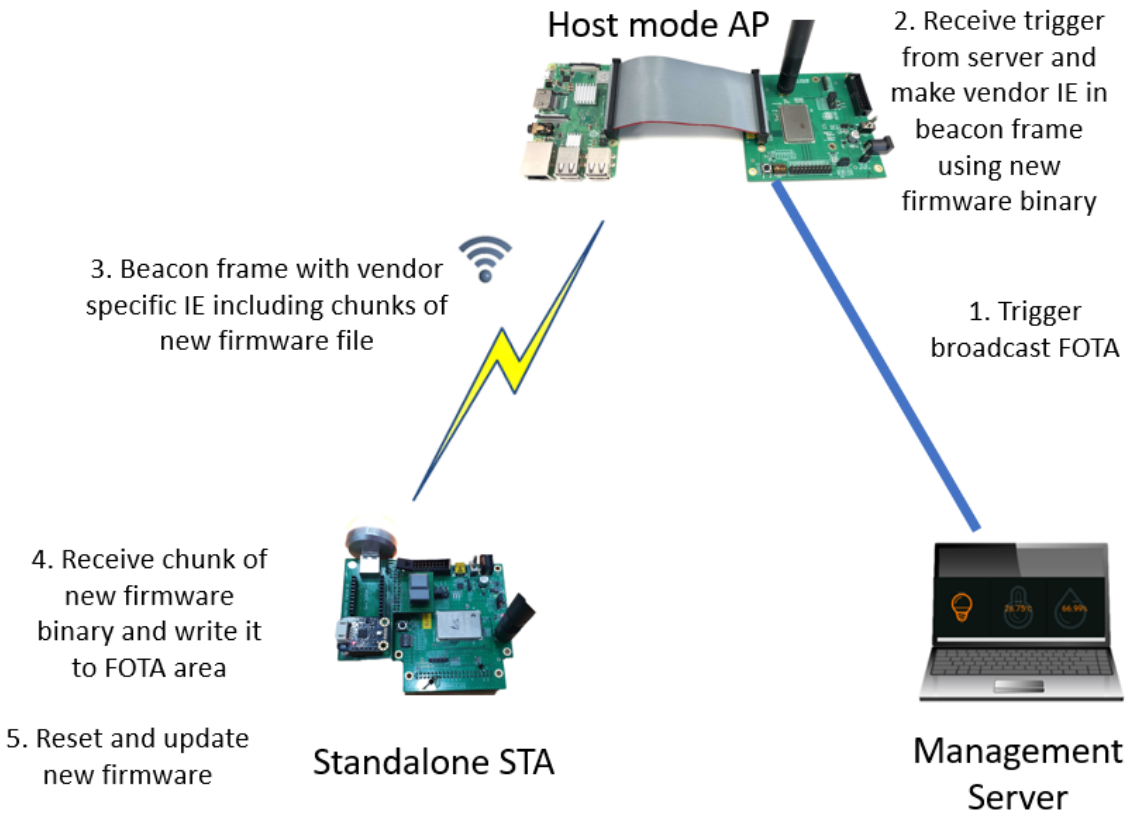

Figure 1.2 Broadcast FOTA procedure

2 HTTP(S) Server

For the example application, a Python-based Simple HTTP Server will be utilized, which functions as both a web server and a file server. To start the server, a simple one-liner terminal command can be used:

```
$python3 -m http.server <port number>
```

In addition, the sample FOTA application also supports an HTTPS environment with MbedTLS implementation. For executing the HTTPS Server, a separate script file is provided. Once the server is running, it can be accessed remotely using a browser by entering the IP address and port number of the machine hosting the server. For instance, if the server's host IP is 192.168.1.11 and the port is set to 12345, typing "192.168.1.11:12345" in the browser's search bar will redirect to the index page of the server. The initial page displayed in the browser will show the list of files in the directory from which the server was started. Further instructions on setting up the server will be covered in a later chapter, providing more detailed information.

Figure 2.1 Directory listing via a web browser

The directory contains three types of files:

file name	Description
***.bin:**	The files ending with the '.bin' extension are firmware binaries.
crc.py:	This Python script is used by the server to compute the CRC bits of firmware binaries for integrity check.
version:	The JSON formatted version file consists following fields:

1. “version” : the version identifier of the latest firmware,
2. “crc” : the computed CRC bit sequence corresponding to the latest firmware binary and
3. “fw_name” : the file name of the latest firmware binary.
4. “force” : the force update flag : 1(true) or 0 (false)

```
{
    "version" : "1.1.0",
    "crc" : "3cad437a",
    "force" : "0",
    "fw_name" : "nrc7394_standalone_xip_hello_world.bin"
}
```

Figure 2.2 An example version file

```python
#!/usr/bin/python
import sys
import zlib

def unsigned32(n):
    return n & 0xFFFFFFFF

def getCRC32(filename):
    try:
        f = open(filename, 'rb')
        crc = zlib.crc32(f.read())
        return unsigned32(crc)
    except IOError:
        print("Error: Cannot fine such file.")
        #return -1
        exit(1)

if len(sys.argv) == 1:
    print("Error: There's no file name")
    print("Usage:")
    print("    $python crc.py <file name>")
    exit(1)

print("%x" % getCRC32(sys.argv[1]))
```

Figure 2.3 CRC32 Python script (crc.py)

3 FOTA Sample Procedure

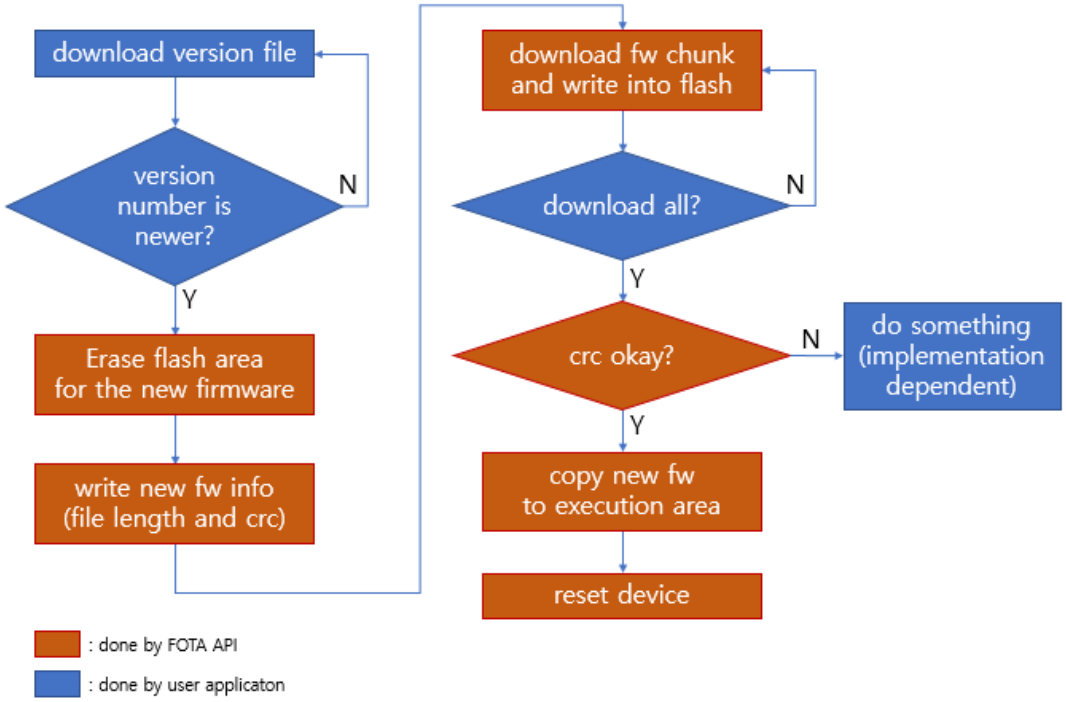

Figure 3.1 FOTA procedure in sample application

The diagram above shows the flowchart of the sample FOTA procedure. The module starts out by periodically fetching the version file from the HTTP server every 10 seconds. The version file contains information including the latest firmware version identifier, the CRC bits corresponding to the firmware binary file and the file name of the latest firmware binary. If the version available on the server is newer than the existing version on the module, the module proceeds with the update procedure.

The update procedure consists of multiple steps. First, the module erases the flash address block designated for the meta-information (file length and expected CRC bits) and the firmware binary. Then, the meta-information derived from the version file and the HTTP header is written to the flash and the download begins. Once the download is complete, the CRC bits corresponding to the downloaded firmware binary is calculated once again on the module and compared with the expected CRC bits stored in the received meta-information. If the two CRC bit sequences match, the new firmware binary is copied to the XIP memory block and the module will reset on its own to run the new firmware binary. A mismatch between the two CRC bit sequences indicates the corruption of the downloaded binary file and its handling is implementation-dependent, although a reasonable approach would be to reattempt the download procedure.

The FOTA API is required for implementing the FOTA procedure. The SDK API document contains more detailed information about the FOTA API usage.

3.1 FOTA server location

The FOTA HTTP server can either run on the AP or elsewhere on another external host.

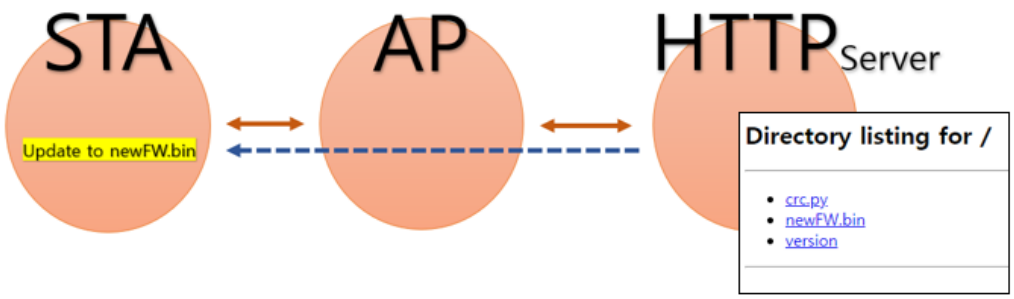

Figure 3.2 FOTA network diagram

3.2 FOTA sample application file configuration

```
#if defined( SUPPORT_MBEDTLS ) && defined( RUN_HTTPS )
#define SERVER_URL "https://192.168.200.1:4443/"
#else
#define SERVER_URL "http://192.168.200.1:8080/"
#endif
#define CHECK_VER_URL SERVER_URL "version"
#define CHUNK_SIZE 2048
```

Figure 3.3 configuration in sample_fota.c

The following parameters in the sample source file "**sample_fota.c**" are relevant for FOTA operation:

- **SERVER_URL.**
  The FOTA HTTP/HTTPS Server URL.
- **CHECK_VER_URL**
  EX.) 'http://192.168.10.199/version'
- **CHUNK_SIZE**
  Chunk Size Definition.

※ SAMPLE version is existed in sample_fota_version.h

3.3 FOTA server operation

The HTTP server requires three files: ‘crc.py’, ‘newFW.bin’, and ‘version’.

```
root@damonoh-H81N:~/FOTA# ls -al
total 644
drwxr-xr-x  2 root root   4096 11월  15 11:35 .
drwx------ 32 root root   4096 11월  15 14:01 ..
-rwxr-xr-x  1 root root    501 10월  28 15:24 crc.py
-rwxr-xr-x  1 root root 642328 10월  29 17:50 newFW.bin
-rwxr-xr-x  1 root root     23 11월  15 11:03 version
```

Figure 3.4 Three files on HTTP_Server

To update the version file, calculate the CRC value of newFW.bin using 'crc.py’.

- Run crc.py ($ python crc.py newFW.bin)
- Update the generated CRC value to the ‘version’ file

```
root@damonoh-H81N:~/FOTA# python crc.py newFW.bin
97cb8611
```

Figure 3.5 Run crc.py

The firmware version, the CRC value of the new firmware and the name of the new firmware binary must be updated in the JSON version file.

```
{
    “version” : “110”,      -> Update firmware if the version is higher than current version installed.
    “crc” : “97cb8611”,              -> Created CRC value
    “fw_name” : “new_FW.bin” -> New firmware file name (defined by user)
}
```

Figure 3.6 Edit ‘version’ file

HTTP Server execute depending on whether Python2 or Python3 is used, one of the following terminal commands can be used in the directory containing the three files to start the FOTA server operation.

python version	command
python2	python -m SimpleHTTPServer 8080
python3	python3 -m http.server 8080

```
root@damonoh-H81N:~/FOTA# python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

Figure 3.7 Starting the FOTA server operation (HTTP)

HTTPS Server execute depending on whether Python2 or Python3 is used (**Use 'Https_Server.py'**)

```
root@damonoh-H81N:~/FOTA# python3 Https_Server.py
Serving HTTP on 0.0.0.0 port 4443..

root@damonoh-H81N:~/FOTA# python2 Https_Server.py
Serving HTTP on 0.0.0.0 port 4443..
```

Figure 3.8 Starting the FOTA server operation (HTTPS)

3.4 FOTA Sample SDK operation (sample_fota)

In the sample_fota application, the module periodically fetches the latest firmware version identifier **every 10 seconds** from the HTTP server. If a new version is available (over 105), new firmware binary is downloaded, and the existing application is updated.

- Step 1) Check the firmware version in the server.
- Step 2) Format the flash memory block for downloading OTA firmware.
- Step 3) Download the new firmware binary and Update
- Step 4) Replace the old binary and Reboot(Execution new firmware)

```
========================================
STEP 1. Check firmware version in server.
========================================
[httpc_parse_url] scheme:http, host:10.198.1.214, port:8080, uri:/version
s1g: 0 update beacon interval(1000)
s1g: 0 listen_interval: 100 beacon_interval: 1000
s1g: 0 update short beacon interval(intv: 100)
s1g: 0 listen_interval: 100 short beacon_interval: 100
[httpc_connect] result:0, handle:0x00000000
--- Request Header ---
GET /version HTTP/1.1
Host: 10.198.1.214:8080

[httpc_recv] size=200

[httpc_recv] size=80

[httpc_recv] size=0
[httpc_close] httpc_close()
[HTTP Response Length] 280
-----Recvd Data-----
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.8.10
Date: Thu, 09 Sep 2021 17:27:16 GMT
Content-type: application/octet-stream
Content-Length: 80
Last-Modified: Wed, 08 Sep 2021 20:59:08 GMT

{
    "version" : "106",
    "crc" : "f8cbb153",
    "fw_name" : "new_FW.bin"
}

[parse_response_version] data length : 280, body length : 80
[parse_response_version] version data : {
    "version" : "106",
    "crc" : "f8cbb153",
    "fw_name" : "new_FW.bin"
}

[parse_response_version] version : 106
[parse_response_version] crc : f8cbb153
[parse_response_version] URL : new_FW.bin
[parse_response_version] version: 106,  crc: f8cbb153  fw_url: http://10.198.1.214:8080/new_FW.bin
====================================================
STEP 2. Erase flash area for OTA firmware to be downloaded.
====================================================
Erasing......Done.

========================================
STEP 3. Download new firmware and update.
========================================

[httpc_parse_url] scheme:http, host:10.198.1.214, port:8080, uri:/new_FW.bin
[httpc_connect] result:0, handle:0x00000000
--- Request Header ---
GET /new_FW.bin HTTP/1.1
Host: 10.198.1.214:8080

[httpc_recv] size=204
-- head content length: 204

[HTTP Response Length] 272866032
-----Recvd Data-----
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.8.10
Date: Thu, 09 Sep 2021 17:27:19 GMT
Content-type: application/octet-stream
Content-Length: 828224
Last-Modified: Wed, 08 Sep 2021 20:57:56 GMT


[parse_response_content] fw_len = 828224

FW Size: 828224
FW CRC: 0xF8CBB153
```

Figure 3.9 Sample FOTA application log (pre-update binary)

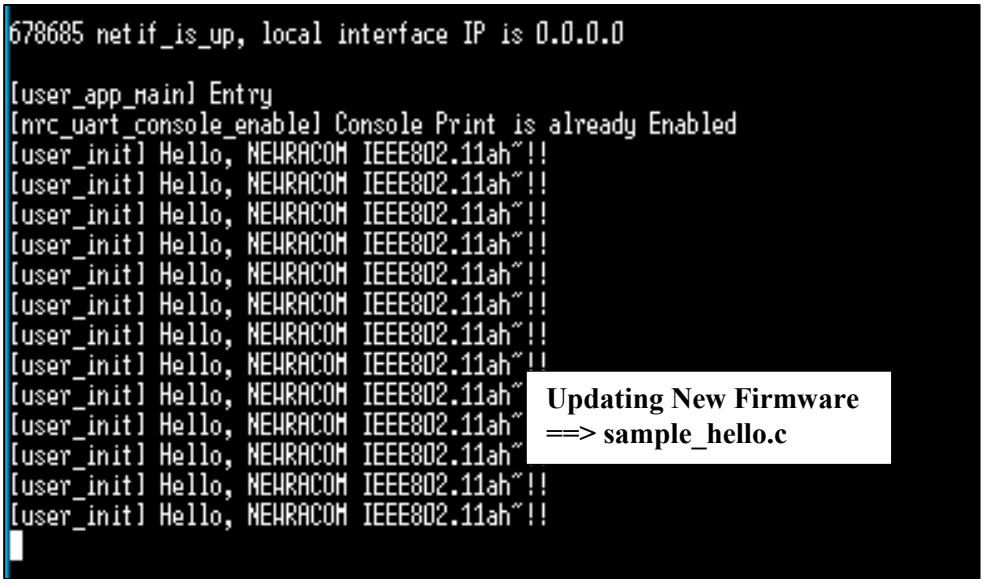

Figure 3.10 Sample FOTA application log (post-update binary)

4 Broadcast FOTA

Broadcast FOTA is a feature designed to enable multiple stations that are typically connected to an Access Point (AP) to receive new firmware updates via a vendor-specific Information Element (IE) embedded within the beacon frame. This innovative method facilitates the distribution of firmware updates to a multitude of devices simultaneously, primarily utilizing a wireless network infrastructure. The overarching goal is to streamline the firmware update process by efficiently disseminating these upgrades to numerous devices linked to a central AP.

In essence, broadcast FOTA enhances the efficiency of firmware updates by ensuring that multiple devices receive the necessary updates in a synchronized manner. This is particularly useful for scenarios where a large number of devices need to be updated concurrently, such as in IoT deployments or large-scale networks.

Broadcast FOTA relies on three key elements: app_name, app_version, and chip id. These components collaborate to ensure precise delivery of firmware updates to specific devices. By effectively utilizing these elements, broadcast FOTA establishes an efficient approach for delivering updates across diverse connected devices. Notably, the chip id is determined within the running firmware, while the app_name and app_version are set using APIs in firmware.

4.1 Access point preparation

Since the AP firmware is distributed binary only, the host mode nrc_pkg version 1.4.1 and after will have the broadcast enabled by default.

There is a python script that will initiate the broadcast can be found in host mode SDK package as sample_broadcast_fota.py and broadcast_fota.py.

4.2 Firmware preparation

The firmware that will support broadcast FOTA should call API’s below to initialize the device to prepare for incoming firmware data.

```
nrc_bcast_fota_init();
nrc_bcast_fota_enable(true);
nrc_bcast_fota_set_mode(BC_FOTA_MODE_CONNECTED);
nrc_set_app_version(&app_version);
nrc_set_app_name(“application name”);
```

Once the firmware prepared with above API’s initiated, the firmware in action will start accepting new firmware from the broadcast FOTA server and start the upgrading process.

For the detailed reference how this is working by checking sample_bcast_fota and sample_softap_uart_tcp_server.

4.2.1 nrc_bcast_fota_init

Initialize broadcast FOTA.

4.2.2 nrc_bcast_fota_enable

Enable/Disable broadcast FOTA.

4.2.3 nrc_bcast_fota_set_mode

Set broadcast FOTA mode
BC_FOTA_MODE_ANY – Run broadcast FOTA without AP connection
BC_FOTA_MODE_CONNECTED – Run broadcast FOTA when AP connection made.

4.2.4 nrc_set_app_version

This function is used to set the application version. The version is represented using the VERSION_T structure, which includes the major, minor, and patch components of the version number. It is mandatory for using broadcasting FOTA.

4.2.5 nrc_set_app_name

Set application name. It is mandatory for using broadcasting FOTA.

4.3 Starting Broadcast FOTA

This section explains two ways to run Broadcast FOTA from the access point (AP):

- 4.3.1: with a Python helper script
- 4.3.2: with a standalone Linux binary

Both methods inject firmware chunks into Wi-Fi beacons so paired stations can receive, verify and install the image.

4.3.1 Using python script.

Prepare new firmware and place it in the same folder with broadcast_fota.py and sample_broadcast_fota.py.

Prepare stations and pair with the access point.

Initiate broadcast FOTA by starting sample_broadcast_fota.py.

sample_broadcast_fota.py will accept 8 arguments – firmware file name, chip id (i.e., 7292), firmware version (dot separated firmware version, i.e., 1.0.0), application name, app version (dot separated application version, i.e., 1.0.0), beaconing mode, retry (whether this session is retry or not), and forcefully update without checking version.

python2 sample_broadcast_fota.py [filename] [chip id] [fw ver] [app ver] [app name] [mode] [retry] [force]

i.e. python2 sample_broadcast_fota.py nrc7292_standalone_xip_sample_bcast_fota.bin 7292 1.0.1 1.0.0.0 sample_bcast_fota 1 0 0

※ Note that current broadcast scheme only supports fast mode (1) which disable short beacon, and set full beacon interval to 10 ms. In the future update, there will be normal mode (2) which disable short beacon, and full beacon interval will remain at the same and low traffic mode (3) which remain the short beacon as is, and full beacon interval will become 10 times of current beacon interval. Below picture is taken from the device console when the broadcast FOTA initiated from AP.

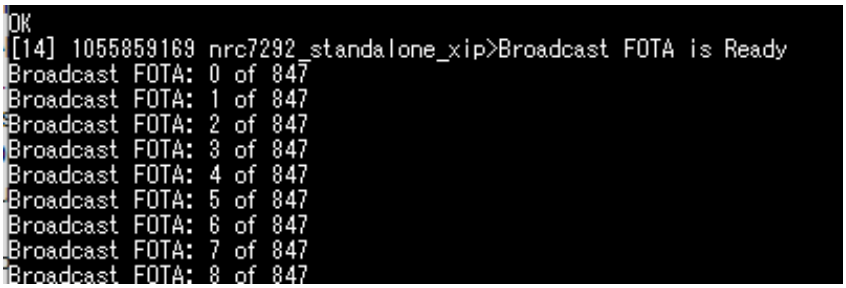

Figure 4.1 FOTA Initiation

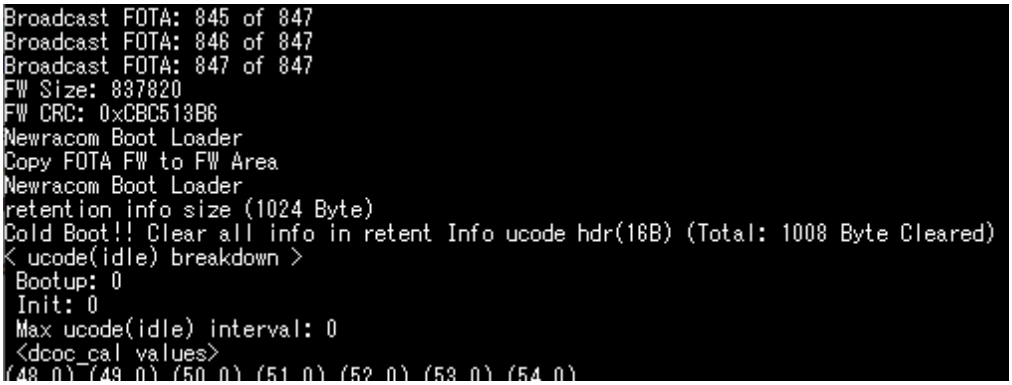

Figure 4.2 FOTA completion

4.3.2 Using the executable Binary

A C implementation, nrc_broadcast_fota.c, controls the network interface (via netlink) and uses "hostapd-cli" to append firmware chunks to becons.

Build requirements

Install dependencies:

```
sudo apt-get install libnl-3-dev libnl-genl-3-dev
sudo apt-get install libcjson-dev
```

Figure 4.3 Dependencies install

Build with the provided Makefile in the same directory as nrc_broadcast_fota.c :

```
make
```

Figure 4.4 Build

Prepare runtime files

Place these files together (same directory):

- The built binary: nrc_broadcast_fota
- The firmware image (e.g., nrc7394_standalone_xip_sample_bcast_fota.bin)
- A JSON configuration file named "bcast_fota.json"

Sample bcast_fota.json shown below.

```
{
        "server_ip"             : "192.168.200.1",
        "server_port"           : 3333,
        "filename"              : "nrc7394_standalone_xip_sample_bcast_fota.bin",
        "chip_id"               : "7394",
        "fw_version"            : "1.4.0",
        "app_version"           : "1.0.1",
        "app_name"              : "sample_bcast_fota",
        "ignore_version"        : 1,
        "auto_retry"            : 1,
        "retry_interval"        : 120,
        "polling_bcn_cnt"       : 3,
        "beacon_interval"       : 20,
        "first_recursion"               : 1,
        "mcs"                   : 10
}
```

Figure 4.5 Sample bcast_fota.json

Key fields (what they mean):

- server_ip, server_port: Control/telemetry socket used by the tool.
- filename: Firmware image to broadcast.
- chip_id: Target chip family identifier.
- fw_version, app_version, app_name: Versioning metadata for receivers.
- ignore_version: if 1, receivers will update regardless of their current version.
- auto_retry: If 1, the AP will automatically retry failed or incomplete sessions.
- retry_interval (seconds): Delay between retries when “auto_retry” is enabled.
- polling_bcn_cnt: Number of beacon intervals used for polling/progress pacing.
- beacon_interval (ms): Full beacon interval to use during broadcast (tool adjusts AP beaconing accordingly).
- first_recursion: Internal pacing/control flag for initial beacon scheduling (leave as is unless instructed).
- mcs: MCS index used during transmission (choose an index suitable for your link budget and station capability).

Run the broadcast

From the directory containing the files:

```
./nrc_broadcast_fota
```

Figure 4.6 Executing nrc_broadcast_fota

What happens on the device (receiver)

1. If the running firmware supports Broadcast FOTA, it erases the dedicated FOTA flash partition and prepares to receive chunks.
2. As beacons arrive, the device validates checksums and stores each chunk.
3. After all chunks are received, the device automatically reboots.
4. On boot, the bootloader validates the image in the FOTA partition.
   - If valid, it copies the image fro the FOTA partition to the primary firmware partition and boots the new firmware.

5 Revision history

Revision No	Date	Comments
Ver 1.0	7/17/2023	Initial version
Ver 1.1	8/30/2023	Added broadcast FOTA
Ver 1.2	10/24/2025	Added nrc_broadcast_fota how-to.