HUB'O:

DESCRIPTION OF THE EXCHANGES WITH THE SERVER

NOTICE

Embedded software is based on Nke Watteco proprietary drivers and applicative code and operates on the Contiki kernel from the SICS (Swedish Institute of Computer Science).

http://www.nke-watteco.com/

DOCUMENT HISTORY

Date	Revision	Modification Description
August 2017	1.0	Creation
January 2018	1.1	Adding versions, sensor silence alarm, etc.
July 2018	1.2	Correction on the KO answer request
February 2019	2.0	Add the modifications corresponding to firmware 02.00

CONTENTS

1 INTRODUCTION

Hub’O is the first local LoRaWAN network gateway designed by nke Watteco. Hub’O works in partnership with a distant http server that hosts or communicate with the applicative back-end of our client. It can works, as well, with a ModBus master through a TCP or RS485 link.

In order, for the complete system, to work correctly, the distant http server needs to have some specific behavior to communicate correctly with Hub’O.

Thus, this document describes all the different exchanges that can exist between the distant server and Hub’O. It details as well all the files exchanged between Hub’O and the distant server.

This is why this document firstly gives a general description about the link between these two entities. Then it gives a general way of sending a file to Hub’O and to receiving a file from it. Finally, the biggest part of this document gives all the details about the files themselves: how and when they have to be used, and their content.

2 HUB’O – SERVER LINK: GENERAL DESCRIPTION

Hub’O, the nke Watteco’s gateway, uses either Ethernet or GSM/GPRS/3G as the Physical link with the distant server.

Hub’O can work with both of these links, and can change dynamically the interface while running. The choice between Ethernet and GSM/GPRS/3G is done through a physical switch on Hub’O board.

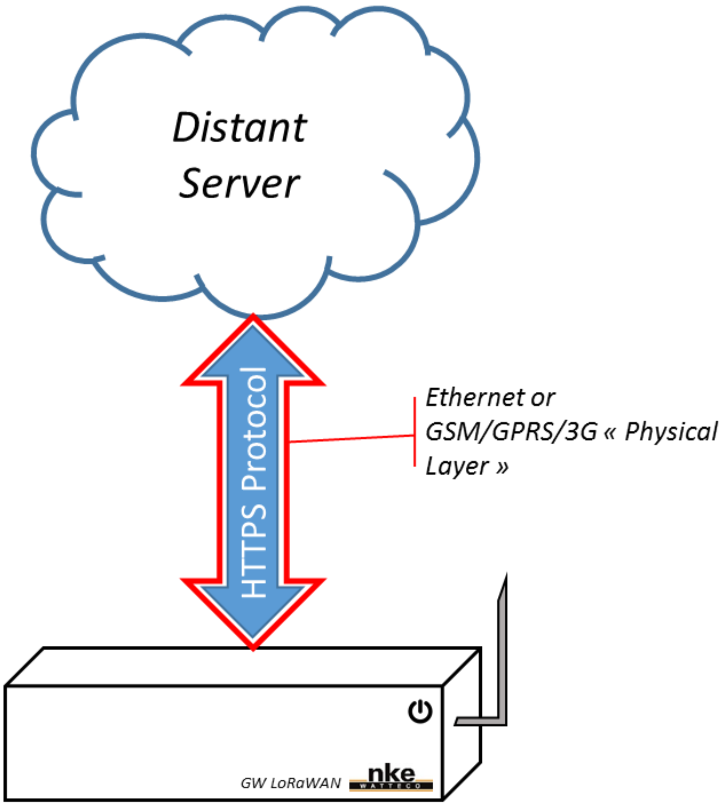

FIGURE 1 - HUB'O - SERVER LINK ILLUSTRATION

The files transfers between Hub’O and the distant server is done thanks to the HTTPS protocol. Thereby, Hub’O sends the different files (data, alarm, topology, etc.) with the POST request and uses the User-Agent header of the HTTPS protocol to identify itself. To get files from the distant server, Hub’O polls it regularly in order to see if the server has a new file to send.

Both of these operations (send and receive) are detailed in the following paragraphs.

3 How to Send a file to Hub’O

The Hub’O Gateway does not have a server running on its side, thus it is not possible to send directly a file to it.

This is why, Hub’O regularly polls the distant server in order to get what is called the “Link file”. The polling period is configurable by the distant server thanks to the Hub’O configuration file. The “Link file” actually contains all the files that Hub’O needs to download from the Server. So, the server has to update this “Link file” with the right names in order to make Hub’O download the right files.

Moreover, in order to optimize the data volume exchanged an ETag system is used. That means that Hub’O checks the ETag associated to the link file. If the ETag is the same as the last time, it means for Hub’O that nothing has changed. So it does not need to download it. But if the ETag has changed, Hub’O downloads all the files named in the link file.

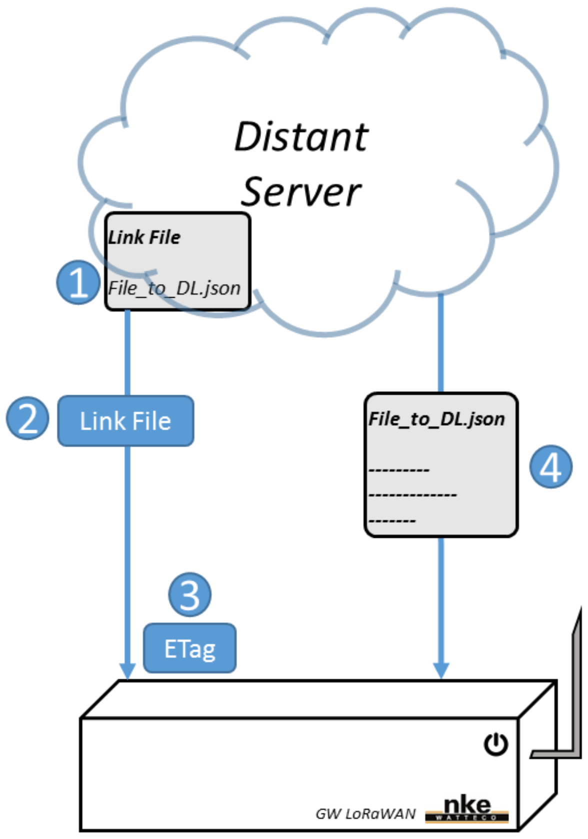

FIGURE 2 - HUB'O GETTING A FILE FROM THE SERVER ILLUSTRATION

Here above can be seen the full process of getting a file from the server for Hub’O.

1. The Server adds the filename of the file to send to Hub’O inside the Link file
2. Hub’O downloads the Link File
3. Hub’O gets the ETag of the Link file and notices that it has changed since the last time
4. Hub’O downloads the file(s) which have their names in the Link file.

4 How To Receive a file from Hub'O

The Hub'O gateway sends regularly files to the distant server. It is either data files, association request files, end-devices alarm files, gateway alarm files or topology files.

Almost all of the quoted files are sent without any period when an event appears: when an alarm is triggered (end-devices alarm or gateway alarm), when an end-device requests an association or after that an end-device has been associated. The data files are the only ones that are send periodically by the gateway and not in a spontaneous way. The period of data files sending is configurable through the Hub'O configuration file.

The illustration here below illustrates how these files are sent by Hub'O.

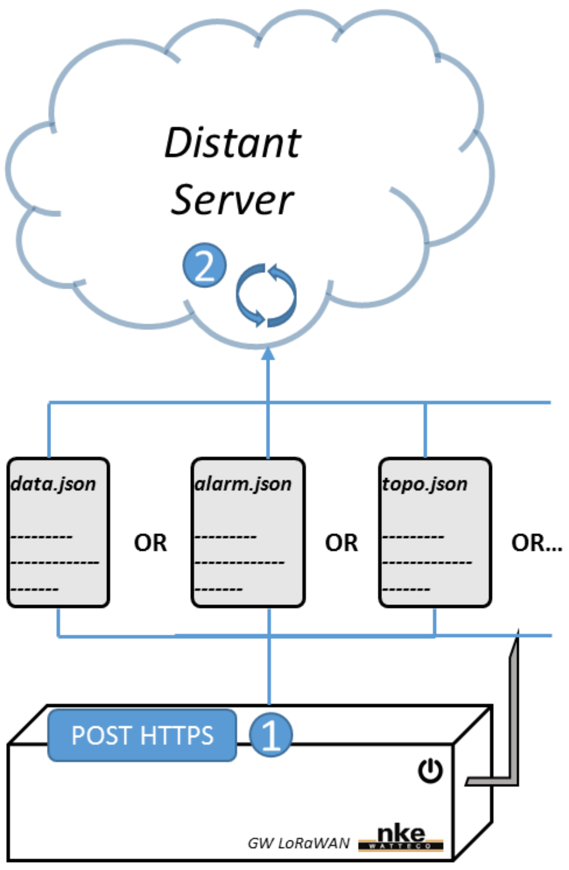

FIGURE 3 - HUB'O SENDING A FILE TO THE SERVER ILLUSTRATION

1. Hub'O does a POST command on the HTTPS link to send the file
2. The distant server process a bit the POST request in order to get and save the file

On the distant server side, a little bit of processing is necessary to save the different files or answer to Hub'O (if the file updated is an asso_request). More details will be given, in the next paragraphs, about each file and the behavior that the distant server needs to have for each of them.

Here below can be seen an example of php code that can be found on the server side, in the Data/ directory. This code allows to save the files sent by Hub'O and to answer OK for each association request.

Thanks to this php code, the files sent by Hub'O are stored in a subdirectory named "Logs" that needs to be created inside the directory "Data".

```php
<?php
// Set the destination directory
$target_dir = "./Logs/";

// Get the date with the wanted format
$date = date("Ymdhis");

// Form the new file name of the file that will be saved
$underscore = "_";
$extension = ".json";
$filename = urldecode($_GET["Filename"]);

// Get the file content from the POST
$json_data = file_get_contents("php://input");

// Check if the file is an association request
if( stristr($filename, "asso_request_") == FALSE )
{
        if( stristr($filename, "d_") != FALSE )
        {
                $head = "data_";
                $devEUI = substr($filename, 2, 16);
        }
        else if( stristr($filename, "a_") != FALSE )
        {
                $head = "alarm_";
                $devEUI = substr($filename, 2, 16);
        }
        else if( stristr($filename, "topology_lora") != FALSE )
        {
                $head = "paired_devices";
                $devEUI = "";
        }

        $target_file = $target_dir . $head . $devEUI . $underscore . $date . $extension;

        // Save this content in the new created file
        file_put_contents($target_file, $json_data);
}
else
{
        // The file is an association request
        // Get the end-device devEUI
        $devEUI = substr($filename, 13, 16); // Get rid of "asso_request_" and ".json"

        // OK file
        $fileToAns =
"{\n\"LoRa_GW_Asso_Answer_File\":{\n\"End_Device_ID\":{\n\"DevEUI\":\"$devEUI\",\n},\n\"Ass
o_status\":\"OK\"\n}\n}";

        // KO file
        //$fileToAns =
"{\n\"LoRa_GW_Asso_Answer_File\":{\n\"End_Device_ID\":{\n\"DevEUI\":\"$devEUI\",\n},\n\"Ass
o_status\":\"KO\"\n}\n}";

        // WAIT file
        //$fileToAns =
"{\n\"LoRa_GW_Asso_Answer_File\":{\n\"End_Device_ID\":{\n\"DevEUI\":\"$devEUI\",\n},\n\"Ass
o_status\":\"WAIT\"\n}\n}";
```

```
        header('Content-Type: application/json');

        // Send the file as an answer to the POST
        echo $fileToAns;

        $head = "asso_request_";
        $target_file = $target_dir . $head . $devEUI . $underscore . $date . $extension;

       // Save this content in the new created file
        file_put_contents($target_file, $json_data);
}
?>
```

5 FILES DETAILED DESCRIPTIONS, UTILITIES AND EXAMPLES

In this paragraph will be detailed all the files used during the Hub'O – Distant Server exchanges. All these files are JSON files.

For each file, an example of file is given.

5.1 LINK FILE

Filename in the exchanges: ***corresp_file.json*** *(name configurable by the Hub'O configuration file)*

As it has been said in §3, the link file function is to allow the distant server to send files to Hub'O.

Indeed, when the distant server needs to send a file to Hub'O, it adds the name of this file in the Link File. As the Link file has been changed, its ETag is changed.

Afterwards, at the polling period (configurable by the Hub'O Configuration file: see §5.2), Hub'O checks the Link file's ETag. If this ETag has changed since the last time, Hub'O downloads the Link file. If the ETag has not changed then nothing happen.

Then, if the Link file has been downloaded, Hub'O gets all the filenames present in the Link file and downloads all the corresponding files.

To have a clear illustration of this process, please see the Figure 2.

File content example:

```
{
        "LoRa_GW_Link_File": {
                "File_Names_To_Download": [
                        { "File_Name": "c_010_0008.json" },
                        { "File_Name": "c_0009_70B3D5E75F0000AB.json" },
                        { "File_Name": "c_0009_70B3D5E75E00AABB.json" },
                        { "File_Name": "p_010_0020.json" }
                ]
        }
}
```

5.2 HUB'O CONFIGURATION FILE

<u>Filename in the exchanges</u>: **c_010_VVVV.json**

With: - VVVV: configuration file version

Name example: **c_010_0008.json**

This file is sent by the distant server to configure a lot of parameters that allow to control the Hub'O's behavior. This file can also be uploaded thanks to an USB stick (cf. ***Hub'O_Getting_Started_X_X.pdf*** document for more details about uploading with USB).

These parameters are sorted in different categories:

- "**Version**": Contains the file's format version. It has to be less than, or equal to, the Firmware version running on Hub'O. If not, the configuration file is not taken into account by Hub'O
- "**Lan**": Contains all the parameters needed for the Local Area Network connection
- "**Wwan**": Contains the parameters necessary if Hub'O uses the GSM/GPRS/3G connection
- "**Time**": Contains the parameters necessary to get correctly the time through SNTP
- "**Service**": Contains the parameters necessary for the application good working
- "**Debug**" : Contains the parameters that may be used for debug

Here below can be found all the parameters detailed category by category.

- "**Lan**":
  - "**IPFixe**":
    - Value : ***true*** or ***false***
    - Allows to parameter Hub'O IP as static (value=true) or dynamic (value=false)
  - "**IPAddr**":
    - Value example : "192.168.4.192"
    - Hub'O IP address value in the case of a static IP. If the DHCP is used ("IPFixe"=false), this parameter is not taken into account
  - "**IPMask**":
    - Value example: "255.255.255.0"
    - Netmask used for the LAN. If the DHCP is used ("IPFixe"=false), this parameter is not taken into account
  - "**IPGw**":
    - Value example: "192.168.4.1"
    - IP address on the LAN of the gateway that allows the connection to Internet. If the DHCP is used ("IPFixe"=false), this parameter is not taken into account
- "**Wwan**":
  - "**PIN**":

    - Value example: "1234"
    - PIN code of the the SIM card on the GSM/GPRS/3G dongle (if used)

  - "**PUK**":
    - Value example: "12345678"
    - PUK code of the the SIM card on the GSM/GPRS/3G dongle (if used)

  - "**IPType**":
    - Value example: "**IP**", "**IPV6**" or "**IPV4V6**"
    - Kind of IP used for the connection on the GSM/GPRS/3G dongle (if used): IPV4, IPV6 or IPV4 and IPV6.

  - "**APN**":
    - Value example: "orange.fr"
    - Access Point Name to be used by the GSM/GPRS/3G dongle (if used)

  - "**APNUser**":
    - Value example: "user"
    - APN username to be used by the GSM/GPRS/3G dongle (if used)

  - "**APNPwd**":
    - Value example : "password"
    - APN password to be used by the GSM/GPRS/3G dongle (if used)

- "**Time**":
  - "**SNTPServer**":
    - Value example: "time.nist.gov"
    - SNTP server name used to get the time and date used by Hub'O

  - "**TimeZone**":
    - Value example: "Europe/Paris"
    - Time zone where Hub'O is located

- "**Service**":
  - "**DNSServer**":
    - Value example: "91.121.161.184"
    - DNS server IP address used for the DNS Service

  - "**PFSComActivated**": *(only available from firmware version 02.00)*
    - Possible values: **true** or **false**
    - Allow to enable or disable the communications with the distant server. If it is disabled, Hub'O does not ask for association validation anymore, it always accepts the sensors asking for association if they are in the authorized end-devices list. Moreover, it does not send data or alarm files to the http server anymore. And, of course, Hub'O does not poll the link file anymore. This way or working has been implemented in the case of Hub'O can only be used through its ModBus interface. The same behavior is obtained if the "PFSUrl" value (cf. here below) is an empty string (i.e. "").

- "**PFSUrl**":
  - Value example: "https://192.168.4.3"
  - URL used by Hub'O to connect to the distant server, it contains the protocol used and the IP address. The protocol can be http or https here.

- "**PFSPort**":
  - Value example: 443
  - Port used to connect to the distant server with the HTTP(S) protocol

- "**PFSUser**":
  - Value example: "User_1"
  - User login, if there is one, to use to connect on the distant server

- "**PFSPwd**":
  - Value example: "Pwd_1"
  - Password, if there is one, to use to connect on the distant server

- "**PFSDataDirectory**":
  - Value example: "Data"
  - Directory name, on the distant server side, where Hub'O sends the following files: alarm files, data files, topology files and association requests files.

- "**PFSConfigDirectory**":
  - Value example: "Config"
  - Directory name, on the distant server side, where Hub'O gets the link file and the different configuration files needed (when they appear in the link file) such as end-devices configuration file or the Hub'O configuration file itself.

- "**PFSLinkFileName**":
  - Value example: "corresp_file.json"
  - Name of the link file that Hub'O will used for downloading it from the distant server.

- "**PFSProvisionningFileName**":
  - Value example: "p"
  - Name beginning of the file containing all the allowed end-devices on the site. Hub'O add "_010_VVVV.json" to have the complete filename. This file contains all the information necessary to communicate with these end-devices through LoRaWAN protocol (see §5.3)

- "**PFSDataFilePeriodMinutes**":
  - Value example: 60
  - Interval of time, in minutes, between two data uploads for Hub'O. For each upload, Hub'O checks if there is new data to send. If it is the case, it uploads them on the distant server, if not it does nothing.

- "**PFSLinkFilePeriodMinutes**":
  - Value example: 60
  - Interval of time, in minutes, between two HTTPS polls on the distant server by Hub'O. This poll consists of getting the link file's ETag, checking if it has

changed and downloading it if this is the case. For the complete polling process, please see §5.1 and §3

- "**EndDeviceSilenceTimeOutHours**":
  - Value example: 25
  - After this interval of time (in hours) without receiving any frame from one of the provisioned sensor, Hub'O sends an "End-device silence alarme" file to the distant server, containing the devEUI of the silent sensor.
- "**FTPSUrl**":
  - Value example: "ftp://192.168.1.27"
  - URL used by Hub'O to connect to the FTPS server that hosts the available new firmware versions. Hub'O will download its new firmware version from this server when the name of a new firmware is put inside the link file.
- "**FTPSPort**":
  - Value example: 21
  - Port used to connect to the FTPS server used for firmware update
- "**FTPSUser**":
  - Value example: "lora-gw"
  - User login, if there is one, to use to connect on the FTPS server
- "**FTPSPwd**":
  - Value example: "password"
  - Password, if there is one, to use to connect on the FTPS server

➢ "**Debug**":
- "**SSHServer**":
  - Posible values: **true** or **false**
  - SSH sessions allowed (=true) or not (=false) on Hub'O
- "**SSHPort**":
  - Value example: 8322
  - Port to use to connect on Hub'O with SSH when it is allowed
- "**LogLevel**":
  - Value example: 2
  - Log level wanted while the Hub'O application is running
- "**Watchdog**":
  - Value example: true or false
  - Variable used to enable or not the Watchdog on Hub'O
- "**LogUpload**":
  - Value example: true or false
  - When set at true, it asks Hub'O to upload several logs file. If the configuration file has been downloaded from the distant server, the log files are uploaded on the FTPS server, if the configuration file comes from an USB stick, the log files are uploaded on the USB stick.

The procedure to send Hub'O's configuration file is the same as the one described in §3.

File content example:

```
{
        "LoRa_GW_Configuration_File": {
                "Version": "01.00",
                "Lan": {
                        "IPFixe": true,
                        "IPAddr": "192.168.4.192",
                        "IPMask": "255.255.255.0",
                        "IPGw": "192.168.4.1"
                },
                "Wwan": {
                        "PIN": "",
                        "PUK": "",
                        "IPType": "IP",
                        "APN": "",
                        "APNUser": "",
                        "APNPwd": ""
                },
                "Time": {
                        "SNTPServer": "time.nist.gov",
                        "TimeZone": "Europe/Paris"
                },
                "Service": {
                        "DNSServer": "91.121.161.184",
                        "PFSComActivated": true,
                        "PFSUrl": "https://192.168.4.3",
                        "PFSPort": 443,
                        "PFSUser": "",
                        "PFSPwd": "",
                        "PFSDataDirectory": "Data",
                        "PFSConfigDirectory": "Config",
                        "PFSLinkFileName": "corresp_file.json",
                        "PFSProvisionningFileName": "p",
                        "PFSDataFilePeriodMinutes": 60,
                        "PFSLinkFilePeriodMinutes": 60,
                        "EndDeviceSilenceTimeOutHours": 25,
                        "FTPSUrl": "ftp://192.168.1.27",
                        "FTPSPort": 21,
                        "FTPSUser": "lora-gw",
                        "FTPSPwd": "password"
                },
                "Debug": {
                        "SSHServer": true,
                        "SSHPort": 8322,
                        "LogLevel": 2,
                        "WatchDog": true,
                        "LogUpload": false
                }
        }
}
```

Inside the configuration file name (**c_010_VVVV.json**), a version VVVV can be seen. Hub'O checks this version before downloading the file. It allows to optimize the volume of data downloaded from the distant server.

If the version in the filename is the same as the last one, Hub'O does not download the file. Thus, the value VVVV in the file name has to be incremented each time that a new configuration needs to be uploaded on Hub'O.

5.3 MODBUS INTERFACE CONFIGURATION FILE

Filename in the exchanges: **c_modbus_010_VVVV.json**

With: - VVVV: modBus configuration file version

Name example: **c_modbus_010_0012.json**

This file can be sent by the distant server to configure a lot of parameters about the ModBus interface. This file can be uploaded as well thanks to an USB stick (cf. ***Hub'O_Getting_Started_X_X.pdf*** document for more details about uploading with USB).

The available parameters are sorted in different categories:

- “**Version**”: Contains the file’s format version. It has to be less than, or equal to, the Firmware version running on Hub’O. If not, the configuration file is not taken into account by Hub’O. The ModBus configuration file is taken into account only by the firmware if it has a version superior to 02.00.
- “**ModBus_ON**”:
  - Possible values: **true** or **false** *(by default)*
  - This field allows to enable or disable the Hub’O ModBus interface
- “**Interface**”:
  - Possible values: “**TCP**” *(by default)* or “**RS485**”
  - This field allows the user to choose between the TCP ModBus interface or the RS485 bus
- “**Slave_Address**”:
  - Possible value: **100** *(by default)*
  - ModBus address used by Hub’O (and is used as offset to the end-devices ModBus addresses)
- “**TCP_Params**”: Contains the parameters used when the TCP ModBus is enable
- “**RS485_Params**” : Contains the parameters used when the RS485 ModBus is enable

Here below can be found all the parameters detailed category by category.

- « **TCP_Params** » :
  - « **ModBus_Port** » :
    - Possible value : **502** *(by default)*
    - Port used in the TCP protocol to send and receive ModBus frames
- « **RS485_Params** » :
  - « **Baud_rate** » :
    - Possible values : **115200** *(by default)*, 57600, 38400, 19200, 9600, 4800, 2400 or 1200
    - Used baud rate on the **RS485** bus

- « **Data_bits** » :
  - Possible values : **8** *(by default)* or 7
  - Parameter « Data bits » used during the communication on the **RS485** bus

- « **Parity** » :
  - Possible values : « **None** » *(by default)*, « Odd » or « Even »
  - Parameter « Parity » used during the communication on the **RS485** bus

- « **Stop_bits** » :
  - Possible values : **1** *(by default)* or 2
  - Parameter « Stop bits » used during the communication on the **RS485** bus

File content example:

```
{
        "LoRa_GW_ModBus_Configuration_File": {
                "Version": "02.00",
                "ModBus_ON": false,
                "Interface": "TCP",
                "Slave_Address": 100,
                "TCP_Params": {
                        "ModBus_Port": 502
                },
                "RS485_Params": {
                        "Baud_rate": 115200,
                        "Data_bits": 8,
                        "Parity": "None",
                        "Stop_bits": 1
                }
        }
}
```

Inside the configuration file name (**c_modbus_010_VVVV.json**), a version VVVV can be seen. Hub'O checks this version before downloading the file. It allows to optimize the volume of data downloaded from the distant server.

If the version in the filename is the same as the last one, Hub'O does not download the file. Thus, the value VVVV in the file name has to be incremented each time that a new configuration needs to be uploaded on Hub'O.

5.4 Allowed End-devices file

Filename in the exchanges: ***p_010_VVVV.json*** *(configurable by the Hub'O configuration file)*

With: - VVVV: allowed end-devices file version

Name example: **p_010_0011.json**

This file contains all the end-devices allowed on the site, where one or several Hub'O gateway(s) are located. For each end-device present in the file, all the parameters necessary to associate it to Hub'O are detailed. Hub'O supports both ABP and OTAA association.

This file needs to be sent to Hub'O in order to be able to associate end-devices with the latter. For more details about the association process please see §5.5 and §5.6.

A field "Version" can be found at the beginning of the file. This field contains the file's format version. It has to be less than, or equal to, the Firmware version running on Hub'O. If not, the file is not taken into account by Hub'O

Here below can be found all the parameters needed for each end-device, detailed category by category.

- "**End_Device_ID**": contains the parameters needed to identify the end-device
  - "**DevEUI**":
    - Value example: "70B3D5E75F0000D8"
    - DevEUI of the current end-device allowed on the site
  - "**DevAddr**":
    - Value example: "010000D8"
    - DevAddr of the current end-device allowed on the site
- "**Asso_Infos**": contains the first part of the association parameters needed
  - "**Activation_Mode**":
    - Possible values: "**OTA**" or "**ABP**"
    - Activation mode to use with this end-device : OTAA or ABP
  - "**Class**":
    - Possible values: "**A**" or "**C**"
    - LoRaWAN class of the end-device : A (sleeping) or C (awake)
- "**OTA_Fields**": contains the fields needed in the case of OTAA association ("Activation_Mode"="OTA")
  - "**AppEUI**":
    - Value example: "70B3D5E75F600000"
    - AppEUI of the current end-device allowed on the site
  - "**AppKey**":
    - Value example: "5B7E151628AED2A6ABF7158809CF4F3C"
    - AppKey of the current end-device allowed on the site

- "**ABP_Fields**": contains the fields needed in the case of ABP association ("Activation_Mode"="ABP")
    - "**NwkSKey**":
        - Value example: "2B7E151628AED2A6ABF7158809CF4F3C"
        - NwkSKey of the current end-device allowed on the site
    - "**AppSKey**":
        - Value example: "4B7E151628AED2A6ABF7158809CF4F3C"
        - AppSKey of the current end-device allowed on the site

Each end-device object can only have either the "**OTA_Fields**" or the "**ABP_Fields**" parameters. It depends if the association type used is OTA ("**Activation_Mode**"="OTA") or ABP ("**Activation_Mode**"="ABP").

The allowed end-devices file can be sent following the process described in §3.

File content example:

```
{
	"LoRa_GW_Allowed_End_Dev_File": {
		"Version": "01.00",
		"End_Device_Objects": [
			{
				"End_Device_ID": {
					"DevEUI": "70B3D5E75F0000D8",
					"DevAddr": "010000D8"
				},
				"Asso_Infos": {
					"Activation_Mode": "OTA",
					"Class": "A"
				},
				"OTA_Fields": {
					"AppEUI": "70B3D5E75F600000",
					"AppKey": "4B7E151628AED2A6ABF7158809CF4F3C"
				}
			},
			{
				"End_Device_ID": {
					"DevEUI": "70B3D5E75F0000D9",
					"DevAddr": "010000D8"
				},
				"Asso_Infos": {
					"Activation_Mode": "ABP",
					"Class": "A"
				},
				"ABP_Fields": {
					"NwkSKey": "2B7E151628AED2A6ABF7158809CF4F3C",
					"AppSKey": "4B7E151628AED2A6ABF7158809CF4F3C"
				}
			},
                    {
                          "End_Device_ID": {
                                "DevEUI": "70B3D5E75F0000DA",
                                "DevAddr": "010000DA"
                          },
                          "Asso_Infos": {
                                "Activation_Mode": "OTA",
                                "Class": "C"
                          },
                          "OTA_Fields": {
                                "AppEUI": "70B3D5E75F600000",
                                "AppKey": "5B7E151628AED2A6ABF7158809CF4F3C"
                          }
                    }
              ]
        }
}
```

Inside the allowed end-devices filename (**p_010_VVVV.json**), a version VVVV can be seen. Hub'O checks this version before downloading the file. It allows to optimize the volume of data downloaded from the distant server.

If the version in the filename is the same as the last one, Hub'O does not download the file. Thus, the value VVVV in the file name has to be incremented each time that a new end-devices list needs to be uploaded on Hub'O.

From Hub'O firmware version 02.00, this file can be uploaded thanks to an USB stick (cf. ***Hub'O_Getting_Started_X_X.pdf*** document for more details about uploading with USB).

5.5 End-device Configuration file

Filename in the exchanges: **c_VVVV_XXXXXXXXXXXXXXXX.json**

With: - VVVV: configuration file version
- XXXXXXXXXXXXXXXX: devEUI of the concerned End-device

Name example: **c_0009_70B3D5E75F0000D8.json**

This file is sent by the distant server to send a configuration to a LoRaWAN end-device through Hub'O. A configuration corresponds to a downlink LoRaWAN frame.

This file allows as well to configure an Alarm rule on Hub'O. That means that if a frame is received by Hub'O from the end-device with the given DevEUI and the frame's payload matches the configured rule, then an alarm is sent immediately to the distant server. It allows to have reactivity when it is important. Indeed, for the other data, there are upload regularly with the data file upload period.

The complete sequence of process and exchanges between the distant server and Hub'O can be found here below in the diagram and its description.

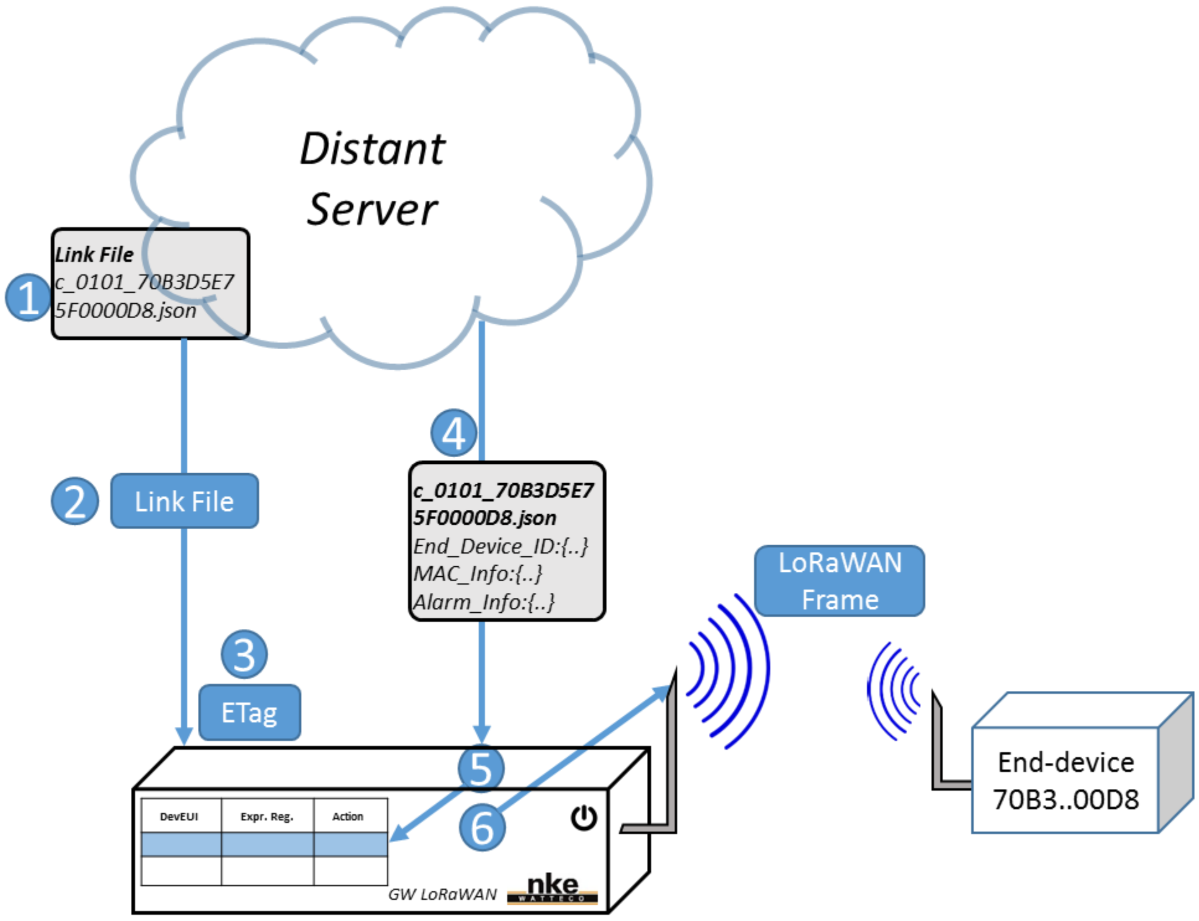

FIGURE 4 - END-DEVICE COMPLETE CONFIGURATION THROUGH HUB'O ILLUSTRATION

In order, can be seen on the diagram:

1. The Server adds the end-device configuration filename to send to Hub'O inside the Link file
2. Hub'O downloads the Link File

3. Hub’O gets the Link file’s ETag and notices that it has changed since the last time
4. Hub’O downloads the end-device configuration file
5. Hub’O saves the Alarm rule if there is one or delete the one existing (see how later in this paragraph)
6. Hub’O sends the configuration payload(s) to the right End-device

After all these steps, if everything went right, the downloaded configuration file is deleted by Hub’O.

Here below can be found all the different fields present inside the json end-device configuration file. They are detailed category by category. Two models of this file can be used. Either the “**old**” one, that allows to send only one downlink frame (firmware version **inferior to 02.00**) or the “**new**” one, that allows to send up to five downlink frames to an end-device with only one configuration file (firmware version **superior or equal to 02.00**). The difference between these two formats is located around the “MAC_Info” location, and it is detailed here below.

- “**Version**”: Contains the file’s format version. It has to be less than, or equal to, the Firmware version running on Hub’O. If not, the file is not taken into account by Hub’O. This is this version that allow to say Hub’O to interpret the file as an **old** (**V01.XX**) or as a **new** (**Version >= 02.00**) format.

- “**End_Device_ID**”: contains the parameters needed to identify the end-device
  - “**DevEUI**”:
    - Value example: “70B3D5E75F0000D8"
    - DevEUI of the end-device for which the configuration is

- Old format : “**MAC_Info**”: contains the fields necessary to send the configuration to the end-device
  - “**FPort**”:
    - Value example: 125
    - Applicative port used in the LoRaWAN to send the configuration

  - “**FrmPayload**”:
    - Value example: “110600500000064180 0F801E050004000000”
    - Applicative payload to send through LoRaWAN in order to configure the end-device. This is the “configuration”

- New format : “**Configuration_Frames**” : a json array with 0 to 5 objects containing “**MAC_Info**” objects corresponding to the “**MAC_Info**” object in the old format

- “**Alarm_Info**”: contains the information relative to an Alarm rule to add or to delete
  - “**Is_Alarm**”:
    - Value example: 0 or 1
    - There are three cases possible with this value.
      - If 1, then a new alarm is configured.
      - If 0, and no regular expression is defined (“Reg_Ex”=””) then no alarm is configured.
      - If 0, and a regular expression is defined, then if an alarm rule exists with this regular expression and for this devEUI, then this alarm rule is deleted.

- "**Reg_Ex**":
  - Value example: "^110A0050000641"
  - Regular expression used to determine if a payload received from the given end-device is an alarm or not. If the payload received matches with the regular expression, then it is considered as an alarm.

- "**Act_On_Alarm**":
  - Value example: 0
  - Type of action to do when an alarm is triggered. For the moment there is just one action possible ("Act_On_Alarm"=0), it uploads directly a corresponding alarm file on the distant server

File content example (**old format**):

```
{
	"LoRa_End_Device_Config_File": {
		"Version": "01.00",
		"End_Device_ID": {
			"DevEUI": "70B3D5E75F0000D8"
		},
		"MAC_Info": {
			"FPort": 125,
			"FrmPayload": "1106005000000641800F801E050004000000"
		},
		"Alarm_Info": {
			"Is_Alarm": 1,
			"Reg_Ex": "^110A0050000641",
			"Act_On_Alarm": 0
		}
	}
}
```

File content example (**new format**):

```
{
	"LoRa_End_Device_Config_File": {
		"Version": "02.00",
		"End_Device_ID": {
			"DevEUI": "70B3D5E75F0000D9"
		},
		"Configuration_Frames": [
			{
				"MAC_Info": {
					"FPort": 125,
					"FrmPayload" : "1106040200000029803C803C0000"
				}
			},
			{
				"MAC_Info": {
					"FPort": 125,
					"FrmPayload" : "110004020000"
				}
			},
			{
				"MAC_Info": {
					"FPort": 125,
					"FrmPayload" : "1105800400000801"
				}
			}
		],
                "Alarm_Info": {
                        "Is_Alarm": 1,
                        "Reg_Ex": "^110A0050000641",
                        "Act_On_Alarm": 0
                }
        }
}
```

Inside the end-device configuration filename (**c_VVVV_XXXXXXXXXXXXXXXX.json**), a version VVVV can be seen. Hub’O checks this version before downloading the file. It allows to optimize the volume of data downloaded from the distant server.

If the version in the filename is the same as the last one, Hub’O does not download the file. Thus, the value VVVV in the file name has to be incremented each time that a new end-devices configuration needs to be uploaded on Hub’O.

5.6 Association Request file

Filename in the exchanges: **asso_request_XXXXXXXXXXXXXXXX.json**

With: - XXXXXXXXXXXXXXXX: devEUI of the End-device trying to associate itself on Hub'O

Name example: **asso_request _70B3D5E75F0000D8.json**

This file is sent by Hub'O to the distant server when an end-device, present in the allowed end-devices file (cf. §5.3), tries to communicate for the first time on the local LoRaWAN network with Hub'O.

The fact that Hub'O sends an association request file to the distant server allows to have several gateways on one site. And it returns to the distant server to choose which Hub'O Gateway will communicate with a given end-device. In order to make the choice possible, the association request file contains information such as RSSI and SNR seen by Hub'O for the given end-device.

In the HTTP(S) answer to Hub'O after the reception of an association request file, the distant server needs to give an association answer file containing either "WAIT", "OK" or "KO". The association answer file will be detailed in the following paragraph.

Nevertheless, the complete process of an end-device association to Hub'O can be found here below in the diagram and its description.

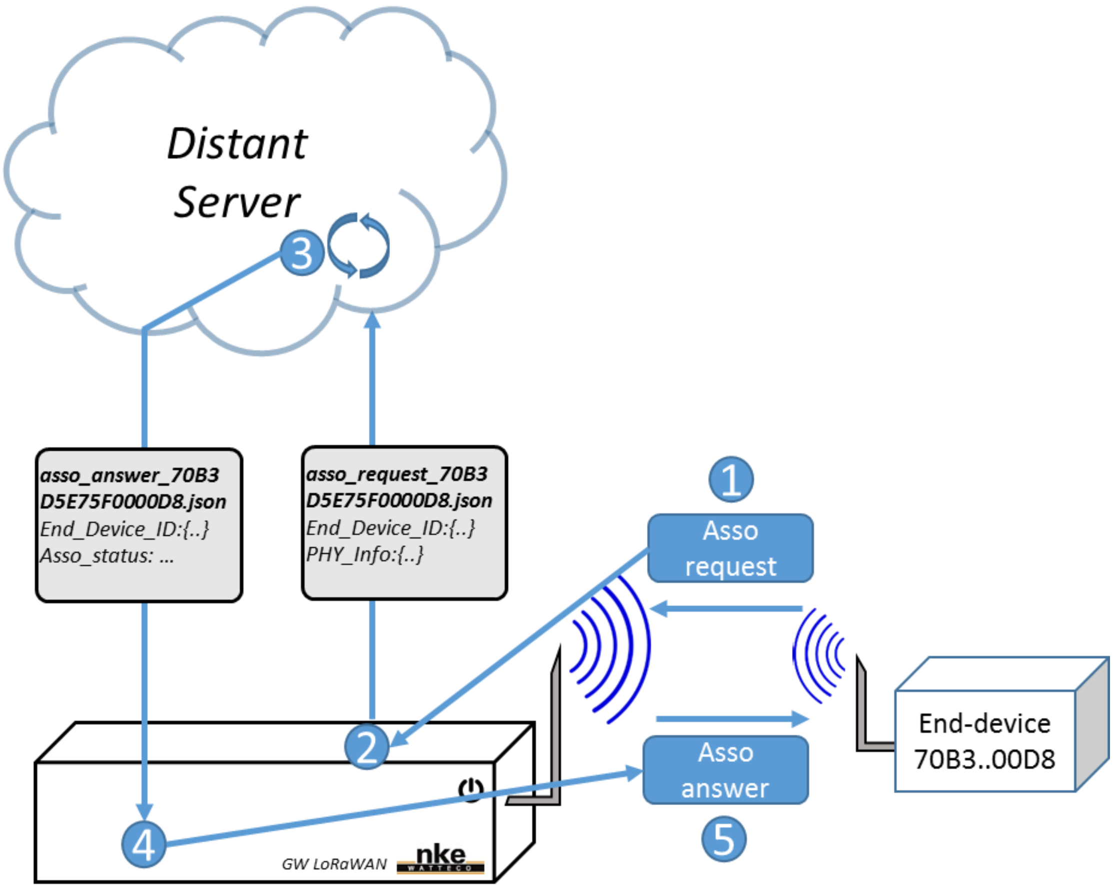

FIGURE 5 - END-DEVICE COMPLETE ASSOCIATION PROCESS ILLUSTRATION

1. The end-device sends its first frame on the LoRaWAN network (a Join Request or a confirmed frame)
2. Hub’O gets the frame, checks if the end-device is in the “Allowed End-devices file” and that it is the first time it sends a frame. If all these configurations are filled, an association request file is sent.
3. The distant server gets the association request file, processes it and gives its answer in the HTTPS response, it could be “OK”, “KO” or “WAIT” (cf. §5.6). Indeed the distant server can ask Hub’O to wait if several Hub’O Gateways are on the site and that the server wants to have all the data before giving its verdict.
4. Hub’O gets the association answer file in the HTTPS response and applies the instruction :
   - If OK, the end-device is saved on Hub’O and Hub’O starts answering to its frames
   - If KO, the end-device will be ignored from now by Hub’O (to work on site with several Hub’O gateways)
   - If WAIT, Hub’O will wait a bit and try again later by sending again an association request file
5. In the illustration, the distant server answers “OK”. Then Hub’O can answer the end-device. Thus, the end-device will considerate itself as “associated”

Here below can be found all the different fields present inside the json association request file. They are detailed category by category.

- “**End_Device_ID**”: contains the parameters needed to identify the end-device
  - “**DevEUI**”:
    - Value example: “70B3D5E75E00AABB"
    - DevEUI of the end-device requesting the association
  - “**DevAddr**”:
    - Value example: “0000AABB”
    - DevAddr of the end-device requesting the association
- “**PHY_Info**”: contains the physical radio data about the end-device transmission seen by Hub’O
  - “**Freq**”:
    - Value example: “868500000”
    - Frequency on which the end-device sent its transmission (in MHz)
  - “**RSSI**”:
    - Value example: -108
    - RSSI seen by Hub’O during the end-device frame reception (in dBm)
  - “**SNR**”
    - Value example: 10
    - SNR seen by Hub’O during the end-device frame reception (in dBm)

File content example:

```
{
        "LoRa_GW_Asso_Request_File": {
                "End_Device_ID": {
                        "DevEUI": "70B3D5E75E00AABB",
                        "DevAddr": "0000AABB"
                },
                "PHY_Info": {
                        "Freq": 868500000,
                        "RSSI": -108,
                        "SNR": 10
                }
        }
}
```

5.7 Association Answer file

Filename in the exchanges: **asso_answer_XXXXXXXXXXXXXXXX.json**

With: - XXXXXXXXXXXXXXXX: devEUI of the end-device for which the distant server gives its answer

Name example: **asso_answer _70B3D5E75F0000D8.json**

This file is sent by the distant server immediately, in the HTTPS response following the HTTPS POST done by Hub'O to send the association request file. For more details about the complete association process, please see §5.5.

The association answer file is a really simple file containing just the device ID of the end-device asking for the association and the distant server answer. As it has been said in the previous paragraph, there are three answers possibles:

- **OK**: The distant server accepts the end-device association with the Hub'O gateway receiving this file. In this case, the Hub'O gateway will communicate with the end-device from now.
- **KO**: The distant server does not accept that the given end-devices associates itself with the Hub'O gateway receiving this file. This answer is given when there are several Hub'O gateways on the site. It means that another gateway is used to communicate with this end-device. In this case the Hub'O gateway will ignore the end-device from now.
- **WAIT**: The distant server still has not taken its decision, and asks the Hub'O Gateway to wait. For example the distant server can wait to see if another Hub'O gateway heard the transmission in better condition than the first one. When this answer is received by Hub'O, it will send again the association request file 30 seconds later.

Here below can be found all the different fields present inside the json association answer file.

- "**End_Device_ID**": contains the parameters needed to identify the end-device
  - "**DevEUI**":
    - Value example: "70B3D5E75E00AABB"
    - DevEUI of the end-device requesting the association
- "**Asso_status**": contains the distant server answer: "OK", "KO" or "WAIT". For further explanations see above.

In the paragraph §4 can be found a php code example than can be used by a really simple distant server to answer an association request. Indeed, it can be seen in the last part that the code prepare the answer to the POST. In this code it can be seen that the requests are systematically accepted.

File content example:

```
{
       "LoRa_GW_Asso_Answer_File": {
              "End_Device_ID": {
                     "DevEUI": "70B3D5E75E00AABB",
              },
              "Asso_status": "OK"
       }
}
```

5.8 HUB'O ALARM FILE

Filename in the exchanges: **a_010_VVVV.json**

With: - VVVV: configuration file version

Name example: **a_010_0005.json**

This file is sent by Hub'O to the distant server in order to notify a problem occurring on the site. The alarm file is sent using the process described in §4.

Especially, the alarm can be used to let the distant server know that:

- The back-up battery's level is low
  - File content:

```
{
        "Alarm_File": {
                "Descriptor": {
                        "Type": "GW_ALARM",
                        "Sub-Type": "LOW_BACKUP_BATTERY"
                },
                "Data": [
                ]
        }
}
```

- A power cut occurred
  - File content:

```
{
        "Alarm_File": {
                "Descriptor": {
                        "Type": "GW_ALARM",
                        "Sub-Type": "POWER_CUT"
                },
                "Data": [
                ]
        }
}
```

As it can be seen here below, for the Hub'O gateway alarm, only the descriptor fields are used. The Data field is not used. This field is used only for the end-devices alarm.

In the descriptor, it can be seen that the type is always, of course, "GW_ALARM" (for Gateway Alarm). Then, the subtype corresponds actually to the kind of problem that occurred on the Hub'O gateway.

Some files are sent as well at the end of the alarm event. Here below can be found these files.

- End of low back-up battery level
    - File content:

```
{
	"Alarm_File": {
		"Descriptor": {
			"Type": "GW_ALARM",
			"Sub-Type": "END_OF_LOW_BACKUP_BATTERY"
		},
		"Data": [
		]
	}
}
```

- End of power cut:
    - File content:

```
{
	"Alarm_File": {
		"Descriptor": {
			"Type": "GW_ALARM",
			"Sub-Type": "END_OF_POWER_CUT"
		},
		"Data": [
		]
	}
}
```

5.9 END-DEVICE ALARM FILE

File name in the exchanges: **a_XXXXXXXXXXXXXXXX.json**

With: - XXXXXXXXXXXXXXXX: devEUI of the end-device from which the frame triggering an alarm has been received

Name example: **a _70B3D5E75F0000D8.json**

This file is sent by Hub'O to the distant server when a frame that has been received by Hub'O from a paired end-device triggered an alarm rule saved on Hub'O. In order to know how to configure an alarm rule on Hub'O, please refer to the paragraph §5.4.

The alarm file is sent using the process described in §4.

The complete process of an end-device alarm trigger and sending can be found here below in the diagram and its description.

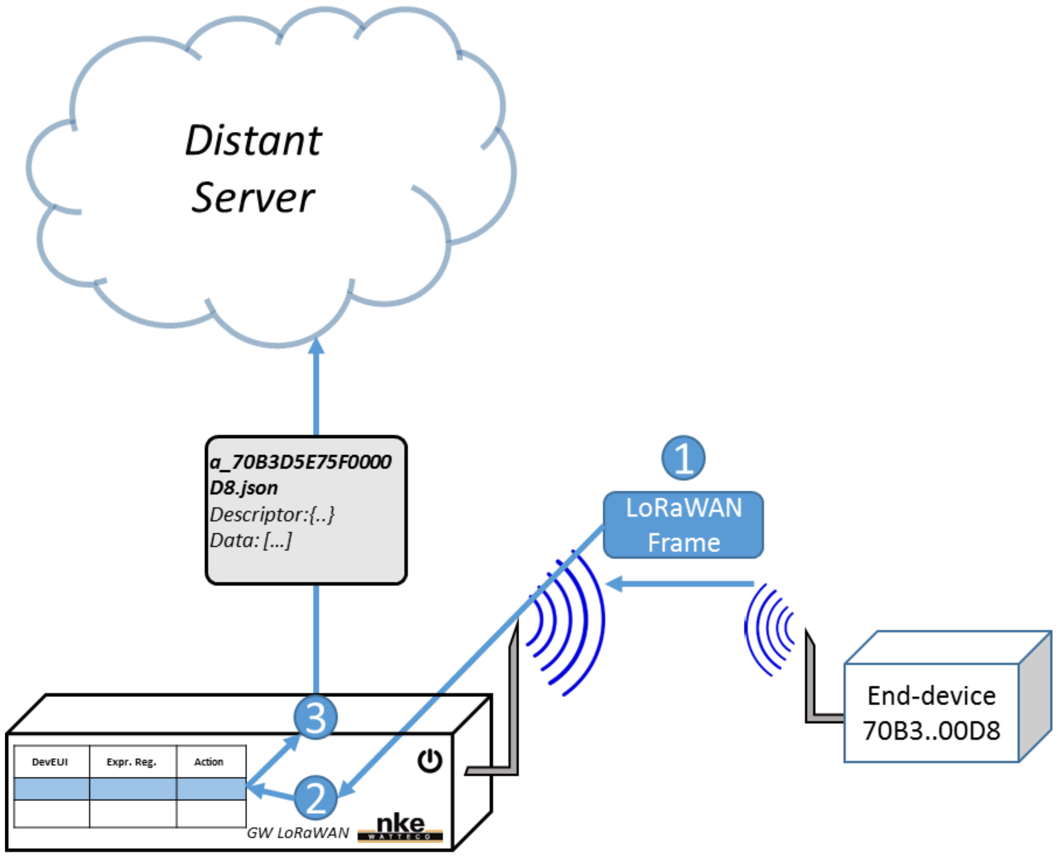

FIGURE 6 - ALARM TRIGGER AND SENDING PROCESS ILLUSTRATION

1. An associated end-device sends a frame to the Hub'O gateway
2. Hub'O checks if the applicative payload of the frame matches an alarm rule configured for this end-device
3. A match has been found, thus Hub'O triggers an alarm file POST to the distant server.

As the Hub’O gateway alarm file, an end-device alarm file contains descriptor fields. Of course they are different for an end-device alarm file:

- “Descriptor”:
  - “Type”: “END_DEVICE_ALARM”
  - “Sub-Type”: “DATA_ALARM”

Moreover, the Data array is used for the end-device alarm files. It contains all the needed information about the frame that triggered the alarm. The different fields present inside a frame object can be seen here below.

NB: *It can be noticed that the applicative payload is decrypted but not decoded. It is an nke Watteco’s choice to not decode the ZCL used by its own end-devices. It allows Hub’O to be interoperable with other end-devices manufacturers. Nevertheless, if the ModBus interface is used by the Hub’O user, it allows to get decoded data for most of the data send by nke Watteco sensors. For more information about the ModBus interface on Hub’O, please see the* ***Hub'O_ModBus_Interface_V1_0.pdf*** *document.*

- “**TimeStamp**”:
    - Value example: “Wed, 12 Jul 2017 12:24:49 +0200”
    - Date and Time, in RFC 822 format, of the frame reception

- “**End_Device_ID**”: contains the parameters identifying the end-device that sends the frame
  - “**DevEUI**”:
    - Value example: “70B3D5E75F0000AB"
    - DevEUI of the end-device that sent the frame

  - “**DevAddr**”:
    - Value example: “010000AB”
    - DevAddr of the end-device that sent the frame

- “**PHY_Info**”: contains the physical radio data about the end-device frame
  - “**Freq**”:
    - Value example: “868500000”
    - Frequency on which the end-device sent its frame (in MHz)

  - “**RSSI**”:
    - Value example: -108
    - RSSI seen by Hub’O during the end-device frame reception (in dBm)

  - “**SNR**”
    - Value example: 10
    - SNR seen by Hub’O during the end-device frame reception (in dBm)

  - “**CryptFrame**”:
    - Value example: “803a2f00008045007d981c892d4301bc7736f9150d09fc671f488262”
    - Complete crypted frame sent by the end-device

- "**MAC_Info**": contains the data about the received frame on the MAC and applicative level
  - "**FPort**":
    - Value example: 125
    - Applicative port used in the LoRaWAN to send the frame
  - "**FCntUp**":
    - Value example: 70
    - LoRaWAN sequence number, or Up counter, of the received frame
  - "**FrmPayload**":
    - Value example: "1106005000000641800F801E050004000000"
    - Decrypted applicative payload received inside the LoRaWAN frame

File content example:

```
{
	"Alarm_File": {
		"Descriptor": {
			"Type": "END_DEVICE_ALARM",
			"Sub-Type": "DATA_ALARM"
		},
		"Data": [
			{
				"TimeStamp": "Wed, 12 Jul 2017 12:24:49 +0200",
				"End_Device_ID": {
					"DevEUI": "70B3D5E75F0000AB",
					"DevAddr": "010000AB"
				},
				"PHY_Info": {
					"Freq": 868300000,
					"RSSI": -100,
					"SNR": 9,
					"CryptFrame":
"803a2f00008045007d981c892d4301bc7736f9150d09fc671f488262"
				},
				"MAC_Info": {
					"FPort": 125,
					"FCntUp": 70,
					"FrmPayload": "110a8007000141072303040000000"
				}
			}
		]
	}
}
```

5.10 Silent End-device Alarm file

File name in the exchanges: **a_silence_XXXXXXXXXXXXXXXX.json**

With: - XXXXXXXXXXXXXXXX: devEUI of the end-device from which the frame triggering an alarm has been received

Name example: **a _silence_70B3D5E75F0000D8.json**

This file is sent by Hub'O to the distant server when a silence is detected on one of the provisioned sensor. Hub'O sends an alarm to warn about a silent sensor when no frames have been received from this sensor since more than 25 hours by default. This threshold is configurable thanks to the field "EndDeviceSilenceTimeOutHours" in the Hub'O configuration file (cf. §5.2).

The alarm file is sent using the process described in §4.

As the Hub'O gateway alarm file, a silent end-device alarm file contains descriptor fields.

- "**Descriptor**":
  - "**Type**": "END_DEVICE_ALARM"
  - "**Sub-Type**": "SILENCE_ALARM"

Moreover, the Data array is used for the silent end-device alarm files. It contains all the DevEUI of the silent sensor. It allows to identify the sensor that seems being silent for too long. These fields are detailed here below.

- "**End_Device_ID**": contains the parameters identifying the end-device that sends the frame
  - "**DevEUI**":
    - Value example: "70B3D5E75F0000AB"
    - DevEUI of the end-device that sent the frame

File content example:

```
{
	"Alarm_File": {
		"Descriptor": {
			"Type": "END_DEVICE_ALARM",
			"Sub-Type": "SILENCE_ALARM"
		},
		"Data": [
			{
				"End_Device_ID": {
					"DevEUI": "70B3D5E75F0000AB"
				}
			}
		]
	}
}
```

5.11 TOPOLOGY FILE

Filename in the exchanges: **topology_lora.json**

This file contains the end-devices ID of all the end-devices paired with Hub'O. This file is sent by Hub'O to the distant server on several occasions:

- At Hub'O start
- Each time that a new end-device is paired with Hub'O (after receiving an OK answer from the distant server)
- Each time a paired end-device is deleted by the distant server (by deleting it from the allowed end-device file and sending it to Hub'O)

To see how this file is sent to the distant server, please refer to §4.

The file is mainly constituted of a json array containing the paired end-devices ID. The different fields present for each paired end-devices can be found here below.

- "**End_Device_ID**": contains the parameters identifying the paired end-device
  - "**DevEUI**":
    - Value example: "70B3D5E75F0000DA"
    - DevEUI of the associated end-device

  - "**DevAddr**":
    - Value example: "010000DA"
    - DevAddr of the associated end-device

File content example:

```
{
        "LoRa_GW_Topology_File": {
                "End_Device_Objects": [
                        {
                                "End_Device_ID": {
                                        "DevEUI": "70B3D5E75F0000D8",
                                        "DevAddr": "010000D8"
                                }
                        },
                        {
                                "End_Device_ID": {
                                        "DevEUI": "70B3D5E75F0000D9",
                                        "DevAddr": "010000D8"
                                }
                        },
                        {
                                "End_Device_ID": {
                                        "DevEUI": "70B3D5E75F0000DA",
                                        "DevAddr": "010000DA"
                                }
                        }
                ]
        }
}
```

5.12 DATA FILE

File name in the exchanges: **d_XXXXXXXXXXXXXXXX.json**

With: - XXXXXXXXXXXXXXXX: devEUI of the end-device from which the data frame has been received

Name example: **d_70B3D5E75F0000D8.json**

A data file contains all the frames, received from a given end-device on a given period of time that are not alarm related (see §5.8). This period corresponds to the upload data file period. To see how this file is sent to the distant server, please refer to §4.

A data file is mainly constituted of a json array containing "Data Objects". These objects correspond to LoRaWAN frames received from a given end-device. The different fields present inside a data object can be seen here below.

NB: *It can be noticed that the applicative payload is decrypted but not decoded. It is an nke Watteco's choice to not decode the ZCL used by its own end-devices. It allows Hub'O to be interoperable with other end-devices manufacturers. Nevertheless, if the ModBus interface is used by the Hub'O user, it allows to get decoded data for most of the data send by nke Watteco sensors. For more information about the ModBus interface on Hub'O, please see the* ***Hub'O_ModBus_Interface_V1_0.pdf*** *document.*

- "**TimeStamp**":
    - Value example: "Wed, 12 Jul 2017 12:20:45 +0200"
    - Date and Time, in RFC 822 format, of the frame reception

- "**End_Device_ID**": contains the parameters identifying the end-device that sends the frame
    - "**DevEUI**":
        - Value example: "70B3D5E75F0000D8"
        - DevEUI of the end-device that sent the frame
    - "**DevAddr**":
        - Value example: "010000D8"
        - DevAddr of the end-device that sent the frame

- "**PHY_Info**": contains the physical radio data about the end-device frame
    - "**Freq**":
        - Value example: "868500000"
        - Frequency on which the end-device sent its frame (in MHz)
    - "**RSSI**":
        - Value example: -108
        - RSSI seen by Hub'O during the end-device frame reception (in dBm)
    - "**SNR**"
        - Value example: 10
        - SNR seen by Hub'O during the end-device frame reception (in dBm)
    - "**CryptFrame**":
        - Value example: "803a2f00008045007d981c892d4301bc7736f9150d09fc671f488262"

        - Complete crypted frame sent by the end-device

- “**MAC_Info**”: contains the data about the received frame on the MAC and applicative level
    - “**FPort**”:
        - Value example: 125
        - Applicative port used in the LoRaWAN to send the frame
    - “**FCntUp**”:
        - Value example: 70
        - LoRaWAN sequence number, or Up counter, of the received frame
    - “**FrmPayload**”:
        - Value example: “1106005000000641800F801E050004000000”
        - Decrypted applicative payload received inside the LoRaWAN frame

<u>File content example:</u>

```
{
        "LoRa_GW_Data_File": {
                "Data_Objects": [
                {
                        "TimeStamp": "Wed, 12 Jul 2017 12:20:14 +0200",
                        "End_Device_ID": {
                                "DevEUI": "70B3D5E75F0000D8",
                                "DevAddr": "010000D8"
                        },
                        "PHY_Info": {
                                "Freq": 868500000,
                                "RSSI": -95,
                                "SNR": 9,
                                "CryptFrame": “80D80000018001007d9cb6d1fb7b4aabc2a2fc98f1"
                        },
                        "MAC_Info": {
                                "FPort": 125,
                                "FCntUp": 1,
                                "FrmPayload": "110a800700014100"
                        }
                },
                {
                        "TimeStamp": "Wed, 12 Jul 2017 12:20:45 +0200",
                        "End_Device_ID": {
                                "DevEUI": "70B3D5E75F0000D8",
                                "DevAddr": "010000D8"
                        },
                        "PHY_Info": {
                                "Freq": 868300000,
                                "RSSI": -108,
                                "SNR": 10,
                                "CryptFrame":
"80D80000018002007dab84d7c0e572ad72ec1d044d13f8880acc16c5"
                        },
                        "MAC_Info": {
                                "FPort": 125,
                                "FCntUp": 2,
                                "FrmPayload": "110a800700014107230304000000000"
                        }
                }
                ]
        }
}
```