Atmel AVR2058: BitCloud OTAU User Guide

Introduction

Designers and providers of embedded wireless systems continue to be challenged by the rapid evolution of the ZigBee® standard. The continuous evolution of the standard requires that these systems must be made “future-proof”, that is, the system engineers must design them to be easily upgraded systems to the next version even after the system has been deployed.

The ability to upgrade networks also depends on the individual devices having enough hardware resources to accommodate the next version of the specification. But even when these hardware resource requirements are met, there still must be a defined and interoperable mechanism for efficient over the air upgrade.

The over-the-air upgrade support in ZigBee networks is covered by the Over-the-Air Upgrade Cluster specification, which complements ZigBee Smart Energy 1.1 specification. The OTAU functionality relies on the use of standard ZigBee data transfer and network management facilities to transfer firmware images to any node on the network. The standard covers on air message exchange, but leaves the details of the architecture and implementation up to the vendor.

This document outlines Atmel’s architecture and implementation of over-the-air upgrade and describes how to add over-the-air upgrade support to embedded applications built on top of BitCloud C API. The guide also introduces PC-side tools for initiating OTAU, explores practical considerations in performing a network upgrade, and provides key performance characteristics of our implementation in a real-world scenario.

8-bit Atmel Microcontrollers

Application Note

Rev. AVR-05/11

1 Architecture

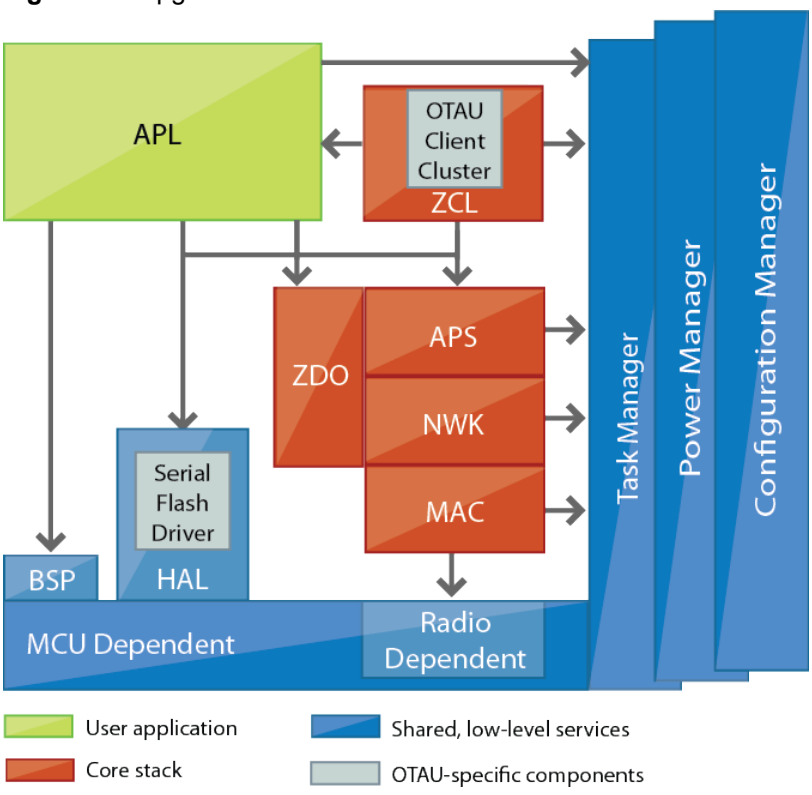
1.1 Architecture Building Blocks

Since ZigBee Over-the-Air Upgrade (OTAU) relies on existing ZigBee PRO services for service discovery and data transmission across the network, its building blocks fit nicely into the standard ZigBee architecture and BitCloud® implementation thereof. The main components of the architecture are:

1. the OTAU client, which resides on one of the end points on every upgradeable device;
2. the OTAU server, which resides on whichever devices initiates the upgrade process;
3. a HAL driver responsible for writing the transferred images to persistent storage on every upgradeable device,
4. an OTAU-capable bootloader for transferring firmware images from persistent storage into microcontroller's flash, also present on every upgradeable device.

The server side of the OTAU cluster resides on what is commonly referred to as the upgrade access point (UAP). This may be a dedicated physical device which implements the server-side cluster or a multi-function in-network device which implements the OTAU service as an add-on piece of functionality. The first case corresponds to an application scenario where a maintenance person temporarily adds a new device to the network for the purpose of upgrading other devices (see Section 1.1.2). The second case corresponds with an application scenario where an in-network device (such as an Energy Service Portal) is used as a permanent access point on the network through which over the air upgrades can be initiated at some point in a network's lifetime .

Figure 1-1 Upgrade Client Architecture



Regardless of how UAP is realized, there is always an implicit backchannel, usually in the form of another network or serial connection outside of the ZigBee network, which is used to transfer the firmware images and control commands to the UAP.

1.1.1 Client side architecture

The high-level software architecture of the OTAU client side (that is, the upgradeable device) is illustrated in Figure 1-1.

According to the OTAU specification [1], the OTAU client part of the architecture is realized as a client side of the OTAU cluster. This implies that the ZigBee Cluster Library (ZCL) framework must be present whenever OTAU is to be enabled on a device, which also restricts the use of OTAU to applications making use of ZCL, specifically ZigBee Smart Energy applications for which OTAU was designed. The standard does not take into account manufacturer-specific profiles and OEM applications.

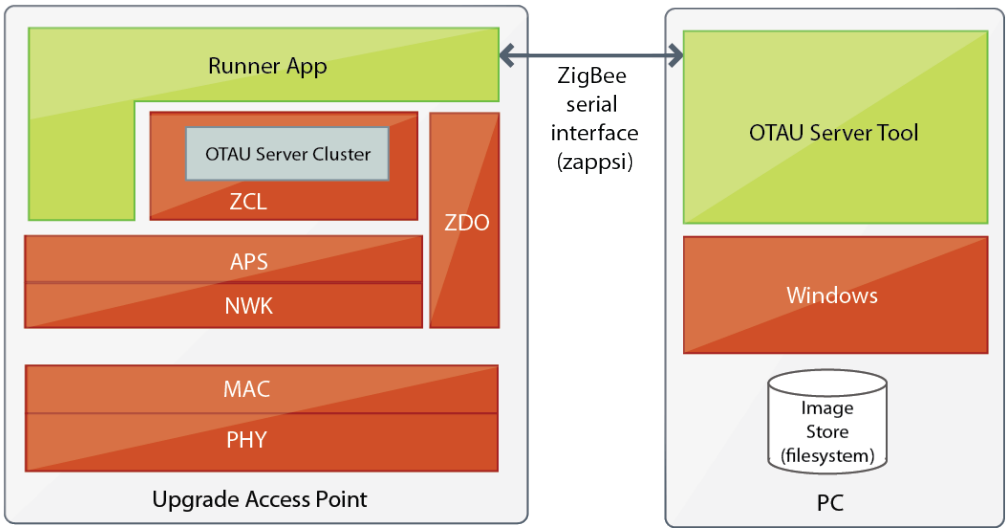
Our architecture does not impose this limitation and enables OTAU for all ZigBee PRO applications, including those that do not use ZCL.

1.1.2 The case of a dedicated UAP

In a reference implementation of this scenario, the upgrade access point (UAP) is a device with the Runner application onboard. The UAP is connected serially to a PC, and the OTAU Bootloader PC application is used as the OTAU Server tool to manage the attached UAP in order to control the entire upgrade process.

The users can also implement a controlling PC tool by themselves, although this is not necessary, because the OTAU Bootloader utility can perform all necessary control operations over the UAP. The Runner application and the OTAU Bootloader tool are explored in Section 2.3. The high-level software architecture of an upgrade server is illustrated in Figure 1-2.

Figure 1-2 Upgrade Server Architecture



1.1.3 The case of an in-network UAP

For this case, an in-network upgrade access point (UAP) is a device joined to the network and supporting the server side of the OTAU cluster in addition to functionality of a common network device. The implementation of the OTAU server cluster support is very much similar to the client cluster support (see Chapter 2).

In order to start and control upgrade process an in-network UAP is connected serially to a PC, and the OTAU Bootloader PC application or a similar custom utility is used.

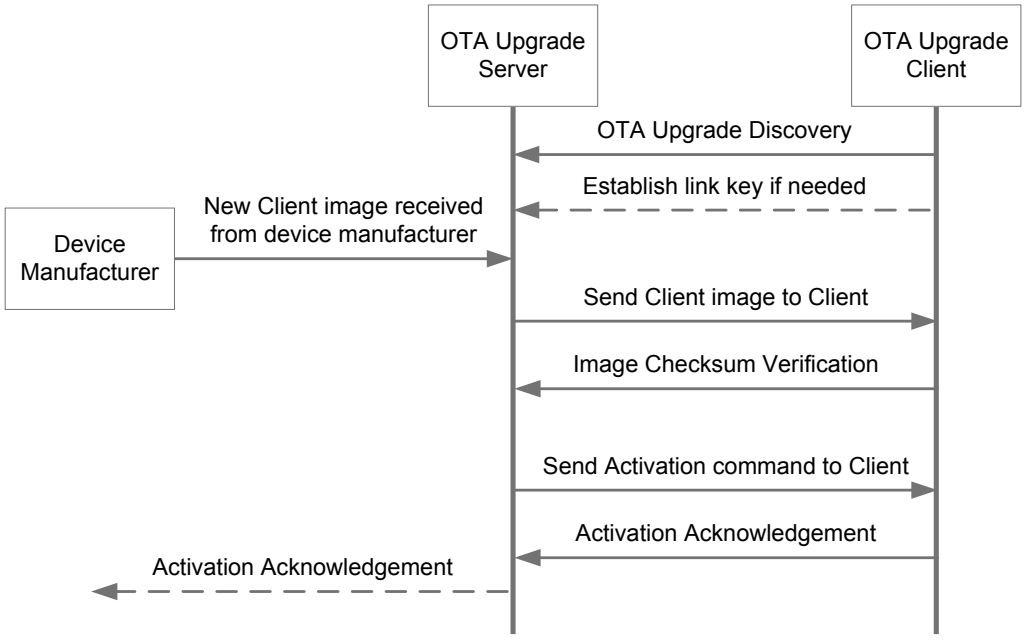
1.2 Basic Protocol and Control Flow

Once UAP appears on the network (assuming it possesses the necessary security material to join it), it must be discovered by the OTAU clients (upgradeable nodes), that is, by those devices that implement the client side of the OTAU cluster. These client nodes issue periodic service discovery commands to discover the OTAU service cluster. In application scenarios where UAP is always present, this discovery will happen once when the client is powered on. In application scenarios where UAP is introduced for a period of time, this discovery will happen continuously during the network's lifetime, until the UAP address is located.

Before clients can proceed with any OTAU specific actions, they must secure the link to the server. The security settings applied to this link are the same as the security requirements for the specific ZigBee application profile of the network being upgraded. Thus for profiles that require high security, the client must negotiate with the trust center for an application link key with the OTAU server. OEM and private profiles can use their own security settings including forgoing security altogether.

Once the link is secured, the clients can begin querying the server for the next image. If the server indicates that a new image is available, then a client starts requesting individual firmware image blocks from the server, eventually completing the download. When the download is complete, the server can tell the client when to actually begin running the new firmware image. The whole sequence of steps is illustrated in Figure 1-3.

Figure 1-3 Network Protocol for Transferring Firmware Images



1.3 Application/OTAU Interaction

An application that wishes to use OTAU functionality must initialize the OTAU client service to run alongside it and to participate in the interactions with the server as shown in Figure 1-3. This is done with a simple call to `ZCL_StartOtauService()` described in Section 2.2.2.

Although the OTAU client service is completely self-contained, that is, no further action needs to be taken by the application during the upgrade process, the application may choose to get OTAU-specific indications from the cluster. These include indications for a new available image, download completion, and various types of error conditions. To receive indications the application simply registers a callback with the OTAU cluster when the OTAU client service is initialized.

Once the application initiates the OTAU client server, both will run in parallel. The application will continue to execute as usual, but it may experience degradation in performance consistent with the amount of traffic generated by the OTAU upgrade. Well-behaved applications should not see any other adverse effects besides proportionally decreased data throughput. The OTAU client cluster runs in the ZCL task handler and at the priority of ZCL layer, so the OTAU tasks will have slightly higher priority than the application tasks.

Note that sleeping devices, which receive a new image available response from the server in the first step, will suspend sleeping until the image is fully received and installed. The application can detect the interrupted sleep by subscribing to the OTAU client cluster indications as described above (see Section 2.4.4).

1.4 Embedded Bootloader and External Image Store

Due to unpredictable and dynamic nature of network links in multi-hop wireless networks like ZigBee, the firmware image can only be transferred to the client using the best effort facilities for data transfer. At any given point in time the client which has downloaded only a part of the image may become disconnected from the network. Since in the general case it is impossible to avoid such a scenario, it helps to ensure that the *entire* image is received before *any* part of it irreversibly overwrites any part of the currently running image.

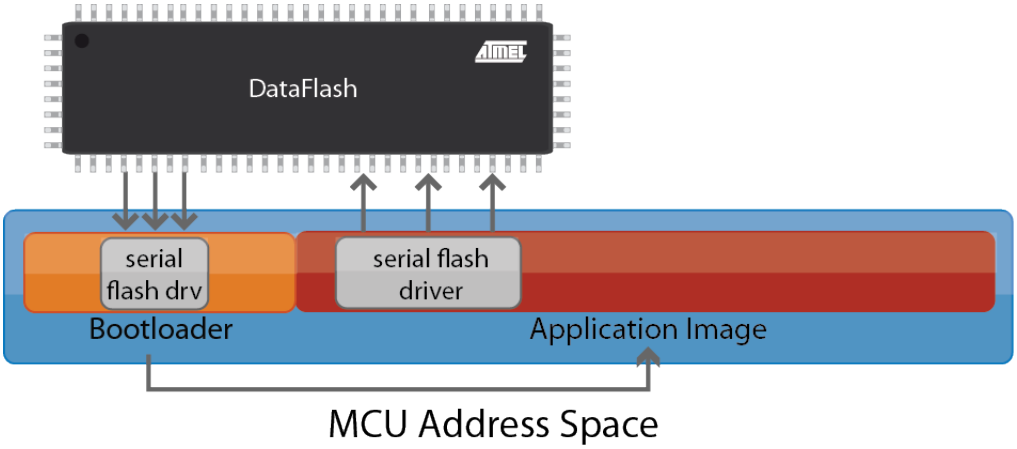
This requires an architecture where whole and partially downloaded firmware images can be stored in a dedicated, non-volatile firmware image store decoupled from the flash memory holding the currently-executing application image. The simplest version of such a system is the one that stores firmware images in a serial flash connected to an upgradable client node. Once the OTAU client cluster receives parts of the image it verifies its integrity (security is already provided by the profile security applied to all in-network communications) and writes it to serial flash. Once the whole image is received, it can be swapped from serial flash into microcontroller's internal flash memory.

Embedded bootloader typically refers to some code that runs on the embedded device after the device is reset, but before the control is passed to the application image. One piece of functionality shared by most bootloaders is the ability to modify parts or the entire application image residing in other parts of flash. Thus bootloaders are commonly used to upgrade embedded devices over a serial connection.

In context of OTAU, the embedded bootloader is extended to also cover interactions with the serial port as well as with the external firmware image store. Thus an OTAU-enabled bootloader may receive images either over serial input or from an external non-volatile memory. Figure 1-4 illustrates this bootloader architecture.

The ability of both the OTAU client cluster and the OTAU bootloader to interact with the external serial flash is critical, so both must be made aware of the external storage interface. Note that since bootloader and application image (including the OTAU client cluster) run in different address spaces, parts of the interface driver must be duplicated. Section 2.2 explores the serial flash driver in more detail.

Figure 1-4 Embedded Bootloader Architecture



2 Implementation

This chapter describes how OTAU is implemented in BitCloud and provides instructions on making devices participate in OTAU as OTAU clients or OTAU servers. Both BitCloud and BitCloud Profile Suite packages provide reference applications which demonstrate OTAU functionality and can be used as templates for adding OTAU support to other reference applications or to the user's own custom applications.

To enable OTAU the user shall consider

- Hardware setup
- Build configuration to include OTAU-enabled libraries
- OTAU parameters configuration
- Use of OTAU cluster API
- Use of PC tools to initiate and control OTAU process

The following sections address all these considerations in more detail.

The OTAU has the classical client-server architecture. Both a client and a server shall support the OTAU cluster, but in different modes: an OTAU client shall register the OTAU cluster as an output cluster, while a server shall support an input OTAU cluster to be able to process incoming requests. The network may contain multiple servers, although typically there is only one OTAU server in the network. A server can be either a common in-network device with OTAU cluster support or a device temporally added to the network, which in the reference implementation is a Runner device.

2.1 Supported Platforms

BitCloud is Atmel's professional-grade implementation of ZigBee PRO standard for wireless monitoring and control [2]. BitCloud Profile Suite is an enhanced version of the BitCloud package, which also adds support for applications targeting public profiles, namely ZigBee Smart Energy, ZigBee Home Automation and ZigBee Building Automation.

OTAU support has been added to BitCloud and BitCloud Profile Suite packages for Atmel's single chip RF transceiver (megaRF) as well as select XMEGA™ and megaAVR™ microcontrollers. The full list of OTAU-enabled software packages is provided in Table 2-1.

Table 2-1. Supported Packages

Package Name	Microcontroller	Radio Frequency Transceiver	Supported External DataFlash
BitCloud Profile Suite SDK for XMEGA	ATxmega256A3	AT86RF231 AT86RF212	AT45DB041 (only 264 byte pages)
BitCloud Profile Suite SDK for MEGARF	ATmega128RFA1	N/A	AT45DB041 (only 264 byte pages)
BitCloud SDK for XMEGA	ATxmega256A3	AT86RF231 AT86RF212	AT45DB041 (only 264 byte pages)
BitCloud SDK for MEGARF	ATmega128RFA1	N/A	AT45DB041 (only 264 byte pages)
BitCloud SDK for ZigBit/ZigBit Amp/ZigBit 900/RCBs	ATmega1281	AT86RF231 AT86RF230 AT86RF212	AT25F2048 AT45DB041

2.2 Hardware Setup

Since the embedded bootloader and the hardware abstraction layer (HAL) of the BitCloud stack are provided in source code, the user may extend the range of supported external flash devices by modifying the driver to interface it with any unsupported chipset. The driver maintains a consistent API for the OTAU cluster to rely upon, which makes it possible to integrate core stack libraries with custom drivers. Of course, the serial flash driver must then also be replicated in the bootloader section.

Note that to switch from one external Flash device to another, the user shall recompile the embedded bootloader and the application itself. In the application the type of external Flash is specified in the `configuration.h` file. For example, if AT45DB041 is used, it will contain with the following line

```
#define EXTERNAL_MEMORY AT45DB041
```

To switch the application to AT25F2048 change `AT45DB041` in the line above to `AT25F2048`, recompile the application, program devices with a new embedded bootloader, and load new application images using the Bootloader PC utility.

2.2.1 Image Storage Driver

The OTAU cluster server side communicates with the image storage, which transfers images to the server, through a special component called the Image Storage driver (ISD). ISD is designed to wrap all hardware related issues providing the OTAU cluster with a public API. In the reference implementation, ISD uses the UART interface to communicate with a PC. The user's custom implementation may involve any other interface with an only requirement to implement all ISD API functions.

Since ISD occupies UART, sample applications code that makes use of UART is disabled with a help of defines if the OTAU cluster server side is going to be supported. Therefore UART cannot be used in sample application for devices serving as OTAU servers. For example, if the WSNMonitor application is used to observe network topology of devices programmed with WSNDemo images, the coordinator

node should not be made an OTAU server, because it transfers data for WSNMonitor through UART.

2.3 Compile-time Configuration

2.3.1 Choosing build configuration

Applications supporting OTAU are compiled with libraries that unlike common reference applications include the OTAU part of the ZCL component. The user shall choose one of appropriate build configurations supporting OTAU. A good example of a sample application provided with the BitCloud SDK that can be configured to work with OTAU is WSNDemo [3].

The way a particular project configuration is selected depends on whether makefiles are used (within the toolchain consisting of AVR Studio and the GCC compiler or command-line compilation with IAR Embedded Workbench) or IAR Embedded Workbench projects.

If makefiles are used, the Makefile located in the root directory of the reference application controls which of the project configurations will be selected at application compile time. Users wishing to enable OTAU simply need to change `PROJECT_NAME` and `CONFIG_NAME` to point to the right configuration. By default, Makefile lists all available project names and configurations. Project names for OTAU-enabled configurations include `OTA`. For example, on the ATmega128RFA1 platform, the project name for OTAU-enabled configurations is `STK600_OTA`. Selecting (uncommenting) `STK600_OTA` at compile time will point the build system at one of the series of Makefiles located in the `\makefiles` directory. In turn, the selected Makefile will define `APP_USE_OTAU`, which will then be defined for the duration of the build process.

If IAR Embedded Workbench is used, open the project file located in the `\iar_projects` directory and select one of configurations including `OTA` in its name. `APP_USE_OTAU` is already defined by such configuration.

When building applications with OTAU enabled, the build system will automatically compile in the serial flash driver and link with the stack library which includes the ZCL component with the client side of the OTAU cluster included. The resulting application image includes all of the components illustrated in Figure 1-1.

2.3.2 Setting OTAU parameters

The `configuration.h` file of the application includes a set of parameters enabled when `APP_USE_OTAU` equals 1 and disabled otherwise. These parameters consist of several parameters controlled by ConfigServer (see Section 2.3.2.1) and additional options not related to ConfigServer such as:

- `APP_USE_FAKE_OFD_DRIVER` which is 0 by default, but when set to 1 substitutes the real Flash driver with the fake Flash driver, which implements all driver operations as empty functions that do not send data anywhere. This feature may be useful to test OTAU operation on evaluation boards.
- `EXTERNAL_MEMORY` which defines what external Flash device should be used
- `OTAU_CLIENT/OTAU_SERVER` which specifies whether the device operates as an OTAU client or an OTAU server. The user shall uncomment the required option and comment out the other.

2.3.2.1 ConfigServer parameters for OTAU

Parameters of the Configuration Server component (ConfigServer) of BitCloud that customize the OTAU operation are presented in Table 2-2. Note that parameters are

applied either on the client or the server side. The role of a device is determined by whether OTAU_CLIENT or OTAU_SERVER in uncommented in the configuration.h file of the application.

Table 2-2. ConfigServer Parameters for OTAU

Parameter	Used on	Description
CS_ZCL_OTAU_DISCOVERED_SERVER_AMOUNT	Client	The maximum number of OTAU servers in the network whose responses the client can process
CS_ZCL_OTAU_CLIENT_SESSION_AMOUNT	Server	The maximum number of clients served by the server simultaneously. If equals 1, the server will upgrade one client at a time.
CS_ZCL_OTAU_SERVER_DISCOVERY_PERIOD	Client	The duration between two attempts to find an OTAU server
CS_ZCL_OTAU_DEFAULT_UPGRADE_SERVER_IEEE_ADDRESS	Client	The default OTAU server address. If the user specifies a valid extended address of a device in the network, the client will send server discovery requests to this particular address. If a broadcast address is specified (0x0000000000000000 or 0xFFFFFFFFFFFFFFFF), the client will broadcast server discovery requests.

To find out more details about ConfigServer parameters refer to [5] or [6].

As it can be observed from Table 2-2, the network can include several OTAU servers, although the user rarely needs more than one server. A client device periodically attempts to discover upgrade servers by sending service discovery requests either to all devices in the network (broadcast) or to a particular address if it is predefined. An upgrade server can also specify the number of simultaneous client sessions, which the server serves in parallel.

2.4 Run-time Interaction with OTAU Cluster for Client/Server

If a correct build configuration is chosen and parameters are specified, the application can enable OTAU functionality through several simple steps described in Section 2.4.1. After the OTAU service has been launched, the application can control the OTAU operation by processing notifications about various events in the callback function specified at OTAU service start and by calling other OTAU cluster API functions. See Section 2.4.3 for detail.

A custom application can safely reuse code from reference applications as described in Section 2.4.2.

2.4.1 Running the OTAU service on a client/server

The OTAU cluster is implemented as a service, such that the user would only need to properly configure the service and start it with a single function call. Since the process is almost identical for both a client and a server, a general procedure is given below highlighting differences between the server and the client side whenever necessary.

To enable OTAU functionality the user shall do the following:

1. Specify whether the device is a client or a server by enabling OTAU_CLIENT or OTAU_SERVER in the configuration.h file of the application.
2. Get the OTAU cluster structure of the ZCL_OtauCluster_t type by calling
    a. ZCL_GetOtauClientCluster() for a client
    b. ZCL_GetOtauServerCluster() for a server

3. Configure and register the endpoint where the OTAU service will reside via the `ZCL_RegisterEndpoint()` function. Note the differences in endpoint configuration for a client and a server:
    a. For a client specify the OTAU cluster in the *out* clusters list and assign the pointer to the structure received in Step 2 to the `clientCluster` field of the endpoint.
    b. For a server specify the OTAU cluster in the *in* clusters list and assign the pointer to the structure received in Step 2 to the `serverCluster` field of the endpoint.
4. After a network start, run the OTAU service by calling

```
ZCL_StartOtauService(&otauInitParams, otauClusterIndication);
```

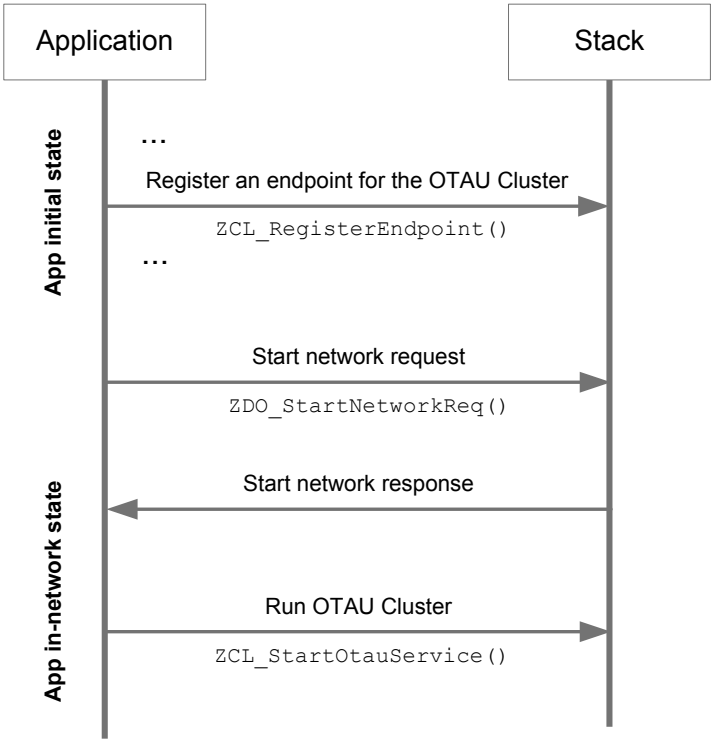
    a. In `otauInitParams` set `clusterSide` to `ZCL_CLIENT_CLUSTER_TYPE` for a client and `ZCL_SERVER_CLUSTER_TYPE` for a server.
    b. `otauClusterIndication` is a callback function executed upon certain events related to the OTAU operation.

The first three steps are performed before the device entered a network. `ZCL_StartOtauService()` is called after the network start. Interaction between the application and the stack is illustrated in Figure 2-1. For source code examples refer to Section 2.4.2.

IMPORTANT

After the OTAU service has been launched, no more actions are required for a server. However, a client must process the `OTAU_DEVICE_SHALL_CHANGE_IMAGE` notification, which indicates that the device is ready to swap images. Refer to Section 2.4.4 for detail.

Figure 2-1 Application/Stack Interaction to Support the OTAU Cluster



2.4.2 Example code for the OTAU service usage

The user can use reference implementation of OTAU support (for example, in WSNDemo) as a template. In the WSNDemo application invocation of the OTAU cluster API is localized in the `WSNZclManager.c` file. Parameters for API functions are configured in the `appZclManagerInit()` function, which is called from the application state machine during application initialization. The OTAU service is started by the `runOtauService()` function called from the application state machine after the network start.

The following code examples illustrate the procedure described in Section 2.4.1.

2.4.2.1 Define variables and constants

Variables holding parameters for API functions are defined in the file scope as follows:

```
static ZCL_Cluster_t otauCluster;
static ClusterId_t otauClusterId = OTAU_CLUSTER_ID;
static ZCL_OtauInitParams_t otauInitParams;
static ZCL_DeviceEndpoint_t otauClusterEndpoint;
```

In addition, define constants for the number of *in* and *out* clusters, which differ for a client and a server:

```
#if defined(OTAU_CLIENT)
#define OUT_CLUSTERS_COUNT 1
#define IN_CLUSTERS_COUNT 0
#elif defined(OTAU_SERVER)
#define OUT_CLUSTERS_COUNT 0
#define IN_CLUSTERS_COUNT 1
#endif
```

2.4.2.2 Get the OTAU cluster

The following code gets the OTAU cluster information for both a client and a server:

```
#if defined(OTAU_CLIENT)
  otauCluster = ZCL_GetOtauClientCluster();
#elif defined(OTAU_SERVER)
  otauCluster = ZCL_GetOtauServerCluster();
#endif
```

2.4.2.3 Register the endpoint

The OTAU service requires an endpoint, which shall be registered by a call to the `ZCL_RegisterEndpoint()` function, rather than via the APS component function used to register endpoints for data transfer. The following code prepares and registers the endpoint:

```
  otauClusterEndpoint.simpleDescriptor.endpoint = APP_OTAU_CLUSTER_ENDPOINT;
  otauClusterEndpoint.simpleDescriptor.AppProfileId = PROFILE_ID_SMART_ENERGY;
  otauClusterEndpoint.simpleDescriptor.AppDeviceId = WSNDEMO_DEVICE_ID;
  otauClusterEndpoint.simpleDescriptor.AppInClustersCount = IN_CLUSTERS_COUNT;
  otauClusterEndpoint.simpleDescriptor.AppOutClustersCount = OUT_CLUSTERS_COUNT;
#if defined(OTAU_CLIENT)
```

```
  otauClusterEndpoint.simpleDescriptor.AppInClustersList = NULL;
  otauClusterEndpoint.simpleDescriptor.AppOutClustersList = &otauClusterId;
  otauClusterEndpoint.serverCluster = NULL;
  otauClusterEndpoint.clientCluster = &otauCluster;
#elif defined(OTAU_SERVER)
  otauClusterEndpoint.simpleDescriptor.AppInClustersList = &otauClusterId;
  otauClusterEndpoint.simpleDescriptor.AppOutClustersList = NULL;
  otauClusterEndpoint.serverCluster = &otauCluster;
  otauClusterEndpoint.clientCluster = NULL;
#endif
  ZCL_RegisterEndpoint(&otauClusterEndpoint);
```

Again, the code above is valid for both a client and a server. A custom application may change the value assigned to `AppDeviceId` to whatever value needed. It is also assumed that the application defines all constants with the `APP` prefix. Here it is the endpoint identifier, which can take any value from 1 to 240 not occupied by other endpoints.

2.4.2.4 Prepare parameters for OTAU initialization

Parameters for the `ZCL_StartOtauService()` function, which is called after a network start to run the OTAU service, may be configured immediately after the endpoint registration.

```
#if defined(OTAU_CLIENT)
  otauInitParams.clusterSide = ZCL_CLIENT_CLUSTER_TYPE;
#elif defined(OTAU_SERVER)
  otauInitParams.clusterSide = ZCL_SERVER_CLUSTER_TYPE;
#endif
  otauInitParams.firmwareVersion.memAlloc =
APP_OTAU_SOFTWARE_VERSION;
  otauInitParams.otauEndpoint = APP_OTAU_CLUSTER_ENDPOINT;
  otauInitParams.profileId = PROFILE_ID_SMART_ENERGY;
```

Note that `otauEndpoint` shall be set to the identifier of the endpoint registered for the OTAU service.

2.4.2.5 Run the OTAU service

As a second argument `ZCL_StartOtauService()` requires a callback function which is called upon OTAU-related events. The simplest callback implementation may look like this:

```
static void otauClusterIndication(ZCL_OtauAction_t action)
{
  if (OTAU_DEVICE_SHALL_CHANGE_IMAGE == action)
  {
    // Device has finished uploading image and can be reset. The
    // application can perform additional actions here before the
    // reset.
    HAL_WarmReset();
  }
}
```

Provided the callback function is declared as stated above, the following line starts the OTAU service:

```
ZCL_StartOtauService(&otauInitParams, otauClusterIndication);
```

2.4.3 OTAU cluster API overview

Table 2-3 lists OTAU cluster API functions. More details, arguments specifications, etc. can be found in [5] and [6].

Table 2-3. OTAU Cluster API Functions

Function	Valid For	Description
`ZCL_GetOtauClientCluster()`	Client	Retrieves the OTAU cluster information on an OTAU client, which should be passed to the endpoint registration function while registering the endpoint for the OTAU service
`ZCL_GetOtauServerCluster()`	Server	Retrieves the OTAU cluster information on an OTAU server, which should be passed to the endpoint registration function while registering the endpoint for the OTAU service
`ZCL_StartOtauService()`	Client and Server	Starts the OTAU service. The function shall be called after a network start.
`ZCL_StopOtauService()`	Server	Stops the OTAU service; is implemented for a server only
`zclIsOtauBusy()`	Client and Server	Checks whether the OTAU cluster is busy or not
`ZCL_UnsolicitedUpgradeEndResp()`	Server	Sends an upgrade end response to a client specifying the duration to wait before swapping firmware images

2.4.3.1 Send an upgrade end response

According to OTAU Cluster Specification when a device loads all parts of a firmware image, it sends to the server an upgrade end request. Upon receiving such request, the server should send an upgrade end response to notify the device of the time after which it should swap firmware images.

In BitCloud implementation of the OTAU cluster all clients receive upgrade end responses that ask to wait infinitely. Following the OTAU Cluster Specification, a client will then send an upgrade end request every hour. To force a device to change its firmware to the uploaded image, the server must send an upgrade end response one more time specifying a finite duration this time. To achieve this the application on the server side shall call the `ZCL_UnsolicitedUpgradeEndResp()`. The duration is calculated as the difference between `upgradeTime` and `currentTime` specified in the function argument. After this duration from receiving the server response elapses, the OTAU client cluster on the device raises an indication that it is ready to change the image. Namely, the callback function specified in `ZCL_StartOtauService()` is called with the `OTAU_DEVICE_SHALL_CHANGE_IMAGE` status.

If the OTAU Bootloader tool is used to launch an upgrade, the user shall send an unsolicited upgrade end response to the upgraded device manually (see Section 2.5.4).

2.4.4 Process OTAU notifications

The OTAU cluster informs the application of various events that occur during its operation via the callback function specified in the `ZCL_StartOtauService()` call (see Section 2.4.2.5). A callback function has the following signature:

```
typedef void (* ZCL_OtauStatInd_t)(ZCL_OtauAction_t action);
```

The `action` variable of the `ZCL_OtauAction_t` enumeration type indicates the type of the event. The enumeration is defined in the `zclOTAUCluster.h` file located in the `BitCloud\Components\ZCL\include` directory.

The application on a server is not obliged to process any events, while a client shall process at least the `OTAU_DEVICE_SHALL_CHANGE_IMAGE` notification, which is called when the time received with the upgrade end response from the server elapses. Thus this notification indicates that the device is ready to swap image, and the application shall respond by invoking hardware reset. Note that the device is not reset automatically after loading the image. This allows the application to make necessary preparations before switching the application image.

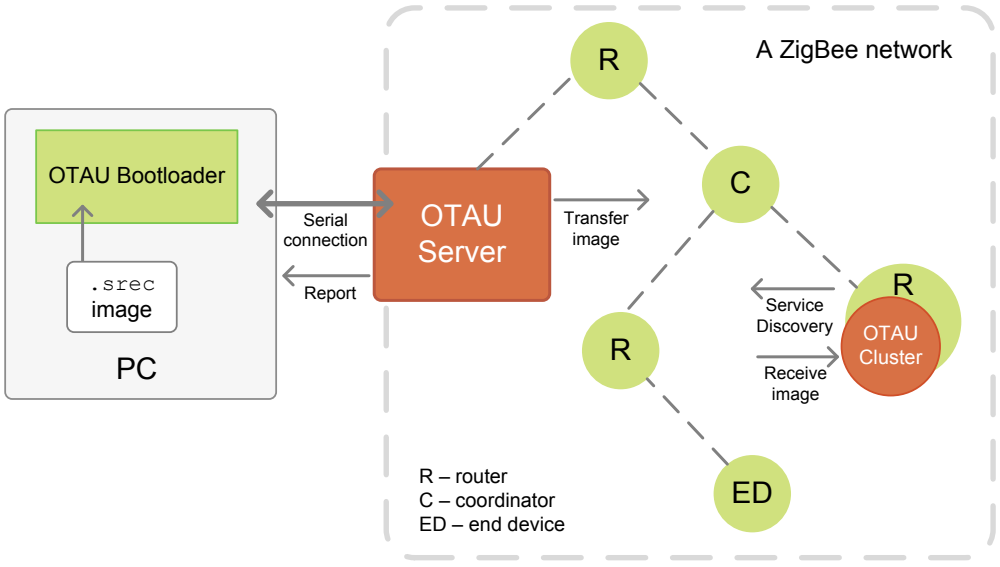
2.5 Upgrade Access Point Tools

Once the network is up and running, the user may proceed with starting the upgrade process with the help of PC tools provided with the BitCloud SDK. In case of a dedicated upgrade access point (UAP), the user shall first configure a Runner device, which will serve as an OTAU server, while in case of an in-network UAP it is assumed that the server device is already present in the network. In both cases the server device shall be connected over a serial link to the OTAU Bootloader tool serving as an OTAU Server tool from Figure 1-1 and installed on a PC.

The OTAU Bootloader tool is used to initiate and control the upgrade process. Another tool involved is the Image Converter utility used to convert `*.srec` images into compatible `*.zigbee` images, which can be uploaded to the devices. The OTAU Bootloader tool and the Image Converter utility are provided with every OTAU-capable BitCloud and BitCloud Profile Suite package.

The overall upgrade process is illustrated in Figure 2-2. The OTAU Server depicted in the illustration may be either a Runner device or an in-network OTAU server. Since a Runner device should first join the network, the procedure of connecting the OTAU Bootloader tool to an OTAU server differs for these two cases. For instance, in case of a Runner device network settings such as the PAN identifier and security keys shall be specified. Proceed to Section 2.5.2 for instructions on using the tool with a Runner device and to Section 2.5.3 to find out how the tool is used with an in-network server (with the so called passive mode enabled). The procedure for updating the network after the server is connected is given in Section 2.5.4.

Figure 2-2 Over-the-Air Upgrade with the OTAU Bootloader Tool



2.5.1 The Runner application

The user may choose to set up the server side of the OTAU cluster on the dedicated UAP device. This is achieved by setting up the standard configuration of the Runner application on a device.

The Runner application can process a wide set of commands received from a PC application through the serial protocol called Zappsi. Thus a device with the Runner application, in addition to serving as an OTAU server, can behave as a common network node and participate in all network communications. The user can write their own application using Python to manage a Runner device, but for the purposes of the over-the-air upgrade the OTAU Bootloader tool is more convenient. Ready-to-use images of the embedded Runner application are provided with the SDK. For details on how to program a device refer to [3] or [4].

2.5.2 Connecting the OTAU Bootloader tool to a Runner device

The following steps illustrate a typical sequence to connect a Runner device with the OTAU Bootloader tool and to prepare an upgrade for the target network:

1. Program a device with the Runner application and connect it to a PC.
2. Start the OTAU Bootloader tool.
3. Switch to the `OTAU` tab as shown in Figure 2-3 below.
4. Specify connection settings to match the port where the Runner device is connected.
5. Click the `Init` button to force the program to collect information about node's configuration such as security settings, network parameters, etc.
6. Click the `Update` button beside the `Custom settings` string under the `Networks` section to collect information about existing networks.
7. Choose appropriate network according to its extended PANID from the list or choose `Custom settings` and insert extended PANID manually.
8. If the target network uses security, click the `Set keys` button to manually set security keys such as the network key and TC link key for the Runner device.
9. Click the `Start` button. The Runner device will try to join the specified network. If the operation fails try again.

Note that you can update several devices simultaneously.

2.5.3 Connecting the OTAU Bootloader tool to an in-network server (passive mode)

The following steps illustrate a typical sequence to connect the OTAU Bootloader tool to an in-network OTAU server and prepare to upgrade the target network:

1. Start the OTAU Bootloader tool.
2. Switch to the `OTAU` tab as shown in Figure 2-3 below.
3. Specify connection settings to match the port where the OTAU server device is connected.
4. Click the `Start passive mode` button. The program will listen to the specified port to detect the OTAU server device.

2.5.4 Updating the network with the OTAU Bootloader tool

The following steps illustrate a typical sequence to upgrade a single device on the network:

1. Connect the tool to a server as described in Section 2.5.2 (if a Runner device is a server) or in Section 2.5.3 (if an in-network device is a server).
2. The utility will automatically populate the list of devices that support the OTAU functionality (that is, applications that include the OTAU client cluster). By default, only devices programmed with application images with OTAU support should be shown in the list as seen in Figure 2-2. The operation may take up to one minute and more depending on end devices' sleep periods.
3. Start the Image Converter utility (you will also be able to convert images in the OTAU Server tool although it is not possible to set metadata information there):
    a. Select `*.srec` image(s) you wish to upload to a remote OTA-capable device over the air.
    b. Fill in image metadata information in fields below and click `Convert`. You can specify the firmware version and the stack version.

NOTE

An `*.srec` image may contain an EEPROM payload. This is configured in the Image Converter utility through the checkbox near the `Erase` label. If it is checked, the image generated by the utility will contain the EEPROM part and the device's EEPROM will be cleared during update with this image. If the `Erase` checkbox is left unchecked, the firmware will not contain the image for EEPROM so that after the update device's EEPROM data will stay unchanged.

4. Return to the OTAU Server tool:
    a. Click on the `Update` button next to device information.
    b. In the window that has opened specify the folder containing firmware images either in the `*.zigbee` or the `*.srec` format. `*.srec` images can be converted in place by clicking on `Convert`.
    c. Select a suitable image and click `Upload` and the upload process will begin.
    d. Once the image is uploaded the progress bar next to the updated device will be replaced by a button. Click on this button to send an update end response to the device, informing it that it can swap application images. Note switching to a new firmware can take additional time.

Upload progress for each device being updated will be shown in the OTAU server tool window depicted on Figure 2-4. When a sleeping end device uploads a new firmware image, it suspends sleeping, but it still uses the polling mechanism to request data from its parent. Therefore for end devices with greater `CS_INDIRECT_POLL_RATE` parameter, which specifies the period of time between two poll requests, it will take longer to download the firmware image.

Figure 2-3 The OTAU Server tool main screen

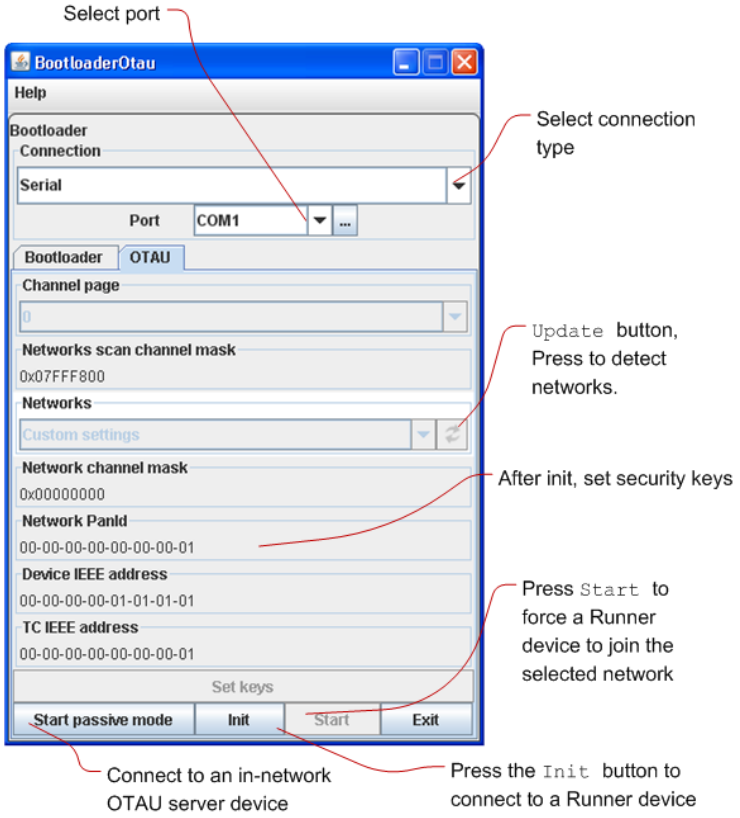
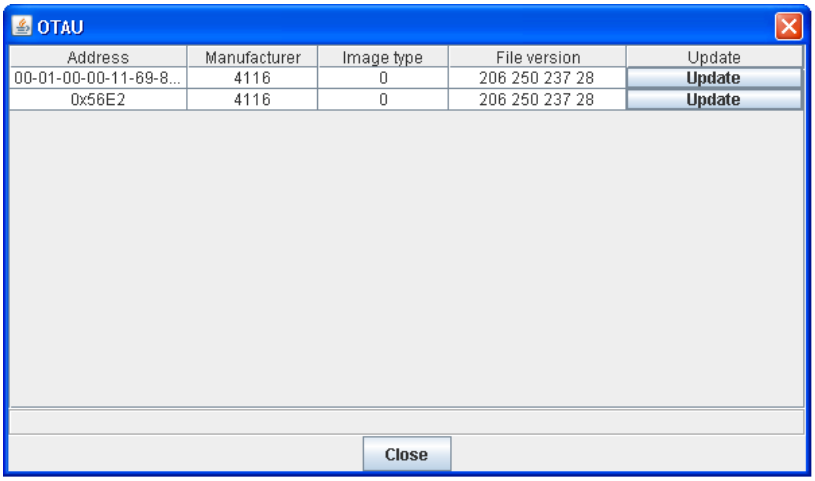


Figure 2-4. The OTAU Server tool devices' screen



3 Performance

One of the key aspects of the architecture's performance is its speed, that is, how fast an individual node and the whole network can be upgraded. The upgrade speed depends on several application-specific parameters, including the type of the device being upgraded, the number of hops separating the device from the UAP, the type of

security scheme applied to the upgrade payload, the data rate and the flash read and write performance.

This chapter describes some preliminary performance results, including the time it takes to upgrade a node under different network conditions. All measurements were made using ATmega128RFA1 and OTAU-capable WSNDemo application. The ATmega128RFA1 device includes 128K of flash on board and 8K of EEPROM. Typical application images sizes range between 80K and 120K. The following subsections present the preliminary performance data.

3.1 Image Transfer over Multiple Hops

This test measured the time it takes to upload the WSNDemo firmware image to a router node one, two, and three hops away from the UAP. The size of the transferred image was 121Kb for standard security and 86.8Kb for no security. These sizes include an EEPROM portion, which is also transferred and programmed into microcontroller's EEPROM alongside the flash. Note that the choice of the transferred image (and its size) depended on the type of security settings applied to the whole network. Flashing a node with a secured image on a non-secured network would mean that the new firmware would be non-functional.

Table 3-1. Multi-hop upgrade of a router node

For Platform	**Hops**			
	1		**2**	
	Upload time (min:sec)	Normalized time per Kb (s)	Upload time (min:sec)	Normalized time per Kb (s)
No security (107Kb)	4:15	2.38	6:30	3.64
Standard security (121.5Kb)	6:40	3.29	10:15	4.50
High security (123.2Kb)	9:00	3.90	12:34	6.08

Note how security settings add a 94% overhead to the time it takes to upload the firmware image over 1 hop, 92% over 2 hops. Some of this slow down can be attributed to the reduction in the maximum packet payload from 94 bytes to 65 bytes (44% decrease) when standard security is enabled. The rest can be attributed to the overhead of CCM* encrypt and decrypt operations applied to every packet's payload when the standard security mode is on. Even though the CCM* stream cipher relies on the underlying hardware implementation of AES-128, the software still imposes some additional overhead.

3.2 Image Transfer to End Devices

The data for the OTAU upgrade over 1 hop is similar to the data obtained for the router case. This is due to the fact that once the OTAU client initiates the transfer of the image, it suspends the sleep cycle. Thus the end device remains on throughout the duration of the upload.

Table 3-2. Multi-hop upgrade of an end device node

For Platform	**Polling Rate**	
	100	**200**

For Platform	Polling Rate			
	100		200	
	Upload time (min:sec)	Normalized time per Kb (s)	Upload time (min:sec)	Normalized time per Kb (s)
No security (107Kb)	14:10	7.94	23:50	13.36

When working with OTAU-enabled applications, system designers should be aware of how the overall battery life of an end device will be affected by the multi-minute on times required for the over-the-air upgrade to complete.

3.3 Bootloader Performance

Once the image is transferred and saved on the external DataFlash device, it must be programmed into microcontroller's internal flash. In our tests, this operation also included an optional backup procedure, that is, the current contents of microcontroller's internal flash were copied into the external DataFlash before the new image was copied in. Thus, the time it took the bootloader to swap the images included an internal flash read and write, as well as external flash read and write.

Swap time is the time it takes the bootloader to transfer the image from external DataFlash into the microcontroller's internal flash memory while backing up the current image to the external DataFlash. Since the swap time could not be measured directly for remote devices, the first on-air packet from the new firmware image was deemed an indicator of how long an image swap took.

In our tests, swap time ranged between 90 and 98 seconds. This corresponds to a period of time when the device is completely unavailable to other network nodes, which is not the case during network upload. For networks where availability is critical, the time can be further reduced by disabling the backup routine for the current image thus reducing the swap time to as little as 50-60 seconds.

3.4 Footprint Overhead

Linking in the ZigBee Cluster library and OTAU flash drivers into the application image has an effect on the application footprint both in terms of flash and RAM. For the WSNDemo application, which typically does not require the ZCL component, the OTAU overhead using GCC toolchain is about 16K of flash and 1400 bytes of RAM. The flash overhead is slightly less (about 12K) for IAR compiler toolchain. For the SE Meter application, which includes the ZCL component by default, the overhead of OTAU is about 6K of flash and 500 bytes of RAM.

This data suggests that about 6K of flash overhead comes from OTAU-specific functionality and another 6K comes from the generic ZCL support.

3.5 Additional Performance Considerations

In general, the time it would take to upgrade the entire network is proportional to the time it takes to upgrade a single node. Assuming that only one UAP is performing all the upgrades, we can easily estimate the time it takes to upgrade the whole network. Considering a network of 100 routers, the entire upgrade could take approximately 7 hours for a non-secure network, 11 hours for a secure network, and 14 hours for a high-security network.

Additional speed-ups can be gained by introducing multiple UAPs. In certain network topologies featuring well-defined network segments, a dedicated UAP may upgrade its own network segment. The mechanisms employed by the OTAU client cluster do not preclude having multiple servers, each node discovering and associating with its own neighborhood UAP. In this application scenario, the upgrades can happen in parallel, and the time it would take to upgrade the network would be inversely proportional to the number of UAPs performing the upgrade.

4 Reference

[1] 095264r12 ZigBee Over-the-Air Upgrading Cluster Specification

[2] 053474r19ZB_CSG-ZigBee Specification

[3] AVR2052: BitCloud Quick Start Guide

[4] AVR2055: BitCloud Profile Suite Quick Start Guide

[5] BitCloud Stack API Reference

[6] BitCloud Profile Suite API Reference

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Milennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chou-ku, Tokyo 104-0033
JAPAN
Tel: (+81) 3523-3551
Fax: (+81) 3523-7581