User Guide

SmartConnect 6LoWPAN

MCU WIRELESS

Introduction

Atmel SmartConnect 6LoWPAN Software Development Kit (SDK) provides a complete solution for IP-mesh connectivity over 802.15.4 links that can be used in various applications.

This document describes.

- SmartConnect 6LoWPAN architecture, package directory structure, understanding of low-power MAC and stack configurations.
- Example application provided in the release, supported platforms and references for help.
- Project files for Atmel Studio and gcc support.

Features

- Portable, low-power 6LoWPAN solution
- Low Power MAC
- RPL, IPHC, IPv6, IPv4 protocol support.
- UDP, TCP, HTTP,and MQTT.
- Small foot-print size (upto 200-300 nodes).
- OTAU support with internal flash.
- Support for ATMEL Cortex-M0 microcontrollers – SAMR21 and SAMD21.
- Support for ATMEL transceivers - AT86RF233 and AT86RF212B.

Table of Contents

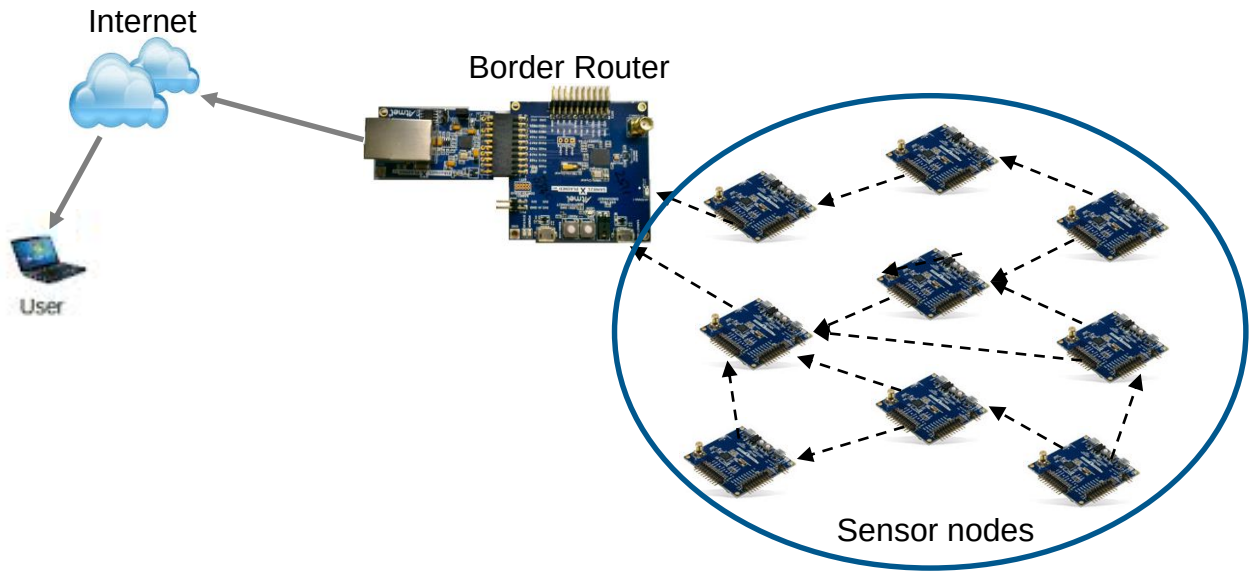
1 Overview

Atmel SmartConnect 6LoWPAN Software Development Kit (SDK) provides a complete solution for IP-mesh connectivity over 802.15.4 links that can be used in various applications.

Typical SmartConnect 6LoWPAN network is composed from end nodes (for example sensor, actuators, range extenders, etc) and a border router (bridge device) as shown on Figure 1-1.

The wireless connection between the edge nodes and border router happens via IEEE 802.15.4 and IP layer brings in the connectivity to the outside world.

Figure 1-1. Typical SmartConnect 6LoWPAN network



1.2 SmartConnect 6LoWPAN SDK

The main items provided as part of Atmel SmartConnect 6LoWPAN are:

- Source code and Project files for Atmel reference applications:
  - UDP applications (see Sections 5.3, 5.4, 5.5)
  - Mesh node application (see Section 5.6)
  - Border router applications (see Sections 5.7, 5.8)
  - MQTT application (see Section 5.9)
  - OTAU applications (see Section 5.10 5.11)
- Source code for 6LoWPAN stack components (see Chapter 2)
- Atmel Smart Upgrade tool and user guide for the same.

1.3 SmartConnect 6LoWPAN Directory Structure

SmartConnect 6LoWPAN is available in *thirdparty/wireless/SmartConnect_6LoWPAN* folder of Atmel Software Framework (ASF) code base [5]. Table 1-1 explains directory structure in the SmartConnect 6LoWPAN module of ASF

Table 1-1. SmartConnect 6LoWPAN Directory Structure

Directory	Description	Notes
./apps	This directory contains the application modules that are used in conjunction with core stack.	
./core/lib	This module contains management modules that are required by stack to use like memory management, list management, etc	
./core/sys	This module contains system modules like timers, process, auto-start module, etc.	
./core/net	Contains the protocol stack	See Section 2.1
./dev	This folder contains transceiver related files	
./examples	This folder contains example projects and related application files	See Chapter 5
./services	This folder contains the modules that are used in conjunction with stack	

1.4 Supported Hardware Platforms and IDEs

The supported hardware platforms are shown in Table 1-2

Table 1-2. Supported Hardware Platforms

Name in this document	Microcontroller	Supported RF Transceivers	Supported Evaluation Kits	Supported IDEs
SAMR21	ATSAMR21G18A	Built-in 2.4GHz	ATSAMR21-XPRO	Atmel Studio v6.2
SAMD21	ATSAMD21J18A	AT86RF233 AT86RF212B	ATSAMD21-XPRO REB233-XPRO ATZB-RF-233-1-C ATZB-RF-212B-0-CN	Atmel Studio v6.2

1.5 General prerequisites

The following tools and tool-chains are used for building the applications from this MAC package:

- Atmel Studio 6.2 [6]
- ARM Code Sorcery GCC Toolchain for Windows (IA32 Windows Installer) [11]

1.6 Building applications

1.6.1 Using GCC makefiles for standalone-projects

INFO **Makefiles will be used for building ASF projects downloaded as stand-alone rather than through Atmel Studio. Atmel Software Framework documentation [5] has details on how to download stand-alone projects.**

Each application should be built using the provided Makefiles. Please follow the procedure as described:

- Change to the directory where the Makefile for the desired platform of the corresponding application is located, for example:

```
cd D:\ASF\thirdparty\wireless\SmartConnect_6LoWPAN\examples\hello-world
cd samr21_xpro\gcc
```

- Run the desired Makefile, for example:

```
make –f Makefile
```

Makefile builds a binary optimized for code size without Serial I/O support, whereas Makefile_Debug builds a version for better debug support without optimization but with additional Serial I/O support

- After running one of the Makefiles the same directory contains both a hex-file and an elf-file which can be downloaded onto the hardware.
- The above procedure for building the Makefiles is common for all platforms.`

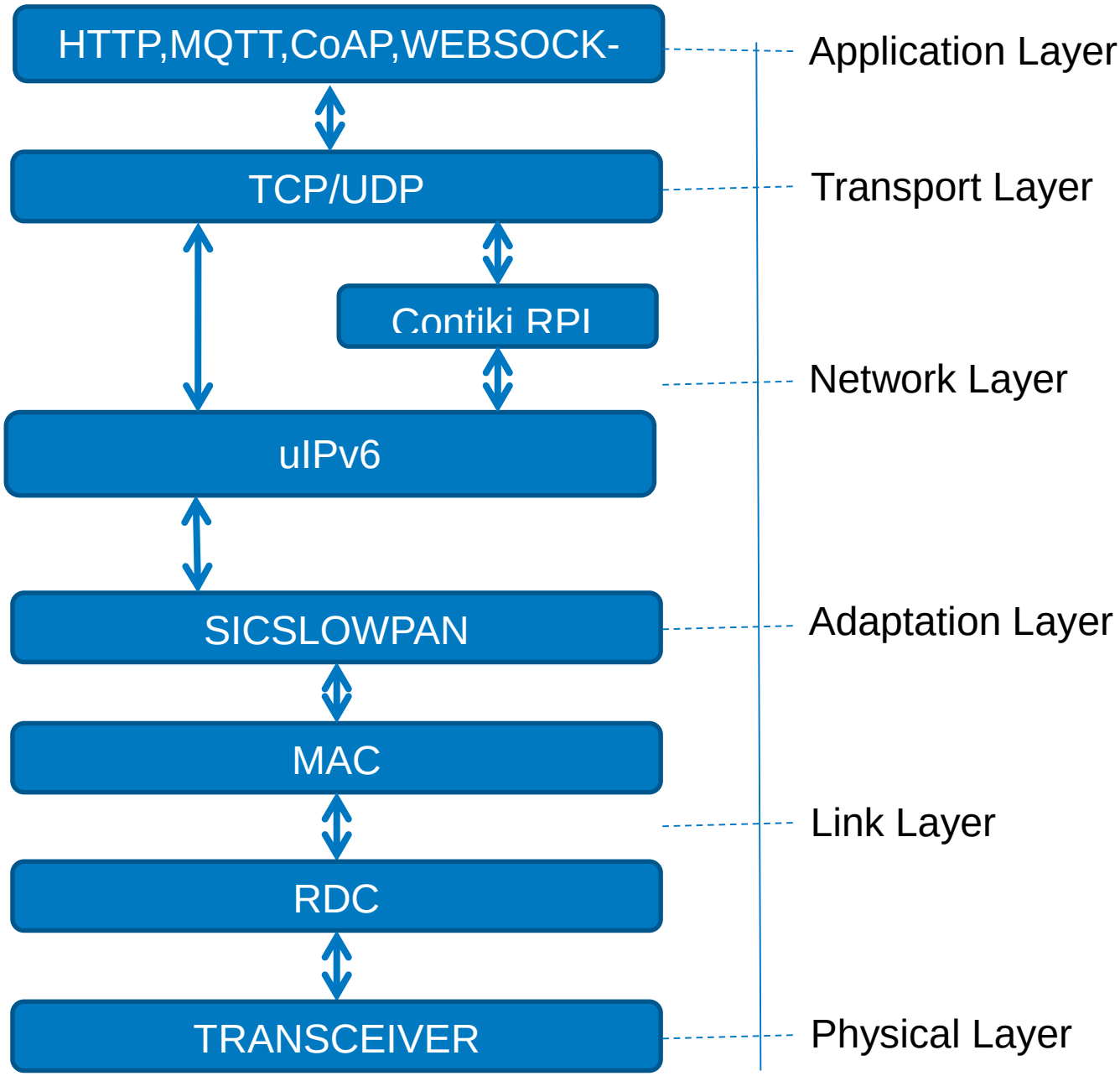
For details and description about platform porting please refer to Atmel Software Framework documentation [5] and Atmel Studio-help [6].

2 General Architecture

SmartConnect 6LoWPAN follows modular approach and has well-defined communication model between the layers.

General architecture of SmartConnect 6LoWPAN is given in Figure 2-1 and layers themselves are described in details in Section 2.2.

Figure 2-1. SmartConnect 6LoWPAN architecture



2.2 Stack Layers

Table 2-1 lists protocols supported in SmartConnect 6LoWPAN layers. All of them are configurable in nature. So based on the need in hand, any module can be added, removed or reconfigured.

Table 2-1. SmartConnect 6LoWPAN layers

Layer	Supported Protocols
Application (Section 2.1.7)	HTTP, MQTT
Transport (Section 2.16)	UDP, TCP
Network (Section 2.15)	IPv6 / RPL
Adaptation (Section 2.14)	Sicslowpan
MAC (Medium Access) (Section 2.13)	CSMA, NULLMAC
RDC (RADIO DUTY CYCLE) (Section 2.1.2)	CONTIKIMAC, NULLRDC
Physical (Section 2.1.1)	IEEE 802.15.4

2.2.2 Transceiver layer

This layer contains the transceiver specific functionality used for IEEE 802.15.4 packets transmission and reception.

Atmel 2.4GHz (AT86RF233) and sub-GHz (AT86RF212B) transceivers are supported.

The following components are implemented inside the Transceiver Layer:

1. Frame transmission unit
2. Frame reception unit
3. Interrupt handling
4. Initialization and reset
5. Power management

2.2.3 RDC Layer

SmartConnect 6LoWPAN stack supports two RDC layers:

- NULL RDC layer that always keeps the radio in RX_ON . No duty cycling is performed by this layer.
- CONTIKIMAC RDC layer is a low power radio duty cycling layer which keeps the radio most of the time OFF to achieve minimum power consumption, meanwhile synchronized with the nodes in the network in way to receive packets. This is achieved by the use of real timer module.

2.2.4 MAC Layer

SmartConnect 6LoWPAN support two MAC layers:

- NULL MAC Layer: it takes packets from adaptation layer and transmits via RDC layers. This layer will not re-transmit any packets in case of any MAC failures.
- The CSMA layer: receives and transmits packets using RDC layer. When MAC failure occurs, this layer will re-transmit the packets automatically.

2.2.5 Adaptation Layer

The adaptation layer uses the SICSLOWPAN protocol and handles the conversion of IP packets to IEEE 802.15.4 packets and vice versa.

The functions of the layers are,

- Receive IP/RPL packets from network layer.
- Fragmentation of IP packets whose size is greater than IEEE 802.15.4 MAC frames MAX size (128).
- IPv6 header compression/decompression.
- Re-assembly of received IPv6 fragmented packets.
- Provide interface for MAC layer to send the received MAC frame to IP layer.

2.2.6 Network layer

Network layer of SmartConnect 6LoWPAN uses uIP implementation for an IP protocol and relies on RPL as routing protocol for Lossy and Low Power WPAN (RPL).

RPL has a tree-like topology with one or more roots and it trickles downward to the leaf nodes.

Contiki RPL is a lightweight and power efficient version of standard IETF RPL.

Contiki RPL is implemented alongside the uIP layer and RPL layer uses ICMPv6 for sending RPL packets.

2.2.6.1 RPL

The features of RPL layer are,

- RPL DAG functional handling.
- RPL Instance management.
- RPL parent discovery and table management.
- Objective function management.
- RPL Rank calculation handling.
- RPL timer management.
- RPL ICMPv6 message handling.

INFO **Refer to draft specifications released by the ROLL working group for detailed study of RPL – *RFC6550***

2.2.6.2 uIP layer

uIP Layer handles network layer functionalities. The functions of this layer are,

- IPv6 header encoding and decoding.
- Retrieval of routes from RPL layer.
- ICMPv6 message handling.
- Interface to transport layer – send/receive data from transport layer.
- IP data forward management.
- Neighbor discovery and table management.
- Packet Queue and Buffer management.

2.2.7 Transport layer

SmartConnect 6LoWPAN transport layer has both TCP and UDP support. And the main functions are,

- TCP socket management and interface of TCP socket APIs for application layer.
- UDP socket management and interface of UDP socket APIs for application layer.

- Simple UDP – This component of transport layer abstracts the user from handling complex UDP APIs and provides the application layer with simple UDP APIs for ease of use.

2.2.8 Other Network modules

There are some components in SmartConnect 6LoWPAN, which provides application layer some networking functionalities and they are listed below.

- MDNS – Provides support for Domain Name Space features, to discover IP address of Destination from web address.
- DHCPv4 – Provides support to configure, the application with a dynamic IPv4 address. This module is used especially on a Border Router node (see Section 5.7 for reference application).
- IP64 – This component provides support to convert IPv6 address to IPv4 address. This module is used on a border router (see Section 5.7).
- IP64 webserver – This component provides support to dump HTML page from node. This module is used in border router webserver application (see Section 5.8).
- Simple-rpl – This module provides support user, a simplified version of RPL APIs for ease of use.
- MQTT – This module provides support to user the register, subscribe and publish data over MQTT protocol to the user. This module is used in mqtt-example application (see Section 5.9).
- HTTP-socket – This module provides support for HTTP GET and HTTP PUT APIs to the user.

2.3 Other stack components

Other stack components include HAL services such as timer, sleep module,

2.3.1 Timer module

Stack Module uses three types of timers

- etimer (Event Timer):

  Event timers provide a way to generate timed events.

  An event timer will post an event to the process that set the timer when the event timer expires. An event timer is declared as a struct etimer and all access to the event timer is made by a pointer to the declared event timer.

  Etimer uses a base hardware timer which runs in 8MHz.It generates interrupt every 8ms from which the timer value is calculated.

- ctimer (Callback Timer):

  The ctimer module provides a timer mechanism that calls a specified callback function when a ctimer expires.

  Callback timer module also uses the same hardware timer for etimer and this will call the callback function on timer expiry.

- rtimer (Real Timer Module):

  The real-time module handles the scheduling and execution of real-time tasks (with predictable execution times).

  The Real timer module is used for time critical tasks where the event cannot wait for other processes to finish. This uses a base hardware timer running at very lower resolution of 31.25 KHz.(rtimer priority).This Rtimer Should run in highest priority as this is for time critical events and to resume when in power down modes.

Hardware dependent timer module files can be found in the directory services\timer\sam0

Stack dependent timer files can be found in the directory core\sys

2.3.2 Sleep module

For battery powered end nodes there's a low power wakeup mechanism. In the power saving cycle both the transceiver and MCU are put sleep. Sleep module is an add-on for the Low power MAC. It Uses the Internal RTC which is in synchronization with the radio duty cycling as a wakeup source and puts the MCU in a lowest Power Mode (Standby Mode).

In the contiki MAC module It can be configured whether MCU also enters into deep sleep Mode.It can be configured using the Macro CONF_MCU_SLEEP

Sleep Module functions can be found in the directory services\timer\rtimer-arch.c

2.3.3 Resource management

Core stack has a module for Memory block allocation and Management

The memory block allocation routines provide a simple yet powerful set of functions for managing a set of memory blocks of fixed size. A set of memory blocks is statically declared with the MEMB ()http://contiki.sourceforge.net/docs/2.6/a01684.html - gaf31774d02a69fd3f1c2b282454438cba macro. Memory blocks are allocated from the declared memory by the memb_alloc () function, and are deallocated with the memb_free () function. This Module is inside the core\sys.

The managed memory allocator is a fragmentation-free memory manager.

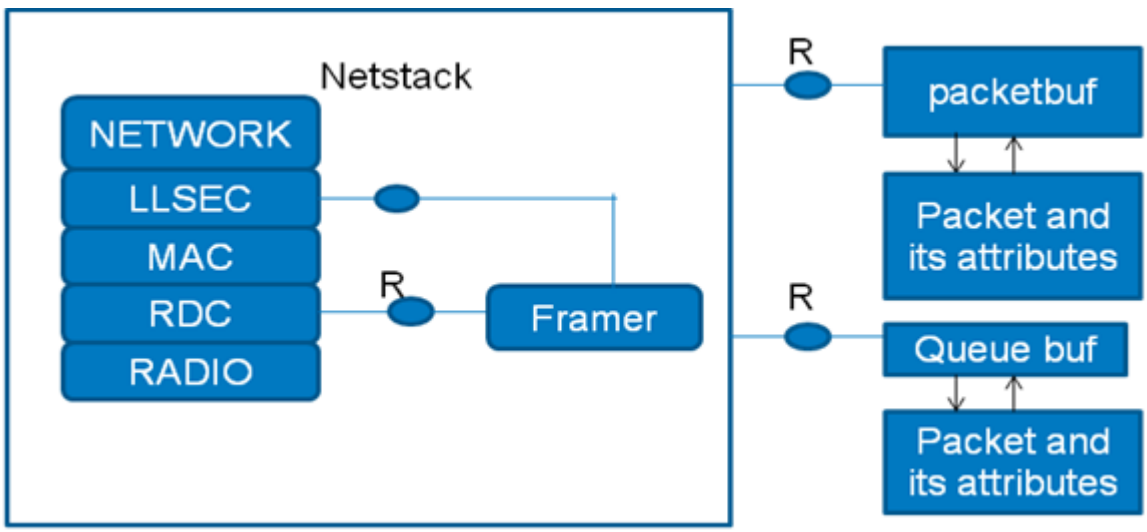
It keeps the allocated memory free from fragmentation by compacting the memory when blocks are freed. A program that uses the managed memory module cannot be sure that allocated memory stays in place. Therefore, a level of indirection is used: access to allocated memory must always be done using a special macro.

Memory module files can be found in the directory core\lib\ memb.c

2.3.4 Link layer security

Link layer security is implemented as a netstack layer, which is located in between the MAC and the NETWORK layer. Figure 2-2 gives a block diagram of link layer security implementation in SmartConnect 6LoWPAN.

Figure 2-2. Link layer security



noncoresec is a noncompromise-resilient 802.15.4 security implementation, which uses a network-wide key. `coresec` is a compromise-resilient LLSEC driver. `coresec` implements the 802.15.4 security sublayer, the Adaptable Pairwise Key Establishment Scheme, as well as the Easy Broadcast Encryption and Authentication Protocol. Currently noncoresec support is available based on software aes.

ENABLE_AES Build switch configures the Link layer security in stack.

LLsec files can be found in the directory core\net\llsec

3 Stack configuration

The stack used by SmartConnect 6LoWPAN can be highly configured according to the application needs. This requires a variety of build switches to be set appropriately at compile time.

3.1 Macro definitions

Table 3-1. Macro defintions for stack configuration

Macro definition	Value	Meaning
NETSTACK_CONF_MAC	nullmac_driver csma_driver	Select NULLMAC or CSMA by configuring either one using this macro
NETSTACK_CONF_FRAMER	framer_802154	This Build switch defines the framer and used here is IEEE 802.15.4 framer according to the IEEE MAC 802 15.4 Specification.
NETSTACK_CONF_NETWORK	sicslowpan_driver	This Build switch defines the 6lowpan layer which format packets between the 802.15.4 and the IPv6 layers. Here it is configured to use the default siclowpan driver

Macro definition	Value	Meaning
NBR_TABLE_CONF_MAX_NEIGHBORS	20	This build switch configures the Neighbor table size for the Maximum number of neighbors. Please note while changing this value, RAM size also needs to be considered.
SICSLOWPAN_CONF_FRAG	1	This build switch is used to configure the SICSLOWPAN layer level fragmentation. Currently by default this macro is enabled.
UIP_CONF_BUFFER_SIZE	400	This build switch configures the UIP buffer size. This MACRO is has to be increased, if the application payload is expected to be greater than the configured size. Default value is 400. Please note while changing this value, RAM size also needs to be considered.
UIP_CONF_MAX_ROUTES	32	This build switch configures the Maximum number route entries that can be stored in the Route table. Please note while changing this value, RAM size also needs to be considered.
DATA_RATE	DATA_RATE BPSK_20 DATA_RATE BPSK_40 DATA_RATE OQPSK_SIN_RC_100 DATA_RATE OQPSK_SIN_250	This Build switch is for sub-GHz band. This is to configure the band/channel page in sub-GHz. This build switch is used when using SUB-GHz transceiver (RF212B).

3.2 MAC ID and IPv6 address assignment

Every node maintains its own 8-byte link-address that matches the mac-id programmed in the device. It forms the LS bytes of the IPv6 address).

In the stack, this is stored in a global variable called, "*linkaddr_node_addr*" defined in "*thirdparty\wireless\SmartConnect_6LoWPAN\core\net\linkaddr.c*" and of the type:

```
typedef union {
  unsigned char u8[LINKADDR_SIZE];
} linkaddr_t;
```

With `LINKADDR_SIZE` set to 8.

3.2.1 SAMR21 Link address assignment

- Link address is read from EDBG_I2C module from a slave address 0x28 after writing the request command in a 2-byte buffer, {0x51, 0xd2}.
- This is done by the function, "***edbg_eui_read_eui64()***" defined in "*thirdparty\wireless\SmartConnect_6LoWPAN\services\sam0\edbg-eui.c*"

3.2.2 SAMD21 Link address assignment

- Link address is read serially from the transceiver module at the address, "EEPROM_EUI64_ADDRESS" set to 0x98.
- This is done by the function, "***at24mac602_arch_read_eui64()***" defined in "*thirdparty\wireless\SmartConnect_6LoWPAN\services\at24mac602\at24mac602-arch.c*"

3.2.3 IPv6 address assignment

- The link address fetched above is used to set the IPv6 address of the node.
- The IPv6 assignment is done using the function *uip_ds6_addr_add()*.
- This address, once assigned, is used by RPL module to send RPL messages and join a network.

INFO **For more information on RPL messages, please refer IETF RPL RFC 6550[7]**

4 MAC power optimization

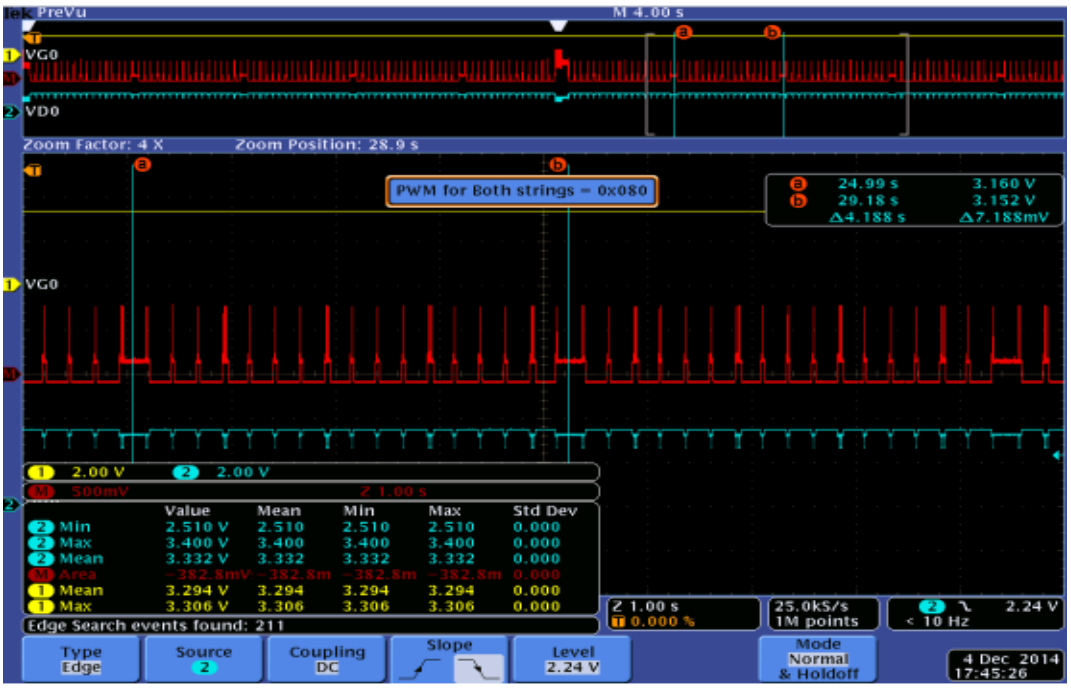
ContikiMAC is a simple low power radio duty cycling layer (RDC). ContikiMAC uses only asynchronous mechanisms, no signaling messages, and no additional packet headers. ContikiMAC is a radio duty cycling protocol that uses periodical wake-ups to listen for packet transmissions from neighbors. If a packet transmission is detected during a wake-up, the receiver is kept on to be able to receive the packet. When the packet is successfully received, the receiver sends a link layer acknowledgment. To transmit a packet, a sender repeatedly sends its packet until it receives a link layer acknowledgment from the receiver. Packets that are sent broadcasts do not result in link-layer acknowledgments. Instead, the sender repeatedly sends the packet during the full wake-up interval to ensure that all neighbors have received it.

Radio duty cycle can be configured according to the application by the Macro in contiki-conf.h.

```
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 4
```

The below scope shot shows the periodic wakeups and CCA cycle of the MCU.

Figure 4-1. Contiki MAC power profiling data



Results from above power measurements with `NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 4` configuration are,

Average Current Consumption: 137uA (for one duty cycle)

Maximum Current Consumption: 9.44mA

Minimum Current Consumption: 3.35uA

INFO **These power measurements are done with no transmission packets in the channel and in lab scenario.**

IMPORTANT **Please note this measurement may vary. These results are shown here are just for the reference.**

5 Example applications

The SmartConnect 6LoWPAN software package includes a variety of example applications which can be flashed on the supported hardware platforms and be executed immediately. On the other hand applications' source code is provided thus helping application developers to understand the proper use the stack and to allowing to make modifications required for custom applications. If the example application makes use of the UART interface (through the EDBG port), the baud rate is set to 115200.

These applications will be explained in more detail in the subsequent sections.

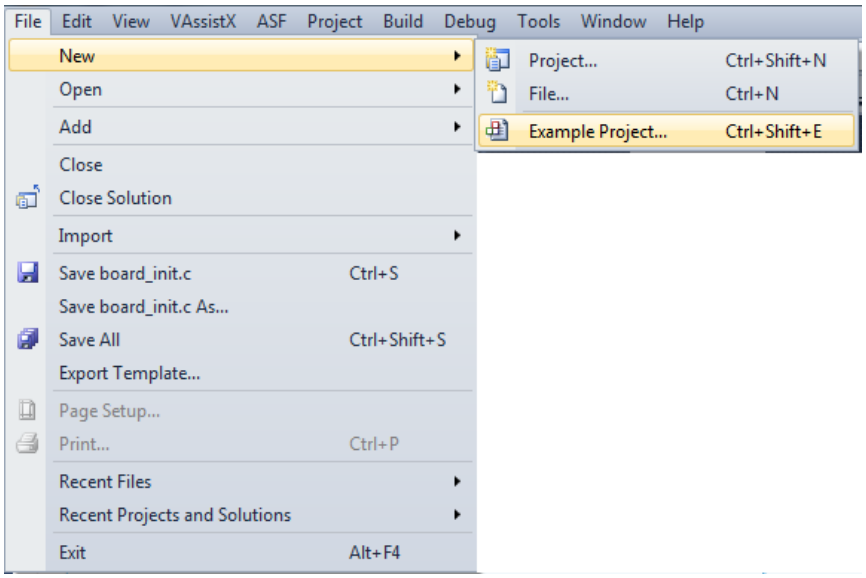
Table 5-1. List of SmartConnnect 6LoWPAN application examples

Example Project	Description	Section
Hello World	Communicate to the MQTT host configured and establish the MQTT connection	5.2
UDP broadcast (Null RDC)	Create a UDP socket and successfully send and receive UDP broadcast data over the network.	5.3
UDP broadcast (ContikiMac RDC)	Same as above but with radio-duty-cycling enabled.	
UDP Unicast Receiver	Establish UDP connection with multiple devices and successfully receive data from network.	5.4
UDP Unicast Sender	Establish UDP connection and send UDP uni-cast data to a device	5.5
Mesh node	Create DAG network, join the existing network and get notification from UIP module	5.6
MQTT	Communicate to the MQTT host configured and establish the MQTT connection	5.9
Border Router	Create a DAG root and act as a Border gateway to the nodes which wants to communicate outside the network.	5.7
Border Router – Webserver	Create a DAG root and act as a Border gateway to the nodes which wants to communicate outside the network. This example uses Webserver module to create HTML pages views of configured information which can be viewed from a browser outside the network	5.8
OTAU – Server	Server-side (Host) app for over-the-air upgrade of stack firmware on top of an external bootloader	5.10
OTAU - Client	Client-side (Target) app for over-the-air upgrade of stack firmware on top of an external bootloader	5.11

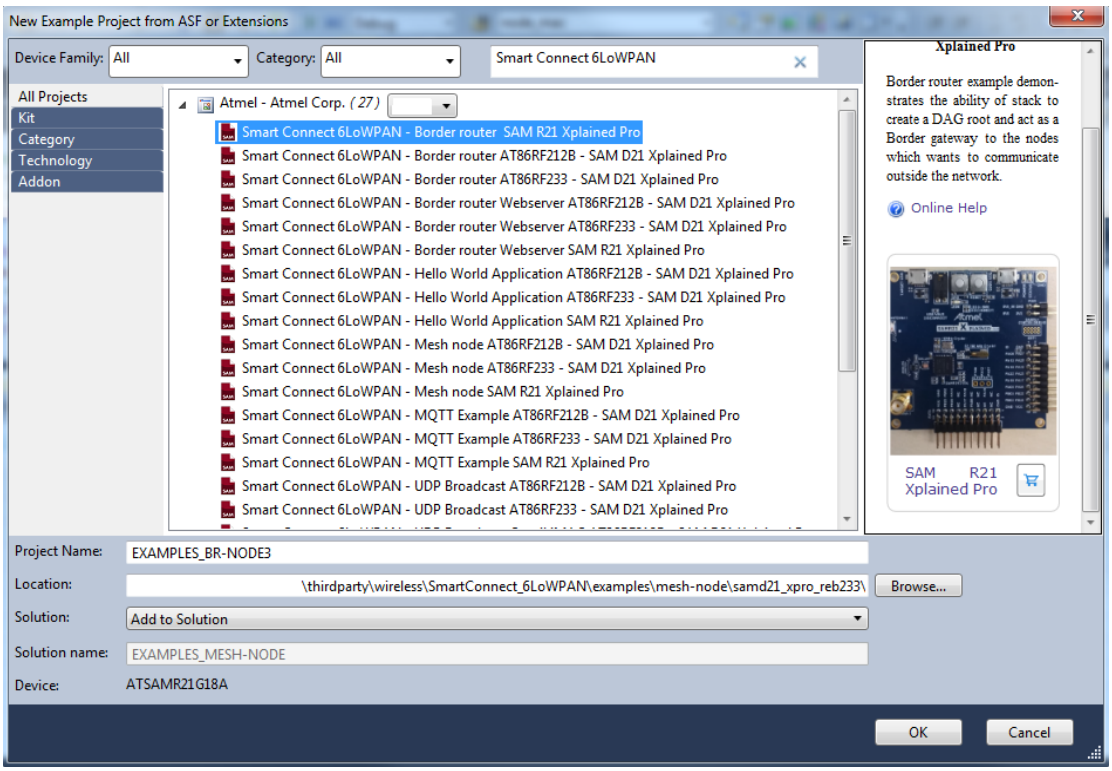
5.1 ASF example projects

SmartConnect 6LoWPAN application examples are distributed as part of Atmel Software Framework (ASF) [5] available via Atmel Studio. The following steps show how to access these ASF example projects from Studio:

1. Open Studio and close any existing project or solution to prevent the new ASF project from being appended to an existing solution containing another project. Now open File->New->Example Project



2. Search for SmartConnect 6LoWPAN and from available options, user can create example projects.



For more details on ASF directory structure please refer to Atmel Software Framework

5.2 hello-world example

This example is given just to understand the initialization procedure of the stack.

Once, Hello-world example is started, it will initialize all the configured required modules, initiate the RPL connection and starts a process thread to print “hello-world” in COM port of Host PC.

5.3 udp-broadcast-example

5.3.1 Introduction

This example demonstrates the stack ability to create a UDP socket and successfully send and receive UDP broadcast data over the network.

5.3.2 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.1

1. UDP port has to be mentioned for which Socket will be created.

```
#define UDP_PORT                    1234
```

2. Time interval for Periodic timer has to be configured.

```
#define SEND_INTERVAL               (5 * CLOCK_SECOND)
```

5.3.3 Application Setup:

1. This application requires 2 nodes of SAMR21 and SAMD21. For SAMD21, 2 transceiver boards (RF233/RF212B) are additionally required.
2. Create the example application as mentioned in Section 5.1 and flash the boards after building the example.
3. Connect the Transceiver board (RF233/RF212B) in EXT2 for SAMD21 projects.

5.3.4 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in broadcast-example-main.c)

Step 2: Start broadcast_example_process.

Step 3: Get link address and register UDP socket with simple_udp_sendto API API and also register callback API to receive data for this UDP port.

Step 4: Start periodic timer with SEND_INTERVAL as periodic value.

Step 5: Once periodic timer expired, send broadcast data using the registered UDP socket.

Step 6: Repeats step 5 for every Periodic timer expiry.

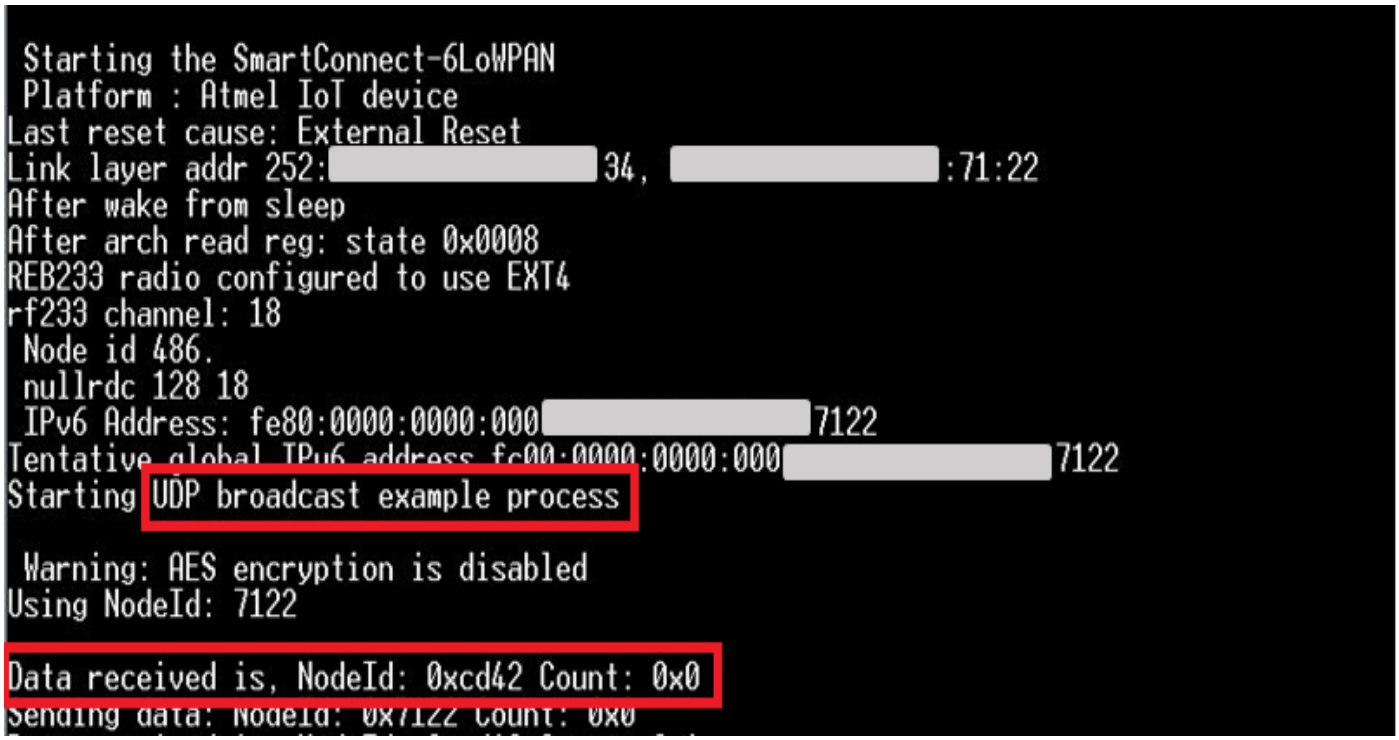
Step 7: In parallel, the callback function will receive data from other nodes.

INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

5.3.5 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 5-2. UDP broadcast node serial log



5.4 udp-unicast-receiver

5.4.1 Introduction

This example demonstrates the ability of stack to establish UDP connection with multiple devices and successfully receive data from network.

5.4.2 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.1

1. UDP port has to be mentioned for which Socket will be created.

```
#define UDP_PORT                    1234
```

2. Time interval for Periodic timer has to be configured.

```
#define SEND_INTERVAL               (5 * CLOCK_SECOND)
```

3. Server register ID

```
#define SERVICE_ID                  190
```

5.4.3 Application Setup:

1. This application requires 2 nodes of SAMR21 and SAMD21. For SAMD21, 2 transceiver boards (RF233/RF212B) are additionally required.
2. Create the example application as mentioned in Section 5.1 and flash the boards after building the example.

3. Connect the Transceiver board (RF233/RF212B) in EXT2 for SAMD21 projects.

5.4.4 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in udp-unicast-receiver-main.c)

Step 2: Start unicast_receiver_process.

Step 3: Initialize the servreg_hack module.

Step 4: Set global address using function set_global_address ().

Step 5: Make the device as root of the DAG network using function create_rpl_dag ().

Step 6: Register in servreg_hack module with defined `SERVICE_ID.`

Step7: Register UDP socket with simple_udp_sendto API.

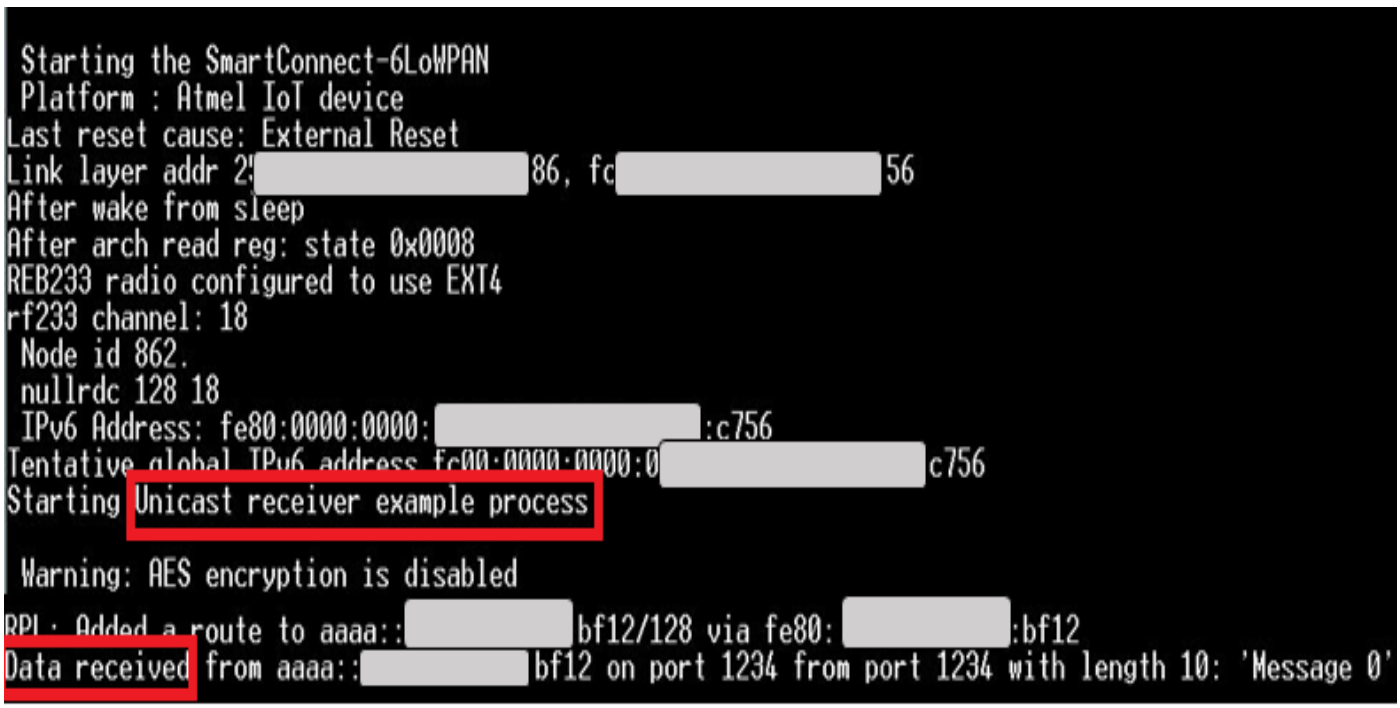
Step8: Wait for packets.

INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

5.4.5 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 5-3. UDP Unicast receiver serial log



5.5 udp-unicast-sender

5.5.1 Introduction

This example demonstrates the ability of stack to establish UDP connection and send UDP uni-cast data to a device.

5.5.2 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.1.

1. UDP port has to be mentioned for which Socket will be created.

```
#define UDP_PORT                    1234
```

2. Time interval for Periodic timer has to be configured.

```
#define SEND_INTERVAL               (5 * CLOCK_SECOND)
```

3. Server register ID

```
#define SERVICE_ID                  190
```

5.5.3 Application Setup:

1. This application requires 2 nodes of SAMR21 and SAMD21. For SAMD21, 2 transceiver boards (RF233/RF212B) are additionally required.
2. Create the example application as mentioned in Section 5.1 and flash the boards after building the example.
3. Connect the Transceiver board (RF233/RF212B) in EXT2 for SAMD21 projects.

5.5.4 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in udp-unicast-sender-main.c)

Step 2: Start unicast_sender_process.

Step 3: Initialize the servreg_hack module.

Step 4: Set global address using function set_global_address ().

Step 5: Register UDP socket with simple_udp_sendto API.

Step 7: Start periodic timer with SEND_INTERVAL as periodic value.

Step 7: Once periodic timer expired, lookup for `SERVICE_ID` in the servreg_hack module and check any receiver device has registered with the `SERVICE_ID`. If found, get the address of the receiver device.

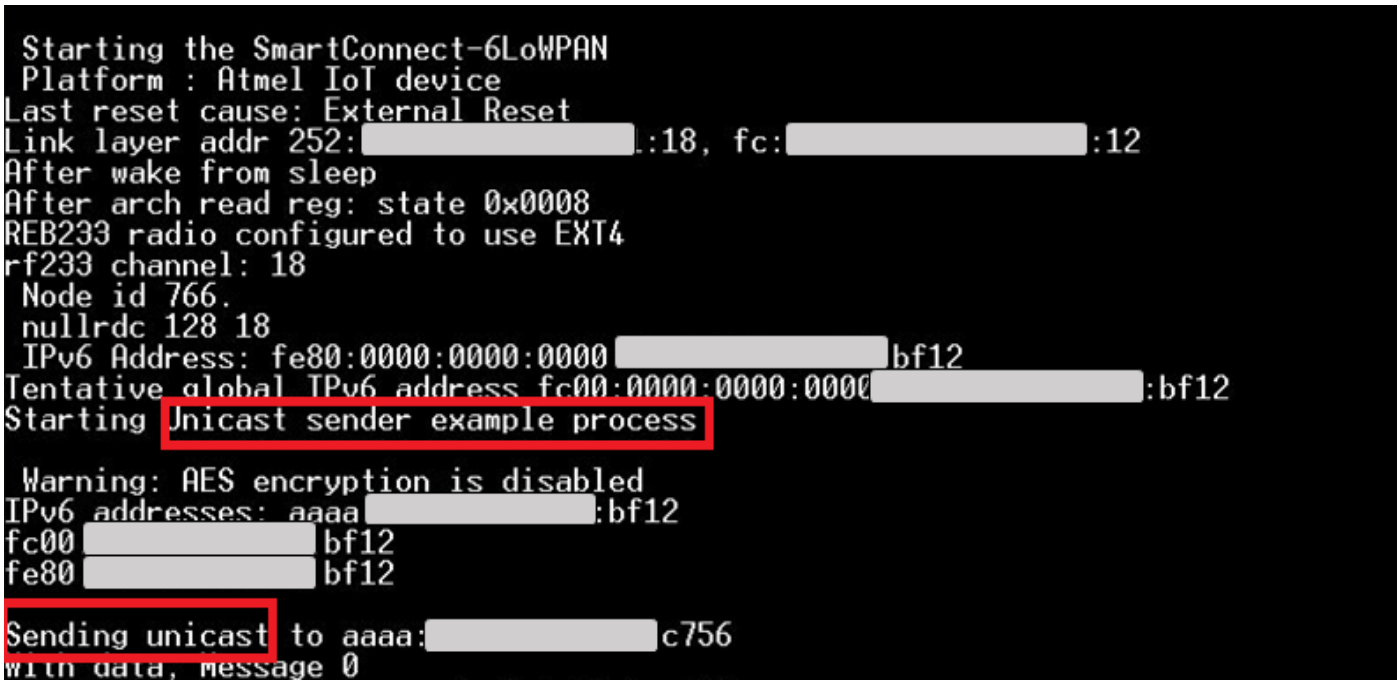
Step 8: After looked-up of global address of the receiver device, send UDP uni-cast message to the device.

INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

5.5.5 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 5-4. UDP unicast sender node serial log



5.6 mesh-node

5.6.1 Introduction

This example application has two options.

1. Mesh-node – Join an existing DAG network. Wait till DAG network is found.
2. Mesh-root – Create a DAG network and becomes root of the DAG network.

This example demonstrates the ability of stack to create DAG network, join the existing network and get notifications from UIP module.

5.6.2 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.1.

1. Define MESH_ NODE = 1 if the application has to behave as a node in the network.
2. Define MESH_NODE = 0 if the application has to be the root of the network.

TIP **To define MESH_NODE in go to Project>> Properties >> Toolchain >> Symbols and change the value of the MESH_NODE. To know more refer Atmel_Studio >> help**

5.6.3 Application Setup:

1. This application requires 2 nodes of SAMR21 and SAMD21. For SAMD21, 2 transceiver boards (RF233/RF212B) are additionally required.
2. Create the example application as mentioned in Section 5.1 and flash the boards after building the example.
3. Connect the Transceiver board (RF233/RF212B) in EXT2 for SAMD21 projects.

5.6.4 Application sequence

The sequence with which the example application executes, are given below,

5.6.4.1 Mesh-node

Step 1: Stack initialization (done mesh-node-main.c)

Step 2: Start mesh-node process

Step 3: Switch ON the LED.

Step 4: Add the call back function in the UIP-DS6 notification list using uip_ds6_notification_add (). So when route added for the node, UIP module will indicate the application by the calling the registered callback function.

Step 5: Once the callback function is called after adding a route, switch off the LED and print the route details.

5.6.4.2 Mesh-root

Step 1: Stack initialization (done in mesh-node-main.c)

Step 2: Start mesh-root process

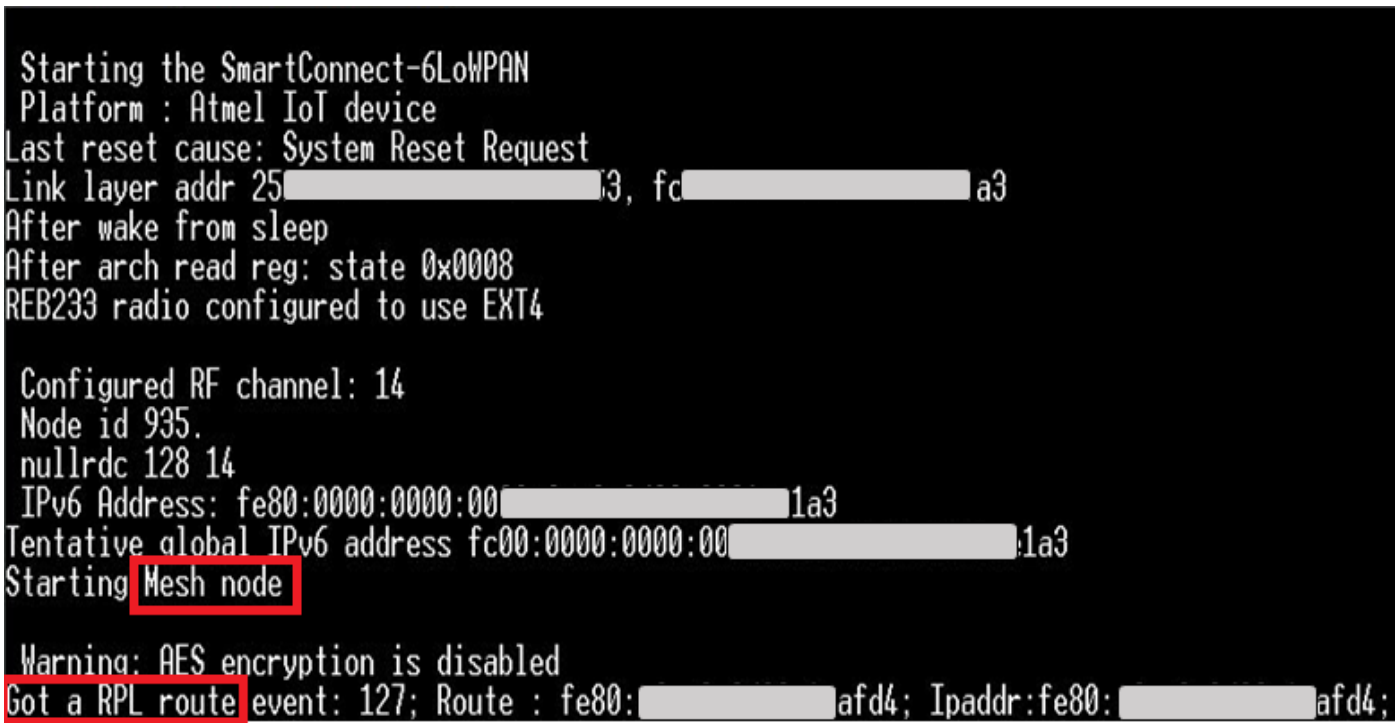
Step 3: Create the DAG root using simple_rpl_init_dag () function.

INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

5.6.5 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 5-5. Mesh node serial log



5.7 br-node

5.7.1 Introduction

Border Router example application uses IP64 module to communicate outside 6LoWPAN network, DHCP module to get an IPv4 address and simple RPL module to create RPL DAG root

This example demonstrates the ability of stack to create a DAG root and act as Border gateway to the nodes which wants to communicate outside the network.

This application uses IP64 module which converts IPv6 address of a node to IPv4 address and creates an entry in IP64 table and maintains it.

This Application uses DHCPv4 module to get an IPv4 address from outside the network.

5.7.2 Application setup

5.7.2.1 SAMR21 setup

- SAMR21_xpro boards – 4
- Ethernet1_xpro board - 1
- Compile and flash br-webserver-node to one of SAMR21_xpro + Ethernet1_xpro board.
- Compile and flash MQTT examples to 3 SAMR21_xpro boards.
- Once DHCP is configured, MQTT device can communicate to configured MQTT server. Refer Section 5.9 to know about MQTT device configuration.

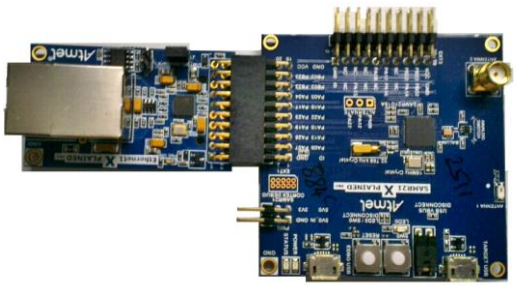
5.7.2.2 SAMD21 Setup

- SAMD21_xpro boards – 4
- RF233/RF212B boards – 4
- Ethernet1_xpro boards – 1
- Connect the Transceiver board (RF233/RF212B) in EXT2 for SAMD21 projects.
- Follow same steps as SAMR21 setup for bringing up the setup.

5.7.2.3 Usage of Ethernet1_xpro board

Connect Ethernet1_xpro board in EXT1 of SAMR21 board as shown below and connect Ethernet cable to Ethernet1_xpro board. For SAMD21 projects also connect Ethernet1_xpro boards in EXT1.

Figure 5-6. Border router node (SAMR21_xpro + Ethernet1_xpro)



5.7.3 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.1.

5.7.4 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in br-node-main.c)

Step 2: Start router_node_process.

Step 3: Call simple_rpl_init_dag () function to setup DAG network and become root of it.

Step4: Call ip64_init () function, to initialize IP64 module.

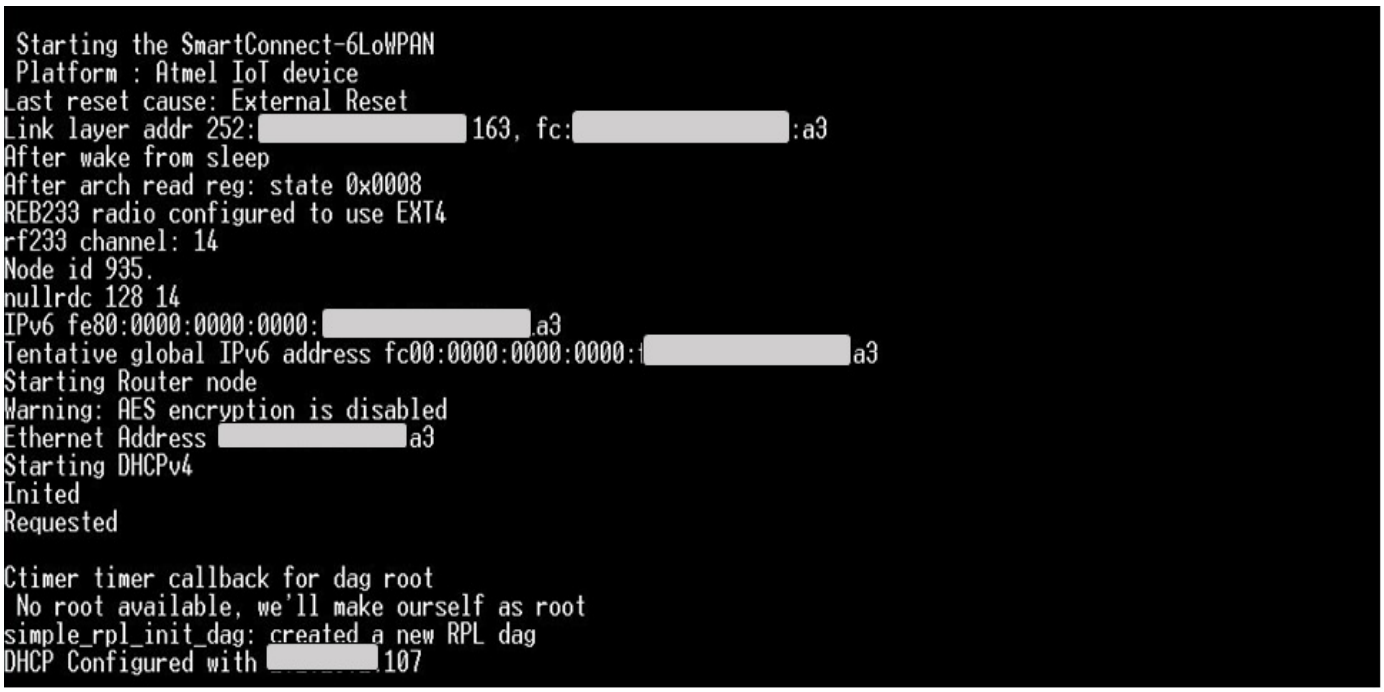
Step5: The IP64 module in turn will call DHCPv4 process and get dynamic IPv4 address.

INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

5.7.5 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 5-7. Border Router serial log



5.8 br-webserver-node

5.8.1 Introduction

This example demonstrates the ability of stack to create a DAG root and act as Border gateway to the nodes which wants to communicate outside the network.

5.8.2 Application setup

5.8.2.1 SAMR21 setup

- SAMR21_xpro boards – 4

- Ethernet1_xpro board - 1
- Compile and flash br-webserver-node to one of SAMR21_xpro + Ethernet1_xpro board.
- Compile and flash mesh-node examples to 3 SAMR21_xpro boards.
- Once DHCP is configured, view the mesh-node information from browser.

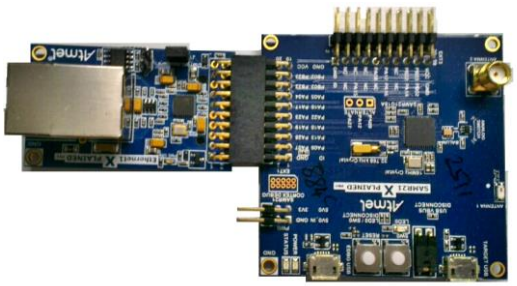
5.8.2.2 SAMD21 Setup

- SAMD21_xpro boards – 4
- RF233/RF212B boards – 4
- Ethernet1_xpro boards – 1
- Connect the Transceiver board (RF233/RF212B) in EXT2 for SAMD21 projects.
- Follow same steps as SAMR21 setup for bringing up the setup.

5.8.2.3 Usage of Ethernet1_xpro board

Connect Ethernet1_xpro board in EXT1 of SAMR21 board as shown below and connect Ethernet cable to Ethernet1_xpro board. For SAMD21 projects also connect Ethernet1_xpro boards in EXT1.

Figure 5-8. Border router web-server node (SAMR21_xpro + Ethernet1_xpro)



5.8.3 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.1.

5.8.4 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in br-node-main.c)

Step 2: Start router_node_process.

Step 3: Call simple_rpl_init_dag () function to setup DAG network and become root of it.

Step 4: Call ip64_init () function, to initialize IP64 module.

Step 5: Call ip64_webserver_init () function to initialize the IP64-WEBSEVER module and create default HTML pages.

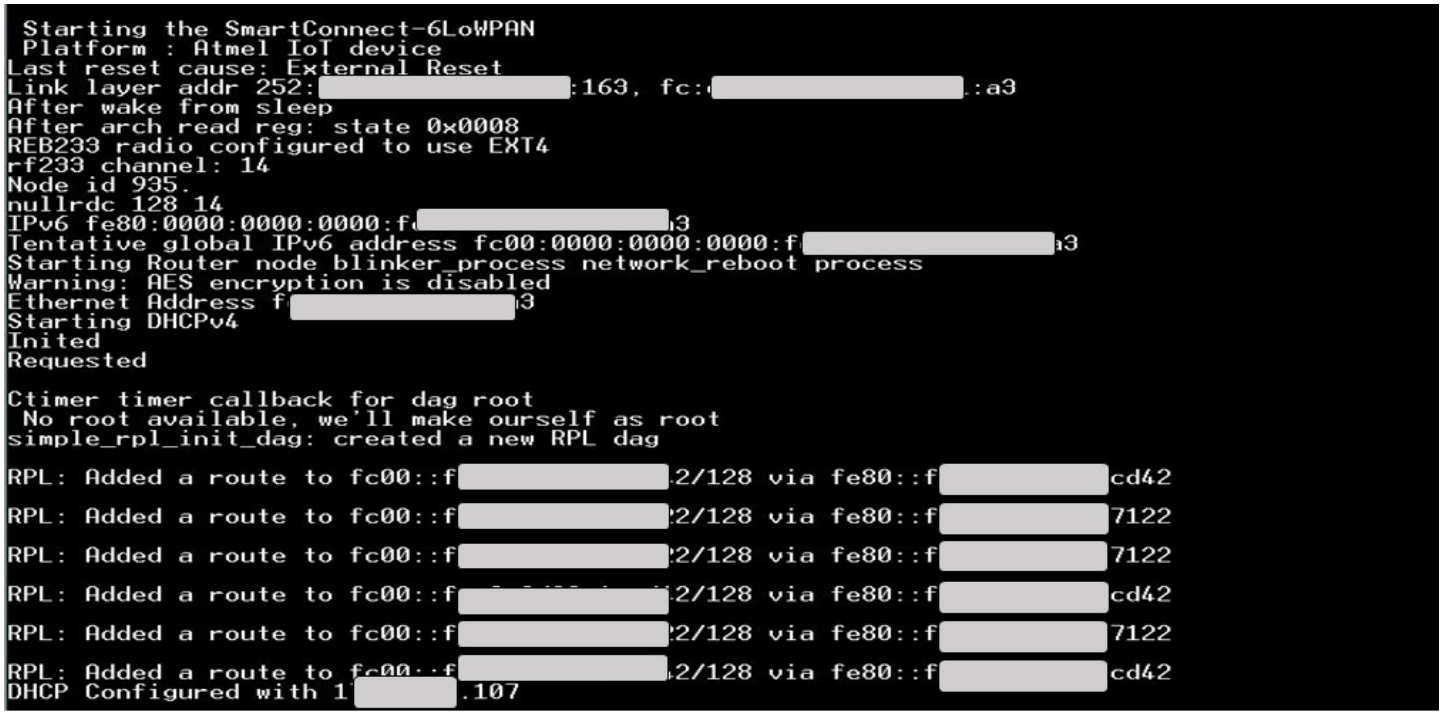
Step 6: The IP64 module in turn will call DHCPv4 process and get dynamic IPv4 address.

INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

5.8.5 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 5-9. Border router web server serial log

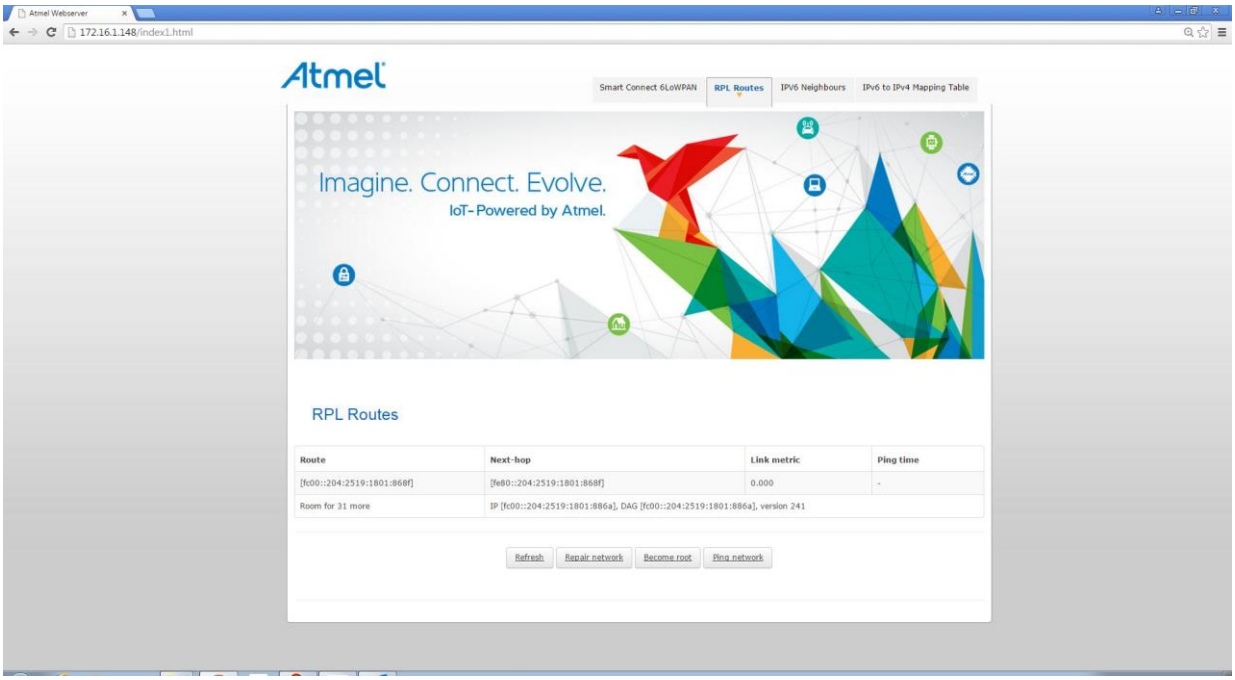


Once the DHCP is configured with IPv4 address, user can view the HTML pages from web server from a PC connected inside the LAN network. Type the IPv4 address shown in the Tera Term in the Web browser to view the information about the nodes.

This web-server has a feature to view the devices inside the 6LoWPAN network and ping them individually.

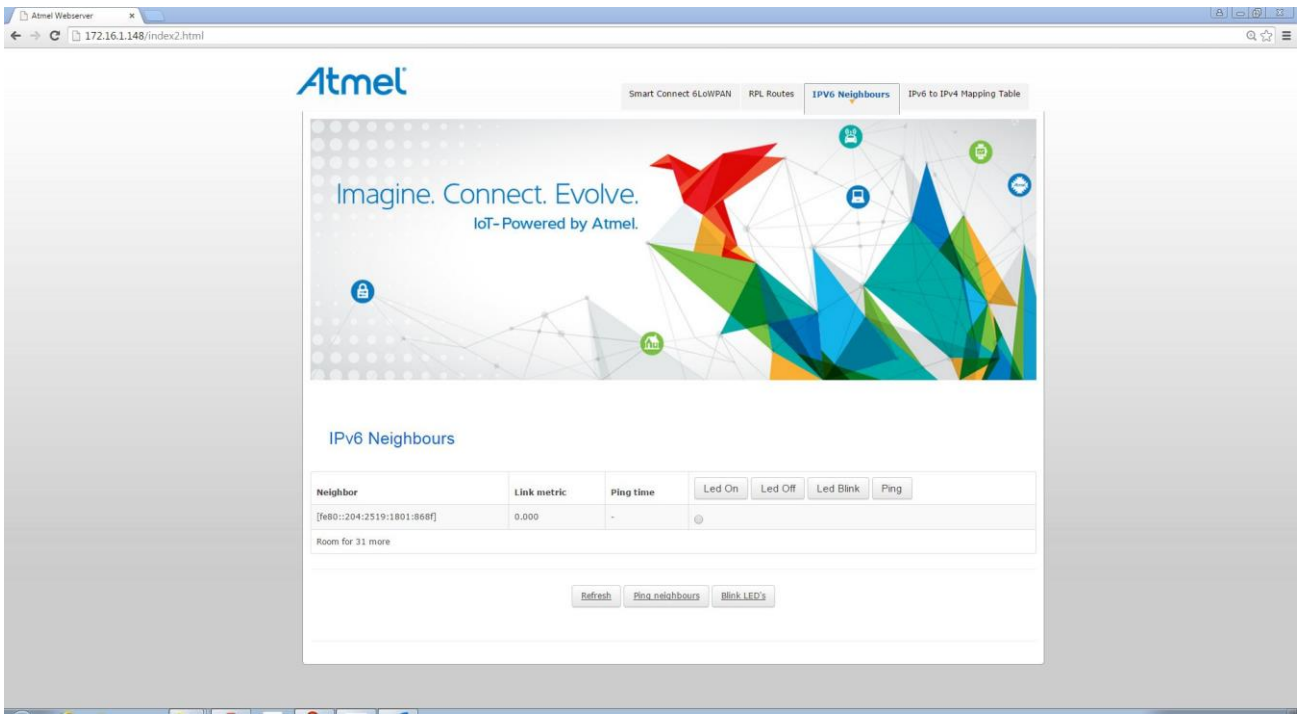
Additionally, Neighbor table and IP64 table also can be viewed from Web-Server.

Figure 5-10. Border Router Web Page – RPL Routes



The RPL routes view will show the RPL route information to the nodes connected to the border router node.

Figure 5-11. Border Router Web Page - IPv6 Neighbors



We can perform one of the four actions (LED on, off, blink and Ping) on individual nodes – by selecting its corresponding radio button:

1. **Ping:**

After clicking ping, we get a screen saying, “Ping scheduled”. Upon refreshing after some time, we get the echo delay as shown below.

2. LED On, Off, Blink:

The border router, after receiving command from browser, sends unicast UDP message to the target node on a dedicated port (defined in contiki-conf.h as LED_UDP_PORT) indicating the LED command. The neighbor nodes will have a UDP receive callback registered for this dedicated port. The callback calls the appropriate routine to control the LED GPIO.

Also the commands “Ping neighbors” and “Blink LEDs” will target the entire neighbor list.

Figure 5-12. Border Router - LED On/OFF

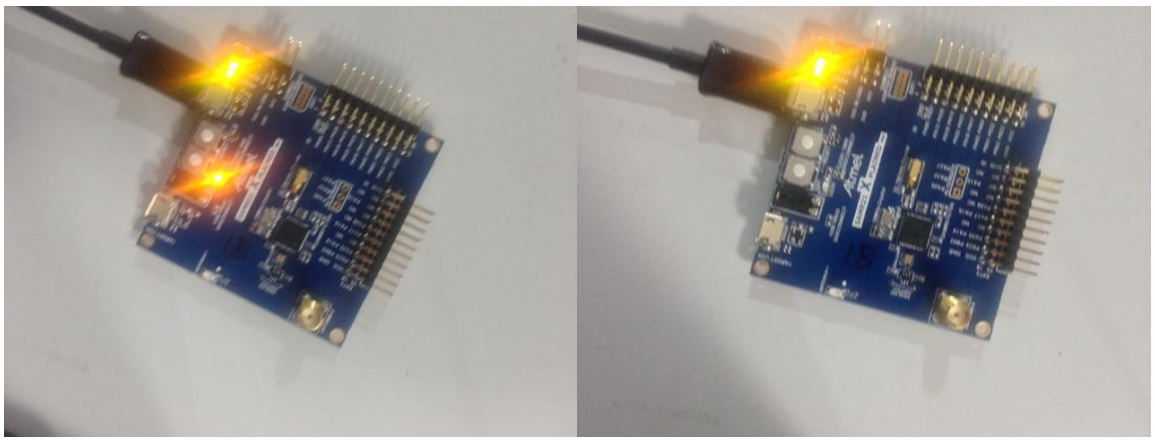


Figure 5-13. Border Router Web Page - Ping Scheduled

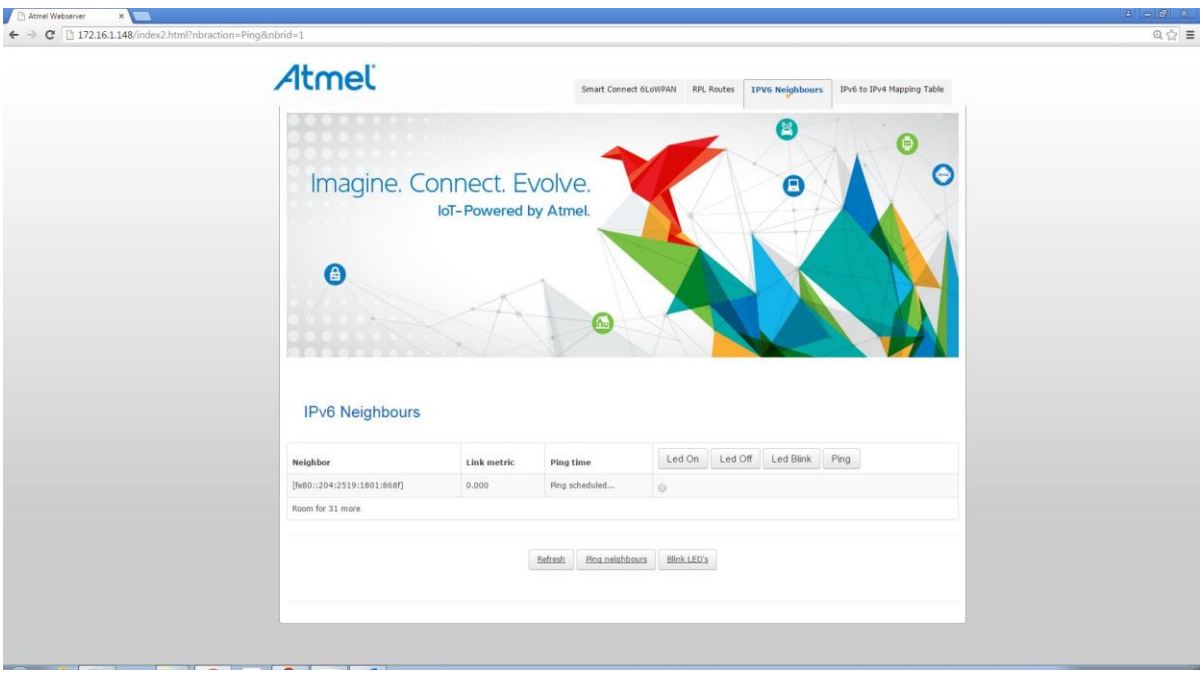


Figure 5-14. Border Router Web Page - Updated ping time to the node

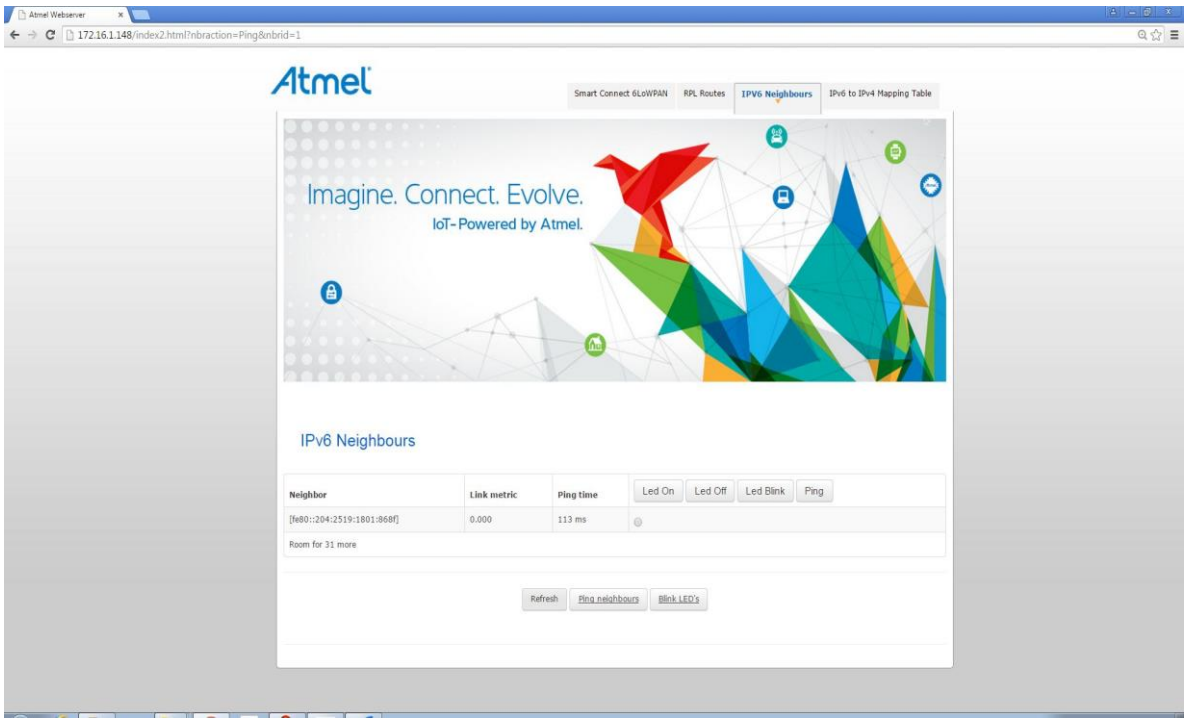
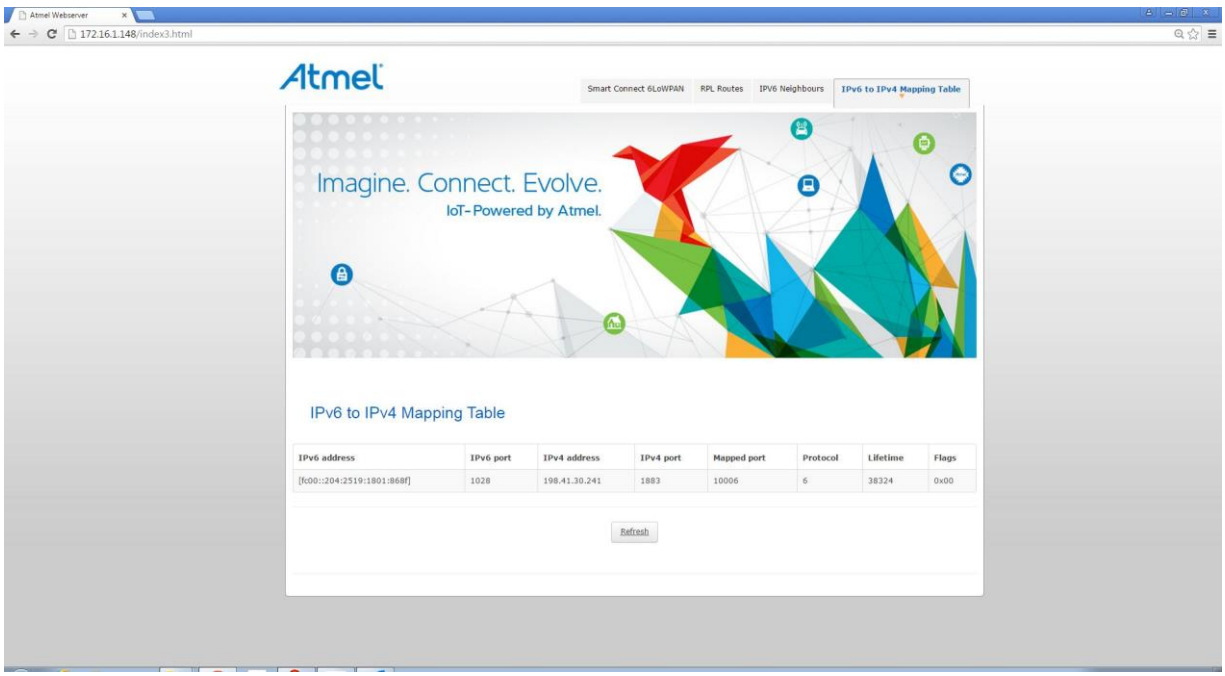


Figure 5-15. Border Router Web Page - IPv6 to IPv4 Mapping table



NAT mapping is used to address the IPv6 end nodes using the gateway's DHCP IPv4 address to talk to the external world through an IPv4 based ISP. The port mappings are listed for various destination ports (and destination IPv4 addresses).

5.9 mqtt-example

5.9.1 Introduction

This example,

1. Will publish data to the MQTT host broker configured with configured topic.
2. Uses IO1_xpro board, to get Temperature and Light sensor details of the current environment and send them to MQTT broker in JSON format.
3. Uses Border router to connect outside the 6LoWPAN network.
4. Uses MDNS module, to get IP address of MQTT host broker configured. MDNS server used here, is Google DNS server (::ffff:8.8.8.8).

5.9.2 Application setup

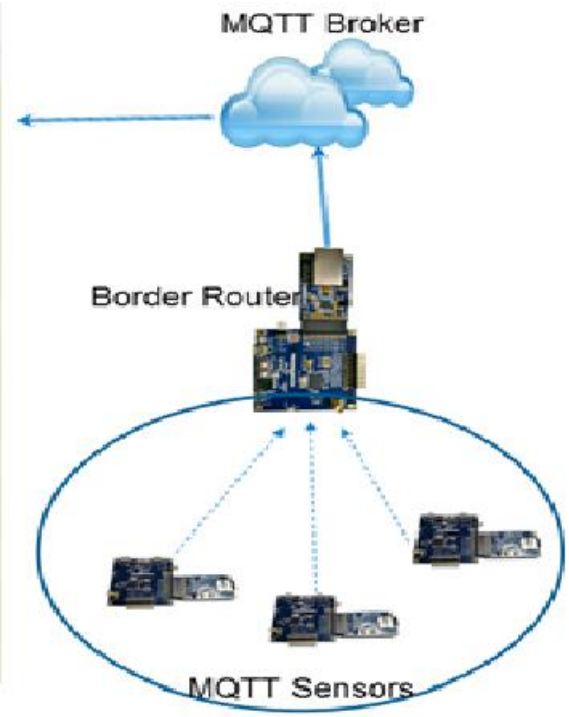
5.9.2.1 SAMR21 setup

Refer to Section 5.7 or Section 5.8 for the Border Router setup details.

Connect IO1_xpro board to SAMR21 board. Compile and flash MQTT example code in SMR21 board.

Example setup looks as shown on below figure.

Figure 5-16. MQTT Setup



5.9.2.2 Usage of IO1_xpro board

Connect IO1_xpro board in EXT1 of SAMR21 board as shown.

Figure 5-17. MQTT Setup

5.9.3 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.1.

In mqtt-example.c, following macros has to be modified in each device for a MQTT Application to run in that specific device.

```
/* MQTT Configuration details */
#define HOST          "m2m.eclipse.org"
#define VERSION       "v1"
#define PRIORITY      "p0"
#define UUID          "atmeld"
```

INFO **The UUID will be the MQTT Client name and it'll register with MQTT Host Broker with same name.**

5.9.3.1 MQTT Client Configuration

The topics of MQTT Client are currently defined as follows.

1. `/<version>/<priority>/<UUID>/sensor` - topic for subscribe. This is the topic, we need specify in MQTT app (MQTT Lens for browser or MY MQTT android app) to get sensor data from each MQTT device.
2. Currently both Light and Temperature sensor data are integrated (IO1 Xpro board).
3. All sensor data are sent in JSON format as seen below.

```
{"timestamp": "141397","SENS_TEMPERATURE": "74.7","SENS_LIGHT_LEVEL": "2612","sender_id": "02[redacted]79"}
```

4. `/<version>/<priority>/<UUID>/led` – topic for publishing to client. Currently to publish data to MQTT device, use the above topic and send data as "on" or "off" to make LED ON/OFF.

5.9.4 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in mqtt-example-main.c)

Step 2: Start mqtt_example_process.

Step 3: Call uip_ds6_get_link_local () to get link local address, which sent in Publish data.

Step 4: Configure topics based on configured details and set periodic timer to publish MQTT data.

Step 5: Resolve the MQTT host broker address using MDNS module.

Step 6: Register MQTT connection with given `UUID`.

Step 7: Initialize IO1_xpro board to get Temperature and Light sensor details.

Step 8: Find RPL parent and add route to it using simple_rpl_parent ().

Step 9: Connect to MQTT broker using mqtt_connect () API.

Step 10: Once MQTT connection is established, Publish Status Topic stating the device is “online” and Subscribe to the LED topic.

Step 11: Once periodic timer is expired, Collect Temperature and Light sensor details from IO1_xpro board.

Step 12: Put the collected details, timestamp and MAC address of the device in JSON format.

Step 13: Send the JSON formatted data to MQTT broker.

Step 14: Repeat from **Step 11** for every periodic timer expiry.

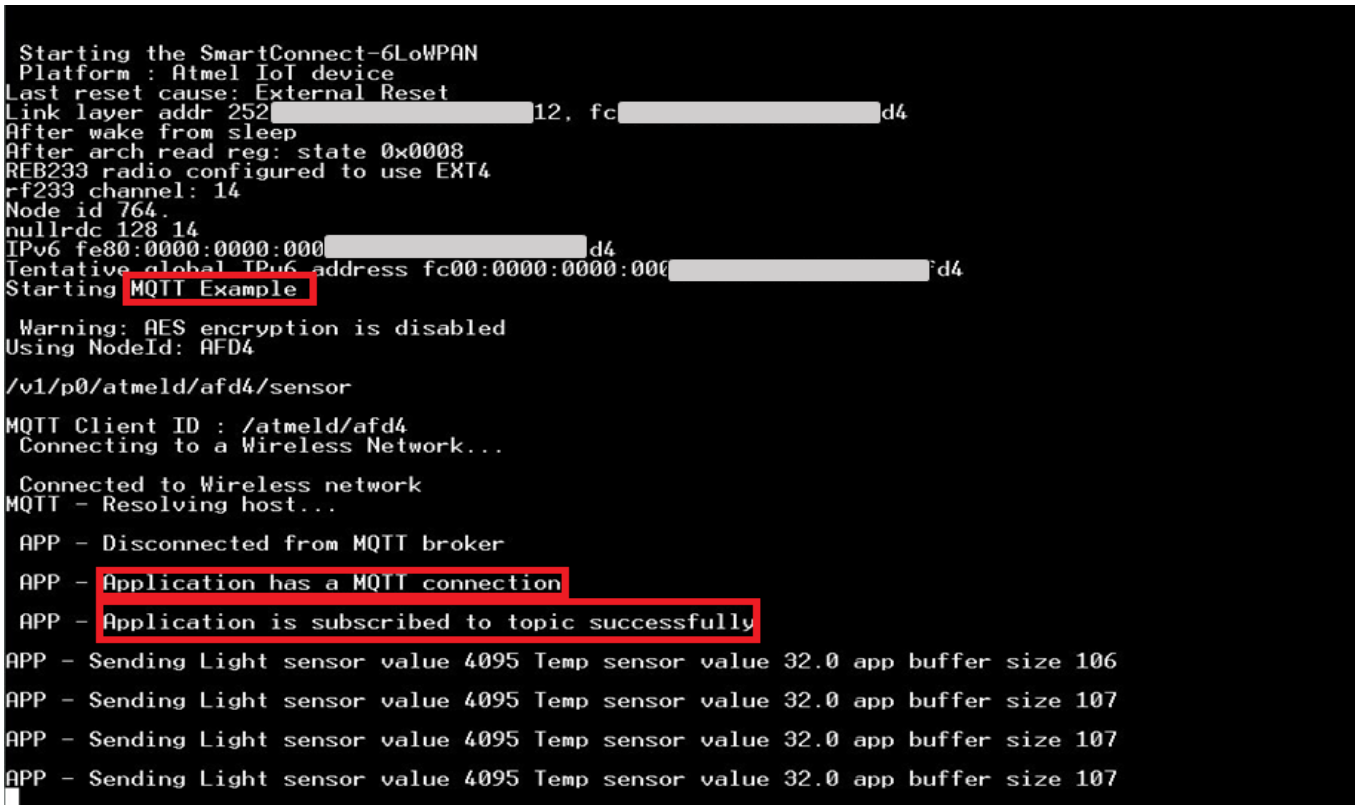
INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

5.9.5 Application execution snapshot

5.9.5.1 Device serial snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

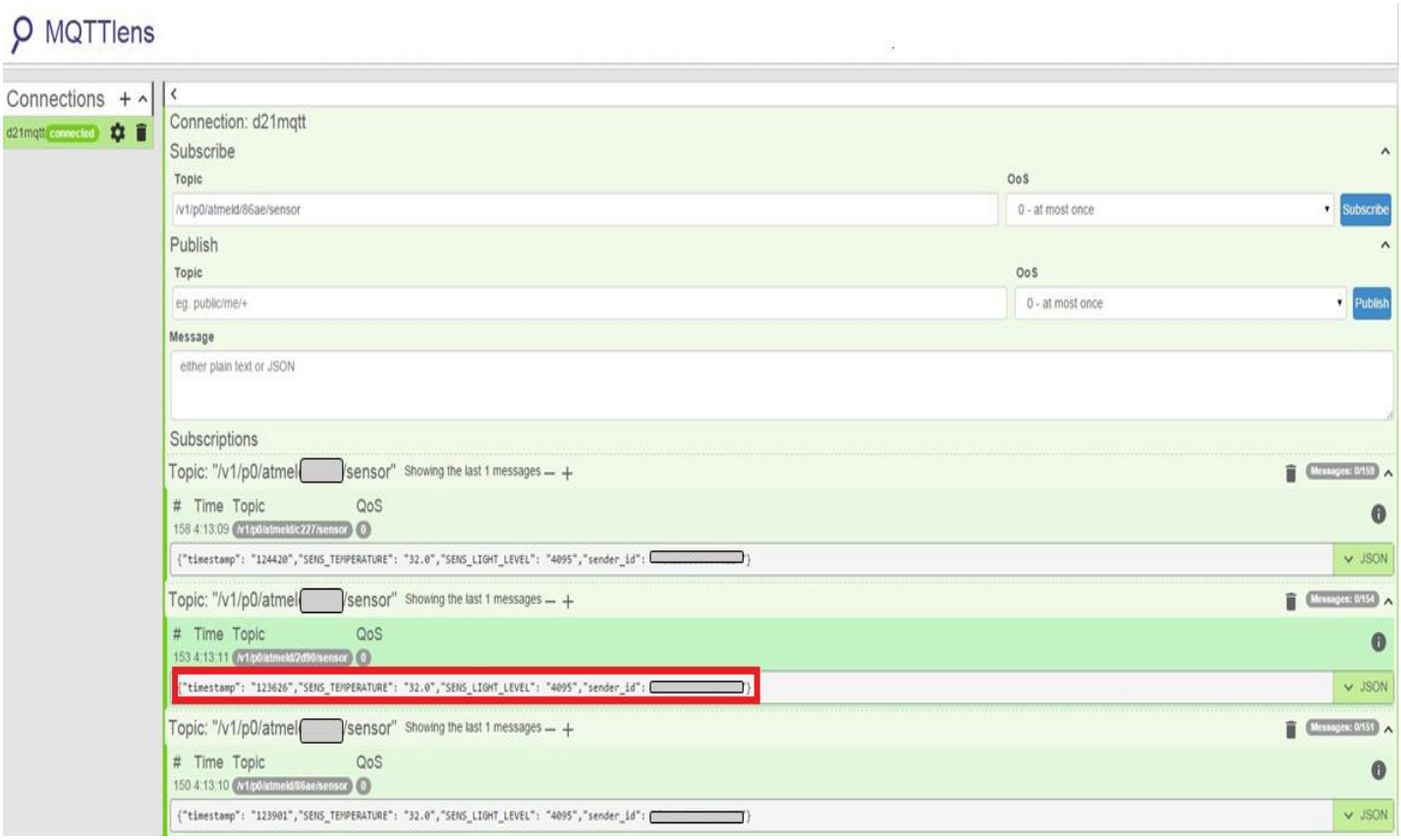
Figure 5-18. Sensor node serial log



5.9.5.2 MQTT lens screenshot.

1. Install MQTT lens in Chrome browser.
2. Establish a connection to the MQTT host broker.
3. After connected to MQTT broker, subscribe for the topic configured in each device in the setup. For eg: in above screen shot, the topic is `/v1/p0/atmeld/afd4/sensor.`
4. From the MQTT lens Application, you can each device published data in JSON format.

Figure 5-19. MQTT Subscription logs in MQTT lens app



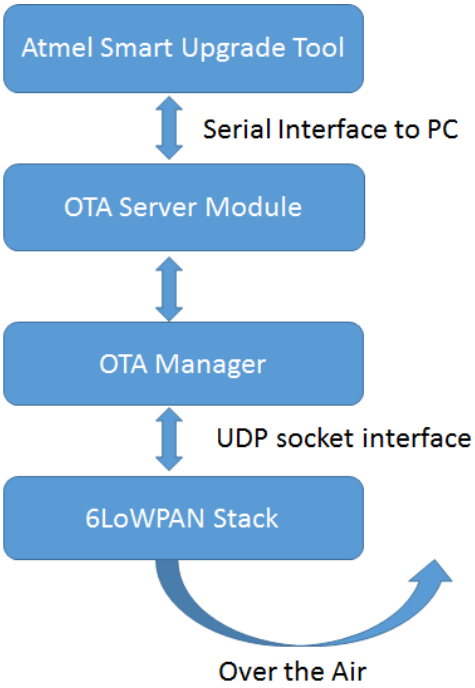
5.10 Otau-server example

5.10.1.1 Introduction

OTAU server example application receives/transmits the command from/to PC through UART/USB. Transmits required frames through stack over UDP socket to reach the clients, in order to upgrade it.

There are three components on top of 6LoWPAN stack in Over-The-Air Server application,

- Atmel Smart Upgrade Tool – the PC tool used to select OTA client node and upgrade the same in the process with new image.
- OTA Server module – OTA server module takes care of Receive packets from PC tool and forwards the same to OTA manager to send the packet to respective client and vice versa. OTA server module is distributed in the form of library.
- OTA manager – OTA manager is the interface between OTA server module and 6LoWPAN stack. This module receives the data from OTA server module and translates into UDP packet. The UDP packet then sent to 6LoWPAN stack to be sent to OTA client node.

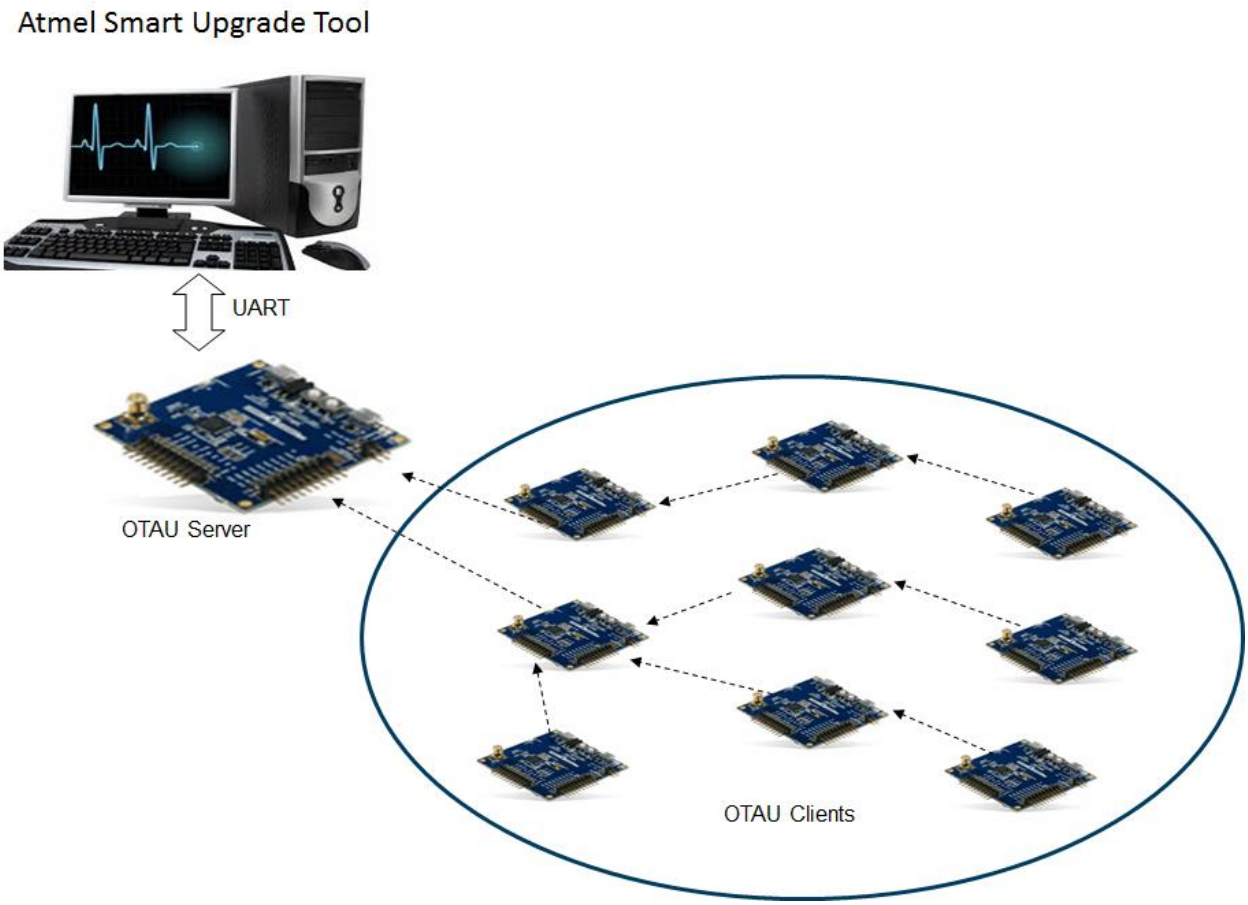


5.10.1.2 Application setup

- 2 SAMR21_Xpro boards.
- Atmel Smart upgrade tool installed in PC.
- Open, Build and flash OTA server application image in the SAMR21_Xpro board.
- Connect the Atmel Smart Upgrade tool to OTA Server node.
- Open, Build and Flash OTA Client image in the SAMR21_Xpro board.
- Refer SmartConnect Upgrade Tool user guide to get details on upgrading a new image in OTA client node.

INFO **SmartConnect Upgrade tool can be found in /doc folder**

Figure 5-20. OTA Upgrade Setup



5.10.1.3 Configuration

Application configurations are

```
/* UDP Port to used for send/receive OTA data */

#define UDP_PORT 5321

/* Service ID for sending OTA server address to all OTA client */

#define SERVICE_ID 190
```

5.10.1.4 Application Sequence

Step 1: Stack initialization (done in ota-main.c)

Step 2: Start `otau_server_process`

Step 3: Call simple_rpl_init_dag () function to setup DAG network and become root of it.

Step 4: Intialization of ota_manager and udp socket is done.

Step 5: Wait for data to be received from OTA Server Module and create the UDP data to forward it to 6LoWPAN stack. Similarly forward the data received from OTA client to OTA server module.

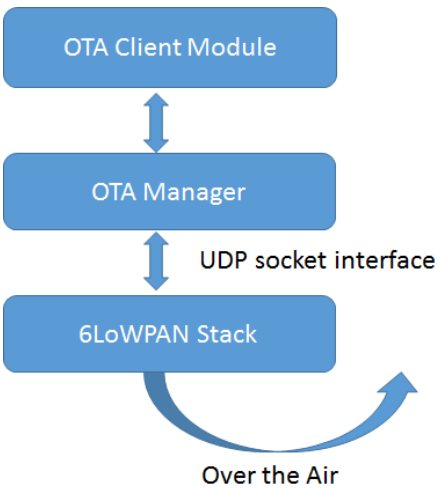
5.11 Otau-client example

5.11.1.1 Introduction

OTAU client application features are as follows,

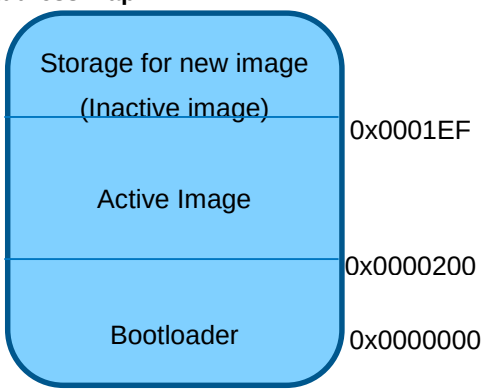
- OTAU client application updates image information to OTAU server regularly.
- When OTAU server indicates a new image is available, OTAU client initiates Upgrade procedure and requests OTAU server for new image.
- After receiving the new image, OTAU client switches to boot-loader and to switch to new image.
- OTAU client uses internal flash to store the new image in inactive section, which is described below.

Figure 5-21. OTA Client module



Getting image from server and validating its integrity is the responsibility of OTAU Client module in stack. Once the image is available in inactive space, the OTAU client will write the details of new image to end of flash and it will reset to boot-loader. So the inactive image has to be overwritten on active image. In order to do this, the AVR2054 serial boot-loader has to be modified as described below.

Figure 5-22. OTA client address map



Bootloader start address = 0x00000000

Bootloader size = 8192 (0x2000) bytes

Node information start address = 0x0003FFC0

Node information size = 64 bytes

So in order to divide the remaining flash into two equal segments,

Active image start address = 0x00002000

Active image size or Inactive image size = (Total flash size – (Bootloader size + Node information size)) / 2

= (0x00040000 – (0x00002000 + 0x40)) / 2

*= **0x0001EFE0** bytes*

Inactive image start address = Active image start address + Active image size

= 0x00002000 + 0x0001EFE0

*= **0x00020FE0***

Note: Current setting for Inactive image start address on OTAU Client manager is 0x0001EFE0, this needs to be corrected to 0x00020FE0. We will fix this in upcoming version.

OTAU client contains following components,

- OTAU client module –
  - This module is responsible for updating the server about client nodes image information and parent details.
  - Upgrade of new image handling. From the new image information received from OTAU server, requests OTAU server for image block.
  - Write the received image blocks to the inactive region of the memory.
  - Verification of integrity after receiving full image. After integrity of image is verified, send switch request to OTAU Server.
  - Switch to boot-loader after receiving switch confirm is also done by this module.
- OTAU manager –
  - This module is responsible for finding the OTAU server address.
  - Send/Receive data from OTAU server.
  - UDP socket creation and timer handling.

5.11.1.2 AVR2054 Serial boot-loader changes

Below mentioned code is used to read from inactive space to active space during boot up.

```
#define FLASH_SIZE                    0x40000 /* 256 kB */
#define FLASH_PAGE_SIZE       64
#define FLASH_ADDR                    (0x00000000U) /**< FLASH
base address */


#define NVM_DEVICE_INFO_LOCATION      ((FLASH_ADDR +
FLASH_SIZE) - 64UL)
//Index to retrieve new image's starting address from node info
#define NEW_IMAGE_START_INDEX         (0)
//Index to retrieve new image's size from node info
#define NEW_IMAGE_SIZE_INDEX          (4)
//Index to identify whether new image available in inactive
region
#define NEW_IMAGE_INDEX               (8)


#define NEW_IMAGE_LOCATION            (0x1EFE0UL)


void main(void)
{
      uint8_t new_image;
      uint32_t new_image_loc = NEW_IMAGE_LOCATION;
      new_image = *(uint8_t *)(NVM_DEVICE_INFO_LOCATION +
      NEW_IMAGE_INDEX);
      if (new_image == 1)
      {
            uint16_t read_block[32];
            uint32_t row = 0;
            uint32_t new_image_start = (uint32_t) *((uint32_t
            *)(NVM_DEVICE_INFO_LOCATION +
            NEW_IMAGE_START_INDEX));
            uint32_t new_image_size = (uint32_t) *((uint32_t
            *)(NVM_DEVICE_INFO_LOCATION +
            NEW_IMAGE_SIZE_INDEX));
            while(row < new_image_size)
            {
                  memcpy(read_block, (uint8_t *)(new_image_loc
            + row), FLASH_PAGE_SIZE);
                  HAL_FlashWrite(new_image_start + row,
            (uint8_t *)&read_block[0], FLASH_PAGE_SIZE);
                  if ((row + FLASH_PAGE_SIZE) > new_image_size)
                  {
                    row += new_image_size - row;
```

```
                    else
                    {
                      row += FLASH_PAGE_SIZE;
                    }
              }
              memcpy(&appStartAddr, &new_image_start,
              sizeof(new_image_start));
              flashUpdateAppStartAddr();
              hwStopWdt();
              jumpToApplication();
        }
}
```

Following function needs to be added in flashloader.c of AVR2054 Serial bootloader…

```
void HAL_FlashWrite(uint32_t address,uint8_t *data,uint16_t length)
{
      uint8_t temp_buffer[FLASH_PAGE_SIZE];
      uint8_t page_offset;
      uint16_t currentPage = 0;
      uint16_t rem_len,curr_len;
      curr_len = rem_len = length;
      uint32_t page_start,page_address;

      page_address = address;
      while(rem_len)
            {
                  page_offset = page_address%FLASH_PAGE_SIZE ;
                  page_start = page_address-page_offset;
                  currentPage = page_start/FLASH_PAGE_SIZE;

                  if((page_offset!=0) &&  (page_offset + rem_len >
      FLASH_PAGE_SIZE))
                  {
                        curr_len = FLASH_PAGE_SIZE - page_offset;
                  }
```

```
            else if (rem_len/FLASH_PAGE_SIZE)
            {
            curr_len = FLASH_PAGE_SIZE;
            }
            else
            {
            curr_len = rem_len;
            }

            for(uint16_t index = 0; index < FLASH_PAGE_SIZE;
index++)
            {
                  temp_buffer[index] =    *((uint8_t
*)page_start+index);
            }

            hwEraseFlashPage(currentPage*FLASH_PAGE_SIZE);

            for(uint16_t index = page_offset; index <
page_offset+curr_len; index ++)
            {
                  temp_buffer[index] = *data++;
            }

            hwFillFlashPageBuffer(currentPage*FLASH_PAGE_SIZE,
FLASH_PAGE_SIZE, &temp_buffer[0]);
            hwWriteFlashPage(currentPage*FLASH_PAGE_SIZE);

            rem_len-=curr_len;
            page_start+= FLASH_PAGE_SIZE;
            page_address = page_start;
   }
```

INFO **Please refer AVR2054 Serial and OTA boot-loader [9] [10] for more details of boot-loader and boot-loader tool.**

5.11.1.3 Application configuration

Application configurations in sc6-ota-mgr.c are

```
/* UDP Port to used for send/receive OTA data */
#define UDP_PORT 5321
/* Service ID for sending OTA server address to all OTA client */
#define SERVICE_ID 190
```

Following definitions inside SmartConnect_6LoWPAN/services/ota_mgr/ota_mgr.h, reflects the implementation on inactive region for bootloader image swapping. This can be modified to adjust the offset based on application size.

```
/* Address of end of flash for storing device information */
#define NVM_DEVICE_INFO_LOCATION      ((FLASH_ADDR + FLASH_SIZE) - 64UL)

/* Address of mid of Flash page with 64byte at end for storing device
information */
#define OFFSET_ADDRESS    (0x1EFE0UL)
```

6 Abbreviations

6LoWPAN – IPv6 packets over Wireless Personal Area Network

ASF – Atmel Software Framework

AES – Advanced Encryption Standard

CoAP – Constrained Application Protocol

CSMA – Carrier Sense Multiple Access

DHCP – Dynamic Host Control Protocol

HTTP – Hypertext Transfer Protocol

IPHC – IP packet Header Compression

IPv6 – IP version 6

IP – Internet Protocol

MAC – Medium Access Control

MCU – Micro Controller

MDNS – Multicast Domain Name System

MQTT – Message Queue Telemetry Transport

RDC – Radio Duty Cycle

RPL – Routing Protocol for Lossy and Low power Networks

UDP – User Datagram Protocol

TCP – Transmission Control Protocol

7 References

[1] Atmel Wireless MCU Software Website

[2] Atmel Wireless Support

[3] IEEE Std 802.15.4™-2006 Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)

[4] Atmel documents for supported families and boards

[5] Atmel Software Framework

[6] Atmel Studio

[7] IETF RPL RFC 6550

[8] IETF Transmission of IPv6 packets over IEEE 802.15.4 networks RFC 4944

[9] Atmel AVR2054 Serial and OTA boot-loader

[10] Atmel AVR2054 Serial and OTA boot-loader User Guide.

[11] ARM GCC Release version.

ATMEL EVALUATION BOARD/KIT IMPORTANT NOTICE AND DISCLAIMER

This evaluation board/kit is intended for user's internal development and evaluation purposes only. It is not a finished product and may not comply with technical or legal requirements that are applicable to finished products, including, without limitation, directives or regulations relating to electromagnetic compatibility, recycling (WEEE), FCC, CE or UL. Atmel is providing this evaluation board/kit “AS IS” without any warranties or indemnities. The user assumes all responsibility and liability for handling and use of the evaluation board/kit including, without limitation, the responsibility to take any and all appropriate precautions with regard to electrostatic discharge and other technical issues. User indemnifies Atmel from any claim arising from user's handling or use of this evaluation board/kit. Except for the limited purpose of internal development and evaluation as specified above, no license, express or implied, by estoppel or otherwise, to any Atmel intellectual property right is granted hereunder. ATMEL SHALL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMGES RELATING TO USE OF THIS EVALUATION BOARD/KIT.

ATMEL CORPORATION
1600 Technology Drive
San Jose, CA 95110
USA

Revision History

Doc Rev.	Date	Comments
A	04/2015	Initial document release.

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**