Application Note

Connect One TI 15.4-Stack Sensor to Multiple Gateways

Andres Mondragon Clavijo

ABSTRACT

The TI 15.4-Stack is a complete software solution for developing applications that require extremely low-power, long-range, reliable, robust and secure wireless star-topology based networking solutions. This application note describes how to add multiple gateway support to a network using the TI 15.4-Stack with the SimpleLink™ CC13xx and CC26xx wireless microcontrollers (MCUs). Multiple gateway support enables additional network features such as node balancing, increased robustness and extended network coverage. First, a brief overview of what the TI 15.4-Stack offers is presented. Next, some of the benefits of having multiple gateway support are discussed. And finally, the necessary steps to enable multiple gateway support for Non-beacon and Beacon modes are described.

Table of Contents

Trademarks

SimpleLink™ and LaunchPad™ are trademarks of Texas Instruments.
Zigbee™ is a trademark of Zigbee Alliance.
Wi-SUN® is a registered trademark of Wi-SUN Alliance.
All trademarks are the property of their respective owners.

Copyright © 2021 Texas Instruments Incorporated

1 Introduction

The TI 15.4-Stack is a wireless stack component of the SimpleLink™ MCU platform that implements the standard IEEE 802.15.4e and 802.15.4g specification. For Sub-1 GHz frequencies, the TI 15.4-Stack also implements a frequency-hopping scheme adopted from the Wi-SUN® field area network (FAN) specification, offering a complete solution based on a star-network topology for connecting long-range, low power sensors in smart home, building and city applications.

Among other things, the TI 15.4-Stack offers network formation and maintenance with scalable network features, three modes of operation (synchronous mode, asynchronous mode, frequency hopping mode), and MAC level packet encryption based on AES 128 CCM*.

Visit the Sub-1 GHz overview to learn more about the key network features.

2 Benefits of Having Multiple Gateway Support

Even though the TI 15.4-Stack offers scalable network features out-of-the-box, features such as node balancing, extended network coverage, and increased node count, need to be implemented at the application layer. This is because a TI 15.4 network operates in a star topology in which only one node serves as PAN coordinator and the rest of the nodes simply report to it.

Having said this, there is nothing in the TI 15.4-Stack that prevents an application from relying on multiple PAN coordinators to form a larger network composed by several TI 15.4 networks.

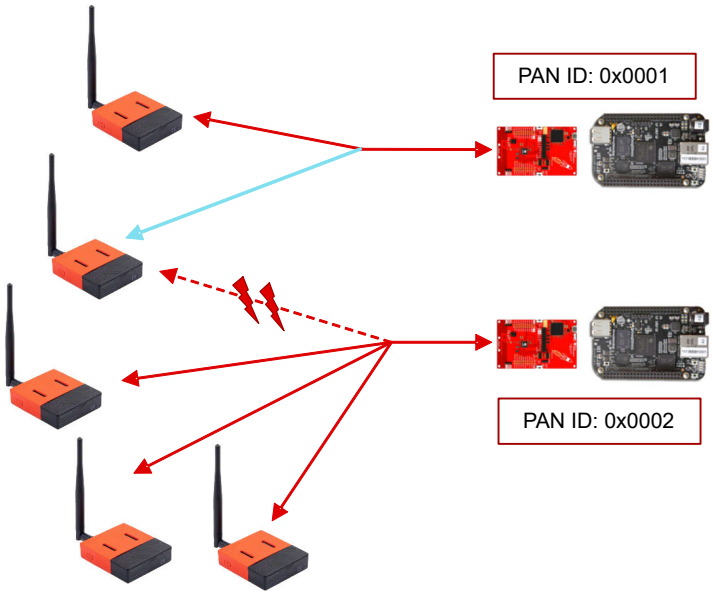
The following section describes the previously mentioned features, so that the reader can determine whether or not implementing multiple gateway support might be suitable for a particular use-case or end product.

2.1 Node Balancing

In the context of a TI 15.4 network, node balancing refers to the network's capability to distribute network traffic across different PAN coordinators. This is necessary because if there is only one PAN coordinator to control a high number of sensor devices, the network might become congested and the coordinator will not be able to add new devices to the network. By distributing the traffic load evenly, the overall responsiveness of the application improves.

Figure 2-1 shows how node balancing works. The bottom coordinator initially has four associated sensors while the top coordinator only has one. In order to balance the load, one of the sensors switches its coordinator.

Copyright © 2021 Texas Instruments Incorporated



TI 15.4 Network PAN ID	Number of devices (before node balancing)	Number of devices (after node balancing)
0x0001	1	2
0x0002	4	3

Figure 2-1. Node Balancing With Multiple Gateways

2.2 Robustness

Robustness refers to the ability of a network to withstand failures and perturbations. This feature is particularly relevant for a TI 15.4 network because it relies on a star topology in which one node (the PAN coordinator) is a single point of failure for the network.

Suppose for example that a TI 15.4 network is deployed with some sensor devices and after some time the PAN coordinator fails. If the PAN coordinator does not recover, all the sensors in the network go into what is known as an orphan state, and try (in vain) to rejoin the PAN coordinator until their power source fails. By having a network in which there are multiple gateways acting as PAN coordinators, sensor devices can at any point in time switch to a different PAN coordinator making the network more robust and resilient.

2.3 Extended Coverage and Network Redundancy

Extended coverage and redundancy in “difficult” RF spots are two additional network features that are enabled with multiple gateway support. The extended coverage feature is self-explanatory. If multiple gateways are part of the network and they are positioned so that they cover different areas, the overall range of the network increases.

Redundancy in “difficult” RF spots, on the other hand, refers to improving the availability of the network by positioning gateways in such a way that there is a slight overlap in their individual network coverage. The overlap allows the network to support nodes which might have been deployed in what could be described as a “difficult” RF spot (a location where RF signals are attenuated or where there is a significant amount of RF noise). Therefore, if for some reason the sensor cannot communicate properly with its PAN coordinator, it can join a different PAN coordinator that belongs to the same network and which might be in range.

Copyright © 2021 Texas Instruments Incorporated

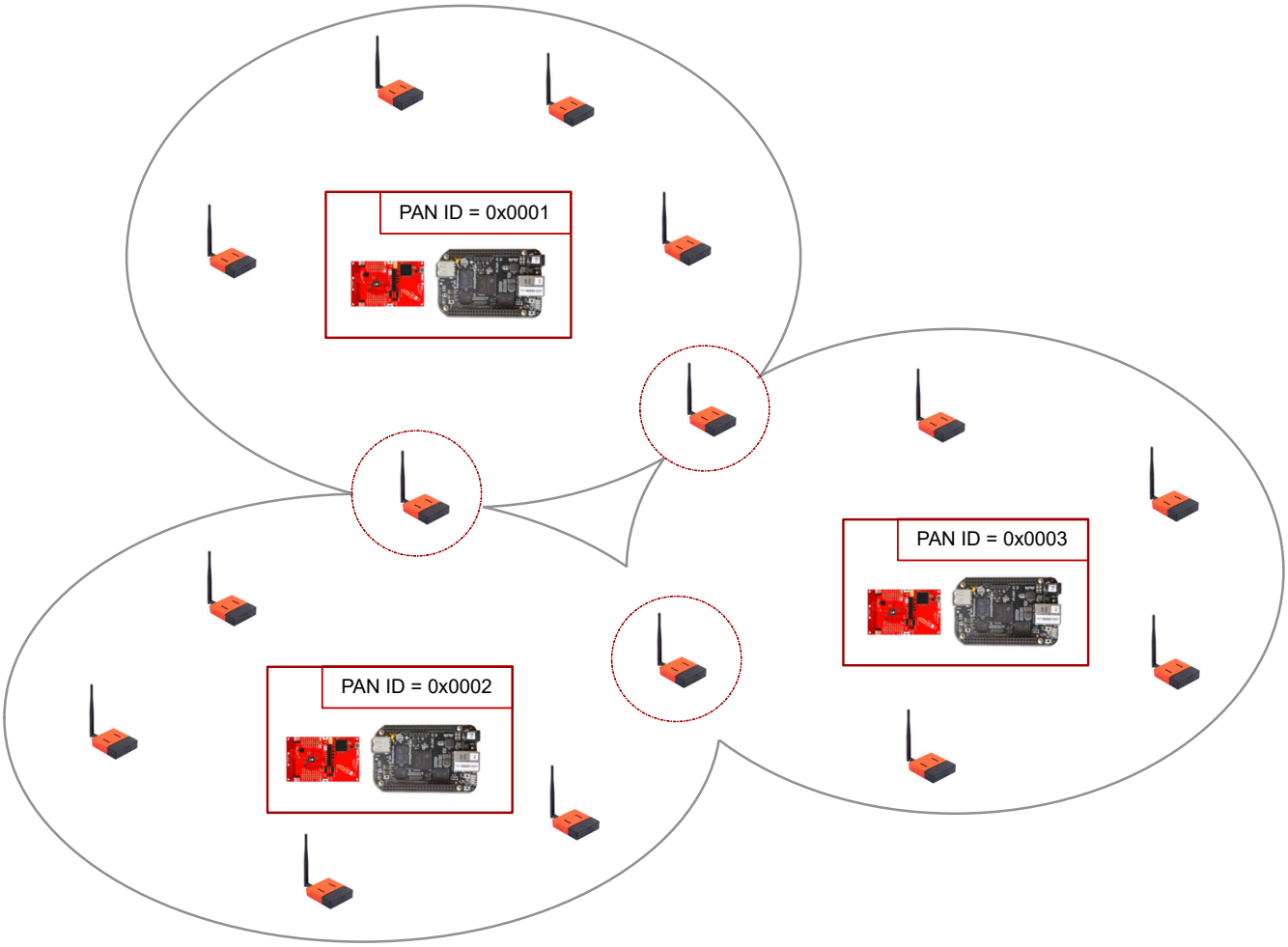


Figure 2-2. Extended Coverage and Network Redundancy for "Difficult" RF Spots

3 Current SDK Examples and Coprocessor Configuration

Among the TI 15.4-Stack examples included in the SimpleLink SDK, there are three examples which are particularly relevant for this application note. The first two correspond to the sensor and collector examples which showcase the scenario in which a single TI 15.4 network is formed and maintained by a single collector device, and then one or more sensor devices join the network. The third example, on the other hand, corresponds to the co-processor example which could be described as an extension of the collector example that adds gateway capabilities to the PAN coordinator of the TI 15.4 network.

Even though these examples only cover the use-case when there is one collector device acting as PAN coordinator and one or more sensor devices reporting to it, this does not mean that the TI 15.4-Stack cannot be used to form a more complex network with extended network coverage, increased node count, redundancy for nodes in "difficult" RF spots, and node balancing capabilities.

To get a clear understanding of how to do this, first consider the coprocessor configuration described in the TI 15.4-Stack User's Guide.

In this configuration, the protocol stack runs on a CC13xx or CC26xx device while the application is executed on an external MPU or MCU. Because the host application executes on an external MPU or MCU, it can run protocol stack layers over the IEEE 802.15.4 e/g MAC and PHY (for example, generic IP over 6LoWPAN, Zigbee™ IP, or Zigbee Pro) making the co-processor configuration suitable for applications that require long-range wireless connectivity. In simple terms, in the co-processor configuration the collector device operates not only as a TI 15.4 PAN coordinator but also as a local gateway capable of communicating with other local gateways.

Copyright © 2021 Texas Instruments Incorporated

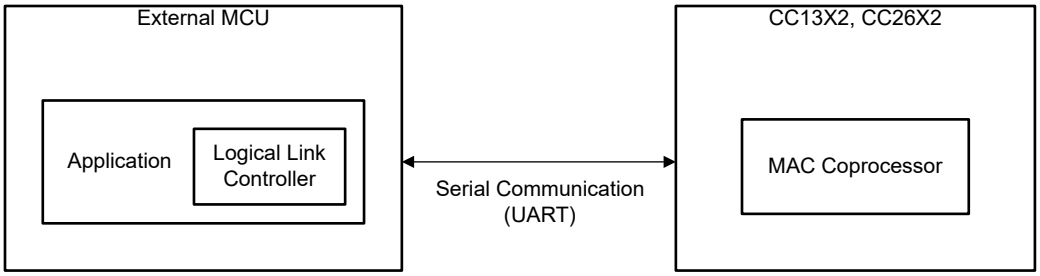


Figure 3-1. High-Level Software Architecture of the TI 15.4-Stack Coprocessor Configuration

Consequently, if a single source, which from now on will be referred as central gateway, coordinates a group of local gateways, it is possible to create a larger network that is composed of many TI 15.4 local networks and that is capable of supporting the previously mentioned network features.

4 Central Gateway

The scope of this application report is not to provide a reference design for a central gateway, but to describe the technical aspects associated with enabling a multiple gateway environment when using the TI 15.4-Stack. Nonetheless, it does make sense to at least include a description of the parameters related to the TI 15.4-Stack that a central gateway implementation should consider.

Before describing the tasks associated with the central gateway, it is first useful to understand what a local gateway is, and what a local gateway does. As mentioned before, a local gateway is nothing more than a collector device with added capabilities to run protocol stack layers over the IEEE 802.15.4 e/g MAC and PHY. The collector device acts as a PAN coordinator for the TI 15.4 network, meaning that it is in charge of creating and maintaining the network, while also providing synchronization services to devices such as sensors (that have joined the network), and other network coordinators.

The reason behind providing synchronization services to other network coordinators is that before a collector forms a TI 15.4 network, it first needs to check that no other coordinator is operating on the same channel or with the same PAN ID.

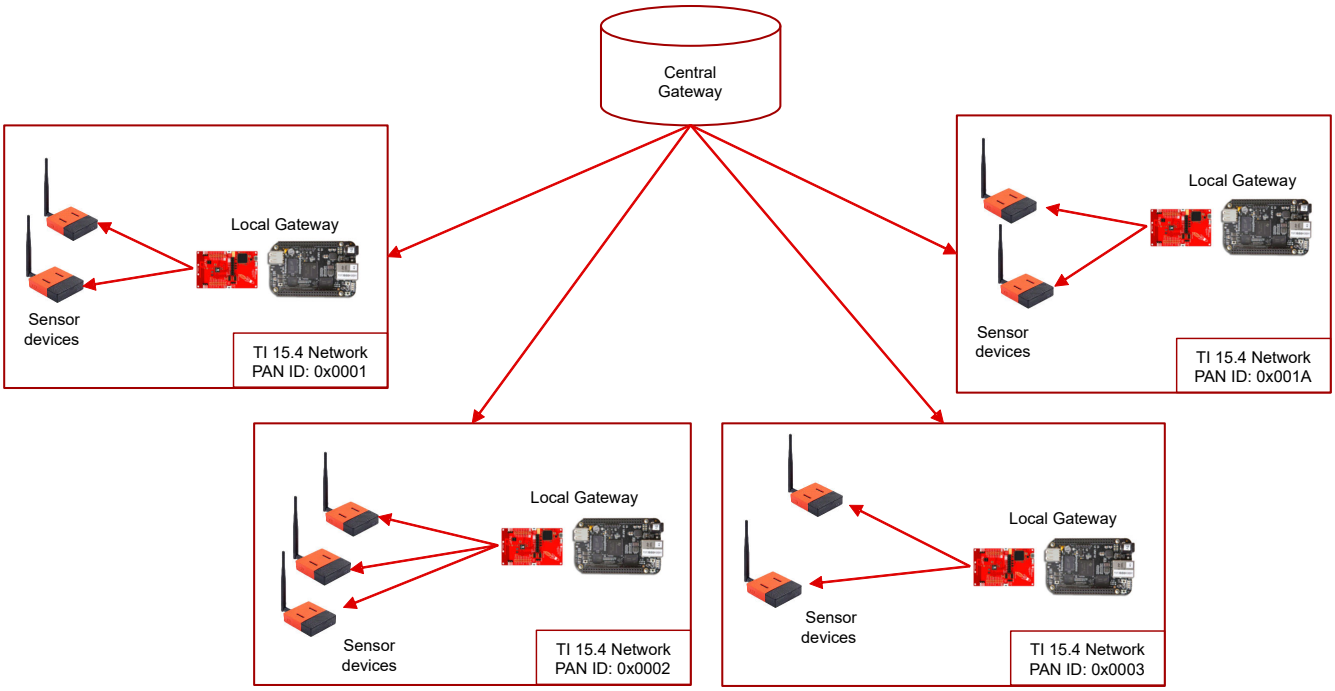


Figure 4-1. Example of a Large Network With a Central Gateway Controlling Multiple Local Gateways

Moreover, the local gateway must be able to identify which sensor devices can join the network and which sensor devices are currently active in the network. It does this by relying on an association procedure that

Copyright © 2021 Texas Instruments Incorporated

verifies that the sensor device has the appropriate network key, and by adding approved sensor devices to an “accept list” if the current resources available on the PAN are sufficient. This “accept list” contains the following information:

- PAN ID of the device
- Short address
- Extended address
- Frame counter
- Device capability information

Consequently, a central gateway implementation must also provide synchronization services, and function as the coordinator for the local gateways that compose the network. Having said this, even though the central gateway must keep the information about the local TI 15.4 networks, it does not need to communicate with the PAN coordinators over the IEEE 802.15.4 e/g MAC and PHY.

The following information related to each local TI 15.4 network needs to be managed by the central gateway:

- Short address mapping: The central gateway should allocate short address spaces to each of the local PAN coordinators to avoid assigning duplicate short addresses to the devices in the network.
- MAC encryption key: Current SDK examples use pre-shared keys for the sensors and coordinators.
- 16-bit PAN identifier of the network
- 16-bit short address of the PAN coordinator
- 64-bit IEEE extended address of the PAN coordinator
- Security: Enabled or disabled
- Mode and operating channel: Network operating mode (beacon, non-beacon, frequency hopping)
- 16-bit short address of the sensor devices in the local network (assigned by the PAN coordinator)
- 64-bit IEEE extended address of the sensor devices in the local network
- Frame counter (replay protection)

5 Enabling Multiple Gateway Support

A critical part of enabling the previously described network features has to do with providing a sensor device with the capability to switch between different PAN coordinators.

Consider for instance a network which is initially configured with one local gateway and a large amount of sensor devices. Given the distance between some of the sensor devices and the gateway, packets are being dropped, affecting the quality of the network. As new devices are added to the network, it is determined by the network’s administrator, that a second local gateway must be deployed to act as PAN coordinator for new sensor devices, and to improve the network coverage for previously deployed sensors.

Nevertheless, given that a sensor device can only exchange packets with its current PAN coordinator, the newly deployed local gateway, does not have a beneficial impact on the network’s coverage for the already deployed sensors. In order for the network to fully benefit from the addition of the second local gateway, the application must consider a procedure so that the sensors that have been previously deployed can switch between PAN coordinators.

This section describes the modifications needed so that an application can support PAN coordinator switching and therefore, multiple gateway support. The following steps require no modification to the TI 15.4-Stack and use the sensor and collector examples included in the SimpleLink SDK as base projects. Furthermore, the steps cover Beacon and Non-Beacon mode for the two most-common coordinator switching scenarios.

5.1 PAN Coordinator Switching Due to Sync Loss

Even though a sync loss between a sensor device and its PAN coordinator is usually something temporal, it can become permanent if the PAN coordinator is no longer reachable.

If this is the case, the sensor device transitions into what is known as an orphan state. In this state, the sensor device enters a low power mode, and only periodically awakes to attempt to reconnect to its PAN coordinator. If the PAN coordinator is in fact no longer reachable because it has failed or it has been removed, the sensor will indefinitely (or until its power source fails) attempt to reconnect to it. Therefore, if the sensor is to remain active in the network, the implementation of a PAN coordinator switching procedure is necessary.

Copyright © 2021 Texas Instruments Incorporated

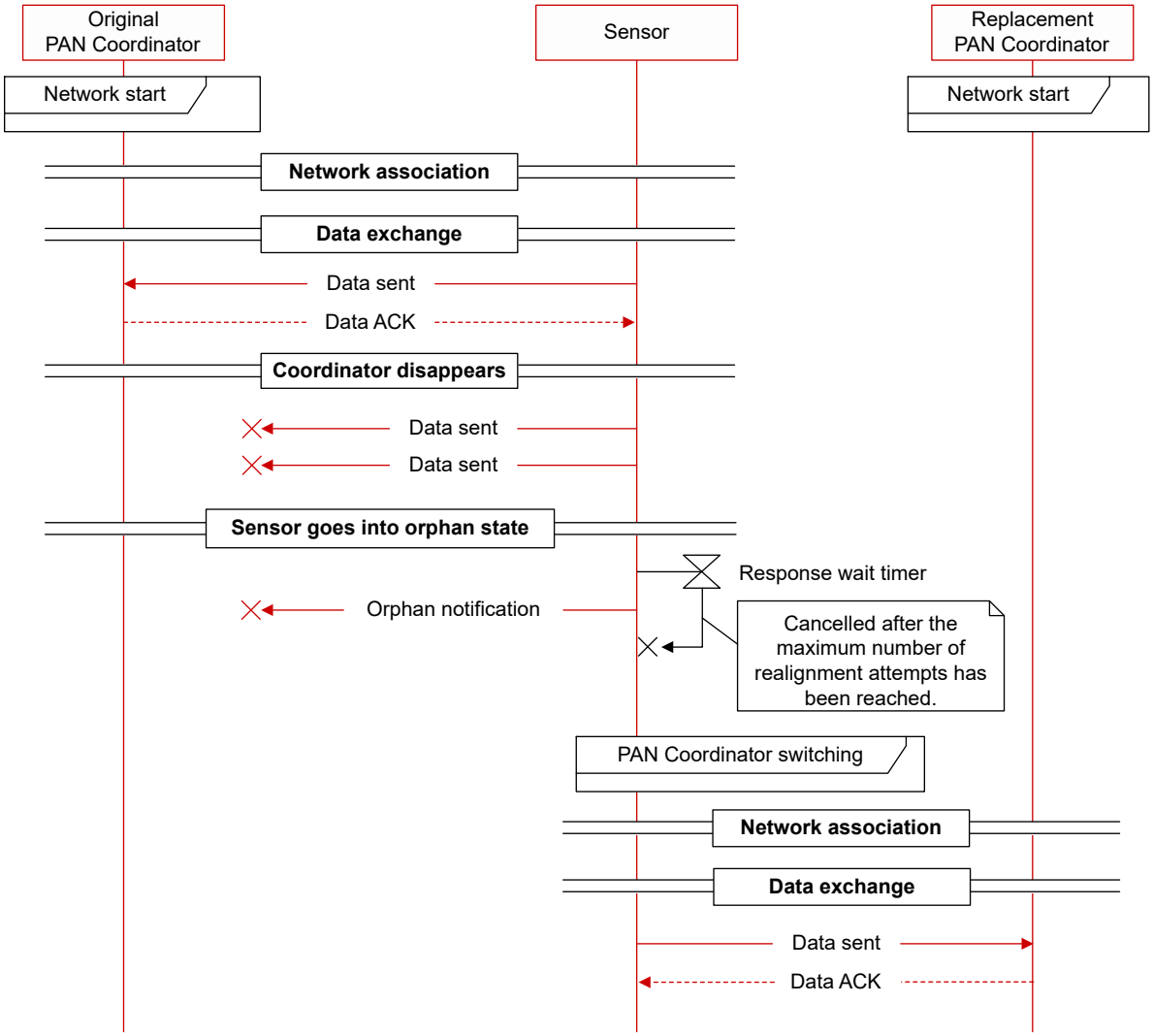


Figure 5-1. PAN Coordinator Switching When the Sensor Device Goes Into an Orphan State

This first switching scenario is rather simple, as the most basic implementation only requires a modification to the application layer of the sensor example. Taking this into account, the aspects that must be considered when implementing the coordinator switching procedure at the application layer of the sensor device are:

- The condition under which the procedure is triggered: How much time should the sensor spend trying to rejoin its original PAN coordinator.
- The implementation of a network disassociation process which does not rely on an acknowledge from the PAN coordinator. In the existing implementation of the disassociation process, either the sensor or the coordinator initiates the process, and the process must be acknowledged by the other side.
- The periodical use of active or passive scans (depending on the TI 15.4 mode of operation) to search for a replacement PAN coordinator.

5.2 PAN Coordinator Switching Due to a Command Coming From the Central Gateway

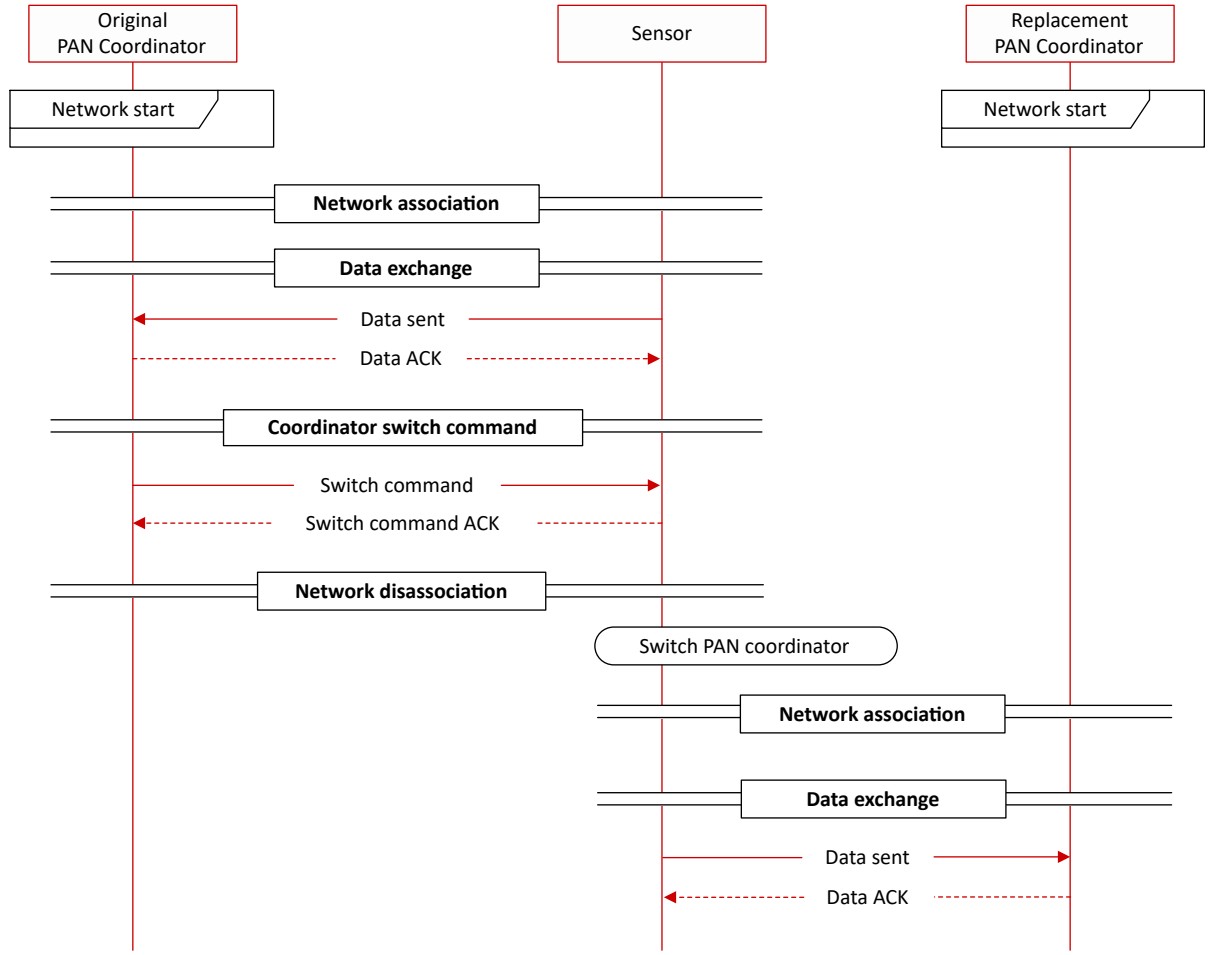
This scenario is more complex and it is particularly pertinent to this application note because it directly relates to most of the network features described in the first sections of this document. Features such as node balancing, extended network coverage, and increased node count, require an application layer capable of performing coordinator switching triggered by a command sent by the central gateway.

Having said this, the approach is similar to the one previously described for coordinator switching due to a sync loss, except that in this case the application layer of the collector also needs to be modified.

Copyright © 2021 Texas Instruments Incorporated

The following steps assume that an existing network has at least two local gateways and some associated sensors. For practical purposes, the first local gateway will be referred as LG1 and the second local gateway will be referred as LG2.

1. The network is formed and the central gateway coordinates LG1 and LG2. The local networks managed by LG1 and LG2 use different PAN IDs and short address ranges, and each PAN coordinator has some associated sensor devices.
2. The central gateway sends a command to LG1 and LG2 notifying both local gateways that a particular sensor device in the network of LG1 should switch PAN coordinator and connect to the local network set up by LG2. That is, LG2 will replace LG1 as PAN coordinator for that particular sensor.
3. Both local gateways acknowledge the command sent by the central gateway.
4. LG2 opens its local TI 15.4 network to allow new devices to join.
5. LG1 notifies the sensor device of the switch request by sending a command that may include (depending on the application) the following things:
   - The PAN ID of LG2
   - The channel currently being used by LG2
6. The sensor receives the switch request, acknowledges it, and sends a disassociation request to LG1.
7. Once the disassociation request succeeds, the sensor performs an active scan (Non-Beacon mode) or a passive scan (Beacon mode) to detect other PAN coordinators.
8. If the scan succeeds, the sensor verifies that the coordinator data sent from LG1 corresponds to the data coming from LG2, and if it does, it starts the association procedure.
9. Once the association procedure succeeds, LG2 notifies the central gateway that the sensor is now part of its local network.



NOTE: This sequence does not cover communication between the local gateways and the central gateway.

Figure 5-2. PAN Coordinator Switching Due to a Command From the Central Gateway

Copyright © 2021 Texas Instruments Incorporated

6 Basic Implementation of PAN Coordinator Switching

The following code snippets are meant to demonstrate how PAN coordinator switching can be implemented in a very simple way by only slightly modifying the application layer of the existing sensor and collector examples.

To visualize how the PAN coordinator switching process takes place, the following test setup was used:

- One CC13x2 LaunchPad™ development kit used as sensor device. This device will switch between PAN coordinators.
- Two CC13x2 LaunchPad kits used as collector devices.
- Two CC13x0 LaunchPad kits used as packet sniffers.
- Smart RF Packet Sniffer and Wireshark for network traffic visualization.

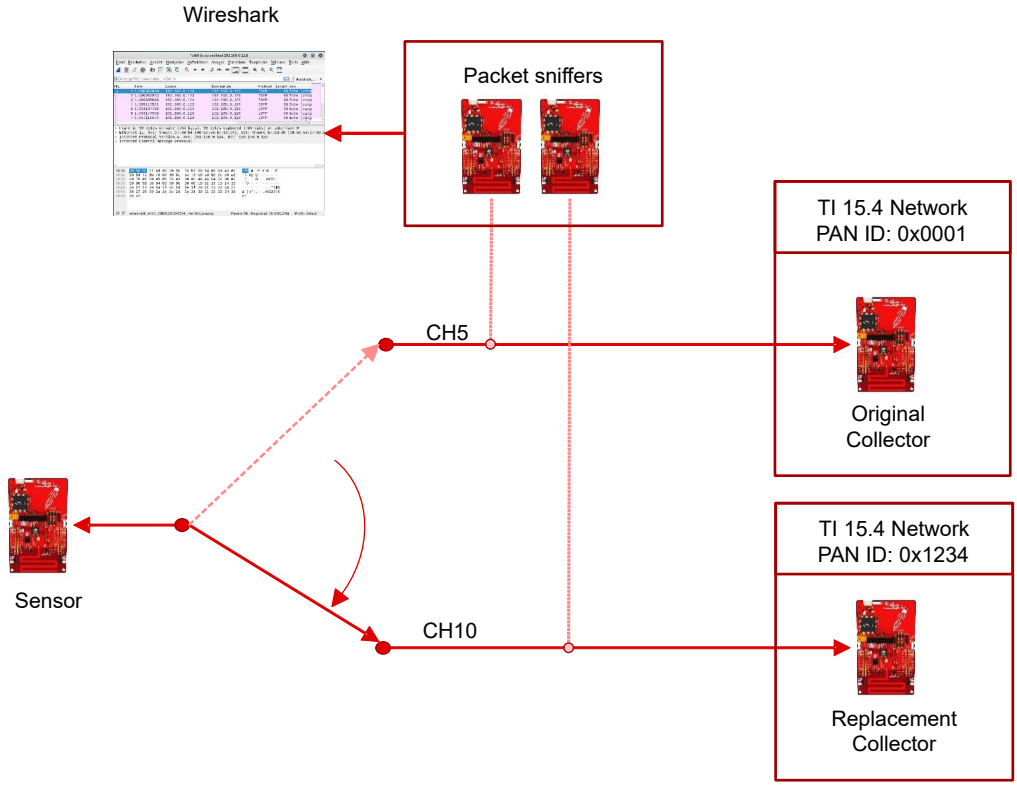


Figure 6-1. Test Setup to Validate PAN Coordinator Switching

For this particular test setup, CC1352 LaunchPad kits were used for the collectors and sensor, two CC1310 LaunchPad kits were used for the packet sniffers, and Smart RF Sniffer Agent v1.9.0 Alpha and Wireshark v3.0.14 were used to keep track of the networks' traffic.

See the SmartRF Packet Sniffer User's Guide to find more information about how to configure and use LaunchPad kits as packet sniffers.

6.1 PAN Coordinator Switching Due to Sync Loss

For this particular scenario, it is enough to modify the application layer of the sensor device for a basic implementation of PAN coordinator switching.

1. Import the following example projects into the IDE workspace:
   a. TI 15.4-Stack Sensor project
   b. TI 15.4-Stack Collector project (used as original collector)
   c. TI 15.4-Stack Collector project (used as replacement collector)

Copyright © 2021 Texas Instruments Incorporated

Note

To differentiate the collectors when using the Common User Interface (CUI), modify the SFF_MENU_TITLE macro in *csf.c*, so that each project is properly identified. It may also be necessary to rename the projects.

2. Configure the sensor and collector projects so that they operate in the same mode, and with the same PHY. Also consider the following:
   - The PAN ID field for the sensor should be set as 0xFFFF.
   - Each collector device should use a different PAN ID.
   - Each collector device should use a different channel mask.
   - The sensor device should use a channel mask that covers the channels at which both collectors operate.
   - Use different short addresses for each coordinator. The short address can be modified in the file *advanced_config.h*.
   - Choose an appropriate value for the Orphan Back-off Interval in the Sysconfig configuration for the sensor (for example 5000 ms).

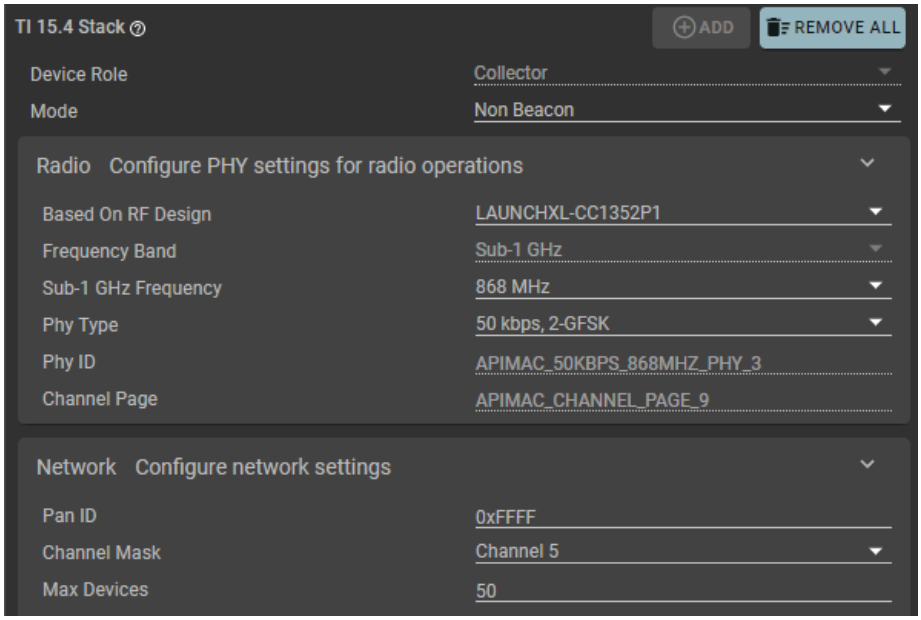


Figure 6-2. TI 15.4-Stack SysConfig Configuration for First Collector Device

Copyright © 2021 Texas Instruments Incorporated

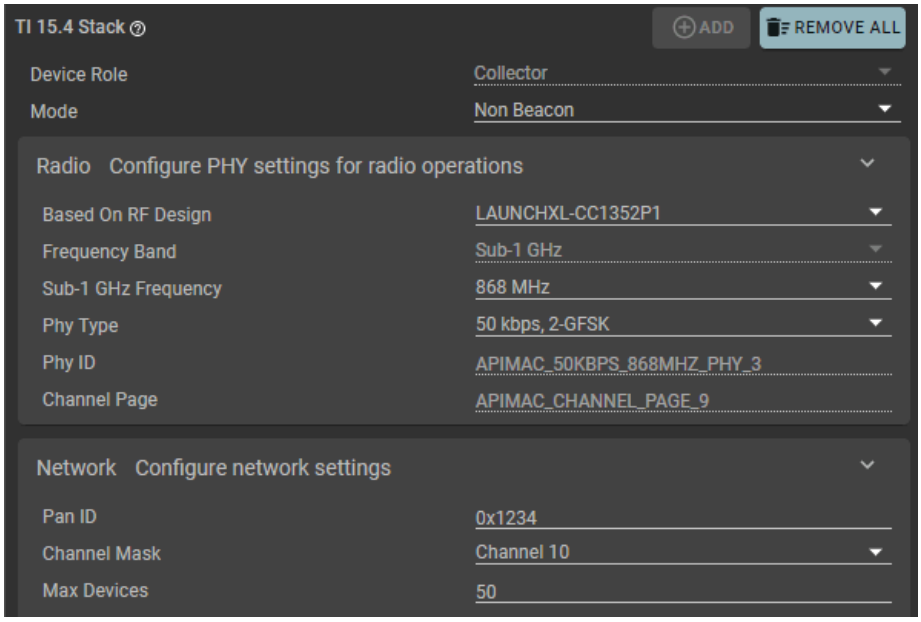


Figure 6-3. TI 15.4-Stack SysConfig Configuration for Second Collector Device

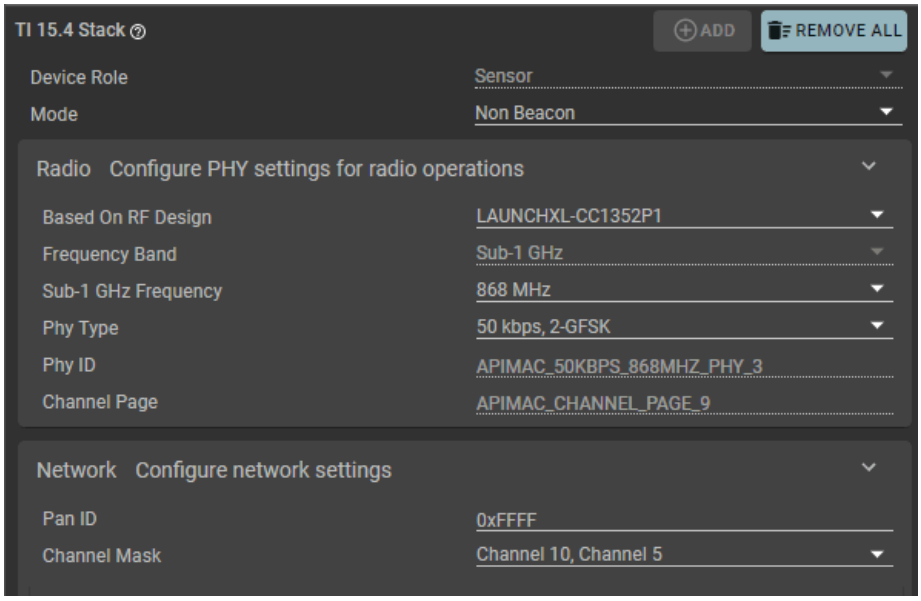


Figure 6-4. TI 15.4-Stack SysConfig Configuration for Sensor Device

3. Build the collector projects and flash the LaunchPad kits.
4. Open the file *jdllc.c* in the sensor project, and add a local function prototype of the function that will be used to perform the network disassociation.

```
static void abandonNetwork(void);
```

5. In *jdllc.c* in the sensor project, add the following lines in the section for constants and definitions:

```
/* Maximum score used to evaluate a channel when looking for second coordinator */
#define JDLLC_MAX_ENERGY                    255
#define JDDLC_MAX_RECONNECTION_ATTEMPTS       5
```

The first macro definition is used as a starting point for the energy detect scans that the sensor device must execute. And the second macro definition indicates the number of times the sensor will attempt to reconnect to its original PAN coordinator before switching to a different PAN coordinator.

Copyright © 2021 Texas Instruments Incorporated

6. In *jdllc.c*, define the local function used to perform the network disassociation.

```
void abandonNetwork(void)
{
    /* stop polling */
    if(!CONFIG_RX_ON_IDLE)
    {
        Ssf_setPollClock(0);
    }

    /* enable looking for new parent */
    parentFound = false;

    /* set devInfoBlock back to defaults */
    devInfoBlock.panID = CONFIG_PAN_ID;
    devInfoBlock.channel = JDLLC_INVALID_CHANNEL;

    if(!CONFIG_FH_ENABLE)
    {
      devInfoBlock.coordShortAddr = 0xFFFF;
    }
    else
    {
      devInfoBlock.coordShortAddr = FH_COORD_SHORT_ADDR;
    }

#ifdef FEATURE_MAC_SECURITY
    ApiMac_secDeleteDevice(&devInfoBlock.devExtAddr);
#endif

    /* Reset devInfoBlock to default values */
    memset(&devInfoBlock.coordExtAddr[0], 0x00, APIMAC_SADDR_EXT_LEN);
    devInfoBlock.devShortAddr = 0xFFFF;
    memset(&devInfoBlock.devExtAddr[0], 0x00, APIMAC_SADDR_EXT_LEN);
    devInfoBlock.beaconOrder = CONFIG_MAC_BEACON_ORDER;
    devInfoBlock.superframeOrder = CONFIG_MAC_SUPERFRAME_ORDER;
    devInfoBlock.currentDevState = Jdllc_deviceStates_scanActive;
    devInfoBlock.prevDevState = Jdllc_deviceStates_scanActive;
    devInfoBlock.dataFailures = 0;
    devInfoBlock.pollInterval = CONFIG_POLLING_INTERVAL;

    /* Stop the reporting timer */
    Ssf_setReadingClock(0);
    Ssf_clearNetworkInfo();
}
```

This function could be described as a “hard” disassociation that does not require an acknowledge from the PAN coordinator. This type of disassociation is valid in this case, because the PAN coordinator is no longer reachable and the sensor considers that it no longer exists.

7. In *jdllc.c*, modify the process scan confirm callback to support coordinator switching.

For this particular case, the coordinator switching takes place after the sensor has repeatedly attempted to reconnect with its PAN coordinator using orphan scans. Once the number of failed orphan scans reaches the limit defined by the JDDLC_MAX_RECONNECTION_ATTEMPTS macro, the sensor makes a call to *abandonNetwork()*and triggers an energy detect scan.

The energy detect scan is used to perform an energy measurement on all available channels. If there is another PAN coordinator operating in one of those channels, the sensor device will tune to that channel and trigger either an active scan (Non-Beacon mode) or a passive scan (Beacon mode). Once this occurs, the sensor joins the new PAN coordinator.

```
static void scanCnfCb(ApiMac_mlmeScanCnf_t *pData)
{
    if(pData->status == ApiMac_status_success)
    {
        if(pData->scanType == ApiMac_scantype_active)
        {
            // Existing code in the example
        }

        // Existing code in the example

        else if(pData->scanType == ApiMac_scantype_energyDetect)
```


Copyright © 2021 Texas Instruments Incorporated

```
        {
            uint8_t *pResults;
            uint8_t chan;
            uint8_t currEnergy;
            uint8_t lastEnergy = JDLLC_MAX_ENERGY;
            uint8_t coordStartChan = 0;

            pResults = pData->result.pEnergyDetect;

            /* Get channel with best energy level */

            for(chan = 0; chan < APIMAC_154G_MAX_NUM_CHANNEL; chan++)
            {
              /*
               * A measured RSSI value is converted to a score between 0 and 255,
               * using the minimum measured RSSI (-90 dBm) and saturation
               * energy (-5 dBm) values and the formula:
               * ED = (255 * (RSSI + 90))/85. A lower energy detect
               * measurement represents a more suitable channel.
               */
                if(JDLLC_IS_CHANNEL_MASK_SET(defaultChannelMask, chan))
                {
                    currEnergy = pResults[chan];
                    if(currEnergy < lastEnergy)
                    {
                        coordStartChan = chan;
                        lastEnergy = currEnergy;
                    }
                }
            }

            ApiMac_mlmeSetReqUint8(ApiMac_attribute_logicalChannel,
                                   coordStartChan);
            panIdMatch = true;

            if(CONFIG_MAC_BEACON_ORDER == JDLLC_BEACON_ORDER_NON_BEACON)
            {
                /* non beacon network */
                sendScanReq(ApiMac_scantype_active);
            }
            else if((CONFIG_MAC_BEACON_ORDER > 0) &&
                    (CONFIG_MAC_BEACON_ORDER < JDLLC_BEACON_ORDER_NON_BEACON))
            {
                /* beacon network */
                updateState(Jdllc_states_joining);
                sendScanReq(ApiMac_scantype_passive);
            }
    }
    else
    {
        // Existing code in the example

        if(((pData->scanType == ApiMac_scantype_orphan) && (pData->status
                        == ApiMac_status_noBeacon)))
        {
            static uint8_t orphanCounter;

            if (orphanCounter <= JDDLC_MAX_RECONNECTION_ATTEMPTS)
            {
                devInfoBlock.prevDevState = devInfoBlock.currentDevState;
                Ssf_setScanBackoffClock(CONFIG_ORPHAN_BACKOFF_INTERVAL);
                orphanCounter++;
            }
            else
            {
                /* Give up trying to reconnect to the same coordinator
                 * and abandon network */

                abandonNetwork();
                panIdMatch = false;

                /* Perform an ED scan to check available channels */
                sendScanReq(ApiMac_scantype_energyDetect);
                orphanCounter = 0;
            }
        }
    }
    // Existing code in the example
}
```


Copyright © 2021 Texas Instruments Incorporated

8. Build the project, make sure that there are no build errors, and flash the modified sensor example.
9. Verify that the PAN coordinator switching is working as expected by following these steps:
   a. All three CC1352 LaunchPad kits should be flashed. Follow the instructions in the README file of the projects to display the CUI (Common User Interface) of the examples.
   b. Configure and start the two packets sniffers. Each packet sniffer should track a different PAN coordinator.
   c. Open Wireshark and start the capture.
   d. Start the first collector and set its join permit to ON.
   e. Start the sensor, and let it join the network of the first collector.
   f. Packet exchange between the sensor and the collector should be visible in the Wireshark GUI (see Figure 6-6).
   g. Start the second collector and set its join permit to ON.
   h. Remove power from the first collector to cause a sync loss. The sensor will continue to send some packets before going into an orphan state, and then it will start sending orphan notifications.
   i. After a given number of orphan notifications, the sensor will perform an energy detect scan (not visible in Wireshark), and when detecting the second collector, it will execute an active or passive scan (depending on the mode) to start a network association with the second collector.
   j. The PAN coordinator switching concludes with an acknowledge to the association response sent from the second PAN coordinator, and communication between the sensor and its new PAN coordinator resumes (see Figure 6-7).

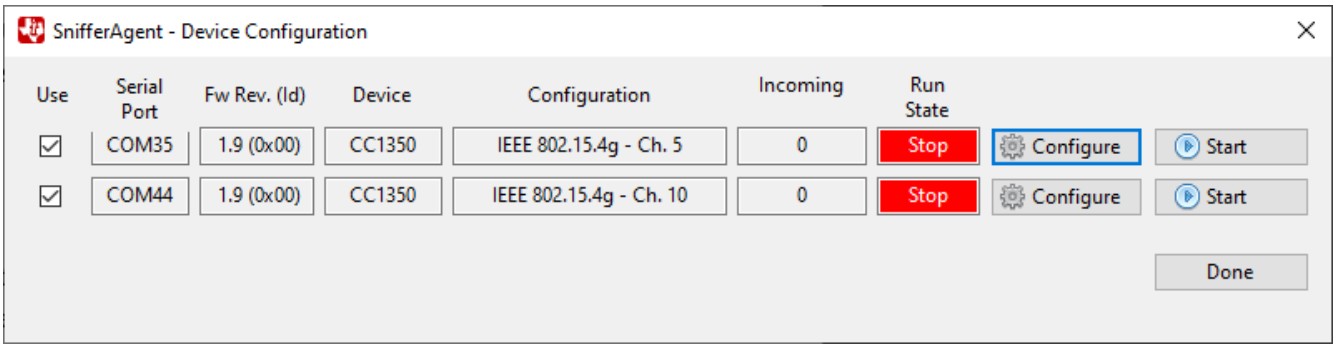


Figure 6-5. Device Configuration in Smart RF Sniffer Agent for Both Packet Sniffers

No.	Time	Source	Destination	Protocol	Frame Control Field	Info
1	0.0000…		0xffff	TI 802.15.4GE	0x0803	Beacon request
2	0.0147…	0xaabb		TI 802.15.4GE	0x8000	Beacon, Src: 0xaabb
3	14.847…		0xffff	TI 802.15.4GE	0x0803	Beacon request
4	14.859…	0xaacc		TI 802.15.4GE	0x8000	Beacon, Src: 0xaacc
5	1.2901…	00:12:4b:1…	0xaabb	TI 802.15.4GE	0xc823	Association request
6	1.2973…			TI 802.15.4GE	0x0002	Ack
7	2.2364…	00:12:4b:1…	0xaabb	TI 802.15.4GE	0xc863	Data request
8	2.2431…			TI 802.15.4GE	0x0012	Ack
9	2.2554…	00:12:4b:1…	00:12:4b:1…	TI 802.15.4GE	0xcc63	Association response, PAN: 0x0001 Addr: 0x0001
10	2.2635…			TI 802.15.4GE	0x0002	Ack
11	2.2908…	0x0001	0xaabb	TI 802.15.4GE	0x8863	Data request
12	2.2966…			TI 802.15.4GE	0x0002	Ack
13	3.2725…	0x0001	0xaabb	TI 802.15.4GE	0x9869	Data, Dst: 0xaabb, Src: 0x0001

Figure 6-6. Packet Exchange Showing Association Request Between Sensor and First Collector

Copyright © 2021 Texas Instruments Incorporated

No.	Time	Source	Destination	Protocol	Frame Control Field	Info
42	10.269908	0x0001	0xaabb	TI 802.15.4GE	0x8863	Data request
43	10.290450	0x0001	0xaabb	TI 802.15.4GE	0x8863	Data request
44	10.308672	0x0001	0xaabb	TI 802.15.4GE	0x8863	Data request
45	10.328504	00:12:4b:1…	0xffff	TI 802.15.4GE	0xc843	Orphan notification
46	15.963575	00:12:4b:1…	0xffff	TI 802.15.4GE	0xc843	Orphan notification
47	17.198376	00:12:4b:1…	0xffff	TI 802.15.4GE	0xc843	Orphan notification
48	22.834622	00:12:4b:1…	0xffff	TI 802.15.4GE	0xc843	Orphan notification
49	24.069379	00:12:4b:1…	0xffff	TI 802.15.4GE	0xc843	Orphan notification
50	29.706799	00:12:4b:1…	0xffff	TI 802.15.4GE	0xc843	Orphan notification
51	30.941576	00:12:4b:1…	0xffff	TI 802.15.4GE	0xc843	Orphan notification
52	36.583641	00:12:4b:1…	0xffff	TI 802.15.4GE	0xc843	Orphan notification
53	34.214628		0xffff	TI 802.15.4GE	0x0803	Beacon request
54	39.566263		0xffff	TI 802.15.4GE	0x0803	Beacon request
55	39.584550	0xaacc		TI 802.15.4GE	0x8000	Beacon, Src: 0xaacc
56	40.217372	00:12:4b:1…	0xaacc	TI 802.15.4GE	0xc823	Association request
57	40.224560			TI 802.15.4GE	0x0002	Ack
58	41.160222	00:12:4b:1…	0xaacc	TI 802.15.4GE	0xc863	Data request
59	41.166929			TI 802.15.4GE	0x0012	Ack
60	41.177501	00:12:4b:1…	00:12:1…	TI 802.15.4GE	0xcc63	Association response, PAN: 0x1234 Addr: 0x0001
61	41.185618			TI 802.15.4GE	0x0002	Ack
62	41.199925	0x0001	0xaacc	TI 802.15.4GE	0x8863	Data request
63	41.205647			TI 802.15.4GE	0x0002	Ack
64	42.372459	0x0001	0xaacc	TI 802.15.4GE	0x9869	Data, Dst: 0xaacc, Src: 0x0001
65	42.392286			TI 802.15.4GE	0x0002	Ack
66	44.367973	0x0001	0xaacc	TI 802.15.4GE	0x9869	Data, Dst: 0xaacc, Src: 0x0001

Figure 6-7. Packet Exchange Showing the Sensor Going Into an Orphan State and Switching PAN Coordinators

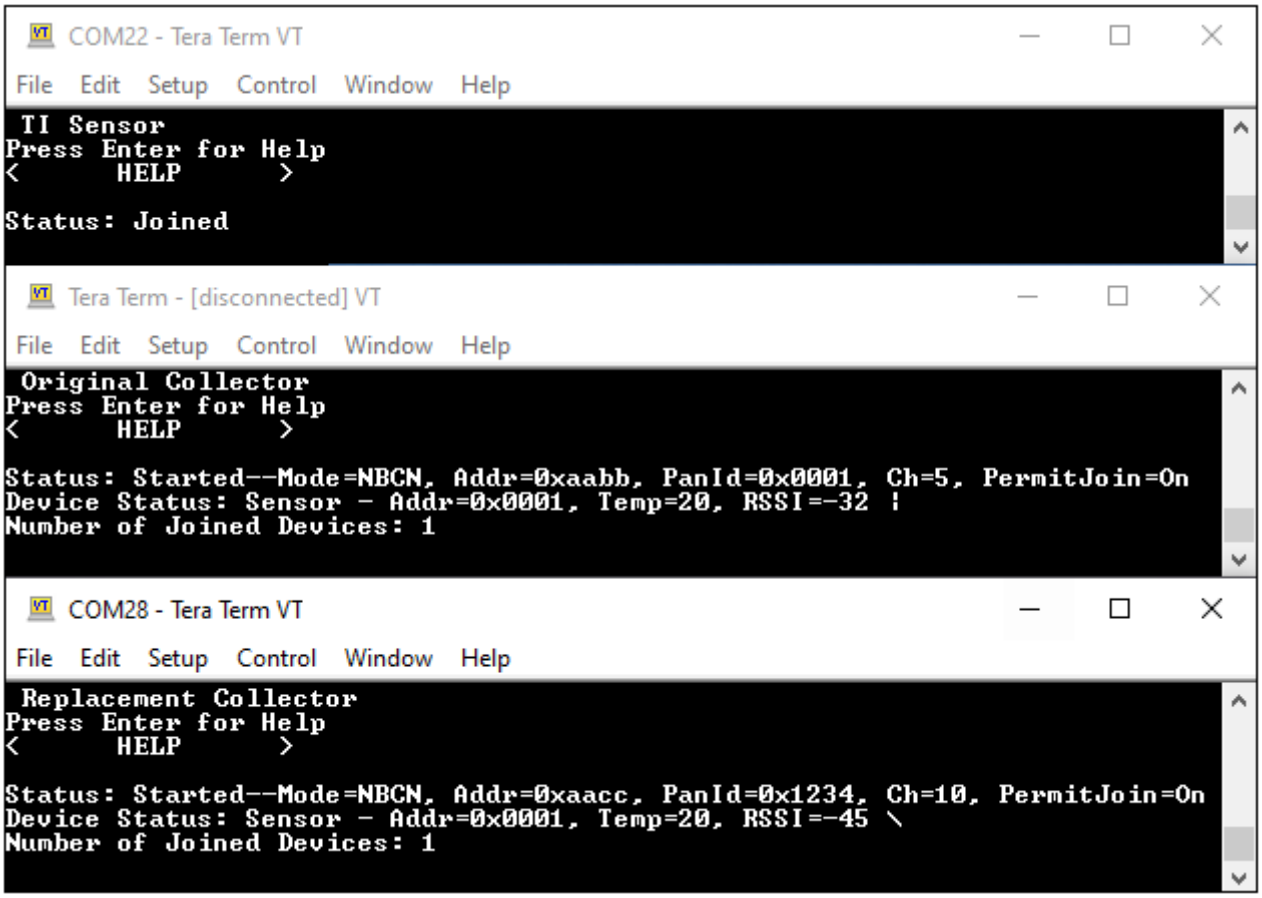


Figure 6-8. Common User Interface for the Sensor and Two Collectors After the First Collector is Disconnected

Copyright © 2021 Texas Instruments Incorporated

6.2 PAN Coordinator Switching Due to a Command Coming From the Central Gateway

This coordinator switching scenario requires a modification to both the sensor and the collector examples. Even though some of the steps are common to both the sensor and the collector, a separate list of steps is provided for each example project.

To keep the implementation simple, instead of relying of a message coming from a central gateway, the CUI of the collector example is modified to add a new type of user action, and it is the user the one that initiates the coordinator switching process by selecting the correct option in the CUI. By picking this option, the PAN coordinator sends a new type of message to the sensor device which process and uses the information in it to switch to a different PAN coordinator.

First steps are common for both the sensor and the collector examples:

1. Import the following example projects into the IDE workspace:
   a. TI 15.4-Stack Sensor project
   b. TI 15.4-Stack Collector project (used as original collector)
   c. TI 15.4-Stack Collector project (used as replacement collector)

   Note

   To differentiate the collectors when using the Common User Interface (CUI), modify the SFF_MENU_TITLE macro in *csf.c*, so that each project is properly identified.
2. Configure the sensor and collector projects so that they operate in the same mode, and with the same PHY. Also consider the following:
   - The PAN ID field for the sensor should be set as 0xFFFF.
   - Each collector device should use a different PAN ID.
   - Each collector device should use a different channel mask.
   - The sensor device should use a channel mask that covers the channels at which both collectors operate.
   - Use different short addresses for each coordinator. The short address can be modified in the file *advanced_config.h*.

   The following steps are the changes to the sensor project:
3. Open the *smsgs.h* file, and add two new message types to the *Constants and definitions* section. One for the request (sent by the coordinator) and one for the reply (sent by the sensor).

```
/******************************************************************************
 Constants and definitions
 *****************************************************************************/
/*! Switch Collector Request message length (over-the-air length) */
#define SMSGS_SWITCH_COLLECTOR_REQUEST_MSG_LEN 3
/*! Switch Collector Response message length (over-the-air length) */
#define SMSGS_SWITCH_COLLECTOR_RESPONSE_MSG_LEN 2

/*!
 Message IDs for Sensor data messages.  When sent over-the-air in a message,
 this field is one byte.
 */
 typedef enum
 {
    // Existing message types
    /* Switch collector message, sent from the collector to the sensor */
    Smsgs_cmdIds_switchCollectorReq = 18,
    /* Switch collector response msg, sent from the sensor to the collector */
    Smsgs_cmdIds_switchCollectorRsp = 19

 } Smsgs_cmdIds_t;

/******************************************************************************
 Structures - Building blocks for the over-the-air sensor messages
 *****************************************************************************/
// Existing structures in the example

/*!
 Switch collector Request message: sent from controller to the sensor.
 */
typedef struct _Smsgs_switchcollectorreqmsg_t
{
```


Copyright © 2021 Texas Instruments Incorporated

```
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
    /*! PAN ID of replacement coordinator - 2 bytes */
    uint16_t switchPanId;
} Smsgs_switchCollectorReqMsg_t;

/*!
 Switch collector Response message: sent from the sensor to the collector
 in response to the Switch collector Request message.
 */
typedef struct _Smsgs_switchcollectorrspmsg_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
    /*! Acknowledge from the sensor - 1 byte */
    uint8_t sensorAck;
} Smsgs_switchCollectorRspMsg_t;
```

4. Open the file *sensor.c* and add a function prototype of the function that will be used to trigger the coordinator switching.

```
static void processCoordinatorSwitchMsg(ApiMac_mcpsDataInd_t *pDataInd);
```

5. In *sensor.c*, add the following global variable:

```
extern bool switchCoordinatorFlag;
```

6. In *sensor.c*, define the local function used to trigger the coordinator switch:

```
static void processCoordinatorSwitchMsg(ApiMac_mcpsDataInd_t *pDataInd)
{
    uint8_t *pBuf = pDataInd->msdu.p;

    switchCoordinatorFlag = true;
    newCoordinatorPanId = Util_buildUint16(pBuf[1], pBuf[2]);
    Jdllc_sendDisassociationRequest();
}
```

7. In *sensor.c*, modify the data indication callback to consider the new type of message previously defined:

```
static void dataIndCB(ApiMac_mcpsDataInd_t *pDataInd)
{
    // Existing code in the example

        switch(cmdId)
        {
            // Existing code in the example

            case Smsgs_cmdIds_switchCollectorReq:

                /* Make sure the message is the correct size */
                if(pDataInd->msdu.len == SMSGS_SWITCH_COLLECTOR_REQUEST_MSG_LEN)
                {
                    /* only send data if sensor is in the network */
                    if ((Jdllc_getProvState() == Jdllc_states_joined) ||
                            (Jdllc_getProvState() == Jdllc_states_rejoined))
                    {
                        /* send the response message directly */
                        cmdBytes[0] = (uint8_t) Smsgs_cmdIds_switchCollectorRsp;
                        cmdBytes[1] = true;

                        Sensor_sendMsg(Smsgs_cmdIds_switchCollectorRsp,
                            &pDataInd->srcAddr, true,
                            SMSGS_SWITCH_COLLECTOR_RESPONSE_MSG_LEN,
                            cmdBytes);

                        processCoordinatorSwitchMsg(pDataInd);
                    }
                }
                break;
```

8. Open the file *jdllc.c* and add the following global variable:

```
bool switchCoordinatorFlag = false;
uint16_t replacementCoordinatorPanId = 0xFFFF;
```


Copyright © 2021 Texas Instruments Incorporated

9. In *jdllc.c*, modify the disassociate confirm callback to check *switchCoordinatorFlag* and act accordingly.

```
static void disassoCnfCb(ApiMac_mlmeDisassociateCnf_t *pData)
{
    // Existing code in the example

    if(switchCoordinatorFlag)
    {
        /* We need to return this flag to false, since we'll be
         * connecting to a new PAN ID.
         */
        panIdMatch = false;

        abandonNetwork();

        if(CONFIG_MAC_BEACON_ORDER == JDLLC_BEACON_ORDER_NON_BEACON)
        {
            /* non beacon network */
            sendScanReq(ApiMac_scantype_active);
        }
        else if((CONFIG_MAC_BEACON_ORDER > 0) &&
                (CONFIG_MAC_BEACON_ORDER < JDLLC_BEACON_ORDER_NON_BEACON))
        {
            /* beacon network */
            updateState(Jdllc_states_joining);
            sendScanReq(ApiMac_scantype_passive);
        }
        switchCoordinatorFlag = false;
    }
```

10. In *jdllc.c*, modify the beacon notification callback so that the sensor only connects to the appropriate the PAN coordinator.

```
static void beaconNotifyIndCb(ApiMac_mlmeBeaconNotifyInd_t *pData)
{
    /* check beacon type */
    if(pData->beaconType == ApiMac_beaconType_normal)
    {
        if(parentFound == false)
        {
            /* Check if association bit permit is set */
            if(APIMAC_SFS_ASSOCIATION_PERMIT(pData->panDesc.superframeSpec))
            {
                /* Check for beacon order match */
                if(checkBeaconOrder(pData->panDesc.superframeSpec) == true)
                {
                    if(devInfoBlock.panID == JDLLC_INVALID_PAN)
                    {
                        /* Device can join any network, associate with
                         * first coordinator from which beacon is received */
                        devInfoBlock.panID = pData->panDesc.coordPanId;
                        panIdMatch = true;
                        devInfoBlock.channel = pData->panDesc.logicalChannel;
                        devInfoBlock.coordShortAddr = pData->panDesc
                                        .coordAddress.addr.shortAddr;
                    }
                    /* Check the incoming PAN ID to see if it's a valid coordinator */
                    else if (devInfoBlock.panID == pData->panDesc.coordPanId)
                    {
                        panIdMatch = true;
                        numSyncLoss = 0;
                        ApiMac_mlmeSetReqBool(ApiMac_attribute_autoRequest, true);
                        devInfoBlock.channel = pData->panDesc.logicalChannel;
                        devInfoBlock.coordShortAddr = pData->panDesc
                                        .coordAddress.addr.shortAddr;
                    }
                    if(APIMAC_SFS_BEACON_ORDER( pData->panDesc.superframeSpec)
                                    != JDLLC_BEACON_ORDER_NON_BEACON)
                    {
                        devInfoBlock.beaconOrder = APIMAC_SFS_BEACON_ORDER(
                                        pData->panDesc.superframeSpec);
                        devInfoBlock.superframeOrder =
                            APIMAC_SFS_SUPERFRAME_ORDER(pData->panDesc.superframeSpec);
                    }
                    if((devInfoBlock.beaconOrder == JDLLC_BEACON_ORDER_NON_BEACON)
                        && (panIdMatch == true))
                    {
                        parentFound = true;
```


Copyright © 2021 Texas Instruments Incorporated

```
                        Ssf_stopScanBackoffClock();
                    }
                }
            }
        }
    }
    // Existing code
}
```

The following steps are the changes to the collector projects:

11. Repeat step (3) for the collector projects.
12. Open *collector.h* and declare a function to send the coordinator switch request message.

```
extern Collector_status_t Collector_sendCollectorSwitchRequest(
                ApiMac_sAddr_t *pDstAddr);
```

13. Open *collector.c* and add the definition for the previously declared function:

```
Collector_status_t Collector_sendCollectorSwitchRequest(ApiMac_sAddr_t *pDstAddr)
{
    Collector_status_t status = Collector_status_invalid_state;

    /* A hard-coded value is used for simplicity. In a real implementation
     * the gateway should inform the coordinator about the approved PAN ID
     * for the switch.
     * Use the appropriate value depending on which collector project is
     * being modified.
     */
    uint16_t newPanId = 0x1234;

    /* Are we in the right state? */
    if(cllcState >= Cllc_states_started)
    {
        Llc_deviceListItem_t item;

        /* Is the device a known device? */
        if(Csf_getDevice(pDstAddr, &item))
        {
            uint8_t msgBuf[SMSGS_SWITCH_COLLECTOR_REQUEST_MSG_LEN];
            uint8_t *pBuf = msgBuf;

            /* Build the message */
            *pBuf++ = (uint8_t) Smsgs_cmdIds_ switchCollectorReq;
            pBuf = Util_bufferUint16(pBuf, newPanId);

            sendMsg(Smsgs_cmdIds_switchCollectorReq, item.devInfo.shortAddress,
                    item.capInfo.rxOnWhenIdle,
                    SMSGS_SWITCH_COLLECTOR_REQUEST_MSG_LEN,
                    buffer);

            status = Collector_status_success;
        }
        else
        {
            status = Collector_status_deviceNotFound;
        }
    }
    return(status);
}
```

Note

If the suggested changes are implemented on both collector projects, make sure to set the appropriate value for the *newPanId* variable.

14. Open *collector.c* and add a local function prototype for the processing of the response coming from the sensor:

```
static void processCollectorSwitchResponse(ApiMac_mcpsDataInd_t *pDataInd);
```

15. In *collector.c*, also add the function necessary to process the response coming from the sensor.

```
static void processCollectorSwitchResponse(ApiMac_mcpsDataInd_t *pDataInd)
{
```


Copyright © 2021 Texas Instruments Incorporated

```
    /* Make sure the message is the correct size */
    if(pDataInd->msdu.len == SMSGS_SWITCH_COLLECTOR_RESPONSE_MSG_LEN)
    {
        // Add necessary code to send message to Gateway
    }
}
```

16. Open *csf.h*, and add a constant for the coordinator switch action that will trigger the switching process.

```
#define SENSOR_ACTION_SWITCH              4
```

17. In *csf.h*, declare a function that will be used to indicate that a sensor device has responded to the coordinator switch command.

```
extern void Csf_switchResponseReceived(ApiMac_sAddr_t *pSrcAddr, bool switchState);
```

18. In *csf.c*, declare a local function. One will be used as a response to an action coming from the CUI, and the other one will be used to update the CUI about the status of the request.

```
static void sensorCollectorSwitchAction(int32_t menuEntryInex);
static void sendCollectorSwitchAndUpdateUser(uint16_t shortAddr);
```

19. In *csf.c*, define the previously mentioned functions:

```
static void sensorCollectorSwitchAction(int32_t menuEntryInex)
{
    Csf_events |= COLLECTOR_SENSOR_ACTION_EVT;
    Csf_sensorAction = SENSOR_ACTION_SWITCH;

    // Wake up the application thread when it waits for clock event
    Semaphore_post(collectorSem);
}

void Csf_switchResponseReceived(ApiMac_sAddr_t *pSrcAddr, bool switchState)
{
    /* This is an optional function used to visually notify the user that the sensor
     * has acknowledged the coordinator switch. For example, toggle a LED.
     */
}

static void sendCollectorSwitchAndUpdateUser(uint16_t shortAddr)
{
   {
        ApiMac_sAddr_t recipientDev;
        recipientDev.addr.shortAddr = shortAddr;
        if(recipientDev.addr.shortAddr != CSF_INVALID_SHORT_ADDR)
        {
            recipientDev.addrMode = ApiMac_addrType_short;
            Collector_sendCollectorSwitchRequest(&recipientDev);
#ifndef CUI_DISABLE
            CUI_statusLinePrintf(csfCuiHndl, deviceStatusLine,
                                 "Switch Collector Request Sent - Addr=0x%04x",
                                 recipientDev.addr.shortAddr);
#endif /* CUI_DISABLE */
        }
   }
}
```

20. In *csf.c,* add a menu item to the CUI that will show inside the APP submenu. Do this by first modifying the number items from five to six, and by adding the item with its associated callback.

```
CUI_SUB_MENU(appSubMenu,"<     APP     >", 6, csfMainMenu)  // Value was originally 5
CUI_MENU_ITEM_ACTION("< SWITCH COLLECTOR >", sensorCollectorSwitchAction)
```

21. In *csf.c*, add the appropriate case to the *CSf_processEvents()* function:

```
void Csf_processEvents(void)
{
    /* Did a key press occur?
    if(Csf_events & CSF_KEY_EVENT)
    {
        // Existing code in the example
    }
    // Existing code in the example
```


Copyright © 2021 Texas Instruments Incorporated

```
    if(Csf_events & COLLECTOR_SENSOR_ACTION_EVT)
    {
        // Existing code in the example

        switch(Csf_sensorAction)
        {
            // Existing code in the example

            case SENSOR_ACTION_SWITCH:
            {
                /* send Collector Switch if CUI */
                sendCollectorSwitchAndUpdateUser(SelectedSensor);
                break;
            }
        // Existing code in the example
        }
}
```

22. Build the projects, make sure that there are no build errors, and flash the modified sensor and collector examples.
23. Start the collectors, set their join permit to ON, and start the sensor so that it connects to one of the collectors.
24. Set up the packet sniffers using the same steps as before, and verify the packet exchange in Wireshark.
25. Using the keyboard, navigate the CUI of the collector that is connected to sensor. First go to the APP submenu, and then go to the “SWITCH COLLECTOR” item.
26. Press the ENTER key to trigger the coordinator switching. Verify that the process is successful with Wireshark and with the CUIs of the examples.

No.	Time	Source	Destination	Protocol	Frame Control Field	Info
73	36.322…	0x0001	0xaabb	TI 802.15.4GE	0x9869	Data, Dst: 0xaabb, Src: 0x0001
74	36.342…			TI 802.15.4GE	0x0002	Ack
75	37.295…	0x0001	0xaabb	TI 802.15.4GE	0x8863	Data request
76	37.300…			TI 802.15.4GE	0x0012	Ack
77	37.310…	0xaabb	0x0001	TI 802.15.4GE	0x9879	Data, Dst: 0x0001, Src: 0xaabb
78	37.319…			TI 802.15.4GE	0x0002	Ack
79	37.329…	0x0001	0xaabb	TI 802.15.4GE	0x9869	Data, Dst: 0xaabb, Src: 0x0001
80	37.338…			TI 802.15.4GE	0x0002	Ack
81	37.349…	00:12:4b:1…	0xaabb	TI 802.15.4GE	0xc863	Disassociation notification
82	37.356…			TI 802.15.4GE	0x0002	Ack
83	37.368…		0xffff	TI 802.15.4GE	0x0803	Beacon request
84	37.383…	0xaabb		TI 802.15.4GE	0x8000	Beacon, Src: 0xaabb
85	52.216…		0xffff	TI 802.15.4GE	0x0803	Beacon request
86	52.227…	0xaacc		TI 802.15.4GE	0x8000	Beacon, Src: 0xaacc
87	52.866…	00:12:4b:1…	0xaacc	TI 802.15.4GE	0xc823	Association request
88	52.873…			TI 802.15.4GE	0x0002	Ack
89	53.807…	00:12:4b:1…	0xaacc	TI 802.15.4GE	0xc863	Data request
90	53.814…			TI 802.15.4GE	0x0012	Ack
91	53.823…	00:12:4b:1…	00:12:4b:1…	TI 802.15.4GE	0xcc63	Association response, PAN: 0x1234 Addr: 0x0001
92	53.832…			TI 802.15.4GE	0x0002	Ack
93	53.846…	0x0001	0xaacc	TI 802.15.4GE	0x8863	Data request
94	53.852…			TI 802.15.4GE	0x0002	Ack
95	54.686…	0x0001	0xaacc	TI 802.15.4GE	0x9869	Data, Dst: 0xaacc, Src: 0x0001
96	54.706…			TI 802.15.4GE	0x0002	Ack

Figure 6-9. Packet Exchange Showing the Sensor Switching PAN Coordinators

Messages 73 to 76 correspond to a normal message exchange between the sensor and its original PAN coordinator (with short address 0xaabb). Message 77 corresponds to the message sent by the original PAN coordinator indicating the sensor to switch to a different PAN coordinator. This message is sent when the user interacts with the CUI.

The message is acknowledged, and then a reply message is sent by the sensor (messages 78 and 79). This is followed by a disassociation notification sent by the sensor (message 81) and a beacon request to which both collectors reply. Notice that the original PAN coordinator also replies to the beacon request sent by the sensor.

Copyright © 2021 Texas Instruments Incorporated

The sensor then makes an association request to the replacement coordinator (with short address 0xaacc) and once this succeeds, communication resumes.

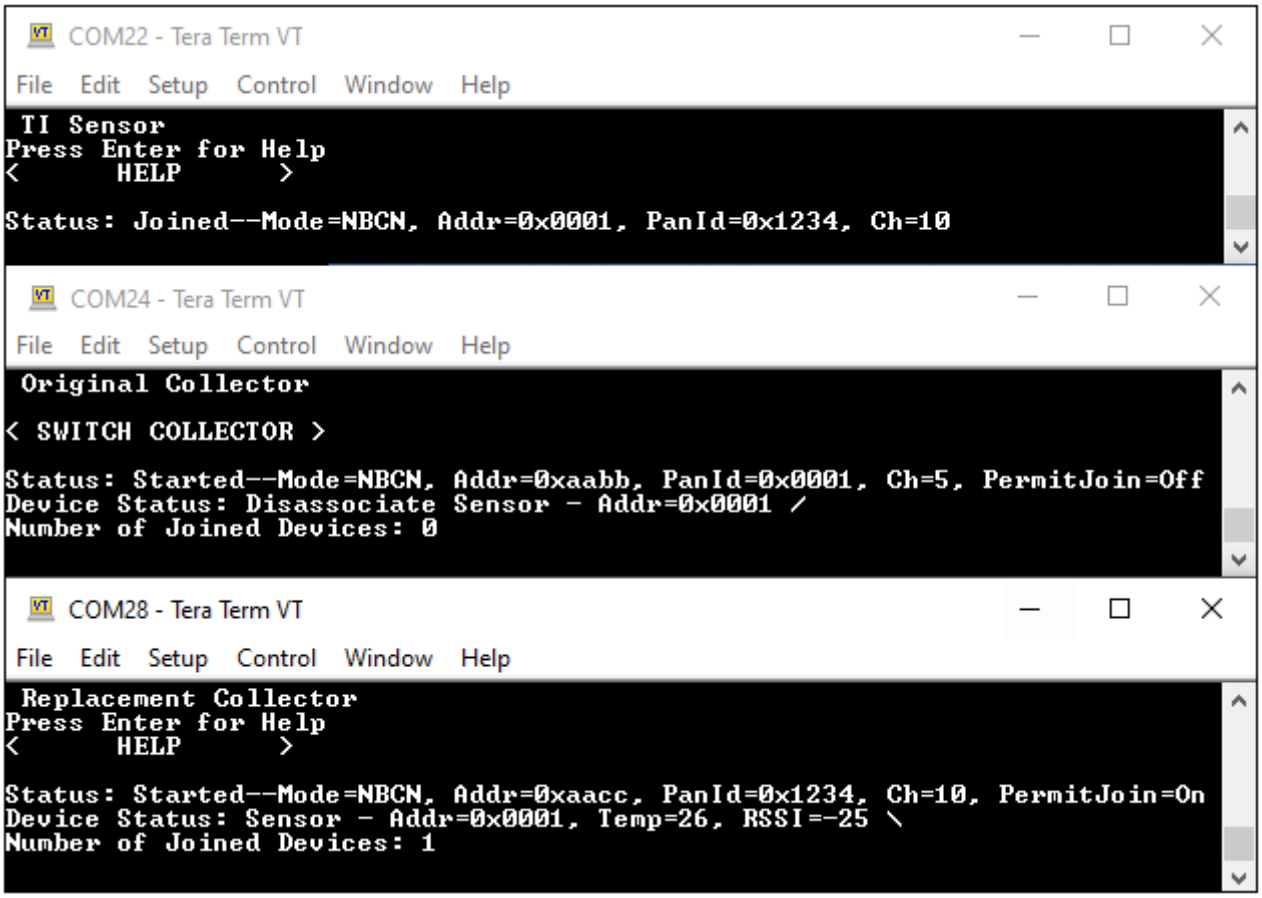


Figure 6-10. Common User Interface for the Sensor and Two Collectors After the First Collector Sends a Switching Coordinator Request

7 Summary

This application report discusses how having multiple gateways benefits the robustness, range and node balancing capabilities of a network that relies on the TI 15.4-Stack. The application report introduced the concept of a central gateway which provides synchronization services between local gateways that act as PAN coordinators for local TI 15.4 networks and communicate directly with sensor devices. Furthermore, the application report describes the steps to implement PAN coordinator switching which is a requirement for the enabling of multiple gateway support. Lastly, the application report discusses two use-case scenarios for PAN coordinator switching and presents some example code to demonstrate how multiple gateway support does not require modifications to the TI 15.4-Stack.

8 References

1. SimpleLink™ CC13xx/CC26xx SDK TI 15.4-Stack User's Guide
2. SimpleLink™ Linux SDK TI 15.4-Stack User's Guide
3. SmartRF Packet Sniffer 2 v1.9.0 User's Guide

Copyright © 2021 Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2022, Texas Instruments Incorporated