Mesh Provisioning

Hardware academy mesh workshop

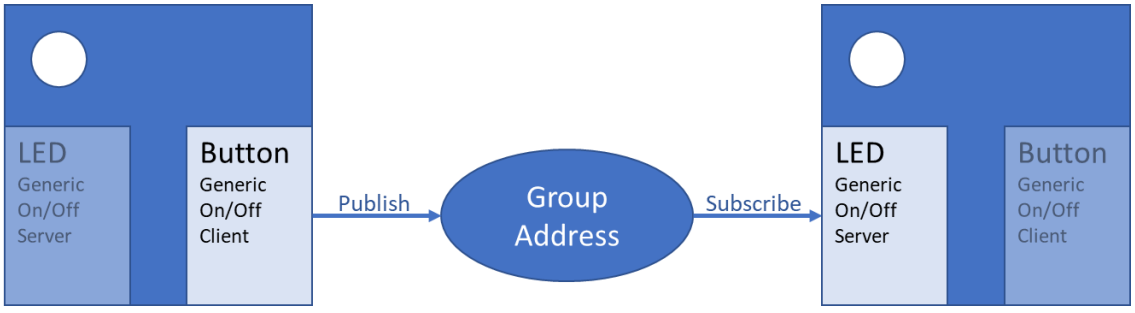
1 Scope

The scope of this hands-on is to get experience with the mesh provisioning app, available for iOS and Android.

The session is based on a mesh demo developed for the nRF52 development kits, that allows the button on the devkit to act as a generic on off client, and the LED on the devkit to act as a generic on off server.
Using the provisioning app on the phone, the user can provision the Devkits into a mesh, and configure the generic on off client and server.
The client can be configured to publish messages to an address every time the button is pushed, and the server can be configured to subscribe to addresses so that the LED is updated whenever someone publishes to the address.
By using group addresses multiple clients can publish and multiple servers can subscribe to the same address.

Since each participant only have a single devkit, everyone should form up in groups of 2-4 people for this exercise.

The first task is to set up a simple mesh with two devkits and configure the client in one to publish to the same group address as the server in the other. This will have the LED on one devkit be controlled by the button on the other:



The second task requires all the devkits to be provisioned into the mesh, and two separate group addresses to be used. The devkits will be split into two groups that can be individually controlled, and two of the devkits should be set to publish into these group addresses. This will allow two buttons to turn on and off either half of the LED's.

1.1.1 PLEASE NOTE:

- There is a troubleshooting section at the end of this document if you experience issues during the hands-on.

1.2 Prerequisites

1) Each participant will receive one nRF52840 DK.

2) Each group needs a single smart phone with the provisioning app installed, available for Android and iOS

1.3 Setting up the Mesh SDK

1) Download the modified mesh SDK from here:
https://github.com/NordicPlayground/nrf52-mesh-light-switch-client-server-proxy

   Download the nRF5 SDK v15.0.0 from here:
https://developer.nordicsemi.com/nRF5_SDK/nRF5_SDK_v15.x.x/

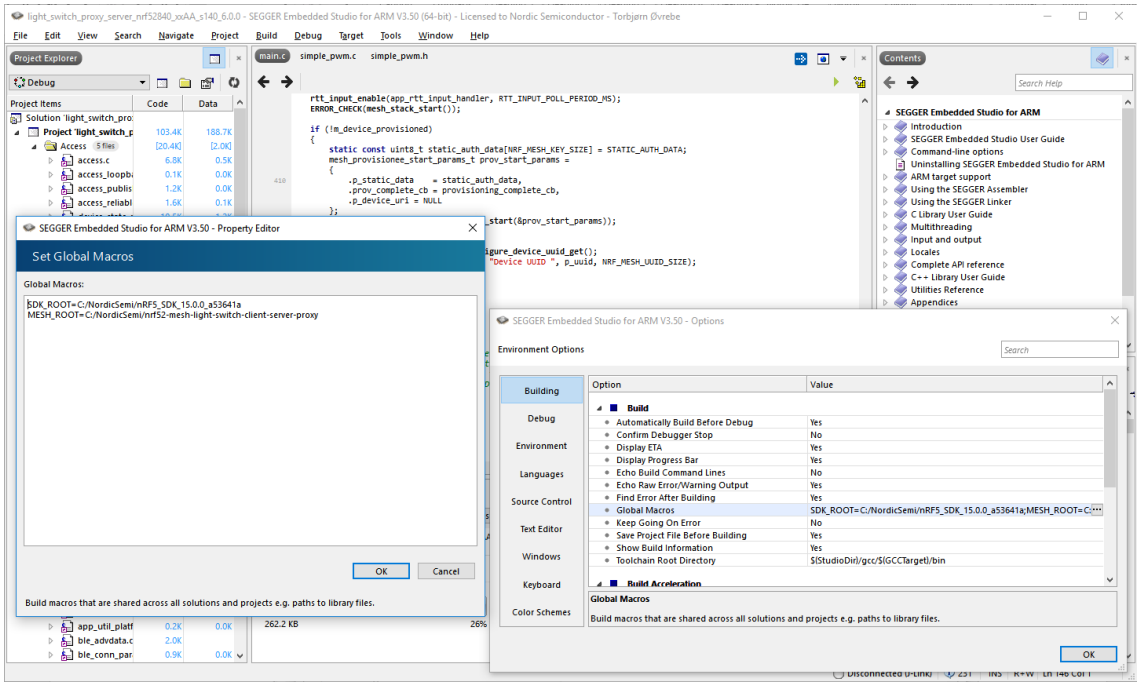
   It is recommended to put both the mesh SDK and the nRF5 SDK in the following folder: c:\NordicSemi\

   Make sure to use Segger Embedded Studio v3.40 or newer.

2) Navigate to: *C:\NordicSemi\nrf52-mesh-light-switch-client-server-proxy\examples\light_switch_london\proxy_server* & open the *light_switch_proxy_server_nrf52840_xxAA_s140_6_0_0.emproject* file

3) Click on Tools -> Options -> Building tab -> double click on Global Macros

   Make sure that the SDK_ROOT & MESH_ROOT point to the correct locations:



4) Open the main.c file and change the DEVICE_NAME define on line 93 to something unique.
This is to avoid accidently provisioning someone else's node once the hands on starts.

5) Compile the example by pressing F7 or *Build -> Build light_switch_proxy_server_nrf52840_xxAA_s140_6.0.0*

6) Make sure your nRF52840 DK is connected to the PC over USB and turned on.
Flash the example by selecting *Target -> Download light_switch_proxy_server_nrf52840_xxAA_s140_6.0.0*, or you can start a new debug session by pressing F5 (in this case the board will be flashed automatically).
If asked to update the board firmware click yes, and wait for the update procedure to complete.

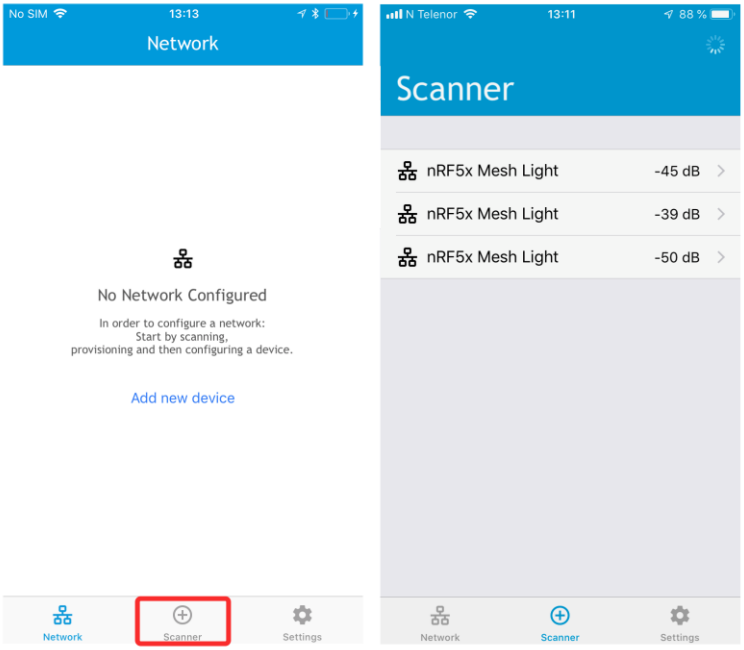
7) Go through the rest of the guide. The proxy server example contains both the generic on/off client & server models. They are slightly simplified compared to the generic models, but still contain most of the same functionality. This enables flexibility to use a device as either a client (e.g. switch) and/or server (e.g. light).

2 Hands-on

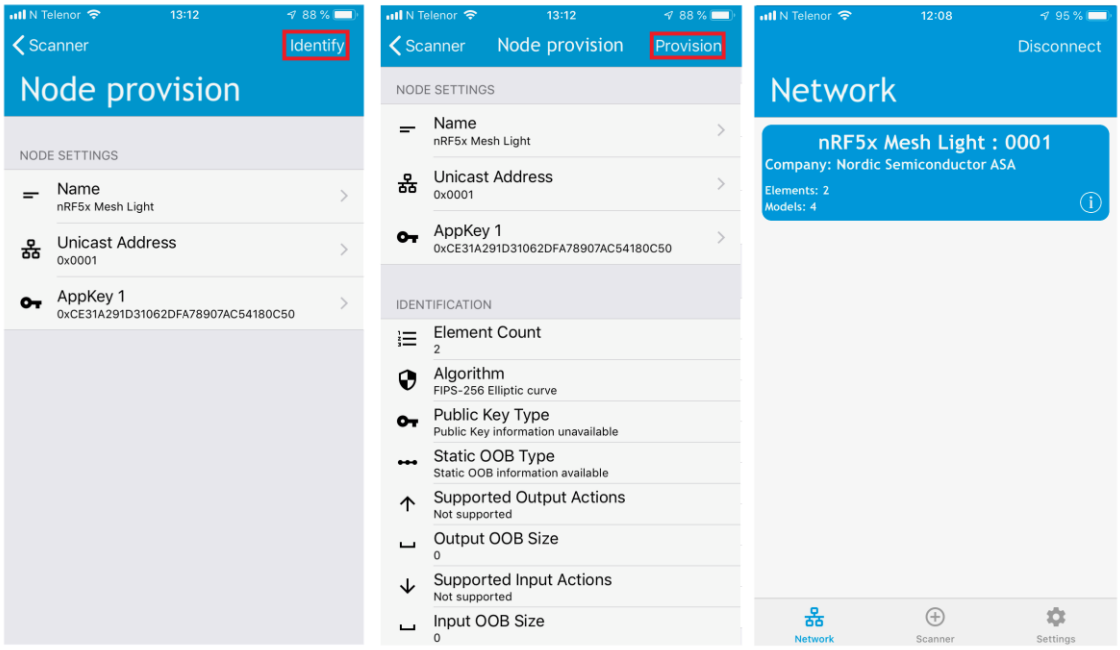
Unless otherwise noted, all the following steps should be performed by one member of the group. Try to make sure that everyone gets to see what is happening, and that everyone in the group gets some experience using the app.

2.1 Mesh provisioning basic example

1) Download and run the nRF Mesh app for iOS or Android on a single phone in the group. Navigate to the scanner section, and verify that all the devkits with your device name are present in the list:



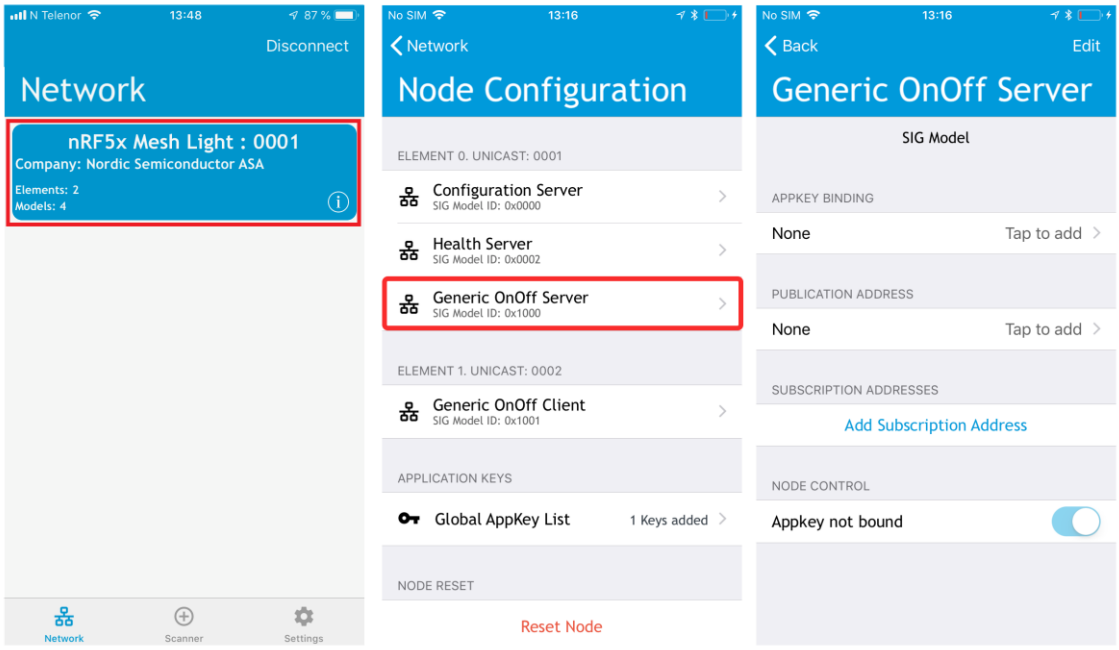
2) Select one of the devkits in your group. Click “IDENTIFY”, and next click “PROVISION”. Wait for the provisioning process to complete.



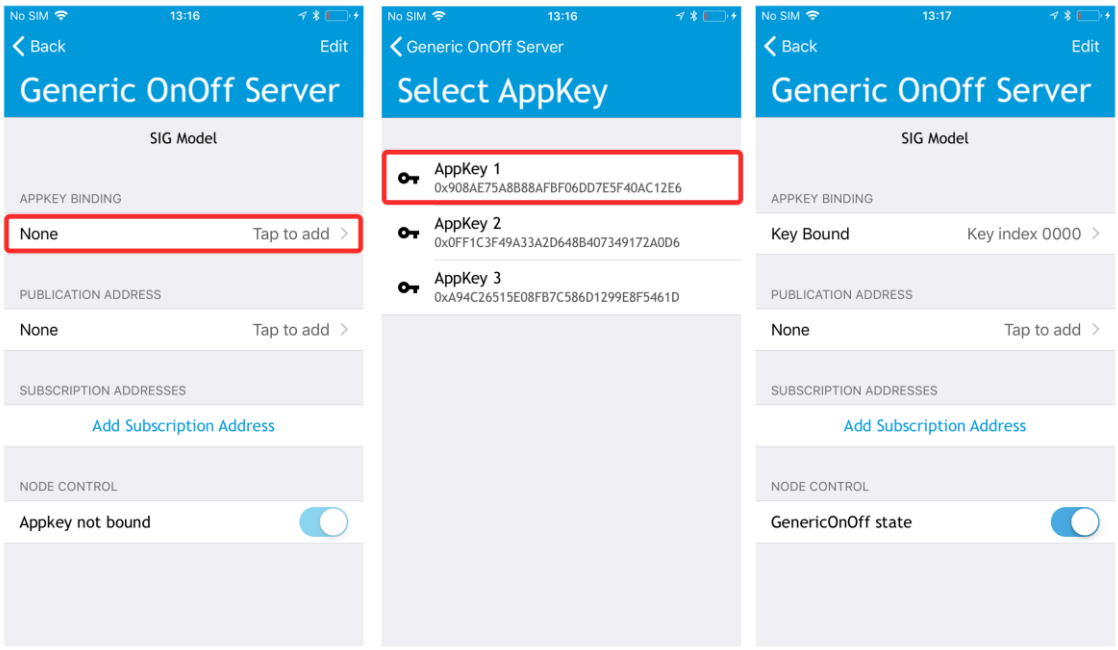
3) Once a devkit is provisioned and configured we can finalize the configuration by binding app keys and assigning addresses to the models.

From the network overview click on the devkit you just provisioned to open the Node Configuration screen.
You should see two elements that can be expanded by clicking the arrow symbol, revealing three models in the first element and one element in the second as shown in the picture below.

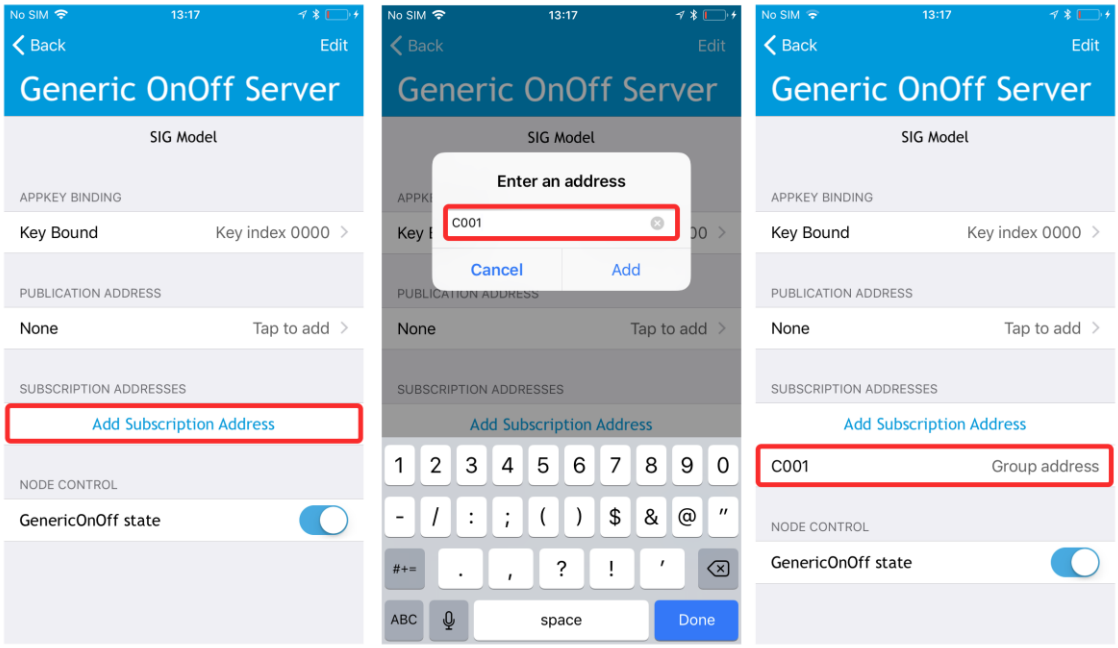
Click on the item called “Generic On Off Server”



4) Before the model can publish or subscribe to an address we need to bind it to an app key. Click “Tap to add” under “APPKEY BINDING”, select “AppKey 1”, and confirm that the key was bound successfully to the model as shown below:

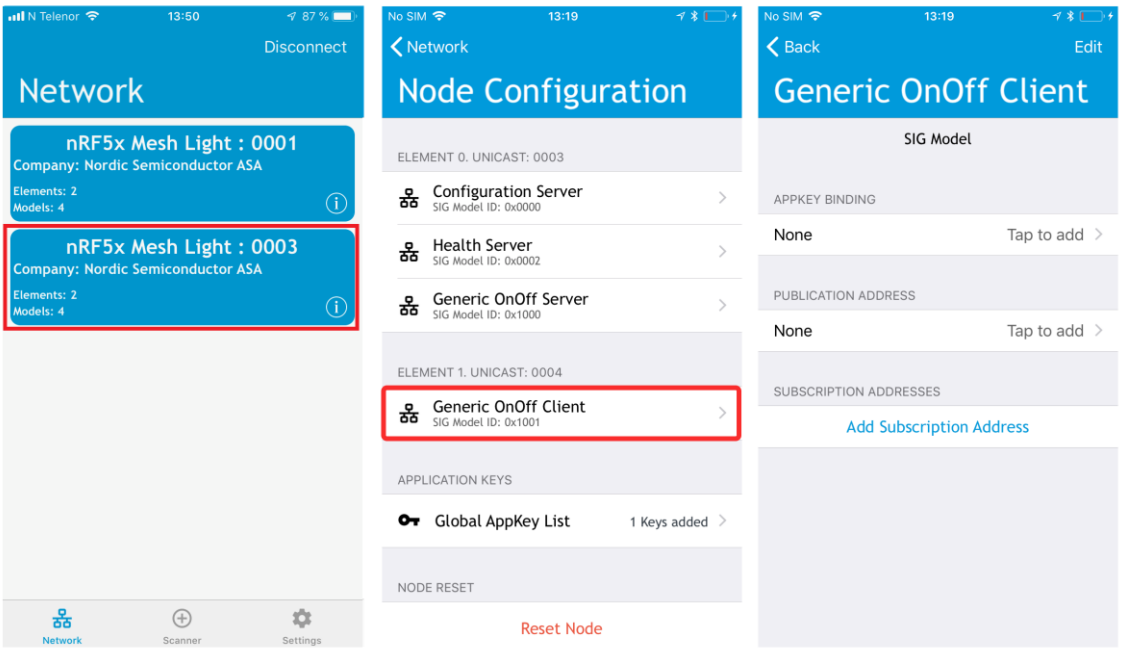


5) Now we can subscribe to a group address. To do this click “Add Subscription Address” and enter the address 0xC001 (or any other address starting with 0xC).



6) Now we should provision a second devkit into the mesh. To do this simply repeat the steps described in 1) and 2).

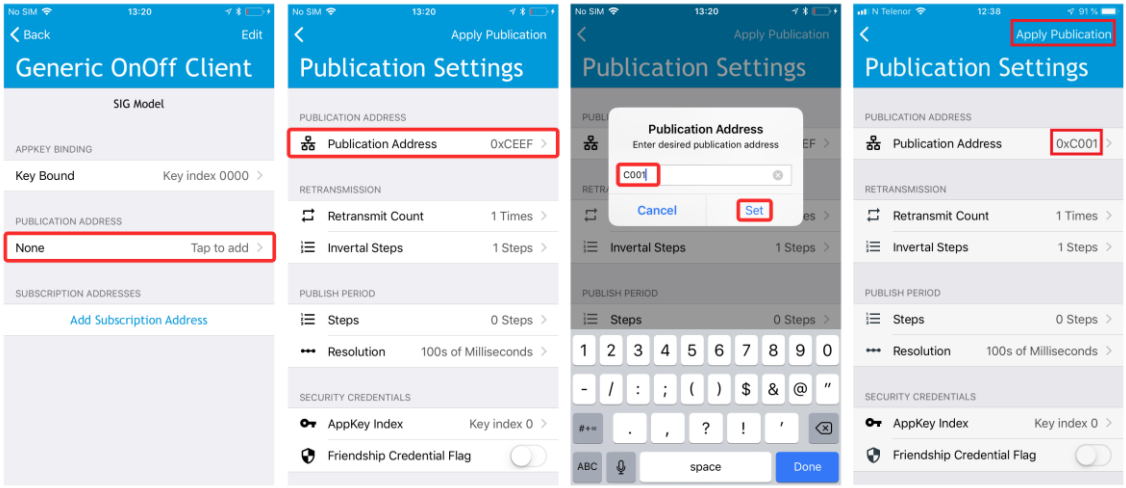
7) Go back to the network overview and click on the second devkit in the list.
Click on the “Generic OnOff Client” item to open the Generic OnOff Client configuration screen.



8) Once again, we need to bind the model to an app key, similar to step 4). Make sure to select the same key as before.

9) Click on “Tap to add” below the “PUBLICATION ADDRESS” header, click on the “Publication Address 0xCEEF” field, and enter the same address as you configured in step 5).

   Press “Set” & then the “Apply Publication” button and verify that the publish address is set.



10) Check that the two devkits are configured correctly, by clicking the button on the second devkit and seeing the LED toggle on the first devkit.

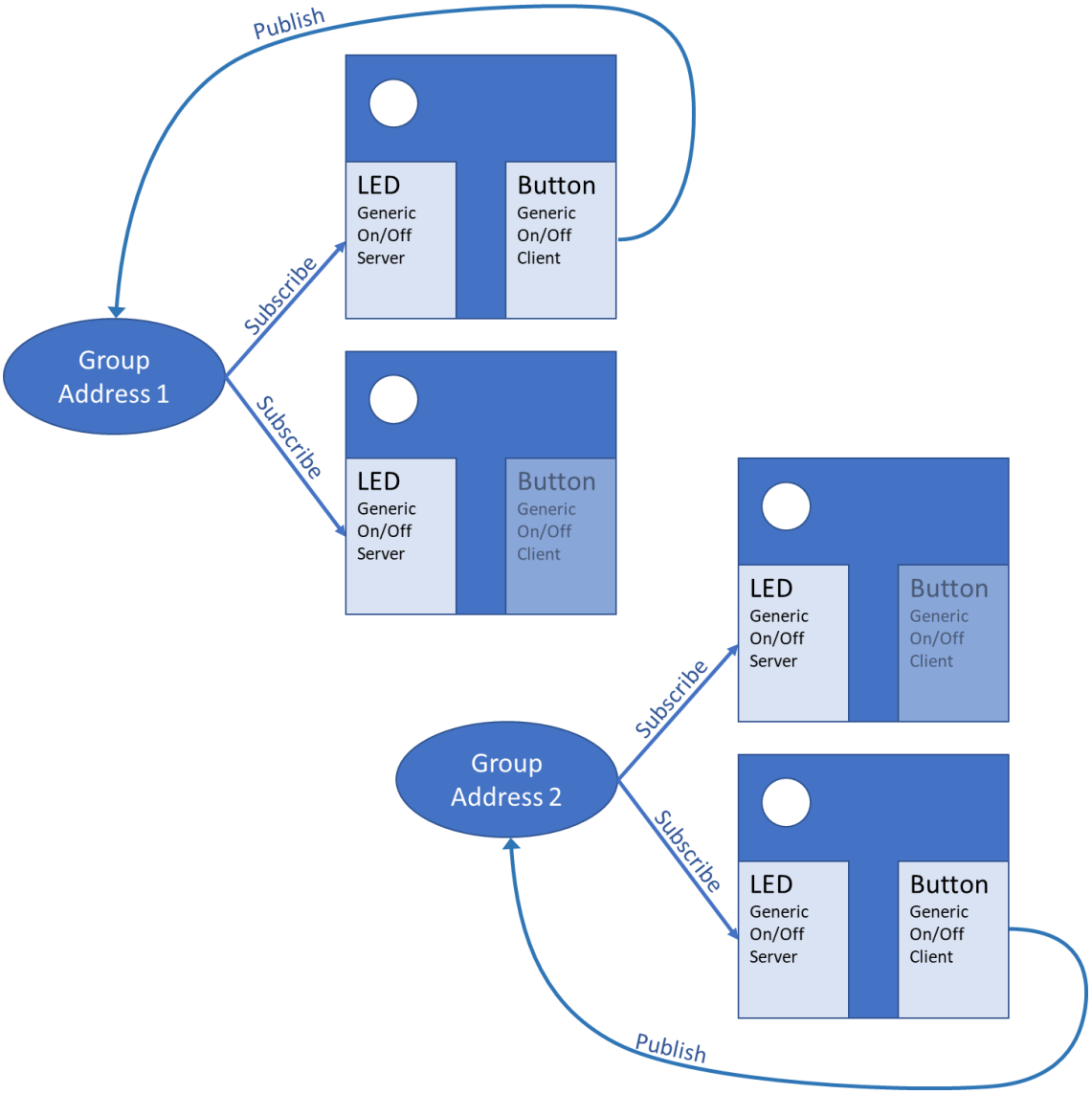
    Congratulations! You have just provisioned and configured your first mesh network using the provisioning app ☺

2.2 Extend the mesh

Note: For the hardware academy workshop you should skip this section, as time is limited, and move to hands on exercise 2.3.

A mesh consisting of two nodes is not very impressive, and the next task is to add any remaining devkits to the mesh network and assign them to different groups.

Unless the group has more than 4 devkits available it might be necessary for some of the devkits to act both as buttons and LED's, by configuring both the client and server. An example of how this can be done is shown in the picture below:



1) Start by provisioning any remaining devkits, as described in the previous section.

2) Partition all the available devkits into two groups and assign each group to its own group address (0xC001 and 0xC002 for example).

   This includes both the original 2 devkits, as well as the new devkits provisioned in step 1)

   Make sure that the Generic On Off Server subscribes to the group address that the devkit belongs to. For a reminder on how to do this, refer to steps 3-5 in section 2.2.

3) Let two devkits publish to either of the group addresses, allowing you to control the two light groups individually.

   For a refresh on setting up publication, refer to steps 7-9 in section 2.2.

   Hint: A single devkit can be both a subscriber and publisher at the same time. Just remember to subscribe on the simple on off server (LED), and publish on the simple on off client (button).

4) Bonus task: Try to add all devkits to a third group (0xC003 for example) and have one of the devkits publish to this address. This will allow all the LED's to be controlled at the same time, while still allowing the two previously defined groups to be used.

   Hint: A server can subscribe to several group addresses.

2.3 PWM integration

The purpose of this hands on is to modify parts of the code in the mesh example to change the behaviour of the application.

The task involves controlling the LED's through PWM (pulse width modulation) to allow fading or pulsing of LED's instead of just turning them on or off.

To simplify the task a small PWM driver has been provided, and the PWM interface is described through the simple_pwm.h header file.

1) Initialize the PWM driver in main(), as indicated by the TODO comment
After initialization, set one of the LED's to pulse or blink repeatedly.

```
int main(void)
{
    initialize();
    execution_start(start);

    // TODO: Hands on 2.3 - Initialize the simple_pwm library to use the 4 LED's on the devkit
    //                      To verify that the PWM driver works, set one of the LED's to blink or pulse in a loop
    //                      Hint: Look at the comments in simple_pwm.h for examples of how to use the simple_pwm library
    //                      Hint2: The LED's on the board are defined by the BSP_LED_x defines, where x is 0-3

    for (;;)
    {
        (void)sd_app_evt_wait();
    }
}
```

2) Modify the on_off_server_set_cb() callback in main.c to fade LED0 in and out when the LED status is changed, rather than have it set or cleared immediately

```
static bool on_off_server_set_cb(const generic_on_off_server_t * p_server, bool value)
{
    // TODO: Hands on 2.3 - After initializing the PWM library in main(), change this function to use the PWM Driver instead of the hal_led_.. functions
    //                      Try to make the LED's fade in and out when the callback occurs, rather than having it set/cleared immediately
    uint32_t err_code;
    __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "Got SET command to %u\n", value);
    if (value)
    {
        hal_led_pin_set(ONOFF_SERVER_0_LED, true);
        m_led_flag = true;
    }
    else
    {
        hal_led_pin_set(ONOFF_SERVER_0_LED, false);
        m_led_flag = false;
    }

    return value;
}
```

3) Look for any remaining references to hal_led_ functions in main.c, and replace them with calls to the PWM library.

4) **(optional)** Currently nothing happens in the code when the "Identify" button is pressed in the nRF Mesh app during provisioning.

In order to change this it is necessary to make changes to some of the files in the mesh SDK:

- In mesh_provisionee.c, the switch case in the prov_evt_handler() function needs to be extended to handle the NRF_MESH_PROV_EVT_INVITE_RECEIVED case, and forward an event to the application informing it of the event.

- The mesh_provisionee_start_params_t struct in mesh_provisionee.h needs to be

extended with an additional callback, which should occur when the INVITE_RECEIVED event occurs.

- Add code in prov_evt_handler() to call this callback when the NRF_MESH_PROV_EVT_INVITE_RECEIVED event occurs.

- In main.c, add a function to handle the invite received callback, and make sure to include it in the provisionee configuration in the start() function

- In the callback function you just created, add code to blink or pulse a LED when the invite callback occurs

3 Troubleshooting

3.1 Lost connection to the mesh

Sometimes the link between the phone and one of the mesh nodes could be broken. In this case it is necessary to reconnect manually from the app on the phone to get access to the mesh:

1) Click the “Connect” button in the app.

2) Click on any of the devices showing up in the list (each node in the mesh can act as a gateway to the phone)

Now the phone should be connected again, and the nodes can be configured.

3.2 Mesh stability issues

If any problems occur during the provisioning or configuration it might be necessary to reset the devkits and the app and start from the beginning.

To reset the devkit, click Button 3 while the kit is running.
Alternatively, do a full erase and reflash the kit with the mesh firmware.

To reset the app, navigate to the “settings” section of the app and press the “Reset Network” button.