Lab: Thread

Introduction

The purpose of this lab is to get you some hands-on experience with Thread. This will come in a couple of different forms:

- Create a network of thread devices
- Send data over IP between devices in a network
- Visualize the network topology

This lab will be a little different from the last one. We won't actually be writing any code. Instead we'll use a Command Line Interface (CLI) on our thread boards to manually configure the network.

Goals

- Understand steps to create a Thread network and join it
- Explore UDP communication over a Thread link
- Consider the topology of a mesh network

Equipment

- Computer (at least one non-MacOS computer)
- nRF52840DK + USB cable (3 total for the group)

Github Classroom

- None today. No code to write this time.

Partners

- This lab should be done with **your group of three**

Submission

- Write your answers up for each task and submit a PDF to Gradescope.

  Remember: I'm not looking for a formal lab report. Just your answers in any format that makes sense. The goal is to prove that you did the lab and spent some time thinking about it.

Table of Contents

Introduction

Table of Contents

List of Tasks

1. Software Requirements

2. Upload Thread CLI

3. Build your own thread network

4. Send UDP messages between boards

5. Observe Network Topology

List of Tasks

- Section 3.1: Show your Thread network details
- Section 4.1: Demonstrate UDP communication
- Section 5.1: Demonstrate the topology tool working

1. Software Requirements

This lab will require two things: programming boards and opening serial consoles to boards.

For programming boards, you have a few options:

1. The “nRF Connect for Desktop” programmer app works just fine for those of you on Windows.

2. You can use `nrfjprog` to flash boards from terminal.

   PlatformIO should have installed it for you in the BLE lab, but you can also get it directly from Nordic Semiconductor's website: https://www.nordicsemi.com/Products/Development-tools/nrf-command-line-tools/download

   To program a hex file onto a board, use the following command:
   `nrfjprog --program <HEXFILE> -f nrf52 --chiperase --verify`
   (where <HEXFILE> should be replaced with the path to the hex file you want to upload)

3. One option for a “terminal” experience on Windows is to open PlatformIO and select the “PlatformIO Core CLI” option from there. In that, you can run the same `nrfjprog` command from above, even on Windows.

For serial consoles you also have a few options:

1. The `miniterm` python library.

   To install it, you need to pip install pyserial which you likely already did to get BLE Wireshark scanning working anyways!

   To run it on Linux: `miniterm -f direct <PORT> 115200`
   The “`-f direct`” part is only important for this lab, as the CLI output has console escape codes.

   On MacOS the command isn’t usually installed, but you can invoke it from python directly: `python3 -m serial.tools.miniterm -f direct <PORT> 115200`

   In both cases, the `<PORT>` is the serial device. Something like `/dev/ttyACM0` or `/dev/tty.usbserial1312222`

2. You could open a serial console in the PlatformIO command line. After you open PlatformIO, select the “PlatformIO Core CLI” to get a command line. Then you can run the command: `pio device monitor -b 115200 -f direct`

   You might need to add the “`-p`“ flag and a `<PORT>` if you have more than one device.

3. PuTTY works fine on Windows. This guide is fine: https://pbxbook.com/voip/sputty.html You don’t have to do anything special to make the console escape codes work on it.

2. Upload Thread CLI

The Thread Command Line Interface has already been compiled into a hex file for you. You can download the file here:
https://drive.google.com/file/d/19X5wqs7QQ97DiUdpLYNqfRuJeBqTSsN2/view?usp=share_link

- Upload that file to a board
- Power cycle the board (off and back on)
- Connect a serial console to the board at 115200 baud. Type “help” and hit enter. You should see something like the following:

```
[brghena@ubuntu thread] $ miniterm -f direct /dev/ttyACM1 115200
--- Miniterm on /dev/ttyACM1  115200,8,N,1 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
help
Please press the <Tab> button to see all available commands.
You can also use the <Tab> button to prompt or auto-complete all commands or its subcommands.
You can try to call commands with <-h> or <--help> parameter for more information.

Shell supports following meta-keys:
  Ctrl + (a key from: abcdefklnpuw)
  Alt  + (a key from: bf)
Please refer to shell documentation for more details.

Available commands:
  clear              :Clear screen.
  device             :Device commands
  devmem             :Read/write physical memory
                      Usage:
                      Read memory at address with optional width:
                      devmem address [width]
                      Write memory at address with mandatory width and value:
                      devmem address <width> <value>
  flash              :Flash shell commands
  gpio               :GPIO commands
  help               :Prints the help message.
  history            :Command history.
  kernel             :Kernel commands
  nrf_clock_control  :Clock control commands
  ot                 :OpenThread subcommands
                      Use "ot help" to get the list of subcommands
  resize             :Console gets terminal screen size or assumes default in
                      case the readout fails. It must be executed after each
                      terminal width change to ensure correct text display.
  sensor             :Sensor commands
  shell              :Useful, not Unix-like shell commands.
uart:~$
```

- If you serial terminal isn’t handling escape codes correctly, you’ll see a bunch of “[22C” and things like that. Be sure to add the “`-f direct`” filter for miniterm. Other serial consoles have different ways of accomplishing the same thing.

```
[22Cterminal width change to ensure correct text display.
  sensor             :Sensor commands
  shell              :Useful, not Unix-like shell commands.
[1;32muart:~$ [m
```

3. Build your own thread network

Once you have at least two boards with the Thread CLI loaded onto them, you can build your own network.

I recommend connecting to each board on a separate computer. That'll make it easier to remember which console is which board and will allow you to add some physical distance between the boards.

We're going to be using the Commissioning process, which requires that a board be authenticated and authorized to become part of the network. One of your boards will have at least three roles: Thread Leader, Commissioner, and Router. The other board will either be a Router or an End Device, depending on your configuration. Commissioning in Thread is described here (we'll be doing "on-mesh" commissioning without a Border Router): https://developer.nordicsemi.com/nRF_Connect_SDK/doc/2.2.0/nrf/ug_thread_commissioning.html

- Here is a guide to the CLI commands: https://github.com/openthread/openthread/blob/main/src/cli/README.md

  A warning, there are some commands on there that aren't in our CLI implementation. Maybe the docs are newer (or older) than our version?

  You'll need to look through a bunch of those, as they'll be very important. All of them are behind the "ot" command to start with, even though they don't show it in the examples. So something like "`ot scan`", not just "`scan`".

- Generally, to get started, you should follow major steps 2 ("Disabling the Thread Network") through 4 ("Adding the Joiner") here: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/2.2.0/nrf/ug_thread_commissioning.html#configuring-on-mesh-thread-commissioning

  A couple of notes for problems I ran into:

  a. When running "ot commissioner joiner add…", the goal is to allow a specific device to join the network if it has the key passphrase. So the address there is the EUI64 from the device that is joining the network. Also, you can use your own passphrase, but it has to follow the passphrase creation rules (At least six characters, no `I`, `O`, `Q`, or `Z`.)

  b. When you actually join with the secondary device, it should work the first time. I ran into an issue though where it totally failed the second time. It turns out the device caches the PAN ID from the last network it connected to. You can set the PAN ID to 0xffff to overwrite this and get joining working again.

- Get all three of your boards connected together into a single network.

- Make one of the three boards into a child device.

  All boards by default act as routers. Not right away as the join: remember that all thread devices join as the child of an existing router. But they quickly (10-30 seconds) upgrade to be routers.

  You can make a board become and stay and End Device (child) though. To do so, take a look at the `ot routereligible` and the `ot state` commands.

1. **TASK:** Show me details of the network you created
   a. The network Dataset (on the commissioner)
   b. The IP addresses of all three devices
   c. The Router Table on the commissioner
   d. The Child Table on the commissioner

4. Send UDP messages between boards

Now that you have a network, it would be nice to be able to send data between the boards. Since Thread uses IP, we can use a standard protocol for communication: UDP.

- Send UDP packets between boards.

  Here are details on the `ot udp` command with examples:
  https://github.com/openthread/openthread/blob/main/src/cli/README_UDP.md

  You should definitely look at the `ot udp connect` option for long-running communication. With three boards you could create a triangle of communication, or you could do a two-way connection and slowly realize you've made a wireless chat client.

  Some things to be aware of:
  - Only some (maybe just one) of the IP addresses given to nodes are actually going to work for this. For example, the link-local address will not work.
  - The port number is up to you, but both sides need to agree.

1. **TASK:** Demonstrate UDP transmissions
   a. Show me both sides of the communication (sender and receiver)
   b. Also demonstrate a packet longer than a 802.15.4 payload, such that it must have been fragmented and reconstructed beneath the UDP layer of the stack

5. Observe Network Topology

Since Wireshark extcap is pretty janky, we're going to skip that this time and use a different tool. Instead of decoding raw packets, we'll introspect network topology.

- The tool to do this is the "nRF Thread Topology Monitor": https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_nrf_ttm%2FUG%2Fnrf_ttm%2Fttm_introduction.html&cp=10_6

  Note that this works on Windows or Linux (tested on Ubuntu, but probably others too).

  It does NOT work on MacOS. You will need to create a VM with Linux if everyone in your group has a Macbook. It's not going to work at all on ARM (MacOS M1/M2) though. If everyone in your group has an ARM MacOS computer, let me know.

- You'll need to load the hex file from the downloaded zip onto one of your boards.

  Warning: don't do this to the commissioner! You need that one. Pull one of the other boards off the network.

  You can use the "nRF Connect" Programmer app or `nrfjprog` to upload the hex file. The hex you want is for the PCA10056 (nRF52840DK) and should use UART (not USB). When I downloaded it, it was named "nrf52840_xxaa_pca10056_uart.hex".

- Start up the tool.

  When you open the tool, you'll need to connect to your board. I recommend doing this on a computer that isn't running one of the two boards still in the Thread network.

  You'll need to enter some details from the Dataset on the commissioner: Channel, PAN ID, and Network Key.

  When I got this working, it took a couple of tries and some patience. It takes like 20 seconds for the tool to join your network if the parameters are right. It's easy to mess up one of the numbers in the Network Key too. If things aren't working: close the tool, power cycle the board, and then pull it all back up and try again.

1. **TASK:** Demonstrate the topology tool working
   a. Take a screenshot of your network
   b. Change the non-commissioner node to a different role (router/child) and take a screenshot of the changed topology
   c. Include the Router Table and Child Table from your commissioner