Open Thread Border Router on Linux Ubuntu

REVISED from the a project named "A Thread Border Router with Pi 4 and nRF9160"' which was NOT SUCCESSFUL.This version will use the nRF52840

Things used in this project

Hardware components

Nordic Semiconductor nRF52840 Dongle × 1

Thread Border Router × 1

OpenThread Border Router Codelab tutorial × 1

Software apps and online services

Nordic Semiconductor nRF Connect SDK

Microsoft Visual Studio Code Extension for Arduino

Ubuntu Linux

Ubuntu 22.04.3 LTS The latest LTS version of Ubuntu, for desktop PCs and laptops.
LTS stands for long-term support

Story

Revised: September 26,2023 for the Make it Matter! Smart Home Design Contest.

I spent a lot of time trying to get this original project to work. I finally got frustrated and gave up.

I applied o FREE hardware and I was awarded a Nordic nRF522840 Dongle for the Contest and decided to revise this project and see I i get any further. My goal is to use an Open Thread Border Router (OTBR) to implement Matter Over Thread.

The changes to the document involbe the following two fundamental changes.

- I'm using the **nRF522840 Dongle** instead of the Nordic NRF9160 DK.
- I am implementing the OTBR on my development Linux PC which is running Ubuntu. I decided to use the OpenThread Border Router docker image instead of Installing OTBR Manually.

The original OS running on the Raspberry PI was "Raspbian GNU/Linux 10 (buster)", This is a 32 bit OS. The contest page recommends Ubuntu 64bit V20.04 as the Linux OS. I decided to try Ubuntu 22.04.3 LTS on my PI 4 2

GB. I quickly found it to be too SLOW for development. I switched gears and decided to use my PC development box running Ubuntu. I will highlight what I do differently and hopefully report a successfull results.

```
I will place the old content that does not work in a BLACK BOX
```

OK now on with the revision.

INTRODUCTION

A Thread Border Router connects a Thread network to other IP-based networks, such as Wi-Fi or Ethernet. A Thread network requires a Border Router to connect to other networks. The Thread BR has two RF sides WiFI and the Thread Network

To Setup the Thread Border Router on a Linux PC. I followed the detailed steps, on the Thread Border Router page in the nRF Connect SDK documentation.

Typically, a Border Router solution consists of the 2 applications

1. An application based on the Network co-processor (NCP) design or its Radio co-processor (RCP) variant compatible with the IEEE 802.15.4 standard. This application can be implemented, for example, on an nRF52 device.

REVISED: I connected my Nordic nRF52840 Dongle to my Ubuntu PC and used it as RCP in this revise edition

```
PREVIOUS: I am using the nRF9160 DK with the nRF52/nRF91 switch set to nRF52.
```

2. A host-side application, usually implemented on a more powerful device with an incorporated Linux-based operating system.

REVISED: I am using my Ubuntu PC to host the OTBR Application.

The nRF Connect SDK does not provide a complete Thread Border Router solution. For development purposes, you can use the OpenThread Border Router (OTBR) released by Google, an open-source Border Router implementation that you will set up on Linux

```
PREVIOUS: I implemented the OTBR on the Rasberry Pi using Raspbian:GNU/Linux 10 (buster), a version of the Deblian distribution.
```

First you need to configure a radio co-processor (RCP), which provides the required radio capability to your Linux PC.

Configuring a radio co-processor

Instructions to accomplish this, are on the this page: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/protocols/thread/tools.html#configuring-a-radio-co-processor

The OTBR must have physical access to the IEEE 802.15.4 network that is used by the Thread protocol. A Linux PC has no such radio capability; you must connect an external nRF device that serves as a radio co-processor.

REVISED: I plan to connect my Nordic nRF52840 Dongle to the Linux PC and use it as an RCP in this revised edition. But first I need to get the Nordic RCP example onto my Dongle.

```
PREVIOUS: I am using the nRF9160 DK with the nRF52/nRF91 switch set to nRF52.
```

Complete the following steps, to configure the nRF52840 Dongle device with the RCP sample application.

REVISED: use the following 2 tabs "nRF52840 dongle (USB transport)" from the page link above. The dongle is pluged into a USB connection on the machine used for the OTBR.

You will need to install the Nordic toolchain for the nRF Connect SDK on the Linux PC to build the RCP firmware to put on the Dongle.

Download the NRF Connect SDK for Linux at Download nRF Connect for Desktop The file that is downloaded has a .AppImage extension. This is an installer file. You will need to make this runnable so you can run it. The following instructs you how to make it runnable. It's quite simple to run AppImages. All you have to do is download them, make them executable and run them. This can either be done using the GUI or via the command line.

To use the GUI, follow these steps:

- Open your file manager and browse to the location of the AppImage
- Right-click on the AppImage and click the 'Properties' entry
- Switch to the Permissions tab and
- Click the 'Allow executing file as program' checkbox if you are using a Nautilus-based file manager (Files, Nemo, Caja), or click the 'Is executable' checkbox if you are using Dolphin, or change the 'Execute' drop down list to 'Anyone' if you are using PCManFM
- Close the dialog
- Double-click on the AppImage file to run

Once the you get the toolchain installed, run nRF Connect for desktop and install and open the Toolchain Manager. Install the version of the SDK you desire. I used version 2.4.2 as shown below.

As shown above, from the pull down menu to the right of the sdk install bar, Select "Open Bash". this will bring you to a bash command window, that has the west tool build environment setup for you. You are also dropped into the tip directory of the SDK version you just installed. Run through all the steps from this bash command window. I have included my successful output to all the commands in the Code section at the end of this Story, Titled "Terminal output for executing the commands to configure a radio co-processor".

The commands were executed on a WINDOW10 OS and Ubuntu 20.04.2 LTS

STEP 1 - Build the Thread: Co-processor sample for the hardware platform

```
$ west build -p always -b nrf52840dongle_nrf52840 nrf/samples/openthread/coprocessor/ -- -DOV
ERLAY_CONFIG="overlay-usb.conf" -DDTC_OVERLAY_FILE="usb.overlay"
```

Step 2 - Install nRF Util as follows:

- Download the nrfutil executable file from https://www.nordicsemi.com/Products/Development-tools/nRF-Util

- I copied the downloaded executable to the root directory of my command line directory where I'm building everything. I needed to place a ./ in front of the nrfutil command to run it which I sometimes forget to do.

Step 2.1 - Run nrfutil install nrf5sdk-tools. When nrf5sdk-tools is installed, nRF Util behaves in the same way as nRF Util 6.1 and earlier.

```
$ ./nrfutil install nrf5sdk-tools
```

Step 3 - Generate the RCP firmware package:

```
$ ./nrfutil pkg generate --hw-version 52 --sd-req=0x00 \
 --application build/zephyr/zephyr.hex --application-version 1 build/zephyr/zephyr.zip
```

Step 4 - Install the RCP firmware package onto the dongle

I was able to build and program the Dongle using the Nordic CLI commands on Windows10 and also on Ubbutu.

1. Connect the nRF52840 Dongle to the USB port.

2. Press the RESET button on the dongle to put it into the DFU mode. The LED on the dongle starts blinking red. NOTE: the RESET button is next to the white SW1 button. The REST button is a side mounted button marked reset I was pressing the wrong button and the LED was not blinking and it was not in DFU mode. The last command was not working for me until I pressed the correct one!!

3. Run the following command, with /dev/ttyACM0 replaced with the device node name of your nRF52840 Dongle:

On Windows I used this command

```
$ ./nrfutil dfu usb-serial -pkg build/zephyr/zephyr.zip -p COM5
```

On Ubuntu I used this command, my dongle was on ACM0

```
$./nrfutil dfu usb-serial -pkg build/zephyr/zephyr.zip -p /dev/ttyACM0
```

Now I'm ready to attach the dongle to my Linux PC to see if works as the RF end of the OTBR. in the next section. As you will see that I had a lot of problems and was unable to get a ODBR running because of it.

This worked a lot better then my previous experience that had originally tried in my first project.

```
PREVIOUS:·
Build the  Thread: Co-processor  sample on the following Tab
used the 2 tabs nRF52840 Development Kit (UART transport)
problems with west.

CLI command
west build -p always -b nrf52840dk_nrf52840 nrf/samples/openthread/coprocessor/
I used VS-CODE and used the sample at nrf/samples/openthread/coprocessor/ under the 2.1.0-rc2
release use
I received a Build error? in 2.02....tried 2.1.0-rc2 and had success..
· Complete the following steps on a successful build
a. Program the image using West (Zephyr's meta-tool):
i. west flash –erase
ii. I flashed it from VS-CODE
b. Disable the Mass Storage feature on the device, so that it does not interfere with the cor
e RCP functionalities. Also, force Hardware Flow Control to avoid potential race conditions r
elated to the auto-detection:
c. HOW TO DO THIS? do I NEED THE CLI TOOLS? MAYBE I SHOULD INSTALL THEM on the nRF9160?
JLinkExe -device NRF52840_XXAA -if SWD -speed 4000 -autoconnect 1 -SelectEmuBySN SEGGER_ID
```

```
J-Link>MSDDisable
Probe configured successfully.
J-Link>SetHWFC Force
New configuration applies immediately.
J-Link>exit
Replace SEGGER_ID with the SEGGER ID of your nRF52840 Development Kit. This setting remains v
alid even if you program another firmware onto the device.
d. Power-cycle the device to apply the changes.
```

Installing OTBR Manually

This option provides most of the functionalities available in the OTBR, such as border routing capabilities needed for establishing Thread communication with a mobile phone on a Wi-Fi network. However, you will need to download the OTBR repository and install the Border Router manually on the Linux PC. I tried this for the Raspberry pi and was not successfull in getting an OTBR working. The steps I followed, 2 and 3 of the official OpenThread Border Router Codelab tutorial did not work for me so I abandoned setting up an OTBR manually and started reviewing the Docker ODBR image running on Ubuntu.

Your welcome to try and get Steps 2 and 3 working on your Raspberry PI and if you get it to work, please leave me your finding and which OS you used, in the comment section of this project.

The introduction page is worth reading, to gain some knowledge what a “Thread Border Router is”. Most importantly these 4 points. Lets really think what each is doing. Because these concepts will come up in the configuration, and are important to understand for a successful implementation of the Thread Network.

Open Thread Border Router (OTBR) is released by Google and is an open-source implementation of the Thread Border Router. A Thread Border Router minimally supports the following functions:

- Bidirectional IP connectivity between Thread and Wi-Fi/Ethernet networks.
- Bidirectional service discovery via mDNS (on Wi-Fi/Ethernet link) and SRP (on Thread network).
- Thread-over-infrastructure that merges Thread partitions over IP-based links.
- External Thread Commissioning (for example, a mobile phone) to authenticate and join a Thread device to a Thread network.

```
An example of this last point would be commisioning a Thread device running firmware (Thingy:
53 running Matter Weather station sample firmware for example) to authenticate and join a Thr
ead network using a  Matter controller on my Phone running CHIP Tool for Android
```

Here’s what I will take away from this exercise.

- How to set up OTBR
- How to form a Thread network with OTBR
- How to build an OpenThread CLI device with the SRP feature
- How to register a service with SRP
- How to discover and reach a Thread end device.

Running OTBR using Docker

I decided to try to use a docker Container implementation of the Google OTBR to get a openthread network up an running for my project. I was successfull in getting an Open Thread Border Router configured and running on my Ubuntu PC.

- My PC is an Intel i3-2330M CPU @ 2.20GHz × 4 running Ubuntu 20.04.2 LTS GNOME: V3.36.8
- Docker: version (24.0.5-0ubuntu1~20.04.1)
- OTBR image:
  REPOSITORY TAG IMAGE ID
  nrfconnect/otbr 84c6aff a8ab2150ebe9
- A Nordic nRF52840 Dongle (with USB transport firmware) attached to the USB port.

Set up OTBR

Use docker and perform the steps outlined from the document
Running OTBR using Docker

For development purposes, you can run the OpenThread Border Router on any Linux-based system using a Docker container that already has the Border Router installed. This solution can be used when you are only interested in direct communication between your Border Router and the Thread network. For example, you can use the Docker container when you want to establish IP communication between an application running on Linux (such as the CHIP Tool Matter controller) and an application running on a Thread node.

To install and configure the OpenThread Border Router using the Docker container on an Ubuntu operating system, complete the following steps:

---- Install the Docker daemon:

```
sudo apt-get install docker.io
```

OK

--- Start the Docker daemon:

```
sudo systemctl start docker
```

--- Create an IPv6 network for the OpenThread Border Router container in Docker:

```
sudo docker network create --ipv6 --subnet fd11:db8:1::/64 -o com.docker.network.bridge.name=otbr0 otbr
```

RETURNED: 4b17674028a0c35d76228168f0ec1f8eb4ac4df93e8b19b56f5506e7cd9175ee

--- Download the compatible version of the OpenThread Border Router docker image by running the following command:

```
docker pull nrfconnect/otbr:84c6aff
```

RETURNED:

Digest: sha256:49cbfd716359bd3e998b336be5c417003e7c50e9615cf2ffe2576413bdfd0d84

Status: Downloaded newer image for nrfconnect/otbr:84c6aff

docker.io/nrfconnect/otbr:84c6aff

--- Connect the radio co-processor that you configured in Configuring a radio co-processor to the Border Router device.

--- Start the OpenThread Border Router container using the following commands:

```
sudo modprobe ip6table_filter
sudo docker run -it --rm --privileged --name otbr --network otbr -p 8080:80 \
--sysctl "net.ipv6.conf.all.disable_ipv6=0 net.ipv4.conf.all.forwarding=1 net.ipv6.conf.all.forwarding=1" \
--volume /dev/ttyACM0:/dev/radio nrfconnect/otbr:84c6aff --radio-url spinel+hdlc+uart:///dev/radio?uart-baudrate=1000000
```

Replace `/dev/ttyACM0` with the device node name of the OpenThread radio co-processor.

Form a Thread network with OTBR Using the WEB GUI

Step 7 gives 2 options to forma a network.

```
Form the Thread network using one of the following options:
1.Follow the instruction in the OpenThread Border Router Codelab tutorial step 3.
2.Open the OTBR Web GUI in a web browser and choose Form from the menu.
```

I followed the 2nd option because it said "One of the following" .

Use the OpenThread Border Router (OTBR) Web GUI to configure and form, join, or check the status of a Thread network. Web GUI docs page: https://openthread.io/guides/border-router/web-g

Open the `http://localhost:8080/` address in a web browser and choose **Form** from the menu.

Note down the selected On-Mesh Prefix value. For example, `fd11:22::/64`.

HERE IS My On-Mesh Prefix: `fd11:22::`

--- Make sure that packets addressed to devices in the Thread network are routed through the OpenThread Border Router container in Docker. To do this, run the following command that uses the On-Mesh Prefix that you configured in the previous step (in this case, `fd11:22::/64`):

```
sudo ip -6 route add fd11:22::/64 dev otbr0 via fd11:db8:1::
```

my command contained no "/64" so here is my command:

```
sudo ip -6 route add fd11:22:: dev otbr0 via fd11:db8:1::
```

--- Check the status of the OpenThread Border Router by executing the following command:

```
sudo docker exec -it otbr sh -c "sudo service otbr-agent status"
```

RETURN: * otbr-agent is running

-- Check the status of the Thread node running inside the Docker:

```
sudo docker exec -it otbr sh -c "sudo ot-ctl state"
```

RETURN:

```
disabled
Done
```

The WEB GUI status screen also shows that the things are not normal

More commands need to be executed

This is the final step in the document that I was following and the status of the thread node running inside the docker container is disabled. At this point i assume that this cannot be a good thing, so I seek some advice from Nordic Devzone and the Discord MakeItMatter Server. I receive some helpful advice from a Nordic moderator on DISCORD. It turns out, that I need to execute some more commands to get the thread node running.

So at this point I executed all the following commands:

```
$ sudo docker exec -it otbr sh -c "sudo ot-ctl ifconfig down"
Done

$ sudo docker exec -it otbr sh -c "sudo ot-ctl thread stop"
Done

$ sudo docker exec -it otbr sh -c "sudo ot-ctl dataset init new"
Done

$ sudo docker exec -it otbr sh -c "sudo ot-ctl dataset commit active"
Done

$ sudo docker exec -it otbr sh -c "sudo ot-ctl ifconfig up"
Done

$ sudo docker exec -it otbr sh -c "sudo ot-ctl thread start"
Done

$ sudo docker exec -it otbr sh -c "sudo ot-ctl state"
leader
Done

$ sudo docker exec -it otbr sh -c "sudo ot-ctl netdata show"
Prefixes:
fd48:5276:8095:1::/64 paos low 7400
Routes:
fd48:5276:8095:2:0:0::/96 sn low 7400
fc00::/7 sa med 7400
Services:
44970 01 24000500000e10 s 7400
44970 5d fda89b12bc80717e93014a1b555b31abd11f s 7400
Contexts:
fd48:5276:8095:1::/64 1 c
Done

$ sudo docker exec -it otbr sh -c "sudo ot-ctl ipaddr"
```

```
fda8:9b12:bc80:717e:0:ff:fe00:fc11
fd48:5276:8095:1:474b:7213:103a:3cd3
fda8:9b12:bc80:717e:0:ff:fe00:fc10
fda8:9b12:bc80:717e:0:ff:fe00:fc38
fda8:9b12:bc80:717e:0:ff:fe00:fc00
fda8:9b12:bc80:717e:0:ff:fe00:7400
fda8:9b12:bc80:717e:9301:4a1b:555b:31ab
fe80:0:0:0:4c1e:6966:69fb:e265
Done
```

```
$ sudo docker exec -it otbr sh -c "sudo service otbr-agent status"
```

```
otbr-agent is running
```

RESULTS

Here are the results of executing the extra commands and the status of the docker container WEB UI showing the status as well.

- Now the status of the Thread Node is now reporting "leader".

```
$ sudo docker exec -it otbr sh -c "sudo ot-ctl state"
leader
Done
```

- The WEB GUI showing the status and the Form menu.

Conclusions

I spent a very long time trying to get this up and running. The Nordic documentation on Step 7 "Forming a Network" is very misleading. I am a novice Ubuntu, Docker, and OpenThread user. Nordic needs to update the Docker OTBR section in the doc page section: Running OTBR using Docker

I had to get advice from the Discord contest moderator @jens to finally get advice to run the extra commands. I was unable to get advice from a post on Nordic Devzone.

This section should also contain more detail in what each command is used for, so the novice user can understand what is being configured.

Hopefully someone from Nordic Docs sees this and gets this section rewritten.