Running an OpenThread Mesh Network with Linux and Zephyr

FOSDEM 2022

Stefan Schmidt <stefan.schmidt@huawei.com>
Principal Solutions Architect, Huawei OSTC

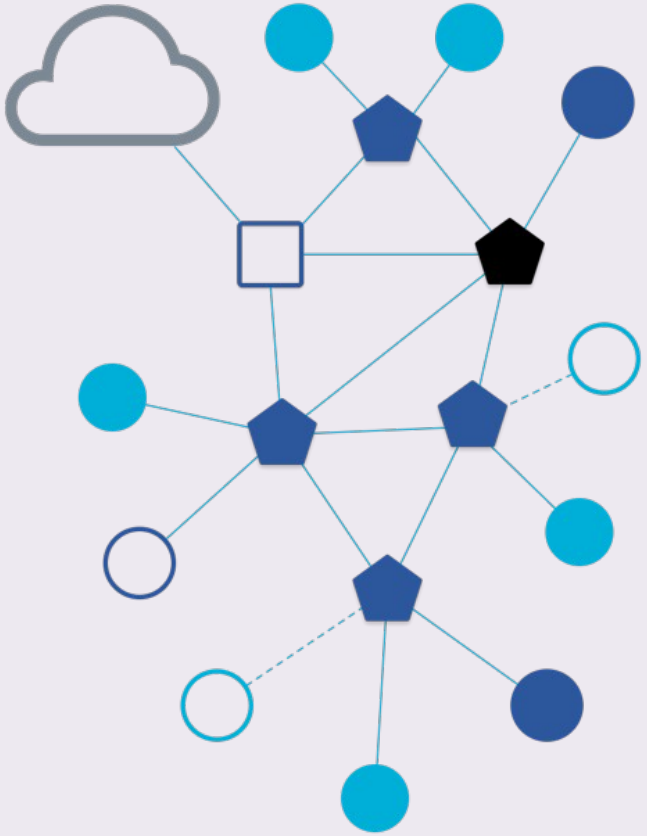
oniro

Agenda

- Motivation
- Blueprint and integration work
- Technical background
  - OpenThread border router on Linux
  - Mesh node on Zephyr
- Building with Yocto
- Onboarding process
- Future work & Summary

Motivation

- Mesh to ensure good coverage and redundancy
- Power usage small enough to run on coin cell battery
- IPv6 end-to-end connectivity
- Internet-enabled microcontroller



Source: https://openthread.io/

OpenThread Project

- Open source (BSD-3) project implementing the Thread network protocol
- Thread specification version 1.1.1 available for everyone
- New versions for Thread group members first
- Active project and community
- Versatile code base that can run on bare-metal, RTOSes and Linux
- https://openthread.io/

OPENTHREAD
released by Google

- Build to re-use existing IEEE 802.15.4 silicon/transceivers
- Build on existing 6lowpan technology (see appendix)
- Filling the gaps to make it a turnkey solution

IoT Gateway Blueprint

- Oniro blueprint to design an IoT Gateway with modern technology
- Aiming for 70% work done towards a real product
- Off the shelf components, hardware BoM and documentation

Gateway:

- Raspberry Pi 4B board (>=2GB RAM)
- Raspberry Pi USB-C Power supply
- microSD card (>=8GB size)
- Makerdiary nrf52840 MDK USB dongle

Mesh node:

- Arduino Nano 33 BLE
- Nordic Semiconductor nRF52840 ARM Cortex-M4 @ 64 Mhz
- 1MB flash memory / 256 KB SRAM
- Bluetooth and Thread compatible radio

- Android mobile phone to run onboarding application

Integration Work

- Yocto/OpenEmbedded recipes (ot-br-posix, ot-daemon, wpantund, etc)
- Meta-oniro-staging layer
  https://booting.oniroproject.org/distro/oniro/-/tree/kirkstone-next/meta-oniro-staging/recipes-connectivity/openthread
- Systemd units for services
- Script for default configuration settings
- Meta-oniro-blueprints layer
  https://booting.oniroproject.org/distro/meta-oniro-blueprints/-/tree/dunfell/recipes-connectivity/openthread
- Upstream fixes for musl build problems
- Hardware enablement & Zephyr application
- Documentation
- Demonstration video

OpenThread Border Router with Linux

- OpenThread border router as service on the gateway
- USB dongle as Thread radio
- Yocto recipe for ot-br-posix
- Systemd unit files
- Configuration script
- Forwarding enabled on interfaces
- WiFi AP to allow easy mobile phone onboarding

Interactive shell to explore OpenThread:

$ ot-ctl

```
# Bring up a OpenThread border router
ot-ctl thread stop
ot-ctl ifconfig down
ot-ctl dataset clear
ot-ctl dataset init new
ot-ctl dataset panid 0x1357
ot-ctl dataset extpanid 11112222deadbeef
ot-ctl dataset networkname OniroThread
ot-ctl dataset channel 26
# J01NME is the Commissioner Credential to be used in the Android app
# J01NU5 is the Joiner Credential, set in the node and QR code
ot-ctl dataset pskc $(pskc J01NME 11112222deadbeef OniroThread)
ot-ctl dataset commit active
ot-ctl ifconfig up
ot-ctl thread start
ot-ctl netdata register
```

OpenThread Mesh Node with Zephyr

- Pick a board with a well-supported SoC in Zephyr (nRF52840)
- Some hardware enablement (e.g. storage partition)
- USB serial console's access

- System configuration from prj.conf in application
- Config options to auto join default network
- Empty business logic in main()

Interactive shell over USB serial:

```
$ screen /dev/ttyACM0 115200
uart:~$ ot
```

```
...
CONFIG_NET_L2_OPENTHREAD=y
CONFIG_OPENTHREAD_DEBUG=y
CONFIG_OPENTHREAD_L2_DEBUG=y
CONFIG_OPENTHREAD_L2_LOG_LEVEL_INF=y
# Default values used on OTBR
# PAN ID 0x1357 to decimal 4951
CONFIG_OPENTHREAD_PANID=4951
CONFIG_OPENTHREAD_CHANNEL=26
CONFIG_OPENTHREAD_NETWORK_NAME="OniroThread"
CONFIG_OPENTHREAD_XPANID="0x11112222deadbeef"
CONFIG_OPENTHREAD_JOINER=y
CONFIG_OPENTHREAD_JOINER_AUTOSTART=y
CONFIG_OPENTHREAD_JOINER_PSKD="J01NU5"
CONFIG_OPENTHREAD_SLAAC=y
```

Building with Yocto

```
$ mkdir oniroproject; cd oniroproject
$ repo init -u https://booting.oniroproject.org/distro/oniro
$ repo sync  —no-clone-bundle
```

Linux Border Router:

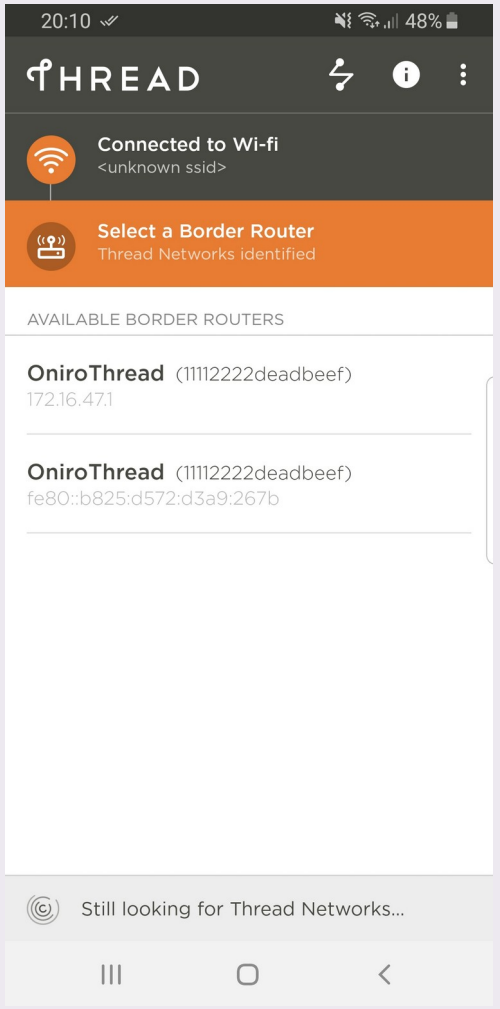
```
$ TEMPLATECONF=../oniro/flavours/linux . ./oe-core/oe-init-build-env build-oniro-linux
$ DISTRO="oniro-linux-blueprint-gateway" MACHINE=raspberrypi4-64 bitbake blueprint-gateway-image
```

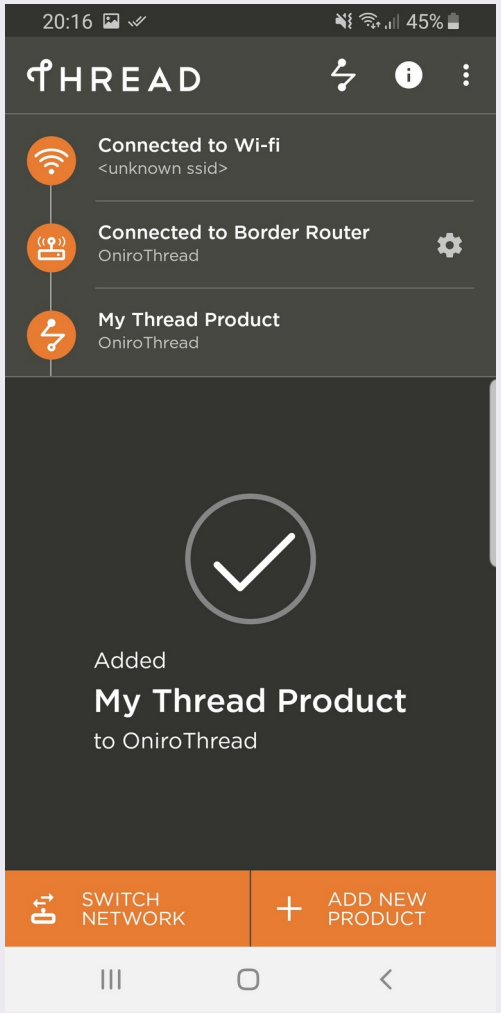
Zephyr Mesh Node:

```
$ TEMPLATECONF=../oniro/flavours/zephyr . ./oe-core/oe-init-build-env build-oniro-zephyr
$ DISTRO="oniro-zephyr-blueprint-gateway" MACHINE=arduino-nano-33-ble bitbake zephyr-blueprint-gateway-node
```

Onboarding

- Generate QR code for node onboarding
  v=1&&eui=00124b001ca77eef&&cc=J01NU5
- Thread 1.1 Commissioning App on Android
- Camera view to scan the generated QR code for the mesh node
- Start Arduino Nano 33 BLE
  to finish the onboarding process





Future Work

- Updating to later OpenThread version and catchup on changes
- Matter protocol integration as a modern IoT connectivity standard
- Flesh out the whole IPv6 connectivity story for the blueprint
- Use an Arduino Nano as Thread radio for border router
- Think about a native Linux-wpan platform for the OpenThread Border Router
- Simpler workflow to build and flash firmware for border router radio

Summary

- Turnkey blueprint for sensors on a wireless link with a small power budget
- Gateway as well as mesh node cases covered
- All open source and ready to be hacked on
- Start building your own software on top of the provided transport layer

Demonstration Video

Documentation

Thank you!

oniroproject.org

Appendix

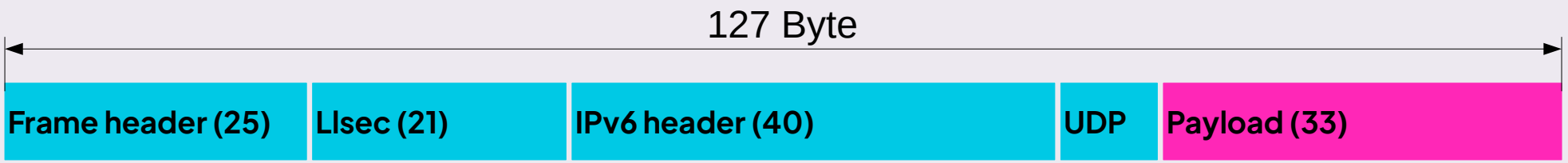
6lowpan in a nutshell

Encapsulation:
- The 6LoWPAN adaptation layer sits between data-link and original network layer
- IPv6 allows for a maximum packet size of 1280 bytes
- Impossible to handle in the 127 byte MTU of IEEE 802.15.4
- Therefore 6lowpan brings back a fragmentation scheme
- But fragmentation can still lead to bad performance in loosy networks, best to avoid

Header Compression:
- Removing information found elsewhere in the frame/packet
- Reduce size by giving up some flexibility
- A few iterations: HC1 & HC2, IPHC, NHC and GHC

Header Size Problem

- Worst-case scenario calculations
- Maximum frame size in IEEE 802.15.4: 127 byte
- Reduced by the max. frame header (25 byte): 102 byte
- Reduced by highest link-layer security (21 byte): 81 byte
- Reduced by standard IPv6 header (40 byte): 41 byte
- Reduced by standard UDP header (8 byte): 33 byte
- This leaves only **33 byte** for actual payload
- The rest of the space is used by headers



6lowpan Header Compression

- IP Header Compression (IPHC) is a core part of 6lowpan

- Defining some default values in IPv6 header
- E.g. version = 6, traffic class & flow-label = 0, hop-limit only well-known values (1, 64 or 255)
- Remove the payload length (available in 6lowpan fragment header or data-link header)

Biggest saving is re-usage of the L2 address for IPv6
- Eliding the IPv6 prefix (global known by network, link-local defined by compression)
- Using the EUI-64 L2 address

Header Size Solution

- Calculations with plain 6lowpan usage for optimal case
- IPv6 with link-local and UDP on top
- IPHC with NHC for UDP
- The 48 byte IPv6 + UDP header could in the best cases be reduced to **6 bytes**
- Double initial payload