Matter and Thread as Connectivity Solution for Embedded

FOSDEM 2023

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

oniro

Agenda

- Scope of this talk
- Matter overview
- Matter on Yocto/OpenEmbedded and Zephyr
- OpenThread overview
- Mesh Capabilities
- Border Router
- mDNS Discovery Proxy & Service Registration Protocol
- Transparent Gateway Blueprint

Scope

- **Low-power** and low-rate **wireless** systems
- **IPv6 End-to-End** with **power budget** in mind
- **Mesh-capabilities** and **Sleepy End Devices**

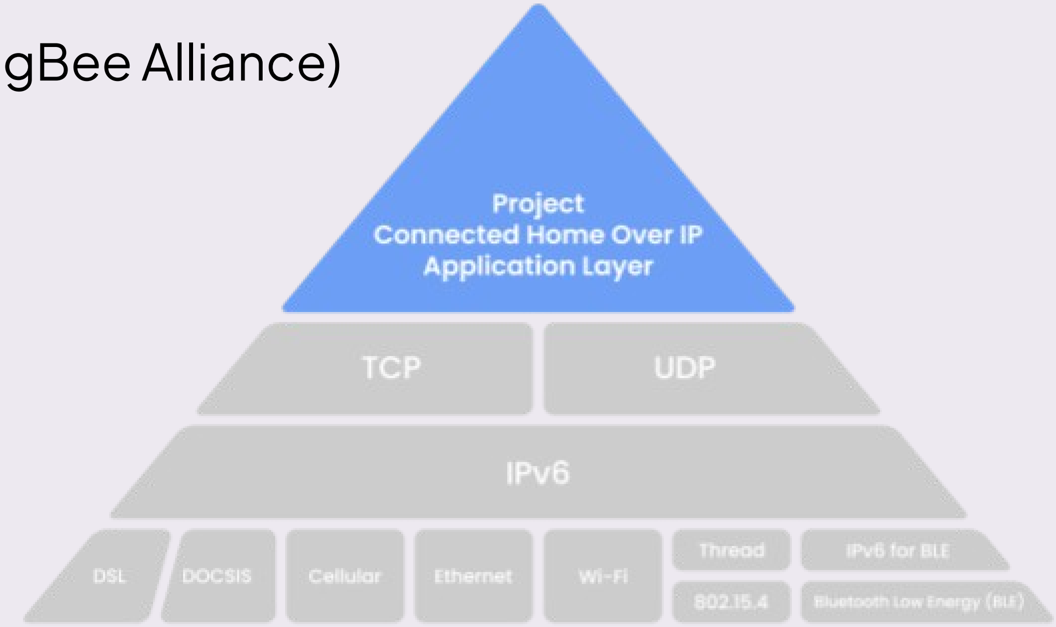
- Focus on **Open Source solution** that can be used within the Oniro project

Matter (former CHIP)

Matter

Github repositories (https://github.com/project-chip/connectedhomeip)

- Open Source SDK for Matter
- Governed by Connectivity Standards Alliance (former ZigBee Alliance)
- Application layer protocol running over IPv6 with Thread as its low-power option
- 1.0 specification and SDK released in October 2022
- Big players set out to work together on this
- Multi-admin to allow devices to connect to multiple platforms
- Opportunity for devices from small companies to work with the major platforms



Matter in Yocto/OpenEmbedded

- Matter SDK uses gn (generate ninja) and pigweed as build tooling
- Apache 2.0 licensed
- Core libraries of protocol implemenation as well as device examples
- Platform abstraction for Linux, other OSes and vendor SDKs
- Matter recipe with gn.bbclass in Oniro (https://gitlab.eclipse.org/eclipse/oniro-core/oniro/-/tree/kirkstone/meta-oniro-staging/recipes-connectivity/matter)
- Meta-matter layer from NXP (https://github.com/NXPmicro/meta-matter)

Matter in Zephyr

- PoC on matter module for Zephyr upstream is work in progress
- Matter had existing integrations for Nordic and Telink SDKs (based on Zephyr)
- Now also standalone Zephyr upstream platform support with CMake and KConfig build support

```
module.yml:
name: matter
build:
  cmake-ext: Ture
  kconfig: config/zephyr/Kconfig
  depends:
  - openthread
```

Matter Devices and Miscellaneous

Device Types:

- Device type examples available in SDK
- Based on ZigBee Cluster Library work
- Good base for own devices

Miscellaneous:

- QR code or setup PIN, NFC in the future as well
- Ethernet and Wi-Fi need no special handling
- Bluetooth Low-Energy for device onboarding only
- How to handle device functionalities not in the spec?

OpenThread Overview

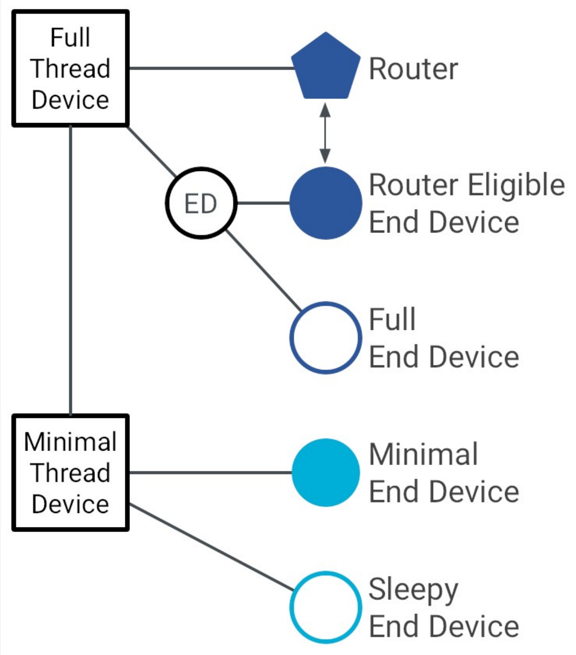
Overview

- Thread Group is the governance body
- Membership with fee, access to working groups and specification development
- Thread 1.1 specification available to the public (members have access to 1.3 already)
- Certification through the Thread Group

- OpenThread as Open Source project under BSD-3–Clause
- Very active and welcoming project (Google CLA for contributions needed)
- Driven by Google/Nest
- Established code base for commercial products

Device Types

Thread Primer documentation:
https://openthread.io/guides/thread-primer



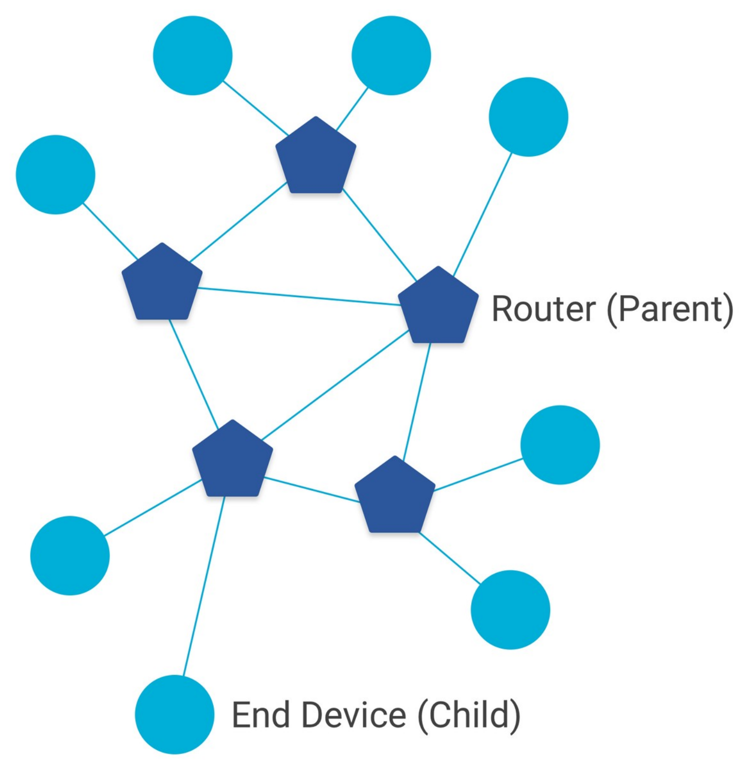
Full Thread Device

- Router: forwards packets, always-on transceiver
- Router Eligible End Device (REED): stand-by router, can get promoted
- Full End Device: end device only

Minimal Thread Device

- Minimal End Device:
- **Sleepy End Device**: transceiver off, wakes up periodically to send or poll messages from parent
- Synchronized Sleepy End Devices: synchronized schedule with parent (needs coordinated sampled listening from IEEE 802.15.4–2015)

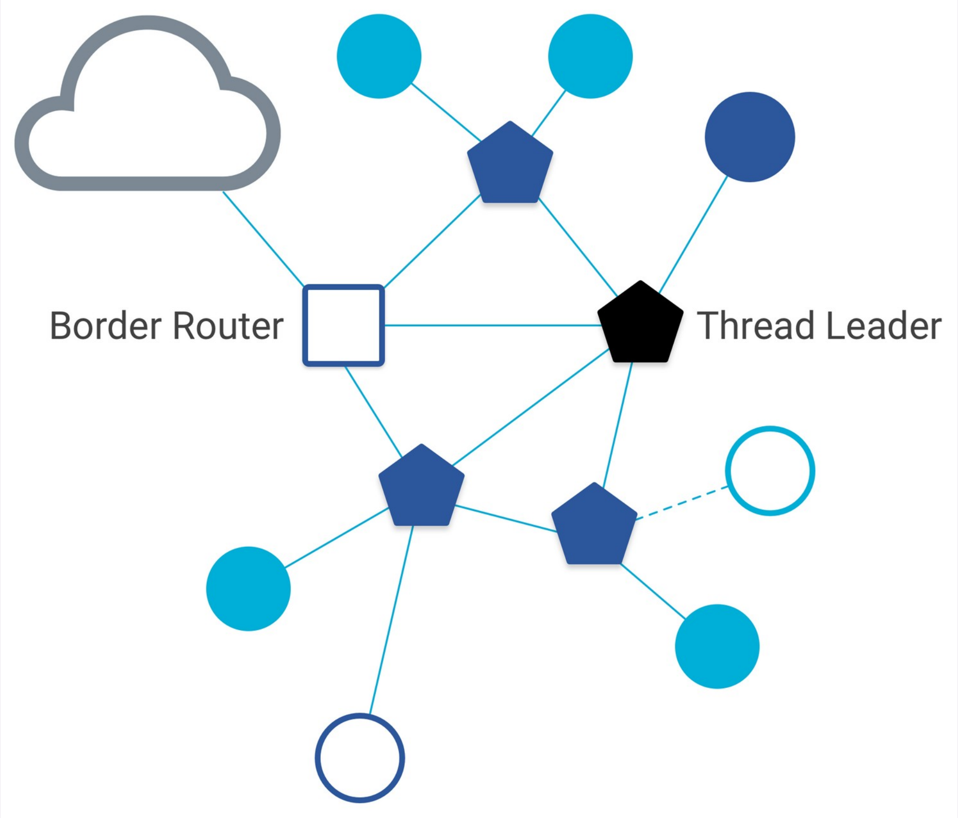
Device Roles



The two main different roles are:

- **Router**, which forwards packets and acts as a parent to child devices.
- **End Device**, communications with a parent, does not forward packets and can thus disable the radio when not in use.

- In addition the there a dynamically selected **Leader** of the Thread network, in charge of coordinating the routers.
- A **Border Router** has an additional non-Thread network where it forwards packets back and forth.



Thread Primer documentation:
https://openthread.io/guides/thread-primer

Addressing

- **Link-local**: all direct reachable addresses
- In a Thread network these use the fe80::/16 IPv6 prefix

- **Mesh-local**: all reachables addresses within the Thread mesh network
- These use the fd00::/8 IPv6 prefix

- **Global**: all reachable addresses, depending on your Border Router this can be the whole IPv6 internet
- In an isolated network the scope could be you campus, home or factory as well
- Thread also uses a Routing Locator (RLOC) to identify a network in the topology
- More details: https://openthread.io/guides/thread-primer/ipv6-addressing

OpenThread Components

Github repositories (https://github.com/openthread/)

- **openthread**: core implementation (with ot-daemon as native posix service)
- ot-{kw41z,ifx,samr21,nrf528xx,cc13x2-cc26x2,qorvo,efr32,nxp,cc2538,b91,esp32} to run on bare metal with vendor hal
- ot-rtos: FreeRTOS with lwIP
- **ot-br-posix**: OpenThread Border Router
- wpantund: network co-processor interface daemon (low maintenance)

- **Openthread module for Zephyr** (https://github.com/zephyrproject-rtos/openthread)

mDNS Discovery Proxy & Service Registration Protocol

Border Router

- Needed for Thread connectivity
- Unlike a vendor/platform specific hubs it will be integrated in other devices
- Border router functionality is already available in Apple homepods & TV, Google Nest, Amazon Echo & Eero and Nanoleaf devices
- Many existing hubs announced support as well

mDNS Discovery Proxy

- Specified in IETF RFC8766
- With Multicast DNS operating on link-local multicast packets it can be problematic for a Thread network
- Allowing multicats traffic into the Thread network (where its handled as broadcast) can bring down network performance and decrease device battery life
- For an efficient use the Discovery Proxy uses Multicast DNS to discover records on its link-local interface offer them over Unicast DNS.
- It would sit on the Border Router and help the Thread as well as the non-Thread network to discover devices and services

Service Registration Protocol

- Currently being worked on: draft-ietf-dnssd-srp
- An additional way of register a service with DNS Service Discovery
- Using the DNS unicast instead of mDNS and thus avoiding the multicast traffic
- Thread devices can register the services they offer with the available Border Routers
- In combination with mDNS Discovery Proxy, SRP allows for less broadcast and more unicast communication inside a Thread subnet

- Optimization, the IPv6 end-to-end paradigma still holds up
- Different to application level translations often used before with non-IP networks

Transparent Gateway Blueprint

Transparent Gateway Blueprint

- Oniro Project blueprint to design an IoT Gateway with modern technology
- Current demo with OpenThread and 6lowpan over IEEE 802.15.4
- Yocto/OE recipes for otbr and matter
- Zephyr application sample with OpenThread settings
- Turn-key solution with mobile application device onboarding

Thank you!

Join us @ oniroproject.org

Appendix

Options

Bare-metal	RTOS	Linux
Many vendors support this via a SDK/HAL approach	ot-rtos with FreeRTOS and LwIP OpenThread module for Zephyr	ot-daemon to run as normal router or node (RCP or NCP) OTBR to run as full border router Wpantund to run as node via NCP
SoC design	SoC design	Co-Processor (RCP, NCP) design
Sleepy End Device Minimal End Device (Full End Device)	Sleepy End Device Minimal End Device Full End Device Router Eligible End Device Router	Full End Device Router Eligible End Device Router Border Router
Battery powered	Battery powered /main powered	Main powered

Power Budget

Power Budget

THREAD GROUP

Typical average current measurements on existing IEEE 802.15.4 radios

The actual current consumption varies between different IEEE 802.15.4 radios and platforms. The measurements from an existing IEEE 802.15.4 platform are:

- Sleep: 1.6 µA
- Data Poll at 1 second period: 22 µA
- Data Transfer (36-byte UDP payload) at 1 second period: 37 µA

Estimated lifetime of existing IEEE 802.15.4 radios

Three components model the current consumption for a peripheral device:

- Average current for sleep: 1.6 µA
- Average current for reporting data every 60 seconds: (37 µA - 1.6 µA) / 60 seconds = 0.59 µA
- Average current for polling every 4 seconds: (22 µA - 1.6 µA) / 4 seconds = 5.1 µA

Summing the above components, the total average current is 7.29µA.

Assuming a device with a CR2032 battery and a typical energy capacity of 200mAh, the battery lifetime is 200mAh/(7.29µA/1,000)/(24 × 365) = 3.13 years.

Environmental sensor profile:

- Reports data every 60 seconds which is one or more hops away.
- Checks in with the parent every 4s for pending data (queued at parent)
- Application data (payload) is 36 bytes (enough for sensor data)

Excerpt taken from the ThreadGroup White Paper: The Value of Low Power

https://www.threadgroup.org/Portals/0/documents/support/TheValueofLowPowerWhitepaper_2454_2.pdf

IEEE 802.15.4 & Thread

- Reduced function devices (RFD) to allow very low power end devices
- e.g. IEEE 802.15.4 Data Request command to poll parent

  -> Received ACK has frame pending bit set
- Sleepy End Device (child) to parent relationship used in OpenThread
- Short addresses can be used to reduce frame size for transmit

NRF52840: radio current consumption:
https://infocenter.nordicsemi.com/pdf/nwp_039.pdf

TX power	Operation	Measurement
	Sleep current	3.74 µA
0 dBm	Data request	15.97 µC
	CoAP NON 57 B	27.96 µC
	CoAP NON 125 B	43.71 µC
	Data request + CoAP ACK	136.7 µC
8 dBm	Data request	26.37 µC
	CoAP NON 57 B	51.37 µC
	CoAP NON 125 B	90.97 µC
	Data request + CoAP ACK	170.03 µC

Table 7: Thread operations measured on nRF52840 v2.0.0 Development Kit

Wi-Fi & BLE

- Established solutions with Wi-Fi and BLE
- If they cover your use-cases there migth be no reason to switch

- Wi-Fi works well for devices with a bigger battery or main powered
- It allows to use existing infrastructure
- It allows for huge amounts of data being transferred

- BLE is prime for direct mobile to accessory connection
- Especially with existing Bluetooth profiles

6lowpan

History

- Initially only home-grown network protocols, TCP/IP was deemed to wasteful
- Interoperability only through proxy applications, or not at all
- Focus on vertical stacks and isolated sub-networks (no end-to-end)

- IP stacks for microcontroller, e.g. µIP, has been in development for a long time (2003)
- Focused on getting the network stack fit in memory, instead of an optimized frame size

- IEEE 802.15.4 standard released in 2003 as kick-off to explore IP on low-rate and low-power wireless
- IETF started from 2007 onwards to specify IPv6 encapsulation and header compression mechanisms

- 6lowpan started with IEEE 802.15.4 and was adopted for many more down the road: Bluetooth, NFC, PLC, etc

6lowpan in a nutshell

Encapsulation:

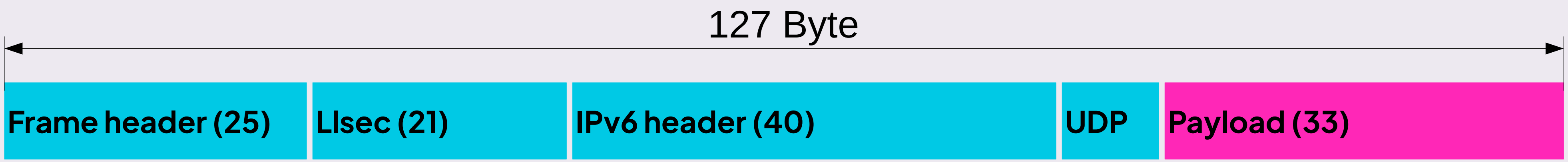
- The 6LoWPAN adaptation layer sits between data-link and the original network layer
- IPv6 allows for a maximum packet size of 1280 bytes
- Impossible to handle in the 127 byte MTU of IEEE 802.15.4
- Therefore 6lowpan brings back a fragmentation scheme
- But fragmentation can still lead to bad performance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

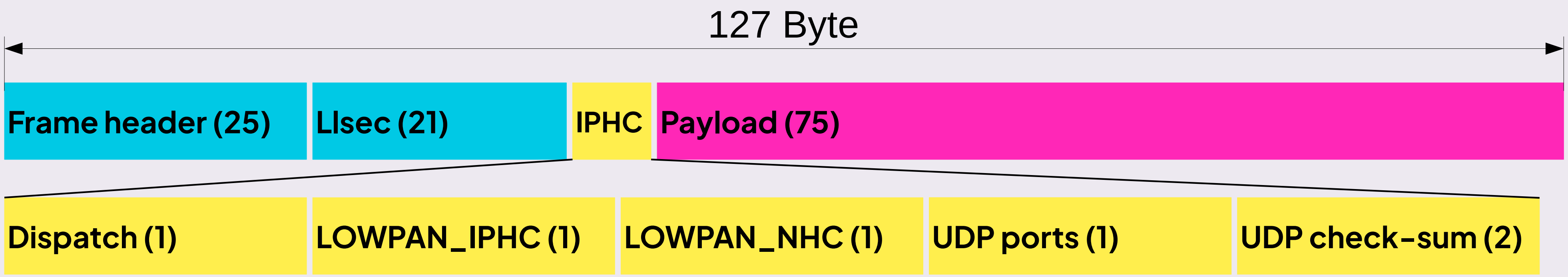
- Defining some default values in IPv6 header, leave out during transmit
- E.g. version = 6, traffic class & flow-label = 0, hop-limit only well-known values (1, 64 or 255)
- Remove the payload length (available in 6lowpan fragment header or data-link header)

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



6lowpan Misc Tips

- Design application protocols and payload to fit into frames and avoid fragmentation
- Enable reliability features in phy and mac layers (IEEE 802.15.4 allows ack/retransmit)
- **One lost frame in a bigger packet can have a serious performance impact**

- UDP with DTLS to avoid 3 way handshake latencies for TLS connections

- IETF also specifies RFC's for an application protocol (CoAP) to fit with 6lowpan