
Universidade Federal de Pernambuco

Centro de Informática

Graduação em Engenharia da Computação

# Análise comparativa de sistemas operacionais para Nordic nRF52840 para implementação na meta-plataforma de internet das coisas KNoT

Vítor Barros Aquino

Trabalho de Graduação

Recife, Pernambuco

Dezembro de 2019

Universidade Federal de Pernambuco
Centro de Informática

Vítor Barros

# Análise comparativa de sistemas operacionais para Nordic nRF52840 para implementação na meta-plataforma de internet das coisas KNoT

Trabalho apresentado ao Curso de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

**Orientador:** Prof. Dr. Kiev Gama

Recife, Pernambuco
Dezembro de 2019

*Dedico este trabalho a Deus e minhas famílias, a que eu ganhei e a que eu escolhi.*

# Agradecimentos

Primeiramente gostaria de agradecer a Deus, que sua historia e ensinamentos sempre serviu de exemplo para meu direcionamento.

Sou grato pelas minhas escolhas, pelos meus acertos e, principalmente, pelos meus erros. Esse trabalho conclui anos de bastante aprendizado, muito amadurecimento e muitas descobertas.

Agradeço a meus pais e meu irmão que sempre me apoiaram. Eles me deram liberdade, com responsabilidade, para que eu me descobrisse profissionalmente, para trilhar uma carreira próspera e, principalmente, feliz.

Queria agradecer a minha namorada, que foi a pessoa que mais me acolheu e motivou nesse último ano graduação. Gostaria de agradecer também a todos os meu amigos, que me ajudaram a formar o caráter e homem que sou hoje.

Já nessa caminhada conheci pessoas incríveis, que acreditaram no meu potencial. Quero agradecer ao Centro de Informática da UFPE, que por muitas vezes foi minha segunda casa. Quero agradecer a todos os professores que de alguma forma me ensinaram e me fizeram crescer como pessoa e como profissional. Quero agradecer a todos os meus colegas e amigos que fiz nessa jornada acadêmica, pelo apoio, incentivo e gargalhadas.

Por fim, não poderia deixar de agradecer o Centro de Estudos e Sistemas Avançados do Recife - CESAR. Empresa que me acolheu como bolsista e que me ensinou muito sobre a realidade do mercado de trabalho. Encontrei nela verdadeiros mestres que acreditaram no meu potencial, me incentivaram e me ensinaram tudo que podiam ensinar. Agradeço a todos os amigos que fiz, por me ensinarem o que é união dentro de uma empresa.

Sou grato por cada pessoa e acontecimento que tive em minha vida. Seja positivo ou negativo, bom ou ruim, feliz ou triste, tudo o que aconteceu na minha vida me trouxe a esse momento. E eu sou extremamente grato por estar onde estou.

*"Não importa o que as pessoas te dizem,*
*palavras e ideias podem mudar o mundo."*

—Prof. Keating (Sociedade dos poetas mortos)

# Resumo

Em Internet das Coisas (IoT) o objetivo se dá por entregar conectividade e inteligência para coisas antes não conectadas, agregando valor a solução. Na área existe um grande pluralidade de tecnologias que agregam valor em IoT, desde plataforma de Nuvem, de software embarcado e hardware. É nessa diversidade que o KNoT atua, visando integrar diferentes plataformas para que todas se comuniquem de forma facilitada. O KNoT, por sua natureza, é uma meta-plataforma de código livre, fim-a-fim e multi protocolo, agregando, por exemplo, diferentes tecnologias de conectividade. Sua arquitetura primária é formada por quatro grandes áreas: dispositivo, gateway, nuvem e aplicação. No escopo do dispositivo, um novo hardware passou a ser utilizado na meta-plataforma KNoT, o chip da Nordic, nRF52840, em conjunto com o protocolo de comunicação Thread. Porém dada a pluralidade de sistemas operacionais para internet das coisas disponíveis no mercado, um desafio é decidir qual o melhor para ser usado em determinado caso de uso, levando em conta a motivação do KNoT e as características do nRF52840 e do protocolo Thread. Este trabalho realiza uma análise comparativa dos sistemas operacionais Zephyr e RIOT OS, selecionados a partir dos requisitos do projeto, identificando a aplicabilidade deles para o KNoT.

**Palavras-chave:** IoT, Sistemas Embarcados, Sistema Operacional, KNoT, nRF52840, Thread.

# Abstract

In Internet of Things (IoT) the goal is to deliver connectivity and intelligence to previously unconnected things, adding value to the solution. In the area there is a wide range of technologies that add value to the IoT solution, such as cloud platform, embedded software and hardware. It is in this complexity that KNoT operates, aiming to integrate different platforms so that all communicate easily. KNoT is an open source, end-to-end and multi-protocol meta-platform, bringing together different connectivity technologies and platforms. Its primary architecture is made up of four major areas: device, gateway, cloud, and application. Within the scope of the device, new hardware has been used on the KNoT meta-platform, Nordic's nRF52840 chip, in conjunction with the Thread communication protocol. However, given the plurality of IoT operating systems available on the market, one challenge is deciding which is best to use in a given use case, given the motivation of KNoT and the characteristics of nRF52840 and the Thread protocol. This paper performs a comparative analysis of the Zephyr and RIOT operating systems, selected from the project requirements, identifying their applicability to KNoT.

**Keywords:** IoT, Embedded System, Operating System, KNoT, nRF52840, Thread.

# Lista de Tabelas

# Lista de Figuras

# Sumário

# Capítulo 1

# Introdução

## 1.1 Introdução

O conceito de Internet das Coisas (IoT) não possui uma homogeneidade na literatura. É possível observar que diferentes conceitos e visões são expostos por diferentes pesquisadores [1]. De um modo agregador, podemos definir Internet das coisas (IoT) como a missão de conectar e dar inteligência a dispositivos antes não conectados, agregando valor a solução.

A Microsoft publicou uma cartilha de oportunidades em IoT [2] escrita por um parceiro, a Solliance. Nesta cartilha é apontado que IoT é um mercado em expansão que está crescendo em ritmo acelerado, onde se espera que o gasto em IoT com produtos e serviços cresça de 201 bilhões de dólares em 2018 para mais de 500 bilhões de dólares em 2023, tendo empresas que estimam uma projeção ainda mais ousada, de 13 trilhões de dólares em gastos com IoT para 2030.

O estudo aponta que das 743 empresas participantes, apenas 16% possuem soluções em produção e 69% estão explorando as possibilidades em IoT. O que mostra que IoT ainda é um conceito recente que nos leva a prestar atenção às barreiras para o desenvolvimento de produtos e serviços em IoT.

A introdução de novas tecnologias de IoT e as tecnologias existentes trazem uma extensa complexidade, trazendo a tona um dos grandes problemas, a interoperabilidade [3]. Ou seja, o poder de fazer as diferentes tecnologias funcionarem em conjunto. Atacar esse desafio vai ser um dos elementos chaves para que 40% do valor potencial da IoT seja gerado [4]. Sendo essa uma das três barreiras, junto com segurança e retorno do investimento, mostradas também no estudo da Microsoft.

Sendo assim, como principal objetivo para ser atingido, o KNoT Network of Thing (KNoT) [5] surge, para atacar o problema da interoperabilidade. Uma meta-plataforma que serve como uma cola entre protocolos e plataformas, para fazerem todos funcionarem em conjunto. O KNoT é uma plataforma fim a fim, que junta hardware e software. O KNoT também é multiprotocolo, em todas as camadas de comunicação, permitindo que cada aplicação selecione a pilha de comunicação que melhor se adapte às suas necessidades. É formado por quatro componentes principais: Thing (devices), Gateway, Cloud e Protocolo KNoT.

Em sua primeira versão, o KNoT utilizou a plataforma de hardware Arduino [6] para implementar os dispositivos (Things). Com o desenvolvimento da plataforma e a necessidade de abraçar novas tecnologias de conectividade e aumentar a gama de soluções, principalmente as que necessitam de mais processamento, a plataforma se propôs a implementar uma solução de hardware, para o Thing, mais robusta, baseada no System on Chip (SoC), da Nordic, nRF52840.

O SoC nRF52840 [7] é equipado com um microprocessador com maior capacidade de processamento, memória flash e memória RAM que o Arduino Pro Mini utilizado na primeira versão do KNoT. É multi protocolo, suportando diversas tecnologias de conectividade. Além de possuir uma família de placas para desenvolvimento e empresas terceiras que produzem módulos com esse chip.

Sendo assim, um novo sistema de software se faz necessário, para atender as características do SoC escolhido e os aspectos da plataforma KNoT. Um desafio por causa da quantidade de sistemas operacionais para sistemas embarcados presentes no mercado, com diferentes características.

## 1.2 Objetivo

O principal objetivo deste trabalho é fazer uma análise comparativa entre os principais sistemas operacionais para sistemas embarcados do mercado, tendo como base as características e limitações da meta plataforma KNoT e do SoC nRF52840.

Como objetivos específicos espera-se:

- Realizar o reconhecimento das principais características do KNoT e do SoC nRF52840;
- Fazer uma análise comparativa das características de sistemas operacionais para sistemas embarcados que se adequem às características da plataforma KNoT e do SoC nRF52840;

- Apontar qual sistema operacional mais adequado para determinado caso de uso.

## 1.3 Estrutura do Trabalho

O presente trabalho será estruturado da seguinte forma:

- Capítulo 2: introduzir conceitos e as principais características da meta plataforma KNoT, da plataforma de hardware nRF52840 e do protocolo Thread. Além disso, realizar uma pesquisa na literatura acerca do tema.
- Capítulo 3: análise comparativa dos sistemas operacionais.
- Capítulo 4: conclusão do trabalho fazendo um apanhado da análise e propondo trabalhos futuros.

# Capítulo 2

# Contextualização

## 2.1 KNoT Network Of Things

O KNoT Network of Things é uma meta plataforma de internet das coisas, desenvolvido pelo Centro de Estudos e Sistemas Avançados do Recife - CESAR. É considerada uma meta plataforma por não ser mais uma plataforma, mas sim uma integradora de plataformas já existentes. Ou seja, se propõe em ser uma plataforma que permite a interoperabilidade entre já existentes plataformas de IoT, seja ela responsável por gerenciamento de nuvem ou plataforma de hardware.

### 2.1.1 Principais características

Suas principais características são:

- Código aberto: Todo o código do KNoT, do hardware e software, é aberto e está hospedado e disponível no GitHub [8]. Sendo assim, o projeto é gratuito para baixar e usar até mesmo comercialmente, além de ser aberta a contribuições da comunidade.

- Fim-a-fim: O KNoT implementa uma solução de integração fim-a-fim, ou seja, hardware e software. Do hardware do dispositivo até o software da Cloud.

- Multi-protocolos: O KNoT trabalha com diversidades nas camadas física, de link de dados, rede e transporte, permitindo que cada aplicação selecione a pilha de rede que melhor atenda às suas necessidades.

- Modelo de dados semânticos: O KNoT também define um modelo de dados semântico

na camada de aplicação. Isso significa que todas as aplicações na plataforma conhecem o significado dos dados uns dos outros, permitindo que eles compartilhem facilmente os dados.

- Nuvem distribuída: O KNoT implementa o conceito de Fog, onde uma instância reduzida e local da Cloud recebe os dados dos dispositivos que estão no mesmo local e os sincronizam de maneira hierárquica, permitindo conexões com milhões de dispositivos.

- Compartilhamento em tempo e espaço: O modelo de compartilhamento de dados para a plataforma KNoT permite que os usuários compartilhem seus dados por tempo (você pode acessar dados apenas às segundas-feiras, 14h por exemplo) e espaço (você pode acessar os dados somente se estiver acessando-os de um local específico, por exemplo).

- Baixo custo: O KNoT é desenvolvido para custar pouco e ser produzido em massa. O objetivo é produzir os dispositivos com um custo final menor de 5 dólares.

### 2.1.2 Principais módulos

As quatro principais partes da implementação são:

- KNoT Protocol: É um protocolo binário, leve, que define diversas mensagens para executar sua operação.

- KNoT Thing: São os dispositivos da borda, formado fundamentalmente por microcontrolador e rádio, ligados a uma fonte de alimentação, e possivelmente sensores e/ou atuadores. O hardware é controlado por um sistema operacional responsável pelo controle dos sensores e/ou atuadores, implementação do KNoT Protocol, além da lógica da aplicação.

- KNoT Cloud: É um serviço que recebe os dados e os deixa disponível para uso da aplicação. Como o objetivo é integrar diversas outras plataformas, o KNoT prevê suportar diversas Clouds diferentes

- KNoT Gateway [9]: Um gateway IoT é um dispositivo intermediário utilizado para conectar um dispositivo que não fala IP a internet, ou seja, no caso conectar o KNoT Thing a KNoT Cloud. Sendo assim é responsável por traduzir o protocolo KNoT utilizado para comunicação com o KNoT Thing para qualquer que seja o protocolo utilizado pela KNoT Cloud, além de gerenciar esses dispositivos conectados.

Esses módulos podem ser observados na Arquitetura KNoT 1.0, que pode ser vista na Figura 2.1, retirada do website do KNoT. O estudo de arquiteturas em IoT foi base para que o chefe de pesquisa em IoT do CESAR, Tiago Barros, junto com autoridades nacionais, encaminhasse uma recomendação para um padrão de referência de arquitetura de dispositivos IoT para o setor de padronização em telecomunicações da União Internacional de Telecomunicações (ITU-T). Essa recomendação foi publicada em julho de 2019 com o nome ITU-T Y.4460 [10].

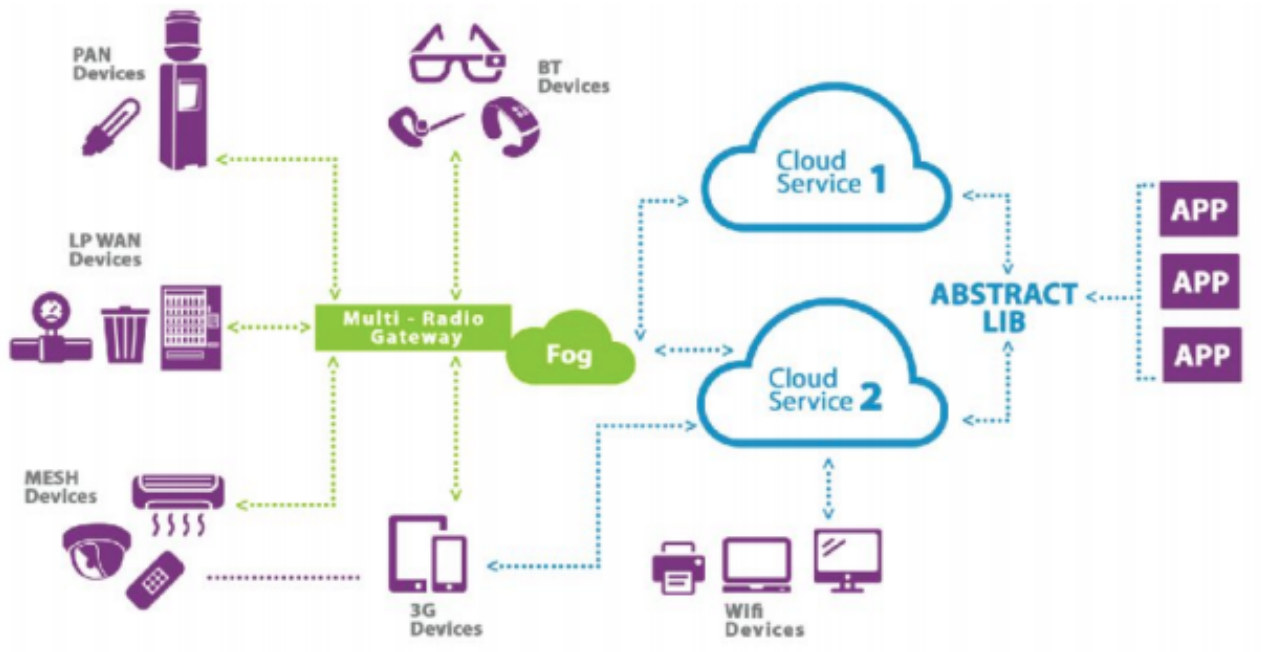


Figura 2.1: Arquitetura KNoT 1.0

### 2.1.3 Arquitetura de dispositivos IoT

A recomendação ITU-T Y.4460 [10] descreve um modelo arquitetural para dispositivos em IoT com base no poder de processamento e na capacidade de comunicação.

A capacidade de processamento define como os dispositivos podem executar tarefas computacionais e executar algoritmos. Sendo possível classificar um dispositivo como:

- Sem capacidade de processamento: São dispositivos sem poder de processamento, considerados dispositivos passivos.

- Baixa capacidade de processamento: São dispositivos que têm capacidade de processamento apenas suficiente para ler e escrever dados de/para sensores e/ou atuadores e enviar e receber esses dados como mensagens para aplicativos de IoT. Eles não têm capacidade de processamento suficiente para tomar decisões ou executar algoritmos complexos.

- Alta capacidade de processamento: São dispositivos que possuem capacidade de processamento suficiente para tomar decisões ou executar algoritmos complexos, como algoritmos de inteligência artificial e aprendizagem de máquina.

A capacidade de comunicação define como os dispositivos podem se conectar às redes de comunicação. Como a comunicação é um recurso obrigatório para um dispositivo IoT, é possível classificar um dispositivo como tendo:

- Baixa capacidade de conectividade: Aquele que não se conecta diretamente com a rede de comunicação.
- Alta capacidade de conectividade: Aquele que se conecta diretamente com a rede de comunicação.

Como essas duas definições foram elaboradas três arquiteturas para três tipos de dispositivos diferentes, que são:

### Arquitetura para dispositivo de baixo processamento e baixa conectividade (LPLC)

Nesse caso, os dispositivos atuam basicamente como coletores de informações ou atuadores no meio. Eles não possuem conectividade direta com a internet e não possuem recurso de processamento suficiente para tomar decisões e executar algoritmos, por isso se faz necessário a adição de um componente na arquitetura que é responsável por fazer a comunicação entre o dispositivo e a internet. Esse dispositivo intermediário é chamado de gateway.

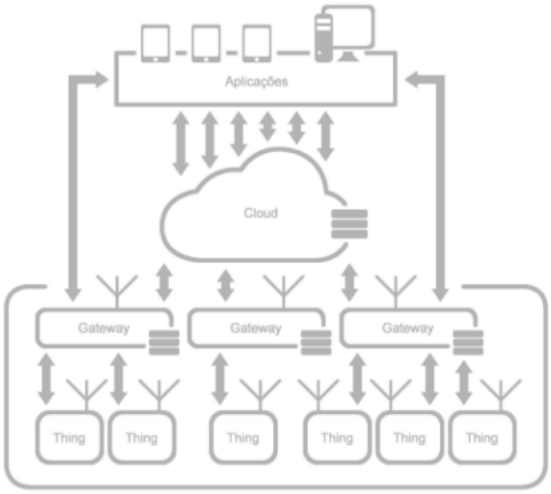


Figura 2.2: Arquitetura LPLC: Thing depende de um gateway para se comunicar com a Cloud

**Arquitetura para dispositivo de baixo processamento e alta conectividade (LPHC)**

Os dispositivos LPHC já possuem a possibilidade de se comunicar diretamente com a internet, logo não necessitam do dispositivo intermediário, o gateway.

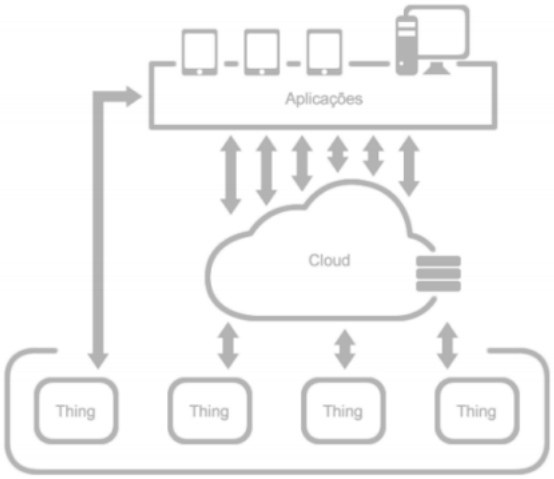


Figura 2.3: Arquitetura LPHC: Thing não depende de um gateway, pode se comunicar diretamente com a Cloud

**Arquitetura para dispositivo de alto processamento e alta conectividade (HPHC)**

Os dispositivos HPHC adicionam o alto poder de processamento, quando comparado com os dispositivos LPHC. Esses dispositivos são autônomos; eles tomam decisões sobre suas próprias funções e também podem coordenar outros dispositivos.

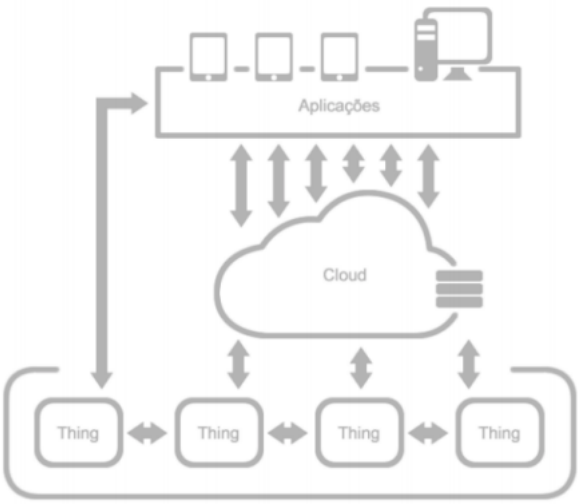


Figura 2.4: Arquitetura HPHC: Thing não depende de um gateway e possui inteligencia para controlar/trocar dados com outros Things diretamente

### 2.1.4 Objetivos Futuros

O KNoT, em sua primeira versão, implementa a arquitetura para dispositivo de baixo processamento e baixa conectividade (LPLC). O hardware do dispositivo nessa primeira versão

é formado por um Arduino Pro Mini e o rádio nRF24l01.

Em seu mapa futuro de desenvolvimento, a próxima versão do KNoT se propõe a ser uma transição para a arquitetura 2.0, referente a arquitetura para dispositivo de baixo processamento e alta conectividade (LPHC).

Para o inicio da transição foi definido a escolha de um novo conjunto de hardware e software para o dispositivo de borda, o KNoT Thing, com a adição de uma nova tecnologia de rádio.

A nova tecnologia de rádio definida foi a rede Thread [11]. Thread é um protocolo de rede baseado em IPv6, 6LoWPAN, projetado para dispositivos IoT de baixa potência em uma rede mesh sem fio, comumente chamada de WPAN (Rede de área pessoal sem fio). Utiliza a pilha da camada física e de enlace da IEEE 802.15.4.

Para o hardware do dispositivo, foi definido a utilização do System on Chip nRF52840 da Nordic Semiconductor.

## 2.2 nRF52840

O nRF52840 [7][12] é um chip da familia nRF52 da Nordic Semiconductor. Conta com um ARM® Cortex®-M4 32-bit processador com FPU, 64 MHz, 1 MB de memória Flash e 256 kB de memória RAM. Possui ARM® TrustZone® Cryptocell 310 que é subsistema de segurança que fornece root de confiança (RoT) e serviços de criptografia para um dispositivo, além do mecanismo de Padrão de Criptografia Avançada (AES). Na área de conectividade possui suporte de protocolo para Bluetooth 5, Bluetooth mesh, Thread, Zigbee, 802.15.4, ANT e pilhas proprietárias de 2.4 GHz.

Para o desenvolvimento será utilizado os modelos de referencia para desenvolvimento da própria Nordic, que são a nRF52840 Devenlopment Kit (DK) e a nRF52840 Dongle. Elas podem ser vistas nas Figuras 2.5 e 2.6, respectivamente.

A nRF52840 Dongle é menor e cerca de 5 vezes mais barata, alimentada e programada via USB. A DK possui diversos recursos para prototipação e testes, possui antena NFC, encaixe para bateria e o depurador SEGGER J-Link.

## 2.3 Thread

O protocolo Thread é mantido pelo Thread Group [13], organização formada por centenas de empresas com o objetivo de criar a melhor maneira de conectar e controlar produtos em

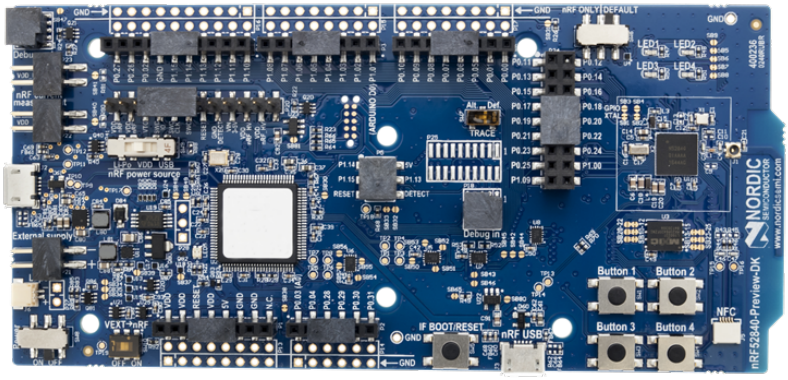


Figura 2.5: nRF52840 Development Kit

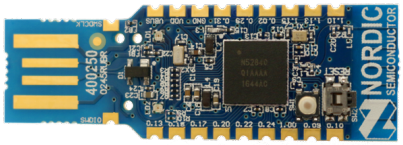


Figura 2.6: nRF52840 Dongle

residências e edifícios.

O Thread [11] é um protocolo de rede baseado em IPv6 projetado para dispositivos Internet of Things de baixa potência em uma rede em malha sem fio IEEE 802.15.4. O protocolo busca ser simples, seguro (autenticação e criptografia), confiável, eficiente e escalável. Como pode ser visto na Figura 2.7, utiliza as camadas física e de enlace do IEEE 802.15.4 e implementa da camada de rede até apresentação, seguindo como referencia as camadas do Modelo OSI.

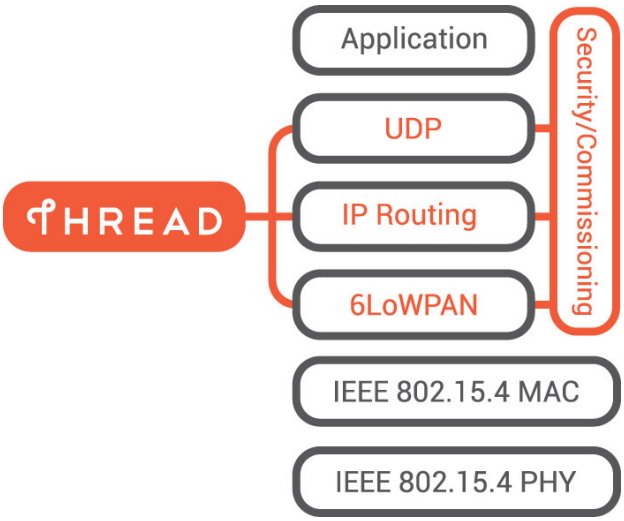


Figura 2.7: Pilha de implementação do protocolo Thread

OpenThread [11] é uma implementação de código aberto lançada pela Google Nest. Esse

projeto implementa todas as camadas e funcionalidades de uma rede Thread, de acordo com a especificação Thread 1.1.1. Essa é a principal biblioteca suportada pelos sistemas operacionais de código livre.

## 2.4 Sistemas operacionais para IoT

Sistema operacional (OS) é um software responsável por gerenciar os recursos do sistema e prover abstrações para a execução das aplicações.

Os trabalhos na literatura [14][15][16][17][18][19] buscam definir as principais características que sistemas operacionais precisam possuir para atender ao ambiente de internet das coisas e ajudar a selecionar o mais apropriado sistema para cada aplicação. A seguir é listado uma intercessão das principais características que aparecem nos trabalhos da literatura.

- Footprint: os dispositivos IoT possuem limitações rm relação a memória, energia e processamento. Logo, se faz necessário que o sistema operacional seja capaz de funcionar com tais limitações.

- Escalabilidade: o sistema operacional deve ser escalável, ou seja, deve ser capacitado a rodar em diferentes arquiteturas e no máximo de dispositivos possível.

- Portabilidade: o sistema operacional deve ser capaz de funcionar em diferentes plataformas de hardware, ou possuir um mecanismo que torne fácil adicionar suporte a uma nova plataforma.

- Modularidade: o sistema operacional deve ser modular, ou seja, permitir que funcionalidades, bibliotecas, protocolos, entre outros serviços, sejam adicionais sob demanda, quando se fazer necessário.

- Conectividade: A conectividade de rede é essencial para a Internet das Coisas. Então é importante o sistema operacional suportar diferentes protocolos de conectividade, como Ethernet, Wi-Fi, BLE, IEEE 802.15.4, 6LoWPAN, entre outros.

- Segurança: os dispositivos IoT estão conectados a internet e/ou em uma rede de sensores e atuadores. Muitas vezes possuem funcionalidades criticas, que demanda recursos que garantam a segurança da aplicação e dos cumprimentos dos requisitos da aplicação.

Desse modo, se faz necessário que o sistema operacional possua recursos, como bibliotecas criptográficas e protocolos de segurança.

- Confiabilidade: para determinadas aplicações os dispositivos IoT possuem funcionalidades críticas, sendo necessário que o sistema operacional ateste sua capacidade de cumprir com os requisitos e tenha uma execução confiável. Geralmente os sistemas passam um certificações para se declarar aptos a executar a aplicação.

Não existe uma característica mais importante que outra. Para cada finalidade de uso do sistema operacional, alguma característica pode ser mais importante que a outra.

## 2.5 Trabalhos relacionados

Na literatura é possível encontrar alguns trabalhos relacionados que buscam comparar ou listar características de sistemas operacionais para o ambiente de internet das coisas.

Dong et al. [14], em 2010, fizeram um trabalho de pesquisa com o objetivo de identificar as características e desafios para o uso de sistemas operacionais em redes de sensores sem fio.

Aleksandar et al. [15] apresentam em seu trabalho critérios importantes para a seleção de um sistema operacional que possa ser usado em uma plataformas IoT, apresentando um survey de sistemas operacionais para IoT.

Borgohain et al. [16] seguem a mesma linha e mostram um survey de sistemas operacionais, mostrando quais protocolos os OS implementam e para quais plataformas os OS possuem portabilidade.

Gaur et al. [17] se aprofundaram mais na características técnicas, definindo e justificando as principais características que um sistema operacional para IoT deveria ter. Realizaram uma pesquisa com treze sistemas operacionais analisando-os em relação as características que definiram. Por fim, propuseram uma arquitetura genérica para um sistema operacional para IoT.

Os trabalhos encontrados na literatura possuem um viés técnico e pouco experimental, focando na comparação com base nas características gerais dos sistemas. Nenhum trabalho dos listados demonstrou uma experiência prática com o sistema operacional, além de não possuírem um critério inicial de escolha dos sistemas a serem comparados, como por exemplo um OS para um chipset especifico.

# Capítulo 3

# Análise Comparativa

## 3.1 Metodologia

Para atingir os objetivos do trabalho, o estudo foi feito em cinco etapas. Essas etapas podem ser vistas no diagrama da Figura 3.1.

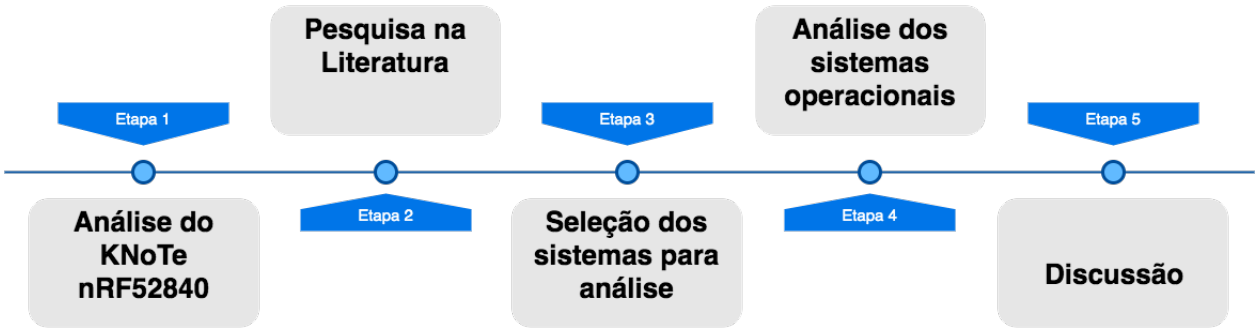


Figura 3.1: Metodologia

Na primeira etapa foi feito uma análise completa dos requisitos dados, ou seja, da meta-plataforma KNoT Network of Things e do SoC nRF52840. Essa análise foi feita para identificar as principais características e definições dos sistemas utilizados, para ser feita uma seleção e análise direcionada dos sistemas operacionais.

Em um segundo momento foi realizado uma pesquisa na literatura por trabalhos semelhantes, que realizavam analise de sistemas operacionais para internet das coisas ou que definiam características desejáveis em sistemas operacionais. Essa pesquisa é importante para saber quais aspectos são de principal interesse no momento de se analisar um sistemas operacional para IoT e para identificar o que é mais pesquisado a cerca do tema, para saber os principais avanços e dificuldades encontradas.

Na terceira etapa foi realizado uma seleção de quais sistemas operacionais entrariam em uma análise mais detalhada, levando em consideração a análise das características do KNoT e do SoC nRF52840.

Posteriormente, em um quarto momento, os sistemas selecionados foram analisados e testados. A análise é para identificar as características gerais, o suporte ao usuário e como é a comunidade do sistema operacional. O objetivo do experimento prático é buscar analisar a viabilidade de portar o KNoT para os sistemas operacionais e assim identificar possíveis problemas para a implementação, e, principalmente, identificar como é a experiencia do usuário ao utilizar o sistema operacional, podendo identificar questões em relação, por exemplo, a facilidade de configuração inicial e qualidade da documentação.

Por fim, na quinta etapa, é possível discutir para qual caso de uso cada sistema poderia ser usado de forma mais eficiente.

## 3.2 Seleção dos sistemas operacionais

Para o ambiente de sistemas embarcado para internet das coisas existem dezenas de sistemas operacionais, diferentemente da quantidade de sistemas operacionais para computadores convencionais, que podemos listar uma hegemonia do Windows, Linux e macOS.

A primeira categorização que pode ser feita é por sistemas que são de código livre ou não. Na lista de sistemas proprietários [18] podemos encontrar em destaque: Android Things, Windows 10 IoT, WindRiver VxWorks, Micrium C/OS, Micro Digital SMX RTOS, MicroEJ OS, Express Logic ThreadX, TI RTOS, Freescale MQX, Mentor Graphics Nucleus RTOS, Green Hills Integrity, Particle. E para sistemas de código livre, os principais são: TinyOS, RIOT OS, Contiki, Mantis OS, Nano RK, LiteOS, Amazon FreeRTOS, Apache Mynewt, Zephyr, Ubuntu Core 16 (Snappy), ARM Mbed, Yocto, Raspbian.

Como o KNoT é um projeto de código aberto, o sistema operacional deve ser de código aberto também. Sendo assim, só iremos considerar a lista de OS de código livre para essa análise.

Outro principal requisito para o objetivo do trabalho é que a sistema vai ser executado no SoC nRF52840, logo o sistema deve ter suporte a essa plataforma. Dessa forma chegamos a uma lista de quatro sistemas: RIOT OS, Amazon FreeRTOS, Zephyr e ARM Mbed OS.

Amazon FreeRTOS [20] é uma extensão do sistema FreeRTOS com bibliotecas para conec-

tividade, segurança e atualizações remotas. O FreeRTOS [21] é um sistema com arquitetura microkernel, com camada de rede especifica, suporte a diversas arquiteturas de plataformas, como AVR e ARM, com escalonador cooperativo e preemptivo e suporte a sistema de tempo real. Em 2018 era, depois do Linux e do Windows, o sistema operacional mais utilizado [22]. A Amazon adicionou bibliotecas para atualização remota, Bluetooth LE, interoperabilidade com POSIX, TLS, Wi-Fi, pilha TCP/IP e bibliotecas para integração com serviços da AWS. Apesar de ser promissor e suportar a placa nRF52840, o Amazon FreeRTOS não possui suporte ao protocolo de comunicação Thread, que é um dos requisitos traçados pelo KNoT.

O Mbed OS [23] faz parte do ecossistema para desenvolvimento de dispositivos IoT da ARM Mbed, da empresa britânica ARM. Esse ecossistema é formado por:

- Mbed OS: sistema operacional para dispositivos;
- Mbed Linux OS: sistema operacional baseado em Linux para gateways e/ou dispositivos IoT complexos;
- Mbed Tools: ferramentas gratuitas de desktop, navegador e CLI, para auxiliar o desenvolvimento;
- Mbed TLS: biblioteca de segurança para dispositivos embarcados;
- Mbed HDK: Projetos de referência, esquemas e layouts de placas para desenvolver hardware de produção;
- Pelion IoT Platform: plataforma IoT, do dispositivo ao consumo dos dados para aplicação.

O sistema possui porte para a placa nRF52840, porém não possui uma implementação estável do protocolo de comunicação Thread. Além disso o Mbed OS só funciona em plataformas que possuem arquitetura ARM, o que não ajudaria, no futuro, o KNoT integrar o mesmo sistema para outras arquiteturas, dificultando arquivar o problema da interoperabilidade.

Dessa forma, sobram o Zephyr e o RIOT OS. Eles serão analisados de forma mais detalhada, pois ambos atendem os requisitos do KNoT e do SoC nRF52840.

## 3.3 Zephyr

O Zephyr [24] é um projeto da Linux Fundation, organização que hospeda grandes projetos de código aberto, como o Linux. Foi lançado em 2016 e tem por trás dele grandes empresas

como Intel, NXP e a fabricante do nRF52840, Nordic.

### 3.3.1 Características gerais

O Zephyr [25] é um sistema operacional nanokernel ou microkernel. É multithread com diferentes algoritmos de escalonamento, preemptivo ou não-preemptivo, Earliest Deadline First (EDF), requisição de interrupção e Timeslicing em threads preemptivas.

Ele possui serviço para gerenciamento de energia como o modo tickless idle. Possui um serviço de alocação de memória, uma API compatível com POSIX pthreads e sistemas de sincronização com semáforos e mutex, além de um serviço para passagem de mensagem por filas ou streams de dados.

O Zephyr implementa uma pilha de comunicação completa e otimizada, com suporte aos protocolos Bluetooth, Bluetooth LE, WiFi, 802.15.4, 6LoWPAN, CoAP, IPv4, IPv6 e NFC. Tem suporte ao OpenThread, implementação em código aberto do protocolo Thread para redes mesh. Possui um Framework completo para logs e gerenciamento para armazenamento não volátil.

É projetado para consumir pouca memoria, necessita de em média 2 kB a 8 kB de RAM (Random Access Memory) e em média 50 kB de ROM (Read-only Memory).

A arquitetura do Zephyr é modular, permitindo que a aplicação só incorpore no sistema aquilo que será utilizado. A arquitetura geral do Zephyr pode ser vista na Figura 3.2, que é divulgada na documentação.

A arquitetura é dividida em três partes. A primeira parte é o Kernel, onde se encontra as abstrações para os diferentes hardwares, o gerenciador de energia e os serviços do kernel, além do escalonador de tarefas. A segunda parte é onde se encontra os serviços do sistema operacional, onde se encontra drivers de barramentos de comunicação, toda a implementação e logica dos protocolos e serviços de redes e o gerenciador de dispositivos. Por ultimo tem a camada de serviços de aplicação que possui bibliotecas de segurança, serviços de gerenciamento de filas e protocolos da camada de aplicação.

O suporte as placas é bastante extenso. O sistema possui porte para mais de 170 placas, das arquiteturas ARM, ARC, x86, RISCV, XTENSA e NIOS II. O Zephyr possui suporte as duas placas mostradas nesse trabalho, a nordic nRF52840 Development Kit e a Dongle.

O projeto também possui um grande foco em segurança, com extensa documentação para garantir as certificações do sistema.

O Zephyr é licenciado permissivamente usando a licença Apache 2.0. Podendo ter alguns

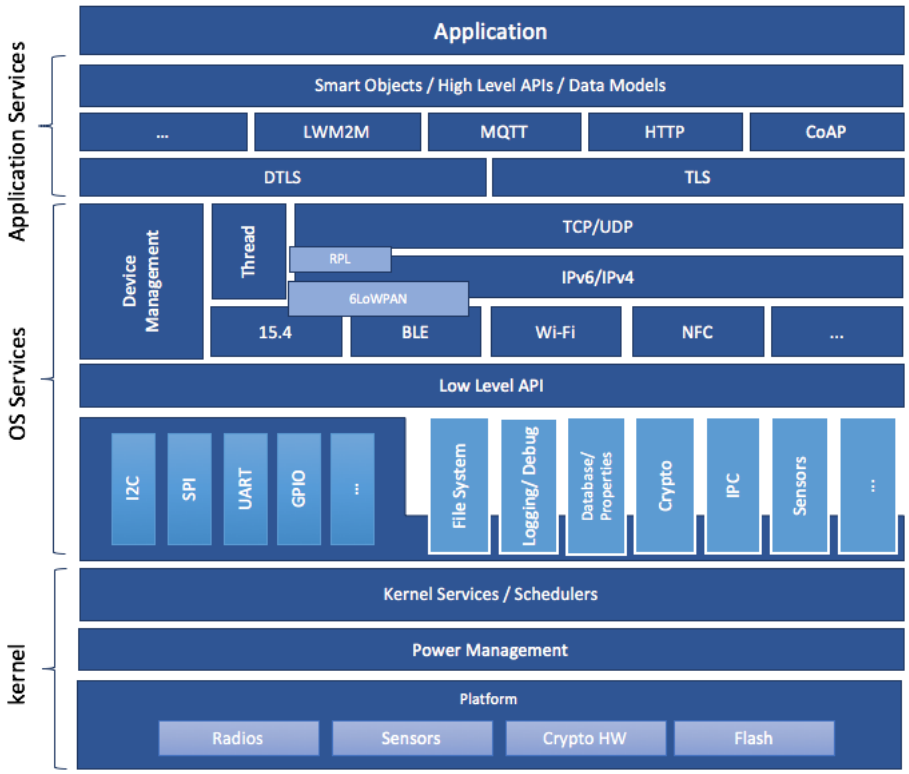


Figura 3.2: Arquitetura Zephyr

componentes que possuem outra licença. Dessa forma permite que o código do sistema seja utilizado também em projetos proprietários de código fechado.

### 3.3.2 Suporte e Comunidade

O Zephyr permite o desenvolvimento completo no Linux, macOS e Windows. Possui um porte nativo POSIX, facilitando o desenvolvimento de aplicações, pois o pode ser desenvolvido e testado em ambiente Linux. O sistema mostra ser amigável para facilitar o desenvolvimento, como, por exemplo, ter uma interface Shell para configuração completa do sistema. Ele é implementado em C e C++, assim como seus programas também podem ser desenvolvidos nessas linguagens. Para auxilio diversos exemplos e testes estão disponíveis no repositório do projeto.

Todo o código do Zephyr está hospedado no GitHub [26]. Até novembro de 2019 foram mais de 33 mil commits, 523 autores e 46 repositórios. Nessa mesma época, possuía mais de 6600 tarefas (issues) fechadas e 1000 abertas.

O Zephyr dispõe na documentação um guia para contribuição. Nele está exposto todo o necessário para um programador fazer contribuição para o projeto, como o layout do repositório,

onde se encontra as tarefas em aberto, como funciona a integração continua do repositório e possui o fluxo completo para realizar uma contribuição. O projeto possui um Coding Style bem definido e um checkpatch para checar se o código está seguindo o padrão, assim como possui um guia de formatação das mensagens de commit e dos Pull Requests. Todo commit passa por um processo de revisão para assegurar a qualidade do código e garantir a segurança do sistema. O sistema conta com as clássicas listas de e-mail de desenvolvimento, tanto para contribuição como para usuários. Para uma comunicação mais direta, o Zephyr adota o Slack como ferramentas para troca de mensagens. Esse canal de comunicação é aberto e organizado de forma eficiente, com integração com o GitHub e diversos canais para objetivos e/ou partes diferentes do sistema.

### 3.3.3 Na prática

Para utilizar o Zephyr é necessário instalar algumas dependências, no guia oficial são sete passos gerais para a instalação, build e flash de uma aplicação. Ele utiliza um serie de ferramentas (Toolchain) utilizado para construir uma imagem de uma aplicação Zephyr e uma ferramenta desenvolvida pelo projeto Zephyr para facilitar o desenvolvimento com o sistema operacional Zephyr.

Para poder analisar a experiência de um usuário com o sistema, foi executado na prática o exemplo "Echo Client", ele implementa um cliente UDP/TCP que envia pacotes IPv4 ou IPv6, espera o retorno dos dados e verifica se ele corresponde aos dados que foram enviados. A placa utilizada no teste foi a nRF54840 Development Kit. O teste foi realizado na versão 1.14.1 do Zephyr. O código pode ser encontrado no próprio repositório do Zephyr.

Depois de gerado o Makefile da aplicação, toda a configuração do sistema foi feita pela interface gráfica no Shell.

O Zephyr se mostra um sistema completo, porém complexo. A preparação para o uso é de certa forma simplificada pelas ferramentas oferecidas, mas para o uso mais profundo do sistema se faz necessário entender a estrutura do sistema.

Uma aplicação é formada por basicamente três arquivos. O primeiro é um arquivo na linguagem C que possui a função principal e o código da aplicação. O segundo é uma lista de configuração para o CMake, ele contém um conjunto de diretrizes e instruções que descrevem os arquivos de origem e destinos do projeto. O terceiro é um arquivo que possui as configurações do Kernel para ser utilizado no projeto.

Dessa forma a tarefa de mudar as configurações da aplicação de exemplo se mostra árdua. Ponto positivo é que a documentação possui um guia muito detalhado de como construir uma aplicação.

Apesar de complexo a organização e padronização do código é um ponto positivo por deixar simples entender como o código funciona e como contribuir com aquele código. O que faz com que depois do aprendizado inicial em uma primeira aplicação, as seguintes fiquem mais simples de fazer.

## 3.4 RIOT OS

O RIOT OS [27] é um sistema operacional que foi oficialmente lançado em 2013, porém teve seu inicial em 2008 com o FeuerWare, um sistema operacional para redes de sensores sem fio.

### 3.4.1 Características gerais

O RIOT OS [28] também é um sistema operacional microkernel. Implementa um escalonador preemptivo, tickless baseando em prioridades.

Funciona com sistemas de threads, sincronizadas via mutex e utilizando um sistema de comunicação entre processos (IPC) para trocar informações entre as threads, de forma síncrona ou assíncrona.

RIOT OS tem em seus destaques a grande gama de tecnologias de rede e comunicação que implementa, como IPv6, 6LoWPAN, RPL, UDP, CoAP, CBOR, além da pilha tradicional TCP/IP. O RIOT também possui suporte ao OpenThread, porém está defasado em comparação ao usado pelo Zephyr.

O sistema ocupa em média 2 kB de RAM (Random Access Memory) e em média 5 kB de ROM (Read-only Memory).

A arquitetura do RIOT também é modular e pode ser vista na Figura 3.3, retirada do site principal do RIOT OS.

O projeto é dividido em quatro grandes partes. A primeira é a abstração de hardware, para o sistema se fazer compatível com diversas plataformas diferentes. Essa camada faz com que o RIOT consiga dividir bem o código especifico para o hardware e o código especifico para a aplicação, fazendo com que o mesmo código funcione em diferentes plataformas. A segunda parte é o Kernel, que é composto basicamente por três principais partes: o escalonador

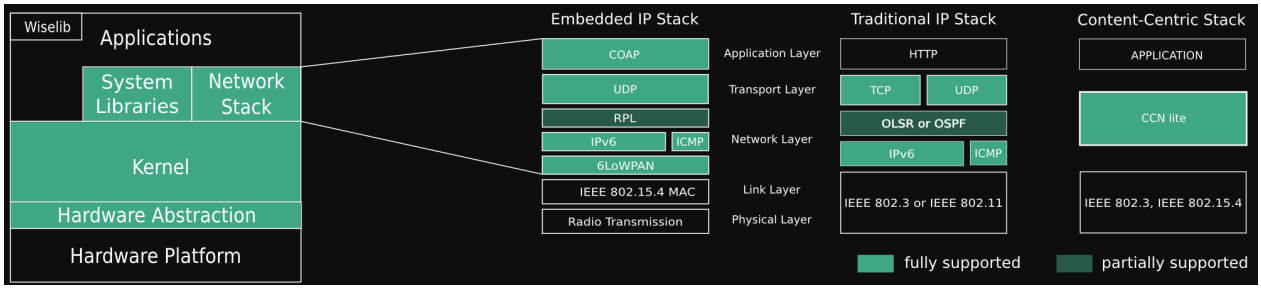


Figura 3.3: Arquitetura RIOT OS

preemptivo, o IPC e o mutex para gerenciar a memória compartilhada. A terceira parte é o pilha de rede. O RIOT possui uma camada genérica [29] que busca abstrair e trazer um acesso único a todas as interfaces de redes suportadas pelo RIOT. Por último, tem uma camada que possui bibliotecas de integração do sistema com a aplicação.

O RIOT OS possui suporte microcontroladores de 8, 16 e 32 bits. Possui porte para placas das arquiteturas AVR, ARM, ARC, x86, ESP8266, ESP32, MIPS32, MSP430, PIC32 e RISC-V. Entre as placas está a nRF52840 Development Kit, porém não possui suporte a versão Dongle.

O RIOT OS possui a licença GNU Lesser General Public License (LGPL). Essa licença permite que terceiros utilizem o código do RIOT OS sem exigir que você necessariamente abra o código do produto.

### 3.4.2 Suporte e Comunidade

O RIOT OS também possui um porte nativo para executa-lo no Linux, dessa forma todo o desenvolvimento do projeto pode ser feito e testado em um computador com Linux. Além disso, o sistema é todo implementado em C e C++, assim como as aplicações também podem ser feitas nessas linguagens.

Todo o código do RIOT é hospedado no GitHub [30]. Até novembro de 2019 foram mais de 25 mil commits e 228 contribuidores.

O RIOT utiliza listas de e-mail para usuários e desenvolvedores, e utiliza um canal no IRC para comunicação e chat.

### 3.4.3 Na prática

O uso do RIOT OS é bastante simples. O próprio projeto se intitula como "um sistema operacional amigável para IoT".

O RIOT OS utiliza o sistema de build GNU make, que usa arquivos de configuração Makefile

para configurar o build da aplicação. Em adição utiliza o conjunto de ferramentas gcc-arm-embedded para microprocessadores ARM, como é o caso da nRF52840. Desse modo, para compilar uma aplicação só é necessário a instalação, em um passo, do gcc-arm-embedded e utilizar um único comando para buildar e flashar a aplicação na plataforma.

Para poder analisar a experiência de um usuário com o sistema, foi executado na prática o exemplo "openthread", um exemplo que exporta uma aplicação de linha de comando com acesso as funcionalidades implementadas pelo OpenThread. A placa utilizada foi a nRF52840. O código pode ser encontrado no próprio repositório do RIOT OS.

A estrutura de aplicação do RIOT é bastante simples, só é necessario indicar quais módulos dos serviços a aplicação deve utilizar e qual a plataforma de hardware e, dessa forma, o próprio sistema configura as demais camadas até a abstração de hardware.

A estrutura do código também é simples, são somente dois arquivos. O primeiro é o código fonte da aplicação e o segundo é um aquivo Makefile com as configurações da aplicação.

O exemplo que foi utilizado, o openthread, não estava funcionando como descrito pela documentação. Para o funcionamento correto do build da aplicação, passos adicionais tiveram que ser feitos. O primeiro foi a adaptação para a placa nRF52840 e o segundo foi a instalação das dependências da instalação do OpenThread. O teste foi realizado na versão 2019.10 do RIOT OS.

Nesse processo de adaptação pôde se observar que a documentação é distribuída. Parte está na Wiki do GitHub do projeto, parte está na página de documentação web do projeto e parte só se encontra analisando o código. Essa documentação distribuída e não organizada faz existir uma maior dificuldade no entendimento do código.

## 3.5 Discussão

A quantidade de sistemas operacionais com foco em internet das coisas que existem no mercado é enorme. Uma das justificativas por existir tamanha quantidade é que no âmbito de internet das coisas, os casos de usos são diversos. Então existe determinados sistemas que atendem melhor a necessidade de determinado caso de uso.

Uma das formas que podemos observar, com a análise que foi feita nesse capítulo, é para qual situação cada sistema se mostraria com uma melhor escolha. Ou seja, para qual uso o sistema teria suas qualidades aproveitadas e suas fraquezas minimizadas.

Tabela 3.1: Características gerais dos sistemas operacionais analisados

| OS | Arquitetura | Escalonador | Multi-thread | Protocolos de Comunicação | Suporte Openthread | Min RAM | Min ROM | Arquiteturas de Hardware suportadas | Documentação |
|---|---|---|---|---|---|---|---|---|---|
| Zephyr | Nanokernel ou microkernel | Cooperativo, prioritário, Earliest Deadline First(EDF), não preemptivo e preemptivo. | Sim | Bluetooth, Bluetooth LE, WiFi, 802.15.4, 6LoWPAN, CoAP, IPv4, IPv6 e NFC | Sim | 2 kB a 8 kB | 50 kB | ARM, ARC, x86, RISCV, XTENSA e NIOS II | Completa e centralizada |
| RIOT OS | Microkernel | Preemptivo, baseado em prioridades | Sim | IPv6, 802.15.4, 6LoWPAN, RPL, UDP, CoAP, CBOR, LoRaWAN | Sim | 2 kB | 5 kB | AVR, ARM, ARC, x86, ESP8266, ESP32, MIPS32, MSP430, PIC32 e RISC-V | Incompleta e descentralizada |

Em relação as características gerais, ambos os sistemas analisados mais profundamente, o Zephyr e o RIOT OS, possuem dados muito parecidos, como pode ser visto na Tabela 3.1.

Ambos são modulares com arquitetura microkernel. Possuem escalonadores preemptivos baseados em prioridades, como modo para economia de energia. São multithreads, com comunicação entre threads com IPC de interface facilitada. Contam com mecanismos de sincronização por mutex. Aceitam parecidas arquiteturas de plataforma de hardware e protocolos de comunicação.

Apesar das características básicas serem semelhantes, o Zephyr apresenta mais recursos alternativos. Como diferentes algoritmos de escalonamento, possibilidade de ser nanokernel, diferentes interfaces de programação para troca de mensagens entre as threads e diferentes modos de sincronização entre threads.

O Zephyr também possui uma implementação mais nova do OpenThread e comumente atualiza a sua versão em decorrência de novos lançamentos da implementação oficial do OpenThread.

O RIOT OS possui suporte a arquitetura AVR e tem porte para o Arduino já utilizado pelo KNoT, o que poderia ser uma vantagem na transição de arquitetura.

Outra diferença que podemos destacar é a documentação. No Zephyr ela está completa, centralizada e organizada de forma lógica quanto a complexidade do sistema. No RIOT OS a documentação ainda não está completa e faltam guias para se aprofundar na implementação do sistema. Essa organização se estende a comunidade. A escolha da utilização do Slack como ferramenta de interação entre desenvolvedores facilita a organização dos assuntos e troca de experiência.

O ponto forte do RIOT OS está na simplicidade. Como foi dito, ele se intitula como "O sistema operacional amigável para a Internet das Coisas"e cumpre esse papel.

O RIOT OS possui uma estrutura simples. A configuração inicial para poder compilar e usar uma aplicação feita com o RIOT OS é bastantes simples, se resuma a duas etapas. A interface oferecida para interação e uso dos recursos do sistema também seguem a linha da simplicidade. Além de possuir um requisito de memória menor que o Zephyr.

Sendo assim, podemos destacar que o Zephyr é um sistema operacional mais complexo e profissional. Interessante para criação de produtos para o mercado. É um projeto da Linux Foundation, com um foco percebido em segurança, seguindo princípios das boas práticas de código livre, completamente modular com diversas opções de bibliotecas/protocolos/interfaces e suporte atualizado e certificado a tecnologias de rede. O Zephyr traz o programador para mais perto do metal, ou seja, dá maior controle do sistema ao desenvolvedor.

O RIOT OS é um sistema amigável. Interessante para uso rápido, para criar provas de conceito e para fins educacionais. Mesmo com foco em simplicidade, o RIOT é um sistema complexo e bastante completo. Possui suporte a tecnologias importantes e uma comunidade bastante ativa.

# Capítulo 4

# Conclusão

Esse trabalho propôs realizar uma análise de sistemas operacionais para internet das coisas com o objetivo de identificar sistemas de acordo com os requisitos para implementação na meta plataforma de internet das coisas KNoT.

Foi visto neste trabalho as principais características do KNoT e dos requisitos definidos da meta plataforma, o SoC nRF52840 e o protocolo Thread. Foram selecionados os principais sistemas que atendiam previamente a todos os requisitos, o Zephyr e o RIOT OS.

O Zephyr é um sistema operacional complexo, com diferentes configurações e implementações das características básicas do seu Kernel. É um sistema bem documentado e com uma comunidade organizada e ativa. O RIOT é um sistema que possuir todas as características básicas necessárias para um bom sistema operacional, porém é simples. Sendo assim, possui um nível de abstração mais alto e uma curva de aprendizado menor para o uso dos recursos da plataforma.

Dessa forma foi possível cumprir o objetivo proposto, discutindo sobre as analises dos sistemas e prevendo para quais casos cada sistema eles seriam melhor utilizados.

## 4.1 Dificuldades encontradas

A maior dificuldade encontrada na realização desse trabalho foi a seleção de sistemas operacionais que seriam analisados de forma mais profunda. A quantidade de sistemas no mercado é enorme e por isso foi necessário fazer uma análise previa de todos os sistemas para ver se eles se adequavam aos requisitos mínimos dados no escopo do trabalho.

## 4.2 Trabalhos Futuros

A partir da análise feita nesse trabalho, próximos passos podem ser dados em trabalhos futuros, como:

- Realizar uma análise das implementações das camadas de redes empregadas nos sistemas operacionais para internet das coisas.
- Analisar como a organização e participação da comunidade afeta na aceitação e adesão de um sistema operacional para internet das coisas.
- Implementação do KNoT Protocol como uma biblioteca em um sistema operacional para internet das coisas.
- Projetar nova placa da meta plataforma KNoT. Levando em consideração a definição do sistema operacional do dispositivo.

# Referências Bibliográficas


[1] L. Atzoria, A. Ierab, and G. Morabitoc, "The internet of things: A survey," *Computer Networks*, 2010.

[2] "Inthernet of things: Microsoft practice development playbook." `https://partner.microsoft.com/en-us/campaigns/iot-playbook`.

[3] G. Blair, M. Paolucci, P. Grace, and N. Georgantas, "Interoperability in complex distributed systems," pp. 1–26, 01 2011.

[4] "Unlock the potencial of the internet of things." `https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/The%20Internet%20of%20Things%20The%20value%20of%20digitizing%20the%20physical%20world/Unlocking_the_potential_of_the_Internet_of_Things_Executive_summary.ashx`. Acessado em 17/12/2018.

[5] "Knot website." `https://www.knot.cesar.org.br/`.

[6] "Arduino website." `https://www.arduino.cc/`.

[7] "nrf52840 product website." `https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840`.

[8] "Cesar github." `https://www.github.com/CESARBR`.

[9] T. Barros, C. Takahasi, V. Aquino, P. S. Filho, R. Ribeiro, J. A. Neto, C. Batista, P. V. Silva, E. Cavalcante, and T. Batista, "A multi-radio gateway architecture and implementation for consumer electronics," *2019 IEEE International Conference on Consumer Electronics (ICCE)*, 2019.

[10] "Itu-t y.4460: Architectural reference models of devices for internet of things applications." `http://handle.itu.int/11.1002/1000/13921`.

[11] "Openthread website." https://openthread.io.

[12] "nrf52840 information website." https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_nrf5x_cltools%2FUG%2Fcltools%2Fnrf5x_nrfjprogexe.html&cp=6_1_3.

[13] "Thread group website." https://www.threadgroup.org/.

[14] W. Dong, C. Chen, X. Liu, and J. Bu, "Providing os support for wireless sensor networks: Challenges and approaches," *IEEE Communications Surveys  Tutorials*, 2010.

[15] A. Milinković, S. Milinković, and L. Lazic, "Choosing the right rtos for iot platform," *INFOTEH-JAHORINA*, 2015.

[16] T. Borgohain, U. Kumar, and S. Sanyal, "Survey of operating systems for the iot environment," *International Journal of Advanced Networking and Applications*, 2015.

[17] P. Gaur and M. P. Tahiliani, "Operating systems for iot devices: A critical survey," *2015 IEEE Region 10 Symposium*, 2015.

[18] "Devopedia. 2019. "iot operating systems."version 10, september 23.." https://devopedia.org/iot-operating-systems. Acessado em 12/11/2019.

[19] Y. B. Zikria, H. Yu, M. K. Afzal, M. H. Rehmani, and O. Hahm, "Internet of things (iot): Operating system, applications and protocols design, and validation techniques," *Future Generation Computer Systems*, 2018.

[20] "Amazon freertos website." https://aws.amazon.com/pt/freertos/.

[21] "Freertos website." https://www.freertos.org/.

[22] "Brown, eric. 2018. "linux still rules iot, says survey, with raspbian leading the way."linuxgizmos, april 30." Acessado em 04/05/2018.

[23] "Mbed os website." https://os.mbed.com/.

[24] "Zephyr website." https://www.zephyrproject.org/.

[25] "Documentação do zephyr." https://docs.zephyrproject.org/.

[26] "Código fonte do zephyr." https://github.com/zephyrproject-rtos/zephyr.

[27] "Riot os website." http://riot-os.org/.

[28] "Documentação do riot os." https://doc.riot-os.org/.

[29] M. Lenders, "Analysis and comparison of embedded network stacks," Master's thesis, Freien Universitat Berlin, Apr. 2016.

[30] "Código fonte do riot os." https://github.com/RIOT-OS/RIOT/.