UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Filipe Landerdahl Albanio

ANÁLISE DA COMUNICAÇÃO POR BLUETOOTH LOW ENERGY COM PILHA IPV6 USANDO OS SISTEMAS OPERACIONAIS ZEPHYR E CONTIKI

Santa Maria, RS
2017

Filipe Landerdahl Albanio

ANÁLISE DA COMUNICAÇÃO POR BLUETOOTH LOW ENERGY COM PILHA IPV6 USANDO OS SISTEMAS OPERACIONAIS ZEPHYR E CONTIKI

Monografia apresentada à Universidade Federal de Santa Maria – UFSM, como requisito parcial para obtenção do título de **Engenheiro de Computação**.

Orientador: Dr. Carlos Henrique Barriquello

Santa Maria, RS
2017

Filipe Landerdah Albanio

ANÁLISE DA COMUNICAÇÃO POR BLUETOOTH LOW ENERGY COM PILHA IPV6 USANDO OS SISTEMAS OPERACIONAIS ZEPHYR E CONTIKI

Monografia apresentada à Universidade Federal de Santa Maria – UFSM, como requisito parcial para obtenção do título de **Engenheiro de Computação**.

Aprovado em 04 de dezembro de 2017

Carlos Henrique Barriquello, Dr.
(Presidente/ Orientador)

Saul Azzolin Bonaldo, Dr, UFSM

Ricardo Nunes Marchesan, Eng, UFSM

Santa Maria, RS
2017

AGRADECIMENTOS

Nesse espaço gostaria de deixar minha gratidão às pessoas que contribuíram para construção desse trabalho.

Inicio pelos meus pais Sirineu Albanio e Noeli Terezinha Landerdahl, por sempre me incentivarem a buscar conhecimento, investirem na minha educação e por serem exemplos de como dedicação gera bons frutos.

À minha esposa Mariana Milbradt Corrêa pelo companheirismo e apoio durante todos esses anos juntos, pela compreensão nos finais de semestres e por ouvir atentamente meus trabalhos, mesmo não sendo da área.

Ao meu irmão Mateus Landerdahl Albanio que esteve sempre bem disposto para me ajudar no que fosse necessário durante esses últimos anos.

Ao meu orientador Carlos Henrique Barriquello, que mesmo com um grupo grande de orientandos aceitou meu pedido e me auxiliou em todas as etapas deste trabalho, desde a definição do tema, configurações, testes até os resultados finais.

Ao meu tutor, amigo e banca professor Saul Azzolin Bonaldo, o responsável por despertar meu interesse por eletrônica e sistemas embarcados em 2009, por sempre me orientar e assistir desde então, bem como por ter aceitado contribuir também nesse trabalho de conclusão.

Ao meu colega de trabalho e grande amigo Ricardo Nunes Marchesan, pela disponibilidade em me auxiliar nos problemas e dúvidas que surgiram ao longo desses cinco anos de engenharia e no final ter aceitado participar como banca da minha defesa.

Ao meu chefe Tiago Martini Sanchotene que foi compreensivo e flexível nos horários próximos aos finais dos semestres e por emprestar alguns equipamentos necessários para esta pesquisa.

Aos meus colegas de curso Guilherme Vieira Hollweg, Gabriel Sandin Falcão e Felipe Jaskulski Maia pela parceria e troca de experiências durante a faculdade.

Por fim aos professores Mateus Beck Rutzig e Cesar Tadeu Pozzer que foram especiais para minha formação.

RESUMO

ANÁLISE DA COMUNICAÇÃO POR BLUETOOTH LOW ENERGY COM PILHA IPV6 USANDO OS SISTEMAS OPERACIONAIS ZEPHYR E CONTIKI

AUTOR: Filipe Landerdahl Albanio
ORIENTADOR: Carlos Henrique Barriquello

Com a recente popularização do Bluetooth Low Energy (BLE), diversos dispositivos, que antes não possuíam acesso à internet, passarão a ter. Com isso, uma forma de se realizar uma comunicação robusta e eficiente é de suma importância. Portanto, para este trabalho analisamos dois diferentes sistemas operacionais de tempo real (RTOS, do inglês: Real-Time Operating System) usando o módulo nRF52832, da Nordic Semicondutor, que possui uma comunicação Bluetooth 4.2. Os RTOSs são o Contiki OS e o Zephyr OS, e ambos possuem sua pilha IPv6 para BLE. Para a realização do trabalho, o protocolo da camada de aplicação escolhido foi o MQTT, por se tratar de uma tecnologia leve e voltada para o mundo da IoT. Foi avaliado o consumo de energia durante a publicação de mensagens, e a taxa máxima de transferência de dados, variando o tamanho da mensagem e a distância entre o dispositivo e o nó central. O projeto aborda também todas as configurações necessárias para o correto funcionamento da comunicação entre um computador e o módulo com os RTOSs utilizados. Nesse sentido ficou demonstrado que o Zephyr OS possui um consumo de energia médio de 5 a 10% menor que o Contiki OS, e taxa de transferência de dados entre 3 a 4 vezes maior. Entretanto, aparentemente o Contiki apresentou menos falhas de comunicação do que o Zephyr durante os testes executados, embora não se possa afirmar se as falhas observadas no Zephyr foram causadas por falhas no software ou causas externas, como interferência.

Palavras-chave: 6LoWPAN, MQTT, nRF52 e RTOS.

ABSTRACT

ANALYSIS OF BLUETOOTH LOW ENERGY COMMUNICATION WITH IPV6 STACK USING ZEPHYR AND CONTIKI OPERATING SYSTEMS

AUTHOR: Filipe Landerdahl Albanio
ADVISOR: Carlos Henrique Barriquello

With the recent popularization of the Bluetooth Low Energy (BLE), several devices, that previously did not have access to the Internet, will have. With this, one way of achieving robust and efficient communication is of extreme importance. Therefore, for this work, we analyze two different real-time operating systems (RTOS) using the nRF52832 module, from Nordic Semiconductor, which has Bluetooth 4.2 communication hardware. Contiki OS and Zephyr OS are the RTOSs used, and both have their own IPv6 stack for BLE. In order to perform this work, the chosen protocol for the application layer is MQTT, because it is a light protocol and aimed for the world of IoT. It was evaluated the energy consumption during message publishing, and the maximum of data rate transfer, varying the size of the message and the distance between the devices. The project also includes all the necessary configurations for the proper functioning of the communication between a computer and the module with the RTOSs used. In this sense it has been shown that Zephyr OS has an average energy consumption of 5 to 10% lower than the Contiki OS, and data transfer rate between 3 to 4 times higher. However, Contiki appeared to have fewer communication failures than Zephyr during the tests performed, although it cannot be stated whether the failures observed in Zephyr were caused by software failures or external causes, such as interference.

Key-words: 6LoWPAN, MQTT, nRF52 and RTOS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE SIGLAS

BLE	Bluetooth Low Energy
LE	Low Energy
IoT	Internet of Things
IEEE	Institute of Electrical and Electronics Engineers
IPv6	Internet Protocol version 6
IPv4	Internet Protocol version 4
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
WPAN	Wireless Personal Area Networks
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
LR-WPAN	Low-Rate Wireless Personal Area Networks
LAN	Local Area Network
MAN	Metropolitan Area Network
RADVD	Router Advertisement Daemon
RFC	Requests For Comment
IETF	Internet Engineering Task Force
RTOS	Real Time Operating System
OS	Operating System
MQTT	Message Queue Telemetry Transport
LED	Light-Emitting Diode
PCB	Printed Circuit Board
CI	Circuito Integrado
PTH	Plated Through-Hole

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Os recentes avanços na área de tecnologia permitiram que a comunicação entre diferentes dispositivos se tornasse uma realidade cada vez mais presente na vida de todos. Com a Internet das Coisas (IoT – Internet of Things), praticamente todos os itens de uma casa, desde lâmpadas e sensores até equipamentos mais complexos como cafeteiras e ar condicionado, poderão possuir conexão com a internet. Com tantos novos equipamentos conectados à rede, é necessário utilizar uma forma viável e energeticamente eficiente de comunicação sem fio para conexão com a internet.

Com o lançamento do Bluetooth 4.0 em 2010, que continha um hardware extra totalmente diferente das versões anteriores, se tornou mais fácil fazer comunicação sem fio de forma eficiente em equipamentos onde o principal foco é o baixo consumo de energia. Esse hardware extra adicionado foi batizado de Bluetooth Low Energy (BLE, do inglês: Bluetooth de baixo consumo de energia) e virou concorrente direto do padrão IEEE[1] 802.15.4[2], pois ambas tecnologias focam no baixo consumo de energia, baixa transferência de dados e redes de pequenas distâncias.

Atualmente, as duas tecnologias mais utilizadas para comunicação sem fio de baixo consumo são a IEEE 802.15.4 e o BLE. Embora o IEEE 802.15.4 seja um dos padrões mais antigos desenvolvidos para a comunicação sem fio de baixa energia, o BLE apresenta melhor taxa de comunicação de dados e consumo de energia quando ambos implementam pilha[3] IPv6 (Internet Protocol versão 6). (TRELSMO; DI MARCO; SKILLERMARK; CHIRIKOV; OSTMAN, 2017).

---

[1] O Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) é a maior organização profissional do mundo dedicada ao avanço da tecnologia em beneficio da humanidade, uma parte desta organização se dedica a família IEEE 802, dos padrões que especificam as LANs e MANs.

[2] O padrão IEEE 802.15.4 foi definido em 2003 especifica a camada física e efetua o controle de acesso para Redes de Área Pessoal Sem Fio de Baixas Taxas de Transmissão (low-rate wireless personal area networks - LR-WPANs). O ZigBee é o padrão mais famoso a utilizar como base o IEEE 802.15.4.

[3] Pilha vem da palavra inglesa “stack”, significando que algumas tecnologias possuem varias camadas empilhadas uma sobre a outra, cada camada com um propósito e objetivo diferente de operação. No final o conjunto de camadas, a pilha, faz com que a tecnologia funcione.

Com a popularização do Bluetooth, que está atualmente presente em praticamente todos os computadores e smartphones, foi lançado em outubro de 2015 o padrão RFC[4] 7668. Este padrão especifica a troca de pacotes IPv6 sobre um link BLE, permitindo assim a conexão de equipamentos que possuem BLE à internet.

Essa popularização gerou um aumento na utilização desta tecnologia, e várias empresas/grupos lançaram suas pilhas IPv6 para BLE. Infelizmente, até o presente momento não há nenhuma comparação dessas diferentes implementações. Com isso, o objetivo deste trabalho será fazer uma comparação de duas implementações de pilha IPv6 para BLE, para que futuramente alguém que pretenda utilizá-lo possa ter uma base para escolher a implementação mais adequada para a aplicação desejada.

1.2 METODOLOGIA

O trabalho prático foi dividido em duas etapas, a primeira parte trata de uma comunicação utilizando o pilha IPv6 sobre o BLE entre dois receptores Bluetooth conectados a um computador rodando o BlueZ[5] cada um. O objetivo desta é realizar todas as configurações no computador para que ele esteja apto a receber e redirecionar corretamente os pacotes IPv6 vindos pelo BLE. Um dos computadores serve, dessa forma, como um nó, contendo apenas uma conexão BLE, enquanto o outro tem o papel de gateway, possuindo BLE e Ethernet. A segunda parte é uma análise da pilha IPv6 implementada por dois diferentes RTOSs (Zephyr OS e o Contiki OS) para o CI nRF52832, que realizará a comunicação entre o módulo e o computador com adaptador USB rodando a pilha BlueZ. Para realizar o funcionamento da comunicação, o computador com o Linux passará a ser uma espécie de gateway, já configurado anteriormente. Possibilitando assim o CI nRF52 acesse o broker MQTT que está em outra máquina apenas com conexão Ethernet, para realizar uma publicação em MQTT.

[4] Os RFCs (Request for Comments) são documentos técnicos desenvolvidos pela Internet Engineering Task Force (IETF). O IETF é um grupo informal internacional aberto que desenvolve padrões para internet.

[5] O BlueZ é uma pilha IPv6 BLE para sistemas operacionais baseados em Linux, que tem como objetivo especificar padrões para comunicação sem fio para o Linux

Para realização dos testes e medições, uma PCB (Printed Circuit Board – Placa de Circuito Impresso) foi confeccionada. Esta PCB é constituída de uma bateria, LEDs, botões e uma unidade sensória para servir como informação a serem enviadas para o broker. Foi medido na PCB o consumo de energia do CI enquanto em standby e também durante as publicações, bem como o tempo total de envio e tempo de compressão do pacote IPv6. Um teste para medir a taxa de transferência de dados máxima de cada OS (Sistema Operacional) também foi realizado. Para finalizar, foi feita uma análise da troca de pacotes na comunicação utilizando o software Wireshark[6], podendo assim diferenciar o funcionamento das diferentes implementações.

1.3 DIVISÃO DO TRABALHO

O trabalho é desenvolvido em cinco capítulos (inclusa esta introdução), sendo assim distribuídos: o capítulo 2 intitulado de revisão do estado da arte detalha informações e funcionamento a respeito das tecnologias utilizadas neste projeto. O capítulo 3 descreve a primeira parte prática do trabalho, mostrando todos os passos e informações necessárias para a configuração dos softwares para a realização do projeto. O capítulo 4 é o capitulo de desenvolvimento, mostrando a confecção da PCB, os testes realizados e os resultados finais obtidos. Por fim, o capítulos 5 trata da conclusão.

[6] Wireshark é um software desenvolvido inicialmente para plataformas Linux onde é possível analisar e observar o tráfego de dados de qualquer rede ligada ao computador. Com o Wireshark é possível monitorar a entrada e saída de dados do computador, em diferentes protocolos e portas, ou da rede à qual o computador está ligado.

2 REVISÃO DO ESTADO DA ARTE

Neste capítulo são descritas detalhadamente as tecnologias, e suas características, utilizadas para o desenvolvimento deste projeto.

2.1 BLUETOOTH LOW ENERGY – BLE

Em 2006 a Nokia introduzia sua nova tecnologia voltada ao baixo consumo de energia, chamado de Wibree, que posteriormente, em 2010, foi integrada no Bluetooth versão 4.0. Com esta incorporação, o Wibree foi rebatizado para Bluetooth Low Energy e vendido pelo nome de Bluetooth Smart[7].

Desde a fusão de 2010 o Bluetooth possui dois tipos de sistemas, o Clássico e o de baixo consumo (LE). Os dois sistemas não são interoperáveis[8], um adaptador pode implementar os dois, mas apenas um pode estar sendo usado por vez (SPÖRK, 2016).

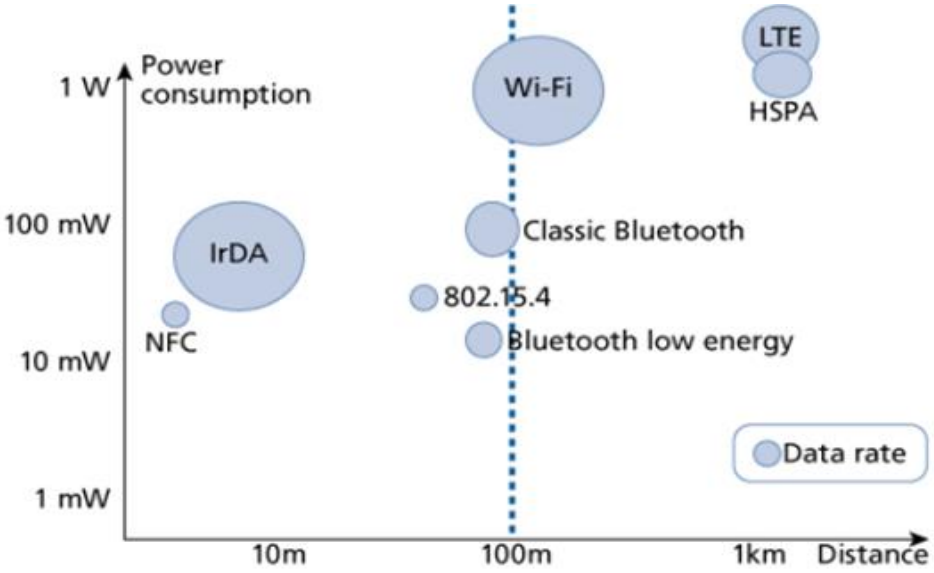
O Bluetooth LE, como também é chamado, é uma expansão do Bluetooth clássico, mas projetado para um consumo mínimo de energia. Mesmo o BLE tendo muitas semelhanças ao Bluetooth clássico, ele é considerado uma tecnologia separada, que possui métodos de operações diferentes. Enquanto a maioria das tecnologias de transmissão sem fio tende ao aumento da taxa de transferência, o BLE possui uma abordagem diferente, focando no mínimo consumo de energia e custo de equipamento, fazendo com que as altas transferências de dados se tornassem a moeda de troca. O BLE tem como objetivo conectar dispositivos que historicamente não possuem tecnologia de comunicação sem fio (KLASMAN, 2017).

Na figura 1 apresentada a seguir, podemos ver uma análise de várias tecnologias de comunicação sem fio. Nessa ilustração podemos observar a distância que a comunicação alcança (no eixo X do gráfico), o consumo de energia para as transmissões (no eixo Y do gráfico) e a taxa de transferência de dados representada pelo tamanho dos círculos. Observamos no gráfico que o BLE possui o melhor custo benefício em relação ao consumo de energia versus distância. Esse bom custo

[7] Atualmente o BLE é desenvolvido e mantido pelo Bluetooth SIG (Bluetooth Special Interest Group), organização responsável pelo desenvolvimento dos padrões Bluetooth e licença da marca.
[8] A interoperabilidade é a capacidade de diversos sistemas e organizações trabalharem em conjunto (interoperar) de maneira transparente de modo a garantir que pessoas, organizações e sistemas computacionais interajam para trocar informações.

benefício acarreta uma perda de transferência de dados, mostrando que neste parâmetro o BLE perde para praticamente todas as tecnologias analisadas no gráfico em questão.

Figura 1 Comparação de diversas tecnologias de comunicação sem fio



Fonte: (ANDERSSON, 2015, p.08).

O Bluetooth clássico não obteve muito sucesso quando utilizado para WPANs[9], rede de sensores e IoT em comparação com o sucesso do IEEE 802.15.4. Já o BLE exibe várias vantagens em relação ao IEEE 802.15.4.Várias medições mostraram que o BLE possui menor consumo de energia que o IEEE 802.15.4, especialmente nos dispositivos periféricos (SPÖRK, 2016).

Outra ótima vantagem do BLE é que ele está presente em praticamente todos os computadores, smartphones, smartwhatches e outros diversos equipamentos. O custo da tecnologia diminui devido à grande popularização e, com isso, os celulares e computadores podem atuar como roteadores, entre o BLE e WI-FI, para os periféricos que possuem apenas o BLE.

[9] Rede pessoal sem fio (Wireless Personal Area Network – WPAN), também chamada de rede doméstica sem fio, são redes de curto alcance normalmente utilizadas para conectar dois equipamentos próximos. As tecnologias mais famosas a utilizar WPANs são o Bluetooth, o HomeRF (descontinuado em 2003) e o ZigBee.

2.1.1 Principais Características do BLE

As diversas características do BLE estão mostradas a seguir (SPÖRK, 2016):

a) Disponível em todo o globo:

   Por operar em 2,4GHz, o BLE faz parte dos equipamentos das faixas de frequência ISM[10] podendo assim ser utilizado de forma global sem a necessidade de pagar licenças. Pelo fato da banda de 2.4GHz ser livre, muitas outras tecnologias como o WI-FI também utilizam essa frequência, por isso o BLE utiliza uma técnica de salto de frequência adaptativa para neutralizar a interferência de outros dispositivos e aparelhos;

b) Rápida conexão:

   O padrão BLE especifica três canais de rádio que são utilizados apenas para realizar o anuncio (advertising) do dispositivo e a configuração da comunicação. O BLE necessita verificar apenas estes canais para uma requisição de conexão, diferente do Bluetooth clássico que necessita verificar todos os canais utilizados para verificar uma solicitação de conexão;

c) Radio quase sempre desligado:

   O estado padrão do BLE é desligado, nesse estado o rádio é desligado e o equipamento consome muito pouca energia;

d) Funcionalidade reduzida:

   O BLE possui muito menos comandos e funcionalidades quando comparado com o Bluetooth Clássico. Ele não suporta operações como Scatternet[11], canais de voz, comunicação contínua ou papel mestre/escravo que estão disponíveis no Bluetooth clássico. A redução de funcionalidade simplifica a complexidade da pilha de comunicação do BLE, ele necessita muito menos memória estática e dinâmica, reduzindo bastante o tamanho do Circuito Integrado em relação ao Bluetooth Clássico;

e) Operações otimizadas:

   Enquanto os equipamentos centrais normalmente operam com grandes baterias ou até conectados a uma fonte de energia continua, os periféricos

[10] As ISM bands (industrial, scientific and medical radio bands) são faixas de rádio específicas reservadas para uso internacional sem a necessidade de pagamento ou licenciamento para utilização, diferente da grande maioria das bandas de frequência de rádio que são utilizadas para telecomunicações.

[11] Scatternet é um tipo de rede sem fio ad hoc (wireless ad hoc network – WANET) utilizado em redes de comunicação sem fio de tecnologia Bluetooth.

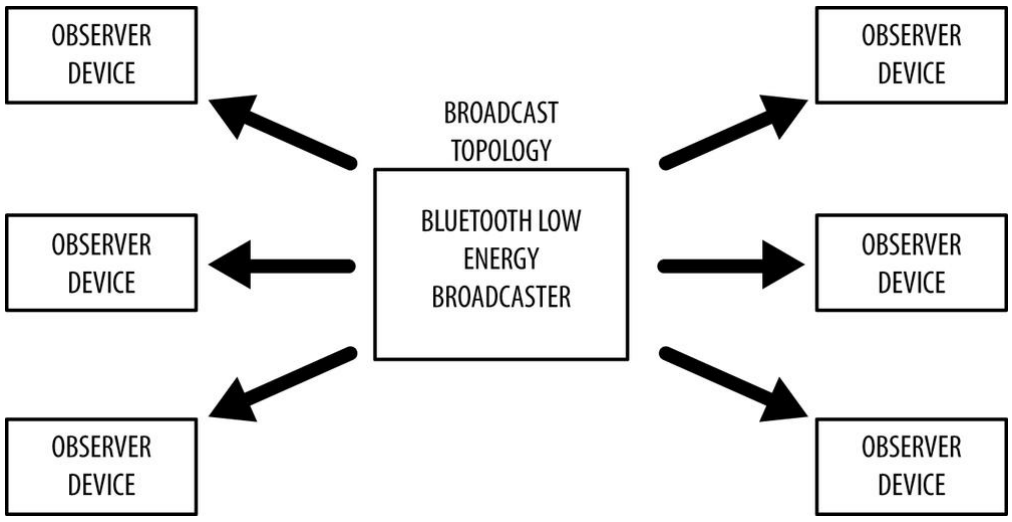
normalmente operam com um uma bateria reduzida. Em seu projeto o BLE já minimiza o consumo de energia do periférico, compensando essa otimização com um equipamento central não tão otimizado energeticamente.

2.1.2 Topologia de Rede

Dispositivos BLE podem ter dois papéis diferentes, um como dispositivo central e outro como periférico. Dispositivos centrais normalmente são equipamentos com alto processamento de CPU (computadores, smartphones) e os periféricos normalmente são sensores de baixo consumo de energia, que se conectam ao dispositivo central.

Um dispositivo BLE pode enviar dois tipos de dados, os pacotes de anúncio (Advertising Packets) e os dados de resposta de escaneamento (Scan Response Data). A comunicação entre dois dispositivos próximos pode ocorrer de duas maneiras, por broadcast ou por conexões. Quando um dispositivo está em broadcast, ele envia pacotes de anúncio sem conexões periodicamente para qualquer observador que esteja escutando. O observador, portanto, repetidamente escaneia a área para receber pacotes. Quando um observador recebe um pacote de anúncio, ele pode solicitar dados de resposta de escaneamento. Esta topologia está ilustrada pela figura 2.

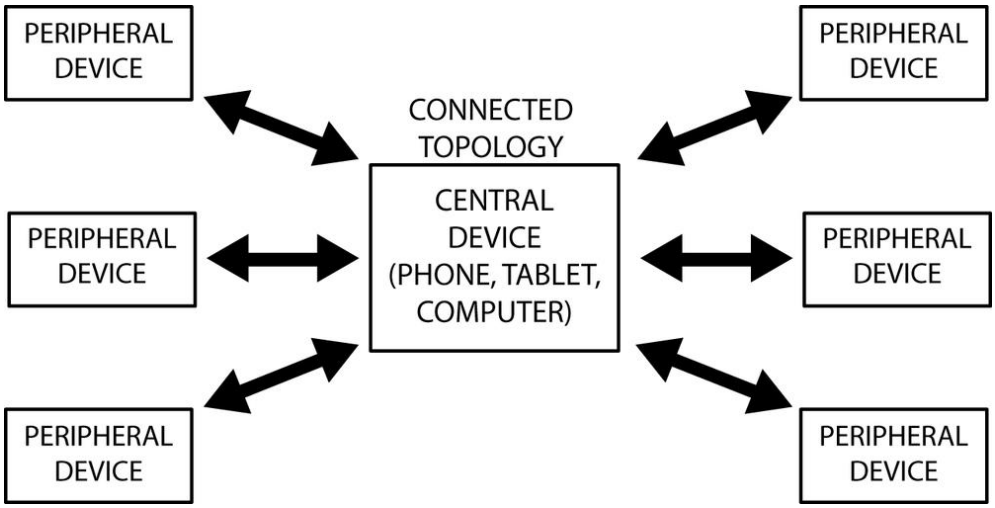
Figura 2 Topologia de broadcast do BLE



Fonte: (MILOVANOVIC, 2016)

A outra maneira de ocorrer uma comunicação por dispositivos BLE é por conexão. Quando em uma conexão, uma permanente troca periódica de pacotes entre os dois dispositivos é necessária. Para iniciar a conexão, o dispositivo central escaneia as frequências procurando por pacotes de anuncio. Quando encontra um pacote adequado, uma conexão é iniciada e, após o estabelecimento da conexão, o dispositivo central controla o tempo e a troca periódica de pacotes. Quando conectados, os dois dispositivos realizam uma troca periódica de pacotes conhecida como evento de conexão (connection event), sendo essa uma das principais características para a economia de energia dos periféricos. Estes dois dispositivos podem ser inicializados, trocar pacotes, e voltar a dormir até o próximo evento de conexão (passando a maior parte do tempo com o radio desligado). Esta topologia está ilustrada pela figura 3

Figura 3 Topologia de conexão do BLE



Fonte: (MILOVANOVIC, 2016)

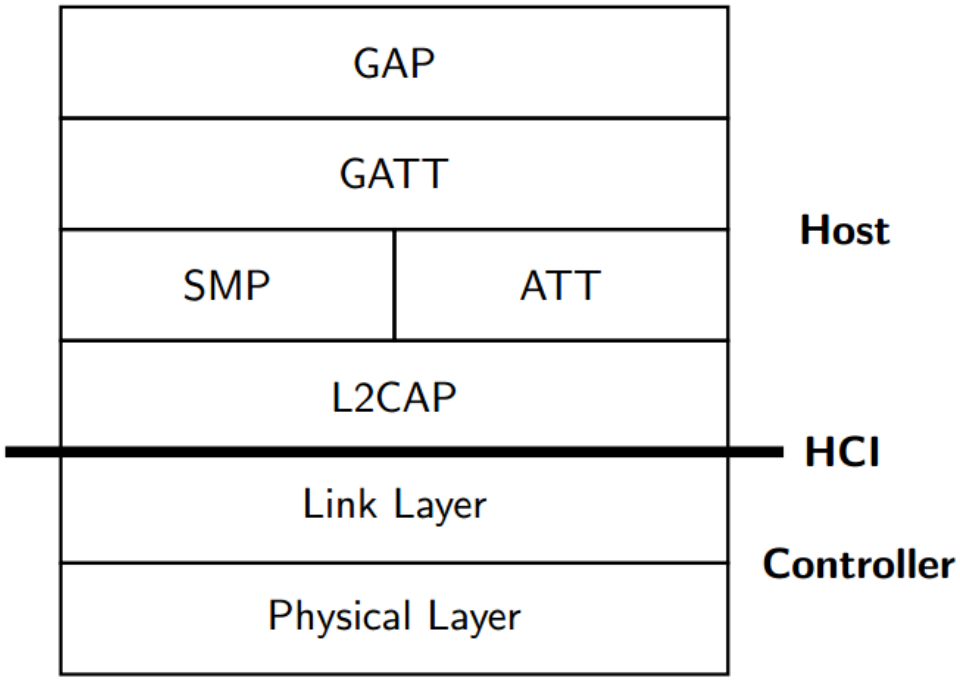
2.1.3 Pilha do BLE

A pilha de comunicação do BLE é dividida em sete camadas, com duas divisões principais: as camadas do controlador (controller) e as camadas do hospedeiro (host).

O controlador normalmente é implementado em um pequeno SoC (System on Chip) e comporta as duas camadas mais baixas da pilha de comunicação, a camada física (Physical Layer) e a camada de enlace (Lynk Layer). Já o hospedeiro é executado em um processador de propósito geral e ele implementa as cinco camadas mais altas da pilha. Tais camadas são: O Protocolo de Controle e Adaptação de Enlace Lógico (Logical Link Control and Adaptation Protocol - L2CAP), o protocolo de atributos (Attribute Protocol - ATT), o perfil de atributo genérico (Generic Attribute Profile - GATT), o protocolo gerenciador de segurança (Security Manager Protocol - SMP) e, por fim, o perfil de acesso genérico (Generic Access Profile - GAP).

O hospedeiro e o controlador podem trocar informações entre si através da interface controladora de hospedeiro (Host Controller Interface - HCI). A figura 4 a seguir ilustra melhor as camadas e os papéis do controlador e hospedeiro.

Figura 4 Pilha de comunicação do BLE



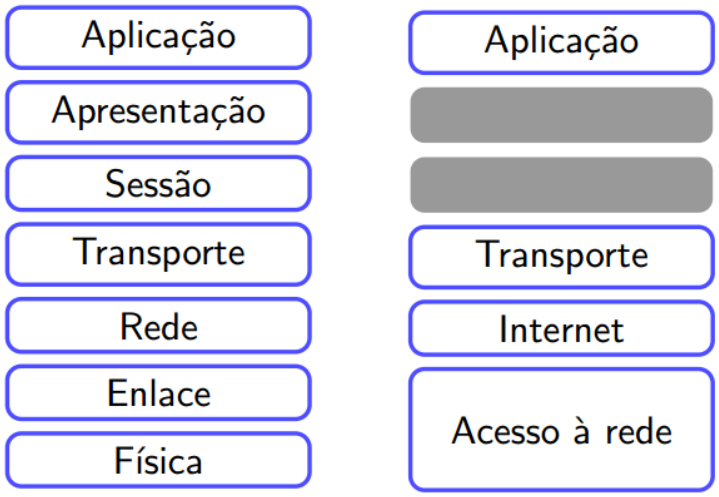
Fonte: (SPÖRK, 2016)

2.2 INTERNET PROTOCOL VERSION 6 – IPv6

Em 1983 a DARPA[12] lançou a ARPANET, a primeira rede de comutação de pacotes[13] a implementar o protocolo TCP/IP, sendo este a base para o lançamento da Internet. O TCP/IP da ARPANET contava com o Protocolo de Internet (IP) versão 4 (Internet Protocol version 4 - IPv4) utilizado até hoje.

O protocolo de internet está presente na camada de rede (terceira camada), tanto no modelo OSI[14] como no modelo TCP-IP[15]. Na figura 5 a seguir é possível observar as diferenças nas camadas desses dois modelos.

Figura 5 Comparação dos modelos OSI x TCP-IP



Fonte: (TANENBAUM; WETHERALL, 2011)

[12] A Agência de Projeto de Pesquisa Avançada de Defesa (Defense Advanced Research Projects Agency – DARPA), dos Estados Unidos da América, foi criada, inicialmente com ARPA (Advanced Research Projects Agency), em resposta ao lançamento do satélite soviético sputnik 1. O objetivo da DARPA é de formular, executar pesquisas e desenvolver projetos em prol da expansão tecnológica e cientifica.

[13] Rede de comutação de pacotes são aquelas em que os pacotes são individualmente encaminhados entre nós da rede através de ligações de dados tipicamente partilhadas por outros nós. Diferentemente da comutação de circuitos que estabelece uma ligação virtual entre os nós, tendo um meio dedicado entre os nós da comunicação. Rede de comutação de pacotes

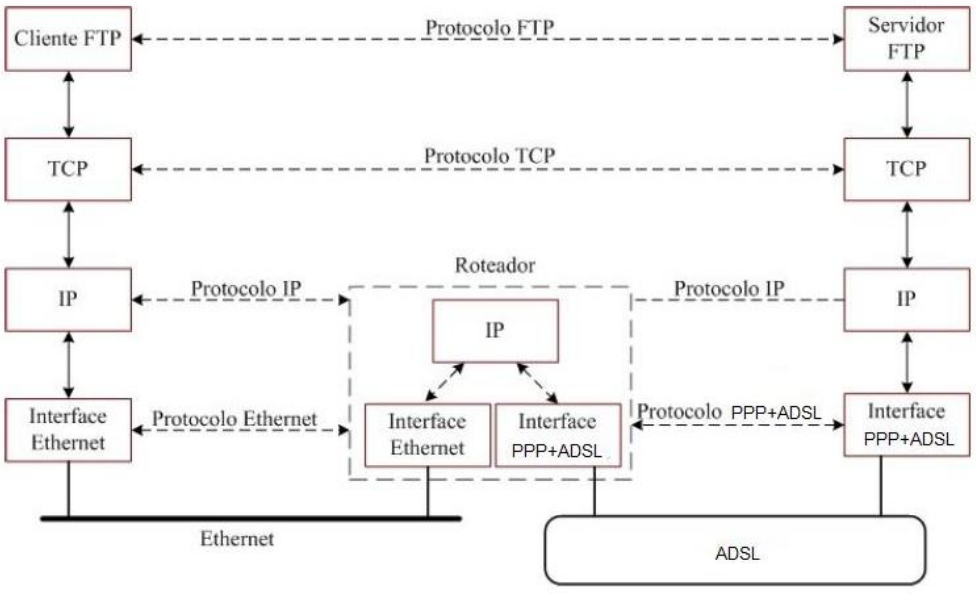
[14] O modelo OSI foi criado em 1971 e formalizado em 1983 com um modelo de rede de computador referência da ISO. Este modelo possui sete camadas bem definidas cada uma com sua função, criando assim uma abstração entre elas.

[15] O modelo TCP-IP é semelhante ao OSI, mas com apenas quatro camadas. As duas camadas mais baixas da OSI foram transformadas em apenas uma de acesso à rede já que esta está normalmente ligada diretamente ao hardware. As duas camadas subsequentes são semelhantes, e as camadas de seção, apresentação e aplicação da OSI estão todas em uma única camada da TCP-IP (camada de Aplicação). Atualmente o modelo TCP-IP é o utilizado pela internet.

A finalidade desta camada é enviar pacotes da origem de qualquer rede na internet e fazê-los chegar ao seu destino, independentemente do caminho e das redes que tomem para chegar lá. Isso faz com que o IP seja o protocolo que faz a interligação de todas as redes e diferentes protocolos a baixo dele. Ele é o único protocolo que deve estar presente em todos os equipamentos do inicio ao fim de uma comunicação entre dois computadores.

Podemos observar na figura 6 como funcionam as camadas do modelo TCP/IP. Temos na imagem dois computadores, um ligado a uma rede por Ethernet e outro por ADSL, e entre os computadores há um roteador fazendo a conexão entre os dois. Observamos que os protocolos acima do IP (camadas de transporte e aplicação) são utilizados apenas nos nós das pontas da comunicação, indiferente para o transporte qual é a comunicação acima utilizada. A camada de acesso à rede abaixo do IP é diferente ao longo de uma comunicação, tornando assim o IP de extrema importância, pois ele se torna um elo entre todos os protocolos. Como comentado anteriormente, o IP deve estar presente em todos os nós (roteadores e afins) ao longo do transporte dos dados.

Figura 6 Funcionamento do protocolo de internet (IP)



Fonte: (TANENBAUM; WETHERALL, 2011)

Devido a essa dependência de estar presente em todos os equipamentos, é extremamente difícil mexer no IP, por isso que desde 1983 o IPv4 ainda é largamente utilizado. Entretanto, cada dia que passa o IPv4 tem se tornado cada vez mais obsoleto, possuindo um campo de endereço de 32 bits, capaz de endereçar apenas 4 bilhões de equipamentos. Quando criado na década de 80 nunca foi imaginado que a internet seria utilizada para recreação, nem que ela possuiria tantos usuários como possui hoje. Já foram necessários vários remendos[16] ao IPv4 original para poder seguir sendo utilizado hoje em dia, pois com o aumento de usuários e principalmente de equipamentos conectados à internet o número de 4 bilhões de equipamentos foi ultrapassado. (ICANN[17], 2011)

Em 1994 foi constatado que o IPv4 encontraria seu fim nas próximas décadas, o IPv6, então, surgiu em 1998 especificado no RFC 2460, e em 2012 entrou em operação. Nesta nova geração do IP a principal diferença é o tamanho de endereçamento, que antes constava com 32 bits, agora consta com 128 bits de endereço. Se 33 bits já duplicava o número de endereços possíveis, 128 multiplicou por 79,2 octilhões o número de endereços do IPv4, que era 4 bilhões, podendo endereçar agora até 3,4x10^38 dispositivos[18]. (TANENBAUM; WETHERALL, 2011). Acredita-se que por um tempo relativamente grande o IPv6 possa endereçar todos os equipamentos com um endereço IP único para cada um.

Além do aumento do campo de endereço, o IPv6 possui também:

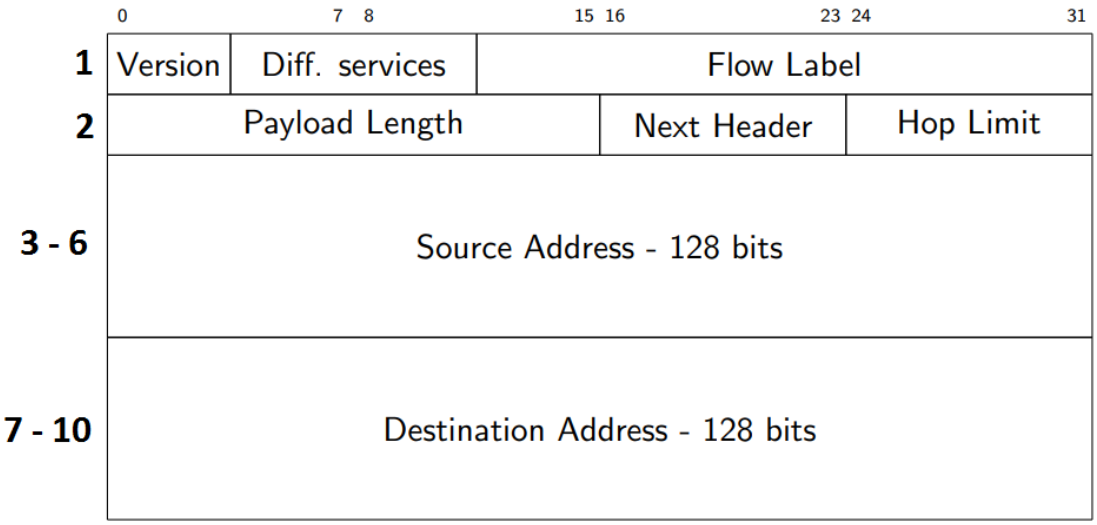
a) Cabeçalho mais simples que o IPv4;
b) Cabeçalhos de extensão – campos opcionais de informação onde é possível adicionar informações extras no futuro sem a necessidade de mexer na estrutura do protocolo;
c) Suporte a qualidade diferenciada – podem ser escolhidos diferentes tipos de tratamento por cada aplicação, dependendo de suas exigências de qualidade;
d) Segurança – Extensões permitem o uso de autenticação, integridade e confidencialidade dos dados.

[16] O remendo mais famoso ao IPv4 foi o NAT (Network address translation) que consiste em reescrever os endereços IP de origem de um pacote que passam por um roteador de maneira que um computador de uma rede interna possa acessar a internet. O problema disso é que todo o computador de uma mesma rede interna passa a ter o mesmo endereço IP externo. Isso criou uma hierarquia que acaba indo contra os princípios do IPv4 que previa um IP único para cada nó na rede.
[17] Corporação da Internet para Atribuição de Nomes e Números, empresa responsável pela alocação de endereços IPs e gerenciar os nomes de domínios.
[18] Como um comparativo, é possível endereçar cada célula de um corpo humano de cada pessoa viva no planeta Terra e ainda sobra endereços no IPv6. É possível também colocar 667 sestilhões de equipamentos ligados à internet por metro quadrado terrestre.

2.2.1 Cabeçalho IPv6

O cabeçalho IPv6 possui 40 Bytes, desses: 32 são para endereço (128 bits de endereço para origem e para destino), os outros 8 Bytes são utilizados para parâmetros e configurações, como mostra a imagem a seguir:

Figura 7 Formato do cabeçalho do IPv6



Fonte: (TANENBAUM; WETHERALL, 2011)

O cabeçalho principal é constituído de 10 linhas de 4 Bytes[19] cada, totalizando os 40 Bytes. Os cabeçalhos de extensão opcionais quando presentes são concatenados ao final dos 40 bytes do cabeçalho principal, assim como os dados (payload) que são concatenados ao final do cabeçalho efetivo. A descrição dos campos do cabeçalho são os seguintes:

a) Version: Diferenciação entre IPv4 e IPv6;
b) Diff. Service: Distinção entre pacotes com tratamento diferenciado[20];

[19] A escolha de cada linha possuir 32 bits é devido ao grande número de roteadores e equipamentos da rede de possuir arquitetura do processador de 32 bits, facilitando assim os cálculos e evitando o desperdício de energia e tempo nos cálculos necessários durante a transmissão.
[20] Um exemplo de tratamento diferenciado é o tempo de entrega, em que a aplicação pode selecionar tempo real, este pacote portanto tendo prioridade sobre os outros durante a transmissão.

c) Flow label: Configuração de pseudoconexão entre a origem e o destino com prioridades específicas para o fluxo de pacotes[21];
d) Payload Length: Informa o número de bytes de dados (payload) que seguem após o cabeçalho;
e) Next header: Indica a existência de cabeçalho de extensão se existir, ou serve pra indicar o protocolo de transporte (TPC ou UDP, por exemplo);
f) Hop Limit: é um contador que é decrementado em cada salto por um roteador, dessa maneira os pacotes não tem uma duração eterna e após no máximo 255 saltos eles são descartados;
g) Addresss: Endereços de fonte e de destino, normalmente escrito na forma de oito grupos de quatro hexadecimais cada. Cada grupo contem 16 bytes[22].

2.3 IPv6 OVER LOW-POWER WIRELESS PERSONAL AREA NETWORK – 6LoWPAN

Como mostrado anteriormente, o BLE tem algumas vantagens quando comparado com as tecnologias semelhantes de baixo consumo, como o IEEE 802.15.4 para aplicações relacionadas à Internet das Coisas, onde uma quantidade muito pequena de dados é transmitida a partir de um dispositivo em tempos regulares. Entretanto, para garantir uma comunicação interoperável com equipamentos da IoT, os equipamentos com BLE não podem utilizar seu simples pacotes Bluetooth. O melhor método para se realizar essa troca de pacotes no mundo da Internet das Coisas é utilizando IPv6.

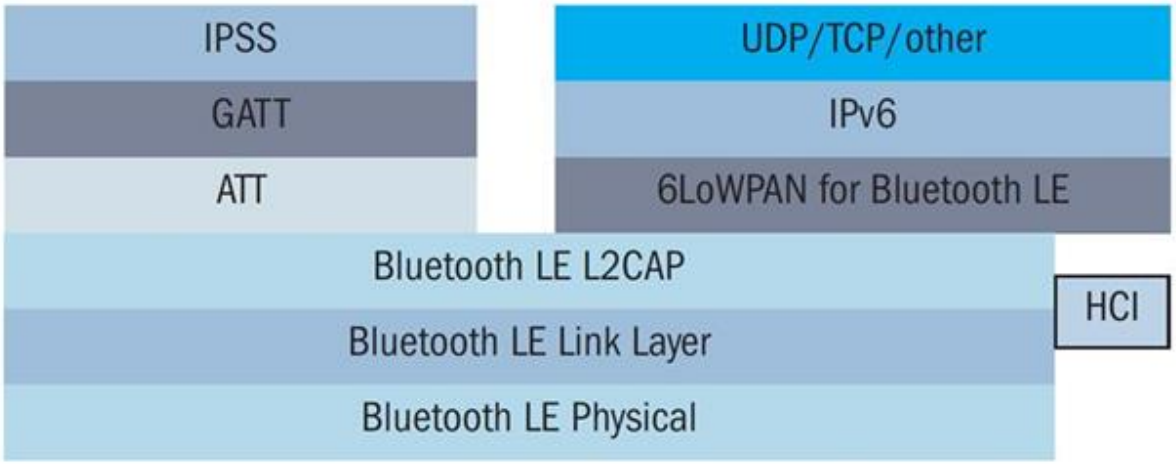
Entretanto, o Bluetooth 4.0 foi introduzido com um tamanho máximo de pacote a ser enviado por vez de 33 bytes, isso sem considerar ainda os cabeçalhos que cada camada da pilha do BLE adiciona, reduzindo para menos de 33 bytes de informação a ser transportada. Com isso, a utilização de técnicas de compressão de cabeçalhos se tornou de extrema importância, para se ter uma comunicação mais rápida e energeticamente eficiente.

[21] A pseudoconexão permite ao roteador reservar largura de banda para certo fluxo de pacotes de uma origem x até um destino y, podendo ter uma resposta em tempo real ou não. Com isso é possível manter uma flexibilidade no protocolo ao poder estabelecer prioridades aos pacotes.
[22] Exemplo de endereço IPv6: 2017:07cc:0000:0000:13e6:2681:2b38:ee42. Outro modo de se escrever esse mesmo endereço é quando há grupos de zeros seguidos no meio do endereço, de ser substituído por dois dois-pontos, como o exemplo a seguir: 2017:7cc::13e6:2681:2b38:ee42.

O 6LoWPAN é um protocolo definido pelo IETF, inicialmente apresentado no RFC 4944 de 2007, com o intuito de definir a transmissão de pacotes IPv6 sobre redes IEEE 802.15.4. O padrão mais atual é o RFC 8066 de 2017 que estabelece como deve ser a troca de pacotes e compressão dos datagramas IPv6 para qualquer rede pessoal de baixo consumo (Low-Power Wireless Personal Area Network – LoWPAN), englobando nesse caso também as redes que utilizam Bluetooth Low Energy.

O Bluetooth SIG lançou um Perfil de Suporte ao Protocolo da Internet (Internet Protocol Support Profile - IPSP), que executa na camada de enlace (Link Layer) no controlador e permite estabelecer uma conexão para o transporte de pacotes IPv6. Para implantar o IPv6 sobre o BLE, algumas modificações na pilha do BLE foram feitas. A principal modificação é possuir uma pilha IPv6 funcionando em paralelo com uma pilha GATT no lado do hospedeiro. A imagem a seguir ilustra essa nova pilha.

Figura 8 Pilha BLE com 6LoWPAN



Fonte: (WOOLEY, 2015, p.32)

A pilha relacionada ao GATT é necessária para realizar a descoberta de nós que suportam IPSP. A camada do 6LoWPAN[23], abaixo da camada do IPv6, realiza

[23] A camada do 6LoWPAN no IEE 802.15.4 é levemente diferente, pois ela está localizada diretamente entre a camada de rede (IPv6) e camada de enlace (Link Layer), portanto o 6LoWPAN do IEE 802.15.4 também deve contar com as funções de fragmentação de desfragmentação de

as compressões de cabeçalhos e pacotes do IPv6, para que eles possam ser enviados por BLE. O IPSP rodando na camada de enlace (Link Layer) é o responsável por realizar uma conexão BLE de acordo com os padrões do 6LoWPAN.

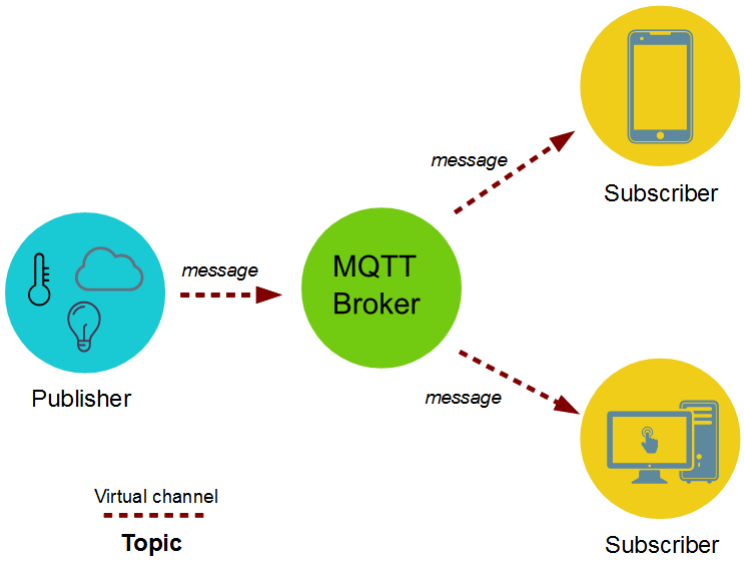
As principais funções da camada do 6LoWPAN para o BLE são as seguintes:

a) Compressão/descompressão de cabeçalhos IPv6 e UDP;
b) Manter as interfaces do 6LoWPAN e realizar o mapeamento delas para os canais do L2CAP;
c) Criar a interface de pilhas de IP;

2.4 MESSAGE QUEUING TELEMETRY TRANSPORT – MQTT

Desenvolvido em 1999 pela IBM com o objetivo de ser um protocolo mais eficiente que os existentes do ponto de vista de largura de banda e de energia, o MQTT é um protocolo (padrão ISO/IEC PRF 20922) de mensagens leves baseado em publicação e subscrição (Publish and Subscriber) que roda sob o protocolo TCP/IP, como indica a figura 9 com um exemplo de topologia.

Figura 9 Topologia de rede MQTT



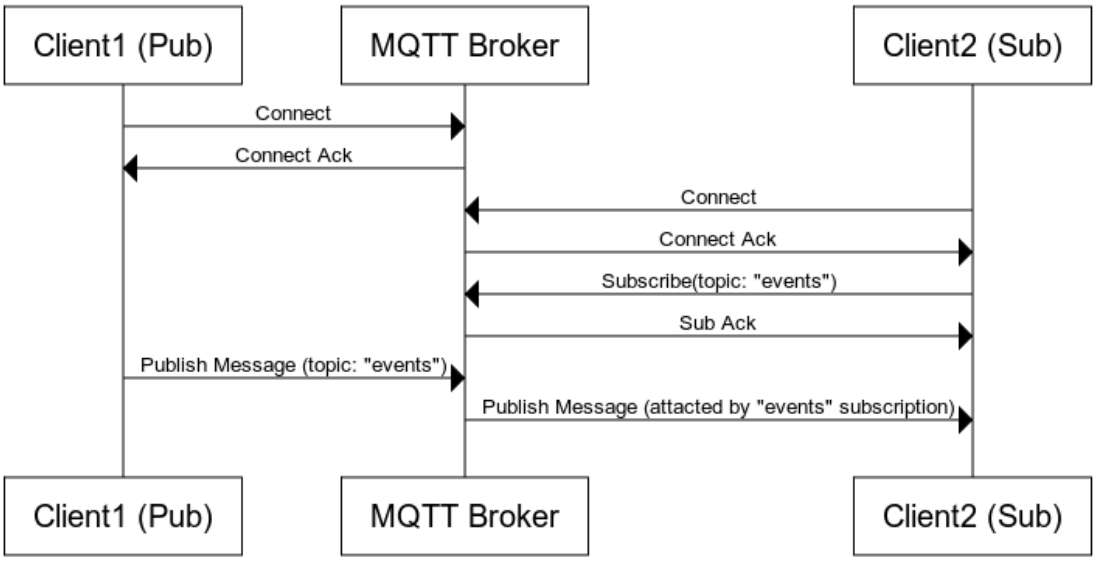
Fonte: (Azzola, 2015)

pacotes. Mas o RFC7668 que especifica o IPv6 diretamente ao BLE, não contem essa exigência pois a parte de fragmentação e desfragmentação estão na camada de L2CAP do BLE, fazendo com que o 6LoWPAN do BLE trate apenas as compressões e descompressões de cabeçalho, o resto é feito pelas camadas já existentes.

O MQTT foi projetado para redes não confiáveis e de alta latência, destinado principalmente para sensores e pequenos dispositivos móveis com uma pequena largura de banda de conexão com a internet. A conexão do cliente ao broker possui uma opções de login (usuário e senha) e uso de criptografia (SSL/TLS).

O padrão de troca de mensagens no MQTT é o publicador/subscritor. Neste padrão, quando um nó da rede deseja receber uma determinada informação, ele deve realizar uma subscrição a um broker em um determinado tópico. Quando outro nó da rede realiza uma publicação para o broker, ele repassa a mensagem de publicação todos os nós que estiverem subscritos naquele tópico. Estes passos da comunicação estão ilustrados na figura 10. Apesar do broker parecer como um elo fraco na rede ao centralizar as comunicações, ele permite um desacoplamento entre os nós que estão trocando informações, algo não possível em modelos de comunicação do tipo cliente/servidor.

Figura 10 Passo a passo para uma publicação MQTT



Fonte: (PIERRE-LUC, 2015)

2.5 RTOS PARA IoT

Reforçando o que foi apresentado na introdução, com a chegada da Internet das Coisas, em pouco tempo diversos equipamentos que jamais foram pensados

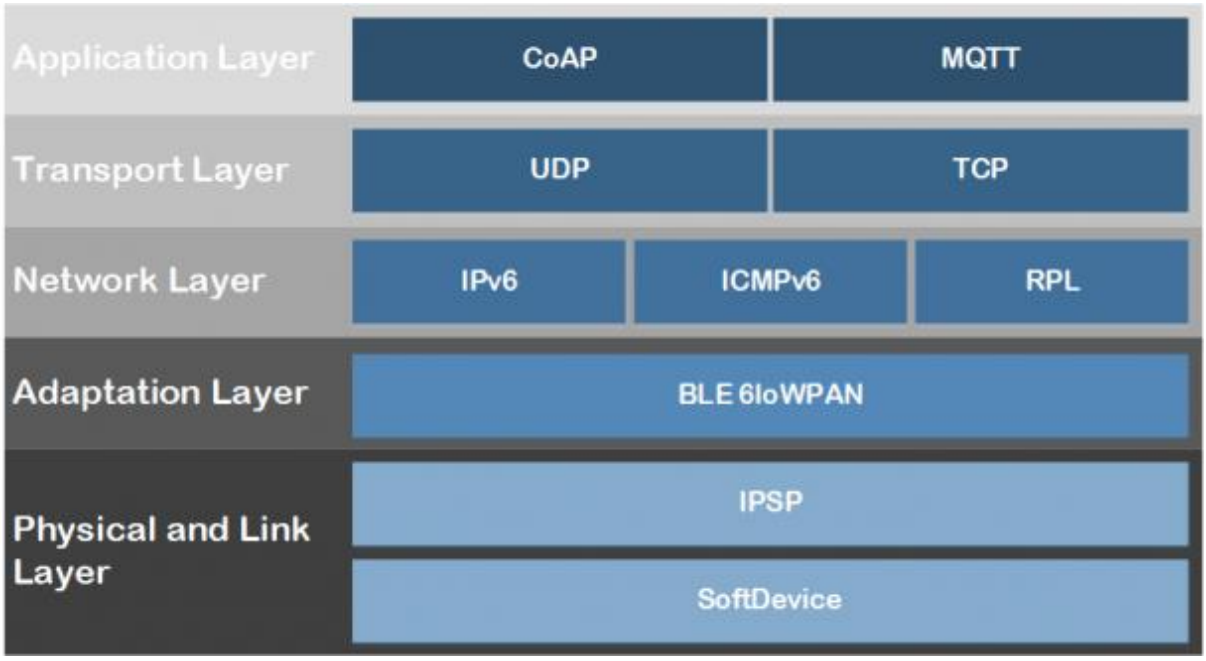
com a possibilidade de se ter acesso à internet, virão a ter. Auxiliado pelo Protocolo de Internet versão 6 que disponibiliza uma quantidade absurda de endereços, o crescimento de equipamentos conectados à internet será exponencial. E realizar uma conexão para transferência de dados eficiente será de suma importância.

Foi com esse pensamento, que várias empresas de Sistemas Operacionais de Tempo Real (Real-Time Operating System – RTOS) para microcontroladores começaram a adicionar um suporte enorme para as conexões de rede voltadas para IoT. Atualmente há 4 grandes Sistemas Operacionais que implementam modelos de pilha de comunicação para o BLE com protocolos semelhantes aos da figura 11. Os RTOSs de código aberto (Open Source) mais famosos para IoT atualmente são:

a) Contiki OS;
b) Zephyr OS;
c) RIOT OS;
d) Mbed OS.

Como os dois OSs utilizados nesse trabalho foram o Contiki e o Zephyr, os dois próximos sub-capitulos serão uma breve explicação das características de cada um.

Figura 11 Exemplos de protocolos implementados por RTOSs



Fonte: (BAKKE, 2015)

2.4.1 Contiki OS

O Contiki[24] é um sistema operacional para sistemas com pouca memória e acesso a rede, voltado para equipamentos que possuem um foco em redes de comunicação sem fio de baixo consumo de energia e Internet das Coisas. Escrito em linguagem C e é muitas vezes referenciado como o “Sistema Operacional de Código Aberto para a Internet das Coisas” (Open Source OS for the Internet of Things).

Como o Contiki foi projetado para pequenos equipamentos, ele opera com menos de 10kB de RAM e 30kB de ROM, e ainda provê uma pilha IP completa com suporte para UDP, TCP e HTTP, entre outros protocolos. Em adição, a pilha IP do Contiki tem suporte pra todos os recentes padrões de protocolos da IETF, tais como 6LoWPAN, camada de adaptação, RPL, CoAP, entre outros.

Contiki roda numa grande variedade de pequenas plataformas de hardware, desde processadores 8051, MSP430, AVR e uma grande variedade de dispositivos ARMs. Na tabela 1 estão listadas todas as plataformas com 100% de suporte do Contiki OS.

Para a utilização do Contiki com o Nordic nRF52832, que foi utilizado neste projeto, é necessário também gravar em parte da memória do nRF52 o SoftDevice. SoftDevice é um software binário e pré-compilado desenvolvido pela Nordic Semiconductor para seus equipamentos, responsável por gerenciar as duas camadas mais baixas da pilha BLE, indicada na figura 11 a cima. Portanto, os sistemas operacionais que rodam em seus equipamentos podem optar em implementar uma pilha completa, ou implementar até o 6LoWPAN e deixar o gerenciamento de fragmentações de pacotes, envios e recebimentos para o SoftDevice.

O Contiki implementa, para o nRF52832, todo o gerenciamento desde o protocolo MQTT até o 6LoWPAN, mas necessita do SoftDevice gravado em parte da memória para realizar os envios e recebimento de pacotes pelo BLE.

[24] O Contiki teve sua primeira versão lançada em 2003 e foi desenvolvida desde então por vários grupos, tais como: Texas Instruments, Atmel, Cisco, ENEA, ETH Zurich, Redwire, RWTH Aachen University, Oxford University, SAP, Sensinode, Swedish Institute of Computer Science, ST Microelectronics, Zolertia, e mais alguns outros.

Tabela 1 Plataformas que o Contiki OS suporta

MCU/SoC	Radio	Platforms
TI CC2538	Integrated / CC1200	RE-Mote
nRF52832	Integrated	nRF52 DK
RL78	ADF7023	EVAL-ADF7023DB1
TI CC2538	Integrated	cc2538dk
TI MSP430x	TI CC2420	exp5438, z1
TI MSP430x	TI CC2520	wismote
Atmel AVR	Atmel RF230	avr-raven, avr-rcb, avr-zigbit, iris
Atmel AVR	TI CC2420	micaz
Freescale MC1322x	Integrated	redbee-dev, redbee-econotag
TI MSP430	TI CC2420	sky
TI MSP430	TI CC1020	msb430
TI MSP430	RFM TR1001	esb
Atmel Atmega128 RFA1	Integrated	avr-atmega128rfa
Microchip pic32mx795f512l	Microchip mrf24j40	seed-eye
TI CC2530	Integrated	cc2530dk
6502	-	apple2enh, atari, c128, c64
Native	-	native, minimal-net, cooja

Fonte: Contiki Hardware (disponível em: http://www.contiki-os.org/hardware.html)

2.4.2 Zephyr OS

O Zephyr[25] é um sistema operacional de tempo real para dispositivos conectados e com recursos limitados (bateria, memória e processamento) de várias arquiteturas de hardware. Lançado em 2015, o Zephyr desde a sua criação foi desenvolvido voltado ao mundo do IoT. Pensado para equipamentos com pouca memória, o Zephyr executa em microcontroladores de até 8kB de memória RAM, possui pilha IP completa, com suporte para Bluetooth, BLE, WI-FI e IEEE 802.15.4. Suporta também vários padrões de comunicação como 6LoWPAN, CoAP, IPv4, IPv6 e NFC.

[25] O Zephyr lançado em 2015 pelo nome de Rocket, atualmente conta com uma equipe de desenvolvedores das seguintes empresas: Intel, NXP Semiconductors, Synopsys, and UbiquiOS Technology.

O Zephyr pode ser executado numa gama enorme de plataformas, desde arquitetura ARM, como também x86, ARC, NIOS II e XTENSA. Todas as plataformas suportadas estão mostradas na tabela 2 a seguir:

Tabela 2 Plataformas que o Zephyr OS suporta

x86 Boards	Arduino/Genuino 101	X86 Emulation (QEMU)
	Galileo Gen1/Gen2	Quark D2000 Development Board
	MinnowBoard Max	tinyTILE
ARM Boards	96Boards Carbon	ST Nucleo F334R8
	96Boards Carbon nRF51	ST Nucleo F401RE
	96Boards Neonkey	ST Nucleo F411RE
	96Boards Nitrogen	ST Nucleo F412ZG
	Arduino/Genuino 101 (BLE)	ST Nucleo F413ZH
	Arduino Due	ST Nucleo L432KC
	CC2650 SensorTag	ST Nucleo L476RG
	CC3220SF LaunchXL	OLIMEX-STM32-E407
	Curie (BLE)	OLIMEX-STM32-P405
	ST Disco L475 IOT01	OLIMEXINO-STM32
	EFM32WG-STK3800	ARM Cortex-M3 Emulation (QEMU)
	NXP FRDM-K64F	SAM4S Xplained
	NXP FRDM-KL25Z	SAM E70 Xplained
	NXP FRDM-KW41Z	STM3210C-EVAL
	Hexiwear	STM32373C-EVAL
	Hexiwear KW40Z	STM32 Minimum Development Board
	ARM V2M MPS2	ST STM32F3DISCOVERY
	MSP-EXP432P401R LaunchXL	STM32F411E-DISCO
	nRF51-PCA10028	ST STM32F412G Discovery
	nRF51-VBLUno51	ST STM32F429I-DISC1 Discovery board
	nRF52840-PCA10056	ST STM32F469I Discovery
	Redbear Labs Nano v2	ST STM32F4DISCOVERY
	nRF52-PCA10040	ST STM32L496G Discovery
	nRF52-VBLUno52	NXP USB-KW24D512
	ST Nucleo F030R8	ARM V2M Beetle
ARC Boards	Arduino/Genuino 101 (Sensor Subsystem)	
	DesignWare(R) ARC(R) EM Starter Kit	
NIOS II Boards	Altera MAX10	
XTENSA Boards	ESP32	
	Xtensa Emulation (QEMU)	
	Xtensa simulator	

Fonte: Autores com dados obtidos de Zephyr Project

Diferente do Contiki OS, o Zephyr OS implementa toda a pilha BLE para o funcionamento no nRF52832, portanto não é necessário realizar a gravação anterior do SoftDevice no CI. Embora ele suporte o nRF52, a utilização da internet pelo BLE não é facilmente configurável e várias modificações no código exemplo foram necessárias para se ter um envio satisfatório de publicações em MQTT.

3 METODOLOGIA DO TRABALHO

Neste capítulo serão descritos os equipamentos utilizados, bem como os softwares e configurações necessárias para realizar a comunicação IPv6 por BLE. Primeiramente a comunicação foi realizada entre duas máquinas virtuais rodando Sistema Operacional Linux e em seguida entre uma máquina virtual e o CI nRF52832. A validação da conexão IPv6 por BLE foi feita inicialmente através de comandos de PING entre as máquinas e posteriormente realizando publicação MQTT.

Os equipamentos utilizados nesse trabalho foram:

a) Computador com adaptador Bluetooth AR3012[26] integrado;
b) Três máquinas virtuais executando, em cada uma, o Linux Mint 18.2 Sony com o Kernel versão 4.8;
c) Adaptador Bluetooth BCM20702[27] com conexão USB;
d) Gravador in-circuit ST-LINK/V2 da STMicroeletronics para realizar a gravação do CI nRF52832;
e) Módulo nRF52832 com conexão PTH;
f) Osciloscópio e analisador lógico USB Analog Discovery da DIGILENT;
g) Placa de circuito impresso (PCB) desenvolvida para este projeto[28].

Os principais softwares utilizados neste projeto foram:

a) gcc-arm-none-eabi, compilador e montador de linguagem C para ARM. Utilizado nas compilações dos firmwares para o módulo nRF52.
b) BlueZ, para o gerenciamento do 6LoWPAN no Linux;
c) RADVD[29], para gerenciar e alocar os endereços IPv6s aos nós da rede;
d) Wireshark, para analisar o tráfego de pacotes entre as redes BLE e Ethernet;
e) OpenOCD[30], para realizar as gravações de firmware no nRF52832 com o ST-LINK/V2;
f) WaveForms[31], para análise das leituras do Analog Discovery;

[26] AR3012 é um adaptador Bluetooth 4.0 da ATHEROS.
[27] BCM20702 é um adaptador Bluetooth 4.0 da Broadcom.
[28] A PCB desenvolvida foi um equipamento essencial para o projeto, mas ela só vai ser detalhada no próximo capítulo que envolve o desenvolvimento do trabalho.
[29] RADVD (Router Advertisement Daemon) é o serviço de semelhante ao DHCP do IPv4 mas para IPv6.
[30] OpenOCD é um software para gravação de código aberto para diversas plataformas e arquiteturas

g) Mosquitto MQTT, broker da comunicação MQTT;
h) Oracle VM VirtualBox, para gerenciar as máquinas virtuais.

3.1 CONFIGURAÇÃO DO AMBIENTE PARA OS TESTES

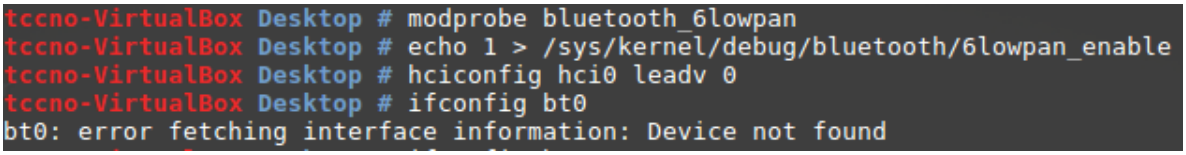
Para iniciar o trabalho, foi realizado um teste para verificar se haveria uma comunicação satisfatória utilizando os dois adaptadores Bluetooth 4.0 (adaptador interno do computador e o externo USB). Para efetuar esse teste de comunicação, foram criadas duas máquinas virtuais com o sistema Linux Mint, uma denominada de gateway e a outra denominada de nó. Em ambas as máquinas foi instalado o software BlueZ.

O objetivo do teste é a realização de PING entre as maquinas por ipv6 sobre BLE. Para isso, os passos são: instalar adaptadores e drivers, realizar conexão BLE, identificar adaptadores de rede, inserir o comando PING. Com o sucesso do teste, definiu-se então a rede utilizada no restante do trabalho.

Na maquina nó foram executados alguns comandos para que o adaptador Bluetooth conectado naquela máquina ativasse o modo de anúncio (advertising) do BLE. A imagem 12 a seguir ilustra os passos necessários:

a) Inserir o módulo do 6LoWPAN no Linux;
b) Inicializar o 6LoWPAN;
c) Ativar o modo de anúncio, leadv = Low Energy Advertising.

Figura 12 Comandos para ativar o modo de anúncio 1/2



Fonte: Autores

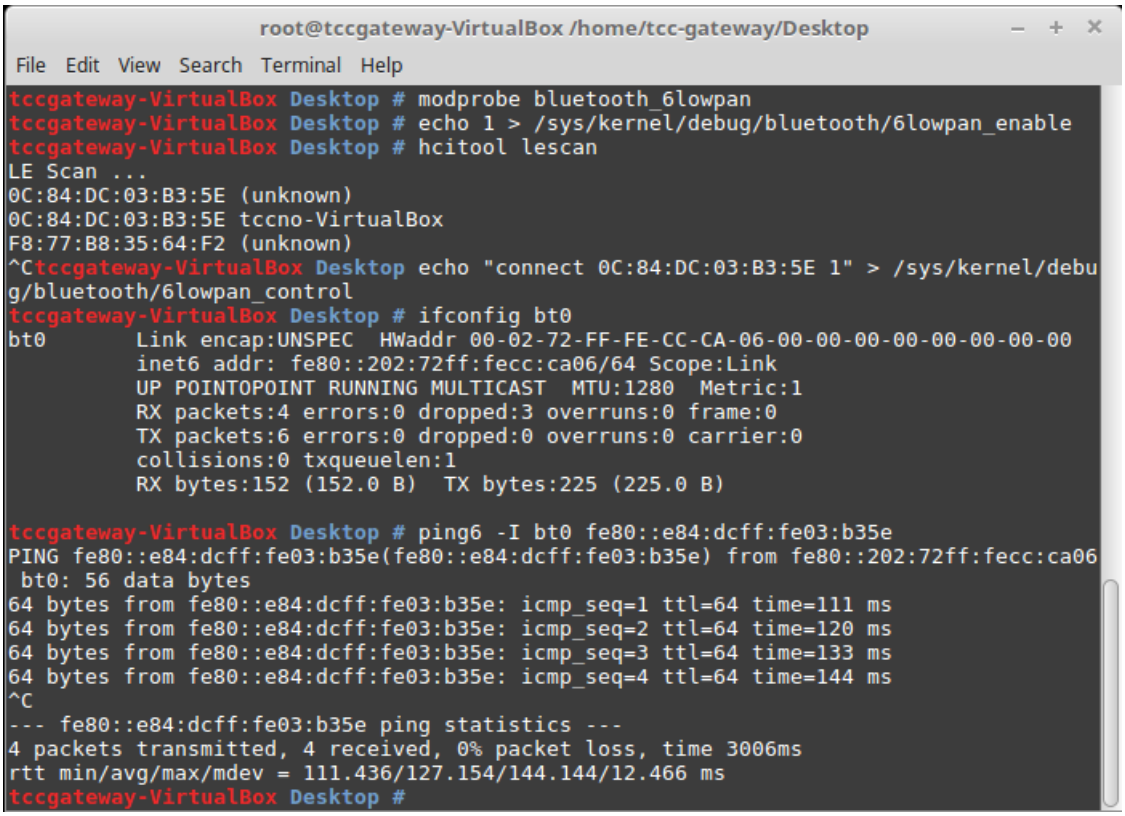
Na figura 12 observamos que após inserir o ultimo comando, o adaptador não foi encontrado, isso ocorre devido à forma como o BLE funciona. Após o periférico

[31] WaveForms é um software da DIGILENT que interagem com com seus osciloscópios USB.

iniciar seu anúncio, o dispositivo central é que deve iniciar a conexão, só após isso é que o adaptador bt0 aparece. Depois que o anuncio é feito, é necessário configurar o nosso gateway. Essa configuração foi feita a partir dos seguintes passos (ilustrados pela figura 13):

a) Inserir o módulo do 6LoWPAN no Linux;
b) Inicializar o 6LoWPAN;
c) Ativar o escanemanto de dispositivos em anúncio, lescan = Low Energy Scan;
d) Depois de aparecer o dispositivo desejado, é necessário realizar uma conexão com o mesmo;
e) É possível, portanto, observar o adaptador bt0, gerenciado pelo BlueZ;
f) Executar o comando PING[32] com o endereço da máquina nó para validar a conexão.

Figura 13 Comandos para conexão de um dispositivo em anúncio

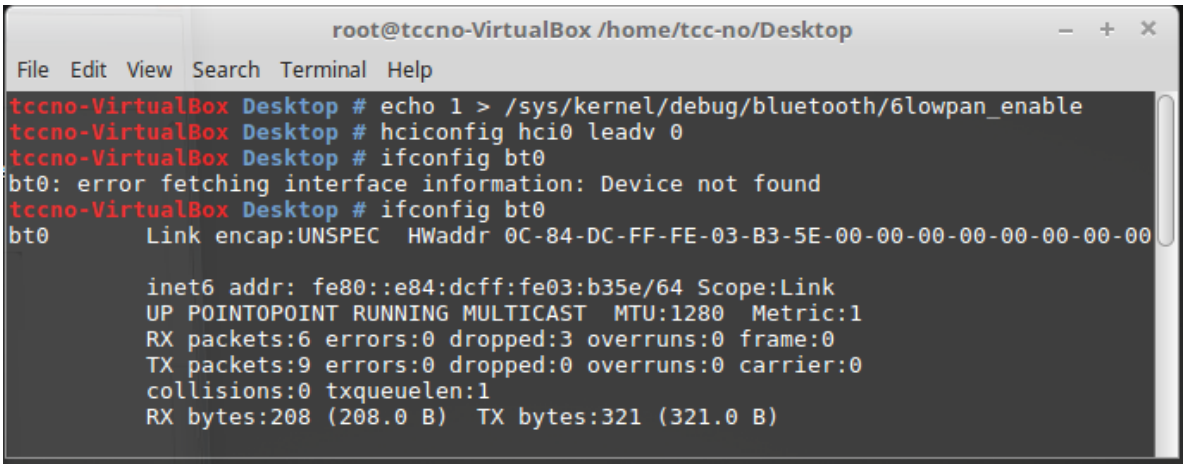


Fonte: Autores

[32] O comando PING executado necessitou do parâmetro "-I bt0", esse parâmetro é necessário quando o IP do PING é um IP local, e não global, sendo necessário especificar o adaptador que será a rota para a localização daquele endereço IP. Todos os endereços de IPv6 que inicializam com "fe80" são endereços locais.

Podemos observar a alta latência na comunicação mesmo com os dispositivos pertos um do outro, isso é característica da conexão por BLE. Após a realização da conexão (com o comando do passo d), o adaptador de rede bt0 também aparece na máquina nó. Foi dessa forma que foi descoberto o IP de link local[33] para que fosse possível executar o comando PING. A imagem 14 ilustra o terminal da máquina nó. Ressalta-se que entre os dois ifconfig inseridos foi realizada a conexão pelo dispositivo central.

Figura 14 Comandos para ativar o modo de anúncio 2/2



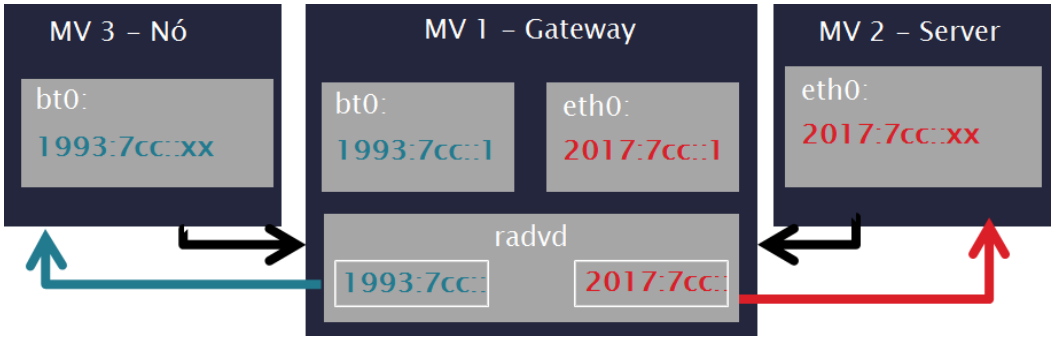
Fonte: Autores

Com isso, os passos para a correta conexão foram definidos e certificamos que o funcionando do BlueZ estava adequado. Passamos a configurar, então, uma interligação entre as redes Bluetooth e Ethernet pelo gateway. Para isso foi necessário utilizar o serviço RADVD do Linux. Esse serviço é responsável por realizar os anúncios dos endereços de rotas IPv6 e por anunciar os prefixos das rotas utilizando o NDP[34], assim todos os computadores e equipamentos conectados a essas redes passaram a possuir um endereço IPv6 global fornecido pelo RADVD.

[33] O IP de link local, que sempre inicia em fe80, serve pra trocar pacotes apenas entre os dois nós da rede que existe a conexão. Se um adaptador possui apenas o IP local, ele não consegue acessar outros nós da rede, apenas o que ele está conectado. O RADVD suprime esse problema, entregando um IP global aos nós e realizando a interligação entre as diferentes redes.
[34] O NDP (Neighbor Discovery Protocol) é um protocolo do TCP/IP utilizado com o IPv6 para realizar a descoberta de vizinhos da rede.

A figura 15 ilustra a rede, definida, que foi utilizada ao longo de todo o projeto. Esta contendo endereços IPv6 para o RADVD na máquina virtual gateway gerenciar. Foi configurado, portanto, um prefixo de endereço IP para cada adaptador do gateway. Os endereços definidos foram os seguintes: prefixo 1993:7cc para os equipamentos conectados no adaptador bt0 e prefixo 2017:7cc para o adaptador eth0. Com isso todos os computadores conectados na mesma rede que o gateway receberam um endereço de IP válido com o prefixo 2017:7cc e com o uso do RADVD os equipamentos da rede eth0 puderam acessar os da bt0 e vice-versa.

Figura 15 Rede gerenciada pelo RADVD

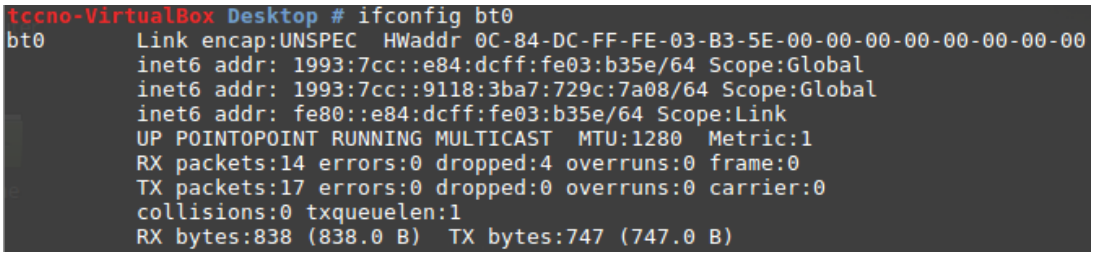


Fonte: Autores

Após a configuração e inicialização do RADVD foi possível observar que todos os computadores ganharam endereços IPv6 de suas respectivas redes. A máquina virtual nó recebeu um endereço da rede 1993:7cc como mostra a figura 16. O arquivo de configuração fica localizado em “/etc/radvd.conf” no Linux. No apêndice A é possível ver o arquivo de configuração utilizado neste projeto.

Inclusive o computador com o Windows (host das maquinas virtuais) recebeu um endereço 2017:7cc do gateway, visto que está na mesma rede. Dessa forma podemos verificar se a máquina nó que possui apenas a conexão por BLE consegue ser acessada pelo computador Windows que possui apenas a conexão WI-FI.

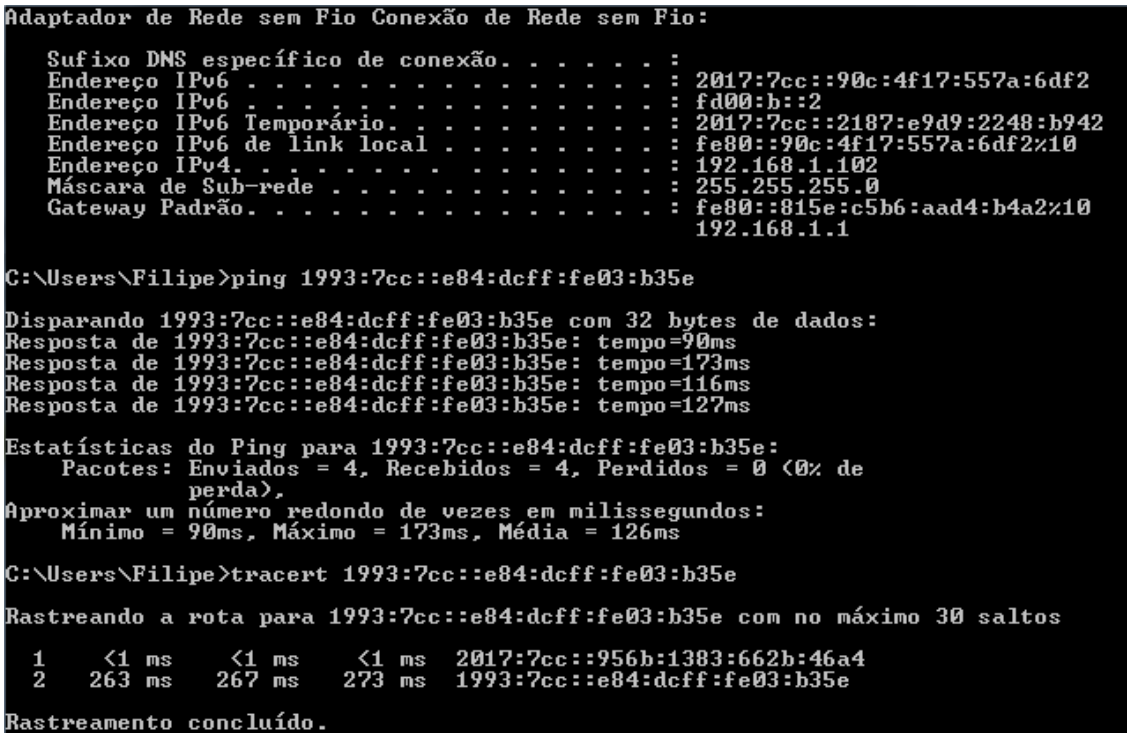
Figura 16 Endereço IPv6 na MV nó



Fonte: Autores

Na figura 17 podemos observar o endereço IPv6 que o Windows recebeu por estar na mesma rede que o gateway. Em seguida realizamos um PING para testar a conexão e usamos o comando de traçar a rota (tracert). Com ele podemos ver que o primeiro salto é extremamente rápido, pois é a comunicação até o gateway que está na mesma rede. Já o segundo salto é um pouco mais lento, por se tratar da comunicação pelo BLE.

Figura 17 Terminal do Windows realizando PING em dispositivo BLE



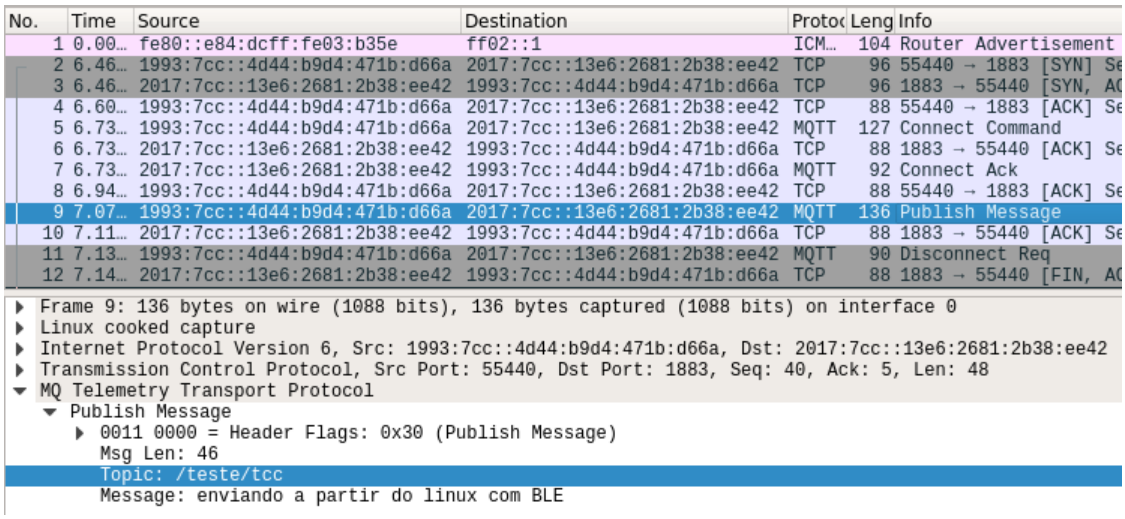
Fonte: Autores

Para concluir a configuração e validação do ambiente, um último teste realizado foi o envolvendo o MQTT. Foi instalado o broker mosquitto na terceira máquina virtual Linux denominada server, e na máquina nó foi instalado o mosquitto publisher, ambos os softwares adquiridos através dos comandos do Linux de acesso ao repositório.

A configuração da rede ficou exatamente como na Figura 15 anterior, o broker (MV server) possuindo apenas conexão cabeada e o publicador (MV nó) somente a conexão por BLE. O gateway serviu como o elo central da comunicação, e nele foi analisada a troca de pacotes durante uma publicação MQTT.

Com o software Wireshark foi analisada a troca de pacotes IPv6 entre as redes bt0 e eth0 como ilustra a Figura 18. Observando a imagem é possível perceber toda a troca de pacotes para uma publicação bem sucedida, desde abertura da conexão TCP, conexão MQTT, publicação MQTT, e desconexão. O tempo total para a publicação é de cerca de 700 ms, devido a troca de pacotes pelo BLE.

Figura 18 Troca de pacotes IPv6 de uma publicação MQTT com o Linux



Fonte: Autores

Dessa forma, finalizamos todos os testes e configurações para a correta comunicação no gateway e broker. Passamos, portanto, a etapa de testes do nRF52832. Temos em mente que qualquer erro ou mal funcionamento da

comunicação daqui pra frente acontecerá por uma má configuração do módulo. O que nos poupa tempo, pois não será preciso ficar procurando os erros nos dois lados da comunicação, apenas no nó de envio dos dados.

3.2 nRF52

O Circuito Integrado nRF52832 da Nordic Semiconductor foi lançado em 2016 e superou em vários aspectos seu antecessor nRF51822. A tabela 3 a seguir mostra uma comparação entre ele, seu sucessor e um modelo bem famoso da Texas Instruments que também possui BLE. É possível observar que a frequência de operação do nRF52 é 4 vezes superior ao nRF51, e o consumo de energia cai quase pela metade, mostrando que o nRF52832 é um excelente dispositivo para ser utilizado quando o consumo de energia é de suma importância.

Tabela 3 Comparação entre três SoCs com BLE

	nRF51822 / nRF51422	CC2640 / CC2650	nRF52832
Manufacturer	**Nordic Semiconductor**	**Texas Instruments**	**Nordic Semiconductor**
Processor	Cortex-M0 16MHz	Cortex-M3 at 48MHz	Cortex-M4F 64MHz
Flash	128kB / 258kB	128kB	512kB
RAM	16kB / 32kB	20kB	64kB
RX Sensitivity	-93dBm	-97dBm	-96dBm
Protocol Support	Bluetooth Smart (BLE), ANT, 2.4GHz	Bluetooth Smart (BLE), 2.4GHz, Zigbee	Bluetooth Smart (BLE), ANT, 2.4GHz
ADC	10-bit	12-bit 200ksps	12-bit 200ksps
Peripherals	SPI, I2C, UART, Quadrature Demodulator	Sensor Controller Engine, Comparators, I2C, SPI, TDC	SPI, I2C, UART, I2S, DMA, Quadrature Demodulator
BLE Output Power	-20dBm to +4dBm	Up to 5dBm	-20dBm to +4dBm
Power Consumption (TX/RX) @ 0dBm	9.7mA / 8mA	6.1mA / 5.9mA	5.5mA / 5.5mA
Package	6x6mm QFN, 3.5×3.83mm WLCSP	4x4mm, 5x5mm, 7x7mm QFN	6x6mm QFN, 3.0×3.2mm WLCSP

Fonte: (Argenox Technologies, 2015)

A Nordic Semiconductor é pioneira em comunicação com baixo consumo de energia. Em 2015, a Aislelabs, empresa voltada para comunicações e localização utilizando tecnologia sem fio, fez um comparativo com quatro diferentes CIs de BLE.

O teste foi uma medição do consumo de energia e estimativa de tempo que diversas baterias durariam dos periféricos realizando anúncios periódicos. Foi variado o intervalo dos anúncios, de 100ms, 645ms e 900ms e mantido uma potência de -12dBm, dando um raio de cobertura de aproximadamente 15 metros, para cada um dos diferentes CIs[35].

A figura 19 mostra os resultados dos testes. Os CIs utilizados foram os seguintes: TI CC254x da Texas Instruments, nRF51822 da Nordic Semiconductor, BLE112/BLE113 da Bluegiga, e o controlador proprietário da Gimbal. Podemos observar que mesmo não se tratando do novo modelo da Nordic, o desempenho dele foi excelente, pois com uma bateria de apenas 1000 mAh ele se manteria ligado por quase 4 anos, fazendo anúncio a cada 900ms.

Tabela 4 Consumo de energia de difersos CIs em modo de anúncio

Chipset Battery LIfe

Aislelabs

		cr2032 small 240mah	cr2450 medium 620mah	cr2477 big 1000 mah
Chipset	**Advertisement Interval**	**Est. Battery cr2032**	**Est. Battery cr2045**	**Est. Battery cr2477**
Gimbal	100ms	1.0 month	2.5 month	4.0 month
Gimbal	645ms	n/a	n/a	n/a
Gimbal	900ms	n/a	n/a	n/a
Nordic	100ms	1.1 months	2.8 months	4.5 months
Nordic	645ms	5.4 months	13.9 months	22.4 months
Nordic	900ms	11.0 months	28.7 months	46.3 months
Bluegiga	100ms	0.6 months	1.6 months	2.6 months
Bluegiga	645ms	5.8 months	15.1 months	24.3 months
Bluegiga	900ms	13.9 months	35.8 months	57.1 months
TI CC254x	100ms	0.7 months	1.8 months	2.9 months
TI CC254x	645ms	4.3 months	11.2 months	18.0 months
TI CC254x	900ms	5.6 months	14.3 months	23.1 months

Fonte: (Aislelabs, 2015)

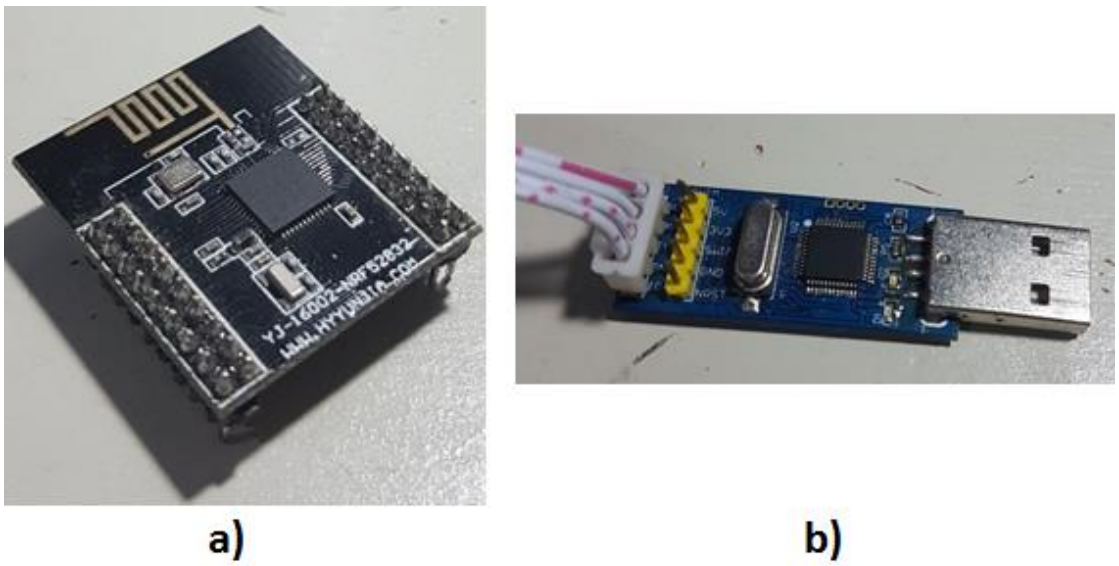
[35] A técnica de manter o dispositivo sempre em anúncio é normalmente utilizada em equipamentos de rastreamento ou identificação.

3.2.1 Configuração do nRF52

Com as configurações do Linux finalizadas e um lado da comunicação validado, teve início o trabalho em volta do módulo nRF52832. O módulo utilizado, ilustrado na figura 19-a, é o modelo mais básico encontrado no mercado, com o CI nRF52832, componentes básicos para seu funcionamento e a sua antena. A placa do módulo conta com duas barras de pinos 2x9 180º para as conexões de energia do módulo e portas do CI. O esquemático do módulo pode ser encontrado no anexo A.

A configuração do módulo envolve a gravação de firmwares com o Contiki OS e o Zephyr OS no nRF52 e verificação do correto funcionamento da comunicação. Para averiguar a comunicabilidade, os firmwares desenvolvidos tentam fazer publicações periódicas em MQTT, ocorrendo tal comunicação, a configuração está concluída. Antes de realizarmos a gravação do firmware o software de gravação openOCD foi instalado para ser utilizado em conjunto ao gravador in-circuit ST-LINK/V2 da STMicroeletronics, ilustrado na figura 19-b.

Figura 19 a) Módulo nRF52832 com dual header, b) Gravador ST-LINK/V2



a) b)

Fonte: Autores

Para a instalação do openOCD foi clonado o repositório oficial no gitHub. Entretanto o openOCD nativo não tem suporte ainda para o nRF52832, por se tratar de um CI relativamente novo. Por isso, a Nordic lançou um patch para o openOCD. Para a instalação do patch é necessário puxar (comando git pull) o patch do endereço da web fornecido pela Nordic. Esse pull gera alguns conflitos de arquivos que foram resolvidos manualmente. Após a resolução dos conflitos foi possível instalar o software (com o comando make install) e utilizá-lo com sucesso.

3.2.1.1 Contiki OS

Passamos ao processo de configuração do Contiki OS versão 3.x. De acordo com a documentação, foi observado que era necessário primeiramente instalar o SoftDevice. É possível baixar o SoftDevice, responsável pelo gerenciamento das duas camadas mais inferiores do BLE, diretamente do repositório oficial do nRF5 SDK mantido pela Nordic. O repositório nRF5 SDK também deve estar presente e deve ser configurado o endereço do diretório dele antes de realizar uma compilação do Contiki, como esta mostrado no apêndice B.

Para o teste de gravação do Contiki foi compilado um exemplo de beacon, que utiliza o BLE em modo de anúncio. Para a gravação desse exemplo é necessário primeiramente gravar o SoftDevice e posteriormente o arquivo .hex compilado do exemplo. Todas as gravações foram feitas utilizando o openOCD com o ST-LINK/V2. Após a gravação foi possível encontrar o dispositivo em anúncio ativando a descoberta no Linux com o comando lescan ilustrado na figura 13 anterior.

Com a gravação funcionando, passamos para um exemplo que utiliza MQTT, que foi feito para ser utilizado com o nRF52DK[36]. Após a sua gravação, foi possível identificá-lo com a descoberta de anúncios pelo Linux, mas não foi possível fazer uma conexão. Quando tentávamos conectar, o nRF52 não respondia a nenhum comando. Se sucederam várias tentativas e testes sem sucesso, e a partir de uma pesquisa no fórum da Nordic, foi encontrado uma publicação confirmando que a versão mais atual do Contiki possuía uma incompatibilidade com o nRF52. Um

[36] O nRF52DK é a placa de desenvolvimento oficial da Nordic para o nRF52832. Ela possui botões, LEDs e sensores, junto com um gravador J-TAG integrado.

administrador da Nordic, portanto, postou um link para o repositório do Contiki dele, de uma versão anterior da atual.

Com o novo repositório em mãos, foi possível compilar e testar o exemplo de publicação em MQTT com sucesso. Não foi observada nenhuma diferença entre as duas versões, fora o fato de que com a mais antiga a comunicação funcionava satisfatoriamente. Para finalizar, a única configuração necessária para realizar a publicação em MQTT a partir do Contiki, com o exemplo desenvolvido para o nRF52DK, foi a do endereço IP do broker, configurado no arquivo project-conf.h e o nome de anúncio, modificado no arquivo contiki-conf.h na pasta da plataforma nRF52DK. Os passos e comandos para compilação e gravação de exemplos do Contiki podem ser visualizados no apêndice B.

3.2.1.2 Zephyr OS

Após a configuração do Contiki, começamos os testes utilizando o Zephyr OS versão 1.8. Foi baixado o repositório oficial deles e, diferente do Contiki, ele não possui exemplos especificamente para o nRF52, portanto o teste foi realizado utilizando exemplos genéricos e durante a compilação selecionamos o tipo de plataforma desejada. Primeiramente um exemplo de beacon foi testado também para verificar o funcionamento do BLE. Os comandos para compilação e gravação do Zephyr podem ser observados no apêndice C.

Após a compilação e gravação do exemplo, que não possui a necessidade do SoftDevice previamente gravado na memória do CI[37], o anúncio pelo BLE pôde ser observado pelo Linux. Após este teste, foi compilado e gravado um exemplo do IPSP para Bluetooth. Diferente do Contiki, o Zephyr não possui um exemplo utilizando o MQTT com o BLE, portanto o utilizado foi o IPSP para o BLE e toda a parte de comunicação em MQTT foi adicionada a partir de outro exemplo de MQTT e da documentação do OS.

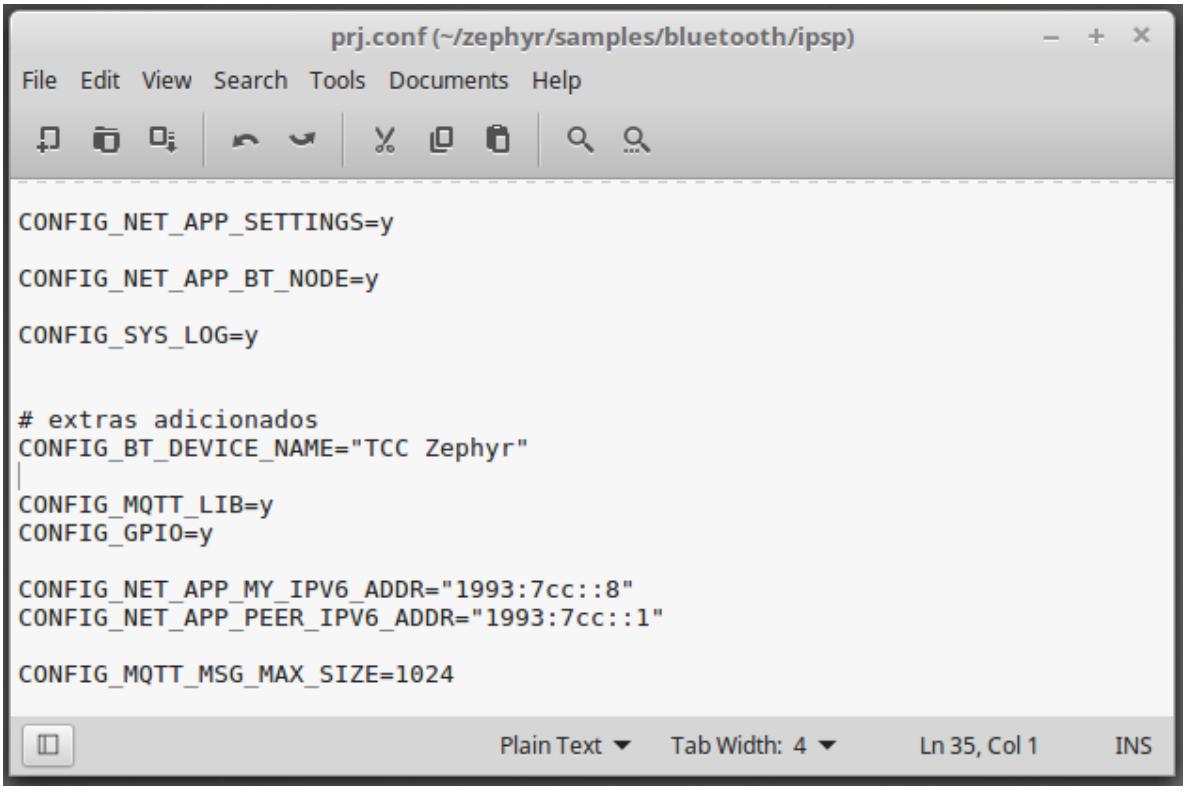
A configuração do Zephyr para o funcionamento do MQTT foi um pouco mais complexa. No arquivo prj.conf[38] foi necessário informar os endereços IP do gateway

[37] Não é necessário o SoftDevice para o Zephyr pois ele implementa todas as camadas

[38] O arquivo de configuração é utilizado durante a compilação para o OS saber quais subsistemas e bibliotecas utilizar e compilar para a plataforma definida. É possível criar um arquivo de configuração manualmente, ou através de uma interface gráfica chamada de menuconfig acessada pelo comando

e do Zephyr. Ao contrário do Contiki que já ganhava endereços IP a partir do RADVD, o Zephyr não implementa essa funcionalidade, o que gerou a necessidade de definir um endereço IP estático para ele. Foi necessário também adicionar nesse arquivo algumas outras configurações como a utilização da biblioteca MQTT, utilização de GPIO e tamanho máximo da mensagem MQTT (necessário para os testes realizados no próximo capítulo). Foi definido também o nome de anúncio. A figura 20 a seguir mostra parte do arquivo de configuração com as informações extras adicionadas.

Figura 20 Parte do arquivo de configuração do Zephyr OS



Fonte: Autores

A configuração do endereço IP do broker foi definida direto no código fonte antes de realizar a conexão em MQTT. Após essas configurações, o projeto foi compilado e gravado com sucesso. Entretanto foi mais demorado para estabelecer a

---

“make menuconfig”, em que é possível selecionar os periféricos utilizados com o auxilio de uma interface gráfica.

comunicação entre o dispositivo e o Linux, pois depois de vários testes sem sucesso foi descoberto que o Zephyr mostra no anúncio um endereço MAC virtual, portanto quando o comando de conexão do 6LowPan é inserido no Linux o parâmetro que para o Contiki e o BlueZ era “1”, deve ser substituído por “2”. O comando em questão é ilustrado na figura 13 como o quarto comando inserido. Após a correta comunicação, o nRF52 executando o Zephyr OS fez uma publicação em MQTT com sucesso.

4 DESENVOLVIMENTO DO TRABALHO

Com a validação das comunicações, entre o Linux e o nRF52 com os dois OSs, começou a análise da comunicação. Para avaliação dos dois sistemas operacionais foi analisado o consumo de energia durante o anúncio do BLE, durante a conexão estabelecida sem troca de dados e durante a publicação. Foi verificado o tempo total de transmissão para publicações grandes e pequenas variando a distância e obstáculos entre o periférico e o computador. Foi observado também como é feita a comunicação (a troca de pacotes) com o software Wireshark e os tempos de compressão e descompressão dos cabeçalhos IPv6 pelo nRF52.

4.1 DESENVOLVIMENTO DA PCB

Para realizar todos os testes informados, uma placa de circuito impresso foi confeccionada. A PCB serve para realizar a conexão do módulo nRF52832 com a alimentação de energia 3,3V, LEDs, botões e uma unidade sensora. A PCB também possui um módulo carregador de baterias Íon de Lítio modelo TP4056 e uma conexão para bateria. Para realizar a medição do consumo de energia um resistor de 1Ω foi adicionado em série com a entrada de energia do módulo, e dois pinos header de 180º inseridos entre o resistor para o fácil acoplamento do osciloscópio USB.

Os LEDs e botões da PCB estão nas mesmas portas de propósito geral do nRF52 que a placa de desenvolvimento do nRF52 (nRF52DK – nRF52 Development kit). Dessa forma é mais fácil executar exemplos para a nRF52DK nesta placa desenvolvida sem se preocupar inicialmente com as portas utilizadas. Foi adicionado pinos header de 180º no catodo de todos os LEDs, para poder ser analisado pelo osciloscópio USB os tempos de compressão e descompressão dos cabeçalhos IPv6.

A PCB conta também com um conector para gravação do nRF52832, que é conectado diretamente ao gravador ST-LINK/V2. A unidade sensora da placa é uma leitura da tensão da bateria, através de um divisor resistivo[39] com um filtro passa baixas para reduzir o nível de ruído na entrada de conversão analógica-digital do CI.

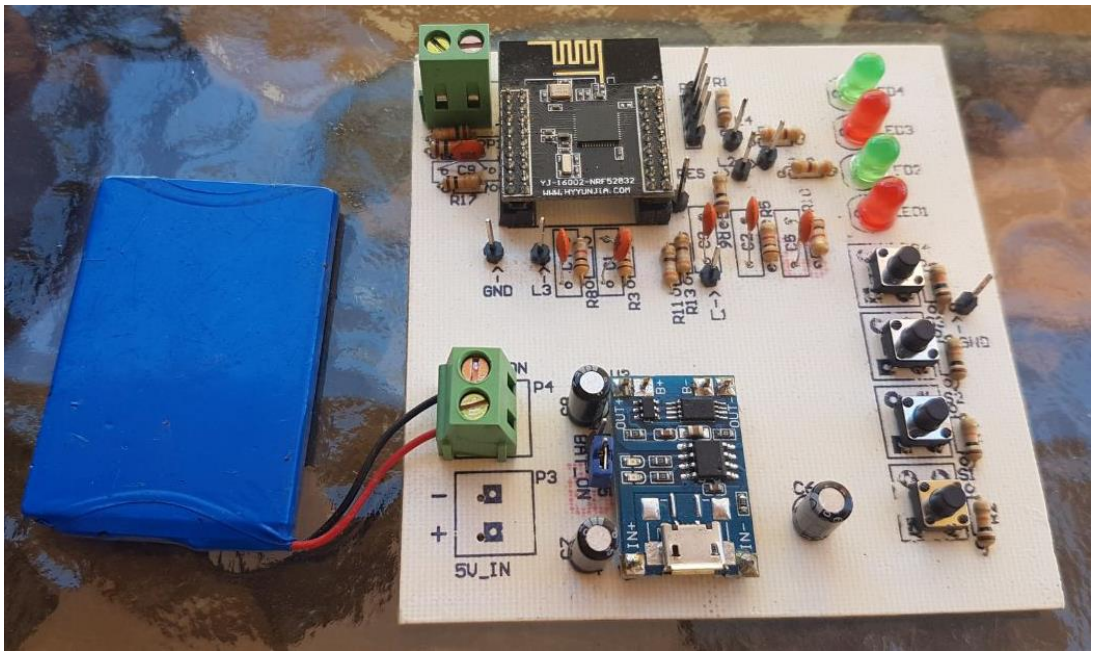
[39] O divisor resistivo é necessário devido à tensão da bateria chegar até 4.2V e a tensão de operação do nRF52832 ser de 3.3V.

Os LEDs foram conectados entre o Vdd e o pino do módulo para que a corrente seja drenada para o GND, não influenciando a medida de energia consumida pelo módulo quando o LED está ligado. Os botões possuem filtros RC, formando um debouncer, para reduzir o repique dos botões.

A PCB foi confeccionada através do processo de redução de cobre, utilizando percloreto de ferro. Para a sua confecção, após a finalização do layout, em software, ele foi impresso em uma impressora laser com papel fotográfico. Com a impressão em mãos é utilizado um processo de transferência térmica para que as trilhas impressas passem para o lado de cobre da placa. A placa, portanto, recebe um banho de percloreto de ferro até remover todo o cobre não coberto pela impressão. Para finalizar, os furos são feitos e uma camada de verniz é aplicada para que o cobre não venha a oxidar. Adicionalmente, um processo de transferência de impressão pode ser realizado na parte superior da placa para criar uma serigrafia com os desenhos dos componentes.

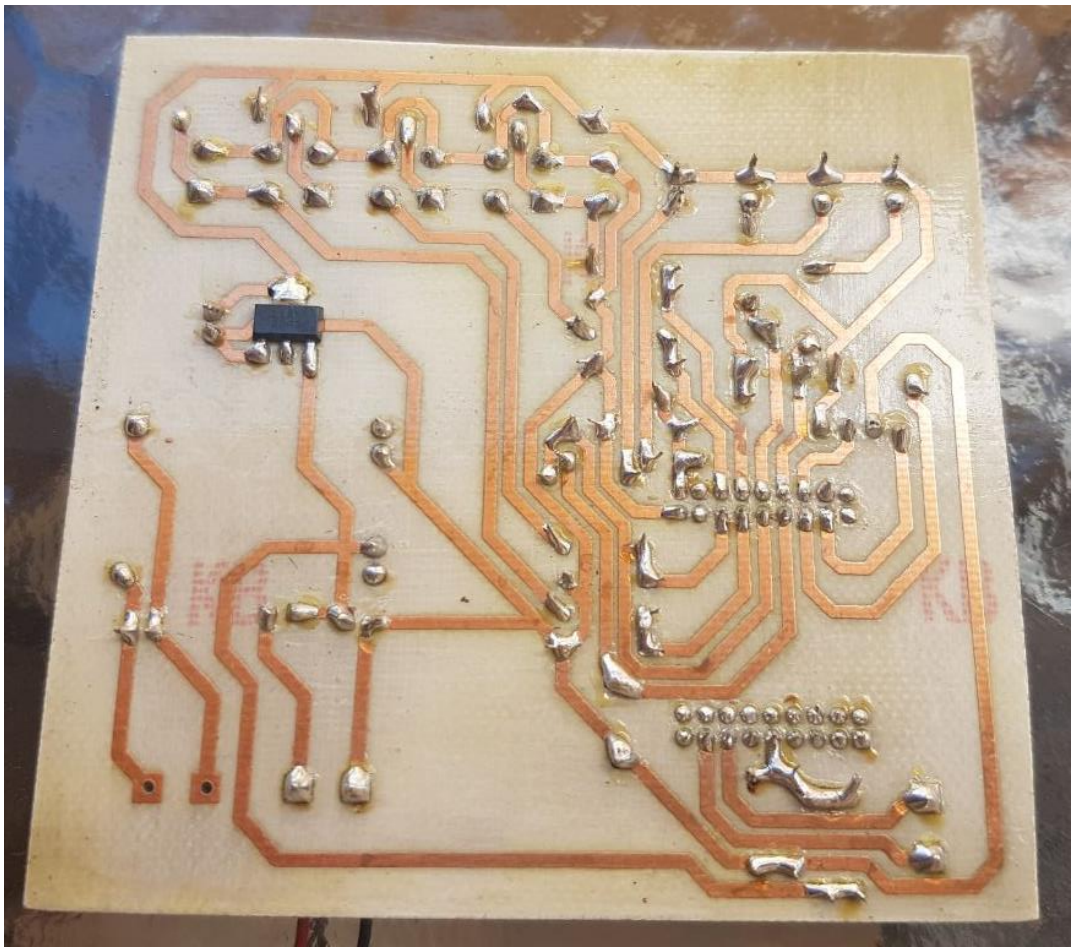
Após a finalização da PCB, todos os componentes dela foram devidamente soldados. A figura 21 ilustra a PCB montada com seus componentes. E a figura 22 mostra o lado inferior da PCB e o regulador de tensão 3,3V que é SMD.

Figura 21 PCB com os componentes



Fonte: Autores

Figura 22 Face inferior da PCB



Fonte: Autores

O material da PCB escolhido foi o composite[40], e a placa possui apenas uma face de cobre. Devido a grande maioria dos componentes serem PTH[41] não foi necessário utilizar nenhum jumper na placa, mesmo ela sendo de apenas uma camada. Os apêndices D e E mostram o esquemático e layout da placa, respectivamente, e o apêndice F a lista de componentes utilizados na montagem da PCB. Com a finalização da placa podemos dar início aos testes desenvolvidos.

[40] O composite é um compósito de fibra de carbono, mais resiste que o comum fenolite e bem menos duro que a fibra de vidro, que danifica muito a broca das fresadoras.
[41] Componentes PTH (Plated Through Hole) são componentes que ao contrário dos SMD (Surface Monted Device) precisam de furos nas placas para serem soldados no lado inferior ao que são posicionados.

4.2 TESTES REALIZADOS

Antes de começar os testes, um formato de mensagem a ser utilizado na publicação foi definido. Essa mensagem deveria enviar o tempo em segundos que o dispositivo estava ligado e também o nível de tensão da bateria. O tópico em que a mensagem publicava era do seguinte formato: tcc/nrf52/”os”, em que “os” deveria ser substituído por contiki ou zephyr, dependendo de qual estava sendo implementado. A figura 23 a seguir ilustra uma subscrição ao broker quando o Zephyr OS estava publicando.

Figura 23 Subscrição ao broker com publicação do Zephyr

Fonte: Autores

A leitura do nível da bateria foi utilizada para servir como uma unidade sensora, dando ao equipamento uma funcionalidade a mais do que apenas enviar pacotes para testes. Para realizar a leitura da tensão o periférico SAADC[42] do nRF52 precisou ser utilizado. Devido ao fato do Zephyr ainda não suportar conversões AD, as leituras foram implementadas realizando acesso direto aos registradores do SAADC.

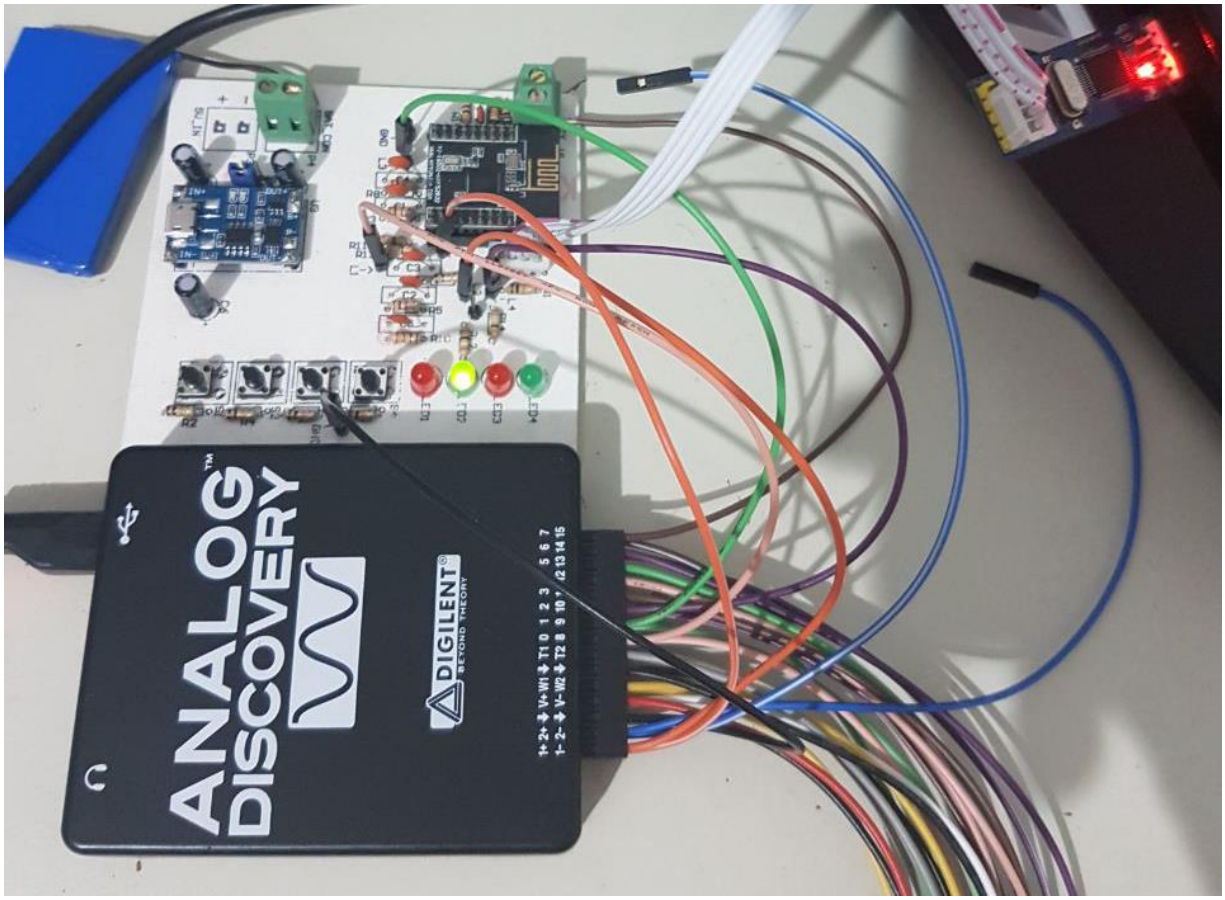
Alguns exemplos da Nordic foram utilizados como base, mas não foi possível utilizar a implementação deles, pois eles utilizavam interrupções para avisar quando a conversão havia sido finalizada. Para a utilização de interrupções com os RTOSs, drivers e APIs deveriam ser criadas. Portanto foi necessário analisar o funcionamento da conversão e adicionar uma espera (while) enquanto a conversão não havia terminado para realizar o envio. O apêndice G descreve as duas funções utilizadas para inicialização do SAADC e para realização das leituras.

[42] O SAADC (Successive approximation analog-to-digital converter) possui um recurso de realizar extras conversões analógico-digital e aumentar a resolução da leitura, por exemplo o nRF52 possui resolução máxima de 12 bits mas com as leituras extras (oversampling) ele alcança uma resolução de 14 bits.

4.2.1 Consumo de energia e tempo de publicação

Os primeiros testes realizados foram os relacionados ao consumo de energia. Foram efetuadas 3 medições diferentes para cada RTOS, uma em modo de anúncio, outra durante a conexão sem troca de dados e uma última durante a publicação em MQTT. Para medir a energia consumida e os tempos, foi utilizado um osciloscópio e analisador lógico USB da DIGILENT chamado de Analog Discovery, a figura 24 ilustra como feita a conexão entre o osciloscópio e o gravador ST-LINK/V2 com a PCB.

Figura 24 Conexão do Osciloscópio USB com a PCB

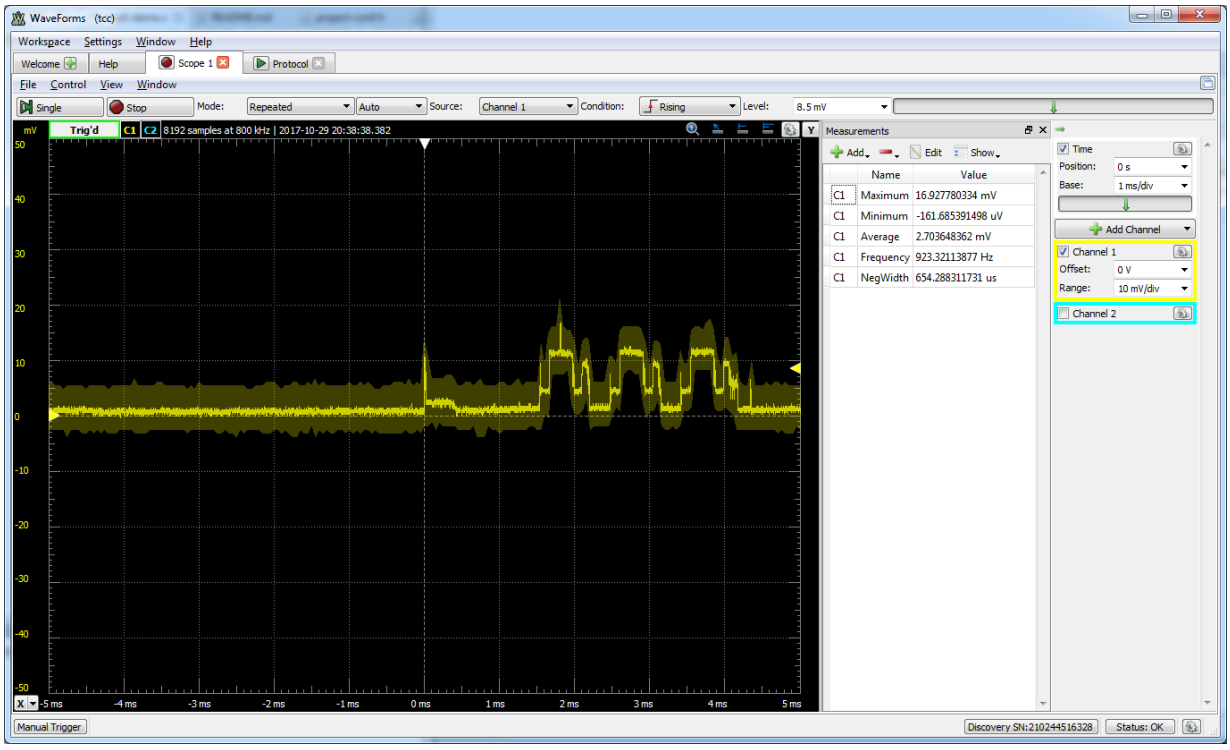


Fonte: Autores.

4.2.1.1 Consumo durante anúncio (advertising)

As figuras 25 e 26 a seguir mostram a medição da corrente de entrada[43] do módulo durante o anúncio. Por mais semelhantes que sejam, é possível perceber que cada implementação é diferente. Enquanto o Zephyr realiza a troca de canal e mantém o rádio ligado, o Contiki desliga o rádio entre as trocas, gerando um tempo total de anúncio maior. É possível observar também que o anúncio é realizado em três canais diferentes, conforme explicado anteriormente nas características de funcionamento do BLE. O intervalo de repetição dos anúncios também é diferente, enquanto o Contiki repete a cada 340ms, o Zephyr repete a cada 100ms. Analisando, portanto, a corrente média durante um ciclo completo para ambos, temos que o Contiki consome 1,38mA e o Zephyr 1,29mA.

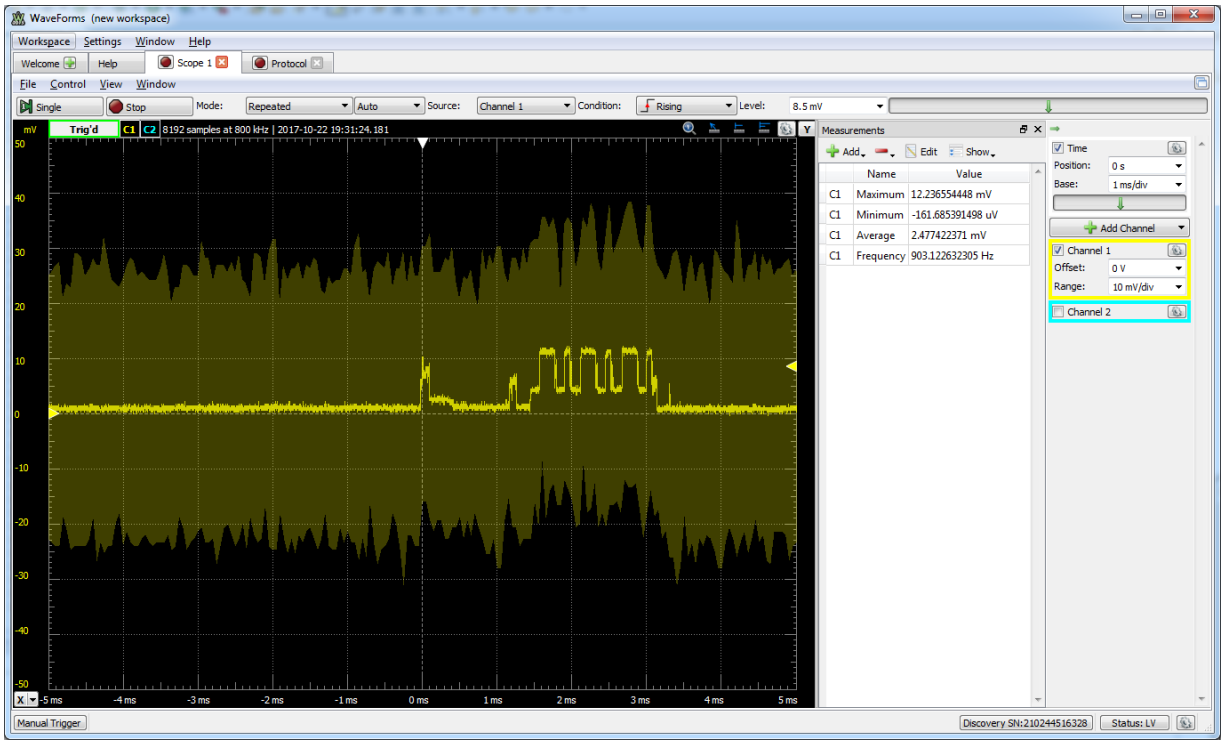
Figura 25 Consumo do Contiki OS em Advertising



Fonte: Autores

[43] O fato de aparecer tensão e não corrente nas imagens é devido ao fato do osciloscópio estar medindo a tensão sobre o resistor de 1Ω na entrada do módulo. Portanto o valor em tensão pode ser convertido para o mesmo valor em corrente.

Figura 26 Consumo do Zephyr OS em Advertising



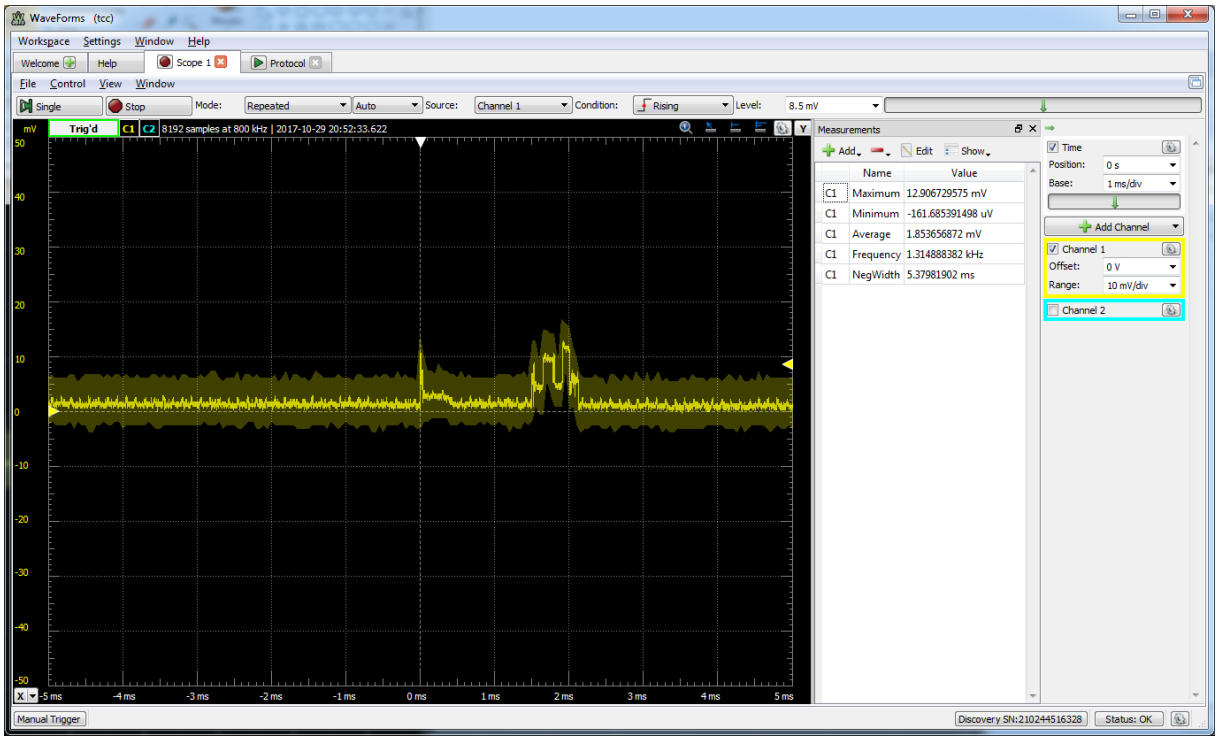
Fonte: Autores.

4.2.1.2 Consumo durante conexão sem transferência de dados

Com o módulo em anúncio, foi realizada uma conexão pelo Linux com ele. Essa conexão, mesmo que não haja transferência efetiva de dados, faz com que o periférico mantenha uma troca de informações continua. Essa troca de informações está descrita nas características do BLE como um evento de conexão. A cada intervalo regular o rádio é ligado e há uma troca de informações. As informações trocadas durante a conexão, quando não há transferência de pacotes em IPv6 são uma espécie de keep-alive da conexão. Pois no momento que o nRF52 é desligado o Linux já detecta a perda de conexão.

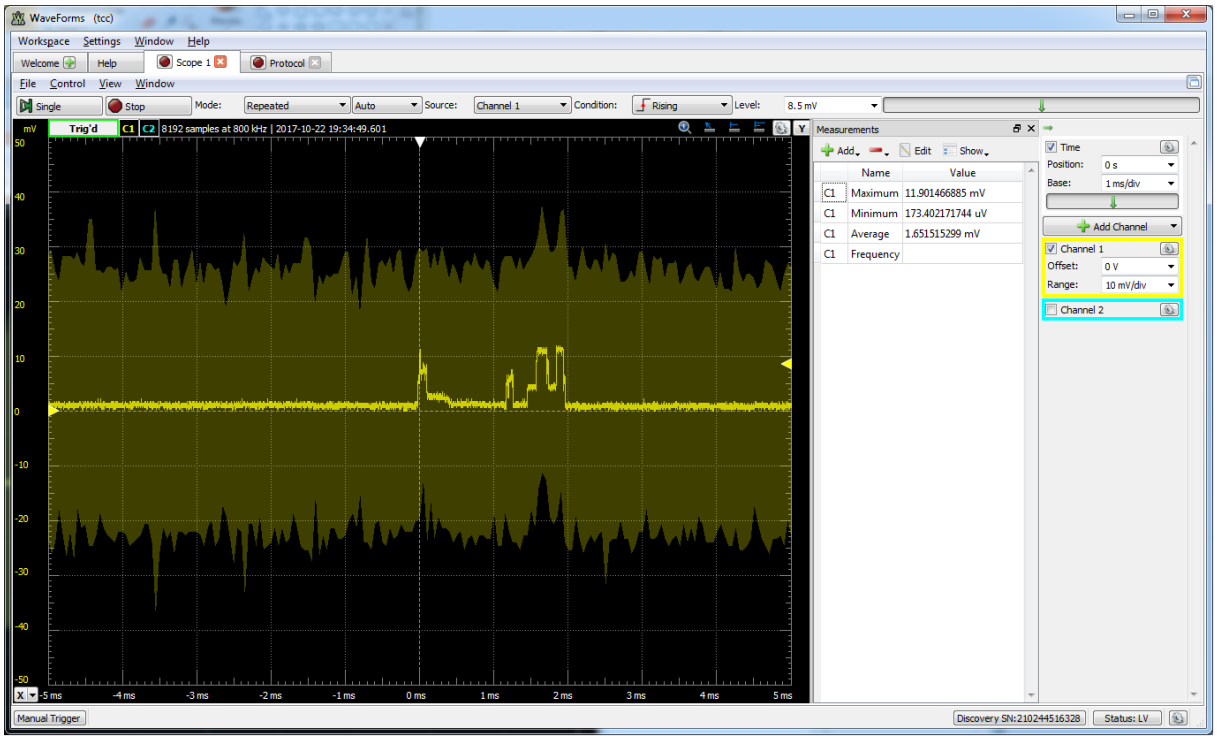
Observamos então pelas figuras 27 e 28 que o consumo de energia muda de “formato”, agora comunicando em apenas um canal. O intervalo de comunicação também muda, enquanto o Zephyr reduz de 100ms para 50ms, o Contiki reduz de 340ms para 70ms o intervalo de comunicação. Isso causa um leve aumento no consumo de energia. Medindo agora a corrente média durante todo o período o Contiki passa a ter 1,46mA e o Zephyr 1,36mA de consumo.

Figura 27 Consumo do Contiki OS em conexão



Fonte: Autores

Figura 28 Consumo do Zephyr OS em conexão



Fonte: Autores

4.2.1.3 Consumo durante publicação e compressão de cabeçalho

Após coletados os dados de consumo em modo de anúncio e em conexão sem troca de dados, tiveram início os testes relativos ao consumo de energia durante a publicação e ao mesmo tempo as medições de tempo total de envio e de compressão/descompressão de cabeçalho. Para análise desses tempos propostos, a biblioteca de GPIO foi utilizada em ambos RTOSs. Foi definido então que o LED 1 da PCB ficasse ligado durante o tempo de publicação, para ser analisado o tempo total de publicação, e os LEDs 3 e 4 utilizados para os tempos de descompressão e compressão de cabeçalho.

Para descobrir o local onde as compressões e descompressões são realizadas no código do Contiki OS e do Zephyr OS foi necessário uma análise da estrutura de cada sistema operacional. No caso do Contiki OS a descompressão do pacote IPv6 acontece na função input do arquivo sicslowpan.c localizado no diretório core/net/ipv6. Nessa função foi adicionada uma chamada para ligar o LED 3 enquanto a descompressão era realizada. A compressão é realizada no mesmo arquivo, na função output. Para a compressão, o LED 4 foi ligado.

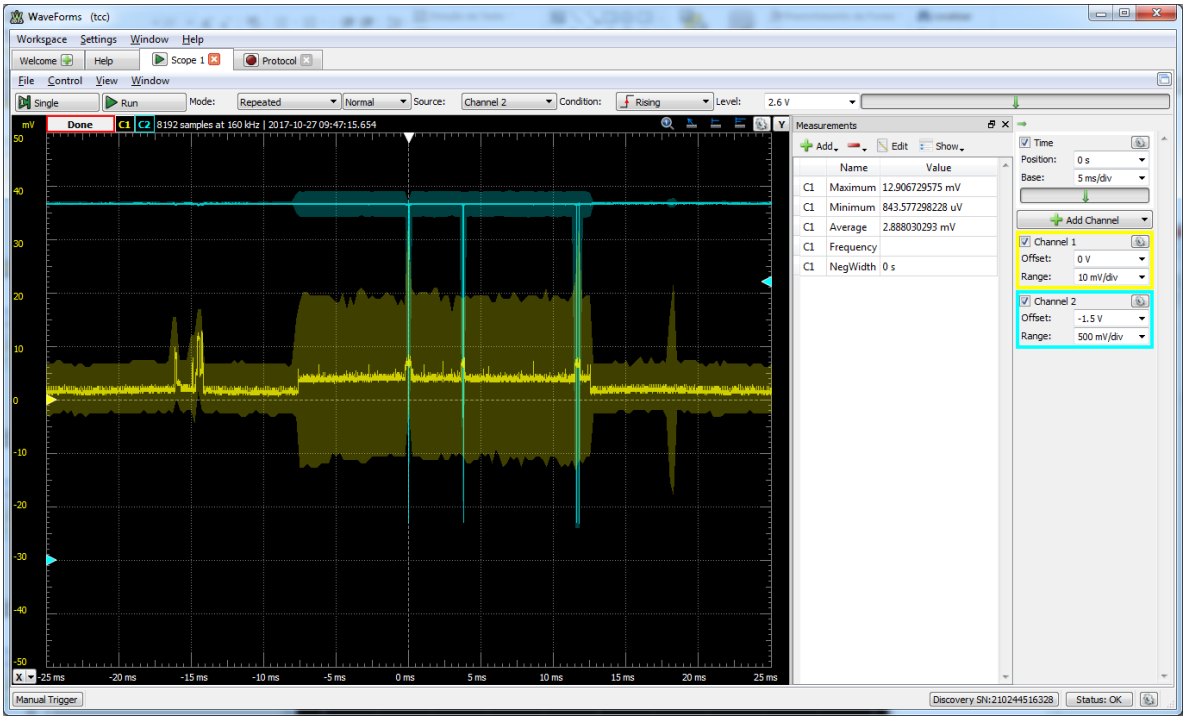
Para o Zephyr OS, as funções de compressão e descompressão se encontram no arquivo 6lo.c localizado no diretório subsys/net/ip, com o nome de net_6lo_compress e net_6lo_uncompress para compressão e descompressão respectivamente. Da mesma forma que o Contiki, foi adicionado no código para ligar os LEDs enquanto essas funções estão sendo processadas.

Em um primeiro teste, foi verificado o quanto a compressão do pacote IPv6 influência no consumo de energia. Para isso foram utilizados os dois canais do osciloscópio USB, o canal 1 com o consumo de energia e o canal 2 para o LED 4 que ligava durante as compressões. A figura 29 ilustra o resultado de uma publicação com o Zephyr OS mostrando o consumo de energia em relação ao tempo de compressão.

Observamos que para um pacote de publicação MQTT com o Zephyr, 3 compressões são realizadas. Entretanto, mesmo sendo possível notar que durante a compressão, que varia de 30 a 200 us, o consumo de energia do módulo é maior, esse aumento é quase insignificante quando comparado com o consumo de energia total da publicação ou o gasto de energia do keep-alive da conexão. O Contiki OS

apresenta a mesma relação de tempo de compressão versus consumo de corrente do módulo.

Figura 29 Consumo de energia versus copressão na publicação

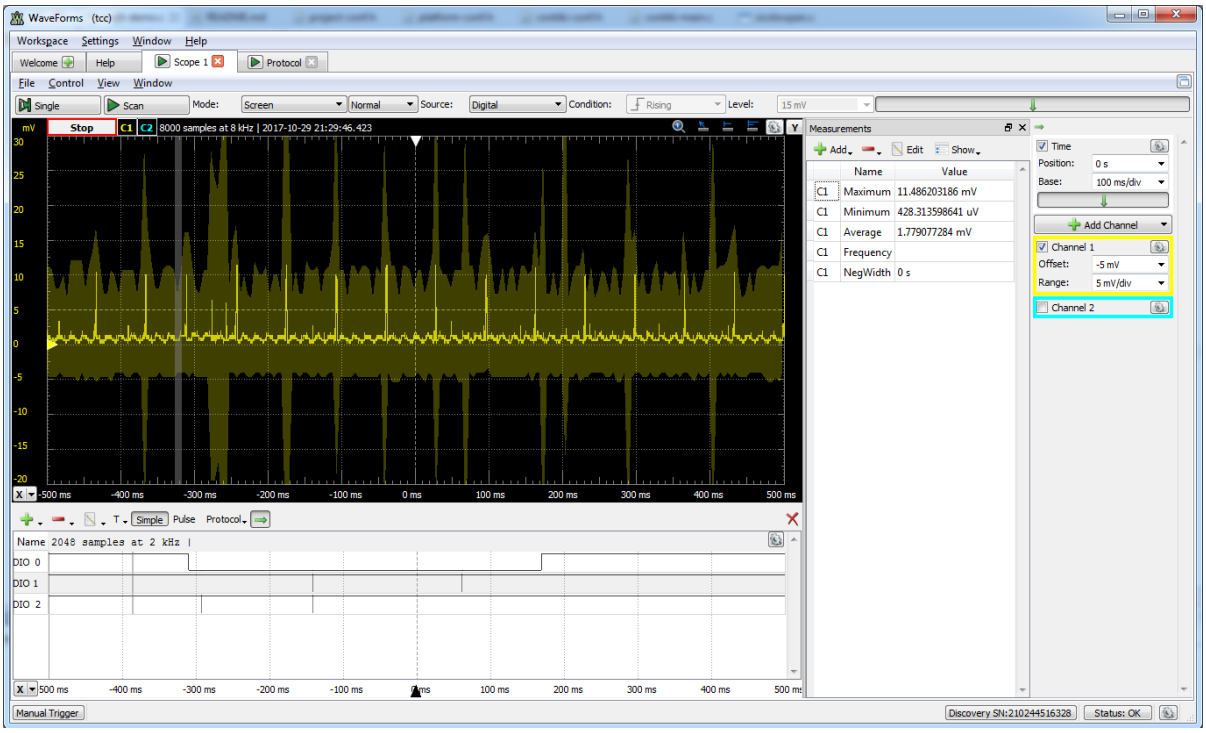


Fonte: Autores

4.2.1.4 Tempo total de publicação

Utilizando o osciloscópio USB, conectamos ao canal 1 o resistor para medir o consumo de corrente do módulo e às entradas digitais 0, 1 e 2 ligadas aos LEDs 1, 3 e 4, respectivamente, marcando o tempo total de publicação, tempo de descompressão e tempo de compressão, nesta ordem. O primeiro a ser testado foi o Contiki OS e a figura 30 demonstra o comportamento do módulo durante a transferência. O tempo total de publicação dura cerca de 500 ms (canal digital DIO 0), acontecem duas compressões (DIO 2) e duas descompressões (DIO 1). A figura 31 ilustra a captura dos pacotes dessa comunicação a partir do software Wireshark.

Figura 30 Tempo total de publicação do Contiki OS



Fonte: Autores

Figura 31 Troca de pacotes analizadas no Wireshark referente a figura 30

2	0.841003627	1993:7cc::…	2017:7cc::…	TCP	108 [TCP segment of a reassembled PDU]
3	0.841188949	2017:7cc::…	1993:7cc::…	TCP	76 1883 → 1026 [ACK] Seq=1 Ack=33 Win=28800 Len=0
4	1.042095030	1993:7cc::…	2017:7cc::…	MQTT	90 Publish Message
5	1.042255334	2017:7cc::…	1993:7cc::…	TCP	76 1883 → 1026 [ACK] Seq=1 Ack=47 Win=28800 Len=0

Fonte: Autores

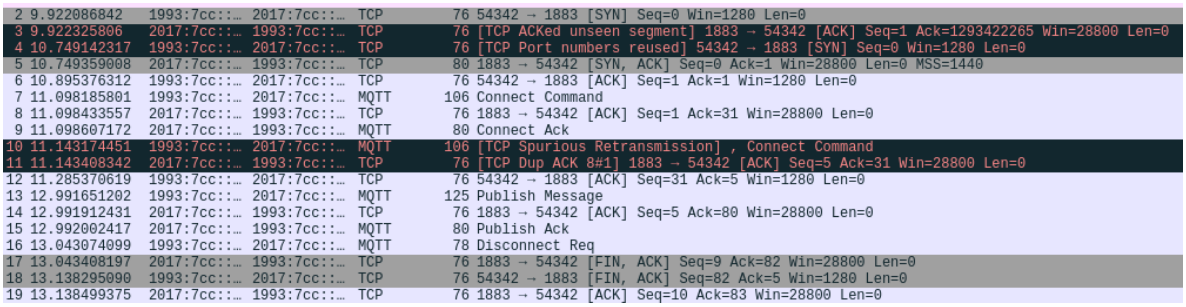
Podemos perceber que no começo há uma compressão para o primeiro pacote TCP enviado, confirmado pela figura 31 mostrando que esse pacote foi enviado do módulo para o broker[44]. Logo após, o broker responde com um ACK que quando chega ao módulo é descomprimido. Após a confirmação do ACK o módulo já comprime o pacote de publicação MQTT. Para finalizar, quando o broker recebe a publicação, ele retorna um ACK que novamente é descomprimido ao chegar no

[44] É possivel perceber que o primeiro pacote da figura 31 foi enviado do módulo para o broker porque o endereço IP de envio pertence a rede 1993:7cc:: definido para os dispositivos BLE.

módulo, ilustrado na figura 30 na última descompressão. Poucos milissegundos depois o módulo termina a publicação.

Após essa análise com o Contiki, começamos a do Zephyr OS. E as análises iniciais não foram nada satisfatórias, o tempo total de publicação do Zephyr OS era de aproximadamente 2,45 segundos, tempo absurdamente elevado. Com análise do Wireshark, demonstrado pela figura 32 é possível perceber que a implementação dos dois sistemas operacionais é bem diferente. Enquanto o Contiki OS após a conexão por BLE, já conectava ao broker e mantinha a conexão aberta, o Zephyr toda vez que publicava abria uma nova conexão TCP e MQTT, publicava MQTT e fechava a conexão.

Figura 32 Troca de pacotes do Zephyr OS

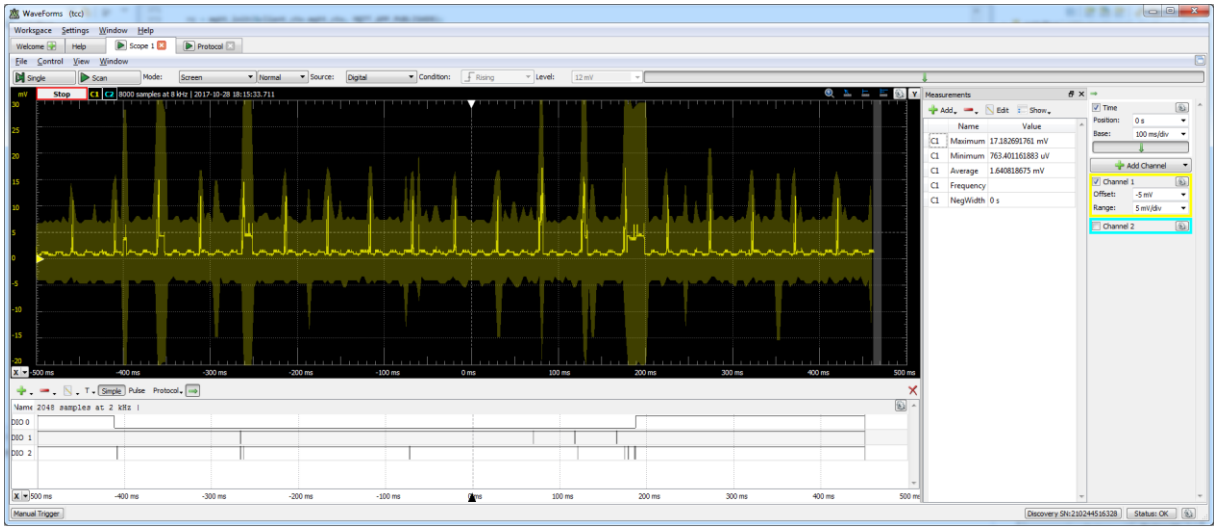


Fonte: Autores

Mesmo que o Zephyr faça a abertura e fechamento de conexão em toda publicação, esse não é o principal problema. Se observado o tempo de cada pacote na figura 32, é possível perceber que entre os pacotes 12 e 13 há uma demora de 1,7 segundos. Após uma extensa análise do código, foi descoberto que no exemplo de publicação MQTT, disponível do Zephyr, após a conexão em MQTT entrava em uma espera de 1,5 segundos. Essa espera era utilizado para garantir que a conexão havia sido estabelecida, sem analisar o ACK de retorno. Foi modificado então o código e adicionado um call-back na interrupção que trava as respostas da comunicação. Assim quando o ACK retornava ao módulo ele automaticamente disparava a função de publicação. Dessa forma a função apenas aguardava o ACK para publicar.

Com a modificação, o tempo total de publicação foi reduzido para 580 ms. Podemos observar na figura 33 como o módulo se comporta durante toda a conexão MQTT. Desde a abertura da conexão TCP, publicação e fechamento da conexão. A figura 34 mostra os pacotes capturados no software Wireshark durante essa comunicação. É possível perceber agora que o atraso entre os pacotes de ACK da conexão e de publicação (pacotes número 10 e 11) foi reduzido de 1,7 segundos para cerca de 250ms, latência normal de uma comunicação por BLE.

Figura 33 Tempo total de publicação do Zephyr OS após a primeira melhoria do código



Fonte: Autores

Figura 34 Troca de pacotes analizados no Wireshark referente à figura 33

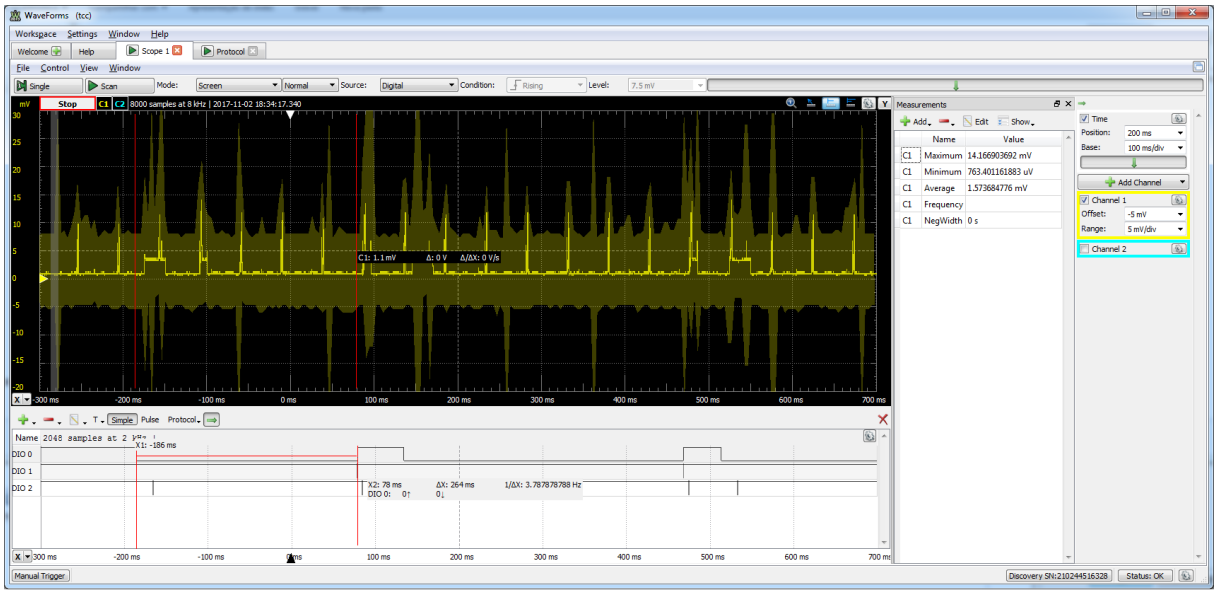
2	5.223640590	1993:7cc::…	2017:7cc::…	TCP	76 38510 → 1883 [SYN] Seq=0 Win=1280 Len=0
3	5.223843225	2017:7cc::…	1993:7cc::…	TCP	80 1883 → 38510 [SYN, ACK] Seq=0 Ack=1 Win=28800 Len=0 MSS=1440
4	5.370007357	1993:7cc::…	2017:7cc::…	TCP	76 38510 → 1883 [ACK] Seq=1 Ack=1 Win=1280 Len=0
5	5.565160815	1993:7cc::…	2017:7cc::…	MQTT	106 Connect Command
6	5.565418213	2017:7cc::…	1993:7cc::…	TCP	76 1883 → 38510 [ACK] Seq=1 Ack=31 Win=28800 Len=0
7	5.568755841	2017:7cc::…	1993:7cc::…	MQTT	80 Connect Ack
8	5.617140846	1993:7cc::…	2017:7cc::…	MQTT	106 [TCP Spurious Retransmission] , Connect Command
9	5.617668947	2017:7cc::…	1993:7cc::…	TCP	76 [TCP Dup ACK 6#1] 1883 → 38510 [ACK] Seq=5 Ack=31 Win=28800 Len=0
10	5.760072798	1993:7cc::…	2017:7cc::…	TCP	76 38510 → 1883 [ACK] Seq=31 Ack=5 Win=1280 Len=0
11	6.003818822	1993:7cc::…	2017:7cc::…	MQTT	123 Publish Message
12	6.004033847	2017:7cc::…	1993:7cc::…	TCP	76 1883 → 38510 [ACK] Seq=5 Ack=78 Win=28800 Len=0
13	6.010074968	2017:7cc::…	1993:7cc::…	MQTT	80 Publish Ack
14	6.055108668	1993:7cc::…	2017:7cc::…	MQTT	78 Disconnect Req
15	6.058672826	2017:7cc::…	1993:7cc::…	TCP	76 1883 → 38510 [FIN, ACK] Seq=9 Ack=80 Win=28800 Len=0
16	6.103607480	1993:7cc::…	2017:7cc::…	TCP	76 38510 → 1883 [FIN, ACK] Seq=80 Ack=5 Win=1280 Len=0
17	6.103818911	2017:7cc::…	1993:7cc::…	TCP	76 1883 → 38510 [ACK] Seq=10 Ack=81 Win=28800 Len=0

Fonte: Autores

Mesmo com essa melhoria, quando era tentado enviar uma mensagem atrás da outra em uma frequência de envio mais alta, a comunicação começava a perder pacotes de conexão TCP exponencialmente e em poucos segundos a conexão pelo 6LoWPAN era perdida. Foi necessário então realizar outra modificação com o objetivo de deixar a troca de pacotes do mesmo formato que o Contiki OS faz. Foi alterado, portanto, a função de publicação de modo que a conexão MQTT se mantém aberta e o módulo apenas execute publicações sucessivas. Isso resultou numa melhora excepcional da comunicação reduzindo de 580ms para cerca de 264ms o tempo total de transmissão.

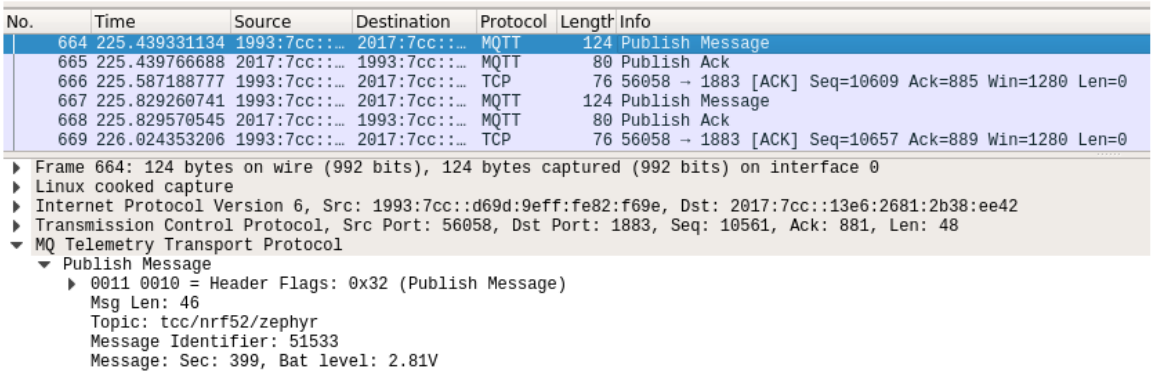
Foi configurado, portanto, o módulo para realizar as publicações MQTT com o Zephyr OS uma logo após a outra. Na figura 35 a seguir, podemos ver que o tempo total de transmissão é de 264ms, e que logo após acabar um envio ele já começa o outro. Na figura 36 é possível ver a troca de pacotes que agora entre uma publicação e outra possui apenas o pacote de publicação e os ACKs, mantendo a conexão aberta entre as duas portas. Na figura 35 também podemos ver a compressão para o envio do pacote e a descompressão do ACK recebida do broker e a compressão do último ACK enviado ao broker finalizando a comunicação.

Figura 35 Tempo total de publicação do Zephyr OS após a segunda melhoria do código



Fonte: Autores

Figura 36 Troca de pacotes[45] analizados no Wireshark referente à figura 35



Fonte: Autores

4.2.2 Taxa de transferência

Após conclusão dos testes relacionados ao consumo de energia, tempo total de transmissão e compressões/descompressões de pacotes IPv6, e conclusão das melhorias do código que não haviam sido previamente previstas, começaram os testes relacionados à máxima transferência de dados em uma publicação MQTT variando a distância física entre os nós e o tamanho da mensagem publicada. Para todas as publicações foi utilizado o QoS 0 (Quality of Service 0), o mais simples, pois não se preocupa em garantir a entrega da mensagem na camada de aplicação. O MQTT repassa a mensagem para a camada de transporte e ela se preocupa em enviar a publicação, não sendo necessário devolver ao MQTT um ACK da mensagem publicada.

Inicialmente os testes realizados foram com o tamanho de mensagem padrão, especificado anteriormente e realizando uma variação na distância. A variação na distância ocorreu da seguinte forma: foram analisados as transmissões em 4 posições diferentes do módulo longe do computador com sua posição fixa. A primeira posição foi a 30 cm do computador, sem nenhum obstáculo, a próxima foi a 5 metros sem obstáculos, outra posição foi a 3 metros com uma parede de

[45] A tensão da bateria aparece como 2,81V. Isso se deve ao fato de que quando a bateria está desconectada, a PCB pode ser alimentada pelo gravador ST-LINK/V2, direto no barramento de 3,3V. Por causa das características contrutivas do regulador de 3,3V da PCB, se ele não estiver alimentado e for colocado 3,3V nos pinos de saída dele, ele acaba “jogando” para a entrada 2,81V, queda de +/- 0,5V devido a alguma junção interna.

alvenaria[46] entre os dispositivos, e a última posição foi a 8 metros também com uma parede de alvenaria de obstáculo.

Não foi testado em maiores distâncias devido ao fato de que em ambos RTOSs quando ultrapassados 9 metros entre o módulo e o computador a conexão 6LoWPAN simplesmente era perdida. É possível descobrir o módulo em anúncio a 10 metros de distância, mas uma conexão não é estabelecida.

4.2.2.1 Taxa de transmissão para publicações pequenas

Utilizando a mensagem padrão definida anteriormente de 46 Bytes, foram modificados os dois códigos para assim que a comunicação estivesse apta a receber uma nova mensagem ela já começasse a enviar os dados novamente. Verificando-se assim a máxima capacidade de transferência tanto para o Zephyr OS quanto para o Contiki OS. A tabela 5 a seguir mostra os resultados obtidos para o Contiki OS. E a Tabela 6 para o Zephyr OS.

Percebemos que o Zephyr OS possui um desempenho muito superior, conseguindo enviar mais que o dobro de dados no mesmo período. É possível notar também que a distância e obstáculos praticamente não altera a taxa de transmissão até o ponto que a conexão simplesmente é desfeita, quando os 10 metros de distância é alcançado.

Tabela 5 Taxa de publicações com o Contiki OS

Distância	Duração do teste	Publicações realizadas	Taxa
30cm	23 segundos	26	1,13 pub/s
5m	22 segundos	26	1,18 pub/s
3m com obstáculo	21 segundos	25	1,19 pub/s
8m com obstáculo	20 segundos	22	1,10 pub/s

Fonte: Autores

[46] A parede de alvenaria é um obstáculo muito comum para as redes WI-FI, e se no futuro o BLE for aderido em vários equipamentos para acesso a rede de internet, ele deveria ter a capacidade de possuir pelo menos a mesma distância de abrangência que as redes WI-FI atuais.

Tabela 6 Taxa de publicações com o Zephyr OS

Distância	Duração do teste	Publicações realizadas	Taxa
30cm	11 segundos	30	2,73 pub/s
5m	11 segundos	32	2,91 pub/s
3m com obstáculo	11 segundos	30	2,73 pub/s
8m com obstáculo	11 segundos	26	2,36 pub/s

Fonte: Autores

4.2.2.2 Taxa de transmissão para publicações grandes

Realizado o teste para publicações pequenas, o tamanho da mensagem foi aumentado, antes enviando 46 bytes de informação útil na publicação, agora passamos a ter 1020 bytes. Para aumentar o tamanho da mensagem, parte de uma musica do Iron Maiden foi adicionada ao payload da publicação. O apêndice H mostra como a publicação é feita no Contiki OS. No Zephyr é basicamente a mesma coisa, só mudam as chamadas de funções. O apêndice I mostra um subscritor ao broker recebendo esse tipo de mensagem.

Inicialmente o objetivo era enviar toda a letra da música, mas o Zephyr OS possui uma limitação em que cada publicação MQTT pode ter no máximo 1024 bytes. O tamanho máximo da mensagem de publicação do Zephyr OS deve ser configurado no arquivo de configurações prj.conf. Quando um valor maior que 1024 é inserido, um erro ocorre durante a compilação informando que o tamanho é muito grande, mesmo sendo de 4kB o tamanho máximo da mensagem de publicação MQTT definida pela IBM. Já o Contiki OS nunca apresentou nenhuma limitação durante os testes, provavelmente devido ao fato dele sempre quebrar os pacotes maiores que 32 Bytes.

Após a configuração de ambos os RTOSs, os testes foram realizados da mesma maneira que para as mensagens pequenas. A tabela 7 e 8 a seguir mostram os resultados obtidos para o Contiki OS e Zephyr OS respectivamente. Para a unidade da taxa dessas tabelas seguintes, foi decido por utilizar segundo por publicação, o inverso das tabelas anteriores devido ao fato de agora as publicações demorarem mais do que um segundo.

Tabela 7 Taxa de publicações de mensagem grande com o Contiki OS

Distância	Duração do teste	Publicações realizadas	Período
30cm	215 segundos	28	7,68 s/pub
5m	174 segundos	21	8,28 s/pub
3m com obstáculo	181 segundos	19	9,52 s/pub
8m com obstáculo	199 segundos	23	8,65 s/pub

Fonte: Autores

Tabela 8 Taxa de publicações de mensagem grande com o Zephyr OS

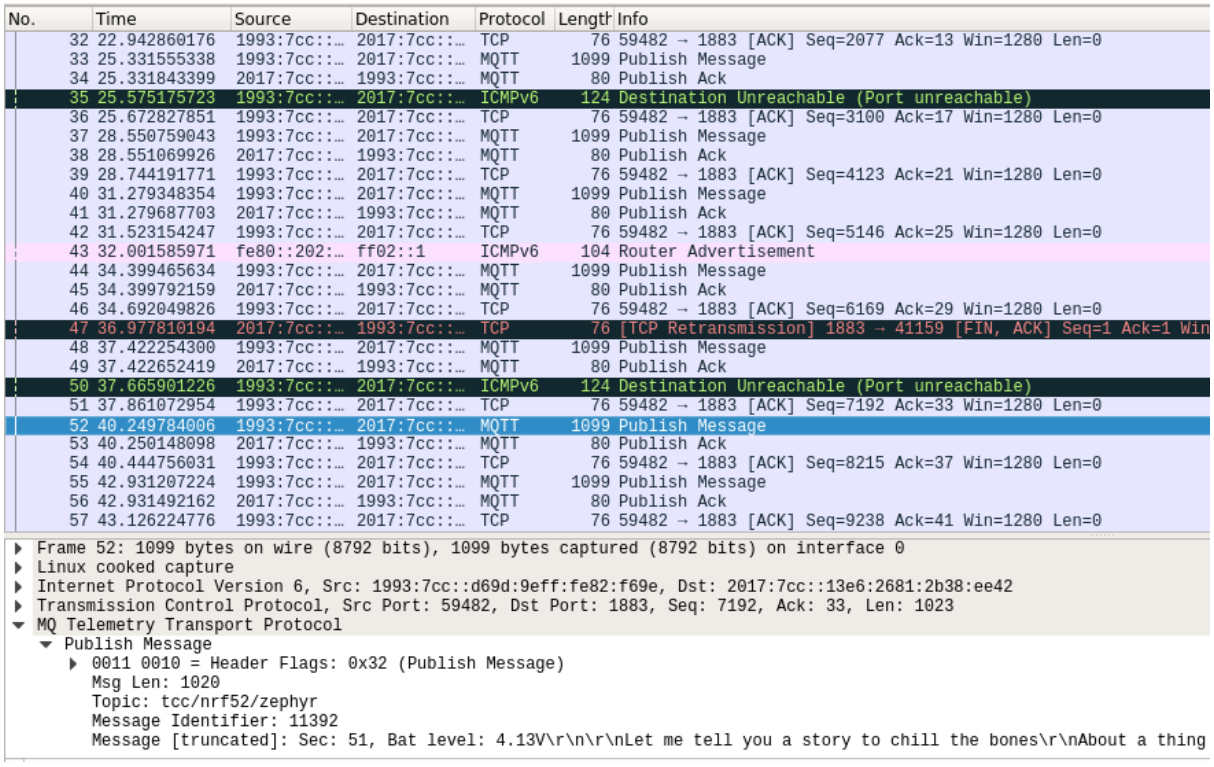
Distância	Duração do teste	Publicações realizadas	Período
30cm	75 segundos	30	2,50 s/pub
5m	82 segundos	34	2,41 s/pub
3m com obstáculo	97 segundos	37	2,62 s/pub
8m com obstáculo	112 segundos	36	3,11 s/pub

Fonte: Autores

Percebemos novamente que o Zephyr OS possui um desempenho entre três e quatro vezes maior que o Contiki OS para grandes mensagens publicadas. Isso ocorre devido ao modo com que sua pilha IPv6 foi implementada. Diferentemente do Contiki OS, o Zephyr OS envia um pacote grande MQTT de 1099 bytes contendo os 1020 bytes de dados útil. A camada de enlace da pilha do BLE que se preocupa em fragmentar esse grande pacote para ser enviado pelo BLE. Na figura 37 a seguir é possível verificar que o pacote de publicação possui os 1099 bytes.

Na figura 37, é possível observar também que o Zephyr OS realiza o envio de apenas um pacote grande, depois o broker responde um ACK e o módulo confirma o ACK. A figura 38 a seguir mostra o consumo de energia durante a publicação que dura 2,3 segundos. Durante a publicação, apenas um pacote é comprimido bem no início do envio (o pacote de publicação MQTT), e no final do envio é possível ver a descompressão do ACK e a compressão da confirmação.

Figura 37 Pacotes enviados do Zephyr com mensagem de publicação grande

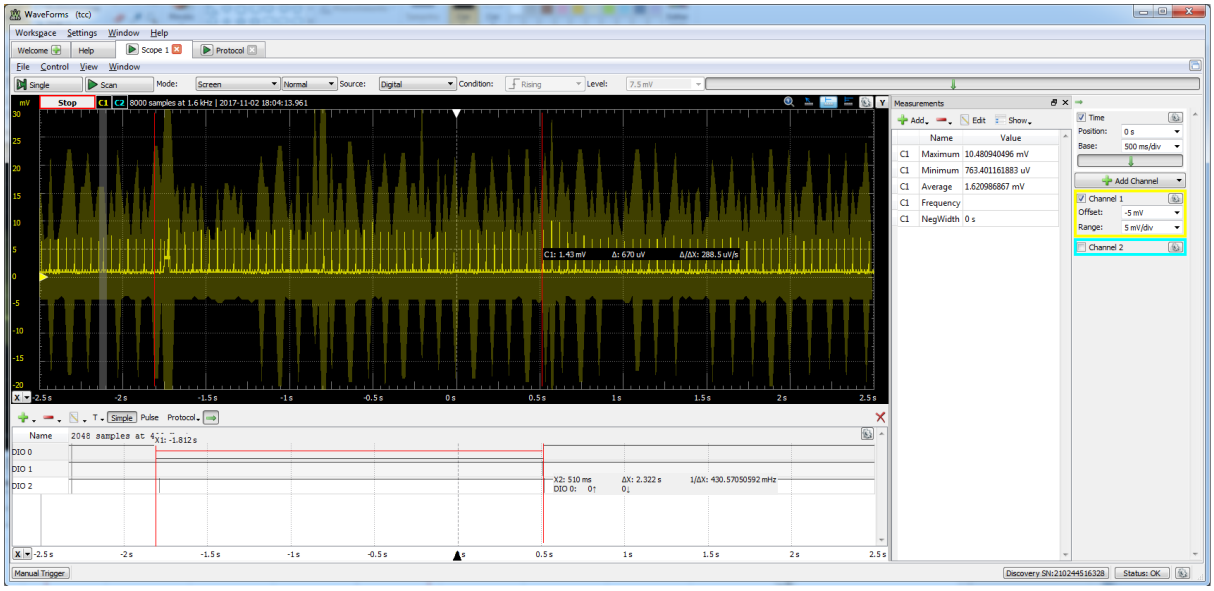


No.	Time	Source	Destination	Protocol	Length	Info
32	22.942860176	1993:7cc::…	2017:7cc::…	TCP	76	59482 → 1883 [ACK] Seq=2077 Ack=13 Win=1280 Len=0
33	25.331555338	1993:7cc::…	2017:7cc::…	MQTT	1099	Publish Message
34	25.331843399	2017:7cc::…	1993:7cc::…	MQTT	80	Publish Ack
35	25.575175723	1993:7cc::…	2017:7cc::…	ICMPv6	124	Destination Unreachable (Port unreachable)
36	25.672827851	1993:7cc::…	2017:7cc::…	TCP	76	59482 → 1883 [ACK] Seq=3100 Ack=17 Win=1280 Len=0
37	28.550759043	1993:7cc::…	2017:7cc::…	MQTT	1099	Publish Message
38	28.551069926	2017:7cc::…	1993:7cc::…	MQTT	80	Publish Ack
39	28.744191771	1993:7cc::…	2017:7cc::…	TCP	76	59482 → 1883 [ACK] Seq=4123 Ack=21 Win=1280 Len=0
40	31.279348354	1993:7cc::…	2017:7cc::…	MQTT	1099	Publish Message
41	31.279687703	2017:7cc::…	1993:7cc::…	MQTT	80	Publish Ack
42	31.523154247	1993:7cc::…	2017:7cc::…	TCP	76	59482 → 1883 [ACK] Seq=5146 Ack=25 Win=1280 Len=0
43	32.001585971	fe80::202:…	ff02::1	ICMPv6	104	Router Advertisement
44	34.399465634	1993:7cc::…	2017:7cc::…	MQTT	1099	Publish Message
45	34.399792159	2017:7cc::…	1993:7cc::…	MQTT	80	Publish Ack
46	34.692049826	1993:7cc::…	2017:7cc::…	TCP	76	59482 → 1883 [ACK] Seq=6169 Ack=29 Win=1280 Len=0
47	36.977810194	2017:7cc::…	1993:7cc::…	TCP	76	[TCP Retransmission] 1883 → 41159 [FIN, ACK] Seq=1 Ack=1 Win
48	37.422254300	1993:7cc::…	2017:7cc::…	MQTT	1099	Publish Message
49	37.422652419	2017:7cc::…	1993:7cc::…	MQTT	80	Publish Ack
50	37.665901226	1993:7cc::…	2017:7cc::…	ICMPv6	124	Destination Unreachable (Port unreachable)
51	37.861072954	1993:7cc::…	2017:7cc::…	TCP	76	59482 → 1883 [ACK] Seq=7192 Ack=33 Win=1280 Len=0
52	40.249784006	1993:7cc::…	2017:7cc::…	MQTT	1099	Publish Message
53	40.250148098	2017:7cc::…	1993:7cc::…	MQTT	80	Publish Ack
54	40.444756031	1993:7cc::…	2017:7cc::…	TCP	76	59482 → 1883 [ACK] Seq=8215 Ack=37 Win=1280 Len=0
55	42.931207224	1993:7cc::…	2017:7cc::…	MQTT	1099	Publish Message
56	42.931492162	2017:7cc::…	1993:7cc::…	MQTT	80	Publish Ack
57	43.126224776	1993:7cc::…	2017:7cc::…	TCP	76	59482 → 1883 [ACK] Seq=9238 Ack=41 Win=1280 Len=0

```
▸ Frame 52: 1099 bytes on wire (8792 bits), 1099 bytes captured (8792 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 6, Src: 1993:7cc::d69d:9eff:fe82:f69e, Dst: 2017:7cc::13e6:2681:2b38:ee42
▸ Transmission Control Protocol, Src Port: 59482, Dst Port: 1883, Seq: 7192, Ack: 33, Len: 1023
▾ MQ Telemetry Transport Protocol
  ▾ Publish Message
    ▸ 0011 0010 = Header Flags: 0x32 (Publish Message)
      Msg Len: 1020
      Topic: tcc/nrf52/zephyr
      Message Identifier: 11392
      Message [truncated]: Sec: 51, Bat level: 4.13V\r\n\r\nLet me tell you a story to chill the bones\r\nAbout a thing
```

Fonte: Autores

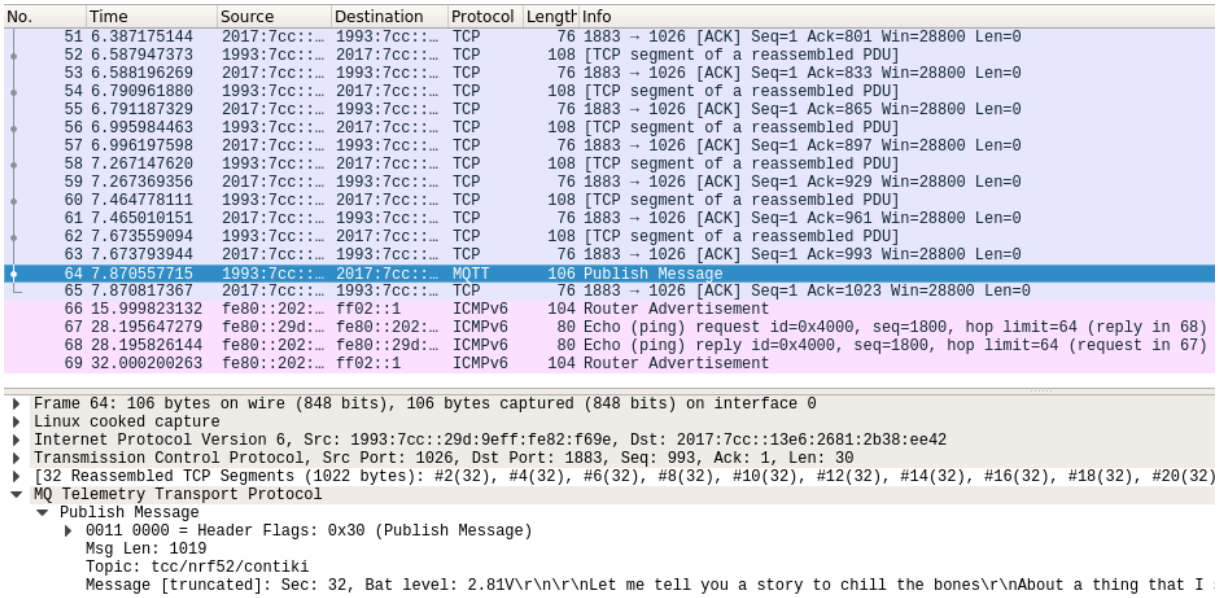
Figura 38 Análise da publicação de mensagens grandes com o Zephyr OS



Fonte: Autores

Já o Contiki OS fragmenta a mensagem a ser enviada na camada de transporte, criando vários pacotes TCPs. Como é possível observar na figura 39 abaixo, o pacote de publicação possui apenas 106 bytes, e ele informa que a mensagem de 1022 bytes foi reagrupada a partir de 32 mensagens que possuíam 32 bytes de informação cada uma. Essa implementação gera um overhead altíssimo, pois para cada 32 bytes de mensagem útil enviado o pacote possui 106 bytes com as informações dos protocolos, enquanto o Zephyr OS envia 1099 bytes com 1022 bytes úteis. Portanto, no total são muito menos bytes a serem transferidos, por isso a taxa de transferência é muito mais alta.

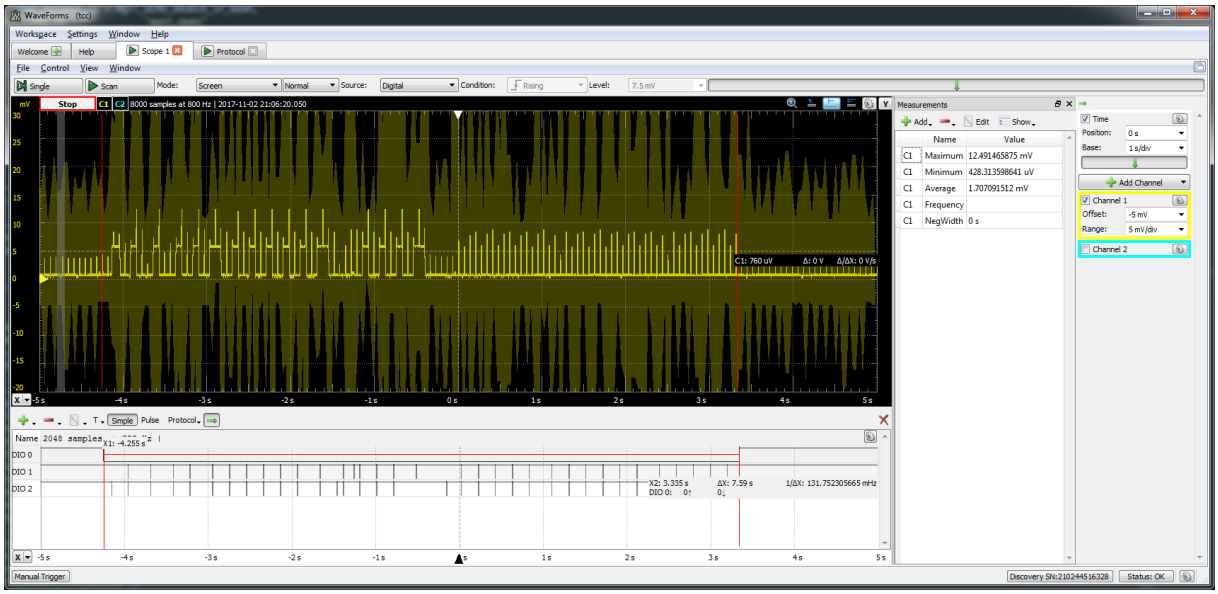
Figura 39 Pacotes enviados do Contiki com mensagem de publicação grande



Fonte: Autores

É possível observar, para o Contiki OS, que o consumo de energia e compressões/descompressões na figura 40 a seguir ocorre exatamente como na análise de dados com o software Wireshark, demonstrado na figura 39 anterior. Observando a imagem vemos que durante a publicação que é de cerca de 7,6 segundos, acontecem várias compressões e descompressões, uma para cada pacote TCP que é recebido e enviado durante uma publicação do Contiki OS.

Figura 40 Análise da publicação de mensagens grandes com o Contiki OS



Fonte: Autores

4.2.2.3 Intensidade do sinal (RSSI) nas posições avaliadas

Após a realização dos testes, foi analisado o nível de intensidade do sinal (RSSI - Received Signal Strength Indication) em dBm nas posições escolhidas. Para medir o RSSI foi utilizado Contiki OS, visto que a camada de rede dele calcula a intensidade do sinal de cada pacote que ele recebe durante uma comunicação, ficando fácil para as camadas superiores solicitarem essa informação.

Foi configurado, portanto, para o Contiki OS enviar uma publicação por segundo informando o RSSI da ultima transmissão como mensagem da publicação. O teste foi executa por 15 segundos em cada um dos locais escolhidos, e a tabela 9 a seguir mostra o resultado do teste, onde é possível observar que com o aumento da distância e de objetos no meio da comunicação, a intensidade de sinal diminui.

Observamos também que mesmo para uma distância pequena, a intensidade do sinal é mais baixo que o normal. Diversos equipamentos com BLE especificam que o RSSI varia de -24 a -100 dBm com a distância, entretanto com as nossas configurações o melhor valor alcançado nunca foi maior que -56 dBm.

Tabela 9 RSSI do sinal nas diferentes posições

Distância	Valor médio	Valor máximo	Valor mínimo
30cm	-60,9 dBm	-56 dBm	-64 dBm
5m	-76,6 dBm	-70 dBm	-82 dBm
3m com obstáculo	-82,9 dBm	-80 dBm	-89 dBm
8m com obstáculo	-83,2 dBm	-78 dBm	-90 dBm

Fonte: Autores

4.3 ANÁLISE DOS RESULTADOS

A partir dos testes descritos na seção 4.2 deste trabalho, os seguintes dados foram extraídos. A tabela 10 a seguir mostra o consumo de energia médio quando o módulo está em modo de anúncio e também depois de realizar a conexão. Ao lado da potência média dos dados do Contiki OS é informado quanto maior ele é quando comparado com o Zephyr OS na mesma categoria. O intervalo de transmissão para o modo de anúncio é o tempo em que o módulo realiza uma transmissão. Esse tempo poderia ser bem maior, causando um menor gasto de energia, mas aumentando o tempo em que a primeira conexão é realizada. O intervalo de transmissão durante a conexão é o tempo em que um evento de conexão acontece e informações são trocadas entre o módulo e o computador. Aumentar esse tempo reduziria o consumo de energia, mas também reduziria a capacidade de transmissão e aumentaria a latência da comunicação.

Tabela 10 Consumo de energia durante anúncio e conexão

	Int. de transmissão	Corrente média	Potência média	
Contiki – Anúncio	340ms	1,38 mA	4,55 mW	+6,8%
Zephyr – Anúncio	100ms	1,29 mA	4,26 mW	
Contiki – Conectado	70ms	1,46 mA	4,82 mW	+7,34%
Zephyr - Conectado	50ms	1,36 mA	4,49 mW	

Fonte: Autores

A tabela 11 a seguir mostra o consumo médio durante a publicação pequena com 46 bytes de payload MQTT e o tempo total para realizar uma transmissão desta mensagem. Ao lado da potência média dos dados do Contiki OS é informado quanto maior ele é quando comparado com o Zephyr OS na mesma categoria.

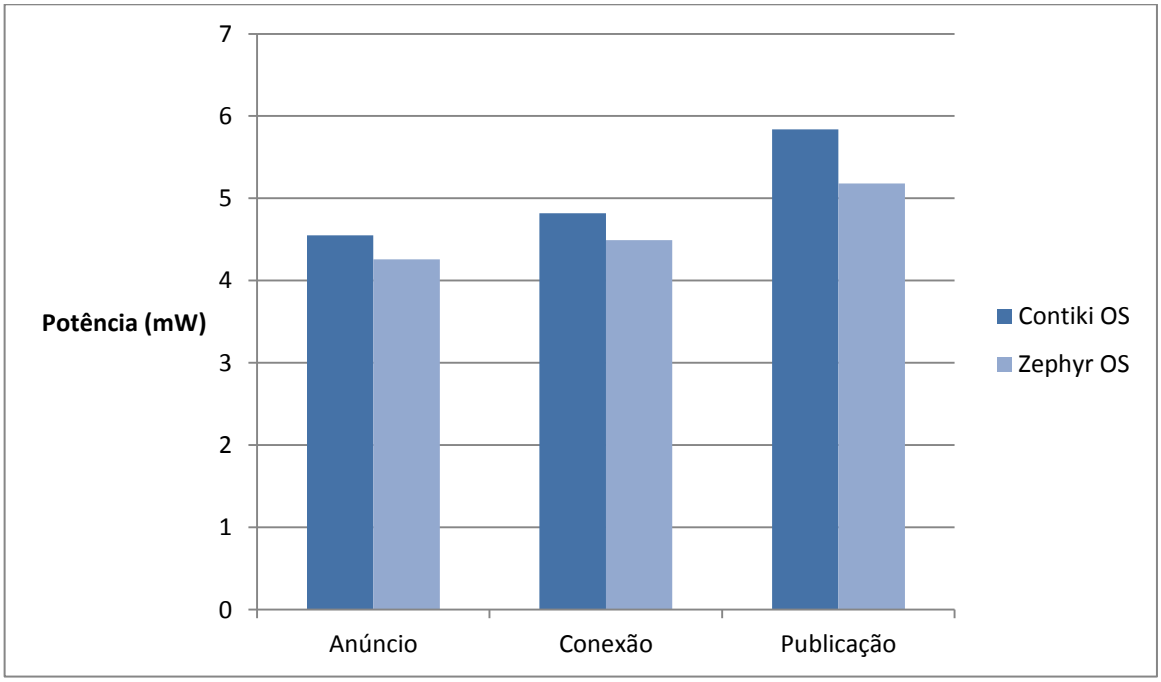
Tabela 11 Consumo de energia durante publicação de 46 bytes

	Tempo de comunicação	Corrente média	Potência média	
Contiki	522ms	1,77 mA	5,84 mW	+12,7%
Zephyr	234ms	1,57 mA	5,18 mW	

Fonte: Autores

O gráfico 1 a seguir ilustra os dados referentes a potência média das tabelas 10 e 11. Podemos observar que, mesmo se tratando de uma pequena diferença, o consumo de energia do Contiki OS sempre é um pouco maior que o do Zephyr OS. Mesmo quando em anúncio, com um intervalo de 3,4 vezes maior como mostrado na tabela 10, o Contiki ainda possui uma potência média maior.

Gráfico 1 Comparação do consumo de energia



Fonte: Autores

Após análise da potência, a tabela 12 foi compilada a partir dos dados extraídos dos testes da taxa de transferência. Ela demonstra o resultado para os testes envolvendo as mensagens grandes e pequenas em uma distância de 30 cm entre os equipamentos. Observamos que para as mensagens pequenas, o Zephyr OS possui uma taxa de transferência de 2,41 vezes (241%) maior que o Contiki, e para as mensagens grandes essa diferença é maior que o triplo. Esse contraste se deve em sua grande parte ao fato do Contiki OS realizar uma fragmentação do pacote de publicação MQTT na camada de transporte. Dessa maneira, cada pacote TCP de 108 bytes transporta apenas 32 bytes da mensagem TCP fragmentada, fazendo com que haja uma transferência de 3,3 vezes mais bytes do que a informação útil.

Já o Zephyr que faz a fragmentação na camada de enlace, envia um único pacote de 1099 bytes com uma mensagem útil de 1020 bytes, possui um overhead muito pequeno. Ou seja, enquanto o Contiki envia 3,3 vezes mais informação que a mensagem útil em si, o Zephyr envia 0,08 (8%) vezes mais informação.

Outro fato importante que reduz a taxa de transferência do Contiki, é o fato de que a comunicação quando conectada possui um intervalo de 70ms, nesse caso um evento de transmissão é gerado a cada 70ms. Diferente do Zephyr que possui um intervalo de 50ms. A tabela 12 mostra esses dados discutidos, e ao lado da taxa efetiva do Zephyr é informado quantas vezes ela é maior que a do Contiki para o mesmo teste.

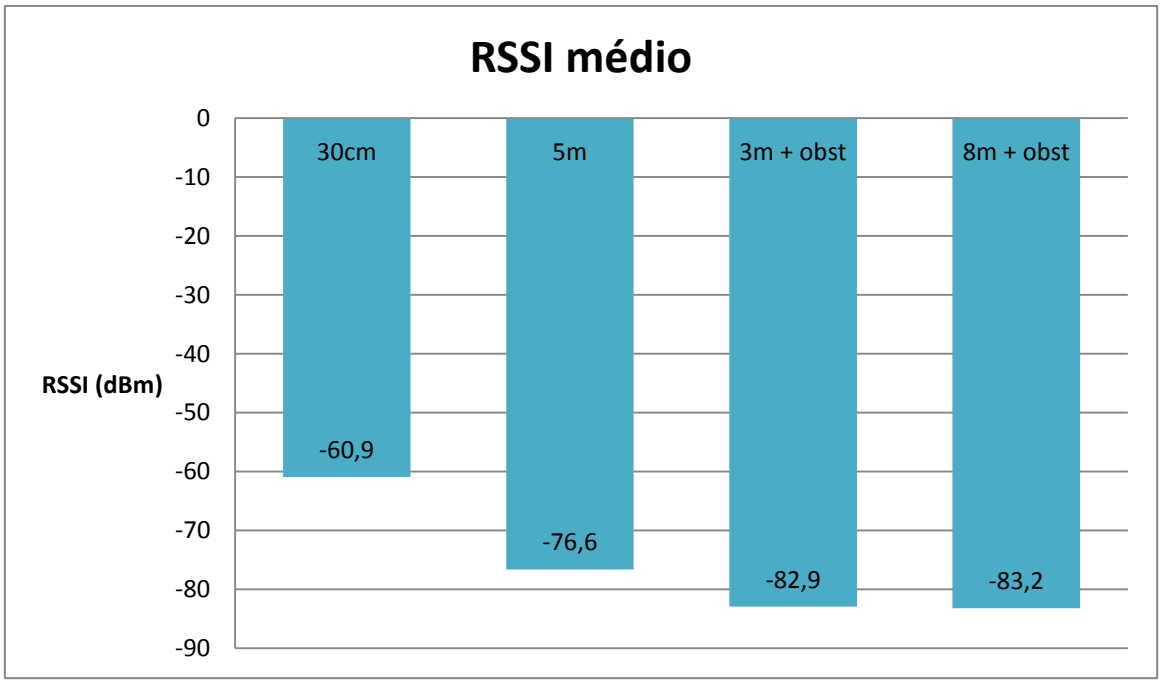
Tabela 12 Taxa efetiva de transferência dos RTOSs a 30 cm de distância

	Tamanho da mensagem MQTT	Publicações por segundo	Taxa efetiva de dados	
Contiki OS	46 Bytes	1,13	51,98 B/s	
Zephyr OS	46 Bytes	2,73	125,58 B/s	+241%
Contiki OS	1019 Bytes	0,13	132,47 B/s	
Zephyr OS	1020 Bytes	0,40	408,00 B/s	+307%

Fonte: Autores

O gráfico 2 exibe as intensidades de potência do sinal recebido pelo módulo nas diferentes posições em que os testes de taxa de transferência de dados foi avaliado. Observa-se que o aumento da distância é diretamente proporcional à redução da intensidade do sinal recebido e que o obstáculo também gera uma grande redução do RSSI.

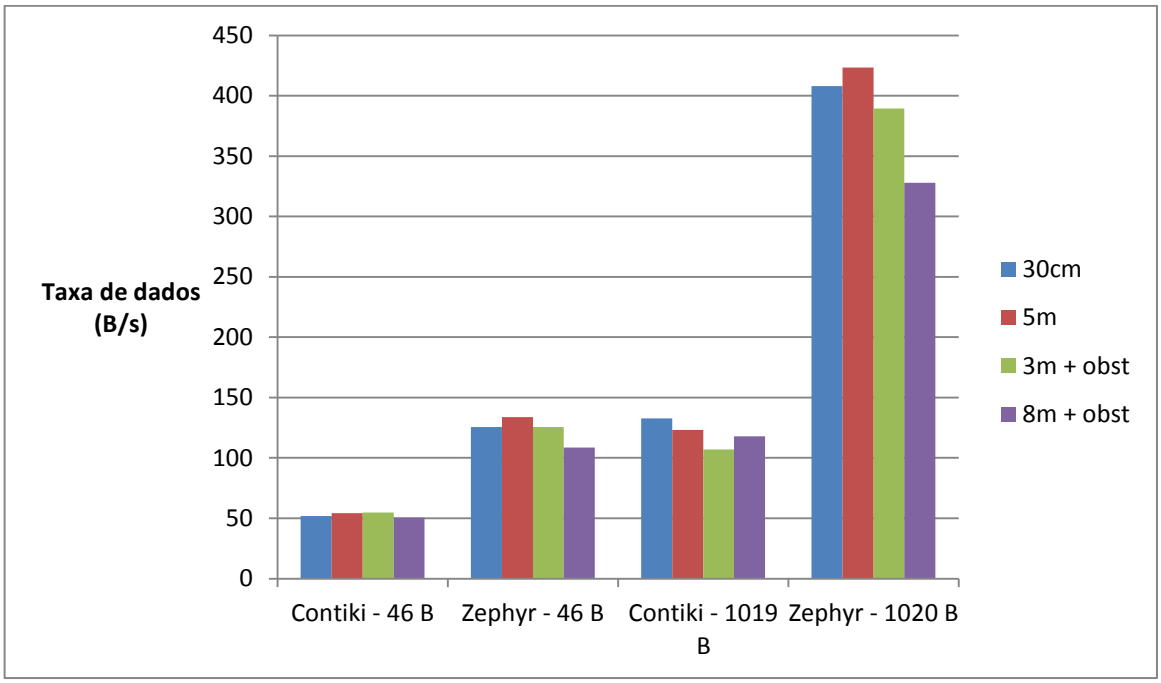
Gráfico 2 RSSI médio



Fonte: Autores

No gráfico 3 a seguir temos a comparação da taxa de Bytes transferidos entre os dois RTOSs e com as variações de distância. Podemos perceber que a variação da distância praticamente não altera a taxa de transferência, mesmo com a redução da intensidade do sinal, como apontado pelo gráfico 2 anterior. O único que sofre um pouco com o aumento da distância foi o Zephyr OS enviando as mensagens longas. Devido ao fato dele possuir uma taxa bem maior que a do Contiki, ele fica mais vulnerável às interferências do ambiente.

Gráfico 3 Comparação da taxa efetiva de transferência de Bytes



Fonte: Autores

5 CONCLUSÃO

Com a finalização desde trabalho, fica claro que o Bluetooth Low Energy é uma ótima tecnologia para ser utilizada na transferência de dados sem fio com eficiência. Mesmo no pior dos casos averiguados (demonstrados no gráfico 1), temos uma potência de 5,84 mW (1,77 mA), que é extremamente baixa para um dispositivo processar e enviar dados em uma camada de aplicação alta, como o MQTT sobre o protocolo de internet versão 6.

A respeito da distância da comunicação, o BLE teórico em condições ótimas alcança 100 metros sem obstáculos. A justificativa para o fato dos testes realizados nesse trabalho não alcançarem mais de 10 metros com obstáculos se deve ao fato de ambos os equipamentos possuírem antenas integradas na sua placa de circuito impresso. Tanto o módulo nRF52 quanto o dongle USB Bluetooth possuem pequenas antenas, diferente de uma comunicação WI-FI em que o computador possui uma antena normalmente presa ao LCD e o roteador uma antena com grande ganho. Mesmo a uma distância de 30 centímetros entre os dispositivos, a potência de recepção do sinal (RSSI) é muito baixa, possivelmente se o dispositivo central tivesse uma antena de alto ganho a distância máxima entre os dispositivos seria bem maior.

Quanto aos sistemas operacionais de tempo real avaliados, ficou claro que o desempenho do Zephyr OS foi superior, com uma diferença muito grande na taxa de transferência de dados, e uma leve vantagem no consumo de energia. Entretanto, para o Zephyr OS funcionar de tal maneira foi necessário realizar várias modificações no código de exemplo MQTT disponibilizado no repositório oficial dele. Enquanto o Contiki OS sempre funcionou perfeitamente, após encontrada a versão estável para o nRF52.

Durante todas as análises de pacotes no Wireshark, o Contiki OS nunca apresentou retransmissões, perdas de pacotes ou pacotes com erros. Já o Zephyr OS, em todas as análises algum pacote teve que ser retransmitido ou algum erro aconteceu na transmissão. Isso ocorre provavelmente devido ao fato do Zephyr OS ser muito novo, e ainda precisar de várias melhorias, enquanto o Contiki OS já possui mais de uma década e está muito mais estável, possuindo uma comunicação mais robusta e hierarquizada.

A configuração do Contiki OS também foi muito mais fácil que a do Zephyr OS, que exigiu várias horas de trabalho para desempenhar um correto funcionamento. Entretanto o Zephyr OS apresenta uma característica interessante em que o próprio main do programa é tratado como uma task, facilitando assim uma utilização inicial do código, pois não é necessário instalar uma task para programas simples e compactos.

Para finalizar, devido ao consumo de energia semelhante, é sugerível a utilização do Zephyr OS apenas se for necessário uma alta taxa de dados a ser transmitida. Se a comunicação for mais espaçada e com mensagens pequenas, o mais recomendado é o uso do Contiki OS, por causa da sua robustez e por ser menos propício a falhas. Entretanto, para um Sistema Operacional de Tempo Real com menos de 2 anos, o Zephyr tem um potencial enorme. Se as corretas decisões forem tomadas pelos desenvolvedores em pouco tempo ele poderá se tornar o melhor e mais eficiente RTOS para IoT.

REFERÊNCIAS BIBLIOGRÁFICAS

AISLELABS. **The Hitchhikers Guide to iBeacon Hardware: A Comprehensive Report by Aislelabs (2015)**. 2015. Disponível em: <https://www.aislelabs.com/reports/beacon-guide/>. Acesso em: 20 de setembro de 2017.

ANDERSSON, Mats. **Short range low power wireless devices and Internet of Things (IoT)**. 2015. Disponível em: <https://www.u-blox.com/sites/default/files/ShortRange-InternetOfThings_WhitePaper_%28UBX-14054570%29.pdf>. Acesso em: 28 de setembro de 2017.

ARGENOX TECHNOLOGIES. **Nordic Semi Announces nRF52 Series of BLE Devices**. 2015. Disponível em: <http://www.argenox.com/blog/nordic-announces-new-line-of-cortex-m4-ble-socs/>. Acesso em: 08 de agosto de 2017.

AZZOLA, Francesco. **MQTT Protocol Tutorial: Step by step guide**. 2015. Disponível em: <https://www.survivingwithandroid.com/2016/10/mqtt-protocol-tutorial.html>. Acesso em: 03 de outubro de 2017.

BAKKE, Glenn Ruben. **IPv6-Brewed Coffee Over Bluetooth Smart**. 2015. Disponível em: <http://teamarin.net/2015/03/03/ipv6-brewed-coffee-bluetooth-smart/>. Acesso em: 05 de agosto de 2017.

CONTIKI OS, **Contiki Hardware**. Disponível em: <http://www.contiki-os.org/hardware.html>. Acesso em: 08 de novembro de 2017.

ICANN, **Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied**. 2011. Disponível em: <https://www.icann.org/en/system/files/press-materials/release-03feb11-en.pdf>. Acesso em: 12 de novembro de 2017.

KLASMAN, Joakim. **Internet Access Possibilities in Bluetooth Low Energy Sensors**. 2017. 51 p. Dissertação - Lund University, Lund, 2017.

MILOVANOVIC, Viktor. **Bluetooth Low Energy – Part 1: Introduction to BLE**. 2016. Disponível em: <https://learn.mikroe.com/bluetooth-low-energy-part-1-introduction-ble/>. Acesso em: 16 de agosto de 2017.

PIERRE-LUC. **Node and MQTT, do something on message**. 2015. Disponível em: <https://stackoverflow.com/questions/32538535/node-and-mqtt-do-something-on-message>. Acesso em: 20 de setembro de 2017.

SPÖRK, Michael. **IPv6 over Bluetooth Low Energy using Contiki**. 2016. 80 p. Dissertação (Master's degree programme: Telematics) - Graz University of Technology, Graz, 2016.

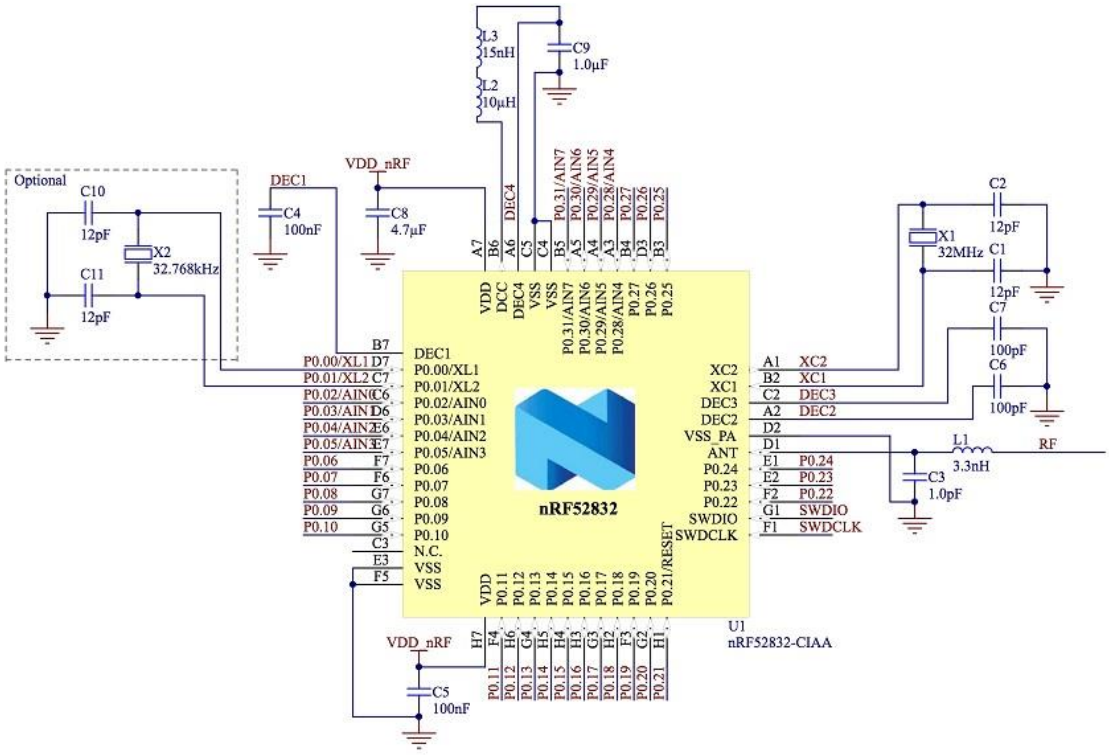
TANENBAUM, Andrew Stuart; WETHERALL, David. **Redes de Computadores**. 5. ed. Brasil: Editora Pearson, 2011, 600p.

TRELSMO, P.; DI MARCO, P.; SKILLERMARK, P.; CHIRIKOV, R.; OSTMAN, J. **Evaluating IPv6 Connectivity for IEEE 802.15.4 and Bluetooth Low Energy**. 2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW). São Francisco. 04 de maio de 2017. Disponível em: <http://ieeexplore.ieee.org/document/7919088/>. Acesso em: 03 de agosto de 2017.

WOOLEY, Martin. **Connecting to the Internet just got even easier**. Newelectronics. 10 de fevereiro de 2015. pp31 – 32. Disponível em: <http://www.newelectronics.co.uk/electronics-technology/connecting-to-the-internet-just-got-even-easier-says-the-bluetooth-sig/73741/>. Acesso em: 18 de outubro de 2017.

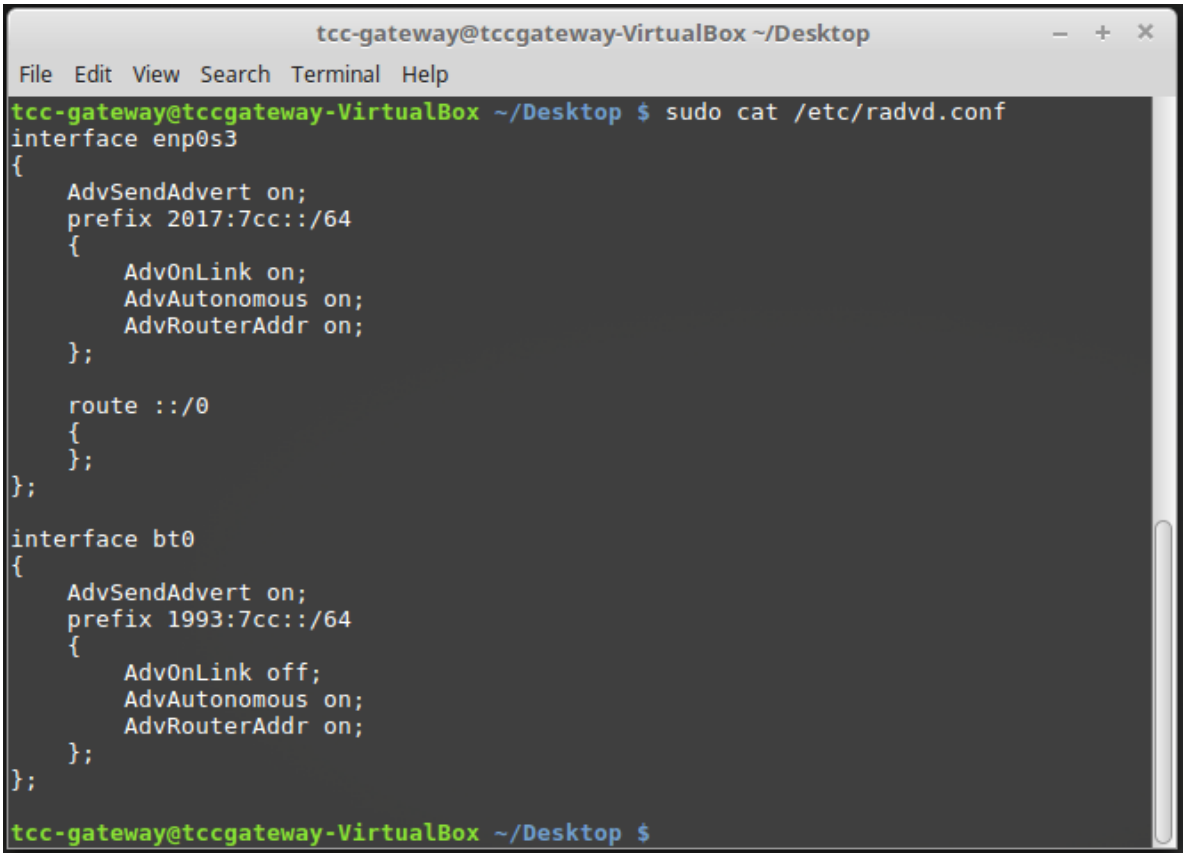
ZEPHYR PROJECT, **A small, scalable open source RTOS for IoT embedded devices**. Disponível em: <https://www.zephyrproject.org/>. Acesso em: 13 de setembro de 2017.

ANEXO A - ESQUEMÁTICO MÓDULO NRF52832



Esquemático de referência da Nordic (Disponível em: http://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.nrf52832.ps.v1.1%2Fref_circuitry.html. Acesso em 10.10.2017)

APÊNDICE A - ARQUIVO DE CONFIGURAÇÃO RADVD.CONF[47]



[47] No arquivo do RADVD o adaptador de ethernet eth0 aparece com o nome de enp0s3 devido ao fato de ser utilizado o Linux na máquina virtual, não mudando a funcionalidade do adaptador.

APÊNDICE B - COMANDOS PARA COMPILAÇÃO E GRAVAÇÃO DO CONTIKI OS

```
tcc-gateway@tccgateway-VirtualBox ~/contiki-nrf52dk-pr/examples/nrf52dk/mqtt-demo $ pwd
/home/tcc-gateway/contiki-nrf52dk-pr/examples/nrf52dk/mqtt-demo
tcc-gateway@tccgateway-VirtualBox ~/contiki-nrf52dk-pr/examples/nrf52dk/mqtt-demo $ cat compila
export NRF52_SDK_ROOT=$HOME/nrf5x-sdk
make TARGET=nrf52dk NRF52_USE_RTT=0 NRF52_WITHOUT_SOFTDEVICE=0
```

Foi criado um arquivo chamado de “compila” dentro da pasta onde é feita a gravação do programa do Contiki, esse arquivo possui os dois comandos necessários para a compilação. O primeiro é um export do path de onde se encontra o nRF5 SDK, que é utilizado quando o Contiki é compilado, e o outro comando é a compilação em si, em que é informado a placa utilizada, o não uso de RTT[48] e a utilização do SoftDevice.

Para a gravação do firmware são inseridos os seguintes comandos no openOCD:

a) reset halt: Reseta e trava o processamento do nRF52;

b) nrf52 mass_erase: limpa toda a memória do nRF52;

c) program “SoftDevice[49]” verify: Programa e verifica o softdevice;

d) program “contiki[50]” verify: Programa e verifica o firmware;

e) reset: reseta o nRF52 e executa o programa.

[48] O RTT (Real Time Transfer) é utilizado quando é feito transferência de dados tipo terminal através de um gravador como o J-Link da SEGGER. O ST-LINK/V2 não tem suporte a RTT.
[49] No lugar do SofDevice, deve ser substituído pelo path do arquivo hex dele no computador.
[50] No lugar do contiki, deve ser substituído pelo path do arquivo hex dele no computador.

APÊNDICE C - COMANDOS PARA COMPILAÇÃO E GRAVAÇÃO DO ZEPHYR OS

```
tcc-gateway@tccgateway-VirtualBox ~ $ cd zephyr/samples/bluetooth/ipsp
tcc-gateway@tccgateway-VirtualBox ~/zephyr/samples/bluetooth/ipsp $ cat compila
export ZEPHYR_BASE=/home/tcc-gateway/zephyr/
export GCCARMEMB_TOOLCHAIN_PATH="~/gccarmemb/"
export ZEPHYR_GCC_VARIANT=gccarmemb

make BOARD=nrf52_pca10040
```

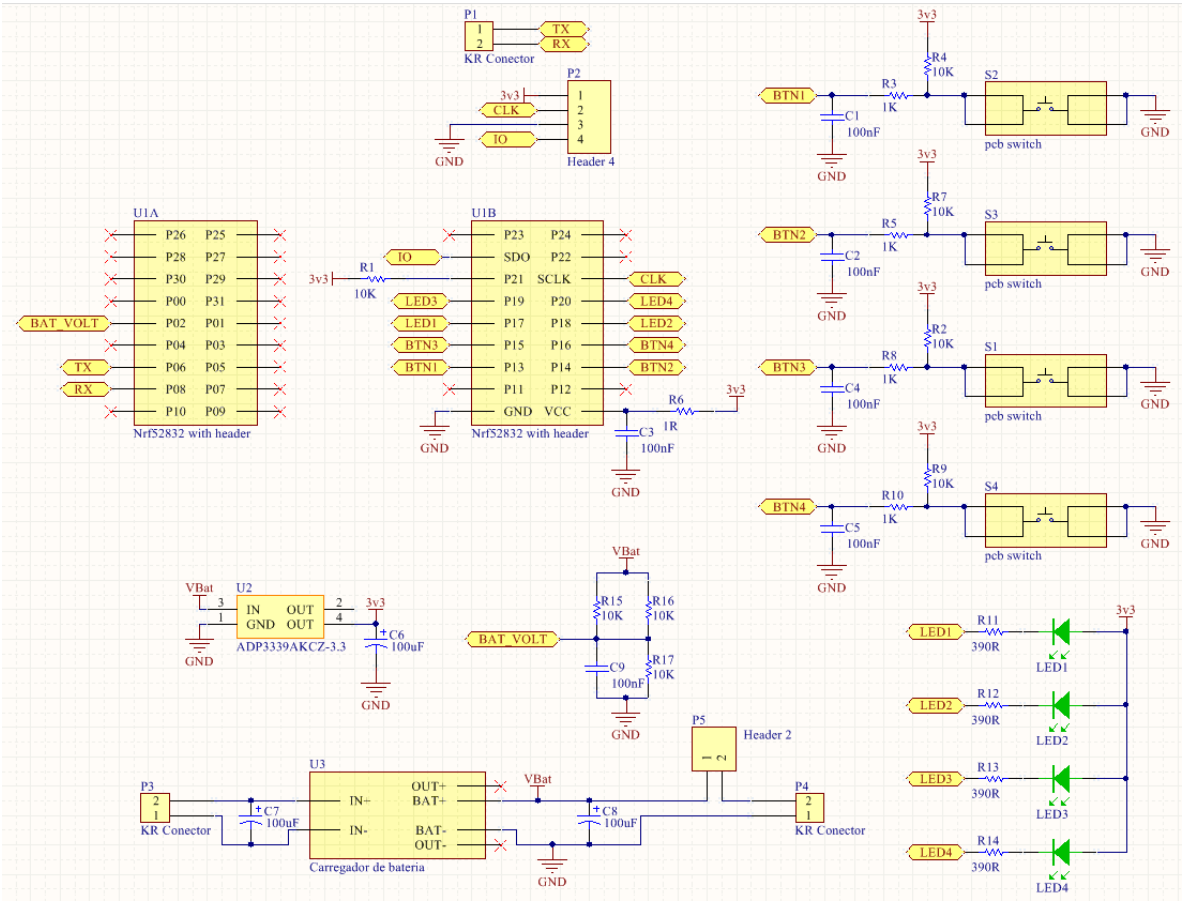
Foi criado um arquivo chamado de “compila” dentro da pasta onde é feita a gravação do programa do Zephyr, esse arquivo possui os quatro comandos necessários para a compilação. O três primeiros são exports dos paths utilizados pelo makefile para encontrar as ferramentas de compilação. O ultimo comando é o de compilação em si, em que é necessário definir a plataforma utilizada.

Para a gravação do firmware são inseridos os seguintes comandos no openOCD:

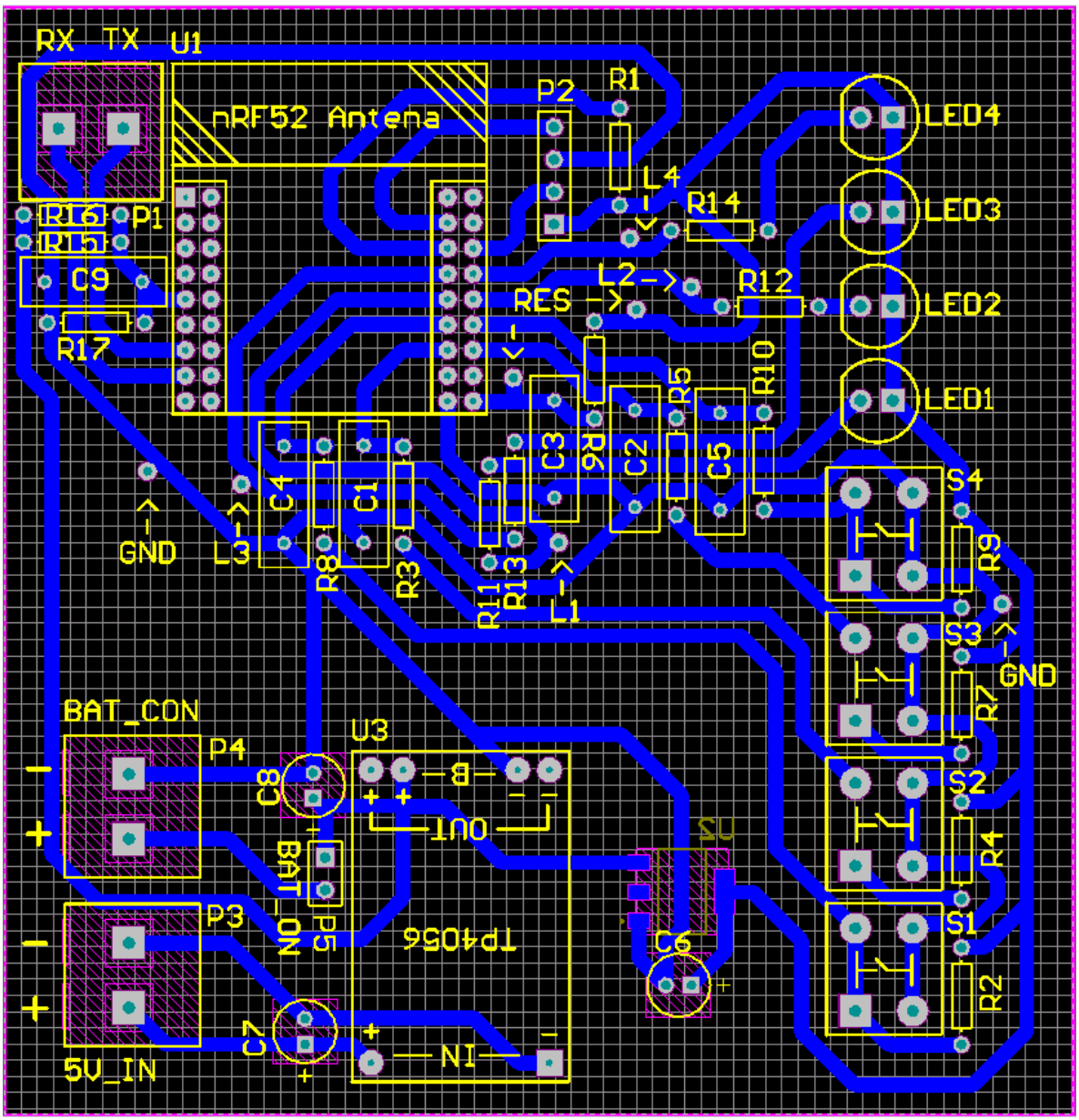
a) reset halt: Reseta e trava o processamento do nRF52;
b) nrf52 mass_erase: limpa toda a memória do nRF52;
c) program “zephyr[51]” verify: Programa e verifica o firmware;
d) reset: reseta o nRF52 e executa o programa.

[51] No lugar do zephyr, deve ser substituído pelo path do arquivo hex dele no computador.

APÊNDICE D - ESQUEMÁTICO DA PCB DESENVOLVIDA



APÊNDICE E - LAYOUT DA PCB DESENVOLVIDA



APÊNDICE F - LISTA DE COMPONENTES DA PCB

Componente	Designador na placa	Quantidade
Capacitor de cerâmica 50V 100nF	C1, C2, C3, C4, C5, C9	6
Capacitor eletrolítico 16V 100uF	C6, C7, C8	3
LED 5mm	LED1, LED2, LED3, LED4	4
Conector KRE	P1, P3, P4	3
Barra de pinos 4 posições 180º	P2	1
Barra de pinos 2 posições 180º	P5	1
Single pino 180º	*[52]	8
Resistor de carbono 1/8W 10kΩ	R1, R2, R4, R7, R9, R15, R16, R17	8
Resistor de carbono 1/8W 1kΩ	R3, R5, R8, R10	4
Resistor de carbono 1/8W 1Ω	R6	1
Resistor de carbono 1/8W 390Ω	R11, R12, R13, R14	4
Switch para PCB	S1, S2, S3, S4	4
Módulo nRF52832	U1	1
Regulador de tensão LDO 3.3V 1.5A SOT223-3	U2	1
Módulo carregador de bateria TP4056	U3	1

[52] Pinos utilizados para medições com o osciloscópio USB, não possuem designadores, mas estão marcados na PCB com seus nomes apontados por uma seta. Os pinos são: 2x GND, 2x RES, 1x L1, 1x L2, 1xL3, 1x L4.

APÊNDICE G - FUNÇÕES PARA LEITURA ANALÓGICA

```
205 void saadc_init(void)
206 {
207     // Seta resolução de 10 bits
208     NRF_SAADC->RESOLUTION = SAADC_RESOLUTION_VAL_10bit;
209 
210     // Configura o canal do SAADC para o AIN0 (P0.02) no modo normal, não diferencial
211     NRF_SAADC->CH[0].PSELP = SAADC_CH_PSELP_PSELP_AnalogInput0 << SAADC_CH_PSELP_PSELP_Pos;
212     NRF_SAADC->CH[0].PSELN = SAADC_CH_PSELN_PSELN_NC << SAADC_CH_PSELN_PSELN_Pos;
213 
214     // Configura o modo para Single Ended (não diferencial), 20uS de tempo de aquisição,
215     // referência para 1/4 do Vdd, ganho de 4x e resistores de bypass
216     NRF_SAADC->CH[0].CONFIG = (( SAADC_CH_CONFIG_MODE_SE         << SAADC_CH_CONFIG_MODE_Pos  ) |
217                                (  SAADC_CH_CONFIG_TACQ_20us      << SAADC_CH_CONFIG_TACQ_Pos  ) |
218                                (  SAADC_CH_CONFIG_REFSEL_VDD1_4  << SAADC_CH_CONFIG_REFSEL_Pos ) |
219                                (  SAADC_CH_CONFIG_GAIN_Gain1_4   << SAADC_CH_CONFIG_GAIN_Pos   ) |
220                                (  SAADC_CH_CONFIG_RESN_Bypass    << SAADC_CH_CONFIG_RESN_Pos   ) |
221                                (  SAADC_CH_CONFIG_RESP_Bypass    << SAADC_CH_CONFIG_RESP_Pos   ));
222 
223     // Configura o frequência de amostras (sample rate)
224     NRF_SAADC->SAMPLERATE = ( SAADC_SAMPLERATE_MODE_Timers << SAADC_SAMPLERATE_MODE_Pos) |
225                             ( 2000 << SAADC_SAMPLERATE_CC_Pos );
226 
227     // Configura o ponteiro para a varíavel onde deve ser salva a leitura AD
228     NRF_SAADC->RESULT.PTR  =(uint32_t)&leituraAD;
229 
230     // Seta o número de leituras consecutivas após cada trigger.
231     NRF_SAADC->RESULT.MAXCNT = 1;
232 }
233 
234 void saadc_read()
235 {
236     // Habilita leitura
237     NRF_SAADC->ENABLE = (SAADC_ENABLE_ENABLE_Enabled << SAADC_ENABLE_ENABLE_Pos);
238 
239     // Inicializa SAADC
240     NRF_SAADC->TASKS_START = 1;
241 
242     // Aguarda inicialização
243     while (NRF_SAADC->EVENTS_STARTED == 0);
244     NRF_SAADC->EVENTS_STARTED = 0;
245 
246     // Inicia amostragem
247     NRF_SAADC->TASKS_SAMPLE = 1;
248 
249     //Aguarda completar amostragem, após isso a variável leituraAD foi atualizada com a leitura
250     while (NRF_SAADC->EVENTS_END == 0);
251     NRF_SAADC->EVENTS_END = 0;
252 }
253 
```

APÊNDICE H - FUNÇÃO DE PUBLICAÇÃO NO CONTIKI OS

```
static void
publish(void)
{
  saadc_read();

  int16_t batLevel = (int16_t)(((double)leituraAD)*(3.3/1024.0)*(3.0/2.0)*100.0);

  snprintf(app_buffer, APP_BUFFER_SIZE, "Sec: %ld, Bat level: %d.%dV\r\n\r\n\
Let me tell you a story to chill the bones\r\n\
About a thing that I saw\r\n\
One night wandering in the everglades\r\n\
Id one drink but no more\r\n\
I was rambling, enjoying the bright moonlight\r\n\
Gazing up at the stars\r\n\
Not aware of a presence so near to me\r\n\
Watching my every move\r\n\
Feeling scared and I fell to my knees\r\n\
As something rushed me from the trees\r\n\
Took me to an unholy place\r\n\
And that is where I fell from grace\r\n\
Then they summoned me over to join in with them\r\n\
To the dance of the dead\r\n\
In to the circle of fire I followed them\r\n\
Into the middle I was led\r\n\
As if time had stopped still\r\n\
I was numb with fear, but still, I wanted to go\r\n\
And the blaze of the fire did no hurt upon me\r\n\
As I walked on to the coals\r\n\
Then I felt I was in a trance\r\n\
And my spirit was lifted from me\r\n\
And if only someone had the chance\r\n\
To witness what happened to me\r\n\
And I danced and I pranced and I sang with them\r\n\
All had death in their eyes\r\n\
Lifeless figures they were undead all of them\r\n\r\n",
           clock_seconds(), batLevel/100, batLevel%100);

  mqtt_publish(&conn, NULL, pub_topic, (uint8_t *)app_buffer,
               strlen(app_buffer), MQTT_QOS_LEVEL_0, MQTT_RETAIN_OFF);

  printf("APP - Publish!\n");
}
```

APÊNDICE I - SUBSCRIÇÃO AO BROKER COM A PUBLICAÇÃO GRANDE DO CONTIKI OS