

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA**

RICARDO SCHONS CIRIO

**IMPLEMENTAÇÃO E ANÁLISE DO CONTROLE
ADAPTATIVO DA POTÊNCIA DE TRANSMISSÃO EM
UMA REDE PESSOAL SEM FIO**

FLORIANÓPOLIS, 2021

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA**

RICARDO SCHONS CIRIO

**IMPLEMENTAÇÃO E ANÁLISE DO CONTROLE
ADAPTATIVO DA POTÊNCIA DE TRANSMISSÃO EM
UMA REDE PESSOAL SEM FIO**

Trabalho de conclusão de curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de engenheiro eletrônico

Orientador:
Prof. Dr. Eng. Renan Augusto Starke

FLORIANÓPOLIS, 2021

Ficha de identificação da obra elaborada pelo autor.

Cirio, Ricardo

Implementação e análise do controle adaptativo da potência de transmissão em uma rede pessoal sem fio / Ricardo Cirio ; orientação de Renan Starke. - Florianópolis, SC, 2021.
57 p.

Trabalho de Conclusão de Curso (TCC) - Instituto Federal de Santa Catarina, Câmpus Florianópolis. Bacharelado em Engenharia Eletrônica. Departamento Acadêmico de Eletrônica.
Inclui Referências.

1. Rede Pessoal Sem Fio. 2. Internet das Coisas.
3. Controle Adaptativo da Potência de Transmissão. I. Starke, Renan. II. Instituto Federal de Santa Catarina. Departamento Acadêmico de Eletrônica. III. Título.

IMPLEMENTAÇÃO E ANÁLISE DO CONTROLE ADAPTATIVO DA POTÊNCIA DE TRANSMISSÃO EM UMA REDE PESSOAL SEM FIO

RICARDO SCHONS CIRIO

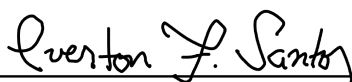
Este Trabalho foi julgado adequado para obtenção do Título de Engenheiro Eletrônico em Abril de 2021 e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação Ciência, e Tecnologia de Santa Catarina.

Florianópolis, 26 de Abril de 2021.

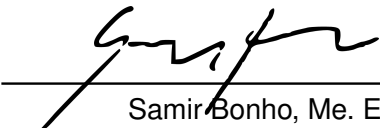
Banca Examinadora:



Renan Augusto Starke, Dr. Eng.



Everton Luiz Ferret dos Santos, Me. Eng.



Samir Bonho, Me. Eng.

AGRADECIMENTOS

Sou eternamente grato aos meus pais Leila Maria e Antonio Ricardo, por me introduzirem à eletrônica e por seus esforços imensuráveis para me proporcionar uma educação de qualidade.

Agradeço a minha esposa Paola, por me motivar durante a graduação, compreendendo o quanto isso significa para mim e me incentivando a seguir meus sonhos. Também sou grato à sua família por todo o apoio e acolhimento.

Agradecimentos ao meu irmão João Antônio, a minha madrinha de coração Carmem, e a toda a minha família por nossa amizade e pelo amor e carinho que valorizo tanto.

Aos meus amigos e colegas que foram indispensáveis para a minha formação.

À empresa 8N1 Sistemas Embarcados pela oportunidade profissional que motivou o tema deste trabalho.

Por fim, minha gratidão ao Instituto Federal de Santa Catarina pelos 11 anos de ensinamentos e experiências incríveis, excepcionalmente ao Departamento Acadêmico de Eletrônica por disponibilizar um dos melhores cursos de Engenharia Eletrônica do Brasil. Também à CAPES e ao CNPq pelas oportunidades de intercâmbio e pesquisa.

RESUMO

O ajuste da potência de transmissão em redes sem fio é uma importante técnica utilizada na manutenção da qualidade de conexão e na otimização da eficiência energética. Este trabalho traz uma implementação genérica de um algoritmo encontrado na literatura para controle adaptativo da potência de transmissão. Escrita em linguagem de programação C, essa implementação é compatível com diferentes protocolos e plataformas. Uma análise de sua operação em um dispositivo de função reduzida de uma rede pessoal sem fio é apresentada. Constatou-se que a implementação do algoritmo em um nRF52840, operando em rede OpenThread dentro de um teste controlado, resultou em uma redução de 43 % na energia utilizada para transmissão, em comparação ao uso de uma potência de transmissão fixa. Além disso não houve diminuição na taxa de recepção de pacotes.

Palavras-chave: Rede Pessoal Sem Fio, Internet das Coisas, Controle Adaptativo da Potência de Transmissão.

ABSTRACT

Adjusting the transmission power in wireless networks is an important technique used for maintaining link quality and optimizing energy efficiency. This work presents a generic implementation of an adaptive transmission power control algorithm found in the literature. Written in the C programming language, this implementation is compatible with different protocols and platforms. An analysis of its operation in a reduced-function device of a personal wireless network is presented. It was found that running the implementation on an nRF52840 as an OpenThread network device in a controlled setup resulted in a 43% reduction in the transmission energy, compared to the use of a fixed transmission power. In addition, there was no decrease in the packet reception rate.

Key-Words: *Wireless Personal Area Network, Internet of Things, Adaptive Transmission Power Control.*

LISTA DE ILUSTRAÇÕES

Figura 1 – Modelo de referência híbrido	15
Figura 2 – Arquitetura do protocolo de rede Thread	16
Figura 3 – Exemplos de topologia	17
Figura 4 – Comunicação assíncrona entre RFD e FFD	18
Figura 5 – Ciclo típico de atividade de um RFD	24
Figura 6 – Configuração experimental	27
Figura 7 – Máquina de estados	28
Figura 8 – Placa de desenvolvimento nRF52840 Dongle	31
Figura 9 – Fluxograma da aplicação	32
Figura 10 – Registro da aplicação de testes	33
Figura 11 – Relação entre PRR e RSSI médio a diferentes distâncias em um parque	34
Figura 12 – Relação entre PRR e RSSI médio a diferentes distâncias em um quintal	35
Figura 13 – Inicialização com distância de 6 m entre transceptores	36
Figura 14 – Operação com distância de 6 m entre transceptores	37
Figura 15 – Inicialização com distância de 4 m entre transceptores	38
Figura 16 – Operação com distância variável entre transceptores	39

LISTA DE ABREVIATURAS E SIGLAS

ATPC	<i>Adaptive Transmission Power Control</i> - Controle Adaptativo da Potência de Transmissão
DTPC	<i>Dynamic Transmission Power Control</i> - Controle Dinâmico da Potência de Transmissão
FFD	<i>Full-Function Device</i> - Dispositivo de Função Completa
IEEE	<i>Institute of Electrical and Electronics Engineers</i> - Instituto de Engenheiros Eletricistas e Eletrônicos
IoT	<i>Internet of Things</i> - Internet das Coisas
IP	<i>Internet Protocol</i> - Protocolo de Internet
ISO	<i>International Organization for Standardization</i> - Organização Internacional para Padronização
LQI	<i>Link Quality Indicator</i> - Indicador de Qualidade de Conexão
LR-WPAN	<i>Low-Rate Wireless Personal Area Network</i> - Rede Pessoal Sem Fio de Baixa Taxa de Transmissão
MAC	<i>Medium Access Control</i> - Controle de Acesso ao Meio
OSI	<i>Open System Interconnection</i> - Interconexão de Sistemas Abertos
OCTP	<i>Optimal Common Transmit Power</i> - Potência de Transmissão Ideal
PHY	<i>Physical Layer</i> - Camada Física
PID	Proporcional Integral Derivativo
PRR	<i>Packet Reception Ratio</i> - Taxa de Recepção de Pacotes
RAM	<i>Random Access Memory</i> - Memória de Acesso Aleatório

RFD	<i>Reduced-Function Device</i> - Dispositivo de Função Reduzida
RSSI	<i>Received Signal Strength Indicator</i> - Indicador de Intensidade do Sinal Recebido
SoC	<i>System on a Chip</i> - Sistema em um <i>Chip</i>
TCP	<i>Transmission Control Protocol</i> - Protocolo de Controle de Transmissão
TPSO	<i>Transmission Power Self-Optimization</i> - Auto-Otimização da Potência de Transmissão
WPAN	<i>Wireless Personal Area Network</i> - Rede Pessoal Sem Fio
WSN	<i>Wireless Sensor Network</i> - Rede de Sensores Sem Fio

LISTA DE SÍMBOLOS

A	ampere - Unidade de corrente elétrica
dBm	decibel miliwatt - Nível de potência em decibel relativo a 1 miliwatt
h	hora - Unidade de tempo referente a 3600 segundos
Hz	hertz - Unidade de frequência (ciclos por segundo)
J	joule - Unidade de energia
m	metros - Unidade de comprimento
s	segundo - Unidade de tempo
V	volt - Unidade de tensão elétrica
W	watt - Unidade de potência

SUMÁRIO

1	INTRODUÇÃO	13
1.1	Justificativa	13
1.2	Definição do problema	14
1.3	Objetivo Geral	14
1.4	Objetivos Específicos	14
2	FUNDAMENTAÇÃO TEÓRICA	15
2.1	Camadas de comunicação	15
2.2	Protocolo de rede OpenThread	16
2.3	Técnicas de controle da potência de transmissão	18
2.4	Controle adaptativo da potência de transmissão	21
2.5	Plataforma de hardware	23
2.6	Consumo energético utilizado para transmissão de dados	23
3	METODOLOGIA	26
3.1	Métodos aplicados	26
4	IMPLEMENTAÇÃO E AVALIAÇÃO DO ALGORITMO ATPC	28
4.1	Concepção de uma implementação genérica	28
4.2	Elaboração de uma aplicação de testes	30
4.3	Definição dos parâmetros de qualidade de conexão	33
4.4	Análise durante inicialização e operação do algoritmo	35
4.5	Avaliação do desempenho energético	40
5	CONSIDERAÇÕES FINAIS	41
	REFERÊNCIAS	43
	APÊNDICE A – APLICAÇÃO DE TESTES	46

1 INTRODUÇÃO

Tecnologias em comunicação sem fio e de circuitos integrados de baixo consumo energético evoluíram consideravelmente ao longo das últimas décadas. Além disso, surgiu a Internet das Coisas (*Internet of Things* - IoT). Utilizando a computação em nuvem como estrutura unificadora para o compartilhamento de informações, a IoT possibilita aplicações inovadoras através da análise de dados e interconexão entre sensores e atuadores. (GUBBI et al., 2013)

De acordo com Lee e Lee (2015), Redes de Sensores sem Fio (*Wireless Sensor Network* - WSN) se destacam como uma das tecnologias essenciais utilizadas no desenvolvimento de produtos e serviços baseados em IoT. Logo, foi criado o conceito de Redes Pessoais Sem Fio de Baixa Taxa de Transmissão (*Low-Rate Wireless Personal Area Network* - LR-WPAN) para atender a necessidade de WSNs de baixo custo, para uso no monitoramento e controle de funções não críticas, explica Gutierrez (2004). Santos (2013) esclarece que LR-WPANS são padronizadas pelo Instituto de Engenheiros Eletricistas e Eletrônicos (*Institute of Electrical and Electronics Engineers* - IEEE).

1.1 Justificativa

Redes Pessoais Sem Fio (*Wireless Personal Area Network* - WPAN) possuem importantes aplicações na área de saúde, embarcadas em veículos e em automação residencial, industrial e agrícola. Alguns de seus aspectos podem ser aprimorados, como o consumo de energia, segurança, confiabilidade e integridade dos dados. (SANTOS, 2013)

A grande utilização de WSNs se dá por serem compostas por dispositivos alimentados por bateria, apresentando baixo custo e facilidade de instalação. Consequentemente, a busca por soluções de *hardware* e *software* para otimizar o seu consumo energético tem sido o objetivo de diversas pesquisas. (BRINI; DESLANDES; NABKI, 2019)

1.2 Definição do problema

A implementação de um algoritmo de controle da potência de transmissão de um dispositivo em uma rede pessoal sem fio, pode possibilitar maior eficiência energética mantendo qualidade na transmissão de seus dados?

1.3 Objetivo Geral

Implementar um algoritmo de controle da potência de transmissão em um dispositivo de uma rede pessoal sem fio, analisando os impactos na qualidade da conexão e no consumo energético.

1.4 Objetivos Específicos

- Pesquisar e selecionar um algoritmo adequado para a aplicação;
- Aplicar o algoritmo escolhido em um dispositivo de uma rede pessoal sem fio;
- Definir um conjunto de testes para análise do desempenho energético e de conexão do dispositivo;
- Executar os testes definidos com e sem a implementação do algoritmo;
- Analisar e comparar os resultados.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo realiza uma revisão de conceitos de redes pessoais sem fio e um estudo de técnicas de controle da potência de transmissão. A estratégia de controle adaptativo apresentada por Lin et al. (2006) é descrita de forma mais detalhada, por ser a utilizada neste trabalho. Também são apresentados o protocolo de rede e a plataforma de *hardware* utilizados no Capítulo 4.

2.1 Camadas de comunicação

Um modelo de referência utilizado no estudo de redes de computadores é o baseado no Protocolo de Controle de Transmissão (*Transmission Control Protocol* - TCP), no Protocolo de Internet (*Internet Protocol* - IP) e na arquitetura de Interconexão de Sistemas Abertos (*Open System Interconnection* - OSI) definida pela Organização Internacional para Padronização (*International Organization for Standardization* - ISO). Este modelo híbrido é composto pelas camadas descritas na Figura 1. (TANENBAUM; WETHERALL, 2011)

Figura 1 – Modelo de referência híbrido

5	Aplicação
4	Transporte
3	Rede
2	Enlace
1	Física

Fonte: Adaptado de Tanenbaum e Wetherall (2011).

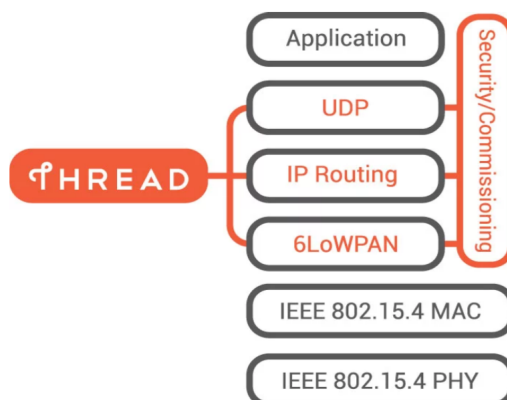
A Camada Física (*Physical Layer* - PHY) detalha a forma de transmissão de bits por um meio de comunicação, enquanto a Camada de Enlace especifica como transmitir mensagens entre dispositivos com um nível de confiabilidade determinado. Além disso, a Camada de Enlace possui uma subcamada chamada de Controle de Acesso ao Meio (*Medium Access Control* - MAC). Essa subcamada apresenta-se importante para redes sem fio cuja transmissão é de natureza difusa, pois possui protocolos que organizam o uso dos canais de acesso múltiplo evitando colisões. (TANENBAUM; WETHERALL, 2011)

A Camada de Rede se encarrega da formação de redes a partir de múltiplas conexões entre dispositivos, lidando com questões como roteamento e topologias de rede. A forma com que os dados são enviados é abstraída pela Camada de Transporte. E a Camada de Aplicação é constituída por programas que utilizam a rede. (TANENBAUM; WETHERALL, 2011)

2.2 Protocolo de rede OpenThread

OpenThread é uma implementação de código livre do protocolo Thread, lançada pela empresa Google. Thread foi desenvolvido para dispositivos de baixo consumo de energia, operando em redes *mesh* (de malha) no padrão IEEE 802.15.4-2006, e possui amplas aplicações de IoT. De acordo com o Thread Group (2021), este protocolo é utilizado por produtos de empresas como Apple, Eve Systems, Google e Siemens. Thread é responsável pelas camadas de transporte e rede, conforme demonstra a Figura 2 (THREAD GROUP, 2018).

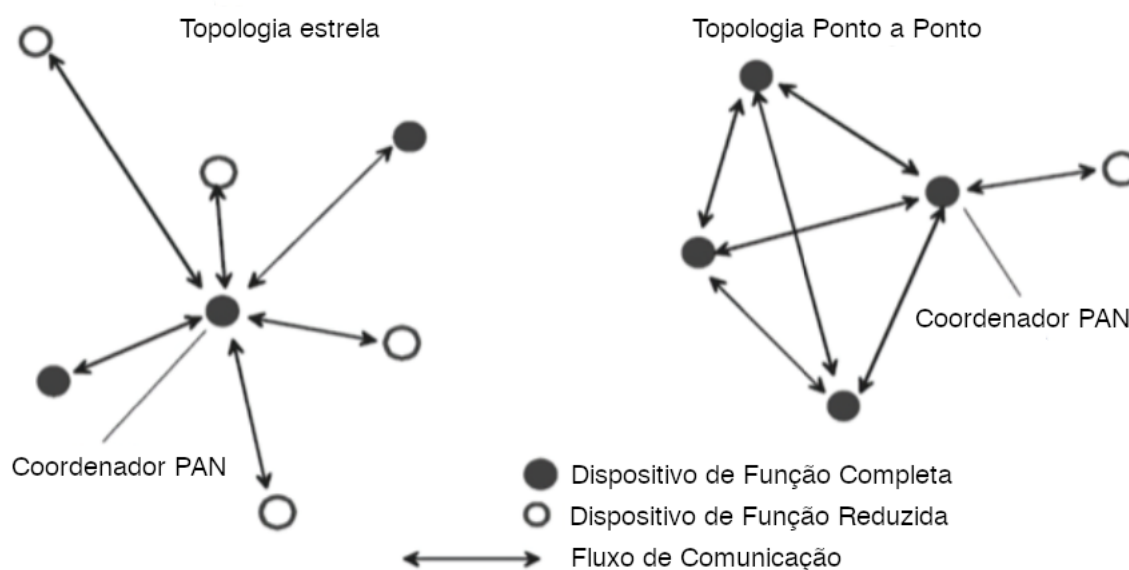
Figura 2 – Arquitetura do protocolo de rede Thread



Fonte: Thread Group (2018).

O padrão técnico IEEE 802.15.4 estabelece definições para as camadas PHY e MAC de LR-WPANs. Estas visam a transferência de dados com confiabilidade, além de custo e consumo energético baixos. Embora a formação da rede seja responsabilidade das camadas superiores, o IEEE 802.15.4 prevê as duas topologias exibidas na Figura 3. (IEEE COMPUTER SOCIETY, 2020)

Figura 3 – Exemplos de topologia

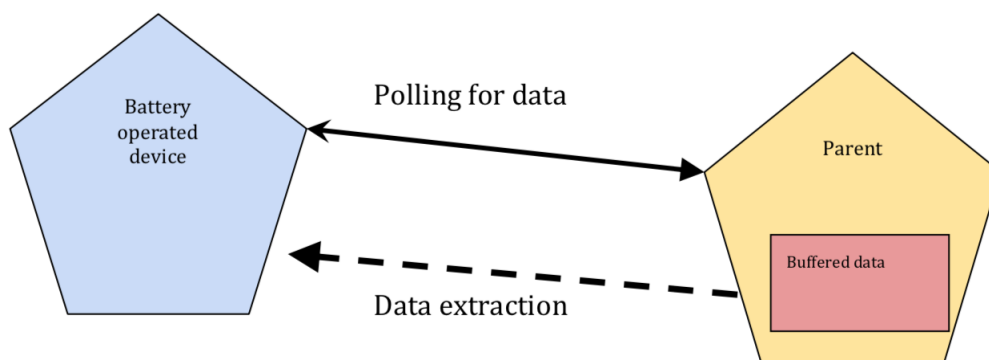


Fonte: Adaptado de IEEE Computer Society (2020).

Na topologia em estrela, um coordenador concentra todo o fluxo de informação, já na ponto a ponto, um dispositivo pode se comunicar diretamente com qualquer outro que estiver dentro do seu alcance. Em uma rede 802.15.4, um nó pode ser um Dispositivo de Função Completa (*Full-Function Device* - FFD) ou um Dispositivo de Função Reduzida (*Reduced Function Device* - RFD). A diferença é que FFDs têm a capacidade de se tornar coordenadores da rede, enquanto RFDs assumem aplicações mais simples e normalmente não exigem muitos recursos. (IEEE COMPUTER SOCIETY, 2020)

Thread utiliza o modo de operação assíncrono do IEEE 802.15.4. Desta forma, dispositivos podem economizar energia desabilitando suas funções e entrando em modo *sleep* (dormir). A Figura 4 demonstra um exemplo de como ocorre a comunicação assíncrona. A seta contínua representa o envio de uma mensagem *poll* (captação), onde o RFD identificado em azul verifica se o FFD, em amarelo, possui alguma mensagem para ele. Caso existam dados para o RFD, estes são transmitidos em uma mensagem representada pela seta tracejada. (THREAD GROUP, 2018)

Figura 4 – Comunicação assíncrona entre RFD e FFD



Fonte: Thread Group (2018).

Embora dispositivos em redes *mesh* mantenham conexões entre todos os seus nós adjacentes, RFDs são uma exceção no OpenThread. Nele, RFDs se comunicam exclusivamente a um roteador, sem realizar encaminhamento de pacotes. São os únicos dispositivos que podem operar em modo de economia de energia, desligando seu transceptor de rádio. Por outro lado, roteadores devem manter seu rádio sempre habilitado. (OPENTHREAD, 2020)

2.3 Técnicas de controle da potência de transmissão

Cui, Zhang e Shang (2007) apontam a utilização da menor potência de transmissão possível, atendendo a um nível de qualidade de conexão aceitável, como uma estratégia para a economia de energia em redes de sensores sem fio. Diversos estudos buscam esse tipo de otimização, com algoritmos que variam de acordo com a forma de escolha de vizinhos, métrica de otimização utilizada e experimentos com parâmetros distintos. (JEONG; CULLER; OH, 2007).

Existem situações em que se pode considerar que a potência de transmissão ideal é a mínima que mantém a conectividade da rede quando aplicada a todos os nós. Exemplos são casos de transceptores simples, de potência de transmissão fixa, ou quando existe dificuldade em alterá-la. Esse conceito é conhecido como Potência de Transmissão Ideal Comum (*Optimal Common Transmit Power* - OCTP). (PANICHPAPIBOON; FERRARI; Tonguz,

2006). Entretanto, esta abordagem vem se tornando cada vez menos necessária, considerando a facilidade em se ajustar a potência de transmissão de dispositivos de Sistema em um Chip (*System on a Chip* - SoC) e transceptores modernos.

Uma solução baseada no conceito de vizinho eficaz é introduzida por Jeong, Culler e Oh (2007). Nela, um nó A é considerado vizinho eficaz de B caso A saiba que B é capaz de receber suas mensagens com Indicador de Intensidade do Sinal Recebido (*Received Signal Strength Indicator* - RSSI) acima de um valor estipulado. Assim, cada dispositivo realiza uma busca por vizinhos eficazes. Um algoritmo chamado de Controle Dinâmico da Potência de Transmissão (*Dynamic Transmission Power Control* - DTPC) é implementado para convergir a quantidade de vizinhos eficazes para um valor predeterminado. Em comparação com o uso de potência de transmissão fixa, este apresentou redução de 16 % na energia consumida. (JEONG; CULLER; OH, 2007)

Lavratti et al. (2010) sugerem o uso de uma medida de eficiência da comunicação, caracterizada pela razão entre mensagens recebidas e enviadas. Esta medida pode ser utilizada como parâmetro para atingir uma redução no consumo energético. A técnica de Auto-Otimização da Potência de Transmissão (*Transmission Power Self-Optimization* - TPSO) envolve o ajuste da potência de transmissão de acordo com a eficiência observada ao longo do tempo, por meio de um algoritmo simples de comparação. Com a TPSO, os autores obtiveram uma redução de 18 % no consumo energético, comparado ao uso de uma potência de transmissão fixa, mantendo a eficiência acima de 90 %. (LAVRATTI et al., 2010)

Priyesh e Bharti (2017) também implementaram um DTPC, mas utilizando um controlador Proporcional Integral Derivativo (PID). Eles obtiveram bons resultados, reduzindo o consumo energético em mais de 50 % se comparado ao uso de uma potência fixa. Porém, a dificuldade em realizar o ajuste inicial dos parâmetros do controlador apresenta-se como uma desvantagem desta abordagem.

A partir de estudos empíricos, Lin et al. (2006) identificaram significativa variação na qualidade do sinal ao longo do tempo entre dispositivos de uma

rede de sensores sem fio. Também perceberam uma correlação entre a potência de transmissão e RSSI e o Indicador de Qualidade de Conexão (*Link Quality Indicator* - LQI). Por conseguinte, eles desenvolveram o algoritmo de Controle Adaptativo da Potência de Transmissão (*Adaptive Transmission Power Control* - ATPC). Neste, cada dispositivo mantém um modelo matemático da relação entre sua potência de transmissão e qualidade do sinal recebido por seus vizinhos. O modelo é atualizado por um mecanismo de realimentação, permitindo que mensagens sejam transmitidas de forma otimizada. Os autores obtiveram uma redução no consumo energético de 46 % com relação ao uso da máxima potência de transmissão, e de 78 % com relação ao uso da técnica OCTP. Além disso, obtiveram uma Taxa de Recepção de Pacotes (*Packet Reception Ratio* - PRR) acima de 98 % com o algoritmo. (LIN et al., 2006)

Um comparativo entre as técnicas mencionadas é realizado na Tabela 1. As informações de redução no consumo energético apresentadas são em relação ao uso de uma potência de transmissão fixa.

Tabela 1 – Comparação de técnicas de controle da potência de transmissão

Técnica	Redução no consumo energético (%)
ATPC	46
DTPC	16
DTPC (PID)	>50
TPSO	18

Fonte: Elaboração própria.

Segundo Priyesh e Bharti (2017), ATPC é um dos algoritmos mais populares para controle da potência de transmissão, baseado em um modelo linear do RSSI. Para Jeong, Culler e Oh (2007), ele também pode ser utilizado para ajustar o alcance de vizinhos desejado. Percebe-se que o ATPC proporciona um bom custo-benefício com relação a sua complexidade e seus resultados. Portanto, uma descrição mais detalhada deste algoritmo é realizada na próxima seção.

2.4 Controle adaptativo da potência de transmissão

Este algoritmo utiliza um modelo preditivo que envolve um vetor TP e uma matriz R. O vetor TP contém N possíveis níveis de potência de transmissão tp_j , $TP = \{tp_1, tp_2, \dots, tp_N\}$. Durante sua inicialização, mensagens do tipo *beacon* são enviadas utilizando cada nível de potência disponível. Os valores de RSSI correspondentes a estas mensagens, medidos por n vizinhos, são armazenados na matriz R, composta por vetores R_i , $R = \{R_1, R_2, \dots, R_n\}^T$. Cada vetor R_i é composto por elementos r_i^j , onde i é o índice do vizinho e j é o índice do nível de potência de transmissão. A relação entre a potência de transmissão e o RSSI é aproximadamente linear e caracterizada pela Equação 2.1. (LIN et al., 2006)

$$r_i^j = a_i \cdot tp_j + b_i \quad (2.1)$$

Onde:

r_i^j = RSSI medido no vizinho de índice i correspondente ao *beacon* transmitido com potência tp_j (dBm)

tp_j = nível da potência de transmissão de índice j (dBm)

a_i = coeficiente angular da relação entre tp_j e r_i

b_i = coeficiente linear da relação entre tp_j e r_i

Os coeficientes a_i e b_i podem ser estimados pelo método dos mínimos quadrados. A soma do quadrado dos resíduos da Equação 2.1 é apresentada na Equação 2.2.

$$S = \sum_{j=1}^N (r_i^j - a_i tp_j - b_i)^2 \quad (2.2)$$

Onde:

S = soma do quadrado dos resíduos

Utilizando o método descrito por Chapra (2012), deriva-se S com respeito a cada coeficiente desconhecido, conforme Equação 2.3, igualando a zero para encontrar seu valor mínimo.

$$\begin{aligned}\frac{\partial S}{\partial a_i} &= -2 \sum_{j=1}^N [(r_i^j - a_i t p_j - b_i) t p_j] = 0 \\ \frac{\partial S}{\partial b_i} &= -2 \sum_{j=1}^N (r_i^j - a_i t p_j - b_i) = 0\end{aligned}\quad (2.3)$$

Assim, chega-se no sistema de duas equações e duas variáveis desconhecidas da Equação 2.4.

$$\begin{cases} (\sum_{j=1}^N t p_j^2) a_i + (\sum_{j=1}^N t p_j) b_i &= \sum_{j=1}^N r_i^j t p_j \\ (\sum_{j=1}^N t p_j) a_i + N b_i &= \sum_{j=1}^N r_i^j \end{cases}\quad (2.4)$$

A solução deste sistema permite o cálculo dos coeficientes desejados, conforme Equações 2.5 e 2.6.

$$a_i = \frac{\sum_{j=1}^N r_i^j \sum_{j=1}^N t p_j - N \sum_{j=1}^N r_i^j t p_j}{(\sum_{j=1}^N t p_j)^2 - N \sum_{j=1}^N t p_j^2}\quad (2.5)$$

$$b_i = \frac{\sum_{j=1}^N r_i^j t p_j \sum_{j=1}^N t p_j - \sum_{j=1}^N r_i^j (\sum_{j=1}^N t p_j)^2}{(\sum_{j=1}^N t p_j)^2 - N \sum_{j=1}^N t p_j^2}\quad (2.6)$$

De acordo com as observações realizadas por Lin et al. (2006), os coeficientes variam ao longo do tempo, portanto é importante que o modelo seja constantemente atualizado. Enquanto o coeficiente angular sofre poucas variações, o coeficiente linear varia significativamente de maneira contínua e lenta. Sendo assim, é possível negligenciar a atualização de a_i , atualizando b_i com base na Equação 2.7:

$$b_i(t) = b_i(t-1) - \frac{\sum_{t=1}^K [RSSI_{LQ} - r_i(t-1)]}{K}\quad (2.7)$$

Onde

K = número de notificações recebidas do vizinho de índice i no instante $t-1$

r_i = RSSI de valor indesejado, medido no vizinho i

$RSSI_{LQ}$ = valor-alvo de RSSI (dBm)

t = tempo (s)

Essa atualização pode ser realizada durante a comunicação entre os nós da rede, onde cada dispositivo monitora o RSSI das mensagens que recebe. No caso de detectar um RSSI fora de uma margem estipulada, em torno de um valor predefinido, envia-se uma mensagem de notificação identificando a discrepância. Experimentos indicam que uma troca de mensagens por hora entre um par é suficiente para manter o modelo atualizado. (LIN et al., 2006)

2.5 Plataforma de hardware

O SoC nRF52840 é compatível com o protocolo OpenThread. Este dispositivo possui um núcleo ARM Cortex-M4 com memória *flash* de 1 MB, 256 kB de RAM e transceptor de rádio de 2.4 GHz, conforme identificado pela fabricante Nordic Semiconductor (2019).

O nRF52840 possui potência de transmissão entre -20 dBm e 8 dBm, configurável em passos de 4 dBm. Ao operar com tensão regulada de $3,3$ V, a corrente típica I_j , absorvida pelo rádio do SoC para realizar a transmissão de dados em cada nível de potência de transmissão j é apresentada na Tabela 2. (NORDIC SEMICONDUCTOR, 2019)

Tabela 2 – Consumo de corrente do rádio (transmissão)

j	Potência (dBm)	I_j (mA)
1	-20	5,6
2	-16	6,0
3	-12	6,4
4	-8	7,2
5	-4	8,1
6	0	10,6
7	4	21,4
8	8	32,7

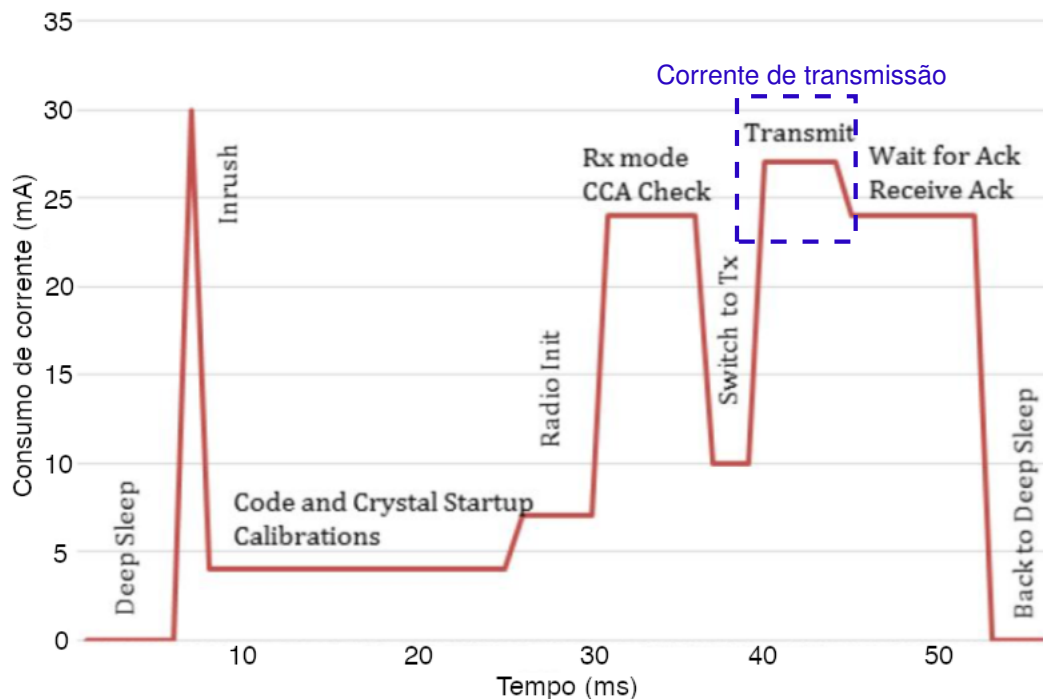
Fonte: Adaptado de Nordic Semiconductor (2019).

2.6 Consumo energético utilizado para transmissão de dados

Um exemplo do consumo de corrente apresentado por um RFD em uma rede Thread é apresentado na Figura 5. A taxa de transmissão nesse

tipo de rede é de 250 kbps. Logo, o tempo de transmissão de cada *byte* é de $T_B = 32 \mu s$. Observa-se destacado em azul o comportamento da corrente durante a etapa de transmissão de dados. (THREAD GROUP, 2018)

Figura 5 – Ciclo típico de atividade de um RFD



Fonte: Adaptado de Thread Group (2018).

É possível calcular o valor da potência média durante a transmissão de dados por meio da Equação 2.8 adaptada de Hart (2011).

$$P_j = \frac{1}{T_B} \int_0^{T_B} v(t)i(t) dt = VI_j \quad (2.8)$$

Onde:

P_j = potência média durante transmissão utilizando potência de índice j (W)

v = tensão elétrica (V)

i = corrente elétrica (A)

t = tempo (s)

T_B = período de transmissão de um *byte*

V = tensão constante aplicada sobre o rádio (V)

I_j = corrente média utilizada pelo rádio para transmissão utilizando a potência de índice j (A)

Desta forma, a energia utilizada pelo rádio na transmissão de um *byte*, em função da corrente média, é encontrada conforme Equação 2.9, também adaptada de Hart (2011).

$$E_j = P_j T_B = (VI_j)T_B = (3,3)(I_j)(32 \times 10^{-6}) = 10,56I_j \times 10^{-6} \quad (2.9)$$

Onde:

E_j = energia de transmissão de um *byte* utilizando a potência de transmissão de índice j (J)

A Tabela 3 apresenta os valores calculados para cada nível de potência do nRF52840 utilizando a Equação 2.9 e os dados da Tabela 2.

Tabela 3 – Energia de transmissão por *byte* para cada nível de potência

j	Potência (dBm)	E_j (μ J)
1	-20	0,591
2	-16	0,634
3	-12	0,676
4	-8	0,760
5	-4	0,855
6	0	1,12
7	4	2,26
8	8	3,45

Fonte: Elaboração própria (2021).

3 METODOLOGIA

A princípio, apresentou-se a problemática de aumentar a eficiência energética, mantendo a qualidade de conexão de um RFD utilizando potência de transmissão adaptativa. Definiram-se então os objetivos do trabalho, buscando responder o problema levantado através de uma pesquisa pré-experimental, de natureza exploratória. Realizou-se então uma revisão da literatura, envolvendo o estudo de técnicas disponíveis e possibilitando a escolha do algoritmo ATPC como ferramenta para alcançar os objetivos estipulados. Este foi selecionado prezando por sua complexidade moderada combinada aos resultados promissores apresentados por Lin et al. (2006).

Devido a indisponibilidade de uma implementação pronta do algoritmo, decidiu-se por realizar uma elaboração própria, possibilitando compatibilidade com diferentes aplicações e plataformas, sendo genérica e de código livre. Conjuntamente, desenvolveu-se uma aplicação de testes, que inclui o módulo ATPC por meio de *link* estático, para realização da implementação e avaliação da técnica selecionada em uma Rede de Área Pessoal (*Personal Area Network* - PAN) operando no padrão IEEE 802.15.4. Nela, um RFD envia mensagens periodicamente para um FFD. Definiu-se o RFD como principal objeto de estudo, devido a importância em manter seu consumo energético eficiente.

3.1 Métodos aplicados

Um estudo da relação entre RSSI e PRR foi realizado para definir os parâmetros de qualidade da conexão a serem utilizados pelo ATPC. Esse procedimento envolveu a escolha de um valor-alvo de RSSI e seus limites de tolerância, para realização de ajustes no caso de alguma mensagem estiver fora dos limites esperados.

A avaliação de desempenho da implementação tem como objetivo identificar se o algoritmo pode resultar em maior eficiência no consumo de energia mantendo uma boa qualidade de conexão. Para isso, verificou-se a variação de RSSI e PRR ao longo do tempo sob diferentes condições, utilizando a potência de transmissão adaptativa e fixa, para fins de comparação, e realizando

uma estimativa do consumo energético. Desta forma também foi possível avaliar o funcionamento da implementação desenvolvida.

As medidas realizadas ao longo deste trabalho foram efetuadas mantendo-se a mesma posição entre as placas de desenvolvimento, evitando a influência de diferentes acoplamentos entre suas antenas, conforme ilustrado na Figura 6. Os experimentos se deram a céu aberto e com os dispositivos a 45 cm do solo, tendo a distância entre os mesmos e o período de envio das mensagens como variáveis independentes. As comunicações sem fio dos computadores portáteis conectados às placas foram desligadas para evitar possíveis interferências.

Figura 6 – Configuração experimental

(a) Dispositivos a uma distância de 2 m



(b) RFD



(c) FFD



Fonte: Elaboração própria (2021).

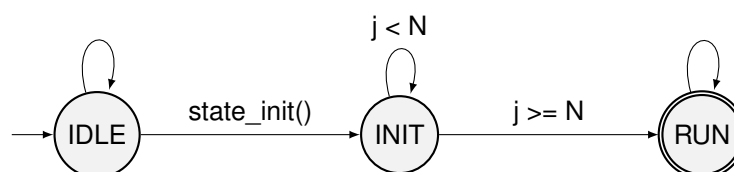
4 IMPLEMENTAÇÃO E AVALIAÇÃO DO ALGORITMO ATPC

Neste capítulo, encontra-se uma descrição do desenvolvimento da implementação do algoritmo e de uma aplicação de testes. São apresentados os resultados referentes a definição dos parâmetros utilizados nos experimentos e a avaliação do seu funcionamento.

4.1 Concepção de uma implementação genérica

Desenvolveu-se a implementação genérica do algoritmo ATPC disponibilizada no GitHub por Cirio (2021). Escrito em linguagem de programação C, o código implementa uma máquina de estados finitos, ilustrada pela Figura 7, e composta por três estados: IDLE (inativo), INIT (inicialização) e RUN (operação). A aplicação deve chamar a função *atpc_process* periodicamente para executá-la. O uso de exclusão mútua ou semáforos pode ser necessário, caso haja recursos compartilhados utilizados pelo módulo ATPC.

Figura 7 – Máquina de estados



Fonte: Elaboração própria (2021).

O dispositivo inicia no estado inativo, onde se encontra ocioso. Uma chamada da função *atpc_init* resulta na transição para o estado de inicialização, onde mensagens do tipo *beacon* são emitidas utilizando cada nível de potência disponível. Aguarda-se então por respostas de seus vizinhos entre a transmissão de cada *beacon*. Quando o índice *j*, referente aos *beacons* enviados, atinge um valor igual ou superior ao número de possíveis níveis de potência de transmissão *N*, o dispositivo muda para o seu estado final, pois concluiu o envio dos *beacons*. Logo, entra-se no estado de operação, no qual são calculados os coeficientes *a* e *b* da parametrização entre potência de transmissão e RSSI para cada vizinho, atualizando o coeficiente *b* conforme necessário. Os cálculos são

realizados utilizando aritmética de ponto fixo, com o objetivo de manter maior compatibilidade entre diferentes plataformas.

Antes de iniciar o módulo ATPC, a aplicação deve configurá-lo por meio da função *atpc_conf*, conforme exemplo apresentado no Código 4.1. Durante a configuração, são passados por referência as *callbacks* (chamadas de retorno) utilizadas pelo módulo e um vetor contendo os possíveis níveis de potência de transmissão. As *callbacks* consistem de funções para operação de listas duplamente encadeadas, indicar mudanças de estado, enviar mensagens e obter o tempo atual do sistema.

Código 4.1 – Exemplo de chamada da função *atpc_conf*

```
1 atpc_conf(&atpc_cbs,
2         DEFAULT_TX_POWER, TX_POWER, sizeof(TX_POWER)/sizeof(int8_t),
3         RSSI_SETPPOINT, RSSI_UPPER_THRESH, RSSI_LOWER_THRESH,
4         MULTICAST_ADDR, BEACON_TIMEOUT);
```

Fonte: Elaboração própria (2021).

Também são parâmetros da função de configuração os seguintes valores: nível de potência de transmissão padrão da aplicação, valor-alvo de RSSI; limites superior e inferior de tolerância de RSSI; endereço utilizado pela rede para difundir os *beacons* de inicialização; e tempo que se deve esperar pelas respostas dos *beacons*.

Deve-se indicar ao módulo ATPC o recebimento de mensagens, passando por referência uma estrutura contendo suas informações. Um exemplo é apresentado no Código 4.2.

Código 4.2 – Exemplo de indicação de recebimento de mensagem

```
1 static void handle_receive(uint16_t src_addr, int8_t rssi,
2                           void *buff, uint8_t length) {
3     atpc_data_ind_t data_ind;
4     atpc_msg_t msg;
5     data_ind.src_addr = src_addr;
6     data_ind.rssi = rssi;
7     switch(buff[0]) {
8         case APP_MSG:
9             msg.type = ATPC_QUALITY_MONITOR;
10            data_ind.data = &msg;
11            break;
12            case ATPC_MSG:
```

```

13         data_ind.data = (atpc_msg_t *) (buff + sizeof(app_msg_type_t))
14     ;
15         break;
16     default:
17         break;
18 }
19 atpc_data_ind(&data_ind);
20 }

```

Fonte: Elaboração própria (2021).

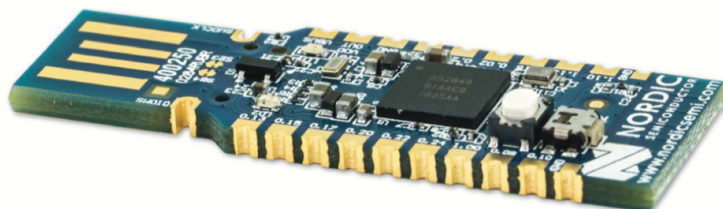
Além disso, é responsabilidade da aplicação chamar funções para registrar e remover vizinhos. Destaca-se que as funções mencionadas independem da máquina de estados, pois são executadas no escopo da aplicação. Portanto, um dispositivo no estado IDLE pode responder *beacons* e monitorar o sinal de seus vizinhos que estiverem implementando o algoritmo ATPC, sem a necessidade de inicializá-lo em sua própria instância.

Os cálculos necessários são realizados utilizando aritmética de ponto fixo, mantendo maior compatibilidade entre plataformas. O arquivo de biblioteca estática do módulo ATPC gerado durante compilação é de 28 182 *bytes*, e o seu uso de Memória de Acesso Aleatório (*Random Access Memory* - RAM) depende do número de dispositivos vizinhos.

4.2 Elaboração de uma aplicação de testes

Uma aplicação simples foi desenvolvida utilizando o protocolo OpenThread e a plataforma nRF52840. A relevância do Thread na indústria e a licença de *software* permissiva do OpenThread impactaram em sua escolha. Além disso, devido às características do nRF52840 e ao seu baixo custo, utilizaram-se duas placas de desenvolvimento como a demonstrada na Figura 8 para executar o *firmware* desenvolvido.

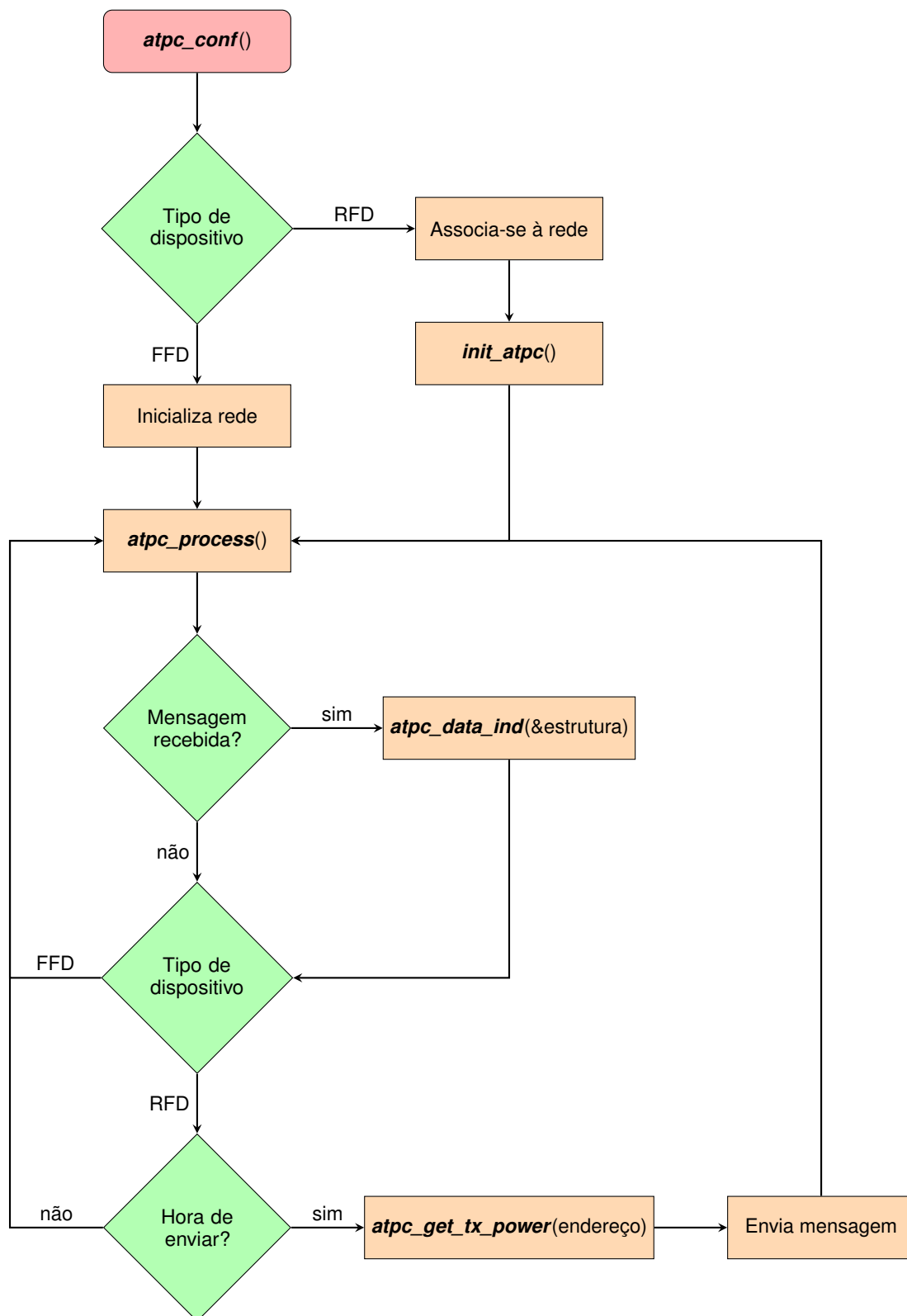
Figura 8 – Placa de desenvolvimento nRF52840 Dongle



Fonte: Nordic Semiconductor (2021).

A aplicação, que se encontra no Apêndice A, é baseada em exemplos de uso do OpenThread, e resulta em *firmwares* para FFD e RFD. Nela, o FFD cria uma rede LR-WPAN, à qual o RFD é conectado. Ocorre então o envio de mensagens do RFD para o FFD periodicamente. Estas mensagens possuem tamanho fixo e são transmitidas utilizando a potência adaptativa determinada pelo algoritmo. Após um segundo do envio de cada mensagem com potência adaptativa, esta é reenviada utilizando potência de transmissão fixa, para fins de comparação. O FFD registra o RSSI e um identificador das mensagens recebidas, enquanto o RFD armazena a potência de transmissão e identificador das mensagens enviadas, bem como informações a respeito das notificações recebidas. A Figura 9 ilustra um fluxograma simplificado para esta aplicação.

Figura 9 – Fluxograma da aplicação



Fonte: Elaboração própria (2021).

A Figura 10 apresenta o registro da aplicação durante um dos testes realizados. Nela, as medidas do FFD estão apresentadas à esquerda, e as informações do RFD à direita.

Figura 10 – Registro da aplicação de testes

```
-85 0
-75 0
-85 1
-74 1
-80 2
-75 2
-84 3
-73 3
-82 4
-73 4
-83 5
-73 5
-83 6
-74 6
-84 7
-74 7
-90 8
-81 8
-82 9
-82 9

[atpc] Control model of 68: sum_tp_squared = 960
, sum_tp = -48, sum_ri = -616, sum_tp_ri: 4292,
a = 7/8, b = -573/8
ri = ([-86., -88., -84., -79., -75., -73., -68.,
-63.])
-12 0
-12 1
-12 2
-12 3
-12 4
-12 5
-12 6
-12 7
-12 8
[atpc] Model updated for 68: b = -629/8
-4 9
-4 10
[atpc] Model updated for 68: b = -669/8
0 11
0 12
0 13
[atpc] Model updated for 68: b = -629/8
-4 14
-4 15
-4 16
[atpc] Model updated for 68: b = -581/8
-12 17
-12 18
-12 19
[atpc] Model updated for 68: b = -621/8
```

Fonte: Elaboração própria (2021).

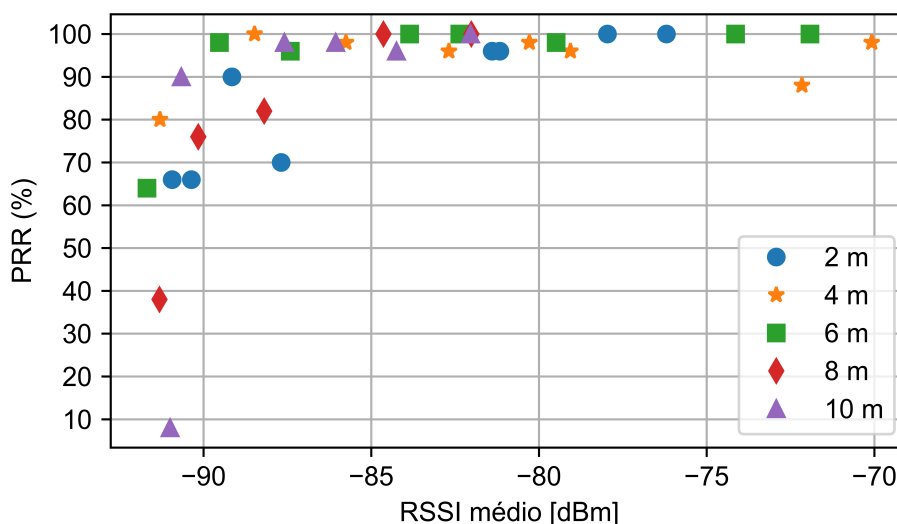
4.3 Definição dos parâmetros de qualidade de conexão

Um procedimento foi adotado para identificar a relação entre PRR e RSSI. Este envolve o envio de 50 pacotes do RFD para o FFD em cada nível de potência de transmissão disponível durante um curto período de tempo, variando a distância entre ambos e registrando os valores de RSSI observados e número de pacotes recebidos. Realiza-se a média dos valores de RSSI para cada nível de potência de transmissão, relacionando-a ao respectivo PRR.

Inicialmente, executou-se este procedimento em um parque, resultando nas medidas apresentadas na Figura 11. Com base nesses dados, definiu-se um limite inferior para a margem de tolerância de RSSI de -87 dBm, com o objetivo de manter o PRR acima de 95 %. O limite superior foi definido em -79 dBm, levando em consideração a maior diferença na média de valores de

RSSI apresentados entre duas potências de transmissão. Assim, estabeleceu-se um valor-alvo de RSSI de -83 dBm.

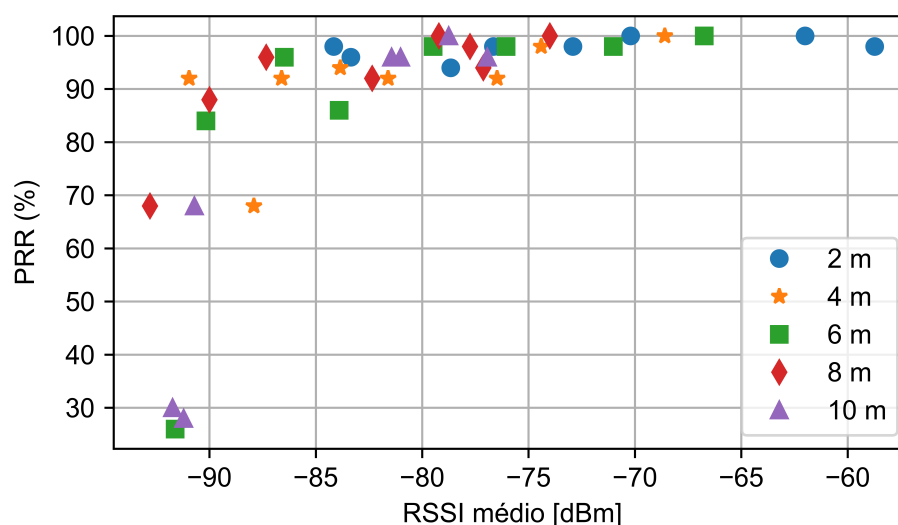
Figura 11 – Relação entre PRR e RSSI médio a diferentes distâncias em um parque



Fonte: Elaboração própria (2021).

Entretanto, observou-se que as comunicações sem fio do computador conectado ao FFD não haviam sido devidamente desabilitadas durante o experimento inicial. Este pode ser o motivo pelo qual alguns valores de RSSI médio observados nas distâncias de 4 m e 6 m foram maiores que os de 2 m. Sendo assim, repetiu-se o teste em um quintal, localizado em ambiente residencial, após corrigido este erro. Assim, obteve-se os resultados da Figura 12.

Figura 12 – Relação entre PRR e RSSI médio a diferentes distâncias em um quintal



Fonte: Elaboração própria (2021).

Comparado ao teste realizado no parque, observa-se uma variação menor no PRR. Ainda, percebe-se que o limite de RSSI inferior determinado para o parque garante um PRR de aproximadamente 90 % no quintal, e que o limite superior e RSSI alvo continuam compatíveis neste caso. Logo, optou-se por manter os parâmetros determinados anteriormente.

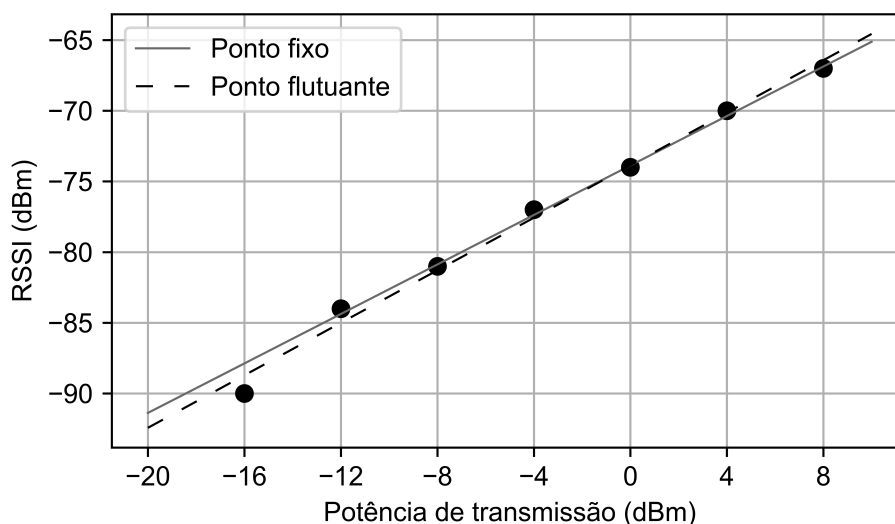
4.4 Análise durante inicialização e operação do algoritmo

No mesmo quintal do procedimento da Seção 4.3, realizou-se um teste de inicialização e operação contínua do algoritmo ATPC. Foram definidos inicialmente uma distância de 6 m entre os transceptores, e um intervalo de 15 minutos entre mensagens enviadas do RFD para o FFD. Este intervalo foi definido com base em aplicações reais, onde sensores transmitem leituras com longos intervalos entre aquisições.

A Figura 13 demonstra a inicialização do algoritmo realizada no RFD. São apresentadas as medidas de RSSI realizadas no FFD conforme o recebimento dos *beacons* transmitidos pelo RFD em cada nível de potência. Nota-se a ausência do *beacon* transmitido em -20 dBm, causada pelo mesmo,

ou por sua resposta, não terem sido recebidos. Essa situação é prevista na implementação desenvolvida, que automaticamente ignora o *beacon* perdido.

Figura 13 – Inicialização com distância de 6 m entre transceptores



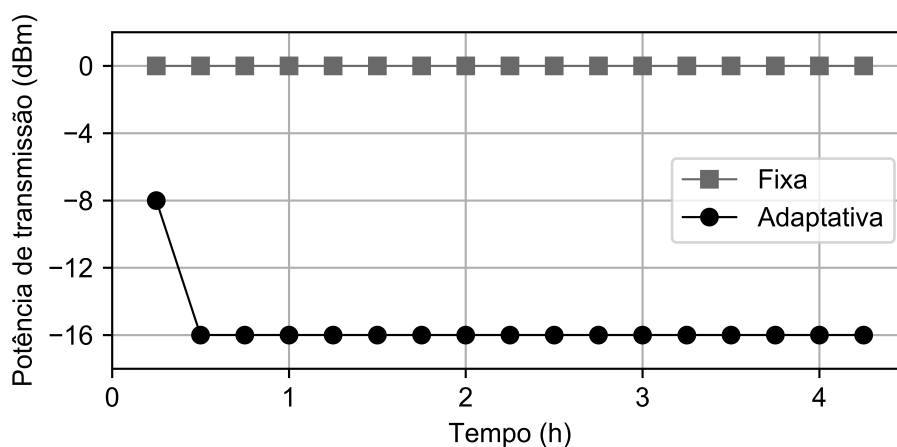
Fonte: Elaboração própria (2021).

Também percebe-se uma pequena diferença entre a parametrização realizada pela plataforma, representada em linha contínua, e uma parametrização realizada posteriormente em uma simulação de computador, representada em linha tracejada. Esta discrepância ocorre em decorrência da resolução de 0.125 utilizada nos cálculos de ponto fixo realizados pelo *firmware* utilizando formato de número Q.3, em contraste com a resolução de 1×10^{-15} utilizada em simulação. A implementação utiliza aritmética de ponto fixo, pois foi desenvolvida com objetivo de ser compatível com diferentes plataformas. Por este motivo a unidade de ponto flutuante do nRF52840 não foi utilizada pelo módulo de ATPC.

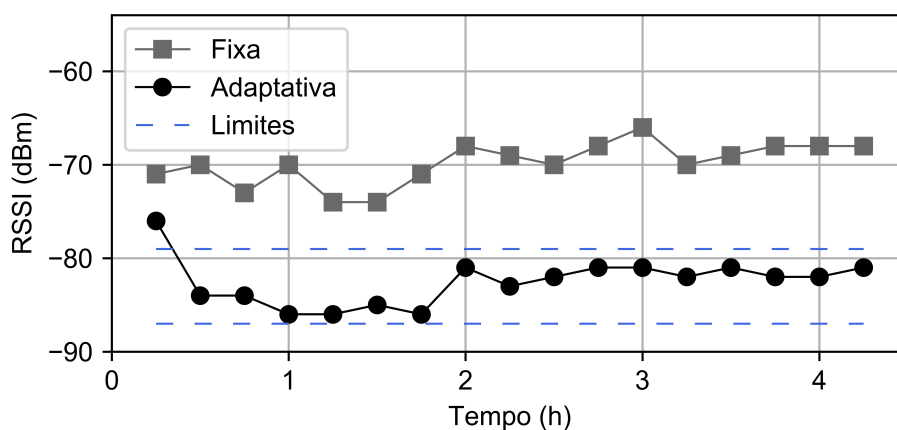
O comportamento durante a operação do algoritmo pode ser observado na Figura 14. Nesta etapa, o RFD envia mensagens utilizando potência de transmissão fixa e adaptativa, com um intervalo de 1 segundo, a cada 15 minutos, conforme Figura 14a. Os valores de RSSI das mensagens recebidas pelo FFD podem ser visualizados na Figura 14b, onde as linhas tracejadas azuis destacam os limites de RSSI estipulados durante a configuração do algoritmo.

Figura 14 – Operação com distância de 6 m entre transceptores

(a) Potência de transmissão utilizada pelo RFD



(b) Respectivos valores de RSSI observados no FFD



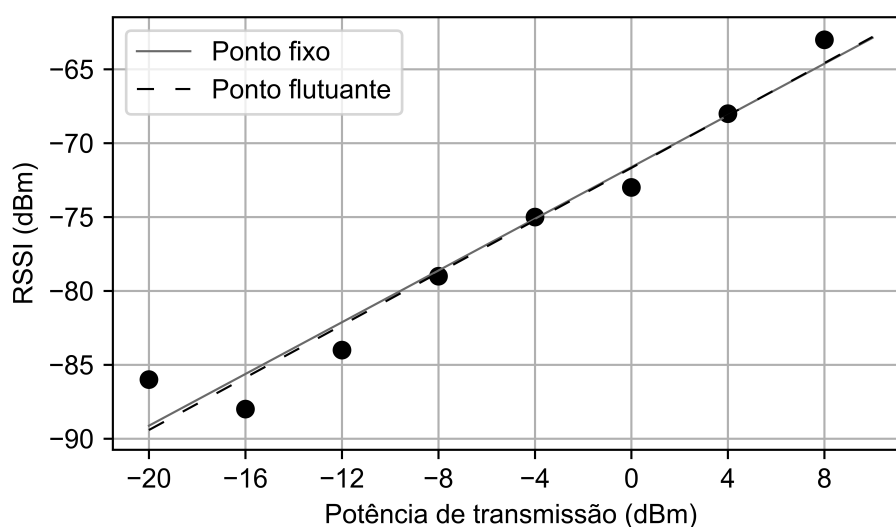
Fonte: Elaboração própria (2021).

Percebe-se que o RFD utilizou uma potência de -8 dBm em sua primeira mensagem, resultando em um RSSI acima do limite superior. Logo, foi enviada uma notificação para a atualização do modelo, causando a utilização da potência de -16 dBm nas mensagens seguintes, que foram recebidas com RSSI dentro da margem esperada. Destaca-se que obteve-se um PRR de 100 % durante todo o teste.

Contudo, esta primeira análise mostra-se insatisfatória para uma verificação mais detalhada da operação. Isto pois não houveram perturbações significativas no meio físico que provocassem suficientes variações na potência de transmissão utilizada. Sendo assim, efetuou-se um novo teste, com distância variável entre os transceptores. Assim possibilitando uma melhor avaliação do comportamento do algoritmo em um tempo curto, além de permitir uma nova análise de seu funcionamento, operando com distância dinâmica.

A inicialização, exibida na Figura 15, foi realizada com 4 m entre as placas. Neste caso, todos os *beacons* de inicialização foram recebidos pelo FFD e seus valores informados ao RFD. Além disso, houve uma discrepância menor entre a parametrização da plataforma e a simulada.

Figura 15 – Inicialização com distância de 4 m entre transceptores



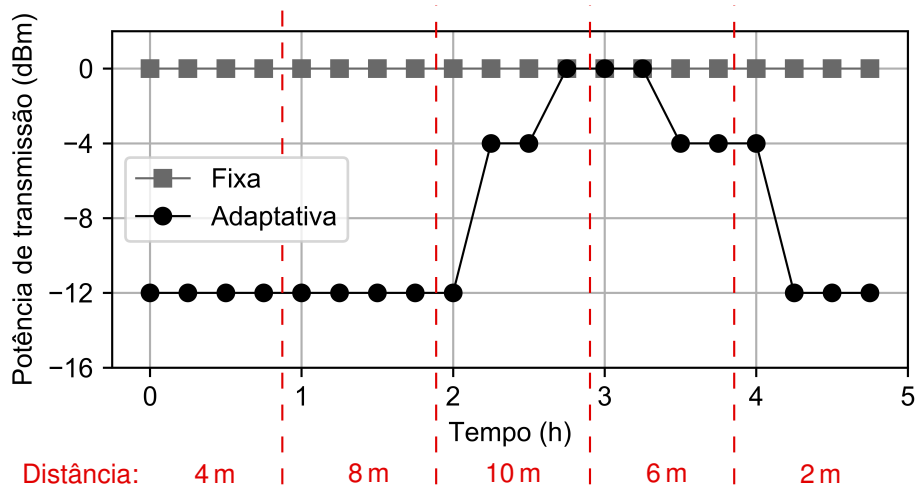
Fonte: Elaboração própria (2021).

A Figura 16 apresenta os resultados durante a operação com variação na distância. As linhas verticais tracejadas em vermelho delimitam os intervalos de tempo em que as placas estiveram separadas por cada distância. Na Figura 16a constata-se que o algoritmo adaptatou a potência de transmissão entre -16 dBm e 0 dBm. Isto ocorreu conforme o recebimento de mensagens cujo

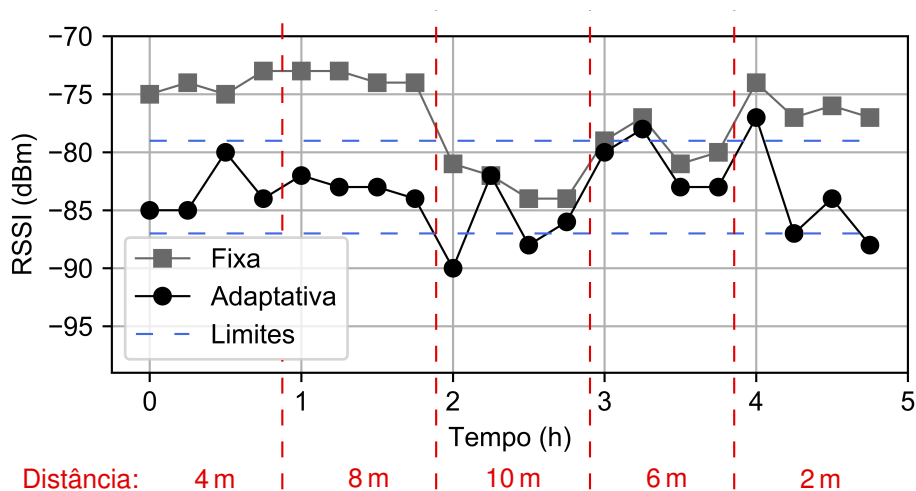
RSSI estava fora dos limites destacados na Figura 16b pelas linhas horizontais tracejadas em azul.

Figura 16 – Operação com distância variável entre transceptores

(a) Potência de transmissão utilizada pelo RFD



(b) Respectivos valores de RSSI observados no FFD



Fonte: Elaboração própria (2021).

Assim como no experimento anterior, registrou-se um PRR de 100 %. Em média, a potência adaptativa utilizada foi menor que a fixa, sem apresentar redução na qualidade da conexão. Com isso, verificou-se que a implementação do ATPC desenvolvida funcionou conforme esperado. Ainda, mostrou-se que

é possível aplicar esta técnica com bons resultados em uma aplicação com distância variável.

4.5 Avaliação do desempenho energético

As transmissões realizadas pelo RFD durante os experimentos da Seção 4.4 tiveram tamanho fixo. Considerando a energia de transmissão por *byte* apresentada na Tabela 3, e a potência utilizada durante a transmissão de cada mensagem, verificou-se o desempenho energético da aplicação. Com o uso da potência adaptativa, houve uma redução de aproximadamente 43 % na energia utilizada pelo RFD para transmitir mensagens no teste com distância fixa de 6 m, e de cerca de 30 % no teste com distância variável. Entretanto, ressalta-se que esta análise não leva em consideração a energia utilizada pelo RFD durante a inicialização, nem a consumida durante os demais estágios de operação do SoC.

5 CONSIDERAÇÕES FINAIS

Após revisão bibliográfica e seleção de uma técnica para controle da potência de transmissão, elaborou-se uma implementação genérica e de código livre, baseada no algoritmo ATPC publicado por Lin et al. (2006). Uma aplicação utilizando o protocolo de rede OpenThread foi desenvolvida para avaliar o funcionamento da estratégia selecionada. Seu desempenho na qualidade de conexão e eficiência energética foi analisado para o caso específico de um SoC nRF52840 desempenhando a função de RFD.

O funcionamento da implementação desenvolvida foi confirmado por meio dos experimentos realizados. Mostrou-se que, além das aplicações com distância fixa, é possível aplicar o algoritmo em situações onde a distância varia. Nesta situação, a adaptabilidade do controle da potência de transmissão é mais evidente. Ainda, os resultados utilizando a potência adaptativa foram comparados com os obtidos utilizando uma potência de transmissão fixa. Assim, verificou-se que o uso do ATPC possibilitou maior eficiência energética na transmissão dos dados de um RFD, mantendo a qualidade na conexão. Portanto, cumpriram-se os objetivos geral e específicos previamente estabelecidos, esclarecendo o problema definido inicialmente.

Uma das maiores contribuições deste trabalho é a publicação da implementação desenvolvida utilizando licença de *software* livre. Contudo, nem todas as condições de funcionamento do algoritmo foram avaliadas, mas ficam como sugestão para trabalhos futuros. Experimentos mais longos, com outras distâncias entre dispositivos e períodos diferentes de envio de mensagens, podem ajudar a compreender melhor os limites de funcionamento. Também podem ser efetuados testes com mais dispositivos conectados, e a análise em diferentes topologias de rede.

Outra recomendação para demais trabalhos, é a realização de uma investigação mais profunda a respeito da definição dos parâmetros de qualidade. Seria interessante determinar as melhores técnicas para a escolha do valor-alvo de RSSI e de seus limites de tolerância. A quantidade de níveis de potência de transmissão disponíveis e o intervalo entre eles poderiam ser considerados na

determinação desses parâmetros. Um ajuste automatizado dos valores pode ser importante para a aplicação do algoritmo em produtos de IoT aplicados em diferentes cenários.

REFERÊNCIAS

- BRINI, O.; DESLANDES, D.; NABKI, F. *A System-Level Methodology for the Design of Reliable Low-Power Wireless Sensor Networks*. **Sensors**, Basel, Suíça, v. 19, n. 8, p. 1800, 2019. Disponível em: <<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6514601/>>. Acesso em: 12 jul. 2020. Citado na página 13.
- CHAPRA, S. ***Applied Numerical Methods with MATLAB for Engineers and Scientists***. 3. ed. Nova Iorque, EUA: McGrall-Hill, 2012. Citado na página 21.
- CIRIO, R. S. ***ATPC: Adaptive Transmission Power Control***. 2021. Disponível em: <<https://github.com/ricardocirio/atpc>>. Acesso em 15 abr. 2021. Citado na página 28.
- CUI, X.; ZHANG, X.; SHANG, Y. *Energy-saving Strategies of Wireless Sensor Networks*. In: **2007 International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications**. Hangzhou, China: IEEE, 2007. p. 178–181. DOI 10.1109/MAPE.2007.4393575. Citado na página 18.
- GUBBI, J. et al. *Internet of Things (IoT): A vision, architectural elements, and future directions*. **Future Generation Computer Systems**, v. 29, n. 7, p. 1645–1660, 2013. Citado na página 13.
- GUTIERREZ, J. A. *On the use of IEEE 802.15.4 to enable wireless sensor networks in building automation*. In: **2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications**. [S.l.: s.n.], 2004. v. 3, p. 1865–1869. Citado na página 13.
- HART, D. ***Power electronics***. New York: McGraw-Hill, 2011. ISBN 978-0-07-338067-4. Citado 2 vezes nas páginas 24 e 25.
- IEEE COMPUTER SOCIETY. ***IEEE Standard for Low-Rate Wireless Networks***. [S.l.], 2020. ISBN 978-1-5044-6689-9. Citado 2 vezes nas páginas 16 e 17.
- JEONG, J.; CULLER, D.; OH, J. *Empirical Analysis of Transmission Power Control Algorithms for Wireless Sensor Networks*. In: **2007 Fourth International Conference on Networked Sensing Systems**. Braunschweig, Germany: IEEE, 2007. p. 27–34. DOI 10.1109/INSS.2007.4297383. Citado 3 vezes nas páginas 18, 19 e 20.

LAVRATTI, F. et al. *Towards a transmission power self-optimization in reliable Wireless Sensor Networks*. In: **2010 11th Latin American Test Workshop**. Punta del Este, Uruguay: IEEE, 2010. p. 1–3. DOI 10.1109/LATW.2010.5550356. Citado na página 19.

LEE, I.; LEE, K. *The Internet of Things (IoT): Applications, investments, and challenges for enterprises*. **Business Horizons**, v. 58, n. 4, p. 431–440, 2015. ISSN 0007-6813. Citado na página 13.

LIN, S. et al. *ATPC: Adaptive Transmission Power Control for Wireless Sensor Networks*. **SenSys**, Boulder, EUA, 2006. Disponível em: <<https://www.cs.virginia.edu/~stankovic/psfiles/ATPC.pdf>>. Acesso em: 20 jul. 2020. Citado 8 vezes nas páginas 15, 19, 20, 21, 22, 23, 26 e 41.

NORDIC SEMICONDUCTOR. **nRF52840 Product Specification v1.1**. 2019. Disponível em: <https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf>. Acesso em 3 mar. 2021. Citado na página 23.

NORDIC SEMICONDUCTOR. **nRF52840 Dongle**. 2021. Disponível em: <<https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/nRF52840-Dongle-product-brief.pdf>>. Acesso em 3 mar. 2021. Citado na página 31.

OPENTHREAD. **What is Thread?** 2020. Disponível em: <<https://openthread.io/guides/thread-primer>>. Acesso em 3 mar. 2021. Citado na página 18.

PANICHPAPIBOON, S.; FERRARI, G.; Tonguz, O. K. *Optimal Transmit Power in Wireless Sensor Networks*. **IEEE Transactions on Mobile Computing**, v. 5, n. 10, p. 1432–1447, 2006. DOI 10.1109/TMC.2006.155. Citado na página 19.

PRIYESH, P. P.; BHARTI, S. K. *Dynamic transmission power control in wireless sensor networks using P-I-D feedback control technique*. In: **2017 9th International Conference on Communication Systems and Networks (COMSNETS)**. Bangalor, Índia: IEEE, 2017. p. 306–313. DOI 10.1109/COMSNETS.2017.7945391. Citado 2 vezes nas páginas 19 e 20.

SANTOS, E. L. F. A IEEE 802.15.4 como plataforma de comunicação de dados. **Revista Ilha Digital**, v. 4, p. 97–105, 2013. ISSN 2177-2649. Citado na página 13.

TANENBAUM, A. S.; WETHERALL, D. J. **Computer Networks**. 5. ed. [S.l.]: Prentice Hall, 2011. Citado 2 vezes nas páginas 15 e 16.

THREAD GROUP. ***The Value of Low Power***. 2018. Disponível em: <https://www.threadgroup.org/Portals/0/documents/support/TheValueofLowPowerWhitepaper_2454_2.pdf>. Acesso em 30 dez. 2020. Citado 4 vezes nas páginas 16, 17, 18 e 24.

THREAD GROUP. ***Thread Certified Products***. 2021. Disponível em: <<https://www.threadgroup.org/What-is-Thread/Thread-Benefits#certifiedproducts>>. Acesso em 3 mar. 2021. Citado na página 16.

APÊNDICE A – APLICAÇÃO DE TESTES

Código A.1 – main.c

```

1 /*
2  * Copyright (c) 2021, Ricardo Cirio (INSTITUTO FEDERAL DE SANTA
   CATARINA).
3  * All rights reserved.
4  *
5  * This code was adapted from OpenThread's CLI example and from
   OpenThread's
6  * codelabs. This software is not endorsed in any way by the OpenThread
   project
7  * or by its authors. Original copyright notice as follows:
8  *
9  * Copyright (c) 2016, The OpenThread Authors.
10 * All rights reserved.
11 *
12 * Redistribution and use in source and binary forms, with or without
13 * modification, are permitted provided that the following conditions
   are met:
14 * 1. Redistributions of source code must retain the above copyright
15 *    notice, this list of conditions and the following disclaimer.
16 * 2. Redistributions in binary form must reproduce the above copyright
17 *    notice, this list of conditions and the following disclaimer in
   the
18 *    documentation and/or other materials provided with the
   distribution.
19 * 3. Neither the name of the copyright holder nor the
20 *    names of its contributors may be used to endorse or promote
   products
21 *    derived from this software without specific prior written
   permission.
22 *
23 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "
   AS IS"
24 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
   THE
25 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
   PURPOSE
26 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
   CONTRIBUTORS BE
27 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
28 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
29 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
   BUSINESS
30 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER

```

```

    IN
31  *   CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
    OTHERWISE)
32  *   ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
    OF THE
33  *   POSSIBILITY OF SUCH DAMAGE.
34  */
35
36 #include <assert.h>
37 #include <openthread-core-config.h>
38 #include <openthread/config.h>
39 #include <openthread/cli.h>
40 #include <openthread/diag.h>
41 #include <openthread/tasklet.h>
42 #include <openthread/platform/logging.h>
43 #include <openthread/platform/time.h>
44 #include <openthread/instance.h>
45 #include <openthread/thread.h>
46 #include <openthread/thread_ftd.h>
47 #include <string.h>
48 #include <openthread/message.h>
49 #include <openthread/udp.h>
50 #include <openthread/dataset_ftd.h>
51 #include <openthread/link_raw.h>
52
53 #include <atpc.h>
54 #include <list.h>
55
56 #include "openthread-system.h"
57 #include "cli/cli_config.h"
58 #include "utils/code_utils.h"
59
60 #if OPENTHREAD_EXAMPLES_SIMULATION
61 #include <setjmp.h>
62 #include <unistd.h>
63
64 jmp_buf gResetJump;
65
66 void __gcov_flush();
67 #endif
68
69 #ifndef OPENTHREAD_ENABLE_COVERAGE
70 #define OPENTHREAD_ENABLE_COVERAGE 0
71 #endif
72
73 #define UDP_PORT 1212
74 #define MULTICAST_ADDR 0xFFFF

```

```

75 #define DATA_REPORT_PERIOD    900*1000000
76 #define BEACON_TIMEOUT        3*1000000
77 #define DEFAULT_TX_POWER      0
78 #define RSSI_SETPOINT         -83
79 #define RSSI_LOWER_THRESH     -87
80 #define RSSI_UPPER_THRESH     -79
81
82 typedef enum {
83     APP_MSG,
84     ATPC_MSG,
85 } app_msg_type_t;
86
87 /*****
88  Local function prototypes
89 *****/
90
91 static void setNetworkConfiguration(otInstance *aInstance);
92 static void handleNetifStateChanged(uint32_t aFlags, void *aContext);
93 static void initUdp(otInstance *aInstance);
94 static void sendUdp(otInstance *aInstance);
95 static void handleUdpReceive(void *aContext, otMessage *aMessage,
96                             const otMessageInfo *aMessageInfo);
97 static void data_report_process(otInstance *aInstance);
98 static bool atpc_send_msg(uint16_t dest_addr, int8_t tx_power,
99                          uint8_t *data, uint16_t len);
100 static void atpc_state_change(state_t state);
101 static void *list_new_wrapper(void);
102 static void list_add_wrapper(void *list, void *data);
103 static void list_remove_wrapper(void *list, void *data);
104 static void *list_next_wrapper(void *list);
105 static void atpc_log(const char *aFormat, ...);
106
107 /*****
108  Local variables
109 *****/
110
111 static otInstance *instance;
112
113 static otUdpSocket sUdpSocket;
114 static otIp6Address neighborIp6;
115
116 static const char UDP_PAYLOAD[] = "01 , IFSC.\n";
117 static const char UDP_MULTICAST_ADDR[] = "ff03::1";
118
119 static int8_t TX_POWER[] = {-20, -16, -12, -8, -4, 0, 4, 8};
120
121 static atpc_callbacks_t atpc_cbs = {.send_msg = atpc_send_msg,

```

```

122         .state_change = atpc_state_change,
123         .list_new = list_new_wrapper,
124         .list_add = list_add_wrapper,
125         .list_remove = list_remove_wrapper,
126         .list_next = list_next_wrapper,
127         .get_time = otPlatTimeGet,
128         .log = atpc_log};
129 static state_t atpc_state = STATE_IDLE;
130
131 /*****
132  Public Functions
133  *****/
134
135 #if OPENTHREAD_CONFIG_MULTIPLE_INSTANCE_ENABLE
136 void *otPlatCalloc(size_t aNum, size_t aSize) {
137     return calloc(aNum, aSize);
138 }
139
140 void otPlatFree(void *aPtr) {
141     free(aPtr);
142 }
143 #endif
144
145 void otTaskletsSignalPending(otInstance *aInstance) {
146     OT_UNUSED_VARIABLE(aInstance);
147 }
148
149 int main(int argc, char *argv[])
150 {
151     #if OPENTHREAD_EXAMPLES_SIMULATION
152         if (setjmp(gResetJump)) {
153             alarm(0);
154         }
155     #if OPENTHREAD_ENABLE_COVERAGE
156         __gcov_flush();
157     #endif
158     execvp(argv[0], argv);
159 }
160 #endif
161
162 #if OPENTHREAD_CONFIG_MULTIPLE_INSTANCE_ENABLE
163     size_t otInstanceBufferLength = 0;
164     uint8_t *otInstanceBuffer = NULL;
165 #endif
166 pseudo_reset:
167
168     otSysInit(argc, argv);

```

```

169
170 #if OPENTHREAD_CONFIG_MULTIPLE_INSTANCE_ENABLE
171     /* Call to query the buffer size */
172     (void)otInstanceInit(NULL, &otInstanceBufferLength);
173
174     /* Call to allocate the buffer */
175     otInstanceBuffer = (uint8_t *)malloc(otInstanceBufferLength);
176     assert(otInstanceBuffer);
177
178     /* Initialize OpenThread with the buffer */
179     instance = otInstanceInit(otInstanceBuffer, &otInstanceBufferLength);
180 #else
181     instance = otInstanceInitSingle();
182 #endif
183     assert(instance);
184
185 #if OPENTHREAD_CONFIG_CLI_TRANSPORT == OT_CLI_TRANSPORT_UART
186     otCliUartInit(instance);
187 #endif
188
189     /* Register Thread state change handler */
190     otSetStateChangedCallback(instance, handleNetifStateChanged, instance
191 );
192
193     /* Override default network credentials */
194     setNetworkConfiguration(instance);
195
196     /* Configure ATPC */
197     atpc_conf(&atpc_cbs,
198             DEFAULT_TX_POWER, TX_POWER, sizeof(TX_POWER)/sizeof(int8_t),
199             RSSI_SETPOINT, RSSI_UPPER_THRESH, RSSI_LOWER_THRESH,
200             MULTICAST_ADDR, BEACON_TIMEOUT);
201
202     /* Start the Thread network interface (CLI cmd > ifconfig up) */
203     otIp6SetEnabled(instance, true);
204
205     /* Start the Thread stack (CLI cmd > thread start) */
206     otThreadSetEnabled(instance, true);
207
208     initUdp(instance);
209
210     while (!otSysPseudoResetWasRequested()) {
211         otTaskletsProcess(instance);
212         otSysProcessDrivers(instance);
213         otSysButtonProcess(instance);
214         atpc_process();
215         data_report_process(instance);

```

```

215     }
216
217     otInstanceFinalize(instance);
218 #if OPENTHREAD_CONFIG_MULTIPLE_INSTANCE_ENABLE
219     free(otInstanceBuffer);
220 #endif
221
222     goto pseudo_reset;
223
224     return 0;
225 }
226
227 /*
228  * Provide, if required an "otPlatLog()" function
229  */
230 #if OPENTHREAD_CONFIG_LOG_OUTPUT == OPENTHREAD_CONFIG_LOG_OUTPUT_APP
231 void otPlatLog(otLogLevel aLogLevel, otLogRegion aLogRegion,
232               const char *aFormat, ...) {
233     va_list ap;
234     va_start(ap, aFormat);
235     otCliPlatLogv(aLogLevel, aLogRegion, aFormat, ap);
236     va_end(ap);
237 }
238
239 void otPlatLogLine(otLogLevel aLogLevel, otLogRegion aLogRegion,
240                   const char *aLogLine) {
241     otCliPlatLogLine(aLogLevel, aLogRegion, aLogLine);
242 }
243 #endif
244
245 /*****
246  Local Functions
247  *****/
248
249 /**
250  * Override default network settings, such as panid, so the devices can
251  join a
252 network
253 */
254 static void setNetworkConfiguration(otInstance *aInstance) {
255     static char aNetworkName[] = "ATPC_example";
256     otOperationalDataset aDataset;
257     otLinkModeConfig aConfig;
258
259     memset(&aDataset, 0, sizeof(otOperationalDataset));
260
261     /*

```

```

261     * Fields that can be configured in otOperationDataset to override
    defaults:
262     *     Network Name, Mesh Local Prefix, Extended PAN ID, PAN ID,
263     *     Delay Timer, Channel, Channel Mask Page 0, Network Master Key,
    PSKc,
264     *     Security Policy
    */
265
266     aDataset.mActiveTimestamp = 1;
267     aDataset.mComponents.mIsActiveTimestampPresent = true;
268
269     /* Set Channel to 15 */
270     aDataset.mChannel = 15;
271     aDataset.mComponents.mIsChannelPresent = true;
272
273     /* Set Pan ID to 2222 */
274     aDataset.mPanId = (otPanId)0x2222;
275     aDataset.mComponents.mIsPanIdPresent = true;
276
277     /* Set Extended Pan ID to 1F5C001F5C001F5C */
278     uint8_t extPanId[OT_EXT_PAN_ID_SIZE] = {0x1F, 0x5C, 0, 0, 0, 0, 0x1F,
    0x5C};
279     memcpy(aDataset.mExtendedPanId.m8, extPanId,
280            sizeof(aDataset.mExtendedPanId));
281     aDataset.mComponents.mIsExtendedPanIdPresent = true;
282
283     /* Set master key to 1F5C0000001F5C0000001F5C */
284     uint8_t key[OT_MASTER_KEY_SIZE] = {0x1F, 0x5C, 0, 0, 0, 0x1F, 0x5C,
    0, 0, 0,
285                                         0x1F, 0x5C};
286     memcpy(aDataset.mMasterKey.m8, key, sizeof(aDataset.mMasterKey));
287     aDataset.mComponents.mIsMasterKeyPresent = true;
288
289     /* Set Network Name */
290     size_t length = strlen(aNetworkName);
291     assert(length <= OT_NETWORK_NAME_MAX_SIZE);
292     memcpy(aDataset.mNetworkName.m8, aNetworkName, length);
293     aDataset.mComponents.mIsNetworkNamePresent = true;
294
295     otDatasetSetActive(aInstance, &aDataset);
296
297 #if OPENTHREAD_FTD
298     /* Set the router selection jitter to override the 2 minute default.
299        CLI cmd > routerselectionjitter 20
300        Warning: For demo purposes only - not to be used in a real product
301        */
302     uint8_t jitterValue = 20;
303     otThreadSetRouterSelectionJitter(aInstance, jitterValue);

```

```

303     OT_UNUSED_VARIABLE(aConfig);
304 #endif
305
306 #if OPENTHREAD_MTD
307     aConfig.mRxOnWhenIdle = 0;
308     aConfig.mDeviceType = 0;
309     aConfig.mNetworkData = 0;
310     otThreadSetLinkMode(aInstance, aConfig);
311     otLinkSetPollPeriod(aInstance, 500);
312     otLinkSetMaxFrameRetriesDirect(aInstance, 0);
313 #endif
314 }
315
316 static void handleNetifStateChanged(uint32_t aFlags, void *aContext) {
317     if ((aFlags & OT_CHANGED_THREAD_ROLE) != 0) {
318         if (otThreadGetDeviceRole(aContext) == OT_DEVICE_ROLE_CHILD) {
319             otThreadGetLeaderRloc(aContext, &neighborIp6);
320             /* dummy msg for parent to register child's local addr */
321             sendUdp(aContext);
322             /* Register parent within ATPC and initialize beacons */
323             if (atpc_register_neighbor(neighborIp6.mFields.m16[7]) ==
324
325                 1) {
326                 atpc_init();
327             }
328         }
329     }
330 }
331
332 /**
333  * Initialize UDP socket
334  */
335 static void initUdp(otInstance *aInstance) {
336     otSockAddr listenSockAddr;
337
338     memset(&sUdpSocket, 0, sizeof(sUdpSocket));
339     memset(&listenSockAddr, 0, sizeof(listenSockAddr));
340
341     listenSockAddr.mPort = UDP_PORT;
342
343     otUdpOpen(aInstance, &sUdpSocket, handleUdpReceive, aInstance);
344     otUdpBind(aInstance, &sUdpSocket, &listenSockAddr);
345 }
346
347 /**
348  * Send a UDP datagram
349  */
350 static void sendUdp(otInstance *aInstance) {

```

```

349     otError      error = OT_ERROR_NONE;
350     otMessage *   message;
351     otMessageInfo messageInfo;
352
353     memset(&messageInfo, 0, sizeof(messageInfo));
354
355     messageInfo.mPeerAddr    = neighborIp6;
356     messageInfo.mPeerPort    = UDP_PORT;
357
358     message = otUdpNewMessage(aInstance, NULL);
359     otEXPECT_ACTION(message != NULL, error = OT_ERROR_NO_BUFS);
360
361     uint8_t messageType = APP_MSG;
362     #if OPENTHREAD_MTD
363     int8_t tx_power = DEFAULT_TX_POWER;
364     if(atpc_state == STATE_RUN) {
365         tx_power = atpc_get_tx_power(neighborIp6.mFields.m16[7]);
366     }
367     error = otPlatRadioSetTransmitPower(aInstance, tx_power);
368 #endif /* OPENTHREAD_MTD */
369     error = otMessageAppend(message, &messageType, sizeof(uint8_t));
370     otEXPECT(error == OT_ERROR_NONE);
371
372     error = otMessageAppend(message, UDP_PAYLOAD, sizeof(UDP_PAYLOAD));
373     otEXPECT(error == OT_ERROR_NONE);
374
375     error = otUdpSend(aInstance, &sUdpSocket, message, &messageInfo);
376
377     exit:
378     if (error != OT_ERROR_NONE && message != NULL)
379     {
380         otMessageFree(message);
381     }
382 }
383
384 /**
385  * Function to handle UDP datagrams received on the listening socket
386  */
387 static void handleUdpReceive(void *aContext, otMessage *aMessage,
388                             const otMessageInfo *aMessageInfo) {
389     OT_UNUSED_VARIABLE(aContext);
390
391     #if OPENTHREAD_FTD
392     static bool neighbor_status = false;
393     if(neighbor_status == false) {
394         neighborIp6 = aMessageInfo->mPeerAddr;
395         if( (neighborIp6.mFields.m16[4] == 0) &&

```

```

396         (neighborIp6.mFields.m16[5] == 0xff00) &&
397         (neighborIp6.mFields.m16[6] == 0xfe) ) {
398             /* Register child's RLOC addr */
399             atpc_register_neighbor(neighborIp6.mFields.m16[7]);
400             neighbor_status = true;
401         }
402     }
403 #endif
404
405     char buff[otMessageGetLength(aMessage)];
406     otMessageRead(aMessage, 0, buff, otMessageGetLength(aMessage));
407
408     atpc_data_ind_t data_ind;
409     atpc_msg_t msg;
410     data_ind.src_addr = neighborIp6.mFields.m16[7];
411     data_ind.rssi = ((otThreadLinkInfo *) (aMessageInfo->mLinkInfo))->mRss
;
412     switch(buff[0]) {
413         case APP_MSG:
414             msg.type = ATPC_QUALITY_MONITOR;
415             data_ind.data = &msg;
416             /* Display message */
417             otPlatLog(OT_LOG_LEVEL_DEBUG, OT_LOG_REGION_API,
418                     buff + sizeof(app_msg_type_t) + sizeof(uint16_t));
419             break;
420         case ATPC_MSG:
421             OT_UNUSED_VARIABLE(msg);
422             data_ind.data = (atpc_msg_t *) (buff + sizeof(app_msg_type_t))
;
423             break;
424         default:
425             OT_UNUSED_VARIABLE(msg);
426             break;
427     }
428     atpc_data_ind(&data_ind);
429 }
430
431 static void data_report_process(otInstance *aInstance) {
432     if((otThreadGetDeviceRole(aInstance) == OT_DEVICE_ROLE_CHILD) &&
433         (atpc_state == STATE_RUN)) {
434         static uint64_t timestamp = 0;
435         /* Send message using adaptive transmission power every
436            DATA_REPORT_PERIOD */
437         if(((otPlatTimeGet() - timestamp) >= DATA_REPORT_PERIOD)) {
438             sendUdp(aInstance);
439             timestamp = otPlatTimeGet();
440         }

```

```

441     }
442 }
443
444 static bool atpc_send_msg(uint16_t dest_addr, int8_t tx_power,
445                          uint8_t *data, uint16_t len) {
446     otError      error = OT_ERROR_NONE;
447     otMessage *   message;
448     otMessageInfo messageInfo;
449     uint8_t messageType = ATPC_MSG;
450     otIp6Address  destinationAddr;
451
452     memset(&messageInfo, 0, sizeof(messageInfo));
453
454     if(dest_addr == MULTICAST_ADDR) {
455         otIp6AddressFromString(UDP_MULTICAST_ADDR, &destinationAddr);
456     } else {
457         otThreadGetLeaderRloc(instance, &destinationAddr);
458         destinationAddr.mFields.m16[7] = dest_addr;
459     }
460     messageInfo.mPeerAddr = destinationAddr;
461     messageInfo.mPeerPort = UDP_PORT;
462
463     message = otUdpNewMessage(instance, NULL);
464     otEXPECT_ACTION(message != NULL, error = OT_ERROR_NO_BUFS);
465
466     error = otMessageAppend(message, &messageType, sizeof(uint8_t));
467     otEXPECT(error == OT_ERROR_NONE);
468
469     error = otMessageAppend(message, data, len);
470     otEXPECT(error == OT_ERROR_NONE);
471
472     #if OPENTHREAD_MTD
473         error = otPlatRadioSetTransmitPower(instance, tx_power);
474     #else
475         OT_UNUSED_VARIABLE(tx_power);
476     #endif
477
478     error = otUdpSend(instance, &sUdpSocket, message, &messageInfo);
479
480 exit:
481     if (error != OT_ERROR_NONE && message != NULL)
482     {
483         otMessageFree(message);
484     }
485
486     if(error == OT_ERROR_NONE) {
487         return true;
488     }

```

```
488     }
489     return false;
490 }
491
492 static void atpc_state_change(state_t state) {
493     OT_UNUSED_VARIABLE(state);
494     atpc_state = state;
495     if(state == STATE_RUN) {
496         otLinkSetPollPeriod(instance, DATA_REPORT_PERIOD);
497     }
498 }
499
500 static void *list_new_wrapper(void) {
501     return (void *)list_new();
502 }
503
504 static void list_add_wrapper(void *list, void *data) {
505     list_add((list_t *)list, data);
506 }
507
508 static void list_remove_wrapper(void *list, void *data) {
509     list_remove((list_t *)list, data);
510 }
511
512 static void *list_next_wrapper(void *list) {
513     return list_next((list_t *)list);
514 }
515
516 static void atpc_log(const char *aFormat, ...) {
517     va_list ap;
518     va_start(ap, aFormat);
519     otCliPlatLogv(OT_LOG_LEVEL_DEBUG, OT_LOG_REGION_API, aFormat, ap);
520     va_end(ap);
521 }
```