Radio Duty-Cycle Aloha Protocol: A ContikiMAC Redesign for Tmote Sky Simulated in Cooja

Jonathan K. dos Santos Silva, Paulo F. Candido Barbosa, and Renato M. de Moraes, *Senior Member, IEEE*

***Abstract*—In IoT networks, energy consumption is a critical factor, especially for battery-powered devices. To mitigate this, MAC protocols with Radio Duty Cycle (RDC) management are commonly used to improve wireless communication efficiency. This article presents a reformulation of the MAC sublayer for the Tmote Sky device, replacing the IEEE 802.15.4-based protocol with a pure Aloha implementation compatible with Contiki. Simulations using Cooja were conducted to evaluate the impact on energy consumption compared to the original MAC protocol. The results demonstrated that the proposed RDC Aloha protocol achieved significantly lower energy consumption compared to the native CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) protocol, with an average reduction of 40% and gains reaching up to 87.5% in best-case scenarios. These improvements are attributed to the simplicity of the Aloha protocol, the elimination of idle listening, and, most notably, the efficient duty cycling strategy introduced in this study. Although the Aloha-based design led to a reduced delivery rate due to its non-deterministic nature, the protocol proved to be a viable alternative for applications that prioritize energy efficiency and operate with low data volumes—typical characteristics of wireless sensor networks (WSNs).**

Link to graphical and video abstracts, and to code: https://latamt.ieeer9.org/index.php/transactions/article/view/9443

***Index Terms*—Aloha, Contiki, Cooja, Energy consumption, IoT, Tmote Sky, WSN.**

I. Introdução

UM nó IoT é um dispositivo capaz de se conectar à Internet, coletar e processar dados, e enviá-los para a rede. Esses dispositivos frequentemente enfrentam restrições, como tempo de vida e consumo de energia, especialmente em locais de difícil acesso. A maior parte do consumo de energia ocorre na comunicação entre os nós, tornando essencial a otimização do uso de energia e a minimização de perdas. O desenvolvimento de métodos de comunicação eficientes em redes IoT é um problema atual de grande interesse. [1].

Tecnologias de comunicação recentes, como o LoRaWAN (*Long Range Wide Area Network*), são estudadas por acadêmicos e indústrias devido ao seu grande alcance, baixa potência e baixo custo para aplicações IoT. O LoRaWAN oferece cobertura de longo alcance, de até vários quilômetros [2]. Ele é classificado em três classes: A, B e C. A classe A, que utiliza o protocolo MAC Aloha Puro, é o foco deste estudo, enquanto as classes B e C adicionam funcionalidades para sincronização e maior disponibilidade, com a classe C mantendo o dispositivo sempre ligado.

Em dispositivos IoT alimentados por bateria, além de métodos de comunicação eficientes, é essencial o uso de um sistema operacional eficaz para minimizar o consumo de energia. O Contiki, um sistema operacional de código aberto desenvolvido para dispositivos embarcados com restrições de energia e memória, é amplamente utilizado, especialmente em redes de sensores sem fio (WSNs). De acordo com [3], o termo ”Contiki AND OS” foi utilizado em mais de 2000 publicações revisadas por pares na base de dados Scopus.

Um aspecto importante na escolha de dispositivos para redes IoT é que eles devem ter características bem definidas e comportamento previsível. O Tmote Sky, um dispositivo de baixo custo e com código e hardware abertos, é amplamente utilizado como benchmark na literatura. Seu comportamento é bem conhecido, o que permite sua emulação em simuladores como o Cooja [4], [5], [6].

O presente trabalho propõe um protocolo MAC Aloha Puro com ciclos de carga (*duty-cycles*) para o Contiki no Tmote Sky, detalhando sua implementação e analisando os cenários nos quais essa abordagem supera o desempenho energético do protocolo MAC nativo IEEE 802.15.4 do Tmote Sky. Além disso, essa solução permite ao Contiki integrar outros protocolos baseados em Aloha, como o LoRaWAN, ampliando suas possibilidades de aplicação. Os códigos-fonte para reproduzir os experimentos estão disponíveis publicamente[1], e há uma patente registrada em [7].

O artigo está organizado da seguinte forma: a Seção II apresenta a fundamentação teórica, discutindo dispositivos IoT com limitações de recursos e protocolos MAC. A Seção III descreve a proposta e os ajustes feitos nos protocolos CSMA/CA e ContikiMAC para implementar o Aloha Puro com RDC. A Seção IV mostra os resultados dos experimentos e suas análises. Por fim, a Seção V apresenta as conclusões e possíveis direções para trabalhos futuros.

The associate editor coordinating the review of this manuscript and approving it for publication was Rodolfo Meneguette (*Corresponding author: Paulo Barbosa*).

This research was supported in part by Fundação de Amparo à Ciência e Tecnologia de Pernambuco (FACEPE).

J. K. D. S. Silva, and R. M. D. Moraes are with Centro de Informatica da Universidade Federal de Pernambuco, Av. Jorn. Aníbal Fernandes, s/n, Cidade Universitária, Recife, Brazil (e-mails: jkss@cin.ufpe.br, and brrenatomdm@cin.ufpe.br).

Paulo F. C. Barbosa is with Universidade de Pernambuco, Campus Petrolina, Rodovia BR 203, Km 2 s/n, Vila Eduardo, Petrolina, Brazil (e-mails: pfcb@cin.ufpe.br).

[1] https://github.com/pafil137/IEEELatin

II. Fundamentos e Considerações

As redes de sensores sem fio empregadas em sistemas IoT podem ser definidas como uma rede de dispositivos autônomos e autoconfiguráveis que possuem um transceptor de rádio, microcontrolador, memória, fonte de energia e um sensor (transdutor). Esse tipo de rede traz funcionalidades como: fácil implantação, confiabilidade, mobilidade, escalabilidade, baixo consumo de energia e configuração automática da rede [8].

A. Tipos de Dispositivos

Vários dispositivos podem ser utilizados para redes IoT a depender das restrições impostas no caso a ser implementado. Este trabalho abordará dispositivos IoT conhecidos como motes, que são placas que já possuem sensores integrados, estão prontos para uso desde sua compra e que são *open-source*, tanto em código fonte quanto quanto em hardware. Entretanto, para fins de uso em larga escala, e portanto com baixo custo, eles possuem limitações de recursos como baixo poder de processamento, pouca memória e são operados à bateria. Assim essas tecnologias demandam soluções que são desafiadoras. As Tabelas I e II, adaptadas de [9] e [5], descrevem a comparação de alguns dispositivos amplamente utilizados na literatura e possuem suporte no simulador Cooja.

TABELA I
Motes - Processador, Clock, RTC (*Real Time Clock*) e Sistemas Operacionais Compatíveis

Nome	**Modelo do Microprocessador**	**Clock**	**RTC**	**OS Compatíveis**
MicaZ	Atmel ATMega 128L	8 MHz	32768 kHz	TinyOS
Tmote Sky	MSP430F1611	8 MHz	32768 kHz	Contiki TinyOS
Zolertia Z1	MSP430F2617	16 MHz	Não especificado	TinyOS Contiki RIOT

TABELA II
Motes - Conectividade, Sensores, Consumo, RAM, ROM, Protocolo de Rede Compatível

Nome	**Exemplos de sensores**	**Corrente de consumo**	**RAM**	**ROM**	**Protocolo de rede**
MicaZ	Temperatura, Umidade, Luminosidade, Pressão, Acelerômetro	Transmitindo: 11 a 17.4 mA Recebendo: 19.7 mA Sleep: 15 µa	4 kB	512 kB	IPv6
Tmote Sky	Temperatura, Umidade, Luminosidade	Transmitindo: 8.5 a 17.4 mA Recebendo: 19.7 mA Sleep: 2.6 µa	10 kB	1024 kB (Flash Externa)	IPv6
Zolertia Z1	Acelerômetro, Temperatura	Transmitindo: 8.5 a 17.4 mA Recebendo: 19.7 mA Sleep: 2.6 µa	8 kB	92 kB	IPv6

B. Tmote Sky

Para este trabalho foi escolhido o dispositivo Tmote Sky por sua compatibilidade com protocolos baseado em IPv6, sistema operacional programável Contiki, compatível com simuladores de rede e interoperabilidade com outros dispositivos WSN/IoT, sugerido na literatura como sensor de fácil implementação de alterações no código do protocolo MAC e nas demais camadas, e amplamente utilizado [5], [6], [10]. Por outro lado, o Tmote Sky é bem mapeado no simulador Cooja, fazendo com que o porte para o dispositivo real ocorra de maneira esperada. Algumas de suas características são descritas a seguir e seus principais parâmetros de operação estão na Tabela III.

1) Na Camada Física: O Tmote Sky possui um rádio transceptor Chipcon CC2420 com taxa de transmissão de 250 kbps, frequência da portadora em 2.4 GHz e é compatível com o padrão IEEE 802.15.4. Cada octeto (um byte) é mapeado em dois símbolos de dados e então transmitidos pelo rádio [11].

2) Na Camada de Enlace: O Tmote Sky emprega nativamente o padrão IEEE 802.15.4 que foi desenvolvido para ser utilizado em redes sem fio de baixa potência (LP-WAN, do inglês *Low Power Wireless Area Networks*), com baixo custo, baixa velocidade de transmissão e código aberto [12].

Esse padrão emprega o CSMA/CA como protocolo MAC numa versão simplificada focada no consumo energético. Antes de transmitir, o dispositivo primeiro verifica se o canal está livre e se estiver, transmite os dados. Caso ocorra uma colisão, é feita uma espera de tempo aleatória, chamada de *backoff*, do pacote que foi enviado para que seja feita uma nova tentativa.

Por outro lado, o ContikiMAC gerencia o protocolo MAC CSMA/CA implementado no Contiki OS. Seu funcionamento consiste em enviar pacotes de dados empregando o CSMA/CA em um intervalo de tempo ou até que um pacote de confirmação (ACK, do inglês *acknowledgement*) seja retornado. Desse modo, o protocolo garante que o receptor, que emprega ciclos de carga (*duty-cycles*), possa detectar o pacote durante a verificação do meio e então obter a mensagem [10]. Para que o receptor consiga obter pacotes que estão sendo transmitidos na rede, uma certa temporização é necessária. Para isso o ContikiMAC tem bem definidos os tempos de transmissão e de checagem da rede.

A Fig. 1 indica as variáveis associadas ao tempo de envio/detecção de um pacote:

- ti - tempo entre pacotes;
- tr - tempo necessário para definir um nível de sinal (RSSI) mínimo, necessário para uma checagem do canal (CCA) estável;
- tc - tempo entre cada CCA;
- ta - tempo entre o recebimento do pacote e o envio do pacote ACK;
- td - tempo necessário para detectar um ACK do receptor.

O termo CCA (*Clear Channel Assessment*) é a checagem do canal em que um mecanismo avalia se o meio de comunicação

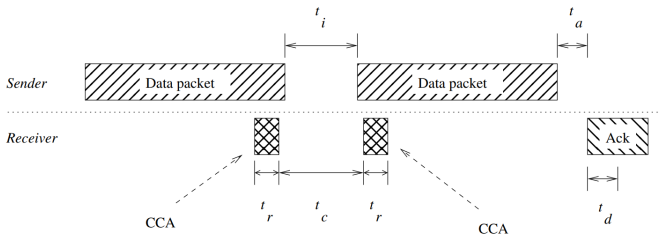

Fig. 1. Temporização do CCA ContikiMAC [10].

está livre ou não, medindo a intensidade do sinal captado pelo rádio (RSSI, do inglês *Received Signal Strength Indicator*). O ContikiMAC ainda define que o tamanho do pacote deve ser grande o suficiente para que o mesmo não aconteça entre dois CCAs. Já ts é o menor tempo de transmissão de um pacote:

$$ta + td < ti < tc < tc + 2tr < ts. \quad (1)$$

Do formato do quadro no IEEE 802.15.4 empregado no ContikiMAC é possível obter algumas informações como o valor de ta, que é definido como 12 símbolos. Um símbolo representa meio octeto, ou seja, 4 bits, onde a taxa de transmissão é de 250 kilobits por segundo (kbps), logo temos 4/250 milisegundos (ms), dando o valor de ta = 48/250 = 0.192 ms [10]. O padrão IEEE 802.15.4 pode detectar o recebimento de um pacote ACK após um preâmbulo de 4 bytes, onde temos 8 símbolos, 1 byte do SFD, onde temos mais 2 símbolos e o byte do início do quadro, que representa mais dois símbolos, resulta 12 símbolos o que fica 48/250 ms, totalizando o valor de td, que é o tempo para detectar um ACK oriundo do receptor, isto é, 48/250 = 0.192 ms. Por fim, temos o valor de tr dado pelo rádio CC2420, que é 0.192 ms. A equação (1) se torna então

$$0.352 < ti < tc < tc + 0.384 < ts. \quad (2)$$

As demais variáveis, o ContikiMAC define como: ti = 0.4 ms; tc = 0.5 ms; ts = 0.884 ms [10].

3) Sistema Operacional Contiki: O Contiki é um sistema operacional de código aberto, focado em redes de sensores, com uma arquitetura modular, suporte a agendamento e preempção multi-threading. Utiliza o Rime na camada de rede e é projetado para alta eficiência no gerenciamento de energia e memória. Escrito em C, o Contiki suporta programação baseada em eventos (protothreads), permitindo programação paralela eficiente. Ao contrário de outros sistemas, o Contiki carrega apenas as partes da aplicação necessárias, tornando-o leve em termos de uso de memória. Para economizar energia em redes IoT, o Contiki permite que os nós entrem em modo de baixo consumo (*sleep*) quando não estão enviando ou recebendo dados, sendo os mecanismos de conservação de energia implementados pela aplicação e protocolo de rede [13].

4) Consumo de Energia no Tmote Sky: Para calcular o consumo de energia, utiliza-se a ferramenta Powertrace, já implementada no sistema Contiki. Para incluí-la no binário final, é necessário informá-la ao compilador durante a compilação. O Powertrace utiliza o módulo Energest para estimar o consumo de energia, monitorando o tempo gasto em diferentes estados: CPU (processador ativo), LPM (*Low Power Mode*), Tx (transmissão de dados) e Rx (recepção de dados). O estado CPU ocorre quando o nó está ativo sem utilizar o transceptor, LPM é o modo de baixo consumo, Tx refere-se à transmissão e Rx à recepção de mensagens [14], [15].

O consumo de energia, segundo [14], pode ser calculado como o produto

$$E_{estado} = V_{cc} \times I_{estado} \times t_{estado}, \quad (3)$$

onde E_{estado} é a energia consumida em um determinado estado, V_{cc} é a tensão da bateria, I_{estado} é o consumo de corrente naquele estado, e t_{estado} é o tempo que o dispositivo permaneceu no referido estado. Por sua vez, [10] define que a energia de um nó é calculada como

$$E = \sum_{n,m} P_{m,n} T_{m,n}, \quad (4)$$

em que $P_{m,n}$ é a potência relativa ao consumo de energia do componente m em um estado n, e $T_{m,n}$ é o tempo que o componente m levou no estado n. Isto é, a energia total do nó é dada pela soma da energia em cada estado. Logo, é possível obter a equação da soma das energias de cada estado como

$$E = E_{CPU} + E_{LPM} + E_{Tx} + E_{Rx}. \quad (5)$$

Aplicando a equação (3) à equação (4) chega-se a

$$E = V_{cc}(I_{CPU}T_{CPU} + I_{LPM}T_{LPM} + I_{Tx}T_{Tx} + I_{Rx}T_{Rx}), \quad (6)$$

onde as variáveis I_{Tx}, I_{Rx}, I_{CPU} e I_{LPM} são dadas, respectivamente, pelos valores de consumo de corrente com o rádio ligado em modo Tx, consumo de corrente com o rádio ligado em modo Rx, consumo de corrente com CPU ligada com rádio desligado, e consumo de corrente em modo *sleep*. T_{CPU}, T_{LPM}, T_{Tx} e T_{Rx} se refere ao tempo que o nó permaneceu em cada estado. Os valores de corrente de cada estado podem ser obtidos do *datasheet* do dispositivo e estão na Tabela III.

TABELA III
PARÂMETROS DE OPERAÇÃO TÍPICA DO TMOTE SKY

Parâmetro	Mínimo	Nominal	Máximo	Unidade
Tensão de alimentação (V_{cc})	2.1		3.6	Volt (V)
Consumo de corrente: MCU ligado, Rádio Rx (I_{Rx})		21.8	23	mA
Consumo de corrente: MCU ligado, Rádio Tx (I_{Tx})		19.5	21	mA
Consumo de corrente: MCU idle, Rádio desligado (I_{CPU})		54.5	1200	µA
Consumo de corrente: MCU standby (I_{LPM})		5.1	21	µA

Entretanto, os valores retornados pelo módulo Energest da ferramenta Powertrace são dados em pulsos de clock do RTC, sendo necessária sua conversão para segundos, dada como

$$T_{estado} = \frac{t_{pulsos\_estado}}{RTIMER\_ARCH\_SECOND}, \quad (7)$$

onde T_{estado}, dado em segundos, equivale ao tempo em que a quantidade de pulsos de clock do dispositivo ficou naquele estado e é uma constante utilizada pelo Contiki para definir quantos pulsos de clock equivalem a um segundo. Para o dispositivo Tmote Sky esse valor é de 32768 pulsos de clock. Aplicando a equação (7) na equação (6) chega-se ao consumo total de energia por nó e por salto [14], como sendo

$$E_{ihop} = V_{cc} \times \frac{Txt_{Tx}+Rxt_{Rx}+CPUt_{CPU}+LPMt_{LPM}}{RTIMER\_ARCH\_SECOND}. \quad (8)$$

Nesta equação, t_{Tx}, t_{Rx}, t_{CPU} e t_{LPM}, se referem aos tempos, em pulsos de clock, que o dispositivo permaneceu naquele estado, e V_{cc} é a tensão de alimentação do dispositivo. Caso seja necessário calcular a potência em Watts consumida pelo dispositivo, é necessário obter o período que foi realizada a medição desses valores, isto é, o tempo entre quando os valores foram medidos [9]. Logo,

$$P = \frac{E_{ihop}}{t} = \frac{V_{cc}}{t} \times \frac{Txt_{tx}+Rxt_{Rx}+CPUt_{CPU}+LPMt_{LPM}}{RTIMER\_ARCH\_SECOND}. \quad (9)$$

C. O Protocolo Aloha Puro

O Protocolo Aloha Puro, protocolo de acesso ao meio (MAC) utilizado em redes sem fio, foi um dos primeiros protocolos de comunicação para redes sem fio criados [16]. A ideia básica por trás desse protocolo é simples: os dispositivos transmitem um pacote de dados sem coordenação com os demais nós da rede. Porém pode haver colisão se mais de um dispositivo transmitir ao mesmo tempo, o que pode acarretar em elevado consumo de energia devido às retransmissões dos pacotes que colidiram. Um dos pontos positivos nesse tipo de protocolo é não precisar de mensagens de sincronismo entre os nós. Um outro exemplo de tecnologia que adota essa estratégia como protocolo MAC é o LoRaWAN [2].

D. Simulador Cooja

Desenvolver uma nova funcionalidade em um dispositivo IoT tende a ser um grande desafio, já que alguns erros intrínsecos ao hardware podem aparecer (tempo de compilação, carregamento do código na placa, etc.). Logo para acelerar o desenvolvimento de funcionalidades e prever complicações que podem acontecer no hardware, pode ser empregado um simulador como o Cooja, emulando o Contiki OS na plataforma Tmote Sky [3], [4], [10].

O Cooja é um simulador de redes para o Contiki que permite a simulação completa de motes IoT [4]. O Simulador cria os motes e então compila o código do Contiki e o carrega dentro desses motes. Assim é possível, por exemplo, posicionar esses motes em uma área de duas dimensões (x,y), configurar taxa de sucesso de transmissão, verificar o status das mensagens de rádio, visualizar o *duty-cycle* do rádio e executar testes automatizados utilizando uma linguagem mista entre JavaScript e Java.

III. Aloha RDC: Proposta e Implementação

Para facilitar a análise, o ContikiMAC pode ser dividido em duas subcamadas: MAC e RDC. A subcamada MAC é a responsável por realizar a transmissão dos pacotes usando o CSMA/CA, implementar o *backoff* e tratar do recebimento de pacotes ACKs. A subcamada RDC tem como objetivo tratar o *duty-cycle* do rádio, quanto tempo este deve ficar ligado para transmissão e recebimento correto de pacotes, quanto tempo deve permanecer em *sleep*, verificar o canal e informar a subcamada MAC de colisões, ACKs de pacotes, e pacotes que não conseguiram ser enviados.

O Aloha foi implementado remodelando o protocolo CSMA/CA do ContikiMAC devido a maturidade do código original, já que possui fila de pacotes, lista de vizinhos, tratamento em caso de ACK recebido e não recebido e outros itens que são do funcionamento do próprio sistema operacional que são relevantes e já foram testados previamente, dando robustez ao código final.

A principal contribuição desta implementação é a adaptação do ContikiMAC para o protocolo Aloha. Isso envolveu a remoção dos mecanismos de colisão do CSMA/CA, a reestruturação da lógica de recepção de pacotes e a implementação de um Radio Duty Cycling (RDC) compatível com o Aloha. A abordagem foi estruturada em seis etapas, com destaque para as etapas 1 e 4, conforme ilustrado na Fig.2.

A. Remodelagem do Transmissor ContikiMAC para Aloha

O protocolo Aloha puro não emprega detecção de portadora antes do envio de pacotes, logo o CCA na transmissão não é aqui empregado na remodelagem do ContikiMAC e retira-se a parte de transmissão em rajada (*burst*) de pacotes nativo do Contiki, já que isto também não pertence ao protocolo Aloha puro [17].

Fig. 2. As fases de operação da implementação do Aloha para o Contiki-OS são: 1) Remodelagem do Transmissor; 2) Análise e revisão do código-fonte original do ContikiMAC; 3) Remoção dos mecanismos de tratamento de colisão do CSMA/CA, conforme a especificação IEEE 802.15.4; 4) Remodelagem da recepção de pacotes; 5) Reestruturação da lógica de recepção para adequar-se ao funcionamento do Aloha com RDC; e 6) Implementação do mecanismo de RDC compatível com Aloha, com testes e ajustes nos parâmetros de *backoff* para otimização do desempenho do Aloha com RDC.

Entretanto, é necessário manter ainda o *burst* de pacotes para o tipo *broadcast*, visto que estes servem para que a camada de rede (e o roteamento) consiga montar a lista de vizinhos. Durante a execução das simulações foi percebido que sem este tratamento a camada de rede demora ou não consegue montar a lista de vizinhos. Como essa implementação não

realiza CCA, caso o rádio esteja em uso pelo receptor no início, é retornado uma mensagem interna NoACK. E caso após a tentativa de envio do pacote não seja recebido ACK, o transmissor faz *backoff*. A partir dessa remodelagem foi obtido o Algoritmo 1 chamado de Transmissor Aloha para ContikiMAC.

Algoritimo 1 Transmissor Aloha para ContikMAC

```
1: Início
2: Preparar o pacote para envio
3: se recebendo pacote? então
4:     Retornar NoACK
5: senão
6:     se pacote é broadcast então
7:         Enviar burst de pacotes
8:     senão
9:         Enviar pacote
10:        Esperar um tempo para receber ACK
11:        se recebeu ACK então
12:            Fim
13:        senão
14:            Faz backoff
15:            Volta para Início
16:        fim se
17:    fim se
18: fim se
19: Fim
```

B. Remodelagem da Recepção de Pacote no Aloha ContikiMAC com RDC

O Contiki divide sua pilha de comunicação em MAC e RDC, como explicado anteriormente. É necessário, portanto, fazer uma remodelagem da parte do RDC do Contiki, onde o responsável principal é o ContikiMAC que envia rajadas do pacote, com o objetivo de que um receptor consiga perceber que há eventos no canal e consiga obter essas informações após acordar.

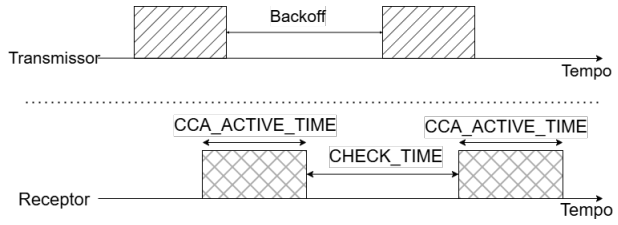

Fig. 3. Exemplo de como o *backoff* pode afetar a recepção do pacote.

No caso do RDC ContikiMAC para o recetor Aloha, a janela de tempo de checagem do canal (CCA), tr, deve obter desde o preâmbulo do pacote até o seu fim, em outras palavras, a mesma deve estar ligada por tempo suficiente para que o pacote seja identificado de forma correta. Além disso, a utilização de dois CCAs não é o ideal, pois ao utilizar desta forma, a checagem da janela de canal poderia cair no problema exibido na Fig. 3 em que o *backoff* no transmissor coincide quase que totalmente com o CCA no receptor.

A Fig. 4 apresenta o funcionamento ideal do envio de um pacote no RDC Aloha. A janela de checagem do canal, tr, deve detectar o preâmbulo do pacote e ter duração para o recebimento completo do pacote. Caso não consiga obter o preâmbulo, o protocolo não conseguirá decifrar o pacote. Como informado antes, um outro ponto importante a ser levado em consideração é o tempo de *backoff* do pacote.

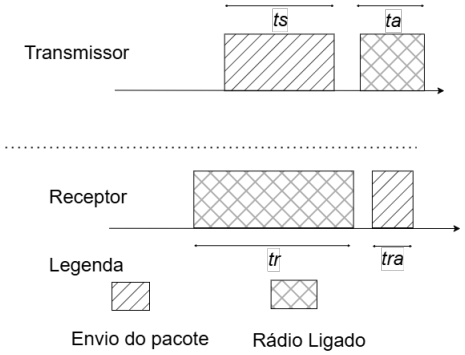

Fig. 4. Exemplo de envio de pacote no Aloha RDC.

Conforme exibido na Fig. 3, o tempo de *backoff* pode resultar em um ciclo infinito onde o transmissor não consegue se comunicar com o receptor porque não consegue transmitir o pacote quando o receptor está com CCA do canal ativo. Assim, após a detecção de atividade, essa janela deve ficar ativa o tempo suficiente para que o pacote seja lido corretamente. Após recebimento do pacote com sucesso, o transmissor espera um tempo ta, que é o tempo necessário para obter um ACK do receptor, sendo tra o tempo que o pacote ACK leva pra ir do receptor ao transmissor, onde [10] estabelece esse valor como 0.16 milisegundos.

Segundo [10] e analisando o próprio código do ContikiMAC, o tempo de CHECK_TIME definido é de 125 ms. A variável CCA_ACTIVE_TIME é o tempo que a janela de checagem do canal fica ativa no receptor, isto é, o tempo que o dispositivo fica com o rádio ligado executando CCA para verificar se há pacotes no meio, conforme exibido no Algoritmo 2 onde é mostrado o $protothread$ inicializado para controlar o ciclo de $sleep$, realizando verificações de temporização do $timer$ para determinar se foi expirado, e detecção de pacotes no enlace através de CCA. Caso o pacote seja detectado, ele mantém o rádio ligado até completar o recebimento.

A função responsável por agendar o envio dos pacotes utiliza uma estrutura chamada *ctimer*, abreviação para *callback timer*, que recebe valores do tipo *clock time t*, medidos em ciclos de *clock*. Utilizando a plataforma Tmote Sky, o Contiki define que a variável CLOCK SECOND tem valor 128, ou seja, um segundo equivale a 128 pulsos de clock [3], [18].

Com base nessas informações é possível calcular valores para o *backoff*. Entretanto é necessário definir antes alguns valores para o CCA_ACTIVE_TIME, estes valores têm como unidade pulsos de *clock*. A plataforma Tmote Sky tem o RTC com *clock* de 32768 Hz, conforme visto na Tabela I. Logo, por exemplo, considerando 500 pulsos de clock para o valor

Algoritimo 2 Recebimento de pacote no Aloha RDC

```
1: Inicia protothread (responsável pelo ciclo de sleep)
2: enquanto Verdadeiro faça
3:     se Timer expirado então
4:         Desliga o rádio e põe um temporizador para chamar novamente a função
5:     senão
6:         Faz CCA
7:         se Existe pacote no meio então
8:             Mantém o rádio ligado
9:             se Terminou o recebimento do pacote então
10:                Continue
11:            senão
12:                Volta ao início do recebimento do pacote
13:            fim se
14:        senão
15:            se Realizou 2 CCAs então
16:                Volta ao início do ciclo de sleep
17:            senão
18:                Faz nova tentativa de CCA
19:            fim se
20:        fim se
21:    fim se
22: fim enquanto
```

de CCA_ACTIVE_TIME obtém-se

$$CCA\_ACTIVE\_TIME = \frac{500}{32768} = 0.015259\ s \approx 15.26\ ms. \quad (10)$$

Com isso podemos calcular o *duty-cycle* referente a este valor, assim

$$Duty\,Cycle = \frac{CCA\_ACTIVE\_TIME}{CCA\_ACTIVE\_TIME + CHECK\_TIME}, \quad (11)$$

que resulta percentualmente como

$$Duty\,Cycle\,(\%) = \frac{15.26}{15.26+125} \times 100 = 0.1088 \times 100 \approx 11\%. \quad (12)$$

Logo para o valor de 500 pulsos de *clock* de CCA_ACTIVE_TIME obtém-se um *duty-cycle* de 11%. Considerando alguns valores de *duty-cycle* na Tabela IV pode-se ajudar numa estimativa para o *backoff*.

TABELA IV
DUTY-CYCLES E TEMPOS ASSOCIADOS

Duty-Cycle (%)	CCA_ACTIVE _TIME (ms)	CCA_ACTIVE _TIME (pulsos de *clock*)	Tempo total do ciclo (ms)
1	1.25	41	126.25
3	3.88	127	128.88
5	6.57	216	131.57
10	13.89	455	138.39
20	31.25	1024	156.25
30	53.57	1755	178.57
40	83.33	2731	208.33
50	125	4096	250
60	187.50	6144	312.5

Com base nos valores apresentados na Tabela IV, o valor do limite inferior do *backoff* deve ser superior ao menor *duty-cycle* considerado — neste caso, 1%, que é o valor padrão adotado pelo Contiki-OS, e inferior ao maior *duty-cycle* analisado, que é de 60%. Define-se uma faixa apropriada com base nos ciclos de *clock* da seguinte forma

$$\frac{1.25\times128}{1000} < backoff < \frac{187.50\times128}{1000} \quad (13)$$

$$0.16 < backoff < 24. \quad (14)$$

Fazendo então a função teto desses dois valores chega-se a

$$1 < backoff < 24. \quad (15)$$

A partir desta relação e usando a função $random\_rand()$ do Contiki, que retorna valores de 0 a 65535, que será dividida em módulo 22 pra gerar valores entre 0 e 21 e é somado 2 pra gerar valores entre 2 e 24, obtém-se a equação a seguir que foi utilizada no código implementado no simulador

$$clock\_time\_t\, backoff = random\_rand()\%22 + 2. \quad (16)$$

Estes ciclos de clock são baseados no valor da variável CLOCK SECOND, que tem valor de 128 pulsos de *clock*. Em outras palavras, o tempo de *backoff* será de 2 a 23 pulsos de *clock*. Pode-se transformar esses valores em milissegundos utilizando a relação

$$t = \frac{backoff}{CLOCK\_SECOND}. \quad (17)$$

Portanto, os valores de *backoff* de 2 a 23 pulsos de *clock* resultam o intervalo entre 15 e 180 milissegundos.

IV. RESULTADOS E ANÁLISE

Os experimentos foram realizados no simulador Cooja usando três dispositivos espaçados de forma uniforme, com distância entre cada dispositivo de 45 metros (m) em topologias de um ou mais saltos, que representa um cenário clássico de teste *daisy-chain*, conforme exemplo na Fig. 5 para os nós 1, 2 e 3. Para os experimentos foi realizado uma comparação do consumo de energia dos protocolos da seguinte maneira:

- Aloha sem RDC (Rádio sempre ligado);
- Aloha com RDC (Variando *duty-cycle*);
- CSMA/CA com RDC do ContikiMAC padrão.

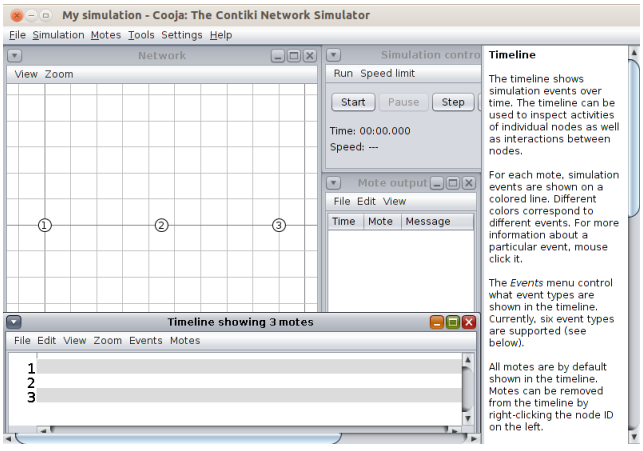

Fig. 5. Interface do simulador Cooja.

Para cada experimento foi criado um arquivo CSV contendo os dados de quanto tempo o dispositivo permaneceu nos estados de CPU, LPM, TX, RX, a quantidade de saltos, a distância total (d), a taxa de transmissão (R) e o tamanho do pacote (Nb) utilizado, conforme o exemplo de estrutura exibida na Tabela V.

TABELA V
ESTRUTURA DO ARQUIVO CSV

CPU (ms)	LPM (ms)	Tx (ms)	Rx (ms)	Saltos (*hop*)	d (m)	R (kbps)	Nb (byte)
63026	924324	8640	1101	1	45	250	52
22800	304673	384	3828	2	90	250	52

Para a geração dos gráficos foram utilizados os valores de parâmetros nominais informados na Tabela III e o tamanho do pacote foi de 52 bytes. Além disso, para ser justo na comparação, foi utilizada a energia por bit (E_{ihop}/bit), isto é, a energia consumida foi dividida pelo tamanho do pacote, sendo dada em unidades de Joules por bit (J/bit).

Os valores de *duty-cycle* escolhidos estão inidicados na Tabela IV. Para todas as simulações foi considerada a topologia de rede exibida na Fig. 6, onde cada nó está espaçado 45 metros um do outro, esse valor foi escolhido pois foram utilizados os valores padrões da simulação do Cooja, onde um dispositivo do tipo do Tmote Sky possui um alcance de rádio de 50 metros. Como era necessário obter dados com mais de um salto (isto é, o nó 2 roteia mensagens do nó 3), foi utilizado esse valor para evitar sobreposição de cobertura de sinal do nó 1 com o nó 3. Foi considerado também que o nó 1 é sempre o nó que recebe as informações, denominado de *sink*.

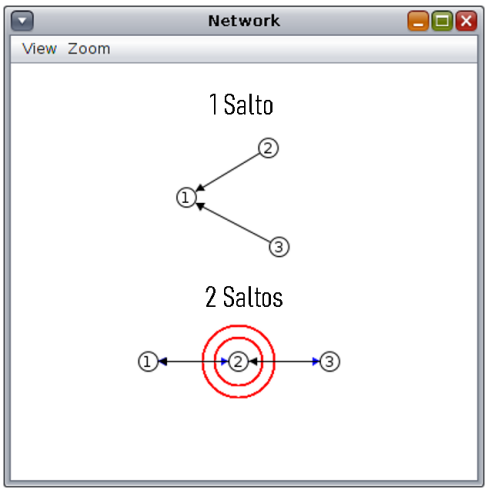

Fig. 6. Topologias de rede usada nas simulações Cooja.

Os resultados foram obtidos com um intervalo de confiança de 95% com grau de liberdade de valor 10 (dez) [19]. Esse grau de liberdade implica realizar 11 (onze) vezes os experimentos usando uma semente aleatória em cada execução.

A Tabela VI apresenta a comparação do consumo médio de energia por nó entre os protocolos ContikiMAC com CSMA/CA, nativo do Contiki OS, e o ContikiMAC com Aloha sem RDC, um dos protocolos implementado neste trabalho. Os dados foram obtidos por meio de simulações no Cooja, utilizando as classes padrão do Contiki. Para o ContikiMAC com CSMA/CA e RDC de 1%, foram recebidas 15770 mensagens, enquanto para o Aloha sem RDC, com o rádio permanentemente ligado, foram recebidas 15764 mensagens.

Como esperado, os resultados indicam que o consumo energético do Aloha puro é significativamente mais elevado em relação ao protocolo com CSMA/CA e RDC. Esse comportamento se deve à ausência de mecanismos de economia de energia no Aloha sem RDC, como a escuta periódica e o controle do tempo de atividade do rádio, o qual permanece continuamente ligado para transmissão e recepção de pacotes.

TABELA VI
COMPARATIVO DE CONSUMO MÉDIO DE ENERGIA ENTRE PROTOCOLOS (J/BIT)

Protocolos	1 Salto	2 Saltos
CSMA/CA	$4.02 \times 10^{-}5$	$3.34 \times 10^{-}5$
Aloha sem RDC	$1.57 \times 10^{-}3$	$1.58 \times 10^{-}3$

A. *ContikiMAC Aloha com RDC*

As Figs. 7 e 8 mostram o desempenho do Contiki Aloha com RDC para 1 e 2 saltos, respectivamente, em diferentes casos de ciclos de carga. Nota-se que quanto menor o percentual do *duty-cycle* empregado, menor o consumo médio por bit transmitido. Sabe-se que ao se reduzir o percentual de tempo que o nó fica ativo, a quantidade média de mensagens trocadas ao longo do tempo tende a diminuir.

Assim, a quantidade de mensagens entregues no mesmo tempo de simulação não é igual para todos, isto pode ser observado na Tabela VII, já que a quantidade de mensagens é função do tempo em que os dispositivos ficam ligados para executar a comunicação.

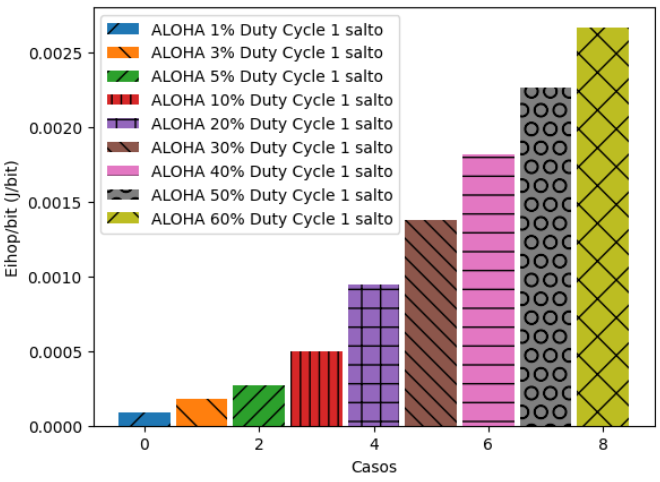

Fig. 7. Consumo de energia versus o *duty-cycle* no Aloha RDC para 1 salto.

A partir da Tabela VII e das Figs. 7 e 8 é possível observar que embora as quantidades de pacotes recebidos sejam menores para *duty-cycles* baixos, como 1% e 3%, ainda foi

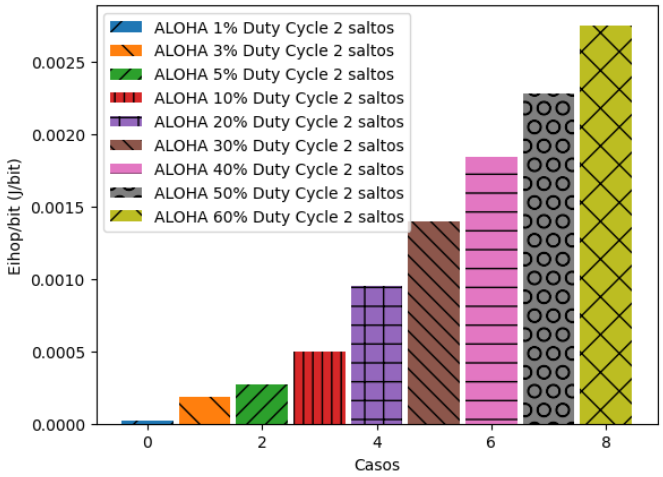

Fig. 8. Consumo de energia versus o *duty-cycle* no Aloha RDC para 2 saltos.

possível o envio destes pacotes, pois ocorreram casos em que o transmissor encontrou o receptor checando o canal e logo no início da checagem do canal, o que possibilitou a leitura do pacote por parte do receptor. Além disso, se o consumo médio de energia for o ponto principal da aplicação de interesse, desconsiderando atrasos na entrega das mensagens, esses valores de *duty-cycles* resultaram uma maior eficiência energética, o que poderia ocasionar uma longevidade maior aos nós e à rede. Mas se for necessário uma taxa maior de entrega, deve-se empregar um percentual de *duty-cycle* mais elevado, observando-se assim que há um compromisso entre a taxa de entrega de pacotes e o consumo de energia em função do *duty-cycle*.

TABELA VII
MENSAGENS ENTREGUES EM FUNÇÃO DO DUTY-CYCLE

Duty-cycle (%)	Intervalo de tempo do *duty-cycle* (ms)	Quantidade de mensagens obtidas pelo nó *sink*
1	126.25	371
3	128.88	1427
5	131.57	2205
10	138.39	3957
20	156.25	6894
30	178.57	10503
40	208.33	12885
50	250	12529
60	312.5	12420

B. Comparação entre os Protocolos

As Figs. 9 e 10 exibem a comparação do consumo de energia por bit para diferentes valores de *duty-cycle* para os protocolos Conitki MAC CSMA/CA RDC, ConitkiMAC Aloha sem RDC, e o ContikiMAC Aloha com RDC, para 1 e 2 saltos, respectivamente. Observa-se que o Aloha com RDC de 1% obteve o menor consumo de energia médio por bit entre todos os experimentos sendo melhor que até mesmo o próprio CSMA/CA padrão do Contiki com RDC também de 1%. Isso se deve ao fato de que o rádio passa menos tempo ligado devido ao menor *duty-cycle*, o que ajuda na redução do tempo de escuta do canal, diminuindo a energia gasta neste estado e também pelo fato de que o Aloha não faz checagem (CCA) do canal.

Na Fig. 11 é possível observar com maior detalhe que a nova proposta do protocolo Aloha RDC com 1%, 3% e 5% teve menor consumo médio de energia por bit do que o CSMA/CA nativo do ContikiMAC com RDC de 1%, consequência do protocolo Aloha não precisar fazer checagem do canal para enviar dados.

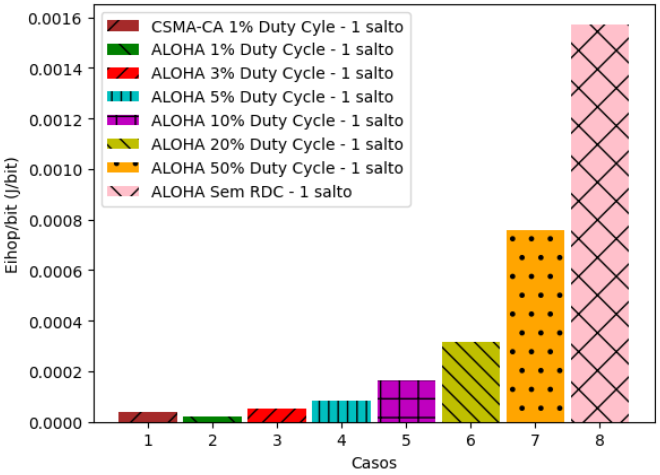

Fig. 9. consumo de energia por bit para diferentes valores de *duty-cycle* para os protocols Conitki MAC CSMA/CA, ConitkiMAC Aloha sem RDC, e o ContikiMAC Aloha com RDC para 1 salto.

A partir da Tabela VIII é possível tirar outras conclusões. Embora apresentem menor consumo energético, os valores de *duty-cycle* de 1%, 3% e 5% do Aloha RDC possuem baixa entrega de mensagem, em outras palavras, isso significa que a taxa de recebimento (quantidade de pacotes recebidos dividido pela quantidade de pacotes enviados) é reduzida se comparada ao CSMA/CA. Como os nós passam em média mais tempo desligados é natural que isso ocorra já que não foi implementado aqui nenhuma estratégia de sincronismo entre os nós. Assim, o Contiki Aloha RDC aqui proposto se adequa melhor às aplicações que necessitem reduzido consumo de energia, mas possuam baixa vazão e tolerem atrasos como nos casos de sensoriamento de temperatura ambiente para fins climáticos, ou aplicações em agricultura para obter informações como a umidade do solo [20].

TABELA VIII
COMPARAÇÃO DE QUANTIDADE DE MENSAGENS RECEBIDAS

Protocolo	Quantidade de mensagens de 1 salto	Quantidade de mensagens de 2 saltos	Total de mensagens
Aloha RDC - 1% Duty-cycle	369	2	371
Aloha RDC - 3% Duty-cycle	1419	8	1427
Aloha RDC - 5% Duty-cycle	2195	10	2205
CSMA-CA	7884	7886	15770

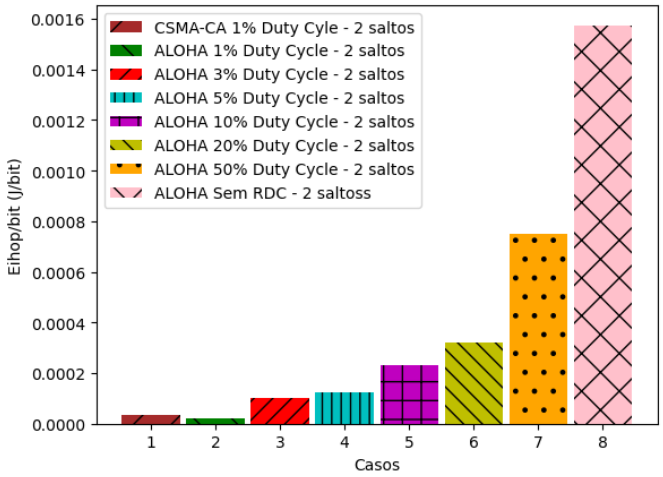

Fig. 10. consumo de energia por bit para diferentes valores de *duty-cycle* para os protocols Conitki MAC CSMA/CA, ConitkiMAC Aloha sem RDC, e o ContikiMAC Aloha com RDC para 2 saltos.

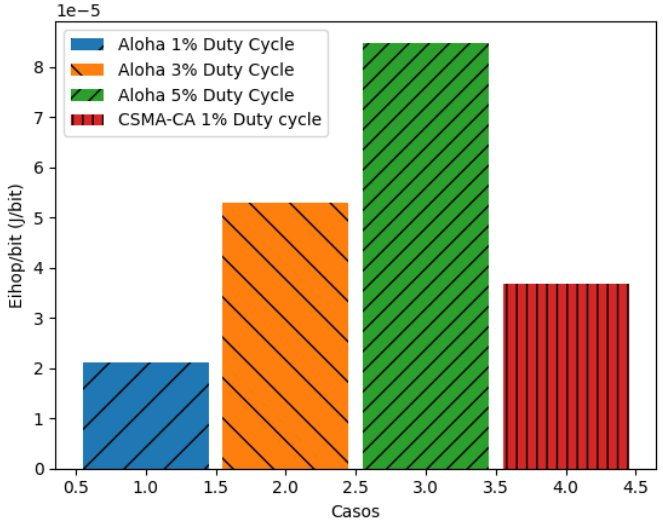

Fig. 11. consumo de energia por bit para os protocols Conitki MAC CSMA/CA com 1% de *duty-cycle* e ConitkiMAC Aloha RDC (com *duty-cycle* 1%, 3%, 5%) para 1 salto.

V. Conclusões

Este trabalho implementou o protocolo Aloha Puro com gerenciamento de RDC no sistema operacional Contiki, como alternativa ao MAC CSMA/CA antivo. A contribuição é relevante por introduzir suporte ao Aloha na plataforma, ampliando as possibilidades de pesquisa e viabilizando futuras implementações baseadas nesse protocolo, como o LoRa.

Simulações no Cooja compararam o desempenho do Aloha Puro, com e sem RDC, ao CSMA/CA do ContikiMAC, mostrando que o Aloha com RDC foi 40% mais eficiente energeticamente, com redução de até 87.5% em alguns cenários, o que pode aumentar em até um terço a vida útil das baterias dos dispositivos Tmote Sky. No entanto, houve uma queda na taxa de entrega de pacotes, especialmente em topologias multi-hop.

Apesar das limitações, o Aloha-RDC demonstrou eficiência em topologias de até um salto, sendo adequado para aplicações com baixo volume de dados e baixa frequência de atualização, como detecção e monitoramento em ambientes urbanos e agrícolas. Nesses casos, a prioridade por eficiência energética supera a necessidade de alta vazão, tornando o protocolo uma alternativa viável para aplicações tolerantes a atrasos e com baixo tráfego.

Para trabalhos futuros, sugere-se a melhoria do mecanismo de *backoff* adaptado ao *duty cycle*, a implementação de técnicas de sincronização entre os nós para aumentar a vazão, e a adaptação do Aloha-RDC à tecnologia LoRaWAN, visando seu uso em redes de longo alcance e baixa potência.

References

[1] Z. Almudayni, B. Soh, H. Samra, and A. Li, "Energy inefficiency in iot networks: Causes, impact, and a strategic framework for sustainable optimisation," *Electronics*, vol. 14, no. 1, p. 159, 2025.

[2] I. Araújo, V. Barbosa, L. N. Lima, L. G. Silva, C. Brito, I. Fé, L. Lopes, E. Andrade, E. Leão, and F. A. Silva, "Assessing the performance of a fault tolerant lorawan architecture with a focus on the sensor layer and data retransmission strategy," *Cluster Computing*, vol. 28, no. 4, p. 275, 2025.

[3] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The contiki-ng open source operating system for next generation iot devices," *SoftwareX*, vol. 18, p. 101089, 2022, doi: 10.1016/j.softx.2022.101089.

[4] Contiki-NG Development Team, "Cooja network simulator: Part of the contiki-ng operating system," 2018, accessed: October 30, 2024. [Online]. Available: https://github.com/contiki-ng/contiki-ng

[5] M. Abdulkarem, K. Samsudin, F. Z. Rokhani, and M. F. A Rasid, "Wireless sensor network for structural health monitoring: A contemporary review of technologies, challenges, and future direction," *Structural Health Monitoring*, vol. 19, no. 3, pp. 693–735, 2020, doi: 10.1177/1475921719854528.

[6] S. Sreenivasamurthy and K. Obraczkaa, "Clustering at the edge: Load balancing and energy efficiency for the iot," *Ad Hoc Networks*, vol. 154, no. 103433, pp. 1–17, 2024, doi: 10.1016/j.adhoc.2024.103433.

[7] J. K. S. Silva, A. R. S. Gois, P. F. C. Barbosa, and R. M. de Moraes, "Protocolo aloha puro para sistema operacional contiki," Patent Software BR512 024 003 629-6, 2024.

[8] E. J. Costa e Silva, K. Y. Goncalves Vieira Guedes, P. A. Araújo da Silva de Almeida Nava Alves, P. R. de Almeida Ribeiro, and A. Oliveira Barradas Filho, "Systematic review of radio wave techniques for indoor positioning systems," *IEEE Latin America Transactions*, vol. 23, no. 3, pp. 205–215, 2025.

[9] P. F. Barbosa, B. A. da Silva, C. Zanchettin, and R. M. de Moraes, "A multi-protocol energy optimization method for an adaptable wireless mac system through machine learning," *Annals of Telecommunications*, vol. 80, no. 1, pp. 1–15, 2025, doi: doi.org/10.1007/s12243-023-01004-2.

[10] S. Homayouni and R. Javidan, "Era-contikimac: An adaptive radio duty cycling layer in internet of things," in *2018 9th International Symposium on Telecommunications (IST)*, 2018, pp. 74–79, doi: 10.1109/ISTEL.2018.8661144.

[11] T. S. Datasheet, "Moteiv corporation," *Saatavissa (viitattu 1.10. 2017)*, 2006, accessed: October 30, 2024. [Online]. Available: http://www.crew-project.eu/sites/default/files/tmote-sky-datasheet.pdf

[12] A. Koubâa, M. Alves, and E. Tovar, "Gts allocation analysis in ieee 802.15. 4 for real-time wireless sensor networks," in *Proc. 20th IEEE Int. Parallel & Distrib. Process. Symp.* IEEE, 2006, pp. 8–pp, doi: 10.1109/IPDPS.2006.1639415.

[13] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *29th Annu. IEEE Int. Conf. Local Comput. Netw. (LCN)*. IEEE, 2004, pp. 455–462, doi: 10.1109/LCN.2004.38.

[14] V. G. Guimaraes, R. M. de Moraes, K. Obraczka, and A. Bauchspiess, "A novel iot protocol architecture: Efficiency through data and functionality sharing across layers," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019, pp. 1–9, doi: 10.1109/ICCCN.2019.8846919.

[15] M. Amirinasab Nasab, S. Shamshirband, A. T. Chronopoulos, A. Mosavi, and N. Nabipour, "Energy-efficient method for wireless sensor networks low-power radio operation in internet of things," *Electronics*, vol. 9, no. 2, p. 320, 2020, doi: 10.3390/electronics9020320.

[16] N. Abramson, "The aloha system: Another alternative for computer communications," in *Proceedings of the November 17-19, 1970, Fall Joint Computer Conference*, 1970, pp. 281–285, doi: 10.1145/1478462.1478502.

[17] S. Penchala, S. K. Bandari, V. V. Mani, and A. Drosopoulos, "Controlled wireless channel using multi-antenna multi-irs assisted communication system: A comprehensive performance analysis," *IEEE Latin America Transactions*, vol. 23, no. 2, pp. 114–124, 2025.

[18] Contiki-OS, "The open source os for the internet of things," 2012, accessed: October 30, 2024. [Online]. Available: http://www.contiki-os.org

[19] B. R. Haverkort, *Performance of Computer Communication Systems: A Model-Based Approach*. USA: John Wiley & Sons, Inc., 1998, isbn: 0471972282.

[20] M. S. Farooq, S. Riaz, A. Abid, T. Umer, and Y. B. Zikria, "Role of iot technology in agriculture: A systematic literature review," *Electronics*, vol. 9, no. 2, p. 319, 2020, doi: 10.3390/electronics9020319.

Jonathan Kilner dos Santos Silva received his bachelor's degree in Electronics from the Federal Institute of Pernambuco (IFPE) in 2017 and in Computer Engineering from the Federal University of Pernambuco (UFPE) in 2024, with a solid technical foundation spanning electronic systems to software engineering. Currently, he works as a software developer focused on web solutions.

Paulo Filipe Cândido Barbosa professor at Universidade de Pernambuco (UPE), with a Post-Doctorate (2024) and Ph.D. (2023) in Computer Science from Universidade Federal de Pernambuco (UFPE). Experienced in Computer Networks, IoT, Sensors, Web Design, and AI, with expertise in machine learning, embedded systems, evolutionary computation, multi-agent systems, AI in healthcare, and cellular automata modeling.

Renato Mariz de Moraes received his bachelor's degree in electrical engineering from Universidade Federal de Pernambuco in 1996, his master's from Universidade Estadual de Campinas in 1998, and his Ph.D. from the University of California, Santa Cruz, in 2005. He is an Associate Professor at Centro de Informática, Universidade Federal de Pernambuco, Brazil, and an IEEE Senior member. He has published over 60 journal and conference papers. His research interests include ad hoc and sensor networks, IoT, and computer networks. He has served on the TPC for various IEEE conferences and as Technical Program Chair for several events.