Codificação Linear Aleatória em Redes TSCH de dispositivos IoT

Luiz Filipe Menezes Vieira [1], **Vinícius Braga Freire**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte – Minas Gerais – Brasil

{lfvieira,vinicius.braga}@dcc.ufmg.br

Abstract. *The Internet of Things (IoT) has revolutionized everyday life by connecting devices and enhancing their communication and intelligence capabilities. However, excessive energy consumption remains a critical challenge, particularly in low-power wireless networks. This study investigates the application of Random Linear Network Coding in TSCH (Time Slotted Channel Hopping) networks, an approach that promises to enhance energy efficiency and network robustness. Implementations and simulations conducted in the Contiki-NG/Cooja environment demonstrated that combining TSCH with network coding techniques reduces latency and the number of exchanged messages, optimizing network performance under varying packet loss conditions. The results highlight the relevance of these techniques for IoT networks, encouraging their evaluation in physical environments.*

Resumo. *A Internet das Coisas (IoT) tem revolucionado o cotidiano, conectando dispositivos e ampliando sua capacidade de comunicação e inteligência. Entretanto, o consumo energético excessivo representa um desafio crítico, especialmente em redes sem fio de baixa potência. Este trabalho explora a aplicação da Codificação Linear Aleatória em redes TSCH (Time Slotted Channel Hopping), uma abordagem que promete aumentar a eficiência energética e a robustez dessas redes. Implementações e simulações conduzidas no ambiente Contiki-NG/Cooja demonstraram que a combinação do TSCH com técnicas de codificação em rede reduz a latência e o número de mensagens trocadas, otimizando o desempenho da rede em cenários com diferentes níveis de perda de pacotes. Os resultados ressaltam a relevância dessas técnicas para redes IoT, incentivando sua avaliação em ambientes físicos.*

1. Introdução

O crescimento da Internet das Coisas (IoT, do inglês *Internet of Things*) tem revolucionado a forma como dispositivos interagem, tornando sistemas cotidianos mais inteligentes e interconectados. À medida que a IoT continua a se expandir em ambientes industriais, urbanos e domésticos, a eficiência energética e o desempenho das redes tornam-se desafios críticos. A necessidade de otimizar a comunicação em redes sem fio de baixa potência tem impulsionado pesquisas sobre protocolos e técnicas que melhoram tanto a confiabilidade quanto a eficiência energética, garantindo operações robustas e duradouras.

Um problema persistente em redes IoT é o alto consumo de energia causado por transmissões frequentes de rádio, agravado por perdas de pacotes que exigem retrans-

missões. O protocolo TSCH (*Time Slotted Channel Hopping*) aborda alguns desses desafios ao programar transmissões e minimizar colisões (Hermeto et al. 2017), mas ainda enfrenta ineficiências em cenários com altas taxas de perda de pacotes. Em tais ambientes, abordagens tradicionais de comunicação muitas vezes são insuficientes para manter a robustez e a eficiência da rede.

Neste trabalho, investigamos a aplicação da Codificação Linear Aleatória em Redes (RLNC) sobre redes TSCH com o objetivo de melhorar a eficiência da comunicação e a economia de energia em ambientes IoT. Demonstramos que a redução das retransmissões, viabilizada pela codificação em rede (Vieira and Vieira 2017), permite uma economia significativa de energia nos dispositivos. Além disso, mostramos que a combinação entre TSCH e RLNC aumenta a robustez da rede, permitindo uma manutenção mais eficiente da comunicação, mesmo em cenários com altas taxas de perda de pacotes. Por meio de simulações realizadas no ambiente Contiki-NG/Cooja, verificamos uma diminuição notável na latência e na sobrecarga de mensagens em comparação a redes TSCH convencionais, evidenciando a relevância desta abordagem para redes IoT de baixa potência.

Enquanto estudos anteriores exploraram TSCH e codificação em rede de forma independente, este trabalho se concentra exclusivamente em sua aplicação combinada. Demonstramos como a sinergia entre essas técnicas pode levar a uma rede mais eficiente em termos de energia e robustez, representando uma melhoria significativa em relação às soluções existentes para ambientes IoT.

O restante deste trabalho está estruturado da seguinte forma: a Seção 2 aprofunda mais sobre a problemática. A Seção 3 fornece o embasamento teórico. A seção 4 mostra os trabalhos relacionados, destacando o estado da arte em TSCH e codificação em rede. A Seção 5 detalha a implementação da solução proposta. A Seção 6 descreve a metodologia experimental, e a Seção 7 apresenta e analisa os resultados. Por fim, a Seção 8 conclui o trabalho e discute perspectivas futuras.

2. Motivação

2.1. Problemática

Uma das principais problemáticas enfrentadas pelos dispositivos IoT é o elevado consumo de energia associado à comunicação via rádio. O uso contínuo do rádio para enviar e receber dados consome uma quantidade considerável de energia, o que é agravado pela necessidade de retransmissões em redes suscetíveis a interferências e perda de pacotes. A comunicação sem fio, cujo um dos maiores desafios é sua eficiência energética, representa um gargalo para a implementação de redes IoT de baixa potência e longa duração.

Além disso, redes IoT operam em ambientes frequentemente desafiadores, como áreas urbanas densas, indústrias ou zonas rurais, onde interferências e falhas de conexão são comuns. Isso resulta em uma alta taxa de perda de pacotes, o que implica a necessidade de retransmissões e, consequentemente, em um aumento do consumo energético.

2.2. Solução Proposta

Para mitigar o problema do consumo excessivo de energia em dispositivos IoT, a combinação de duas abordagens se mostra promissora: o padrão TSCH e o *Network Coding*. O TSCH, por organizar as transmissões em *slots* de tempo específicos e alternar

entre canais de frequência, reduz significativamente o tempo em que os rádios dos dispositivos precisam estar ativos, permitindo que entrem em modo de economia de energia nos intervalos de inatividade.

Complementando o TSCH, o *Network Coding* introduz um método eficiente de transmissão ao combinar pacotes de dados (Zhu et al. 2021). Em vez de enviar pacotes individuais, os nós da rede podem realizar combinações e, posteriormente, decodificar os dados durante o processo de transmissão. Isso reduz a necessidade de retransmissões, uma vez que, durante o percurso, a informação deste pacote pode ter sido combinada a outro pacote (Rajan et al. 2023). A solução proposta neste artigo é a combinação dessas técnicas, que promete aumentar a robustez da rede e, ao mesmo tempo, diminuir o consumo de energia, tornando-a uma solução ideal para redes IoT de baixa potência.

3. Base Teórica

3.1. Salto de Canais com Intervalo de Tempo

O TSCH (*Time Slotted Channel Hopping*, ou Salto de Canais com Intervalo de Tempo em português) é um protocolo da camada MAC (*Medium Access Control*, ou Controle de Acesso ao Meio em português). Este depende de uma estrita organização das transmissões entre nós de uma rede para que o mesmo seja determinístico e remova ao máximo colisões de transmissões (Hermeto et al. 2017).

A principal ideia deste protocolo gira em torno de intervalos de tamanho fixo no qual o tempo é dividido. Em acréscimo, o protocolo fraciona o tempo igualmente em cada um dos canais de frequência disponíveis, tornando-se então uma operação bidimensional. Chama-se de células um intervalo de tempo em uma frequência específica, como pode ser visto na figura 1 no canto inferior direito.

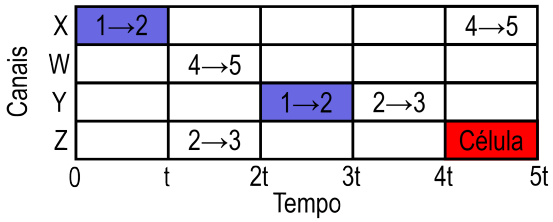

Figure 1. Exemplo de funcionamento do TSCH

Para atingir os objetivos do protocolo citados, o mesmo utiliza da seguinte política: um mesmo *link* (i.e. uma conexão) recebe um conjunto de células. Somente este link pode utilizar das células que lhe foram atribuídas, evitando assim a colisão entre diferentes *links*. Como os nós de uma rede só transmitem ou recebem durante uma célula a eles alocada, todos os outros nós durante aquele intervalo de tempo e naquela frequência podem entrar em modo de inatividade, economizando assim energia.

É importante ressaltar que o TSCH também faz o que chamamos de *Channel Hopping* (i.e. Salto de Canais, em português). Um *link* após utilizar uma célula de um canal X irá usar, na próxima vez que se tornar ativo, outro canal de frequência Y onde a condição $Y \neq X$ é satisfeita. Isto reduz interferências e desvanecimento por múltiplos

caminhos (*Multipath Fading*, em inglês) (Hermeto et al. 2017). Isto está exemplificado na figura 1, na qual o *link* do nó 1 para o nó 2 utiliza, no primeiro intervalo, o canal X e, posteriormente, o canal Y.

3.2. Codificação em Rede

Em uma rede de computadores, cada nó exerce o papel de retransmitir informação de uma conexão de entrada para um conjunto de conexões de saída. Do ponto de vista da teoria da informação, não existe razão para limitar o funcionamento de um nó somente para o comportamento descrito acima (Ahlswede et al. 2000). O conceito de Codificação em Rede (*Network Coding* em inglês) surge a partir desta visão, na qual os nós também são capazes de alterarem as informações que recebem em vez de simplesmente as "encaminhar para frente" de forma intacta.

Sob essa perspectiva, os nós passam a ser codificadores, recebendo informações dos múltiplos *links* de entrada, codificando essas mensagens juntas e, posteriormente, as propagando para as conexões de saída.

A figura 2 nos mostra um exemplo de funcionamento de uma rede. Neste exemplo, dois dispositivos estão tentando, cada um, enviar uma mensagem para o outro. Inicialmente, 2 dispositivos propagam suas respectivas mensagens M_1 e M_2. Posteriormente, um nó roteador necessita propagar as mensagens enviadas para que cheguem ao destino (*sink node*).

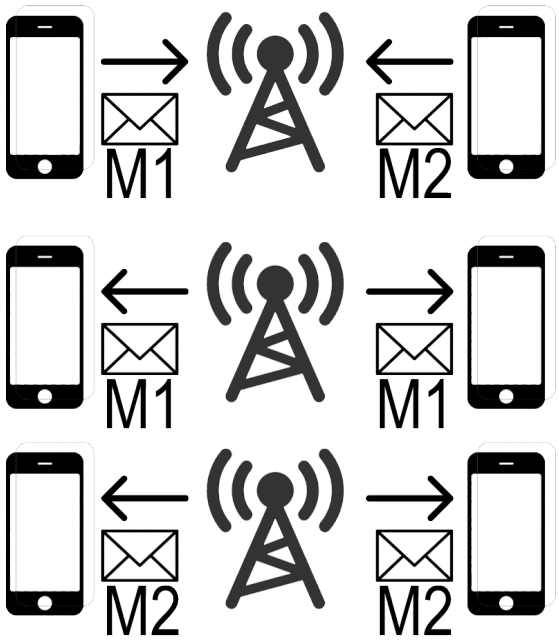

Figure 2. Rede sem *Network Coding*

Pela progressão do tempo (de cima para baixo), notamos que o roteador necessita propagar cada mensagem separadamente. Nota-se que, neste exemplo, ambos os dispositivos irão "escutar" mensagens que para eles eram desnecessárias de se receber (neste caso, a sua própria mensagem). Isto se deve à natureza dos meios de propagação compartilhados.

Já na figura 3 vemos que o nó de roteamento, ao escutar as duas mensagens, as combina através de uma função f. Esta função f é o que chamaremos de função de codificação, que dá a capacidade de codificar e decodificar mensagens.

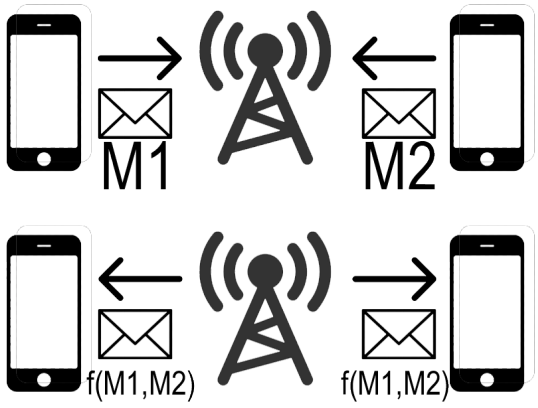

Figure 3. Rede com *Network Coding*

Portanto, f é tal que:

$$f(M_1, M_2) = X \tag{1}$$

$$f(M_1, X) = M_2 \tag{2}$$

$$f(M_2, X) = M_1 \tag{3}$$

Onde X é uma mensagem obtida após a codificação de outras 2.

Logo, na figura 3 temos que os dispositivos, ao receberem a mensagem combinada, são capazes de obter a mensagem de seu par através da decodificação. Por exemplo, o dispositivo que enviou a mensagem M_1 irá receber a mensagem $f(M_1, M_2)$ e por ter conhecimento da mensagem M_1, pois foi ele quem enviou, é capaz de executar $f(f(M_1, M_2), M_1) = f(X, M_1) = M_2$.

Esta técnica traz vários benefícios às redes que as implementam. Esta provou aumentar a vazão das redes, afinal, como visto anteriormente, em uma só transmissão *multicast* é possível carregar informação de mais de uma mensagem. Além disto, é capaz de reduzir o consumo de energia e aumentar a robustez da rede (Vieira and Vieira 2017).

3.2.1. Codificação Aleatória em Rede

A forma como os pacotes serão combinados, isto é, a definição concreta da função f (equação 1) é um fator relevante na forma como a Codificação em Rede será implementada.

Dentre as mais simples maneiras se encontra a Codificação Linear (*Linear Coding* em inglês). Esta solução encara o dado como um vetor (em uma dada base, como por exemplo binária) e permite que os nós apliquem transformações lineares (e.g., $f = a_1 * M_1 + a_2 * M_2$, onde a_1 e a_2 são coeficientes lineares e M_1 e M_2 são as mensagens) sob um vetor antes de transmiti-lo adiante (Li et al. 2003). Um exemplo claro e simples é a função XOR (ou exclusivo).

A Codificação Linear Aleatória (*Random Linear Coding* em inglês) é uma forma de Codificação em Rede. Nesta, os nós escolhem independentemente e de forma aleatória os coeficientes lineares. Desta maneira, para combinar mensagens, basta aplicar a função $f = a_1 * M_1 + a_2 * M_2$.

Note que neste caso será necessário enviar, juntamente com a mensagem codificada, os coeficientes que geraram essa mensagem. Isto incrementa o tamanho da mensagem, porém tende a se tornar insignificante mediante o crescimento do corpo da mesma. A figura 4 mostra como os coeficientes são concatenados à mensagem para posterior decodificação.

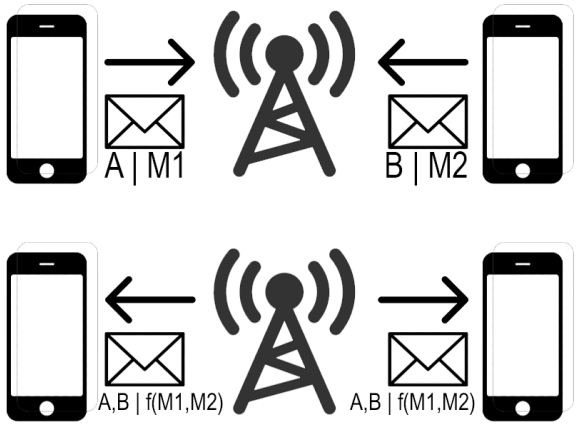

Figure 4. Rede com *Random Linear Network Coding*

Nesta figura podemos ver dois dispositivos enviando as mensagens M_1 e M_2. Porém, vemos que, diferente dos casos anteriores, é necessário enviar no corpo da mensagem um cabeçalho virtual (algo como um cabeçalho deste protocolo) que contém os pesos (coeficientes) de cada nó que já codificou sob a mensagem em questão. Desta maneira, as mensagens da parte superior da figura 4 seriam $Y_1 = A * M_1$ e $Y_2 = B * M_2$.

A mensagem codificada pelo nó intermediário é, portanto, $X = A * M_1 + B * M_2$ e leva consigo os coeficientes A e B para posterior decodificação. Os dispositivos, ao receberem a mensagem codificada, possuem, respectivamente, os seguintes sistemas para resolver:

$$\begin{cases} Y_1 = A * M_1 \\ X = A * M_1 + B * M_2 \end{cases} \tag{4}$$

$$\begin{cases} Y_2 = B * M_2 \\ X = A * M_1 + B * M_2 \end{cases} \tag{5}$$

Onde o sistema 4 é referente ao dispositivo da esquerda e o sistema 5 é referente ao da direita. Assim, podemos ver claramente que é possível decodificar a mensagem enviada pelo dispositivo par apenas resolvendo os sistemas obtidos.

3.3. Combinando técnicas

A integração do TSCH com técnicas de codificação em rede apresenta uma vantagem significativa em relação ao número de mensagens transmitidas na rede. Como pode ser observado nas figuras 5 e 6, essa combinação permite uma disseminação de dados mais eficiente, reduzindo a quantidade total de transmissões necessárias para alcançar todos os nós. Isto significa também uma possível redução na latência da transmissão dos dados.

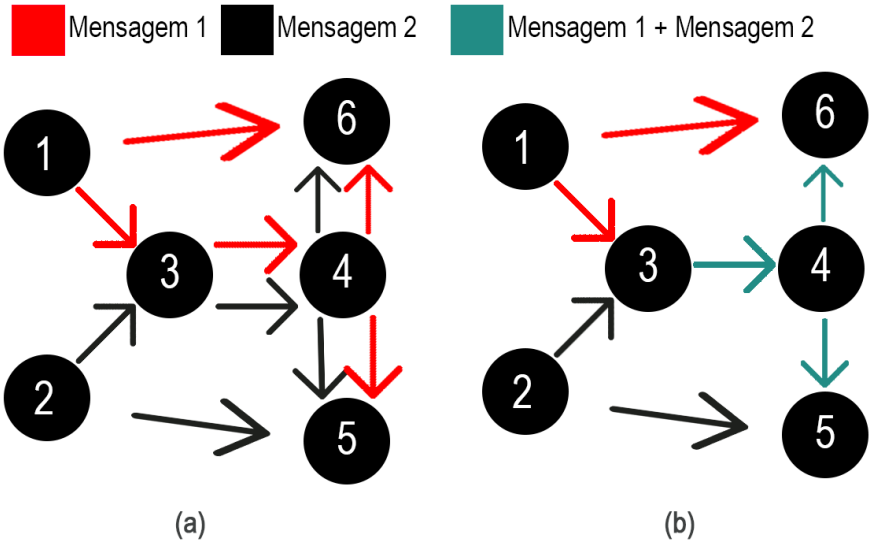

Figure 5. Tráfego de mensagens em rede TSCH sem (a) e com (b) *Network Coding*

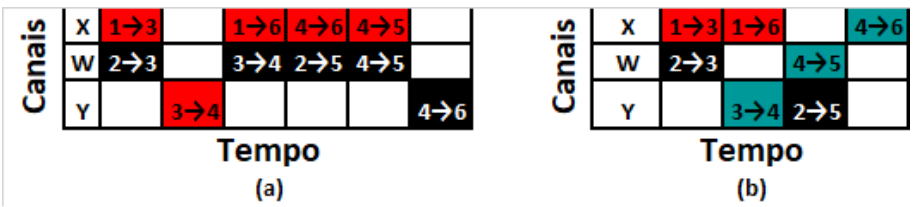

Figure 6. Mensagens trocadas em uma rede TSCH com (a) e sem (b) *Network Coding*

Isso ocorre porque a codificação em rede permite o envio de pacotes combinados que atendem múltiplos destinos simultaneamente, minimizando retransmissões e aproveitando melhor a capacidade do agendamento determinístico do TSCH.

Comparado ao estado da arte, que geralmente utiliza retransmissões simples ou flooding controlado, essa abordagem se mostra mais escalável e eficaz, especialmente em redes densas ou com restrições de energia.

4. Trabalhos Relacionados

Em relação a TSCH, (Hermeto et al. 2017) expuseram os prós e contras do escalonamento em saltos lentos de canal (*slow channel hopping*) na camada MAC (*Medium Access Control*, ou Controle de Acesso ao Meio em português) para redes industriais de baixa potência. Além disso, foram apresentados, sistematicamente, os diferentes algoritmos de escalonamento e suas características.

Além disso, os mesmos também destacam que requisitos de comunicação em tempo real, modelos de tráfego e características dos enlaces de comunicação são fatores a se considerar nos algoritmos de escalonamento de dispositivos IoT.

O TSCH também foi adaptado e avaliado no sistema operacional Contiki por (Duquennoy et al. 2017), que discutem os principais desafios e adaptações do protocolo 6TiSCH para o ambiente Contiki. Esse trabalho foca nas peculiaridades da integração de TSCH com o protocolo IPv6 em ambiente simulado, explorando aspectos de desempenho e avaliação em redes de sensores distribuídos.

Além de TSCH, a codificação em rede (Network Coding) tem se mostrado promissora para melhorar a disseminação de dados em redes sem fio. O trabalho seminal de (Li et al. 2003) introduziu o conceito de codificação em rede linear, enquanto (Ahlswede et al. 2000) estabeleceram as bases teóricas do fluxo de informação em redes. Estes estudos são fundamentais para entender a aplicação de técnicas de codificação em redes de comunicação.

No contexto de redes 5G e comunicação Device-to-Device (D2D), (Vieira and Vieira 2017) exploram o uso de codificação em rede, destacando os benefícios para o aumento da confiabilidade e eficiência. Em redes de sensores sem fio, (Júnior et al. 2017) introduziram o *CodeDrip*, um protocolo que melhora a disseminação de dados usando codificação em rede, resultando em maior robustez e eficiência.

Complementando os estudos sobre codificação em rede, trabalhos recentes destacam avanços na redução da complexidade computacional e na melhoria da decodificação progressiva, visando viabilizar seu uso em dispositivos com recursos limitados (Zhu et al. 2021). Além disso, análises de desempenho confirmam os benefícios da técnica em termos de latência, confiabilidade e vazão em redes sem fio (Rajan et al. 2023), reforçando seu potencial para aplicações futuras.

5. Implementação

5.1. Contiki-NG

O Contiki-NG é um sistema operacional de código aberto para dispositivos IoT, amplamente utilizado em ambientes de pesquisa e desenvolvimento devido à sua arquitetura modular e sua capacidade de executar protocolos de comunicação de baixa potência, ideal para dispositivos com restrições físicas de *hardware*, como memória na ordem de 100 kB (Oikonomou et al. 2022).

Para este trabalho, o Contiki-NG foi escolhido como base para o desenvolvimento e simulação do protocolo de rede utilizando o TSCH (que já está implementado no próprio sistema operacional) e técnicas de codificação de rede. A escolha deste sistema operacional se justifica por sua robustez e por permitir a customização e integração com diversas bibliotecas e funcionalidades, além de suportar a plataforma Cooja para simulação de cenários de redes complexas.

5.2. Cooja

O Cooja é um simulador de redes de sensores sem fio que se integra ao Contiki-NG, proporcionando um ambiente versátil para a emulação de diferentes arquiteturas de rede e a análise do desempenho de protocolos em um ambiente controlado.

No contexto deste trabalho, o Cooja foi utilizado para simular cenários de redes de sensores implementando TSCH e codificação de rede, possibilitando uma análise detalhada do comportamento da rede em termos de latência, perda de pacotes e consumo energético. Este simulador permite testar e validar algoritmos em uma variedade de dispositivos e topologias, o que contribui para a verificação da viabilidade e a eficácia das soluções propostas antes de uma eventual implantação em *hardware* real.

A utilização do Cooja como ferramenta de simulação neste estudo é crucial para a obtenção de resultados que podem guiar futuros desenvolvimentos e aprimoramentos nos protocolos de comunicação em redes de sensores sem fio antes de executar os testes em campo, que são muito mais caros e trabalhosos.

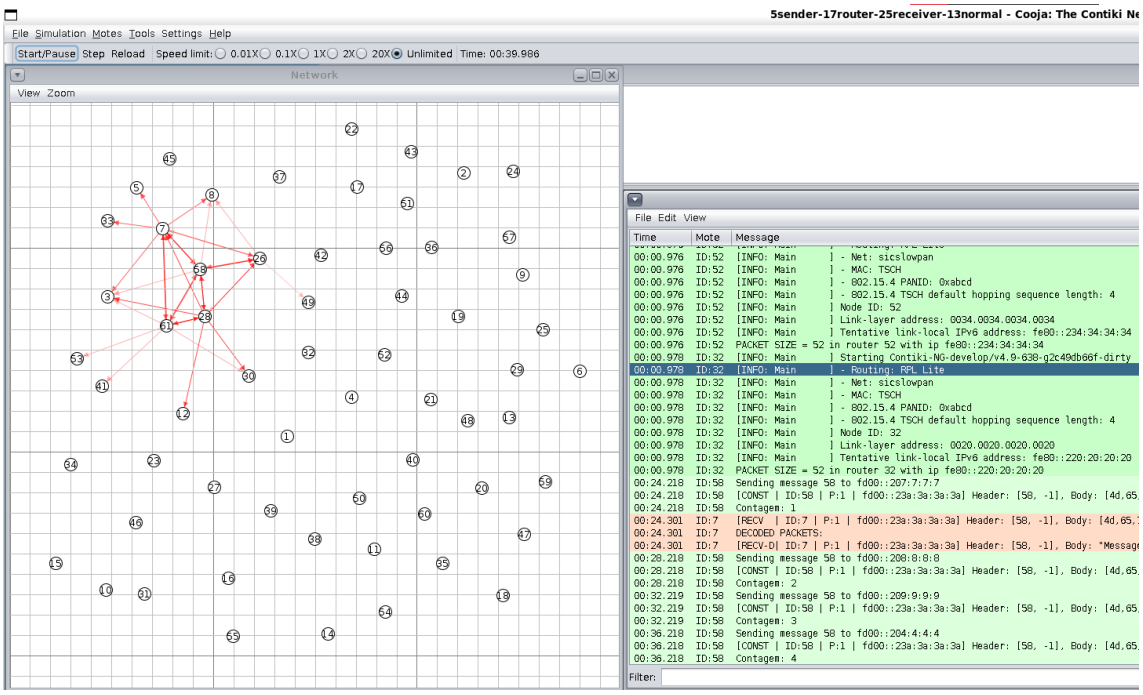

Figure 7. Interface do Simulador Cooja

Na figura 7 vemos a interface do Cooja. O bloco mais à esquerda é a àrea de trabalho onde a rede e seus nós são dispostos espacialmente. Cada esfera enumerada é um nó da rede e seu respectivo identificador.

As setas vermelhas indicam uma mensagem trafegando em um *link* de um nó de origem para um nó destino. Essas setas vão aparecendo e sumindo dinamicamente no decorrer da execução da simulação. À direita temos uma sessão de *log* onde o usuário consegue depurar a execução do protocolo através de registros feitos pelo mesmo.

Por fim, é importante notar que a rede forma um grafo conexo de nós. As setas representam também a existência de um enlace físico entre os mesmos. É possível dizer que cada uma destas arestas do grafo possui um peso, que é a probabilidade de recepção/transmissão de mensagens no enlace físico. Este parâmetro será utilizado durante os experimentos.

5.3. Codificação em rede

A implementação do *Network Coding* foi feita em cima do *Contiki-NG*, que já possui o TSCH embutido no mesmo e, portanto, não foi necessário implementar. É possível segmentar a implementação em 4 partes que coincidem com os 4 tipos de nós (ou funções) que podem compor uma rede que utiliza de *Network Coding*: expedidores (S), roteadores (RT), receptores (RX) e comuns (N).

Primeiramente será apresentada uma versão simplificada deste protocolo, que utiliza a função XOR como codificador/decodificador (função f). Antes de adentrar na implementação dos nós, discutamos sobre a unidade básica de informação neste protocolo: os pacotes.

5.3.1. Pacotes

Como discutido anteriormente, para que seja possível fazer a decomposição (equação 2) das mensagens já combinadas, é necessário saber quais foram as mensagens que compuseram a mesma. Para tal, neste protocolo cada mensagem será embutida de um pseudo-cabeçalho.

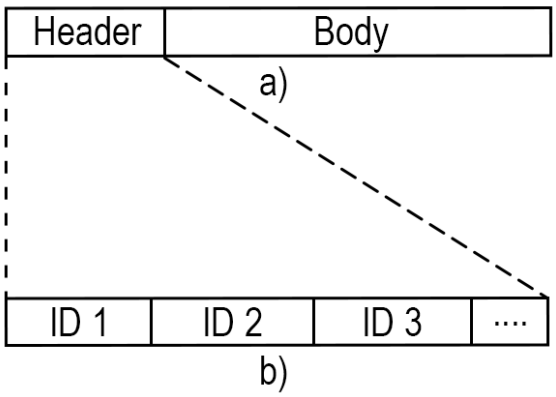

Figure 8. Composição interna de um pacote no protocolo *Network Coding*

Na figura 8 vemos como os pacotes neste protocolo são compostos internamente. Considerando um pacote como um vetor de *bits*, temos que este vetor é segmentado em duas partes, como é possível ver na figura 8-a. A segunda é o corpo (do inglês *body*),

que contém a mensagem transmitida propriamente dita após a aplicação da função de combinação f.

A primeira parte, o cabeçalho (do inglês *header*), é a responsável por carregar um identificador único que diz quais foram os pacotes que compõem esta mensagem. A partir de agora, para descrever um pacote usaremos a função $P(h)$, onde h é uma lista de identificadores das mensagens que compõem este. Para um caso onde h tem tamanho 3, $P(h)$ é tal que o conteúdo deste pacote é:

$$P(h) = f(f(B_{h_1}, B_{h_2}), B_{h_3}) \tag{6}$$

Onde h_i é o i-ésimo identificador único de mensagem na lista $h =< h_1, h_2, h_3 >$. Assim, B_{h_i} é o corpo da mensagem de identificador h_i.

Assim, uma mensagem $P(< A, B, C >)$ que trafega na rede é resultado da combinação das mensagens A, B e C e possui um corpo X:

$$X = f(f(B_A, B_B), B_C) \tag{7}$$

Já seu cabeçalho deve possuir os identificadores únicos das mensagens A, B e C. Assim, caso um nó receba a mensagem X e, posteriormente, receba as mensagens B e C, será capaz de extrair a mensagem A, como mostrado nas equações 1 a 3.

5.3.2. Nós Expedidores (tipo S)

Estes são os nós da rede responsáveis por gerar as mensagens originais e que, necessariamente, se destinam a um nó receptor. Uma mensagem original não é nada menos que $P(< i >)$, onde i é o identificador único dessa mensagem. Note que o corpo dessa mensagem também é aplicado sobre a função f. Para a função XOR, $f_{XOR}(A) = A$.

5.3.3. Comuns (tipo C)

Estes nós são responsáveis por não executar nenhuma ação sob os pacotes. Desta maneira, esses nós são responsáveis apenas por retransmitir as mensagens que recebem, atuando como pontos intermediários de retransmissão entre os expedidores e receptores.

5.3.4. Roteadores (tipo RT)

Os nós roteadores são aqueles responsáveis por interceptar as mensagens que trafegam na rede e aplicar as devidas codificações sob as mesmas. De forma simplificada, um nó roteador é capaz de fazer 3 coisas ao receber um pacote: armazená-lo para combinações futuras (não o propagando de volta na rede), combinando este com algum pacote previamente armazenado ou não executando nenhuma operação em cima do mesmo, apenas o propagando de volta à rede (atua apenas como um nó comum).

Perceba, portanto, que para codificar pacotes no futuro, um nó terá de armazenar um ”histórico” de mensagens. Com isto, este será capaz de recuperar, desse armazém de

pacotes anteriores, um que seja compatível para ser combinado com o pacote que está trafegando no dispositivo neste exato momento.

Para que um nó seja capaz de armazenar mensagens das quais passaram por ele, é necessário que o mesmo possua um *buffer* para os armazená-los. Este deve ter um tamanho limitado e reduzido, afinal o foco deste protocolo são dispositivos com recursos computacionais limitados. Assim, deve haver um balanço entre memória gasta e o quão curto é o "histórico" de mensagens deste nó.

Sempre que um nó que está atuando como roteador e recebe um pacote, ele executa o algoritmo 1.

Algorithm 1 Codifica pacote

input : O nó $node$ a executar a codificação
input : O pacote $packet$ a ser combinado com outro
output: O pacote resultante da codificação

```
1 if ShouldCombinePacket(node) then
2     packetToCombine ← GetPacketToCombine(node, packet)
3     if packetToCombine ≠ NULL then
4         return CombinePackets(packet, packetToCombine)
5     StorePacket(node, packet)
6     return NULL
7 return packet
```

Cada nó neste protocolo é atribuído com um valor PC (probabilidade de combinar). PC representa, de forma simplificada, qual a probabilidade de um nó codificar um pacote ao recebê-lo. Um nó comum, por exemplo, tem $PC = 0$, pois ele nunca codifica mensagens, retransmitindo de forma inalterada.

A função $ShouldCombinePacket$ no algoritmo 1 é a função responsável por gerar um número aleatório e compará-lo com PC. Assim, ao ajustar PC, esta função é capaz de aumentar ou diminuir a probabilidade de um nó roteador combinar ou não um pacote.

Algorithm 2 Combina pacotes

input : O pacote 1 $p1$ a ser combinado
input : O pacote 2 $p2$ a ser combinado
output: O pacote combinado

```
8  combined ← new packet()
9  combined.header ← p1.header · p2.header
10 combined.body ← f(p1.body, p2.body)
11 return combined
```

A função $CombinePackets$ no algoritmo 1 está apresentada no algoritmo 2. Nela vê-se que o cabeçalho do pacote codificado é um vetor com o identificador dos pacotes de cada pacote concatenados. O corpo é a aplicação da função f (e.g. $f_{XOR}(A, B) = A \oplus B$).

Já a função $StorePacket$ no algoritmo 1 é responsável por armazenar um pacote no *buffer* de mensagens do nó. A função $GetPacketToCombine$ acessa esse *buffer* e obtém um pacote compatível com $packet$. Isto é, que pode ser combinado com $packet$ de forma a gerar informação útil (e.g. $f_{XOR}(A, A) = 0$ não gera nenhuma informação a agregar na rede).

A definição se um pacote é compatível com outro varia de acordo com a definição da função f. Como visto, na função XOR gerar pacotes que tenham um número par de combinações de uma mesma mensagem não gera nenhuma informação útil e, portanto, não deve ser combinado.

5.3.5. Receptores (tipo R)

Os nós receptores são aqueles que irão receber e decodificar as mensagens. Note que estes nós podem funcionar como nós comuns (não executam codificação) quando alguma mensagem tem sua rota passando por este dispositivo.

Algorithm 3 Decodifica pacote

input : O nó $node$ a executar a decodificação
input : O pacote $packet$ a ser decodificado
output: A lista $decoded$ de pacotes decodificados

```
12 decoded ← new list()
13 packetsToDecode ← new list(packet)
14 StorePacket(node, packet)
15 while packetsToDecode has more packets do
16     for packetToDecode in packetsToDecode do
17         newDecoded ← DecodeOverStorage(node, packetToDecode, decoded)
18         remove packetToDecode from packetsToDecode
19         packetsToDecode ← packetsToDecode · newDecoded
20 return decoded
```

O algoritmo 3 é o responsável por receber um pacote e, dado o histórico de mensagens do nó receptor, gerar uma lista de pacotes que foram obtidos através da decodificação deste conjunto de objetos. Note que mesmo decodificando, o nó receptor armazena $packet$ em seu *buffer*, afinal é necessário gerar o histórico de dados para futuras decodificações.

O método $DecodeOverStorage$, que é onde maior parte da lógica de decodificação ocorre, está descrito no algoritmo 4.

Algorithm 4 Decodifica pacote

input : O nó $node$ a executar a decodificação
input : O pacote $packetToDecode$ a ser decodificado com outros pacotes no *buffer*
input : A lista $decoded$ de pacotes decodificados com informação de apenas 1 mensagem
output: A lista $rawDecoded$ de pacotes decodificados

```
21 rawDecoded ← new list()
22 for storedPacked in node.buffer do
23     decodedPacket ← resolvePackets(storedPacked, packetToDecode)
24     didInsert ← tryTemporarilyStorePacket(node, decodedPacket)
25     if didInsert then
26         rawDecoded ← rawDecoded · decodedPacket
27         if decodedPacket size is 1 then
28             decoded ← decoded · decodedPacket
29 return rawDecoded
```

Este método, para cada mensagem presente no *buffer* tenta decodificar um novo pacote. Caso o resultado da nova decodificação seja inédito (não foi gerado ainda), então este deve ser inserido em $rawDecoded$ para que mais decodificações sejam feitas em cima do mesmo. Caso esse pacote decodificado tenha informação de apenas 1 mensagem, então um pacote ”final” foi gerado.

O método $tryTemporarilyStorePacket$ é quem faz o controle de pacotes inéditos. Sempre que a decodificação for acontecer, o nó gera um espaço em memória de tamanho reduzido para armazenar alguns pacotes que o mesmo já decodificou, imitando o *buffer*. Ao fim do processo de decodificação este *buffer* temporário é liberado.

Isto se faz necessário para que o nó consiga manter um histórico gerado pela recursividade da decodificação. Assim, o processo de decodificação se torna capaz de gerar uma gama maior de pacotes decodificados, ao em vez de poder gerar apenas decodificações em cima dos pacotes que estão presentes no *buffer* original com o $packetToDecode$.

Em uma questão espacial, este *buffer* temporário não gera custos elevados, pois está na mesma ordem de grandeza do *buffer* original. Além disto, este só ocupa espaço durante o processo de decodificação, que é uma parcela reduzida do tempo operacional do dispositivo de rede.

O método $resolvePackets$ executa a lógica explicitada nas equações 1 a 3.

5.4. Codificação Linear Aleatória

A implementação definida anteriormente define a forma do *Network Coding*. Note que não foi especificado qual era a implementação da função f. O algoritmo ser genérico ao ponto que apenas pequenas partes têm de ser alteradas para mudar a função de codificação facilita na modularização e teste de diferentes codificadores.

Em uma primeira etapa, foi implementada a função $f_{XOR}(A, B) = A \oplus B$. Esta função, dado dois vetores de *bits*, aplica a operação XOR entre cada par de *bits* dos dois

vetores, como mostra a figura 9, onde a_i e b_i são *bits*, respectivamente, das mensagens A e B.

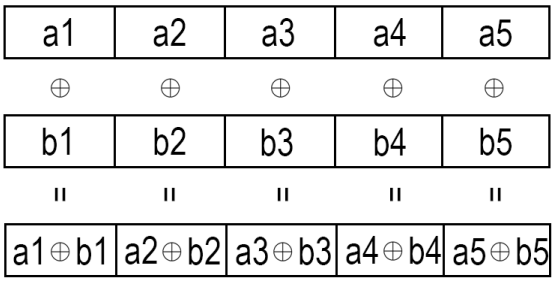

Figure 9. Aplicação da função f_{XOR} sob um vetor de *bits*

O que irá mudar na Codificação Linear Aleatória é como os vetores de bits são combinados. A figura 10 nos dá um exemplo de como f_{linear} opera. Para cada pacote, o corpo é multiplicado com o peso da mensagem original. No fim, os corpos são somados e os cabeçalhos das 2 mensagens originais passam a compor o cabeçalho do pacote combinado.

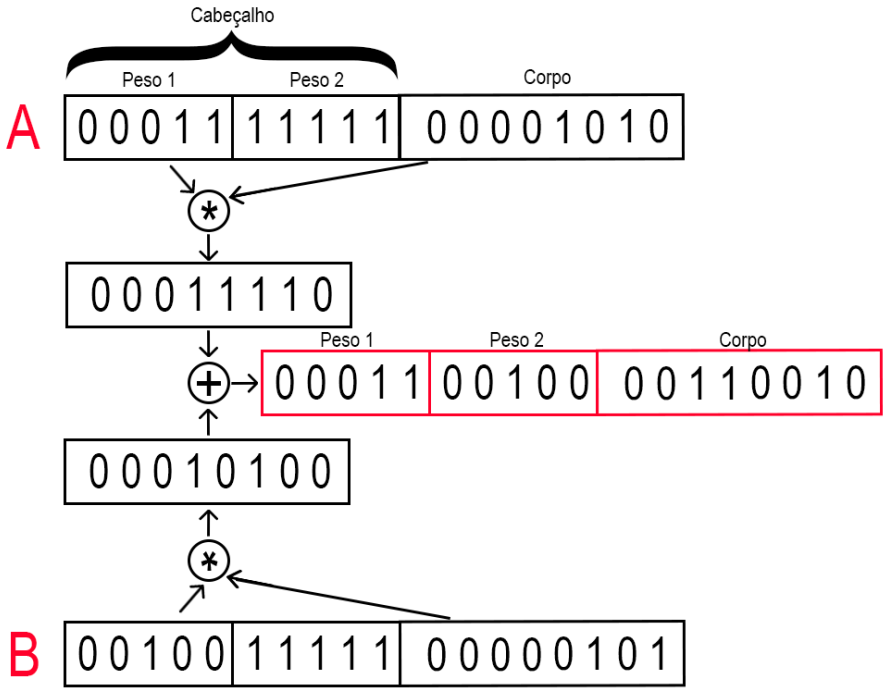

Figure 10. Aplicação da função f_{linear} sob dois pacotes

Como o cabeçalho dos pacotes é um vetor de pesos com tamanho pré-alocado, o protocolo precisa discernir uma posição preenchida com um peso de uma posição vaga. Para tal, é estabelecido um valor reservado para representar a ausência de um peso no cabeçalho. Na figura 10 vemos que o valor 11111 (maior valor para o número de *bits* usado para representar os pesos no cabeçalho) foi utilizado para representar uma posição vaga. Assim, o protocolo sabe que este pacote possui um espaço vago de E e pode ser

combinado com algum pacote que seja composto por até E outras mensagens. Neste caso, como o protocolo do exemplo permite combinações de no máximo 2 pacotes, E é sempre 0 ou 1.

6. Metodologia

Dado tais detalhes de implementação e uma robusta base teórica, é possível descrever a metodologia deste trabalho. Neste, o objetivo é implementar o protocolo apresentado e comparar o comportamento da aplicação de *Network Coding* em cima do *TSCH* em comparação a uma rede utilizando puramente *TSCH*.

6.1. Métricas

A base teórica apresentada anteriormente demonstra diversos benefícios do uso de *Network Coding*. Assim, a proposta é verificar, dentre esses benefícios, uma redução no tempo de propagação das mensagens na rede, também conhecida como latência ou tempo de disseminação. Essa redução indica que a rede consome menos *slots* de tempo, deixando a rede mais disponível para futuras comunicações, pois a disseminação de uma unidade de informação completa (i.e. uma mensagem final já decodificada, e não um pacote do protocolo) se dá de maneira mais rápida. Assim, avaliar a vazão de mensagens se torna uma métrica essencial.

Outro objetivo é observar o número de mensagens trocadas na rede durante o processo de comunicação. Com uma redução do número de mensagens, normalmente há uma redução no consumo de energia dos dispositivos, afinal, há um menor uso do rádio. Assim, isto servirá como uma métrica de eficiência.

Estes dois comportamentos, que podem estar correlacionados, serão observados em função de distintas variáveis. Assim, será variada a taxa de falhas nos enlaces físicos entre os nós (taxa de erro de transmissão/recepção), tal como o tamanho da rede (número de nós) para maior compreensão do comportamento prático das redes.

Neste artigo não se variou o tamanho do *buffer* de mensagens em cada nó. Esta decisão foi tomada em cima do resultado de que a máxima utilização do *buffer* (i.e. número de posições do mesmo que ficam ocupadas na média durante o tempo) não escala em função do tamanho da rede (Júnior et al. 2017).

6.2. Implementação

Em relação ao algoritmo de *Network Coding* utilizado, ambos os algoritmos apresentados foram implementados (f_{XOR} e f_{linear}). Apesar disso, os testes foram feitos todos sobre f_{linear}, pois é o algoritmo final desejado deste artigo. A função f_{XOR} foi implementada somente para comprovação teórica do funcionamento do protocolo e como uma etapa intermediária anterior à implementação da codificação linear aleatória.

É necessário ressaltar que para as simulações feitas no *Cooja*, as redes utilizaram todos os tipos de nós descritos na seção anterior. A disposição destes nós em um espaço bidimensional foi feita de forma aleatória. A única garantia feita é que exista um caminho entre todo e qualquer nó da rede, formando assim um grafo conexo. Além disto, foi utilizada a implementação de TSCH já fornecida pela plataforma.

Os testes sempre funcionarão da seguinte maneira: S nós expedidores irão enviar uma mensagem reservada para cada um dos R nós receptores. Caso o nó destino receba

a mensagem, o nó expedidor não a envia mais, pois seu objetivo foi cumprido. Um nó expedidor só para de enviar mensagens quando cada um dos R receptores receberem suas mensagens reservadas. Assim, os testes só terminam quando $R \times S$ mensagens forem recebidas.

É importante ressaltar que no percurso, que é aleatório devido à disposição aleatória dos nós, podem haver nós roteadores para executar a codificação de mensagens. Durante este processo, a perda de pacotes ou a propagação de mensagens modificadas são comportamentos normais e é sob estes efeitos que a rede será testada.

As taxas de perdas de pacotes nos enlaces físicos inserem no sistema um fator de aleatoriedade. Portanto, este fator aleatório obriga que, para uma mesma configuração e disposição da rede, mais de uma simulação seja feita, de tal forma que um comportamento possa ser observado em uma distribuição de simulações. Desta maneira, para cada experimento, 10 simulações serão executadas.

7. Resultados

No primeiro conjunto de testes, testamos o número de nós receptores (tipo R) que receberam todas as S mensagens destinadas a eles em função do tempo, comparando a rede sob o uso puro do TSCH em comparação a uma rede com TSCH e *Network Coding*.

É possível observar, na figura 11, o comportamento no tempo para uma rede de 40 nós (5 do tipo S, 10 do tipo RT, 15 do tipo R e 10 do tipo C) densamente povoada com 20% de chance de perda de um pacote em todos os enlaces.

A figura mostra que para os momentos iniciais, ambos protocolos se comportam de maneira similar. As diferenças aparecem após um certo tempo, onde a rede sem *Network Coding* passa a ter dificuldades de entregar o restante das mensagens reservadas (cerca de 70% das mesmas). Isso pode ser explicado pelas mensagens iniciais entregues terem sido as mensagens dos nós expedidores para os nós receptores mais próximos, sofrendo assim pouco impacto da taxa TX (i.e. taxa de transmissão) reduzida. Ao sobrarem apenas pares de combinações de nós $(expedidor, receptor)$ com um grande número de nós entre si, o efeito de TX aumentou o tempo de transmissão das mensagens reservadas.

Em contrapartida, para o *Network Coding*, foi possível observar que este efeito não ocorreu. Devido à população dos *buffers* com mensagens durante os momentos iniciais, o efeito da taxa TX foi reduzido, pois em caso de perda de pacotes, nós longínquos possuem tais mensagens já armazenadas, não fazendo necessária a retransmissão do mesmo. Isto reduz em muito o percurso que a mensagem tem que percorrer, reduzindo então a sua probabilidade de ser perdida nos meios de transmissão novamente. Portanto, é possível observar uma grande diferença na latência total da rede (i.e. tempo total para todos os receptores receberem todas as mensagens reservadas). Este fato também demonstra uma maior robustez da rede sob este protocolo.

No segundo teste foi explorado o tempo de transmissão (latência) necessário para o fim do teste (i.e. recebimento nos receptores de todas as S mensagens reservadas) em função do aumento do erro na taxa de transmissão em todos os enlaces físicos. De maneira similar, o terceiro teste analisa o número de mensagens trocadas em toda a rede (eficiência) em função do aumento da taxa de erro dos enlaces.

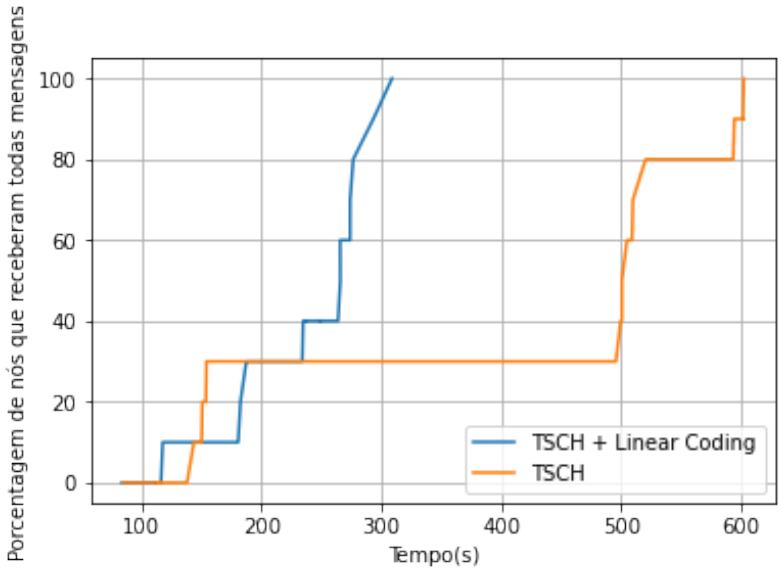

Figure 11. Latência na recepção de todas mensagens reservadas entre protocolos

Na figura 12 é possível ver o resultado do segundo experimento. Neste, foi utilizada uma rede com mesma configuração de nós do anterior. O teste demonstra que para taxas TX de perda nos enlaces, a rede com *Network Coding* possui menor latência total. Este fato é justificado pela mesma explicação do exemplo anterior. Com o aumento da taxa TX, mais difícil se faz para uma mensagem chegar a nós receptores longínquos. Assim, o armazenamento intermediário das mensagens dá maior robustez que evitam a retransmissão de mensagens, consequentemente reduzindo a latência de propagação.

Algo que pode ser notado é que em probabilidades TX baixas o uso de *Network Coding* deixa a rede com maior latência em relação ao TSCH puro. Isto comprova que a probabilidade de combinação de pacotes em redes com baixa perda de pacotes apenas atrasa a entrega dos pacotes, afinal a probabilidade de que os pacotes sejam perdidos nos enlaces físicos é baixa. Assim, é possível dizer que nestes casos a probabilidade de combinação de pacotes atua quase como uma taxa TX de perda de pacotes. É importante ressaltar que em redes reais dificilmente as perdas de pacotes são tão baixas assim (Aguayo et al. 2004).

Na figura 13 vemos o resultado do terceiro teste. Neste, usando a mesma configuração de nós anterior, foi variada a taxa TX e se observou o número total de mensagens trocadas na rede. É possível observar que com *Network Coding* para valores maiores de TX a rede transmite menos mensagens. Isto pode ser explicado pelos mesmos motivos apresentados nos 2 últimos experimentos mostrados. Desta maneira, um menor número de mensagens trocadas implica em um menor consumo de energia por parte dos dispositivos da rede.

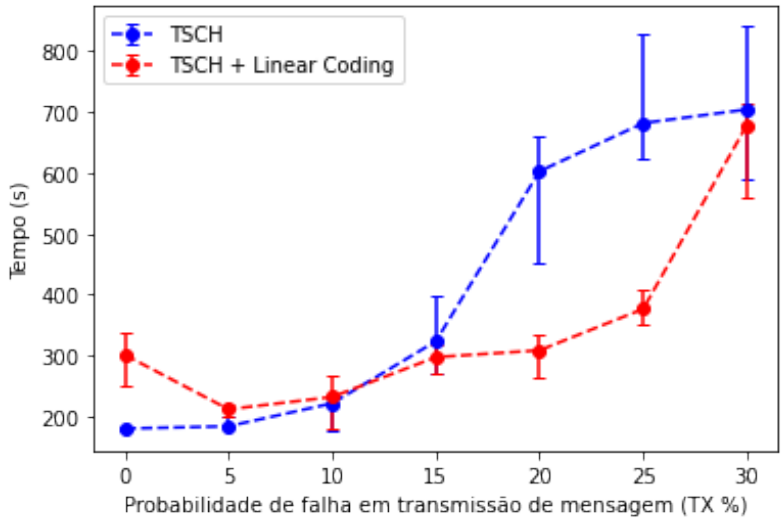

Figure 12. Latência total da rede em função da probabilidade de perda de pacotes

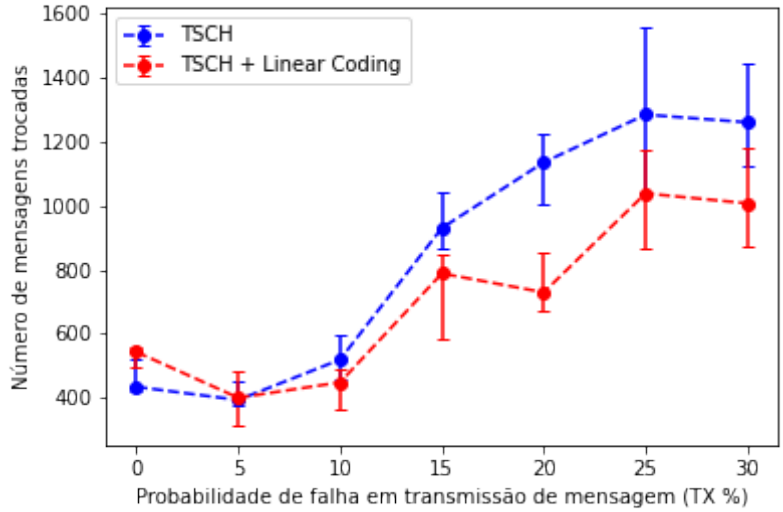

Figure 13. Número de mensagens trocadas na rede em função da probabilidade de perda de pacotes

Por fim, o quarto teste busca entender como a eficiência do protocolo reage ao aumento do número de nós da rede. É possível observar na figura 14 que em todos os casos a rede com *Network Coding* desempenha melhor que em comparação às topologias com somente TSCH. O gráfico nos mostra que um menor número de mensagens é trocado na rede, indicando maior eficiência energética da mesma. É necessário salientar que com o aumento do número de nós, a proporção entre os tipos de dispositivos foi mantida (i.e. proporção entre nós do tipo S, R, RT e C).

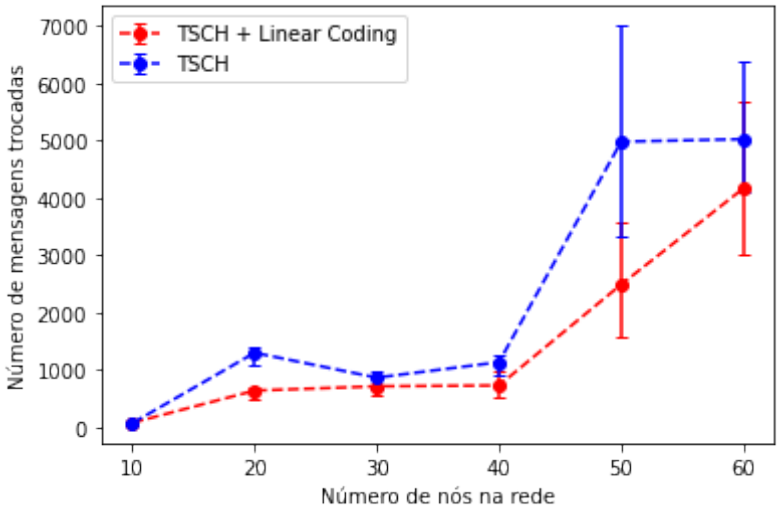

Figure 14. Número de mensagens trocadas na rede em função do tamanho da mesma

8. Conclusão

Este trabalho investigou a implementação e análise do impacto da Codificação Linear Aleatória em redes TSCH simuladas para dispositivos IoT, destacando as vantagens e limitações dessa abordagem em cenários simulados. Os resultados indicam que o uso combinado de TSCH e técnicas de codificação em rede apresenta benefícios significativos, como redução da latência e do consumo energético, particularmente em ambientes com elevadas taxas de perda de pacotes.

No entanto, é necessário salientar que as avaliações foram realizadas em um ambiente simulado. A transição para um ambiente físico, com as complexidades inerentes de hardware, interferências externas e comportamentos imprevisíveis, constitui um próximo passo essencial para validar a aplicabilidade prática e a robustez do protocolo proposto.

Por fim, este estudo contribui para a literatura ao demonstrar que a integração de TSCH com técnicas de codificação de rede pode viabilizar a criação de redes IoT mais eficientes e resilientes, estabelecendo um ponto de partida para futuras investigações que explorem a implementação em dispositivos reais e a otimização de algoritmos para contextos específicos.

References

[Aguayo et al. 2004] Aguayo, D., Bicket, J., Biswas, S., Judd, G., and Morris, R. (2004). Link-level measurements from an 802.11 b mesh network. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 121–132.

[Ahlswede et al. 2000] Ahlswede, R., Cai, N., Li, S.-Y., and Yeung, R. W. (2000). Network information flow. *IEEE Transactions on information theory*, 46(4):1204–1216.

[Duquennoy et al. 2017] Duquennoy, S., Elsts, A., Nahas, B. A., and Oikonomou, G. C. (2017). Tsch and 6tisch for contiki: Challenges, design and evaluation. *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 11–18.

[Hermeto et al. 2017] Hermeto, R. T., Gallais, A., and Theoleyre, F. (2017). Scheduling for ieee802. 15.4-tsch and slow channel hopping mac in low power industrial wireless networks: A survey. *Computer Communications*, 114:84–105.

[Júnior et al. 2017] Júnior, N. d. S. R., Tavares, R. C., Vieira, M. A., Vieira, L. F., and Gnawali, O. (2017). Codedrip: Improving data dissemination for wireless sensor networks with network coding. *Ad Hoc Networks*, 54:42–52.

[Li et al. 2003] Li, S.-Y., Yeung, R. W., and Cai, N. (2003). Linear network coding. *IEEE transactions on information theory*, 49(2):371–381.

[Oikonomou et al. 2022] Oikonomou, G., Duquennoy, S., Elsts, A., Eriksson, J., Tanaka, Y., and Tsiftes, N. (2022). The contiki-ng open source operating system for next generation iot devices. *SoftwareX*, 18:101089.

[Rajan et al. 2023] Rajan, V., Marimuthu, T., Londhe, G. V., and Logeshwaran, J. (2023). A comprehensive analysis of network coding for efficient wireless network communication. In *2023 IEEE 2nd International Conference on Industrial Electronics: Developments Applications (ICIDeA)*, pages 204–210.

[Vieira and Vieira 2017] Vieira, L. F. and Vieira, M. A. (2017). Network coding for 5g network and d2d communication. In *Proceedings of the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, pages 113–120.

[Zhu et al. 2021] Zhu, F., Zhang, C., Zheng, Z., and Farouk, A. (2021). Practical network coding technologies and softwarization in wireless networks. *IEEE Internet of Things Journal*, 8(7):5211–5218.