DIOGO FERREIRA LIMA FILHO

PROJETO E IMPLEMENTAÇÃO DE UM NOVO ALGORITMO E PROTOCOLO DE ENCAMINHAMENTO DE PACOTES BASEADO EM CÓDIGOS CONVOLUCIONAIS USANDO TCNet: TRELLIS CODED NETWORK

São Paulo
2015

DIOGO FERREIRA LIMA FILHO

PROJETO E IMPLEMENTAÇÃO DE UM NOVO ALGORITMO E PROTOCOLO DE ENCAMINHAMENTO DE PACOTES BASEADO EM CÓDIGOS CONVOLUCIONAIS USANDO TCNet: TRELLIS CODED NETWORK

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Ciências

São Paulo
2015

DIOGO FERREIRA LIMA FILHO

PROJETO E IMPLEMENTAÇÃO DE UM NOVO ALGORITMO E PROTOCOLO DE ENCAMINHAMENTO DE PACOTES BASEADO EM CÓDIGOS CONVOLUCIONAIS USANDO TCNet: TRELLIS CODED NETWORK

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Ciências

Área de concentração:
Sistemas Eletrônicos

Orientador:
Prof. Dr. José Roberto de Almeida Amazonas

São Paulo
2015

Dedicatória

Aos meus pais, *in memoriam,* Diogo e Elza.

À minha esposa Rose e filhos,
Vinicius, Patrícia e Júlia.

Agradecimentos

Ao Prof. Dr. José Roberto de Almeida Amazonas por sua orientação, paciência e acima de tudo confiança;

Aos Profs. Dr. Reginaldo Palazzo Jr. e Dr. Alexandre B. de Lima pelas sugestões e recomendações;

Ao Prof Dr. Marlim P. Menezes peja ajuda na validação prática dessa pesquisa;

Ao aluno de graduação Mateus Oliveira Françani, com o auxílio da Bolsa financiada pela Associação dos Engenheiros Politécnicos – USP, pelo apoio nas simulações teóricas;

À Ana Maria Badiali pelo auxílio na revisão do texto.

Em especial à minha esposa Rose e à filha Júlia por terem entendido minha ausência nas muitas horas dedicadas a essa pesquisa.

Resumo

Os Wireless sensor networks (WSNs) evoluíram a partir da idéia de que sensores sem fio podem ser utilizados para coletar informações de ambientes nas mais diversas situações. Os primeiros trabalhos sobre WSNs foram desenvolvidos pelo *Defense Advanced Research Projects Agency* (DARPA)[1], com o conceito de *Smart Dust* baseados em *microelectromechanical systems* (MEMS), dispositivos com capacidades de detectar luminosidade, temperatura, vibração, magnetismo ou elementos químicos, com processamento embarcado e capaz de transmitir dados via *wireless*. Atualmente tecnologias emergentes têm aproveitado a possibilidade de comunicação com a *World Wide Web* para ampliar o "*rol*" de aplicações desta tecnologia, dentre elas a Internet das Coisas (*I*nternet of Things) – IoT.

Esta pesquisa estuda a implementação de um novo algoritmo e protocolo que possibilita o encaminhamento dos dados coletados nos microsensores em cenários de redes *ad hoc* com os sensores distribuídos aleatoriamente, em uma área adversa.

Apesar de terem sido desenvolvidos vários dispositivos de hardware pela comunidade de pesquisa sobre WSN, existe um esforço liderado pela *Internet Engineering Task Force* (IETF)[2], na implementação e padronização de protocolos que atendam a estes mecanismos, com limitações de recursos em energia e processamento. Este trabalho propõe a implementação de novos algoritmos de encaminhamento de pacotes utilizando o conceito de códigos convolucionais. Os resultados obtidos por meio de extensivas simulações mostram ganhos em termos da redução de latência e do consumo de energia em relação ao protocolo AODV. A complexidade de implementação é extremamente baixa e compatível com os poucos recursos de hardware dos elementos que usualmente compõem uma rede de sensores sem fio (WSN). Na seção de trabalhos futuros é indicado um extenso conjunto de aplicações em que os conceitos desenvolvidos podem ser aplicados.

Palavras-Chaves: Rede de sensores. Códigos convolucionais. Wireless. Decodificador de treliça. Protocolos. Redes codificadas.

[1] Agência do Departamento de Defesa Americano, responsável pelo desenvolvimento de novas tecnologias para uso militar.

[2] É uma organização que estabelece padrões para Internet, sem a adesão formal. Todos os participantes são voluntários e gestores, embora seu trabalho normalmente é financiado por seus empregadores ou patrocinadores.

Abstract

Wireless sensor networks (WSNs) have evolved from the idea that small wireless sensors can be used to collect information from the physical environment in a large number of situations. Early work in WSNs were developed by *Defense Advanced Research Projects Agency* (DARPA)[1], so called *Smart Dust*, based on *microelectromechanical systems* (MEMS), devices able to detect light, temperature, vibration, magnetism or chemicals, with embedded processing and capable of transmitting wireless data. Currently emerging technologies have taken advantage of the possibility of communication with the *World Wide Web* to expand to all applications of this technology, among them the Internet of Things – IoT.

This research, studies to implement a new algorithm and protocol that allows routing of data collected in micro sensors in ad hoc networks scenarios with randomly distributed sensors in adverse areas.

Although they were developed several hardware devices by the research community on WSN, there is an effort led by *Internet Engineering Task Force* (IETF)[2], in the implementation and standardization of protocols that meet these mechanisms, with limited energy and processing resources. This work proposes the implementation of new packets forwarding algorithms using the concept of convolutional codes. The results obtained by means of extensive simulations show gains in terms of latency and energy consumption reduction compared to the AODV protocol. The implementation complexity is extremely low and compatible with the few hardware resources usually available in the elements of a wireless sensor network (WSN). In the future works section a large set of applications for which the developed concepts can be applied is indicated.

Keywords: Sensor Network. Convolutional codes. Wireless. Trellis Decoder. Protocols. Encoded networks.

[1]Agency of the United States Department of Defense responsible for the development of new technologies for use by the military.

[2]It is an organization that sets standards for the Internet, without formal membership. All participants are volunteers and managers, although his work is usually funded by their employers or sponsors.

Lista de Figuras

Lista de Tabelas

LISTA DE ABREVIATURAS E SIGLAS

ACK	Acknowledgement
AODV	Ad Hoc On Demand Distance Vector
Bcast ID	Identificador de Broadcast
DSDV	Destination Sequenced Distance Vector Routing
DSR	Dynamic Source Routing
dHam	distância de Hamming
dE	distância Euclidiana
dLog	distância Lógica
Dest ID	Identificador de Destino
Dest Seq Num	Número de Sequência do destino
DSR	Dynamic Source Routing
Fonte ID	Identificador do nó Fonte
FSM	Finite State Machine
IETF	Internet Engeneering Task Force
IoT	Internet of Things
LLNs	Low power and Lossy Networks
Λ	Lattice
ML	Maximum Likelihood
MM	Máquina de Mealy
QoS	Qualidade de Serviço
ROLL	Routing Over Low Power Lossy Networks
RREQ	Route Request

RREP	Route Reply
Rx	Receptor
Sec Seq Num	Número de sequência da fonte
TTL	Tempo de vida da mensagem
Tx	Transmissor

Sumário

1 INTRODUÇÃO

Apresenta um cenário de redes de sensores sem fio, com enfoque nas limitações dos protocolos de roteamento que motivaram esta pesquisa, seus objetivos, metodologia de desenvolvimento do trabalho e a organização do texto.

1.1 CONTEXTO E MOTIVAÇÃO

As redes de sensores sem fio (*Wireless Sensor Networks*) – WSNs [2], [3], têm despertado grande interesse na última década. A ideia do emprego de dispositivos com limitados recursos de processamento, comunicação e energia, embarcados, na coleta de informações do ambiente físico e a possibilidade de comunicação com o mundo virtual utilizando a *World Wide Web* tem inspirado o surgimento de tecnologias emergentes, dentre elas a Internet das Coisas (*Internet of Things*) – IoT [4].

As várias aplicações das redes de sensores sem fio - WSNs, constituem uma importante infraestrutura para a Internet das Coisas, mas esbarra no problema de implementação de bons protocolos de roteamento que atendam às limitações dos dispositivos que constituem as redes de sensores, tornando-se a questão de grande interesse na discussão desta pesquisa.

As WSNs configuram um cenário de redes ad hoc, onde os nós assumem o papel de roteadores devido à falta de estrutura e a característica adversa da qualidade da transmissão. Além disso, a dificuldade dos enlaces em cobrir grandes áreas, sugere uma administração distribuída dos recursos com protocolos inteligentes. Os dispositivos *ad hoc* também estão submetidos a danos que podem torná-los inoperantes. Por tais motivos, é necessário que as redes *ad hoc* sejam tolerantes a falhas de modo que parâmetros importantes como latência, perda de pacotes, vazão e consumo de energia não sejam drasticamente afetados.

Um sistema tolerante a falhas deve manter um desempenho adequado em relação a latência e ainda manter a conectividade entre os nós no caso da perda de um ou mais caminhos entre o nó falho e os seus vizinhos. As mensagens de atualização dos *paths* não devem sobrecarregar o sistema, devendo haver uma solução distribuída com o balanceamento dessas informações.

Os vários protocolos de roteamento utilizados atualmente são classificados como: "*estado de enlace*" e "*vetor de distância*" [5], baseados em tabelas de rotas. A necessidade de um elevado tráfego

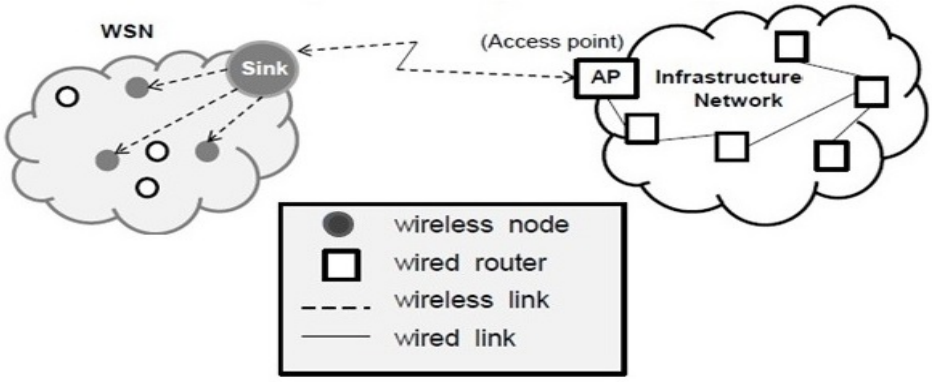
de sinalização para gerar e atualizar as tabelas de rotas inerentes a esses protocolos é proibitivo para uma rede em que os nós roteadores têm poucos recursos. Para identificação das rotas, no caso os protocolos utilizados nas *Low-power and Lossy Networks* (LLNs) [1] adotam soluções baseadas no tráfego de sinalizações do tipo *route request* (RREQ) e *route reply* (RReply) [7], causando aumento da latência. Esse trabalho apresenta uma proposta de modo a contribuir com a redução da latência, melhoria do consumo de energia e robustez no caso de ocorrência de falhas.

A característica aberta do padrão IP incentivou nos últimos 15 anos o surgimento de soluções de arquiteturas e protocolos proprietários aplicados às WSNs sem os cuidados de interoperabilidade entre os dispositivos, o que motivou o *Internet Engineering Task Force* (IETF) através do grupo de trabalho denominado *Routing Over Low-Power and Lossy Networks* (ROLL) [6], a estabelecer padrões e especificar protocolos IP para redes de dispositivos LLNs.

1.2 OBJETIVO

O objetivo desta pesquisa é propor um algoritmo e protocolo de encaminhamento de pacotes que possa ser utilizado pelas WSNs, onde uma das possibilidades é permitir o acesso às redes com infraestrutura IP, conforme mostrado na Figura 1.1, onde o nó SINK assume a responsabilidade de *gateway* da rede, possibilitando a adaptação dos FRAMES das redes WSNs com as redes IP.

Figura 1.1: Cenário de rede de sensores (WSN) no mesmo universo de redes com protocolo IP [1]



Fonte: autor

O fato dos dispositivos *ad hoc* possuírem grande mobilidade faz com que sejam considerados os casos extremos de adaptação e reconfiguração das rotas. Nos casos apresentados neste trabalho, apesar de ser considerada a aleatoriedade da rede quanto ao posicionamento físico dos nós, não foram

[1] Termo utilizado pelo Grupo de trabalho ROLL para se referir a redes de dispositivos com limitados recursos em CPU, memória e energia [6].

testados exaustivamente quanto à mobilidade e seus efeitos nos enlaces, o que deve ser considerado em trabalhos futuros [8], [9].

Para atingir os objetivos deste trabalho, foram feitas análises dos protocolos de roteamento disponíveis para as redes de sensores sem fio, identificação das limitações e proposta de um novo protocolo de encaminhamento que ofereça as seguintes vantagens:

- Eliminação da tabela de rotas;
- Redução da latência provocada pelos pacotes de sinalização, usualmente empregados nos protocolos tradicionais;
- Possibilidade de seguir rotas obtidas a partir de *algorithms for multirestictive routing*, [10];
- Implementação de protocolos que sejam robustos na presença de falhas na rede.

Para atender estes objetivos a proposta desta tese é fazer uma associação entre os estados de uma Máquina de Estados Finitos (*Finite State Machine*) - FSM [11], utilizada nos códigos convolucionais e os nós de uma rede em que a trajetória seja representada através dos ramos do diagrama de uma treliça, de modo que possa ser identificada como uma rota através da rede de sensores sem fio. Para isso, o algoritmo desenvolvido está baseado na teoria de decodificação do algoritmo de Viterbi [12].

As WSNs que venham utilizar estes protocolos são designadas neste trabalho como TCNet - (*Trellis Coded Network*).

1.3 METODOLOGIA DE DESENVOLVIMENTO DO TRABALHO

A hipótese científica deste trabalho consiste em adotar uma analogia entre uma rede e a sequência de estados percorrido por um código convolucional em um diagrama de estados, sendo possível desenvolver algoritmos de encaminhamento de pacotes que ofereçam características interessantes em termos de baixa latência, baixo consumo de energia e robustez na ocorrência de falhas.

A metodologia de trabalho consiste:

- Estudo dos protocolos hoje adotados nas redes de sensores sem fio;
- Desenvolvimento da analogia entre uma rede e a sequência de estados percorrido por um código convolucional em um diagrama de estados;
- Validação por meio de simulações;
- Proposta de campos de aplicação.

1.4 ORGANIZAÇÃO DO TEXTO

O Capítulo introdutório apresenta o contexto e motivação para identificação de protocolos que atendam aos limitados recursos das WSNs e os objetivos desta pesquisa.

O Capítulo 2 apresenta uma revisão teórica dos tópicos dividido nas Subseções: protocolos de roteamento existentes para as redes WSNs, suas vantagens e desvantagens; teoria dos códigos convolucionais, representação em diagrama de estados, representação em diagrama de treliça; distância lógica usando os conceitos de *lattice*[2].

O Capítulo 3 apresenta a proposta de associar os estados de um código convolucional aos nós de uma rede e como implementar um protocolo de encaminhamento de pacotes que não dependa de tabelas de rotas. Mostra que percorrer o diagrama de treliça corresponde a percorrer uma rota em uma rede, podendo-se usar o algoritmo de Viterbi para fazer o roteamento. Mostra o comportamento da análise da treliça para atender um aumento de concorrência dos nós na rede.

O Capítulo 4 apresenta os resultados das simulações e resultados experimentais divididos nas seguintes situações: cenários de simulação de latência e energia consumida pelo algoritmo TCNet durante um *query*; análise de desempenho do algoritmo TCNet em relação ao algoritmo AODV [14]; demonstra a robustez do algoritmo TCNet na presença de falha de nós e colisão de pacotes e cenários de potencial aplicação do algoritmo TCNet.

O Capítulo 5 apresenta as conclusões e os trabalhos futuros.

[2]Princípio de *Set Partitioning* usado em *trellis coded* para definir distâncias na treliça [13]

2 REVISÃO TEÓRICA

O Capítulo apresenta uma revisão teórica resumida dos principais tópicos da pesquisa, com o objetivo de realizar comparações com os mecanismos de roteamentos existentes para as redes WSNs.

2.1 PROTOCOLOS DE ROTEAMENTO EXISTENTES PARA REDES WSNs

As constantes pesquisas para se obter protocolos que atendam às redes WSNs são desafios devido às características dinâmicas dessas redes [9], [15], [16]. As tentativas de se adaptar os protocolos de roteamento das redes com infraestrutura [17], [18], [19], aos casos de redes *ad hoc*, são muitas vezes inconsistentes para atenderem aos aspectos como: frequentes alterações de topologias, baixa qualidade dos enlaces, largura de banda restrita e restrições das fontes de energia.

Em [20], [21] e [22] são apresentados algoritmos denominados *Geometric routing* usando as coordenadas geográficas dos nós. Esses algoritmos inicialmente são atrativos por utilizarem *distâncias Euclidianas* para decidir a posição dos nós vizinhos, porem tornam-se inviáveis devido às características dos sistemas GPS serem sensíveis às constantes obstruções.

Em [23] é proposta a utilização de coordenadas virtuais, baseado em *distâncias Euclidianas* e *Hamiltonian path* [24], não sendo conhecidos resultados teóricos com a aplicações desse conceito.

Em [25] foram apresentados novos fundamentos às aplicações de *Geometric routing*, baseando-se em *espaços hiperbólicos* [26], na tentativa criar novas soluções para as rotas dinâmicas.

Em [27] foi apresentada uma técnica utilizando *clustered WSNs* como uma solução para gerenciar a escalabilidade e latência da rede. Cada *cluster* sendo gerenciado por um *gateway node* (GN), interligado aos demais GN, de modo a permitir rotas: fonte – GN – GN – destino.

Em [28] é utilizado um protocolo de roteamento *multipath*, assistido por *hotline* para atender às solicitações prioritárias. O cálculo do trajeto considera o número de *hops end to end*, latência e mínima energia do nó.

Em [29] é analizada a melhor rota numa determinada topologia considerando o efeito da potência mínima de transmissão na taxa de erro (BER) em canais *multipath Rayleigh fading*.

Este trabalho, buscando encontrar nova proposta de protocolo que atendesse às redes *ad hoc* aplicados a WSNs, apresenta um conceito baseado em códigos convolucionais, demonstrado através de análise de eficiência comparado a um protocolo de rota escolhido (AODV), por ser o mais utilizado atualmente, levando em consideração às seguintes características: menor latência, otimização dos custos das métricas e robustez na presença de falhas. Para atender a essas premissas, os protocolos foram identificados e classificados levando em conta as seguintes considerações [7]:

- Mecanismos de atualização das informações das rotas:
  - **Pró-ativos**, baseados em tabelas com as informações da topologia da rede, onde estão relacionadas as ligações entre os nós da rede. Essas informações são atualizadas por um processo de difusão através da rede. Sempre que um nó necessitar encontrar um caminho até o destino, a tabela será consultada e um protocolo de "*estado de enlace*" (algoritmo de Dijkstra) é utilizado [30], [5], [31];
  - **Reativos**, não possuem o conhecimento global da rede, são descentralizados e a única informação que um nó possui são os custos dos enlaces com os nós vizinhos. O caminho de menor custo é encontrado de modo interativo e distribuído, onde cada nó pesquisa a rota sabendo apenas os custos dos enlaces diretamente ligados a ele utilizando o algoritmo do tipo "*vetor de distância*" (algoritmo de Bellman – Ford) [5]. As tabelas são atualizadas à medida que o nó em questão recebe nova informação sendo distribuída no modo de difusão pela rede.

- Baseados em informações passadas e futuras das rotas:
  - São protocolos que não possuem o conhecimento global da rede, baseiam-se em informações temporais dos estados passados e futuros da rede de modo a tomarem decisões sobre o roteamento [5];

- Baseados em recursos de QoS :
  - São protocolos que utilizam cenários de recursos disponíveis da rede como: energia (*power aware*), largura de banda disponível, escalabilidade. Esses parâmetros configuram as características dos protocolos aplicados [5].

A Tabela 2.1 mostra os protocolos de roteamento estudados, apresentando suas limitações que procurarão ser superadas pela técnica proposta neste trabalho. A tabela não é exaustiva, mas é representativa dos protocolos existentes que podem ser usados nas WSNs.

Analisando-se as comparações da Tabela 2.1, nota-se a necessidade de um algoritmo que atenda os parâmetros da dinâmica das WSNs de um modo global. As propostas apresentadas procuram adaptar os conceitos de *estado de enlace* e *vetor de distância* empregados nas redes fixas para serem utilizados nas redes *ad hoc*.

Entre os protocolos apresentados na Tabela 2.1, os protocolos OLSR e DSDV não apresentam características relevantes, por serem projetados para redes fixas, não possuindo as características di-

nâmicas que se procura para as redes *ad hoc*. Os protocolos DSR e AODV apresentam as seguintes características dinâmicas procuradas para comparações de eficiência:

- Não necessita conhecer previamente uma rota;
- Utiliza a sinalização como negociação para descobrir uma rota;
- São protocolos reativos (*on demand*);

Apesar das semelhanças entre o DSR e o AODV, este trabalho decidiu usar o AODV para comparações, por ser o mais usado nas redes WSNs.

Tabela 2.1: Classificação dos protocolos de roteamento

Atualização da rede		Usa informação temporal	Sensível a QoS	Característica
Pró-ativo	**Reativo**	-	-	-
OLSR [32]	-	Atualiza a tabela de modo otimizado	Otimiza a difusão dos pacotes utilizando o mecanismo *multipoint relaying*	Projetado para redes fixas, seu emprego nas redes ad hoc é motivado pela redução dos mecanismos de controle evitando congestionamento da rede e redução da latência
DSDV [33]	-	Decide baseado em informações de estados futuros obtido pela atualização periódica da tabela	não	Projetado para redes fixas, possui elevada latência que aumenta com o crescimento do número de nós
	DSR [34]	Decide a rota baseado em RREQ e RReply pelos nós vizinhos. Tempo de vida do RREQ $<$ 10 *hops*	não	Estabelece a rota propagando um RREQ e aguardando um RReply do nó destino. Não possui alternativas para perda de enlace. Possui elevada latência causada pelos protocolos de sinalização. O pacote de dados especifica todo o percurso da rota. Capacidade reduzida de nós
	AODV [14]	Atualiza periodicamente os enlaces através da variável *RouteError*	não	Usa pacote de dados com as variáveis (*SrcSeqNum*, *DestSeqNum*, *SrcID*, *DestID*, *BcastID*, *time to live*) para estabelecer a rota. Capacidade $>$ 200 nós

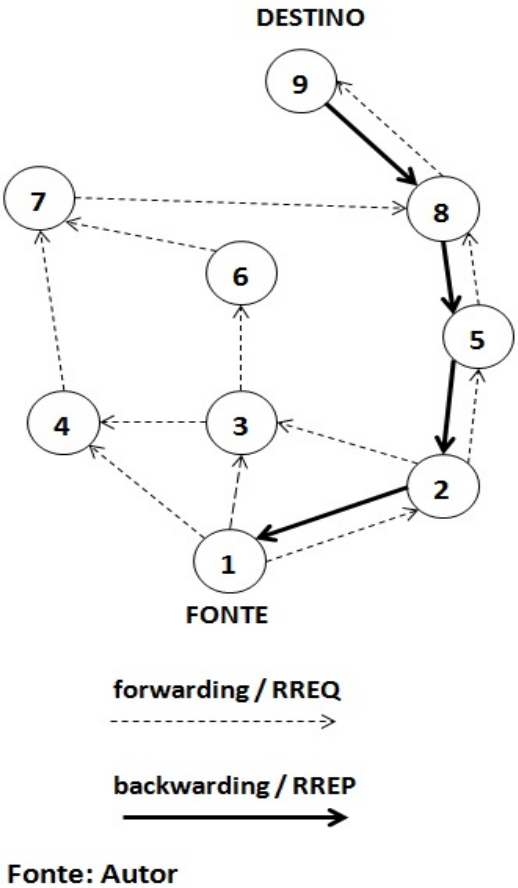
2.1.1 Cenário de funcionamento do AODV

Tendo sido o AODV o protocolo escolhido para fazer as comparações com o TCNet, esta Seção faz uma descrição de suas principais características.

A Figura 2.1 mostra um cenário resumido do funcionamento do AODV, onde o nó *FONTE* e os nós intermediários estabelecem comunicação com os nós vizinhos através de HELLOs (periódicos), de modo a manter uma tabela de rotas atualizada, [14], [35], [7]. O mecanismo de descoberta de rota é iniciado pelo nó *FONTE* emitindo um *broadcast* com mensagem de *Route Request* (RREQ), desencadeando um *rebroadcasting* pelos demais nós da rede. A mensagem RREQ é configurada com os campos contendo as seguintes informações:

- Identificação da fonte (SrcID);
- Identificação do destino (DestID);
- Sequência com os dados da fonte (SrcSeqNum);
- Tempo de vida da solicitação (TTL)

Figura 2.1: Cenário de funcionamento do protocolo AODV



No exemplo da Figura 2.1, o nó 1 (FONTE) deseja enviar uma mensagem para o nó 9 (DESTINO). O nó 1 (FONTE) envia um RREQ que é recebido pelos nós 2, 3 e 4, e pelos demais nós através

de *rebroadcasting* (linha tracejada). O cenário mostra a atuação dos nós vizinhos ao nó 1 (FONTE) recebendo o RREQ e tomando as decisões:

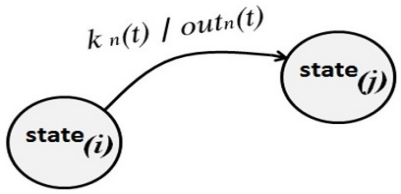
- Sendo o nó 9 (DESTINO), este envia um Route Reply (RREP) (linha cheia);
- No caso do nó 8 que recebe respostas do nó 7 e do nó 9, será considerada para o cenário, a resposta do nó 9 por ter sido recebida antes, portanto a rota que é estabelecida é: 9, 8, 5, 2, 1, sendo desprezada a resposta do nó 7;
- A mensagem continua sendo *rebroadcasting* até expirar o tempo útil (TTL)

2.2 ESTRUTURA DOS CÓDIGOS CONVOLUCIONAIS

O conceito de código convolucional foi apresentado por Elias [36] em 1955, como uma alternativa para transmissão da informação em canais adversos, com aplicação nas últimas quatro décadas em modulação e teoria de códigos. Neste trabalho foi utilizada a estrutura de grafo dos códigos convolucionais para modelar redes de sensores.

Os códigos convolucionais utilizam o conceito de *autômato finito* ou Máquina de Estados Finitos (*Finite State Machine*-FSM) [11]. Em geral, a FSM é definida como uma máquina geradora admitindo uma sequência de símbolos de entrada $k_n(t)$, onde uma função de transferência $(k_n(t)/out_n(t))$ gera como saída palavras códigos $out_n(t)$, correspondendo à operação da máquina, do estado i para o estado j conforme mostra a Figura 2.2.

Figura 2.2: Máquina de Estados Finitos (FSM), representada pelo estados i, j e função de transferência $(k_n(t)/out_n(t))$



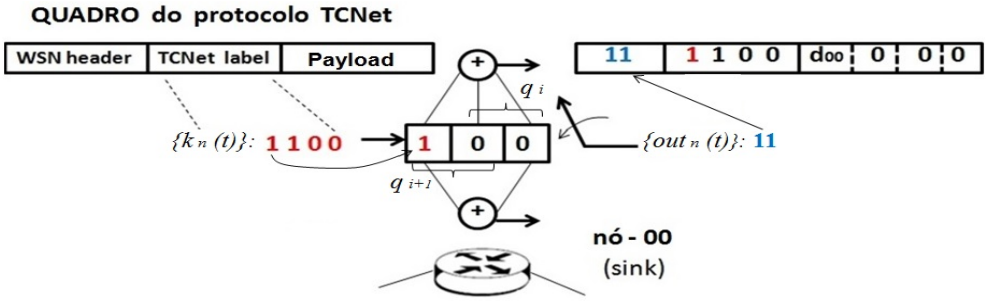
Fonte: Autor

Um método de descrever o código convolucional é considerar o funcionamento de uma Máquina de Estados Finitos (FSM)[1]. No contexto deste trabalho será utilizada a máquina de Mealy (MM) [37], conforme mostra o exemplo da Figura 2.3, onde a sequência de entrada $k_n(t) = 1\ 1\ 0\ 0...$ deslocando-se através dos registradores da MM, altera os estados da MM, nos instantes correspondentes a $(\cdots t_i, t_{i+1} \cdots)$, para $(\cdots \mathbf{q_i}, \mathbf{q_{i+1}} \cdots)$. O resultado desta operação a cada instante t é uma sequência

[1]Uma FSM pode ser implementada utilizando-se Máquina de Mealy ou Máquina de Moore [11].

de saída $out_n(t) = (c_1, c_2)$, realizada pela portas XOR (soma módulo -2) entre os dados ligados pelos ramos aos registradores, representando uma sequência codificada pela máquina geradora.

Figura 2.3: Exemplo de uma sequência de entrada $k_n(t)$ gerando uma sequência de saída $out_n(t) = (c_1, c_2)$ através da MM

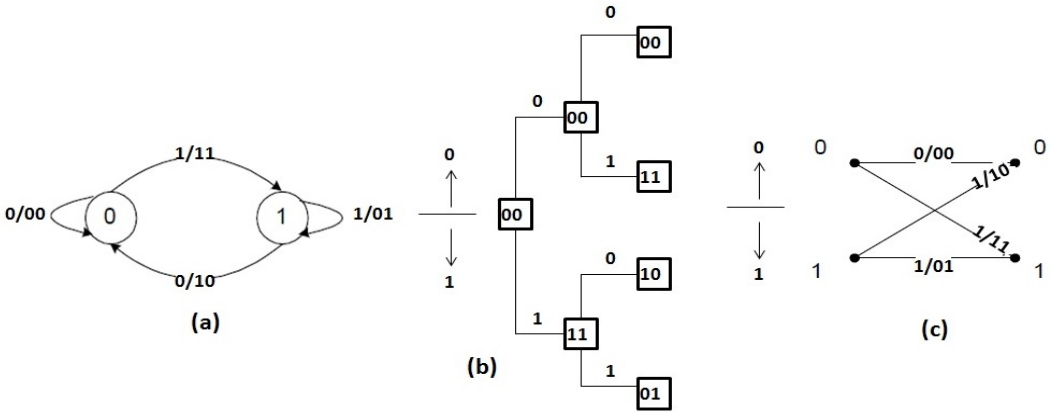


Fonte: autor

2.3 RELAÇÃO ENTRE DIAGRAMA DE ESTADOS, ÁRVORE E TRELIÇA

O funcionamento de um codificador convolucional pode ser representado pelo seu diagrama de estados ou, equivalentemente, pelo diagrama de árvore ou pelo diagrama de treliça, conforme Figura 2.4, [37], [38].

Figura 2.4: (a)Diagrama de estados correspondendo a um código convolucional com dois estados; (b)Diagrama de árvore; (c)Diagrama de treliça

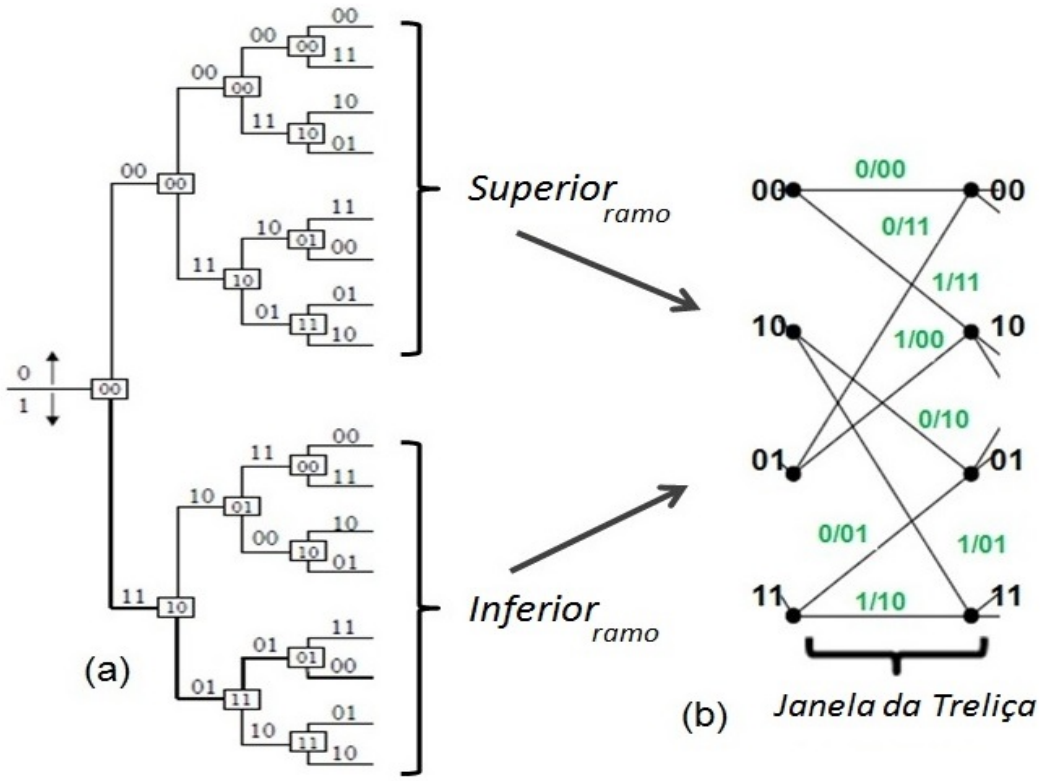


Fonte: Autor

No diagrama de árvore a convenção usada para representar o diagrama de estados corresponde ao sentido dos ramos, onde um símbolo (0) de entrada especifica o ramo superior de uma bifurcação, ao passo que o símbolo (1) especifica o ramo inferior. Assim, um percurso específico sobre a árvore descreve a sequência de entrada e a respectiva sequência codificada. Observando-se, por exemplo, a entrada (1 0 ...) sobre os ramos da árvore, teremos a sequência codificada de saída (11 10 ...). O mesmo resultado pode ser observado se for considerada a treliça.

À medida que o número de estados de um código convolucional aumenta, o diagrama de árvore resultante se expande, tornando-se inviável a análise sobre a árvore. Observando-se o exemplo da Figura 2.5(a) verifica-se que a partir da terceira derivação os ramos da ávore se repetem, ou seja, o conjunto de ramos superiores e inferiores são idênticos. Essa propriedade pode ser aplicada para obtenção de uma estrutura simplificada com as mesmas propriedades do diagrama de árvore, evitando-se uma explosão da estrutura da árvore. Concentrando-se a análise apenas em um dos conjuntos de ramos da árvore, obtém-se uma configuração de treliça conforme mostra a Figura 2.5(b), resultando em um método poderoso para análise de estruturas que utilizam códigos convolucionais.

Figura 2.5: (a) Diagrama de árvore corespondente a um código convolucional com quatro estados; (b) Diagrama de treliça resultante



Fonte: Autor

Este trabalho utiliza as propriedades do algoritmo de Viterbi [37] proposto em 1967, para decodificação de códigos convolucionais, nessa nova proposta de decodificação de rotas em uma rede, pela semelhança da análise de ramos sobre um diagrama de treliça em função de uma sequência de dados recebido.

O princípio do algoritmo de Viterbi consiste em decodificar uma sequência recebida, estimando a menor *distância de Hamming*, no caso de *hard decision*[2], entre uma sequência de símbolos recebidos e o peso do ramo ou *métrica* no trecho considerado na treliça, configurando um caso de decisão por *máxima verossimilhança*[3] - (*maximum likelihood*) - ML. O mecanismo de decisão do melhor trajeto

[2]A decisão entre as palavras códigos é quantizada em dois níveis: "0" e "1s" [37].
[3]Método probabilístico de estimar valores de modo a maximizar a probabilidade condicional dos dados [37].

sobre a treliça consiste em associar cada ramo da treliça a um valor denominado *métrica* e encontrar o caminho cuja soma das métricas seja mínima. Este procedimento configura a *máxima verossimilhança*, ou seja, a obtenção de um valor estimado $\hat{\mathbf{x}}$ entre um conjunto de símbolos transmitidos $\mathbf{x}$. Seja a sequência de K símbolos originalmente transmitidos, representada por (2.1):

$$\mathbf{x} = \{x_0, x_1, \cdots, x_{K-1}\} \tag{2.1}$$

Considerando a transmissão em canais sujeitos a erros [39], a tarefa do algoritmo de Viterbi é processar a sequência $y(t)$ recebida de modo a obter a sequência estimada $\hat{\mathbf{x}}$ (2.2):

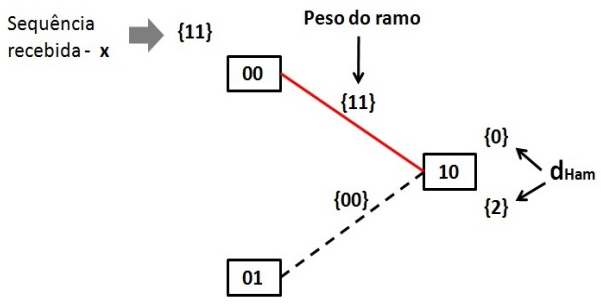
$$\hat{\mathbf{x}} = \{\hat{x}_0, \hat{x}_1, \cdots, \hat{x}_{K-1}\} \tag{2.2}$$

O problema resulta em minimizar a probabilidade de erro de uma sequência $y(t)$, que representa uma rota na rede, gerada originalmente por uma sequência $k_n(t)$. Esse procedimento consiste na obtenção da probabilidade condicional entre as possíveis sequências recebidas e pela melhor escolha entre elas. Em outras palavras, a decodificação decide pelo valor de $y(t)$ em relação a $\hat{\mathbf{x}}$, conforme (2.3):

$$P[y(t)|\hat{\mathbf{x}}] = max_{(\text{all } \mathbf{x})} P[y(t)|\mathbf{x}] \tag{2.3}$$

A Figura 2.6 mostra um exemplo de decisão sobre um ramo da treliça, onde o estado - 10 decide pela origem da sequência enviada, para estabelecer um *ramo sobrevivente*. Usando os conceitos do algoritmo de Viterbi, o estado em questão analisa os ramos adjacentes, calculando a *distância de Hamming* entre a sequência recebida - $y(t)$ e os respectivos pesos dos ramos (*métricas*), decidindo pelo ramo que apresentar menor *distância de Hamming* (operação de *hard decision*). Este exemplo mostra uma decisão simples com a ausência de ambiguidade.

Figura 2.6: Exemplo de *hard decision* usado pelo estado - 10 para decidir por um ramo sobre a treliça, baseado na sequência recebida = 11

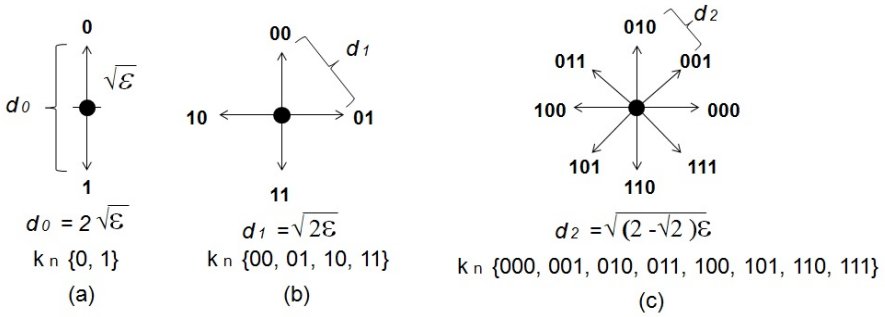


Fonte: Autor

2.4 A IMPORTÂNCIA DA DISTÂNCIA NO CÁLCULO DO CUSTO PARA DECISÃO DA ROTA

Considerando-se que a decisão da rota em um enlace está relacionada aos custos dos caminhos de menor custo [5], onde a distância é um fator importante para o cálculo desses custos, obtidos a partir da distância *Euclidiana* (d_{E}), acrescenta-se a isso a analogia da decisão da rota utilizando-se a decodificação sobre os ramos da treliça, onde são necessárias outras distâncias para os cálculos desses custos, nesse caso a distância de *Hamming* (d_{Ham}) é aplicada na decisão dos ramos entre os nós codificados da treliça. A limitação da (d_{Ham}) aplicada aos casos de *hard decision* reduzem as alternativas dos ramos que saem de um nó. Esse trabalho utiliza o conceito de *distância livre* (d_{livre}) [37], como uma alternativa para ampliar as opções de ligações entre os nós de uma rede, configurando a suavização da (d_{Ham}) conforme mostra a Figura 2.7, onde a quantidade de símbolos ν, dos nós codificados correspondem às 2^{ν} quantidades de ligações que saem de um nó. No exemplo são mostradas configurações de nós com: $2^{\nu} = 2$, $2^{\nu} = 4$ e $2^{\nu} = 8$ saídas, suas respectivas (d_{livre}) e combinações de símbolos das sequências $k_n(t)$.

Figura 2.7: Configurações de nós e suas respectivas d_{livre} entre os símbolos da sequência $k_n(t)$, onde $d_0 > d_1 > d_2$. (a)Nó da treliça correspondente a sequência $k_n(t) = \{0, 1\}$; (b)Nó da treliça correspondente a sequência $k_n(t) = \{00, 01, 10, 11\}$; (c)Nó da treliça correspondente a sequência $k_n(t) = \{000, 001, 010, 011, 100, 101, 110, 111\}$



Fonte: Autor

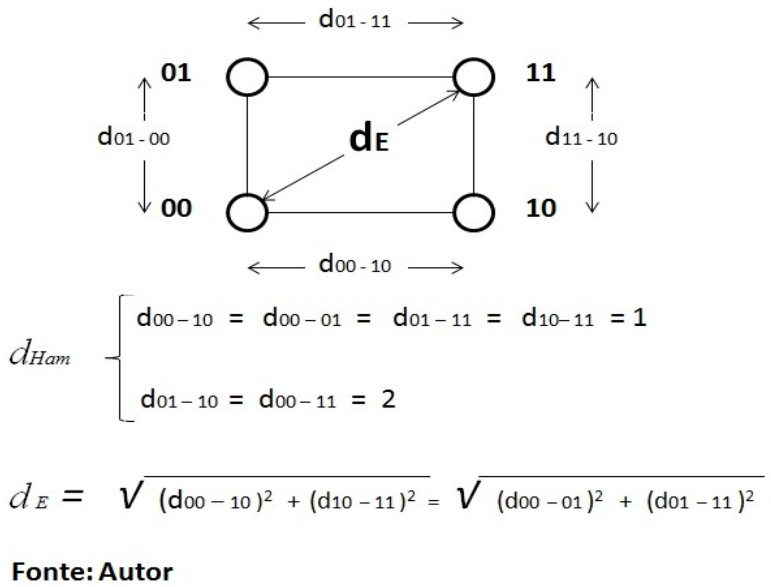
A utilização das distâncias de *Hamming* (d_{Ham}) e *Euclidiana* (d_{E}) para decisão dos ramos sobre a treliça, apresenta um problema devido à ocorrência de ambiguidades, conforme Figura 2.8. No exemplo, pode-se observar as ambiguidades de resultados para as distâncias de *Hamming* (d_{Ham}) e *Euclidiana* (d_{E}), considerando-se uma rede com 4 nós:

- As distâncias de *Hamming* (d_{Ham}) entre os nós: (00 e 10), (00 e 01), (01 e 11) e (10 e 11) são iguais a "1". Pode-se também notar as ambiguidades nas distâncias de *Hamming* (d_{Ham}) entre os nós: (01 e 10) e (00 e 11) ambos iguais a 2;

- As distâncias *Euclidiana* (d_{E}) considerando-se os percursos (00-01-11) e (00-10-11) são do mesmo modo iguais e dadas por (2.4):

$$d_E = \sqrt{(d_{00-10})^2 + (d_{10-11})^2} = \sqrt{(d_{00-01})^2 + (d_{01-11})^2} \tag{2.4}$$

Figura 2.8: Arranjo de uma rede com 4 nós mostrando os casos de ambiguidades das distâncias *Hamming* (d_{Ham}) e *Euclidiana* (d_{E})



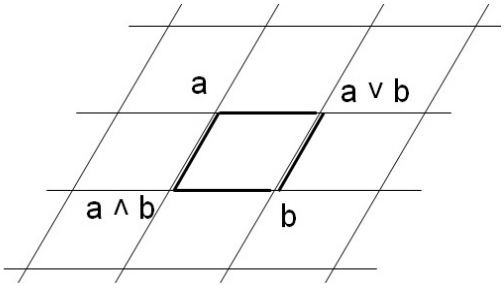
Fonte: Autor

O trabalho propõe a utilização da *distância lógica* para resolver o problema da ambiguidade, baseado no conceito de *lattice partitioning*, que será explicado a seguir.

2.4.1 Definição matemática de *lattices*

Conforme Grätzer (1978) [13], [40], sendo o *lattice* uma estrutura em treliça, conforme Figura 2.9, podem ser formuladas as seguintes definições:

Figura 2.9: Representação da estrutura de *lattices* e as operações $(a \vee b)$ e $(a \wedge b)$ obtidas sobre o *lattice*



Fonte: Autor

- **Definição 1.** *Lattice* $(\mathbf{\Lambda})$ é definido como um *Partially ordered set* (*Poset*) [41], onde os elementos $a, b \in (\mathbf{\Lambda})$ possuem um *supremum (least upper bound)* identificado pela operação $(a \vee b)$ e também um *infimum (greatest lower bound)* cuja operação é $(a \wedge b)$.

- **Definição 2.** *Partially ordered set (Poset)* é expresso pelo par $\mathcal{P} = (P, \varrho)$, onde "P" é um conjunto não vazio e "ϱ" é uma *Partial ordering relation* especificada por "$\leq$", satisfazendo as seguintes propriedades básicas, para $a,b,c \in P$:

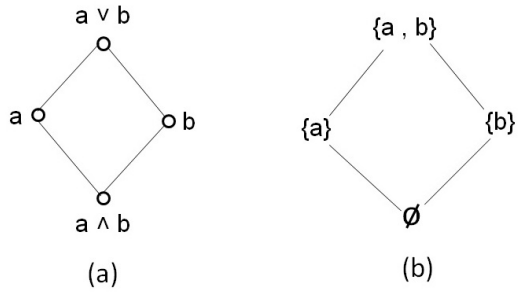
  - (**P1**) $a \leq a$, (*reflexividade*);
  - (**P2**) se $a \leq b$ e $b \leq a$, então $a = b$, (*assimetria*);
  - (**P3**) se $a \leq b$ e $b \leq c$, então $a \leq c$, (*transitividade*)

Considerando um *Partially ordered set* (P, ϱ), satisfazendo as propriedades (**P1**), (**P2**) e (**P3**), para todo $a,b,c \in P$, temos que:

* *reflexividade*:$(a,a) \in \varrho$ é verdadeiro;
* *assimetria*:(a,b), $(b,a) \in \varrho$ implica que $a = b$;
* *transitividade*:(a,b), $(b,c) \in \varrho$ implica que $a,c \in \varrho$

Uma maneira de representar a **Definição 1** de *Lattice* $(\mathbf{\Lambda})$ é através do diagrama de Hasse, Figura 2.10, ilustrando o *Partially ordered set* obtido a partir do conjunto $\{a,b\}$, correspondendo a $\{\{a,b\},\{a\},\{b\},\{\}\}$.

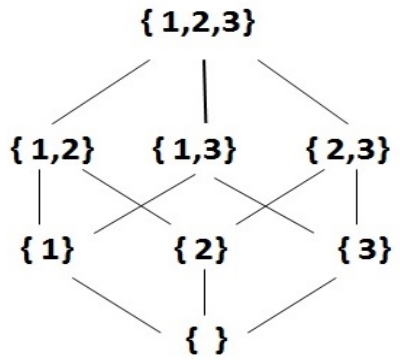
Figura 2.10: (a) *lattice* $(\mathbf{\Lambda})$; (b) Diagrama de Hasse correspondente ao *lattice* $(\mathbf{\Lambda})$



Fonte: Autor

Seja dado como exemplo o conjunto $P : \{1,2,3\}$ e a relação "$\varrho : \leq$". Considerando inicialmente o *Total ordered set*: $\{\{1,2,3\},\{1,2\},\{1,3\},\{2,3\},\{1\},\{2\},\{3\},\{\}\}$, resulta como exemplo de *Partially ordered set* a decomposição nos subconjuntos disjuntos aplicando-se o diagrama de Hasse, conforme mostra a Figura 2.11.

Figura 2.11: Decomposição do conjunto $P : \{1, 2, 3\}$ em Poset



Fonte: Autor

Outro conceito utilizado neste trabalho é o *isomorfismo*, um recurso utilizado nas demonstrações matemáticas de modo a ampliar os conhecimentos de um fenômeno em outro, conforme mostra a **Definição 3**.

- **Definição 3.** Um *isomorfismo* entre os *lattices* $\mathbf{\Lambda_0} = (\mathbf{L_0},\ \varrho)$ e $\mathbf{\Lambda_1} = (\mathbf{L_1},\ \varrho)$ existe, se ocorrer a bijeção $\varphi : \mathbf{L_0} \rightarrow \mathbf{L_1}$, de modo que $x\ \varrho\ y \Leftrightarrow \varphi(x)\ \varrho\ \varphi(y)$. Também considerada uma relação de equivalência "ϱ", denotado por "$\equiv$".

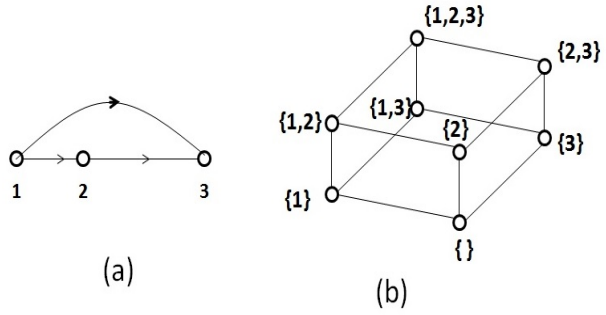
Por exemplo [13], dada a relação $\varrho = \{(a,b)\text{: } a,b \in \mathbb{N}$ e $(a + b)$ é um número par$\}$, observa-se que a relação "ϱ" obedece as propriedades (**P1**), (**P2**) e (**P3**) de um *Partially ordered set (Poset)*: é *reflexiva* devido $(a + a)$ ser par; é *simétrica* devido $(a + b)$ ser par, então $(b + a)$ também é par; é *transitiva* devido $(a + b)$ ser par e $(b + c)$ ser par, então $(a + c)$ também é par.

2.4.2 Aplicações de *lattices*

Considerando que os elementos de um *lattice* podem representar distâncias, áreas ou volumes, a solução de casos de fronteiras e distâncias mínimas em comunicações, influenciaram Ungerboeck (1982) e Forney (1988) em casos de *set partitioning* de constelação de sinais M-ary. Utilizando partições de *lattices* em *sublattices* e *cosets*, formularam a implementação de códigos para canais de baixa capacidade, [42]. Neste trabalho, será utilizado o conceito de *lattice* para formular distâncias em grafos, para aplicações em redes com recursos limitados.

Considerando-se o dígrafo representando um conjunto de pontos $\{1,2,3\}$ e a relação "$\varrho : \leq$", Figura 2.12(a), pela aplicação do diagrama de Hasse, pode-se obter o conjunto parcialmente ordenado destes pontos, ou seja um *Partially ordered set (Poset)* dos pontos do dígrafo, conforme Figura 2.12(b), equação (2.5):

Figura 2.12: (a) Conjunto de pontos de um dígrafo; (b) Exemplo de combinação em *posets* dos pontos do dígrafo usando-se o diagrama de Hasse



Fonte: Autor

$$\{\{1,2,3\},\{1,2\},\{1,3\},\{2,3\},\{1\},\{2\},\{3\},\emptyset\} \tag{2.5}$$

Nos trabalhos de Calderbank e Sloane [42], foram utilizados os conceitos de *lattice partitioning* analisando-se o *lattice* ($\mathbf{\Lambda}$) como um conjunto de *sublattices* ($\mathbf{\Lambda}'$), onde o *sublattice* ($\mathbf{\Lambda}'$) é indicado na equação (2.6):

$$\mathbf{\Lambda}' = \begin{pmatrix} n \\ k \end{pmatrix} \quad 0 \leq k \leq n \tag{2.6}$$

e o *lattice* ($\mathbf{\Lambda}$) é a combinação em *posets* dos subconjuntos de *sublattices* ($\mathbf{\Lambda}'$), na equação (2.7):

$$\mathbf{\Lambda} = \bigcup \begin{pmatrix} n \\ k \end{pmatrix} \quad 0 \leq k \leq n \tag{2.7}$$

Nesse caso, o exemplo de *Partially ordered set (Poset)* obtido na Figura 2.12, será considerado em termos dos conjuntos de *sublattices* ($\mathbf{\Lambda}'$):

$$\mathbf{\Lambda}'_{\emptyset} \longmapsto \begin{pmatrix} 3 \\ 0 \end{pmatrix} = \{\ \}$$

$$\mathbf{\Lambda}'_{1} \longmapsto \begin{pmatrix} 3 \\ 1 \end{pmatrix} = \{1\}\ \{2\}\ \{3\}$$

$$\mathbf{\Lambda}'_{2} \longmapsto \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \{1,2\}\ \{1,3\}\ \{2,3\}$$

$$\mathbf{\Lambda}'_{3} \longmapsto \begin{pmatrix} 3 \\ 3 \end{pmatrix} = \{1,2,3\}$$

ou seja,

$$\mathbf{\Lambda} = \mathbf{\Lambda'_0} \bigcup \mathbf{\Lambda'_1} \bigcup \mathbf{\Lambda'_2} \bigcup \mathbf{\Lambda'_3}$$

A aplicação de *lattices* em casos de distâncias, é utilizada neste trabalho em termos da combinação dos subconjuntos de *posets*, distribuídos entre um *supremum* e um *infimum* de *sublattices*.

Assim, considerando-se uma rede com 2^n nós e usando-se um isomorfismo com a *distância de Hamming*, a distância entre os nós do grafo pode ser analisada como a distância entre os pares (r, q), parcialmente ordenados pelo *sublattice* ($\mathbf{\Lambda}'$), dado por (2.8):

$$\mathbf{\Lambda}'_{k} = \begin{pmatrix} 2^n \\ k \end{pmatrix} \quad 0 \leq k \leq 2^n \tag{2.8}$$

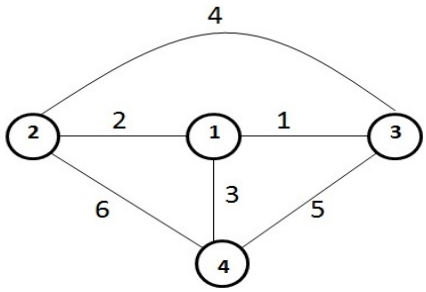
A Tabela 2.2 mostra o critério de construção e ordenação da *distância lógica* utilizada neste trabalho.

Tabela 2.2: Distribuição dos pesos dos ramos do grafo, para construção da *distância lógica*

(r, q)	Pares (r, q) obtidos a partir de $\mathbf{\Lambda}'_k$
d_{Ham}	*Distância de Hamming* relativas a (r, q)
Pesos	Sequências de elementos disjuntos $\subseteq \mathbb{Z}^+$

Como exemplo, seja o grafo com 2^n nós, onde $n = 2$, Figura 2.13, com os pesos dos ramos distribuídos de modo disjunto e $\subseteq \mathbb{Z}^+$. Para o exemplo os nós possuem a seguinte correspondência binária: nó-1 (00), nó-2 (01), nó-3 (10) e nó-4 (11).

Figura 2.13: Grafo com $2^n = 4$ nós e pesos dos ramos distribuídos de modo disjunto



Fonte: Autor

Os pares (r, q) a seguir, obtidos a partir do *sublattice* ($\mathbf{\Lambda}'_2$), (2.9):

$$\mathbf{\Lambda}'_2 \longmapsto \begin{pmatrix} 4 \\ 2 \end{pmatrix} = \{1,2\}\ \{1,3\}\ \{1,4\}\ \{2,3\}\ \{2,4\}\ \{3,4\} \tag{2.9}$$

estão relacionados na Tabela 2.3, de modo a manterem um isomorfismo com a *distância de Hamming* entre os nós, e distribuídos segundo uma sequência de pesos disjuntos $\subseteq \mathbb{Z}^+$.

Tabela 2.3: Distribuição dos pesos dos ramos para o exemplo

(r, q)	(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)
d_{Ham}	d00-d01	d00-d10	d00-d11	d01-d10	d01-d11	d10-d11
Pesos	1	2	3	4	5	6

Tabela 2.4: *Distância lógica* entre nós do grafo dado no exemplo

(r, q)	00	01	10	11
00	∞	1	2	3
01	1	∞	4	5
10	2	4	∞	6
11	3	5	6	∞

A Tabela 2.4 resultante, obtida a partir da distribuição dos pesos dos ramos na Tabela 2.3, representa as *distâncias lógicas* entre os nós do grafo para o exemplo, correspondendo a um *lattice* em $\mathbb{Z}^2$, onde as submatrizes simétricas são *sublattices*.

Para grafos de redes maiores, as tabelas com as *distâncias lógicas*, são obtidas utilizando-se o mesmo conceito de *lattice* em $\mathbb{Z}^2$, onde os *posets* são *sublattices* ($\mathbf{\Lambda}'_k$), parcialmente ordenados em submatrizes simétricas.

3 DESENVOLVIMENTO TEÓRICO

De acordo com a metodologia proposta no Capítulo 1, este capítulo descreve a possibilidade de associar os estados de um código convolucional aos nós de uma rede e a proposta do algoritmo TCNet.

3.1 ANALOGIA COM OS CÓDIGOS CONVOLUCIONAIS

Considerando-se que uma das possíveis formas de representar e analisar uma rede é por meio de uma modelagem em dígrafos e dos teoremas derivados da Teoria dos Dígrafos D=(V,E) em que:

- V é o conjunto dos vértices v_i que representam os nós ou equipamentos;
- E é o conjunto de elementos (v_i, v_j) denominados arestas que representam as conexões físicas ou lógicas entre os vértices v_i e v_j.

O conceito de código convolucional baseado em *autômato finito* ou Máquina de Estados Finitos (*Finite State Machine*) - FSM [11], apresentado no Capítulo 2 deste trabalho, utilizando uma topologia de arestas orientadas, em que os nós representam os estados da FSM, sugere uma analogia com os elementos de uma rede.

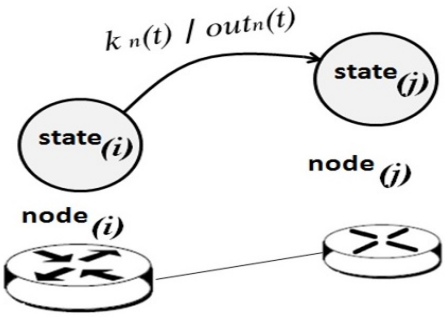
Conforme a Figura 3.1, um símbolo de entrada de uma sequência $k_n(t)$ no instante t está relacionado com uma palavra código de saída $out_n(t)$ pela função de transferência $(k_n(t)/out_n(t))$, correspondendo a operação de uma FSM do estado i para o estado j tendo como analogia um enlace do nó i para o nó j.

Baseando-se nesta analogia, este trabalho aplicou os mesmos conceitos de códigos convolucionais na solução de estruturas de redes WSNs devido à semelhança de topologia de grafos e a transição de estados se comportar como uma transmissão de pacotes de dados.

3.1.1 Modelo de roteamento de uma rede WSN utilizando o protocolo TCNet

Usando os mesmos conceitos dos códigos convolucionais, é possível gerar uma rota específica através dos nós de uma rede WSN usando diferentes critérios para definir a melhor rota, inclusive a

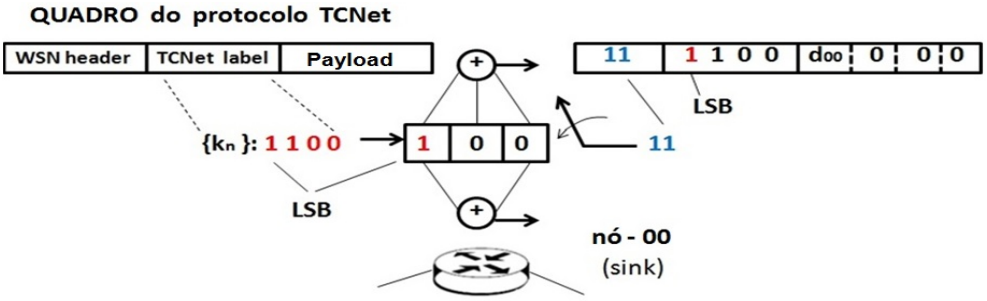
Figura 3.1: Nós da rede usando como modelo os estados de uma Máquina de Estados Finitos - FSM



Fonte: autor

QoS. O modelo possibilita configurar uma sequência $k_n(t)$ com características desejadas de (latência, perda de pacotes, largura de banda, baixo consumo de energia ou outras características), podendo ser alteradas sem muita complexidade. Este trabalho utiliza a FSM como uma máquina de Mealy - MM implementada em cada nó da rede, por ser um gerador de baixa complexidade, construído com registradores de deslocamento e portas OU-Exclusivo (XOR) [11]. O mecanismo consiste em deslocar uma sequência $\{k_n\}$ modelada pela característica desejada, utilizada como *label* do protocolo TCNet, através da MM e obter palavras códigos que serão utilizadas como *header* dos QUADROS com as informações da rota sobre a rede. É importante observar que as rotas são definidas por meio de um procedimento *off-line*. Os critérios de otimização são definidos em função das características das aplicações, e o descobrimento da rota pode ser feito por exemplo utilizando o *Algorithms for multirestrictive routing hop by hop* [10], onde cada rota é associada a uma sequência $k_n(t)$. O algoritmo [10] é interessante devido levar em conta simultaneamente diferentes métricas: de desempenho, lineares, não lineares e inclusive métricas que definam resultados econômicos.

Figura 3.2: Exemplo da montagem do QUADRO do protocolo TCNet



Fonte: autor

Considerando o cenário mostrado na Figura 3.2, o protocolo TCNet pode ser explicado através dos seguintes passos:

- *Passo 1*: O nó-00 (*sink*), nó com atribuições de gerenciamento da rede WSN, deseja interrogar um conjunto de nós da rede. Esses nós são considerados destinos dos QUADROS TCNet.

Para realizar a operação utiliza uma sequência $k_n(t)$ com as características da rota desejada, transportada no campo TCNet *label* do QUADRO;

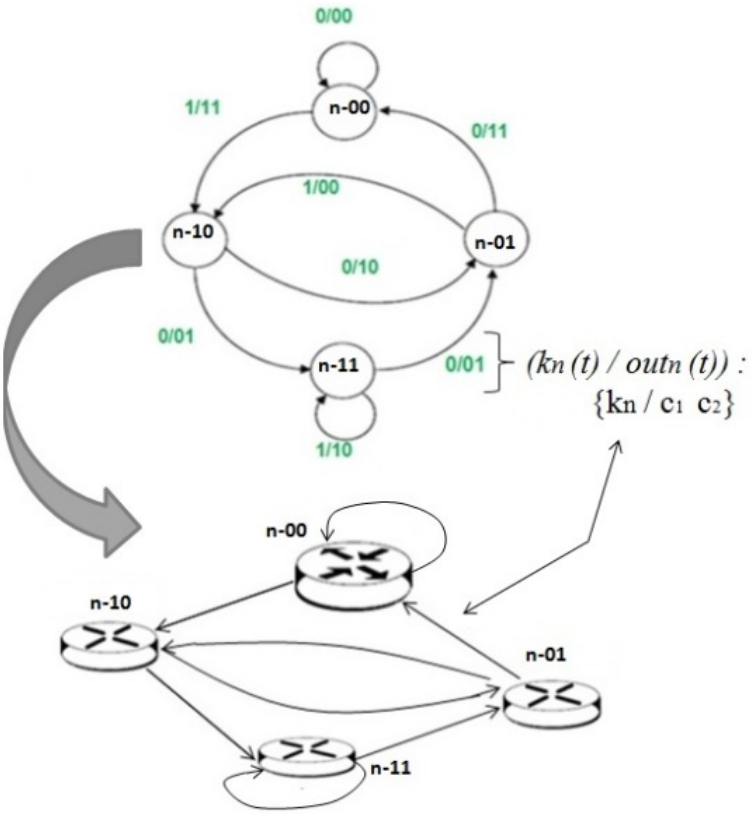
- *Passo 2*: O mecanismo consiste em deslocar a sequência através da MM como mostra a Figura 3.2, em que os registradores da MM inicialmente no estado zero, têm seus estados alterados nos instantes seguintes $(\cdots t_i, t_{i+1} \cdots)$ para $(\cdots \mathbf{q_i}, \mathbf{q_{i+1}} \cdots)$, correspondendo aos nós da rede;

- *Passo 3*: O resultado dessa operação a cada instante t é uma sequência de saída $out_n(t) = (c_1, c_2)$, resultante da operação lógica entre os dados dos registradores e soma módulo-2, que será utilizada como *header* do QUADRO, como peso do ramo da treliça, para decisão do endereço do nó que está sendo interrogado.

- *Passo 4*: Esse processo se repete ao longo da rota, em que o nó que é destino atualiza uma nova sequência de saída $out_n(t) = (c_1, c_2)$ que será transportada como *header* do QUADRO da rede WSN, e após atualizar o *payload data*, encaminha esse QUADRO ao próximo nó da rota.

A Figura 3.3 mostra um cenário de uma rede WSN com quatro nós modelado por uma FSM de quatro estados e as correspondências existentes entre os nós da rede e os estados da FSM, enfatizando a função de transferência $(k_n(t)/out_n(t))$: $\{k_n/c_1\,c_2\}$ indicada nos arcos do diagrama de estados.

Figura 3.3: Rede WSN com quatro nós usando o modelo de uma FSM de quatro estados



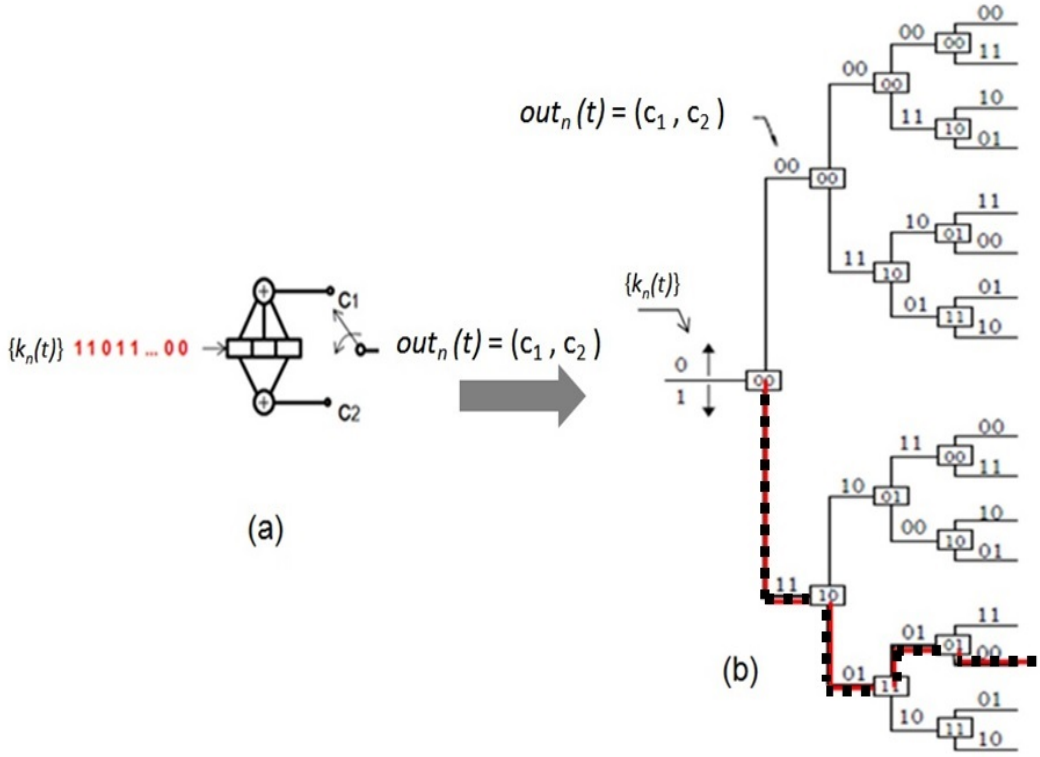
Fonte: autor

Pode-se observar que a estrutura da rede obtida possui analogia a um diagrama de estados gerado por uma FSM de 4 estados, controlada pela sequência de símbolos de entrada $k_n(t)$.

3.2 DECODIFICADOR DOS PROTOCOLOS COM ESTRUTURA DE CÓDIGOS CONVOLUCIONAIS

Assim, como foi visto no Capítulo 2, o diagrama de árvore pode ser usado para descrever a evolução dos estados, neste trabalho, serão considerados como nós sobre a árvore. Em função disto, é possível traçar um trajeto sobre a árvore determinado pela sequência de entrada $k_n(t)$ na MM, como mostra a Figura 3.4, onde o símbolo (1) da sequência $k_n(t)$ especifica um ramo inferior de uma bifurcação na árvore, enquanto o símbolo (0) especifica o ramo superior.

Figura 3.4: (a) MM - geradora com a sequência de entrada $k_n(t)$ e palavra código de saída $out_n(t) = (c_1, c_2)$; (b) Diagrama de árvore baseado na MM - geradora e detalhe do trajeto na "linha tracejada" sobre a árvore, associado à sequência de entrada $k_n(t)$ na MM - geradora



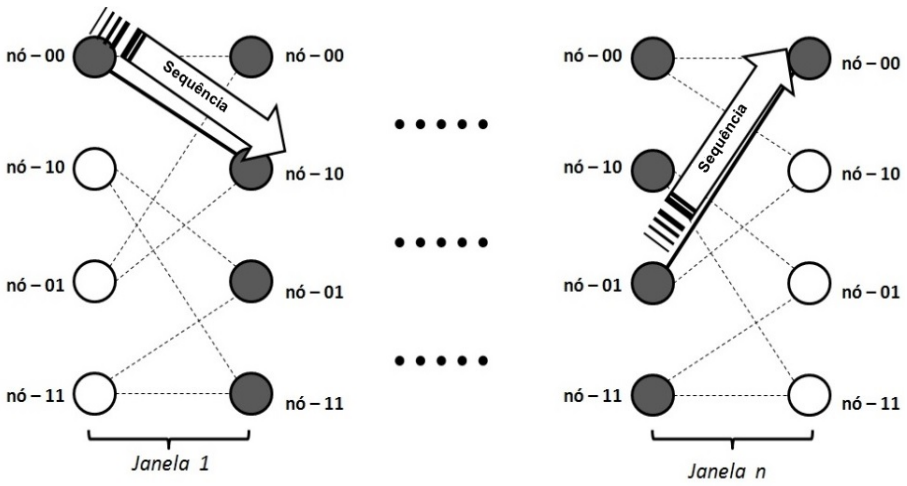
Fonte: autor

Considerando que a estrutura do diagrama de árvore não é a melhor representação para análise de roteamento, as simplificações obtidas devido à eliminação das redundâncias do diagrama de árvore, torna o diagrama de treliça uma das ferramentas utilizadas para análise das configurações geradas pelas máquinas de estados finitos [38], [37]. Assim como no diagrama de estados e no diagrama de árvore, no diagrama de treliça podem ser identificados os mesmos componentes, como: *estados*, *ramos*

e *rotas*, descritos a seguir, conforme Figura 3.5:

- Os estados representam os nós da rede;
- Cada ramo que sai de um estado anterior para um novo estado representa um enlace da rota, definido por uma sequência de entrada $k_n(t)$;
- Um trajeto sobre a treliça é formado por um conjunto de ramos representando uma rota estabelecida por um mapeamento lógico, obtido pela repetição da janela da treliça com todas as propriedades do diagrama de estados, até que toda a sequência $k_n(t)$ seja transmitida.

Figura 3.5: Janelas da Treliça utilizadas para compor uma rota



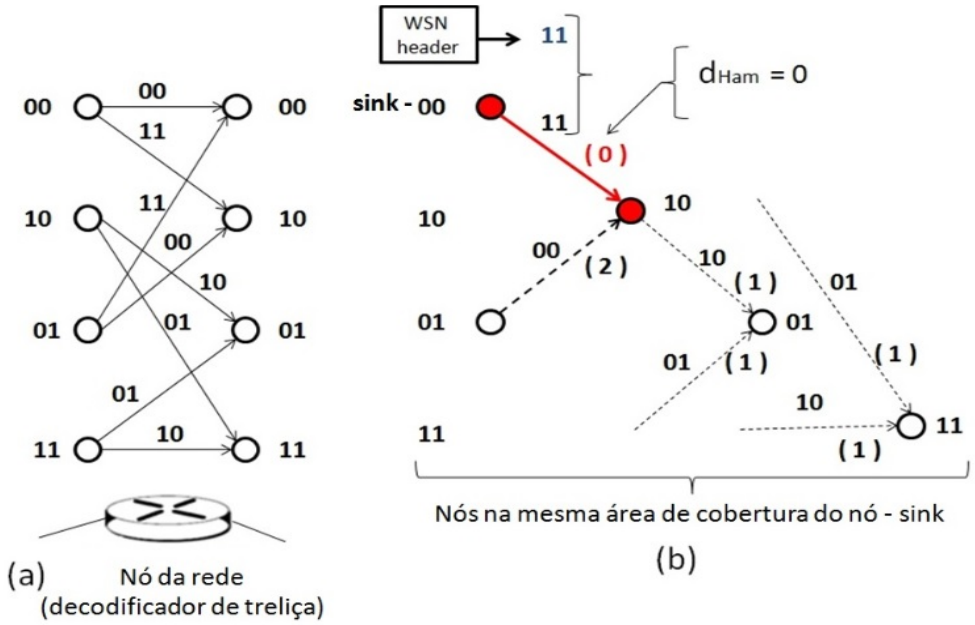
Fonte: autor

A utilização do algoritmo de Viterbi [37] para tomadas de decisão sobre a treliça, pode ser vista em um cenário típico de *broadcast*, em que inicialmente os nós da rede encontram-se na mesma área física e um nó pode estar recebendo sinal de todos os outros. A Figura 3.6 mostra uma situação simples, em que cada nó recebe informação apenas de dois outros nós. Como exemplo, o nó (10) recebe uma solicitação com a informação $(c_1, c_2) = (11)$, desencadeando uma consulta ao *decodificador de treliça* residente no nó, e sendo informado com as seguintes possibilidades de recepção: nó (00) pode gerar o código $(c_1, c_2) = (11)$ e nó (01) pode gerar o código $(c_1, c_2) = (00)$, Figura 3.6(a).

A Figura 3.6(b) mostra as operações realizadas pelos nós na mesma área de cobertura do emissor, nó (sink-00). A operação realizada pelo nó (10) consiste em obter d_{Ham} entre a palavra código transportada no *header* do QUADRO e a informação da palavra código registrada no ramo da treliça, resultando:

- ramo como origem o nó (00) $\Rightarrow$ $d_{\text{Ham}} = 0$;
- ramo como origem o nó (01) $\Rightarrow$ $d_{\text{Ham}} = 2$.

Figura 3.6: (a)Decodificador de treliça residente no nó; (b)Nós na mesma área de cobertura do nó (sink)



Fonte: autor

O valor $d_{\text{Ham}} = 0$ indica que o QUADRO recebido pelo nó (10) foi enviado pelo nó (sink-00), desencadeando assim as demais operações pelo nó (10), decorrentes da solicitação recebida: atualizar o *payload* e determinar a próxima palavra código $(c_1, c_2) = (01)$ usando a sequência $\{k_n\}$ recebida no *label* do QUADRO, para ser atualizada no *header* do QUADRO.

Os demais nós situados na mesma área de cobertura realizam o mesmo procedimento do nó (10) não obtendo $d_{\text{Ham}} = 0$. Neste caso, os demais nós permanecem aguardando uma nova solicitação.

Considere o exemplo para o caso do decodificador de treliça possuir mais concorrência entre os nós, conforme Figura 3.7 [37], obtida a partir de uma nova MM geradora. Neste caso não houve um aumento da rede, o que ocorreu foi uma alteração na configuração da máquina usada. O aumento dos nós da rede acarreta um procedimento semelhante ao do aumento da concorrência da informação no nó, sendo utilizado o mesmo procedimento de decisão para os casos mais simples conforme exemplo da Figura 3.6. Nesse caso o nó (01) obtém a d_{Ham} entre a palavra código transportada no *header* do QUADRO e as palavras códigos registradas nos ramos concorrentes da treliça, resultando:

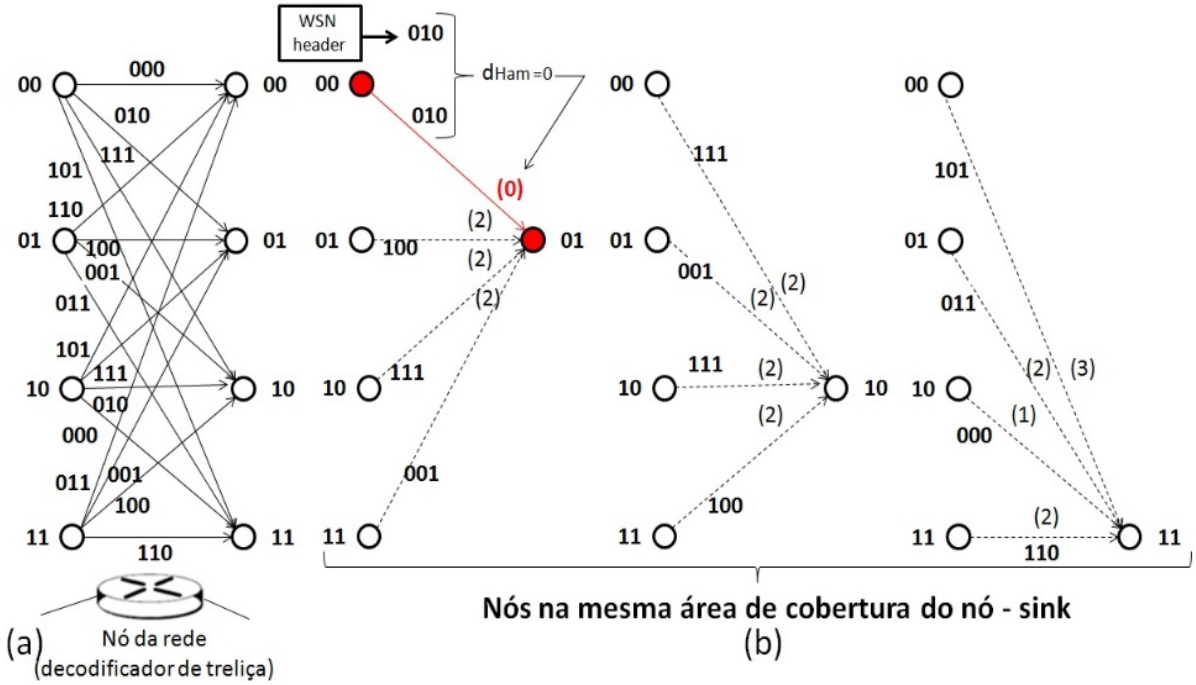
- ramo como origem o nó (00) $\Rightarrow$ $d_{\text{Ham}} = 0$;
- ramo como origem o nó (01) $\Rightarrow$ $d_{\text{Ham}} = 2$;
- ramo como origem o nó (10) $\Rightarrow$ $d_{\text{Ham}} = 2$;
- ramo como origem o nó (11) $\Rightarrow$ $d_{\text{Ham}} = 2$;

O valor $d_{\text{Ham}} = 0$ indica que o QUADRO recebido pelo nó (01) foi enviado pelo nó (sink-00), descartando assim as outras possibilidades, de modo que, o nó (01) assume as demais operações

decorrentes da solicitação recebida: atualizar o *payload* e determinar a próxima palavra código (c_1, c_2) a ser atualizada no *header* do QUADRO.

Os demais nós situados na mesma área de cobertura realizam o mesmo procedimento do nó (01), não obtendo $d_{\text{Ham}} = 0$ permanecem aguardando uma nova solicitação.

Figura 3.7: (a)Decodificador de treliça residente no nó; (b)Caso de nós na mesma área de cobertura do nó (sink) apresentando maior concorrência



Fonte: autor

As situações mais genéricas como casos de ambiguidades e exemplos de casos maiores de redes serão apresentados nas simulações citadas no Capítulo 4.

4 SIMULAÇÃO E RESULTADOS EXPERIMENTAIS

Neste Capítulo serão descritos os cenários de simulação do algoritmo TCNet e as comparações com o algoritmo AODV utilizando a ferramenta de simulação OMNeT++.

4.1 DESCRIÇÃO DO SIMULADOR

Este trabalho utiliza como ambiente de simulação o OMNeT++ [43] baseado em C++ e orientado a objetos. Atualmente tem sido amplamente utilizado em pesquisas por ser um software aberto com aplicações em simulações e modelagem de redes de tráfego, servindo como referência para comparações entre outras técnicas devido aos *frameworks*[1] disponíveis.

O ambiente de simulação OMNeT++ disponibiliza uma biblioteca de componentes de redes que agrupados em módulos permite realizar simulações de forma escalável, ou seja, com muitos nós de rede tanto para arquiteturas fixas ou móveis (*wireless*). Os resultados das simulações podem ser interpretados de modo gráfico e animações no próprio ambiente ou serem exportados para ambientes gráficos e estatísticos como R [45]. A portabilidade do OMNeT++ permite ser utilizado nos sistemas operacionais mais comuns (Linux, Mac OS/X e Windows), no caso deste trabalho foi feita a opção pelo sistema operacional Windows.

4.2 CENÁRIO DE SIMULAÇÃO

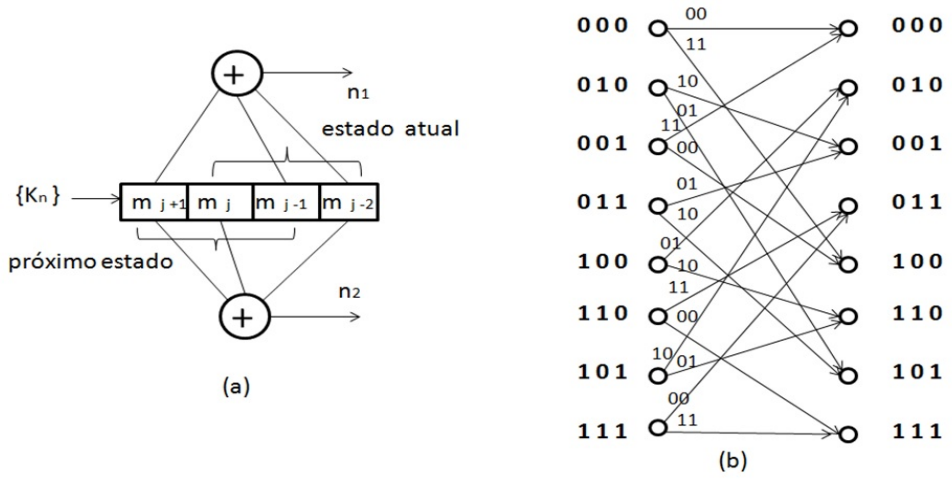
Mostra o funcionamento do algoritmo TCNet utilizando-se o cenário de uma rede com 8 nós para análise de desempenho por meio da latência e da potência consumida pelos diferentes nós da rede, que serão detalhados a seguir.

[1] Em desenvolvimento de software é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica [44].

4.2.1 Cenário de implementação do algoritmo TCNet

Para demonstrar o mecanismo do algoritmo TCNet foram feitos inicialmente testes com uma rede de 8 nós, considerando um caso ideal sem falhas de nós, onde o nó SINK envia um *query* com um tráfego CBR para verificar a alcançabilidade dos nós. A Figura 4.1 mostra o modelo do nó utilizado, configurado por uma Máquina de Mealy (MM) com taxa k/n=1/2 e o respectivo decodificador de treliça.

Figura 4.1: (a)Máquina de Mealy (MM) com taxa k/n=1/2, resultando na saída palavras códigos (n_1, n_2); (b)Diagrama de treliça correspondente à MM



Fonte:Autor

O cenário considerado, Figura 4.2, compreende uma rede fixa no modo *Multicast* em uma mesma área de cobertura, permitindo a alcançabilidade de todos os nós pela a utilização da sequência $k_n(t) = 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0$ escolhida previamente e transportada no FRAME transmitido pelo SINK.

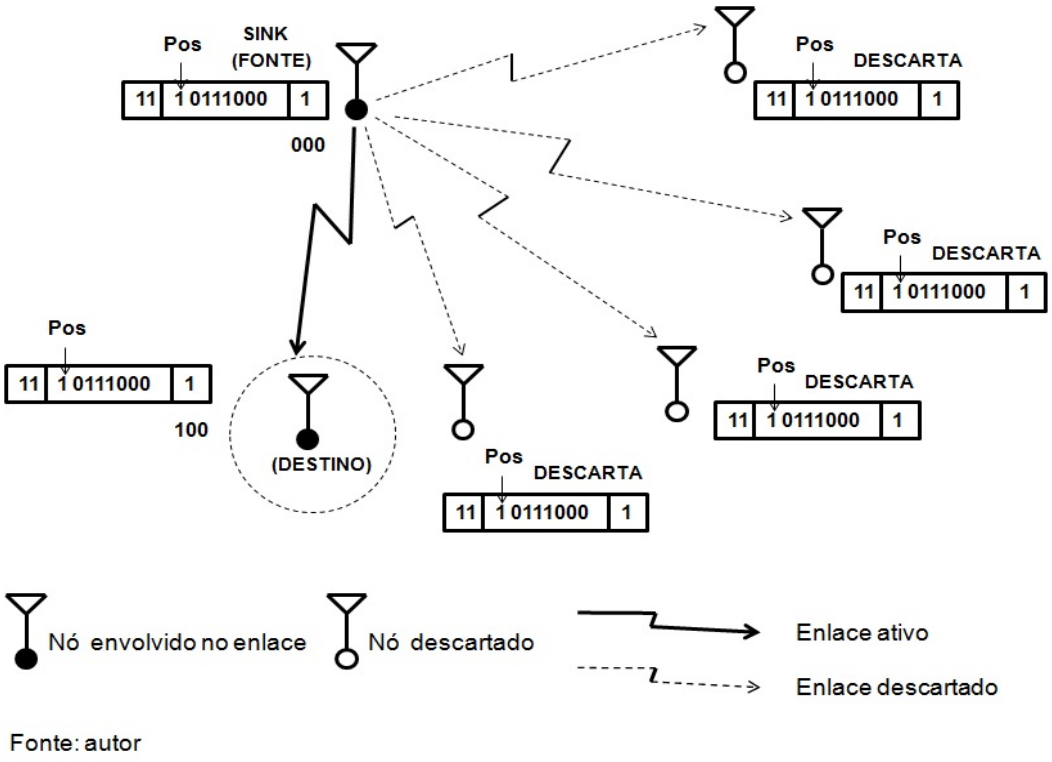
A composição do FRAME está estruturada com os seguintes dados:

- **HEADER**, resultado das palavras códigos (n_1, n_2) na saída da MM;
- **SEQUÊNCIA**: $k_n(t) = 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0$;
- **POS**, indica a posição na sequência durante os deslocamentos no registrador da MM;
- **PAYLOAD**, transporta os dados coletados nos nós.

A Figura 4.2 mostra o cenário da simulação onde o 1º *Multicast* do FRAME iniciado pelo SINK é recebido pelos demais nós da rede. No exemplo, todos os nós recebem e realizam o processamento da sequência transportada no FRAME, até à posição (**POS**) indicada, possibilitando assim decidir qual é o nó atual e o próximo nó da rede, de modo a identificar o nó destino e o ramo correspondente da

rota. Após definida a identificação, os demais nós não reconhecendo a solicitação descartam o FRAME recebido e ficam esperando um novo *Multicast* que será desencadeado pelo último nó destino, após a atualização do FRAME por esse nó.

Figura 4.2: Cenário do 1º *Multicast* da rede, realizado pelo SINK e a identificação pelo nó (100) - DESTINO



Para obter os dados das simulações realizadas neste trabalho foi utilizado o seguinte hardware:

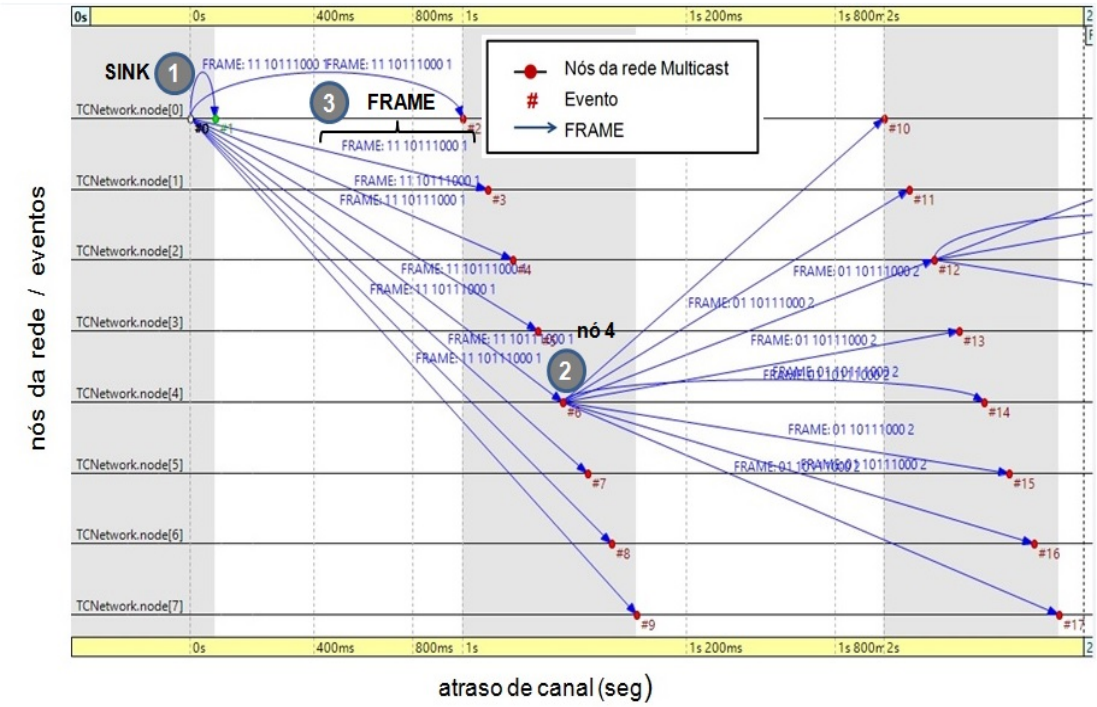
- Processador Intel Core i3-2.7 GHz;
- Memória: 4GB;
- Sistema Operacional: Windows 8.1

A Figura 4.3 mostra o detalhe da simulação do 1º *Multicast*, iniciado pelo SINK e a identificação pelo nó DESTINO - nó 4 (100) realizado no OMNeT++ usando o modo *Eventlog* [2]. Analisando-se a Figura 4.3 observa-se o SINK (1) iniciando o 1º *Multicast*, sendo recebido por todos os nós e reconhecido apenas pelo nó 4 (2), que assume a atualização do FRAME (3) e desencadeia o *Multicast* seguinte.

Considerando o resultado teórico da rota para o cenário escolhido, pode-se observar pela Figura 4.4 a rota resultante, onde os ramos estão definidos sobre uma treliça representando o percurso desde o SINK, passando por todos os nós da rede e retornando ao próprio SINK após a conclusão do *query*.

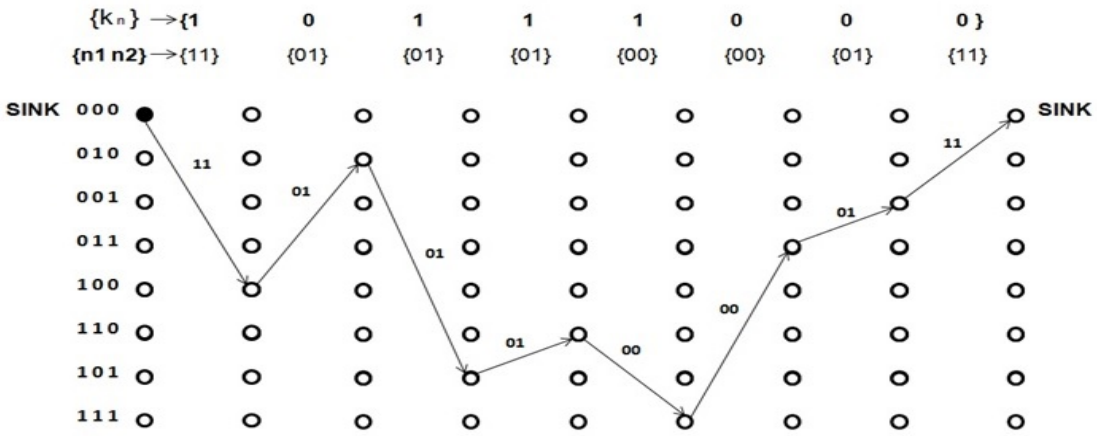
[2]Resultado do OMNeT++ que auxilia no entendimento passo a passo da simulação

Figura 4.3: Resultado simulado no OMNeT++ no modo *Eventlog* mostrando detalhe do 1º *Multicast* pelo SINK



Fonte: autor

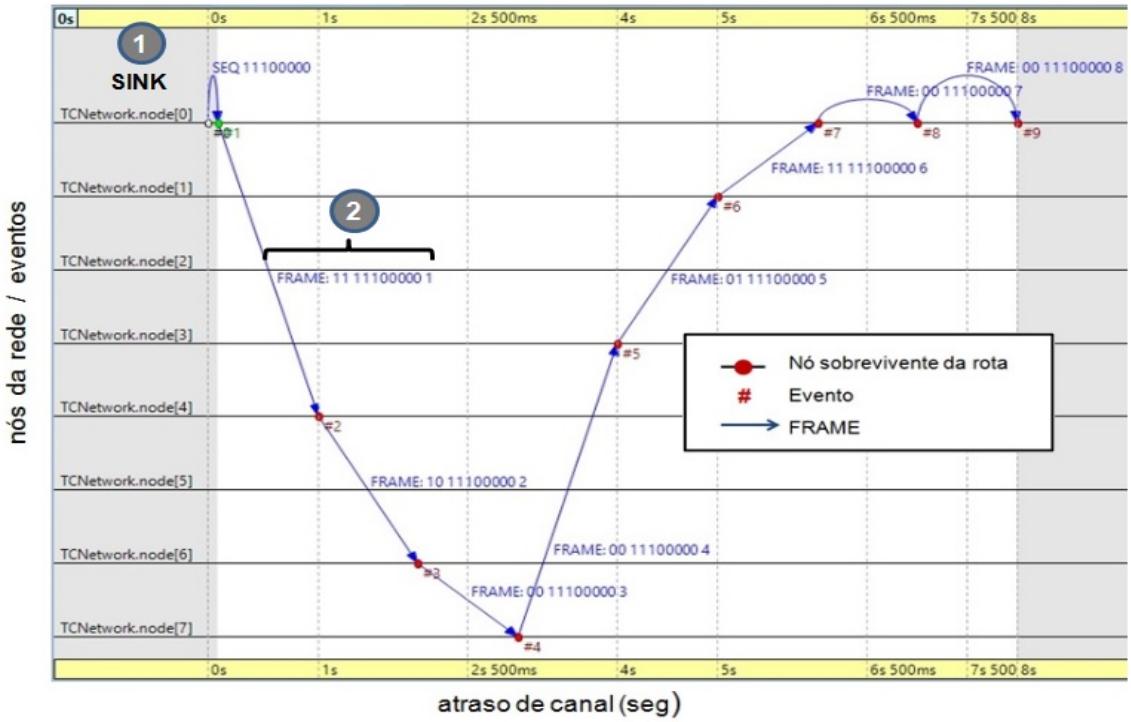
Figura 4.4: Resultado teórico da rota sobre a treliça devido a um *query*



Fonte: autor

Pode-se confirmar o resultado teórico utilizando-se o modo *Eventlog* do OMNeT++ conforme mostra a Figura 4.5. Analisando-se a Figura 4.5 observa-se a rota resultante, iniciada pelo SINK (1) e o reconhecimento pelos nós pertencentes à rota, mostrando ainda detalhe do FRAME (2).

Figura 4.5: Resultado simulado no OMNeT++ no modo *Eventlog* mostrando a rota sobre a treliça devido a um *query*



Fonte: autor

4.2.2 Latência do algoritmo TCNet durante um *query*

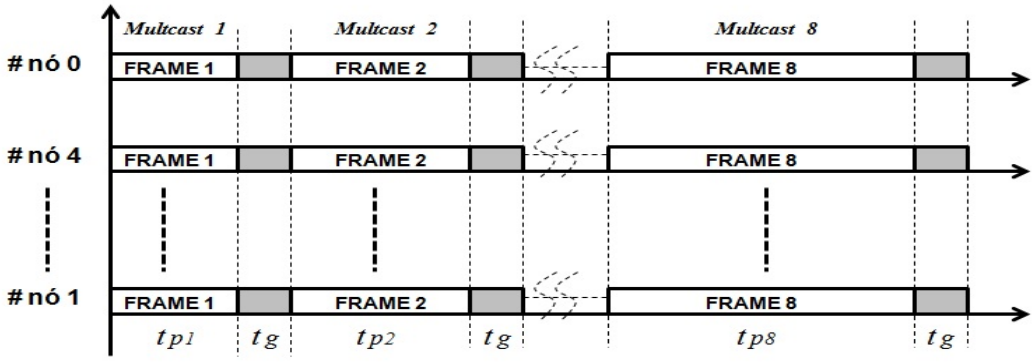
Embora os tempos de processamento dos FRAMES ($t_p n$) executados pela MM sejam diferentes, devido às posições (**POS**) nos deslocamentos da sequência de entrada na MM, todos os nós da rede estarão ocupados executando a mesma sequência recebida durante o *Multicast*, com isso as colisões serão evitadas. Mesmo assim, foram consideradas bandas de guarda (t_g) para resguardar possíveis colisões devido a atrasos de canal.

Na Figura 4.6, o Diagrama de Tempo ilustra os tempos durante o processamento dos *Multicasts* da rede considerada, para demonstrar o funcionamento do algoritmo TCNet.

Para obtenção da latência da rede durante um *query* foram normalizados os seguintes dados:

- **Tempo de processamento** (é o tempo ($t_p n$) considerado durante o deslocamento da sequência na MM):
  - Tamanho máximo da sequência: 2^{16} bits

Figura 4.6: Diagrama de tempo do processamento dos *Multicasts* na rede TCNet com 8 nós durante um *query*



Fonte: autor

– Clock: 1 Mbps

Assim,

$t = 2^{16} \ / \ 10^6 \ \ s \longrightarrow 65 \ \ ms$

$t_{p\,n} = 2(t) = 0,13 \ \ s \longrightarrow 0,1 \ \ s$ (valor normalizado)

- **Atraso de canal** (é o tempo de propagação (t_c) no canal de comunicações *wireless* considerando):

  – Distância máxima: $d_{max} = 1000 \ \ m$ e $c = 3.10^8 \ \ m/s$

  Assim,

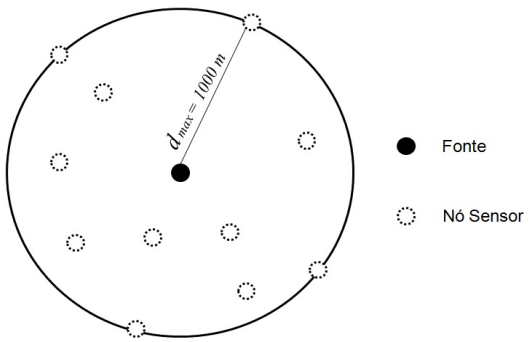
  $t_c = d_{max}/(3.10^8) \ \ s \ \ = \ \ 3,3.10^{-6} \ \ s \longrightarrow 0,000003 \ \ s$

- **Banda de Guarda** (é o intervalo de tempo (t_g) de segurança considerado pelos nós entre os *Multicasts* dos FRAMES para reduzir as chances de colisões):

  $t_g = 0,2 \ \ s$

A Figura 4.7 mostra o cenário teórico considerado, com a distribuição aleatória dos nós em uma área de cobertura com a distância máxima ($d_{max} = 1000 \ \ m$).

Figura 4.7: Área de cobertura considerando a distância máxima ($d_{max} = 1000 \ \ m$)



Fonte: Autor

Tabela 4.1: Contribuição da latência individualmente pelos nós da rede

$\Sigma T_{L0} = 0,1 + 0,000003 + 0,2 = 0,300003 \quad s$
$\Sigma T_{L4} = \Sigma T_{L0} + (0,2 + 0,000003 + 0,2) = 0,700006 \quad s$
$\Sigma T_{L2} = \Sigma T_{L4} + (0,3 + 0,000003 + 0,2) = 1,20001 \quad s$
$\Sigma T_{L5} = \Sigma T_{L2} + (0,4 + 0,000003 + 0,2) = 1,80001 \quad s$
$\Sigma T_{L6} = \Sigma T_{L5} + (0,5 + 0,000003 + 0,2) = 2,50002 \quad s$
$\Sigma T_{L7} = \Sigma T_{L6} + (0,6 + 0,000003 + 0,2) = 3,30002 \quad s$
$\Sigma T_{L3} = \Sigma T_{L7} + (0,7 + 0,000003 + 0,2) = 4,20002 \quad s$
$\Sigma T_{L1} = \Sigma T_{L3} + (0,8 + 0,000003 + 0,2) = 5,20002 \quad s$

Utilizando-se a equação (4.1) que define a latência total da rede, pode-se obter a contribuição dos nós para latência da rede considerada.

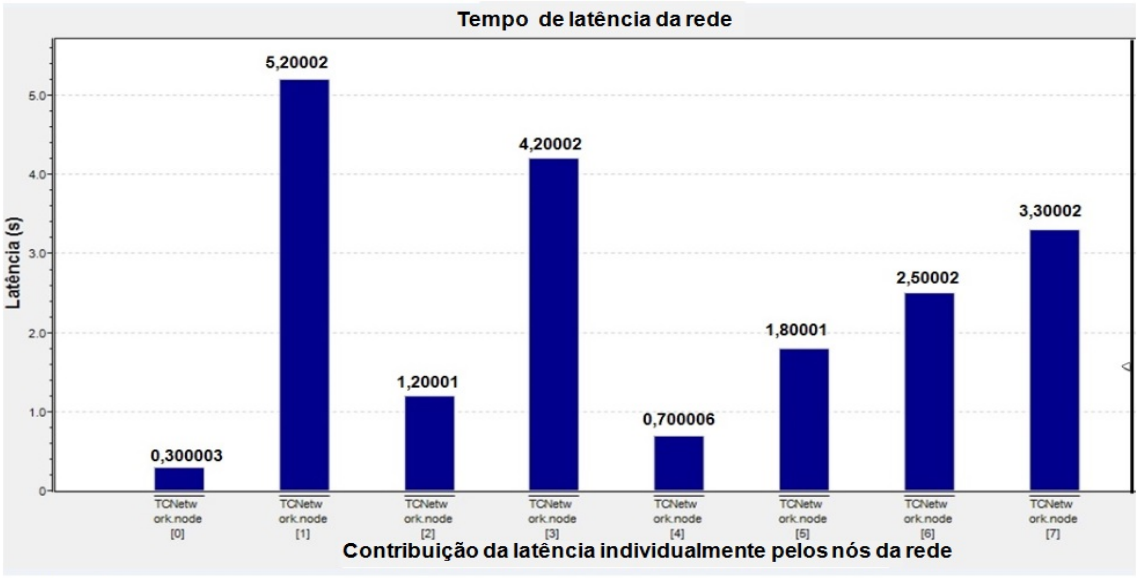
$$\Sigma T_L = t_p + t_c + t_g \tag{4.1}$$

Inicialmente foram levantados os valores apresentados na Tabela 4.1, que mostra a soma acumulada da latência individualmente dos nós, organizados na mesma sequência da rota considerada: ΣT_{L0} $(nó\ 0)$, ΣT_{L4} $(nó\ 4)$, ΣT_{L2} $(nó\ 2)$, ΣT_{L5} $(nó\ 5)$, ΣT_{L6} $(nó\ 6)$, ΣT_{L7} $(nó\ 7)$, ΣT_{L3} $(nó\ 3)$ e ΣT_{L1} $(nó\ 1)$.

A Tabela 4.1 mostra uma variação progressiva da latência, devido ao processamento da sequência nas respectivas MM configuradas nos nós da rede.

A simulação usando-se o OMNeT++, Figura 4.8, demonstra os resultados teóricos obtidos na Tabela 4.1, onde o nó 1 (ΣT_{L1}) contribui com a maior latência por ser o último nó identificado pela rede, resultando no maior tempo de execução pela MM.

Figura 4.8: Resultados dos tempos de processamento durante os *Multicasts* na rede TCNet com 8 nós



Fonte: autor

4.2.3 Energia consumida pelo algoritmo TCNet durante um *query*

A energia consumida numa WSN é um fator fundamental do projeto devido às limitações das fontes que na maioria das vezes não são substituíveis. Este trabalho deu importância à energia consumida pelos nós na utilização do algoritmo TCNet, levando em conta a distribuição das potências nas seguintes situações: *Transmissão (tx)*, *Recepção (rx)*, *Processamento (proc)* e *Banda de guarda (bg)*. Considerando-se a distribuição dos sensores aleatoriamente em uma área de cobertura com distância máxima ($d_{max} = 1000 \quad m$), foram normalizados os seguintes valores de potência para este trabalho, conforme o padrão IEEE 802.11b [7]:

- Potência de *Transmissão* (P_{tx}): 2 mW;
- Potência de *Recepção* (P_{rx}): 1 mW;
- Potência gasta no *Processamento* (P_{proc}): 1 mW;
- Potência gasta durante a *Banda de guarda* (P_{bg}): situação de *Power save*[3] (consumo desprezível de energia).

Utilizando-se a equação (4.2) para definir a energia total consumida pelo nó, pode-se obter a contribuição dos nós para a o consumo total da rede.

$$\Sigma E_{(n)} = E_{tx} + E_{rx} + E_{proc} + E_{bg} \tag{4.2}$$

[3]Situação de reduzido consumo de energia geralmente da ordem de (pW), utilizado pelos dispositivos com pouca infraestrutura, enquanto aguardam serem chaveados, [46].

Normalizando-se as diferentes parcelas de tempo de consumo de energia da rede, pelos valores da Tabela 4.1, obtém-se os seguintes intervalos de tempos: ΔT_{tx}, ΔT_{rx}, ΔT_{proc} e ΔT_{bg}.

Seja o exemplo do consumo de energia referente ao nó 0:

- Intervalos de tempos considerados:

$\Delta T_{tx} =$ 0,1 s;
$\Delta T_{rx} =$ 0,1 s + 0,000003 s (atraso de canal) $\simeq$ 0,1 s;
$\Delta T_{proc} =$ 0,1 s;
$\Delta T_{bg} =$ 0,2 s (tempo não considerado para o consumo de energia).

- Resulta no seguinte consumo de energia:

$E_{tx} = (2.10^{-3}).10^{-1} = 2.10^{-4}$ J;
$E_{rx} = (1.10^{-3}).10^{-1} = 1.10^{-4}$ J (considerando o atraso de canal);
$E_{proc} = (1.10^{-3}).10^{-1} = 1.10^{-4}$ J;
$E_{bg} =$ (desprezível).

- Energia total consumida pelo nó 0:

$\Sigma E_{(0)} = 2.10^{-4} + 1.10^{-4} + 1.10^{-4} = 4.10^{-4}$ J

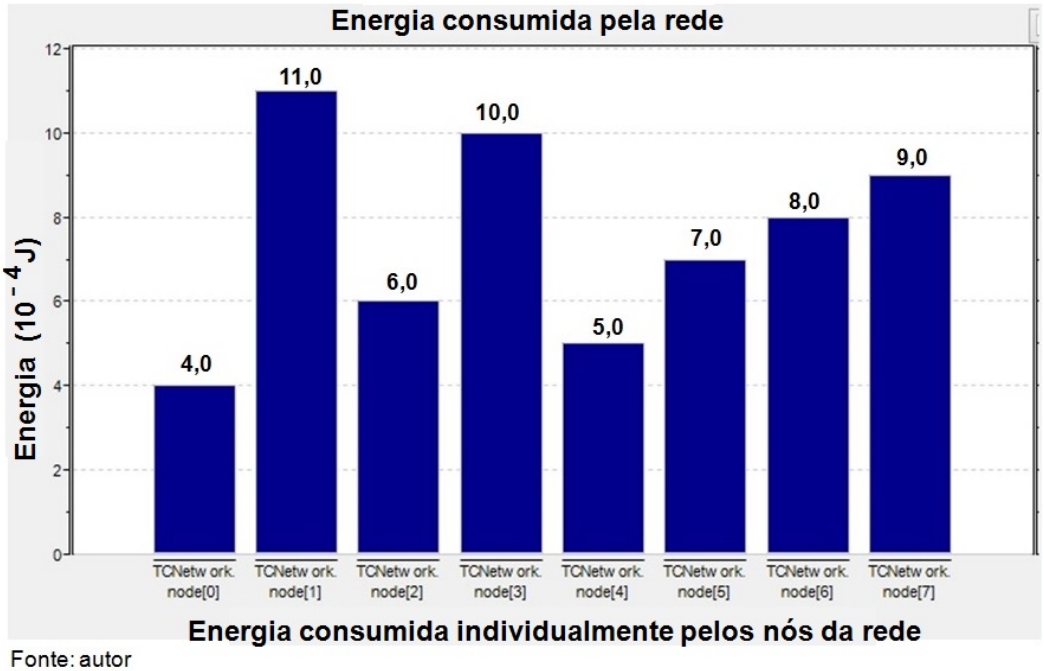
Os valores levantados na Tabela 4.2, mostram o consumo individual de energia pelos nós, organizados na mesma sequência da rota considerada.

Tabela 4.2: Contribuição individualmente da energia $\Sigma E_{(n)}$ consumida pelos nós da Rede

$\Sigma E_{(0)} = 4.10^{-4}$	J
$\Sigma E_{(4)} = 5.10^{-4}$	J
$\Sigma E_{(2)} = 6.10^{-4}$	J
$\Sigma E_{(5)} = 7.10^{-4}$	J
$\Sigma E_{(6)} = 8.10^{-4}$	J
$\Sigma E_{(7)} = 9.10^{-4}$	J
$\Sigma E_{(3)} = 10.10^{-4}$	J
$\Sigma E_{(1)} = 11.10^{-4}$	J

O resultado da simulação usando-se o OMNeT++, Figura 4.9, demonstra os valores teóricos obtidos na Tabela 4.2, onde o nó 1 (ΣE_1) é o responsável pelo maior consumo de energia devido à sua posição na rota considerada, exigindo maior tempo de processamento da MM.

Figura 4.9: Resultado da energia consumida pelos nós durante os *Multicasts* na rede TCNet com 8 nós



Fonte: autor

4.3 ANÁLISE DE DESEMPENHO DO ALGORITMO TCNet EM RELAÇÃO AO ALGORITMO AODV

Será mostrada uma análise de desempenho do algoritmo TCNet através de comparações de latência e energia consumida, com o algoritmo AODV.

4.3.1 Cenário inicial de uma rede com 8 nós considerando os mesmos critérios para nós específicos (*Fonte* e *Destino*)

A análise de desempenho visa identificar cenários justos de comparação entre o TCNet e o AODV assumindo as mesmas condições de parâmetros e quantidades de nós, de modo a realizar medidas de latência e energia consumida pela rede. O cenário de simulação foi definido como:

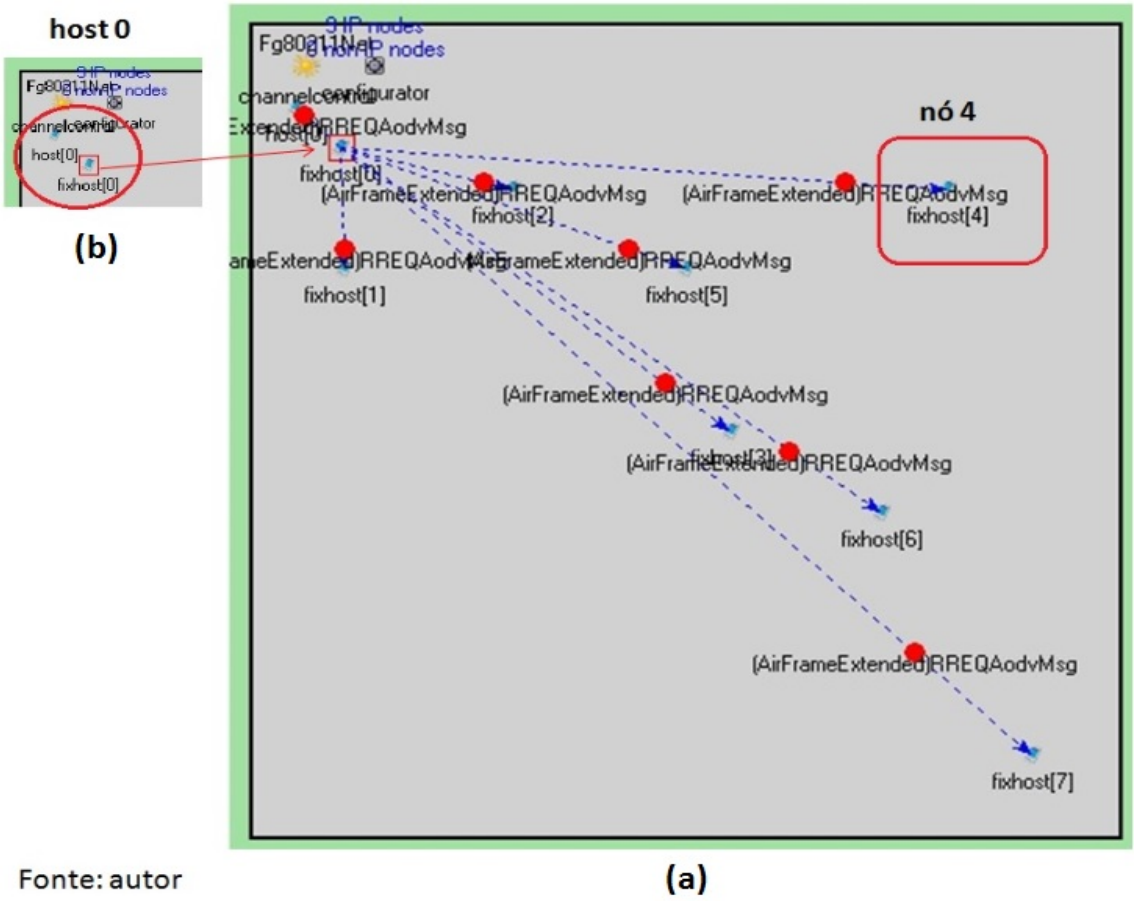
- **TCNet**
  - O cenário considerado é fixo;
  - Número de nós: 8;
  - Topologia simulada: nós posicionados aleatoriamente em uma área com ($d_{max} = 1000 \ \ m$), conforme mostra a Figura 4.7, onde o nó fonte é o SINK que inicializa a comunicação com os demais nós por meio de *queries* na rede;
  - Protocolo: MAC proprietário (hardware utilizado nas simulações práticas), Anexo A;
  - Tráfego: fluxo CBR com pacotes de 512 bytes;
  - Tempo de simulação: duração de 1 *query*;
- **AODV**
  - O cenário considerado é fixo;
  - Número de nós: 8;
  - Topologia simulada: nós posicionados aleatoriamente em uma área com ($d_{max} = 1000 \ \ m$), conforme mostra a Figura 4.7, onde o host(0) é o nó fonte e tenta comunicação com os demais nós da rede utilizando rotas alternativas obtidas através de sucessivos *floodings*;
  - Protocolo: MAC Protocolo 802.11b [7];
  - Tráfego: fluxo CBR com pacotes de 512 bytes;
  - Tempo de simulação: até a ocorrência do 1º *flooding* na rede, com as negociações (RREQ e RRep) e a confirmação do ACK pelo nó fonte host(0).

Considerando-se inicialmente a alcançabilidade da rede para o TCNet e o AODV, foram feitas medidas da latência para alcançar os nós destinos: nó 4, nó 6, nó 7 e nó 1, seguindo os critérios de acessibilidade: nós mais afastados fisicamente para ambos os casos ou nós considerados críticos na rota, como são os casos dos nós visitados por último conforme a sequência $k_n(t)$ do TCNet, o que corresponde a um tempo maior de processamento.

- **Comparação da latência de um nó específico**

  – A Figura 4.10 mostra o cenário de decisão da rota no AODV. Usando-se o modo *Eventlog* do OMNeT++ pode-se observar os detalhes do *flooding* realizado pelo nó *Fonte*: host(0), na tentativa de encontrar uma rota para o nó *Destino*: (nó 4).

Figura 4.10: (a)Cenário do AODV, mostra um *flooding* emitido pelo nó *Fonte* host (0) para todos os nós da rede e detalhe do nó 4 (*Destino*), resultado do OMNeT++; (b)Detalhe do nó host(0)

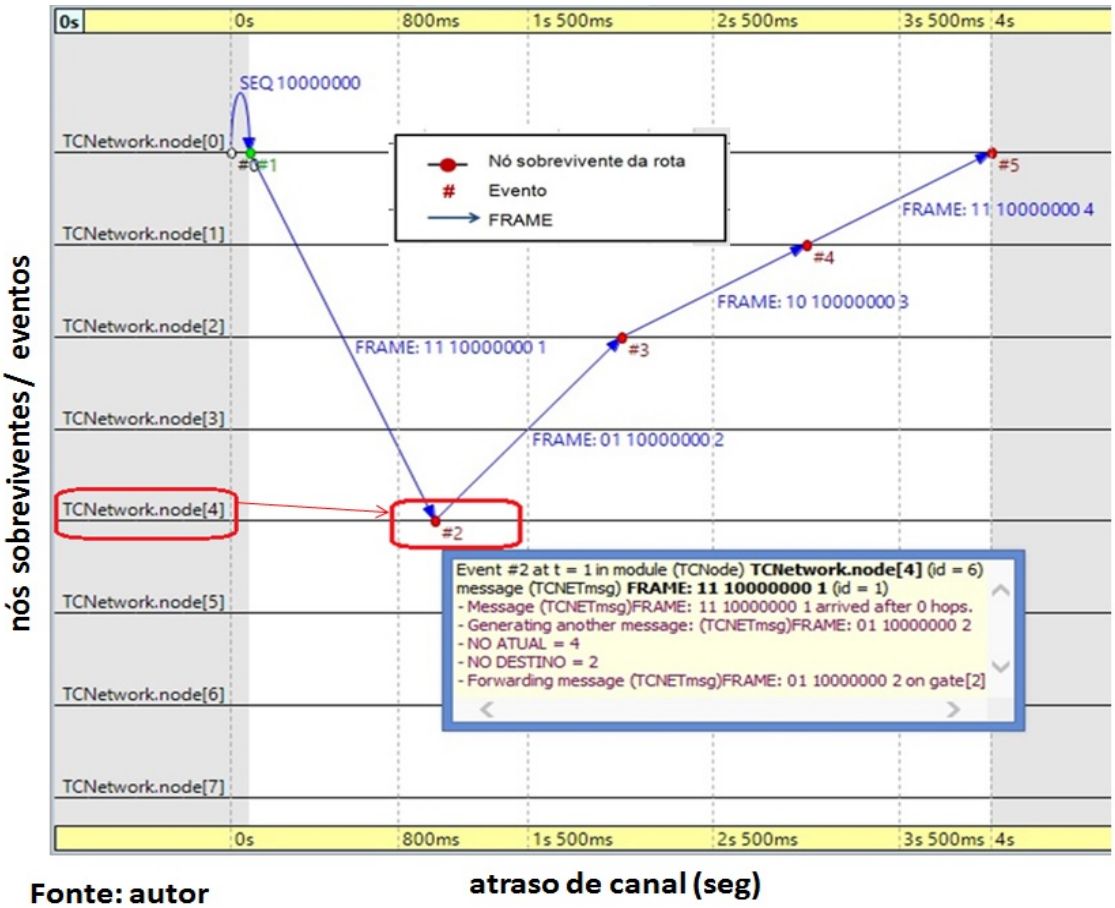


  – A Figura 4.11 mostra o cenário do algoritmo TCNet onde o nó *Fonte* (SINK) estabelece a rota para consultar o nó *Destino* (nó 4), usando o modo *Eventlog* do OMNeT++. Para a rede considerada, a consulta ao nó 4 é estabelecida pela sequência transportada no FRAME: $k_n(t) = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0$.

  – Diferente do AODV, o TCNet não necessita consultar todos os nós da rede para obter os dados de um nó específico, resultando em economia de tempo de processamento e consumo

de potência. É importante observar que o tamanho da sequência $k_n(t)$ é igual ao número de nós que devem ser visitados. No caso de um cenário onde todos os nós forem visitados, o aumento de nós tem impacto direto no aumento da sequência $k_n(t)$. Assim a sequência está ligada à rota e não à topologia da rede.

Figura 4.11: Cenário do TCNet mostrando a rota correspondente ao *query* emitido pelo *Fonte*: SINK (nó 0), para consultar do nó *Destino*: (nó 4), resultado obtido no OMNeT++

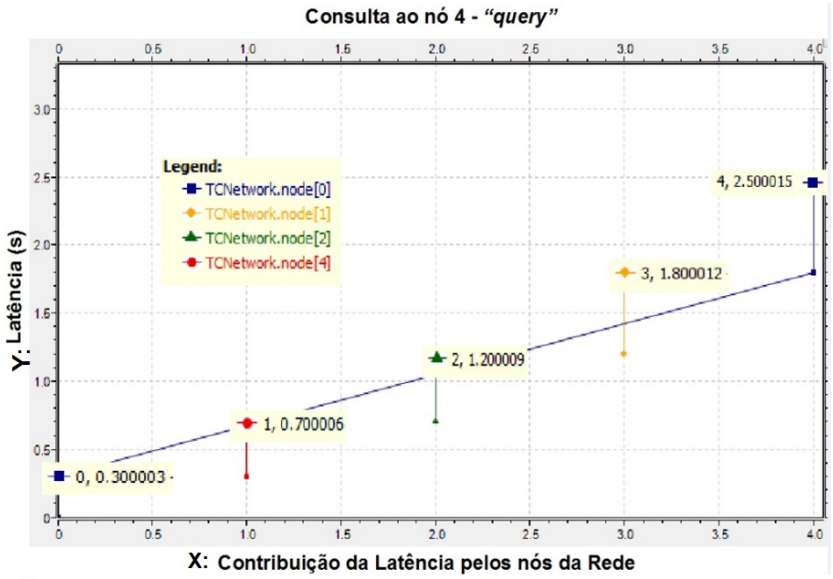


Fonte: autor

– A Figura 4.12 mostra o resultado da simulação obtida no OMNeT++, representando a latência dos nós envolvidos na consulta pelo SINK ao *Destino*: (nó 4), durante 1 *query*. O eixo X do gráfico identifica o nós que participam da rota: nó 0, nó 1, nó 2 e nó 4. O eixo Y mostra a duração do processamento dos respectivos nós, em segundos.

– Os resultados nos extremos do gráfico da Figura 4.12 mostram o processamento inicial do SINK: $\Sigma T_{L0} = 0,300003$ s e o processamento final do SINK: $\Sigma T_{L0} = 2,500015$ s, representando a duração de 1 *query* que será levado em conta para comparação entre o TCNet e o mecanismo de sinalização do AODV.

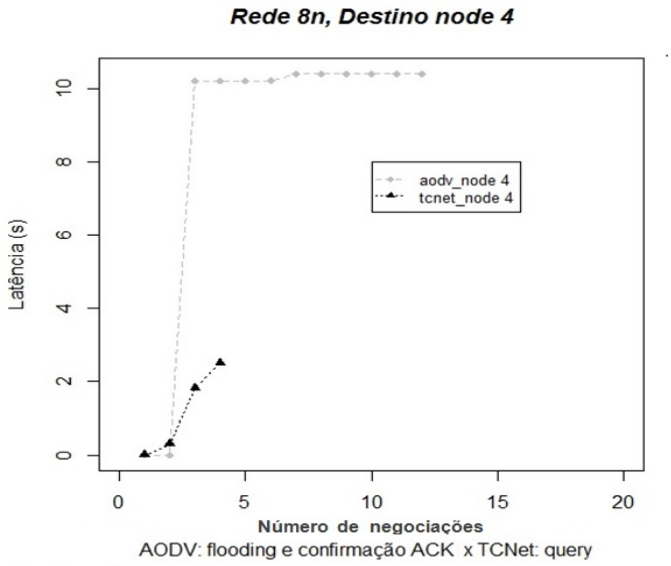
Figura 4.12: Cenário do TCNet, mostrando a rota correspondente ao *query* emitido pelo nó *Fonte* (SINK) para consultar do nó *Destino* (nó 4), obtido no OMNeT++



Fonte: autor

– A Figura 4.13 compara a latência dos cenários AODV e TCNet durante a consulta ao nó 4, pertencente a uma rede com 8 nós distribuídos aleatoriamente em áreas com a mesma cobertura de propagação. A latência do AODV é 75% maior do que a latência do TCNet devido à simplicidade do TCNet em realizar 1 *query*, não necessitando de tarefas de sinalizações.

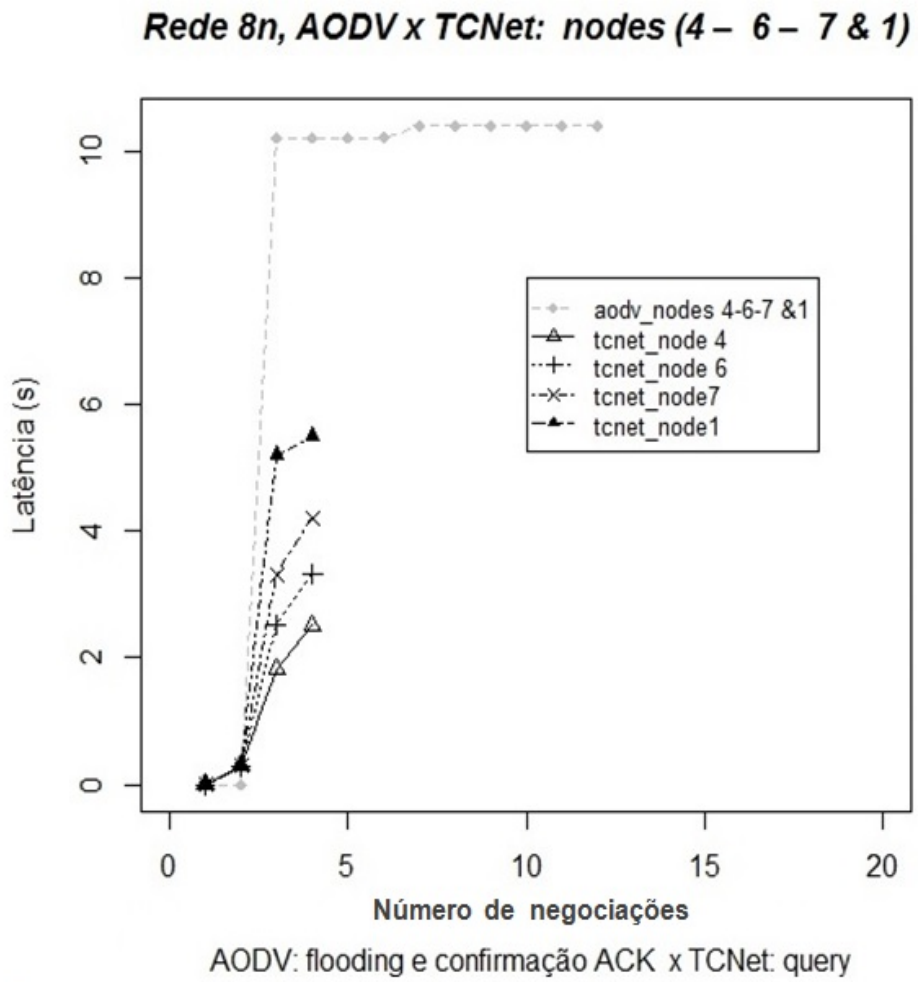
Figura 4.13: Comparação de 1 *query* do TCNet e o mecanismo de sinalização do AODV para estabeler uma rota até o nó 4 da rede



Fonte: autor

- **Comparação da latência dos demais nós da rede**
  - A Figura 4.14 compara os *queries* do TCNet para os outros nós da rede, além do nó 4, com o mecanismo de estabelecimento de rota do AODV, para os mesmos nós: 4, 6, 7 e 1.
  - No TCNet a latência aumenta à medida que os nós correspondem às últimas posições da sequência $k_n(t)$, ocasionando maior tempo de processamento da MM durante a tomada de decisão do nó destino.
  - No AODV percebe-se uma demora inicial no estabelecimento da rota e uma estabilização temporária da latência. No TCNet o pior caso de latência corresponde a 50% da latência do AODV, no cenário considerado para uma rede de 8 nós.

Figura 4.14: Cenário de comparação dos *queries* do TCNet para uma rede com 8 nós em relação ao AODV



Fonte: autor

- **Comparação do desempenho de consumo de energia**

  Metodologia usada para medida de desempenho do consumo de energia no TCNet e no AODV:

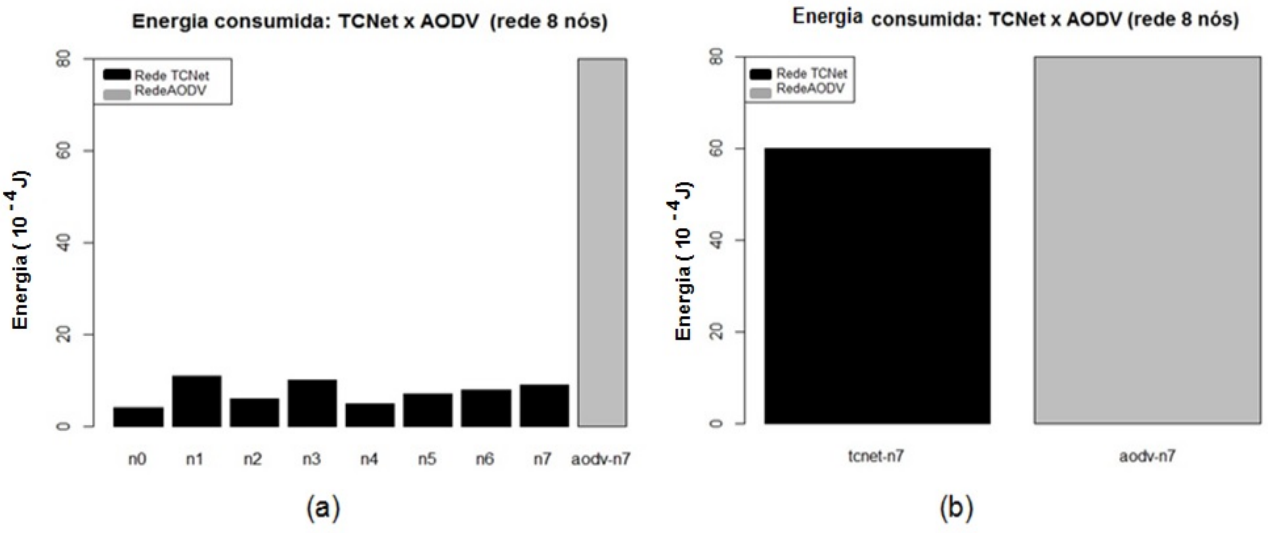
  – No TCNet foram usadas as medidas de consumo diretamente em cada nó, utilizando-se a equação (4.2), onde o consumo total da rede dado por: $\Sigma E_{(n)}$;

  – No AODV foi considerado o tempo total (ΣT_{Lat}) em que ocorrem as negociações usando a sinalização (RREQ, RREP e ACK) para estabelecer a rota para o nó destino, levando-se em conta que, em cada evento do AODV ocorre consumo de energia na transmissão, recepção e processamento, respectivamente dados por: E_{tx}, E_{rx} e E_{proc}. Assim o consumo de energia considerado para a rede AODV será dada pela equação (4.3):

$$E_{Taodv} = \Sigma T_{Lat}\,(P_{tx} + P_{rx} + P_{proc}) \tag{4.3}$$

  – Para a medida de comparação do consumo de energia: TCNet x AODV, foi considerada uma rede com 8 nós de modo que todos os nós participam da rota o que corresponde ao caso de maior consumo de energia, conforme mostram os resultados na Figura 4.15.

Figura 4.15: Cenário de comparação do consumo de energia numa rede com 8 nós, entre um *query* do TCNet para o destino: nó 7 em relação ao mecanismo de sinalização do AODV para alcançar o nó 7; (a)Energia distribuída pelos nós do TCNet em relação à energia consumida pelo AODV; (b)Comparação entre as energias totais consumidas pelos mecanismos do TCNet x AODV



Fonte: Autor

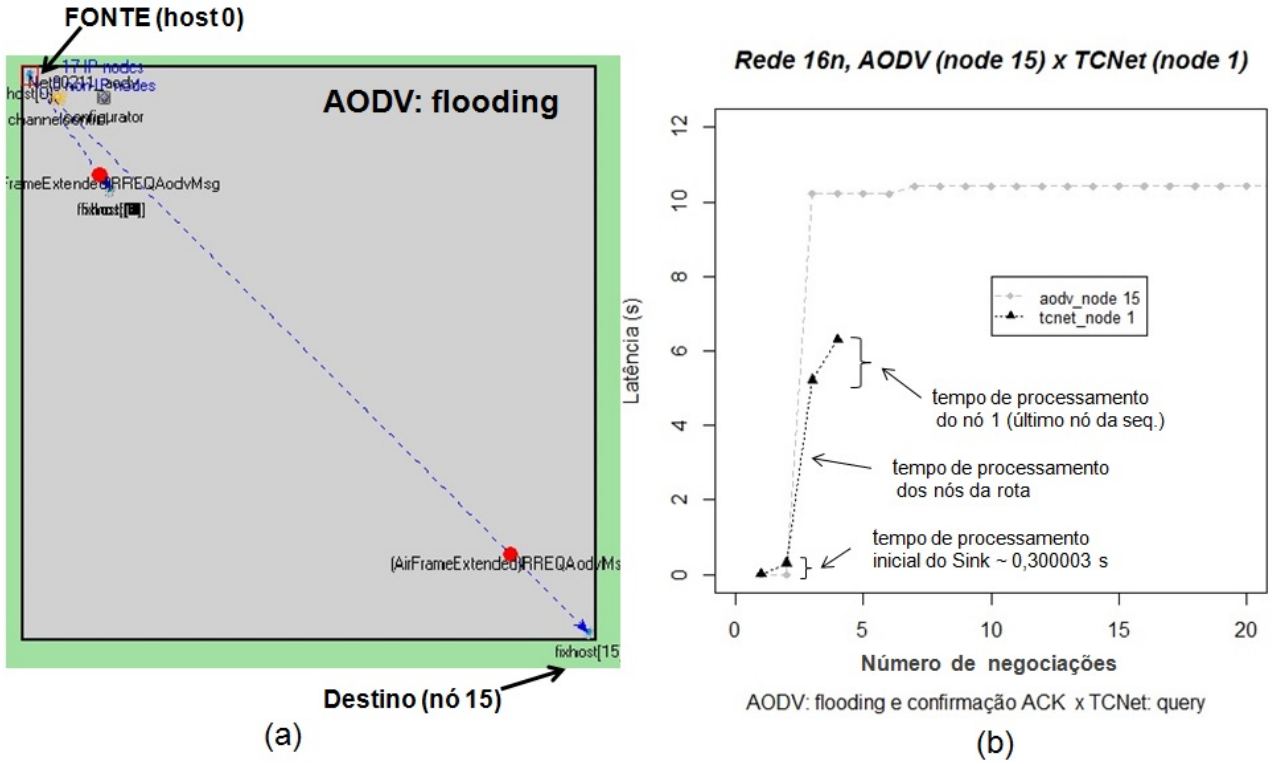
  – Pode-se observar que a energia consumida pelo TCNet é 25% menor em relação à energia consumida pelo AODV.

4.3.2 Análise de desempenho do algoritmo TCNet em relação ao algoritmo AODV com cenários expandidos

A análise de desempenho visa identificar cenários justos de comparação entre o TCNet e o AODV assumindo as mesmas condições de parâmetros e quantidades de nós, de modo a realizar medidas de latência e energia consumida pela rede. A metodologia utilizada foi definida como:

- **Dificuldade em acessar o nó**
  - No AODV foi considerada a "distância física", ou seja, o nó que se encontra mais afastado da FONTE, no *playground*;
  - No TCNet foi considerada a posição do nó na sequência, o que vai fazer diferença para o tempo gasto pela MM configurada em realizar o processamento.
- **Mecanismos de simulação**
  - No AODV foi considerado o mecanismo de estabelecimento da rota considerando o *flooding* e as respectivas sinalizações (RREQ, RREP e ACK);
  - No TCNet foi considerado o mecanismo do *query* submetido a uma sequência com o objetivo de estabelecer uma rota.
- **Topologia da simulação**
  - O cenário considerado é fixo;
  - Os nós da rede estão distribuídos aleatoriamente numa área de 1000 m x 1000 m;
  - O nó *Fonte*: do AODV e nó SINK do TCNet inicializam a comunicação.
- **Comparação da latência aumentando-se progressivamente o número de nós da rede**
  - A Figura 4.16(a) mostra o cenário de decisão da rota no AODV. Usando-se o modo *Eventlog* do OMNeT++ pode-se observar os detalhes do *flooding* realizado pelo nó *Fonte*: host(0), na tentativa de encontrar uma rota para o nó *Destino*: (nó 15). A Figura 4.16(b) compara a latência dos cenários AODV e TCNet para uma rede de 16 nós, durante a consulta aos nós que representam maior demora no acesso, respectivamente: AODV (nó 15) e TCNet (nó 1). A latência do AODV é 40% maior do que a latência do TCNet, neste caso. Pode-se observar a distribuição da latência do TCNet conforme detalhado na Figura 4.16(b): tempo de processamento inicial do Sink = 0,300003 s, tempo de processamento dos nós da rota e tempo de processamento do último nó da rota (nó 1). O tempo gasto pelo AODV para estabelecer a rota está distribuído nas sinalizações: RREQ, RREP e ACK, até atingir uma estabilização temporária, onde novas atualizações ocorrerão através de *HELLOS* na rede via *flooding*.

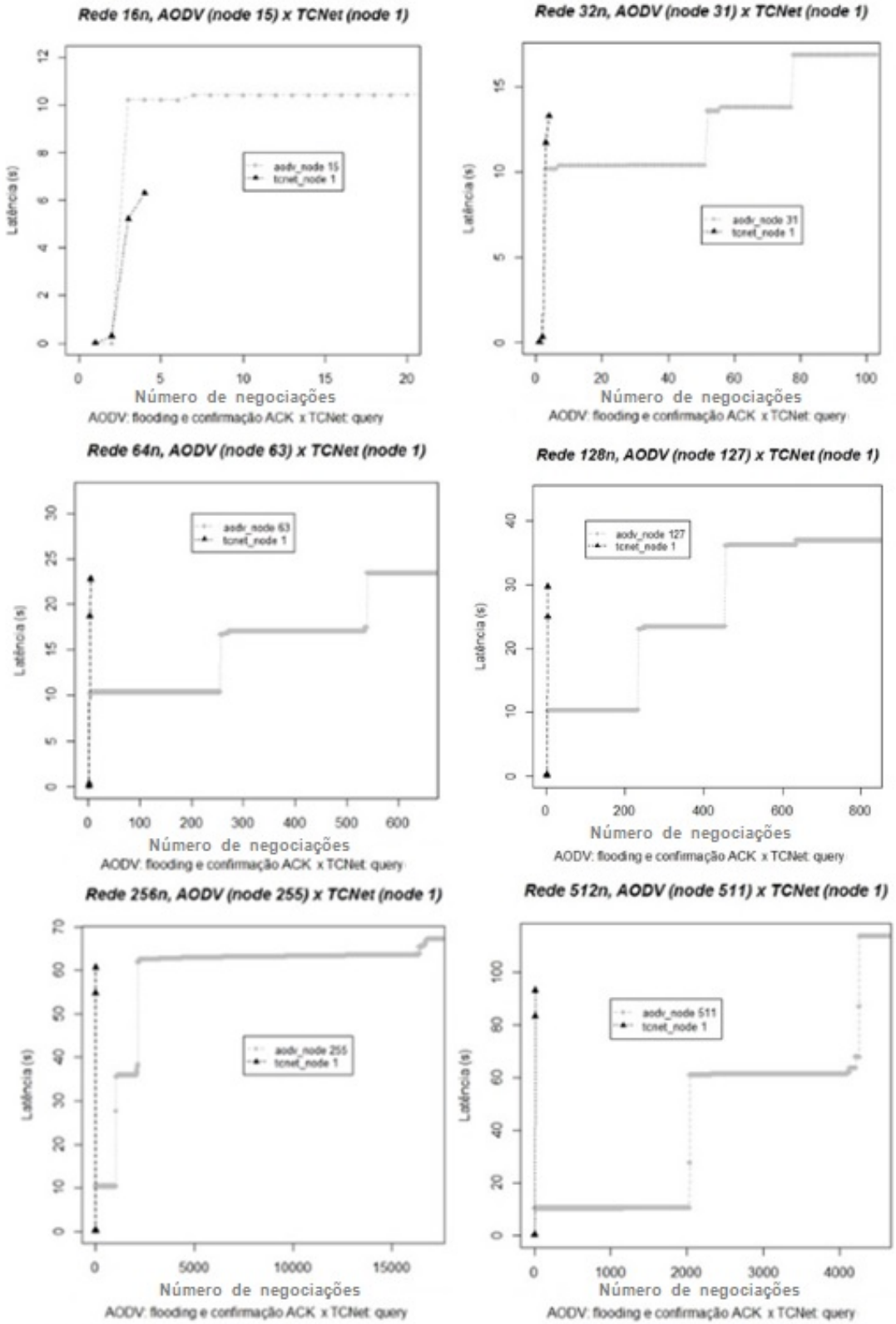
Figura 4.16: (a)Cenário mostrando o *flooding* do AODV em uma rede de 16 nós, obtida no modo *Eventlog* do OMNeT++; (b)Comparação da latência: AODV x TCNet para alcançar o nó com maior dificuldade da rede



Fonte: Autor

– Os resultados mostrados na Figura 4.17 referem-se à comparação das latências em se estabelecer as rotas para os nós considerados com maior dificuldade de acesso, à medida que a densidade de nós da rede aumenta.

Figura 4.17: Comparação da latência: AODV x TCNet à medida que ocorre aumento progressivo da quantidade de nós na rede



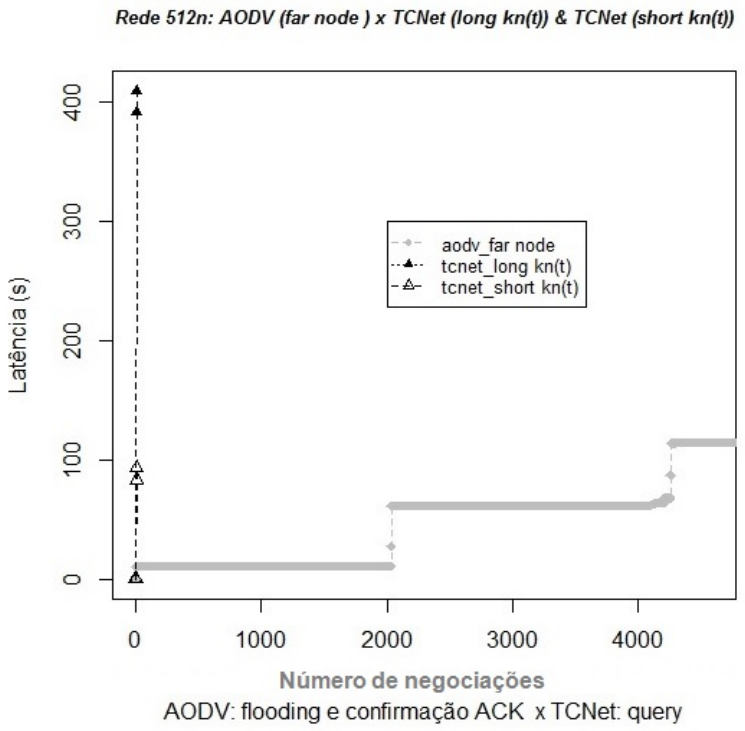
Fonte: autor

Tabela 4.3: Comparações do aumento da latência do TCNet e AODV, com o aumento da densidade de nós na rede

Dimensões da Rede	Latência do TCNet (s)	Latência do AODV (s)
16 nós	6,5	10,5
32 nós	13,3	17,5
64 nós	20,7	25
128 nós	29,9	40
256 nós	60,7	100
512 nós	93	250

– Os resultados levantados na Tabela 4.3 referem-se à comparação da latência entre o TCNet e o AODV, considerando-se o aumento geométricamente progressivo dos nós da rede. Verifica-se que a razão média do TCNet para a configuração da MM considerada (k/n = 1/2) é 1,68 vezes, enquanto a razão média do AODV é de 1,92 vezes, considerando comparações justas com relação à dificuldade de acesso ao nó considerado DESTINO. Esses valores podem ser melhorados em favor do TCNet utilizando-se melhores rotas estabelecidas pela sequência $k_n(t)$, como mostra a Figura 4.18 ou alterando-se o código convolucional de tal maneira que a treliça correspondente ofereça rotas mais curtas, assim o novo código será implementado por uma nova MM sem grande aumento de complexidade de implementação.

Figura 4.18: Cenário de comparação usando uma rede de 512 nós, na decisão pela rota mais curta: TCNet x AODV



Fonte: Autor

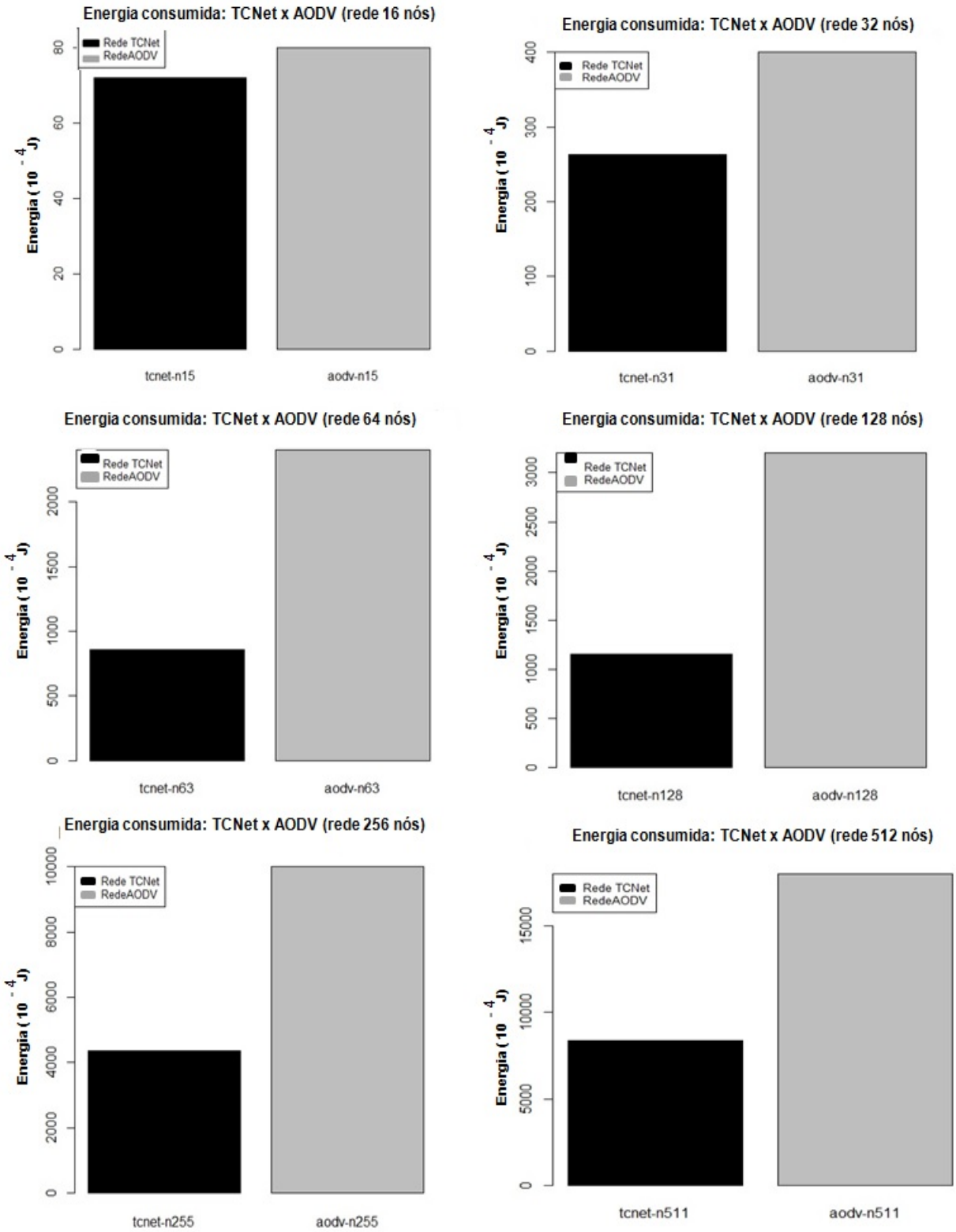
A Figura 4.18 mostra as opções do TCNet para decisão de rotas com menor latência. O cenário usa os resultados de uma rede com 512 nós e as opções de sequências $k_n(t)$: *long* - $kn(t)$ ou *short* - $kn(t)$, que irá ajudar na decisão da rota com menor latência. Enquanto o AODV após estabelecer uma rota para nó DESTINO (*far node*), depende de novas atualizações que possam ocorrer na rede, reduzindo assim suas opções na escolha da rota com menor latência.

- **Comparação do consumo de energia aumentando-se progressivamente o número de nós da rede**

  A energia consumida nas redes WSN é um fator fundamental do projeto devido às limitações das fontes de energia dos nós da rede. Este trabalho faz uma comparação do consumo de energia de uma rede TCNet com uma rede AODV, procurando usar os mesmos parâmetros de comparação. A Figura 4.19 mostra uma comparação do consumo de energia consumida usando as mesmas dimensões das redes.

  A Tabela 4.4 mostra uma comparação do consumo de energia entre o TCNet e o AODV, considerando-se o aumento geometricamente progressivo dos nós da rede. Verifica-se que a razão média do aumento do consumo de energia no TCNet é de 2,7, enquanto no AODV o aumento médio da razão da energia consumida é de 3,6.

Figura 4.19: Comparação do consumo de energia: AODV x TCNet à medida que ocorra aumento progressivo da quantidade de nós na rede



Fonte: Autor

Tabela 4.4: Comparações do aumento da energia consumida no TCNet e AODV, com o aumento da densidade de nós na rede

Dimensões da Rede	Energia do TCNet (10^{-4} J)	Energia do AODV (10^{-4} J)
16 nós	72	80
32 nós	260	400
64 nós	800	2500
128 nós	1200	3200
256 nós	4200	10000
512 nós	8000	19000

4.4 ROBUSTEZ DO ALGORITMO TCNet NA PRESENÇA DE FALHA DE NÓS E COLISÃO DE PACOTES

Será discutida a robustez do algoritmo TCNet em caso de falha de nós de uma rede configurada sobre a treliça, aproveitando a capacidade de regeneração da treliça. Será mostrada a eficiência do processo através de medidas da latência.

4.4.1 Capacidade de recuperação da rede usando a regeneração da treliça

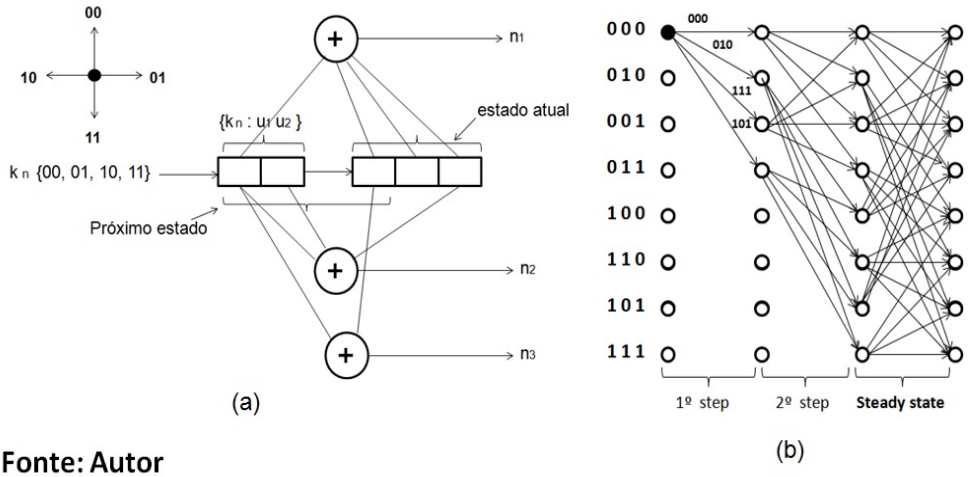
A capacidade do TCNet em estabelecer rotas sobre uma treliça está associada à complexidade da máquina de Mealy. Os cenários apresentados neste trabalho utilizam uma configuração de MM com taxa k/n=1/2 devido à simplificação de hardware, com o objetivo de atender às características dos nós das WSNs. Outra estratégia para atender às simplificações das MM foi a opção pela *decisão abrupta* (*hard decision*) que utiliza a *distância de Hamming* [37]. Embora a MM utilizada como referência atenda às simplificações de hardware exigidas para este trabalho, isso reduz as alternativas de ramos que saem do nó, limitando a capacidade de decisão da treliça como veremos mais à frente.

As "distâncias" são parâmetros importantes a serem considerados nas decisões utilizadas pelo decodificador de treliça, como mostra a Seção 2.4 deste trabalho, onde são explicadas a *distância de Hamming* (d_{Ham}), *distância livre* (d_{livre}) e *distância lógica* (d_{Log}). As MM que utilizam a (d_{Ham}) do tipo (*hard decision*) reduzem as alternativas dos ramos que saem de um nó, limitando assim as possibilidades de rotas. Uma alternativa para ampliar as opções de ligações entre os nós de uma rede é utilizar a suavização da (d_{Ham}), através do conceito de *distância livre* (d_{livre}), conforme foi demonstrado na Seção 2.4.

Um exemplo de configuração de MM geradora, proposta por Ungerboek [42], conforme mostra a Figura 4.20, representa uma MM com taxa k/n=2/3, onde a sequência $k_n(t)$ é composta por palavras com $\nu = 2$ símbolos: $\{u_1u_2\}$ deslocando-se nos registradores da MM com saídas $\{n_1n_2n_3\}$, resultando no aumento da capacidade de ligações dos nós conforme mostra a treliça equivalente da

MM considerada. Um fator importante desta configuração é o aumento da capacidade de estabelecer rotas sobre a treliça e a redução do tempo de estabilização da treliça, o que significa atingir o *steady state* em poucos passos, conforme a Figura 4.20(b)

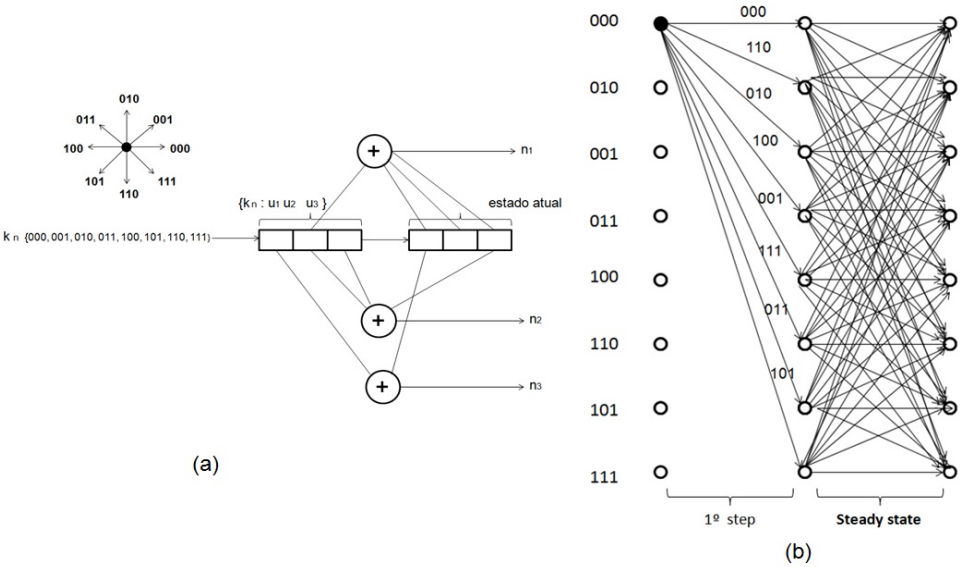
Figura 4.20: (a)Configuração da MM com taxa k/n=2/3, mostrando detalhe da capacidade de ligações do nó; (b)Diagrama de treliça resultante com $2^{\nu} = 4$ ramos ligando os nós da treliça e o instante em que a treliça atinge o *steady state*



Fonte: Autor

Outro exemplo mostrado na Figura 4.21, representa uma MM com taxa k/n=3/3, onde a sequência $k_n(t)$ é composta por palavras com $\nu = 3$ símbolos: $\{u_1u_2u_3\}$ deslocando-se nos registradores da MM com saídas $\{n_1n_2n_3\}$. O acréscimo de 1 registrador na configuração da MM aumenta a capacidade de decodificação da treliça, resultando numa configuração *full mesh*, eliminando assim o problema do retardo na estabilização da treliça, conforme a Figura 4.21(b).

Figura 4.21: (a)Configuração da MM com taxa k/n=3/3, mostrando detalhe da capacidade de ligações do nó; (b)Diagrama de treliça resultante com $2^{\nu} = 8$ ramos ligando os nós da treliça e o instante em que a treliça atinge o *steady state*



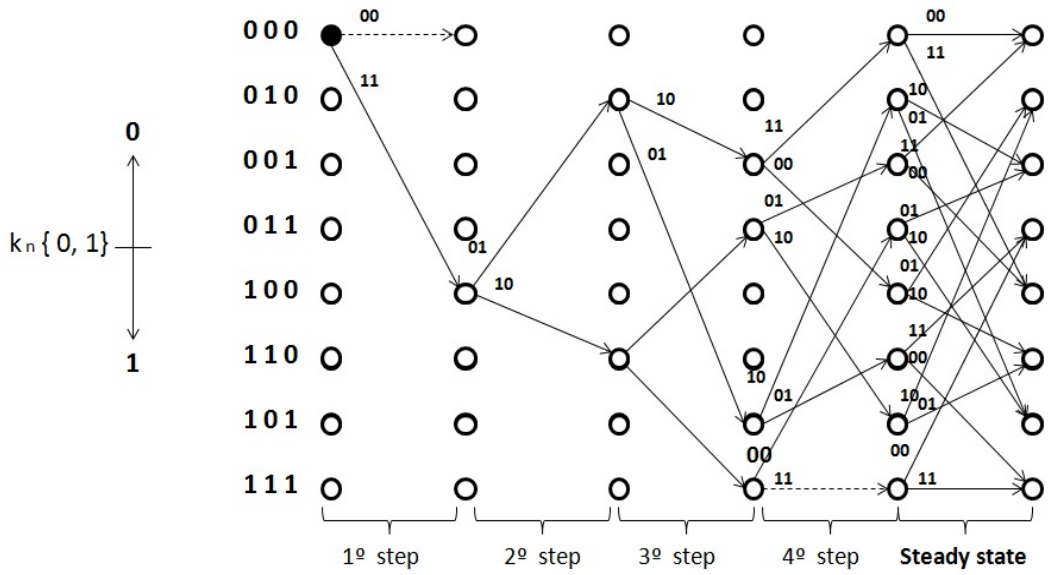
Fonte: Autor

4.4.2 Cenário de simulação com falhas de nós na rede considerando uma mesma área de cobertura

O cenário inicial considerado utiliza uma MM com taxa k/n=1/2 submetida a uma sequência $k_n(t) = 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0$ que permite a alcançabilidade de uma rede com 8 nós, de modo que se possa realizar comparações com os resultados obtidos. Serão consideradas as limitações da máquina na escolha dos nós com falhas, utilizando-se os critérios de exceções das regiões da treliça onde não foi alcançada a estabilização da treliça ou *steady state*, conforme a Figura 4.22:

- 1º step: O nó(100) é imprescindível para inicialização da rota;
- 2º step: Os nós (010) e (110) pertencem à inicialização da treliça, apenas um dos nós pode falhar;
- 3º step: Os nós (001), (011), (101) e (111) pertencem à inicialização da treliça, apenas dois nós podem falhar simultaneamente.

Figura 4.22: Configuração da treliça gerada pela MM com taxa k/n=1/2, mostrando a estabilização da treliça (*steady state*) após o 4º step



Fonte: Autor

- **Falha de nós específicos em uma rede com 8 nós**

As medidas de eficiência da rede, na presença de falhas de nós, será mostrada através da latência durante recuperação da rota. Será utilizada a equação (4.1) com o acréscimo do parâmetro *time out* (t_{out}), representando o tempo que a rede espera pela resposta do nó solicitado até que possa ser assumido por outro nó da rede:

- t_{out} = *0,5 s*

Utilizando-se a equação (4.4) que define a latência total da rede para o caso de falha do nó, pode-se obter a contribuição dos nós para latência da rede considerada.

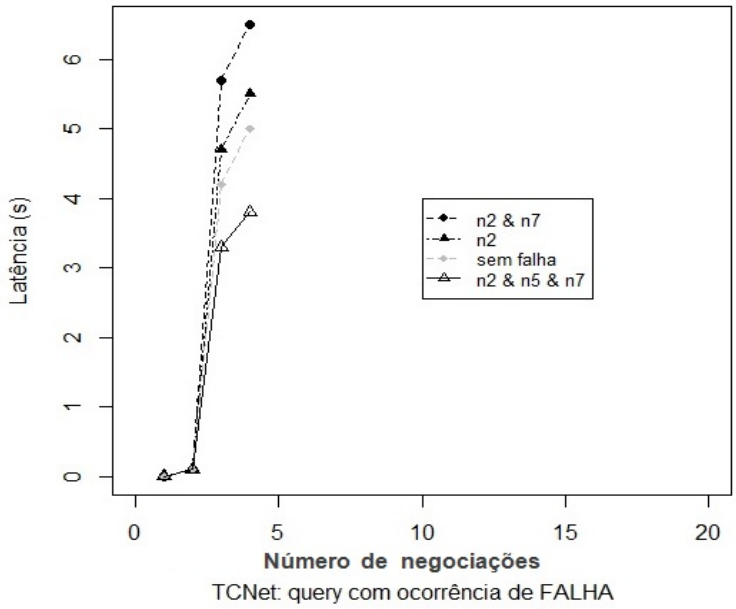
$$\Sigma T_{Lf} = t_p + t_c + t_g + t_{out} \tag{4.4}$$

Para o cenário considerado de falhas numa rede com 8 nós, foram especificados os nós que no caso de falha possam comprometer a recuperação da treliça. Nesse caso foram selecionados nós localizados na região instável da treliça como mostrado na Figura 4.22:

- Falha de 1 nó: nó (010): 2 localizado no 2º step da inicialização da treliça;
- Falha de 2 nós simultaneamente: nó (010): 2 e nó (111): 7, localizados nos steps 3º e 4º da treliça;
- Falha de 3 nós simultaneamente: nó (010): 2, nó (101): 5 e nó (111): 7, localizados nos steps 2º, 3º e 4º da treliça.

A Figura 4.23 mostra a comparação da latência da rede na recuperação da rota, em relação ao caso da rota sem falha:

Figura 4.23: Comparação da latência da rede na recuperação da rota para os casos de falhas: com 1 nó, 2 nós e 3 nós



Fonte: Autor

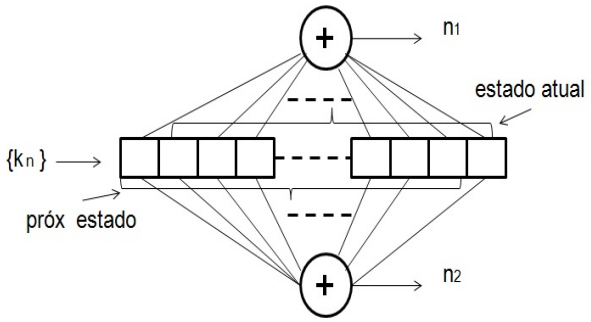
- No caso da falha de 1 nó ocorre um aumento de 10% na latência em relação à rede sem falha, devido o acréscimo do (t_{out}) na rede;
- No caso da falha de 2 nós simultâneos, ocorre um amento de 25% na latência em relação à rede sem falha;

– No caso da falha de 3 nós ao mesmo tempo, deve-se levar em conta que houve uma redução da rede em 50%, ainda assim a rede se recupera e completa o *query* com 70% da latência em relação à rede sem falha mesmo considerando o (t_{out}).

- **Falha de nós específicos em redes estendidas**

Para os casos de redes com maior densidade de nós foram simulados cenários que representassem os nós localizados nas regiões críticas para a recuperação da treliça. Considerando-se que foi utilizada a mesma configuração básica da MM, ou seja, máquinas com taxa k/n=1/2, acrescentando-se progressivamente registradores e ligações das portas 'XOR', pode-se obter resultados que podem servir de comparações para medidas de eficiência do TCNet, conforme a Figura 4.24.

Figura 4.24: Máquina de Mealy (MM) para os casos de generalização da rede



Fonte: Autor

Os critérios para se obter os resultados para as redes expandidas, obedeceram a análise inicial para uma rede com 8 nós, onde foram identificadas as regiões críticas da treliça antes de alcançar o *steady state*:

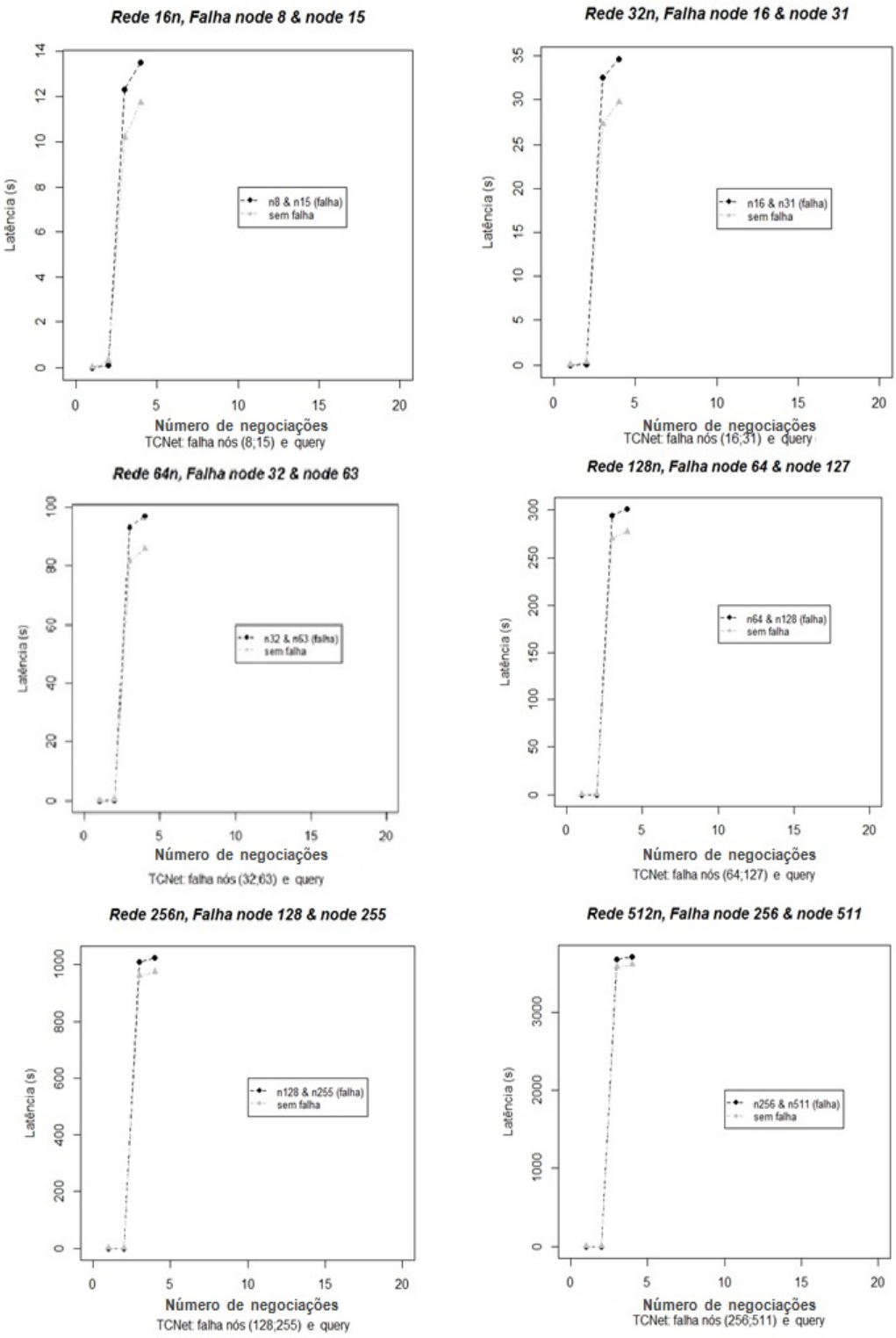
– Nó localizado no 2º step da região de estabilização da treliça e possua a menor *distância lógica* (d_{Log}) em relação ao Sink;

– Nó com a maior *distância lógica* (d_{Log}) em relação ao Sink.

Considerando-se as dificuldades na seleção dos nós à medida que a densidade da rede aumenta, optou-se pela seleção do nó que irá falhar, considerando-se a posição da *distância lógica* (d_{Log}) do nó na rede. Os resultados mostrados na Figura 4.25 referem-se às latências, à medida que a densidade de nós da rede aumenta, obtidas em relação às respectivas rotas sem falha.

Os nós foram selecionados de acordo com as densidades das redes, podendo-se analisar os resultados das latências resultantes pela Tabela 4.5, onde mostra uma redução progressiva da latência para as condições consideradas, à medida que aumenta a quantidade de nós da rede.

Figura 4.25: Comparação da latência e recuperação da rota para redes com maior densidade de nós



Fonte: Autor

Tabela 4.5: Comparações da latência com o aumento da densidade da rede com falhas de nós

Dimensões da Rede	Nós considerados	Acréscimo da Latência (%)
16 nós	$n8$ & 15	20 %
32 nós	$n16$ & 31	18,8 %
64 nós	$n32$ & 63	12,5 %
128 nós	$n64$ & 127	7,3 %
256 nós	$n128$ & 255	5,5 %
512 nós	$n256$ & 511	2,7 %

4.4.3 Cenários que podem resultar em colisões de pacotes

Os principais desafios ao se projetar um algoritmo de roteamento para redes *ad hocs* segundo [7], além dos problemas relacionados à: aleatoriedade dos nós, contenção de recursos, transmissão *brodcast*, estão o terminal oculto e exposto.

- **Considerações sobre o Terminal oculto pelo TCNet**

O terminal oculto contribui com a degradação da taxa de transferência de dados na rede devido às colisões ocasionadas. A prática adotada pelos protocolos convencionais para contornar o problema, consiste na técnica de *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA) que se resume às sinalizações utilizando os controles: *Request to Send, Clear to Send, Data, Acknowledgment* (RTS, CTS, Data e ACK), representando uma solução complexa para a limitada capacidade das WSNs, expondo assim a limitação do protocolo CSMA na solução do problema.

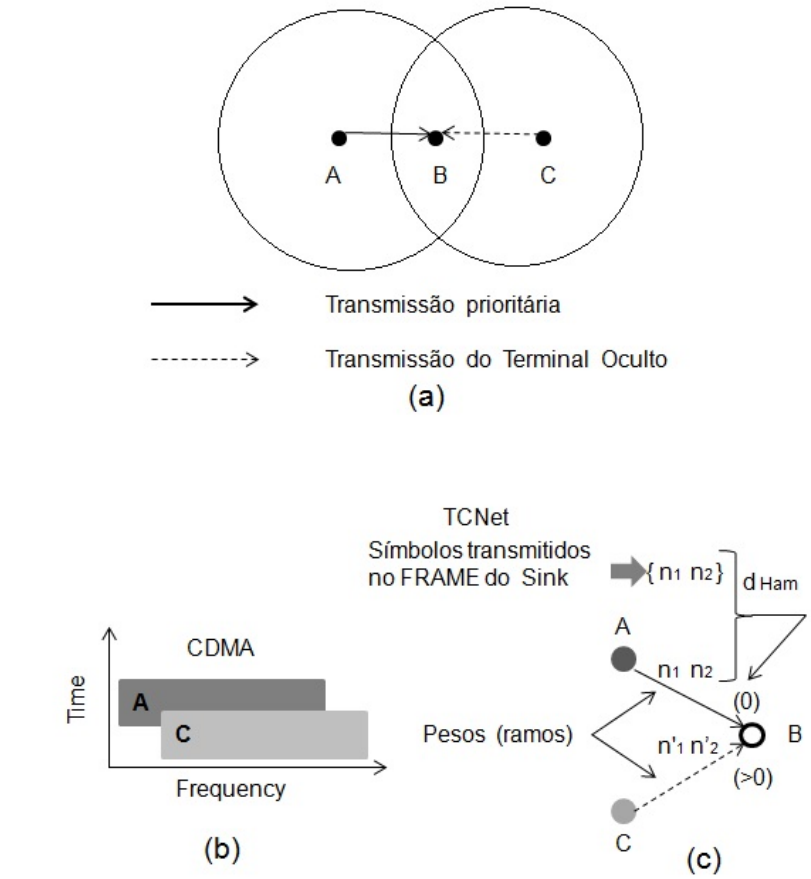
A inovação proposta pelo TCNet consiste na decisão realizada pelo próprio nó, em fazer parte da rota, utilizando máquinas de estados finitos, sem a necessidade de mensagens de sinalização da rede, como: *Route Request* e *Route Reply*. Ainda existe a possibilidade de usar códigos baseados em diversidade como *Code Division Multiple Access* (CDMA) podendo ocorrer o compartilhamento de canal pelos nós.

Considerando que o CDMA possibilita a diversidade a nível de camada física, permite o nó sintonizar diferentes transmissões. A diversidade a nível de camada física é um requisito essencial à implementação das WSNs e portanto a utilização do CDMA não introduz impacto no consumo de energia que seja considerável, sendo um fator comum às várias alternativas. Associando-se a capacidade do algoritmo TCNet em utilizar estratégias que permitem o nó decidir se ele pertence a uma determinada rota, resulta numa combinação que favorece no atendimento de um *query* em um determinado instante.

A Figura 4.26(a) mostra um cenário clássico de terminal oculto usando decisão CSMA/CA, onde os nós A e C transmitem no mesmo instante para o nó B, ocorrendo colisão de pacotes, devido os nós A e C estarem na situação de ocultos ente si.

A solução adotada pelo TCNet usa o conceito de decisão tomada pelo próprio nó, baseado no algoritmo de Viterbi [37], decodifica a sequência recebida estimando a mínima distância de Hamming entre os símbolos da sequência enviada pelo Sink e o peso dos ramos da treliça conforme mostra a Figura 4.26(c), onde o nó B no modo de recepção decide pela $d_{\text{Ham}} = 0$ como resultado entre a sequência de símbolos emitido pelo Sink: $(n_1 n_2)$ e o peso do ramo ligado ao nó A, descartando assim a transmissão enviada pelo nó C devido a $d_{\text{Ham}} > 0$. A solução de compartilhamento do canal implica na utilização da diversidade a nível de camada física com o uso do *Code Division Multiple Acces* (CDMA) conforme a Figura 4.26(b).

Figura 4.26: (a) Cenário clássico do terminal oculto usando (CSMA/CA); (b) Solução usando CDMA possibilitando o compartilhamento do mesmo canal pelos pacotes de dados A e C; (c) Mecanismo de decisão do TCNet usando a decisão pela distância de Hamming



Fonte: Autor

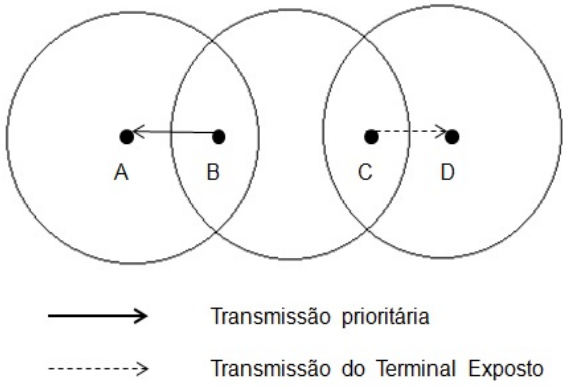
- **Considerações sobre o terminal exposto pelo TCNet**

Outro cenário possível de ocorrer colisão nas redes *ad hocs*, é a existência de nós pertencentes à rede, mas que estão fora da área de cobertura, ficando impossibilitados de receberem a sinalização CSMA/CA para evitar colisões ou esperas desnecessárias por oportunidades de transmissão, configurando assim o caso do terminal exposto.

A Figura 4.27 mostra o problema do terminal exposto, onde uma transmissão do nó B está em execução para o nó A e o nó C necessita iniciar uma transmissão para o nó D. Nos casos

das redes *ad hocs* convencionais recaem nas interações com a camada MAC, com a utilização do CSMA/CA, onde o nó C detecta que o meio está ocupado, permanecendo assim em espera. Nesse caso o nó A estando fora de alcance de C é desnecessária a espera de C em relação ao nó B.

Figura 4.27: (a) Cenário clássico do terminal exposto usando (CSMA/CA)



Fonte: Autor

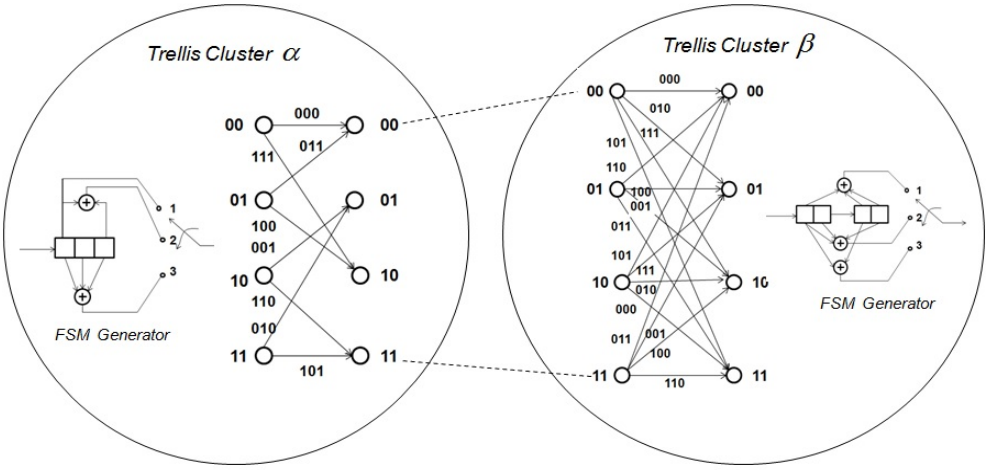
Os cenários de WSN são normalmente aplicados a centenas de nós distribuídos em grandes áreas, sugerindo a subdivisão dessas áreas em *clusters*[4]. O uso do TCNet nesses casos aumenta as alternativas de conexões possibilitando a busca de novas rotas devido à auto configuração da treliça, tornando-se desnecessário o uso dos protocolos de sinalização CSMA/CA.

A Figura 4.28 mostra um cenário com dois *clusters*: α e β configurados por máquinas de Mealy distintas, possibilitando a utilização de diferentes rotas nos *clusters*. No caso de ocorrer o recobrimento de áreas vizinhas, as rotas por serem independentes, será um mecanismo que poderá ser útil na análise do terminal exposto pelo compartilhamento do mesmo canal por múltiplos nós.

A Figura 4.29 mostra rotas distintas geradas por máquinas de Mealy ou *Finite State Machine* (FSM) em diferentes *clusters*, onde os respectivos Sink inicializam os *queries* usando a mesma sequência $k_n(t) = \{1100\}$. Nesse caso a ocorrência do terminal exposto é real. Considere por exemplo que o nó B está respondendo ao *query* solicitado pelo Sink de α e o nó C necessita responder o *query* solicitado pelo Sink de β, assim como no caso do "Terminal oculto", o CDMA possibilita diversidade a nível de camada física, possibilitando o nó B compartilhar o canal, enquanto o TCNet possibilita o nó C identificar que está sendo solicitado pelo Sink de β, eliminando esperas desnecessárias, otimizando assim o uso do canal.

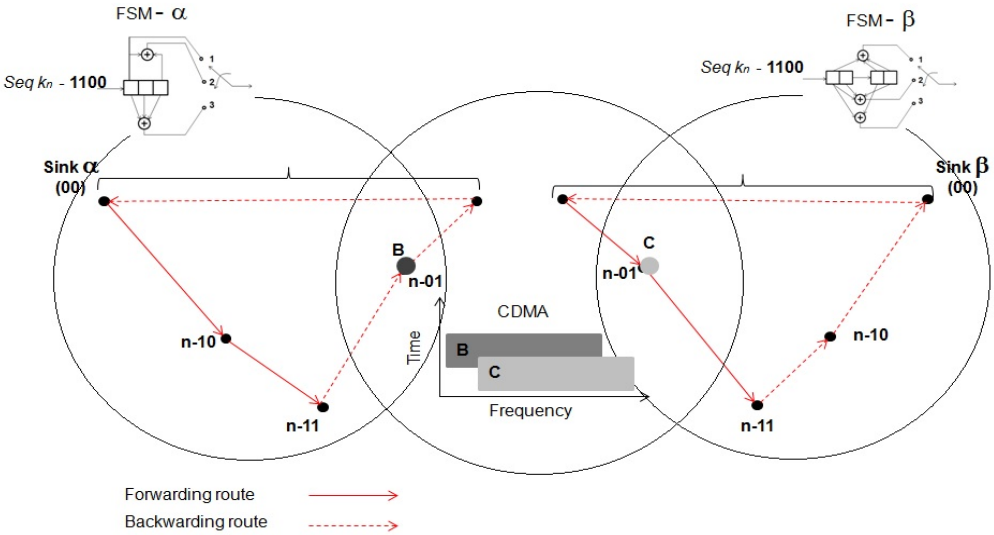
[4]Uma arquitetura *clustered* organiza grupos de sensores coordenados por um *cluster head* que gerencia as mensagens trocadas entre os nós e envia para uma Estação Base (BS), configurada como *access point* (AC) conectado a uma rede, [7].

Figura 4.28: Os *clusters* α e β possuem rotas estabelecidas pelas respectivas máquinas de Mealy



Fonte: Autor

Figura 4.29: Solução do algoritmo TCNet para o caso do terminal exposto, nó C. Enquanto o nó B atende o *query* originado pelo Sink de α, o nó C compartilha o mesmo canal usando diversidade (CDMA), de modo a atender o *query* gerado pelo Sink de β



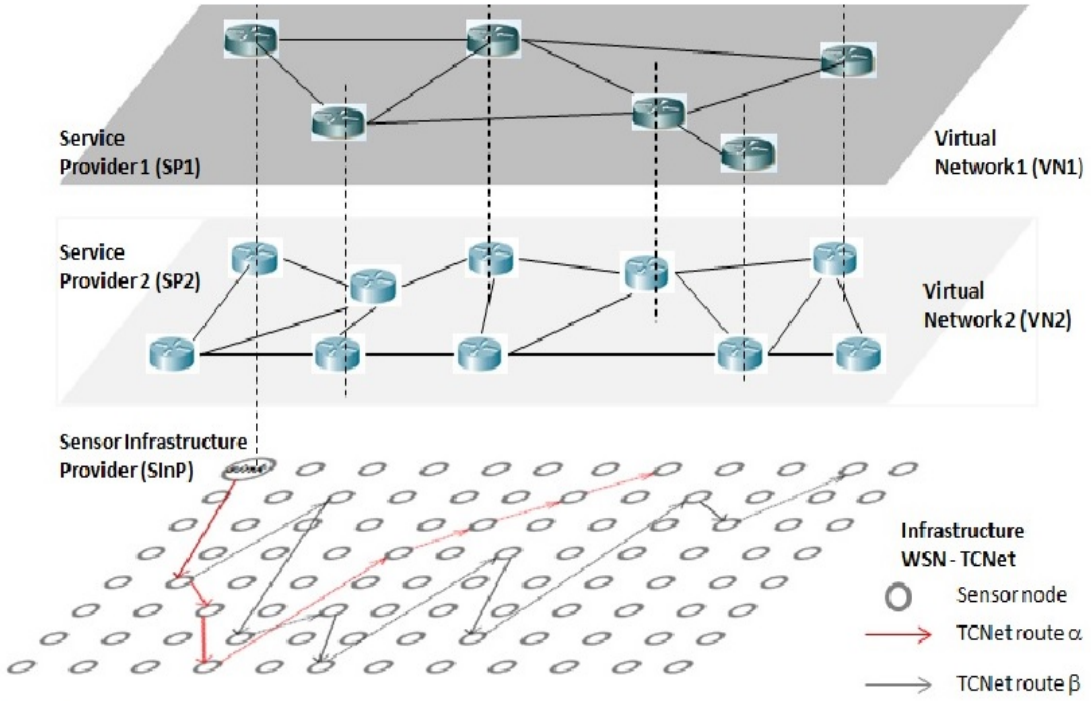
Fonte: Autor

4.5 POTENCIAL DE APLICAÇÃO DO ALGORITMO TCNet

- **Aplicação do TCNet em cenários de Virtualização de Redes de Sensores**

A aplicação do conceito do algoritmo TCNet em casos de Virtualização de Redes de Sensores (VSN) [47], [48] é facilitado devido a flexibilidade do algoritmo TCNet em aplicações de gerenciamento de rotas. O conceito do TCNet pode ser mais uma ferramenta a ser utilizada pelos *Sensor Infrastructure Providers* (SInPs), possibilitando "*end to end services*" conforme a Figura 4.30.

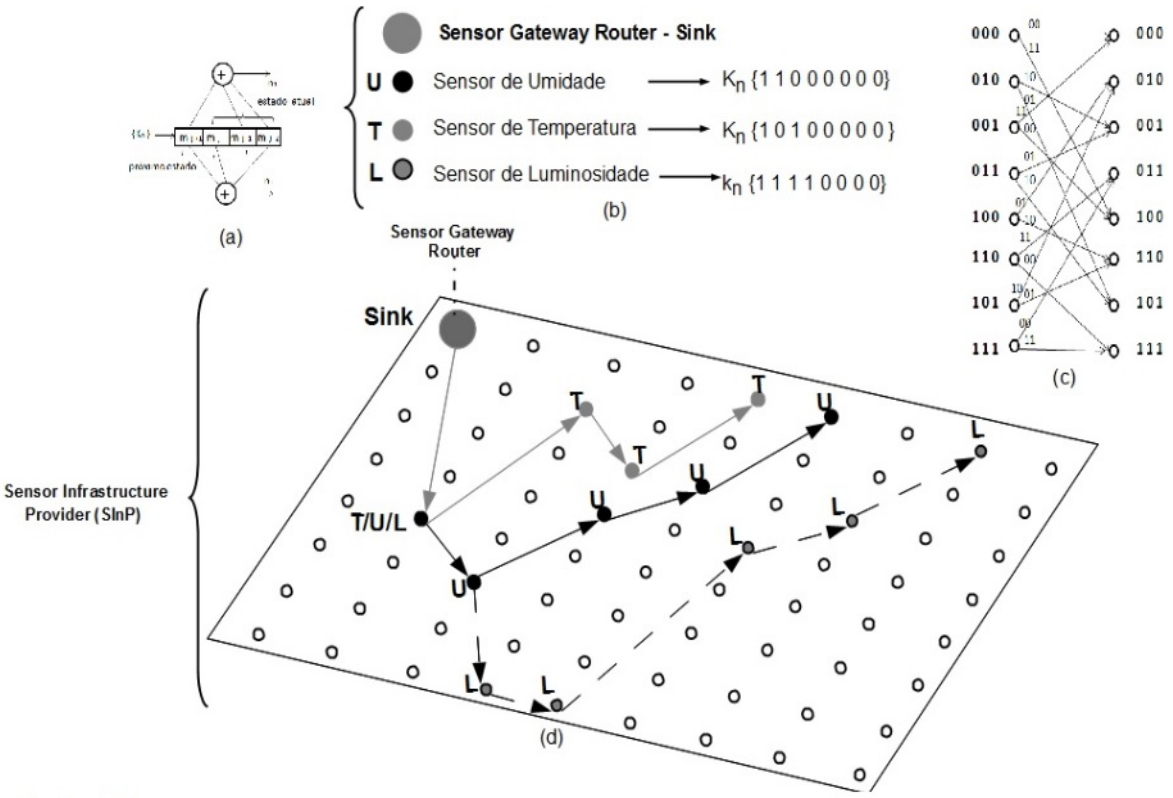
Figura 4.30: Cenário de ambiente de Virtualização de Redes de sensores considerando *Infrastructure WSN* TCNet



Fonte: Autor

A Figura 4.31 mostra um exemplo de aplicação do TCNet em cenários VSN. O ambiente VSN possui um conjunto heterogêneo de sensores coexistindo no mesmo *Sensor Infastructure Provider* (SInP) gerenciado por um *Sensor Gateway Router* localizado no Sink. O TCNet possibilita a ocorrência de diferentes *queries* na mesma rede, associando sequências a grupos de diferentes sensores, como explica a Figura 4.31.

Figura 4.31: (a)FSM com taxa k/n = 1/2; (b)Sequências relativas às rotas dos sensores; (c)Diagrama de Treliça correspondente à FSM; (d)Grupos de sensores no SInP

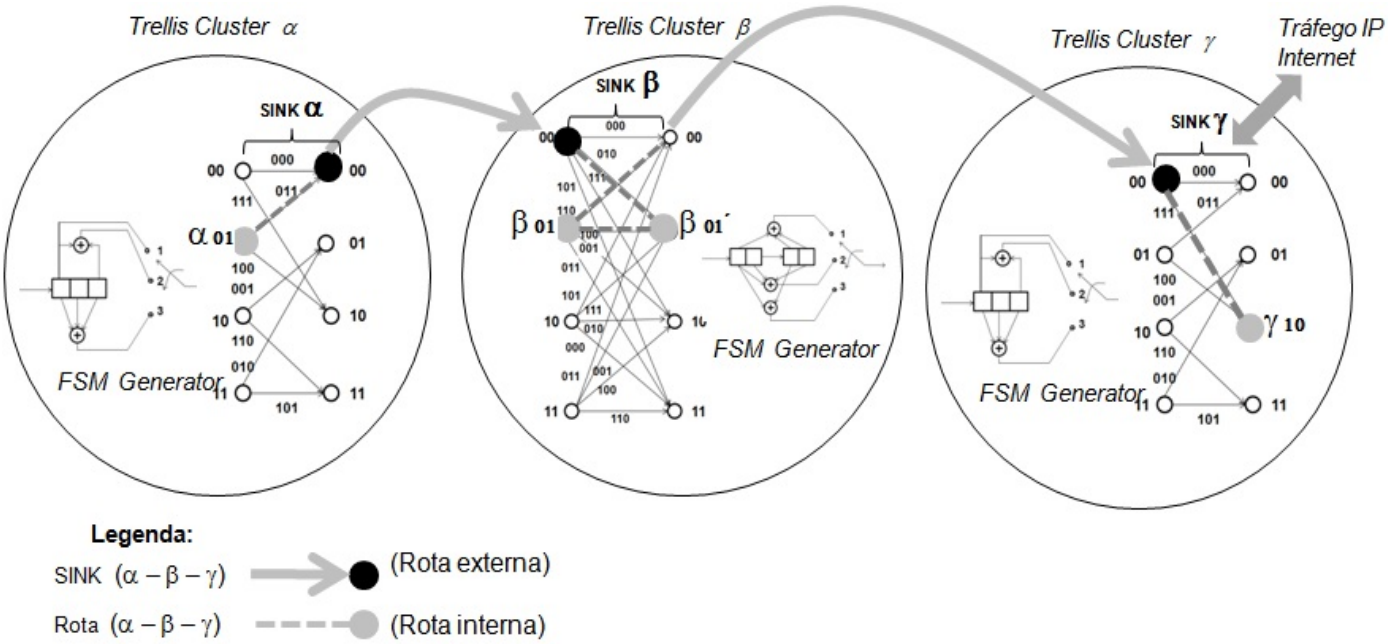


Fonte: Autor

- **Aplicação do TCNet em cenários de *clusters* de nós**

A Figura 4.32 mostra a vantagem da aplicação do TCNet nas ampliações das rotas das redes, onde grupos de nós compondo *clusters*, formados por diferentes configurações de FSM capazes de gerar diferentes treliças, permite ampliação das rotas de modo a atender grandes áreas de coberturas *wireless*, conforme mostra a Figura. Os nós SINK: α, β e γ possibilitam a interligação dos *clusters*, administrando os trechos das rotas nas suas respectivas treliças.

Figura 4.32: *Clusters* correspondentes às FSM: α, β e γ com a representação da migração das rotas entre os *Clusters*



Fonte: Autor

O cenário mostra um exemplo de dados coletados dos nós (α_{01}, β_{01} e γ_{10}), pertencentes à diferentes *clusters*, e sendo transmitidos pelos seus respectivos SINKs, de modo a serem agregados ao tráfego IP pelo SINK γ com a função de *gateway*.

5 CONCLUSÕES E TRABALHOS FUTUROS

O Capítulo apresenta as principais contribuições deste trabalho, assim como sugestões para trabalhos futuros.

5.1 COMENTÁRIOS E CONTRIBUIÇÕES

O objetivo principal deste trabalho é a proposta de um novo algoritmo e protocolo de encaminhamento de pacotes que seja específico para redes com recursos limitados de processamento, comunicação e energia como é o caso das redes WSNs. Como solução foi proposto um novo paradigma baseado em códigos convolucionais.

Para as medidas de desempenho, foram utilizados os parâmetros de latência e consumo de energia, comparados com o desempenho do AODV, um protocolo bastante adotado em aplicações em WSN.

As simulações foram realizadas utilizando-se o ambiente de simulação OMNeT++ e seus *frameworks* disponibilizados nas bibliotecas do simulador para os casos do AODV e tendo que serem implementadas as rotinas para o TCNet conforme mostra o Anexo C desta pesquisa.

As configurações das Máquinas de Mealy - (MM) utilizadas, para geração das rotas das redes, foram modeladas no ambiente OMNet++ utilizando-se o C++ orientado a objetos. Para essas máquinas foram utilizadas as configurações mais simples com taxas k/n $= 1/2$ de modo a serem utilizados poucos recursos de hardware. Isto teve reflexo numa piora dos resultados de latência, à medida que ocorreu um aumento na quantidade de nós da rede, numa razão média de 1,68 vezes para o TCNet enquanto o AODV nas mesmas condições apresentou uma razão média de 1,92 vezes. A latência é um parâmetro importante a ser considerado em qualquer sistema, embora os resultados obtidos com o TCNet podem ser melhorados com uma simples alteração da sequência $k_n(t)$, ou alterando-se a configuração da MM para valores maiores de taxas, como: k/n $= 2/3$ ou k/n $= 3/3$. Outro fator importante a ser considerado é, mesmo com a configuração simplificada da MM utilizada, o TCNet obteve resultados melhores em economia de energia em relação ao AODV, à medida que ocorreu o aumento de nós na rede, representando uma razão média do aumento do consumo de energia de 2,7

para o TCNet, enquanto que o AODV apresenta uma razão de consumo de 3,6 vezes. O aspecto de consumo de energia é muito importante para as WSNs devido às limitações de recursos utilizados e muitas vezes inviável a substituição das baterias, devido aos nós das redes estarem em locais de difícil acesso para a realização de manutenção. Analisando-se o TCNet como um sistema de geração de rotas, pode-se considerar que o algoritmo se aproveita do conhecimento prévio da rota, obtido a partir de *algoritms for multirestictive routing*, [10], enquanto o AODV faz um levantamento periódico da rota através de HELLOs, sobrecarregando o sistema. A vantagem do conhecimento prévio da rota não pode ser utilizada pelo AODV.

Quanto ao alerta da falha de nós, no TCNet isso ocorre sempre ao completar um *query*, pela consulta ao FRAME no campo *payload*, o que facilita tomadas de decisões mais rápidas pelo SINK, com a possibilidade de reconfiguração da rota. No caso do AODV, a descoberta da falha do nó ocorre após atualizações das tabelas de rotas, após os HELLOs periódicos do sistema.

Tentativas de simulações práticas para validação do TCNet foram realizadas, não havendo tempo hábil para conclusões e comparações com a teoria, porém os hardwares e códigos fontes utilizados estão comentados no Anexo C.

5.2 TRABALHOS FUTUROS

Ao longo desta pesquisa inúmeras ideias surgiram, porém não houve tempo hábil para executá-las, exigindo novo trabalho de pesquisa. Algumas sugestões para trabalhos futuros são:

- Estudar os casos de MM com taxas $k/n = 2/3$ e $k/n = 3/3$, de modo a avaliar melhorias na latência e energia consumida;
- Fazer avaliações do TCNet considerando outros parâmetros da rede, como: vazão e perda de pacotes;
- Análise do algoritmo TCNet considerando a mobilidade da rede, de modo a avaliar os efeitos dos enlaces *wireless* no estabelecimento das rotas;
- Estudar outros protocolos de roteamento para comparação com o TCNet;
- Otimizar a busca de rotas, no caso da falha do nó.
- Conclusão da validação prática do TCNet iniciada nesta pesquisa e documentada nos Anexos A e B.

Referências

[1] A.BOUKERCHE. *Algorithms and protocols for wireless and mobile ad hoc networks*. [S.l.]: Hoboken,N.J.:Wiley, 2009.

[2] J.KAHN; R.KATZ; PISTER, K. Next century challenges: mobile networking for "smart dust". *ACM/IEEE International Conference on Mobile Computing and Networking*, p. 271–272, 1999.

[3] J.POLASTRE; R.SZEWCZYK; D.CULLER. Enabling ultra-low power wireless research. *Proc.IPSN SPOTS 05*, 2005.

[4] V.OVIDIU; P.FRIESS. *Internet of Things Global Thechnological and Societal Trends*. [S.l.]: River Publishes, 2011. ISBN 978-87-92329-67-7.

[5] J.F.KUROSE; K.W.ROSS. *Computer Networking A Top-Down Approach Featuring the internet*. fifth. EUA: England: Addison-Wesley, 2009.

[6] GROUP, I. R. W. *ROLL Working Group Routing Over Low-Power and Lossy Networks*. 2009. Disponível em: <http://www.ietf.org/dyn/wg/charter/roll-charter.html>.

[7] MURTHY, C. R.; B.S.MANOJ. *Ad Hoc Wireless Networks - Architectures and Protocols*. 6th. ed. [S.l.]: Prentice Hall, 2008.

[8] HAN, C.; HARROLD, T.; ARMOUR, S.; KRIKIDIS, I.; VIDEV, S.; GRANT, P.; HAAS, H.; THOMPSON, J.; KU, I.; WANG, C.-X.; LE, T. A.; NAKHAI, M.; ZHANG, J.; HANZO, L. Green radio: radio techniques to enable energy-efficient wireless networks. *IEEE Communications Magazine*, v. 49, n. 6, p. 46–54, Jun. 2011. ISSN 0163-6804.

[9] LANEMAN, J. N. *Cooperative diversity in wireless networks: algorithms and architectures*. Tese (Doutorado) — Massachusetts Institute of Technology, Cambridge, MA, 2002.

[10] W.HERMAN. http://www.teses.usp.br. acesso em: maio 2014. In: *Algebraic formulation for modeling algorithms for multirestictive routing hop by hop*. [S.l.: s.n.], 2008.

[11] J.HOPCROFT; J.ULMAN. *Introduction to Automata Theory, Languages and Computation*. [S.l.]: Addison Wesley, 1955.

[12] A.J.VITERBI. Error bounds for convolutional codes and an asynmptotically optimal decoding algorithm. *IEEE Trans. Inform. Theory*, v. 13, p. 260–269, 1967. ISSN 1553-877X.

[13] G.GRATZER. *General Lattice Theory*. [S.l.]: Academic Press, Inc, 1978.

[14] C.E.PERKINS; E.BELDINGROYER; S.DAS. *Ad hoc on-demand distance vector-(AODV) routing*. 2003. Disponível em: <http://www.www.rfc-editor.org/rfc/rfc3561.txt. Acesso em: maio 2014>.

[15] CHO, S.-R.; CHOI, W.; KO, Y.-J.; AHN, J.-Y. Coordinated multipoint transmission in LTE-advanced. In: *Cooperative Cellular Wireless Networks*. Cambridge, Reino Unido: Cambridge University Press, 2011. p. 495–513. ISBN 978-0-521-76712-5.

[16] ROST, P.; FETTWEIS, G. Green communications in cellular networks with fixed relay nodes. In: *Cooperative Cellular Wireless Networks*. Cambridge, Reino Unido: Cambridge University Press, 2011. p. 300–323. ISBN 978-0-521-76712-5.

[17] MOY, J. *Open Shortest Path First*. 1988. Disponível em: <http:www.rfc-editor.org/rfc/rfc2328.txt. Acesso em: maio 2014>.

[18] REKHTER, Y.; LI, T.; HARES, S. *Border Gateway Protocol*. 2006. Disponível em: <http:www.rfc-editor.org/rfc/rfc4271.txt. Acesso em: maio 2014>.

[19] C.HENDRICK. *Routing Information Protocol*. 1988. Disponível em: <http:www.rfc-editor.org/rfc/rfc1058.txt. Acesso em: maio 2014>.

[20] P.BOSE; P.MORIN. Routing with guaranteed delivery in ad hoc wireless networks. *Theoretical Computer Science*, n. 6, p. 609–616, 2001.

[21] B.KARP; H.KUNG. Gpsr: greedy perimeter stateless routing for wireless networks. *Mobile Comput Networking*, p. 243–254, 2000.

[22] F.KUHN; R.WATTENHOFER. Geometric ad-hoc routing: of theory and pratice. *Proc. 22nd ACM Int Symp*, 2003.

[23] A.RAO; C.PAPADIMITRIOU. Geographic routing without location information. *ACM press*, 2003.

[24] N.LINIAL; L.LOVASZ. Rubber bands, convex embeddings and graph connectivity. *ACM press*, 1988.

[25] C.PAPADIMITRIOU; D.RATAJCZK. On a conjecture related to geometric routing. *Theoretical Computer Science*, 2005.

[26] F.COMELLAS; C.DALFó; M.FIOL. The manhattan product of digraphs. *Eletronic Journal of Graph Theory and Applications*, v. 1, p. 11–27, 2013.

[27] A.TUFAIL. Reliable latency aware routing for clustered wsns. *International Journal of Distributed Sensor Networks*, 2012.

[28] A.TUFAIL; K.KIM. *WEAMR-A weighted energy aware multipath reliable routing mechanism for hotline-based WSNs*. 2013. Disponível em: <http://www.scopus.com. Acesso em: maio 2014>.

[29] B.SUMAN; S.TRIPATHI. Minimum transmitting power and other performance metrics in regular wsn in fading environment. *International Journal of New Trands Eletronics and acommunication*, 2014.

[30] T.CORMEN. *Introduction to Algorithms*. [S.l.]: Cambridge: MIT Press, 2001.

[31] R.PERLMAN. *Interconnections: Bridges, Routers, Switches and IP*. [S.l.]: Addison Wesley, 1999.

[32] T.CLAUSEN; P.JACQUET. *Optimized Link State Routing Protocol-(OLSR)*. 2003. Disponível em: <http://www.rfc-editor.org/rfc/rfc3626.txt. Acesso em: maio 2014>.

[33] C.PERKINS; P.BHAGWAT. Highly dynamic destination sequenced distance vector routing. *ACM SIGCOMM*, n. 6, p. 234–244, 1994.

[34] D.JOHNSON; Y.HU; D.MALTZ. *The Dynamic Source Routing Protocol-(DSR)*. 2007. Disponível em: <http://www.rfc-editor.org/rfc/rfc4728.txt. Acesso em: maio 2014>.

[35] C.PERKINS; E.ROYER. http://www.cs.cornell.edu/people/egs/615/aodv.pdf. acesso em: maio 2014. In: *Ad hoc on-demand distance vector-(AODV) routing*. [S.l.: s.n.], 2003.

[36] P.ELIAS. Coding for noisy channels. *Proc. IRE Conv. Rec.*, part 4, n. 2, p. 37–46, 1955.

[37] J.PROAKIS; M.SALEHI. *Digital Communications*. [S.l.]: Mc Graw Hill, 2008.

[38] S.HAYKIN; M.MOHER. *Communication Systems*. [S.l.]: John Wiley & Sons, Inc, 2009.

[39] SHANNON, C. E. A mathematical theory of communication. *Bell System Technical Journal*, v. 27, p. 379–423 e 623–656, 1948.

[40] B.DAVEY; H.PRIESTLEY. *Introduction to Lattices and Order*. [S.l.]: Cambridge University Press, 1990.

[41] B.HERBERT. *Elements of Set Theory*. EUA: Academic Press, 1977.

[42] E.BIGLIERI; D.DIVSALAR; P.J.MCLANE; M.K.SIMON. *Introduction to Trellis-coded Modulation with Applications*. [S.l.]: Macmillan Publishing Company, 1991.

[43] A.VARGA. *OMNeT++ Discrete Event Simulation System*. 2011. Disponível em: <http://www.omnetpp.org/doc/manual/usman.html>.

[44] M.FAYAD; D.JOHNSON. *Building application frameworks: object-oriented foundations of framework design*. [S.l.]: J.Wiley, 1999. ISBN 0471248754.

[45] R.GENTLEMAN, R. . R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, p. 5:299–314, 1996.

[46] J.P.VASSEUR; A.DUNKELS. *Interconnecting Smart Objects with IP The Next internet*. EUA: Morgan Kaufmann, 2010.

[47] T.ANDERSON; L.PETERSON; S.SHENKER; J.TURNER. Overcoming the internet impasse through virtualization. *Computer*, v. 38, p. 34–41, 2005.

[48] K.CHOWDHURY; F.ZAHEER; R.BOUTADA. An identity management framework for network virtualization. *IEEE INFOCOM*, p. 34–41, 2009.

[49] N.ASA. *nRF24L01+ Product Specification*. 2008. Disponível em: <http://www.nordicsemi.com. Acesso em: maio 2014>.

[50] ATMEL. *Atmel Overview and use of The ATMEGA Serial Interface*. 2008. Disponível em: <http://www.atmel.com. Acesso em: maio 2014>.

[51] PRIX. *Open source Arduino environment*. 2013. Disponível em: <http://www.arduino.com. Acesso em: maio 2014>.

ANEXO A

O Anexo apresenta as características do hardware utilizado para obter os resultados práticos, preliminares desta pesquisa.

A.1 Validação prática do algoritmo TCNet

A.1.1 Introdução

Foram realizados testes inicialmente com uma rede WSN de 8 nós, utilizando as seguintes características de hardware:

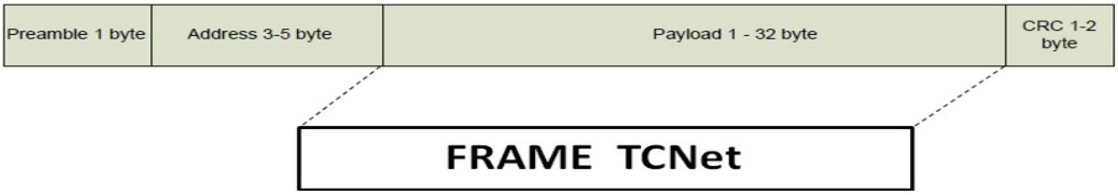
- Rádio nRF24L01+,[49], operando na banda ISM [1]:
  - Carrier Frequency: 2,4 GHz;
  - Radio transmitter Power: 1 mW;
  - Radio Sensitivity: -85 dBm;
  - Radio bitrate: 1 Mbps;
  - GFSK modulation;
- Processador e IDE:
  - Atmega328, [50], (32KB of Flash memory, 2KB of RAM, and 1K bytes of EEPROM);
  - Arduino IDE (Ver 1.5.2), [51];
- Sensores e Atuadores:
  - Transdutores conectados ao conversor A/D do ATmega328 com resolução de 10 bits.
- Estrutura do FRAME rádio no modo ShockBurst:

[1] Banda de frequência destinada ao uso industrial, científico e medicina

– O rádio nRF24L01+ possui sua própria estrutura de FRAME, sendo utilizados os protocolos proprietários do tipo Enhanced ShockBurst ou ShockBurst packet format. Neste trabalho foi utilizado o ShockBurst packet format de modo a não mascarar o FRAME TCNet a ser transportado pelo payload rádio;

– As características do FRAME rádio adotado resumem-se aos seguintes campos: preamble, address, payload e CRC, mostrado na Figura A.1:

Figura A.1: FRAME rádio no modo "ShockBurst packet format" com capacidade de payload (0 - 32 bytes)

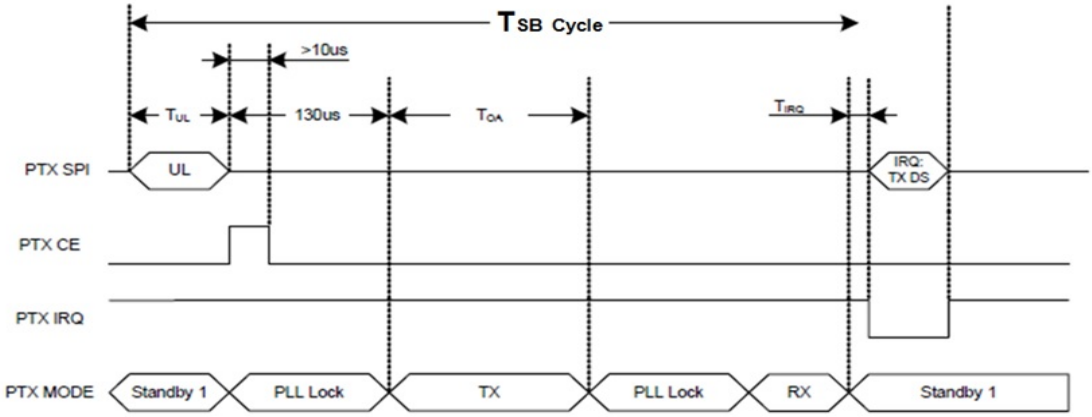


Fonte: http://www.nordicsemi.no (maio 2014), alterado pelo autor

* **Preamble**: é uma sequência de bits utilizado no sincronismo com o formato de um byte (10101010 ou 01010101) de modo a estabilizar os transceptores;
* **Address**: corresponde ao endereço do transmissor e receptor para evitar colisões na rede. Neste trabalho são utilizados 5 bytes (F0F0F0F0E1 e F0F0F0F0D2) correspondendo respectivamente ao transmissor e receptor que serão configurados pelo nós quando assumirem a condição de transmissão ou recepção. Nesse trabalho será mantido o mesmo par de endereço para todos os nós da rede;
* **Payload**: conteúdo a ser transmitido pelo nRF24L01+ com capacidade de 0 a 32 bytes, definido pelo usuário, neste caso o FRAME TCNet que irá gerenciar a rede, ampliando assim a capacidade do rádio na configuração MutiCeiver limitado a uma rede de 6 nós;
* **CRC**: (Cyclic Redundancy Check), mecanismo de detecção de erro na transmissão dos pacotes. Nenhum pacote é aceito pelo modo "ShockBurst" se houver falha de CRC.

– O FRAME do rádio nRF24L01+ deve se ajustar ao Diagrama de Tempo de Transmissão [49], mostrado na Figura A.2, controlado por uma SPI - Serial Peripheral Interface onde encontra-se armazenado o FRAME TCNet que vai determinar os PTX - Peripheral Control TX.

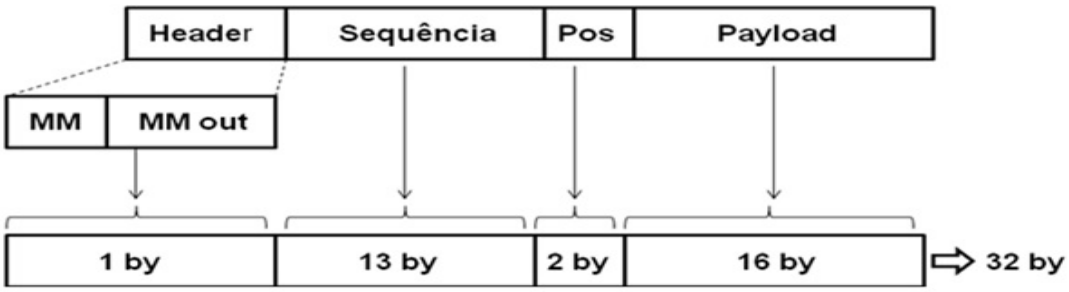
Figura A.2: Diagrama de Tempo de Transmissão do FRAME rádio nRF24L01+: T SB (Time ShockBurst); T UL (Time Upload); T AO (Time on - air); T IRQ (Time IO Request)



Fonte: http://www.nordicsemi.no (maio 2014)

- Estrutura do FRAME TCNet:
  - O FRAME TCNet foi configurado conforme a Figura A.3, de modo a aproveitar os 32 bytes disponíveis no FRAME rádio, onde os campos foram distribuídos em: header, sequência, posição (pos) e payload , com as seguintes funções:
    - **Header**: transporta a informações da Mealy Machine (MM) utilizada para gerar uma rota, subdividindo-se em *nibble* (MSB) - (0000 a 1111) onde armazena os tipos de configurações das MMs previamente armazenadas na rotina da rede. O gerenciamento da escolha da MM é de responsabilidade do nó *sink*, de modo a atender a uma alternativa de rota que venha garantir à alcançabilidade dos nós de interesse. O *nibble* (LSB), MM out - (0000 a 1111) armazena o resultado da MM escolhida, importante para decisão do ramo da treliça que irá representar o ramo da rede indicada pela *distância de Hamming*=> $(d_{\text{Ham}}) = 0$ ou a distância com *Máxima Verossimilhança*.
    - **Sequência**: é a informação de entrada da MM definindo uma rota sobre os ramos da treliça correspondente, e portando os enlaces entre os nós solicitados por um *query* emitido pelo *sink*.
    - **Pos**: é a posição até onde a sequência deve ser inserida nos registradores da MM. Esse procedimento é realizado por todos os nós da rede durante um *multicast*, importante para que o nó reconheça se é o nó solicitado. O campo *Pos* é sempre atualizado pelo último nó visitado.
    - **Payload**: é o campo do FRAME TCNet onde é realizado o *upload* da informação colhida na visita ao nó. Na configuração da rede com 8 nós, dessa validação prática, foram considerados 16 campos, onde cada 2 campos correspondem a 1 byte, de modo a armazenar informações normalizadas (0 a 255) obtidas dos sensores.

Figura A.3: Distribuição do payload de 32 bytes destinado ao FRAME TCNet para uma rede de 8 nós

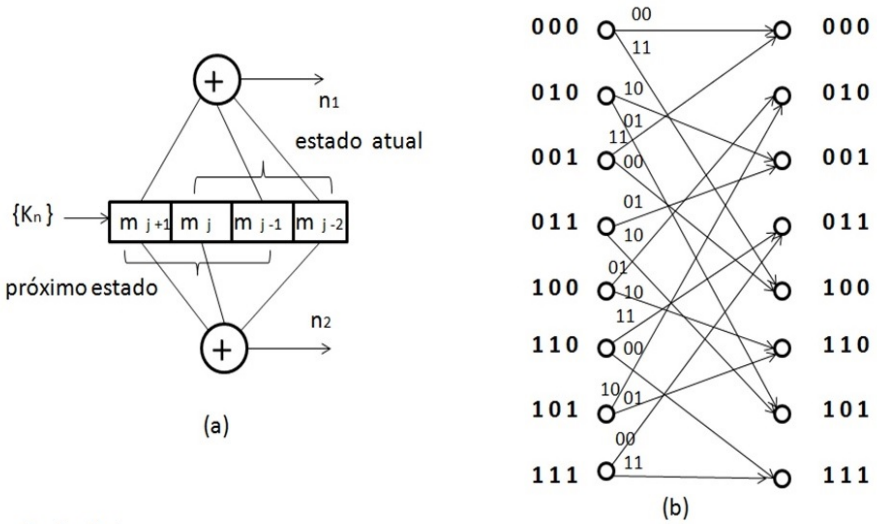


Fonte: autor

A.1.2 Cenário de implementação do TCNet no caso ideal

Inicialmente foi considerada uma configuração de Mealy Machine com taxa k/n = 1/2 de modo a atender a uma rede com 8 nós, mostrada na Figura A.4, onde (a) é o codificador MM 8 e (b) é o respectivo decodificador de treliça.

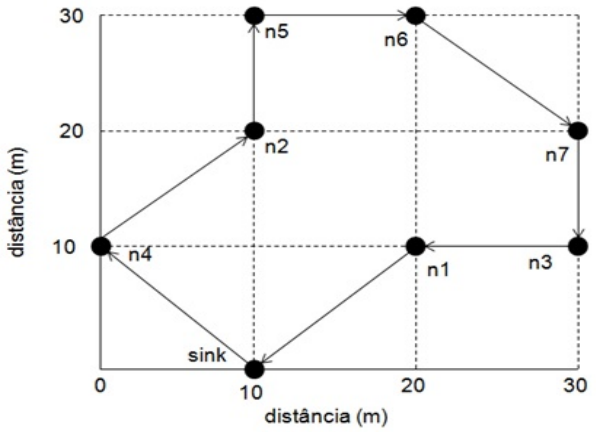
Figura A.4: (a)MM com taxa k/n = 1/2, capaz de gerar palavras códigos ($n_1 n_2$) correspondentes aos pesos dos ramos da treliça; (b)Decodificador de treliça correspondente



Fonte: Autor

O cenário prático estabelecido considerou o pior caso para aplicação do rádio nRF24L01+, como sendo uma área de cobertura *indoor* distribuída conforme mostra a Figura A.5.

Figura A.5: Cenário inicial *indoor*, para testes de uma rede com 8 nós acoplados a sensores



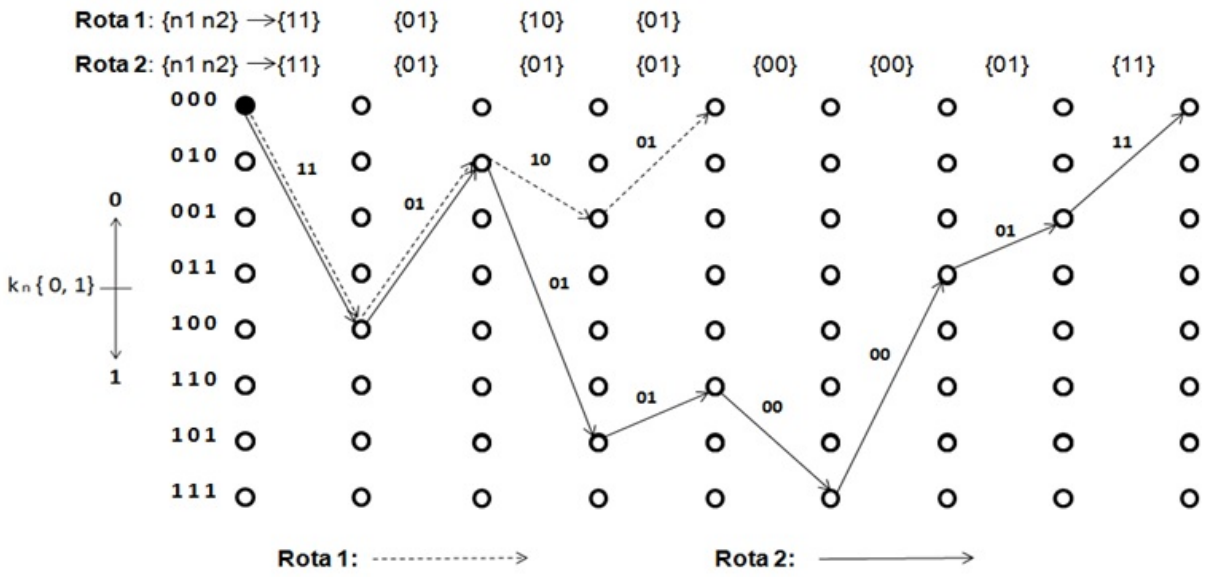
Fonte: autor

Considerando-se as possibilidades das rotas no cenário da Figura A.6, **Rota 1** e **Rota 2**, em que:

- **Rota 1**: refere-se a um grupo de 4 nós de modo a mostrar a flexibilidade do processo em atender a um critério de QoS, como rota mais curta. Para isso foi especificada a sequência $k_n = \{1000\}$ emitida durante um *query* pelo nó *sink* (000);
- **Rota 2**: mostra uma rota de modo a atender a alcançabilidade dos nós da rede, para isso foi especificada a sequência $k_n = \{10111000\}$ referente a um *query* originado pelo *sink* (000).

Para demonstrar o processo, essa pesquisa considerou a **Rota 1** por considerar mais imediata.

Figura A.6: Rotas resultantes para duas sequências $k_n = \{1000\}$ e $k_n = \{10111000\}$

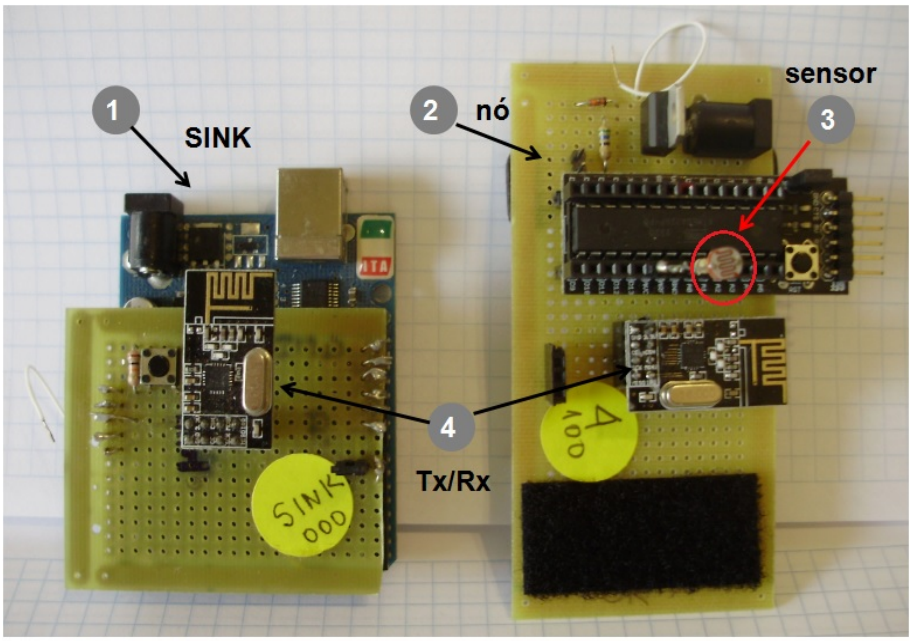


Fonte: autor

A.1.3 Resultados em tempo real da rede considerando a Rota 1

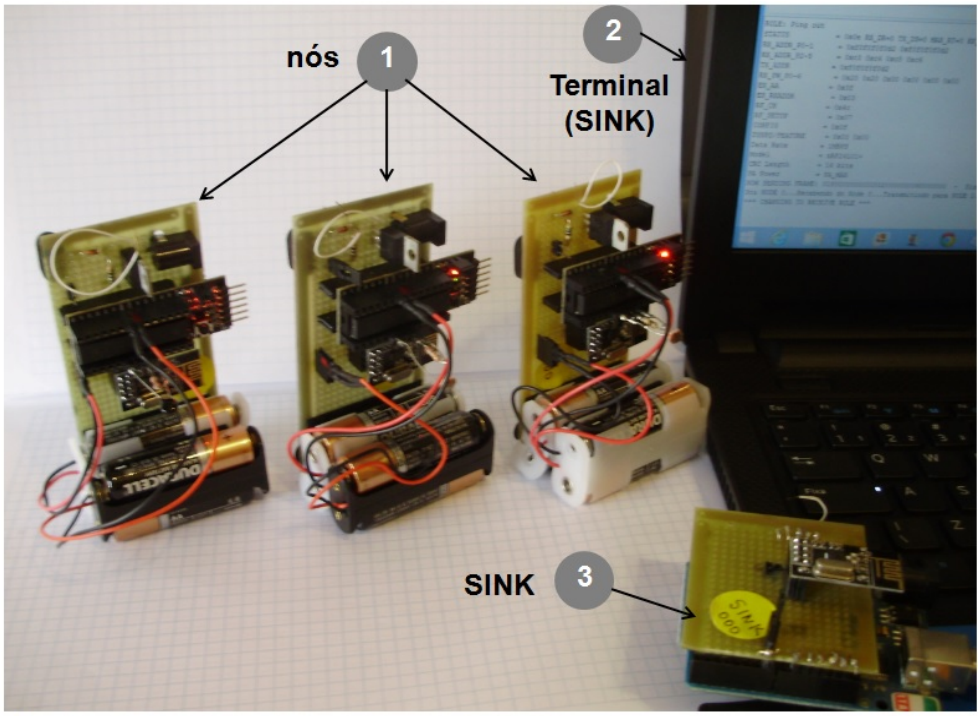
Os ensaios para implementação da **Rota 1**, foram realizados com os protótipos mostrados nas Figuras A.7 e A.8.

Figura A.7: Protótipo do *sink* (1) e do nó da rede (2), mostrando detalhes do sensor (3) e Transceptores (Tx/Rx)(4)



Fonte: autor

Figura A.8: Protótipos dos nós da rede (1), Terminal do (*sink*) (2) e *sink* (3)

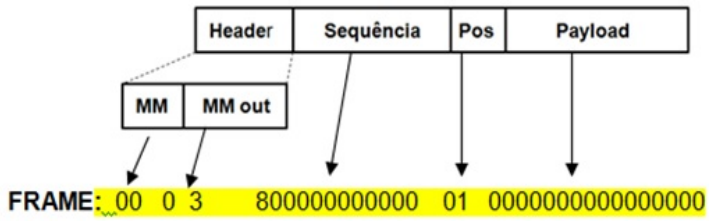


Fonte: autor

A inicialização da rede considerada, é obtida realizando-se os seguintes procedimentos preliminares:

- Carregando-se o FRAME inicial no *sink* (000), com as caractericas desejadas para a rota, como mostra a Figura A.9.

Figura A.9: FRAME inicial inicializado pelo *sink*



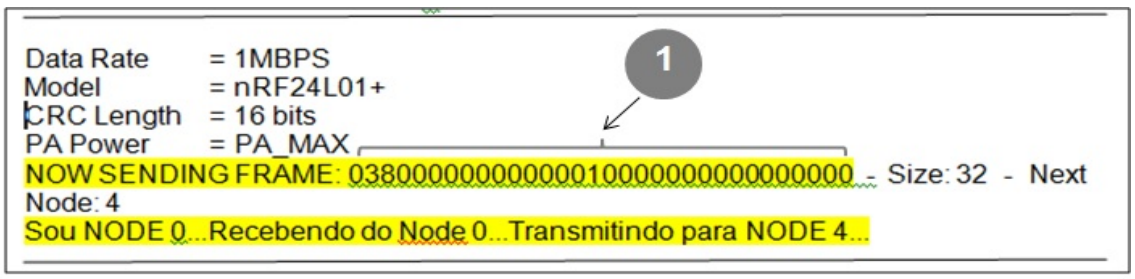
Fonte: autor

- Identificando-se os respectivos *nodes IDs* nas Memórias RAMs, individualmente em cada nó.

Os resultados dos nós em tempo real, podem ser monitorados diretamente durante a solicitação do *query*. Para a rota considerada, os resultados do processo foram confirmados conforme mostram as Figuras a seguir:

- A Figura A.10 mostra o relatório gerado pelo *sink* após ter carregado o FRAME inicial, com as características da rota e resultados da MM, possibilitando o nó destino (nó 4), executar o próximo passo após o recebimento do *multicast*.

Figura A.10: NODE 0 inicializando o *query* com transmissão do FRAME inicial (1)



Fonte: autor

- A Figura A.11 mostra o relatório gerado pelo nó 4, após o recebimento do FRAME (1) e o carregamento do *payload* com a informação do dados do sensor (2). Após atualizar o FRAME para ser enviado (3), realiza um novo *multicast*.

Figura A.11: NODE 4 atualizando a transmissão do FRAME (1)

```
Data Rate      = 1MBPS
Model          = nRF24L01+
CRC Length     = 16 bits
PA Power       = PA_MAX
GOT PAYLOAD size=32 value=03800000000000010000000000000000
idNode = 4  -  Current Node = 4

tPos=02
val=30
tSeq=800000000000
tD0=00
tD1=00
tD2=00
tD3=00
tD4=30
tD5=00
tD6=00
tD7=00
*** CHANGING TO TRANSMIT ROLE ***
NOW SENDING FRAME: 01800000000000020000000030000000 - Size: 32 - Next
Node: 2
Sou NODE 4...Recebendo do NODE 0...Transmitindo para NODE 2...
```

Fonte: autor

- A Figura A.12 mostra o relatório gerado pelo nó 2, após o recebimento do FRAME (1) e atualização do *payload* com a informação do sensor correspondente (2). Após atualizar o FRAME para ser enviado (3), realiza um novo *multcast*.

Figura A.12: NODE 2 atualizando transmissão do FRAME (1)

```
Data Rate      = 1MBPS
Model          = nRF24L01+
CRC Length     = 16 bits
PA Power       = PA_MAX
GOT PAYLOAD size=32 value=01800000000000020000000030000000
idNode = 2  -  Current Node = 2

tPos=03
val=8f
tSeq=800000000000
tD0=00
tD1=00
tD2=8f
tD3=00
tD4=30
tD5=00
tD6=00
tD7=00
*** CHANGING TO TRANSMIT ROLE ***
NOW SENDING FRAME: 02800000000000030000 8f0030000000 - Size: 32 - Next
Node: 1
Sou NODE 2...Recebendo do NODE 4...Transmitindo para NODE 1...
```

Fonte: autor

- A Figura A.13 mostra o relatório gerado pelo nó 1, após o recebimento do FRAME (1) e a atualização do *payload* com a informação do sensor correspondente (2). Após atualizar o FRAME para ser enviado (3), realiza um novo *multicast*.

Figura A.13: NODE 1 atualizando a transmissão do FRAME (1)

```
Data Rate      = 1MBPS
Model          = nRF24L01+
CRC Length     = 16 bits
PA Power       = PA_MAX
GOT PAYLOAD size=32 value=028000000000000300008f0030000000
idNode = 1  -  Current Node = 1

tPos=04
val=32
tSeq=800000000000
tD0=00
tD1=32
tD2=8f
tD3=00
tD4=30
tD5=00
tD6=00
tD7=00
*** CHANGING TO TRANSMIT ROLE ***
NOW SENDING FRAME: 038000000000000400328f0030000000 - Size: 32 - Next
Node: 4
Sou NODE 1...Recebendo do NODE 2...Transmitindo para NODE 0...
```

Fonte: autor

- A Figura A.14 mostra o relatório final gerado pelo *sink* (000), após o recebimento do FRAME (1), onde estão reunidas no *payload*, as informações coletadas dos sensores, pertencentes à rota considerada (2).

Figura A.14: NODE 0 gerando o relatório do *query* com as informas coletadas no *payload*

```
Data Rate      = 1MBPS
Model          = nRF24L01+
CRC Length     = 16 bits
PA Power       = PA_MAX
GOT PAYLOAD size=32 value=038000000000000400328f0030000000 - Size: 32
Next Node: 0
Sou NODE 0...Recebendo do Node 1...Transmitindo para NODE 0...
*** CHANGING TO RECEIVE ROLE ***
GOT PAYLOAD size=2 value=03
idNode = 0  -  Current Node = 1
*** RELATORIO ***
Machine type: 0 0
Header.....: 3 3
Sequence....: - 800000000000
Position....: - 0
Data SINK...: - 0
Data Node 1.: - 32
Data Node 2.: - 8f
Data Node 3.: - 0
Data Node 4.: - 30
Data Node 5.: - 0
Data Node 6.: - 0
Data Node 7.: - 0
```

Fonte: autor

A.1.4 Conclusões

Os resultados obtidos não foram exaustivos, devido às limitações de tempo para concluir esta pesquisa. Embora a rede considerada para os ensaios seja reduzida, foram encontradas dificuldades que podem se propagar quando se considerar redes maiores, como:

- Os problemas de colisões, não sendo suficientes os dados especificados pelo fabricante [49] do transmissor, com relação ao Diagrama de Tempo (Figura A.2), outros parâmetros devem ser considerados, como: atrasos de canal, banda de guarda, reflexões do sinal em ambientes *indoor*;
- A capacidade de Memória do processador utilizado para armazenamento da rotina do nó deve ser considerada, à medida que a rede aumenta a especificação do processador para o projeto pode encontrar limitações, embora seja o objetivo da pesquisa aplicações em sistemas de baixa capacidade;
- As fontes de energia para os nós devem ser autocarregáveis, devido a impossibilidade de se administrar as várias fontes distribuídas pela rede.

Os próximos passos para a validação prática dessa pesquisa consiste em realizar testes em redes com maior número de nós, de modo a levantar especificações de hardware e otimizar as rotinas.

A rotina utilizada nos ensaios iniciais está disponível no Anexo B.

ANEXO B

Código fonte do algoritmo TCNet (Validação prática) rede 4 nós

tcnet_rede_4n.cc

```
1
2   #include <SPI.h>
3   #include "nRF24L01.h"
4   #include "RF24.h"
5   #include "printf.h"
6   // HARDWARE CONFIGURATION
7   // Set up nRF24L01 radio on SPI bus plus pins 9 & 10
8   RF24 radio(9, 10);
9   const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };
10  typedef enum { role_ping_out = 1, role_pong_back } role_e;
11  role_e role;
12  const char* role_friendly_name[] = { "invalid", "Ping out",
13  "Pong back"};
14  const int analogInPin = A0;  // Analog input pin that the
15  potentiometer is attached to
16
17  // CONSTANTES
18  const int max_payload_size = 32;
19  const int size_eng = 4;  // Número máximo de bits da MM.
20  const int size_seq = 48;
21  const int size_hcn = 3;
22
23  // VARIÁVEIS DECLARADAS
24  char receive_payload[max_payload_size + 1]; // +1 to allow
25  room for a terminating NULL char
26  //           038000000000000100000000000000000 - Para 4 nós
27
28  char send_payload[] = "038000000000000100000000000000000";
29  Payload padrão do TCNet (SINK)
30  char *tMM,  // tipo de Máquina de Mealy em hexa - texto
31   *tH,   // header: n1n2 em hexa - texto
32   *tSeq, // sequência em hexa - texto
33   *tPos, // posição em hexa - texto
34   *tD0,  // dado do nó 0 (SINK) em hexa - texto
35   *tD1,  // dado do nó 1 em hexa - texto
36   *tD2,  // dado do nó 2 em hexa - texto
37   *tD3,  // dado do nó 3 em hexa - texto
38   *tD4,  // dado do nó 4 em hexa - texto
39   *tD5,  // dado do nó 5 em hexa - texto
40   *tD6,  // dado do nó 6 em hexa - texto
41   *tD7;  // dado do nó 7 em hexa - texto
```

```
42   int nMM,  // tipo de Máquina de Mealy em decimal
43   nH,   // header: n1n2 em decimal
44   nSeq[size_seq], // sequência em decimal - binário
45   nPos, // posição em decimal
46   nD0,  // dado do nó 0 (SINK) em decimal
47   nD1,  // dado do nó 1 em decimal
48   nD2,  // dado do nó 2 em decimal
49   nD3,  // dado do nó 3 em decimal
50   nD4,  // dado do nó 4 em decimal
51   nD5,  // dado do nó 5 em decimal
52   nD6,  // dado do nó 6 em decimal
53   nD7;  // dado do nó 7 em decimal
54
55   // VARIÁVEIS DEFINIDAS
56   int HCN[] = {0, 0, 0}; // HCN[0] = Header; HCN[1] = Current;
57   HCN[2] = Next;
58   int sensorValue = 0;   // value read from the pot
59   int outputValue = 0;   // value output to the PWM (analog)
60
61   void MM(int seq[], int pos, int hcn[], int tipoMM) {
62   int engine[] = {0, 0, 0, 0}; // A máquina inicia com zeros
63   int e, i = 0, e03, e1, e2; // elem do engine  |e0|e1|e2|e3|
64
65   switch (tipoMM) {
66   case 0:
67     do {
68       for (e = size_eng - 1; e > 0; e--)
69         engine[e] = engine[e - 1];
70       engine[0] = seq[i++];
71     } while (i < pos);
72
73     e03 = engine[0] ^ engine[3];
74     e1  = engine[2] ^ e03;
75     e2  = engine[1] ^ e03;
76     hcn[0] = e1 * 2 + e2; // Header em decimal
77     hcn[1] = engine[1] * 4 + engine[2] * 2 + engine[3];
78     hcn[2] = engine[0] * 4 + engine[1] * 2 + engine[2];
79     break;
80     case 1:
81     // Implementar
82     break;
83 }
84  } // MM
85
86    char* strnmcpy(char str[], int start, int fim) {
87    int i, j;
88    char *sub;
89    if (start >= fim || fim > strlen(str))
90    return NULL;
91    sub = (char *) malloc(sizeof(char) * (fim - start + 1));
92    for (i = start, j = 0; i < fim; i++, j++)
```

```
 93    sub[j] = str[i];
 94    sub[j] = '\0';
 95    return sub;
 96  } // strnmcpy
 97
 98   void hex2bin(int bin[], char hex[]) {
 99   for (int i = 0; i < strlen(hex); i++) {
100    switch (hex[i]) {
101      case '0': bin[i * 4 + 0] = 0; bin[i * 4 + 1] = 0; bin[i *
102      4 + 2] = 0; bin[i * 4 + 3] = 0; break;
103      case '1': bin[i * 4 + 0] = 0; bin[i * 4 + 1] = 0; bin[i *
104      4 + 2] = 0; bin[i * 4 + 3] = 1; break;
105      case '2': bin[i * 4 + 0] = 0; bin[i * 4 + 1] = 0; bin[i *
106      4 + 2] = 1; bin[i * 4 + 3] = 0; break;
107      case '3': bin[i * 4 + 0] = 0; bin[i * 4 + 1] = 0; bin[i *
108      4 + 2] = 1; bin[i * 4 + 3] = 1; break;
109      case '4': bin[i * 4 + 0] = 0; bin[i * 4 + 1] = 1; bin[i *
110      4 + 2] = 0; bin[i * 4 + 3] = 0; break;
111      case '5': bin[i * 4 + 0] = 0; bin[i * 4 + 1] = 1; bin[i *
112      4 + 2] = 0; bin[i * 4 + 3] = 1; break;
113      case '6': bin[i * 4 + 0] = 0; bin[i * 4 + 1] = 1; bin[i *
114      4 + 2] = 1; bin[i * 4 + 3] = 0; break;
115      case '7': bin[i * 4 + 0] = 0; bin[i * 4 + 1] = 1; bin[i *
116      4 + 2] = 1; bin[i * 4 + 3] = 1; break;
117      case '8': bin[i * 4 + 0] = 1; bin[i * 4 + 1] = 0; bin[i *
118      4 + 2] = 0; bin[i * 4 + 3] = 0; break;
119      case '9': bin[i * 4 + 0] = 1; bin[i * 4 + 1] = 0; bin[i *
120      4 + 2] = 0; bin[i * 4 + 3] = 1; break;
121      case 'A':
122      case 'a': bin[i * 4 + 0] = 1; bin[i * 4 + 1] = 0; bin[i *
123      4 + 2] = 1; bin[i * 4 + 3] = 0; break;
124      case 'B':
125      case 'b': bin[i * 4 + 0] = 1; bin[i * 4 + 1] = 0; bin[i *
126      4 + 2] = 1; bin[i * 4 + 3] = 1; break;
127      case 'C':
128      case 'c': bin[i * 4 + 0] = 1; bin[i * 4 + 1] = 1; bin[i *
129      4 + 2] = 0; bin[i * 4 + 3] = 0; break;
130      case 'D':
131      case 'd': bin[i * 4 + 0] = 1; bin[i * 4 + 1] = 1; bin[i *
132      4 + 2] = 0; bin[i * 4 + 3] = 1; break;
133      case 'E':
134      case 'e': bin[i * 4 + 0] = 1; bin[i * 4 + 1] = 1; bin[i *
135      4 + 2] = 1; bin[i * 4 + 3] = 0; break;
136      case 'F':
137      case 'f': bin[i * 4 + 0] = 1; bin[i * 4 + 1] = 1; bin[i *
138      4 + 2] = 1; bin[i * 4 + 3] = 1; break;
139      }
140  }
141 } // hex2bin
142
143  void brokeFrame(char frame[]) {
```

```
144  tMM  = strnmcpy(frame, 0, 1);
145  tH   = strnmcpy(frame, 1, 2);
146  tSeq = strnmcpy(frame, 2, 14);
147  tPos = strnmcpy(frame, 14, 16);
148  tD0  = strnmcpy(frame, 16, 18); // {0, 1, 2, 14, 16, 18, 20,
149  22, 24, 26, 28, 30}
150  tD1  = strnmcpy(frame, 18, 20);
151  tD2  = strnmcpy(frame, 20, 22);
152  tD3  = strnmcpy(frame, 22, 24);
153  tD4  = strnmcpy(frame, 24, 26);
154  tD5  = strnmcpy(frame, 26, 28);
155  tD6  = strnmcpy(frame, 28, 30);
156  tD7  = strnmcpy(frame, 30, 32);
157 } // brokeFrame
158
159  void getDataMainValues()
160  {
161  nMM  = strtol (tMM, NULL, 16);
162  nH   = strtol (tH, NULL, 16);
163  hex2bin(nSeq, tSeq);
164  nPos = strtol (tPos, NULL, 16);
165 } // getDataMainValues
166
167  void getDataNodeValues()
168 {
169  nD0 = strtol (tD0, NULL, 16);
170  nD1 = strtol (tD1, NULL, 16);
171  nD2 = strtol (tD2, NULL, 16);
172  nD3 = strtol (tD3, NULL, 16);
173  nD4 = strtol (tD4, NULL, 16);
174  nD5 = strtol (tD5, NULL, 16);
175  nD6 = strtol (tD6, NULL, 16);
176  nD7 = strtol (tD7, NULL, 16);
177 } // getDataNodeValues
178
179  void printReport()
180 {
181  printf("Machine type: %s - %d\n", tMM, nMM);
182  printf("Header......: %s - %d\n", tH, nH);
183  printf("Sequence....: %s - ", tSeq);
184  for(int x = 0; x < size_seq; x++)
185  printf("%d", nSeq[x]);
186  printf("\n");
187  printf("Position....: %s - %d\n", tPos, nPos);
188  printf("Data SINK...: %s - %d\n", tD0, nD0);
189  printf("Data Node 1.: %s - %d\n", tD1, nD1);
190  printf("Data Node 2.: %s - %d\n", tD2, nD2);
191  printf("Data Node 3.: %s - %d\n", tD3, nD3);
192  printf("Data Node 4.: %s - %d\n", tD4, nD4);
193  printf("Data Node 5.: %s - %d\n", tD5, nD5);
194  printf("Data Node 6.: %s - %d\n", tD6, nD6);
```

```
195  printf("Data Node 7.: %s - %d\n", tD7, nD7);
196 } // printReport
197
198 // Enable to sender...ping
199 void change2sender()
200 {
201  role = role_ping_out;
202  radio.openWritingPipe(pipes[0]);
203  radio.openReadingPipe(1, pipes[1]);
204} // sender
205
206  // Enable to receiver...pong
207 void change2receiver()
208 {
209  role = role_pong_back;
210  radio.openWritingPipe(pipes[1]);
211  radio.openReadingPipe(1, pipes[0]);
212 } // receiver
213
214  char  mode; // R - modo receptor; T - transmissor.
215  const unsigned int idNode = 0; // 0 = sink node.
216  // Start process
217  void setup(void)
218 {
219   if (idNode == 0) {
220    mode = 'T';
221    role = role_ping_out;
222  }
223  else {
224    mode = 'R';
225    role = role_pong_back;
226  }
227  Serial.begin(57600); // 57600
228  printf_begin();
229  printf("ROLE: %s\n\r", role_friendly_name[role]);
230  radio.begin();
231  radio.setRetries(15, 15);
232  radio.openReadingPipe(1, pipes[1]);
234  radio.startListening();
235  radio.printDetails();
236 } // setup
237
238  // Infinite loop...
239  int tf, priorNode;
240  void loop(void)
241 {
242  // Ping out role.  Repeatedly send the current time
243  if (role == role_ping_out) // TRANSMISSOR
244  {
245    radio.stopListening();
```

```
246
247//*********************************************************
248      // Trata o FRAME recebido...
249      brokeFrame(receive_payload);
250      getDataMainValues();
251      MM(nSeq, nPos, HCN, nMM);
252      priorNode = HCN[1];
253
254 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
255   printf("idNode = %d  -  Current Node = %d\n", idNode,
256   HCN[2]);
257      // Forçando impressão do relatório
258   //HCN[2] = 0;
259      //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
260
261      if (HCN[2] == 0) { // é SINK, imprime relatório...
262        getDataNodeValues();
263        printf("*** RELATORIO ***\n");
264        printReport();
265        printf("\n*** CHANGING TO TRANSMIT ROLE ***\n");
266        change2sender();
267      }
268      else if(HCN[2] == idNode) { // é NODE, então atualiza o
269      FRAME para ser reenviado...
270        // Código para atualização do FRAME a ser 271
271
272   //**********************************************
273        nPos++; // Incrementa Position de uma unidade...
274        MM(nSeq, nPos, HCN, nMM); // Recalcula HCN
275//*****************************************************
276        sensorValue = analogRead(analogInPin); // Lê o sensor
277        outputValue = map(sensorValue, 0, 1023, 0, 255);
278        char val[3];
279        if (outputValue < 10)
230        sprintf(val, "0%x", outputValue);
231        else
232          sprintf(val, "%x", outputValue);
233        val[2] = '\0'; // Finaliza para atualização
234        switch (idNode) {
235          case 1: strcpy(tD1, val); break;
236          case 2: strcpy(tD2, val); break;
237          case 3: strcpy(tD3, val); break;
238          case 4: strcpy(tD4, val); break;
239          case 5: strcpy(tD5, val); break;
240          case 6: strcpy(tD6, val); break;
241          case 7: strcpy(tD7, val); break;
242        }
243        // Converte positiondec para valuePosition...
244        if (nPos < 10)
245          sprintf(tPos, "0%x", nPos);
246        else
```

```
247        sprintf(tPos, "%x", nPos);
247
248       // receive_payload[0] = '\0'; // Limpa o frame
249
250        printf("*** CHANGING TO TRANSMIT ROLE ***\n");
251        change2sender();
252      }
253      delay(500);
254      radio.startListening();
255    }
256    delay(500);
257  }
258} // loop
259  // vim:cin:ai:sts=2 sw=2 ft=cpp
```

ANEXO C

Código fonte do algoritmo TCNet (caso ideal) simulação no OMNet++

tcnet.cc

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4   #include <omnetpp.h>
5
6   #include "TCNet_m.h"
7
8   #include <iostream>
9   using namespace std;
10
11  #define MAX 2000
12
13  class TCNode : public cSimpleModule
14  {
15
16     cOutVector Tp;
17     cOutVector Energy;
18
19      int seq[MAX];         // Sequência de entrada
20      int out[MAX];         // Saída da MM
21      int hcn[MAX];         // Dados estraídos da MM,
22
23      int k;                // hcn[2] (para controle)
24      int N;                // Grau de escalabilidade da rede
25
26
27
28     double tx_interval;         // Atraso de Canal
29     double tprocessamento_MM;   // Tempo de proces da MM
30     double t_guarda;            // Banda de guarda
31     double latencia;            // Latência
32
33     double e_tx;                // Energia de Tx
34     double e_rx;                // Energia de Rx
35     double e_proc;              // Energia de proc
36     double Energia_total;       // Energia Total
37
38     private:
39     simsignal_t arrivalSignal;
40
41     protected:
```

```
42     virtual TCNETmsg *generateMessage();
43     virtual void forwardMessage(TCNETmsg *msg);
44     virtual void initialize();
45      virtual void handleMessage(cMessage *msg);
46        };
47
48      Define_Module(TCNode);
49
50      void MealyMachine(int seq[], int pos, int hcn[], int
51      out[], int N)
52      hcn[0] = hcn[1] = hcn[2] = 0;
53      int SIZE_ENG = N+1;
54      int engine[MAX]; // A máquina sempre inicia com zeros
55      for( int q = 0; q < SIZE_ENG; q++)
56      engine[q] = 0;
57
59     // Mealy Machine
60     // Percorre a seq até a posição pos-1 (pois ini zero)
61     int e, i = 0;
62     do {
63     // Desloca todos os bits da engine uma posição dir
64     for(e = SIZE_ENG-1; e > 0; e--) {
65     engine[e] = engine[e-1];
66     }
67
68    engine[0] = seq[i++];
69    } while(i < pos);
70
71   // Efetua as transformações
72   // Obtém o header (n1n2)
73   int ee1, ee2; // elementos do engine
74   ee1 = engine[0];
75   ee2 = engine[SIZE_ENG-1];
76
77   for( int t = SIZE_ENG-1; t > 1; t--){
78   ee1 = ee1^engine[t];
79   }
80
81   for( int s = 0; s < SIZE_ENG-2; s++){
82   ee2 = ee2^engine[s];
83    }
84
85   hcn[0] = ee1 * 2 + ee2; // Em decimal
86   // Obtém o nó corrente (current)
87   int pot = 0;
88   for( int t = SIZE_ENG-1; t > 0; t--){
89   hcn[1] = engine[t]*pow(2,pot) + hcn[1];
90   pot++;
91   }
92
```

```
 93     // Obtém o próximo nó (destino)
 94     int pot2 = N-1;
 95     for( int s = 0; s < SIZE_ENG-1; s++){
 96     hcn[2] = engine[s]*pow(2,pot2) + hcn[2]; pot2--;
 97
 98     }
 99
100     // Obtém a saída (n1n2)
101     out[0] = ee1;
102     out[1] = ee2;
103
104     }
105
106     void TCNode::initialize()
107    {
108
109   // Leitura do parâmetro "Grau de escalabilidade da rede "2^n"
110    N = simulation.getSystemModule()->par("n");
111
112
113    // Sequência de entrada
114    if(N == 3){
115   //Se o número de nós da rede for 8 (2^N),escolher a sequência
116         seq[0]= 1;
117         seq[1]= 0;
118         seq[2]= 1;
119         seq[3]= 1;
120         seq[4]= 1;
121         seq[5]= 0;
122         seq[6]= 0;
123         seq[7]= 0;
124    }else{
125    // Se o número de nós da rede for diferente de 8, a
126   // Gerador da sequência; 0..N/2 = 1, N/2..N = 0
127   for(int i = 0; i < pow(2,N); i++){
128               if(i < pow(2,N)/2 )
129                     seq[i] = 1;
130               else
131               seq[i] = 0;
132         }
133    }
134
135    arrivalSignal = registerSignal("arrival");
136
137    // Atribuição de valores aos parâmetros de análise
138    tx_interval = par("Tx_interval");
139    tprocessamento_MM = par("T_proc");
140    t_guarda = par("T_guard");
```

```
141    e_tx = par("E_tx");
142    e_rx = par("E_rx");
143    e_proc = par("E_proc");
144    Tp.setName("Tempo de processamento");
145    Energy.setName("Energia consumida");
146
147    // Node 0 envia a primeira mensagem
148    if (getIndex()==0)
149    {
150    // Inicia o processo marcando a primeira mensagem
151   char first[20];
152   sprintf(first,"SEQ;
153
154    TCNETmsg *msg = new TCNETmsg(first);
155
156    scheduleAt(0.0 + simTime(), msg);
157
158    }
159  }
```

ANEXO D

Código fonte do algoritmo TCNet (falha do nó) simulação no OMNet++

tcnet_falha.cc

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4   #include <omnetpp.h>
5
6   #include "TCNet_m.h"
7
8   #include <iostream>
9   using namespace std;
10
11  #define MAX 2000
12
13  class TCNode : public cSimpleModule
14 {
15 cOutVector Tp;
16 cOutVector Energy;
17
18  int seq[MAX];          // Sequência de entrada
19  int out[MAX];          // Saída da MM
20  int hcn[MAX];          // Dados estraídos da MM, em decimal
21                         //(Nó atual, próximo nó e header)
22  int k;                 // hcn[2] (para controle)
23  int N;                 // Grau de escalabilidade da rede "2^n
24
25  int position;
26
27  // Parâmetros de análise
28  double tx_interval;          // Atraso de Canal
29  double tprocessamento_MM;    // Tempo de processamento da MM
30  double t_guarda;             // Banda de guarda
31  double t_espera;             // Espera  double latencia;
32 // Latência
33
34  double e_tx;                 // Energia de transmissão
35  double e_rx;                 // Energia de recepção
36  double e_proc;               // Energia de processamento
37  double Energia_total;        // Energia Total
38
39  private:
40 simsignal_t arrivalSignal;
```

```
41  protected:
42  virtual TCNETmsg *generateMessage();
43  virtual void forwardMessage(TCNETmsg *msg);
44  virtual void initialize();
45  virtual void handleMessage(cMessage *msg);
46  };
47
48  Define_Module(TCNode);
49
50  void MealyMachine(int seq[], int pos, int hcn[], int out[],
51  int N) {
52  hcn[0] = hcn[1] = hcn[2] = 0;
53
54  int SIZE_ENG = N+1;
55  int engine[MAX]; // A máquina sempre inicia com zeros
56  for( int q = 0; q < SIZE_ENG; q++)
57  engine[q] = 0;
58  // Supõe-se que "pos" é bem comportado, isto é, pos <= size.
59  // Mealy Machine
60  // Percorre a seq até a posição pos-1 (pois inicia em zero)
61  int e, i = 0;
62  do {
63  // Desloca todos os bits da engine uma posição para direita
64  for(e = SIZE_ENG-1; e > 0; e--) {
65  engine[e] = engine[e-1];
66  }
67  // Atualiza engine[0] com o bit seq[i]
68  engine[0] = seq[i++];
69  } while(i < pos);
70
71   // Efetua as transformações
72  // Obtém o header (n1n2)
73  int ee1, ee2; // elementos do engine
75  ee1 = engine[0];
76  ee2 = engine[SIZE_ENG-1];
77
78   for( int t = SIZE_ENG-1; t > 1; t--){
79   ee1 = ee1^engine[t];
80   }
81
82   for( int s = 0; s < SIZE_ENG-2; s++){
83   ee2 = ee2^engine[s];
84   }
85
86   hcn[0] = ee1 * 2 + ee2; // Em decimal
87  // Obtém o nó corrente (current)
88  int pot = 0;
89  for( int t = SIZE_ENG-1; t > 0; t--){
90  hcn[1] = engine[t]*pow(2,pot) + hcn[1];
91  pot++;
92  }
```

```
 93  // Obtém o próximo nó (destino)
 94  int pot2 = N-1;
 95  for( int s = 0; s < SIZE_ENG-1; s++){
 96  hcn[2] = engine[s]*pow(2,pot2) + hcn[2];
 97  pot2--;
 98  }
 99
100  // Obtém a saída (n1n2)
101  out[0] = ee1;
102  out[1] = ee2;
103
104 }
105
106  void MealyMachine_decisora(int seq[], int pos, int hcn[],
107  int  out[], int N, int Index) {
108  hcn[0] = hcn[1] = hcn[2] = 0;
109
110  int SIZE_ENG = N+1;
111  int engine[MAX];
112  int copy = Index;
113  for( int q = SIZE_ENG-1; q > 0; q--){
114 engine[q] = copy%2;
115 copy = copy/2;
116  }
117  // Atualiza engine[0] com o bit seq[i]
118  engine[0] = seq[pos];
119
120  // Efetua as transformações
121  // Obtém o header (n1n2)
122  int ee1, ee2; // elementos do engine
123  ee1 = engine[0];
124  ee2 = engine[SIZE_ENG-1];
125
126  for( int t = SIZE_ENG-1; t > 1; t--){
127  ee1 = ee1^engine[t];
128  }
129
130  for( int s = 0; s < SIZE_ENG-2; s++){
131  ee2 = ee2^engine[s];
132  }
133
134  hcn[0] = ee1 * 2 + ee2; // Em decimal
135  // Obtém o nó corrente (current)
136  int pot = 0;
137  for( int t = SIZE_ENG-1; t > 0; t--){
138  hcn[1] = engine[t]*pow(2,pot) + hcn[1];
139  pot++;
140  }
```

```
141  // Obtém o próximo nó (destino)
142  int pot2 = N-1;
143  for( int s = 0; s < SIZE_ENG-1; s++){
144  hcn[2] = engine[s]*pow(2,pot2) + hcn[2];
145  pot2--;
146  }
147
148  // Obtém a saída (n1n2)
149  out[0] = ee1;
150  out[1] = ee2;
151
152  }
153
154  void TCNode::initialize()
155  {
156
157  // Leitura do parâmetro "Grau de escalabilidade da rede "2^n"
158   N = simulation.getSystemModule()->par("n");
159
160   // Sequência de entrada
161   if(N == 3){
162   // Se o número de nós da rede for 8 (2^N), escolher a
163   //sequência
164       seq[0]= 1;
165       seq[1]= 0;
166       seq[2]= 1;
167       seq[3]= 1;
168       seq[4]= 0;
168       seq[5]= 0;
170       seq[6]= 0;
171       seq[7]= 0;
172   }else{
173  // Se o número de nós da rede for diferente de 8, a sequência
174   //é formada genericamente
175       // Gerador da sequência; 0..N/2 = 1, N/2..N = 0
176       for(int i = 0; i < pow(2,N); i++){
177             if(i < pow(2,N)/2 )
178                   seq[i] = 1;
179             else
180                   seq[i] = 0;
181       }
182   }
183
184   arrivalSignal = registerSignal("arrival");
185
186   // Atribuição de valores aos parâmetros de análise
187   tx_interval = par("Tx_interval");
188   tprocessamento_MM = par("T_proc");
189   t_guarda = par("T_guard");
190   t_espera = par("Time_out");
191   e_tx = par("E_tx");
```

```
192  e_rx = par("E_rx");
193  e_proc = par("E_proc");
194
195  Tp.setName("Tempo de processamento");
196  Energy.setName("Energia consumida");
197
198  // Node 0 envia a primeira mensagem
199  if (getIndex()==0)
200  {
201  // Inicia o processo marcando a primeira mensagem como
202  //self-message
203      char first[20];
204      sprintf(first,"SEQ )
205      %d%d%d%d%d%d%d%d",seq[0],seq[1],seq[2],seq[3],
206      seq[4],seq[5],seq[6],seq[7]);
207      TCNETmsg *msg = new TCNETmsg(first);
208      scheduleAt(0.0 + simTime(), msg);
209
210  }else
211   latencia = tprocessamento_MM + tx_interval;
212  }
213
214  int inst = 0;
215  void TCNode::handleMessage(cMessage *msg)
216  {
217      bool falha[MAX];
218 for(int i = 0 ; i < pow(2,N); i++)
219      falha[i] = false;
220
221 //// ESCOLHER NÓS QUE VÃO FALHAR
222 ////////////////////////////////////////////
223 //// PARA ISSO, COLOQUE O ÍNDICE DO NÓ QUE DESEJA FALHAR NO
224 //  ÍNDICE DO VETOR
225 //// EX.: falha[ÍNDICE DO NÓ] = true;
226 //// Para que mais nós falhem, copia a sentença e indique
227   // outro índice
228
229    falha[3] = true;
230    falha[1] = true;
231    falha[2] = true;
232
233
234  ////////////////////////////////////////////////////////
235
236 TCNETmsg *ttmsg = check_and_cast<TCNETmsg *>(msg);
237
238 // Obtém a posição a ser executada
239  int pos = ttmsg->getHopCount();
240  if( pos >= pow(2,N) ) pos = 0;
241
242  // Obtém o destino da mensagem
```

```
242  int dest = ttmsg->getDestination();
243
244  // Calcula os possíveis destinos em caso de falha
245  int dest1_falha = dest + pow(2,N)/2;
246  int dest2_falha = dest - pow(2,N)/2;
247
248  bool houve_falha = false;
249  if(falha[dest] == true){
250      houve_falha = true;
251  }
252
253  if(getIndex() == dest){
254      if (falha[getIndex()] == false)
255      {
256            // Mensagem chegou
257            delete ttmsg;
258
259            // Gerar nova mensagem
260
261            // Executar a máquina
262            pos++;
263            MealyMachine(seq, pos, hcn, out, N);
264            position = pos;
265
266            EV << "Generating another message: ";
267            TCNETmsg *newmsg = generateMessage();
268            EV << newmsg << endl;
269            forwardMessage(newmsg);
270
271
272                  // RECEBER MENSAGEM
273                  if(inst != 0)
274                  latencia = latencia + t_guarda + t_espera;
275                  Tp.record(latencia);
276                  recordScalar("Tempo de processamento",
277                   latencia);
278
279                  // PROCESSAR MENSAGEM
230                  latencia = latencia +
231                  (pos)*tprocessamento_MM;
232                  Tp.record(latencia);
233                  recordScalar("Tempo de processamento",
234                   latencia);
235
236                  // ENVIAR MENSAGEM
237                  latencia = latencia + tx_interval;
238                  Tp.record(latencia);
239                  recordScalar("Tempo de processamento",
240                   latencia);
241
242      }else{
```

```
245          pos++;
246
247          // RECEBER MENSAGEM
248          latencia = latencia + t_guarda +
249         ((int)houve_falha)*t_espera;
250          Tp.record(latencia);
251          recordScalar("Tempo de processamento", latencia);
252
253          // PROCESSAR MENSAGEM
254          latencia = latencia + (pos)*tprocessamento_MM;
255          Tp.record(latencia);
256          recordScalar("Tempo de processamento", latencia);
257
258          latencia = latencia + tx_interval;
259
260          delete ttmsg;
261          EV << "DELETANDO MSG" << endl;
262     }
263  }
264
265 inst++;
266 }
267
268    TCNETmsg *TCNode::generateMessage()
269 {
270  // Produz os endereços de nó atual e nó destino
271  int src = hcn[1];
272  int dest = hcn[2];
273
274  // Salva a mensagem como um pacote de caracteres.
275  char msgname[30];
276
277  TCNETmsg *msg = new TCNETmsg(msgname);
278  msg->setSource(src);
279  msg->setDestination(dest);
280  msg->setHopCount(position);
281
282  return msg;
289 }
290
291  void TCNode::forwardMessage(TCNETmsg *msg)
292  {
293  EV << "NO ATUAL = " << hcn[1] << endl;
294  EV << "NO DESTINO = " << k << endl;
295  EV << "Forwarding message " << msg << endl;
296  for(int j=0; j < pow(2,N); j++)
297  {
298     TCNETmsg *copy = (TCNETmsg *) msg->dup();
299     send(copy, "out", j);
300   //send(msg, "out", k);
301  }
```

ANEXO E

Código fonte do algoritmo AODV (framework) simulação no OMNet++ adaptado para essa pesquisa

aodv.ned

```
1  [General]
2  #debug-on-errors = true
3  sim-time-limit = 3000s
4  seed-0-mt = 5
5  network = inet.examples.adhoc.net80211_aodv.Net80211_aodv
6
7  cmdenv-express-mode = true
8  tkenv-plugin-path = ../../../etc/plugins
9
10 description = "Aodv Simple test"
11
12 *.playgroundSizeX = 1000
13 *.playgroundSizeY = 1000
14 *.numFixHosts = 8
15 *.numHosts = 1
16
17 # mobility
18 **.fixhost[7].mobility.x = 899
19 **.fixhost[7].mobility.y = 899
20
21 **.host[0].mobility.x = 1
22 **.host[0].mobility.y = 1
23
24 **.host[*].mobility.x = -1
25 **.host[*].mobility.y = -1
26
27 **.host*.mobilityType = "inet.mobility.NullMobility"
28
29 # udp apps (on)
30 **.host[*].udpAppType = "UDPBasicBurst"
31 **.host[0].numUdpApps = 1
32
33 **.udpApp[0].destAddresses = "fixhost[0]"
34 **.udpApp[0].localPort = 1234
35 **.udpApp[0].destPort = 1234
36 **.udpApp[0].messageLength = 512B  #
37 **.udpApp[0].messageFreq = 0.2s
38 **.udpApp[0].message_freq_jitter = uniform(-0.001s,0.001s)
39 **.udpApp[0].burstDuration = 0
40 **.udpApp[0].activeBurst=true
41 # **.udpApp[0].burstDuration = uniform(1s,4s,1)
42 # **.udpApp[0].time_off = uniform(20s,40s,1)
43 **.udpApp[0].time_off = 0s
44 **.udpApp[0].time_end = 0s
45 ##**.udpApp[0].time_begin =uniform(0s,4s,1)
46 **.udpApp[0].time_begin = 10s
```

```
47  **.udpApp[0].limitDelay = 20s
48  **.udpApp[0].rand_generator = 0
49
50  **.fixhost[0].udpAppType = "UDPSink"
51  **.fixhost[0].numUdpApps = 1
52  **.fixhost[0].udpApp[0].localPort = 1234
53  #**.fixhost[0].x=-1
54  #**.fixhost[0].y=-1
55
56  # tcp apps (off)
57  **.numTcpApps = 0
58  **.tcpAppType = "TelnetApp"
59
60  # ping app (host[0] pinged by others)
61  # ping app (off)
62  **.pingApp.count = 0
63  **.pingApp.startTime = 1s
64  **.pingApp.stopTime = 0
65  **.pingApp.printPing = true
66
67  # tcp settings
68  **.tcp.mss = 1024
69  **.tcp.advertisedWindow = 14336  # 14*mss
70  **.tcp.sendQueueClass = "TCPMsgBasedSendQueue"
71  **.tcp.receiveQueueClass = "TCPMsgBasedRcvQueue"
72  **.tcp.tcpAlgorithmClass = "TCPReno"
73  **.tcp.recordStats = true
74
75  # ip settings
76  **.ip.procDelay = 10us
77  # **.IPForward=false
78
79
80 ##############################################################
81   manet routing
82  **.manetrouting.manetmanager.routingProtocol = "AODV"
83  ######################################################
84
85  # nic settings
86  **.wlan.mgmt.frameCapacity = 10
87  **.wlan.mac.maxQueueSize = 14
88  **.wlan.mac.rtsThresholdBytes = 3000B
89  **.wlan.mac.bitrate = 54Mbps
90  **.wlan.mac.basicBitrate = 6Mbps  # 24Mbps
91  **.wlan.mac.retryLimit = 7
92  **.wlan.mac.cwMinData = 31
93  **.wlan.mac.cwMinBroadcast = 31
94
95  # channel physical parameters
96  *.channelcontrol.pMax = 2.0mW
97
98  **.wlan.radio.transmitterPower=2.0mW
99  **.wlan.radio.bitrate=54Mbps
100**.wlan.radio.sensitivity=-90dBm
101**.wlan.radio.berTableFile="per_table_80211g_Trivellato.dat"
102
103
104**.broadCastDelay=uniform(0s,0.005s)
```