Pós-Graduação em Ciência da Computação

"Avaliação Integrada de Consumo de Energia e Confiabilidade em Rede Sensores Sem Fio usando Modelos"

Por

Antônio Vicente Lourenço Dâmaso

Tese de Doutorado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, Dezembro/2015

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Antônio Vicente Lourenço Dâmaso

"Avaliação Integrada de Consumo de Energia e Confiabilidade em Rede Sensores Sem Fio usando Modelos"

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientador: *Nelson Souto Rosa*
Co-Orientador: *Paulo Romero Martins Maciel*

RECIFE, Dezembro/2015

Esta tese foi financiada pela FACEPE (Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco) através do Processo N° IBPG-0396-1.03/10.

Agradecimentos

Eu gostaria de dedicar esta tese á todas as pessoas envolvidas diretamente ou indiretamente neste trabalho. Aos meus orientadores, Nelson Rosa e Paulo Maciel; à minha família, Lúcia e Mariane; e à minha esposa Dayane. Obrigado a todos pela força e ajuda nessa trajetória.

Resumo

Rede de Sensores Sem Fio (RSSF) é um tipo de rede *ad hoc* formada tipicamente por centenas de pequenos dispositivos, chamados de *nós sensores*, os quais cooperam entre si para coletar e enviar informações até um nó sorvedouro. Esses nós sensores possuem recursos limitados de processamento, armazenamento e energia. Geralmente, a RSSF é implantada para coleta de informações em ambientes de difícil acesso, impossibilitando a substituição dos nós sensores caso apresentem alguma falha ou quando a energia acaba. Sendo assim, planejar e estimar o tempo de vida (consumo de energia) e a qualidade do serviço (confiabilidade) de uma RSSF antes de implantá-la são atividades cruciais.

Existem duas deficiências quando se observam soluções para o problema mencionado: elas se concentram na avaliação do consumo da aplicação ou apenas da infraestrutura de comunicação(e.g., protocolos de comunicação); e, quando os trabalhos avaliam os dois juntos, eles avaliam ou o consumo de energia ou a confiabilidade, e não ambas. Tais deficiências devem ser resolvidas para se ter um bom planejamento da RSSF.

Neste cenário, esta tese propõe uma metodologia para guiar o usuário no desenvolvimento de RSSFs levando em consideração o consumo de energia e a confiabilidade das aplicações e da infraestrutura. A metodologia orienta o usuário no planejamento, codificação, otimização, validação e implantação da RSSF. Com relação à avaliação, a metodologia inclui quatro conjuntos de modelos formais baseados em *Coloured Petri Ned* (CPN) e em *Reliability Block Diagram* (RBD) para avaliar o consumo de energia e a confiabilidade, respectivamente. Esses quatro conjuntos de modelos são criados através de um processo de composição usando pequenos modelos reusáveis.

Todas as atividades da metodologia são suportadas por um conjunto de ferramentas que automatiza a avaliação das RSSFs. Os modelos CPN e RBD foram validados através de experimentos, comparando os resultados obtidos à medições e dados encontrados na literatura. Uma análise de sensibilidade foi realizada para identificar quais fatores tem maior impacto sobre o consumo de energia e a confiabilidade das RSSFs.

As contribuições foram a metodologia, unindo o planejamento da aplicação com da infraestrutura da RSSF e avaliando o consumo de energia e a confiabilidade de forma integrada; os modelos formais baseados nas instancias da RSSF; o conjunto de ferramentas para suportar a metodologia proposta; e a análise de sensibilidade, que mostrou quais fatores afetam mais o consumo e a confiabilidade da RSSF.

Palavras-chave: Rede de Sensores Sem Fio, Consumo de Energia, Confiabilidade, CPN, RBD, Análise de Sensibilidade.

Abstract

Wireless Sensor Network (WSN) is an ad hoc network typically formed by hundreds of small devices called *sensor nodes*, which cooperate to collect and send information to a sink node. These sensor nodes have limited resources processing, storage and energy. Generally, the WSN is deployed to collect information in inaccessible environments, making it impossible to replace the sensors if they have any failure or when the energy is over. In this way, plan and estimate the lifetime (power consumption) and quality of service (reliability) of a WSN, considering the application and infrastructure before deploying are crucial activities.

There are two deficiencies when looking at solutions for the mentioned problem: they focus on evaluating the power consumption of the application or just the infrastructure of WSN; and, when the studies evaluate both together, or they evaluate power consumption and reliability, not both. These deficiencies should be addressed to have a good planning WSN.

In this scenario, this paper proposes a methodology to guide the user in the WSNs development, considering the power consumption and reliability of the application and the infrastructure of the WSN. The methodology guides the user in the design, coding, optimization, validation and deployment of WSN. Regarding the evaluation, the methodology includes four sets of formal models based on Coloured Petri Ned (CPN) and Reliability Block Diagram (RBD) to evaluate the power consumption and reliability of the WSN, respectively. These four models are created by a process of composition using small reusable models.

All activities of the methodology are supported by a set of tools that automates the WSN evaluation. CPN and RBD models were validated by experiments, comparing the results obtained by the models and measurements or study in the literature. A sensitivity analysis was performed to identify which factors impact on power consumption and reliability of the WSN using the proposed models.

The contributions were the methodology, combining the planning of the application with the WSN infrastructure and evaluating power consumption and reliability in an integrated way; formal models based on WSN instances; the set of tools to support the proposed methodology; and the sensitivity analysis, which showed which factors most affect the consumption and reliability of WSN.

Keywords: Wireless Sensor Networks, Power Consumption, Reliability, CPN, RBD, Sensitivity Analysis.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

PN	*Petri Net*
CPN	*Coloured Petri Net*
RSSF	Rede de Sensores Sem Fio
RBD	*Reliability Block Diagram*
SDP	*Sum of Disjoint Products*
SC	*Synchronous Code*
AC	*Asynchronous Code*
EDEN	*Evaluation and Development Enviroment for Wireless Sensor Network*
IDEA4WSN	*Integrated Developed Enviroment Energy-Aware for WSN*
Vident	*Vident is EDEN Evaluator*

1 Introdução

Este capítulo apresenta inicialmente as motivações do trabalho, com foco no problema de como realizar a avaliação integrada do consumo de energia e confiabilidade das Rede de Sensores sem Fio (RSSFs). Em seguida, ele apresenta algumas soluções existentes e suas limitações, e introduz a solução proposta para resolvê-lo. O capítulo termina com a apresentação da estruturação da tese.

1.1 Motivação

O avanço em algumas áreas como microprocessadores e comunicação sem fio possibilitou o surgimento de *nós sensores*, que são circuitos integrados de baixo custo e com baixa capacidade computacional (Agre and Clare, 2000). Eles são formados por uma ou mais placas de sensoriamento que podem ter um ou mais sensores, um microcontrolador e um transmissor (Akyildiz *et al.*, 2002). Geralmente, o nó sensor é alimentado por uma bateria não recarregável e não é operável diretamente, dificultando a sua substituição quando apresenta algum defeito ou quando a energia acaba. O nó sensor possui uma interface padrão de comunicação com a placa de sensoriamento (*e.g.*, MTS300), permitindo acoplar uma variedade de sensores (*e.g.*, vídeo, áudio, sensor de presença, entre outros) e desenvolver vários tipos de aplicações. Uma atividade padrão do nó sensor é monitorar e coletar informações do ambiente, *e.g.*, temperatura, luminosidade, pressão e poluição, usando um ou mais sensores.

Por usar um microprocessador, é possível configurar diversas aplicações no nó sensor, que vão de atividades simples (*e.g.*, como uma aplicação que coleta e envia a temperatura periodicamente) às mais complexas (*e.g.*, uma aplicação com capacidade de reconfiguração de acordo com o contexto do ambiente). O transmissor do nó é utilizado para enviar o dado coletado ou o resultado de um processamento para um usuário, e/ou para

rotear dados dos outros nós sensores. Geralmente, o transmissor é de baixa potência, necessitando a cooperação entre dois ou mais nós sensores em curtas distâncias.

Em função da limitação de recursos, os nós sensores não suportam os sistemas operacionais tradicionais. Por este motivo, diversos sistemas operacionais específicos foram criados, *e.g.*, TinyOS (Levis *et al.*, 2003) e Contiki (Dunkels *et al.*, 2004), para facilitar o desenvolvimento de aplicações, oferecendo módulos já implementados que são usados para abstrair a complexidade do hardware. No caso do TinyOS, os nós sensores só executam uma única aplicação, porque o sistema operacional é compilado junto com ela. Desta maneira, o compilador seleciona os módulos que de fato serão utilizados pela aplicação, diminuindo o arquivo compilado e otimizando a utilização dos recursos.

O conjunto de nós sensores forma uma rede *ad hoc*, chamada Rede de Sensores Sem Fio (RSSF), que não possui topologia fixa e se adapta dinamicamente à entrada e saída dos nós sensores na rede. Geralmente, essa rede é formada por centenas ou milhares de nós sensores coletando dados do ambiente e encaminhando os pacotes dos outros nós sensores através de um mecanismo de multi-saltos. Em alguns casos, os nós sensores podem ser eleitos apenas com o roteamento para otimizar o consumo de energia (Heinzelman, 2000; Heinzelman *et al.*, 2000). Por esse motivo, ela é uma rede altamente escalável e possui a capacidade de auto-configuração à medida que os nós sensores vão entrando ou saindo dela. Devido à quantidade de participantes e à utilização de multi-saltos, os nós sensores mais distantes podem enviar os dados coletados até que cheguem ao usuário. De fato, os nós sensores enviam os dados para o nó sorvedouro que repassa-os para o usuário. Este, geralmente, está conectado a uma fonte de energia contínua e tem conexão com uma rede externa como a Internet.

Outras características dessa rede são os protocolos e as estratégias de implantação. A RSSF possui uma variedade de protocolos que atende aos requisitos da rede (*e.g.*, escalabilidade, autoconfiguração, *ad hoc*) e do nó sensor (*e.g.*, baixo consumo de energia, pouca capacidade de processamento). Existem, por exemplo, protocolos que determinam os caminhos entre os nós sensores e o nó sorvedouro quando a rede é inicializada. Da mesma forma, existem aqueles que só determinam o caminho quando houver a necessidade de enviar um pacote. Por este motivo, alguns protocolos podem considerar a rede plana (todos os nós sensores têm o mesmo papel) ou hierárquica (são formados *clusters* e atribuídos papéis diferentes na rede).

Além dos protocolos, existem várias formas de implantar os nós sensores para construir uma RSSF. É possível colocá-los de forma aleatória (*e.g.*, jogando-os no ambiente) ou organizada (*e.g.*, a distância entre os nós sensores é a mesma). Pode-se também

colocá-los seguindo uma regra específica (*e.g.*, os nós sensores estão entre 10 a 50 metros de distância do nó sorvedouro). A estratégia de implantação adotada irá impactar no tempo de vida da rede (Chen *et al.*, 2005) e, por este motivo, devem-se observar os objetivos da aplicação e as limitações da infra-estrutura.

Independentemente da estratégia de implantação utilizada, é comum existir mais de um nó sensor analisando o mesmo fenômeno (*e.g.*, temperatura) na mesma área ou região. Os nós sensores presentes na mesma região irão enviar o mesmo valor para o nó sorvedouro. Essa estratégia torna a rede mais confiável, permitindo que os dados de qualquer região da RSSF cheguem mesmo havendo falha na rede. Alternativamente, a RSSF também pode agregar os valores em algum ponto da rede, diminuindo o consumo de energia nos nós sensores.

Devido às suas características, as RSSFs têm sido empregadas em uma grande variedade de aplicações (Akyildiz *et al.*, 2002), por exemplo militares (*e.g.*, detecção de intruso, monitoramento de munição, avaliação de ataques químicos, nuclear ou biológico em um terminado local) e monitoramento ambiental (*e.g.*, monitoramento de vulcões, análise de incêndio em florestas densas). Geralmente, a RSSF é colocada em áreas de difícil acesso ou hostis para o ser humano. Dessa maneira, a rede não fica comprometida caso um nó sensor falhe, pois há um alto grau de redundância de fontes de dados e caminhos até o nó sorvedouro.

Independente da aplicação e do local de implantação, é fundamental (1) planejar a rede definindo o comportamento da aplicação, a quantidade de nós sensores, o sistema operacional, os protocolos a serem utilizados, o tamanho da área de atuação, a distribuição dos nós sensores; (2) avaliar o consumo de energia para estimar o tempo de vida da rede; e (3) e avaliar a confiabilidade para estimar a qualidade do serviço oferecido da RSSF. No entanto, essas etapas possuem vários problemas que serão apresentados a seguir.

1.2 Problema

Como foi dito anteriormente, uma RSSF pode ser formada por milhares de nós sensores e é normalmente implantada em ambientes hostis. Por conta da quantidade dos nós sensores e dessas condições do ambiente, dificilmente um nó sensor é trocado quando a energia acaba ou quando apresenta algum defeito. O responsável por projetar a RSSF deve se preocupar em escolher o hardware adequado e desenvolver aplicações com baixo consumo de energia, criando uma rede que dure o maior tempo possível e que mantenha a qualidade do serviço. Sendo assim, o responsável por projetar a RSSF deve definir as

estratégias para diminuir o consumo de energia e para melhorar a confiabilidade da RSSF, se preocupando para que elas não interfiram negativamente na RSSF.

Para entender melhor isso, algumas estratégias serão apresentadas para explicar como elas podem interferir negativamente no consumo de energia ou na confiabilidade da RSSF caso não sejam usadas corretamente. As estratégias escolhidas estão relacionadas à comunicação entre os nós sensores porque é a parte que consome mais energia (Akyildiz *et al.*, 2002).

O usuário pode optar por um protocolo de roteamento que use apenas um único caminho entre o remetente e o destinatário. Essa estratégia irá consumir a energia apenas dos nós sensores participantes do caminho. Para melhorar a confiabilidade, o usuário pode optar por protocolos de roteamento que criam múltiplos caminhos entre o remetente e o destinatário em vez de um único caminho. No entanto, mais nós sensores da RSSF irão participar na transmissão do dado, aumentando o consumo de energia da RSSF (Al-Karaki and Kamal, 2004). É importante ressaltar que a confiabilidade da RSSF pode ser satisfatória para o usuário usando apenas um único caminho, mas ela pode ser melhorada caso ele opte por um protocolo com múltiplos caminhos.

Outra alternativa para melhorar a confiabilidade é a retransmissão de dados. No entanto, os nós sensores podem precisar trocar mais informações entre si e, consequentemente, aumentarão o consumo de energia da RSSF. Existem diversas estratégias de retransmissão de dados para melhorar a confiabilidade, considerando aspectos de consumo de energia, como é descrito por Mahmood *et al.* (2015). No entanto, comparado com o cenário sem retransmissão de dados, qualquer uma dessas estrategias irá aumentar o consumo de energia.

Uma alternativa utilizada para melhorar o consumo de energia é elaborar um esquema para parte da rede ficar ativa (*e.g.*, analisando o ambiente e roteando os dados), enquanto a outra parte fica desativada (*e.g.*, sem nenhum tipo de processamento). Essa estratégia economiza muita energia. No entanto, se não for usada corretamente, pode acarretar em problemas de comunicação entre os nós sensores (Raghunathan *et al.*, 2006), tal como a colisão de pacotes (Suhonen *et al.*, 2009). Tais problemas interferem na confiabilidade da RSSF.

Esses casos descritos anteriormente só reforçam a necessidade de planejar a RSSF e verificar se as estratégias escolhidas de fato estão auxiliando ou prejudicando a RSSF. Essa verificação pode ser feita através da avaliação da RSSF.

Conhecendo o consumo de energia da aplicação em um determinado hardware, é possível estimar o tempo de vida de toda a rede. No entanto, mesmo sabendo o tempo

de vida da rede, não é possível prever se ela, mesmo com energia, está funcionando corretamente. Neste ponto, o funcionamento correto implica em garantias de que dados coletados em qualquer ponto da rede são transmitidos até o nó sorvedouro, ou seja, se a rede é confiável. Na prática, alguns nós da RSSF podem estar sem energia, provocando problemas de conectividade para uma parte da rede até o nó sorvedouro. Logo, é preciso integrar o estudo do consumo de energia com a confiabilidade das RSSFs para entender e mensurar o impacto do primeiro sobre o segundo.

O consumo de energia e a confiabilidade podem ser avaliados em dois momentos: na implantação ou no desenvolvimento da RSSF. No primeiro momento, na implantação, a avaliação é feita utilizando os nós sensores reais e quando a RSSF já está em funcionamento. Em outras palavras, todas as estratégias para melhorar o consumo de energia e a confiabilidade já foram avaliadas, a aplicação já está desenvolvida e o hardware já foi adquirido. No entanto, pode ser um momento tardio para descobrir que umas dessas escolhas está interferindo negativamente em outra, aumentando o custo e o tempo para efetuar qualquer modificação. Por exemplo, a RSSF utiliza redundância (para aumentar a confiabilidade) com um protocolo hierárquico (para melhorar o consumo de energia): os nós sensores enviam mais mensagens (consumindo mais energia), mas o *cluster head* ainda continua sendo o gargalo da comunicação. São duas soluções boas para a confiabilidade e para o consumo de energia, mas podem não trazer um bom resultado quando aplicadas juntas.

Por outro lado, uma alternativa é avaliar o consumo de energia e a confiabilidade na fase de projeto ou de desenvolvimento das RSSFs, permitindo que alterações sejam feitas e avaliadas antes de implantá-la. Nessa fase, é necessário utilizar modelos computacionais para simular ou emular o comportamento dos nós sensores e, consequentemente, da rede. Essa abordagem tem o potencial de produzir resultados confiáveis tanto quanto no primeiro momento (utilizando nós sensores reais) e elimina os problemas citados anteriormente (tempo e custo). No entanto, um único modelo para avaliar o consumo de energia e a confiabilidade pode requerer uma grande capacidade de processamento, necessitando de mais computadores físicos ou tornando a avaliação inviável. Desta maneira, o grande problema a ser resolvido é como avaliar o consumo de energia e a confiabilidade de RSSFs de forma integrada, considerando aspectos da aplicação (*e.g.*, comportamento da aplicação) e da rede (*e.g.*, protocolo de roteamento, quantidade de nós sensores, estratégia de implantação da RSSF, entre outros).

Portanto, a solução proposta é criar um conjunto de modelos, um para o consumo de energia e outro para a confiabilidade, e uma metodologia suportada por um conjunto de

ferramentas para integrá-los. Já existem alternativas para avaliar separadamente (como será mostrado na próxima seção) o consumo de energia e a confiabilidade da RSSF usando modelos computacionais, mas não há uma metodologia para conectá-los. Além disso, o problema do processamento ainda pode existir e pode ser resolvido através da solicitação de servidores em um ambiente de nuvem. Por este motivo, a solução de tais problemas (de como avaliar o consumo de energia e a confiabilidade da RSSF de forma integrada) é o foco desta tese.

1.3 Soluções Parciais

O consumo de energia e a confiabilidade da RSSF podem ser avaliados através de três métodos: medição, sistema computacional, e modelagem analítica. A medição avalia a RSSF usando os próprios nós sensores físicos, os quais executam uma determinada aplicação. Para avaliar a confiabilidade, é necessário apenas dois ou mais nós sensores e uma aplicação para gerar pacotes na rede. No caso do consumo de energia, podem ser necessários diversos equipamentos (*e.g.*, osciloscópio, fonte de tensão contínua, e assim por diante), que estarão conectados a um ou mais nós sensores. A medição é ideal para obter o valor exato do consumo de energia ou a confiabilidade da RSSF por avaliar diretamente o nó sensor. No entanto, ela é custosa, por necessitar de diversos equipamentos; demorada, por trabalhar com o tempo real (por exemplo, avaliar o consumo de energia de 1 hora de transmissão); e tediosa, por ser um processo totalmente manual (permitindo que haja erros humanos). Alguns trabalhos utilizaram esse método para avaliar o consumo de energia de diversos sistemas operacionais (Lajara *et al.*, 2010), algoritmos de criptografia (Chang *et al.*, 2007), e middleware (Hiltunen *et al.*, 2012). Com relação à confiabilidade, existem estudos que mediram as falhas do hardware e, consequentemente, da rede, inserindo-as artificialmente (Cinque *et al.*, 2009) e outros estudos que avaliam as falhas que ocorrem naturalmente na RSSF ocasionadas por ruído, interferências, distância e outros fatores (Parameswaran *et al.*, 2009).

O sistema computacional para simular ou para emular a RSSF é uma alternativa para substituir a medição tanto do consumo de energia, quanto da confiabilidade. Nessa abordagem, pode-se simular ou emular o comportamento do hardware, da aplicação ou da rede, utilizando softwares. Dessa maneira, não é necessário adquirir nenhum equipamento para avaliar a rede, permitindo avaliar redes pequenas ou grandes. Outra vantagem é que esse método simula ou emula o comportamento da RSSF que levaria dias/semanas (tempo real do sistema modelado) em questões de segundos/minutos (tempo da simulação, tempo

necessário para executar o software). Para avaliar o consumo de energia, Haase *et al.* (2011) dividem sistema computacional em três grandes grupos: emuladores de hardware, emuladores de sistema operacional e simuladores de rede. A emulação (hardware ou sistema operacional) necessita do código fonte da aplicação para avaliar o consumo de energia. A vantagem de utilizar a emulação do hardware é avaliar a aplicação tal como na plataforma real, independente do sistema operacional e da linguagem utilizados. No entanto, essa característica tem uma grande desvantagem: ela não é escalável (Haase *et al.*, 2011). Por considerar o comportamento do nó sensor em baixo nível, pois simula os circuitos integrados do hardware, ela dificulta a emulação de diversos nós sensores ao mesmo tempo. Por este motivo, a alternativa é emular o sistema operacional que abstrai o comportamento do hardware, mas vincula a emulação a um sistema operacional. Essa abstração melhora o desempenho da emulação permitindo avaliar mais nós sensores na rede, mas deixa menos preciso o resultado final por ignorar aspectos do hardware, como, por exemplo, o TinyOS, que ignora as manipulações de interrupções feitas pelo microprocessador (Levis *et al.*, 2003). Portanto, caso o foco não seja a aplicação no nó sensor e sim a rede (*e.g.*, elaborar um novo protocolo de roteamento), mais adequado é utilizar um simulador de rede, *e.g.*, NS-2 (de Berkeley, 2011), NS-3 (Henderson *et al.*, 2006), PAWiS (Glaser *et al.*, 2008) e Castalia (Boulis, 2007). Geralmente, esses simuladores não precisam do código fonte da aplicação para simular a rede, pois fornecem um conjunto de modelos que, quando combinados, representam uma RSSF completa.

Os modelos computacionais para simular ou emular a RSSF também podem ser utilizados para avaliar a confiabilidade. Nesse contexto, novos modelos são acrescentados aos simuladores ou emuladores para simular as falhas na rede, por exemplo, ocasionadas por interferências ou ruídos. Por exemplo, existem modelos de propagação simples que não consideram perdas de pacotes causados pelo ambiente (Rappaport *et al.*, 1996); como também os que considera as imperfeiçoeis na comunicação e a interferência do ambiente na comunicação se aproximando da realidade de uma RSSF, *e.g.*, o Radio Irregularity Model (Zhou *et al.*, 2006). Esses modelos adicionados permitem avaliar melhor as condições de perda, colisão ou erro dos pacotes. Esse cenário é ideal para avaliar, por exemplo, novos protocolos de controle de congestionamento, como foi feito por Wang *et al.* (2006).

Por fim, a modelagem analítica utiliza representação matemática para modelar e avaliar a RSSF. A grande vantagem desse método é a rapidez para obter um resultado. A modelagem analítica é muito utilizada para avaliar a confiabilidade por conta da sua simplicidade: é necessário detectar os pontos de falha da rede e associar uma probabili-

dade dela não operar corretamente ou dela falhar. Existem vários estudos que avaliam a confiabilidade da RSSF usando a modelagem analítica. Por exemplo, uma metodologia para avaliar a confiabilidade da RSSF (considerando as falhas no nó sensor e na rede) através da Árvore de Falha ou os protocolos da camada de transporte (Ghaffari and Rahmani, 2008). No entanto, a modelagem analítica não é tão difundida para avaliar o consumo de energia porque existem diversos fatores que influenciam o consumo de energia da RSSF, *e.g.*, número de vizinhos, tamanho do pacote, tempo de transmissão do rádio, seu tempo de recepção e tempo no modo *idle* do rádio, usa da CPU e da memória, assim por diante. Além disso, esses fatores possuem diversos estados, dificultando ainda mais a representação na modelagem analítica. Por esse motivo, os trabalhos encontrados na literatura se limitam a avaliar o consumo de energia de um único aspecto; por exemplo, eles ignoram o comportamento da aplicação e da rede, considerando apenas o comportamento de um protocolo; avaliam o comportamento de um protocolo roteamento (Manjeshwar *et al.*, 2002), o impacto de um protocolo MAC (Sahota *et al.*, 2011) ou os benefícios do LPL (*Low Power Listening*) (Cano *et al.*, 2009).

1.4 Solução Proposta

O principal problema que esta tese pretende resolver é **como avaliar o consumo de energia e a confiabilidade da RSSF de forma integrada**. Para isso, foi definida uma metodologia de desenvolvimento e avaliação da RSSF, foram propostos modelos formais para avaliar a RSSF, e um ambiente Web para suportar a metodologia, facilitando a avaliação da RSSF.

Observando a importância do planejamento da RSSF antes de implantá-la, esta tese propõe uma metodologia de desenvolvimento e avaliação do consumo de energia e da confiabilidade da RSSF. Esta metodologia é formada por uma sequência de atividades que guia o usuário desde o planejamento até a implantação da RSSF. Além disso, ela utiliza modelos formais para avaliar o consumo de energia e a confiabilidade da RSSF.

Antes de apresentar a metodologia e os modelos, é importante entender que o desenvolvimento da RSSF é focado na criação da aplicação e na configuração da rede, conjunto de informações contendo o protocolo de roteamento, posição dos nós sensores, posição do nó sorvedouro, tamanho da área, entre outros. Observando isso, a metodologia proposta define a mesma sequência de atividades para o desenvolvimento da aplicação e para a configuração de rede. Essa sequência de atividades é dividida em cinco fases: *Planejamento*, *Codificação*, *Otimização*, *Validação* e *Implantação*. Na fase *Planeja-*

mento, o usuário deve identificar os requisitos, definir métricas de consumo de energia e confiabilidade e planejar o desenvolvimento da RSSF. Logo após planejar a RSSF, na fase seguinte (*Codificação*), o usuário deve desenvolver o código fonte da aplicação e definir a configuração da infraestrutura da RSSF. Em seguida, ele deve analisar o código fonte da aplicação e a configuração da RSSF para encontrar inconsistências de implementação (*e.g.*, a RSSF utiliza o protocolo DIRECT, mas nem todos os nós sensores tem acesso ao nó sorvedouro), podendo melhorar o consumo de energia e/ou a confiabilidade da RSSF (fase *Otimização*). Para validar a RSSF, o usuário deve avaliar a RSSF (fase *Validação*) verificando se o comportamento e os resultados de consumo de energia e a confiabilidade da RSSF estão de acordo com os definidos na primeira fase. Caso eles estejam, a RSSF pode ser implantada (fase *Implantação*). Caso contrário, ele deve verificar se houve um erro no planejamento (fase *planificação*) ou no desenvolvimento (fase *Codificação*).

Em especial, a fase *Validação* permite que se faça a análise de sensibilidade da RSSF, onde é possível identificar qual fator está afetando mais o consumo de energia e a confiabilidade da RSSF. Adicionalmente, ela permite identificar qual configuração da rede teve o melhor resultado de acordo com os requisitos e métricas definidos na primeira fase.

Esta metodologia considera que a avaliação do consumo de energia e da confiabilidade da RSSF devem ser utilizadas na fase *Otimização* (para justificar as otimizações) e *Validação* (para auxiliar na validação da RSSF). Vários modelos devem ser usados para avaliar o consumo de energia e a confiabilidade da RSSF devido à complexidade de avaliar uma RSSF, *e.g.*, como avaliar a aplicação e como avaliar a configuração da RSSF. Além disso, esta metodologia ilustra como os modelos devem ser criados e avaliados, definindo uma ordem de avaliação. Mesmo existindo vários modelos, a metodologia mostra para o usuário uma visão única, avaliando como se fosse apenas um único modelo.

Esta tese definiu que a CPN (*Coulored Petri Net*) (Jensen, 1997; Jensen *et al.*, 2007; Jensen and Kristensen, 2009) deve ser utilizada para avaliar o consumo de energia e o RBD (*Reliability Block Diagram*) (Venkatesan *et al.*, 2013) para avaliar a confiabilidade da RSSF. O CPN e o RBD foram escolhidos por modelarem sistemas complexos (tal como o consumo de energia da RSSF) e pela simplicidade da modelagem para avaliar a confiabilidade de qualquer sistema.

Além disso, foram criados três conjuntos de modelos de consumo de energia (Modelo de Aplicação, Modelo de Rede e Modelo de Nó Sensor) e um conjunto de modelo de confiabilidade (Modelo de Região). O Modelo de Aplicação avalia o consumo de energia da aplicação baseado no código fonte da aplicação; O modelo de Rede, o consumo de

energia da rede baseado na configuração de rede; e o Modelo de Nó Sensor, permite avaliar o consumo de energia da aplicação e da rede baseados no código fonte da aplicação e na configuração da rede. Por fim, o Modelo de Região avalia a confiabilidade de um conjunto de nós sensores da RSSF (chamado de região) definidas na configuração da rede. Adicionalmente, esta tese considera que uma região é um conjunto de nós sensores redundantes e que estejam próximos, analisando o mesmo fenômenos físico e (caso um nó sensor apresente falha) a região ainda pode continuar ativa (funcionando).

Esses modelos, apesar de muito distintos, possuem uma estratégia em comum: eles são criados usando a estratégia dividir para conquistar. Utilizando essa estratégia, os modelos propostos podem avaliar qualquer código fonte da aplicação e qualquer configuração de rede.

Pequenos modelos reusáveis de consumo de energia, chamados modelos básicos, são combinados para expressar o consumo de energia de operadores das aplicações e de pilhas de protocolos. Por exemplo, modelos básicos que representam o consumo de energia de comandos e de estruturas de uma linguagem de programação são compostos para definir o consumo de uma função. Estes, por sua vez, são compostos para modelar o consumo da aplicação inteira, criando o Modelo de Aplicação.

Da mesma forma, o Modelo de Rede é formado por modelos de protocolos, os quais possuem o comportamento e o consumo de energia dos protocolos. A junção do Modelo de Aplicação com o Modelo de Rede forma o Modelo do Nó Sensor, o qual será utilizado para avaliar o consumo de energia da RSSF, considerando a aplicação e a rede no nó sensor.

Além do consumo de energia, é possível avaliar também a confiabilidade de uma região presente na RSSF. Modelos básicos que representam os pontos de falhas da RSSF (o nó sensor e o enlace de comunicação) são compostos para representar um caminho entre um nó sensor e o nó sorvedouro. Os caminhos são definidos por uma algoritmo de roteamento definido na configuração de rede. Por sua vez, esses caminhos modelados são compostos para representar uma região (chamado de Modelo de Região). Um conjunto de modelos das regiões permite avaliar a confiabilidade da RSSF.

O Modelo de Região considera que o nível de energia do nó sensor deve ser considerado com um dos principais fatores de falha dele. Esta característica reforça a existência de uma ordem de avaliação da RSSF: os modelos de consumo de energia devem ser avaliados antes do modelo de confiabilidade para cacular a confiabilidade dos nós sensores usando o nível de bateria ao longo do tempo.

Essa metodologia é suportada por um ambiente Web, chamado de EDEN (*Evaluation*

and Development Enviroment for Wireless Sensor Network), implantada em um ambiente de nuvem para apoiar todas as fases e atividades da metodologia, *e.g.*, criação da aplicação e da rede, e a composição e a avaliação dos modelos. Fazem parte desse ambiente quatro ferramentas: editor, tradutor, avaliador e gerenciador. O editor, chamado de IDEA4WSN (*Integrated Developed Enviroment Energy-Aware for WSN*), é usado para desenvolver o código fonte das aplicações na linguagem de programação nesC (Gay *et al.*, 2003) e elaborar a configuração de rede (escolhendo os protocolos, quantidade de nós sensores, entre outros parâmetros). Além disso, esta ferramenta analisa e sugere trechos de códigos mais otimizados (que consomem menos energia), tornando-a ciente de energia.

O tradutor (chamado *Sensor2Model*), por sua vez, converte o código fonte da aplicação e a configuração de rede nos respectivos modelos. Para isso, ele é responsável por executar a estrategia dividir para conquistar (explicada anteriormente) para compor os modelos usando como base o código fonte da aplicação e a configuração de rede. O avaliador chamado de *Vident* (*Vident is EDEN Evaluator*), ao receber o modelo gerado pelo tradutor, avalia os modelos para obter o consumo de energia ou a confiabilidade da RSSF. Por fim, o gerenciador é responsável por gerenciar as instâncias das outras ferramentas e a comunicação entre elas. Este ambiente possui uma arquitetura escalável por possibilitar que várias instâncias de editores, tradutores e avaliadores possam existir e possam dividir o processamento entre si.

1.4.1 Contribuições

As contribuições dessa tese estão sumarizadas em quatro partes: metodologia, modelos, o conjunto de ferramentas (ou ambiente Web) e análise de sensibilidade. Esta tese propõe uma metodologia para o desenvolvimento e avaliação do consumo de energia e da confiabilidade da RSSF. A avaliação da RSSF é feita através de modelos formais baseados no código fonte da aplicação e na configuração de rede. Para auxiliar o usuário, um conjunto de ferramentas foi criado para suportar a metodologia, automatizando a avaliação da RSSF. Por fim, a análise de sensibilidade avalia o impacto de cada fator no consumo de energia e na confiabilidade da RSSF.

A seguir, as contribuições são apresentadas separadamente.

Metodologia. A metodologia une os planejamentos da aplicação e da infraestrutura das RSSFs. Além disso, ela considera, como parte da validação, a avaliação do consumo de energia e da confiabilidade.

Modelos. Foram propostos modelos formais, os quais foram construídos com base no código fonte da aplicação e na configuração da infraestrutura da RSSF. Em especial, o

modelo de confiabilidade é construído com base no algoritmo de roteamento utilizado pela RSSF. As informações geradas por um modelo serviram como entrada para outros, criando uma ordem de execução entre os modelos e dando a visão única da solução.

Conjunto de Ferramentas. Esta tese também propõe um conjunto de ferramentas para dar suporte à metodologia proposta. Ele é formado por quatro ferramentas: um editor, para facilitar o desenvolvimento da RSSF; o tradutor, responsável por traduzir a aplicação e a infraestrutura da RSSF em modelos CPN e RBD; um avaliador, o qual avalia o consumo de energia ou a confiabilidade dos modelos gerados; e, por fim, o gerenciador, para gerenciar o processo de avaliação.

Análise de Sensibilidade. A análise de sensibilidade da RSSF foi feita para identificar quais fatores afetam mais o consumo de energia e a confiabilidade da RSSF.

1.5 Estrutura da Proposta

Este documento está estruturado em sete capítulos.

O Capítulo 2 introduz os conceitos básicos necessários ao entendimento deste trabalho, tais como: RSSF (Rede de Sensores Sem Fio), sistema operacional TinyOS, linguagem de programação chamada nesC, a pilha de protocolos da RSSF, CPN (*Coloured Petri Net*), RBD (*Reliability Block Diagram*) e Análise de Sensibilidade.

O Capítulo 3 discute os trabalhos relacionados encontrados na literatura. Estes trabalhos foram divididos em quatro grupos: os que avaliam apenas o consumo de energia da RSSF; os que avaliam apenas a confiabilidade da RSSF; os que avaliam o consumo de energia e a confiabilidade da RSSF; e os que fazem a análise de sensibilidade da RSSF ou variante, *e.g.*, UWSAN (*Underwater Wireless Sensor Array Networks*).

O Capítulo 4 descreve a metodologia proposta para o desenvolvimento e avaliação da RSSF. Ela é dividida em duas partes: a primeira parte está relacionada ao desenvolvimento da RSSF e a segunda parte apresenta o guia de avaliação do consumo de energia e confiabilidade da RSSF utilizando modelos.

Os modelos de consumo de energia e confiabilidade da RSSF citados no capítulo anterior são apresentados, com mais detalhes, no Capítulo 5. Este capítulo descreve os quatro conjuntos de modelos formais baseado no CPN e no RBD para avaliar o consumo de energia e a confiabilidade da RSSF, respectivamente.

O Capítulo 6 apresenta um ambiente elaborado para auxiliar no desenvolvimento e na avaliação do consumo de energia e da confiabilidade da RSSF. O ambiente proposto é formado por quatro ferramentas: o Editor, para auxiliar no desenvolvimento da RSSF; o

tradutor, para criar os modelos formais; o avaliador, para avaliar o consumo de energia e a confiabilidade da RSSF usando os modelos criados pelo tradutor; e o gerenciador, para gerenciar a interação entre as outras ferramentas. Este ambiente foi implantado em um ambiente de nuvem para aumentar a capacidade de processamento, principalmente, com relação a avaliação da RSSF.

O Capítulo 7 apresenta os experimentos utilizados para validar os modelos apresentados no Capítulo 5 e análise de sensibilidade para identificar quais fatores afetam mais o consumo de energia e a confiabilidade da RSSF.

Por fim, o Capítulo 8 apresenta as contribuições, limitações e trabalhos futuros.

2

Conceitos Básicos

Antes de apresentar a solução proposta, é preciso apresentar os conceitos básicos da RSSF (Rede de Sensores Sem Fio). Em seguida, são apresentados os conceitos sobre CPN (*Coloured Petri Net*) e RBD (*Reliability Block Diagram*), os quais foram usados para avaliar o consumo de energia e a confiabilidade da RSSF, respectivamente. E, por fim, é apresentado o conceito de Análise de Sensibilidade, adotado para detectar qual fator influencia mais o consumo de energia e a confiabilidade da RSSF.

2.1 Rede de Sensores Sem Fio

As Redes de Sensores Sem Fio (RSSFs) são resultados do melhoramento de três tecnologias (Agre and Clare, 2000): microprocessadores, comunicação sem fio e tecnologias microeletrônica (miniaturização dos componentes em escala microscópica). Uma RSSF é tipicamente formada por milhares de pequenos nós sensores com baixa capacidade computacional, poucos recursos disponíveis (*e.g.*, memória RAM), pequeno raio de comunicação e, geralmente, bateria não recarregável. O conjunto de nós sensores permite que a rede possa executar tarefas complexas, as quais dificilmente poderiam ser executadas por um único nó sensor (Akyildiz *et al.*, 2002). As RSSFs têm usualmente uma topologia dinâmica e cada nó sensor tem um equipamento de sensoriamento para analisar um fenômeno, *e.g.*, temperatura, luminosidade, abalos sísmicos, entre outros. Através desses equipamentos, ele pode capturar a ocorrência do fenômeno e transmiti-lo para o nó sorvedouro, responsável por disseminar o dado para o usuário remoto. Devido ao grande número de nós sensores participantes na rede, é possível utilizar a comunicação multi-saltos para enviar dados entre nós sensores distantes, usando esses nós sensores mais próximos para retransmitir os dados. Dessa maneira, a RSSF economiza energia na transmissão dos dados (Pottie and Kaiser, 2000).

2.1.1 Características e Classificações

As RSSFs possuem diversas características que podem ser sumarizadas como (Haenggi, 2004; Ruiz, 2003; Loureiro *et al.*, 2003): recurso limitado, rede *ad hoc*, topologia dinâmica, grande número de nós sensores, energia, entre outras.

Recursos Limitados. Os nós sensores possuem recursos restritos de processamento, de bateria e de comunicação. Uma arquitetura típica do nó sensor da RSSF é ilustrada na Figura 2.1. Ela possui um processador, memória RAM, armazenamento, transmissor, fonte de energia e um ou mais sensores. Por exemplo, as plataformas MICAz (MEMSIC, 2014b) e IRIS (MEMSIC, 2014a) são comumente usadas na RSSF e, praticamente, possuem as mesmas configurações: baseados no microcontrolador Atmel, transmissor IEEE 802.15.4, armazenamento de 128 Kb e memória RAM de 4 e 8 Kb, respectivamente.

Geralmente, a fonte de energia consiste de duas pilhas AA não recarregáveis. Como qualquer equipamento eletrônico, o nó sensor irá parar de funcionar quando as pilhas não tiverem mais energia suficiente para suprir a necessidade deste equipamento.

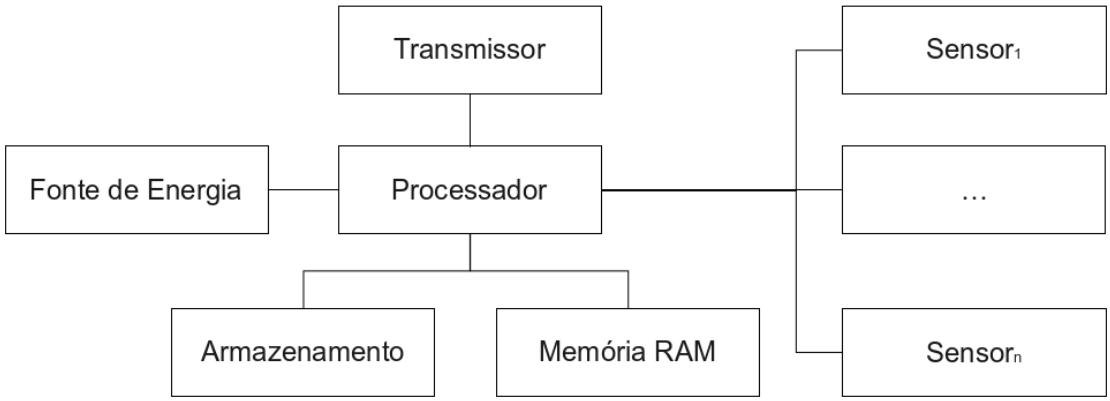


Figura 2.1 Arquitetura típica do nó sensor.

No caso da comunicação, o transmissor mais usado é o rádio por se adaptar melhor ao ambiente do que os outros citados (Vieira *et al.*, 2003). Ele trabalha com várias potências, onde as mais fortes alcançam longas distâncias e consomem mais energia. A quantidade de nós sensores e o multi-saltos permitem que o nó sensor trabalhe com pequenas potências para se comunicar. Algumas plataformas (*e.g.*, IRIS) utilizam a tecnologia ZigBee (IEEE 802.15.4) como padrão de comunicação por consumir menos energia do que o Bluetooth (IEEE 802.15.1), Wi-Fi (IEEE 802.11) e UWB (IEEE 802.15.3) (Lee *et al.*, 2007). Mais recentemente, uma nova tecnologia de transmissão de dados tem sido utilizada: *Bluetooth Low Energy* - BLE (Bluetooth, 2010). Como alguns estudos mostram

(Siekkinen *et al.*, 2012; Dementyev *et al.*, 2013), ela consome menos energia do que o ZigBee.

Sistemas operacionais como o TinyOS possibilitam a comunicação entre nós sensores através de uma abstração baseada no *Active Messages* (Von Eicken *et al.*, 1992). A sua utilização permite enviar 36 bytes em um único pacote, o qual utiliza 1 byte para identificar o nó sensor remetente (TinyOS, 2008). Os demais bytes servem para identificar o nó destinatário, o tipo da mensagem, o tamanho da mensagem e os dados da aplicação.

Rede Ad Hoc. As RSSFs não possuem um nó sensor central responsável exclusivamente pelo roteamento do pacote da rede, tal como um ponto de acesso em uma rede Wi-Fi. Os nós sensores participantes dessa rede possuem duas atividades: sensoriamento e roteamento. A atividade de sensoriamento é a aplicação propriamente dita, a qual interage com o ambiente no qual está instalada e gera pacotes para o nó sorvedouro. A outra atividade é a de auxiliar o roteamento dos pacotes gerados pelos outros nós sensores. A partir do momento que os nós identificam os seus vizinhos, a rede pode ser considerada como montada e os pacotes podem ser roteados.

Topologia Dinâmica. Outra característica relacionada à RSSF é o fato da topologia ser dinâmica. Por causa das variações do ambiente e, principalmente, do nível de energia dos nós sensores, a topologia da RSSF varia frequentemente. Existem estratégias (Jardosh and Ranjan, 2008; Basagni *et al.*, 2004) para os nós sensores informarem quando entram ou saem da rede, ou quando a bateria está se exaurindo.

Escalabilidade. As RSSFs são normalmente formadas por um número elevado de nós, que geralmente estão espalhados de forma assimétrica em um espaço físico. Por este motivo, a RSSF deve ser escalável para trabalhar com grande número de nós sensores. Deste modo, a escalabilidade facilita a formação da rede *ad hoc*, possibilitando que qualquer participante da mesma possa enviar seus dados coletados para o nó sorvedouro, independentemente da distância entre eles.

Energia. Existem diversos meios para reduzir o consumo de energia da aplicação e do hardware. Para a aplicação, o projetista da rede pode escolher protocolos mais eficientes, que consomem menos energia, *e.g.*, optar por protocolos *cross-layer* (Heinzelman, 2000; Heinzelman *et al.*, 2000); diminuir o consumo de energia da CPU evitando o uso de threads (Levis *et al.*, 2005); e esquematizar a atividade dos nós sensores permitindo que uma parte da rede fique inativa, enquanto outra fica ativa. Além da aplicação, o projetista da rede pode escolher um hardware mais eficiente energeticamente. Também é possível recuperar a energia da bateria usando equipamentos de captação de energia, tais como, placas fotovoltaicas.

Tempo de vida. Existe uma forte relação entre o consumo de energia e o tempo de vida da RSSF, pois a rede só existe se houver nós sensores ativos. Por tal condição, é comum avaliar o consumo de energia da RSSF e estimar o tempo de vida dela por conta dessa relação. Além das estratégias de redução do consumo de energia, é possível estender o tempo de vida da rede adicionando novos nós sensores periodicamente já que é difícil substituir as baterias ou os nós sensores com defeitos devido ao local onde as RSSFs normalmente se encontra.

Nó Sorvedouro. As RSSFs possuem um elemento fundamental para funcionar: o nó sorvedouro. Ele se diferencia dos demais por intermediar a comunicação entre a RSSF e uma rede externa como a Internet, e por esse motivo está conectado a uma fonte contínua de energia. Os nós sensores enviam as informações coletadas (*e.g.*, temperatura) para o nó sorvedouro, o qual irá repassá-las para um usuário remoto. Adicionalmente, a comunicação na RSSF é do tipo *event-to-sink*: os nós sensores se comunicam com o nó sorvedouro quando um determinado evento ocorrer. Essa característica é diferente das redes tradicionais, onde a comunicação é tipicamente *end-to-end*. Por ser um elemento chave, uma RSSF com mais de um nó sorvedouro em diferentes pontos pode melhorar o consumo de energia dos nós sensores e, consequentemente, o tempo de vida da rede (Kim *et al.*, 2005a). Uma outra opção é o nó sorvedouro ser móvel dentro da RSSF, equilibrando o consumo de energia (Yang *et al.*, 2010).

Agregação de Dados. Vários nós sensores no mesmo lugar avaliando o mesmo ambiente podem causar um problema: uma quantidade imensa de dados. Como dito anteriormente, os dados coletados são enviados para o nó sorvedouro. Como os dados são roteados através de multi-saltos, a energia dos nós sensores pode se exaurir apenas com o roteamento das informações dos outros nós sensores. No entanto, em vez de enviar vários pacotes, os nós sensores sumarizam (ou agregam) um conjunto de dados e enviam apenas o dado agregado, diminuindo o impacto na rede. A agregação pode ocorrer em dois pontos: no remetente, o nó sensores pode enviar apenas um único pacote com o valor da sumarização, ou no percurso, um nó sensor na rede agrega todos os pacotes que recebeu e envia apenas um para o nó sorvedouro. Dessa maneira, a agregação de dados tem sido uma funcionalidade essencial para roteamento na RSSF (Intanagonwiwat *et al.*, 2000). Vários algoritmos de agregação de dados podem ser utilizados, por exemplo, o CLUDDA (Chatterjea and Havinga, 2003). Da mesma forma, várias estratégias foram propostas para saber o seu impacto na RSSF (Krishnamachari *et al.*, 2002) e, por fim, alguns deles foram incorporados aos algoritmos de roteamento, *e.g.*, LEACH (Heinzelman, 2000; Heinzelman *et al.*, 2000).

As RSSFs têm sido classificadas levando-se em consideração três parâmetros (Ruiz, 2003): configuração, sensoriamento e comunicação. A configuração foca na organização da RSSF e leva em consideração cinco aspectos: composição, organização, mobilidade, densidade e distribuição. A composição indica se a rede é formada por diversos modelos de nós sensores (heterogênea) ou se é formada pelo mesmo modelo (homogênea). A organização observa se a rede é plana ou hierárquica, se possui *clusters*. A mobilidade define se os sensores participantes da RSSF são capazes de se locomover para outro lugar ou são estáticos. A densidade indica a concentração dos nós em relação à rede, se está balanceada ou desregulada (se tem muitos ou poucos nós sensores participantes). A distribuição é bem similar à densidade, mas está relacionada à concentração dos nós em relação à área ocupada. Caso existam mais nós sensores em uma área do que em outra, a rede caracteriza-se como irregular. Caso contrário, é caracterizada como regular.

O sensoriamento está relacionado à coleta dos dados nos nós sensores. Caso os dados sejam coletados em intervalos regulares, o sensoriamento é caracterizado como periódico. Caso os dados apenas sejam coletados quando houver um determinado evento ou quando o nó sensor é acionado, o sensoriamento é reativo. Por fim, é caracterizado como contínuo quando as RSSFs coletam dados sem interrupção, continuamente. O sensoriamento do tipo continuo difere do tipo periódico por não haver um tempo (grande) entre as coletas.

Finalmente, a comunicação está relacionada à forma como os dados são transmitidos na rede. Este item é divido em cinco partes: disseminação, conexão, transmissão, alocação de canal e fluxo de informação. A disseminação observa como os dados são enviados na rede e pode ser do tipo programada, quando os dados são enviados em tempos regulares; contínua, a todo momento enviam dados; ou sob demanda, quando responde à uma requisição do observador ou quando detecta determinados eventos.

De acordo com a topologia da rede e dos tipos dos sensores, a comunicação física entre dois pontos (nós sensores) pode ser apenas unidirecional (assimétrica) ou bidirecional (simétrica). Essa diferença caracteriza o tipo de conexão da rede. A transmissão refere-se à capacidade do nó sensor em decidir o sentido da comunicação, que pode ser imposta pela topologia ou pela aplicação. A comunicação pode ser apenas transmitir dados (*Simplex*), transmitir e receber dados em momentos diferentes (*Half-duplex*) ou transmitir e receber ao mesmo tempo *Full-duplex*. A alocação do canal está relacionada à divisão do canal de comunicação entre os nós para minimizar as interferências (colisões) nas transmissões: estática, divisão igualitária do canal entre os nós; e dinâmica, quando os nós possuem tempo de posse do canal variável. O fluxo de informação está relacionado ao roteamento do dado pela rede. Caso o dado seja enviado em *broadcast*, a rede é

denominada *Flooding*. Caso seja apenas enviado para os seus vizinhos, ela é considerada *Multicast*. Caso seja do tipo *Unicast*, todos os nós sensores têm comunicação direta com o nó sorvedouro. Caso seja *Gossiping*, os nós sensores têm a capacidade de selecionar os nós para rotear os dados. Por fim, caso seja *Bargaining*, há uma negociação entre nós antes da transmissão dos dados.

2.1.2 TinyOS and nesC

O TinyOS (Levis *et al.*, 2005) é um sistema operacional baseado em eventos de código aberto, especialmente desenvolvido para executar em equipamentos com baixa capacidade de processamento e com armazenamento limitado, com os nós sensores da RSSF. O TinyOS é executado em diferentes plataformas, tal como IRIS (MEMSIC, 2014a), MICAz (MEMSIC, 2014b), e provê facilidades para o desenvolvimento das aplicações. O TinyOS foi desenvolvido em nesC (Gay *et al.*, 2003), uma extensão da linguagem C, otimizada para equipamentos com baixa capacidade de armazenamento.

Em termos de programação, uma aplicação em nesC consiste de componentes conectados, em que um componente tem uma especificação e uma implementação. A especificação define as ações dos provedores e do usuários, enquanto a implementação consiste do código nesC que implementa a especificação. Na prática, o provedor caracteriza as funcionalidades do componente e o usuário define as funcionalidades desejadas pelo componente.

Uma interface em nesC é bidirecional e é usada para comunicar os componentes. A interface especifica um conjunto de funções chamadas de comandos e eventos. Esses comandos devem ser implementados pelo provedor da interface, enquanto os eventos devem ser implementados pelos usuários da interface. Desse modo, interações entre componentes podem ser muito complexas e, tipicamente, um componente registra interesse em algum evento que é sinalizado quando ocorre. Por exemplo, um componente que invoca um comando parecido com "leia temperatura"deve implementar um evento "leia temperatura concluído", que é executado logo após o comando "leia temperatura". Na prática, os comandos tipicamente vão da aplicação em sentido ao hardware ("downwards") e os eventos estão no sentido oposto, do hardware para a aplicação ("upwards"). As tarefas são outra função presente no componente. Elas são um tipo especial de função que não retorna nada (`void`) e não têm nenhum argumento. Uma tarefa é usada para atividades demoradas e, diferente dos comandos e dos eventos, é inserida em uma fila para ser executada depois por um *schedule*, seguindo a política FIFO (*First In, First Out*).

A linguagem NesC possui um modelo de execução para evitar conflitos de interesse.

Ela executa uma tarefa por vez (não preemptiva), no entanto, existem os manipuladores de interrupções que são sinalizados assincronamente pelo hardware, os quais são preemptivos. Como foi dito anteriormente, o *scheduler* é responsável por executar as tarefas em uma ordem e elas não podem interromper a atividade uma da outra, ou seja, elas são atômicas entre si. No entanto, elas não são atômicas em relação aos manipuladores de interrupções. Como consequência desse modelo de execução, o código da aplicação nesC pode ser dividido em duas partes: código síncrono (em inglês, *Synchronous Command* - SC) e código assíncrono (em inglês, *Asynchronous Command* - AC). O SC é acessível a partir de tarefas e inclui as funções, comandos, eventos e tarefas. Por outro lado, o AC é acessível a partir de pelo menos um manipulador de interrupção, o qual pode executar eventos diretamente ou indiretamente (tornando-os AC).

Em termos de programação, nesC tem o mesmo conjunto de operadores da linguagem C: matemáticos (+, *, -, /), chamada de função, atribuição (=, +=, -=, *=, /=), manipuladores de bits (<, >, &, |), lógicos (>, >=, <, <=, ==, !=), *cast*, e expressões primárias (identificadores, constantes, *string*, expressões parametrizadas, entre outros). Em especial, nesC introduz novos operadores de chamada de função: `call` α usado para invocar um comando (α); `signal` ε, para chamar um evento (ε); e `post` ω, para invocar uma tarefa (ω).

Adicionalmente, a linguagem tem estruturas divididas em três categorias: seleção, interação e desvios. As estruturas de seleção escolhem um ramo para executar baseado na execução de uma expressão de controle. As estruturas de seleção podem ter vários ramos e cada um tem uma expressão booleana e um corpo. A estrutura de seleção executa o primeiro ramo, o qual irá executar o seu corpo caso a expressão de controle retorne `true`. Esse tipo de estrutura possui dois representantes (*if-then-else* e *switch*) como são mostrados a seguir:

```
if( exp )
  { lista de expressão; }
else if( exp )
  { lista de expressão; }
…
else
  { lista de expressão; }
```

(a)

```
switch( exp )
{
  case 0: lista de expressão;
  case 1: lista de expressão;
   …
  default: lista de expressão;
}
```

(b)

As estruturas de repetição executam um conjunto de operadores (chamado de corpo) repetidamente até a expressão de controle retornar `false` (SGI, 2014). A linguagem nesC tem três estruturas: *while*, *do-while* e *for*. A expressão de controle do *while* é

verificada antes da execução do corpo, enquanto a expressão de controle do *do-while* é verificada depois. Finalmente, o *for* é usado para executar o corpo *n* vezes. Essa estrutura tem três expressões: a primeira expressão é associada à inicialização; a segunda expressão, ao controle, o qual é verificado antes da interação; e a última, usualmente especifica o incremento.

```
while( exp )
{
  lista de exp;
}
```
(a)

```
do
{
  lista de exp;
} while( exp );
```
(b)

```
for( exp ; exp ; exp )
{
  lista de exp;
}
```
(c)

O controle de fluxo em uma estrutura de repetição pode ser alterado por uma estrutura de desvio. Uma estrutura de desvio causa uma transferência de controle (SGI, 2014). Existem três estruturas estruturas de desvios: *continue*, *break* e *return*. O *continue* ignora os operadores subsequentes e passa o controle para a expressão de controle. O *break* termina a execução do *switch* (única estrutura de seleção afetada por ele) ou qualquer estrutura de interação. Por fim, o *return* finaliza uma função.

2.1.3 Pilha de Protocolos

Devido às características das RSSFs, vários protocolos de comunicação foram desenvolvidos para, principalmente, atender aos requisitos de energia e de escalabilidade. Esses protocolos estão organizados em quatro camadas: aplicação, transporte, rede e enlace. Os protocolos da camada de aplicação estão relacionados à atividade da aplicação, *e.g.*, *Constrained Application Protocol* (CoAP) (Shelby *et al.*, 2014) e Lightweight Simple Object Access Protocol (Lightweight SOAP) (Moritz *et al.*, 2011).

A camada de transporte está relacionada ao controle de congestionamento e com a confiabilidade da comunicação fim-a-fim (*end-to-end*) ou ponto-a-ponto (*hop-by-hop*) (Wang *et al.*, 2006). Muitas vezes os protocolos dessa camada são responsáveis por incrementar a segurança (*i.e.*, criptografa o dado transmitido) ou a qualidade do serviço (*i.e.*, a disponibilidade da informação). Os protocolos ESRT (*Event-to-Sink Reliable Transport*) (Sankarasubramaniam *et al.*, 2003) e RMST (*Reliable Multi-Segment Transport*) (Stann and Heidemann, 2003) são exemplos de protocolos desta camada.

A camada de rede é responsável por determinar um percurso entre a origem e o destino (Al-Karaki and Kamal, 2004). Os protocolos dessa camada são tipicamente os protocolos de roteamento e eles podem determinar se a comunicação é direta (único

salto) entre a origem e o destino ou utilizar vários nós sensores para rotear o pacote (multi-saltos). Os protocolos de único salto são utilizados em pequenas redes, onde os nós sensores possuem comunicação direta com o nó sorvedouro.

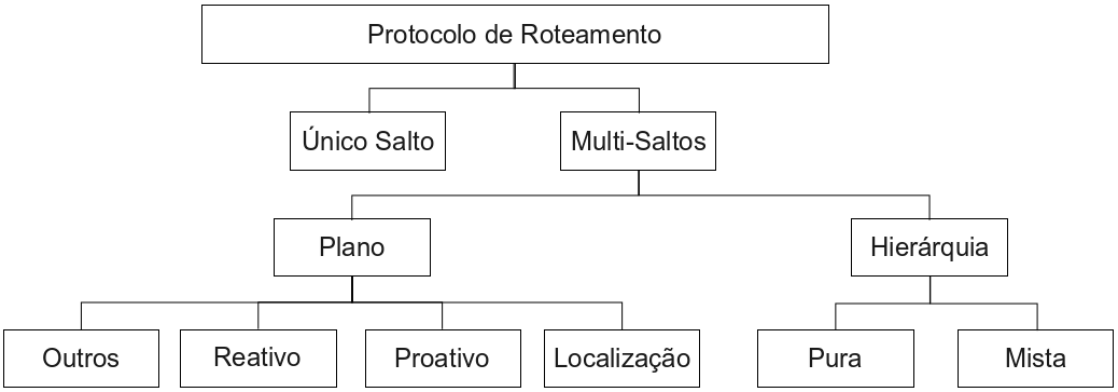


Figura 2.2 Tipos de protocolos de roteamento.

Como os nós sensores possuem um transmissor de baixa potência e, como as RSSFs tendem a ser grandes, nem todos os nós sensores têm comunicação direta com o nó sorvedouro. Para resolver esse problema, existem os protocolos de multi-saltos, os quais podem elaborar redes planas, hierárquicas ou mistas (veja a Figura 2.2). As redes planas atribuem a mesma tarefa a todos os nós sensores: eles cooperam entre si para rotear o pacote até o nó sorvedouro. A diferença entre um protocolo e o outro, quando cria uma rede plana, é o modo como cada um descobre (ou cria) o percurso até o nó sorvedouro. Eles podem ser Proativos, os quais determinam o percurso logo quando o nó sensor for ativado ou quando houver mudanças na RSSF; Reativos, responsáveis por criar um caminho quando houver a necessidade de transmissão; Baseados em Localização, os quais utilizam as localizações dos outros nós sensores para roteador os pacotes; ou aqueles que usam estratégias particulares para transmitir um pacote. Por exemplo, aqueles protocolos que enviam aleatoriamente os pacotes, ou enviam em *broadcast* ou *multicast*, ou decidem enviar pela rota com maior energia.

Protocolos de roteamento que criam redes hierárquicas definem *clusters* na RSSF. Esses *clusters* possuem um líder, chamado de *Cluster Head* (CH), o qual irá receber dados dos participantes do *cluster* e os enviará para o nó sorvedouro. Nas redes hierárquicas puras, os CHs possuem comunicação diretamente com o nó sorvedouro. No caso das mistas, os CHs irão colaborar entre si para rotear os pacotes de outros CHs. Adicionalmente, os CHs, antes de enviar, agregam as informações dos participantes do *cluster*, enviando apenas um único pacote.

A última camada, a camada de enlace, é responsável por corrigir erros nos quadros e controlar o acesso ao meio (MAC). Em especial, os protocolos MAC possuem um papel importante nessa camada: eles criam canais de comunicação e determinam o compartilhamento do ambiente entre os nós sensores (Aslam *et al.*, 2009). Além disso, eles são responsáveis por otimizar o consumo de energia dessa camada para evitar a colisão entre pacotes, a interceptação de pacotes de outros nós sensores (*overhearing*), a criação demasiada de pacotes de controle, entre outros.

Arquiteturas em camadas são muito bem adaptada à rede cabeada, mas não para redes sem fio, porque o ambiente sem fio permite uma comunicação modular mais rica do que a rede cabeada (Srivastava and Motani, 2005). Devido à transparência entre as camadas, os protocolos acabam não cooperando com protocolos de outras camadas, e isso pode gerar mais pacotes na rede. Para resolver esse problema, a solução é adotar uma estratégia *cross-layer*, quebrando esta transparência, criando novas interfaces de comunicação entre as camadas ou unindo as camadas próximas para sugir uma super camada. Um dos protocolos *cross-layer* mais populares é o LEACH (Heinzelman, 2000; Heinzelman *et al.*, 2000), o qual pode ser considerado um protocolo da camada de rede por rotear dados de forma hierárquica, mas se preocupa em evitar colisões.

2.2 Rede de Petri

Redes de Petri (PN) (Desel and Reisig, 1998) é uma família de técnicas formais usadas para modelar uma grande variedade de sistemas através de uma notação gráfica. Atualmente, existem diversas extensões definidas para facilitar a modelagem/avaliação de diferentes cenários. Estas extensões são classificadas em Rede de Petri de baixo nível ou alto nível. Um dos representantes da Rede de Petri de alto nível é a CPN (*Coloured Petri Net*), a qual permite modelar sistemas complexos combinando Rede de Petri com uma linguagem de programação. A rede *Place/Transition* é uma das Rede de Petri de baixo nível e serve de base para as demais extensões. Por este motivo, ela será explicada primeiro para facilitar o entendimento da *Coloured Petri Net*.

2.2.1 Rede Place/Transition

Uma Rede *Place/Transition* é um dígrafo com dois tipos de vértices: os lugares e as transições. Os lugares representam os estados dos sistemas, enquanto as transições, as ações ou eventos. Um arco dirigido é utilizado para conectar os dois tipos de vértices. Caso o arco inicie em um lugar, ele deverá terminar obrigatoriamente em uma transição e

vice-versa. Além desses três elementos, existe o *token*, o qual sinaliza o estado atual do sistema.

Os lugares podem ter um ou mais *tokens*, os quais podem se mudar de um lugar a outro, alterando o estado do sistema. Para a mudança de lugar, é preciso que haja uma transição entre esses dois lugares que, quando disparada, representa uma ação efetuada dentro do sistema. Por fim, para uma transição ser disparada, é necessário que ela esteja habilitada. Para uma transição t ficar habilitada, é necessário existir um *token* para cada arco iniciando no lugar l e terminando na transição t.

A Figura 2.3 ilustra os quatros elementos de Rede de Petri, onde o lugar é representado por um círculo (a); a transição, por um retângulo (b); o arco, por uma reta (c); e o *token*, por uma círculo preto preenchido (d).

Figura 2.3 Elementos básicos da Rede de Petri: (a) lugar, (b) transição, (c) arco e (d) *token*.

Formalmente, uma Rede de Petri *Place/Transition* é definida como LT = (P , T , A , F_P , M_0), onde:

- P é um conjunto de lugares l;
- T é um conjunto de transições t;
- A é um conjunto de arcos *(t,l)* ou *(l,t)*;
- F_P define quantas marcas são necessárias para executar determinada transição e quantas são geradas como resultado da execução de certa transição; e
- M_0 é a marcação inicial da rede.

Algumas propriedades podem ser verificadas sobre o modelo em Rede de Petri, tais como *reachability*, se a marcação M_n é alcançável (se existe uma sequência de disparos) a partir de M_0; *boundedness*, se o número de *tokens* em cada lugar nunca ultrapassa o valor igual a k; *liveness*, se existem transições que nunca serão disparadas, *i.e.*, se existem transições mortas; e *reversibility*, se é possível retornar ao estado inicial (Murata, 1989).

2.2.2 Coloured Petri Net

Coloured Petri net (CPN) (Jensen, 1997; Jensen *et al.*, 2007; Jensen and Kristensen, 2009) é um modelo que é tanto orientado a estado quanto à ação, e combina a capacidade de Rede de Petri com a capacidade de uma linguagem de programação de alto nível. A CPN ML é uma linguagem de programação funcional baseada em ML (Milner *et al.*, 1990), a qual provê as primitivas para a definição de tipos de dados simples ou complexo (chamados de cor) e manipuladores de dados (funções). Além disso, CPN permite que os *tokens* sejam de uma cor e tenham um valor (chamado de *token colour*) atribuído a ele; os lugares armazenam *tokens* de um determinada cor; e as transições podem ler e manipular os *token colours* usando os manipuladores de dados. Adicionalmente, CPN considera a noção de tempo, permitindo capturar o tempo de execução tomado por um evento no sistema. Assim, a CPN pode ser aplicada para avaliar desempenho. É importante ressaltar que a adição da linguagem de programação e os tipos complexos (chamados de cores) possibilitam modelar cenários complexos de maneira simples, fatores fundamentais em estudos de desempenho e de consumo de energia. Por exemplo, Maciel *et al.* (1996) modelou o mesmo problema, jantar dos filósofos, usando PN e CPN, mostrando que o modelo CPN é mais simples do que o PN.

CPN Tools (Westergaard and Verbeek, 2014) é a principal ferramenta usada para elaborar *Coloured Petri net*. Cada lugar da rede tem um nome, uma cor e um marcador inicial. O nome identifica o lugar, a cor determina quais *tokens* podem ser armazenados lá, e o lugar pode também ter um valor inicial. Esses valores iniciais, os quais são transformados em *tokens*, são usados para iniciar o modelo CPN. A Figura 2.4(a) ilustra um lugar (place01) contendo um *token*, cujo o seu valor é zero. Fora do lugar, existem um marcador inicial (atribuído como 0) e uma cor (INT).

A Figura 2.4(b) mostra um arco com uma única anotação que é uma variável (*i*) do mesmo tipo do lugar de saída ou de entrada. Além de uma variável, o arco pode ter uma expressão como anotação que pode elaborar ações no valor do *token* (por exemplo, mudar o valor dele). A Figura 2.4(c) ilustra uma transição que contém um identificador (*trans01*), guarda (*[i = 0]*), tempo (*@+1*) e código. A guarda é uma expressão booleana, a qual informa as condições para habilitar a transição. Quando a transição é disparada, o tempo adiciona um determinado valor ao tempo no modelo CPN e o código atribuído para a transição é disparado. O código é divido em três partes: *input*, *output* e *action*. O *action* contém o código CPN ML para executar uma atividade que processa a entrada (*input*) e retorna uma valor (*output*).

A CPN hierárquica tem como objetivo representar o modelo CPN em diferentes níveis

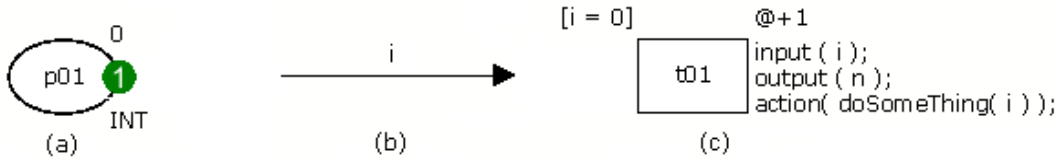


Figura 2.4 Elementos básicos da CPN: (a) lugar e *token*, (b) arco e (c) transição.

de abstrações que podem ser conectadas umas com as outras. Essa característica é muito útil para construir modelos grandes, *i.e.*, para representar uma aplicação. Na prática, essa característica permite dividir o modelo em vários módulos, onde cada módulo tem uma CPN. Além disso, ela permite o reuso dos módulos, o que reduz o tamanho do modelo. Esses modelos independentes (chamados de sub módulos) são representados por uma transição no módulo principal. Esses módulos são conectados através de lugares e os conectores sempre envolvem apenas dois lugares: um para o sub módulo (chamado de porta) e outro para o super módulo (chamado de *socket*). O *socket* passa o *token* para a respectiva porta e vice-versa. Existem três tipos de portas: entrada, saída e I/O. Uma porta do tipo entrada recebe um *token* do *socket*; uma porta do tipo saída envia um *token* para o super módulo e a porta do tipo I/O permite ambas as atividades (recebe e envia um *token*).

CPN Tools tem um importante funcionalidade (chamado de monitor) para extrair periodicamente informações sobre as marcações dos lugares ou parar a simulação quando a transição X for disparada, por exemplo. Um monitor pode ser usado para observar, inspecionar, parar, controlar ou modificar a simulação de um modelo CPN. Na prática, o monitor pode observar e executar ações baseadas na observação. Existem quatro tipos de monitores chamados *breakpoint* (para parar a simulação), *data collector* (para extrair dados numéricos da rede), *write-in-files* (para atualizar um arquivo durante a simulação) e *user-defined* (propósito geral).

Tal como a Rede de Petri anterior, CPN também é representada pela notação algébrica (além da representação gráfica) CPN = (P , T , A , S_{color} , P_{color} , T_{color} , A_{color} , G_{color} , M_0), onde:

- P é um conjunto de lugares l;
- T é um conjunto de transições t;
- A é um conjunto de arcos (t,l) ou (l,t);
- S_{color} representa o conjunto de cores, expressões, guardas e funções disponíveis para serem usadas na Rede de Petri.

- P_{color} representam um conjunto de elementos, os quais mapeiam um lugar (*e.g.*, *p01*) com uma cor (*e.g.*, INT).

- T_{color} e G_{color} representam um conjunto de funções (*e.g.*, `action( doSomeThing(i) )`) associado à uma transição (*e.g.*, *t01*).

- A_{color} representa um conjunto de expressões (*e.g.*, *i*) usadas por um arco

- G_{color} representa um conjunto de guardas (*e.g.*, `[i = 0]`) associada à uma transição (*e.g.*, *t01*).

- M_0 é a marcação inicial (tal como na notação algébrica da Rede de Petri Place/-Transition).

Resumindo, os conjuntos de lugares (P), transições (T), arcos (A) e marcações inicias (M_0) são definidos da mesma forma em CPN e em PN. Tais elementos são encontrados na Figura 2.4. As outras variáveis estão relacionadas com a linguagem de programação e com o mapeamento das cores/funções com os elementos da Rede de Petri (definidos em P, T ou A).

2.3 Reability Block Diagram

Reability Block Diagram (RBD) (Henley and Kumamoto, 1981) permite representar a confiabilidade de um sistema através de um conjunto de blocos, os quais representam a menor unidade do sistema (*e.g.*, o roteador pode ser considerado como a menor unidade em uma rede, considerado como o sistema). O RBD possui um ponto de entrada onde inicia-se o processo de avaliação e um ponto de saída onde termina a avaliação. Entre a entrada e a saída, os blocos são organizados em série, paralelo ou combinando-os. Esses blocos possuem apenas dois estados: funcionando ou em falha. As falhas são independentes umas das outras. Cada bloco possui uma confiabilidade associada: o bloco B_i possui uma confiabilidade $R_i(t)$ de funcionar no tempo t. Similarmente, o seu complemento $(1 - R_i(t))$ representa a falha no tempo t. Adicionalmente, a confiabilidade de um bloco ou de um sistema é dada entre zero (sempre falhará) e 1 (sempre funcionará).

Quando os componentes estão organizados em série, é necessário apenas que um componente falhe para o sistema falhar. Um sistema com n componentes em série, como ilustra a Figura 2.5 (a), possui uma confiabilidade (no tempo t) igual ao produto da confiabilidade dos blocos que o compõe, como mostra a Equação 2.1. Por exemplo, um

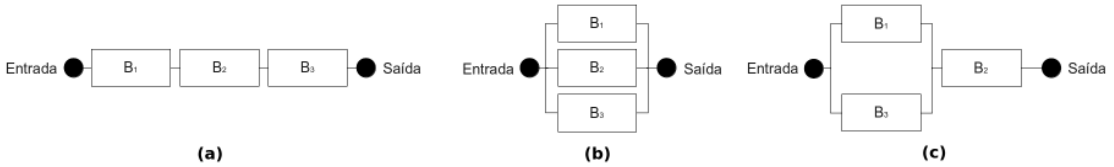


Figura 2.5 Exemplo de RBD quando os componentes estão em (a) série, (b) paralelo ou (c) combinado.

RBD com três blocos, onde cada um possui uma confiabilidade igual á 0,9, possui uma confiabilidade igual à 0,729.

$$R(t) = \prod_{i=1}^{n} R_i(t) \tag{2.1}$$

Quando os blocos estão organizados em paralelo, o sistema só irá falhar quando todos os elementos falharem. Desta maneira, um sistema com n componentes, tal como mostra a Figura 2.5 (b), possui uma confiabilidade igual ao complemento de todos os blocos falharem no tempo t (como mostra a Equação 2.2).

$$R(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t)) \tag{2.2}$$

Por exemplo, a confiabilidade de um sistema com três blocos, onde cada um possui uma confiabilidade igual à 0,9, é igual à 0,999. Adicionalmente, é importante observar como a organização dos blocos interfere no resultado final do sistema.

Além dessas duas organizações, é possível combiná-las, criando outros formatos, tais como: série-paralelo, paralelo-série, ponte e *k-out-of-n*. Esses novos formatos utilizam as Equações 2.1 e 2.2 para calcular a confiabilidade do sistema. Por exemplo, a Figura 2.5 (c) representa um RBD série-paralelo, onde os blocos B_1 e B_2 estão em paralelo, e o B_3 está em série. Inicialmente, é preciso calcular a confiabilidade dos blocos em paralelo e, depois, dos blocos em série. Desta maneira, se cada bloco possui uma confiabilidade igual à 0,9, o resultado final do sistema será igual à 0,891.

2.3.1 Caminhos mínimos e Cortes mínimos

O caminho mínimo é um conjunto de componentes organizados em série que garante a operação do sistema (Kuo and Zuo, 2003). Por outro lado, o corte mínimo é um conjunto de componentes que implica na falha do sistema (Kuo and Zuo, 2003). Por exemplo, a Figura 2.5 (a) tem um único caminho mínimo ($B_1B_2B_3$) e tem três cortes mínimos (B_1,B_2,B_3).

2.3.2 Soma dos Produtos Disjuntos

O método Soma dos Produtos Disjuntos (em inglês, *Sum of Disjoint Products* - SDP) é baseado na álgebra Booleana e avalia a probabilidade de um sistema funcionar através da união dos caminhos mínimos ou de um sistema falhar através da união dos cortes mínimos. Se dois ou mais eventos não tem componente em comum, a probabilidade de um dos eventos ocorrer é a soma das probabilidades dos eventos individuais (Kuo and Zuo, 2003). Se dois eventos A e B tem elementos em comum, temos a seguinte equação para avaliar a probabilidade dos eventos A e B (A ∪ B) (Kuo and Zuo, 2003):

$$R(A \cup B) = R(A) + R(\overline{A}B) \tag{2.3}$$

Sendo assim, um sistema com n eventos (A_1, A_2, ... , A_n) tem:

$$R(A_1 \cup A_2 \cup ... \cup A_n) = R(A_1) + R(\overline{A_1}\, A_2) + ... + R(\overline{A_1}\, \overline{A_2} \, ... \, \overline{A_{n-1}}\, A_n) \tag{2.4}$$

2.4 Análise de Sensibilidade

Um sistema pode ser caracterizado como uma caixa preta que recebe um conjunto de entradas (parâmetros) e gera um conjunto de saídas (resultados). Quando nós mudamos os parâmetros, o sistemas pode gerar novos resultados, ou não. Observando isso, a análise de sensibilidade verifica quais parâmetros (entradas) interferem no resultado (saída) do sistema Hamby (1994).

Uma das formas mais simples de fazer análise de sensibilidade é mudar um parâmetro por vez enquanto mantêm os outros contantes. Assim, é possível comparar o resultado de acordo com as mudança no parâmetro. Além disso, técnicas de planejamento de experimento (*Design of Experiments* - DoE) podem ser usadas para determinar (simultaneamente) os efeitos individuais e interação entre vários parâmetros que podem afetar a saída Jain (1991). Na termologia DoE, os parâmetros são chamados de fatores e os valores de níveis. Para conduzir um experimento usando uma técnica DoE, é necessário definir os fatores e os seus níveis, criar e avaliar todas as combinações possíveis. O número de combinações (C) é calculado como segue Jain (1991):

$$C = \prod_{i=1}^{n} level(f_i) \tag{2.5}$$

onde n é o total de fatores e *level(*f_i*)* é uma função que retorna a quantidade de níveis

de um fator f_i. Dependendo do número de fatores e de níveis, a análise de sensibilidade pode requerer muito processamento (*e.g.*, mais de um computador) e tempo (*e.g.*, mais de um mês trabalhando exaustivamente), os quais podem inviabilizar o estudo. Uma alternativa é limitar o número de fatores e/ou de níveis.

Nós podemos fazer a análise de sensibilidade com apenas dois níveis e com k fatores. Esse tipo de experimento é chamado de 2^k *design* e o número de combinações é igual a 2^k. Mesmo assim, dependendo do número de fatores, o número de combinações pode ser grande. Após obter todos os resultados, nós podemos criar um *ranking* dos fatores e das interações entre eles que influenciaram mais o resultado. Para isso, é necessário calcular o impacto (S) de um fator (θ) em uma determinada saída (Y). O efeito do fator θ é a diferença entre a média do primeiro nível (θ_{-1}) e segundo nível (θ_{+1}), como mostra a seguir:

$$S_a(Y) = T_Y(\theta_{+1}) - T_Y(\theta_{-1}) \tag{2.6}$$

onde T é uma função que retorna a saída Y da média obtida com o primeiro nível (θ_{-1}) e o segundo nível (θ_{+1}) do fator θ. Além disso, nós devemos usar essa equação para calcular a interação entre dois fatores. Por exemplo, a interação entre os fatores A e B é calculado a seguir:

$$\begin{aligned} S_a(Y) = & \frac{T_Y(A_{+1}, B_{+1}) + T_Y(A_{-1}, B_{-1})}{2} \\ & - \frac{T_Y(A_{+1}, B_{-1}) + T_Y(A_{-1}, B_{+1})}{2} \end{aligned} \tag{2.7}$$

onde a função $T_Y(A_{+1}, B_{+1})$ retorna a média da saída Y quando usa o segundo nível dos fatores A e B; a função $T_Y(A_{-1}, B_{-1})$, quando usa o primeiro nível dos fatores A e B; e assim por diante.

2.5 Considerações finais

Este capítulo apresentou os conceitos fundamentais para o entendimento da proposta: RSSF (Rede de Sensores sem Fio), CPN (*Coloured Petri Net*), RBD (*Reliability Block Diagram*) e Análise de Sensibilidade. A RSSF (Rede de Sensores sem Fio) é uma rede *ad hoc* formada por pequenos equipamentos (chamados de nós sensores) com baixa capacidade computacional (processamento, armazenamento, comunicação), geralmente,

alimentados por baterias não recarregáveis. O CPN (*Coloured Petri Net*) e RBD (*Reliability Block Diagram*) são modelos formais utilizados para avaliar o consumo de energia e a confiabilidade, respectivamente, de qualquer sistema (*e.g.*, uma RSSF). Por fim, a análise de sensibilidade é usada para identificar quais fatores (*e.g.*, quantidade de nós sensores e protocolo de roteamento) estão afetando mais as saídas (*e.g.*, consumo de energia e confiabilidade) de um sistema (*e.g.*, uma RSSF).

3

Trabalhos Relacionados

Esta seção apresenta os trabalhos existentes que avaliam o consumo de energia e a confiabilidade de RSSFs. Os trabalhos relacionados foram divididos em quatro grandes grupos: avaliação do consumo de energia, avaliação de confiabilidade, avaliação de consumo de energia e confiabilidade juntos, e análise de sensibilidade. Dentro de cada um destes grupos, os trabalhos foram novamente agrupados de acordo com a técnicas de avaliação utilizada: medição, simulação e modelagem analítica.

3.1 Avaliação do Consumo de Energia

3.1.1 Medição

A medição é a técnica mais usada para avaliar o impacto do consumo de energia de um elemento (hardware ou software) do nó sensor. Por exemplo, Lee *et al.* (2007) avaliou o consumo de energia de diferentes transmissores sem fio de rádio. Através desse estudo, pode-se observar qual tecnologia possui o melhor desempenho e também o menor consumo de energia.

Com relação ao software, é possível citar três exemplos que usaram a medição. Hiltunen *et al.* (2012) avaliaram o desempenho e o consumo de energia do middleware POCOBOS através na plataforma Imote2. Neste caso, o objetivo foi mostrar o impacto do middleware no desempenho e no consumo de energia do nó sensor.

Lajara *et al.* (2010) também usaram a técnica de medição, mas o foco foi comparar o consumo de energia dos sistemas operacionais das RSSFs *e.g.*, TinyOS, Contiki, entre outros. Para efetuar a comparação, duas plataformas foram usadas, Tmote Sky e MICAz, com quatro aplicações diferentes representando atividades comuns, como por exemplo, coletar a temperatura e enviar um valor qualquer.

Chang *et al.* (2007) mediram o consumo de energia de diferentes algoritmos de criptografia na plataforma Mica2. Eles criaram uma aplicação que criptografa uma mensagem e a envia. O resultado do consumo de energia é igual ao consumo de energia do processador para executar o algoritmo mais o consumo de energia para transmitir a mensagem. Neste caso, algoritmos diferentes geram mensagens criptografadas com tamanhos diferentes.

Além de medir o consumo de energia do hardware ou de uma parte do software, existem estudos que usam a medição para auxiliar na simulação ou na modelagem analística. Basicamente, estes trabalhos usam a medição para obter o consumo de energia de cada componente da plataforma, *e.g.*, o processador pode estar ativo, inativo ou em modo econômico. Por exemplo, Prayati *et al.* (2010) ilustra como é feita a medição do estado de um componente da plataforma TelosB.

Em todos os casos, os trabalhos mencionados se utilizam de um ambiente de avaliação consistindo de uma aplicação e um sistema operacional; um hardware, *e.g.*, MicaZ, para instalar o código; uma fonte de tensão contínua, *e.g.*, Icel Manaus PS-1500, para manter a voltagem constante para o nó sensor; um osciloscópio para capturar os dados; e um programa que colete os dados do osciloscópio. Uma limitação inicial da medição é a configuração do ambiente que é composto por vários elementos. Adicionalmente, a RSSF é composta por milhares de nós sensores e, dificilmente, é possível avaliar individualmente o consumo de energia de cada nó sensor.

3.1.2 Simulação e Emulação

Atualmente, existem diversos softwares, emuladores e simuladores, utilizados para diferentes propósitos. Haase *et al.* (2011) divide os softwares em três grandes grupos: softwares que emulam hardware, softwares que emulam sistema operacional, e software que simulam rede. A emulação do hardware e do sistema operacional necessita do código fonte da aplicação para avaliar o consumo de energia. Por outro lado, a simulação já possui o comportamento dos protocolos modelados, e o usuário só precisa determinar quais modelos deseja utilizar para avaliar a sua RSSF.

A emulação do hardware executa a aplicação tal como se fosse na plataforma real, *e.g.*, MICAz. Uma das técnicas mais usadas é o *cycle-accurate*, ideal para representar arquiteturas antigas ou que necessitam de precisão na avaliação (Weaver and McKee, 2008). Para isso, é necessário ter conhecimento sobre a arquitetura desejada e seus componentes, instruções, ordem de execução, entre outras coisas. A vantagem de utilizar essa abordagem é avaliar o consumo de energia da aplicação independente do sistema

operacional ou linguagem utilizada. No entanto, a avaliação poderá demorar tanto quanto uma medição e poderá apresentar problemas caso emule vários nós sensores ao mesmo tempo.

O ATEMU (ATmel EMUlator) (Polley *et al.*, 2004)[1] foi o primeiro simulador de baixo nível para RSSF. A maioria dos simuladores de baixo nível foi desenvolvido para os processadores ou microcontroladores, tal como os emuladores SimulAVR (Rivet and Klepp, 2014), o qual emula o microcontrolador Atmel AVR; Embra (Witchel and Rosenblum, 1996), voltado para o processador MIPS R3000/R4000; e MSPsim, para o microcontrolador MSP430. ATEMU emula vários componentes da plataforma MICA2, permitindo avaliar vários nós ao mesmo tempo e suas respectiva interações. Além disso, o ATEMU permite executar diferentes aplicações na mesma simulação. A configuração da simulação é através de um arquivo XML e ATEMU possui uma interface gráfica para auxiliar o monitoramento da simulação. No entanto, o ATEMU apresenta sérios problemas de escalabilidade e o seu modelo de energia é ineficiente (Haase *et al.*, 2011).

O Avrora (Titzer *et al.*, 2005)[2] foi desenvolvido em Java e provê um modelo de *cycle-accurate* da família da plataforma MICA. No entanto, ele necessita de uma extensão para avaliar o consumo de energia, o AEON (Landsiedel *et al.*, 2004). O Avrora possui uma arquitetura extensível que permite suportar outras plataformas, tais como MICAz. O AvroraZ (de Paz Alberola and Pesch, 2008) é um dos exemplos de extensão e simula e avalia o consumo de energia da plataforma MicaZ. Seu desempenho é melhor do que o ATEMU. No entanto, ele ainda tem um desempenho pior do que os simuladores de sistema operacional (Haase *et al.*, 2011).

O SystemC (Liao *et al.*, 2002) é uma biblioteca da linguagem C++ e uma metodologia que pode ser usada para criar um modelo *cycle-accurate* de um hardware (Synopsys, 2002). Além disso, ele permite criar diversas ferramentas para o hardware modelado, como compiladores e simuladores. Diversas abordagens (Haase *et al.*, 2009; Madureira *et al.*, 2011; Du *et al.*, 2011; Minakov and Passerone, 2013) utilizam o SystemC para modelar uma plataforma da RSSF. No entanto, alguns projetos necessitam utilizar outras ferramentas para simular a rede, *e.g.*, NS-2 ou OMNeT++, pela falta de bibliotecas adequadas em SystemC. Quando usam uma biblioteca disponível, muitas vezes, os simuladores só avaliam a topologia do tipo estrela (Madureira *et al.*, 2011). Por outro lado, o PAWiS (*Power Aware Wireless Sensors*)[3] (Glaser *et al.*, 2008) é um exemplo de simulador baseado em SystemC que utiliza o OMNeT++ para avaliar a rede.

[1] o ATEMU está disponível em `http://www.hynet.umd.edu/research/atemu/`
[2] O Avrora está disponível em `http://avrora.sourceforge.net`
[3] o PAWiS está disponível em `http://pawis.sourceforge.net/`

Enquanto as estratégias anteriores estão relacionadas à uma plataforma específica, Somov *et al.* (2009) apresenta um *framework* para avaliar o consumo de energia de aplicações da RSSF através da simulação de diferentes plataformas. Esse *framework* descreve os possíveis estados de cada componente, memória e rádio, separadamente e cada estado tem o consumo de energia e desempenho associados. Um conjunto desses componentes representa uma plataforma específica. Essa mesma estratégia foi utiliza por e Silva Nogueira (2010) para avaliar o consumo de energia e o desempenho de sistemas embarcados.

A simulação do sistema operacional executa a aplicação de acordo com o comportamento do sistema operacional escolhido. Cada sistema operacional possui um simulador próprio. Por exemplo, TOSSIM (Levis *et al.*, 2003) e COOJA (Eriksson *et al.*, 2009) são simuladores dos sistemas operacionais TinyOS e Contiki (Dunkels *et al.*, 2004), respectivamente. A grande vantagem de utilizar essa abordagem é executar a aplicação independente da plataforma. Isto permite avaliar a mesma aplicação em várias plataformas utilizando o mesmo simulador. No entanto, a aplicação sempre estará vinculada ao mesmo sistema operacional e a mesma linguagem.

O TOSSIM (Levis *et al.*, 2003) é um dos simuladores mais utilizados por simular o comportamento do TinyOS, que é um dos sistemas operacionais mais populares. Ele ignora alguns aspectos do comportamento do hardware, por exemplo, não simula ações de interrupção, mas considera a velocidade do processador. O TOSSIM é construído a partir do código fonte da aplicação: o usuário, quando compila o código fonte da aplicação, cria um simulador exclusivo para ela. Essa característica não permite ao TOSSIM simular diversas aplicações ao mesmo tempo na mesma RSSF. Utilizando apenas o TOSSIM, não é possível avaliar o consumo de energia da RSSF. No entanto, ele possui uma arquitetura extensível, o qual permite acrescentar novos recursos. Por este motivo, várias extensões foram desenvolvidas para avaliar o consumo de energia em diferentes versões do TinyOS e plataformas: PowerTOSSIM (TinyOS 1.x) (Shnayder *et al.*, 2004), PowerTOSSIM2 (Prabhakar *et al.*, 2008), e o PowerTOSSIMz (TinyOS 2.x) (Perla *et al.*, 2008). Adicionalmente, o TOSSIM está relacionado com o MICA e as extensões PowerTOSSIM2 e PowerTOSSIMz, com o MICA2 e MICAz, respectivamente.

Li *et al.* (2009) propõem uma extensão do TOSSIM, chamado H-TOSSIM, para executar uma avaliação híbrida: simular uma RSSF com nós sensores reais e virtuais. O H-TOSSIM permite configurar quais são os nós sensores virtuais que irão se comunicar com os nós sensores reais, e vice-versa. A vantagem de utilizar essa abordagem é simular redes grandes, podendo capturar falhas que são difíceis de serem detectadas/capturadas

através de uma simulação puramente virtual. No entanto, essa abordagem necessita da medição para avaliar o consumo de energia nos nós sensores reais. Além disso, são necessários no mínimo três nós sensores reais: um para representar os nós sensores virtuais; outro para ser o nó sorvedouro; e um para fazer parte da rede que irá interagir com os sensores virtuais.

Muitas vezes, na simulação da rede, o comportamento da aplicação é ignorado, visto que o foco é o comportamento do nó sensor na RSSF. Ou seja, o interesse é avaliar a quantidade de mensagens enviadas e recebidas, o comportamento de um determinado protocolo com uma determina topologia, quantidade de nós sensores participantes, números de vizinhos, entre outras coisas. Geralmente, esses simuladores não necessitam do código da aplicação para simular o comportamento na rede. Em contrapartida, eles possuem os modelos pré-definidos dos protocolos e da rede, os quais combinados formam a RSSF do usuário.

O NS-2 (de Berkeley, 2011) é um dos mais populares simuladores de rede (Kurkowski *et al.*, 2005). Ele foi desenvolvido na linguagem C++ e é um simulador baseado em evento discreto. Os protocolos são modelados na linguagem C++ e as simulações são criadas e controladas através da linguagem de script chamada TCL (*Tool Command Language*) ou OTCL (*Object Tool Command Language*). Originalmente, ele foi desenvolvido para avaliar as redes tradicionais, onde o protocolo IP e o TCP são utilizados. Algumas abstrações são necessárias para avaliar ambientes sem fio, tal como o RSSF. Por este motivo, foram desenvolvidos simuladores exclusivos para a RSSF baseados no NS-2. O SensorSim (Park *et al.*, 2000) é baseado no NS-2 e provê os modelos de energia e da bateria. No entanto, o NS-2 possui um grande problema: a escalabilidade. Ele possui problemas para avaliar redes maiores do que 100 nós sensores (Naoumov and Gross, 2003).

O seu sucessor, chamado de NS-3 (Henderson *et al.*, 2006), possui várias melhorias de desempenho e escalabilidade. Para isso, ele teve duas principais modificações. Primeiro, abandonou a linguagem Tcl/OTcl, permitindo que o simulador possa ser implementado puramente em C++. Segundo, a sua arquitetura foi integrada com o simulador GTNetS (Riley, 2003), o qual possui boas características de escalabilidade. Weingartner *et al.* (2009) mostrou que o NS-3 possui um desempenho melhor do que outros simuladores, *e.g.*, NS-2, OMNeT++, JiST (Barr *et al.*, 2005) e SimPy (Mueller, 2004).

O OMNeT++ (Varga, 2001) é um *framework* modular de propósito geral escrito em C++ para simular redes através de eventos discretos. Uma extensão chamada Mobility Framework (Drytkiewicz *et al.*, 2003) foi desenvolvida para suportar especificamente as

Tabela 3.1 Resumo dos trabalhos relacionados que avaliam o consumo de energia da RSSF.

Método	Trabalho	Usou	Descrição
Medição	Lee (2007)	Nó sensor real	Ele avaliou o consumo de energia de diferentes transmissores sem fio de rádio.
	Hiltunen *et al.* (2012)	Imote2	Eles avaliaram o desempenho e o consumo de energia do middleware POCOBOS.
	Lajara *et al.* (2010)	Tmote/MICAz	O seu foco foi comparar o consumo de energia dos sistemas operacionais da RSSF.
	Chang *et al.* (2007)	Mica2	Eles mediram o consumo de energia de diferentes algoritmos de criptografia.
	Prayati (2010)	TelosB	Ele ilustra como é feita a medição do estado de um componente da plataforma TelosB.
Simulação	Polley *et al.* (2004) Titzer *et al.* (2005)	Emulação de Hardware	Uma das técnicas mais usadas é o *cycle-accurate*, ideal para representar arquiteturas antigas ou que necessitam de precisão na avaliação. No entanto, eles apresentam sérios problemas de escalabilidade.
	Haase *et al.* (2009) Madureira *et al.* (2011) Du *et al.* (2011) Minakov *et al.* (2013)	SystemC	Muitas vezes, os simuladores só avaliam um tipo de topologia, a estrela (Madureira *et al.*, 2011). Exceto o PAWiS (Pases *et al.,* 2013) que utiliza o OmNET++ para avaliar a rede e aceita várias topologias.
	Somov *et al.* (2009) Bruno *et al.* (2010)	Emulação de Hardware	Somov *et al.* (2009) apresenta um framework para avaliar o consumo de energia de aplicações da RSSF através da simulação para diferentes plataformas. Enquanto Bruno *et al.* (2010) utilizou essa estrategia para avaliar o consumo de energia e o desempenho de sistemas embarcados.
	Levis *et al.* (2003)	Simulador do TinyOS	O TOSSIM é um dos simuladores mais utilizados por simular o comportamento do TinyOS,que é um dos sistemas operacionais mais populares. Existem várias extensões para avaliar o consumo de energia na RSSF, tal como PowerTOSSIM (Shnayder *et al.* ,2004), PowerTOSSIM2 (Prabhakar *et al.*, 2008) e PowerTOSSIMz (Perla *et al.*, 2008).
	Eriksson *et al.* (2009)	Simulador do Contiki	O COOJA permte executar a aplicação independente da plataforma, mas a aplicação deve estar vinculada ao sistema operacional Contiki.
	Htossim *et al.* (2009)	Simulador do TinyOS + rede externa	Eles propõem uma extensão do TOSSIM (chamado H-TOSSIM) para executar uma avaliação híbrida: é possível simular uma RSSF com nós sensores reais e virtuais.
	Park *et al.* (2000)	NS-2	O SensorSim é baseado no NS-2 e provê os modelos de energia e da bateria. No entanto, o NS-2 possui um grande problema: a escalabilidade. Ele possui problemas para avaliar redes maiores do que 100 nós sensores (Naoumov *et al.*, 2003).
	Weber *et al.* (2007) Boulis *et al.* (2007) nesct	OMNeT++	Eles propuseram simuladores baseados no OMNeT++ especificos para a RSSF, provendo biblioteca de módulos e suporta mobilidade na rede, ambiente dinâmicos e protocolos desta rede.
Modelagem Análitica	shinghal *et al.* (2011)	Expressão Matemática	Eles estimam o tempo de vida da plataforma desenvolvida por eles.
	Rusli *et al.* (2010) Manjeshwar *et al.* (2002) Sahota *et al.* (2011) Cano *et al.* (2009)	Cadeia de Markov M/G/1 model	Essas abordagens focam na avaliação da rede, mais especificamente, protocolos da RSSF.

redes ad hocs como as RSSF. Alguns simuladores, PAWiS, Castalia e SenSim, e extensões, NesCT, foram desenvolvidos com base no OMNeT++. PAWiS (Weber *et al.*, 2007) provê uma biblioteca de módulos e suporta mobilidade na rede e ambiente dinâmicos. Castalia (Boulis, 2007) tem como ponto forte a precisão do canal sem fio e a modelagem do rádio, incluindo a camada MAC. SenSim (C. Mallanda and Sastry., 2005) adiciona módulos de comunicação, protocolos, energia, entre outros, para representar a RSSF. Ele ainda oferece *templates* que servem como base para futuros módulos. Ele prover um modelo básico de implementação, o qual auxilia no começo de uma nova simulação (Egea-Lopez *et al.*, 2006). Por fim, NesCT (Featherlight, 2011) é uma extensão do OMNeT++ para suportar os nós sensores com o TinyOS e atua como um tradutor entre a linguagem nesC e as classes C++ do OMNeT++.

3.1.3 Modelagem Analítica

Atualmente, as abordagens que usam a modelagem analítica focam na avaliação da rede e não na aplicação: Shinghal *et al.* (2011) criou uma plataforma um modelo analítico para estimam o tempo de vida de uma aplicação executada na sua plataforma; Rusli *et al.* (2010) avaliam o desempenho do protocolo OR (*Opportunity Routing*) (Biswas and Morris, 2004); Manjeshwar *et al.* (2002) modelam, avaliam e comparam o consumo de energia dos protocolos LEACH (Heinzelman, 2000; Heinzelman *et al.*, 2000) e APTEEN (Manjeshwar and Agrawal, 2002); Sahota *et al.* (2011) propuseram e avaliaram um novo protocolo MAC chamado DS-MAC (Lin *et al.*, 2004); e Cano *et al.* (2009) verificou o benefício do LPL (*Low Power Listening*) (Hill and Culler, 2002) dos protocolos MACs.

A Tabela 3.1 sumariza os trabalhos relacionados que avaliam o consumo de energia das RSSFs. Esta tese avalia o consumo de energia e a confiabilidade utilizando o código fonte da aplicação e a configuração de rede. O modelo que avalia o consumo de energia da aplicação está vinculado ao sistema operacional TinyOS. Diferente dos simuladores de rede existentes que exige um conhecimento para usá-los, os modelos propostos são criados e avaliados automaticamente baseados em uma configuração da rede auxiliada por uma ferramenta Web. Além disso, esta tese propõe avaliar a confiabilidade da RSSF usando dados do consumo de energia.

3.2 Confiabilidade

Tal como no consumo de energia, é possível encontrar trabalhos que avaliam a confiabilidade das RSSFs usando as três técnicas de avaliação: medição, simulação e modelagem

analítica.

3.2.1 Medição

No caso da confiabilidade, a medição não precisa usar o osciloscópio para analisar o número de falhas ocorridas no nó sensor real. Existem duas possibilidades de estudo: inserir artificialmente ou identificar naturalmente uma falha.

A Injeção de Falha simula a falha em um nó sensor sem comprometer o funcionamento do sistema. Essa falha é inserida por um software, o qual decide quando deve ocorrer a falha. Em geral, é utilizado em redes para simular falhas dos tipos descarte ou atraso de pacotes. Outra maneira é simular falhas do hardware. O AVR-Injection (Cinque *et al.*, 2009) é um exemplo de ferramenta que insere falhas de diversos tipos. Ele recebe uma aplicação e insere algumas instruções em assembly. No entanto, a avaliação através da Injeção de Falha pode ser tardia, pois precisa de uma aplicação codificada para executar em um nó sensor real. O ideal é avaliar nas fases iniciais do projeto a aplicação ou a RSSF.

A outra alternativa é monitorar a RSSF nas condições reais sem induzir ou forçar a falhar. Uma das técnicas utilizadas nesse contexto é o FFDA (*Field Failure Data Analysis*). Ela consiste em observar arquivos de *log* para detectar as falhas. Ou seja, é necessário que os participantes da RSSF registrem periodicamente o funcionamento do sistema. Por exemplo, Parameswaran *et al.* (2009) utilizaram essa técnica para saber se o RSSI (*Received Signal Strength Indication*) informa a distância correta entre dois nós sensores. O experimento consistiu em comparar a distância real entre dois nós sensores com o valor RSSI obtido em *n* mensagens transmitidas. Através desse experimento, foi observado que o RSSI não é um ótima solução para servir como base para algoritmos de localização. Além disso, eles mostraram que o RSSI é muito irregular quando varia a distância dos nós sensores. No entanto, o FFDA não é viável para a RSSF por não prover nenhum tipo *log*) e os nós sensores devem usar aplicações otimizadas, evitando utilizar APIs desnecessárias para a sua atividade (Carrozza, 2008).

3.2.2 Simulação

Em vez de utilizar o nó sensor real, existe a possibilidade de avaliar algumas falhas através da simulação. Além de avaliar o comportamento do nó sensor e da rede, alguns simuladores acrescentaram modelos para simular as interferências e a propagação do sinal, permitindo trabalhar com uma quantidade maior de nós sensores presentes na

mesma RSSF. Existem modelos de propagação simples que não consideram perdas de pacotes causados pelo ambiente, Espaço Livre (Rappaport *et al.*, 1996), e modelos mais específicos para a RSSF, como é o caso do RIM (*Radio Irregularity Model*) (Zhou *et al.*, 2006).

Esses modelos adicionados aos simuladores permitem avaliar melhor as situações de perda, colisão ou erro de pacote na RSSF. Por exemplo, Wang *et al.* (2006) propuseram um protocolo de controle de congestionamento chamado PCCP (*Priority-based Congestion Control Protocol*) para a RSSF. Através da simulação, o protocolo PCCP mostrou que evita ou reduz perda de pacote e melhora o consumo de energia. Desta maneira, o protocolo PCCP aumenta a disponibilidade dos nós sensores na RSSF.

3.2.3 Modelagem Analítica

Com relação à modelagem analítica, é possível modelar a falha de uma RSSF de várias formas. Essas formas são divididas em dois grupos (Maciel *et al.*, 2011): modelos combinatórios e modelos baseados em espaço de estados. Os modelos combinatórios consideram que o sistema está em falha ou operacional considerando as relações estruturais entre os componentes do sistema. Enquanto os modelos baseados em espaço de estados consideram os estados dos componentes do sistema e os eventos que provocam mudanças de estados. O primeiro grupo é mais simples, no entanto, ele considera apenas falhas persistentes, ou seja, o componente para de funcionar. As falhas transientes não são modeladas. O segundo grupo permite representar cenários mais complexos e, por este motivo, esse grupo é mais difícil de ser criado. Dependendo da quantidade de estados, a avaliação pode ser inviável.

Para os modelos combinatórios, as formas mais utilizadas para avaliar a RSSF são BDD (*Binary Decision Diagram*), Árvore de Falhas e RBD. BBD é um grafo acíclico com dois tipos de vértices, quadrado e círculo. Para avaliar a confiabilidade da RSSF Wang *et al.* (2012), os vértices quadrados representam os dois estados do nó sorvedouro: 0, quando falha, e 1, quando está funcionando. Os nós sensores são representados pelos vértices círculos. Desses vértices saem duas arestas rotuladas, uma com valor 0 e outra com valor 1 representando que o componente falhou ou está funcionando, respectivamente. Um BDD pode ser otimizado caso seja reordenado e reduzido, gerando uma ROBDD (*Reduced Ordered Binary Decision Diagram*). Para avaliar a confiabilidade da BDD ou da ORBDD, são utilizados os operadores booleanos *AND* e *OR* introduzidos por Zang *et al.* (1999).

Wang *et al.* (2012) utilizou o BDD para avaliar a confiabilidade da RSSF. Eles criaram

uma pequena rede com cinco nós sensores, os quais enviavam as mensagem em *broadcast*, e avaliaram diversas características da topologia, tais como conectividade, diâmetro da rede, coeficiente médio de *clustering*. Eles criaram vinte e um cenários e constataram, por exemplo, que quanto maior a conectividade da rede, melhor será a sua confiabilidade. No entanto, eles não apresentaram uma metodologia ou ferramenta para avaliar RSSF maiores. Dependendo do tamanho da rede e se a definição da topologia for totalmente manual, pode ser inviável avaliar a confiabilidade da rede.

Shrestha *et al.* (2006) apresentaram uma metodologia, baseada no ROBDD, para avaliar a confiabilidade de *clusters* em RSSFs. Eles avaliaram apenas a topologia hierárquica, com *clusters* estáticos, considerando como ponto de falha o nó sensor, o enlace de comunicação e algumas causas de falhas, tais como como terremoto e bombas. Eles compararam a confiabilidade dos *clusters* com e sem as falhas. Além disso, eles avaliaram um cenário com reposição de nós sensores, demostrando que melhora a confiabilidade dos *clusters*. Posteriormente, Shrestha *et al.* (2007) estenderam o seu trabalho anterior para avaliar a confiabilidade orientada à cobertura dos *clusters* da RSSF, onde considera-se que existem K nós sensores de N analisando o mesmo fenômeno.

Outra forma utilizada para avaliar a confiabilidade das RSSFs é a Árvore de Falhas, a qual modela as possíveis falhas que podem acontecer na RSSF (Ericson, 2005). Silva *et al.* (2012) criaram uma metodologia para avaliar a confiabilidade e a disponibilidade de um conjunto de nós sensores presentes em uma RSSF. O modelo proposto permite avaliar redes lineares, estrela e *cluster* através da Árvore de Falhas. No entanto, ele não considera o protocolo para montar as rotas entre o conjunto de nós sensores e o *gateway*. A construção das rotas entre o nó sensor até o *gateway* é realizado através de uma busca em profundidade no grafo que mapeia a topologia da rede. Ou seja, ele não considera que a mesma RSSF pode ter confiabilidade diferente usando protocolos diferentes.

O RBD (veja a Seção 2.3) permite modelar os componentes que podem falhar no sistema e cada componente possui uma probabilidade de estar funcionando. Por exemplo, Purohit *et al.* (2008) mostraram todos os elementos do nó sensor envolvidos na confiabilidade da RSSF.

Ghaffari and Rahmani (2008) usaram RBD para avaliar a confiabilidade de dois protocolos da camada de transporte, ESRT (*Event-to-Sink Reliable Transport*) (Sankarasubramaniam *et al.*, 2003) e RMST (*Reliable Multi-Segment Transport*) (Stann and Heidemann, 2003), enviando os pacotes em *broadcast*. Eles consideraram que as rotas possuem o mesmo nível de confiabilidade e dividiram as falhas em toleráveis, falhas que podem ser tratada pela camada de transporte como a perda de pacotes, e intoleráveis,

aquelas decorrentes de fatores externos que são imprevisíveis como o nó sensor parar de funcionar. O protocolo RMST, por retransmitir os pacotes, possui uma confiabilidade maior do que o ESRT.

Em outro estudo, Shaikh *et al.* (2007) avaliaram a confiabilidade do protocolo RMST e do protocolo RBC (*Reliable Bursty Convergecast*) (Zhang *et al.*, 2005), ambos fazem retransmissão de pacotes. As confiabilidades dos dois protocolos foram próximas, mas o protocolo RBC precisa de menos nós sensores para obter uma confiabilidade próxima de 1, enquanto o protocolo RMST precisou de 6 nós sensores. De acordo com os autores, isto indica que o protocolo RBC consumiu menos energia do que o protocolo RMST por usar menos nós sensores para obter uma boa confiabilidade.

Com relação ao modelos baseados em espaço de estados, a Cadeia de Markov permite modelar os estados do sistema e a frequência de mudanças de um estado para o outro. Bein *et al.* (2005) avaliaram o custo e a disponibilidade de uma RSSF considerando que os nós podem ou não serem substituídos. Eles utilizaram Cadeia de Markov para modelar todos os estados possíveis de uma RSSF com 4, 5 e 6 nós sensores. Como esperado, a RSSF que permite substituir os nó sensores possui um resultado melhor do que a rede que não permite. No entanto, tradicionalmente, as RSSFs não permitem o reparo ou a substituição dos nós sensores por causa da quantidade de nós, e implantação em locais difícil acesso.

CTMC (*Continuos Time Markov Chain*) é uma variante da Cadeia de Markov que acrescenta a escala de tempo contínua. Esta variante foi utilizada por Bruneo *et al.* (2010) para avaliar a confiabilidade e a produtibilidade (verifica a capacidade do nó sensor de realizar a sua atividade) da RSSF. Mais especificamente, eles analisaram o impacto da redundância de nós sensores e da topologia da rede na confiabilidade e na produtibilidade da RSSF. É importante ressaltar que esse trabalho considerou a estratégia *duty-cycle*, onde ora o rádio do nó sensor está ativo, ora está desativado para economizar energia. O trabalho considerou ainda o nível de energia do nó sensor como critério de falha e determinou que o consumo de energia diminui continuamente dependendo do estado do *duty-cycle*. Sendo assim, o nó sensor falha completamente quando estiver sem energia ou temporariamente quando o rádio estiver desligado. Eles avaliaram três tipos de topologia: estrela, onde todos os nós possuem comunicação direta com o nó sorvedouro; hierárquica, onde foram criado vários *clusters*; e mesh, quando os nós sensores auxiliam uns aos outros para rotear os pacotes. A topologia estrela teve os melhores resultados, tanto confiabilidade quanto produtibilidade, porque os nós sensores possuem comunicação direta com o nó sorvedouro. Enquanto isto, a rede hierárquica teve o pior resultado porque

Tabela 3.2 Resumo dos trabalhos relacionados que avaliam a confiabilidade da RSSF.

Método	Trabalho	Usou	Descrição
Medição	Cinque et al. (2009)	Injeção de Falha	Ele insere falhas de diversos tipos em uma aplicação da RSSF.
	Parameswaran *et al.* (2009)	FFDA	Os participantes da RSSF registrem periodicamente o funcionamento do sistema para detectar uma falha.
Simulação	Wang *et al.* (2006)	NS-2	Eles propuseram um protocolo de controle de congestionamento chamado PCCP para a RSSF para avaliar a confiabilidade da RSSF.
Modelagem Análitica	Vokkarane *et al.* (2012)	BDD	Eles criaram uma pequena rede com cinco nós sensores, os quais enviaram as mensagem em *broadcast*, e avaliou diversas características (tal como conectividade, diâmetro da rede, coeficiente médio de *clustering*) da RSSF.
	Shrestha *et al.* (2006)	ROBDD	Eles apresentaram uma metodologia para avaliar a confiabilidade dos *clusters* da RSSF. Eles avaliaram apenas a topologia hierárquica (mais especificamente, *clusters* estáticos) considerando como ponto de falha o nó sensor, o enlace de comunicação e as causas comuns de falhas (tal como terremoto e bombas).
	Shrestha *et al.* (2007)	ROBDD	Eles estenderam a proposta anterior para avaliar a confiabilidade orientada a cobertura (considera que existem K nós sensores de N analisando o mesmo fenômeno) dos *clusters* da RSSF.
	Silva *et al.* (2012)	Árvore de Falhas	Eles criaram uma metodologia para avaliar a confiabilidade e a disponibilidade de um conjunto de nós sensores (o que pode ser toda a rede ou parte dela) presentes em uma RSSF. Mas não consideram o protocolo de roteamento para definir os caminhos entre os nós sensores com o nó sorvedouro e não considerou o nível da bateria como fator de falha.
	bein *et al.* (2005)	Cadeia de Markov	Eles avaliaram o custo e disponibilidade de uma RSSF considerando que os nós podem ser (ou não) substituídos. No entanto, tradicionalmente, as RSSFs não permitem o reparo ou a substituição dos nós sensores por causa do ambiente que pode ser hostil ou de difícil acesso.
	Bruneo *et al.* (2010)	CTMC	Eles analisaram o impacto da quantidade de redundância de nós sensores e da topologia da rede na confiabilidade e na produtibilidade da RSSF. Este trabalho considerou a estratégia *duty cycle*, considerou o nível da bateria do nó sensor como critério de falha e determinou que o consumo de energia diminui continuamente dependendo do estado do *duty cycle*.
	Bruneo *et al.* (2011)	NMSPN	
	Purohit *et al.* (2008)	RBD	Eles mostraram os componentes que afetam a confiabilidade do nó sensor.
	Ghaffari *et al.* (2008)	RBD	Eles avaliaram a confiabilidade de dois protocolos da camada de transporte (ESRT e RMST); um com e outro sem retransmissão de pacotes. Eles não consideraram o consumo de energia como fator que afeta a confiabilidade.
	Shaikh *et al.* (2007)	RBD	Eles avaliaram a confiabilidade de dois protocolos da camada de transporte (ESRT e RBC). Novamente, não foi considerado o consumo de energia como fator que afeta a confiabilidade.

vários nós sensores precisam do *cluster head* para se comunicar com o nó sorvedouro, o que o torna um gargalo. No entanto, eles não trabalharam com nenhum protocolo para definir os caminhos entre os nós sensores e o nó sorvedouro. Adicionalmente, eles consideram como confiabilidade a probabilidade de pelo menos k nós sensores estarem funcionando ao tempo t; e como produtibilidade a probabilidade de pelo menos k nós sensores estarem com energia e com a antena ativa ao tempo t, ou seja, serem capazes de se comunicar com o nó sorvedouro.

Rede de Petri (veja a Seção 2.2) também pode ser utilizada para avaliar a confiabilidade da RSSF, representando os estados e as ações através de lugares e transições, respectivamente. Por exemplo, (Bruneo *et al.*, 2011) utilizaram uma variante, denominada NMSPN (*non-Markovian stochastic Petri nets*), e Árvore de Falhas para dar continuidade ao trabalho anterior (Bruneo *et al.*, 2010). A NMSPN permite políticas de preempção e inclui transições cuja função de distribuição não necessite ser exponencial. Mesmo usando outro tipo de modelo, eles mostraram que obtiveram o mesmo resultado usando NMSPN e CTMC.

A Tabela 3.2 mostra o resumo dos trabalhos que tratam da análise de confiabilidade em RSSF. Nenhum dos trabalhos anteriores trata a confiabilidade da RSSF tal como o Modelo de Região apresentado na Seção 5.4. A sequência de passos para obter o Modelo de Região é praticamente a mesma definida por Silva *et al.* (2012). No entanto, esta tese considera que o protocolo de roteamento é fundamental para a definição dos caminhos entre os nós sensores e o nó sorvedouro em vez de utilizar matriz de adjacência. O Modelo de Região também considera o nível de energia como fator que afeta a confiabilidade, tal como o Bruneo *et al.* (2010). Enquanto Bruneo *et al.* (2010) definem um consumo de energia constante, o Modelo de Região define que a confiabilidade da bateria é calculada de acordo com o seu nível, o qual é obtido após simular o comportamento da RSSF.

3.3 Consumo de Energia e Confiabilidade

Poucos trabalhos avaliam o consumo de energia e a confiabilidade de RSSFs simultaneamente. Não foi encontrado nenhum trabalho que usasse a modelagem analítica para avaliar o consumo de energia e a confiabilidade da RSSF. A Tabela 3.3 mostra os trabalhos relacionados que avaliaram o consumo de energia e a confiabilidade da RSSF.

A confiabilidade da transmissão do dado é usualmente definida em termos do BER (*Bit Error Rate*), perda de pacotes e atraso na transmissão do pacote (Balouchestani *et al.*, 2012; Kwon *et al.*, 2005). Neste sentido, simuladores tal como o PowerTOSSIM

Tabela 3.3 Resumo dos trabalhos relacionados que avaliam o consumo de energia e a confiabilidade da RSSF.

Método	Trabalho	Usou	Descrição
Medição	Horvat *et al.* (2012)	Nós sensores reais	Eles analisaram o consumo de energia, a confiabilidade e a propagação do sinal (sensibilidade do receptor, força do sinal recebido) da RSSF formada por módulos XBee ZigBee usando *Adaptive Transmission Power Control* para reduzir o consumo de energia.
Simulação	Balouchestani *et al.* (2012)	C++ e MATLAB	Eles avaliaram o consumo de energia e a confiabilidade da WBAN (variante da RSSF) quando reduzem a amostra de sinais eletrocardiacos (atividades eletricas do coração) com Compressed Sensing Theory (Balouchestani *et al.*, 2011), considerando que a confiabilidade é a probabilidade de transmitir o dado com sucesso.
	Kwon *et al.* (2005)	NS-2	Eles avaliaram o consumo de energia dos protocolos *cross layer*, considerando a restrições da confiabilidade; mais especificamente, restrições de transmitir um pacote fim-a-fim.
	Zonouz *et al.* (2014)	Grafos	Eles avaliaram o consumo de energia e a confiabilidade da RSSF ao mesmo tempo quando simulavam o gráfo, o qual representava a RSSF. Eles consideram o nível da bateria como fator da confiabilidade do enlace de comunicação.

e NS-2, os quais são usados para avaliar o consumo de energia (Balouchestani *et al.*, 2012; Kwon *et al.*, 2005), podem também ser usados para avaliar a confiabilidade da RSSF. Balouchestani *et al.* (2012) avaliaram, usando C++ e MATLAB, o consumo de energia e a confiabilidade da WBAN (*Wireless Body Area Network*) quando há uma redução da atividades cardíaca usando *Compressed Sensing Theory* (Balouchestani *et al.*, 2011). Kwon *et al.* (2005) avaliaram o consumo de energia dos protocolos *cross layer* considerando restrições de confiabilidade, mais especificamente, restrições da transmissão fim-a-fim.

Comum a essas abordagens é o fato de que o enlace de comunicação é considerado o único ponto de falha na RSSF, enquanto consideramos a falha da aplicação, bateria, sistema operacional, middleware, entre outros. Além disso, nós avaliamos a confiabilidade da RSSF usando modelos RBD depois de avaliar o consumo de energia usando modelos CPN. Por fim, consideramos que a confiabilidade dos nós sensores está também relacionada ao nível de energia da bateria.

Similarmente, existem estudos que avaliam o consumo de energia e a confiabilidade da RSSFs usando nó sensores reais. Horvat *et al.* (2012) analisaram o consumo de energia, confiabilidade e propagação do sinal de RSSFs formadas por módulos XBee ZigBee usando *Adaptive Transmission Power Control* para reduzir o consumo de energia.

Uma das limitações iniciais da medição é a configuração do ambiente que é composta por diversos elementos. Por esta razão, é comum avaliar o consumo de energia de um nó sensor e deduzir o consumo de energia para toda a rede visto que os nós sensores possuem a mesma configuração, *e.g.*, aplicação, pilha de protocolo, plataforma. No entanto, algumas configurações são únicas do nó sensor, tais como a quantidade de vizinhos, o raio de comunicação e o caminho até o nó sorvedouro. Tais características afetam o consumo de energia e a confiabilidade, e são frequentemente ignoradas. Além disso, cenários com milhares de nós sensores demandam uma grande quantidade de equipamentos e um longo tempo.

A estratégia criada por Zonouz *et al.* (2014) para avaliar o consumo de energia e a confiabilidade da RSSF é parecida com a definida nesta tese. Eles usaram grafos com expressões matemáticas, *First Order Radio Model*, para representar e avaliar o consumo de energia e a confiabilidade de RSSFs. Eles consideram o nível de energia como fator de confiabilidade para o enlace de comunicação, enquanto consideramos para o nó sensor. Além disso, eles avaliaram o consumo de energia e a confiabilidade da RSSF ao mesmo tempo quando simulavam o grafo. Por outro lado, fazemos a avaliação sequencialmente: primeiro, o consumo de energia usando modelos CPN e, então, a confiabilidade usando RBD. Além disso, nós avaliamos o consumo de energia da aplicação baseado no código fonte da aplicação, enquanto eles definiram um consumo de energia constante. Diferente de Zonouz, todo o processo de avaliação é guiado por uma metodologia e suportado por um conjunto de ferramentas.

3.4 Análise de Sensibilidade

Alguns estudos usam a análise de sensibilidade para avaliar quais os fatores da RSSF influenciam os algoritmos de localização (Sahota, 2013; Ma and Jin, 2014), utilizado para identificar as posição dos nós sensores; ou o protocolo de roteamento (Xu, 2002).

Sahota (2013) estudou projetos de RSSF para agricultura. Ele se concentrou nas camadas de enlace e física e mostrou a importância do algoritmo de localização para esse tipo de aplicação. Em particular, ele usou a análise de sensibilidade para determinar o nível de erro (diferença entre o valor obtido e o valor real) de acordo com os parâmetros usados pelos modelos de localização. Ele mostrou que a localização baseada no tempo de chegar do sinal (quanto menor o tempo, mais próximo os nós sensores estavam) é mais precisa do que a baseada pelo força do sinal (quanto maior a força do sinal, mais próximo os nós sensores estavam).

Similarmente, Ma and Jin (2014) avaliaram os fatores que influenciam a localização em RSSFs para ambientes aquáticos chamada UWSAN (*Underwater Wireless Sensor Array Networks*). Esse tipo de rede é formada por vários nós sensores, os quais são equipados com diversos sensores para coletar e processar sinais eletromagnéticos e acústicos. Nesse tipo de rede, os sinais acústicos são utilizados pelos algoritmos de localização para determinar a posição dos nós sensores. Observando isso, eles buscaram a relação entre o DoA (*Directional of Arrival*), que é o angulo de recepção do sinal, para a localização baseada no método *Least-Square* (Whittle, 1963). Entre vários achados, eles encontraram que o erro de localização aumenta proporcionalmente à distância entre os nós sensores quando o número de nós sensores e o DoA são mantidos.

Diferente dos trabalhos anteriores, o nosso trabalho concentra-se em definir o impacto de cada fator no consumo de energia e na confiabilidade da RSSF, enquanto eles só mostraram a influência no algoritmo de localização.

Xu (2002) propôs um protocolo de conservação de energia para estender o tempo de vida das redes sem fio. Como parte do estudo, ele usou a análise de sensibilidade, através da simulação, para avaliar quais fatores afetam mais os protocolos de conservação de energia (AODV, BECA, AFECA, GAF e CEC) e, consequentemente, o tempo de vida da rede. EM particular, os fatores considerados foram o modelo de mobilidade, trafego de dado, modelo de propagação do rádio, modelo de energia, modelo de localização e densidade da rede. Ele concluiu que o modelo de propagação do rádio e o modelo de mobilidade afetam mais do que os outros fatores.

Nosso trabalho está próximo ao Xu (2002): ambos fizeram a análise de sensibilidade da RSSF e mostraram o impacto dos parâmetros no seu consumo de energia. Entretanto, eles são diferentes em vários aspectos. Por exemplo, ele avaliou o consumo de energia da RSSF usando o NS-2 e nós usamos modelos formais, os quais foram propostos neste trabalho. Ele considerou que o nó sensor pode se mover na rede e usou o modelo de programação do rádio como parâmetros; enquanto nós consideramos que a rede é estática e utilizamos apenas um único modelo de propagação de rádio. Por outro lado, nós consideramos o protocolo de roteamento como parâmetro e avaliamos o impacto de cada fator no consumo de energia e na confiabilidade da RSSF. Além disso, nós propomos uma metodologia para desenvolver e avaliar a RSSF que é totalmente suportada por um ambiente de desenvolvimento e avaliação.

Os trabalhos relacionados que realizaram a análise de sensibilidade de RSSFs são sumarizados na Tabela 3.4. A análise de sensibilidade feita nesta tese é similar à apresentada por Xu (2002).

Tabela 3.4 Resumo dos trabalhos relacionados que fizeram análise de sensibilidade da RSSF ou de uma variante.

Trabalho	Avaliou	Descrição
Sahota (2013)	algoritmos de localização	Ele usou a análise de sensibilidade determinar o nível de erro (saída) de acordo com os parâmetros usados pelos modelos de localização (entrada).
Ma *et al.* (2014)	algoritmos de localização	Eles avaliaram os fatores que influenciam a localização em uma variante da RSSF para ambientes aquáticos (*Underwater Wireless Sensor Array Networks* – UWSAN).
Xu (2002)	Protocolo de roteamento	Ele usou a análise de sensibilidade para avaliar quais fatores afetam mais os protocolos de conservação de energia e o tempo de vida da rede.

3.5 Considerações Finais

Este capítulo apresentou três métodos para avaliar o consumo de energia ou a confiabilidade: medição / experimento, emulação / simulação e modelagem analítica. A medição tende a ser mais precisa, porque está avaliando diretamente no equipamento; enquanto a modelagem analítica tende a ser a mais rápida, porque os modelos são baseados em expressões matemáticas. Os trabalhos relacionados foram divididos em quatro grupos: os que avaliam apenas o consumo de energia, os que avaliam apenas a confiabilidade, os que avaliam o consumo de energia e a confiabilidade, e os que fizeram a análise de sensibilidade. A Figura 3.1 mostra a concentração dos estudos em cada grupo apresentado. A emulação/simulação é o método mais utilizado em três grupos, os quais envolveram o consumo de energia. Diversos fatores (*e.g.*, comportamento da aplicação, taxa de envio de dados, quantidade de nós sensores, estratégia de implantação da RSSF, entre outros) devem ser considerados para analisar e obter o consumo de energia da RSSF. A medição e modelagem analítica são mais utilizadas para avaliar fatores pontuais da RSSF, tal como o consumo de energia de algoritmos de criptografias ou um determinado protocolo. A modelagem analítica é mais utilizada apenas para avaliar a confiabilidade da RSSF visto que existem diversas alternativas.

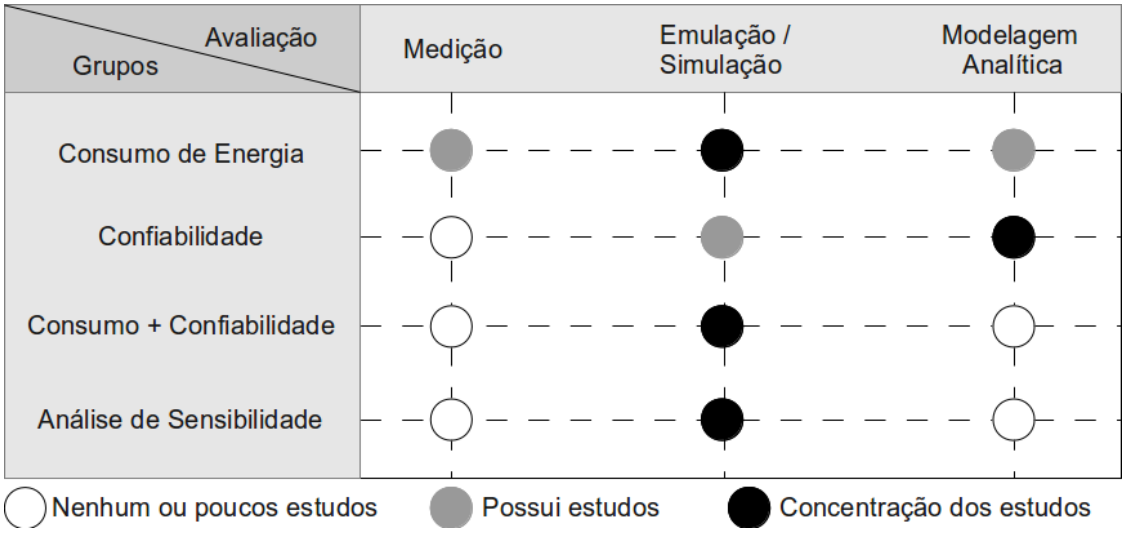


Figura 3.1 Resumo dos trabalhos relacionados.

4

Metodologia

Este capítulo apresenta a metodologia criada para desenvolver e avaliar o consumo de energia e a confiabilidade da RSSF considerando a aplicação e os elementos da rede. Apresentamos inicialmente a notação utilizada para descrever a metodologia e em seguida todas as etapas da metodologia são descritas detalhadamente.

4.1 Elementos Básicos

Antes de apresentar a metodologia, é preciso introduzir os elementos básicos da notação utilizada para descrever a metodologia. Como mostrado na Figura 4.1, a notação possui seis elementos que representam atividades manuais ou automáticas, repositórios, decisões, conectores e artefatos.

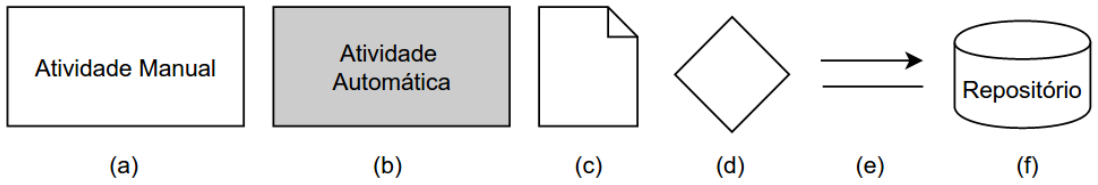


Figura 4.1 Elementos básicos utilizados para descrever a metodologia: (a) atividade manual, (b) atividade automática, (c) artefato, (d) decisão, (e) conector e (f) repositório.

A metodologia considera uma atividade manual quando executada exclusivamente pelo usuário. Uma atividade automática é executada por uma ferramenta de software sem nenhuma ou com pouca intervenção do usuário. O repositório armazena os pequenos modelos reusáveis utilizados para compor os modelos de energia. A decisão permite que o usuário ou o conjunto de ferramentas selecione um fluxo de execução entre os vários fluxos de execução baseado em uma condição. O conector é utilizado para conectar

atividades, decisões e repositórios. Por fim, os artefatos são gerados por uma atividade manual ou automática.

4.2 Visão Geral

Uma RSSF é formada por centenas ou até mesmo milhares de nós sensores trabalhando em conjunto em um determinado ambiente. Antes de vê-la funcionando, é necessário compreender o passo-a-passo do seu desenvolvimento. O primeiro passo é entender que a RSSF pode ser dividida em duas partes: a aplicação e a rede. Esta metodologia considera que uma aplicação é um software desenvolvido em uma linguagem de programação e que é executada em um nó sensor. A rede, por sua vez, é definida como um conjunto de nós sensores se comunicando para formar uma rede *ad hoc*. Para desenvolver uma rede, é preciso definir um conjunto de informações sobre ela, tais como a quantidade de nós sensores, os protocolos utilizados, tamanho da área de implantação, o padrão de implantação, a quantidade e a localização de nós sorvedouros. Em relação aos nós sensores, precisa-se do raio de alcance de comunicação dos nós, o nível de energia e a localização. Aliado a isto, as RSSFs possuem várias peculiaridades que devemos tratar, como por exemplo, a restrição de energia e de processamento e o fato de os nós sensores geralmente não passarem por manutenção quando apresentam uma falha.

Neste contexto, este trabalho propõe uma metodologia, ilustrada na Figura 4.2, com um conjunto de atividades pré-estabelecidas para criar uma RSSF, considerando as peculiaridades da aplicação e da rede. Além disso, esta metodologia avalia o consumo de energia e a confiabilidade da RSSF de forma integrada. Para facilitar a compreensão, o processo de avaliação é tratado como uma caixa preta na Figura 4.2 e será detalhado na Seção 4.6.1.

A metodologia guia o desenvolvimento da aplicação e da rede desde o planejamento até a implantação. Ela é composta por cinco fases: *Planejamento*, *Codificação*, *Otimização*, *Validação* e *Implantação*. A primeira fase é responsável por definir os requisitos e planejar o desenvolvimento da RSSF. Na fase seguinte, chamada *Codificação*, deve-se instanciar a RSSF, desenvolvendo o código da aplicação e definindo a configuração da rede. Em seguida, tem inicio a fase de *Otimização*, cuja responsabilidade é analisar o código fonte da aplicação e a configuração da rede, buscando por inconsistências na implementação da RSSF ou tentando melhorar o comportamento da *e.g.*, aumentar o tamanho do pacote e enviá-lo quando necessário para aumentar o tempo de vida na rede. Após otimizar a RSSF, deve-se avaliar o seu comportamento, consumo de energia e a

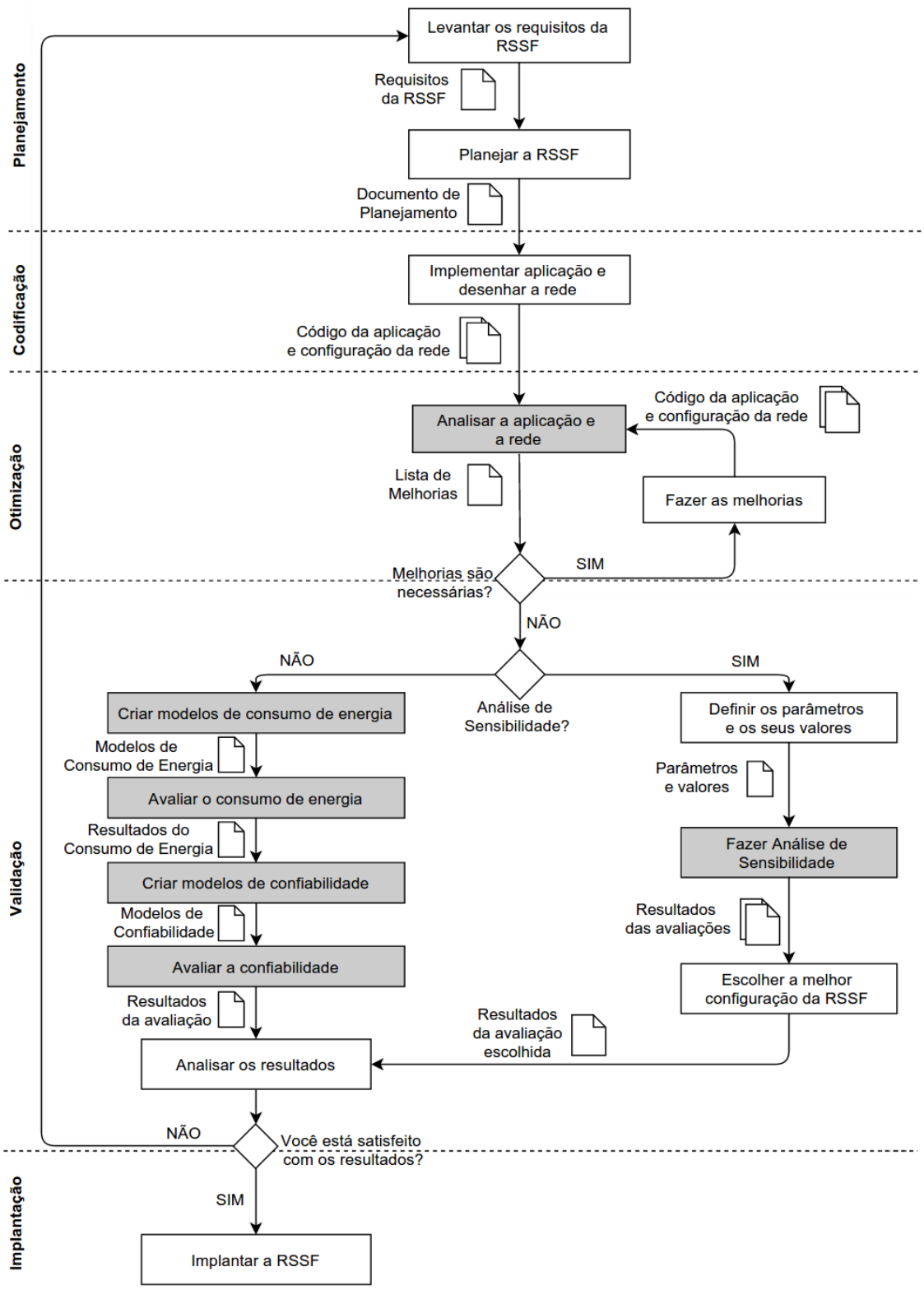


Figura 4.2 Atividades da metodologia proposta.

confiabilidade na fase *Validação*. Esta fase é primordial para verificar se o comportamento da RSSF e os resultados de consumo de energia e de confiabilidade estão de acordo com o planejado na fase *Planejamento*. Caso estejam, segue-se para a última fase, chamada de *Implantação*, que é responsável por implantar a RSSF em campo.

A fase *Validação* requer uma atenção a mais do que as outras fases porque ela utiliza modelos formais para avaliar o consumo de energia e a confiabilidade da RSSF. A avaliação das RSSFs possui três aspectos importantes definidos nesta metodologia: os modelos de consumo de energia e de confiabilidade são baseados em modelos formais; pequenos modelos reusáveis são propostos e combinados para representar o consumo de energia e a confiabilidade de uma determinada RSSF; e existe uma ordem de avaliação dos modelos. Esses três aspectos são fundamentais para avaliar o consumo de energia e a confiabilidade de forma integrada e independente da implementação da aplicação e da configuração da RSSF.

Cada fase será apresentada detalhadamente nas próximas seções.

4.3 Planejamento

A primeira fase, denominada *Planejamento*, está relacionada à identificação dos requisitos da RSSF e com o planejamento da RSSF. Inicialmente, o usuário deve criar dois grupos de requisitos manualmente na atividade *Levantar os requisitos da RSSF*: um conjunto de requisitos para aplicação e outro conjunto para a rede. Requisitos básicos para aplicação são o sistema operacional (*e.g.*, TinyOS ou Contiki) e a linguagem de programação (*e.g.*, nesC ou C) que serão utilizados, o período que o nó sensor irá enviar os dados (*e.g.*, frequentemente ou eventualmente), qual atividade será executada (*e.g.*, coletar temperatura, rastrear objetos ou detectar invasão), se vai agregar os dados e como irá agregá-los, e assim por diante. Com relação à rede, os requisitos básicos podem ser o tipo de plataforma utilizada (*e.g.*, IRIS ou MICAz), o protocolo de roteamento adotado (*e.g.*, LEACH ou DIRECT), o protocolo MAC (*e.g.*, B-MAC), o as dimensões do ambiente onde será implantada a RSSF, a forma como os nós sensores serão dispostos no ambiente, se terá segurança, se terá alguma estratégia de retransmissão de pacotes, qual será a topologia da rede, e assim por diante. Vale ressaltar que, mesmo separando os requisitos da aplicação e da rede, o usuário deve ficar atento para evitar conflitos ou incoerência entre os requisitos dos dois grupos, porque esses requisitos irão definir as características da RSSF.

Por exemplo, o usuário escolhe criar um aplicativo multimídia na RSSF, os nós senso-

res capturam vídeos do ambiente e enviam para o nó sorvedouro. Esse tipo de aplicativo necessita de um protocolo na camada de transporte para controlar o congestionamento de dado e parar criar vários caminhos entre o nó sensor e o nó sorvedouro (Akyildiz *et al.*, 2007). No entanto, o usuário ignora essa necessidade e determina que nenhum protocolo da camada de transporte será utilizado. Esse exemplo representa uma incoerência entre dois ou mais requisitos da RSSF, porque o tipo de aplicativo escolhido (requisito da aplicação) necessitava de uma determinada configuração da infraestrutura da rede (requisito da rede) para funcionar corretamente. Vale ressaltar que a RSSF implantada poderá funcionar como esperado, os nós sensores irão enviar e receber dados; mas o desempenho, consumo de energia e confiabilidade da RSSF poderiam ter sido melhores se um protocolo de transporte adequado para esse tipo de aplicativo fosse considerado. E isso deveria ter sido previsto na fase *Planejamento*.

Complementando esses requisitos, o usuário deve definir métricas como tempo de vida da RSSF, confiabilidade, consumo de energia, desempenho dos nós sensores, qualidade do serviço prestado, custo, entre outros. As duas métricas mais importantes consideradas por esta metodologia são o consumo de energia e a confiabilidade porque elas podem ser obtidas avaliando-se o código fonte da aplicação e a configuração da rede, e podem ainda ser verificadas na fase *Validação* antes mesmo da RSSF ser implantada.

Após identificar os requisitos e definir as métricas de consumo de energia e de confiabilidade, o usuário deve planejar o desenvolvimento da RSSF. Esta atividade também é manual e o usuário pode utilizar qualquer notação para auxiliar nesse trabalho, por exemplo, UML (Rumbaugh *et al.*, 2004) ou SysML (OMG, 2012). Novamente, é importante que o usuário planeje a aplicação e a rede separadamente visto que cada um tem as suas características.

Esta metodologia não determina como os requisitos devem ser identificados/levantados e nem como a RSSF deve ser planejada. Ela apenas exige que um conjunto de documentos, *Requisitos da RSSF* e *Documentos de Planejamento*, seja criado para orientar a implementação da aplicação e a configuração da RSSF, e que possa ser verificado posteriormente na fase *Validação*.

4.4 Codificação

A segunda fase da metodologia é responsável pela instanciação da aplicação e da infraestrutura da RSSF (*e.g.*, definir a quantidade, a posição e a configuração dos nós sensores, os protocolos de roteamento e enlace, a localização do nó sorvedouro, entre outros). Em

outras palavras, o usuário deverá implementar manualmente o código fonte da aplicação e criar a configuração de infraestrutura da RSSF seguindo os requisitos e o planejamento definidos anteriormente. Tal como na fase anterior, as implementações da aplicação e da rede devem ser feitas separadamente. Esta estratégia possui duas vantagens: é possível avaliar separadamente o consumo de energia da aplicação e da rede usando as suas instâncias; e permite criar e avaliar diferentes configurações de rede, identificando qual configuração possui os melhores resultados de consumo de energia e/ou confiabilidade de acordo com os requisitos definidos anteriormente.

O código fonte da aplicação deve ser escrito em uma linguagem de programação e ser suportado por um sistema operacional. Com relação à configuração de rede, esta configuração deve permitir configurar a rede, informando os protocolos, a posição dos nós sensores, as configurações dos nós sensores, entre outras informações.

4.5 Otimização

Após instanciar a aplicação e a infraestrutura da RSSF, a próxima fase é a *Otimização*. Nesta fase, o usuário pode identificar inconsistências na implementação ou adotar sugestões comuns às RSSFs. Algumas inconsistências ou recomendações padrões da RSSF podem ser encontradas ou sugeridas analisando-se apenas as características do código fonte da aplicação ou a configuração da rede.

Esta metodologia propõe que a análise deve ser uma atividade semi-automática por dois motivos: algumas inconsistências e recomendações padrões da RSSF já são conhecidas e podem ser aplicadas a todas as RSSFs; um usuário leigo pode criar uma RSSF otimizada mesmo tendo pouco conhecimento. Um conjunto de otimizações podem ser aplicadas ao código fonte da aplicação e à configuração de infraestrutura da RSSF. Caso haja necessidade, o conjunto de otimizações pode sugerir ao usuário a otimização. Este, por sua vez, decide se a implementa ou não, considerando os requisitos estabelecidos na fase *Planejamento*. Esta metodologia recomenda que o usuário aplique apenas as melhorias propostas que não entrem em conflito com os principais requisitos definidos no *Documento de Planejamento*, garantindo que a RSSF tenha o comportamento tal como foi especificado.

Por exemplo, se um nó sensor não tem comunicação direta com o nó sorvedouro, os protocolos DIRECT e LEACH não são apropriados para esta RSSF, requerendo a utilização de outro protocolo de roteamento ou mudanças na posição dos nós sensores. Esse tipo de análise verifica as características dos nós sensores e não necessita que a RSSF

seja avaliada para detectar essa inconsistência. Com relação à aplicação, um conjunto de sugestões podem ser feitas sobre código fonte para diminuir o consumo de energia do nó sensor.

As otimizações são divididas em dois grupos: otimizações da aplicação e otimizações da rede. Duas otimizações para o código fonte da aplicação foram implementadas. A primeira otimização busca por trechos de códigos que consomem mais energia do que a sua versão otimizada. Os trechos de código são apresentados na Seção 6.3. A outra otimização implementada permite que o usuário compare o consumo de energia de duas ou mais funções implementadas no código fonte da aplicação. Vale ressaltar que outras otimizações podem ser implementadas. Por exemplo, otimizar o código binário que será executado pelo nó sensor.

Com relação à rede, apenas uma única otimização foi implementada: comparar o consumo de energia e a confiabilidade de duas ou mais configurações de rede criadas pelo usuário manualmente. Tal como as otimizações da aplicação, outras otimizações podem ser implementadas para a rede.

Após o usuário implementar todas as otimizações, ele poderá avaliar na próxima fase o comportamento, o consumo de energia e a confiabilidade da RSSF, usando o código fonte da aplicação e a configuração de infraestrutura da RSSF. Adicionalmente, é fundamental passar primeiro pela fase de *Otimização* antes da *Validação* porque os mecanismos de otimização irão auxiliar na fase de validação.

4.6 Validação

A próxima fase está relacionada à validação do comportamento, consumo de energia e confiabilidade da RSSF. Esta fase propõe que a avaliação do consumo de energia e da confiabilidade façam parte da validação da RSSF e que a avaliação seja feita usando modelos. Para facilitar o entendimento, o processo de avaliação é tratado aqui como uma caixa preta que possui entradas e saídas apenas. O detalhamento da avaliação será feito na Seção 4.6.1.

Por ser um processo complexo, a avaliação é realizada com pouca intervenção do usuário. Existem três tipos de processo de avaliação previstos na metodologia: avaliar apenas a aplicação, avaliar apenas os componentes de comunicação da rede, e avaliar os dois elementos anteriores conjuntamente. O primeiro tipo é avaliar o consumo de energia apenas da aplicação sem considerar os aspectos da rede, tais como a quantidade de nós sensores e o roteamento de dados. Para isso, o usuário deve informar o código fonte da

aplicação, que será convertido em um modelo formal usado para avaliar o consumo de energia da aplicação. Este tipo de processo irá retornar o consumo de energia da aplicação em um determinado período, podendo retornar o consumo de energia individual de cada função implementada na aplicação.

O segundo tipo de processo é avaliar o consumo de energia e a confiabilidade apenas da rede sem considerar as características da aplicação. Por exemplo, este modelo não considera se o nó sensor coleta a temperatura ou se monitora abalos sísmicos. Na realidade, a única característica da aplicação considerada é a periodicidade de envio de pacotes. Tal como no primeiro processo de avaliação, o usuário deve informar a configuração de rede que será convertida em modelos formais. Por sua vez, os modelos criados serão avaliados para obter o consumo de energia e a confiabilidade da rede. A saída desse tipo de processo retorna vários relatórios com o resultado do consumo de energia e da confiabilidade da rede. Por exemplo, o tempo de funcionamento de cada nó sensor, nível de energia, a confiabilidade de uma região, entre outras coisas.

A última possibilidade é avaliar o consumo de energia e a confiabilidade considerando a aplicação e os elementos de rede juntos. O processo de avaliação é similar aos dois já apresentados: o usuário deve informar o código fonte da aplicação e a configuração da rede. Estas informações serão convertidas em modelos formais usados na avaliação. A saída desse tipo de avaliação é a mesma do primeiro (dados da aplicação) e do segundo tipo (dados da rede).

O usuário é responsável por escolher qual das três possibilidades de avaliação deseja utilizar. Esta metodologia recomenda que as duas primeiras sejam utilizadas para acompanhar a evolução do consumo de energia e a confiabilidade da aplicação e da rede. Elas devem ser utilizadas, principalmente, quando houver alguma mudança significativa determinada pelo usuário. Por exemplo, mudar a lógica da aplicação ou mudar o protocolo de roteamento. Como o usuário está acompanhando a evolução do consumo de energia e da confiabilidade da aplicação e da rede, ele entenderá melhor o impacto de cada mudança realizada. Logo após terminar o processo de avaliação, o usuário deve voltar para a fase *Planejamento* para verificar os requisitos restantes da RSSF e, em seguida, para a fase *Codificação* para completar a implementação da aplicação e/ou da rede.

A terceira possibilidade deve ser utilizada quando o código da aplicação e a configuração de rede já estão finalizados ou próximos de serem finalizados. Esta tese considera que o código da aplicação e a configuração estão próximos de serem finalizados quando os principais requisitos forem implementados ou atendidos. Adicionalmente, o usuário deve determinar quais são os principais requisitos da aplicação e da rede. Como o usuário

acompanhou a evolução do consumo de energia e da confiabilidade da aplicação e da rede, quando for avaliá-los conjuntamente, ele/ela terá uma melhor noção dos resultados e de qual mudança está impactando a RSSF. Os resultados da terceira possibilidade devem ser os únicos considerados para decidir se a RSSF deve ou não ser implantada.

Para validar a RSSF, o usuário deve comparar os resultados de consumo de energia e de confiabilidade com as métricas definidas na primeira fase. Além disso, deve verificar se o comportamento da RSSF está de acordo com os requisitos definidos e com o planejamento elaborado na fase inicial da metodologia. Em outras palavras, a validação da RSSF corresponde à aprovação do comportamento e dos resultados obtidos, indicando que a RSSF está pronta para ser implantada no ambiente desejado. Caso a RSSF não seja validada, o usuário deve rever se houve erro de planejamento (fase *Planejamento*) ou de implementação (fase *Codificação*).

O usuário deve estar atento a alguns detalhes que podem invalidar a RSSF. Por exemplo, ele pode fazer uma estimativa errada do consumo de energia e de confiabilidade, definir uma quantidade de nós sensores não compatíveis com o tamanho do ambiente onde a RSSF será implantada, escolher um protocolo de roteamento inadequado para a topologia definida, planejar uma determinada configuração de infraestrutura da RSSF (*e.g.*, a comunicação ser *full-duplex*), mas (na realidade) ela se comporta de outro modo (*e.g.*, a comunicação ser *simplex*), e assim por diante.

Esses três tipos de avaliação podem ser usados de duas formas. Na primeira, o usuário pode avaliar o consumo de energia e a confiabilidade de uma aplicação e/ou de uma configuração de rede. Na segunda forma, o usuário pode identificar qual é a melhor configuração de rede de acordo com os requisitos e as métricas estabelecidos anteriormente. Neste caso, o usuário pode criar manualmente e avaliar individualmente diversas configurações de rede diferentes. Esse tipo de atividade é conhecida como a análise de sensibilidade (Seção 2.4). Este tipo de análise ocorre em três passos. No primeiro passo, o usuário especifica manualmente um conjunto de parâmetros com os respectivos valores. Um conjunto de configurações de rede é criado, combinando os valores estabelecidos, e avaliado automaticamente no segundo passo. Por fim, no terceiro passo, o impacto dos parâmetros no consumo de energia e na confiabilidade é calculado e os resultados de todas as configurações de rede são mostrados. Usando esses resultados, o usuário pode escolher qual foi a melhor configuração de rede, considerando os requisitos e as métricas definidas na fase *Planejamento*.

A análise de sensibilidade pode utilizar o segundo e o terceiro tipo de avaliação descritos anteriormente. Caso o usuário escolha avaliar apenas as configurações de rede

criadas, ele deve voltar para a fase *Planejamento* e, em seguida, continuar a implementar a RSSF. Caso o usuário escolha avaliar a aplicação juntamente com as configurações de rede criadas, após escolher a melhor configuração de rede, ele pode verificar se o resultado está satisfatório ou não para implementar a RSSF de acordo com os requisitos e as métricas estabelecidos na fase *Planejamento*.

A análise de sensibilidade poderia ter sido implementada como um mecanismo da fase *Otimização*. No entanto, uma característica dela foi considerada como fundamental para ser implementada na fase *Validação*: a análise de sensibilidade, logo após ser realizada, possibilita que a RSSF seja validada e implantada, porque o usuário pode validar e implantar a RSSF com o melhor resultado, de acordo com o usuário.

4.6.1 Avaliação

Até agora, a metodologia tratou o processo de avaliação da RSSF como uma caixa preta para facilitar a compreensão, mostrando apenas a entrada e a saída, e descrevendo o processo interno resumidamente. Esta seção irá explicar detalhadamente como funciona o processo interno dessa caixa, mostrando como esta tese modelou e avaliou o consumo de energia e a confiabilidade da RSSF.

O processo de avaliação foi definido seguindo três princípios básicos da modelagem da RSSF. Primeiro, esta tese define pequenos modelos reusáveis, chamados modelos básicos, que representam a menor unidade de consumo de energia e de confiabilidade da RSSF. Quando esses pequenos modelos reusáveis são combinados, eles formam os modelos finais utilizados para representar uma aplicação e/ou uma infraestrutura da RSSF e avaliá-los. Os pequenos modelos reusáveis propostos são combinados de acordo com as informações contidas no código fonte da aplicação e na configuração de infraestrutura da RSSF.

Devido à dificuldade de avaliar o consumo de energia e a confiabilidade da RSSF, é inviável avaliar toda a RSSF utilizando apenas um único modelo. Por causa disso, o segundo princípio define a criação de vários modelos e cada um deles avalia um aspecto da RSSF. Mais especificamente, esta metodologia propõe 4 modelos finais diferentes, 3 relacionados ao consumo de energia e 1 à confiabilidade. Devido à complexidade de avaliar o consumo de energia, foi necessário criar um modelo final para avaliar a aplicação, outro para a configuração da rede e um terceiro para avaliar ambos ao mesmo tempo. O modelo final de confiabilidade avalia a confiabilidade de uma região presente na RSSF. Esta metodologia considera que uma região é um conjunto de nós sensores replicados fazendo a mesma atividade e provendo o mesmo serviço. Desta maneira, se

um nó sensor parar de funcionar, a região poderá continuar provendo o serviço.

Quando todos esses 4 modelos finais são avaliados, significa dizer que o consumo de energia e a confiabilidade de toda a rede foi avaliada. No entanto, os modelos finais dependem uns dos outros para avaliarem a RSSF completamente e corretamente. Este problema é tratado no terceiro princípio: esta tese define uma sequência de avaliação dos modelos finais porque a saída de um modelo final serve como entrada para outro. Desta maneira, a sequência de avaliação dos modelos finais cria a impressão de ser uma única solução, como se um único modelo final fosse utilizado: o usuário informa à RSSF e obtêm o consumo de energia e a confiabilidade dela, avaliando todos os modelos finais.

A Figura 4.3 apresenta as sub-atividades da avaliação que estão agrupadas em três partes: atividades de avaliação da aplicação, atividades de avaliação dos elementos de comunicação da rede; e atividades da avaliação integrada. Como já foi mencionado anteriormente, a aplicação e a rede podem ser avaliadas separadamente e em conjunto.

4.6.2 Avaliação da Aplicação

A avaliação da aplicação consiste em avaliar o consumo de energia da aplicação. Para isso, o usuário deve informar o código fonte da aplicação, o qual será usado para construir e avaliar o modelo final da aplicação. Esse modelo final, chamado de Modelo de Aplicação, é formado por pequenos modelos reusáveis, denominados Modelos do Operador e que estão armazenados em um repositório.

O processo de construção do Modelo de Aplicação é o seguinte: os operadores utilizados no código fonte da aplicação são identificados e convertidos em Modelos do Operador. Por sua vez, esses modelos reusáveis são acrescentados ao Modelo de Aplicação na mesma ordem que aparecem os operadores no código fonte da aplicação. Desta maneira, o Modelo de Aplicação pode avaliar o consumo de energia de qualquer aplicação da RSSF. Como o modelo final é baseado no código fonte da aplicação, qualquer alteração no código fonte da aplicação implica na necessidade de se gerar um novo modelo final. Caso contrário, o resultado do consumo de energia não irá representar a aplicação. Adicionalmente, detalhes de implementação desse modelo final são apresentados na Seção 5.2.

Além disso, a avaliação da aplicação permite avaliar a aplicação inteira ou parte dela. Essa característica auxilia o usuário no desenvolvimento da aplicação porque ele pode identificar qual função consome mais energia e acompanhar a evolução do consumo de energia a cada alteração. Para isso, ele deve avaliar individualmente cada função presente na aplicação.

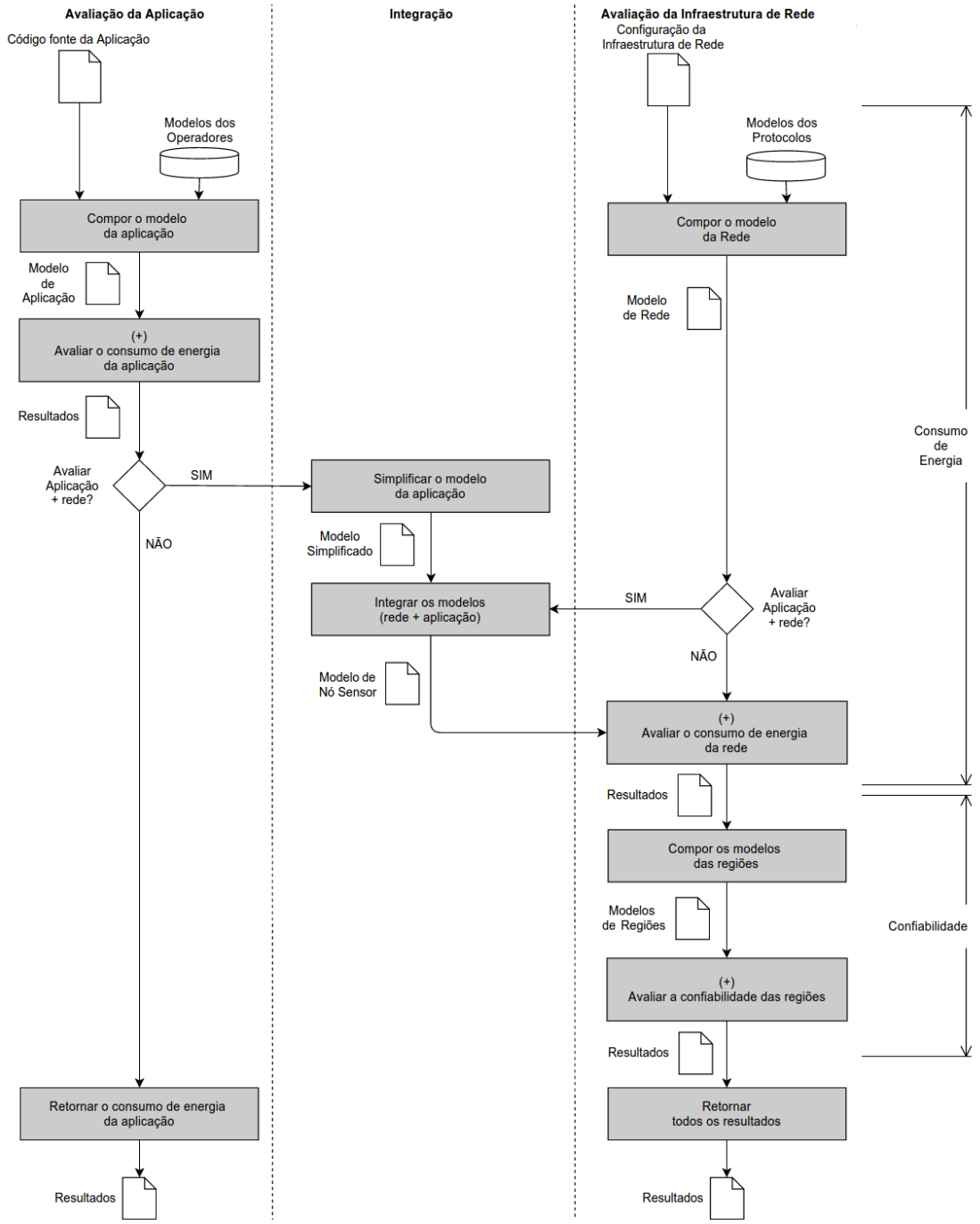


Figura 4.3 Detalhamento da atividade de Avaliação

4.6.3 Avaliação da Rede

A avaliação da rede consiste em avaliar o consumo de energia e a confiabilidade considerando a configuração de rede. Primeiro, este processo avalia o consumo de energia e, depois, a confiabilidade. É importante seguir essa ordem porque os dados da avaliação do consumo de energia são utilizados para avaliar a confiabilidade da RSSF.

O modelo final da rede, chamado Modelo de Rede, é criado de acordo com a configuração de rede. O processo identifica a quantidade e a configuração dos nós sensores, os protocolos utilizados, características do ambiente, entre outras coisas. Em especial, os protocolos nesta tese são representados por modelos reusáveis, chamados de Modelos de Protocolo, também armazenados em um repositório. Esses modelos reusáveis representam o comportamento e o consumo de energia do protocolo. Além desses pequenos modelos representando os protocolos, existe o Modelo de Ambiente que representa o ambiente onde o nó sensor irá ser implantado. Esse modelo do ambiente irá determinar como os dados enviados pelos nós sensores são propagados no ambiente e quais nós sensores irão recebê-los.

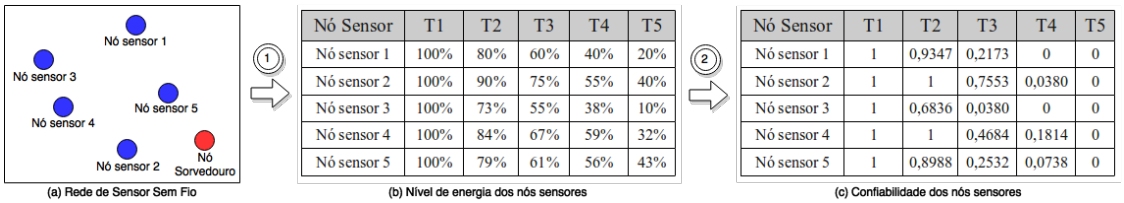
A combinação dos Modelos de Protocolo representa a pilha de protocolos do nó sensor. Por sua vez, essa representação da pilha de protocolos mais o Modelo de Ambiente representam a configuração de infraestrutura da RSSF modelada. Caso haja alguma mudança na configuração de rede, é necessário criar um novo Modelo de Rede. Os detalhes de implementação desse modelo final são apresentados na Seção 5.3.

Após avaliar o consumo de energia da rede, é gerado um artefato com as configurações e com os resultados da avaliação de todos os nós sensores e, consequentemente, de todas as regiões. Como foi dito anteriormente, uma região é um conjunto de nós sensores replicados, fazendo a mesma atividade e provendo o mesmo serviço. Se um nó sensor para de funcionar, a região poderá continuar provendo o serviço.

As informações sobre a infraestrutura da RSSF (*e.g.*, protocolo de roteamento) e sobre os nós sensores (*e.g.*, localização e o nível de energia dos nós sensores) são utilizadas para avaliar a confiabilidade das regiões presentes na RSSF (Seção 5.4). Para modelar uma região, é necessário definir o caminho entre ela e o nó sorvedouro porque a região pode não prover mais o serviço caso haja algum problema no caminho entre eles. Existem diversas maneiras de definir um caminho e, dependendo da quantidade de nós sensores, isso poderia causar um problema de escalabilidade. Para resolver isso, esta metodologia propõe que os caminhos devem ser definidos por um algoritmo de roteamento que precisa ser o mesmo informado pela configuração de rede e usado na avaliação da rede. Como um dos desafios a ser resolvido pelo algoritmo de roteamento é a escalabilidade da RSSF

(Al-Karaki and Kamal, 2004), então, o algoritmo de roteamento definirá as possibilidades de caminhos entre a região e o nó sorvedouro no modelo proposto, tornando a solução escalável.

Antes de definir os caminhos, é necessário estabelecer a confiabilidade dos nós sensores. A Figura 4.4 mostra os dois passos para obter a confiabilidade dos nós sensores. No primeiro passo, após o usuário criar a RSSF, ele deve avaliar o consumo de energia da rede. Um histórico do consumo de energia da RSSF é retornado para o usuário, registrando a configuração dos nós sensores. Esse histórico de consumo de energia é utilizado para criar o histórico de confiabilidade no segundo passo. Para isso, é necessário aplicar a Equação 7.6.1 em cada registro de nível de energia de todos os nós sensores. Com o histórico de confiabilidade, é possível calcular a confiabilidade de uma região, como é explicado, com mais detalhe, na Seção 5.4.



(a) Rede de Sensor Sem Fio

Nó Sensor	T1	T2	T3	T4	T5
Nó sensor 1	100%	80%	60%	40%	20%
Nó sensor 2	100%	90%	75%	55%	40%
Nó sensor 3	100%	73%	55%	38%	10%
Nó sensor 4	100%	84%	67%	59%	32%
Nó sensor 5	100%	79%	61%	56%	43%

(b) Nível de energia dos nós sensores

Nó Sensor	T1	T2	T3	T4	T5
Nó sensor 1	1	0,9347	0,2173	0	0
Nó sensor 2	1	1	0,7553	0,0380	0
Nó sensor 3	1	0,6836	0,0380	0	0
Nó sensor 4	1	1	0,4684	0,1814	0
Nó sensor 5	1	0,8988	0,2532	0,0738	0

(c) Confiabilidade dos nós sensores

Figura 4.4 Passo a passo para calcular a confiabilidade dos nós sensores da RSSF.

Um novo modelo final de confiabilidade é gerado para todos os tempos registrados no histórico do consumo de energia. Isso ocorre porque os caminhos entre a região e o nó sorvedouro podem ser alterados ao longo do tempo dependendo do algoritmo de roteamento utilizado. Por exemplo, no tempo *T1* pode ter um modelo final diferente do tempo *T2* usando um algoritmo de roteamento *X*. Por este motivo, um novo modelo final de região é gerado a cada tempo registrado. Após vários modelos finais serem criados e avaliados, o histórico de confiabilidade das regiões da RSSF é gerado.

4.6.4 Avaliação Conjunta

Para avaliar a RSSF, as duas avaliações anteriores são utilizadas e, por este motivo, uma ordem foi definida para executá-las. O usuário deve informar o código fonte da aplicação e a configuração de infraestrutura da RSSF. O código fonte da aplicação é convertido em um Modelo de Aplicação e a configuração de rede, em Modelo de Rede. Em seguida, o consumo de energia do Modelo de Aplicação é avaliado e simplificado (Seção 5.3.4), gerando o Modelo Simplificado. Este modelo final considera apenas o consumo de energia médio da aplicação e a frequência que ela envia um pacote. Essa

simplificação é importante para viabilizar a avaliação da RSSF, aplicação e infraestrutura da rede juntas. Caso não fosse feita, o modelo final poderia ter tantas atividades para executar que inviabilizaria a sua avaliação.

A integração do Modelo Simplificado com o Modelo de Rede cria um novo modelo final, chamado de Modelo do Nó Sensor. Desta maneira, aspectos da aplicação e da rede são considerados e avaliados. Após o Modelo do Nó Sensor ser avaliado, segue o mesmo fluxo da rede: dados da avaliação são capturados e repassados para a criação e avaliação do modelos finais da confiabilidade.

4.7 Implantação

A última fase desta metodologia define que a RSSF deve ser implantada tal como foi planejada. Se o usuário implantar uma RSSF diferente da planejada, os resultados obtidos não representarão a RSSF implantada. Desta maneira, um novo planejamento da RSSF deve ser feito e validado.

4.8 Considerações Finais

Este capítulo apresentou uma metodologia com um conjunto de atividades pré-estabelecidas para criar e avaliar uma RSSF considerando a sua aplicação e a rede. A metodologia foi dividida em cinco fases: *Planejamento*, planejar o desenvolvimento da RSSF; *Codificação*, instanciar a aplicação e a rede; *Otimização*, encontrar inconsistências de implementação ou fazer melhorias; *Validação*, avaliar e validar o comportamento, o consumo de energia e a confiabilidade da RSSF; e *Implantação*, implantar a RSSF em campo. Em especial, a fase *Validação* foi mais detalhada nesta tese por avaliar o consumo de energia e a confiabilidade da RSSF utilizando modelos formais. Esta metodologia define 4 modelos finais (3 para avaliar o consumo de energia e 1 para a confiabilidade), como eles devem ser criados e a ordem de avaliação desses modelos. Devido à complexidade para criar e avaliar os modelos finais, esta metodologia é suportada por um conjunto de ferramentas apresentadas no Capítulo 6. E o próximo capítulo (Capítulo 5) apresenta os quatro conjunto de modelos mencionados nesta metodologia.

5 Modelos

Este capítulo apresenta os modelos relacionados ao consumo de energia e à confiabilidade da RSSF. Inicialmente, os modelos relacionados ao consumo da aplicação e da rede são apresentados. Em seguida, o modelo de confiabilidade é detalhado.

5.1 Visão Geral

Esta tese definiu quatro conjunto de modelos relacionados ao consumo de energia e à confiabilidade. Por ser mais complexo, o consumo de energia foi divido em três modelos: Modelo de Aplicação (Dâmaso *et al.*, 2013), usado para avaliar o consumo de energia de uma aplicação escrita em nesC; Modelo de Rede (Dâmaso *et al.*, 2014a), usado para avaliar o consumo de energia da rede; e Modelo de Nó Sensor (Dâmaso *et al.*, 2014a), usado para avaliar o consumo de energia da aplicação e da rede ao mesmo tempo. Por sua vez, o modelo de confiabilidade, chamado de Modelo de Região (Dâmaso *et al.*, 2014b), é usado para avaliar a confiabilidade de uma região presente na RSSF em um determinado instante de tempo.

5.1.1 Modelos do Consumo de Energia

Para facilitar a compreensão, esta seção apresenta, inicialmente, os elementos que são considerados pelos modelos relacionados ao consumo de energia da RSSF. A Figura 5.1 ilustra dois principais elementos: nó sensor e ambiente. O nó sensor é responsável por executar uma operação, como coletar temperatura, e cooperar na rede enviando, roteando, recebendo e processando pacotes. A rotina está relacionada com o código da aplicação implementado pelo desenvolvedor, e a cooperação na rede está relacionada com a pilha de protocolos. Como foi mencionado na Seção 2.1.3, a pilha de protocolos

da RSSF é formada por quatro camadas, onde cada uma tem uma função específica. A camada de aplicação possui um protocolo relacionado com a rotina do nó sensor; a camada de transporte, por controlar o congestionamento e melhorar a confiabilidade da comunicação fim-a-fim; a camada de rede, por rotear os pacotes; e a camada de enlace, por controlar o acesso ao meio. Quando um nó sensor envia um pacote, o ambiente assume a responsabilidade por transmitir o pacote do nó transmissor até o(s) nó(s) receptor(es). Adicionalmente, o nó sorvedouro é considerado um nó sensor, mas executa uma rotina diferente.

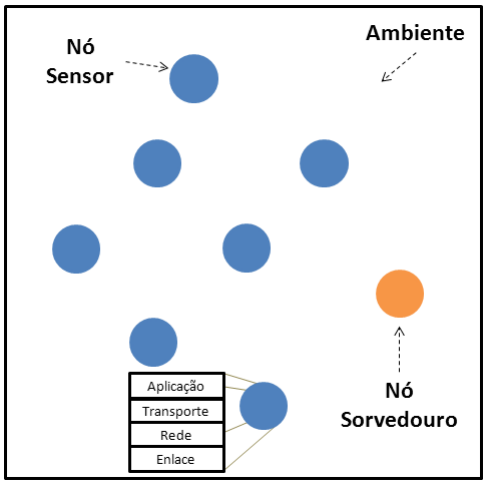


Figura 5.1 Elementos da RSSF considerados no processo de modelagem do consumo de energia

Devido à complexidade do consumo de energia da RSSF, optou-se por modelar separadamente a aplicação (rotinas implementadas pelo desenvolvedor) da rede (ambiente e pilha de protocolos sem a camada de aplicação) para depois uni-las. Adicionalmente, a camada de aplicação da rede foi ignorada por estar relacionada com atividade determinada pelo desenvolvedor, enquanto as demais camadas são genéricas e podem ser utilizadas por qualquer tipo de aplicação. Desta maneira, foram definidos três modelos para avaliar o consumo de energia da rede: Modelo de Aplicação para avaliar o consumo de energia da aplicação, Modelo de Rede para avaliar o consumo de energia da rede e Modelo de Nó Sensor para avaliar o consumo de energia da aplicação e da rede ao mesmo tempo.

Baseado na metodologia, a Figura 5.2 mostra uma visão geral dos modelos e dos passos necessários para obter o consumo de energia da RSSF. Inicialmente, o desenvolvedor implementa uma aplicação e define uma configuração da RSSF para ser avaliada (passo *Elaboração*). Os Modelos dos Operadores e dos Protocolos definem o consumo de energia de pequenas partes da aplicação, *e.g.*, comandos da linguagem, e protocolos,

respectivamente. Esses modelos pertencem à uma biblioteca de modelos básicos reusáveis que podem ser compostos de várias formas para modelar diferentes aplicações (Modelo de Aplicação) e pilha de protocolos (Modelo de Rede). Depois, o Modelo de Aplicação é avaliado e um Modelo Simplificado é criado (passo *Simplificação*). Finalmente, o modelo de consumo de energia do nó sensor (Modelo de Nó Sensor) é obtido integrando-se o Modelo de Aplicação e modelo de Rede (passo *Integração*). Adicionalmente, o passo *Codificação* é equivalente à fase *Codificação* da metodologia; e os outros passos (*Composição*, *Simplificação* e *Integração*) estão relacionados com avaliação do consumo de energia da RSSF detalhado na metodologia (veja o Capítulo 4).

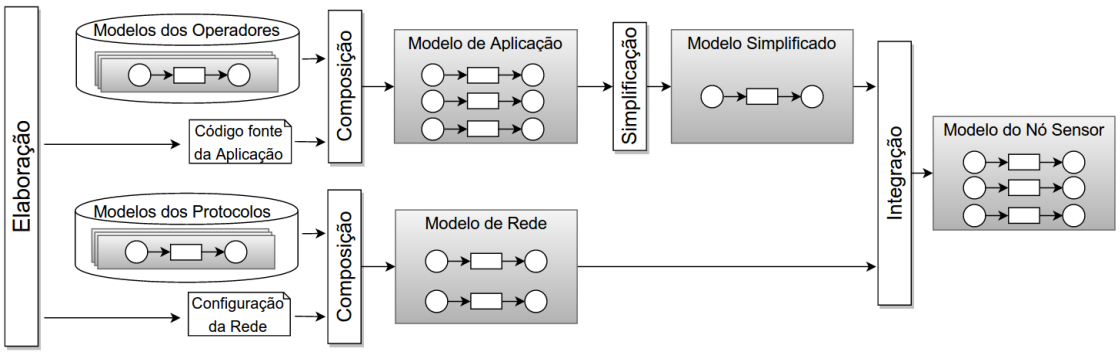


Figura 5.2 Visão Geral da criação dos modelos relacionados ao consumo de energia

Mais detalhadamente, a Figura 5.3 mostra a visão geral para elaborar o Modelo de Aplicação e o Modelo de Rede. Essa visão geral mostra três itens: o repositório, os modelos intermediários e o modelo final. O repositório armazena os modelos básicos reusáveis, os quais são usados para elaborar os Modelos Intermediários (MIs). Os modelos básicos representam o consumo de energia de um operador da aplicação (por exemplo, aritméticos, atribuição, binário, entre outros citados na Seção 2.1.2), chamado de Modelo do Operador (MO); ou representam o consumo de energia de um protocolo, chamado Modelo de Protocolo (MP), da camada de transporte, da camada de rede ou da camada de enlace. O segundo item (os MIs) representa uma parte da aplicação ou da rede modelada. No caso do consumo de energia, os MOs e MPs são agrupados entre si para elaborar, respectivamente, o modelo que representa uma função (no caso da aplicação) ou uma pilha de protocolos (no caso da rede). Como foi dito anteriormente, esses elementos básicos podem ser reutilizados para gerar outros MIs.

O Modelo Final representa a RSSF e é a composição dos MOs criados juntamente com alguns Modelos Pre-definidos (MPDs). Os MIs representam as particularidades (as funções implementados pelo desenvolvedor, no caso da aplicação, e a pilha de protocolos,

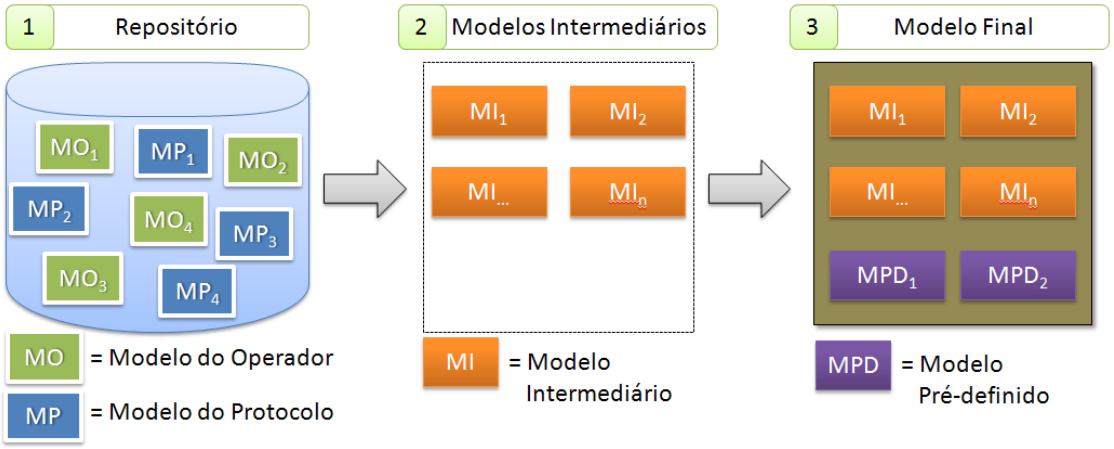


Figura 5.3 Visão Geral da criação do Modelo de Aplicação e do Modelo de Rede

no caso da rede) da aplicação ou da rede, enquanto os MPDs representam os pontos comuns a todas as aplicações ou redes. Por exemplo, todas as aplicações inicializam da mesma forma e os nós sensores usam a comunicação sem fio para transmitir um pacote. Como foi dito anteriormente, existem três Modelos Finais para avaliar o consumo de energia da RSSF: Modelo de Aplicação, Modelo de Rede e o Modelo de Nó Sensor. Entretanto, caso a aplicação ou a rede sofra alterações (*e.g.*, adição de nó sensor na rede), é necessário refazer todos os modelos.

5.1.2 Modelo de Confiabilidade

Tal como os modelos de consumo, é necessário caracterizar os elementos que foram considerados no processo de modelagem antes de apresentarmos o modelo de confiabilidade da RSSF. A Figura 5.4 mostra esses elementos e os seus relacionamentos. Todas as RSSFs são compostas por nós sensores, nós sorvedouros e enlaces de comunicação como foi introduzido na Seção 2.1. Além desses elementos, nós definimos a noção de região e caminho. Uma região consiste de um conjunto de nós redundantes que analisam o mesmo fenômeno físico e que estejam próximos um do outro, ou seja, os nós sensores irão praticamente capturar os mesmos valores do fenômeno físico. Por sua vez, um caminho representa o conjunto de nós sensores e enlaces de comunicação entre um nó sensor de uma região até o nó sorvedouro. Até esse ponto, é importante observar um aspecto chave na nesta tese: o caminho depende do algoritmo de roteamento usado. Então, vários caminhos podem existir entre um nó sensor particular e o nó sorvedouro, pois suas confiabilidades podem ser diferentes. Além disso, a RSSF pode ter várias regiões localizadas em coordenadas diferentes e com quantidades de nós sensores diferentes, *e.g.*,

uma região com 4 nós sensores, e outro com apenas 1 nó sensor). Adicionalmente, o usuário deve criar as regiões de acordo com a sua necessidade e características da rede.

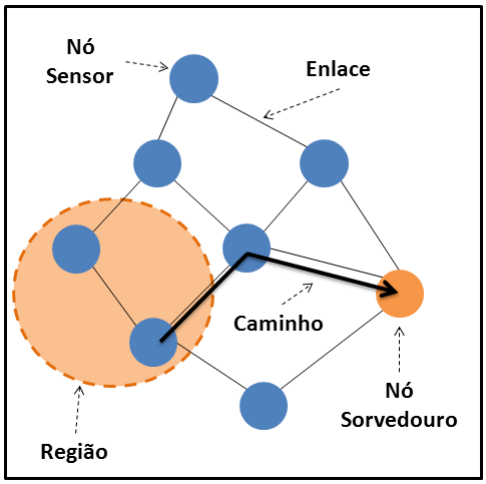


Figura 5.4 Elementos da RSSF considerados no processo de modelagem da confiabilidade

Por usarmos estes elementos (nós sensores, caminhos e regiões), foi definida uma estratégia para modelá-los em RBD. Inicialmente, cada modelo RBD possui um ponto inicial e um ponto final. Na RSSFs, o ponto inicial é qualquer região da rede, enquanto o ponto final é sempre o nó sorvedouro. Segundo, para o propósito de confiabilidade, a RSSF é dividida em regiões e sua confiabilidade é computada individualmente para cada região. Essa divisão é necessária porque a confiabilidade da região é afetada por quatro fatores: a posição dos nós sensores que compõem a região, o algoritmo de roteamento usado, a confiabilidade dos nós sensores e os enlaces de comunicação da RSSF. Finalmente, cada elemento citado anteriormente possui um modelo de confiabilidade associado, que pode ser representado por: blocos básicos, usados para modelar os nós sensores e os enlaces; modelo de caminho, usado para modelar a confiabilidade do caminho entre um nó sensor até o nó sorvedouro; e o Modelo de Região, usado para modelar a confiabilidade de uma região.

A Figura 5.5 apresenta o passo-a-passo da estratégia proposta. A partir da configuração da infraestrutura da RSSF (*e.g.*, posição dos nós sensores e algoritmo de roteamento adotado), o usuário precisa definir manualmente a região a ser avaliada, *e.g.*, localização da região, distância para o nó sorvedouro, quantidade de nós sensores. Os próximos passos serão executados automaticamente por um conjunto de ferramentas e consistirão em definir os caminhos, usando o algoritmo de roteamento adotado na RSSF, para cada nó sensor presente na região. Com esses caminhos definidos, o próximo passo é gerar os

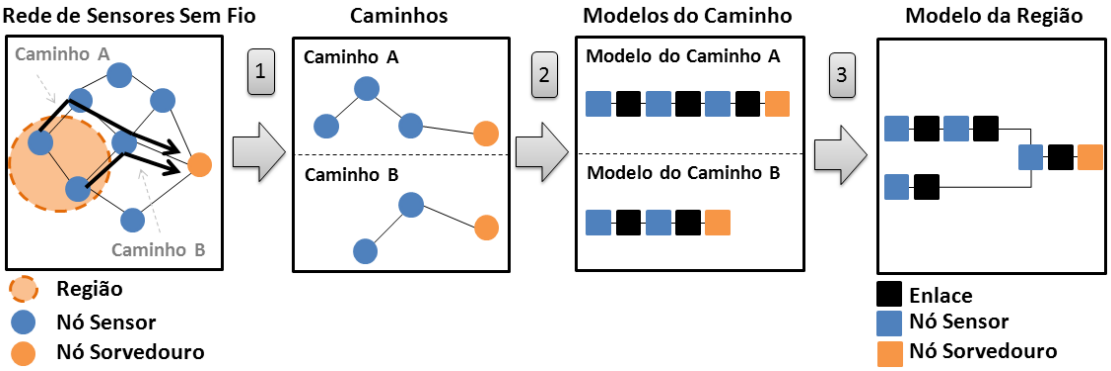


Figura 5.5 Visão geral do processo de criação do Modelo de Região

modelos de confiabilidade para cada um deles. O último passo consiste em compor os modelos dos caminhos criados, produzindo o modelo de confiabilidade da região. Novamente, é importante observar que os nós sensores que compõem um caminho dependem do algoritmo de roteamento utilizado.

5.2 Modelo do consumo de energia da aplicação

A abordagem adotada para explicar o Modelo de Aplicação *bottom-up*. Primeiro, esta seção apresentará os Modelos dos Operadores (MO), os quais modelam o consumo de energia dos operadores e estruturas da linguagem nesC. Depois, é apresentado o Modelo de Função, que consiste de um conjunto de Modelos dos Operadores para representar uma função da aplicação. E, por fim, é apresentado o Modelo Principal, responsável por orquestrar a ordem de execução dos modelos das funções.

5.2.1 Modelos dos Operadores

O primeiro passo para a modelagem da aplicação é definir os modelos CPNs para cada operador e estrutura da linguagem nesC. Como alguns deles possuem a mesma estrutura, é possível agrupá-los em quatro diferentes grupos: operadores, operadores de invocação, estrutura de seleção e estrutura de repetição. As próximas subseções apresentarão os detalhes de cada grupo.

Operadores

Por possuírem a mesma estrutura, os operadores aritméticos (+, -, *, /, e %), comparação (>, >=, <, <=, == e !=), lógica (&& e ||), incremento (++), decremento (–), operadores de

bit (& , | , ^ , « , » e ~) e atribuição (=) foram modelados através de um modelo genérico em CPN, como ilustra a Figura 5.6. A transição *op_1* representa o consumo de energia de um operador.

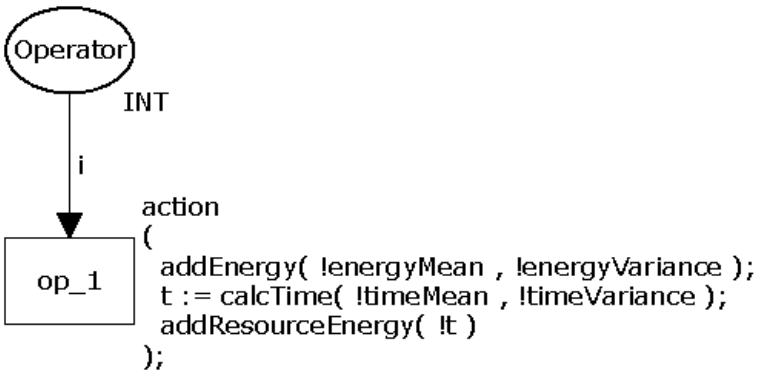


Figura 5.6 Modelo básico de um operador

Quando a transição *op_1* é disparada, o consumo de energia é calculado através de três funções `addEnergy()`, `calcTime()` e `addResourceEnergy()`. A função `addEnergy()` calcula um valor randômico seguindo a distribuição normal, usando os valores passados por parâmetros (`energyMean` e `energyVariance`), do consumo de energia do operador naquele momento. Depois, esta função incrementa o consumo de energia da aplicação com o valor calculado. Por gerar um valor randômico, é possível definir um intervalo de confiança, contendo esse valor. Por sua vez, a função `calcTime()` calcula o tempo para executar o operador naquele momento. De maneira similar à função anterior, a função `calcTime()` também usa a média e a variância para gerar um valor randômico seguindo a distribuição normal. Esse valor é incrementado no tempo da aplicação. Por fim, a função `addResourceEnergy()` calcula o consumo de energia dos recursos do *hardware* (rádio e LEDs) que podem ter sido ativados anteriormente (ver seção 5.2.1). Adicionalmente, a média e a variância do tempo e energia estão relacionadas a cada operador. Esses valores foram obtidos através de medição (ver seção 7.2).

Este modelo tem duas propriedades (ver Seção 2.2): ele é seguro, porque o lugar *Operator* não armazena nenhum *token*; e ele não tem nenhuma transição morta, porque a transição *op_1* sempre será executada.

Estrutura de Seleção

As estruturas de seleção *if-then-else* e *switch* são representadas pelo mesmo modelo CPN. Ambas possuem ramos e apenas um deles será executado (veja a seção 2.1.2). Como uma

estrutura de seleção pode ter vários ramos, é necessário associar uma probabilidade a cada ramo para ele ser executado. O desenvolvedor da aplicação é responsável por associar manualmente esses valores através do comentário `//@valor`. É importante ressaltar que o usuário da aplicação deve conhecer bem o funcionamento da aplicação para atribuir a probabilidade correta no ramo. Por exemplo, no caso abaixo, as probabilidades associadas pelo desenvolver foram: 80% (E_SUCESS), 15% (E_ERROR) and 5% (caso contrário).

```
switch( e ){
   case E_SUCESS: //@0.80
                  value = v; break;
   case E_ERROR:  //@0.15
                  value = -1; break;
   default:       //@0.05
                  value = 0;
}
```

A Figura 5.7 mostra o modelo CPN representando as estruturas *if-the-else* e *switch*. A transição *c1* decide qual ramo deve ser executado, gerando um valor randômico entre 0 e 1, seguindo a distribuição uniforme. Adicionalmente, essa distribuição foi escolhida devido a sua característica de gerar um número finito de resultados (entre dois valores) com chances iguais de acontecer. Após decidir qual ramo deve ser executado, a transição *c1* atribui a sua decisão ao valor do *token*, o qual será utilizado pelas transições subsequentes.

O lugar *el_x* (*e.g.*, *el_1* e *el_2*) representa a expressão de controle de um ramo e *body_y* (*e.g.*, *body_1*, *body_2* e *body_3*) representa o seu corpo. Enquanto isso, as transições *e_x* (*e.g.*, *e_1* e *e_2*) e *b_y* (*e.g.*, *b_1*, *b_2* e *b_3*) calculam o consumo de energia da expressão de controle e do corpo de um ramo, respectivamente.

Como mencionado na Seção 2.1.2, uma estrutura de seleção executa um ramo quando sua expressão de controle retorna `true`. Em outras palavras, a expressão de controle de um ramo consome energia depois de decidir se ele executa o corpo (quando retorna *true*) ou passa para a expressão de controle do próximo ramo (quando retorna `false`). A transição *e_x* (*e.g.*, *e_1* e *e_2*) representa este comportamento no modelo CPN. Esta transição verifica se é para executar o corpo ou se é para passar para o próximo ramo. Por exemplo, o lugar *el_1* verifica o valor do *token* e o move para o lugar *body_1* (quando o valor é igual a 0) ou para o lugar *el_2* (quando o valor é diferente de zero). A transição *e_2* tem um comportamento similar.

Este modelo possui propriedades similares ao modelo anterior quando todos os ramos têm uma probabilidade maior do que zero. Por outro lado, o corpo de um ramo nunca irá ser executado caso o ramo tenha uma probabilidade igual a zero (`//@0.0`). Consequentemente, a transição associada a este ramo (*e.g.*, *b_1*, *b_2* e *b_3* na 5.7) se torna uma transição morta.

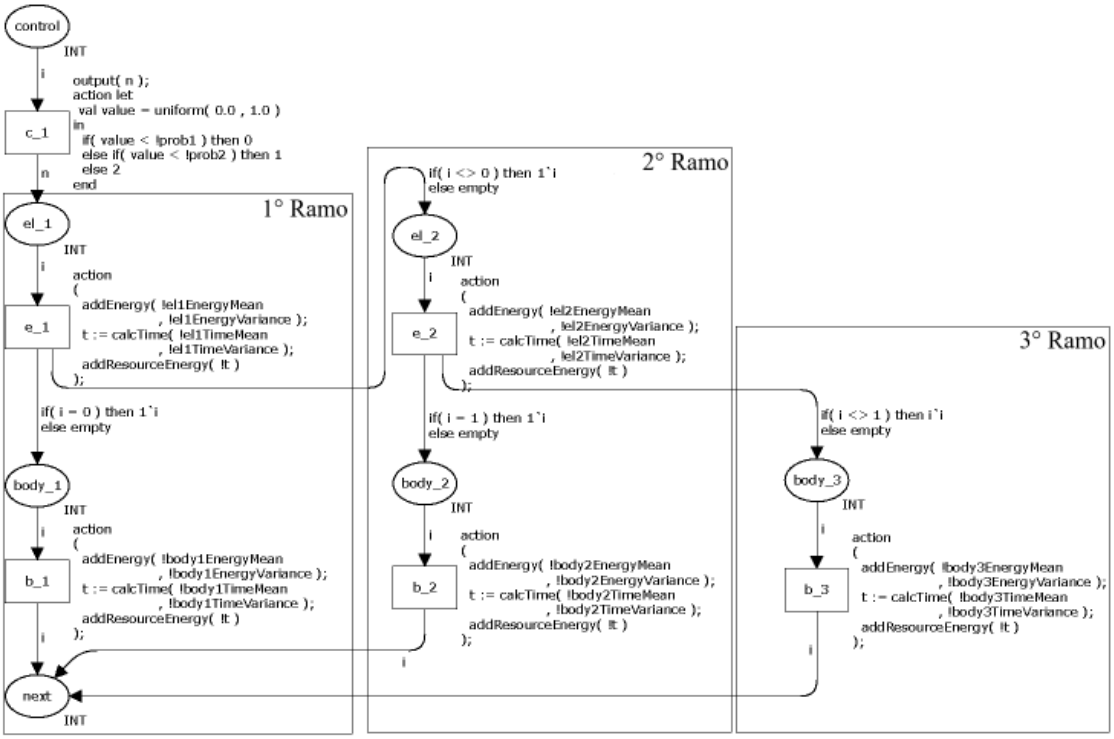


Figura 5.7 Modelo básico do if-then-else e do switch

Estrutura de Repetição

Devido às suas particularidades, cada estrutura de repetição (*while*, *do-while* e *for*) possui um modelo CPN diferente. A estrutura *while* tem um controle de iteração que (1) verifica se a expressão booleana é verdadeira e (2) depois executa o corpo (caso a expressão booleana seja igual a `true`). Quando o corpo termina, o controle de iteração verifica novamente a expressão. Esse ciclo termina quando a expressão booleana retorna `false`. Análogo à estrutura de seleção (*e.g.*, *if-then-else*), é necessário associar a probabilidade da expressão booleana ser igual a `true`, *i.e.*, a probabilidade do corpo ser executado. A probabilidade da expressão booleana é determinada pelo desenvolvedor manualmente usando o comentário `//@valor` próximo à estrutura de repetição *while* e *do-while*.

A Figura 5.8 mostra o modelo CPN para representar o *while*. Além de calcular o consumo de energia da expressão de controle, a transição *c_1* (similar ao adotado na estrutura de seleção) decide se executa o corpo (o *token* move do lugar *control* para *body*) ou não (o *token* move para o lugar *next*). O lugar *body* modela os operadores dentro do corpo, onde a transição *b_1* é responsável por calcular o consumo de energia deles.

De maneira similar à estrutura de seleção, este modelo pode ter transições mortas quando a probabilidade associada à expressão booleana for igual à zero.

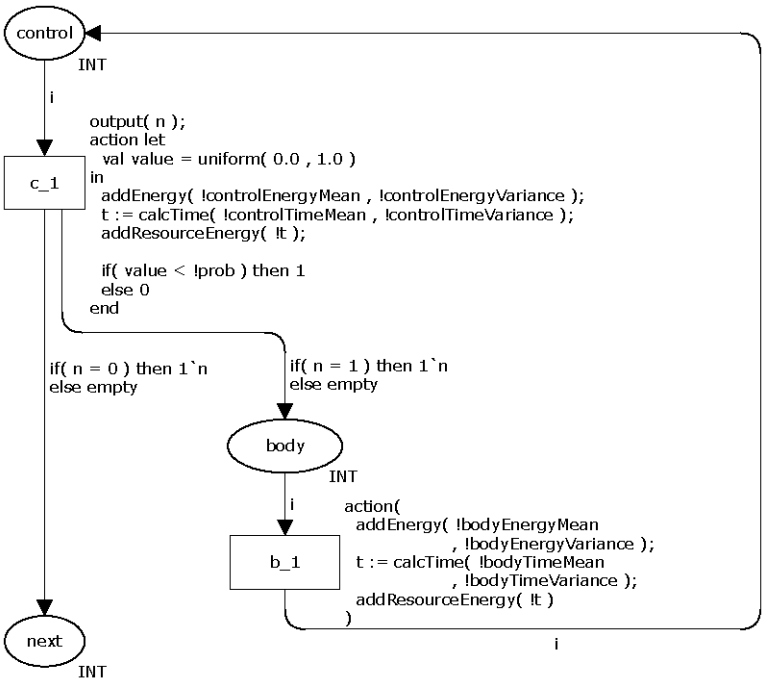


Figura 5.8 Modelo do *while*

O modelo CPN que representa o comando *do-while* é bastante similar ao comando *while*. A única e fundamental diferença entre eles é que o *do-while* garante que o corpo irá executar pelo menos uma única vez. Essa característica significa que o modelo não tem nenhuma transição morta. A Figura 5.9 mostra o modelo CPN proposto para essa estrutura de repetição.

Finalmente, a última estrutura de repetição, chamada *for*, consiste de quatro partes: inicialização, controle, passo e corpo. As primeiras três partes podem ser modeladas da mesma forma como um operador qualquer, como foi mostrado na Seção 5.2.1 por usarem, geralmente, um único operador (por exemplo, a inicialização usa o operador de atribuição; o controle, um operador de lógica; e o passo, o operador de incremento ou de decremento). Por sua vez, o corpo é similar às outras estruturas de repetição apresentadas anteriormente.

Entretanto, diferente das outras estruturas de repetição, é necessário definir o número de vezes que o corpo deve ser executado pelo controle de iteração (em vez de definir uma probabilidade do corpo ser executado), porque o *for* é usado para executar o corpo *n* vezes (veja a Seção 2.1.2). Adicionalmente, esse valor deve ser inteiro e deve ser definido manualmente pelo desenvolvedor usando o comentário `//@valor`.

A Figura 5.10 ilustra o modelo CPN proposto para representar o comando *for*. Os

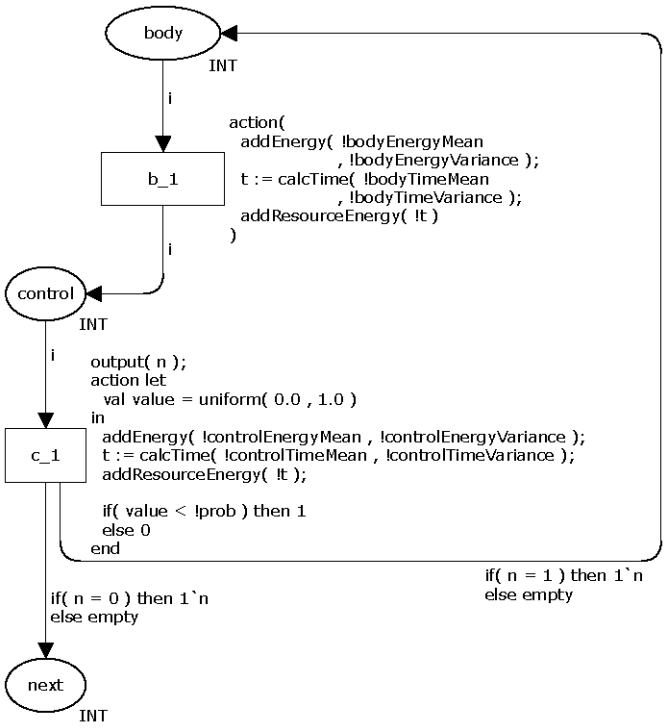


Figura 5.9 Modelo do *do-while*

lugares *assign*, *control*, *inc* e *body* estão relacionados com as partes de inicialização, controle, passo e corpo, respectivamente. A inicialização é executada apenas uma única vez e antes da repetição começar. A transição *c_1* decide o número de vezes (N) que irá repetir o corpo baseando-se no valor informado pelo desenvolvedor da aplicação (usando o comentário `//@valor`).

Adicionalmente, o modelo tem transições morta, caso o número de repetições associado ao comando seja igual à zero.

Operadores de invocação: `call`, `signal` e `post`

Como mencionado na seção 2.1.2, o operador de invocação `call` α é usado para executar um comando (α), o `signal` ε é usado para sinalizar um evento (ε), e o `post` Ω para executar posteriormente uma tarefa (Ω). Além disso, uma função pode habilitar/desabilitar um recurso do *hardware* (rádio ou LEDs), influenciando o consumo de energia da aplicação ao longo do tempo, ou, então, uma função pode sinalizar (após a sua execução) um evento (por exemplo, o comando `send` sinaliza, posteriormente, o evento `sendoDone`).

Devido a essas características, o modelo proposto desses operadores é similar visu-

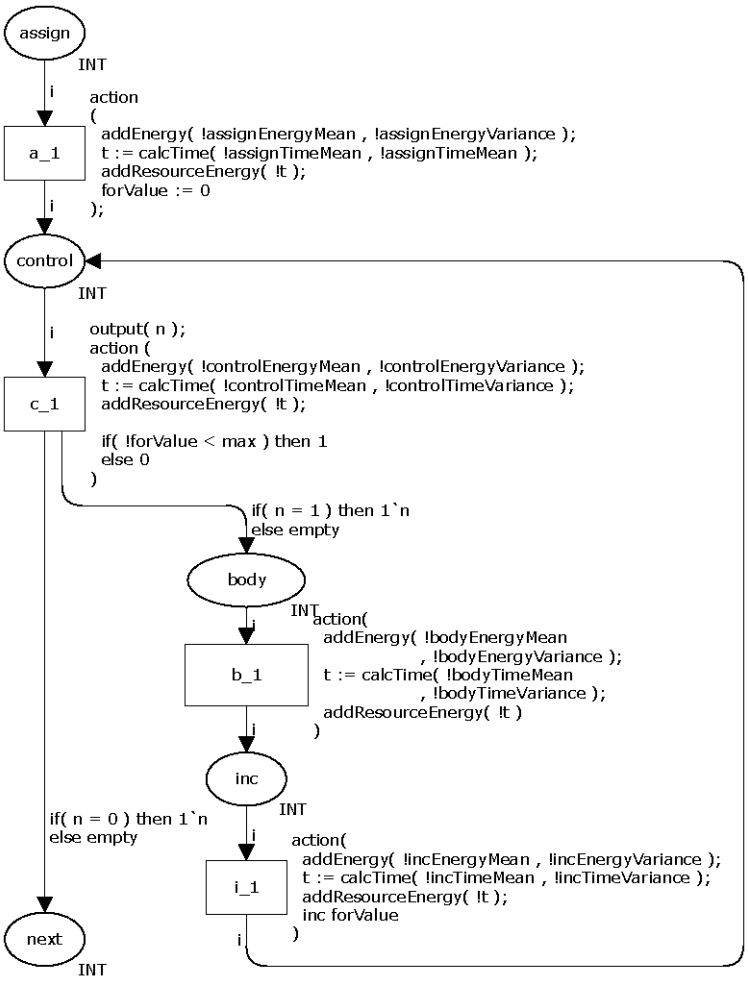


Figura 5.10 Modelo do *for*

almente ao modelo dos operadores apresentados na seção 5.2.1; no entanto, o modelo dos operadores de invocação possuem três particularidades que o diferem do modelo dos operadores apresentados anteriores. Primeiro, o consumo de energia desses operadores estão associados à função chamada, exceto o `post`, que irá apenas indica qual tarefa deve ser executada posteriormente. Segundo, o modelo de um operador de invocação pode acrescentar eventos ou tarefas na lista de funções (descrita na Seção 5.2.3). Por exemplo, no caso de um evento ε (*e.g.*, `sendDone()`) ser sinalizado depois da execução de uma função (*e.g.*, o comando `send()`), o evento ε deve ser acrescentando à lista de funções pelo Modelo de Função. Essa regra também serve para o modelo do operador `post`, que deve acrescentar uma tarefa Ω na lista de funções. Por último, o Modelo de Função deve indicar se habilita ou desabilita um recurso do *hardware* (rádio ou LEDs). No entanto, a segunda e a terceira peculiaridades são opcionais, porque dependem da função modelada.

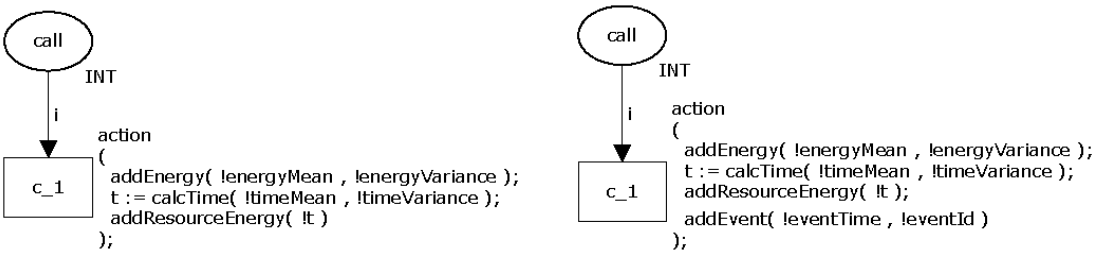


Figura 5.11 Modelo dos operadores *call*, *signal* e *post*

A Figura 5.11 mostra dois modelos CPNs representando uma chamada de comando. Os dois modelos são semelhantes ao modelo apresentado na Figura 5.6. O primeiro modelo (a) representa uma invocação a uma função qualquer, enquanto o segundo modelo (b) representa uma situação em que um evento ou tarefa será executada posteriormente. Neste último caso, a função `addFuction()` é responsável por acrescentar uma função (evento ou tarefa) dentro da lista de funções. Para isso, ela recebe dois parâmetros: o tempo (*functionTime*) e o identificador (*functionId*) da função.

5.2.2 Modelo de Função

Antes de apresentar o Modelo de Função, é necessário relembrar que a implementação de uma aplicação nesC consiste em um conjunto de funções chamadas de tarefas, comandos e eventos (veja Seção 2.1.2). Elas são implementadas de forma similar às funções da linguagem C. Uma função é um bloco que contém um conjunto de operadores e estruturas (tais como atribuição, repetição e seleção), os quais são representados por Modelos dos Operadores, como foi ilustrado na seção anterior, portanto um modelo de função consiste

de um conjunto de Modelos dos Operadores que são colocados juntos seguindo uma regra. Para o melhor entendimento, considere a seguinte função (um comando) nesC:

```
command void Modulo.read(){
  SLEEP_TIME = 100;  //assign
  call Temp.read();  //invocation
}
```

Esta função possui dois elementos, atribuição (=) e invocação (`call`), que são executados sequencialmente. Desse modo, o respectivo modelo dessa função é simplesmente a composição dos Modelos dos Operadores (MOs) desses dois operadores, como mostra a Figura 5.12.

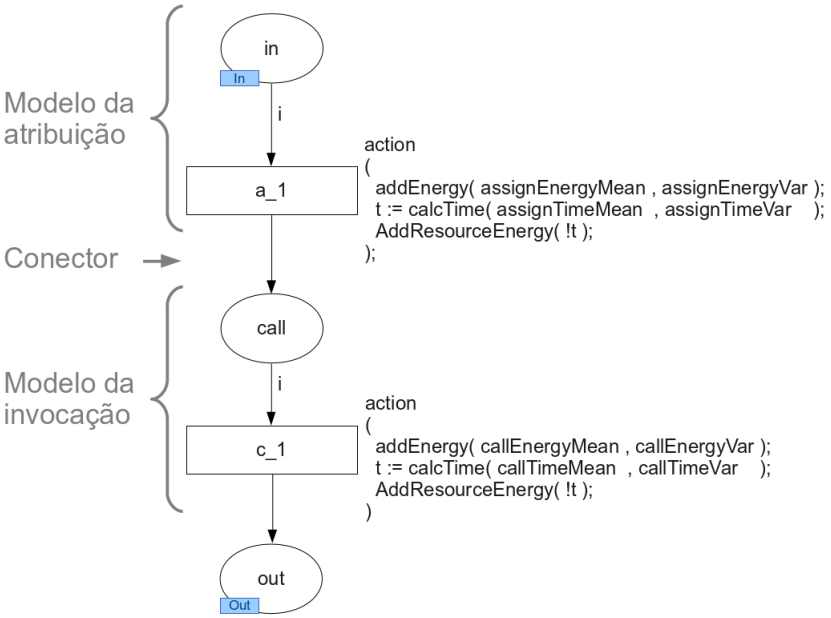


Figura 5.12 Modelo de uma função

O operador de atribuição é modelado através do lugar *in* e da transição *a_1*; enquanto o operador de invocação é modelado pelo lugar *call* e pela transição *c_1* (veja a Seção 5.2.1). Além disso, os lugares *in* e *out* representam o início e o fim da função e são portas de entrada e saída, respectivamente (veja a Seção 2.2). Eles são usados para conectar com o Modelo Principal (veja a Seção 5.2.3).

Esses MOs são conectados de acordo com a seguinte regra: se uma função possuir *n* operadores, o modelo dessa função será formado por *n* MOs. A transição do primeiro MO é conectada com o lugar do segundo MO; a transição do segundo MO é conectada ao lugar do terceiro MO e, assim sucessivamente até o último MO. Por ser o último modelo, a transição do enésimo MO irá se conectar com o lugar *out*, representando o fim

da modelagem da função. Por exemplo, o modelo do comando *Modulo.read()* possui dois MOs: o primeiro representando a atribuição e o segundo, a invocação. A transição do primeiro MO (*a_1*) é conectada com o lugar (*call*) do segundo MO que, por sua vez, a sua transição irá se conectar com o lugar *out*, visto que o segundo MO também é o último MO desta função.

No caso das estruturas seleção e repetição, foi colocado um lugar *next*, nos respectivos modelos, para ilustrar o ponto de conexão entre ele e o MO subsequente. Por exemplo, se uma função tiver dois operadores (o primeiro é uma estrutura de seleção e o segundo for uma atribuição), o lugar *next* do modelo da estrutura de seleção representa o lugar do MO da atribuição.

evento `receive`

Geralmente, os eventos na linguagem nesC representam uma resposta para a execução de um comando. Por exemplo, quando uma aplicação executa o comando `start` para habilitar o rádio, ele descobrirá se o rádio foi ligado através do evento `startDone`. Da mesma forma, quando a aplicação deseja coletar a temperatura. Ele executa o comando *read* e recebe o valor através do evento *readDone*. Diferente dos eventos anteriores, existem eventos que são executados quando um fator externo ocorre. Um exemplo desses eventos é o *receive*, que sinaliza quando o nó sensor recebe um pacote. Caso o Modelo de Aplicação seja avaliado sem o Modelo de Rede, é necessário o desenvolvedor da aplicação associar uma probabilidade (usando o comentário `//@valor`) que indica a chance desse evento ocorrer. A Figura 5.13 apresenta o modelo CPN, chamado de módulo *receive*, desse evento.

O módulo *receive* tem uma transição chamada *check* para decidir se a aplicação recebeu uma mensagem, gerando um número randômico entre 0 e 1 seguindo a distribuição uniforme visto que ele gera números finitos entre dois números com chances iguais de acontecer. Se o valor gerado é menor ou igual à probabilidade associada ao evento, o *token* move do lugar *in* para a transição *receive*. Caso contrário, o *token* move para o lugar *out*. Tal como no Modelo de Função, os lugares *in* e *out* são portas de entrada e saída, respectivamente (veja a Seção 2.2) usados para conectá-los ao Modelo Principal. A transição *body* modela o(s) comando(s) internos do evento. O modelo CPN dos outros eventos também possui os lugares (*in* e *out*) e a transição *body*.

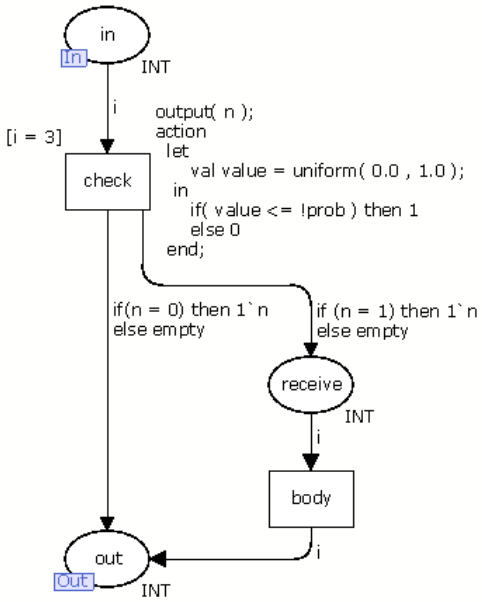


Figura 5.13 Modelo do evento *receive*

5.2.3 Modelo Principal

Para entender como o Modelo de Aplicação é obtido, é importante observar que as aplicações nesC são baseadas em eventos, nos quais a ordem de execução é apenas definida em tempo de execução devido às tarefas e aos eventos. Por este motivo, um elemento chamado *scheduler* foi introduzido no modelo para tratar explicitamente a execução das tarefas e eventos, servindo como um elemento chave para o Modelo de Aplicação. Essas características fazem o Modelo Principal genérico, permitindo representar qualquer aplicação escrita na linguagem nesC independente da plataforma do nó sensor (*e.g.*, IRIS ou MICAz).

Cada Modelo de Aplicação é organizado em módulos CPNs (veja a Seção 2.2) e cada módulo representa uma função em nesC (tarefa, comando ou evento). Eventos que dependem da ocorrência de um fator externo possuem uma probabilidade que deve ser definida pelo desenvolvedor da aplicação. Atualmente, o único evento que depende de fator externo é a recepção de pacotes (veja a Seção 5.2.2).

A Figura 5.14 mostra o esquema do Modelo de Aplicação.

O *scheduler* tem os seguintes elementos básicos: uma fila de função, sete transições (*scheduler*, *start*, *end*, *sleep*, *boot*, *receive*, *nl*) e dois lugares (*in*, *out*). As tarefas e eventos são armazenados em uma fila, onde cada uma possui um identificador e um tempo associado. Desta maneira, uma tarefa ou um evento é removido da fila de acordo com o

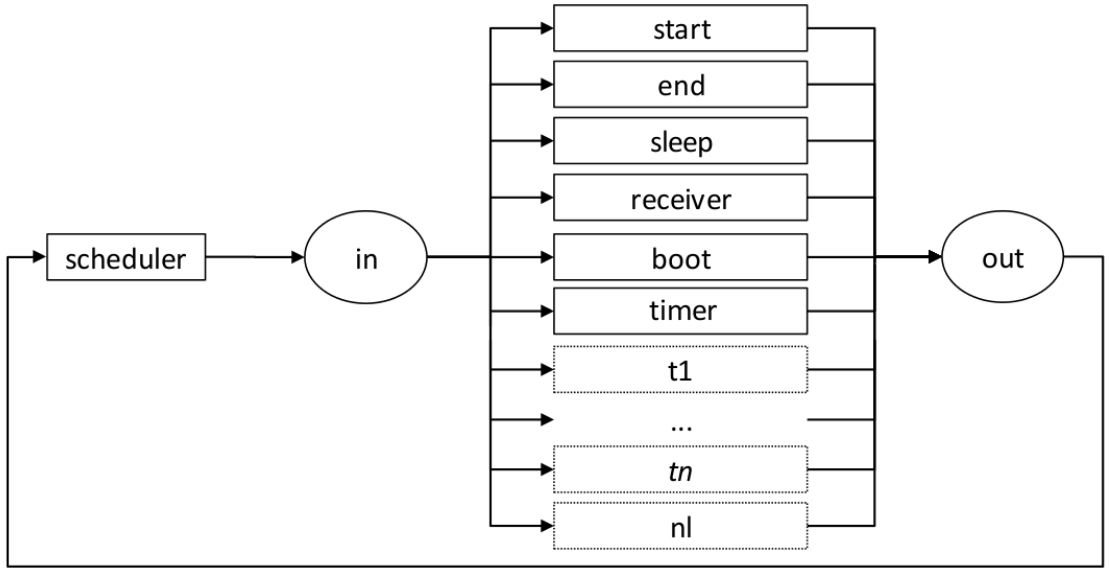


Figura 5.14 Visualização do esquema do modelo principal da aplicação

seu tempo (por exemplo, as funções armazenadas com menor tempo são as próximas a serem executadas). Se existem duas funções diferentes com o mesmo tempo, a função a ser executada será a primeira a ser armazenada (FIFO) tal como o TinyOS (veja a Seção 2.1.2).

A transição *scheduler* atua como um orquestrador que define qual será a próxima transição (ou modulo) a ser executada de acordo com o tempo associado à tarefa ou evento. Adicionalmente, a tarefa ou o evento que é tipo código assíncrono (AC) será selecionado pela transição *scheduler* e ele não poderá parar a execução de uma código síncrono (SC) (veja a Seção 2.1.2). Em outras palavras, uma AC será executada como uma SC (veja a Seção 2.1.2).

Os lugares *in* e *out* são *sockets* e identificam o começo e o fim de uma atividade, respectivamente. Esses lugares são conectados aos lugares *in* e *out* presentes no Modelo de Função e no modelo do evento *receiver*. Ele *in* também é o ponto inicial do modelo, indicando qual é a primeira transição a ser executada (as demais decisões são feitas pela transição *scheduler*). A combinação da transição *scheduler* (que decide qual transição deve ser executada) e o lugar *in* (que inicia uma atividade) torna o modelo reversível, ou seja, o modelo sempre retorna ao seu ponto inicial.

No caso da lista está vazia, ou seja, nenhuma função para ser executada, a transição *sleep* é selecionada, indicando que a aplicação não está executando nenhuma função. A transição *start* é executada apenas uma única vez. Ela inicia todas as variáveis e coloca a primeira função (para executar a transição relacionada à inicialização do nó sensor)

dentro da fila de função. A transição *end* indica o término da execução da aplicação e também é executada apenas uma única vez.

As transições *boot*, *timer*, *t1*, and *tn* dependem do comportamento da aplicação. Essas transições correspondem às tarefas ou eventos implementados na aplicação e cada transição serve como uma ponte entre o módulo *scheduler* e o respectivo Modelo de Função. Por exemplo, as transições *boot*, *timer* e *receive* representam, respectivamente, o módulo *boot* (responsável pela inicialização da aplicação), *timer* (responsável por executar uma atividade periodicamente) e o *receive* (responsável por tratar um pacote de rede quando recebê-lo). As transições *t1* e *tn* representam outras funções (tarefas ou eventos) implementadas na aplicação.

Adicionalmente, a integração do Modelo de Aplicação com o Modelo de Rede interfere em duas transições, *ln* e *receive*, no Modelo Principal. A transição *ln* representa a camada de rede e serve como ponte entre a camada de aplicação e a camada de rede. Essa transição só é necessária quando existir a integração entre os dois modelos uma vez que a camada de rede é de responsabilidade do Modelo de Rede.

A transição *receive* pode assumir dois comportamentos. O primeiro comportamento é tratar essa transição como qualquer outra dentro desse módulo (por exemplo, *boot*, *timer*, *t1*): ela poderá ser disparada quando o seu identificador for inserido na lista de função. Nesse caso, é necessário que o Modelo de Aplicação esteja integrado com o Modelo de Rede, responsável por inserir o identificador dessa transição na lista de função (veja a Seção 5.3.1). O segundo comportamento é tratá-la *receive* diferente das outras: o desenvolvedor terá que associar uma probabilidade dessa transição ser disparada (como explica a Seção 5.2.2); *i.e.*, havendo, portanto, uma probabilidade do nó sensor receber um pacote de outro nó sensor. Isso possibilita duas coisas: (1°) o Modelo de Aplicação avalia o consumo de energia de todos as funções (incluindo um evento que é disparado por um fator externo) e (2°) o Modelo de Aplicação possa ser simulado sem estar integrado com o Modelo de Rede (veja a Seção 5.3.4), avaliando o consumo de energia apenas da aplicação. Além disso, o segundo comportamento interfere no funcionamento da transição *scheduler*: ora o *scheduler* seleciona uma função (tarefa ou evento) da fila, ora ele verifica se recebeu um pacote (disparar a transição *receive*), observando, por fim, que o rádio deve estar habilitado para disparar a transição *receive*.

5.3 Modelo do consumo de energia da rede e do nó sensor

As próximas subseções apresentam detalhadamente os modelos relacionados ao modelo de Rede e ao Modelo de Nó Sensor. O Modelo de Rede é responsável por avaliar o consumo de energia dos protocolos e dos nós sensores presentes na RSSF. Esse modelo é formado por dois modelos de protocolos da camada de rede e de enlace, os quais formam a pilha de protocolo; e por um Modelo de Ambiente, o qual é responsável por identificar e repassar os pacotes para os nós sensores próximos. Por fim, o Modelo de Nó Sensor é responsável por avaliar o consumo de energia da aplicação e da rede juntos.

5.3.1 Modelo dos protocolos da camada da rede

Esta tese não tem um modelo de consumo de energia genérico que pode ser usado para representar diferentes protocolos da camada de rede, visto que existe uma grande variedade de protocolos e características a serem modeladas. Então, foi criado um padrão (veja a Figura 5.15) contendo os elementos básicos.

Esse padrão estabelece que qualquer modelo de protocolo da camada de rede deve incluir dois lugares, *in* e *out*, para interagir com a aplicação (veja a Seção 5.2) e uma transição, *LinkLayer*, que atua como uma ponte para a camada de enlace. Os lugares *in* e *out* são conectados à camada de aplicação através dos lugares *in* e *out* da Figura 5.14, respectivamente.

A transição *AppPckt* representa a criação de um pacote pela aplicação, *e.g.*, quando a ela coleta a temperatura e precisa enviá-la ao nó sorvedouro, enquanto a transição *CtrlPckt* modela a criação de pacotes próprios do protocolo, *e.g.*, uma fase de inicialização ou configuração do protocolo. A transição *select* é responsável por decidir qual será o próximo nó que o pacote deve ser enviado. Essa decisão é tomada usando uma lista de nós (lugar *NodeLst*). É importante observar que esse padrão não específica como esses caminhos são definidos, pois isto vai depender do algoritmo de roteamento adotado pelo protocolo.

Quando o modelo do protocolo dessa camada recebe alguma informação (*e.g.*, temperatura coletada) vinda da camada da aplicação no lugar *in*, ele seleciona o próximo nó sensor e cria dois *tokens*: um para a aplicação (lugar *out*), notificando a transição *scheduler* do Modelo Principal (veja a Figura 5.14) para executar outra transição (outra tarefa ou evento) e o outro *token*, para o modelo da camada de enlace, indicando que um

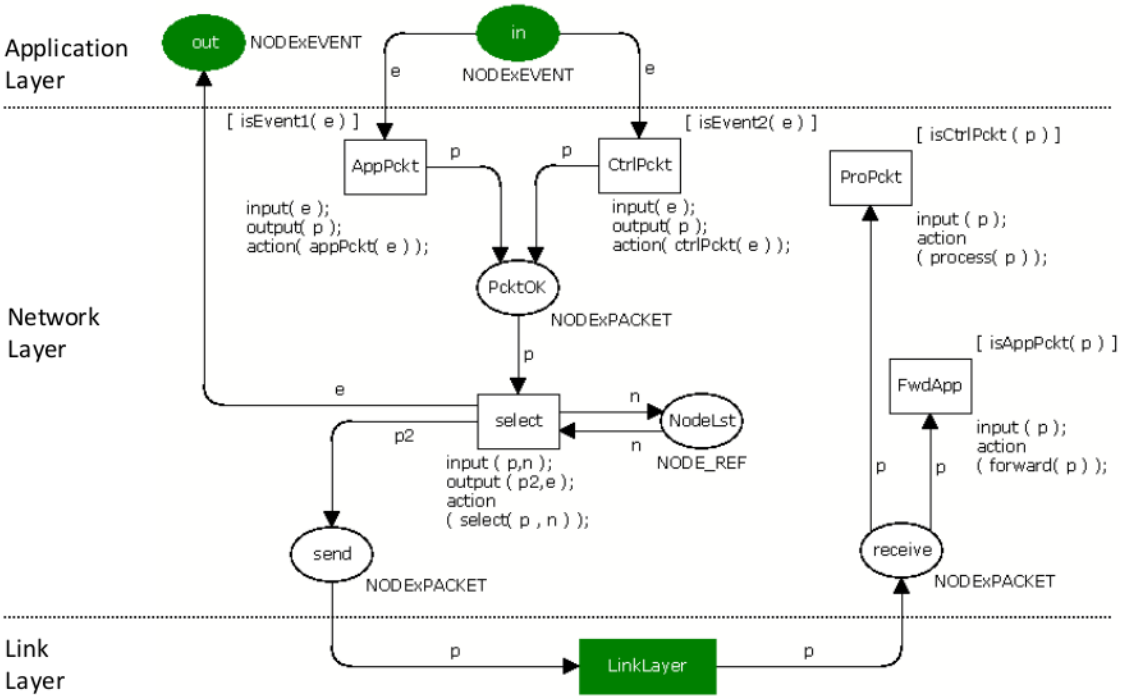


Figura 5.15 Visualização do padrão da camada de rede

pacote deve ser enviado pela rede (lugar *send*).

Quando o modelo do protocolo recebe um quadro da camada de enlace através da transição *LinkLayer*, ele verifica quem o criou. Se o quadro foi criado pela transição *AppPckt* (*i.e.*, um pacote da camada da aplicação), o modelo do protocolo dispara a transição *FwdApp* que adiciona um elemento na lista de funções (Figura 5.14), indicando que a transição *receive* deve ser executada. No entanto, se o pacote foi criado pela transição *CtrlPckt*, o modelo do protocolo dispara a transição *ProPckt* para processá-lo. Adicionalmente, o *token* é um tipo complexo para representar um quadro no modelo. Esse tipo, denominado *NODExPACKET*, possui as informações de no nó sensor que o criou, do tamanho do quadro, qual nó sensor representa e assim por diante. Essas informações são atribuídas pelas transições *AppPckt* e *SinkPckt*; as demais transições apenas processam o *token*.

Portanto, o Modelo de Rede pode usar qualquer modelo de energia. Esta tese usou o *First Order Radio Model* (Heinzelman *et al.*, 2000) por sua simplicidade e por ser amplamente utilizado em RSSFs, pois considera o consumo de energia quando o nó sensor envia e recebe pacotes; ignorando o consumo de energia do processamento do pacote. Desta maneira, o consumo de energia do Modelo de Rede é contabilizado na camada de enlace (veja a Seção 5.3.2), a qual é responsável por enviar e receber pacotes

para/do ambiente. As outras camadas não existe o consumo de energia por processar os pacotes, o qual é ignorado pelo *First Order Radio Model*.

Protocolo DIRECT

O protocolo DIRECT (Senouci *et al.*, 2012) é ideal para redes pequenas, nas quais os nós sensores têm comunicação direta (único salto) com o nó sorvedouro. Nesse caso, este protocolo não necessita rotear os dados através da rede. A Figura 5.16 ilustra o modelo do protocolo DIRECT. É importante observar, nesse caso, que a lista de nós sensores (o lugar *NodeLst*) não é necessária, já que os mesmos têm comunicação direta.

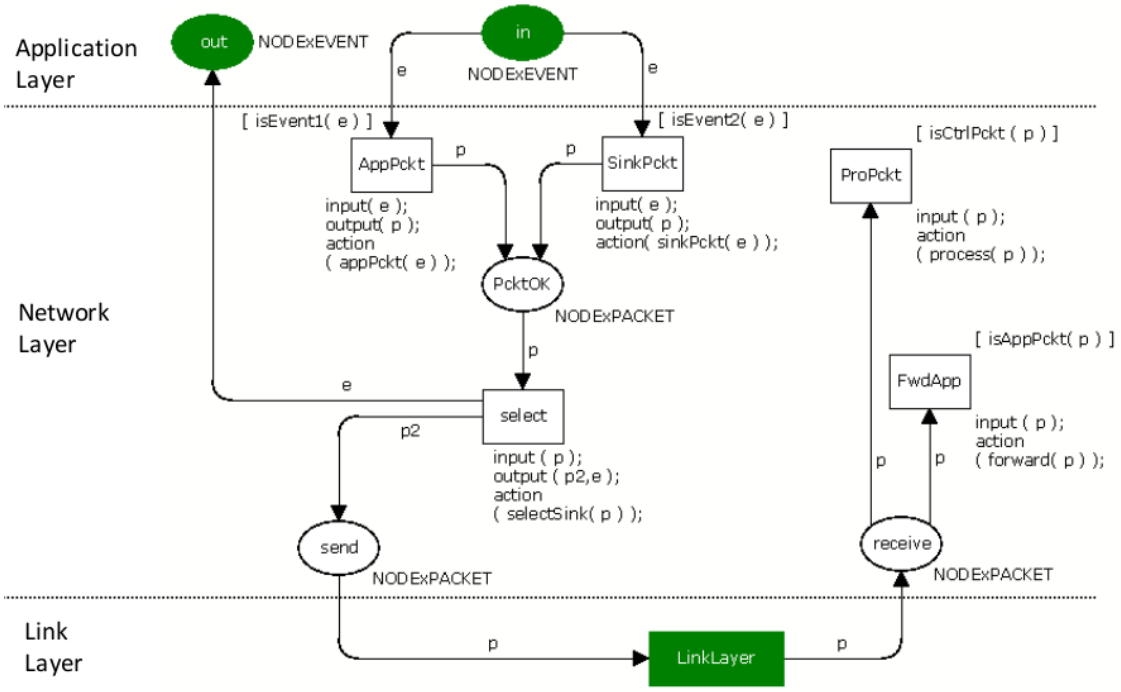


Figura 5.16 Modelo do protocolo DIRECT seguindo o padrão da camada de rede

Na fase de inicialização do protocolo, o nó sorvedouro envia uma mensagem (*SinkPckt*) para todos os nós sensores. Cada nó sensor, ao receber esse pacote, calcula a distância entre ele e o nó sorvedouro e modifica a potência do rádio para o mínimo necessário que é exigido para haver a comunicação entre eles. Essa ação é representada pela transição *ProPckt*. Depois da inicialização, as aplicações dos nós sensores começam a enviar os pacotes (disparando a transição *AppPckt*) para o nó sorvedouro (quando recebe um pacote, notifica sua aplicação através da transição *FwdApp*).

Protocolo FLOODING

Um nó sensor usando o protocolo FLOODING (Ko *et al.*, 2004) envia um pacote in *broadcast*, em que seus vizinhos recebem o pacote e o repassa (também em *broadcast*) até o pacote alcançar o destinatário. Essa característica não precisa preparar e armazenar os caminhos até o destinatário. Então, o modelo do FLOODING (ilustrado na Figura 5.17) também não precisa do lugar *Nodelst*. Nesse caso, a transição *select* apenas atribui o valor *broadcast* para o pacote criado.

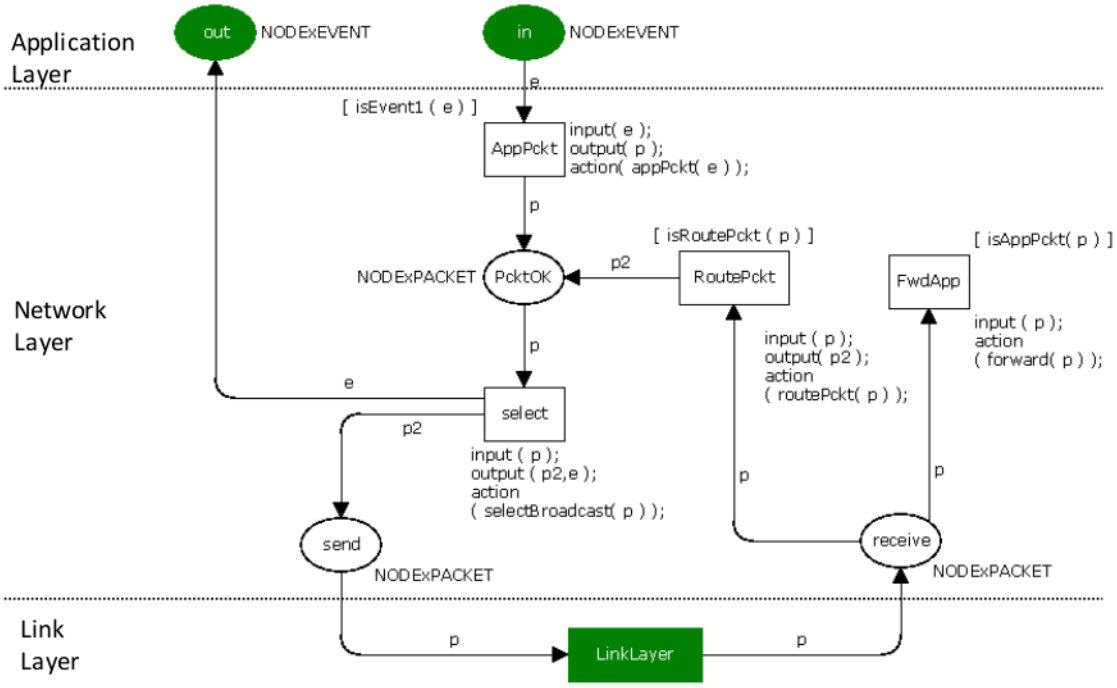


Figura 5.17 Modelo do protocolo FLOODING seguindo o padrão da camada de rede

Quando um nó sensor recebe o pacote, ele verifica se o pacote é destinado a ele ou não. Se for, ele repassa o pacote para a aplicação (transição *FwdApp*). Caso contrário, ele roteia o pacote para seus vizinhos (transição *RoutePckt*).

Protocolo GOSSIPING

O protocolo GOSSIPING (Hedetniemi *et al.*, 1988), diferentemente do protocolo FLOODING, escolhe um nó sensor randomicamente e repassa o pacote para o nó sensor escolhido (como ilustrado na Figura 5.18). Na fase de configuração do protocolo, todos os nós sensores devem enviar um sinal (transição *SigPckt*) em *broadcast*. Quando um nó sensor recebe esse pacote, ele calcula a distância entre ele e o transmissor e muda o valor

da lista de nós (lugar *NodeLst*).

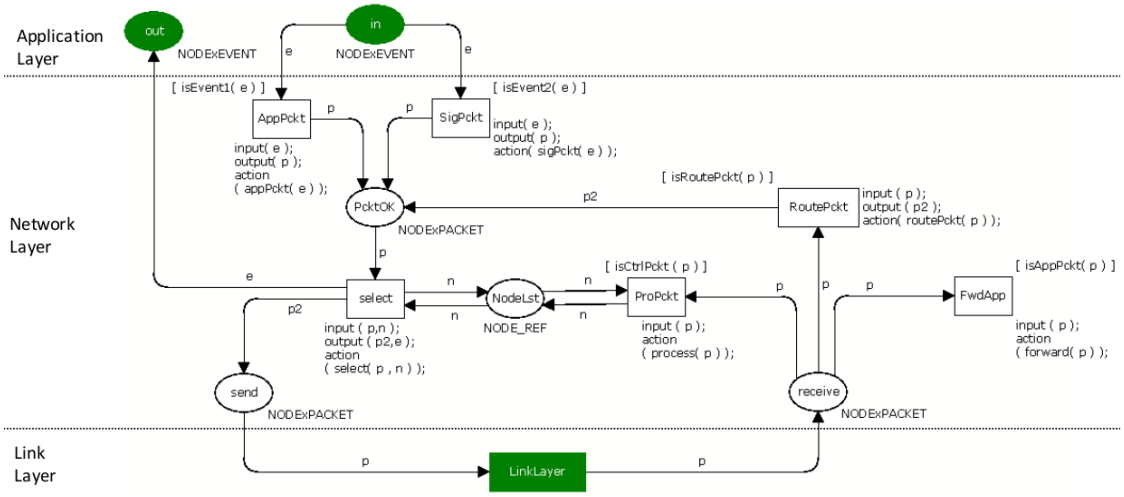


Figura 5.18 Modelo do protocolo GOSSIPING seguindo o padrão da camada de rede

Na fase de execução do protocolo, quando o nó sensor cria um pacote da aplicação (transição *AppPckt*), ele seleciona (transição *select*) outro nó sensor e muda a potência do rádio para um valor mínimo necessário para alcançar o receptor. Quando o nó sensor recebe esse pacote, ele verifica quem é o destinatário. Caso o destinatário seja ele, o pacote é repassado para a camada de aplicação (transição *FwdApp*). Caso contrário, o pacote é enviado para outro nó sensor (transição *RoutePckt*), o qual é selecionado randomicamente (tal como o nó sensor remetente fez).

Protocolo LEACH

O protocolo LEACH (*Low-Energy Adaptive Clustering Hierarchy*) (Heinzelman, 2000; Heinzelman *et al.*, 2000) é um dos mais usados para roteamento hierárquico da RSSF. Esse protocolo dinamicamente cria *clusters* dentro da rede e os nós sensores usam os líderes dos *clusters* (CHs) para rotear os seus pacotes até o nó sorvedouro. De tempos em tempos, um novo nó sensor se torna um líder de um *cluster*. Essa estratégia reduz a dissipação de energia entre os nós sensores, incrementando o tempo de vida da rede. Adicionalmente, esse protocolo assume que todos os nós sensores podem se comunicar com o nó sensor diretamente. Essas características são modeladas na Figura 5.19.

Esse protocolo divide suas atividades em duas fases: fase de configuração e fase estável. A fase de configuração é usada para criar os *clusters*. Nessa fase, todos os nós sensores geram um número randômico entre 0 e 1 para determinar se será um líder de um *cluster* ou não. Se o valor gerado for menor do que a entrada *T(n)*, ele se torna um líder de

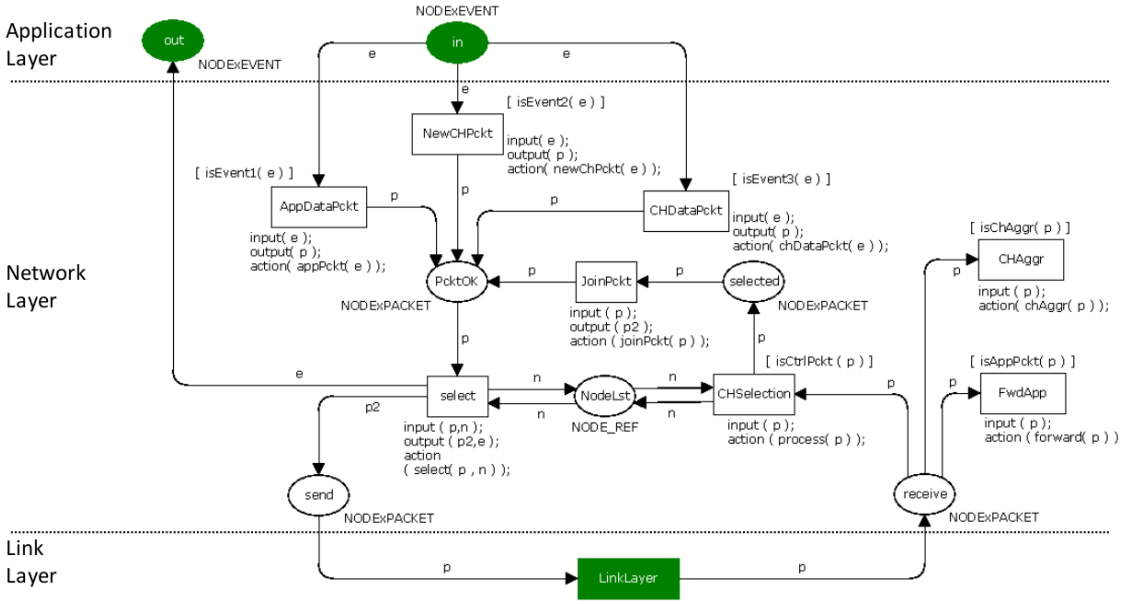


Figura 5.19 Modelo do protocolo LEACH seguindo o padrão da camada de rede

um *cluster* e notifica os demais (transição *NewCHPckt*). Quando os nós sensores recebem essa notificação, eles verificam qual é o líder mais próximo (transição *CHSelection*) e enviam um pacote para o líder escolhido para participar do seu *cluster* (transição *JoinPckt*). No próximo passo (fase estável), os nós sensores enviam periodicamente um pacote para o líder do *cluster* (transição *AppDataPckt*), o qual agrega os dados recebidos (transição *CHAggr*) e os envia para o nó sorvedouro (transição *CHDataPckt*). Após um determinado tempo definido pelo usuário, a fase de configuração começa novamente (transição *NewCHPckt*), criando novos *clusters*. Dessa forma, o LEACH elege novos nós sensores para serem líderes de *clusters* em tempo em tempo.

5.3.2 Modelo dos protocolos da camada de enlace

De maneira similar à camada de rede, esta seção apresenta um padrão (ilustrado na Figura 5.20) a ser usado como base para um protocolo da camada de enlace, especialmente para os protocolos MAC. Esse padrão possui dois lugares (*NetSend* e *NetReceive*) para conectar com o modelo da camada de rede e uma transição (*env*) atuando como uma ponte com o Modelo de Ambiente. O ambiente modela tudo entre o nó transmissor e o receptor, enquanto que as transições *send* e *receive* representam as ações de enviar e receber um pacote para/de o ambiente, respectivamente.

Quando a transição *send* é disparada, o consumo de energia é calculado pela função

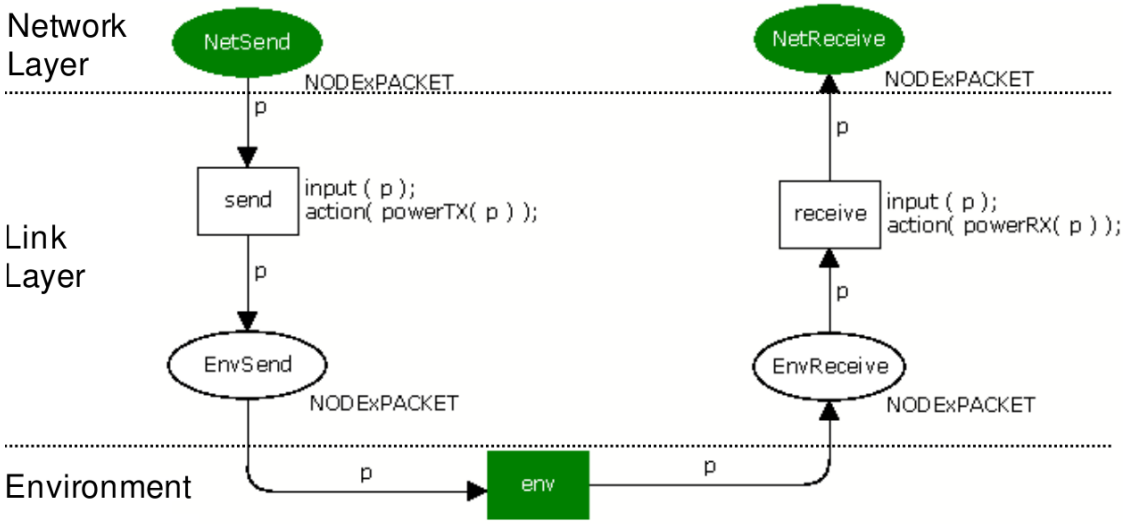


Figura 5.20 Visualização do padrão da camada de enlace

`powerTX()`. Seu pseudocódigo[1] é mostrado a seguir:

```
1 function powerTX( NodexPacket np ){
2    double e1 = eTX * np.getPacketSize();
3    double e2 = eAMP * np.getPacketSize() * np.getRadioRange()
                      * np.getRadioRange();
4    double e = e1 + e2;
5    long n = np.getNodeId();
6    addPowerConsumption( n , e );
7 }
```

Esta função implementa o código apresentado por Heinzelman *et al.* (2000). O consumo de energia de enviar um pacote (e) é a soma do consumo de energia do transmissor eletrônico, responsável por transmitir o pacote, (linha 2) e o consumo de energia do amplificador de transmissão, responsável por aumentar a força de transmissão (linha 3). Além disso, a função `addPowerConsumption()` incrementa *e* o consumo de energia do nó *n*.

Similarmente, quando a transição *receiver* é disparada, o consumo de energia de receber um pacote é contabilizado pela função `powerRX()`. Essa função considera que o consumo de energia para receber um pacote (linha 1) é igual ao consumo de energia do receptor eletrônico (linha 2), como mostrado a seguir:

```
1 function powerRX( NodexPacket np ){
2    double e = EnergyRX * np.getPacketSize();
3    long n = np.getNodeId();
```

[1]Devido à complexidade da sintaxe da CPN ML, foi adotado um pseudocódigo para facilitar o entendimento da função

```
4     addPowerConsumption( n , e );
5 }
```

Finalmente, esse padrão serve como um modelo de consumo de energia de uma camada MAC sem colisões e sem *overhearing*, *i.e.*, um pacote está sempre pronto para ser enviado e todos os pacotes são processados.

Protocolo B-MAC

O protocolo B-MAC (*Berkeley Media Access Control*) (Polastre *et al.*, 2004) é um dos mais conhecidos protocolos MAC. Ele utiliza o CSMA (*Carrier Sense Multiple Access*) para evitar colisões, ilustrado na Figura 5.21. Esse protocolo verifica (transição *ChckEnv*) se o ambiente está livre, ou não, antes de enviar um pacote. Se o ambiente estiver ocupado (lugar *busy*), o nó sensor espera um tempo randômico (transição *wait*) antes de tentar enviar novamente o pacote. Se nenhum nó sensor estiver transmitindo (lugar *free*), ele envia o pacote (transição *send*).

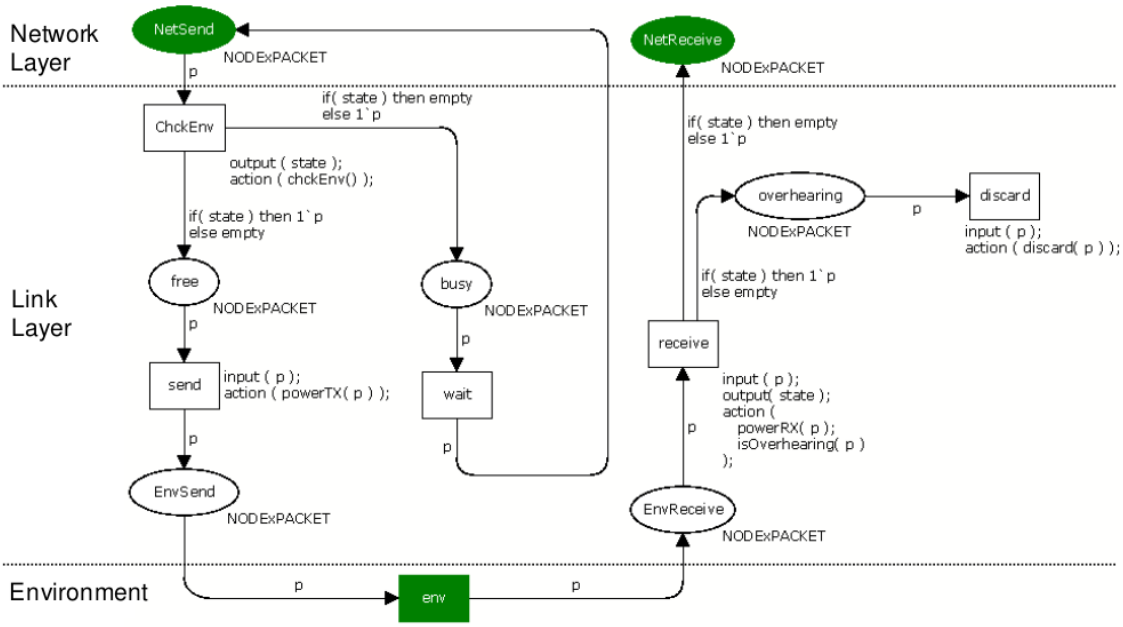


Figura 5.21 Modelo do protocolo B-MAC seguindo o padrão da camada de enlace

Quando um nó recebe um pacote (transição *receive*), ele verifica quem é o destinatário. Se o destinatário for ele, ou caso o pacote tenha sido enviado em *broadcast*, o pacote é repassado para a camada acima (lugar *NetReceive*). Caso contrário, ele considera como um *overhearing* (lugar *overhearing*) e ignora o pacote (transição *discard*).

5.3.3 Modelo de Ambiente

O Modelo de Ambiente modela tudo (no meio) entre o nó transmissor e receptor. Ele identifica (em tempo de execução) as conexões entre os nós sensores e repassa os pacotes de um nó sensor (origem) a outro (destino). Na Figura 5.22, os lugares *send* e *receive* representam o momento em que um nó envia e recebe um pacote, respectivamente. Como já mencionado anteriormente, a transição *env* modela tudo entre os nós.

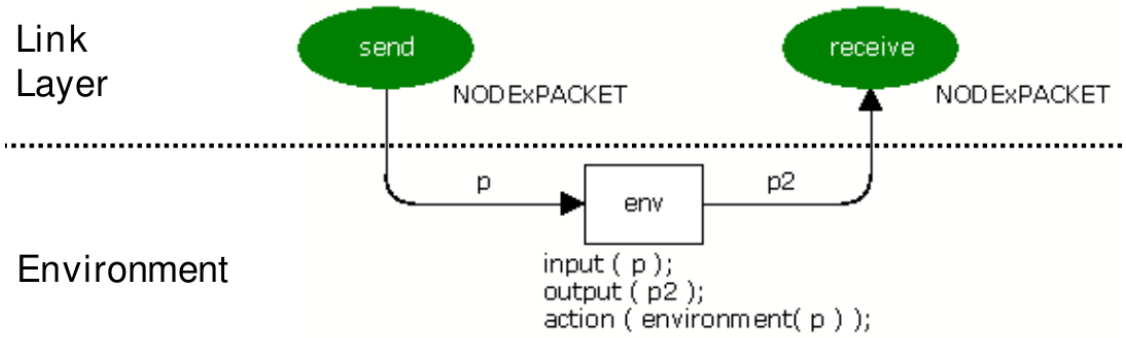


Figura 5.22 Visualização do Modelo de Ambiente

Adicionalmente, o Apêndice E mostra os modelos de propagação do sinal existentes e qual desses modelos o Modelo de Ambiente utilizou para identificar as conexões existentes entre os nós sensores.

5.3.4 Modelo do consumo de energia do Nó Sensor

O consumo de energia do nó da RSSF é obtido através da combinação dos Modelos da Aplicação e da Rede. Devido ao tamanho e à complexidade, é inviável simular essa combinação. Para resolver esse problema, é necessário, inicialmente, simular o Modelo de Aplicação separadamente e obter o consumo de energia e a frequência de envio de quadros. Os valores obtidos através da simulação são usados pelo Modelo Simplificado (ilustrado na Figura 5.23), o qual representa o Modelo de Aplicação na combinação. A transição *PwrConsumption* modela o consumo de energia de toda a aplicação. Dessa maneira, o Modelo de Nó Sensor considera aspectos da aplicação sem mudar a sua estrutura e seu comportamento.

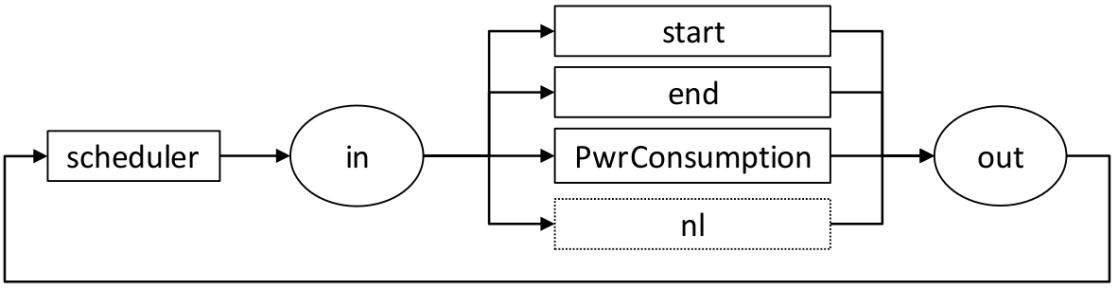


Figura 5.23 Modelo simplificado representando o consumo de energia da aplicação

5.4 Modelo da confiabilidade da região

O Modelo de Região avalia a confiabilidade de uma região na rede em um determinado tempo. Primeiramente, esta seção apresenta os blocos básicos do RBD, os quais representam os nós sensores e os enlaces de comunicação. Em seguida, ela mostrará o modelo de confiabilidade de um caminho, percurso entre um nó sensor e o nó sorvedouro. Por fim, descreverá o modelo de confiabilidade da região e, logo em seguida, mostra alguns exemplos criados de acordo com protocolo de roteamento utilizado.

5.4.1 Blocos Básicos

Os pontos de falha na RSSF são o enlace de comunicação e o nó sensor e, por este motivo, esses dois pontos são modelados como blocos do RBD (chamados de Blocos Básicos), como ilustra a Figura 5.24. As falhas permanentes (compromete o funcionamento permanentemente) são as únicas falhas consideradas nesse estudo. As falhas transientes (compromete o funcionamento temporariamente) devem ser tratadas como falhas permanentes ou devem ser ignoradas. Por exemplo, a interferência na comunicação pode ser considerada como falha transiente no enlace de comunicação. Neste estudo, ela deve ser considerada como falha permanente ou devem ser ignoradas. As falhas transientes sendo consideradas como falhas permanentes representam o pior cenário a ser avaliado. Mesmo não representando a realidade, caso o usuário esteja satisfeito com os resultados da confiabilidade da RSSF, ele terá uma rede mais preparada visto que foi avaliada considerando o pior cenário.

Em particular, o bloco relacionado ao nó sensor é formado pela seguinte sequência de blocos: aplicação (App), sistema operacional (SO), middleware (Middleware), plataforma (Hardware), rádio (Rádio) e nível de energia (Bateria). Como o bloco do nó sensor é formado por esta sequência em série, se um deles falhar, todo o bloco do nó sensor

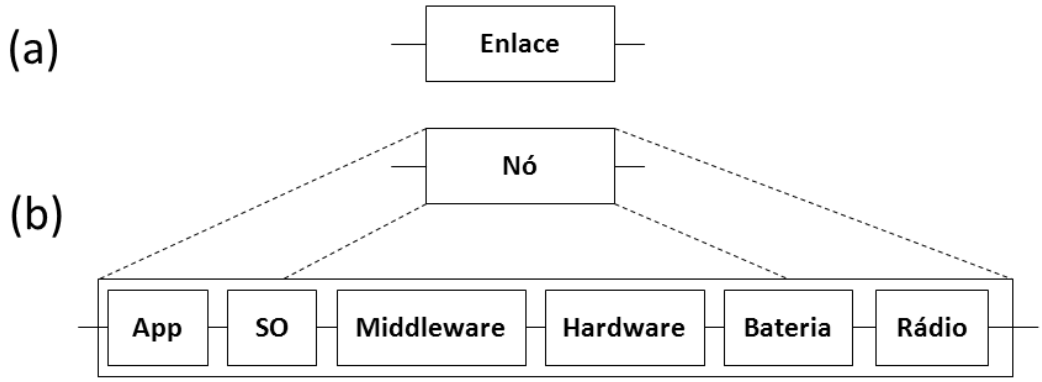


Figura 5.24 Blocos básicos do modelo de confiabilidade

falhará (veja a Seção 2.3). Cada um possui uma confiabilidade associada definida pelo usuário, cujo valor pode ser obtido através de uma especificação do software/hardware (*e.g.*, plataforma) ou através de uma simulação (*e.g.*, nível de energia).

Na realidade, a simulação é usada para saber o *status* (*e.g.*, nível de energia, rádio ativo/inativo) de um componente no tempo *t*. Usando essas informações (*status*), é possível definir a confiabilidade do componente. Por exemplo, a confiabilidade do nó sensor é diretamente afetada pelo seu nível de energia, *i.e.*, baixo nível de energia reflete em um baixa confiabilidade do nó sensor. Ao longo do tempo, a bateria é consumida e pode alcançar um nível que não atenda as necessidades energéticas do nó sensor, as quais não trabalharão corretamente (alta probabilidade de falha) ou morrerão (sempre falhará). Para estimar a confiabilidade da bateria, seu *status* é simulado para definir a sua confiabilidade como foi definido na Seção 7.6.1.

De maneira similar, a confiabilidade do rádio é também definida por simulação. Portanto, é possível capturar o seu *status* no tempo *t* através de simulação. O *status* do rádio assume dois valores: ligado e desligado. O nó sensor pode enviar, receber e repassar um pacote quando o rádio está ligado. Entretanto, quando ele está desligado, o nó sensor se torna "inativo"para a rede e não pode enviar, receber e nem repassar pacotes. Um pacote será perdido se for repassado para um nó sensor com o rádio desligado. Para representar isso em RBD, o bloco Rádio deve ser usado para indicar se o rádio está ativo (confiabilidade > 0,0) ou inativo (confiabilidade = 0,0).

5.4.2 Modelo de Caminho

O modelo de confiabilidade de um caminho (chamado de Modelo de Caminho) é formado por uma sequência dos blocos básicos. Para ser considerado um caminho, ele deve possuir pelo menos dois nós sensores (um salto), onde um nó sensor inicia (origem) e o outro termina (destino) o modelo. Caso haja mais nós sensores envolvidos (um caminho com vários saltos), eles serão colocados entre os blocos considerados como origem e destino.

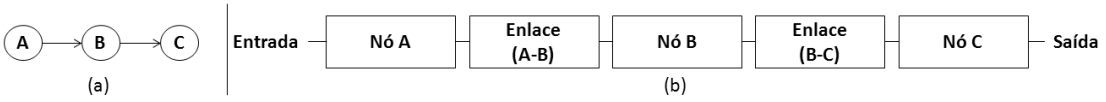


Figura 5.25 Exemplo de Modelo de Caminho

Por exemplo, a Figura 5.25(a) ilustra um caminho com os nós sensores *A*, *B* e *C*; no qual o nó sensor *A* e o *C* são a origem e o destino (respectivamente); e o nó sensor *B* roteia o pacote entre a origem e o destino. O modelo desse caminho é ilustrado na Figura 5.25(b). Tal como o caminho modelado, o nó sensor *A* (representado pelo bloco *Nó A*) começa o modelo, seguido pelos nós sensores *B* e *C*, os quais são representados pelos blocos *Nó B* e *Nó C*, respectivamente. Entre dois blocos *Nó*, existe o bloco representando a comunicação entre eles (*e.g.*, o bloco *Enlace (A-B)* representa a comunicação entre o nó sensor *A* e *B*). Dessa maneira, o modelo explicita que a falha pode ocorrer tanto no nó sensor quanto na comunicação entre eles; e que, caso um deles falhe, o caminho irá falhar. Adicionalmente, esse modelo RBD possui apenas um único caminho mínimo (CM_1 = *Nó A, Enlace (A-B), Nó B, Enlace (B-C), Nó C*).

Além disso, o nó sensor pode enviar em *multicast* (para um conjunto de nós sensores) ou em *broadcast* (para todos os nós sensores), repassando o mesmo pacote para mais de um nó sensor ao mesmo tempo. Por exemplo, a Figura 5.26(a) ilustra o nó sensor *A* enviando em *broadcast* para o nó sensor *B* e *C*. Por fim, esses dois nós sensores enviam o pacote para o mesmo destinatário: o nó sensor *D*.

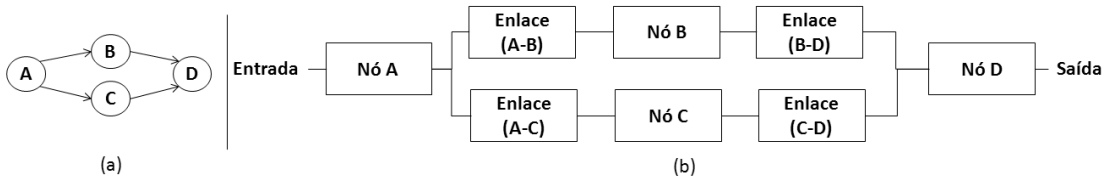


Figura 5.26 Exemplo de modelo de múltiplos caminhos

O modelo de confiabilidade desse caminho terá blocos básicos em paralelo (representando os múltiplos caminhos) nos pontos em que dois ou mais nós sensores repassam o mesmo pacote do mesmo transmissor. No entanto, em algum momento, o modelo considera que esses múltiplos caminhos irão se encontrar (convergir para o mesmo nó sensor), voltando a ficar com um único caminho. Essas duas características dos múltiplos caminhos (divisão e junção) fazem com que a sua representação no RBD comece e termine com um bloco de comunicação. O primeiro bloco de comunicação indica quando os múltiplos caminhos surgiram e o último bloco de comunicação representa quando elas se juntaram. Por exemplo, a Figura 5.26(b) ilustra quando dois caminhos são criados (os nós sensores *B* e *C* repassam o mesmo pacote do nó sensor *A*) e quando eles se juntam (se encontram no nó sensor *D*). Adicionalmente, esse modelo RBD possui dois caminhos mínimos: CM_1 *Nó A, Enlace (A-B), Nó B, Enlace (B-D), Nó D* e CM_2 *Nó A, Enlace (A-C), Nó C, Enlace (C-D), Nó D*.

Além disso, é importante observar que o Modelo de Caminho está relacionado com o algoritmo de roteamento utilizado (ver Seção 5.4.4), porque ele determina quais nós formam o caminho.

5.4.3 Modelo de Região

Após modelar os caminhos, o próximo passo é modelar uma região. É importante observar que uma região pode ter um ou vários nós sensores. Caso tenha apenas um nó sensor, o próprio caminho entre ele até o nó sorvedouro será considerado o caminho da região. No entanto, se a região tiver mais de um nó sensor, é necessário combinar os caminhos desses nós até o nó sorvedouro para representá-la. Em outras palavras, o Modelo de Região é o resultado da combinação dos Modelos dos Caminhos, os quais representam o caminho individual de cada nó sensor presente na região até o nó sorvedouro. Antes de combinar esses Modelos dos Caminhos, é necessário avaliar se eles possuem pontos em comuns: se os caminhos utilizam os mesmos nós sensores para rotear os pacotes de origem diferente.

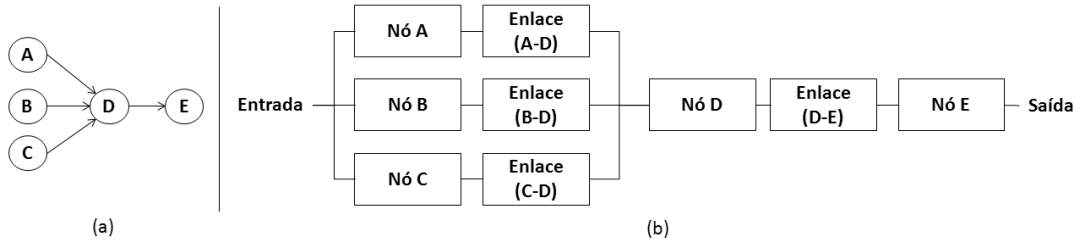


Figura 5.27 Exemplo de modelo de uma região

Por exemplo, a Figura 5.27(a) ilustra uma região e os seus caminhos até o nó sorvedouro. Esta região utiliza um protocolo hierárquico (que cria *clusters* na rede), possui três nós sensores (*A*, *B* e *C*) e tem o nó sensor *D* como líder. Desta maneira, todos os nós dessa região (*A*, *B* e *C*) devem enviar seus pacotes para o líder (*D*). Isso significa dizer que todos os nós sensores possuem um ponto em comum e isso deve ser representado no Modelo de Região (como é ilustrado na Figura 5.27(b)). Caso esse ponto em comum falhe, a região irá falhar independente do caminho. Adicionalmente, esse modelo RBD possui três caminhos mínimos: CM_1 *Nó A, Enlace (A-D), Nó D, Enlace (D-E), Nó E*, CM_2 *Nó B, Enlace (B-D), Nó D, Enlace (D-E), Nó E* e CM_3 *Nó C, Enlace (C-D), Nó D, Enlace (D-E), Nó E.*

Em alguns cenários específicos, como mostra a Figura 5.28(a), o comportamento operacional pode não ser representado por uma composição serie-paralelo em RBD. Nesses casos, os métodos pivotiamento ou SDP (Kuo and Zuo, 2003) devem ser aplicados para avaliar a confiabilidade da região interessada.

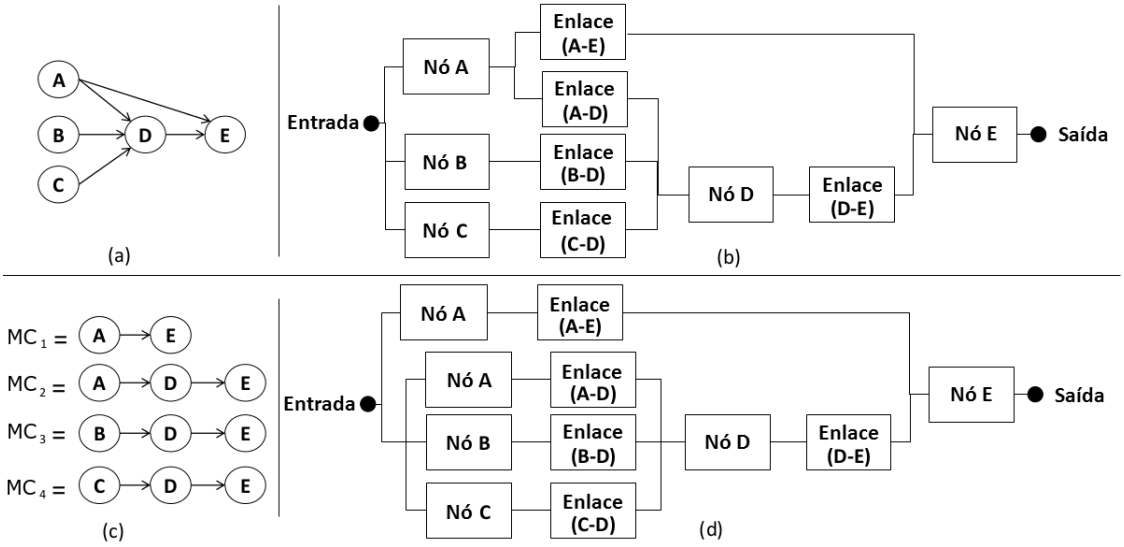


Figura 5.28 Cenário especifico que mostra uma (a) região; um RBD não-serie-paralelo; (c) caminhos mínimos da região; e (d) RBD serie-paralelo equivalente a representação obtidas através do SDP

Por exemplo, o nó sensor *A* envia pacotes em *multicast* para os nós sensores *D* e *E*, criando um RBD não-série-paralelo, como mostra a Figura 5.28(b). Para resolver essa situação, nós precisamos usar apenas o método SDP, mas antes é necessário definir os caminhos mínimos (veja a Seção 2.3.2) como mostra a Figura 5.28(c). Como os caminhos mínimos possuem elementos em comum (*Nó A*, *Nó D* e *Nó E*), a confiabilidade da região usando o método SDP é calculada a seguir (veja a Seção 2.3.2):

$$
\begin{aligned}
R_{regiao} &= R_{CM1} + \overline{R_{CM1}}\, R_{CM2} + \overline{R_{CM1}}\,\overline{R_{CM2}}\, R_{CM3} + \overline{R_{CM1}}\,\overline{R_{CM2}}\,\overline{R_{CM3}}\, R_{CM4} \\
&= R_A R_{L(AE)} R_E \\
&+ \overline{R_A R_{L(AE)} R_E}\, R_A R_{L(AD)} R_D R_{L(DE)} R_E \\
&+ \overline{R_A R_{L(AE)} R_E}\,\overline{R_A R_{L(AD)} R_D R_{L(DE)} R_E}\, R_B R_{L(BD)} R_D R_{L(DE)} R_E \\
&+ \overline{R_A R_{L(AE)} R_E}\,\overline{R_A R_{L(AD)} R_D R_{L(DE)} R_E}\,\overline{R_B R_{L(BD)} R_D R_{L(DE)} R_E}\, R_C R_{L(CD)} R_D R_{L(DE)} R_E \\
&= R_E(R_A R_{L(AE)} + R_D R_{L(DE)}(R_C R_{L(CD)} + R_B R_{L(BD)} + R_A R_{L(AD)}))
\end{aligned}
$$

CM representa um caminho mínimo ilustrado na Figura 5.28(c) e R_x representa a confiabilidade de um nó sensor (*e.g.*, R_A representa a confiabilidade do *Nó A*), enlace de comunicação (*e.g.*, $R_{L(AE)}$ representa a confiabilidade do enlace *Enlace (A-E)*), caminho mínimo (*e.g.*, R_{MP1} representa a confiabilidade do caminho mínimo MP_1), ou região (*e.g.*, R_{regiao}). Se aplicarmos o método SDP, nós iremos obter a expressão $R_E(R_A R_{L(AE)} + R_D R_{L(DE)}(R_C R_{L(CD)} + R_B R_{L(BD)} + R_A R_{L(AD)}))$. Adicionalmente, nós podemos criar um modelo RBD serie-paralelo (representando a confiabilidade da região), como ilustra a Figura 5.28(d), baseado na expressão resultado do SDP. Entretanto, isso é um passo adicional e não é necessário para avaliar a região visto que o método SDP é suficiente.

5.4.4 Impacto do roteamento no Modelo de Região

Como foi mencionado na Seção 5.1, a estratégia de roteamento tem um impacto importante na confiabilidade da RSSF. Cada algoritmo de roteamento define um conjunto diferente de nós sensores que compõe o caminho da região até o nó sorvedouro. Por esse motivo, a confiabilidade de uma determinada região depende diretamente do algoritmo de roteamento adotado. A Figura 5.29 ilustra uma RSSF usada para descrever os Modelos das Regiões criadas por diferentes algoritmos de roteamento.

Esta RSSF possui 20 nós sensores e um nó sorvedouro, e uma região distante do nó sorvedouro com apenas um único nó sensor. O modelo de confiabilidade dessa região será ilustrado quando a RSSF usa os protocolos de roteamento DIRECT (único salto), FLOODING (múltiplos caminhos), GOSSIPING (vários saltos) e LEACH (*cluster*). Esses protocolos foram selecionados por (I) possuirem diferentes estrategias para determinar um caminho, (II) serem fáceis de implementar; (III) serem amplamente utilizados; e (IV) representarem uma categoria de protocolo de roteamento (veja a Seção 2.1). Por exemplo, o LEACH cria *clusters* na RSSF, enquanto o FLOODING cria múltiplos caminhos.

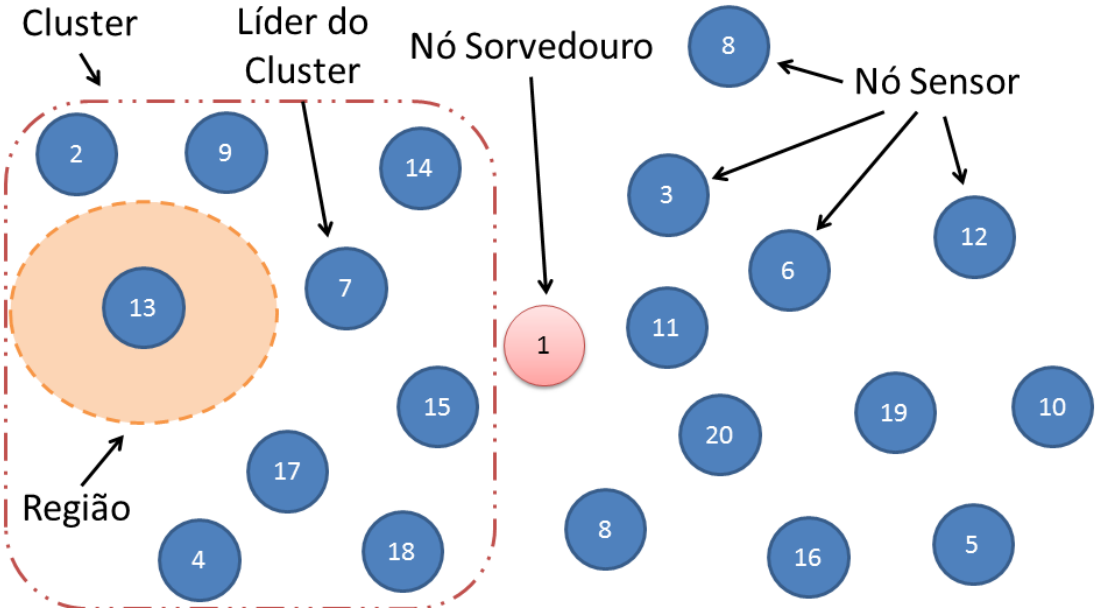


Figura 5.29 Exemplo de RSSF

Protocolo DIRECT

Como mencionado anteriormente, o protocolo DIRECT é ideal para redes pequenas, em que os nós sensores podem se comunicar diretamente com o nó sorvedouro. O seu algoritmo de roteamento seleciona apenas dois nós sensores como ilustrado na Figura 5.30: o transmissor e o receptor. Esse modelo só tem três blocos: o bloco *Nó 13* está representando a região; o *Nó 1* está relacionado ao nó sorvedouro, enquanto que o bloco *Enlace (13-1)* está representando a comunicação entre eles.

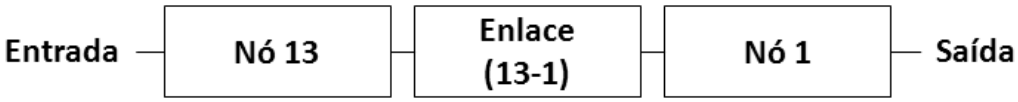


Figura 5.30 Modelo da confiabilidade de uma região usando o protocolo DIRECT

Protocolo FLOODING

O FLOODING é o único protocolo descrito nesta tese que envia o pacote em múltiplos caminhos. Esse protocolo envia uma mensagem em *broadcast* até chegar ao nó sorvedouro. Esta tese usou uma variante do FLOODING (o FLOODING baseado em probabilidade) por consumir menos energia. Inicialmente, o nó sensor cria e envia um pacote para os

seus vizinhos. Quando um vizinho recebe o pacote, ele decide se deve ou não repassar o pacote. Para tomar a decisão, o nó sensor gera um número randômico e só repassará o pacote se o número gerado for menor do que o número limiar *t* (o qual foi definido pelo desenvolvedor da aplicação). Dessa forma, a RSSF não irá utilizar todos os nós sensores da rede para repassar um pacote. Adicionalmente, todos os nós sensores irão se comportar assim, até quando o pacote chegar no nó sorvedouro.

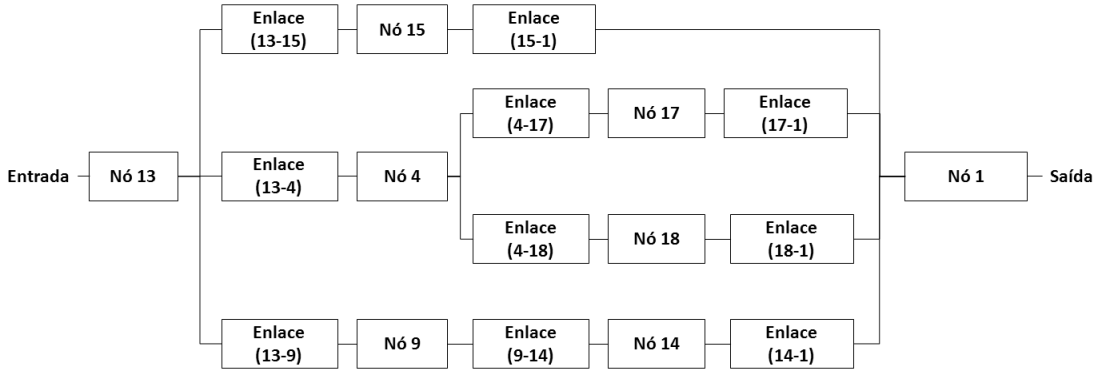


Figura 5.31 Modelo da confiabilidade de uma região usando o protocolo FLOODING

Por exemplo, a RSSF descrita anteriormente possui 21 nós (20 nós sensores e um nó sorvedouro), mas só oito participaram no roteamento do pacote (veja a Figura 5.31). O bloco *Nó 13* representa a origem (o nó sensor da região). Os blocos *Nó 15*, *Nó 4* e *Nó 9* são os nós sensores 15, 4 e 9 (respectivamente) que repassam, ao mesmo tempo, o pacote do nó sensor 13, criando múltiplos caminhos. O mesmo ocorre com os blocos *Nó 17* e *Nó 18*, os quais repassam o mesmo pacote do bloco *Nó 4*. Isso ocorreu porque todos os nós sensores enviaram o pacote em *broadcast*. Todos esses ramos se encontram no *Nó 1* (no destino). Para o pacote não chegar ao nó sorvedouro (*Nó 1*), é necessário que todos os caminhos falhem. Essa característica torna a região mais confiável. No entanto, ela requer muita energia da rede por utilizar mais nós sensores para transmitir o mesmo dado.

Protocolo GOSSIPING

Diferente do protocolo FLOODING, o protocolo GOSSIPING escolhe aleatoriamente um nó sensor para repassar o pacote. Por exemplo, o nó sensor *A* (origem) deseja enviar um pacote para o nó sorvedouro (destino). Inicialmente, o nó sensor *A* (quando deseja transmitir um pacote) escolhe aleatoriamente um nó sensor vizinho (dentro do raio de alcance de transmissão). Esse nó sensor vizinho, ao receber o pacote, verifica se é para ele (tratando o pacote) ou se é para outro nó sensor (escolhendo aleatoriamente o próximo

nó sensor). Isso continuará acontecendo até quando o nó sorvedouro receber o pacote (ou seja, atingir o nó sensor destino).

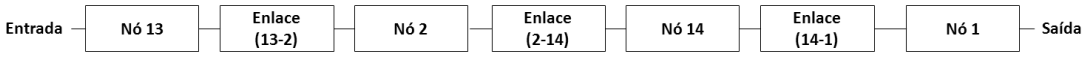


Figura 5.32 Modelo da confiabilidade de uma região usando o protocolo GOSSIPING

A Figura 5.32 mostra o Modelo de Região quando a RSSF usa o protocolo GOSSIPING. Esse modelo não tem múltiplos caminhos (porque a mensagem não foi enviada em *broadcast* ou em *multicast*) e possui mais de dois nós na transmissão do pacote (um caminho com vários saltos). O primeiro e o último blocos (*Nó 13* e *Nó 1*) representam o nó sensor da região e o nó sorvedouro, respectivamente. Enquanto os blocos *Nó 2* e *Nó 14* representam os dois nós sensores (2 e 14) que participaram no roteamento do pacote (os quais foram escolhidos aleatoriamente pelo seu antecessor).

Protocolo LEACH

Enquanto os demais protocolos consideram a rede plana, o protocolo LEACH cria periodicamente *clusters* (criando uma rede hierárquica) elegendo um líder do *cluster* (CH) para receber os pacotes dos nós sensores e repassá-los para o nó sorvedouro. Desta maneira, qualquer nó sensor (7 ou 13) está a um ou dois saltos do nó sorvedouro (1), como ilustra a Figura 5.33.

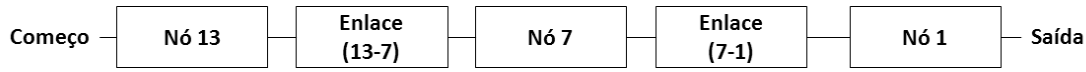


Figura 5.33 Modelo da confiabilidade de uma região usando o protocolo LEACH

Nesse caso, o nó sensor não era o líder do *cluster* (veja a Figura 5.33). O primeiro bloco (chamado de *Nó 13*) representa o nó sensor da região; o segundo (bloco *Nó 7*) é o líder do *cluster* e o último (bloco *Nó 1*) é o nó sorvedouro.

5.5 Considerações Finais

Esse capítulo apresentou os modelos para avaliar o consumo de energia e a confiabilidade da RSSF. Cada avaliação foi feita em um modelo específico: o consumo de energia foi avaliado em CPN; enquanto a confiabilidade, em RBD. Devido à complexidade das RSSFs, esses modelos foram divididos em algumas partes. O consumo de energia foi

divido em três modelos: Modelo de Aplicação, para avaliar o consumo de energia de uma aplicação escrita em nesC; o Modelo de Rede, para avaliar consumo de energia dos protocolos de rede; e o Modelo de Nó Sensor, a junção dos Modelos da Aplicação e da Rede. Por sua vez, o Modelo de Aplicação e de Rede foram divididos em Modelos dos Operadores, os quais representam o consumo de energia dos operadores e das estruturas da linguagem, enquanto que os Modelos dos Protocolos, representando os protocolos da camada de rede e de enlace. Análogo ao consumo de energia, o Modelo de Confiabilidade da RSSF também foi divido em dois blocos básicos (comunicação e nó sensor), os quais representam os pontos de falhas na RSSF. A combinação (em sequência) desses blocos básicos forma um caminho que representam a trajetória dos pacotes da região até o nó sorvedouro. Através dessas informações, é possível avaliar a confiabilidade das regiões definidas na RSSF.

6

Ambiente de Desenvolvimento e de Avaliação

Este capítulo apresenta o ambiente de desenvolvimento e avaliação da RSSF para dar suporte à metodologia proposta no Capítulo 4. Primeiro, a visão geral da ferramenta é mostrada, apresentando as ferramentas que compõe o ambiente e como elas interagem entre si. Em seguida, cada ferramenta é detalhada separadamente.

6.1 Visão Geral

O capítulo anterior apresentou os modelos para avaliar o consumo de energia e a confiabilidade da RSSF usando modelos CPN e RBD, respectivamente. Esses modelos são elaborados de acordo com as características da aplicação e/ou da configuração da rede, sendo um processo complexo para ser executado pelo usuário. Além disso, o usuário deseja focar mais no desenvolvendo da RSSF (por exemplo, programando a aplicação) e utilizar ferramentas para auxiliar ou automatizar o processo de avaliação da RSSF. Observando isso, esta tese criou um ambiente, chamado EDEN (*Evaluation and Development Environment for Wireless Sensor Network*), para facilitar o desenvolvimento e avaliação da RSSF. EDEN é um ambiente Web, distribuído, e composto por um editor, tradutor, avaliador e gerenciador (como é mostrada na Figura 6.1). O ambiente pode ter várias instâncias da mesma ferramenta e está implantado em um ambiente de nuvem para, principalmente, aumentar a capacidade de processamento, principalmente, com relação a avaliação da RSSF.

Juntas, as quatro ferramentas são responsáveis por suportar a metodologia definida na Seção 4. O editor facilita a criação da aplicação e/ou da configuração da rede. O tradutor traduz o código da aplicação ou da configuração da rede para um dos modelos

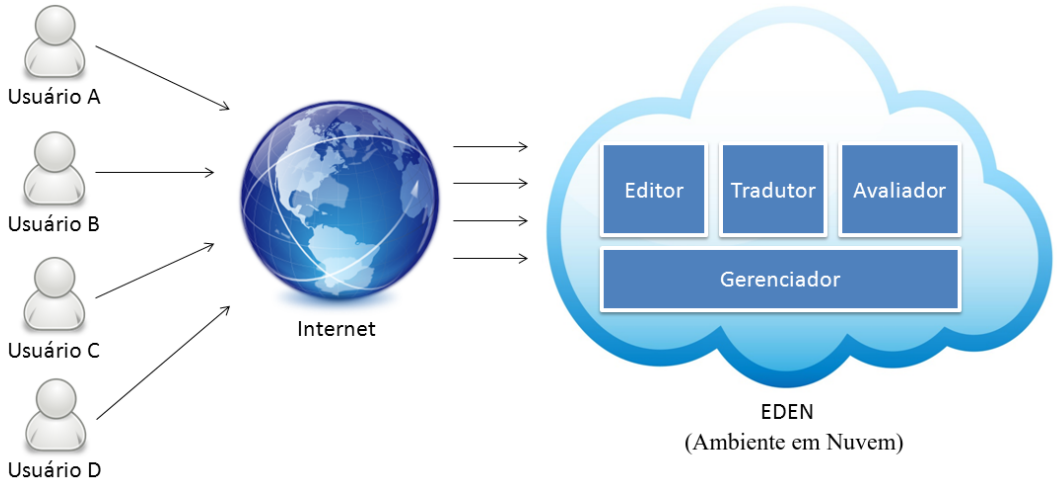


Figura 6.1 Visão geral do EDEN.

apresentados no Capítulo 5. O avaliador é responsável por avaliar o consumo de energia e a confiabilidade da RSSF usando os modelos gerados. E, por último, o gerenciador controla o processo de avaliação, intermediando a comunicação entre as ferramentas. A Figura 6.2 mostra a participação do editor, tradutor e avaliador em cada fase da metodologia apresentada no Capítulo 4.

O editor, chamado de IDEA4WSN, é responsável pelas fases *Planejamento*, *Codificação* e *Otimização*. O editor também é responsável por interligar com as outras ferramentas, as quais estão relacionadas com a *Validação*, mais especificamente na avaliação da RSSF. O tradutor, Sensor2Model, é responsável pelas atividades relacionadas com a composição, simplificação e integração dos modelos da avaliação da RSSF. O Modelo de Aplicação é criado a partir do código da aplicação escrito em nesC, enquanto o Modelo de Rede e o Modelo de Região são derivados do arquivo da rede. Além disso, dois repositórios foram utilizados para armazenar os Modelos dos Operadores e dos Protocolos, os quais são usados para criar o Modelo de Aplicação e o Modelo de Rede, respectivamente. Os Modelos da Aplicação e da Rede são integrados, criando o Modelo de Nó Sensor. O avaliador, chamando de Vident (*Vident is EDEN Evaluator*), é responsável por avaliar o consumo de energia ou confiabilidade dos modelos gerados pelo tradutor. O gerenciador não aparece explicitamente nessa figura (mas é ilustrado na Figura 6.3), pois ele é responsável por intermediar a comunicação entre as ferramentas. Por exemplo, ele é responsável por repassar os modelos criados pelo tradutor para o avaliador.

A Figura 6.2 também ilustra a sequência de operações das quatro ferramentas. Pri-

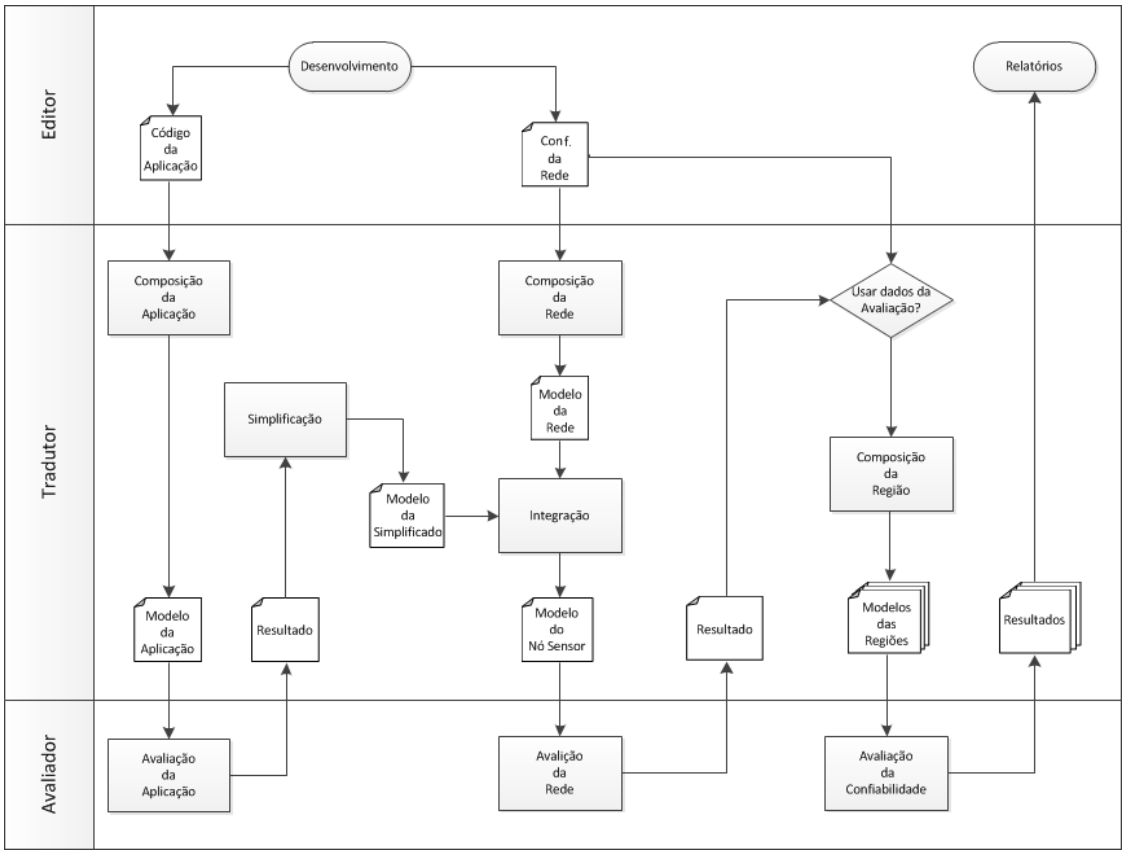


Figura 6.2 Sequência de atividades do EDEN.

meiro, o usuário cria uma aplicação e/ou uma configuração de rede no editor. Depois, alguns parâmetros, tais como diretório da aplicação, arquivo da configuração de rede, plataforma, nível de confiança, são passados para o tradutor, que cria os modelos de consumo de energia e confiabilidade, acessando os repositórios dos modelos básicos. Como foi dito na Seção 5.3.4, o Modelo de Aplicação é o primeiro modelo a ser criado e avaliado. Após a avaliação, o Modelo de Aplicação é simplificado e integrado com o Modelo de Rede, criando o Modelo de Nó Sensor. O último modelo a ser criado e avaliado é o da região porque pode usar informações do Modelo de Nó Sensor (veja a Seção 5.4). Caso não use as informações oriundas desse modelo, o usuário deverá informar manualmente esses dados, *e.g.*, nível de energia dos nós sensores. Como a RSSF pode ter várias regiões, o tradutor cria um modelo para cada região, avaliando-os individualmente. Por fim, quando o avaliador termina de avaliar todas as regiões, vários resultados são criados e passados para o editor, o qual irá mostrar relatórios para o usuário.

6.2 Arquitetura

A Figura 6.3 ilustra a arquitetura do EDEN. O editor tem três componentes: *Gerenciador de Arquivos*, para criar e gravar documentos tais como o código da aplicação; *Proponente de Sugestões*, para sugerir melhorias no consumo de energia e/ou confiabilidade da aplicação ou da infraestrutura da RSSF; e *Analisador de Sensibilidade*, para mostrar o impacto de cada parâmetro no consumo de energia e na confiabilidade da RSSF. O tradutor tem um repositório para armazenar os modelos básicos e dois componentes para criar os modelos CPN (*Tradutor de Modelos de Energia*) e RBD (*Tradutor de Modelo de Confiabilidade*). O avaliador tem três componentes: *Avaliador de Energia* e *Avaliador de Confiabilidade*, para avaliar os modelos CPN e RBD criados pelo tradutor, respectivamente; e o *Gerador de Relatório*, responsável por gerar os relatórios de consumo de energia e confiabilidade da RSSF. Por fim, o gerenciador tem o componente *Gerenciador de Avaliação* para controlar a avaliação do consumo de energia e a confiabilidade da RSSF; e outro componente, chamado de *Gerenciador de Ciclo de Vida*, para controlar as instâncias do editor, tradutor e avaliador.

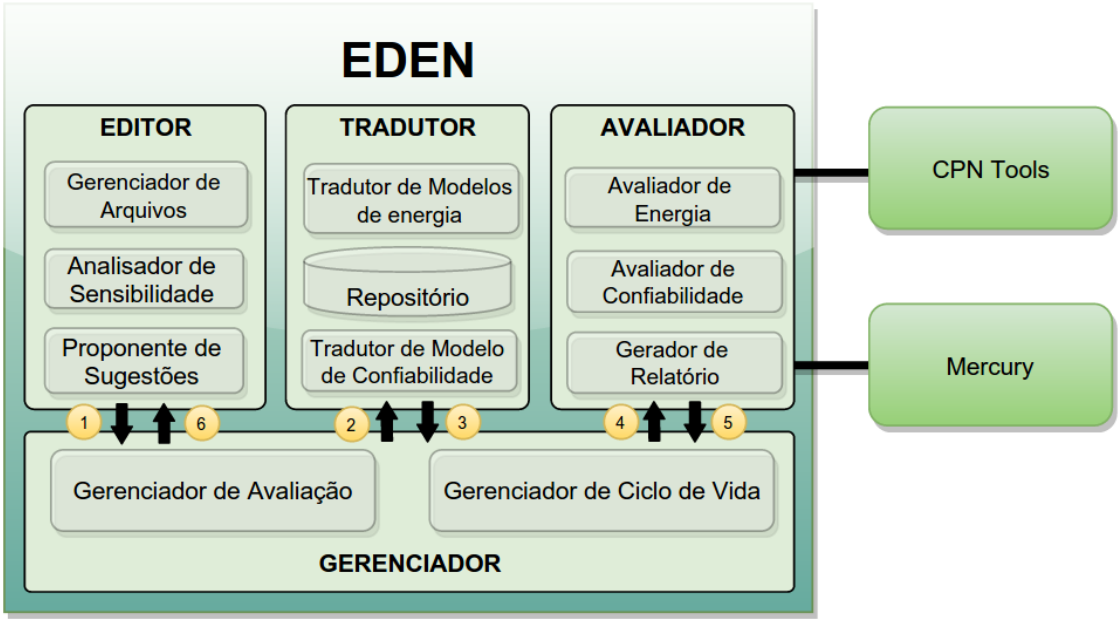


Figura 6.3 Arquitetura do EDEN.

A Figura 6.3 ilustra ainda a sequência de passos para mostrar o processo de avaliação da RSSF, mas (desta vez) inclui o gerenciador no processo de avaliação. Tal como foi mostrado na Figura 6.2, o editor é utilizado pelo usuário para criar a aplicação e a RSSF. Quando o usuário deseja avaliar o consumo de energia e a confiabilidade da RSSF, o

editor informa ao gerenciador (passo 1) que, por sua vez, repassa os arquivos da aplicação e da rede para o tradutor (passo 2). Este irá traduzir os arquivos em modelos CPN e RBD, os quais são repassados para o gerenciador (passo 3). Assim que os recebe, o gerenciador envia-os para o avaliador (passo 4) que vai avaliar o consumo de energia e a confiabilidade da RSSF. Quando terminar a sua atividade, o avaliador gera os relatórios (passo 5) e envia-os para o gerenciador, o qual repassa para o editor (passo 6).

Com relação à funcionalidade chamada *Analisador de Sensibilidade*, o usuário deve definir quais parâmetros, *e.g.*, nível de energia e tamanho do pacote, precisam ser avaliados. Várias configurações de redes são criadas e avaliadas automaticamente, usando essa mesma sequência de passos apresentada anteriormente. Após avaliar todas as redes, o usuário decide qual configuração criada teve o melhor resultado de acordo com o planejamento da RSSF definida por ele. Por exemplo, ele/ela pode preferir que a RSSF tenha um melhor consumo de energia do que confiabilidade.

Vale ressaltar três informações importantes sobre o processo de avaliação da RSSF. Primeiro, o processo de avaliação é assíncrono e baseado em eventos. Por exemplo, o editor é liberado logo após enviar os arquivos da RSSF para serem avaliados. O gerenciador irá notificá-lo lançando um evento quando a RSSF for avaliada. O tradutor e o avaliador também notificam o gerenciador lançando um evento quando eles traduzirem e avaliarem uma RSSF, respectivamente. Segundo, a arquitetura é escalável, permitindo que várias instâncias do editor, tradutor e avaliador existam. O gerenciador é o responsável por controlar e escolher a instância que irá participar da avaliação da RSSF. Por fim, já que existem várias instâncias, o gerenciador faz balanceamento de carga entre elas. Tais características do EDEN são importantes para fazer a análise de sensibilidade da RSSF.

6.3 Editor

Esta seção apresenta o editor IDEA4WSN (*Integrated Environment for Developing Energy-Aware Applications for WSN*). O IDEA4WSN possui duas partes: aplicação, que auxilia o desenvolvedor a construir códigos mais eficientes do ponto de vista de consumo de energia, e rede, que permite avaliar e traçar o perfil energético da rede. Estas duas partes fazem uso do Sensor2Model (veja a Seção 6.4) e Vident (veja a Seção 6.5) para traduzir e avaliar, respectivamente, o consumo de energia e a confiabilidade da RSSF.

6.3.1 Requisitos

Os requisitos do editor são mostrados na Tabela 6.1. O requisito ED01 divide o editor em duas partes: editor de aplicação (ED02 ao ED07) e editor de rede (ED08 ao ED11). Os requisitos ED02 e ED03 são fundamentais para que o usuário possa identificar erros no código fonte. Os requisitos do ED04 ao ED06 são os principais requisitos que tornam o editor da aplicação ciente de energia. Através deles, o editor poderá avaliar e sugerir modificações no código fonte da aplicação, sempre visando aumentar o tempo de vida do nó sensor. O último requisito da aplicação (ED07) serve para mostrar ao desenvolvedor o modelo de energia avaliado.

Tabela 6.1 Requisitos funcionais do editor.

Requisito	Descrição
ED01	Permitir escrever o código fonte da aplicação em nesC e definir a configuração da RSSF.
ED02	Facilitar a implementação de aplicações, destacando palavras-chaves, blocos de código e estruturas em nesC.
ED03	Validar o código em tempo de desenvolvimento, identificando erros e oferecendo opções para tratá-los.
ED04	Sugerir códigos otimizados com a mesma funcionalidade, mas que consumam menos energia.
ED05	Estimar o consumo de energia da aplicação antes de implantá-la no nó sensor real.
ED06	Comparar o consumo de energia de duas ou mais funções implementadas na aplicação.
ED07	Apresentar o modelo de energia da aplicação.
ED08	Criar representação visual da configuração de rede, possibilitando visualizar o alcance do nó.
ED09	Sugerir otimizações visando a redução do consumo de energia na rede.
ED10	Estimar o consumo da rede e identificar pontos críticos.
ED11	Possibilitar a criação do mapa de energia da rede.

Por outro lado, o editor de rede deverá fornecer ao desenvolvedor facilidades para criação de uma rede. Essas facilidades vão desde oferecer a forma mais adequada da disposição dos nós sensores no ambiente (ED08), a escolha do protocolo mais adequado (ED09), como também avaliar e mapear a energia da rede (ED10 e ED11).

6.3.2 Arquitetura e Implementação

A arquitetura do IDEA4WSN (Figura 6.4) é composta pelos módulos *Desenvolvimento*, *Aplicação* e *Rede*. Em todos os componentes foram utilizadas as mesmas tecnologias: JSP (Corporation, 2014) para criar as telas de exibição; JQuery (jQuery Foundation., 2014) para melhorar a interação do usuário com o ambiente de desenvolvimento; Twitter Bootstrap (Twitter, 2014) para melhorar a aparência do ambiente; o Spring (GoPivotal, 2014) para tratar as requisições do usuário no servidor; e o JFreeChart (Limited, 2014) para gerar os gráficos (relatórios). As telas do editor podem ser vistas no Apêndice A.

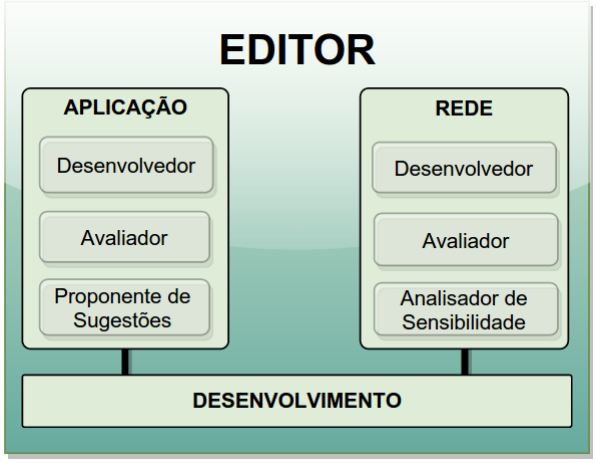


Figura 6.4 Arquitetura do IDEA4WSN.

O módulo *Desenvolvimento* é responsável por oferecer funcionalidades básicas para os outros módulos. Por exemplo, uma de suas funções é identificar o usuário logado, facilitar o acesso aos projetos independente de onde eles estejam armazenados, implementar a interface básica da IDEA4WSN, executar atividades que levam longo tempo para serem concluídas (*e.g.*, avaliar do consumo de energia da aplicação e/ou da rede) em segundo plano e, por fim, configurar o ambiente.

Para facilitar o desenvolvimento das aplicações, foi desenvolvido o módulo *Aplicação*. Ele possui um pacote chamado *Desenvolvedor* responsável por destacar as palavras chaves (através de uma extensão do JQuery chamada ACE Editor (IDE, 2014)), validar o código e sugerir correções de erros. Essas características atendem aos requisitos do ED01 ao ED03. Além disso, esse módulo é responsável por avaliar uma aplicação ou parte dela (pacote *Avaliador*), sugerir modificações (pacote *Proponente de Sugestões*) para diminuir o consumo de energia (veja a Seção 6.3.3) e mostrar os resultados da avaliação, atendendo aos requisitos do ED04 ao ED07. Esse módulo utiliza o gerenciador para

algumas atividades mencionadas anteriormente.

O módulo *Rede* é responsável pelo editor de rede e pelas funcionalidades a fim de avaliar o seu consumo de energia e a sua confiabilidade. Ele foi construído para atender aos requisitos do ED08 ao ED11 e consiste em três partes: o pacote *Desenvolvedor*, responsável pela representação visual do editor e sugestões de otimizações; o pacote *Avaliador*, responsável pela avaliação da energia e da confiabilidade e responsável por elaborar os relatórios após as avaliações; e o pacote *Analisador de Sensibilidade* é responsável por fazer análise de sensibilidade dos parâmetros escolhidos pelo usuário.

O módulo *Avaliador* é responsável por interagir com o gerenciador, enviando e recebendo informações sobre as avaliações.

Entre os pacotes implementados, as sugestões (do módulo *Aplicação*) são as mais importantes por tornar o ambiente de desenvolvimento ciente de energia. Por este motivo, a próxima seção irá explicar quais foram as sugestões implementadas na aplicação.

6.3.3 Sugestões

As otimizações implementadas no editor da aplicação reduzem o consumo de energia da aplicação e não alteram a sua lógica. Individualmente, estas otimizações não representam um ganho significativo de energia (apenas microjoules). No entanto, se uma função otimizada for executada diversas vezes, esse ganho pode se tornar significativo. Além disso, essas sugestões são genéricas o suficiente para serem usadas em qualquer aplicação. Ao todo, foram implementadas cinco otimizações, as quais são apresentadas a seguir.

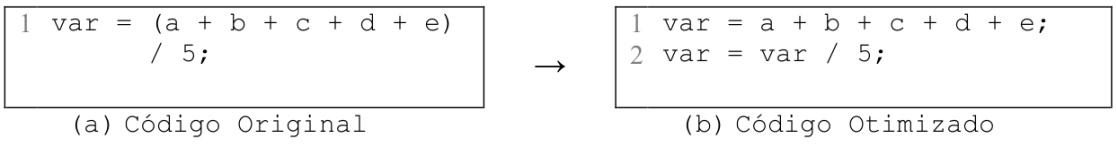


Figura 6.5 Exemplo de código otimizado sem parêntese.

A primeira sugestão (OT01) é evitar o uso de parênteses em operações matemáticas para não forçar uma mudança de prioridade de execução. Para isso, é recomendado tratá-los em outro lugar, atribuindo o valor a uma variável (veja a Figura 6.5). Essa substituição, um parâmetro por uma atribuição, é recomendada pelo fato de a atribuição consumir menos energia do que os parênteses.

A segunda sugestão (OT02) está relacionada à estrutura de repetição `for`. É recomendado evitar o uso dessa estrutura para iniciar *arrays* de tamanho fixo. Em vez disso, é sugerido atribuir individualmente os valores dos elementos do *array*. A Figura 6.6 ilustra

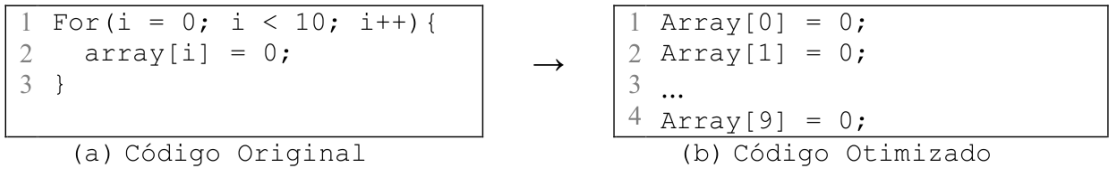


Figura 6.6 Exemplo de código otimizado sem `for`.

dois códigos, o original e a sua versão otimizada.

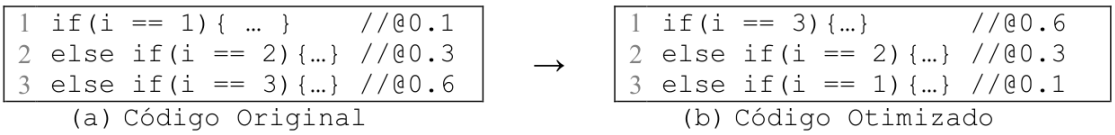


Figura 6.7 Exemplo de código reordenando os ramos de um `if-then-else`.

A terceira sugestão (OT03) está relacionada a outra estrutura muito utilizada, o `if-then-else`. Essa estrutura pode ser otimizada da seguinte forma: os ramos (`if`, `else if` `else`), que possuem maior probabilidades de serem executados, devem ser colocados primeiro. O usuário informa a probabilidade do ramo de ser executado através da expressão `//valor` (como foi mencionado na Seção 5.2.1). Por exemplo, na Figura 6.7 o código original (a) possui um `if-then-else` com 3 ramos: o primeiro com 10%, o segundo, 30% e o terceiro, 60% de ser executado. A versão otimizada (b) muda a ordem em que os ramos aparecem: o ramo com maior probabilidade vem primeiro.

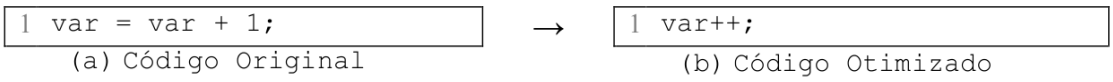


Figura 6.8 Exemplo de código reduzindo o incremento do valor 1 à uma variável.

A quarta sugestão (OT04) está relacionada ao incremento de uma variável. A expressão completa (var = var + 1) consome mais energia do que sua versão simplificada (var++), como é mostrado na Figura 6.8.

A última sugestão (OT05) está relacionada com a melhor forma de atribuir um valor a uma variável. É recomendado atribuir um valor fixo/constante a uma variável ao invés de outra variável. Caso seja utilizada uma variável e essa variável não mude de valor ao longo do tempo, é recomendado torná-la uma constante. Por exemplo, a Figura 6.9, em especial o código original (a), ilustra o cálculo do comprimento de um círculo, em que o valor de π (variável PI) é sempre o mesmo. Então, é recomendável que a variável PI seja uma constante, como ilustra o código otimizado (b).

```
1 float PI = 3.14159265;
2 …
3 float c  = 2 * PI * 10;
```
(a) Código Original

→

```
1 #define PI 3.14159265;
2 …
3 float c  = 2 * PI * 10;
```
(b) Código Otimizado

Figura 6.9 Exemplo de código transformando uma variável em constante.

6.4 Tradutor

O tradutor, por sua vez, chamado de Sensor2Model, é responsável por traduzir o código fonte da aplicação escrito em nesC em um Modelo de Aplicação; e a configuração da RSSF em um Modelo de Rede e um Modelo de Região. Ele também integra o Modelo de Aplicação com o Modelo de Rede, criando o Modelo de Nó Sensor. A arquitetura do Sensor2Model é ilustrada na Figura 6.10.

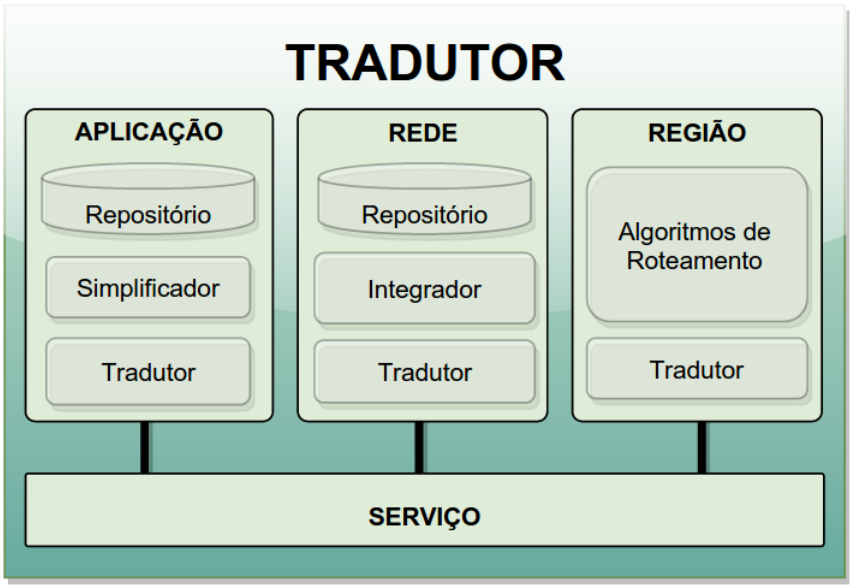


Figura 6.10 Visão geral da arquitetura do tradutor.

O Sensor2Model é divido em 4 módulos: *Serviço*, *Aplicação*, *Rede* e *Região*. O módulo *Serviço* recebe os dados do gerenciador (por exemplo, código da aplicação e configuração da RSSF) e repassa-os para o módulo correspondente. A interface abaixo é implementada por este módulo:

```
1 public interface Tradutor {
2   public void modeloDeAplicacao( Informacao inf );
3   public void modeloDeRede     ( Informacao inf );
3   public void modeloDeNoSensor ( Informacao inf );
4   public void modeloDeRegiao   ( Informacao inf );
5 }
```

Esta interface possui quatro métodos, um para cada modelo; por exemplo, o método *modeloDeAplicacao* irá repassar para o módulo *Aplicação* para criar o Modelo de Aplicação. Todos os métodos recebem o mesmo objeto, mas com informações diferentes. Por exemplo, o *modeloDeAplicacao* irá receber informações relacionadas à aplicação escrita na linguagem nesC; enquanto o *modeloDeRede* irá receber do editor sobre a RSSF.

Além disso, esta interface é responsável por intermediar as ações entre os módulos. Por exemplo, para gerar o Modelo de Nó Sensor, o método `modeloDeNoSensor` implementado pelo módulo *Serviço*, primeiro acessa o módulo *Aplicação*, simplifica-o (usando o sub-módulo *Simplificação*) e, depois, integra-o com o módulo *Rede* (através do módulo *Integração*). Ou, então, após executar o Modelo de Rede, o consumo de energia avaliado é repassado para a *Região* para simular a confiabilidade da RSSF.

Os outros três módulos (*Aplicação*, *Rede* e *Região*) possuem comportamentos similares. Cada um irá converter um modelo (sub-módulo *Tradutor*). Por exemplo, o módulo *Aplicação* está relacionado com o Modelo de Aplicação; o módulo *Rede*, com o Modelo de Rede; e o módulo *Região*, com o Modelo de Região. Em especial, os módulos *Aplicação* e *Rede* possuem os sub-módulos *Simplificador* e *Integrador*, respectivamente, para gerar o Modelo de Nó Sensor. Todos esses módulos irão utilizar o gerenciador para notificar que os modelos foram criados. Além disso, enquanto os Modelos de Operadores e de Protocolos irão fazer parte do Modelo de Aplicação e de Rede (respectivamente), os algoritmos de roteamento irão auxiliar na criação do Modelo de Região, como foi mencionado na Seção 5.4.

6.5 Avaliador

O avaliador, chamado de Vident (*Vident is EDEN Evaluator*), utiliza outros programas, *e.g.*, CPN Tools e Mercury, para avaliar os modelos. Ele atua como ponte entre o tradutor e os programas que irão avaliar de fato os modelos.

A arquitetura do Vident possui apenas três módulos (veja a Figura 6.11). O módulo *Serviço* é responsável por oferecer uma única interface para avaliar tanto o consumo de energia quanto a confiabilidade da RSSF. Essa interface é mostrada abaixo:

```
1 public interface Avaliador {
2   public void consumoDeEnergia( Informacao inf );
3   public void confiabilidade  ( Informacao inf );
4 }
```

Ao receber uma requisição, o *Serviço* repassa a chamada para o módulo apropriado, o qual irá avaliar o consumo de energia (o módulo *Consumo de Energia*) ou a confia-

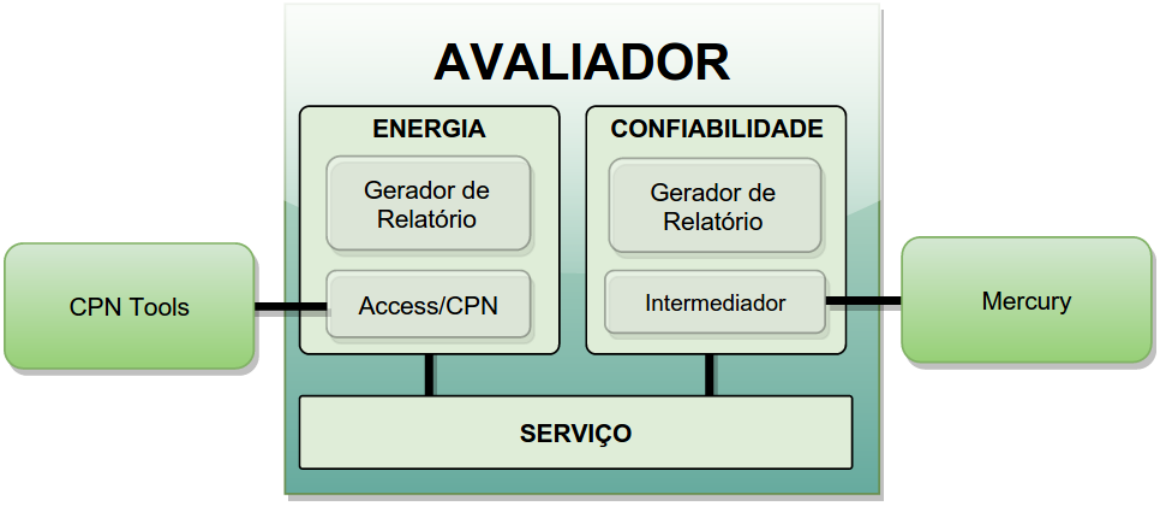


Figura 6.11 Visão geral da arquitetura do avaliador.

bilidade (o módulo *Confiabilidade*). O módulo *Consumo de Energia* é responsável por interagir com o CPN Tools (Westergaard and Verbeek, 2014), utilizando o Access/CPN (Westergaard, 2011), para avaliar os modelos CPN criados pelo tradutor. Por fim, o módulo *Confiabilidade* utiliza o Mercury (Silva *et al.*, 2013) para avaliar a confiabilidade usando os modelos RBDs criados pelo tradutor.

É importante perceber que os métodos não retornam nenhum dado, porque o processo de avaliação é assíncrono: quando a avaliação terminar, o avaliador notifica o gerenciador através do componente *Resultado*. Este componente está presente nos módulos *Consumo de Energia* e *Confiabilidade*.

6.6 Gerenciador

A arquitetura do gerenciador já foi apresentada na Figura 6.3 e, por este motivo, não será apresentada novamente nesta seção. Em vez disso, nós iremos complementar as informações ditas anteriormente. Cada módulo do gerenciador (*Gerenciador de Avaliação* e *Gerenciador de Ciclo de Vida*) implementam uma interface para se comunicar com as demais ferramentas do EDEN. As duas interfaces implementadas são mostradas abaixo:

```
01 public interface Ferramenta {
02   public void addEditor   ();
03   public void addTradutor ();
04   public void addAvaliador();
05 }
06
07 public interface Avaliacao {
```

```
08   public void edicaoConcluida   ( Informacao inf );
09   public void traducaoConcluida ( Informacao inf );
10   public void avaliacaoConcluida( Informacao inf );
11 }
```

A interface *Ferramenta* é implementada pelo módulo *Gerenciador de Ciclo de Vida* e é utilizada para adicionar uma instância do editor (linha 2), tradutor (linha 3) ou avaliador (linha 4). Toda vez que uma ferramenta é instanciada, ela deve notificar o gerenciador através dessa interface. O gerenciador é responsável por identificar qual ferramenta se conectou a ele através dos métodos da interface. Por exemplo, o método *addEditor* significa que um editor acabou de se conectar ao gerenciador. Além disso, cria-se um canal bidirecional entre a ferramenta e o gerenciador. Essa característica permite que o gerenciador utilize a conexão já existente para enviar ou receber notificações do processo de avaliação. Por exemplo, quando o avaliador conclui uma avaliação, ele envia uma notificação por este canal já estabelecido, acionando a interface *Avaliacao*.

A interface *Avaliacao* é utilizada para gerenciar a avaliação e está implementada ao módulo *Avaliação*. Os métodos desta interface indicam qual etapa foi finalizada. Quando um desses métodos é chamado, o gerenciador fica responsável por repassar para a próxima etapa. Por exemplo, quando o método *edicaoConcluida* é chamado, o gerenciador repassa as informações recebidas para o tradutor criar os modelos. Quando os modelos são criados, o método *traducaoConcluida* é executado, dando início à etapa de avaliação, e assim por diante. Adicionalmente, o módulo *Gerenciador de Avaliação* utiliza as informações no módulo *Gerenciador de Ciclo de Vida* para repassar os dados entre as instâncias das ferramentas.

6.7 Considerações Finais

Este capítulo apresentou o EDEN (*Evaluation and Development Environment for Wireless Sensor Network*), o qual foi desenvolvido para ser executado em um ambiente de nuvem. O EDEN é composto por quatro ferramentas: editor, tradutor, avaliador e gerenciador. O editor, chamado de IDEA4WSN (*Integrated Environment for Developing Energy-Aware Applications for WSN*), é responsável por facilitar o desenvolvimento da aplicação e da configuração da RSSF. Além disso, ele oferece um mecanismo que avalia o que está sendo feito e sugere melhorias no código fonte, buscando melhorar o consumo energético. Outra funcionalidade do editor é mostrar para o usuário quais são os parâmetros ideais para a RSSF logo após o usuário informar um conjunto de elementos (*e.g.*, protocolo, tamanho da rede e entre outras coisas). O tradutor (chamado de Sensor2Model) é utilizado para

traduzir o código da aplicação e a configuração da RSSF para os modelos de energia e confiabilidade apresentados anteriormente. Por sua vez, o tradutor necessita do avaliador, chamado de Vident (*Vident is EDEN Evaluator*), para avaliar os modelos gerados. Para intermediar a comunicação entre essas três ferramentas, existe o gerenciador. Por fim, o editor mostra para o usuário relatórios dos modelos avaliados.

7 Avaliação

Este capítulo apresenta os resultados dos experimentos realizados com os modelos de consumo de energia e de confiabilidade da RSSF propostos no Capítulo 5. Primeiro, este capítulo mostra o ambiente de medição para obter o consumo de energia de um nó sensor real. Em seguida, um exemplo de como medimos o consumo de energia dos operadores da linguagem nesC. Depois, os experimentos para validar as sugestões implementadas no EDEN (Seção 6.3.3) e os modelos propostos (Capítulo 5) são apresentados. Por fim, este capítulo mostra a análise de sensibilidade e o impacto de cada fator no consumo de energia e na confiabilidade da RSSF.

7.1 Ambiente de Medição

Para fazer uma medição no nó sensor, são necessários os seguintes elementos: AMALGHMA, osciloscópio, fonte de tensão contínua e um nó sensor. O AMALGHMA (*Advanced Measurement Algorithms for Hardware Architectures*) (Tavares *et al.*, 2008) é o programa responsável por recuperar os dados gerados pelo osciloscópio e transformá-los em informações como o consumo de energia, o valor da corrente, o tempo de processamento, entre outras coisas. O osciloscópio está conectado diretamente ao nó sensor, coletando a energia usada pela aplicação. A fonte de tensão contínua é utilizada para fornecer a mesma tensão durante todo o experimento. Por fim, o nó sensor possui uma aplicação em análise, indicando o momento em que deve contabilizar o consumo de energia.

AMALGHMA é uma ferramenta implementada em Java para automatizar a atividade de medição, capaz de estimar o tempo médio de execução, o consumo de energia e outras estatísticas relacionadas tais como: *bootstrap* e os métodos paramétricos. Para medir o consumo de energia, por exemplo, uma estação de trabalho (*e.g.*, desktop) executando a ferramenta AMALGHMA é conectada ao osciloscópio.

A medição foi realizada com o auxílio de um osciloscópio, modelo DSO3102A Agilent, o qual possui dois ganchos para trabalhar com o nó sensor. Um deles deve ser conectado ao fio sinalizador, que indica o momento de iniciar o cálculo do consumo de energia e quando deve parar. A sinalização é feita de forma simples: quando passa corrente neste local onde está o gancho significa que deve ser começada a contabilização. Quando não passa, a contabilização deve ser encerrada. O outro gancho fica conectado a uma resistência. Essa resistência é a forma que o osciloscópio tem para captar a energia que está sendo usada pelo nó sensor.

O tempo dessa coleta pode ser ajustado de 2 microssegundos a 50 segundos. Este valor está relacionado a um quadro do visor. Como o visor tem 12 quadros, é possível criar cenários que durem de 24 microssegundos (resultado obtido da multiplicação entre o menor valor do quadro e a quantidade de quadro) até uma hora (seguindo o mesmo cálculo). Portanto, esse equipamento possibilita medir trechos pequenos, como uma soma, ou até atividades demoradas, como uma solicitação de reconhecimento de imagem.

Foi necessário modificar o nó sensor para permitir a medição no osciloscópio. A modificação consistiu na adição de um condutor (fio), conectado ao LED vermelho do nó sensor. Quando o LED está desligado, sinaliza para o osciloscópio começar a contabilizar o consumo de energia da aplicação. Quando o LED é ligado, sinaliza que a contabilização deve ser encerrada. Dessa forma, a aplicação pode indicar o trecho de código no qual o consumo de energia deve ser avaliado.

```
1: void medirRotina(){
2:  call Leds.led0Off();                //Desligar o led Vermelho
3:  call fazerUmaAtividade();
4:  call Leds.led0On();                 //Ligar o led Vermelho
5: }
```

Este trecho de código nesC é um exemplo de uma aplicação usada na medição. Na linha 2, a aplicação desliga o LED vermelho (indicado com a função `led0Off()`), sinalizando que a contabilização do consumo de energia deve ser iniciada. A linha seguinte executa a rotina em análise (função `fazerUmaAtividade`). Em seguida, a aplicação liga o LED vermelho (linha 4), sinalizando que a contabilização do consumo de energia deve ser encerrada. Assim, é possível medir o consumo de energia apenas do trecho do código relacionado à atividade em análise. Adicionalmente, essa função pode ser executada repetidamente, iniciando uma nova medição.

7.2 Operadores e Comandos

O objetivo desta seção é mostrar como nós medimos o consumo de energia de cada operador da linguagem nesC e dos comandos disponíveis no TinyOS *e.g.*, send, read. Os resultados do consumo de energia dos operadores avaliados são mostrados no Apêndice C. O consumo de energia coletado é usado para representar o operador ou comando no Modelo de Aplicação (Seção 5.2). A aplicação nesC, a seguir, foi utilizada para medir o consumo de energia de todos os operadores e comandos:

```
 1. implemention {
 2.   event void Boot.booted(){
 3.      call Timer.startPeriodic( 100 );
 4.   }
 5.
 6.   event void Timer.fired(){   // começa a medir
 7.      operator; //primeira vez
 8.      operator; //segunda vez
 9.      ...
10.      operator; //N - 1 vez
11.      operator; //N vez
12.   }                          // para de medir
13. }
```

Para reduzir a interferência no processo de medição, essa aplicação não habilita nenhum recurso, *e.g.*, rádio e LEDs, e repete a medição até atingir um erro médio especificado. O consumo de energia de um operador particular (*operator*) é obtido pela divisão do consumo de energia medido pelo número de repetições. Em todas as medições dos operadores, nós repetimos os operadores 30 vezes e medimos 100 vezes a aplicação. Por exemplo, se a média do consumo de energia das 100 avaliações foi igual a 6,3 joules e nós repetimos 30 vezes o operador, então, podemos concluir que o consumo de energia do operador é igual a 0,21 joule.

Uma abordagem similar foi usada para medir o consumo de energia dos recursos do nó sensor, *e.g.*, rádio e LEDs. No entanto, neste caso, uma aplicação foi executada com e sem um recurso particular habilitado. A diferença entre ambos os consumos de energia é o consumo de energia do recurso.

A linguagem nesC possui 34 operadores básicos, *e.g.*, atribuições, operadores matemáticos, manipuladores de bits, entre outros. Nós observamos que o consumo de energia dos operadores básicos variam de acordo com o tipo utilizado. Por exemplo, se somarmos duas variáveis do tipo `short` (2 bytes), o consumo de energia será menor do que se somarmos duas variáveis do tipo `long` (8 bytes). Observando isso, nós deveríamos fazer 272 medições para avaliar o consumo de energia de todos os operadores usando todos os

tipos. Devido à quantidade de medições para apenas avaliar os operadores, nós medimos apenas os operadores e os comandos que foram utilizados para validar o Modelo de Aplicação. Como mencionado anteriormente, os resultados do consumo de energia dos operadores avaliados são mostrados no Apêndice C.

7.3 Sugestões

O objetivo desta avaliação é validar as otimizações sugeridas pelo IDEA4WSN (Seção 6.3.3), mostrando a redução do consumo de energia quando elas são usadas. Para isso, foram criados seis cenários, um cenário para cada sugestão, nos quais cada um implementa duas aplicações: uma com e outra sem a otimização. O consumo de energia das duas aplicações foi medido 100 vezes.

O primeiro cenário, *Cenário-I*, avalia o consumo de energia dos parênteses. O segundo cenário, *Cenário-II*, avalia o impacto da estrutura de repetição `for`. O próximo cenário, *Cenário-III*, avalia a melhor maneira de implementar o *for*: com incremento ou decremento. O quarto cenário, *Cenário-IV*, avalia a ordem dos ramos da estrutura de seleção `if-then-else`. Por fim, o *Cenário-V*) e *Cenário-VI* avaliam duas atividades comuns na aplicação: a soma e a atribuição, respectivamente.

Como foi mencionado anteriormente, duas aplicações similares foram avaliadas no primeiro cenário para validar a primeira sugestão (*OT01*) apresentada na Seção 6.3.3. Essas aplicações calculam a média de cinco números, onde uma aplicação usa parênteses e outra usa duas atribuições para efetuar o cálculo:

Aplicação 1: RESULTADO = (A + B + C + D + E) / 5

Aplicação 2: TOTAL = A + B + C + D + E
RESULTADO = TOTAL / 5

A Figura 7.1 mostra o consumo de energia do *Cenário-I*. A diferença da média do consumo de energia entre as duas aplicações é de 62,46%. Foi necessário garantir que essa diferença exista e, por isso, foi usado o teste de hipótese. Esse teste retornou um p-valor igual a 1.08015 x 10^{-53}, significando que ambas as aplicações de fato consumiram valores diferentes. Dessa maneira, é possível concluir que é melhor usar duas atribuições em vez de uma única com parênteses. Possivelmente, o parêntese força uma mudança de prioridade na ordem de execução das instruções do processador, aumentando o consumo de energia. Por exemplo, se a aplicação não o usasse, o processador iria dividir antes de somar os cinco números. Quando se usa parênteses, o processador é forçado a somar primeiro os cinco números e, depois, dividi-los por cinco.

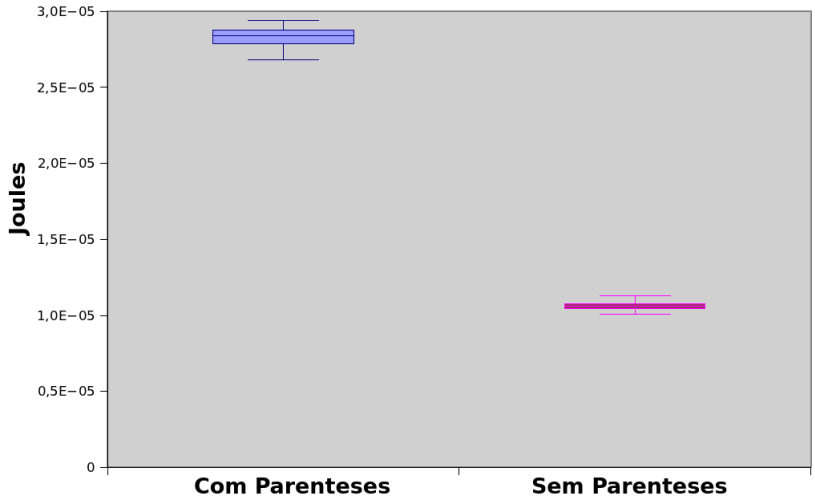


Figura 7.1 Consumo de energia do *Cenário-I*.

O segundo cenário está relacionado à estrutura de repetição `for` (Seção 2.1.2) e com a segunda sugestão (*OT02*) do editor (Seção 6.3.3), uma estrutura que executa um conjunto de operadores várias vezes até a expressão de controle retornar `false`. Essa estrutura de repetição possui três expressões: (1) a primeira expressão é associada à inicialização e executa apenas uma única vez; (2) a segunda expressão é o controle, o qual é verificado antes de executar o corpo; e, finalmente, (3) a última expressão usualmente incrementa algum valor.

Para avaliar o impacto dessa estrutura, duas aplicações foram implementadas: a primeira utiliza um `for` para inicializar um *array* e a segunda inicializa as posições deste mesmo *array* individualmente.

Os resultados apresentados na Figura 7.2 mostram que a diferença média entre o consumo das duas aplicações é de 71,85%. O teste de hipótese foi aplicado para mostrar se há ou não diferença entre eles. O p-valor retornado foi igual a 1,85816 x 10^{-65}, confirmando que as aplicações consumiram valores diferentes. Desta maneira, nós podemos concluir que a estrutura de repetição aumentou consideravelmente o consumo de energia. Isso ocorreu porque essa estrutura de repetição tem três expressões, das quais duas são executadas várias vezes. O consumo de energia das três expressões juntas pode ser representado pela seguinte equação:

$$E_{exp} = E_{init} + nE_{control} + (n-1)E_{inc}, \tag{7.1}$$

onde E_{init}, $E_{control}$ e E_{inc} são o consumo de energia da primeira, segunda e terceira expressões, respectivamente; n é igual à quantidade de vezes que o corpo será executado.

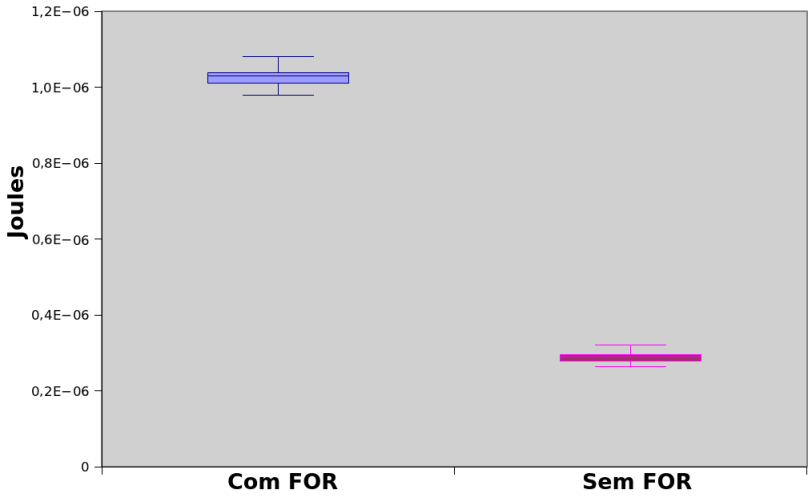


Figura 7.2 Consumo de energia do *Cenário-II*.

O ganho pode ser mais expressivo caso as expressões sejam mais complexas.

Algumas vezes, é necessário usar a estrutura de repetição `for`, por exemplo, para inicializar um *array* com tamanho fixo em tempo de execução. Para esses casos, foi investigado no *Cenário-III* a melhor maneira de usar essa estrutura de repetição. Tal como os outros cenários, foram avaliadas duas aplicações: uma usando o incremento e outra usando o decremento. Essas aplicações inicializam uma *array* com tamanho pré-definido igual a 20, atribuindo o valor zero a todas as posições.

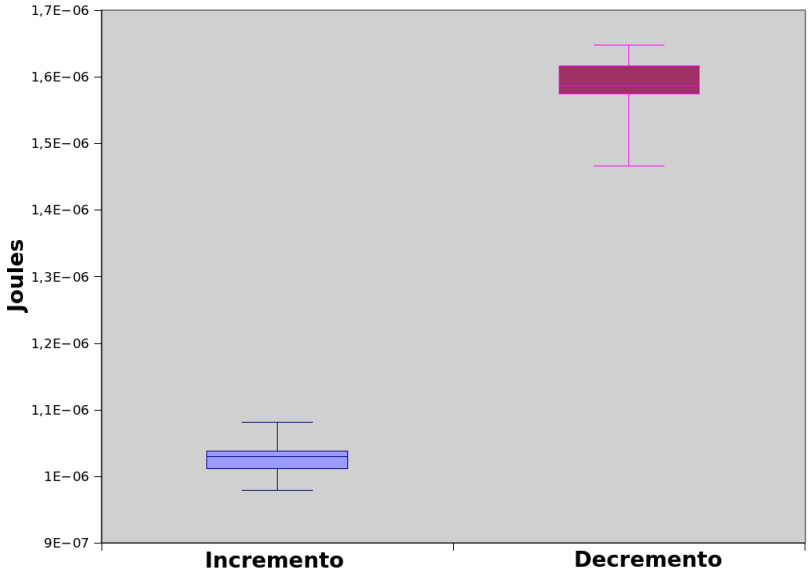


Figura 7.3 Consumo de energia do *Cenário-III*.

A Figura 7.3 mostra o consumo de energia das duas aplicações avaliadas nesse

cenário. Neste caso, a diferença entre as médias foi próxima de 54,78% e o teste de hipótese retornou um p-valor igual a 7,7639 x 10^{-49}, mostrando que os valores retornados foram diferentes. Isso significa dizer que o incremento consome menos energia do que o decremento. Possivelmente, a razão para isso ocorrer deve ser uma otimização feita pelo processador ou pelo compilador, visto que o incremento é mais usado do que o decremento.

Outra estrutura muito usada em qualquer aplicação escrita na linguagem nesC é o `if-then-else`. Essa estrutura de seleção possui vários ramos. Cada um possui uma expressão de controle, exceto a estrutura `else`. Essas expressões verificam se executam o seu corpo ou passam para o próximo ramo. Uma das formas de melhorar o consumo energético do `if-then-else` é mudar a ordem de execução desses ramos e aqueles com maior probabilidade de serem executados devem vir primeiro (Seção 6.3.3).

Para testar se a ordem dos ramos influencia no consumo de energia, foram criadas duas aplicações que possuem uma estrutura de seleção *if-then-else* com cinco ramos. Dentre eles, quatro (A, B, C e D) ocorrem 5% das vezes e um ramo (E) ocorre 80% das vezes. Por exemplo, se essas aplicações forem executadas vinte vezes, o ramo E será executado dezesseis vezes e os outros ramos serão executados apenas uma única vez. A diferença entre as duas aplicações é a ordem dos ramos. A primeira aplicação possui o ramo *E* como sendo o último a ser verificado. Enquanto a segunda aplicação possui o mesmo ramo como sendo o primeiro.

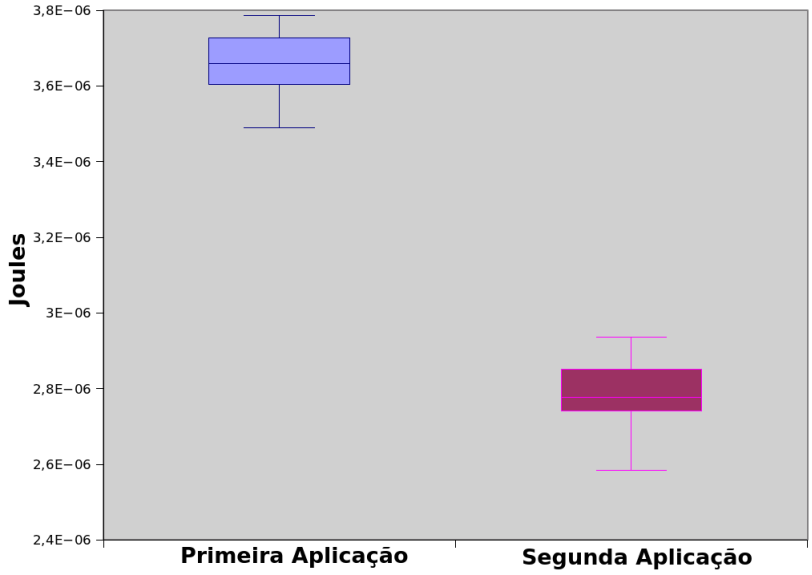


Figura 7.4 Consumo de energia do *Cenário-IV*.

O resultado do consumo de energia desse cenário é ilustrado na Figura 7.4. A

diferença entre a primeira e a segunda aplicação é de 23,71%. Similar aos outros cenários, foi usado o teste de hipótese que retornou um p-valor igual a 3,1065 x 10^{-44}. Esse valor significa dizer que a segunda aplicação consumiu menos energia do que a primeira aplicação. Isso evidencia a importância da organização dos ramos, ordenando-os pela probabilidade de serem executados.

O ganho do consumo de energia da segunda aplicação pode ser representado pelo seguinte cálculo:

$$E_{exp} = \sum_{i=0}^{n} E_i, \tag{7.2}$$

no qual E_i é o consumo de energia da expressão do ramo e n é o número de ramos anteriores. Por exemplo, n é igual a quatro no caso da primeira aplicação. O consumo de energia de cada expressão de cada ramo (E_i) é igual à 2,17 x 10^{-07}. Então, o ganho final será de (E_{exp}) e igual a 8,68 x 10^{-07}.

O *Cenário-V* avalia três possibilidades oferecidas pela linguagem nesC para incrementar o valor 1 em uma variável. Esse experimento está relacionado à quarta sugestão (*TO04*) apresentada na Seção 6.3.3. Para isso, este cenário avalia três possibilidades de incremento: *Tipo1* (`var++`), *Tipo2* (`var += 1`) e *Tipo3* (`var = var + 1`). Foram criadas três aplicações, em que cada uma irá usar um tipo descrito anteriormente.

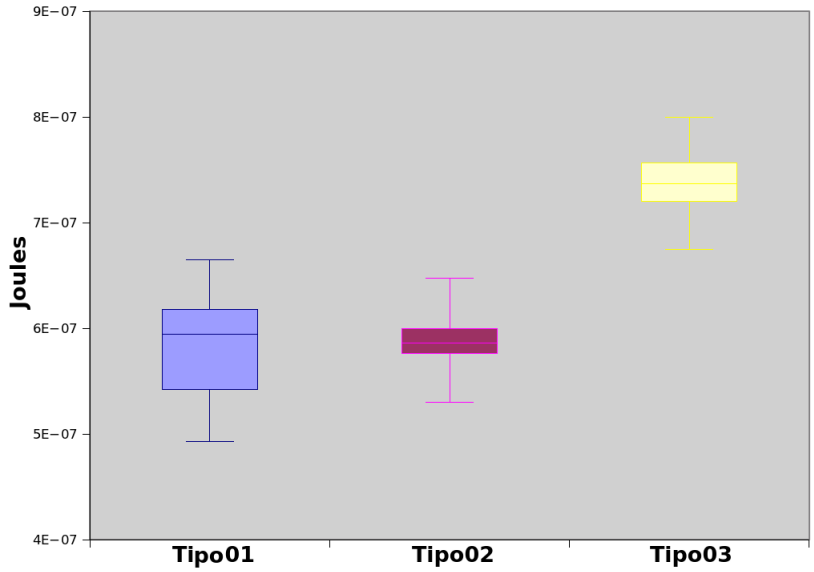


Figura 7.5 Consumo de energia do *Cenário-V*.

O resultado desse cenário é ilustrado na Figura 7.5. O *Tipo1* tem um consumo de energia menor entre as três possibilidades. No entanto, o *Tipo2* tem uma média próxima

ao *Tipo1*, a diferença entre eles foi de 1,23%. Como os valores foram muito próximos, foi decidido, portanto, usar o teste de hipótese para saber se de fato existe essa diferença. Ele retornou um p-valor igual a 0,4632; o que significa dizer que não existem evidências suficientes para informar que esses dois tipos são diferentes. Certamente, o compilador ou o processador executa o *Tipo1* e o *Tipo2* da mesma forma.

De todos os tipos, o *Tipo3* teve o pior consumo de energia. A diferença para o *Tipo1* é de 20,82% e do *Tipo02* é de 19,83%. Similar ao cenário anterior, foi utilizado o teste de hipótese, o qual retornou um p-valor igual à 5,5745 x 10^{-20} e 8,2415 x 10^{-28} quando é comparado com o *Tipo1* e o *Tipo2*, respectivamente. Em ambos os casos significa dizer que o *Tipo3* de fato é o pior caso para incrementar o número 1 a uma variável. Possivelmente, o processador acessa mais vezes a memória (diferente do *Tipo1* e do *Tipo2*).

O último cenário avalia outra atividade comum em uma aplicação: atribuir um valor qualquer a uma variável (Seção 6.3.3). O *Cenário-VI* avalia três modos: constante, valor explícito ou outra variável. A primeira alternativa é atribuir o valor de uma constante (*e.g.*, `var = CONST`), a qual não pode mudar o seu valor logo após ser criada. A segunda maneira é atribuir um valor explicitamente (*e.g.*, `var = 1`). E a última alternativa é atribuir o valor de uma outra variável (*e.g.*, `var = var2`).

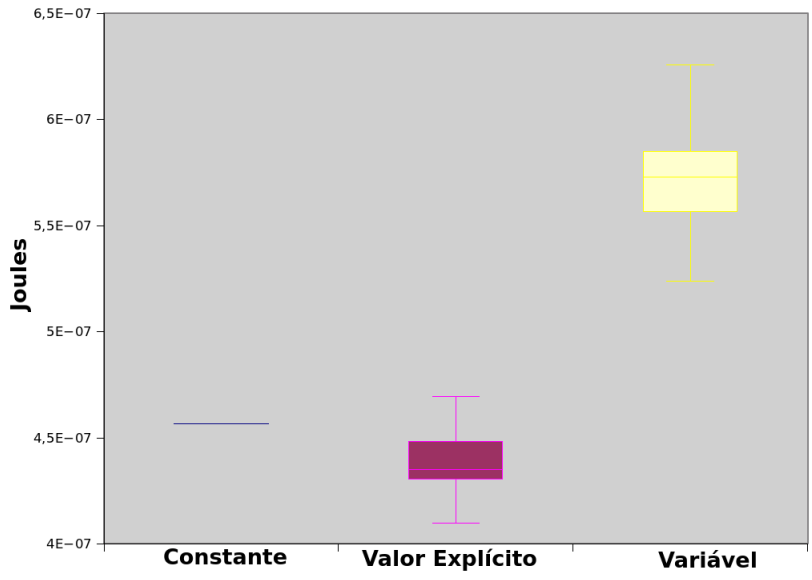


Figura 7.6 Consumo de energia do *Cenário-VI*.

A Figura 7.6 mostra o consumo de energia das três alternativa. A constante e o valor explícito obtiveram praticamente o mesmo valor. A diferença entre eles foi de 0,06%. Como as médias foram praticamente iguais, foi decidido usar o teste de hipótese que

retornou um p-valor igual a 0,9365. Isso significa dizer que não existem evidências suficientes para afirmar que eles são diferentes. Ou seja, o consumo deles dois foi o mesmo. Provavelmente, o compilador substitui o nome da constante pelo valor definido, diminuindo o acesso à memória pela aplicação em tempo de execução.

A terceira maneira, atribuir o valor de uma outra variável, obteve o pior consumo de energia quando comparados com a constante (diferença de 23,31%) e valor explícito (23,27%). Tal como foi feito anteriormente, foi usado o teste de hipótese que retornou um valor igual a 7,82489 x 10^{-33} quando comparado à constante, e 4,7263 x 10^{-34}, quando comparado com o valor explícito. Certamente, a aplicação acessa mais a memória para recuperar o valor da outra variável antes de atribuir esse valor.

7.4 Modelo de Aplicação

O foco dessa seção é validar o Modelo de Aplicação comparando os resultados do consumo de energia obtidos por medição e simulação. Para isso, foram implementadas cinco aplicações escritas em nesC e gerados seus respectivos modelos CPN usando o tradutor Sensor2Model (Seção 6). A primeira aplicação realiza uma atividade simples usando apenas operadores da linguagem nesC (Seção 2.1.2), mais especificamente, essa aplicação (*App01*) calcula a média de cinco números. A segunda aplicação (*App02*) tem uma invocação para coletar a temperatura, uma atividade típica na RSSF. Já a terceira aplicação (App03) usa uma estrutura de seleção (Seção 2.1.2) para agregar valores capturados. A quarta aplicação (App04) calcula a média de cinco valores capturados (App01) usando uma estrutura de interação. A quinta e última aplicação (App05) coleta a temperatura e a envia ao nó sorvedouro.

Foi avaliado um pedaço pequeno do código das aplicações *App01*, *App02*, *App03* e *App04*, enquanto a App05 foi avaliada por inteira. Esses experimentos servem para mostrar como a ferramenta e o modelo proposto são flexíveis de tal modo que eles permitem avaliar um pedaço pequeno de código ou uma aplicação inteira. Todas as aplicações foram avaliadas 100 vezes usando a simulação e a medição.

Antes de apresentar os resultados, é necessário introduzir o critério de parada adotado nas simulações.

7.4.1 Critério de Parada

A simulação do Modelo de Aplicação proposto é estacionária e leva um longo tempo para ser executada. Na prática, é necessário definir um critério de parada para finalizar a

simulação. A *batch means* é uma da melhores soluções para resolver esse problema e é muito utilizada.

A *batch means* divide a simulação em partes, *batch*, com o mesmo tamanho. Depois, ela calcula a média dos *batch* e constrói uma lista com essas médias. Dessa forma, é possível determinar o erro da simulação através dessa lista. O erro é calculado como mostra a seguinte equação:

$$R = \frac{t^{(k-1,1-\alpha)}S}{\sqrt{k}}, \tag{7.3}$$

onde, t é o valor da distribuição T-Student com grau de liberdade *K-1* e nível de confiança (definida pelo usuário) igual a *1-*α, k é o número de elementos dentro da lista e S é o desvio padrão das médias dos valores da lista. É possível comparar o erro do *batch means* com o valor definido pelo usuário, chamado de "erro máximo". A simulação para quando o E é menor ou igual ao erro máximo.

7.4.2 Resultados

A primeira aplicação, App01, é mostrada a seguir:

```
 1. implemention {
 2.  int8_t mean = 0, a = 8, b = 16, c = 32, d = 64, e = 127;
 3.
 4.  event void Boot.booted(){
 5.    call Timer.startPeriodic( 100 );
 6.  }
 7.
 8.  event void Timer.fired(){              // começa a medição
10.    mean = ( a + b + c + d + e ) / 5;
11.  }                                      // para a medição
12. }
```

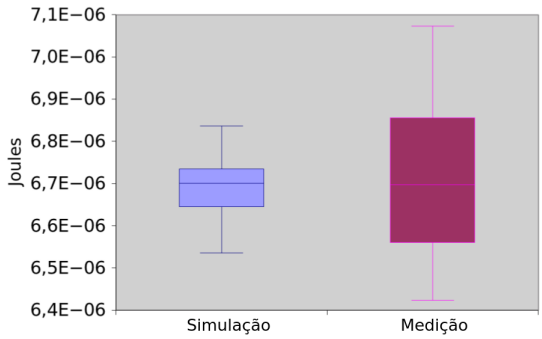


Figura 7.7 Consumo de energia da aplicação App01.

Essa aplicação deveria ter apenas uma única atribuição (Seção 2.1.2). No entanto, essa única atribuição não afetaria o consumo de energia do nó sensor, porque o compilador iria executar uma otimização descartando essa variável por não ser utilizada pela aplicação; consequentemente, reduzindo o consumo de energia da aplicação. O compilador não considera como variável usada quando apenas atribuímos valores a ela. Então, a aplicação *App01*, cujo propósito é avaliar uma atribuição, também inclui operadores aritméticos (+ e /). O consumo de energia desses operadores é mostrado no Apêndice C.1.

A Figura 7.7 mostra o consumo de energia da *App01* quando avaliada pelo modelo e pela medição. A diferença da médias obtidas pelo Modelo de Aplicação e pela medição é de 0,35%. Como o resultado foi próximo, foi decidido verificar se essas médias são significativamente diferentes. O teste de hipótese retornou um p-valor igual a 0,520920. Isso significa dizer que não existem evidências suficientes para afirmar que o modelo e a medição retornam resultados diferentes. Vale ressaltar que a medição variou mais do que o Modelo de Aplicação, porque otimizações feitas em tempo de execução e processamento extra no nó sensor podem ter ocorridos. No entanto, mesmo a medição variando mais, os valores obtidos pelo Modelo de Aplicação e pela medição foram similares e a média foi próxima.

A segunda aplicação, *App02*, simplesmente coleta uma temperatura:

```
1. ...
2. void readTemperature(){ //começa a medição
3.   call Temp.read();
4. }
5.
6. event void Tem.readDone(error_t err, float value){
7.   // do nothing
8. }                      //para a medição
9. ...
```

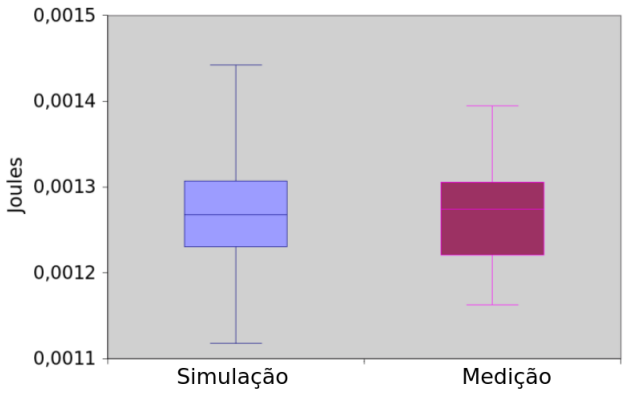


Figura 7.8 Consumo de energia da aplicação App02.

A linha 1 possui um código similar à *App01*. *App02* invoca o comando `read` (linha 3) para coletar uma temperatura e o evento `readDone` é disparado quando termina de ler a temperatura. O consumo de energia deste comando é mostrado no Apêndice C.2.

A Figura 7.8 ilustra que os resultados obtidos pela medição e pela simulação são similares, tendo uma diferença próxima de 0,19% na média das avaliações. O teste de hipótese, retornando um p-valor igual a 0,8276, confirmou que não existe diferença significativa entre o Modelo de Aplicação e a medição.

App03 tem uma função que usa três variáveis para agregar um valor ao longo do tempo com uma estrutura de seleção (`if-then-else`):

```
 1. ...
 2. void aggregate(int8_t value){  //começa a medição
 3.  total = total + value;
 4.  size = size + 1;
 5.
 6.  if( size == 5 ){ //@0.20
 7.   mean = total / 5;
 8.   total = 0;
 9.   size = 0;
10.  }
11. }                              //para a medição
12. ...
```

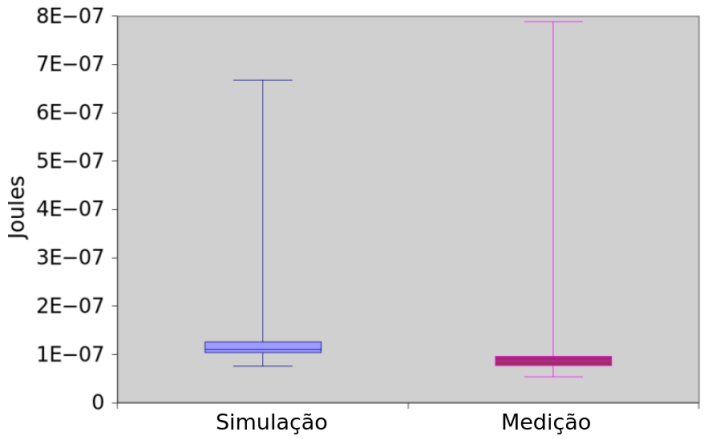


Figura 7.9 Consumo de energia da aplicação App03.

O código de inicialização e as declarações das variáveis são similares à *App01*. A função `aggregate()`, como comumente é encontrada nas aplicações da RSSF, faz um cálculo (*e.g.*, média) com os valores capturados. Mais detalhadamente, cada vez que esta função é executada, a variável `size` é incrementada (linha 4). Quando esta função for executada pela quinta vez (linha 6), ela irá agregar os valores capturados.

Deste modo, a estrutura de seleção `if-then-else` tem associado a probabilidade de 20% (`//0.20`), porque isso ocorre uma vez a cada cinco execuções (na linha 6, `size == 5`). O consumo de energia dos operadores utilizados nesta aplciação é mostrado no Apêndice C.3.

Os resultados obtidos pela simulação e pela medição estão ilustrados na Figura 7.9. A diferença entre os dois métodos foi de 23,30%. Novamente, foi aplicado o teste de hipótese que retornou o valor 0,115351. Logo, este valor mostra que não existem evidências suficientes para assegurar que os dois métodos retornaram valores diferentes.

A quarta aplicação, *App04*, é mostrada a seguir:

```
1 ...
2 void mean(){                        //começa a medição
3  total = 0;
4  for(i = 0; i < 5; i++){ //@5.0
5    total = total + value[i];
6  }
7  mean = total / 5;
8 }                                   //para a medição
```

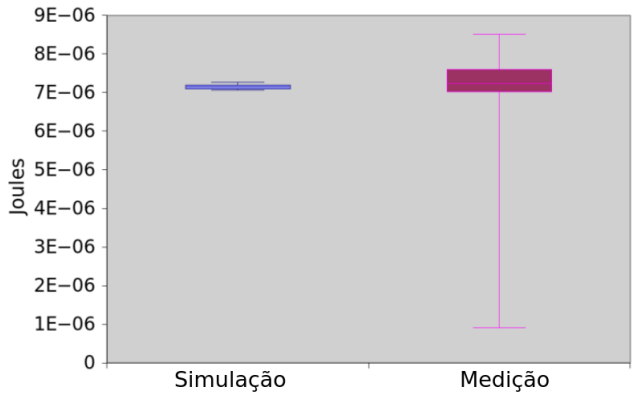


Figura 7.10 Consumo de energia da aplicação App04.

Similarmente à *App01*, esse programa calcula a média de cinco números, mas ele usa uma estrutura de repetição (`for`) para somá-los (4-6). A Figura 7.10 mostra o consumo de energia obtido através da simulação e da medição. Nesse caso, a diferença entre eles foi de 3,09% e o teste de hipótese retornou um p-valor igual a 0,072628. Portanto, não existem evidências mostrando que a simulação e a medição retornaram valores diferentes. Adicionalmente, o consumo de energia dos operadores utilizados nesta aplicação está no Apêndice C.4.

Por fim, a quinta e última aplicação, *App05*, coleta a temperatura e a envia para outro nó sensor, como é ilustrada abaixo:

```
 1. implemention{
 2.   message_t pckt; msg_t* msg;
 3.
 4.   event void Boot.booted(){
 5.      msg = (msg_t*) call Packet.getPayload( &pckt , size( msg_t ) );
 6.      call RadioControl.start();
 7.   }
 8.
 9.   event void RadioControl.startDone(error_t e){
10.      call Timer.startOneShot( 100 );
11.   }
12.
13.   event void Timer.fired(){
14.      call Temp.read();
15.   }
16.
17.   event void Temp.readDone(error_t e, float value){
18.      msg->error = e;
19.      msg->value = value;
20.      call RadioSender.send( BASE_STATION , &pckt , 28 );
21.   }
22.
23.   event void RadioSender.sendDone(error_t e, message_t p){
24.      call Timer.startOneShot( 100 );
25.   }
26. }
```

O comportamento dessa aplicação consiste em ligar o rádio na inicialização (4-7). Quando o rádio está funcionando, a aplicação coleta a temperatura (13-15) a cada 100 milissegundos (9-11). Depois, a aplicação cria uma mensagem e envia o valor coletado para o nó sorvedouro (17-21). Esse comportamento é repetido a cada 100 milissegundos (23-25) em um laço.

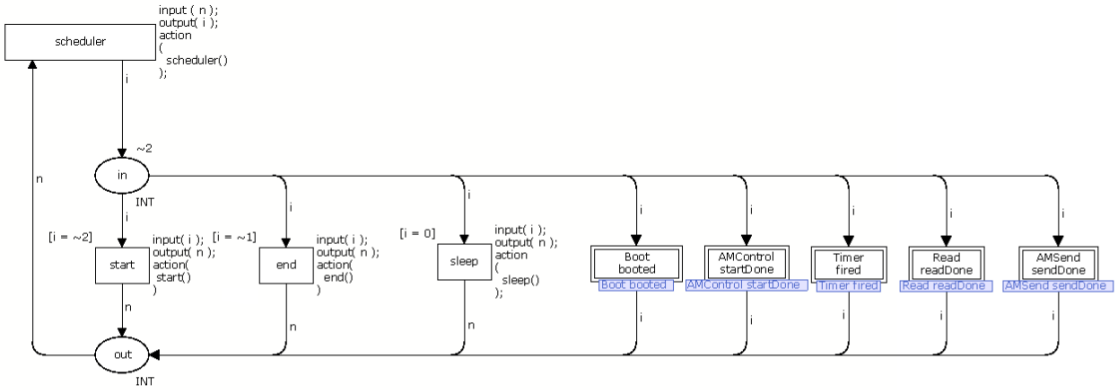


Figura 7.11 Consumo de energia da aplicação App05.

A Figura 7.11 mostra o Modelo Principal (Seção 5.2.3) da *App05*. O consumo de

energia dos operadores e comandos utilizados nesta aplicação é mostrado no Apêndice C.5.

Os resultados obtidos pelos dois métodos são apresentados na Figura 7.12. A diferença entre eles foi de 6,71%. Tal como as aplicações anteriores, foi utilizado o teste de hipótese que retornou um p-valor igual a 0,083238. Portanto, é possível concluir que, para esse experimento, não existem evidências suficientes para afirmar que os resultados são diferentes.

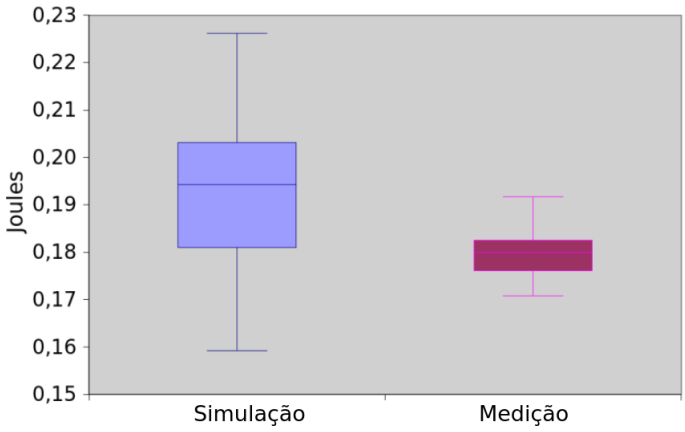


Figura 7.12 Consumo de energia da aplicação App05.

Mesclando os resultados obtidos com duas aplicações anteriores (*App2* e *App5*), nós podemos identificar o impacto do consumo de energia do transmissor e o impacto do consumo de energia da aplicação no nó sensor. Para isso, é necessário fazer três considerações. A segunda aplicação e a quinta aplicação são parecidas (1° Consideração). A diferença entre elas é a inclusão da transmissão de dados e alguns operadores para criar a mensagem, a qual será usada para transmitir o dado. Como os operadores consomem pouca energia, nós podemos considerar que a diferença do consumo de energia entre as duas aplicações é influenciada principalmente pela transmissão do dado (2° Consideração). Arrendondado para cima a média dos valores obtidos com o modelo, nós podemos considerar que a segunda aplicação (aplicação) consumiu 0,0013 joules e a quinta aplicação (aplicação + transmissor) consumiu 0,20 joules (3° Consideração). Desta maneira, nós concluímos que o transmissor consumiu 0,1987 joules, aumentando o consumo de energia em 153 vezes e equivalendo a 99,35% do consumo de energia do nó sensor. Esse tipo de comparação mostra que o processo de transmissão (criar pacote + transmissor ligado + envio do pacote) consome mais energia do que a aplicação, como já é dito na literatura (Akyildiz *et al.*, 2002).

7.5 Modelo de Rede e Modelo de Nó Sensor

Essa seção tem dois objetivos principais: validar a geração automática dos Modelos da Rede e do Modelo de Nó Sensor e mostrar o impacto da aplicação no tempo de vida da RSSF. Para alcançar esses objetivos, foram definidos quatro experimentos:

- **Experimento I**: mostra o tempo de vida da RSSF obtido quando usa o Modelo de Rede e o compara com resultados encontrados na literatura. Foram utilizados quatro protocolos da camada de rede : LEACH, GOSSIPING, DIRECT e FLOODING. Nesse experimento, o consumo de energia de cada nó sensor da RSSF é apenas da camada de rede e nenhuma aplicação é executada neles.

- **Experimento II**: mostra o impacto da aplicação no tempo de vida da rede considerando diferentes protocolos da camada de rede. Aqui foram utilizados os mesmos protocolos usados no *Experimento I*. Nesse experimento, o consumo de energia do nó sensor é devido à aplicação e à pilha de protocolos.

- **Experimento III**: mostra o impacto de diferentes aplicações usando o mesmo protocolo. Tal como o *Experimento II*, o nó sensor contabiliza o consumo de energia da aplicação e da pilha de protocolo.

- **Experimento IV**: mostra o impacto do consumo de energia da rede e o consumo de energia da aplicação na RSSF.

Diferente da validação do Modelo de Aplicação, o Modelo de Rede foi validado comparando os resultados obtidos por ele com os resultados encontrados na literatura. Isso ocorreu por duas dificuldades com a medição: (1) alto custo para adquirir todos os equipamentos (por exemplo, 20 nós sensores) e (2) a dificuldade de medir o consumo de energia de mais de um nó sensor simultaneamente. Como avaliar uma aplicação é simples, esses problemas acabam não aparecendo. No entanto, para avaliar uma rede, eles dificultam ao ponto de tornar a medição extremamente complexa. Aliado a isso, existem vários estudos sobre os protocolos de rede na literatura, o que permitiu escolher e comparar os resultados com um desses estudos.

Antes de apresentar os resultados, é necessário introduzir o critério de parada e a configuração adotados nas simulações.

7.5.1 Critério de Parada

Em todos os experimentos foram adotados dois tipos de critério de parada: um relacionado ao tempo de simulação e outro à confiabilidade dos dados obtidos através da simulação. O primeiro tipo define quando a simulação deve parar. Por exemplo, a simulação deve parar quando o primeiro nó sensor da RSSF estiver com a bateria exaurida. Esse critério de parada é conhecido como FND (*First Node Dies*). Além dele, existem os critérios quando o último nó sensor morrer (LND), quando a metade da rede morrer (HND) ou quando um percentual dela ficar sem bateria (PND).

Por outro lado, o segundo critério de parada verifica se os resultados obtidos são confiáveis, ou não, logo após parar a simulação. No caso da confiabilidade não ser alcançada, a simulação é executada novamente. O critério adotado foi o *batch means*, apresentado na Seção 7.4.1.

7.5.2 Configuração

O Modelo de Rede usa o *First Order Radio Model* (Heinzelman *et al.*, 2000), o qual contabiliza o consumo de energia quando o nó sensor envia e recebe um pacote. Nesse caso, os consumos de energia da CPU, EEPROM e LEDs não são considerados.

Tabela 7.1 Configuração da simulação do consumo de energia da RSSF.

Propriedade	Valor
Protocolos de roteamento	DIRECT, FLOODING, GOSSIPING e LEACH
Nós sensores	20
Nó Sorvedouro	1
Areá	400 x 400 m
Raio de comunicação	200 m
Nível de Energia	1 mJ (Experimento I e II) e 10 mJ (Experimento III)
Periodicidade de envio	a cada 2 segundos
Distribuição dos nós sensores	distribuídos randomicamente entre 10 a 50 metros
Repetições	30 vezes
Modelo de Energia	First Order Radio Model
E_{Tx}	50 nJ/bit
E_{Rx}	50 nJ/bit
E_{Amp}	100 pJ/bit/m^2

A rede consiste em vinte (20) nós sensores e um (1) nó sorvedouro. Os nós sensores da RSSF são distribuídos randomicamente em uma área de 400 x 400 m. Os nós sensores estão no intervalo de 10 a 50 m do nó sorvedouro. O alcance do rádio é de

200 m e o nível de energia é de 1mJ no primeiro experimento e 10J nos demais. Com relação aos protocolos adotados, foram selecionados os protocolos DIRECT, FLOODING, GOSSIPING e LEACH por serem amplamente documentados e usados em RSSFs. Eles possuem as mesmas configurações apresentadas por Senouci *et al.* (2012) e tais informações estão sumarizadas na Tabela 7.1.

Além disso, nós repetimos 30 vezes todos os experimentos. Nós utilizamos os valores padrões do *First Order Radio Model* para avaliar o consumo de energia dos nós sensores: o transmissor eletrônico (E_{Tx}) e o receptor eletrônico (E_{Rx}) consomem 50 nJ/bit e amplificador do transmissor ((E_{Amp}) consome 100 pJ/bit/m^2.

7.5.3 Resultados

No primeiro experimento, similar a Senouci *et al.* (2012), a aplicação envia um pacote a cada 2 segundos e o nó sensor da RSSF possui uma energia inicial de 1 mJ.

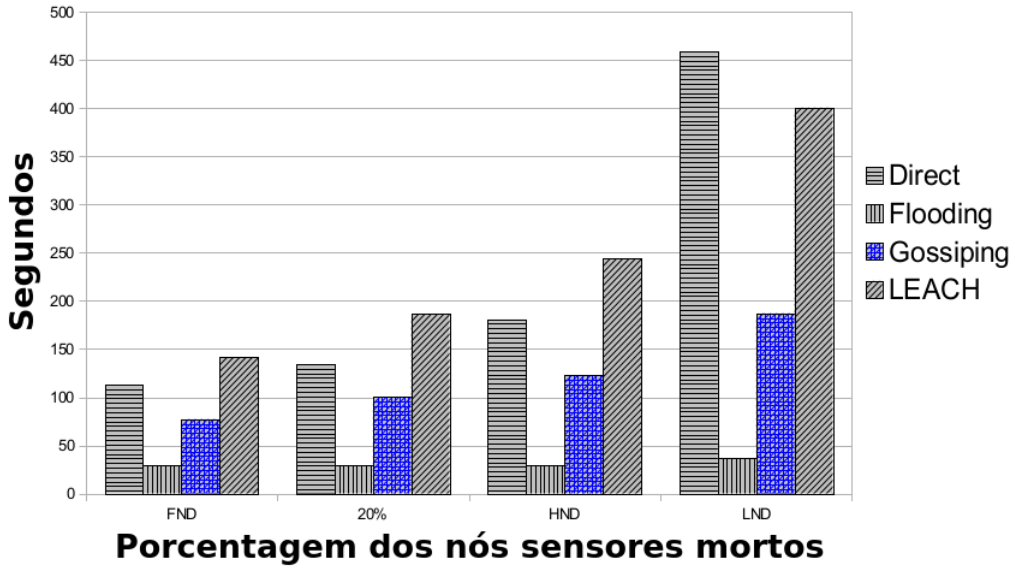


Figura 7.13 Tempo de vida da RSSF no *Experimento I*.

A Figura 7.13 mostra o percentual de nós sensores mortos (devido à energia ter acabado) durante o tempo obtido através da simulação do modelo. Como esperado, o FLOODING tem o pior desempenho. O GOSSIPING teve um desempenho melhor do que o FLOODING, próximo de 3x e 6x para os critérios de paradas FND e LND, respectivamente. Isso ocorreu porque o FLOODING envia um pacote a todos os nós sensores, enquanto o GOSSIPING o repassa para apenas um nó aleatoriamente. O protocolo DIRECT teve um desempenho pior do que o LEACH nos dois primeiros

critérios (FND e PND igual a 20%), porque os nós sensores mais afastados morrem mais cedo (Senouci *et al.*, 2012; Li *et al.*, 2011). Por outro lado, se os nós sensores mais distantes usassem os nós sensores mais próximos (múltiplos saltos) do nó sorvedouro em vez de enviar os dados diretamente, os nós sensores mais próximos do nó sorvedouro morreriam mais rápido. Este problema é conhecido como *energy hole* (Li *et al.*, 2011).

No entanto, nos critérios de parada HND e LND, ele teve um desempenho melhor. Certamente, a distribuição não uniforme dos *Cluster Heads* (CHs) afeta negativamente o consumo de energia da RSSF. Por exemplo, o número de CH não é o mesmo durante a simulação e ele dificilmente mantém os CHs presentes em todas as partes da rede. Consequentemente, alguns nós sensores da RSSF gastam mais energia por não existir um CH próximo. Esses resultados são similares ao apresentado por Senouci *et al.* (2012), exceto no critério HND, no qual o protocolo DIRECT teve um desempenho melhor do que o LEACH. Certamente, isso ocorreu por dois motivos: a quantidade de simulações executadas no nosso experimento (30 vezes) e no experimento de Senouci *et al.* (2012) (não informado) pode ter sido diferente; e o nosso experimento, cada vez que a RSSF era avaliada, os nós sensores mudavam de posição visto que a posição deles era randômica (Senouci *et al.* (2012) não mencionou se foi feito desta forma também). Por estes dois motivos, o critério HND foi diferente nos dois estudos. Como nós avaliamos a RSSF 30 vezes, observamos que, em algumas dessas avaliações, o LEACH obteve um consumo de energia melhor do que o LEACH, obtendo o mesmo resultado que Senouci *et al.* (2012). Adicionalmente, o Apêndice F mostra os mapas de energia dos protocolos utilizados neste experimento.

Como mencionado anteriormente, o *Experimento II* simula uma aplicação com diferentes protocolos da camada de rede e avalia o impacto de cada protocolo no tempo de vida da RSSF. Essa aplicação coleta a temperatura a cada dois segundos e envia-o para o nó sorvedouro. A configuração desse experimento é similar ao anterior, exceto nível inicial da bateria igual a 10 J. A utilização do nível inicial de 10 J e não 1 mj, como no primeiro experimento, foi necessário porque o nó sensor da RSSF considera o consumo de energia da aplicação e da pilha de protocolo. A Figura 7.14 mostra os nós sensores mortos ao longo do tempo do *Experimento II*.

O consumo de energia dos nós sensores da RSSF tem um comportamento similar ao *Experimento I*. Isso significa dizer que os protocolos que tiveram um bom desempenho no *Experimento I* continuam tendo o mesmo padrão no *Experimento II*, mesmo considerando o consumo de energia da aplicação. Portanto, a integração dos modelos da Aplicação e da Rede (Seção 5.3.4) preserva o padrão do consumo de energia da comunicação encontrado

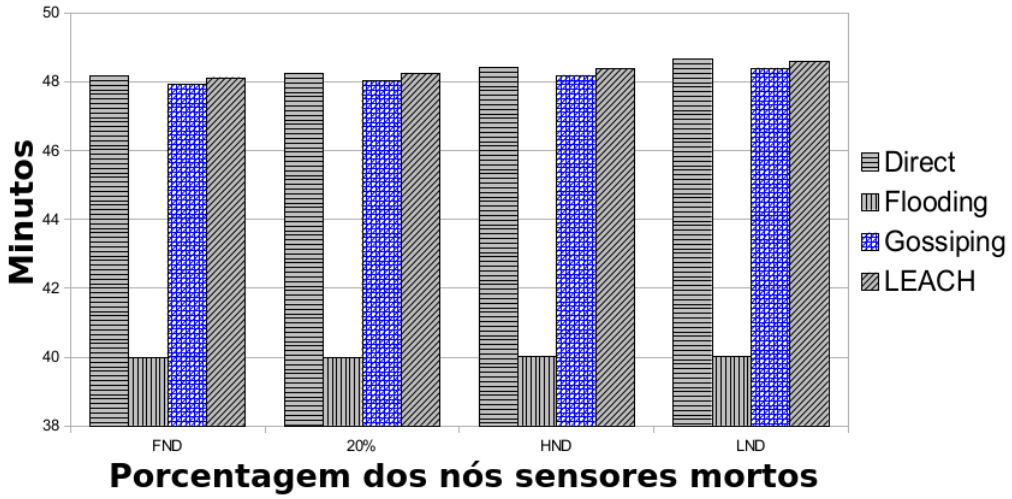


Figura 7.14 Tempo de vida da RSSF no *Experimento II*.

no *Experimento I*.

O terceiro experimento foi criado para avaliar o impacto da aplicação no tempo de vida da RSSF. Por conta disso, foram desenvolvidas três aplicações que coletam a temperatura (a cada dois segundos) e enviam para o nó sorvedouro (similar ao *Experimento II*). A primeira aplicação (*App01*) envia a temperatura imediatamente após capturá-la. A segunda aplicação (*App02*) envia a média de cinco valores coletados. A última aplicação (*App03*) apenas envia uma informação quando a temperatura excede um determinado valor. Nesse caso, foi considerado que 30% das vezes a temperatura excede esse valor. Dessa forma, o protocolo DIRECT foi escolhido por ter o melhor desempenho nos experimentos anteriores.

Como esperado, *App01* tem o pior consumo de energia (Figura 7.15) porque ele envia um pacote imediatamente depois de coletar a temperatura. *App03* teve um desempenho melhor em todos os critérios devido a dois fatores: (1) ele tem menos processamento depois de capturar a temperatura e (2) ele apenas envia um pacote quando é necessário. *App02* teve desempenho moderado por conta da agregação dos dados, o que ajudou a economizar energia. No entanto, o benefício da agregação poderia ter sido maior caso tivesse agregado mais dados em vez de apenas cinco.

O último experimento mostra o impacto da rede e da aplicação no consumo de energia da RSSF. Este experimento consiste de 1 aplicação que coleta a temperatura e envia para o nó sorvedouro. Ela envia periodicamente a cada 2 segundos e consome, em média, 0,0013 J a cada 2 segundos. O valor do consumo de energia da aplicação é o mesmo da segunda aplicação utilizada para validar o Modelo de Aplicação. A quinta aplicação

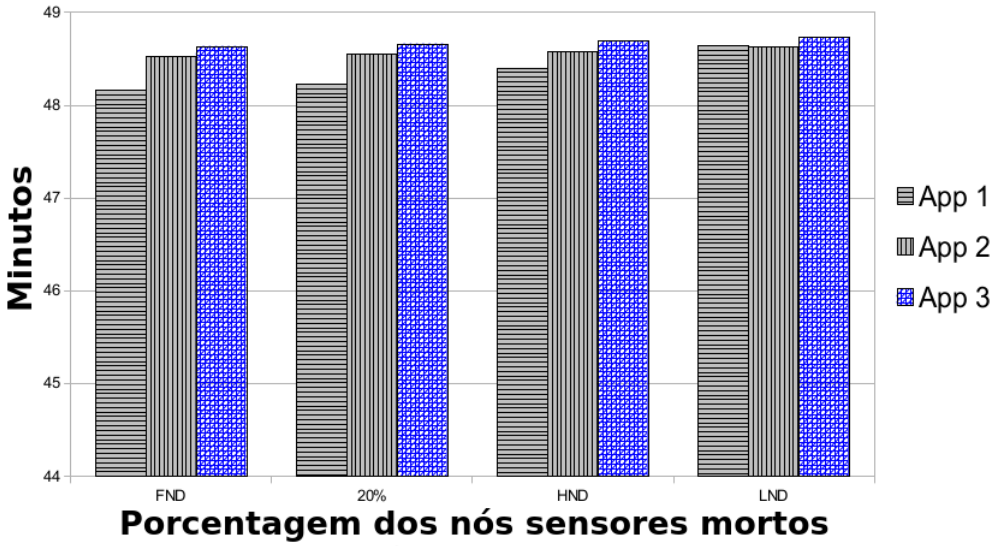


Figura 7.15 Tempo de vida da RSSF no Experimento III.

(Modelo de Aplicação) não foi utilizada porque o resultado do seu consumo de energia considera o consumo de energia da aplicação e da rede juntas, neste caso, queremos apenas o consumo da aplicação. O protocolo utilizado foi o DIRECT, por ser o mais simples e não tentar balancear o consumo de energia dos nós sensores, tal como LEACH. A quantidade de nós sensores foi 100 com o nível de energia igual a 5 joules, distribuídos aleatoriamente em um campo de 500 x 500 metros. O nó sorvedouro está localizado no centro da área (250,250) e os nós sensores possuem um raio inicial de comunicação igual a 250 metros, permitindo que todos se comuniquem com o nó sorvedouro. Além disso, todos os nós sensores enviaram os dados para o nó sorvedouro e a avaliação termina quando o último nó sensor ficar sem energia (LND).

A aplicação envia pacotes com diferentes tamanhos para mostrar o impacto dos elementos da rede (*e.g.*, transmitir um pacote) sobre o consumo de energia da RSSF. Inicialmente, a aplicação envia 100 bytes e depois aumenta para 1.000, 5.000, 10.000 e 15.000 bytes.

A Figura 7.16 mostra o impacto dos elementos da rede (*e.g.*, transmitir um pacote) e da aplicação no consumo de energia da RSSF. Para obter esses valores, a seguinte estratégia foi adotada. Foi calculado o tempo de vida da RSSF (T) utilizando o Modelo de Nó Sensor. Como a aplicação consome 0,0013 joules a cada 2 segundos, nós podemos obter o consumo de energia total da aplicação utilizando a seguinte equação: $Consumo_{app} = 0,0013T/2$. Como o nó sensor possui 5 joules de energia, o consumo de energia da rede pode ser calculado usando a diferença entre o total de energia (5 joules) menos o consumo

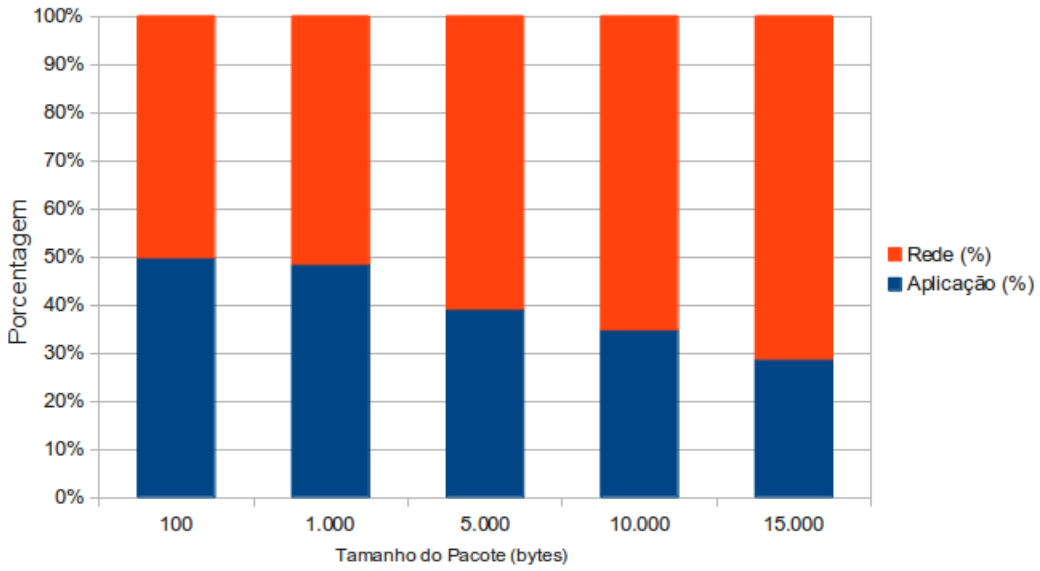


Figura 7.16 Impacto da rede e da aplicação no consumo de energia da RSSF.

da aplicação ($Consumo_{app}$). Por fim, é calculado o impacto da rede e da aplicação (em porcentagem) no consumo de energia da RSSF.

Como é possível observar, o impacto dos elementos da rede aumenta quando aumentamos o tamanho do pacote. Quando o tamanho do pacote é igual a 100 bytes, a influência da rede é igual a 50,24%; esta influência aumenta para 71,31% quando o tamanho do pacote é igual a 15.000 bytes. A influência da rede no consumo de energia da RSSF poderia ter sido maior caso o *First Order Radio Model* considerasse o consumo de energia do transmissor ligado não apenas quando um pacote é enviado ou recebido. Mesmo não contabilizando isso, a rede teve uma influência grande no consumo de energia da RSSF, como já estabelecido na literatura (Akyildiz *et al.*, 2002).

Adicionalmente, o impacto dos elementos da rede e da aplicação no consumo de energia em cada nó sensor varia de acordo com a posição dos nós sensores. Os nós sensores mais distantes morrem mais rápido do que os nós sensores mais próximos do nó sorvedouro quando utilizam o protocolo DIRECT. Sendo assim, a rede deve impactar mais os nós sensores mais distantes, enquanto a aplicação impacta mais os nós sensores mais próximos. Na medida que aumentamos o tamanho do pacote, os elementos da rede começam a influenciar tanto nos nós sensores mais distantes quanto nos mais próximos do nó sorvedouro. O resultado mostrado na Figura 7.16 representa a influência na RSSF.

7.6 Modelo de Região

O objetivo dessa seção é validar o Modelo de Região, mostrando a confiabilidade da mesma RSSF com diferentes protocolos. Foram criados três cenários a partir do *Experimento I*, apresentado na Seção 7.5. O primeiro cenário (Cenário 1) avalia a confiabilidade da mesma região usando os protocolos FLOODING, DIRECT e LEACH. O segundo cenário (Cenário 2) avalia a confiabilidade do mesmo protocolo e da mesma região, mas com quantidade de nós sensores diferentes. Por fim, o último cenário, Cenário 03, avalia a confiabilidade de várias regiões usando o mesmo protocolo (DIRECT).

Não foram considerados o *Experimento II*, por utilizar os mesmos protocolos, e nem o *Experimento III*, por usar apenas um protocolo. Foi também ignorado o protocolo GOSSIPING por não apresentar um padrão no seu comportamento (ele escolhe aleatoriamente os nós sensores para repassar um pacote), consequentemente, a sua confiabilidade oscilava bastante.

Antes de apresentar os resultados, é necessário entender o experimento feito para obter alguns dados da confiabilidade do nó sensor e a configuração da simulação do Modelo de Região.

7.6.1 Confiabilidade da Bateria

A confiabilidade da bateria foi definida através de um experimento usando um nó sensor real. A relação entre a confiabilidade e o nível de energia é determinada pela média de erros ocorridos em uma determinada voltagem. Para ajustar a voltagem, uma fonte de tensão contínua foi usada. É importante observar que o desgaste da bateria ao longo do tempo não foi considerado nesse experimento porque a voltagem não diminui ao longo do tempo tal como na bateria. É necessário mudar a voltagem manualmente usando a fonte de tensão contínua. Adicionalmente, esse experimento usou a plataforma IRIS, que trabalha apropriadamente entre 3,3V e 2,7V (MEMSIC, 2014a), *i.e.*, ele é capaz de enviar e receber pacotes sem falha.

Esse experimento usou um nó sensor para periodicamente enviar mensagens para o computador que detecta se um erro ocorreu no nó sensor ou não. Assim, uma aplicação foi desenvolvida e instalada no nó sensor para coletar a temperatura e enviá-la para o nó sorvedouro, o qual repassa o pacote para o computador. A comunicação entre o nó sensor e o nó sorvedouro era pequena o suficiente para evitar interferências. Dessa maneira, qualquer erro no pacote está relacionado ao nó sensor.

Essa aplicação foi avaliada com diferentes voltagens entre 3,3V (100% da bateria)

e 2,7V (81,81% da bateria). É importante observar que esses valores foram definidos pelo fornecedor do IRIS (MEMSIC, 2014a). Depois, a voltagem foi ajustada de 2,7V (menor valor recomendado pelo fabricante) até quando o nó sensor parou de funcionar. Foi observado que o nó sensor trabalhou corretamente de 2,7V até 1,78V (53,94% da bateria). Abaixo desta voltagem, *e.g.*, *1,77V*, o nó sensor parou de enviar as mensagens para o computador, representando uma falha.

Com esses resultados iniciais, foi criada uma estratégia para definir a confiabilidade da bateria porque é interessante trabalhar com a possibilidade de falha devido ao nível de energia. Por este motivo, a confiabilidade da bateria é 1.0 quando o nível de energia está entre 100% (3,3 V) e 81,81% (2,7 V); e igual a 0,0 quando o nível de energia está abaixo de 53,94% (1,78 V). Quando o nível de energia estiver entre 81.81% e 53,94%, é usada a proporção da confiabilidade da bateria ($\frac{R}{1} = \frac{Level-Min}{Max-Min}$).

Assim, a confiabilidade da bateria é calculada de acordo com a Equação 7.6.1:

$$R_{bateria}(L) = \begin{cases} 1 & \text{se o L estiver entre 100\% a 81,81\%} \\ \frac{L-Min}{Max-Min} & \text{se o L estiver entre 81,81\% a 53,94\%} \\ 0 & \text{se o L estiver menor do que 53,94\%} \end{cases}$$

onde, *Max* e *Min* são iguais a 0,8182 (81,82% do nível de energia) e a 0,5394 (53,94%), respectivamente; e *L* é o valor do nível de energia. Por exemplo, se o nível de energia for igual a 70% (0,70), sua confiabilidade será 0,57. É importante observar que, como mostra essa equação, a confiabilidade da bateria não é influenciada por nenhum outro componente do nó sensor, *e.g.*, rádio, hardware, aplicação, middleware e sistema operacional.

7.6.2 Configuração

Como foi apresentado anteriormente, o *Experimento I* (Seção 7.5) é a base de todos os cenários, com exceção do Cenário 3, que precisou mudar a distribuição dos nós sensores. O *Experimento I* definiu que os nós sensores estão no mínimo a 10 metros do nó sorvedouro e no máximo a 50 metros. As regiões criadas tinham praticamente o mesmo comportamento, o que não era ideal para o Cenário 3. Dessa forma, essa característica foi modificada: os nós sensores ficarão entre 10 e 100 metros dos nós sensores. Assim, foi possível mostrar o impacto da distância dos nós sensores na confiabilidade da região, consequentemente, da RSSF.

Após essas configurações básicas, foram definidas as posições de quatro regiões,

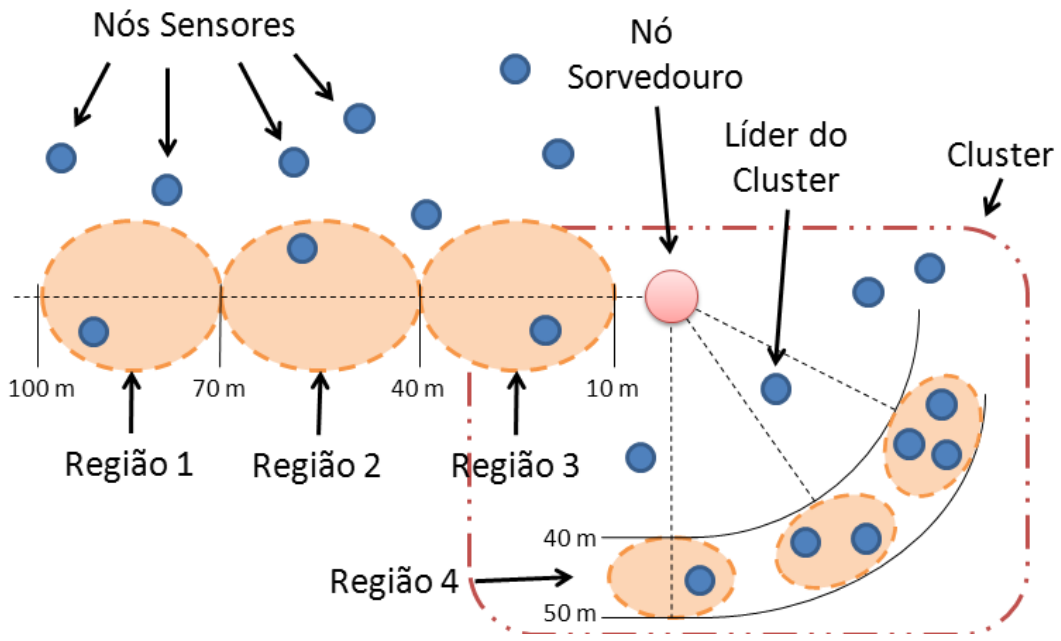


Figura 7.17 Visão Geral das regiões avaliadas pelos três cenários propostos.

como mostra Figura 7.17: a Região 1 fica de 70 a 100 metros do nó sorvedouro; a Região 2, de 40 a 70 metros; a terceira região (Região 3), de 10 a 40 metros; e a última região (Região 4), de 40 a 50 metros. Adicionalmente, as Regiões 1, 2 e 3 foram usadas no Cenário 3, enquanto a Região 4 foi utilizada nos demais cenários.

A confiabilidade da comunicação entre os nós sensores (Seção 5.4.1), aplicação, sistema operacional, middleware e plataforma foi definida como 1.0, *i.e.*, nunca falha. Adicionalmente, a aplicação não utiliza um middleware na comunicação. Para o *middleware* não influenciar na confiabilidade do nó sensor, optou-se por definir seu valor como sendo 1.0. O único valor diferente é a confiabilidade da bateria. Dessa maneira, o nível de energia dos nós sensores irá interferir diretamente na confiabilidade da RSSF.

Não foi definido um valor geral da confiabilidade da bateria, porque esse valor é individual e possui diversos fatores que o influenciam: número de vizinhos, quantidade de mensagens transmitidas, roteadas e recebidas, distância entre os nós sensores, potência do rádio, pilha de protocolo, aplicação, e nível de energia. Por estes motivos, optou-se por definir a confiabilidade da bateria após a avaliação do Modelo de Rede, o qual simula a interferência desses fatores. Além disso, foi utilizada a estratégia apresentada na Seção 7.6.1 para definir o valor da confiabilidade usando o nível de energia gerado após avaliar o Modelo de Rede.

Além disso, nós repetimos 30 vezes todos os experimentos. Nós utilizamos os

valores padrões do *First Order Radio Model* para avaliar o consumo de energia dos nós sensores: o transmissor eletrônico (E_{Tx}) e o receptor eletrônico (E_{Rx}) consomem 50 nJ/bit e amplificador do transmissor ((E_{Amp}) consome 100 pJ/bit/m^2.

Todas essas informações estão sumarizadas na Tabela 7.2.

Tabela 7.2 Configuração da simulação da confiabilidade da RSSF.

Propriedade	Valor
Protocolos de roteamento	DIRECT, FLOODING, GOSSIPING e LEACH
Nós sensores	20
Nó Sorvedouro	1
Area	400 x 400 m
Raio de comunicação	200 m
Nível de Energia	1 mJ
Periodicidade de envio	todos os nós sensores enviam a cada 2 segundos
Distribuição dos nós sensores	entre 10 a 100 metros do nó sorvedouro
Regiões	Região 1 = 100 a 70 metros do nó sorvedouro; Região 2 = 70 a 40 metros; Região 3 = 40 a 10 metros; Região 4 = 50 a 40 metros.
Confiabilidade	Enlace de comunicação = 1,0 middleware = 1,0 Sistema Operacional = 1,0 Aplicação = 1,0 Radio = 1,0 Bateria = Equação 7.6.1
Repetições	30 vezes
Modelo de Energia	First Order Radio Model
E_{Tx}	50 nJ/bit
E_{Rx}	50 nJ/bit
E_{Amp}	100 pJ/bit/m^2

7.6.3 Resultados

O objetivo do primeiro cenário (Cenário 1) é mostrar o impacto do protocolo na confiabilidade de uma região. Como os nós sensores ficarão até 50 metros do nó sorvedouro, foi criada a Região 4 (40 a 50 metros de distância do nó sorvedouro) com apenas um único nó sensor. Esse nó sensor utilizou os protocolos FLOODING, DIRECT e LEACH. A confiabilidade ao longo do tempo de cada protocolo é ilustrada na Figura 7.18.

Como esperado, o FLOODING teve a pior confiabilidade por consumir muita energia. E o protocolo DIRECT teve um desempenho pior do que o LEACH, porque a distância

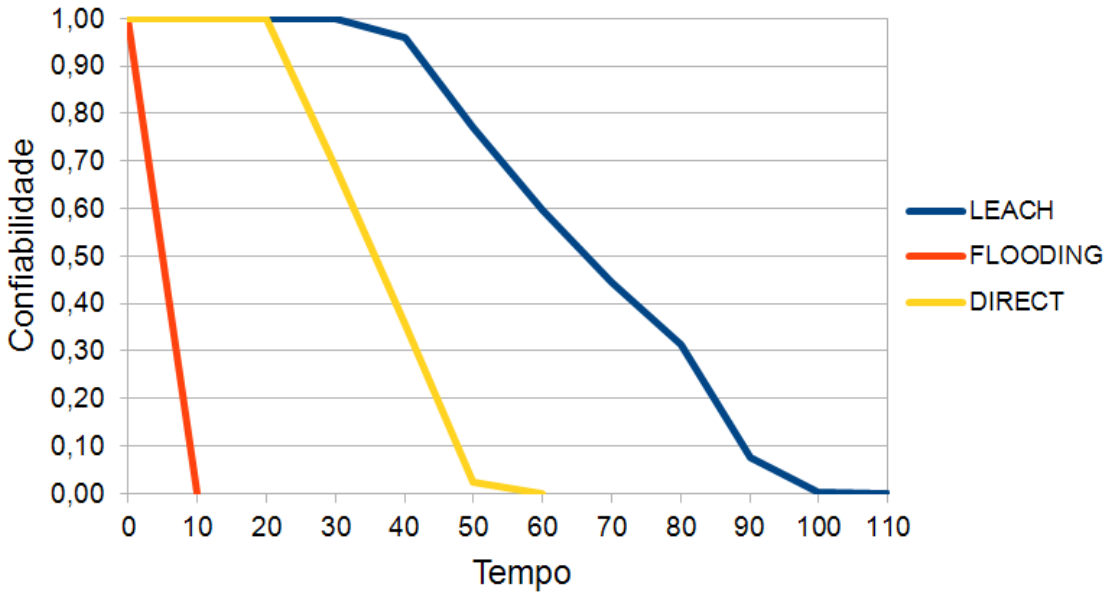


Figura 7.18 Confiabilidade ao logo do tempo da Região 1 quando usa três protocolos diferentes

da região afetou muito mais o DIRECT do que o LEACH: os nós mais afastados morrem mais cedo quando usam esse protocolo. Mesmo o LEACH usando mais nós sensores do que o DIRECT, o nível de energia foi o fator chave para melhorar a confiabilidade do LEACH. O LEACH cria *clusters* periodicamente na RSSF, diminuindo e balanceando o consumo de energia na rede.

É importante observar que o protocolo FLOODING é uma boa alternativa quando os nós sensores e os enlaces de comunicação possuem uma confiabilidade muito baixa e quando o nível de energia não é considerado. Assim, para mostrar essa situação, o Cenário 1 foi refeito, mas com outras configurações: a confiabilidade do nível de energia foi constante (igual a 1,0) e a confiabilidade dos nós sensores e dos enlaces foram iguais a 0,1, *i.e.*, eles têm uma baixa confiabilidade. A Figura 7.19 mostra o resultado desse experimento. O protocolo DIRECT tem o melhor resultado, porque ele só envolveu dois nós sensores e um enlace. Além disso, a confiabilidade do enlace não considera a distância entre o transmissor e o receptor; o que poderia afetar o resultado do DIRECT. O protocolo FLOODING teve o segundo melhor resultado porque ele cria múltiplos caminhos. O protocolo LEACH teve o pior resultado, porque ele utilizou um nó sensor, o líder do *cluster*, a mais do que o DIRECT. Por exemplo, enquanto a confiabilidade da região usando o DIRECT é igual a $0,1^3$, a confiabilidade da mesma região usando o LEACH é igual a $0,1^5$, porque usou um nó sensor a mais na comunicação com o nó sorvedouro (veja a Seção 5.4).

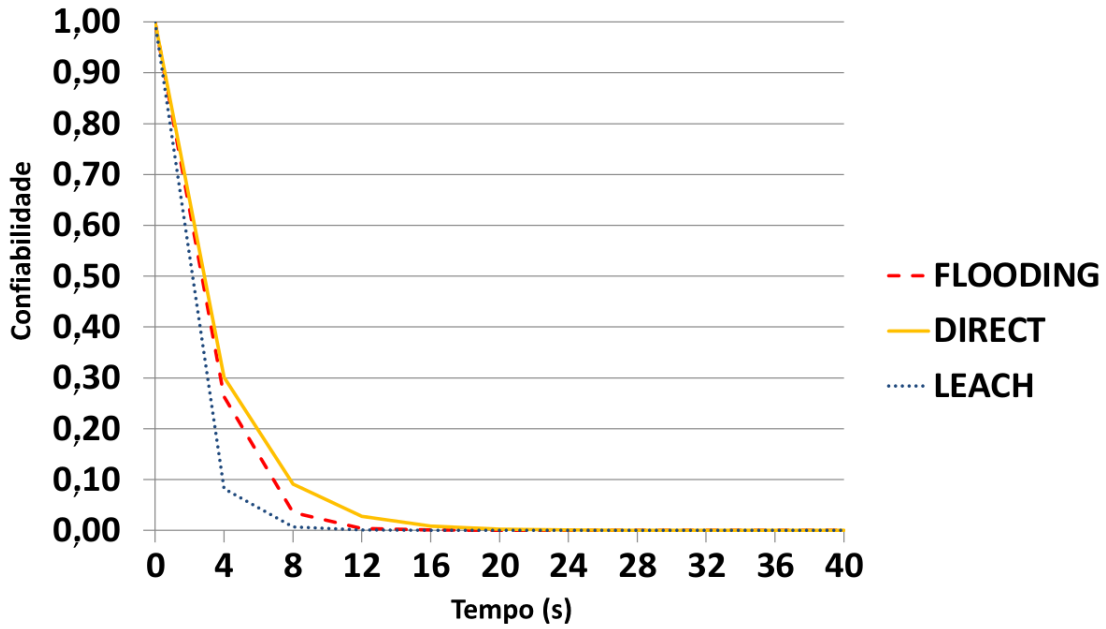


Figura 7.19 Confiabilidade da região quando os nós sensores possuem energia suficiente e baixa confiabilidade

O segundo cenário (Cenário 2) foi utilizado para avaliar o impacto do número de nós sensores na confiabilidade de uma região. Esse cenário utilizou a Região 1 (por ser a mais afasta) com 1, 3 e 5 nós sensores. O protocolo LEACH foi escolhido para ser usado nesse cenário por montar *clusters* na rede, o que é muito adequado para avaliar a cooperação entre os nós sensores.

Como é apresentado na Figura 7.20, a Região 1 com 1 nó sensor teve o pior resultado, enquanto a diferença entre a região com 3 e 5 nós sensores não foi significativa, porque eles possuem um gargalo em comum: o *cluster head*(CH). Se o CH falha, a região falhará, não importa o número de nós sensores dentro, porque todos os caminhos dos nós sensores da mesma região até o nó sorvedouro passam pelo líder do *cluster* (Figura 5.27). A região com 3 e 5 nós sensores foram diferentes em dois pontos: o primeiro ponto em 50 segundos, a região com 3 nós sensores teve a pior confiabilidade, porque esta região possui menos nós sensores; o segundo ponto em 90 segundos, a região com 5 nós sensores teve a pior confiabilidade, porque (certamente) os nós sensores começaram a ficar sem energia. Entretanto, a região com 5 nós sensores ficou mais tempo disponível, porque sua confiabilidade só é igual a zero quando o nível de bateria dos 5 nós sensores for menor do que 53,94%.

O objetivo do último cenário (Cenário 3) é mostrar que, em alguns casos, a localização de uma região interfere na sua confiabilidade. A distribuição dos nós sensores teve que

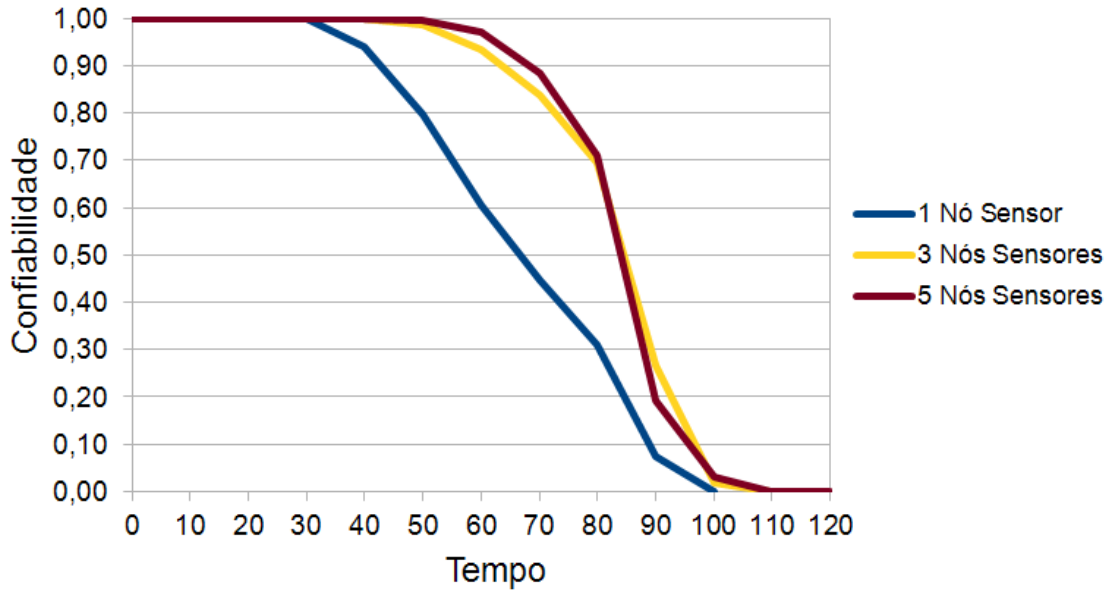


Figura 7.20 Confiabilidade ao longo do tempo da Região 1 com diferentes nós sensores

ser alterada: os nós sensores estavam entre 10 e 100 metros (antes era de 10 a 50 metros) do nó sorvedouro. Além disso, foram avaliadas as Regiões 1, 2 e 3 com apenas um nó sensor em cada uma delas quando eles usavam o protocolo DIRECT. Como mencionado antes, os nós sensores mais afastados morrem mais cedo quando usam esse protocolo. Por este motivo, ele foi escolhido para mostrar o decremento da confiabilidade das regiões em diferentes partes da rede.

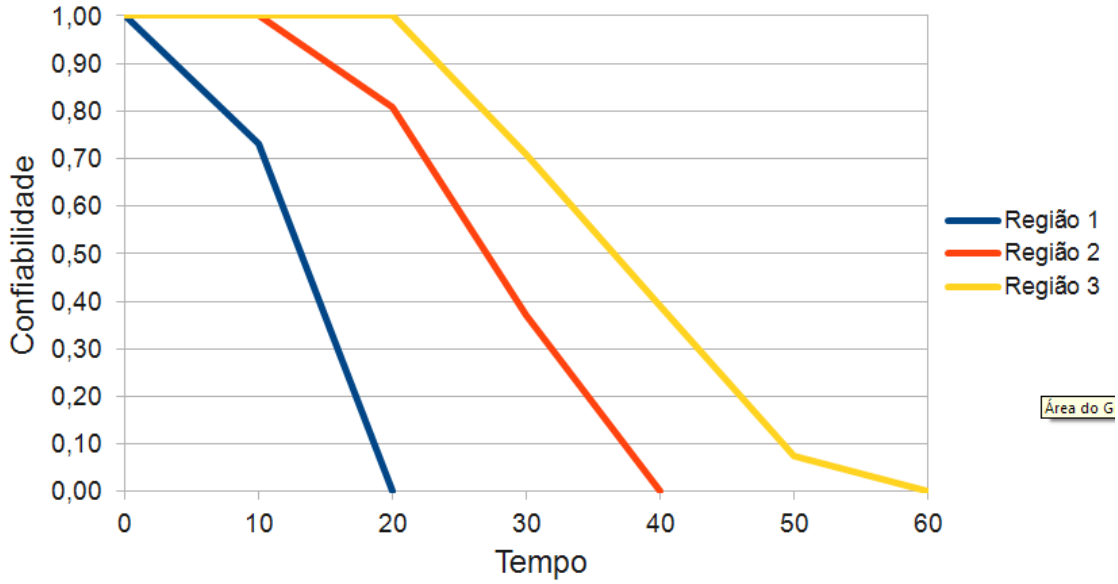


Figura 7.21 Confiabilidade ao longo do tempo das três regiões usando protocolo DIRECT.

A Figura 7.21 mostra o resultado do Cenário 3. Como esperado, a Região 3, por ser mais próxima, teve a melhor confiabilidade, enquanto a Região 1 teve o pior resultado. Os nós sensores mais distantes, quando usam o DIRECT, tendem a morrer mais cedo do que os mais próximos do nó sorvedouro. Além disso, esse cenário reforça a importância de avaliar individualmente cada região da RSSF para avaliar qual delas necessita de mais atenção.

7.7 Análise de Sensibilidade

A atividade *Fazer Analise de Sensibilidade* da metodologia (Capítulo 4) é responsável por criar e avaliar várias configurações da RSSF de acordo com os parâmetros, chamados de fatores, e valores, chamados de níveis, definidos pelo usuários. Este é o ponto chave desta atividade para identificar a influência de cada fator no consumo de energia e na confiabilidade da RSSF.

Para realizar a análise de sensibilidade, nós consideramos os fatores e níveis mostrados na seção a seguir.

7.7.1 Fatores e Níveis

A Tabela 7.3 mostra os 8 fatores com 2 níveis considerados no experimento. Usando a Equação 2.5, 256 combinações foram criadas, sendo cada combinação uma RSSF com um nível de cada fator.

Tabela 7.3 Fatores e níveis considerados na Analise de Sensibilidade

Fator	Sigla	Nível (-1)	Nível (+1)
Nível Inicial de Energia	IEL	0,001 Joules	0,005 Joules
Tamanho do Pacote	PaS	250 Kb	50 Kb
Protocolo de Roteamento	RP	GOSSIPING	DIRECT
Alcance Inicial do Rádio	IRR	1770m	354m
Algoritmo de Implantação	PA	Longe	Perto
Número de Nós Sensores	NN	20	100
Taxa Regular de Envio	SRR	2s	10s
Tamanho do Área	PlS	500m x 500m	100m x 100m

Nós definimos que o primeiro nível (-1) é o pior caso e o segundo nível (+1) é o melhor caso baseado nos resultados do consumo de energia anteriores e em outros trabalhos da literatura (Xu *et al.*, 2005; Xu, 2008; Senouci *et al.*, 2012). Os níveis com valores positivos são cinco vezes maiores do que o próximo nível do mesmo fator, *e.g.*, 0,001

Joules e 0,005 Joules. Desta maneira, nós evitamos favorecer um fator em particular para influenciar o consumo de energia e/ou confiabilidade da RSSF mais do que os outros fatores. Além disso, outros critérios para a definição dos valores para cada parâmetro foram utilizados e serão apresentados nos próximos parágrafos.

Os valores do *Nível Inicial de Energia* (0,001 e 0,005) e a *Número de Nós Sensores* (20 e 100) são baseados em experimentos anteriores e em outros trabalhos da literatura (Senouci *et al.*, 2012; Dâmaso *et al.*, 2014a,b). Além disso, esses são valores pequenos para viabilizar a simulação. No caso da bateria, um valor muito alto poderia aumentar o tempo das avaliações, aumentando o tempo para fazer análise de sensibilidade. No caso da quantidade de nós sensores, um valor muito alto poderia inviabilizar a simulação por demandar uma quantidade imensa de processamento.

Existem uma relação entre os níveis dos fatores *Periodicidade de Envio* e *Tamanho do Pacote*. Se o nó sensor enviar a cada 2 segundos, primeiro nível de *Periodicidade de Envio*, um pacote de 50Kb, primeiro nível de *Tamanho de Pacote*, ele enviará a mesma quantidade de dado no mesmo período caso enviasse a cada 10 segundos um pacote de 250 kb, segundo nível de *Periodicidade de Envio* e *Tamanho do Pacote*, respectivamente. O objetivo disto é verificar se é mais vantajoso enviar pacotes pequenos, mas frequentemente; ou se é melhor enviar um pacote grande em um período maior. Adicionalmente, todos os nós sensores irão enviar o pacote para o nó sorvedouro, e os valores da periodicidade são pequenos para acelerar a avaliação visto que a avaliação só acaba quando todos os nós sensores morrem.

Existe também uma relação entre o *Tamanho da Área* e o *Alcance Inicial do Rádio* dos nós sensores. O nó sorvedouro está localizado no centro da área. Por exemplo, se a área tem 500 x 500 metros, o nó sorvedouro encontra-se na localização (250,250). Para garantir que qualquer nó sensor tenha comunicação direta com o nó sorvedouro, é necessário que todos os nós sensores tenham um raio de comunicação inicial de no mínimo 250m (primeiro nível do fator *raio de alcance*).

Os protocolos de roteamento GOSSIPING e DIRECT foram escolhidos por terem sido, respectivamente, como os piores e o melhores protocolos no consumo de energia na avaliação feita em (Senouci *et al.*, 2012; Dâmaso *et al.*, 2014a,b). Na realidade, o protocolo FLOODING teve o pior resultado. No entanto, o consumo de energia desse protocolo inviabiliza a avaliação da confiabilidade da RSSF, porque a rede não passava muito tempo ativa; e, como a confiabilidade do nó sensor está relacionada diretamente com o nível de energia, a confiabilidade dos nós sensores é praticamente igual a zero durante todo o experimento já que este protocolo consome energia muito rápido.

Por último, os níveis do *Algoritmo de Implantação* são baseados na estratégia *Lifetime-Oriented Deployment* (Xu *et al.*, 2005; Xu, 2008). A RSSF pode ser formada por dois conjuntos de nós sensores, um próximo e outro distante do nó sorvedouro, o qual está localizado no centro da área. O consumo de energia será diferente visto que os nós sensores mais distantes necessitam de mais energia para transmitir os pacotes. Além disso, a confiabilidade da RSSF pode variar de acordo com a distância do nó sensor ao nó sorvedouro, porque a confiabilidade da RSSF é afetada pelo consumo e energia.

A partir desses fatores e níveis, 256 combinações foram criadas usando a Equação 2.5 e cada combinação é uma configuração da RSSF. Para fazer a análise de sensibilidade, nós usamos o componente *Analisador de Sensibilidade* (veja o Capítulo 6), o qual irá criar e avaliar todas as combinações em paralelo usando vários computadores simultaneamente.

7.7.2 Considerações

Antes de apresentar os resultados, nós devemos entender algumas considerações usadas na simulação da RSSF sobre o modelo de consumo de energia, propagação do sinal, enlace de comunicação, tráfego de dados e posição do nó sorvedouro. Tais informações são importantes para compreender o resultados.

- O *First Order Radio Model* (Heinzelman, 2000; Heinzelman *et al.*, 2000) foi usado para calcular o consumo de energia da RSSF por sua simplicidade. Ele só considera o consumo de energia quando o nó sensor envia e recebe pacotes da rede. Ou seja, ele não considera o consumo de energia quando o nó sensor processa o pacote.

- A comunicação entre os nós sensores é perfeita e não tem nenhum obstáculo para degradar o sinal. Por exemplo, se o nó sensor A está dentro do raio de comunicação do nó sensor B, o nó sensor A irá receber todos os pacotes enviados pelo no sensor B. Aliado a isso, foi considerado que a camada de enlace é perfeita, *i.e.*, ela evita colisões de pacote e *overhearing*.

- Foi definida uma região, *Região I*, com um nó sensor localizado em uma das arestas da RSSF, especificamente, entre as coordenadas (0,0) a (10,10). Esta região é usada para avaliar o impacto dos fatores na confiabilidade da RSSF. Na realidade, nós já sabemos que a distância (combinação entre os fatores *Tamanho da Área* e *Algoritmo de Implantação*) e o algoritmo de roteamento influencia a confiabilidade da região. No entanto, nós não sabemos qual fator afeta mais a confiabilidade da RSSF. Colocando a região longe do nó sorvedouro, nós teremos uma melhor noção de quais fatores afetaram a confiabilidade da RSSF; e

- Todos os nós sensores enviam dados para o nó sorvedouro, o qual está localizado no centro da área. Por exemplo, o nó sorvedouro estará localizado na coordenada (50,50) e (250,250) quando o fator *tamanho da área* for 100x100 e 500x500 metros, respectivamente.

7.7.3 Resultados

Nós analisamos o impacto de cada fator e a interação entre dois fatores no consumo de energia e na confiabilidade da RSSF, como mostra a Tabela 7.4. Nós obtivemos esses valores quando usamos as Equações (2.5) e (2.6), descritas na Seção 2.4. Nesta tabela, os valores estão ordenados de forma decrescente: ele começa com o fator ou interação com maior impacto e terminar com aqueles de menos impacto.

Consumo de energia. Os quatro fatores que afetaram mais o consumo de energia da RSSF foram *Protocolo de Roteamento* (RP), *Tamanho do Pacote* (PaS), *Nível Inicial de Energia* (IEL) e *Taxa de Envio Regular* (SRR). O *Protocolo de Roteamento* (RP) tem o maior impacto entre os quatro fatores porque ele decide como os nós sensores irão cooperar entre si para se comunicarem com o nó sorvedouro. Por exemplo, o protocolo DIRECT tem melhor consumo de energia do que o protocolo GOSSIPING porque os nós sensores enviam os dados diretamente para o nó sorvedouro. Usando o protocolo DIRECT, os nós sensores mais próximos do nó sorvedouro consomem menos energia do que aqueles mais distantes, passando mais tempo ativos e contribuindo para a RSSF passar mais tempo ativa.

Para compreender o impacto do *Tamanho do Pacote* (PaS) e *Taxa de Envio Regular* (SRR), é preciso observar que *First Order Radio Model* só considera o consumo de energia quando se envia e recebe um pacote. Portanto, dois fatores são afetados: o tamanho do pacote e o *Alcance Inicial do Rádio* (IRR). Quando nós aumentamos em cinco vezes (*e.g.*, 50kb to 250kb) o *Tamanho do Pacote* (PS), o nó sensor irá consumir cinco vezes mais para enviar um pacote. Quando nós mudamos cinco vezes (*e.g.*, 250m to 1250m) o valor do *Alcance Inicial do Rádio* (IRR), o nó sensor deveria aumentar o consumo de energia em vinte e cinco vezes para enviar um pacote. No entanto, esse comportamento não acontece quando aumentamos o *Alcance Inicial do Rádio* (IRR) porque o *Protocolo de Roteamento* (RP) muda seu valor assim que inicia (veja a Seção 2.1). Por exemplo, o protocolo DIRECT muda o alcance do rádio para a distância entre o nó sensor e o nó sorvedouro. Por esta razão e porque o nó sorvedouro está no centro, o *Tamanho do Área* (PlS) influencia mais o consumo de energia do que o *Alcance Inicial do Rádio* (IRR).

Tabela 7.4 Impacto dos fatores e das interações no consumo de energia e na confiabilidade da RSSF

	Consumo de Energia		Confiabilidade	
	Fator	Impacto	Fator	Impacto
1	RP	2169,39844	RP	0,40058532
2	RP e PaS	-2021,21875	PlS e RP	-0,36882255
3	RP e IEL	-1992,00781	PlS	0,33705977
4	RP e SRR	-1970,54688	RP e PaS	-0,27112969
5	PA e RP	-1886,37500	RP e IEL	-0,27058203
6	PaS	1873,03906	RP e SRR	-0,26114040
7	PaS e IEL	-1843,82813	PlS e PaS	-0,23936691
8	PaS e SRR	-1822,36719	PlS e IEL	-0,23881926
9	IEL	1814,61719	PlS e SRR	-0,22937762
10	IEL e SRR	-1793,15625	IRR e RP	-0,20832578
11	SRR	1771,69531	PA e RP	-0,19865220
12	PA e PaS	-1738,19531	NN e RP	-0,19406566
13	PA e IEL	-1708,98438	IRR e PlS	-0,17656301
14	PA e SRR	-1687,52344	PA e PlS	-0,16688943
15	PA	1603,35156	PlS e NN	-0,16230288
16	PlS e RP	-1459,57813	PaS	0,14167405
17	PlS e PaS	-1311,39844	PaS e IEL	-0,14112640
18	PlS e IEL	-1282,18750	IEL	0,14057874
19	PlS e SRR	-1260,72656	PaS e SRR	-0,13168476
20	IRR e RP	-1241,35156	IEL e SRR	-0,13113711
21	NN e RP	-1210,56250	SRR	0,12169547
22	PA e PlS	-1176,55469	IRR e PaS	-0,07887015
23	IRR e PaS	-1093,17188	IRR e IEL	-0,07832249
24	IRR e IEL	-1063,96094	PA e PaS	-0,06919657
25	NN e PaS	-1062,38281	IRR e SRR	-0,06888086
26	IRR e SRR	-1042,50000	PA e IEL	-0,06864891
27	NN e IEL	-1033,17188	NN e PaS	-0,06461003
28	NN e SRR	-1011,71094	NN e IEL	-0,06406237
29	PA e IRR	-958,32813	PA e SRR	-0,05920728
30	PA e NN	-927,53906	NN e SRR	-0,05462074
31	PlS	749,75781	IRR	0,01606624
32	IRR e PlS	-531,53125	NN	-0,01245400
33	PlS e NN	-500,74219	PA e NN	0,00786746
34	IRR	313,30469	PA e IRR	-0,00639266
35	IRR e NN	-282,51563	PA	-0,00328091
36	NN	251,72656	IRR e NN	-0,00180612

Além do *Tamanho do Pacote* (PS), a *Taxa de Envio Regular* (SRR) também tem um forte impacto no consumo de energia da RSSF. Se o nó sensor envia frenquentemente vários pequenos pacotes, esse comportamento aumenta o seu consumo de energia. Essa razão é bem simples: o nó sensor consome o mesmo nível de energia se ele enviar um grande pacote em uma baixa frequência ou se ele enviar um pequeno pacote frequentemente. Por exemplo, considerando-se dois nós sensores (A e B), o nó sensor A envia um pacote de 50Kb a cada 2 segundos e o nó sensor B envia um pacote de 250kb a cada 10 segundos. O consumo de energia do nó A será igual ao consumo de energia do nó B a cada 10 segundos, porque eles irão transmitir a mesma quantidade de pacote a cada 10 segundos. No entanto, o *Tamanho do Pacote* (PS) influencia mais do que a *Taxa de Envio Regular* (SRR) porque ele afeta diretamente o consumo de energia da RSSF. Por esta razão, é mais importante reduzir o *Tamanho do Pacote* (PS) e aumentar a *Taxa de Envio Regular* (SRR). Desta maneira, a RSSF pode passar mais tempo em campo. Adicionalmente, a interação entre o *Tamanho do Pacote* (PS) e a *Taxa de Envio Regular* (SRR) está na oitava colocação no *ranking*, mostrando que existe uma forte relação entre eles quando comparado com outras interações.

O *Algoritmo de Implantação* (PA) também teve uma forte influência no consumo de energia da RSSF por conta dos *Protocolos de Roteamento* (RP) escolhidos e a interação entre esses dois fatores está na quinta colocação no *ranking*. Como mencionado anteriormente, o protocolo DIRECT ajuda os nós sensores próximos ao nó sorvedouro. Quando os nós sensores são colocados mais próximos do nó sorvedouro, a RSSF passa mais tempo ativa porque diminui a potência de transmissão de dados e, consequentemente, diminui o consumo de energia dos nós sensores quando transmitem pacotes.

Finalmente, o *Número de Nós Sensores* (NN) teve pouco impacto no consumo de energia da RSSF. As razões para isto incluem (i) o protocolo DIRECT não requer ajuda de nenhum outro nó sensor para enviar dados para o nó sorvedouro; (ii) a camada de enlace foi considerada como perfeito, *i.e.*, ela não contabiliza o consumo de energia quando um nó sensor processa um pacote destinado a outro nó sensor por engano ou quando há colisão. Se o experimento tivesse usado outro protocolo na camada de enlace e de rede, o número de nós sensores e a distribuição poderia ter tido um impacto significativo no consumo de energia da RSSF.

Adicionalmente, se tivéssemos considerado a camada de enlace como fator, nós teríamos 512 RSSF para avaliar, levando o dobro do tempo de avaliação, inviabilizando o estudo. Os protocolos de roteamento foram escolhidos a partir de um estudo do consumo de energia e por serem utilizados na literatura. Se formos escolher outros protocolos

na análise de sensibilidade para evidenciar o impacto da quantidade e a posição de nós sensores, de uma certa forma, nós estaríamos influenciando no resultado da análise de sensibilidade.

Confiabilidade. A confiabilidade da RSSF foi calculada no tempo de 25 segundos porque a maioria das RSSFs avaliadas tinham a confiabilidade maior do que 0 nesse tempo. Como o propósito da análise de sensibilidade é usar os mesmos fatores e níveis para avaliar o consumo de energia e confiabilidade da RSSF, nós avaliamos a confiabilidade da *Região I* e apenas consideramos o nível de energia como confiabilidade do nó sensor (Seção 5.4.3). Sem esta consideração, a análise de sensibilidade da confiabilidada da RSSF deveria ser feita considerando outros fatores e níveis, *e.g.*, confiabilidade do enlace de comunicação.

Os quatro fatores que afetaram mais a confiabilidade da RSSF foram *Protocolo de Roteamento* (RP), *Tamanho do Área* (PlS), *Tamanho do Pacote* (PaS) e *Nível Inicial de Energia* (IEL). Como tínhamos mencionado anteriormente, a confiabilidade da RSSF varia de acordo com a distância entre a região e o nó sorvedouro, o protocolo de roteamento usado e a quantidade de nós sensores dentro da região. Como o Modelo do Caminho (Seção 5.4.3) é baseado no algoritmo de roteamento, era esperado que o protocolo de roteamento afetaria a confiabilidade da RSSF. Além disso, como a região está localizada em um dos extremos da rede e o nó sorvedouro está no centro, nós também esperávamos que o *Tamanho do Área* (PlS) fosse um dos fatores que impactasse na confiabilidade da RSSF. O *Número de Nós Sensores* (NN) não teve muito impacto na confiabilidade porque a região tem apenas um único nó sensor independentemente da quantidade de nós sensores presentes na RSSF.

Para entender os outros fatores, é importante relembrar que a confiabilidade do nó sensor é baseada no nível de energia do nó sensor, usando a Equação 7.6.1, *i.e.*, a confiabilidade do nó sensor está fortemente dependente do nível de energia. Note que a partir do *Tamanho do Pacote* (PaS), a ordem de impacto foi mantida (similar à tabela do consumo de energia): *Tamanho do Pacote* (PaS), *Nível Inicial de Energia* (IEL), *Taxa de Envio Regular* (SRR) e assim por diante. A única exceção foi o *Algoritmo de Implantação* (PA), pois ele não teve muito impacto na confiabilidade da RSSF tal como no consumo de energia porque a região sempre esteve no mesmo lugar independentemente da posição dos outros nós sensores. Entretanto, é importante ressaltar que a interação entre o *Algoritmo de Implantação* (PA) e *Algoritmo de Roteamento* (RP) teve um impacto considerado (décima primeira posição) na confiabilidade da RSSF quando comparado com outras interações, *e.g.*, a interação entre *Algoritmo de Implantação* (PA) e *Alcance Inicial do*

Rádio (IRR).

7.8 Considerações Finais

Este capítulo teve como propósito (I) mostrar como foram coletados o consumo de energia de cada operador e dos comandos oferecidos pelo TinyOS; (II) validar os modelos propostos no Capítulo 5; (III) mostrar as vantagens de optar pelas sugestões dadas pelo editor; e (IV) ilustrar o impacto de cada fator no consumo de energia e na confiabilidade da RSSF (*e.g.*, tamanho do pacote). Para mostrar tudo isso, diversos experimentos foram realizados, entre simulações e medições. Resumidamente, foram realizados 6 experimentos para validar as sugestões do editor, obtendo em um dos experimentos uma otimização de 71,85% no consumo de energia; 5 experimentos para validar o Modelo de Aplicação, obtendo resultados similares à uma medição; 3 cenários para validar o Modelo de Rede e de Nó Sensor, comparando os resultados com valores na literatura; 3 simulações para validar o Modelo de Região, mostrando que a confiabilidade de qualquer região é influenciada pelo protocolo de roteamento, pela distância entre a região e o nó sorvedouro, e pela quantidade de nós sensores dentro da região; e, por fim, 1 análise de sensibilidade utilizando os modelos propostos, avaliand o consumo de energia e a confiabilidade de 256 RSSF para determinar o impacto de 8 fatores no consumo de energia e na confiabilidade da RSSF.

8

Conclusões e Trabalhos Futuros

Este último capítulo apresenta a conclusão desse trabalho, as contribuições, as limitações e os trabalhos futuros.

8.1 Conclusões

Observando a importância do planejamento das RSSFs, esta tese propõe uma metodologia para avaliar o consumo de energia e a confiabilidade de aplicações e componentes de comunicação das RSSFs.

A avaliação do consumo de energia foi feita usando modelos CPN organizados em 3 conjuntos de modelos: Modelo De aplicação, para avaliar o consumo de energia da aplicação escrita em nesC; Modelo de Rede, o qual avalia o consumo de energia da rede; e Modelo de Nó Sensor, que avalia o consumo de energia da aplicação e da rede ao mesmo tempo. Em relação à confiabilidade, foi criado apenas um único modelo (Modelo de Região), o qual avalia a confiabilidade de uma região da RSSF.

Esses modelos são criados dinamicamente baseados no código da aplicação ou na configuração da rede. Para isso, foi utilizada a estratégia dividir para conquistar: primeiro, modela o menor ponto de consumo de energia e de confiabilidade da RSSF; e depois combina-os para criar os modelos propostos.

Além da metodologia e dos modelos, também foi proposto o EDEN (*Evaluation and Development Environment for Wireless Sensor Network*): um ambiente Web distribuído para suportar as etapas da metodologia. Ele é formado por um editor, um tradutor, um avaliador e um gerenciador.

8.2 Contribuições

As contribuições desta tese estão sumarizadas como seguem:

Metodologia. A metodologia une o planejamento da aplicação com o planejamento da infraestrutura da RSSF, dando uma visão única do planejamento da RSSF. Além disso, ela considera, como parte da validação da RSSF, a avaliação do consumo de energia e a confiabilidade, além da análise de sensibilidade da RSSF.

Modelos. Foram propostos modelos formais, os quais foram construídos com base no código fonte da aplicação e na configuração da infraestrutura da RSSF. Em especial, o modelo de confiabilidade é construído com base no algoritmo de roteamento utilizado pela RSSF. As informações geradas por um modelo serviram como entrada para outros, criando uma ordem de execução entre os modelos e dando a visão única da solução.

Conjunto de Ferramentas. Esta tese também propõe um conjunto de ferramentas para dar suporte à metodologia proposta. Este conjunto de ferramentas é executado em um ambiente de nuvem.

Análise de Sensibilidade. A análise de sensibilidade das RSSFs foi feita para identificar quais fatores afetam mais o consumo de energia e a confiabilidade da RSSF.

8.3 Limitações

As limitações dessa proposta estão relacionadas com os modelos e se estendem para o conjunto de ferramentas.

Modelo de Aplicação. O Modelo de Aplicação considera que todas as funções são *Synchronous Command* (SC), as quais são executadas sequencialmente. No entanto, aplicações escritas na linguagem nesC também possuem funções *Asynchronous Command* (AC). É importante ressaltar que todas as funções (SC ou AC) serão executadas, não gerando problemas na execução da aplicação.

Modelo de Rede e Modelo. O Modelo de Rede possui duas limitações. A primeira é o fato de que considera-se que todos os nós sensores possuem o mesmo Modelo de Rede, ou seja, uma mesma pilha de protocolos. A segunda limitação, o Modelo de Rede só permite que exista um nó sorvedouro. RSSFs com mais de um nó sorvedouro não podem ser avaliadas por esse modelo.

Modelo de Nó Sensor. Tal como o Modelo de Rede, o Modelo de Nó Sensor considera que a mesma aplicação é executada em todos os nós sensores. Em situações onde é preciso executar várias aplicações diferentes, é necessário criar sub redes agrupando nós

com a mesma aplicação e avaliá-las separadamente.

Modelo de Região. É considerado apenas o algoritmo de roteamento para criar o modelo de confiabilidade. No entanto, existem outras características presentes nas outras camadas que são importantes. Por exemplo, a retransmissão de pacotes presentes na camada de enlace e de transporte não foi modelada.

Análise de Sensibilidade. Algumas características das RSSFs não foram avaliadas na análise de sensibilidade, tal como *Duty Cicle*, reenvio de pacotes, diferentes protocolos da camada de enlace e de transporte, nós sensores se movimentando na RSSF, entre outros. Tais características também influenciam o consumo de energia e a confiabilidade da RSSF.

8.4 Trabalhos futuros

Os próximos passos para esse trabalho estão focados em três pontos: estender os modelos propostos, avaliar as propriedades dos modelos CPN e RBD, monitoramento em tempo real e dependabilidade.

Estender os modelos propostos. Alguns cenários específicos não podem ser avaliados usando os modelos propostos. Por exemplo, avaliar o consumo de energia de uma RSSF com diferentes aplicações executando nós sensores, permitindo que os nós sensores possam se deslocar dentro da rede. Observando isso, os modelos devem ser estendido para avaliar situações específicas da RSSF.

Avaliar outros aspectos dos modelos. Além de avaliar o consumo de energia e a confiabilidade da RSSF, os modelos CPN e RBD podem verificar outras propriedades da RSSF ou servir para fazer outras análises da RSSF. Por exemplo, o Modelo de Aplicação, feito em CPN, poderia verificar se existe algum comando que nunca será executado. Como os comandos são representados por transições no Modelo de Aplicação, nós podemos verificar quais transições nunca serão executadas (propriedade *liveness*). No caso do RBD, nós poderíamos verificar quais nós sensores são importantes na comunicação entre a região e o nó sorvedouro (*Reliability Importance*). Ou, então, nós poderíamos avaliar a criticalidade da RSSF para verificar os nós sensores que falham mais e a gravidade da falha, ou os que estão mais propensos a falhar na RSSF. As ferramentas utilizadas nesta tese (CPN Tools e Mercuy) já possuem mecanismos para avaliar essas propriedades do CPN e do RBD, exigindo algumas adaptações para avaliar os modelos criados. Por exemplo, como nós criamos vários modelos para avaliar a confiabilidade de uma região (veja a Seção 4.6.1), para avaliar qual nó sensor é importante na comunicação entre a

região e o nó sorvedouro, é necessário avaliar todos os modelos criados desta região.

Implementar mais otimizações no editor. Nós citaremos quatro exemplos que poderiam ser implementados. O primeiro é utilizar algoritmos para analisar a configuração de rede e sugerir as posições dos nós sensores para melhorar o consumo de energia e/ou a confiabilidade. Alguns desses algoritmos são explicados por Younis and Akkaya (2008). O segundo exemplo é auxiliar o usuário a posicionar os nós sorvedouros, tal como Kim *et al.* (2005b) propôs. O terceiro exemplo é encontrar um ou mais nós sensores que não tem comunicação com o nó sorvedouro. E o último exemplo de otimização é verificar se o protocolo de roteamento escolhido se enquadra para a configuração de rede criada.

Monitoramento em tempo real. Esta tese está focada nas etapas que antecedem a implantação da RSSF. No entanto, é interessante monitorar a RSSF para verificar se os resultados obtidos estão de acordo com a realidade. Ou seja, é importante monitorar a RSSF logo após ela ter sido implantada e comparar os resultados reais com os obtidos no planejamento. Desta maneira, é possível calibrar os modelos com dados reais e prever, por exemplo, problemas de consumo de energia de um determinado conjunto de nós sensores não visto no desenvolvimento da RSSF.

Dependabilidade. Existem outros aspectos que não são avaliados nessa proposta, tais como segurança, que está relacionada a confiabilidade no sentido de estar livre de danos catastróficos; disponibilidade, a probabilidade de que a RSSF esteja operacional durante um determinado período de tempo; integridade, que tenta garantir que os dados não sejam alterados durante uma comunicação dos nós sensores; entre outros. Essas análises servem como base para outra maior, chamada de análise de dependabilidade (Parhami, 1988), que avalia, com mais detalhes, a qualidade do serviço prestado. Novos modelos devem ser criados e a ordem de avaliações dos modelos deve ser refeita.

Referências Bibliográficas

Agre, J. and Clare, L. (2000). An integrated architecture for cooperative sensing networks. *Computer*, **33**(5), 106–108.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: A survey. *Comput. Netw.*, **38**(4), 393–422.

Akyildiz, I. F., Melodia, T., and Chowdhury, K. R. (2007). A survey on wireless multimedia sensor networks. *Comput. Netw.*, **51**(4), 921–960.

Al-Karaki, J. and Kamal, A. (2004). Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, **11**(6), 6–28.

Aslam, S., Farooq, F., and Sarwar, S. (2009). Power consumption in wireless sensor networks. In *Proceedings of the 7th International Conference on Frontiers of Information Technology*, FIT '09, pages 14:1–14:9, New York, NY, USA. ACM.

Balouchestani, M., Raahemifar, K., and Krishnan, S. (2011). Increasing the reliability of wireless sensor network with a new testing approach based on compressed sensing theory. In *Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on*, pages 1–4.

Balouchestani, M., Raahemifar, K., and Krishnan, S. (2012). Wireless body area networks with compressed sensing theory. In *Complex Medical Engineering (CME), 2012 ICME International Conference on*, pages 364–369.

Barr, R., Haas, Z. J., and van Renesse, R. (2005). Jist: An efficient approach to simulation using virtual machines: Research articles. *Softw. Pract. Exper.*, **35**(6), 539–576.

Basagni, S., Carosi, A., and Petrioli, C. (2004). Sensor-dmac: dynamic topology control for wireless sensor networks. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 4, pages 2930–2935 Vol. 4.

Bein, D., Jolly, V., Kumar, B., and Latifi, S. (2005). Reliability modeling in wireless sensor networks. *International Journal of Information Technology*, **11**(2), 1–8.

Biswas, S. and Morris, R. (2004). Opportunistic routing in multi-hop wireless networks. *SIGCOMM Comput. Commun. Rev.*, **34**(1), 69–74.

Bluetooth, S. (2010). Bluetooth core specification version 4.0. *Specification of the Bluetooth System*.

Boulis, A. (2007). Castalia: Revealing pitfalls in designing distributed algorithms in wsn. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 407–408, New York, NY, USA. ACM.

Bruneo, D., Puliafito, A., and Scarpa, M. (2010). Dependability evaluation of wireless sensor networks: Redundancy and topological aspects. In *Sensors, 2010 IEEE*, pages 1827–1831.

Bruneo, D., Puliafito, A., and Scarpa, M. (2011). Energy control in dependable wireless sensor networks: a modelling perspective. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, **225**(4), 424–434.

C. Mallanda, A. Suri, V. K. S. I. R. K. A. D. and Sastry., S. (2005). Simulating wireless sensor networks with omnet++. `http://scc.acad.bg/ncsa/articles/library/Switches\%20and\%20Networks/OMNET++\%20SensorSimulator.pdf`, acessado em 09 de Novembro de 2015.

Cano, C., Bellalta, B., Barcelo, J., Oliver, M., and Sfairopoulou, A. (2009). Analytical model of the lpl with wake up after transmissions mac protocol for wsns. In *Wireless Communication Systems, 2009. ISWCS 2009. 6th International Symposium on*, pages 146–150.

Carrozza, G. (2008). *Software Faults Diagnosis in Complex OTS-Based Critical Systems*. Ph.D. thesis, Dipartimento di Informatica e Sistemistica - Universita degli Studi di Napoli Federico II.

Chang, C.-C., Nagel, D., and Muftic, S. (2007). Assessment of energy consumption in wireless sensor networks: A case study for security algorithms. In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, pages 1–6.

Chatterjea, S. and Havinga, P. (2003). A dynamic data aggregation scheme for wireless sensor networks. In *ProRISC 2003, 14th Workshop on Circuits, Systems and Signal Processing*.

Chen, Y., Chuah, C.-N., and Zhao, Q. (2005). Sensor placement for maximizing lifetime per unit cost in wireless sensor networks. In *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 1097–1102 Vol. 2.

Cinque, M., Cotroneo, D., Di Martino, C., Russo, S., and Testa, A. (2009). Avr-inject: A tool for injecting faults in wireless sensor nodes. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8.

Corporation, O. (2014). Javaserver pages (jsp). `http://jcp.org/en/jsr/detail?id=245`, acessado em 05 de Janeiro de 2014.

Dâmaso, A., Freitas, D., Rosa, N., Silva, B., and Maciel, P. (2013). Evaluating the power consumption of wireless sensor network applications using models. *Sensors*, **13**(3), 3473.

Dâmaso, A., Rosa, N., and Maciel, P. (2014a). Reliability of wireless sensor networks. *Sensors*, **14**(9), 15760–15785.

Dâmaso, A., Rosa, N., and Maciel, P. (2014b). Using coloured petri nets for evaluating the power consumption of wireless sensor networks. *International Journal of Distributed Sensor Networks (IJDSN)*, **2014**, 13.

Damosso, E. and Correia, L. (1999). *COST Action 231: Digital Mobile Radio Towards Future Generation Systems : Final Report*. EUR (Series). European Commission.

de Berkeley, U. (2011). The network simulator (ns-2). `http://www.isi.edu/nsnam/ns/`, acessado em 05 de Janeiro de 2014.

de Paz Alberola, R. and Pesch, D. (2008). Avroraz: Extending avrora with an ieee 802.15.4 compliant radio chip model. In *Proceedings of the 3Nd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, PM2HW2N '08, pages 43–50, New York, NY, USA. ACM.

Dementyev, A., Hodges, S., Taylor, S., and Smith, J. (2013). Power consumption analysis of bluetooth low energy, zigbee and ant sensor nodes in a cyclic sleep scenario. In *Wireless Symposium (IWS), 2013 IEEE International*, pages 1–4. IEEE.

Desel, J. and Reisig, W. (1998). Place/transition petri nets. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 122–173. Springer Berlin Heidelberg.

Drytkiewicz, W., Sroka, S., Handziski, V., Köpke, A., and Karl, H. (2003). A mobility framework for omnet++. *in Proceedings of the 3rd International OMNeT++ Workshop*.

Du, W., Navarro, D., Mieyeville, F., and O'Connor, I. (2011). Idea1: A validated system c-based simulator for wireless sensor networks. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 825–830.

Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462.

e Silva Nogueira, B. C. (2010). *Avalição de desempenho e consumo de energia de aplicações embarcadas: uma estrategia baseada em modelos da arquitetura de hardware e no código da aplicação*. Master's thesis, Departamento de Ciência da Computacao da Universidade Federal de Pernambuco.

Egea-Lopez, E., Ponce-Marin, F., and Vales-Alonso, J. (2006). Obiwan: wireless sensor networks with omnet++. In *Electrotechnical Conference, 2006. MELECON 2006. IEEE Mediterranean*, pages 777–780.

Ericson, C. A. (2005). *Hazard Analysis Techniques for System Safety*. Wiley.

Eriksson, J., Österlind, F., Finne, N., Tsiftes, N., Dunkels, A., Voigt, T., Sauter, R., and Marrón, P. J. (2009). Cooja/mspsim: Interoperability testing for wireless sensor networks. In *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 27:1–27:7, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Featherlight (2011). Nesct. `http://nesct.sourceforge.net/`, acessado em 05 de Janeiro de 2014.

Ferreia, L. A. (2009). *Estudo de propagação de frequência ISM para aplicação em Rede de Sensores*. Master's thesis, PUC-Campinas, Campinas, Brasil.

Friis, H. (1946). A note on a simple transmission formula. *Proceedings of the IRE*, **34**(5), 254–256.

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., and Culler, D. (2003). The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, PLDI '03, pages 1–11, New York, NY, USA. ACM.

Ghaffari, A. and Rahmani, A.-M. (2008). Fault tolerant model for data dissemination in wireless sensor networks. In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 4, pages 1–8.

Glaser, J., Weber, D., Madani, S., and Mahlknecht, S. (2008). Power aware simulation framework for wireless sensor networks and nodes. *EURASIP Journal on Embedded Systems*, **2008**(1), 369178.

GoPivotal, I. (2014). Spring framework. `http://spring.io/`, acessado em 05 de Janeiro de 2014.

Haase, J., Damm, M., Glaser, J., Moreno, J., and Grimm, C. (2009). Systemc-based power simulation of wireless sensor networks. In *Specification Design Languages, 2009. FDL 2009. Forum on*, pages 1–4.

Haase, J., Molina, J., and Dietrich, D. (2011). Power-aware system design of wireless sensor networks: Power estimation and power profiling strategies. *Industrial Informatics, IEEE Transactions on*, **7**(4), 601–613.

Haenggi, M. (2004). Opportunities and challenges in wireless sensor networks. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, pages 1.1–1.14.

Hamby, D. (1994). A review of techniques for parameter sensitivity analysis of environmental models. *Environmental Monitoring and Assessment*, **32**(2), 135–154.

Hedetniemi, S. M., Hedetniemi, S. T., and Liestman, A. L. (1988). A survey of gossiping and broadcasting in communication networks. *Networks*, **18**(4), 319–349.

Heinzelman, W. (2000). *Application-Specific Protocol Architectures for Wireless Networks*. Ph.D. thesis, Massachusetts Institute of Technology.

Heinzelman, W., Chandrakasan, A., and Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, pages 10 pp. vol.2–.

Henderson, T. R., Roy, S., Floyd, S., and Riley, G. F. (2006). Ns-3 project goals. In *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*, WNS2 '06, New York, NY, USA. ACM.

Henley, E. J. and Kumamoto, H. (1981). *Reliability engineering and risk assessment*, volume 193. Prentice-Hall Englewood Cliffs (NJ).

Hill, J. and Culler, D. (2002). Mica: a wireless platform for deeply embedded networks. *Micro, IEEE*, **22**(6), 12–24.

Hiltunen, J., Ala-Louko, M., and Taumberger, M. (2012). Experimental performance evaluation of pobicos middleware for wireless sensor networks. *CN*, **2012**, 7:7–7:7.

Horvat, G., Sostaric, D., and Zagar, D. (2012). Power consumption and rf propagation analysis on zigbee xbee modules for atpc. In *Telecommunications and Signal Processing (TSP), 2012 35th International Conference on*, pages 222–226.

IDE, C. (2014). Ace editor. `http://ace.c9.io/`, acessado em 05 de Janeiro de 2014.

Intanagonwiwat, C., Govindan, R., and Estrin, D. (2000). Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 56–67, New York, NY, USA. ACM.

Jacinto, R. M. P. (2012). *Modelação da Propagação numa Rede de Sensores sem Fios*. Master's thesis, Faculdade de Ciência e Tecnologia, Universidade de Lisboa, Lisboa, Portugal.

Jain, R. (1991). *The art of computer systems performance analysis*. John Wiley & Sons.

Jardosh, S. and Ranjan, P. (2008). A survey: Topology control for wireless sensor networks. In *Signal Processing, Communications and Networking, 2008. ICSCN '08. International Conference on*, pages 422–427.

Jensen, K. (1997). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Coloured Petri Nets. Springer.

Jensen, K. and Kristensen, L. (2009). *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer.

Jensen, K., Kristensen, L. M., and Wells, L. (2007). Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *Int. J. Softw. Tools Technol. Transf.*, **9**(3), 213–254.

jQuery Foundation., T. (2014). Jquery. `http://www.jquery.com/`, acessado em 05 de Janeiro de 2014.

Kim, H., Seok, Y., Choi, N., Choi, Y., and Kwon, T. (2005a). Optimal multi-sink positioning and energy-efficient routing in wireless sensor networks. In C. Kim, editor, *Information Networking. Convergence in Broadband and Mobile Networking*, volume 3391 of *Lecture Notes in Computer Science*, pages 264–274. Springer Berlin Heidelberg.

Kim, H., Seok, Y., Choi, N., Choi, Y., and Kwon, T. (2005b). Optimal multi-sink positioning and energy-efficient routing in wireless sensor networks. In *Information Networking. Convergence in Broadband and Mobile Networking*, pages 264–274. Springer.

Ko, Y.-B., Choi, J.-M., and Kim, J.-H. (2004). A new directional flooding protocol for wireless sensor networks. In H.-K. Kahng and S. Goto, editors, *Information Networking. Networking Technologies for Broadband and Mobile Networks*, volume 3090 of *Lecture Notes in Computer Science*, pages 93–102. Springer Berlin Heidelberg.

Krishnamachari, B., Estrin, D., and Wicker, S. (2002). The impact of data aggregation in wireless sensor networks. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 575–578.

Kuo, W. and Zuo, M. (2003). *Optimal Reliability Modeling: Principles and Applications*. Wiley.

Kurkowski, S., Camp, T., and Colagrosso, M. (2005). Manet simulation studies: The incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, **9**(4), 50–61.

Kwon, H., Kim, T. H., Choi, S., and Lee, B. G. (2005). Cross-layer lifetime maximization under reliability and stability constraints in wireless sensor networks. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 5, pages 3285–3289 Vol. 5.

Lajara, R., Pelegrí-Sebastiá, J., and Solano, J. J. P. (2010). Power consumption analysis of operating systems for wireless sensor networks. *Sensors*, **10**(6), 5809–5826.

Landsiedel, O., Wehrle, K., and Götz, S. (2004). Accurate prediction of power consumption in sensor networks. In *Proc. IEEE Second Workshop on Embedded Networked Sensors (EmNetS-II)*, pages 37–44.

Lee, J.-S., Su, Y.-W., and Shen, C.-C. (2007). A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pages 46–51.

Levis, P., Lee, N., Welsh, M., and Culler, D. (2003). Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 126–137, New York, NY, USA. ACM.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D. (2005). Tinyos: An operating system for sensor networks. In W. Weber, J. Rabaey, and E. Aarts, editors, *Ambient Intelligence*, pages 115–148. Springer Berlin Heidelberg.

Li, Q.-Q., Gong, H., Liu, M., Yang, M., and Zheng, J. (2011). On prolonging network lifetime through load-similar node deployment in wireless sensor networks. *Sensors*, **11**(4), 3527.

Li, W., Zhang, X., Tan, W., and Zhou, X. (2009). H-tossim: Extending tossim with physical nodes. *Wireless Sensor Network*, **1**(4), 324–333.

Liao, S., Martin, G., Swan, S., and Grötker, T. (2002). *System design with SystemC*. Kluwer Academic Pub.

Limited, O. R. (2014). jfreechart free. `http://www.jfree.org/jfreechart`, acessado em 05 de Janeiro de 2014.

Lin, P., Qiao, C., and Wang, X. (2004). Medium access control with a dynamic duty cycle for sensor networks. In *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, volume 3, pages 1534–1539 Vol.3.

Loureiro, A. A., Nogueira, J. M. S., Ruiz, L. B., de Freitas Mini, R. A., Nakamura, E. F., and Figueiredo, C. M. S. (2003). Redes de sensores sem fio. In *Simpósio Brasileiro de Redes de Computadores (SBRC)*, pages 179–226.

Ma, Y. and Jin, L. (2014). Sensitivity analysis of doa error on least square-based source localization in uwsan. In *Image and Signal Processing (CISP), 2014 7th International Congress on*, pages 979–983.

Maciel, P., Lins, R., and Cunha, P. (1996). *Uma Introdução às Redes de Petri e Aplicações*. Campinas – SP: Sociedade Brasileira de Computação.

Maciel, P., Trivedi, K., Matias, R., and Kim, D. (2011). *Dependability Modeling*. Hershey, Pennsylvania.

Madureira, H., de Medeiros, J., Da Costa, J., and Beserra, G. (2011). System-level power consumption modeling of a soc for wsn applications. In *Networked Embedded Systems for Enterprise Applications (NESEA), 2011 IEEE 2nd International Conference on*, pages 1–6.

Mahmood, M. A., Seah, W. K., and Welch, I. (2015). Reliability in wireless sensor networks: A survey and challenges ahead. *Computer Networks*, **79**, 166 – 187.

Manjeshwar, A. and Agrawal, D. P. (2002). Apteen: A hybrid protocol for efficient routing and comprehensive information retrieval in wireless sensor networks. In *in Proceedings International Parallel and Distributed Processing Symposium (IPDPS)*, volume 2, pages 195–202.

Manjeshwar, A., Zeng, Q.-A., and Agrawal, D. (2002). An analytical model for information retrieval in wireless sensor networks using enhanced apteen protocol. *Parallel and Distributed Systems, IEEE Transactions on*, **13**(12), 1290–1302.

Mattos, G. M. (2006). *Redes de Acesso em Banda Larga utilizando Sistemas VSAT e WiFi*. Master's thesis, PUC-Rio, Rio de Janeiro, Brasil.

MEMSIC (2014a). Wireless module iris: Iris datasheet. `http://www.memsic.com`, acessado em 05 de Janeiro de 2014.

MEMSIC (2014b). Wireless module micaz: Micaz datasheet. `http://www.memsic.com`, acessado em 05 de Janeiro de 2014.

Milner, R., Tofte, M., and Harper, R. (1990). *The Definition of Standard ML*. MIT Press, Cambridge, MA, USA.

Minakov, I. and Passerone, R. (2013). Pases: An energy-aware design space exploration framework for wireless sensor networks. *Journal of Systems Architecture*, **59**(8), 626 – 642.

Moritz, G., Golatowski, F., and Timmermann, D. (2011). A lightweight soap over coap transport binding for resource constraint networks. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 861–866.

Mueller, K. (2004). Simpy documentation. `http://simpy.readthedocs.org/`, acessado em 27 de Dezembro de 2014.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, **77**(4), 541–580.

Najnudel, M. (2004). *Estudo de Propagação em Ambientes Fechados para o Planejamento de WLANs*. Master's thesis, PUC-Rio, Rio de Janeiro, Brasil.

Naoumov, V. and Gross, T. (2003). Simulation of large ad hoc networks. In *Proceedings of the 6th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '03, pages 50–57, New York, NY, USA. ACM.

OMG (2012). OMG Systems Modeling Language (OMG SysML), Version 1.3.

Parameswaran, A. T., Husain, M. I., and Upadhyaya, S. (2009). Is rssi a reliable parameter in sensor localization algorithms: An experimental study. In *Field Failure Data Analysis Workshop (F2DA09)*.

Parhami, B. (1988). From defects to failures: A view of dependable computing. *SIGARCH Comput. Archit. News*, **16**(4), 157–168.

Park, S., Savvides, A., and Srivastava, M. B. (2000). Sensorsim: A simulation framework for sensor networks. In *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '00, pages 104–111, New York, NY, USA. ACM.

Perla, E., Catháin, A. O., Carbajo, R. S., Huggard, M., and Mc Goldrick, C. (2008). Powertossim z: Realistic energy modelling for wireless sensor network environments. In *Proceedings of the 3Nd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, PM2HW2N '08, pages 35–42, New York, NY, USA. ACM.

Polastre, J., Hill, J., and Culler, D. (2004). Versatile low power media access for wireless sensor networks. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 95–107, New York, NY, USA. ACM.

Polley, J., Blazakis, D., McGee, J., Rusk, D., and Baras, J. (2004). Atemu: a fine-grained sensor network simulator. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 145–152.

Pottie, G. and Kaiser, W. (2000). Embedding the internet: wireless integrated network sensors. *Communications of the ACM*, **43**(5), 51–58.

Prabhakar, T., Venkatesh, S., Sujay, M., Kuri, J., and Praveen, K. (2008). Simulation blocks for tossim-t2. In *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, pages 17–23.

Prayati, A., Antonopoulos, C., Stoyanova, T., Koulamas, C., and Papadopoulos, G. (2010). A modeling approach on the telosb {WSN} platform power consumption. *Journal of Systems and Software*, **83**(8), 1355 – 1363. <ce:title>Performance Evaluation and Optimization of Ubiquitous Computing and Networked Systems</ce:title>.

Purohit, N., Varadwaj, P., and Tokekar, S. (2008). Reliability analysis of wireless sensor network. In *Networks, 2008. ICON 2008. 16th IEEE International Conference on*, pages 1–6. IEEE.

Raghunathan, V., Ganeriwal, S., and Srivastava, M. (2006). Emerging techniques for long lived wireless sensor networks. *Communications Magazine, IEEE*, **44**(4), 108–114.

Rappaport, T. S. *et al.* (1996). *Wireless communications: principles and practice*, volume 2. Prentice Hall PTR New Jersey.

Riley, G. (2003). Large-scale network simulations with gtnets. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 1, pages 676–684 Vol.1.

Rivet, B. and Klepp, T. (2014). Simulavr: an avr simulator. `http://www.nongnu.org/simulavr/`, acessado em 05 de Janeiro de 2014.

Ruiz, L. B. (2003). *MANNA: Uma Arquitetura de Gerenciamento para Redes de Sensores Sem Fio*. Ph.D. thesis, Departamento de Ciência da Computacao da Universidade Federal de Minas Gerais.

Rumbaugh, J., Jacobson, I., and Booch, G. (2004). *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education.

Rusli, M., Harris, R., and Punchihewa, A. (2010). Markov chain-based analytical model of opportunistic routing protocol for wireless sensor networks. In *TENCON 2010 - 2010 IEEE Region 10 Conference*, pages 257–262.

Sahota, H., Kumar, R., and Kamal, A. (2011). A wireless sensor network for precision agriculture and its performance. *Wireless Communications and Mobile Computing*, **11**(12), 1628–1645.

Sahota, H. S. (2013). *Wireless sensor network for precision agriculture:Design, Performance Modeling and Evaluation,and Node Localization*. Ph.D. thesis, Iowa State University.

Sankarasubramaniam, Y., Akan, O. B., and Akyildiz, I. F. (2003). Esrt: Event-to-sink reliable transport in wireless sensor networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, MobiHoc '03, pages 177–188, New York, NY, USA. ACM.

Senouci, M. R., Mellouk, A., Senouci, H., and Aissani, A. (2012). Performance evaluation of network lifetime spatial-temporal distribution for {WSN} routing protocols. *Journal of Network and Computer Applications*, **35**(4), 1317 – 1328. <ce:title>Intelligent Algorithms for Data-Centric Sensor Networks</ce:title>.

SGI (2014). C language reference manual. Silicon Graphics Inc. Document Number: 007-0701-150. `http://techpubs.sgi.com`, acessado em 05 de Janeiro de 2014.

Shaikh, F., Khelil, A., and Suri, N. (2007). On modeling the reliability of data transport in wireless sensor networks. In *Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on*, pages 395–402.

Shelby, Z., Hartke, K., Bormann, C., and Frank, B. (2014). Constrained Application Protocol (CoAP). Technical Report draft-ietf-core-coap-07.txt, IETF Secretariat, Fremont, CA, USA.

Shinghal, K., Noor, A., Srivastava, N., and Singh, R. (2011). Power measurements of wireless sensor network node. *Int. J. Comput. Eng. Sci.(IJCES)*, **1**(1), 8–13.

Shnayder, V., Hempstead, M., Chen, B.-r., Allen, G. W., and Welsh, M. (2004). Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 188–200, New York, NY, USA. ACM.

Shrestha, A., Xing, L., and Liu, H. (2006). Infrastructure communication reliability of wireless sensor networks. In *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on*, pages 250–257.

Shrestha, A., Xing, L., and Liu, H. (2007). Modeling and evaluating the reliability of wireless sensor networks. In *Reliability and Maintainability Symposium, 2007. RAMS '07. Annual*, pages 186–191.

Siekkinen, M., Hiienkari, M., Nurminen, J. K., and Nieminen, J. (2012). How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*, pages 232–237. IEEE.

Silva, B., Callou, G., Tavares, E., Maciel, P., Figueiredo, J., Sousa, E., Araujo, C., Magnani, F., and Neves, F. (2013). Astro: An integrated environment for dependability and sustainability evaluation. *Sustainable Computing: Informatics and Systems*, **3**(1), 1 – 17.

Silva, I., Guedes, L. A., Portugal, P., and Vasques, F. (2012). Reliability and availability evaluation of wireless sensor networks for industrial applications. *Sensors*, **12**(1), 806–838.

Somov, A., Minakov, I., Simalatsar, A., Fontana, G., and Passerone, R. (2009). A methodology for power consumption evaluation of wireless sensor networks. In *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1–8.

Srivastava, V. and Motani, M. (2005). Cross-layer design: a survey and the road ahead. *Communications Magazine, IEEE*, **43**(12), 112–119.

Stann, F. and Heidemann, J. (2003). Rmst: Reliable data transport in sensor networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, pages 102–112, Anchorage, Alaska, USA.

Suhonen, J., Hämäläinen, T. D., and Hännikäinen, M. (2009). Availability and end-to-end reliability in low duty cycle multihop wireless sensor networks. *Sensors*, **9**(3), 2088–2116.

Synopsys, I. (2002). System c 2.0.1 - user guide. `http://www.cse.iitd.ac.in/~panda/SYSTEMC/LangDocs/UserGuide20.pdf`, acessado em 05 de Janeiro de 2014.

Tavares, E., Silva, B., and Maciel, P. (2008). An environment for measuring and scheduling time-critical embedded systems with energy constraints. In *Software Engineering*

and Formal Methods, 2008. SEFM '08. Sixth IEEE International Conference on, pages 291–300.

TinyOS (2008). Lesson 4: Composing components to send and receive messages. disponível em `http://www.tinyos.net/nest/doc/tutorial/lesson4.html`. [acessado em 17 de Julho de 2015].

Titzer, B., Lee, D., and Palsberg, J. (2005). Avrora: scalable sensor network simulation with precise timing. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 477–482.

Trivedi, K. S. (2002). *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John˜Wiley˜&˜Sons,˜Inc., second edition.

Twitter, I. (2014). Twitter bootstrap. `http://www.getbootstrap.com/`, acessado em 05 de Janeiro de 2014.

Varga, A. (2001). The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, volume 9, page 319–324. sn.

Venkatesan, L., Shanmugavel, S., and Subramaniam, C. (2013). A survey on modeling and enhancing reliability of wireless sensor network. *Wireless Sensor Network*, **2013**(5), 41–51.

Vieira, M., Coelho, C., da Silva, D.C., J., and da Mata, J. (2003). Survey on wireless sensor network devices. In *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, volume 1, pages 537–544 vol.1.

Von Eicken, T., Culler, D. E., Goldstein, S. C., and Schauser, K. E. (1992). Active messages: a mechanism for integrated communication and computation. *In Proceedings of the 19th annual international symposium on Computer architecture (ISCA '92)*, **20**(2), 256–266.

Wang, C., Sohraby, K., Lawrence, V., Li, B., and Hu, Y. (2006). Priority-based congestion control in wireless sensor networks. In *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, volume 1, pages 34–40.

Wang, C., Xing, L., Vokkarane, V., and Sun, Y. (2012). Reliability analysis of wireless sensor networks using different network topology characteristics. In *Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2012 International Conference on*, pages 12–16.

Weaver, V. and McKee, S. (2008). Are cycle accurate simulations a waste of time. In *Proc. 7th Workshop on Duplicating, Deconstructing, and Debunking*, pages 40–53.

Weber, D., Glaser, J., and Mahlknecht, S. (2007). Discrete event simulation framework for power aware wireless sensor networks. In *Industrial Informatics, 2007 5th IEEE International Conference on*, volume 1, pages 335–340.

Weingartner, E., vom Lehn, H., and Wehrle, K. (2009). A performance comparison of recent network simulators. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–5.

Westergaard, M. (2011). Access/cpn 2.0: A high-level interface to coloured petri net models. In L. Kristensen and L. Petrucci, editors, *Applications and Theory of Petri Nets*, volume 6709 of *Lecture Notes in Computer Science*, pages 328–337. Springer Berlin Heidelberg.

Westergaard, M. and Verbeek, H. E. (2014). Cpn tools. `http://www.cpntools.org/`, acessado em 24 de Janeiro de 2014.

Whittle, P. (1963). Prediction and regulation by linear least-square methods.

Witchel, E. and Rosenblum, M. (1996). Embra: Fast and flexible machine simulation. *SIGMETRICS Perform. Eval. Rev.*, **24**(1), 68–79.

Xu, K. (2008). *"Device Deployment Strategies for Large-scale Wireless Sensor Networks"*. Ph.D. thesis, Kingston, Ontario, Canada.

Xu, K., Hassanein, H., and Takahara, G. (2005). Relay node deployment strategies in heterogeneous wireless sensor networks: multiple-hop communication case. In *Sensor and Ad Hoc Communications and Networks, 2005. IEEE SECON 2005. 2005 Second Annual IEEE Communications Society Conference on*, pages 575–585.

Xu, Y. (2002). *Adaptive Energy Conservation Protocols for Wireless Ad Hoc Routing*. Ph.D. thesis, Los Angeles, CA, USA. AAI3093948.

YACOUB, M. D. (1993). *Fundamentals of Mobile Radio Engineering*. CRC PRESS.

Yang, Y., Fonoage, M. I., and Cardei, M. (2010). Improving network lifetime with mobile wireless sensor networks. *Computer Communications*, **33**(4), 409 – 419.

Younis, M. and Akkaya, K. (2008). Strategies and techniques for node placement in wireless sensor networks: A survey. *Ad Hoc Networks*, **6**(4), 621–655.

Zang, X., Sun, H., and Trivedi, K. (1999). A bdd-based algorithm for reliability analysis of phased-mission systems. *Reliability, IEEE Transactions on*, **48**(1), 50–60.

Zhang, H., Arora, A., ri Choi, Y., and Gouda, M. G. (2005). Reliable bursty convergecast in wireless sensor networks. In *In ACM MobiHoc*, pages 266–276. ACM Press.

Zhou, G., He, T., Krishnamurthy, S., and Stankovic, J. A. (2006). Models and solutions for radio irregularity in wireless sensor networks. *ACM Trans. Sen. Netw.*, **2**(2), 221–262.

Zonouz, A., Xing, L., Vokkarane, V., and Sun, Y. (2014). A time-dependent link failure model for wireless sensor networks. In *Reliability and Maintainability Symposium (RAMS), 2014 Annual*, pages 1–7.

Apêndice

A

Telas do EDEN

O EDEN, como já foi mencionado no Capítulo 6, é formado por quatro ferramentas: editor, tradutor, avaliador e gerenciador. O usuário irá trabalhar apenas no editor, o qual possui uma interface gráfica. As demais ferramentas serão utilizadas pelo editor para fazer a avaliação da RSSF, atendendo parte da metodologia apresentada no Capítulo 4. Este apêndice irá dar uma visão geral de como utilizar o EDEN (principalmente, o editor) para elaborar e avaliar uma RSSF de acordo com a metodologia proposta.

A.1 Tela Inicial

A Figura A.1 mostra a primeira tela do editor. Por ser um ambiente Web, nós entendemos que o EDEN possa ser usado por vários usuários ao mesmo tempo. Por este motivo, foi criar um gerenciador de contas de usuário e, quando o usuário tenta usar o EDEN, é necessário que ele se identifique-se usando uma tela de login (apresentada nesta figura).

Após o usuário se logar, é apresentado a tela de gerenciamento dos projetos RSSF do usuário (como mostra a Figura A.2). Nós entendemos que um projeto seja um conjunto de código relacionados ao desenvolvimento da RSSF (*e.g.*, código fonte da aplicação, configuração da RSSF, entre outros). Cada projeto deve ser independente do outro e deve representar uma RSSF diferente da outra.

Caso o usuário queira criar um novo projeto, ele deve clicar em "new Project"e digitar o nome do novo projeto. Além disso, o usuário deve utilizar metodologia proposta para desenvolver um projeto da RSSF. As telas relacionadas com o projeto é mostrado na seção seguinte.

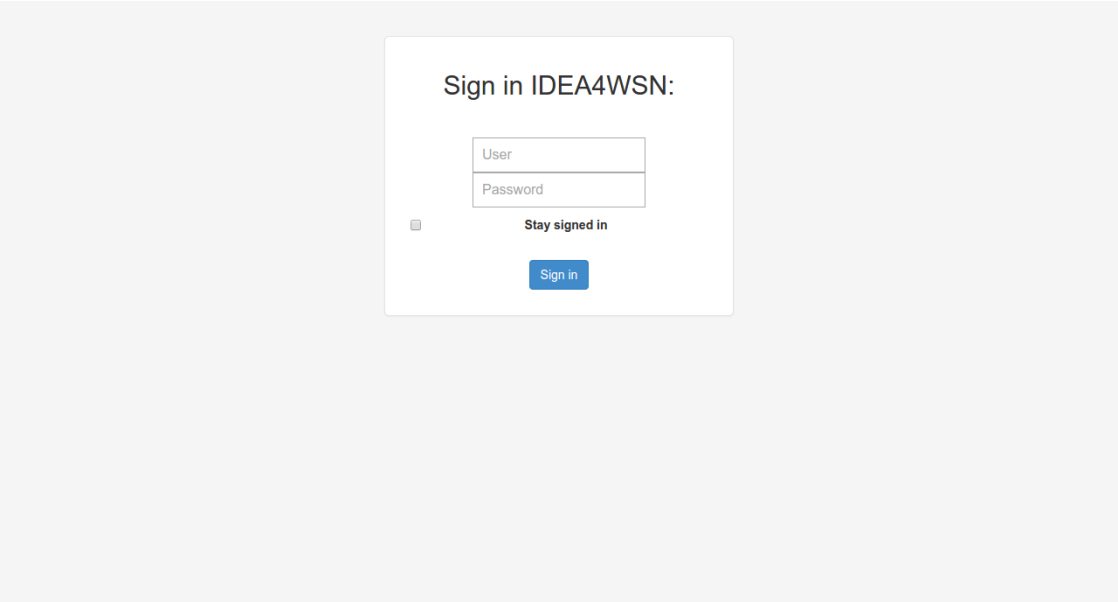


Figura A.1 Tela de login do EDEN.

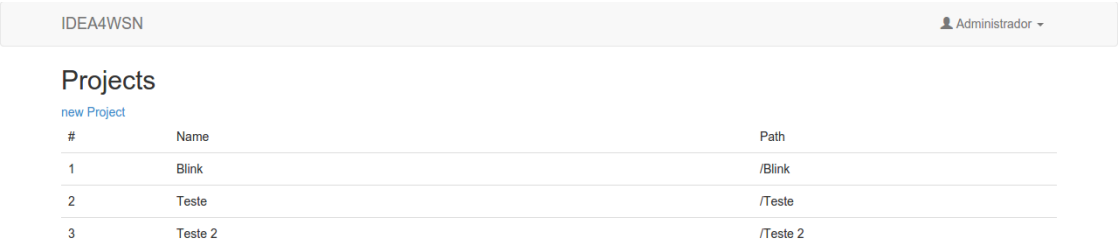


Figura A.2 Tela do gerenciador de projetos.

A.2 Projeto

Cada projeto deve ser desenvolvido em abas diferentes. Por este motivo, quando o usuário abre um projeto a partir da tela de gerenciamento de projetos, uma nova aba é aberta (como mostra a Figura A.3). Usando esta tela, o usuário deve seguir as cinco fases da metodologia definidas no Capítulo 4. Na fase *Planejamento*, o usuário deve criar arquivos textos para auxiliar no levantamento de requisitos e na elaboração do desenvolvimento. Na fase *Codificação*, o usuário deve criar arquivos *.nc* para desenvolver o código da aplicação na linguagem nesC e arquivos *.wsn* para configurar a rede. As Figuras A.3 (a) e A.3 (b) mostram a edição de um arquivo *.nc* e de um arquivo *.wsn*, respectivamente. Adicionalmente, os arquivos *.wsn* são baseados no XML, como foi apresentado no Apêndice B, e o Editor utiliza uma interface gráfica para facilitar a configuração da RSSF e criar os arquivos *.wsn* facilmente.

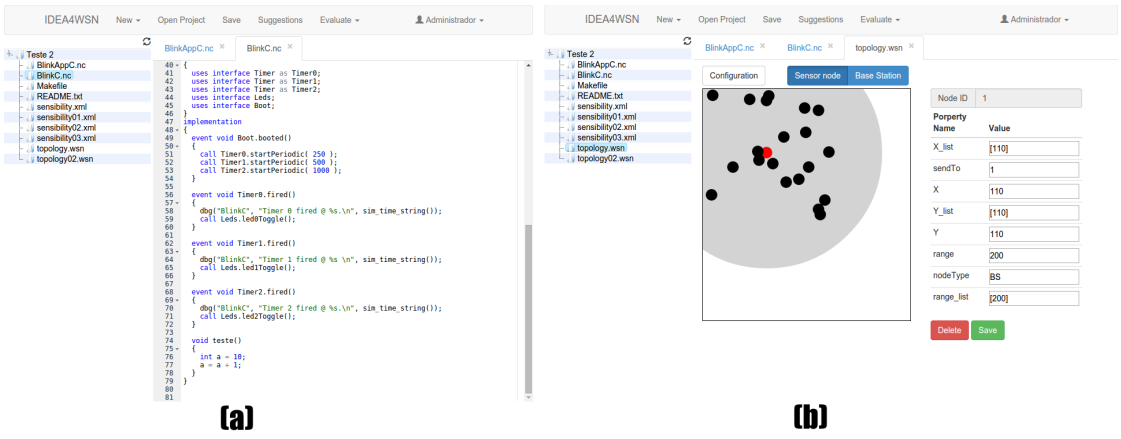


Figura A.3 Tela do projeto aberto.

O menu acima serve para auxiliar em duas fases da metodologia: o menu *suggestions* está relacionado com a fase *Otimização* e o menu *evaluate*, com a fase *Validação*. O menu *suggestions* serve para dar sugestões de melhorias ao projeto atual. Estas sugestões dadas pelo editor foram mostradas na Seção 6.3.3. A Figura A.4 mostra um exemplo de sugestões dadas ao usuário.

O menu *evaluate* deve ser utilizado para avaliar o consumo de energia e a confiabilidade da aplicação e/ou da configuração de rede (fase *Validação*). Além disso, este menu possui opções de comparar o consumo de energia entre duas ou mais funções implementadas na aplicação, e comparar o consumo de energia e a confiabilidade da RSSF (fase *Otimização*). Caso o usuário deseje avaliar várias configurações de rede (*e.g.*, os nós sensores enviando pacotes com tamanhos diferentes, em posições diferentes, entre

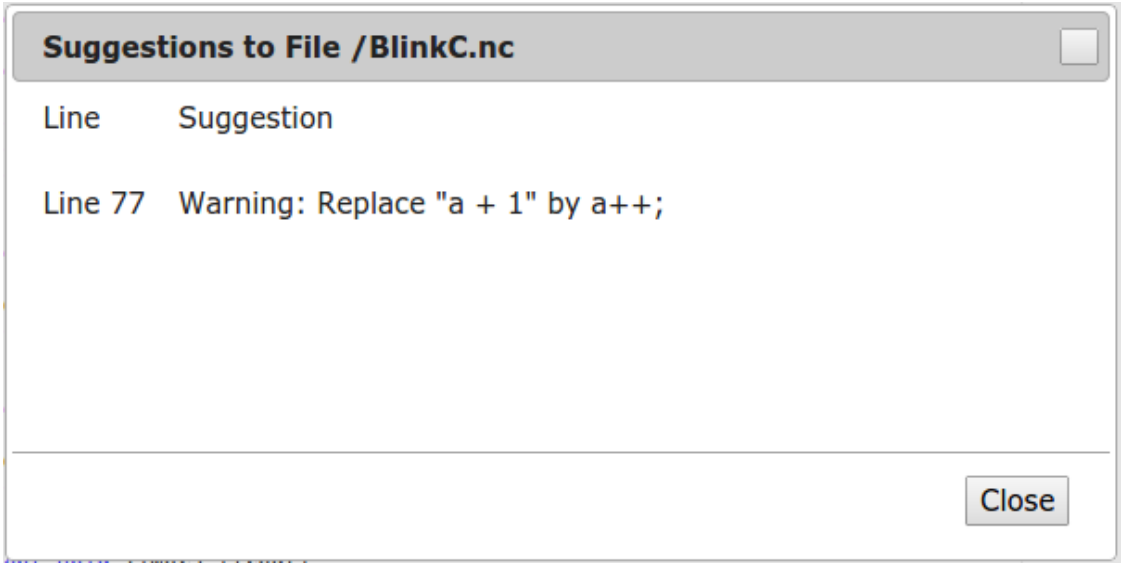


Figura A.4 Sugestões dados ao usuário.

outras coisas), ele pode fazer análise de sensibilidade da RSSF. As telas dessas opções são mostradas na Figura A.5.

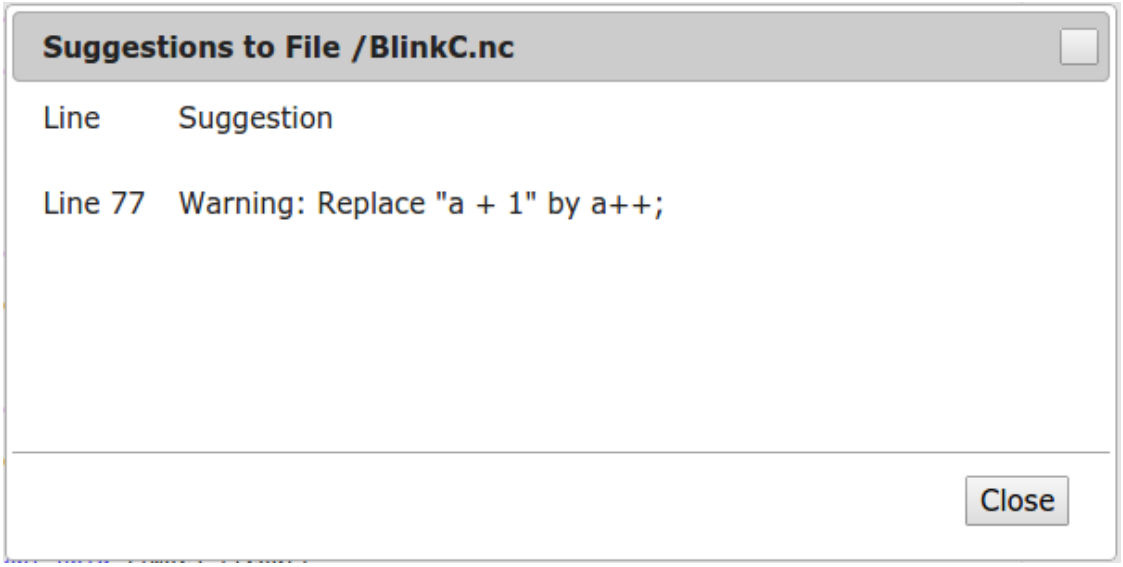


Figura A.5 Telas para avaliar uma RSSF.

Todas as opções desse menu irão utilizar as outras ferramentas (gerenciador, tradutor e avaliador). Essas três ferramentas não possuem interface gráfica. Elas são executadas através da linha de comando e a sua configuração é feita por um aquivo.

B

Arquivos

Este apêndice descreve os formatos de arquivos utilizados para representar a rede da RSSF e um modelo de protocolo.

B.1 Protocolos

Os protocolos de rede (independentemente da camada) é representado por dois arquivos: um arquivo de configuração, contendo as propriedades da rede e dos nós sensores; e um modelo, o modelo funcional do protocolo. O modelo do protocolo já foi apresentado na Seção 5.3. Sendo assim, esta seção irá descrever o outro arquivo, o arquivo de configuração. O arquivo de configuração é baseado em XML, como pode ser visto abaixo:

```
 1. <configuration>
 2.  <name>LEACH</name>
 3.  <layer>network</layer>
 4.  <description>Low-Energy Adaptive Clustering Hierarchy (LEACH).</description>
 5.
 6.  <variable_list>
 7.
 8.   <property>
 9.    <name>leach_setup_time</name>
10.    <type>INT</type>
11.    <default>80</default>
12.    <description>Each X time, the setup phrase is executed.</description>
13.   </property>
14.
15.   <property>
16.    <name>leach_aggregation_time</name>
17.    <type>INT</type>
18.    <default>10</default>
19.    <description>Each X time, the aggregation phrase is executed.</description>
20.   </property>
```

```
21.
22.   <property>
23.    <name>leach_cluster_no</name>
24.    <type>INT</type>
25.    <default>8</default>
26.    <description>Number of cluster in the WSN.</description>
27.   </property>
28.
29.   <property>
30.    <name>leach_round</name>
31.    <type>INT</type>
32.    <default>0</default>
33.    <description>It should be zero.</description>
34.   </property>
35.
36.   <property>
37.    <name>clusterHeadList</name>
38.    <type>list INT</type>
39.    <default>[0]</default>
40.    <description>It should be zero.</description>
41.   </property>
42.
43.  </variable_list>
44.
45.  <node_propreties>
46.
47.   <property>
48.    <name>chId</name>
49.    <type>INT</type>
50.    <default>0</default>
51.    <description>it define the ID of the cluster head (CH) which
                  this node should send the data.</description>
52.   </property>
53.
54.  </node_propreties>
55.
56. </configuration>
```

A tag *configuration* é a principal tag por delimitar o início e o fim da configuração. Dentro dessa tag, a gente pode dividir as demais em três partes: tags informativas, tags de configuração geral e tags de configuração específica do nó sensor. As tags informativas contém informações que serão mostradas para o usuário. Existem apenas três tags que são *name*, está relacionada com o nome do protocolo (neste caso, o LEACH); *layer*, está relacionada com a pilha de protocolo (neste caso, é a camada de rede); e *description*, uma breve descrição do protocolo.

A tag *variable_list* contém as configurações gerais do protocolo (linha 6 a 43) e a *tag node_propreties* contém as configurações do nó sensor (linha 45 a 54). Um configuração geral ou do nó sensor é representada pela tag *property* que as sub tags *name*,

o nome da configuração; *type*, o tipo da variável (*e.g.*, inteiro, decimal, caracter); *default*, valor padrão; e *description*, descrição da configuração. Por exemplo, a configuração denominada *leach_cluster_no* representa a quantidade de *clusters* que deve ter na rede (linha 22 a 27).

B.2 Arquivo de Rede

O Arquivo de Rede (extensão .wsn) é um arquivo baseado em XML com as configurações da RSSF (*e.g.*, protocolos, quantidade de nós sensores, tamanho da rede) e dos nós sensores (*e.g.*, localização, identificador, nível de energia, radio de alcance de comunicação). Abaixo está um exemplo de Arquivo de Rede:

```
 1. <topology>
 2.
 3.  <configuration>
 4.       <property name="sensorNumber" value="2" />
 5.       <property name="routingProtocol" value="DIRECT" />
 6.       <property name="areaSize" value="100x100" />
 7.       ...
 8.  </configuration>
 9.
10.  <nodes>
11.
12.    <node id="1">
13.       <property name="energyLevel" value="0.001" />
14.       <property name="radioRange" value="100" />
15.       <property name="Y" value="33" />
16.       <property name="X" value="10" />
17.       ...
18.    </node>
19.    ...
20.  </nodes>
21. </topology>
```

A tag *topology* é a principal tag do documento por delimitar o início e o fim do documento Arquivo de Rede. A tag *configuration* representa as configurações gerais da rede, as quais são definidas individualmente pela tag *property*; tal como informar a quantidade de nós sensores (linha 4), o tamanho da rede (linha 6) e os protocolos (linha 5). A tag *nodes* define o escopo de configuração dos nós sensores da rede (linhas 12 a 20). Um nó sensor é representado pela tag *node* (linhas 12 a 17) e as suas propriedades são representadas pela tag *property* (linha 13). Por exemplo, a linha 13 define o nível de energia inicial do nó sensor com ID 1. Adicionalmente, as propriedades dos nós sensores estão relacionadas ao protocolos escolhidos (veja a Seção B.1). Em outras

palavras, dependendo do protocolo escolhidos, novas propriedades dos nós sensores poderão existir.

Como a comunicação entre os nós sensores é sem fio, não é necessário definir as ligações entre os nós sensores. Esse tipo de atividade é definido em tempo de simulação usando as coordenadas dos nós sensores, o raio de alcance de comunicação e o modelo de propagação. Adicionalmente, por enquanto, só foi implementado o ambiente livre (sem interferência e atenuação do sinal).

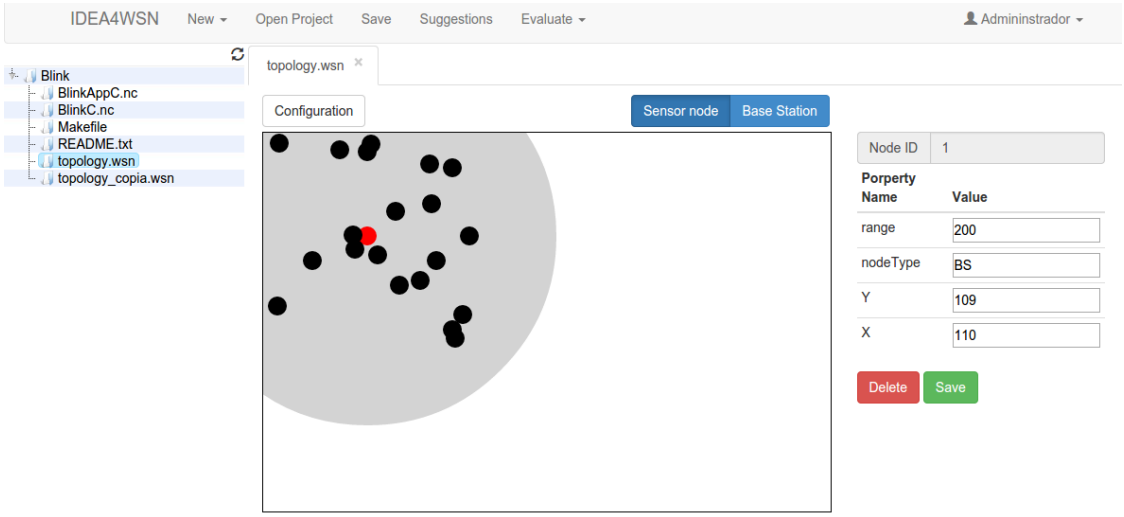


Figura B.1 Tela do IDEA4WSN configurando uma rede da RSSF.

Vale ressaltar que, para o usuário, a estrutura interna do Arquivo de Rede não é importante visto que ele (usuário) só irá trabalhar com a parte gráfica da ferramenta. O usuário irá criar e configurar a rede através de um GUI (como é visto na Figura B.1). A parte central da tela possui os nós sensores (representados por pontos pretos) e nó sorvedouro (representado pelo ponto vermelho). O lado direito possui as informações de configuração do nó sensor selecionado. E o botão *configuration* irá configurar a rede, tal como selecionar os protocolos e o algoritmo de roteamento. Todas as informações são salvadas dentro de um arquivo ".wsn"(como foi dito anteriormente). No exemplo da Figura B.1, o nome do arquivo é "topology.wsn".

B.3 Arquivo de Configuração da Analise de Sensibilidade

O Arquivo de Configuração da Analise de Sensibilidade (extensão .sac) é baseado no XML e deve ser usado para definir os fatores e níveis da análise de sensibilidade da RSSF (*e.g.*, os protocolos de roteamento que devem ser utilizados, o tamanho da rede, a quantidade de nós sensores, entre outros). Um exemplo desse arquivo é apresentado abaixo:

```
 1. <sensibility>
 2.  <wsn_sizes>
 4.   <wsn_size>
 5.     <nodeNumber>100</nodeNumber>
 6.     <area>500</area>
 7.     <bsX>250</bsX>
 8.     <bsY>250</bsY>
 9.   </wsn_size>
10.  </wsn_sizes>
11.  <reliability>
12.    <os>1.0</os>
13.    <application>1.0</application>
14.    <hardware>1.0</hardware>
15.    <link>1.0</link>
16.    <batteryMin>0.0</batteryMin>
17.    <batteryMax>100.0</batteryMax>
18.  </reliability>
19.  <application_layer>
20.   <configuration>
21.     <id>1438346150550</id>
22.     <name>application_model</name>
23.     <conf>{sendTo=1}</conf>
24.     <var>{app_time=2, app_energy=0.0013}</var>
25.   </configuration>
26.  </application_layer>
27.  <network_layer>
28.   <configuration>
29.     <id>1438346150550</id>
30.     <name>direct</name>
31.     <conf>{}</conf>
32.     <var>{net_max_time=5, net_time=1}</var>
33.   </configuration>
34.  </network_layer>
35.  <link_layer>
36.   <configuration>
37.     <id>1438346150550</id>
38.     <name>perfect_mac</name>
39.     <conf>{X=0, Y=0, range=250, nodeType=NODE}</conf>
40.     <var>{packetSize=100, battery=5.0}</var>
41.   </configuration>
```

```
42.   <configuration>
43.     <id>1438346150550</id>
44.     <name>perfect_mac</name>
45.     <conf>{X=0, Y=0, range=250, nodeType=NODE}</conf>
46.     <var>{packetSize=1000, battery=5.0}</var>
47.   </configuration>
48.  </link_layer>
49.  <deploy>
50.   <configuration>
51.     <id>1438346150555</id>
52.     <name>RANDOM UNIFORM</name>
53.     <conf>{}</conf>
54.     <var>{}</var>
55.   </configuration>
56.  </deploy>
57. <regions>
58.  <region>
59.    <id>1</id>
60.    <x>1</x>
61.    <y>1</y>
62.    <width>10</width>
63.    <height>10</height>
64.    <nodes>2</nodes>
65.   </region>
66.  </regions>
67.  <stopCriteriaStorage type="time">
68.   <time>10</time>
69.   <max>100000</max>
70.  </stopCriteriaStorage>
71. </sensibility>
```

Todos os fatores e níveis são definidos dentro da tag *sensibility*, utilizando um dos cinco sub-tags: (*wsn_sizes*, *reliability*, *application_layer*, *network_layer*, *link_layer*, *deploy*, *regions* e *stopCriteriaStorage*). Cada sub-tag será explicada nos próximos parágrafos.

A tag *wsn_sizes* agrupa os fatores das configurações básicas da RSSF (tamanho da rede, quantidade de nós sensores e localização do nó sorvedouro). A tag *wsn_size* representa uma configuração básica da RSSF. Dentro dela, nós devemos definir o tamanho da área (a tag *area*), o posicionamento do nó sorvedouro (as tags *bsX* e *bsY*) e a quantidade de nós sensores (a tag *nodeNumber*). Além disso, a tag *wsn_sizes* permite que várias configurações básicas sejam definidas dentro dela. A única exigência é que as configurações básicas tenham, pelo menos, um fator (tamanho da rede, quantidade de nós sensores ou a localização do nó sorvedouro) com valor diferente das demais configurações. Adicionalmente, a posição do nó sorvedouro pode ser alterada dependendo do algoritmo de implantação escolhido.

A tag *reliability* configura a confiabilidade dos elementos do nó sensor e do enlace de

comunicação (como pode ser visto na Seção 5.4.3). Por exemplo, a tag *os* e *application* definem a confiabilidade do sistema operacional e da aplicação, respectivamente, do nó sensor. As tags *batteryMin* e *batteryMax* são utilizadas para calculo a confiabilidade da bateria de acordo com a Equação 7.6.1.

As tags *application_layer*, *network_layer* e *link_layer* servem para configurar as camadas aplicação, rede e enlace (respectivamente). Cada configuração é representada pela tag *configuration*, a qual possui três sub tags importantes: a tag name, indicando o nome do protocolo utilizado (*e.g.*, DIRECT); a tag *conf*, para configurar os nós sensores da simulação; e a tag *var*, para criar as variáveis globais da avaliação. Esses dados são obtidos no arquivo de configuração do protocolo (veja a Seção B.1). Tal como a tag *wsn_sizes*, essas as tags *application_layer*, *network_layer* e *link_layer* permitem que um conjunto de configurações possam ser definidos, exigindo apenas que um dos campos seja diferente.

A tag *deploy* agrupa as configurações de implantação da rede (*e.g.*, randômica). Ela também utiliza a tag *configuration* para definir o algoritmo de implantação (a tag *name*), as configurações dos nós sensores (tag *conf*) e as configurações globais (tag *var*). Ela também permite que sejam definidas várias configurações do implantação. Adicionalmente, os algoritmos de implantação irão utilizar as informações definidas nas tags *wsn_sizes* e *regions* para criar a RSSF.

A tag *regions* define o conjunto de regiões presentes na RSSF. Cada região é configurada através da tag *region*, a qual necessita saber do número da região (a tag *id*), os nós sensores presentes nela (a tag *nodes*), a localização da região (as tags *x* e *y*) e o tamanho dela (as tags *width* e *heigth*).

Por fim, a tag *stopCriteriaStorage* configura o critério de parada da avaliação (quando a avaliador deve parar se avaliar o consumo de energia da RSSf). Neste caso, a tag do tipo time (type="time") possui duas sub tags para configurar corretamente o critério de parada: a tag *max* para informar o tempo máximo da avaliação e a outra (chamada de *time*) para coletar informações periodicamente sobre a RSSF (e.g., registrar o consumo de energia da RSSF).

Todos as tags são importantes e devem ter, pelo menos, uma configuração, porque a quantidade de configurações de RSSF criadas será igual a quantidade de configurações básicas, vezes a quantidade de procolos utilizados e vezes a quantidade de algoritmos de implantação definidos.

C

Valores do Consumo de Energia dos Operadores e Comandos

Este apêndice mostra o consumo de energia dos operadores da linguagem nesC obtidos através de medição (como foi explicado na Seção 7.1) e que foram usados no Modelo de Aplicação. Esses valores estão associados a plataforma IRIS e, caso o usuário deseja avaliar a aplicação em outra plataforma, é necessário medir e cadastrar os novos valores da plataforma desejada.

Como foi dito no Capítulo 5, o Modelo de Aplicação pode avaliar o consumo de energia de qualquer aplicação (desenvolvida na linguagem nesC) em qualquer plataforma. Este modelo é composto por pequenos modelos reusáveis, chamados de Modelo de Linguagem, o qual representa o consumo de energia de um operador da linguagem nesC. Esses pequenos modelos reusáveis utilizam dados cadastrados para representar o consumo de energia do mesmo operador em diferentes plataformas. Em outras palavras, um pequeno modelo reusável representa um operador X em diferentes plataformas. Ele seleciona o consumo de energia correspondente a plataforma que o usuário deseja avaliar a aplicação.

Os dados cadastrados são o nome do operador, o nome da plataforma, a média e a variância do consumo de energia e do tempo de execução. O nome do operador e o nome da plataforma é utilizado para identificar a qual operador e a qual plataforma pertencem aqueles dados. A média e a variância do consumo de energia e do tempo de execução são necessários para gerar vários randômicos seguindo a distribuição normal. Desta maneira, o Modelo de Aplicação pode estabelecer um intervalo de confiança do consumo de energia da aplicação. Adicionalmente, a distribuição normal foi escolhida por melhor representar a distribuição de energia dos operadores.

As seções seguintes irão apresentar os valores dos operadores de acordo com os

experimentos realizados com o Modelo de Aplicação.

C.1 Primeiro experimento da aplicação

A aplicação *App01* utilizou 4 vezes o operador de soma (+), 1 vez o operador de divisão (/), 1 vez o parentese e 1 vez a atribuição (=). Todos os operadores estão relacionados com o tipo *int_8*. O consumo de energia (em joules) e o tempo de execução (em segundos) de cada operador é ilustrado na Tabela C.1.

Tabela C.1 Consumo de energia e tempo de execução dos operadores utilizados no experimento *App01*.

Informações Básicas		Consumo de Energia		Tempo de Execução	
Tipo	Operador	Média	Variância	Média	Variância
Matemática	/	3,52E-7	5,85E-17	8,83E-6	3,53E-16
Atribuição	=	2,17E-8	1,46E-17	4,28E-7	2,06E-17
Matemática	+	1,18E-8	1,75E-17	2,51E-7	4,89E-17
Parentese	()	1,20E-6	2,12E-15	2,58E-5	1,46E-13

C.2 Segundo experimento da aplicação

A aplicação *App02* utilizou apenas um único comando chamado *send* do módulo *AMSenderC*, o qual implementou a interface *AMSend*. Este comando lança um evento *sendDone* que deve ser implementado pela aplicação. O consumo de energia (em joules) e o tempo de execução (em segundos) do comando *send* é ilustrado na Tabela C.2.

Tabela C.2 Consumo de energia e tempo de execução dos operadores utilizados no experimento *App02*.

Informações Básicas		Consumo de Energia		Tempo de Execução	
Módulo	Comando	Média	Variância	Média	Variância
AMSenderC	send	5.34E-4	3,63E-8	0.0075506667	7,45E-6

C.3 Terceiro experimento da aplicação

A aplicação *App03* utilizou 2 vezes o operador de soma (+), 5 vezes o operador de atribuição (=), 1 vez o de igual (==) e 1 vez o de divisão (/). Todos os operadores estão relacionados com o tipo *int_8*. O consumo de energia (em joules) e o tempo de execução (em segundos) destes operadores é ilustrado na Tabela C.3.

Tabela C.3 Consumo de energia e tempo de execução dos operadores utilizados no experimento *App03*.

Informações Básicas		Consumo de Energia		Tempo de Execução	
Tipo	Operador	Média	Variância	Média	Variância
Matemática	/	3,52E-7	5,85E-17	8,83E-6	3,53E-16
Atribuição	=	2,17E-8	1,46E-17	4,28E-7	2,06E-17
Matemática	+	1,18E-8	1,75E-17	2,51E-7	4,89E-17
Comparação	==	3,63E-8	1,94E-18	6,85E-7	1,29E-18

C.4 Quarto experimento da aplicação

A aplicação *App04* utilizou 1 vez o operador de soma (+), 4 vezes o operador de atribuição (=), 1 vez o operador de menor que (<), 1 vez o de divisão (/) e 1 vez o operador de incremento (++). Todos os operadores estão relacionados com o tipo *int_8*. O consumo de energia (em joules) e o tempo de execução (em segundos) destes operadores é ilustrado na Tabela C.4.

Tabela C.4 Consumo de energia e tempo de execução dos operadores utilizados no experimento *App04*.

Informações Básicas		Consumo de Energia		Tempo de Execução	
Tipo	Operador	Média	Variância	Média	Variância
Matemática	/	3,52E-7	5,85E-17	8,83E-6	3,53E-16
Atribuição	=	2,17E-8	1,46E-17	4,28E-7	2,06E-17
Matemática	+	1,18E-8	1,75E-17	2,51E-7	4,89E-17
Comparação	<	4,23E-8	8,17E-17	7,96E-7	5,01E-16
Incremento	++	3,56E-8	4,41E-17	6,58E-7	1,17E-16

C.5 Quinto experimento da aplicação

A aplicação *App05* utilizou 3 vezes o operador de atribuição (=), 1 vez a função `size()`, 1 vez o comando *getPlayload()* do módulo AMSenderC, o comando `start()` do módulo ActiveMessageC, 2 vezes o comando `startOneShot()` do módulo TimerMilliC, 1 vez o comando `read()` do módulo TempC e 1 vez o comando `send()` do módulo AMSenderC. O operador de atribuição está relacionado com o tipo *float*. O consumo de energia (em joules) e o tempo de execução (em segundos) destes operadores é ilustrado na Tabela C.5.

Além do consumo de energia dos operadores e comando, a aplicação *App05* habilitou

o transmissor, o qual consome energia durante o tempo em que ele estiver ligado. Como o consumo de energia é igual a potência vezes o tempo ($Energia = Potencia * Tempo$) e o tempo é definido pela aplicação, o Modelo de Aplicação necessita do valor da potência para calcular a energia (veja o Apêndice D). Deste modo, nós medimos e obtemos a média da potência do transmissor igual a 0,0419232102 e a variância igual a $4,57E^{-5}$.

Tabela C.5 Consumo de energia e tempo de execução dos operadores utilizados no experimento *App04*.

Informações Básicas		Consumo de Energia		Tempo de Execução	
Tipo/Módulo	Operador/Comando	Média	Variância	Média	Variância
Função	size	2,017E-7	4,77E-16	4,0555E-6	4,65E-17
Atribuição	=	8,27E-8	5,28E-18	1,64E-6	1,61E-18
ActiveMessageC	start	5,69E-5	6,34E-14	5,69E-5	6,34E-14
AMSenderC	send	5,34E-4	3,63E-8	0,0075506667	7,45E-6
AMSenderC	getPlayload	9,89E-7	4,00E-16	1,96E-5	5,86E-16
TimerMilliC	startOneShot	0,00556	7,35E-7	0,248	9,45E-8

D

Funções para Modelo da Aplicação

O Modelo de Aplicação introduzidos na Seçaõ 5.2 usa três funções auxiliares que nós temos definidos para computar o consumo de recursos relacionados: `addEnergy()`, `calcTime()` e `addResourceEnergy()`. As funções *addEnergy()* e *calcTime()* são responsáveis por calcular o consumo de energia e o desempenho (tempo de execução), respectivamente, de um operador modelado. Por outro lado, a função `addResourceEnergy()` é usado para calcular o consumo de energia de um recurso (*e.g.*, radio e LEDs).

O pseudo-código[1] da função *addEnergy()* é apresentado a seguir:

```
1 function addEnergy( real mean , real variance ){
2   var energy = normal( mean , variance );
3   energy\_app = (!energy\_app) + energy;
4   return energy;
5 }
```

Os parâmetros `mean` e `variance` representam a média e a variância de um operador e são usados para calcular um valor randômico seguindo a distribuição normal (linha 2). O valor gerado irá representar o consumo de energia de um operador naquele instante. É importante observar que a função `normal()` é oferecido como biblioteca do CPN Tools. O valor gerado é, então, adicionado a variável que representa o consumo de energia da aplicação (`energy_app`). A função `calcTime()` é similar à função *addEnergy()*, mas ela gera um valor randômico, o qual representa o tempo de execução de um operador.

A infraestrutura proposta suporta várias distribuições de probabilidade, entre elas a Exponencial, Erlang, Polinomial, Normal, entre outras (Trivedi, 2002). No entanto, o sistema considerado nesta tese é bastante estável como pode ser testemunhado através das medidas coletadas e por ser um sistema embarcado no qual cada nó sensor faz principalmente algumas atividades no local sem interferência. Mesmo assim, variações

[1]Devido à complexidade da sintaxe da linguagem CPN ML, nós adotamos um pseudo código para facilitar o entendimento do código da função

no consumo de energia e no tempo de execução ainda existem uma vez que o sistema não é determinístico. Neste estudo, a distribuição de probabilidade normal foi o que melhor se ajustou aos dados medidos.

Finalmente, o pseudo código da função `addResourceEnergy()` é mostrado a seguir:

```
1 function addResourceEnergy ( real time ) {
2   if( radioIsOn ) {
3     var energy = radio\_power * time;
4     energy\_app = (!energy\_app) + energy;
5   }
6   if( led0IsOn ) {
7     var energy = led0\_power * time;
8     energy\_app = (!energy\_app) + energy;
9   }
10  if( led1IsOn ) {
11    var energy = led1\_power * time;
12    energy\_app = (!energy\_app) + energy;
13  }
14  if( led2IsOn ) {
15    var energy = led2\_power * time;
16    energy\_app = (!energy\_app) + energy;
17  }
18 }
```

Esta função calcula o consumo de energia de um recurso (*e.g.*, LEDs) disponível no nó sensor. Neste caso particular, o nó sensor adotado (IRIS) tem 1 radio e 3 LEDs. Cada recurso tem associado uma variável global, indicando se o recurso está ativo (*e.g.*, `radioIsOn`) ou não. Se o recurso estiver ligado, o consumo de energia do recurso é calculado e adicionado a variável que representa o consumo de energia da aplicação (`energy_app`). É importante observar que, como nós não sabemos durante quanto tempo o recurso ficará ligado, esta função está associada em cada transição presente no modelo.

E

Modelos de Propagação

Este apêndice descreve o que é a propagação do sinal, mostra quais são os fatores que interferem e ilustra alguns dos modelos de propagação existem na literatura.

E.1 Introdução

O Modelo de Ambiente é responsável por repassar os pacotes de um nó sensor emissor para os seus vizinhos (nós sensores receptores) (veja a Seção 5.3.3). Para isso, ele utiliza um modelo de propagação para estabelecer o alcance de comunicação de cada nó sensor; em outras palavras, definir até onde as ondas electromagnéticas de um emissor são perceptíveis pelos outros nós sensores.

O estudo de modelos de propagação é uma área de fundamental importância para o projeto de redes sem fio (Rappaport *et al.*, 1996; YACOUB, 1993). Através desse estudo, é possível simular o comportamento da RSSF como se estivesse no ambiente real, permitindo obter informações sobre a conectividade (os enlaces existentes entre os nós sensores) da RSSF. Esse tipo de informação pode ser útil para rever o planejamento da RSSF (*e.g.*, protocolo utilizado, quantidade de nós sensores) visto que a mesma RSSF (a mesma quantidade de nós sensores com as mesmas configurações e nas mesmas posições) usando modelos de propagações de sinais diferentes poderá ter uma conectividade diferente entre os nós sensores; e, consequentemente, poderá ter um consumo de energia e uma confiabilidade diferentes.

Por exemplo, a Figura E.1 mostrar uma RSSF (a) formada por um nó sorvedouro (no centro) e doze nós sensores envolta do nó sorvedouro, usando dois modelos de propagação de sinal diferentes. No primeiro modelo de propagação (b), o nó sorvedouro (no centro) possui conectividade com seis nós sensores por estarem dentro do raio de propagação do sinal. Enquanto, no segundo modelo (c), o nó sorvedouro pode possuir (visto que

os nós sensores não estão completamente dentro do sinal de propagação) conectividade com, no máximo, 4 nós sensores. Observando isso, o primeiro modelo poderá ter uma conectividade melhor entre o nó sorvedouro e os nós sensores do que o segundo modelo. No entanto, o segundo modelo pode ser mais realista do que o primeiro.

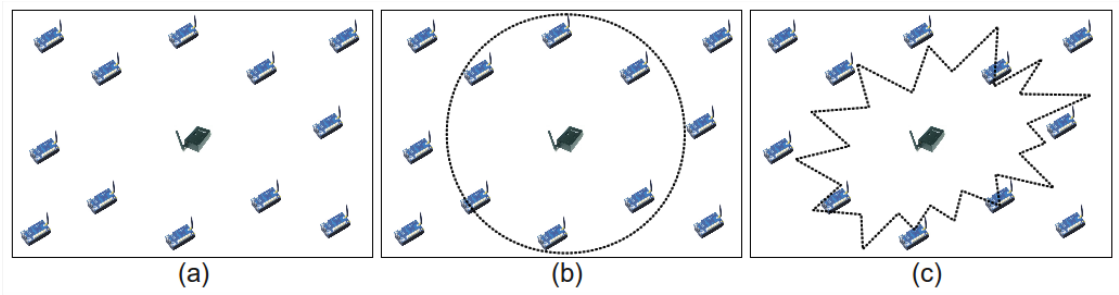


Figura E.1 A mesma RSSF (a) usando dois modelos de propagação de sinal fictícios: (b) Modelo Circular e (c) Modelo Estrela.

Os principais fatores que causam atenuação (perda ou ganho) de sinal são a distância entre o emissor e o receptor, os obstáculos entre eles e o ambiente (*e.g.*, as condições atmosféricas, o tipo de ambiente, entre outros) onde a RSSF está instalada. A distância influência diretamente na potência do sinal visto que quanto mais distante for o receptor, menor será a potência recebida do sinal do transmissor. Essa característica é conhecida como *Path Loss*. Além disso, dependendo do ambiente, pode haver vários obstáculos entre o transmissor e o receptor que podem aumentar (construtiva) ou diminuir (destrutiva) a qualidade do sinal recebido, dependendo do material como é mostrado por Damosso and Correia (1999). No pior caso, quando existe um objeto impermeáveis à penetração de ondas electromagnéticas, pode haver área sem comunicação com os demais integrantes da rede. Essa característica é denominada efeito sombra (*shadowing*) (Jacinto, 2012). Por fim, o ambiente pode interferir também na potência no sinal por causa da irregularidade do terreno, por ser *indoor* ou *outdoor*, se está chovendo ou não (as condições do ambiente). Adicionalmente, ambientes *indoor* tendem ter mais objetos para interferir na potência do sinal do que o *outdoor* (Jacinto, 2012).

Os modelos de propagação devem, ou não, considerar essas características para representar um ambiente, onde a RSSF será implantada. Alguns modelos são apresentados a seguir.

E.2 Modelos

Os modelos de propagação são representações matemáticas teóricas (equações elaborados a partir da observação de um determinado ambiente) e/ou empíricas (equações elaborados a partir de resultados obtidos através de experimentos em um determinado ambiente) da propagação das ondas electromagnéticas. Estes modelos servem para simular o comportamento da propagação do sinal (ondas electromagnéticas) em um determinado ambiente.

A atenuação do sinal pode ser em grande escala (a variação média) ou em pequena escala (variações rápidas). Os modelos podem considerar a variação média (Grande Escala) quando a distância entre o emissor e o recepção é grande (*e.g.*, centenas ou milhares de metros). Ou, então, as variações rápidas (Pequena Escala) ocorridas entre distâncias muito curtas ou entre tempos muito curtos. Esse comportamento de pequena escala está relacionada a chegada do mesmo sinal em diferentes instantes no receptor. Devido à esse comportamento aleatório, esta componente pode ser representada com uma distribuição estatística de Rice ou Rayleigh (Jacinto, 2012).

A seguir, serão apresentados uma breve descrição de alguns modelos de propagação de grande escala utilizados na RSSF.

Simples

O modelo, denominado aqui como simples, considera apenas dois fatores: o raio de comunicação do emissor e a distância entre o emissor e o receptor. Por exemplo, o IRIS MEMSIC (2014a) pode ter uma comunicação entre 50 m (*indoor*) até uns 300 m (*outdoor*). O receptor irá receber os dados do emissor se a distância entre o emissor e o receptor for igual ou menor do que o raio de comunicação do emissor. Caso contrário, não existem a comunicação entre eles. Por exemplo, se o raio de alcance de comunicação do emissor for igual a 50 m e a distância entre eles for de 30 m, o receptor irá receber todos os dados do emissor. Caso a distância entre eles mude para 60 m, o receptor deixará de receber os dados.

Para descobrir a distância entre dois pontos (A e B) quando se usa duas dimensões (no plano cartesiano), basta aplicar o teorema de Pitágoras apresentado a seguir:

$Distancia_{AB} = \sqrt{(X_B - X_A)^2 + (Y_B - Y_A)^2}$

onde X_i e Y_i representam a coordenada no eixo X e no eixo Y, respectivamente, do ponto i (por exemplo, X_B e Y_B representam as coordenadas do ponto B no eixo X e Y, respectivamente).

Mesmo esse modelo não representado a realidade de um ambiente (porque o raio de comunicação do emissor é perfeito em todos os sentidos), ele é utilizado na literatura por sua simplicidade, por exemplo, para validar protocolos de roteamento Senouci *et al.* (2012). Por este motivo, este modelo foi utilizado no Modelo de Rede.

Modelo de Livre Espaço

O modelo de Livre Espaço (Friis, 1946) é um dos modelos teórico mais simples por não considerar obstáculos entre o emissor e o receptor. Ele representa o cenário ideal e é muito utilizado para os sistemas de comunicação de satélites; no entanto, (dificilmente) uma RSSF estará em um ambiente nessas condições (sem obstrução entre o receptor e o emissor). A potência do sinal recebida é dado em função da distância entre esses dois pontos, do ganho das antenas e da potência de emissão. Mas, primeiro, deve-se utilizar o seguinte equação para calcular a atenuação da propagação no Espaço Livre (em dB):

$L_{dB} = 10log(\frac{4\Pi d}{\lambda})^2$

onde d representa a distância entre o emissor e o receptor e λ representa o comprimento de onda. Após calcular a atenuação do sinal, é possível calcular a potência do sinal recebida pelo receptor, como mostrar a seguir:

$P_{RX} = P_{TX} + G_T + G_R - L_{dB}$

onde P_{TX} é a potência de transmissão do emissor, G_T e G_R são os ganhos das antenas do emissor e do receptor em dBm (respectivamente), e (como foi mostrado anteriormente) L_{dB} representa a atenuação do sinal no espaço livre. Para que haja comunicação entre o emissor e o receptor, a potência recebida (P_{RX}) pelo receptor deve ser maior ou igual ao nível de sensibilidade de sua antena. Por exemplo, para existir a comunicação entre dois IRIS, é necessário que a potência recebida (P_{RX}) pelo receptor seja maior ou igual a -101 dBm; nível de sensibilidade do IRIS de acordo com o seu *datasheet* (MEMSIC, 2014a).

Adicionalmente, este calculo pode ser utilizado juntamente com os outros modelos de propagação de sinal (sendo necessário apenas descobrir a atenuação do sinal). Por tal motivo, este calculo não será mais apresentado.

Modelo de 2 raios

Os modelos baseados no traçado dos raios analisam a atenuação do sinal de acordo com a reflexão da radiação de ondas electromagnéticas. Quanto mais objetos refletores existem no ambiente, mais complexo será o calculo e mais tempo levará para ser calculada. Por tal motivo, é comum simplificar os modelos baseados no traçado dos raios para considerar um número limite de raios refletidos (por exemplo, considerar apenas 5 ou 6 raios); como

é o caso do modelo de 2, de 4, de 6 e de 10 raios. Por ser mais simples, essa seção descreverá o modelo de 2 raios.

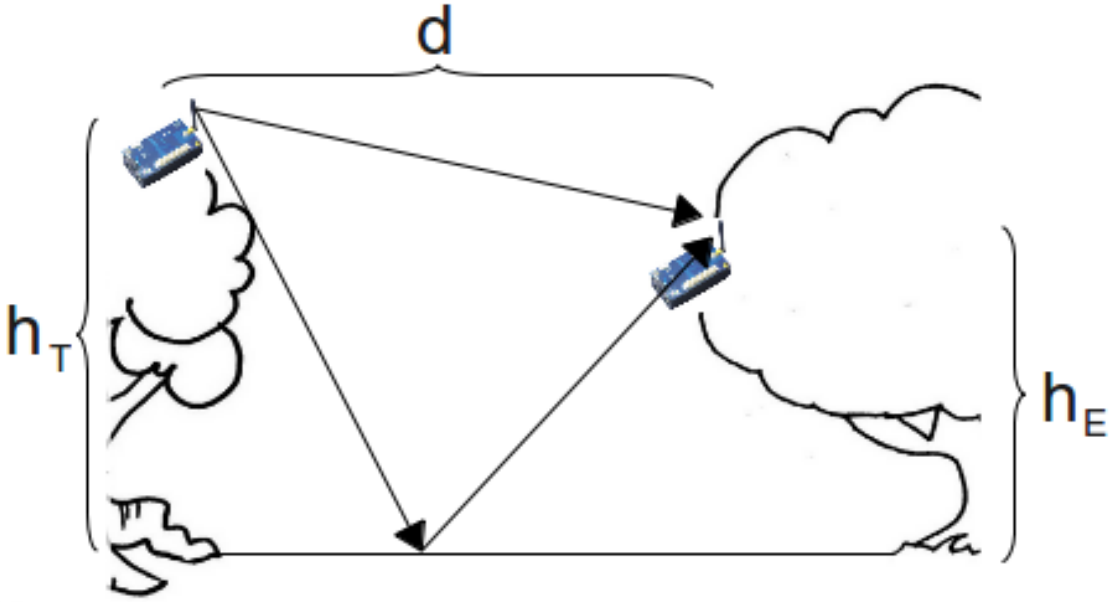


Figura E.2 Ilustração dos 2 raios.

O modelo de 2 raios (Rappaport *et al.*, 1996) considerar apenas a reflexão do solo. Esse modelo é ideal para representar ambientes abertos sem paredes (obstáculos laterais) e sem tetos (obstáculos superiores) para refletir os raios (veja a Figura E.2), tal como uma fazenda. De maneira resumida, a atenuação do sinal (em dB) no modelo de 2 raios pode ser calculado da seguinte forma (Najnudel, 2004; Mattos, 2006):

$L = 40log(d) - 20log(h_T) - 20log(h_R) - G_T - G_R$

onde d é distância entre o emissor e o receptor (em metros); h_T e h_R são a altura do emissor e do receptor (em metros), respectivamente; e, por fim, G_T e G_R representam os ganhos das antenas do emissor e do receptor (em dBi), respectivamente.

Modelo *Log-Distance*

O modelo *Log-Distance* (Rappaport *et al.*, 1996) é um dos mais simples modelos empíricos existentes por se basear em uma distância de referência (denominada L_0). Ele permite calcular a atenuação do sinal para ambientes abertos ou fechados, como é mostrado a seguir:

$L = 10log(\frac{4\Pi d_0}{\lambda})^2 + 10\beta log(\frac{d}{d_0})$

onde L_0 representa o ponto de referências (como foi dito anteriormente), o qual pode assumir o valor igual a 1 ou 100 para representar ambientes internos ou externos,

respectivamente (Ferreia, 2009); e β representa a taxa de diminuição da potência com a distância em um determinado ambiente. Esta variável pode assumir os valores 2 (quando representa um espaço livre), 2,7 a 3,5 (quando representa áreas urbanas), ou 3 a 5 (quando representada áreas urbanas sombreadas) (Ferreia, 2009).

Modelo *Log-Normal Shadowing*

O modelo *Log-Normal Shadowing* (Rappaport *et al.*, 1996) é uma variante do modelo anterior (*Log-Distance*) e não considerar que a comunicação seja perfeita em todos as direções (ou seja, que o modelo de propagação seja um circulo perfeito). Ele considera que existe um grau de incerteza na propagação do sinal e no ambiente de propagação (Ferreia, 2009). Para isso, ele acrescenta uma variável aleatória na equação (chamado de componente de *Shadowing*), como é mostrado a seguir:

$L = 10log(\frac{4\Pi d_0}{\lambda})^2 + 10\beta log(\frac{d}{d_0}) + X_\omega$

onde X_ω é uma uma variável aleatório seguindo a distribuição Gaussiana (em dB) com média nula (em dB) e com um desvio padrão ω (em dB). Os valores de β e ω são definidos por meio de experimentos / medições efetuadas (Jacinto, 2012).

Conclusão

Este apêndice apresentou alguns modelos de propagação que podem ser usados pelo Modelo de Ambiente (veja a seção 5.3.3) para determinar o alcance de comunicação de um nó sensor. Foram apresentados cinco modelos: (I) simples, o qual determinar um raio de alcance de comunicação do emissor; (II) livre espaço, que calcula a potência recebida pelo receptor considerando a distância, o ganho das antenas e a potência de transmissão; (III) 2 raios, que considera a reflexão do raio no solo; (IV) *Log-Distance*, modelo empírico que considera uma propagação diferente em meios diferentes; e, por fim, (V) *Log-Normal Shadowing*, uma variante do modelo anterior que considera a irregularidade na propagação do sinal.

Existem vários outros que não foram apresentados, mas que são usados na literatura. Por exemplo, Modelo RIM (*Radio Irregularity Model*) (Zhou *et al.*, 2006), desenvolvido especificamente para a RSSF, considera a irregularidade na propagação do sinal (tal como o modelo de *Log-Normal Shadowing*).

F

Mapa de Energia

Este apêndice foi criado para mostrar o mapa de energia da RSSF quando utiliza um determinado protocolo. Mais especificamente, este apêndice mostra o consumo de energia da RSSF no primeiro experimento da seção 7.5. Além dos quatro protocolos utilizado neste experimento, este apêndice mostra um protocolo extra não utilizado no experimento apenas para mostrar um problema que ocorre na RSSF quando ele (um protocolo similar) é utilizado.

F.1 Protocolo FLOODING e protocolo GOSSIPING

Os protocolos FLOODING e GOSSPING tem o pior consumo de energia entre outros protocolos aqui listados, como foi mostrado na Seção 7.5. O protocolo FLOODING envia um pacote a todos os nós sensores vizinhos, os quais enviam para os seus vizinhos, e assim sucessivamente até chegar no remetente. O protocolo GOSSIPING escolhe apenas um único vizinho aleatoriamente, o qual faz o mesmo processo até enviar ao remetente. Por essas características, ambos os protocolos consomem muita energia e de forma desnecessária. Entre eles, oprotocolo FLOODING consome mais energia do que o protocolo GOSSIPING.

A Figura F.1 mostra o mapa de energia da RSSF em três momentos distintos para os dois protocolos. Cada momento representa um tempo da avaliação: o primeiro, quando a avaliação foi iniciada; o segundo, quando a avaliação está na metade (ou próximo); e o terceiro, quando a avaliação está perto de terminar. Em todos os três momentos, todos os nós sensores estão (praticamente) com o mesmo nível de energia. Isso ocorre porque todos os nós sensores participam do roteamento do dado ativamente de (praticamente) de todos os nós sensores da RSSF. Por este motivo, os nós sensores tendem a morrem juntos e rápidos.

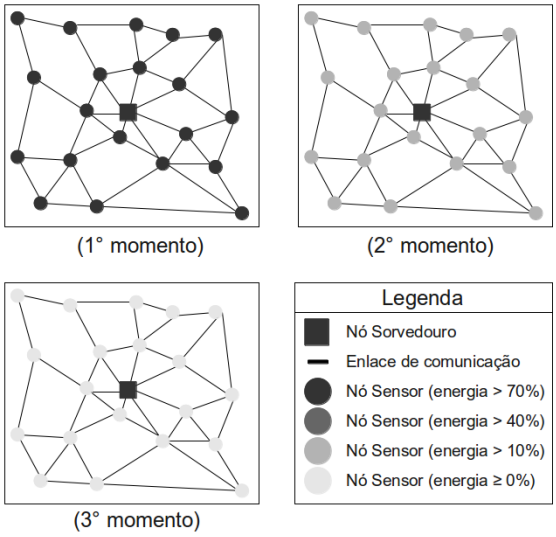


Figura F.1 Mapa de energia da RSSF quando está usando os protocolos FLOODING e GOSSIPING.

F.2 Protocolo LEACH

O protocolo LEACH divide a RSSF em grupos, chamados de *clusters*. Os *clusters* possui um líder e vários participantes, os quais devem enviar os dados para os líder do *cluster* que, por sua vez, envia para o nó sorvedouro. Novos líderes são eleitos em tempos em tempos para equilibra o consumo de energia na RSSF. Quando um líder é eleito, os nós sensores decidem qual *cluster* desejam participar. Assim que os nós sensores decidem, eles ajustam o raio de comunicação do transmissor. A Figura F.2 mostra diferentes líderes de *clusters* em três momentos da avaliação da RSSF.

Os momentos segue os mesmo princípios da figura anterior: o primeiro, quando a avaliação foi iniciada; o segundo, quando a avaliação está na metade (ou próximo); e o terceiro, quando a avaliação está perto de terminar. Algumas características do LEACH interferem no consumo de energia da RSSF. Por exemplo, o protocolo LEACH não garante que existam a mesma quantidade de *clusters* na RSSF. Em outras palavras, o número de líderes de *clusters* pode vaiar ao longo do tempo; por exemplo, no primeiro momento, a RSSF possui quatro líderes de *clusters*, enquanto no segundo momento, existam três líderes de *clusters*. Quando os líderes de *cluster* são eleitos, os nós sensores podem escolher qual *cluster* participar. Eles escolhem o líder de *cluster* mais próximos para economizar energia. No entanto, o líder de *cluster* irá consumir mais energia de acordo com a quantidade de participantes do seu *cluster*.

Como existe a rotatividade de líder de *cluster*, os nós sensores equilibram o consumo

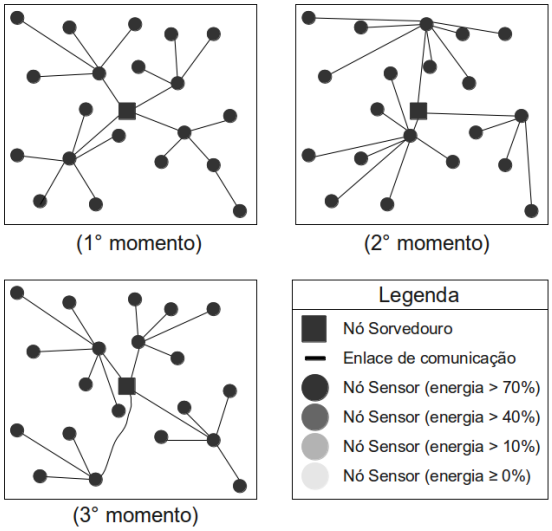


Figura F.2 Mapa de energia da RSSF quando está usando o protocolo LEACH.

de energia da RSSF. No entanto, os nós sensores acabam ficam sem energia ao mesmo tempo.

F.3 Protocolo DIRECT

O protocolo DIRECT envia diretamente os dados para o nó sorvedouro. Para economizar energia, os nós sensores ajustam o raio de comunicação do transmissor para o menor valor necessário para haver a comunicação com o nó sorvedouro. Por exemplo, se a distância entre o nó sensor e o nó sorvedouro for igual a 200 metros, o raio de comunicação do transmissor deve ser ajustado para um valor que seja suficiente para transmitir um dado a 200 metros.

A Figura F.3 mostra o mapa de energia em três momentos da RSSF quando utiliza o protocolo DIRECT. Como é possível perceber, os nós sensores mais distantes tendem a ficar sem energia mais rápido do que os nós sensores mais próximos devido ao ajuste do raio de comunicação do transmissor. Quanto mais distante o nó sensor está do nó sorvedouro, o consumo de energia dele será maior.

F.4 Protocolo Extra

Outros protocolos utilizados na RSSF são os de *short paths*, o qual encontra o menor caminho entre o remetente e o destinatário (geralmente, o nó sorvedouro). Os nós sensores

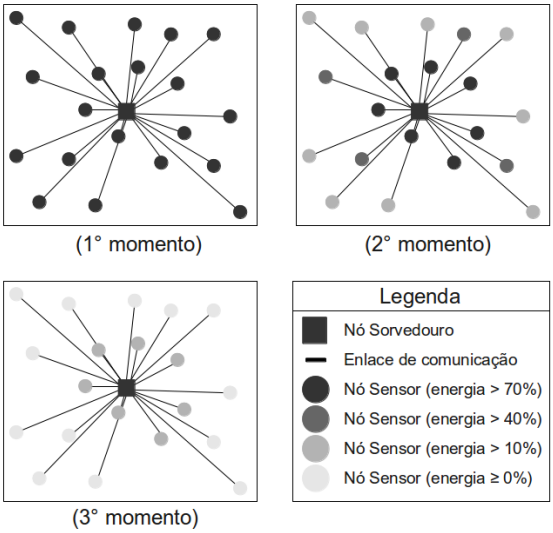


Figura F.3 Mapa de energia da RSSF quando está usando o protocolo DIRECT.

auxiliam a rotear os dados entre si, diminuindo o consumo de energia da RSSF. Os nós sensores mais próximos dos nó sorvedouro tendem a ser mais utilizado e, por este motivo, eles tendem a ficar sem energia mais cedo. Tal problema é ilustrado na Figura F.4.

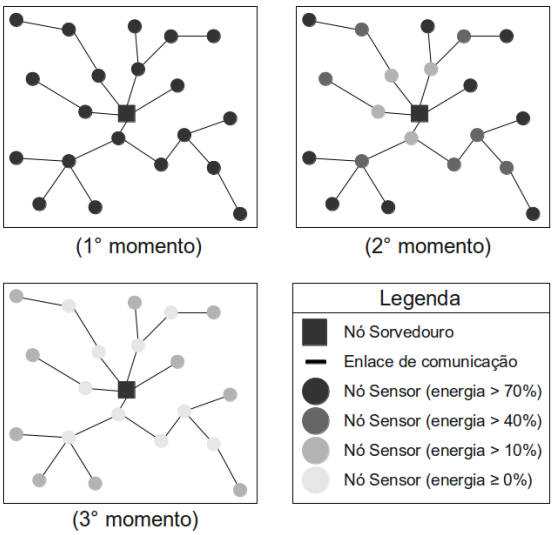


Figura F.4 Mapa de energia da RSSF quando está usando o protocolo DIRECT.