UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

ÉLIO RIBEIRO DE BRITO FILHO

Aplicando a Técnica de Simulação *Hardware-In-the-Loop* no Desenvolvimento de um Simulador Híbrido para *Testbeds* de Redes de Sensores Sem Fio

Goiânia
2014

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Aplicando a Técnica de Simulação *Hardware-In-the-Loop* no Desenvolvimento de um Simulador Híbrido para *Testbeds* de Redes de Sensores Sem Fio

Autor(a): Élio Ribeiro de Brito Filho

Goiânia, 28 de Agosto de 2014.

Élio Ribeiro de Brito Filho – Autor

Dr. Renato de Freitas Bulcão Neto – Orientador

Dr. Iwens Gervasio Sene Junior – Co-Orientador

ÉLIO RIBEIRO DE BRITO FILHO

Aplicando a Técnica de Simulação *Hardware-In-the-Loop* no Desenvolvimento de um Simulador Híbrido para *Testbeds* de Redes de Sensores Sem Fio

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Renato de Freitas Bulcão Neto

Co-Orientador: Prof. Dr. Iwens Gervasio Sene Junior

Goiânia
2014

ÉLIO RIBEIRO DE BRITO FILHO

Aplicando a Técnica de Simulação *Hardware-In-the-Loop* no Desenvolvimento de um Simulador Híbrido para *Testbeds* de Redes de Sensores Sem Fio

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Computação, aprovada em 28 de Agosto de 2014, pela Banca Examinadora constituída pelos professores:

Prof. Dr. Renato de Freitas Bulcão Neto
Instituto de Informática – UFG
Presidente da Banca

Prof. Dr. Iwens Gervasio Sene Junior
Instituto de Informática – UFG

Prof. Dr. Bruno Oliveira Silvestre
Instituto de Informática – UFG

Profa. Dra. Silvana Rossetto
Departamento de Ciência da Computação – UFRJ

Élio Ribeiro de Brito Filho

Graduou–se em Engenharia da Computação, com ênfase em Administração e Gerenciamento de Redes de Computadores, pela Pontifícia Universidade Católica de Goiás (PUC-GO). Durante sua graduação, foi monitor bolsista no departamento de Computação da PUC-GO. Atualmente é mestrando bolsista da CAPES/REUNI do programa de Mestrado em Ciência da Computação, da Universidade Federal de Goiás (UFG), atuando no desenvolvimento de pesquisas direcionadas a integração entre diferentes ambientes de testes para as Redes de Sensores Sem Fio.

Dedico este trabalho ao meus adoráveis pais Élio Ribeiro de Brito e Dulce Fernandes Cotrim de Brito, e a minha amável esposa Daniele de Oliveira Prates.

Agradecimentos

Primeiramente, agradeço à Deus, pela vida e oportunidade, bem como pela manutenção de minhas forças, possibilitando assim a conclusão deste trabalho. Aos meus queridos pais, por todo amor e cumplicidade na composição dos pilares que regem a minha vida. Aos meus irmão e sobrinhos, por todo amor presente. À minha amável e companheira esposa, por toda compreensão, incentivo e suporte emocional nas horas mais difíceis. Aos meus amigos, por contribuírem com a manutenção de uma vida repleta de felicidades. Ao Prof. Dr. Bruno Oliveira Silvestre, pela paciência e compreensão, e por estar sempre presente, contribuindo de maneira significativa para a conclusão deste trabalho. Ao meu orientador Prof. Dr. Renato de Freitas Bulcão e co-orientador Prof. Dr. Iwens Gervasio Sene Junior, pela confiança e importante colaboração. Ao Grupo de Pesquisa em Computação Ubíqua, por todo enriquecimento acadêmico/científico proporcionado nas reuniões. Aos colegas Guilherme Salazar Silva e Roberto Vito Rodrigues Filho, pelo suporte com a inicialização do aprendizado das ferramentas TinyOS e MSPSim, respectivamente. Ao Prof. Dr. José Luiz de Freitas Júnior (CMD/PUC-GO) e a Profa. Msc. Angélica da Silva Nunes (CMD/PUC-GO), pela confiança depositada. Ao Instituto de Informática (INF) e à Universidade Federal de Goiás (UFG), pela oportunidade e apoio estrutural. À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), que através do Programa de Apoio a Planos de Reestruturação e Expansão das Universidades Federais (REUNI), financiou todo este trabalho.

"Não posso dar a nenhum cientista de qualquer idade melhor conselho do que este: a intensidade da convicção de que uma hipótese é verdadeira não tem nenhuma relação com se é ou não verdadeira."

Peter Medawar, 1979,
Advice to a Young Scientist.

Resumo

Brito Filho, Élio R. de. **Aplicando a Técnica de Simulação *Hardware-In-the-Loop* no Desenvolvimento de um Simulador Híbrido para *Testbeds* de Redes de Sensores Sem Fio**. Goiânia, 2014. 85p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Este trabalho apresenta uma alternativa para escalar *testbeds* de redes de sensores sem fio. A abordagem utilizada baseou-se na recorrência da técnica de simulação *Hardware-In-the-Loop* para co-relacionar nós sensores reais com nós sensores virtuais. Os fundamentos constitutivos da pesquisa foram viabilizados a partir de uma revisão literária de caráter exploratório, que possibilitou a definição de predicados para o direcionamento de uma solução funcional. O delineamento da abordagem utilizada foi materializado através da implementação de um simulador híbrido escalável e de alta-fidelidade. Para dar consistência ao empreendimento desenvolvido, foi aplicada uma série de testes/*benchmarks*, que possibilitou avaliar a integridade, o custo operacional e as limitações presentes na infraestrutura. As avaliações aplicadas confirmaram o cumprimento dos objetivos traçados e a identificação de algumas limitações relacionadas ao *hardware*.

Palavras–chave

Redes de Sensores Sem Fio, Simulador híbrido, *Hardware-In-the-Loop*

Abstract

Brito Filho, Élio R. de. **Applying Technique Simulation Hardware-In-the-Loop on Development of a Hybrid Simulator for Testbeds of Wireless Sensor Networks**. Goiânia, 2014. 85p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

This paper presents an alternative to scale testbeds for wireless sensor networks. The approach in this research was based on the recurrence of Hardware-In-the-Loop simulation technique to co-relate real sensor nodes with virtual sensor nodes. A literature review of explanatory feature enabled the fundamentals of this research and allowed the definition of predicates for targeting a functional solution. The design of the approach used was materialized through the implementation of a scalable and high-fidelity hybrid simulator. A series of tests/benchmarks took effect in order to legitimate the current study, that allowed us to evaluate the integrity, operating cost and limitations present in the infrastructure. The applied assessment confirmed the compliance of the outlined goals and the identification of some limitations related to the hardware.

Keywords

Wireless Sensor Networks, Hybrid simulator, Hardware-In-the-Loop

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Códigos de Programas

CAPÍTULO 1

Introdução

Considerada como uma vertente do paradigma Internet das Coisas (IoT), as Redes de Sensores Sem Fio (RSSF) são coleções de pequenos dispositivos autônomos – conhecidos como nós sensores ou *motes* – cuja capacidade de sensoriamento, processamento e transmissão cooperativa de informação, possibilitam que diferentes fenômenos físicos e ambientais sejam monitorados, mesmo em condições inóspitas e adversas.

Devido a sua natureza *hardware*-restritiva caracterizada por suas limitações energéticas, baixo poder de processamento e pouca capacidade de memória [12], os nós sensores não manipulam grandes quantidades de dados, sendo basicamente responsáveis pela digitalização do fenômeno monitorado e encaminhamento das informações ao observador. O observador, que pode está conectado via Internet, satélite ou outro sistema, então colhe as informações através de um ponto de escoamento na rede (representado pelo nó *sink*[1]) e as analisam e/ou armazenam. Todo esse processo (fenômeno $\rightarrow$ sensor $\rightarrow$ observador) pode decorrer de forma constante, periódica ou escalonada, conforme especificações do usuário ou projeto da rede.

Para determinar o gerenciamento das ações de controle de um nó sensor, aplicações são desenvolvidas sobre a plataforma de um sistema operacional de propósito específico (*e.g.*, TinyOS [44], Contiki [27], MANTIS [8], Nano-RK [33], LiteOS [11]) e embarcadas em cada nó necessário. Entretanto, comum ao processo de desenvolvimento de *software*, essas aplicações devem ser avaliadas antes da implantação final. Segundo Li *et al.* [46] e Sundani *et al.* [66], o procedimento de avaliação de *software* é um dos principais desafios na pesquisa e desenvolvimento de redes de sensores, inviabilizado, inclusive, devido à falta de uma infraestrutura física própria, seja por ser difícil, inviável ou impossível a aquisição do *hardware*.

Como recurso à ausência de um ambiente de teste particular, algumas alternativas foram elaboradas e implementadas ao longo do tempo, são elas: os simuladores/emuladores e os *testbeds*. Os simuladores/emuladores são modelos computacionais

[1] O nó *sink* é qualquer dispositivo que seja utilizado como ponto de escoamento de informações da rede de sensores.

capazes de reproduzir as principais características e comportamentos de uma rede de sensores, empregando formalizações em um ambiente controlado, de baixo custo e passível de repetição. Já os *testbeds*, são ambientes desenvolvidos exclusivamente para testes, que em sua maioria são compostos apenas de nós sensores físicos, mas com alguns exemplos de integração entre diferentes tipos de nós (Figura 1.1), como é o caso do WISEBED [15] e Kansei [6].

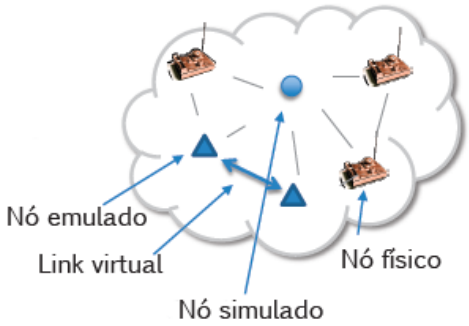


Figura 1.1: *Representação da abordagem de integração entre nós físicos, simulados e emulados no testbed WISEBED.*

1.1 Motivação

Considerando a possibilidade de ofertar serviços de informação e comunicação para solucionar problemas da população em suas cidades, o Ministério da Ciência, Tecnologia e Inovação (MCTI), através da Rede Nacional de Ensino e Pesquisa (RNP), financiou os projetos Construindo Cidades Inteligentes ([CIA]^2)[2] e Testbed para Espaços Inteligentes (GT-TeI), que dentre outras metas, viabilizaram o desenvolvimento de um *testbed* para servir como uma infraestrutura de acesso público para testes preliminares com *softwares* de redes de sensores sem fio.

O *testbed* do projeto [CIA]^2/GT-TeI foi idealizado para contar com uma arquitetura composta por 40 unidades de nós sensores, sendo que 30 são *motes* do tipo MICAz [19] e 10 são *motes* do tipo TelosB [20]. Apesar da quantidade de nós sensores ser equiparável a alguns *testbeds*, como WSNTB [59] e SignetLab [16], o *testbed* do projeto [CIA]^2/GT-TeI, assim como o WSNTB e o SignetLab, apresentam deficiências em relação aos *testbeds* IoT-LAB [34], Kansei [6], WISEBED [15] e Trio [28] quanto a possibilidade de se testar aplicações de nós sensores para projetos de redes de grande escala. Em um comparativo entre os principais *testbeds* para redes de sensores sem fio, a partir da análise da Figura 1.2, podemos constatar o quão é limitada, por exemplo, a arquitetura do *testbed* do projeto [CIA]^2/GT-TeI, quanto ao número de nós sensores disponíveis.

[2] `http://www.nr2.ufpr.br/~cia2`

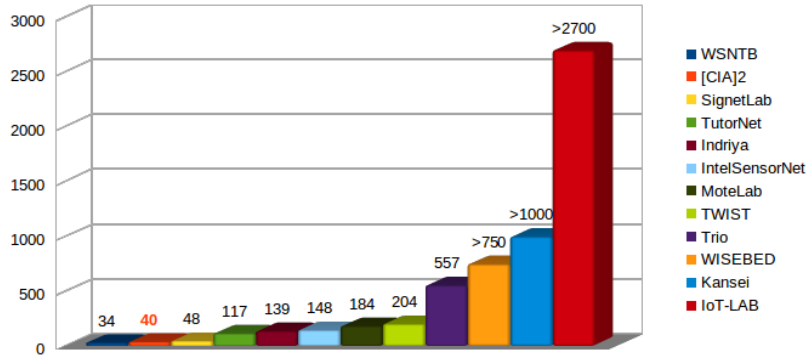


Figura 1.2: *Quantidade de nós sensores em diferentes testbeds.*

Assim, considerando que muitos projetos de redes de sensores sem fio contam com uma infraestrutura de centenas ou até milhares de nós sensores, torna-se difícil para os pesquisadores e desenvolvedores encontrar um *testbed* que viabilize a verificação do comportamento da rede e de seus componentes antes da implementação no mundo real. Entretanto, a impossibilidade de se avaliar protocolos e aplicações em um ambiente de teste com quantidade de nós que divergem do projeto da rede alvo, pode resultar em um considerável aumento de custo temporal do projeto.

1.2 Objetivo

Fundamentado na possibilidade de obter uma alternativa para estender *testbeds* físicos de redes de sensores sem fio a partir da integração com nós sensores virtuais, e alimentado pela motivação em contribuir com o *testbed* do projeto [CIA]^2/GT-TeI, estamos propondo neste trabalho a elaboração de um estudo capaz de identificar uma maneira de integrar os ambientes físicos e virtuais, obedecendo aos seguintes predicados:

1. Um ambiente híbrido totalmente transparente: de modo a comportar-se tanto quanto possível como um *testbed* puramente físico, objetivando minimizar a sobrecarga de transferência do sistema experimental para o sistema final implantado no mundo real.
2. Um ambiente virtual com capacidade de fornecer execuções de granularidade fina: favorecendo a execução de tarefas com um alto grau de fidelidade de processamento.
3. Um sistema capaz de manter total integridade dos dados trafegados: garantindo ao usuário a disponibilidade de informações confiáveis em conformidade com as experimentações reais.

A partir dos pré-requisitos definidos para a nossa proposta, abordamos a utilização do conceito de simulação *Hardware-In-the-Loop* (HIL) para desenvolver um simula-

dor híbrido que fosse capaz de estender um nó sensor físico em algumas unidades virtuais, tratadas como alternativa a aquisição de mais nós sensores físicos para os *testbeds*.

A técnica de simulação HIL consiste da utilização de simuladores interpolando funções e eventos de dispositivos físicos, compondo um sistema híbrido físico-simulado. Com a utilização dessa técnica, pudemos a partir da recorrência de simuladores, modelar nós sensores híbridos capazes de abstrair as funcionalidades dos transceptores físicos para os nós sensores simulados. A expectativa é que a partir do acréscimo no número de nós sensores, seja possível compor cenários que favoreçam testes com aplicações para implementações em maiores escalas.

1.3 Metodologia

As etapas constitutivas do projeto abordaram a revisão bibliográfica acerca de temas como: as Redes de Sensores Sem Fio (RSSF) e seus componentes estruturais; o protocolo IEEE 802.15.4 de especificação da camada física e acesso ao meio; os ambientes de testes para RSSF e a classificação de simuladores/emuladores conforme níveis de abstração de um sistema computacional convencional; e trabalhos relacionados à técnica de simulação *Hardware-In-the-Loop* aplicada a ambientes de testes para RSSF.

Os elementos constituintes da proposta apresentada foram especificados a partir de um minucioso levantamento de requisitos, que contou com a definição: de uma ferramenta de simulação capaz de emular os principais componentes do nó sensor; do *hardware* necessário para embarcar as aplicações que possibilita a integração entre os ambientes físico e virtual; e da infraestrutura do *front-end* de interfaceamento dos componentes e gerência de cada nó sensor simulado.

O *software* do dispositivo de integração físico-simulado ficou a cargo da codificação em linguagem de programação nesC, utilizando componentes do sistema operacional de propósito especifico TinyOS[3]. Já o desenvolvimento das demais aplicações foi baseado em linguagem de programação Java.

Os testes realizados sobre o simulador híbrido considerou a avaliação da integridade dos dados e da capacidade operacional da ferramenta. As limitações apresentadas no *hardware* utilizado foram demonstradas através da análise de leitura observadas. Os *benchmarks* utilizados durante a bateria de testes e análise de dados foram desenvolvidos sobre as plataformas nesC, Java, C e ShellScript.

[3] `http://www.tinyos.net`

1.4 Organização do Trabalho

Neste Capítulo 1 foram apresentadas definições de redes de sensores sem fio e da técnica de integração *Hardware-In-the-Loop*, objetivando criar elementos conceituais que servissem de base para uma melhor compreensão da motivação adotada e dos objetivos traçados para este trabalho.

No Capítulo 2 é abordada uma visão mais detalhada sobre as redes de sensores sem fio, cujo objetivo é servir de plano de fundo para as definições e classificações dos principais sistemas constituídos nos projetos de redes de sensores.

No Capítulo 3 são apresentados os tipos e características dos principais ambientes de testes para redes de sensores, enfatizando os simuladores em uma classificação baseada na abordagem *top-down* em que os sistemas computacionais são organizados. Esse levantamento direcionou a definição da ferramenta de simulação mais adequada para o objetivo deste trabalho.

No Capítulo 4 é apresentada a descrição de toda a arquitetura que compõe o conjunto de ferramentas desenvolvidas para demonstrar a aplicação da técnica de simulação *Hardware-in-The-Loop* em um ambiente físico de testes para redes de sensores sem fio.

No Capítulo 5 é abordada a análise dos resultados obtidos através da execução de testes/*benchmarks*, assim como as ferramentas e técnicas utilizadas. O objetivo é avaliar o comportamento da infraestrutura de pesquisa, procurando dar consistência ao empreendimento.

No Capítulo 6 é apresentada uma conclusão da pesquisa realizada, assim como um confrontamento com trabalhos realizados, e sugestões que direcione possíveis pesquisas futuras.

CAPÍTULO 2

Redes de Sensores Sem Fio (RSSF)

O avanço das tecnologias de telecomunicação, semicondutores e ciências dos materiais contribuiu de forma significativa com o surgimento das Redes de Sensores Sem Fio (RSSF), experimentando grande crescimento nas áreas de integração em grande escala (VLSI), sistemas microeletromecânicos (MEMS) e comunicação sem fio [22].

Com a tecnologia VLSI (*Very-Large-Scale Integration*) tornou-se possível controlar a densidade com que os componentes (transistores e outros elementos) são condensados em um Circuito Integrado (CI), dando origem, inclusive, aos atuais microcontroladores. Já com os MEMS (*Micro-Electro-Mechanical Systems*) foi possível capacitar a integração de elementos mecânicos e eletrônicos em uma pastilha comum de silício, fornecendo soluções compactas e de baixo custo, como sensores e atuadores.

Distinta das tradicionais redes sem fio, as RSSF são caracterizadas pela limitação energética, baixo poder de processamento e restrições de memória. Contudo, devido ao seu baixo custo e diversidade de sensores (*e.g.,* temperatura, umidade, pressão, ruídos, movimento) suas aplicações atingem as mais variadas áreas, como saúde, militar, ambiental e doméstica [3, 12, 21], criando assim oportunidades de introdução à instrumentação no controle de casas, cidades e meio ambiente.

O ambiente monitorado pelas RSSF pode ter caráter físico, químico, biológico ou simulado. Além das funcionalidades de monitoramento, o usuário pode contar com a instrumentação e a reação a eventos e fenômenos. Sendo o usuário qualquer entidade civil, governamental, comercial, militar ou industrial.

O processo de monitoramento que compõe as RSSFs conta com três elementos distintos, porém intimamente interligados, que são: o fenômeno, que pode ser uma manifestação, sinal ou sintoma observado; o sensor, dispositivo capaz de capturar as informações referentes ao fenômeno assistido; e o observador, que é o usuário a quem interessa o conteúdo dos dados monitorados pelo sensor.

2.1 Arquitetura dos Nós Sensores

O nó sensor, também conhecido como *mote*, é o dispositivo mais comum em uma rede de sensores. Composto basicamente por microcontrolador, memória, sensores, fonte de energia e um transceptor de rádio de baixa potência [4], os nós sensores são capazes de coletar informações do ambiente através dos seus sensores, processar dados e comunicar com os outros nós da rede. A Figura 2.1 mostra a infraestrutura básica de um nó sensor.

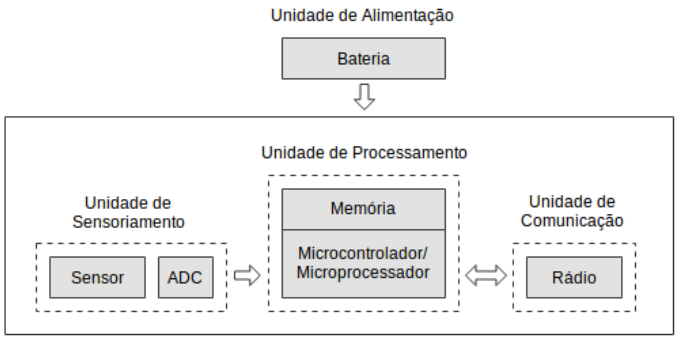


Figura 2.1: *Representação da infraestrutura básica de um nó sensor.*

Os microcontroladores presentes nos *motes* são responsáveis pela execução de tarefas, processamento dos dados e controle de funcionalidades dos outros componentes do nó sensor. Dentre os diversos benefícios provenientes dos microcontroladores, o baixo custo, a flexibilidade para se conectar a outros dispositivos e o baixo consumo de energia são as principais características que o elegem como mais indicados que os microprocessadores para compor a arquitetura dos *motes*.

Basicamente um nó sensor é composto por três tipos de memória, que são as de armazenamento de dados persistentes de configuração do *mote* (*e.g.,* EEPROM, EPROM, Memória Flash); as usadas para armazenar os programas e os dados constantes (*e.g.,* ROM, Memória Flash); e as memórias de acesso rápido, utilizadas para armazenas ponteiros, variáveis globais e pilhas (*e.g.,* RAM). Os requisitos de memória dependem muito da aplicação do usuário. Algumas plataformas de nós sensores apresentam capacidades de armazenamento de programa consideravelmente pequena até mesmo para o universo dos *motes*. Isso impossibilita, inclusive, o carregamento de algumas aplicações.

Com relação aos sensores, eles são pequenos dispositivos de leitura capazes de produzir continuamente sinal analógico em resposta a estímulos específicos provenientes do ambiente que ele está monitorando. As informações coletadas pelos sensores são enviados para um conversor analógico-digital (ADC), que converte esses dados para o formato digital e repassa-os para o microcontrolador. Dentre os diferentes tipos de sensores, têm-se os de leitura de luminosidade, temperatura, pressão, movimento, radiação, som,

orientação, mecânicos, químicos, magnéticos, *etc*. Em alguns casos a suíte de sensores é fornecida em separado da plataforma do *mote*.

Para que um nó sensor tenha potencial elétrico para o seu funcionamento é preciso que se mantenha uma fonte de energia com corrente contínua alimentando os componentes do *mote*. Essa fonte de energia pode ser proveniente de baterias ou interfaces de alimentação externa. As interfaces de alimentação externa, cujas mais comuns permitem energizar o nó sensor através da porta USB de um computador *host*, também podem ser conectadas a geradores de tensão, painéis solares ou capacitores de alta potência. Cada plataforma de nó sensor tem seu conjunto de fontes e/ou interfaces de alimentação.

O transceptor é um dispositivo que combina a capacidade de transmitir e receber informações através da propagação de sinal pelo meio de comunicação ao seu alcance. Dentre os principais veículos de transmissão que compõem o meio de comunicação de um nó sensor, o rádio frequência é o mais comum nos dispositivos de RSSFs. Isso porque as outras opções, como o laser e o infravermelho, requerem um maior campo de visão com os transceptores vizinhos, assim como sofre com a baixa sensibilidade às condições atmosféricas. Devido ao alto consumo de energia, a maioria dos transceptores de RSSF operam a maior parte do tempo em modo sono.

A diversidade de plataformas de nós sensores permite atingir os mais variados tipos de aplicação, comumente utilizadas em projetos de redes de sensores. A Tabela 2.1 mostra um comparativo entre as plataformas de nós sensores mais comuns em pesquisa de redes de sensores, tais como BTnode [7], CM3300 [2], MICA2 [18], MICAz [19], SHIMMER [41], TelosB [20, 52], TmoteSky [53], WiSMote [43], XM1000 [1] e Z1 [78]. O objetivo não é determinar a melhor plataforma, mas reunir e comparar as características de cada uma, possibilitando que o próprio usuário determine qual a plataforma de nó sensor mais adequada para as suas necessidades.

Tabela 2.1: *Comparativo entre as plataformas de nós sensores mais comuns em pesquisa de redes de sensores.*

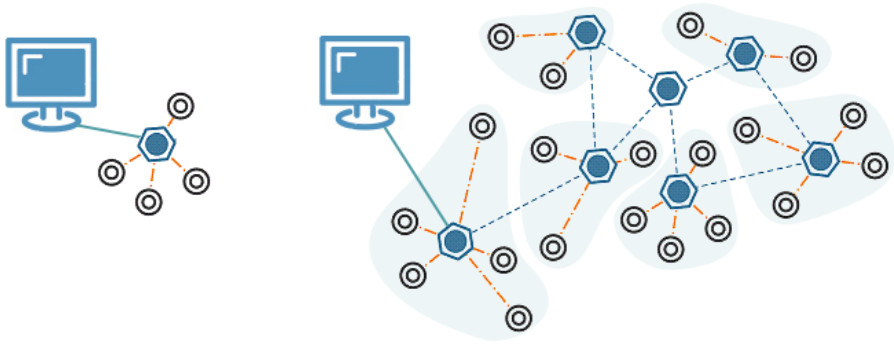
	Microcontrolador		Memória		Sensores	Transceptor					Fonte de Alimentação	
Motes	**MCU**	**Freq. Máx. de CPU**	**Mem. de Programa**	**Mem. RAM**	**Sensores Integrados**	**Chip de Rádio**	**Freq. de Operação**	**Alcance Indoor**	**Alcance Outdoor**	**Vazão Máx.**	**Bateria**	**Alimentação Externa**
BTnode	ATmega128L	8 MHz	128 KB	4 KB	nenhum	CC1000	315, 433, 868 e 915 MHz	--	--	39,3 kbps	2 x AA	J1 e J2
CM3300	MSP430-F1611	8 MHz	48 KB	10 KB	nenhum	CC2420	2400 a 2483.5 MHz	80 a 120 m	800 m	250 kbps	2 x AA	USB
MICA2	ATmega128L	8 MHz	128 KB	4 KB	nenhum	CC1000	315, 433, 868 e 915 MHz	--	150 a 300 m	39,3 kbps	2 x AA	--
MICAz	ATmega128L	8 MHz	128 KB	4 KB	nenhum	CC2420	2400 a 2483.5 MHz	20 a 30 m	75 a 100 m	250 kbps	2 x AA	USB
SHIMMER	MSP430-F1611	8 MHz	48 KB	10 KB	acelerômetro	CC2420	2400 a 2483.5 MHz	--	--	250 kbps	--	USB
TelosB	MSP430-F1611	8 MHz	48 KB	10 KB	(luminosidade, temperatura e humidade)[1]	CC2420	2400 a 2483.5 MHz	20 a 30 m	75 a 100 m	250 kbps	2 x AA	USB
Tmote Sky	MSP430-F1611	8 MHz	48 KB	10 KB	luminosidade, temperatura e humidade	CC2420	2400 a 2483.5 MHz	50 m	125 m	250 kbps	2 x AA	USB
WiSMote	MSP430-F5437	18 MHz	256 KB	16 KB	luminosidade, temperatura e acelerômetro	CC2520	2394 a 2507 MHz	--	--	250 kbps	AAA ou Li-ion	USB
XM1000	MSP430-F2618	16 MHz	116 KB	8 KB	luminosidade, temperatura e humidade	CC2420	2400 a 2483.5 MHz	20 a 30 m	120 m	250 kbps	2 x AA	USB
Z1	MSP430-F2617	16 MHz	92 KB	8 KB	temperatura e acelerômetro	CC2420	2400 a 2483.5 MHz	--	--	250 kbps	2 x AA ou AAA ou 1 x CR2032	USB

[1]. Modelo TPR2420CA

2.2 Arquitetura das Redes de Sensores

As redes de sensores sem fio são compostas por uma variedade de pequenos dispositivos autônomos capazes de monitorar um determinado ambiente e distribuir de forma cooperativa as informações coletadas através de comunicações sem fio. Esses dispositivos compactos, conhecidos como nós sensores ou *motes*, são capazes de realizar pré-processamentos das informações adquiridas e utilizar de maneira inteligente a interface de comunicação, resultando em considerável economia de energia.

Espacialmente distribuídos, os nós sensores transmitem as informações monitoradas para um observador. Um outro tipo de nó (conhecido como nó *sink*) é utilizado como ponto de escoamento dessas informações da rede, intermediando a comunicação da rede de sensores com o observador. O observador, geralmente representado por PC *host*, é composto por uma infraestrutura com processamento e armazenamento mais robusto, e responsável pela análise a armazenamento dos dados recebidos. A Figura 2.2(a) mostra uma representação da infraestrutura básica de uma RSSF.



(a) (b)

Figura 2.2: *(a) Estrutura básica de uma rede de sensores. (b) Estrutura de uma rede de sensores baseada em clusters.*

Em um outro rearranjamento estrutural de uma RSSF, assim como apresentado na Figura 2.2(b), os nós são distribuídos em grupos conhecidos como *clusters*, com um dos nós fazendo a função de roteador (*cluster head*). Essa abordagem é utilizada em situações em que é preciso reduzir os custos de comunicação ou mesmo garantir a entrega de dados, e essencial na implantação de redes em grande escala.

Segundo Jun Zheng [77], projetos de implementações de redes de sensores são específicos para cada aplicação, cujas diferentes características de cada implementação possibilita a classificação das RSSF em diferentes categorias:

1. Redes Estáticas e Redes Móveis: considerando a mobilidade dos nós em uma rede de sensores, a rede pode ser classificada como estática ou móvel. Na rede estática, todos os nós apresentam posicionamento estático, ou seja, sem movimento. Já em

uma rede móvel, existem nós com potencial de mobilidade, não mantendo assim um posicionamento constante durante o funcionamento da rede. Redes móveis, costumam envolver uma maior complexidade tanto da implementação como do gerenciamento e controle. Um exemplo de rede móvel é apresentando em [47], onde animais são supervisionados de maneira autônoma.

2. Redes Determinísticas e Redes Não-Determinísticas: correlacionado com as redes estáticas, o planejamento de uma rede determinística regulamenta que os nós tenham posicionamento fixo. No entanto, nem todas as redes estáticas são determinísticas. Isso porque, uma rede determinística é projetada para que todos os nós tenham o mesmo posicionamento em todas as execuções da rede. Essa condição, contudo, dificulta a implantação dos nós sensores em ambientes agressivos e/ou hostis, onde há riscos de reposicionamento de algum nó. Já as redes não-determinísticas não necessitam de um planejamento prévio de posicionamento, podendo os nós serem implantados de forma aleatória. Isso promove uma maior escalabilidade e flexibilidade da rede, entretanto, requer mais complexidade de gerenciamento e controle.
3. Redes com *Sink* Estático e Redes com *Sink* Móveis: assim como os nós sensores, os nós *sinks* podem ser estáticos ou móveis. Em uma rede com o nó *sink* estático, o nó tem o seu posicionamento constante durante todo o período de execução da rede. Por outro lado, o nó *sink* móvel pode variar a sua posição na rede de acordo às suas necessidades. Com a proximidade dos nós sensores com o *sink*, a carga de tráfego nos nós sensores tende a aumentar drasticamente, provocando o efeito *hotspot* [5, 13], o que pode acarretar em morte prematura dos nós sensores, resultando em partição da rede ou até mesmo colapso e interrupção de operação. Uma solução para aliviar o efeito *hotspot* na rede é a utilização de nó *sink* móvel. Isso permite que o *sink* se movimente em torno da região de monitoramento para coletar os dados dos nós sensores, equilibrando assim a quantidade de tráfego nos nós.
4. Redes com Único *Sink* e Redes com Múltiplos *Sinks*: uma implantação de rede de sensores pode conter um ou mais nós *sinks*. Sendo redes com apenas um nó *sink* conhecidas como redes de único *sink*, e com dois ou mais nós *sinks* conhecidas como redes de múltiplos *sinks*. A quantidade de nó *sink* que será utilizada em uma implantação de rede de sensores é propriamente uma predileção do usuário, guiado pelas necessidades da aplicação do projeto de rede. Entretanto, um outro fator pode ser levado em conta quando se define a quantidade de nós *sinks* presentes na rede, que é o efeito *hotspot*. Quando bem projetadas, redes com múltiplos *sinks* tendem a equilibrar a carga de tráfego nos nós sensores, aliviando assim o efeito *hotspot* na rede.
5. Redes com Único Salto e Redes com Múltiplos Saltos: redes de sensores também

podem ser classificadas de acordo o número de saltos. Redes cujos nós sensores apresentam um único salto até o nó *sink* são conhecidas como redes de único salto. Em uma outra perspectiva, quando se tem dois ou mais saltos em uma comunicação entre nós, a rede é conhecida como sendo de múltiplos saltos. Uma rede de múltiplos saltos, apesar de apresentar maior complexidade de gerenciamento e controle, proporciona consideráveis benefícios, principalmente a projetos de redes de grande escala. Isso porque, em uma rede de múltiplos saltos estima-se ter comunicações sem fio de curto alcance, o que diminui o consumo total de energia na rede, assim como o número de colisões e latências de entregas. Além do mais, a utilização de nós intermediários, comuns em redes de múltiplos saltos, possibilita a agregação de mensagens, eliminando a redundância de dados e consequentemente a quantidade total de tráfego na rede.

6. Redes Autorreconfiguráveis e Redes Não-Autorreconfiguráveis: inerente à configurabilidade dos nós sensores, uma rede pode ser classificada como autorreconfigurável ou não-autorreconfigurável. Em uma rede autorreconfigurável, os nós sensores têm a capacidade de se organizar de forma autônoma na rede, mantendo sua conectividade e colaborando com a realização das tarefas de monitoramento. Já nas redes que não são autorreconfiguráveis, ou seja, redes não-autorreconfiguráveis, o controle dos nós sensores é realizado por um controlador central, assim como o gerenciamento da coleta de informações. Projetos de redes não-autorreconfiguráveis são mais recomendados para implantações de pequena escala.
7. Redes Homogêneas e Redes Heterogêneas: de acordo com a capacidade dos nós sensores, uma determinada rede pode ser classificada como sendo homogênea ou heterogênea. Redes de sensores homogêneas são compostas por plataformas de nós sensores com as mesmas capacidades e características. Antagonicamente, as redes de sensores heterogêneas apresentam diferentes plataformas de nós sensores. A classificação quanto a heterogeneidade de uma rede de sensores envolve características particulares de cada nó sensor. Isso inclui a avaliação da capacidade energética, de computação, armazenamento, comunicação, *etc.* Como benefício, redes heterogêneas podem delegar tarefas mais complexas de processamento e comunicação para os nós sensores mais sofisticados, de modo a melhorar a eficiência energética total da rede.

CAPÍTULO 3

Ambientes de Testes para Redes de Sensores Sem Fio

O projeto de uma rede de sensores é uma tarefa muito específica da aplicação. Devido às peculiaridades encontradas em cada implantação de projeto, como ambientes hostis ou infraestruturas com grande quantidade de nós, torna-se indispensável avaliações preliminares do comportamento da rede.

A avaliação de uma rede de sensores é dada através do uso de ambientes específicos de testes. Um ambiente de teste, também conhecido como *testbed*, é uma infraestrutura física e/ou lógica onde as aplicações dos nós sensores são executadas, obedecendo diretrizes para cada item ou condição avaliada.

Quanto aos *testbeds*, eles podem ser físicos, onde todos os nós sensores são reais; lógicos, com os dispositivos e comportamentos dos nós sensores sendo simulados ou emulados; ou híbridos, em que o ambiente real da infraestrutura física é integrado ao ambiente virtual da infraestrutura lógica, resultando em um novo ambiente real-virtual.

Os *testbeds* físicos apresentam uma alta-fidelidade na avaliação dos experimentos, possibilitando ao usuário definir um planejamento mais detalhado antes da implantação final do projeto. Contudo, esse tipo de infraestrutura muitas vezes sofre com algumas limitações, como a tendência a ser composta de uma arquitetura de pequena escala, em função dos seus altos custos; ter pouca flexibilidade, já que tradicionalmente os nós têm posicionamento fixo; e apresentar heterogeneidade restrita, com pouca variedade de plataformas de nós sensores.

Já os *testbeds* lógicos resolvem alguns problemas presentes na maioria dos *testbeds* físicos, mas ao mesmo tempo, apresentam outras limitações que inviabilizam o seu uso para qualquer tipo de experimento. Apesar dos *testbeds* lógicos serem altamente flexíveis e escaláveis, sendo úteis para, rapidamente, tentar novas ideias, pecam quanto à fidelidade dos experimentos.

Assim, como alternativa às limitações apresentadas para ambos os *testbeds*, físicos e lógicos, surgiram os *testbeds* híbridos. Esse tipo de infraestrutura aborda soluções para resolver muitas das deficiências apresentadas pelos supracitados *testbeds*, como a

adição de flexibilidade ao *testbeds* físicos, e o acréscimo de alta-fidelidade ao *testbeds* lógicos.

Mais detalhes sobre cada um desses ambientes de teste, assim como as peculiaridades de suas principais ferramentas, serão apresentados nas seções seguintes.

3.1 *Testbeds*

Como citado na introdução deste capítulo, o termo *testbed* trata-se apenas da designação de um ambiente de teste. Genérico, por sinal. Contudo, para efeito de organização, trataremos neste documento o *testbed* como sendo uma infraestrutura que obedeça basicamente as seguintes premissas: ser física ou híbrida; e oferecer *softwares* de gerência, onde os usuários possam carregar as aplicações para testes e manipular seus experimentos.

De maneira geral, um *testbed* é apresentado ao usuário a partir do seu sistema de gerência, *software* de acesso local ou remoto onde é possível reservar um horário de uso, assim como definir os recursos que serão usados para o experimento.

Iniciado o período de uso do *testbed*, o usuário pode acessar os nós sensores reservados e carregar suas aplicações para teste. Paralelo à execução das aplicações, o usuário pode utilizar as ferramentas do sistema de gerência para avaliar o experimento.

Na seção seguinte são apresentadas descrições sobre alguns *testbeds* de infraestrutura física; e na Seção 3.1.2, os *testbeds* híbridos.

3.1.1 *Testbeds* Físico

O *testbed* Trio

Criado e implantado nas imediações da Universidade da Califórnia, Berkeley, o *testbed* Trio [28] foi constituído para contar com uma infraestrutura de 557 nós sensores, alimentados por energia solar; 7 *gateways*, para encaminhamento de tráfego entre a rede 802.15.4 do *testbed* e a rede 802.11 do *backbone* de rede sem fio; 2 repetidores; e 1 servidor raiz.

Principais características: o *testbed* Trio oferece alimentação por energia renovável; sua infraestrutura cobre uma área de 50.000 m^2 ao ar livre, possibilitando assim a captura de nuances presentes nesses tipos de ambientes; e apresenta flexibilidade e escalabilidade a prova de falhas, graças aos seus *softwares* de gerência.

O *testbed* MoteLab

MoteLab [74] é um *testbed* composto por 184 nós sensores, conectados e distribuídos ao longo de três andares do prédio de Engenharia Elétrica e Ciência da Computação, na Universidade de Harvard. Através de uma interface Web intuitiva, os usuários podem realizar experimentos locais ou remotos. Favorecidos pelo acesso simplificado e eficientes algoritmos de gerência, os usuários têm acesso a acelerados processos de implantação e depuração de *software*, além da possibilidade do desenvolvimento de registros de dados.

Principais características: o *testbed* fornece manipuladores de reprogramação; permite avaliações *offline*; fornece um nó sensor com um multímetro digital conectado à rede, para monitorar o consumo de energia; e disponibiliza o código-fonte do gerenciador, para ser implantado em outros *testbeds*.

O *testbed* SignetLab

O *testbed* SignetLab [16, 17] foi desenvolvido e implantado no Departamento de Engenharia da Informação, da Universidade de Padova, na Itália. Sua infraestrutura conta com 48 nós sensores do tipo EyesIFXv2 e uma rica ferramenta de gerência, fornecendo o máximo de controle possível ao *testbed*.

Principais características: a ferramenta de controle e gerência do *testbed* SignetLab fornece uma API para que os usuários possam construir e personalizar as suas próprias ferramentas, que originalmente conta com um painel de compilação e execução, um visualizador de saída serial, um painel de entrada de comandos para os nós, e um painel de visualização de terreno. O SignetLab também oferece um eficiente módulo de programação dos nós sensores, garantindo alta-fidelidade na programação ao longo de múltiplos saltos.

A federação IoT-LAB

Conhecido anteriormente como SensLAB [24], o IoT-LAB [34] trata-se de uma federação de *testbeds* para redes de sensores sem fio. Sua corpulenta infraestrutura é composta por mais de 2.700 nós sensores, distribuídos por 6 instituições francesas de pesquisa. Projetado para permitir experimentos em grande escala, o IoT-LAB fornece também suporte a experimentos com nós fixos e/ou móveis. Seu robusto sistema de gerência fornece serviços de análise de dados em tempo real, assim como o armazenamento de resultados para análise posterior.

Principais característica: o IoT-LAB permite que cada nó sensor possa ser configurado como um nó *sink*, possibilitando que informações sejam enviadas através da Internet para qualquer lugar ou dispositivo. Outra peculiaridade do IoT-LAB é a

capacidade de injetar ruídos aos experimentos, através do envio de dados falsos. Assim como permitir que o usuário ligue e desligue os nós sensores com a finalidade de imitar falhas.

O *testbed* TWIST

O *testbed* TWIST [39] foi desenvolvido pelo *Telecommunication Networks Group* (TKN), na Faculdade de Engenharia Elétrica e Ciência da Computação, da Universidade Técnica de Berlim, na Alemanha. Sua implantação foi feita ao longo de três andares, cobrindo uma área fechada de mais de 1.500 m^2. Atualmente, o *testbed* TWIST conta com uma arquitetura escalável e flexível, composta por 204 nós sensores, sendo metade do tipo TmoteSky e a outra metade do tipo eyesIFX. Isso possibilita, inclusive, experimentos heterogêneos.

Principais características: o TWIST disponibiliza todo o *software* de gerência com código aberto, em que é possível destacar o controle ativo de fonte de alimentação dos nós, permitindo fácil transição entre a alimentação energética via USB ou via bateria; a autoconfiguração; e o suporte à seleção dinâmica de topologia. Também foi desenvolvido e disponibilizado para os usuários um *plugin* para o COOJA, possibilitando carregar o código das aplicações dos nós sensores, além de tornar mais fácil a observação da execução do experimento.

O *testbed* Indriya

Implantado ao longo de 23.500 m^3 internos da Universidade Nacional de Singapura, o Indriya [25] foi projetado para ocupar 3 andares da Escola de Computação, compondo uma distribuída arquitetura tridimensional. Sua infraestrutura é composta por 139 nós sensores do tipo TelosB, conectados através de cabos USB ativo (permite distâncias de até 25m) a 6 Mac Mini. Seu sistema de gerência é garantido por uma versão modificada da aplicação baseada em Web, MoteLab, a qual foi desenvolvida para o *testbed* de mesmo nome.

Principais características: o Indriya foi projetado para ser um *testbed* de baixo custo. Suas despesas com a implementação e manutenção atingiram valores inferiores a *testbeds* conceituados para redes de sensores, sem que sua eficiência fosse comprometida.

O *testbed* WSNTB

O *testbed* WSNTB [59] foi desenvolvido no Departamento de Ciência da Computação da Universidade Nacional de Tsing Hua, em Taiwan. Sua infraestrutura é composta por 34 nós sensores do tipo MICAz e TmoteSky, distribuídos por duas salas de laboratório. O WSNTB apresenta uma arquitetura reconfigurável e escalável, composta

por 3 *gateways*, sendo cada um acessível a 17 nós sensores (alguns nós acessam mais de um *gateway*).

Principais características: O WSNTB fornece um servidor com bibliotecas instaladas do Microsoft Visual Basic Runtime e do Java Runtime Environment, a fim de possibilitar que usuários criem portas seriais no seus computadores locais, através de *softwares* de virtualização de porta. O *testbed* também oferece suporte à seleção de arquitetura plana ou hierárquica.

3.1.2 *Testbeds* Híbrido

A federação WISEBED

O WISEBED [15] não é propriamente um *testbed*, mas um conjunto de *testbeds* que compõe uma infraestrutura federada, com participação de um total de 9 universidades e centros de pesquisas espalhados pela Europa. A infraestrutura híbrida do WISEBED é constituída de elementos físicos, simulados e emulados, capazes de fornecer encorpados ambientes para a realização de experimentos em grande escala. Apenas de nós sensores físicos o WISEBED tem mais de 750 unidades.

Principais características: o WISEBED fornece conectividade emulada, permitindo explorar padrões de diferentes conectividades, como adicionar conectividade entre pares de nós sem nenhuma conexão física, ou mesmo construir novas topologias na rede, em cima de uma nova malha física subjacente; fornece um SDK para os usuários desenvolverem e sintetizarem aplicações que serão executadas, isso fornece transparência para o *testbed*, evitando reimplementações de código portados entre os ambientes físicos, simulados e emulados; fornece controle de experimentos programável em modo de arquivo de lote; e possibilita o armazenamento dos resultados dos experimentos para consultas posteriores.

O *testbed* Kansei

Desenvolvido originalmente no Departamento de Ciência e Engenharia da Computação da Universidade do Estado de Ohio, o *testbed* Kansei [6] foi projetado para suportar uma infraestrutura híbrida de mais de 1.000 nós, sendo que 410 unidade são de nós sensores puramente físicos. Subdivididos entre nós fixos, portáteis e móveis, a infraestrutura do *testbed* Kansei suporta experimentos em ambientes internos e externos (ao ar livre), além de composições de robustas redes heterogêneas. Atualmente o *testbed* Kansei pertence à federação KanseiGenie [63, 64].

Principais características: o *testbed* Kansei oferece dois canais para os usuários acessarem e gerenciarem seus experimentos, que são via interface Web ou através do

uso programas específicos, via Web *services*. O usuário também pode utilizar-se de ferramentas no sistema de gerência do Kansei para ingerir eventos aos seus experimentos.

3.2 Simuladores

Inicialmente citados neste capítulo como *testbeds* lógicos, os simuladores compreendem-se de ferramentas que utilizam modelos baseados em representações matemáticas para conjecturar o comportamento de dispositivos físicos. A maneira como são modelados esses dispositivos permite classificar os simuladores de acordo com os três níveis de abstração de um sistema computacional convencional: aplicação, sistema operacional e *hardware*. Entretanto, algumas ferramentas de simulação permitem simulações em que nós de diferentes níveis se comunicam em um processo interativo e dinâmico.

A seguir, descrevemos sobre os principais simuladores para redes de sensores, considerando a abordagem *top-down* em que os sistemas computacionais são organizados; assim como os simuladores multiníveis. Finalizamos então este capítulo, abordando o conceito de simulação híbrida, integração pontualmente empregada nos *testbeds* híbridos, descritos na Seção 3.1.2.

3.2.1 Simuladores em Nível de Aplicação

Simuladores que rodam em nível de aplicação geralmente são simuladores de redes convencionais que contam com modelos de sensores implementados. Eles apresentam grandes deficiências quanto à modelagem com precisão de detalhes específicos da plataforma de *hardware*. A limitação quanto à implementação de tais modelos impossibilita que esses simuladores, por exemplo, averiguem com precisão o consumo de energia do nó e/ou da rede. Esse tipo de simulador é mais adequado para implementações onde o funcionamento do *hardware* específico não é tão importante quanto o da rede em si, tendo uma boa fidelidade, por exemplo, em experimentos que envolvam testes de protocolos de roteamento.

Dentre os principais simuladores para rede de sensores que rodam em nível de aplicação, destaque para:

NS-2 Simulador mais utilizado em experimentos com redes convencionais, o NS-2 [26] foi adaptado, através de algumas extensões (*e.g.*, Sensorsim [55], Scatterweb [57], Mannasim [49]) para dar suporte às simulações de rede de sensores. No entanto, o NS-2 e suas extensões ainda são bastante limitadas em simulações de rede de sensores, pois falta-lhes a capacidade de modelar o *hardware* com a precisão de detalhes necessária para obter um resultado significativo sobre as métricas de desempenho.

NS-3 Mais um *framework* que faz parte da série de simuladores de rede NS (*Network Simulator*), que contém também o NS-1 e o NS-2. No entanto, a sua implementação independe de certa forma do seu antecessor NS-2, sendo o seu código totalmente implementado do zero. Contudo, mesmo ainda sendo considerado novo e não tendo a mesma quantidade de extensões desenvolvidas para o NS-2, o NS-3 [75] vem com uma série de inovações, incluindo atividades emuladas, que possibilitam uma melhor adaptação para os experimentos com rede de sensores.

Castalia Um dos simuladores de alto nível mais utilizados para testes com rede de sensores, o Castalia [9] é baseado no OMNeT++ [69] e apresenta modelos bem definidos de processos físicos, dispositivos de sensoriamento e ruídos, protocolos de roteamento e pilhas de rede [66]. Assim como OMNeT++, o Castalia utiliza a linguagem NED, com possibilidade, inclusive, dos usuários utilizarem a linguagem para modificar parâmetros nos módulos referentes ao comportamento do nó sensor.

J-Sim Como o NS-2 e a maioria dos simuladores que executam algoritmos em nível de aplicação, o J-Sim [60] é um simulador de propósito geral. Contudo, ao contrário do NS-2, onde cada nó é representado como um objeto, o J-Sim utiliza o conceito de componentes. Esse tipo de abordagem adotada pelo J-Sim proporciona uma escalabilidade maior que a do NS-2 [66]. No J-Sim, esses componentes de alto nível são basicamente divididos em três tipos [66]: nó *target*, que produz estímulos na rede; nó sensor, que reage aos estímulos produzidos pelo nó *target*; e nó *sink*, que serve de destino final para o relatório de estímulos gerados dos experimentos.

GloMoSim Bastante conhecidos pelos pesquisadores e desenvolvedores de aplicações para rede de sensores, o extensível GloMoSim [76] é altamente capacitado a ambientes paralelos, o que o torna consideravelmente escalável, sendo implementado com sucesso em memórias compartilhadas e memória distribuída [66]. Além disso, apresenta uma determinada técnica de agregação de nós, cujos benefícios possibilitam um alto desempenho na simulação. Nessa abordagem cada nó representa uma área geográfica da simulação sendo cada entidade, em particular, representada pela posição física do nó.

3.2.2 Simuladores em Nível de Sistema Operacional

Nos simuladores para rede de sensores que rodam em nível de sistema operacional, os *softwares* do nó sensor são compilados sobre a plataforma *host* e, então, a API do *hardware* é abstraída [23]. Como essas abordagens são desenvolvidas de acordo o funcionamento do RTOS (*Real-Time Operating System*), então esses tipos de simuladores têm por características não oferecer flexibilidade em estender o suporte a outras plataformas de sistemas operacionais [23].

Atualmente, têm-se duas ferramentas de simulação que executam projetos de simulação de rede de sensores em nível de sistema operacional, são elas: TOSSIM[1] e COOJA[2].

TOSSIM Primeiro simulador implementado para execuções de código em nível de sistema operacional, o TOSSIM [45] fornece alta-fidelidade em experimentos desenvolvidos em TinyOS. Proporcionando um simulador específico para rede de sensores capaz de prover escalabilidade, extensibilidade e uma melhor precisão, comparado aos simuladores que executam em níveis de aplicação. Isso é possível devido à adoção de um modelo probabilístico que utiliza erros de *bits* para criar abstrações de um ambiente do mundo real. Além da característica inerente dos simuladores que rodam em nível de sistema operacional de não permitirem experimentos heterogêneos quanto ao SO embarcado, o TOSSIM também apresenta algumas limitações que restringem o seu uso em cenários mais abrangentes, como a impossibilidade de executar diferentes programas em uma mesma rede e de ter diferentes compilações para uso no simulador e no nó sensor real.

COOJA Simulador multinível implementado em Java, cujo objetivo principal é dar suporte ao sistema operacional Contiki. No entanto, o COOJA [54] apresenta uma série de novidades em relação ao TOSSIM, tais como: devido à sua natureza modular e extensiva, o COOJA permite fáceis alterações, substituições ou inclusões de plataformas de nó sensor, transceptores de rádio e modelos de transmissão de mensagem; ele também dá suporte à comunicação entre diferentes níveis do sistema, possibilitando, por exemplo, que nós implementados em nível de aplicação se comuniquem com nós que rodam em nível de sistema operacional; além do mais, o COOJA é totalmente heterogêneo, ou seja, ele pode comportar em uma mesma rede tanto nós de diferentes plataformas de *hardware* quanto nós com diferentes sistemas embarcados (quando integrado a outros simuladores). Essas características fazem do COOJA um simulador consideravelmente flexível.

3.2.3 Simuladores em Nível de *Hardware*

A abordagem de simulação em nível de conjunto de instrução de ciclo preciso possibilita a execução dos mesmos binários que são embarcados no *hardware* real, sem que o *software* sofra qualquer modificação. Assim, esses simuladores independem da sintaxe de programação e compiladores utilizados para gerar os binários [23], uma vez que interpretam apenas código de máquina. Além do mais, esses tipos de simuladores, ou

[1] TOSSIM está incluso na distribuição do TinyOS
[2] COOJA está incluso na distribuição do Contiki OS

seja, que provem simulações em nível de conjunto de instrução de ciclo preciso, fornecem alta-fidelidade quanto ao comportamento do microcontrolador da plataforma de nó sensor. Isso é possível devido à execução baseada em eventos permitir tempos precisos enquanto a utilização do processador da máquina *host* é mantida tão baixa quanto possível [32].

Os três principais simuladores que rodam em nível de conjunto de instrução de ciclo preciso, seguindo a ordem cronológica de criação, são: ATEMU [56], Avrora [68] e MSPSim [30].

ATEMU Apontado por muitos como o primeiro simulador de rede de sensores capaz de emular de forma precisa o *hardware* [42, 56], o ATEMU é capaz de emular microcontroladores ATmel AVR, temporizador e a interface de rádio, sendo o restante dos componentes e a comunicação sem fio tendo os seus comportamentos simulados. Entretanto, o ATEMU apresenta algumas deficiências, tais como, o baixo desempenho, a incapacidade de analisar o funcionamento de todos os nós da rede de forma simultânea (ele analisa o nó de forma individual) e a apresentação de problemas relacionados à portabilidade e extensibilidade. Além disso, o ATEMU usa uma estratégia de implementação ciclo-a-ciclo, onde cada nó e cada dispositivo são avançados por um ciclo de relógio a cada rodada. Isso assegura que os nós, seus dispositivos internos e a sua comunicação de rádio estejam corretamente sincronizados [68]. Contudo, mesmo a utilização dessa estratégia alcançando a sincronização de maneira consideravelmente fácil, ela escalona mal, pois cada dispositivo adiciona trabalho a cada ciclo do relógio [68].

Avrora Simulador com suporte à emulação de microcontroladores ATmel AVR, o Avrora é escalável, não intrusivo e com um sistema de fila de eventos que o torna bastante eficiente quanto ao processo de sincronização dos dispositivos com o relógio do processador. Dos dispositivos emulados pelo Avrora, têm-se, além do supracitado microcontrolador, a porta serial, os temporizadores, os pinos de dados conectados externamente e o rádio AM digital (orientado por *byte* e controlado por *software*) [42]. Uma das características mais importante do Avrora é a capacidade de suportar simulações *multithreading*, onde cada nó é carregado em um único *thread*, executando concorrentemente com outros nós em outros *threads* (cada *thread* com seu próprio tempo de simulação). Esse modelo proporciona uma melhor escalabilidade através de execuções paralelas devido ao fato de que cada nó é simulado em seu próprio segmento [42]. Igualmente ao ATEMU, o Avrora também analisa o funcionamento do nó de forma individual. Além do mais, apresenta limitações quanto à carência de um mecanismo uniforme capaz de modelar os efeitos de uma mudança no ambiente e como a rede pode reagir a tais mudanças [42]. Atualmente o Avrora conta com uma versão modificada conhecida como AvroraZ [23], que adi-

ciona suporte a uma nova plataforma de *chip* de rádio, a Texas Instruments Chipcon CC2420.

MSPSim Um dos *frameworks* que compõe a distribuição do Contiki, o MSPSim é um simulador baseado em Java capaz de emular de maneira completa os microcontroladores da série MSP430 e o *chip* de rádio CC2420. O MSPSim fornece uma API para integração com outros *frameworks* de simulação de redes de sensores, tais como o COOJA, que devido à sua natureza multinível permite simulações muito mais opulentas. Além disso, o MSPSim também fornece uma série de ferramentas e serviços que auxiliam o usuário em seus experimentos, como por exemplo, depurador, *breakpoints*, *watchpoints*, *logging* e *single stepping*.

3.2.4 Simuladores Multiníveis

A possibilidade de permitir a comunicação entre nós sensores de diferentes níveis vai desde o suporte nativo do próprio simulador, até a integração de diferentes simuladores. Neste último quesito, as integrações dependem muito das características multiníveis dos simuladores. Alguns simuladores por natureza já dão suporte a essa condição, mesmo não a utilizando em suas simulações.

A simulação entre níveis permite que nós de cada um dos níveis possam coexistir e interagir na mesma simulação. Isso permite implementações híbridas com granulações de detalhes mais precisas, tendo um maior desempenho e escalabilidade.

Um exemplo de uma eficiente integração de simuladores que executam aplicações em diferentes níveis do sistema é:

COOJA/MSPSim/AvroraZ Como já descrito na Seção 3.2.2, o COOJA foi desenvolvido para permitir por padrão a integração multinível. Mesmo não tendo um modelo implementado para o mais baixo nível do sistema (*hardware*), ele fornece uma modelagem bastante intuitiva das simulações em níveis de aplicação e sistema operacional. Para suprir essa deficiência, de falta de suporte ao modelo de simulação em nível de *hardware*, o COOJA é integrado aos simuladores de baixo nível MSPSim e AvroraZ, que fornecem interfaces que possibilitam que essas integrações sejam realizadas de forma bastante prática. Como resultado da integração COOJA/MSPSim/AvroraZ [32] é possível, por exemplo, prover simulações em grande escala utilizando-se de uma abordagem heterogênea com nós simulados em nível de conjunto de instrução de ciclo preciso e nós simulados em níveis superiores. Segundo Österlind *et al.* [54], se combinarmos 1% dos nós emulados com 99% dos nós Java, podemos simular mais de 2.000 nós (20 emulados e 1980 baseados em Java), que por sinal, consumiria o mesmo tempo de execução que uma simulação com apenas 50 nós emulados. A utilização desse tipo de abordagem permite leituras

mais detalhadas sobre o funcionamento do nó sensor, mantendo o alto desempenho e a escalabilidade da rede.

3.2.5 Simuladores Híbridos (*Hardware-In-the-Loop*)

Utilizando-se da técnica de simulação *Hardware-In-the-Loop* (HIL), originária da década de 1950 e comumente aplicada à indústria automobilística, militar e aeroespacial [58], pesquisadores e desenvolvedores interpolam funções e/ou eventos de nós sensores físicos com nós sensores simulados provendo ambientes de testes com características híbridas. Dentre exemplos de ferramentas de simulação para rede de sensores que se utiliza da técnica HIL para corrigir deficiências em simuladores ou prover escalabilidade a *testbeds* físicos, temos: SensorSim [55], MULE [71], EmTOS/EmSim [38], SEMU [48], DiSenS [72, 73], H-TOSSIM [46] e Hy-Sim [62].

SensorSim *Framework* de simulação para redes de sensores capaz de modelar tanto experimentos de simulação em *software* puro como experimentos de simulação híbrida. Para proporcionar essa interação entre componentes físicos e simulados, foi projetado um *middleware* denominado SensorWare [10], que foi baseado na linguagem de *script* TCL, estendida com uma nova API para dar suporte a códigos móveis, permitindo que aplicações com capacidade de locomoção pela rede de sensores fossem desenvolvidas. O SensorSim conta também com uma arquitetura composta por três tipos de nós: (a) *Sensor Nodes*, que utilizam uma série de transdutores para monitorar o ambiente; (b) *Target Nodes*, responsáveis por gerar os estímulos para os *Sensor Nodes*; e (c) *User Nodes*, que são os nós clientes utilizados para administrar a rede de sensores. Já a comunicação entre os nós do SensorSim é facilitada pelos canais separados: *Sensor Channel* e *Network Channel*. Enquanto os canais do tipo *Sensor Channel* fornecem meios que possibilitam que os *Sensor Nodes* se comuniquem com os *Target Nodes*, os *Networks Channels* são utilizados para reencaminhamento de pacotes para outras redes (através de *User Nodes* ou *Gateways*). Segundo os desenvolvedores, a utilização de dois canais pelo SensorSim proporciona facilidades para modelar o complexo comportamento do *Sensor Node*, além de possibilitar transmissões concorrentes.

MULE Simulador de rede de sensores desenvolvido especialmente para estender o TOSSIM para simulações híbridas. Ele combina a facilidade de depuração de um simulador de *software* puro, com a alta-fidelidade adquirida dos dispositivos de rádio e de sensoriamento dos *motes* reais. Para que fosse possível a extensão do TOSSIM a um simulador híbrido, os desenvolvedores do MULE substituíram os componentes responsáveis pela simulação da comunicação e do sensoriamento do

TOSSIM, por componentes que possibilitassem manipular a interação entre os nós simulados e os nós reais.

EmTOS/EmSim O EmTOS/EmSim foi projetado como um conjunto de ferramentas de simulação verdadeiramente modular, extensivo quanto a integração e interoperabilidade entre nós sensores dos tipos *motes* (MICA2 ou MICAz) e *microservers* (iPAQ ou Crossbow Stargate), e capaz de dar suporte tanto a aplicações EmStar/Linux quanto nesC/TinyOS. Mesmo que o EmStar não tenha sido projetado originalmente para dar suporte à escrita de aplicações para nós sensores de dispositivos extremamente restritos, como os *motes*, o desenvolvimento e adição do *framework* EmTOS ao conjunto de ferramentas de simulação do EmStar possibilitou que códigos de aplicações escritas em nesC fossem encapsulados na biblioteca do EmTOS, e passassem a interagir prontamente com nós sensores mais robustos, como os *microservers*. Para permitir a interação heterogênea e híbrida entre nós sensores simulados e reais, o EmStar disponibiliza bibliotecas que implementam primitivas *message-passing* IPC (*Inter-Process Communication*) entre processos Linux, onde cada serviço ou módulo é representado por um único processo, com o seu próprio espaço de endereços.

SEMU Diferentemente dos demais simuladores híbridos para redes de sensores, o SEMU utiliza máquinas virtuais para modelar a rede simulada. Sua infraestrutura baseada em camadas é composta, além da máquina virtual (instância modificada do QEMU), por mais quatro outras camadas, subdivididas em: camada de comunicação, camada de sistema operacional virtual, camada de módulo e camada de sistema operacional nativo. A camada de máquina virtual é justamente a camada superior da pilha. É nessa camada que se encontra os nós sensores simulados. A camada de comunicação possibilita que nós sensores simulados da camada superior, nós sensores reais e a interface gráfica distribuída se comuniquem com os componentes de gerenciamento que compõem a camada de sistema operacional virtual. A camada de sistema operacional virtual é composta por elementos responsáveis pela manutenção e gerenciamento do simulador como um todo, dentre os quais têm-se o *kernel* do SEMU, os componentes de inicialização e as interfaces de comando. A abordagem *top-down* da arquitetura do SEMU finaliza com duas camadas restantes, a camada de módulo e a camada de sistema operacional nativo. A primeira, contém diversas ferramentas para auxiliar os usuários, que vão desde componentes de configuração e verificação de *status* da simulação, a componentes de gerenciamento de execuções paralelas para múltiplos nós. A segunda, é propriamente uma instância do sistema operacional Linux, que serve de fundação para o *framework* de simulação.

DiSenS Simulador para rede de sensores que possibilita experimentos com nós total-

mente simulados ou simulações híbridas com integrações entre nós simulados e reais. No entanto, diferentemente dos outros simuladores híbridos apresentados neste documento, a proposta dos desenvolvedores do DiSenS é: em vez de utilizar dispositivos físicos para aumentar a riqueza de detalhes da simulação, fazer justamente o processo inverso, ou seja, utilizar a flexibilidade e escalabilidade dos simuladores de *software* puro para estender os *testbeds* físicos. Para isso, Wen *et al.* [73] desenvolveram um nó híbrido robusto, o qual denominaram de "super nó". Esses super nós são divididos em duas partes, virtual e físico. Para a parte virtual, tem-se um nó sensor baseado em *software* com algumas modificações, sendo uma delas a adição de mais uma interface de rádio para comunicações externas. Para a parte física, a infraestrutura híbrida do DiSenS utiliza placas Crossbow MIB510 para conexões seriais RS-232, e MIB520 para conexões seriais USB, ambas anexadas a nós sensores reais. Os nós sensores reais e virtuais fariam parte de redes totalmente independente não fossem pela utilização dos super nós. Portanto, toda a comunicação entre os nós sensores físicos é realizada através de rádios reais, e toda a comunicação entre os nós sensores virtuais é através de meios de rádio simulados; o que difere de muitos simuladores híbridos que utilizam o rádio real para a comunicação virtual.

H-TOSSIM Extensão que adiciona suporte a simulações híbridas ao tradicional simulador de rede de sensores para plataformas TinyOS, TOSSIM (ver Seção 3.2.2). A motivação para o desenvolvimento do H-TOSSIM vem da impossibilidade do TOSSIM de revelar erros na definição do comprimento de mensagens enviadas e problemas de sobrecarga em tarefas de cálculo. Além do mais, mesmo tendo uma extensão baseada em *software* para estimativas de consumo de energia (PowerTOSSIM), ela não é capaz de apresentar avaliações tão precisas como as leituras baseadas em *hardware*. Para resolver esses problemas, os desenvolvedores projetaram e implementaram mudanças na arquitetura base do TOSSIM, além de adicionar suporte a simulações híbridas. Dentre as alterações baseadas em *software* estão a inclusão de um modelo modificado do TOSSIM; um componente de transferência e transformação de mensagem; uma ferramenta de *gateway*; e uma interface gráfica. Para a composição do *hardware*, foram utilizados três nós sensores, sendo dois atuando como estações base (*sink*) e um outro nó sensor físico para a execução da aplicação de teste. Para estimar o consumo de energia, um multímetro pode ser conectado ao nó sensor de teste, proporcionando uma precisa avaliação de consumo de energia a um baixo custo de *hardware*.

Hy-Sim *Framework* de simulação para rede de sensores com suporte a características híbridas, que utiliza ferramentas desenvolvidas pela *MathWorks Corporation* para modelar as plataformas de nó sensores de maneira independente da aplicação. Ele

é estruturado em uma arquitetura multibloco capaz de fornecer aos usuários cinco tipos de simulação, distribuídas em três categorias: totalmente simulada, parcialmente simulada e totalmente independente. Para simulações totalmente simuladas, tanto o modelo de rádio como todos os sensores são modelados por *software*. A simulação parcialmente simulada pode ser caracterizada pela utilização de qualquer um dos três tipos distintos de nós: `HIL_SEM` (*Hardware-In-Loop SENsor*), que colhe dados a partir de dispositivos sensores reais, mas continua usando um modelo de rádio simulado para comunicação entre os nós; `HIL_RF` (*Hardware-In-Loop Radio Frequency*), que ao contrário do `HIL_SEM` utiliza transceptores de rádio reais e modelos de dispositivos de sensoriamento virtuais; e `HIL_FULL` (*Hardware-In-Loop FULL*), no qual tanto o rádio como os dispositivos sensores utilizados na simulação são totalmente reais. Já a simulação totalmente independente utiliza apenas nós sensores reais para compor a rede.

CAPÍTULO 4

Implementando um Simulador Híbrido

Considerada uma tendência desde o advento da computação, a simulação é uma técnica de apoio a decisão amplamente utilizada para modelar sistemas reais. Seja como complemento ou mesmo objeto de substituição de sistemas completos. Como ferramenta de reprodução da técnica de simulação, os simuladores caracterizam-se pela tentativa de gerar amostras de cenários representativos para um determinado modelo. Atualmente, suas aplicações atingem diferentes segmentos da indústria manufatureira, além de áreas como pesquisa e entretenimento.

Um fato histórico que reflete muito bem a necessidade da utilização de simuladores na pesquisa científica, refere-se aos testes com o primeiro míssil balístico, na Alemanha Nazista. Dado a inexistência de computadores digitais para simular os lançamentos, o único modo de testar os mísseis era realizando lançamentos reais. Cabendo aos engenheiros examinar todas os pedaços que sobravam do míssil, em busca de evidências que permitissem identificar a causa específica do fracasso. Isso acarretava, inclusive, em um alto custo temporal até outro míssil estar preparado novamente para teste.

Com a oportunidade de portar sistemas reais para o universo digital dos computadores, foi permitido ao homem "prever" o comportamento de determinados sistemas físicos, simplesmente utilizando modelos baseados em representações matemáticas. Isso possibilitou uma visão mais sistemática do conceito de simulação. Contudo, a simulação não é propriamente um modelo matemático, uma vez que não existe expressões analíticas fechadas ou mesmo um conjunto de equações que forneçam resultados sobre o comportamento do sistema a partir de uma forma analítica direta [14]. O que torna a simulação irredutível a um simples cálculo ou fórmula matemática.

Assim como um simulador pode modelar o comportamento de um sistema baseado em *software*, ele também pode fazê-lo a sistemas completos baseados em *hardware*. Quando o processo de simulação exige uma alta-fidelidade quanto ao comportamento e características do sistema alvo, a técnica de simulação emana a um novo conceito, conhecido como emulação.

A emulação, assim como a simulação, possibilita prever o comportamento de sistemas, gerando situações hipotéticas, a partir da entrada de dados específicos,

respeitando um conjunto de premissas. Entretanto, quando define-se um sistema como emulado, espera-se que a sua modelagem tenha um alto grau de fidelidade. Para alcançar tal objetivo, ou seja a alta-fidelidade, a ferramenta de emulação (emulador) utiliza-se do sistema hospedeiro (*hardware* ou *software*) para criar novas funções que atendam às características do comportamento do sistema alvo.

Quando se simula ou emula um sistema, espera-se na maioria dos casos, que esse sistema se comporte parcialmente ou completamente como o seu sistema real correspondente. Entretanto, alguns ambientes e dispositivos que compõem os sistemas reais apresentam um alto grau de complexidade quanto a sua modelagem (*e.g.*, túneis de vento, comunicação sem fio). Em outros casos, são os sistemas reais que são complexos de serem implementados (*e.g.*, alvo para testes de radar, estradas para testes de pneus). Para resolver a maioria desses problemas, os pesquisadores e desenvolvedores passaram a utilizar o conceito de integração com ambientes/dispositivos externos, conhecido como simulação híbrida.

A simulação híbrida, apesar da ambiguidade com relação a aplicação do termo na literatura, refere-se aqui como sendo a técnica de execução de aplicações em uma infraestrutura integrada, composta por ambientes virtuais (simulados ou emulados) e ambientes reais (*i.e.*, dispositivos físicos, fenômenos da natureza, *etc.*).

4.0.6 O Conceito de Simulação Híbrida Aplicado às Redes de Sensores Sem Fio

Conforme mencionado do Capítulo 1, como apoio ao projeto [CIA]^2/GT-TeI, que almeja contribuir com o paradigma Internet das Coisas, a partir do desenvolvimento de um *testbed* público para redes de sensores sem fio. Propomos neste trabalho, a utilização de simuladores como uma alternativa à extensão da infraestrutura do *testbed* em números de nós sensores.

Para que seja possível integrar a rede real nativa do *testbed* com a rede virtual dos simuladores, determinamos a utilização de uma técnica de simulação híbrida proveniente de pesquisas militares na década de 1950, mas que atualmente está atribuída à implementações em diferentes áreas da tecnologia. Essa técnica de integração entre ambientes reais e virtuais, como já descrita na Seção 3.2.5, é conhecida como *Hardware-In-the-Loop* (HIL).

Ao integrar os dois ambientes (real e virtual), procuramos através da nossa abordagem manter uma infraestrutura transparente para o usuário, de forma que não exista restrições quanto ao nó sensor ser real ou virtual. Isso possibilita que o usuário carregue no nó virtual o mesmo binário que ele embarcaria em um nó real, resultando em uma maior fidelização dos experimentos.

Para uma melhor compreensão da nossa abordagem, tomaremos como exemplo a infraestrutura do *testbed* do projeto [CIA]^2/GT-TeI (Figura 4.1). É possível observar que trata-se de uma arquitetura tradicional de *testbeds*, ou seja, um servidor, que comporta o sistema de gerência do *testbed*, fornecendo componentes para que os usuários utilizem os recursos da rede de sensores. Considerando hipoteticamente a possibilidade de adicionar nós sensores virtuais à infraestrutura desse *testbed*, traçamos um panorama de como seria essa nova arquitetura. Assim, como mostrado na Figura 4.2, teríamos um novo elemento (simulador) no *testbed*, com interfaces de comunicação para o sistema de gerência e para a rede de nós sensores físicos. Isso possibilitaria o controle da rede física, da rede simulada, e da rede híbrida. Vale ressaltar que essa integração não foi realizada, tratando apenas de uma representação sistemática.

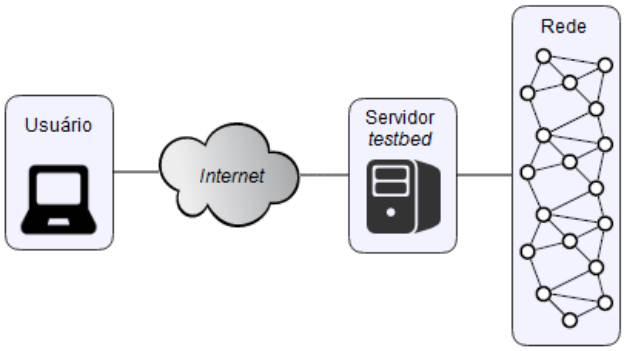


Figura 4.1: *Infraestrutura original do testbed.*

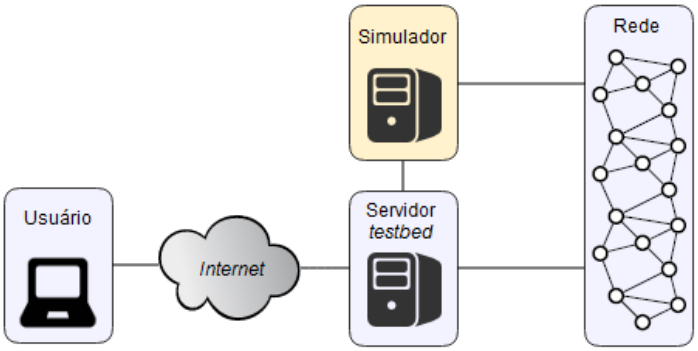


Figura 4.2: *Infraestrutura do testbed com adição do simulador híbrido.*

A expectativa da nossa proposta, com a abordagem que utilizamos (Figura 4.2), é que a partir do acréscimo no número de nós sensores do *testbed*, seja possível aos usuários compor cenários mais robustos que favoreçam pesquisas com aplicações para projetos de implementação em maiores escalas.

Para capacitar a nossa proposta a uma implementação real, entendemos que seria importante a produção de um conjunto de ferramentas funcionais, mesmo o desenvolvimento de uma ferramenta de simulação híbrida não ser o objetivo deste trabalho. Entre-

tanto, através dessas ferramentas seria possível identificar limitações e desbalizações que compõem a infraestrutura de uma rede híbrida.

As seções seguintes deste capítulo descrevem o modelo conceitual e a implementação do supracitado conjunto de ferramentas.

4.1 Abordando de Forma Conceitual o Empreendimento Implementado

Para que a proposta apresentada neste trabalho pudesse se mostrar funcional, elaboramos um projeto de um simulador híbrido, capaz de demonstrar de forma prática a aplicação da técnica *Hardware-In-the-Loop* na produção de um ambiente transparente, que integra nós sensores reais com nós sensores virtuais.

Na literatura é possível encontrar alguns trabalhos que apresentam ferramentas híbridas para testes com redes de sensores, como em [38,46,48,55,62,71,73]. Entretanto, devido às particularidades apresentadas nos propósitos de criação de cada uma dessas ferramentas (ver Seção 3.2.5) não serem os mesmos que desejamos (ver Seções 1.2 e 6.1), o desenvolvimento de uma ferramenta própria se deu necessário.

A priori, foi preciso fazer um levantamento de requisitos para identificar todos os elementos necessários para a concepção do nosso empreendimento. Inclusive, identificar as lacunas não preenchidas nos supracitados trabalhos encontrados na literatura, de modo a abordar soluções que pudessem atender a algumas peculiaridades presentes na nossa proposta.

Com a análise dos requisitos identificados, concluímos que precisaríamos basicamente de dois componentes: um simulador, que fosse possível emular os microcontroladores dos nós sensores reais, pois só assim atenderíamos ao requisito de transparência na infraestrutura híbrida; e um *hardware* dedicado, cuja capacidade permitisse essa integração entre o ambiente virtual do simulador e o ambiente real dos *testbeds* físicos. Entendam-se *testbeds* físicos, ambientes de teste para redes de sensores sem fio, compostos por nós sensores físicos.

Ao integrar o ambiente real do *testbed* físico com o ambiente virtual do simulador, objetivamos a concepção de um novo ambiente, agora híbrido, representando uma nova e ampla infraestrutura de testes para redes de sensores. A Figura 4.3 demonstra uma percepção visual dessa composição híbrida.

A seguir, adotamos uma abordagem *top-down* para descrevermos com mais detalhes a ferramenta de simulação híbrida desenvolvida para este trabalho, assim como os componentes que a constitui.

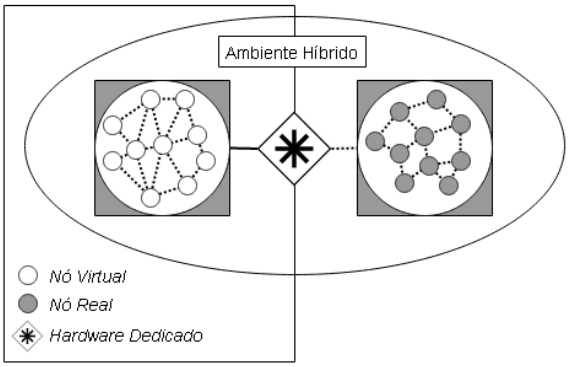


Figura 4.3: *Composição abstrata de um ambiente híbrido para teste.*

4.1.1 A Ferramenta de Simulação Híbrida HS_2N

Com a aplicação da técnica de simulação *Hardware-In-the-Loop*, integramos o ambiente virtual do nosso simulador ao ambiente real dos nós sensores físicos. O resultado foi uma ferramenta híbrida de simulação, cujos nós sensores virtuais passaram a ter totais capacidade de comunicação com os nós sensores reais. Compondo um único ambiente físico-simulado.

Denominado de Simulador Híbrido de Nó Sensor (ou *Hybrid Simulator of Sensor Node* - HS_2N), a nossa ferramenta de simulação instancia de maneira independente nós sensores virtuais, fornecendo uma interface de controle para cada nó virtual construído. A composição da rede virtual é através da comunicação por *sockets*, com a calibragem de atraso de pacote de acordo com o comportamento de experimento em rede real. Uma representação sistemática da nossa ferramenta HS_2N implantada em uma infraestrutura multiescalar pode ser vista na Figura 4.4.

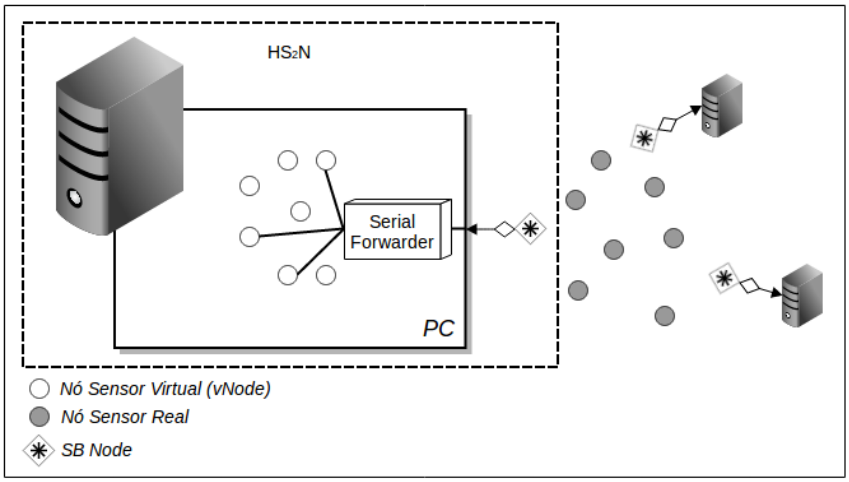


Figura 4.4: *Representação sistemática do simulador híbrido HS_2N implementado em uma infraestrutura multiescalar.*

Basicamente, o HS_2N é composto de apenas dois tipos de componentes: os

vNodes, que são instâncias de nós sensores virtuais; e o SB Node, que é o dispositivo de integração de ambientes. Entretanto, como pode ser visto na Figura 4.4, um outro elemento foi adicionado ao nosso empreendimento, que é a ferramenta de código aberto `SerialForwarder` [40], distribuída em conjunto com o pacote TinyOS.

Com a adição da ferramenta `SerialForwarder` ao nosso empreendimento, possibilitamos que múltiplos vNodes acessem de forma simultânea a porta serial conectada ao SB Node. No entanto, caso a porta serial seja acessível apenas a um único vNode, o uso da `SerialForwarder` se torna opcional.

Para compor os vNodes foram desenvolvidos duas ferramentas, o MSPSim-RX e o Cuke. O MSPSim-RX é o simulador responsável pela concepção e simulação dos nós sensores virtuais. Já o Cuke é o *front-end* encarregado de gerenciar o MSPSim-RX e fornecer as interfaces para controles externos. Ao instanciar um vNode o usuário na realidade estará usando o *front-end* Cuke para criar um novo nó sensor simulado no MSPSim-RX. Sendo que, cada instância do vNode é uma nova instância do Cuke e do MSPSim-RX.

As interações envolvendo os vNodes e os nós sensores reais decorrem a partir da manutenção do fluxo de dados promovido da seguinte forma. O SB Node captura os pacotes acessíveis ao seu dispositivo de rádio e os enviam para o `SerialForwarder` através da porta serial. O `SerialForwarder`, por sua vez, repassa os pacotes recebidos para os vNodes, que trata de aceitar ou rejeitar cada pacote. No processo inverso, os vNodes enviam pacotes para a `SerialForwarder`, que os repassam ao SB Node. Então, o SB Node se encarrega de difundir os pacotes pela rede real do testbed através do seu transceptor de rádio. A transferência de pacotes entre os nós sensores reais é realizada através de comunicação real, assim como os vNodes utilizam enlaces virtuais para se comunicarem.

Mais detalhes sobre os componentes vNode, SB Node, MSPSim-RX e Cuke são apresentados nas seções seguintes.

4.2 Abordando a Implementação do Empreendimento

Para uma melhor organização do processo de implementação do nosso simulador híbrido, quebramos o projeto em etapas gerenciáveis com a finalidade de otimizar todo o seu desenvolvimento. Após a definição das especificações, soubemos como o sistema deveria se comportar, no entanto, precisaríamos definir como cada componente seria construído. Isso nos permitiu, inclusive, conhecer detalhes internos da infraestrutura de cada um desses componentes.

O primeiro componente desenvolvido foi o MSPSim-RX, que consistiu em aplicar alterações pontuais no simulador MSPSim, permitindo sobretudo a transladação dos

frames de rádio para dentro e para fora do simulador. Em seguida, desenvolvemos o Cuke, *front-end* dos nós virtuais, implementado para gerir o MSPSim-RX e fornecer as interfaces para comunicações externas. Por último, idealizamos o componente SB Node, arquitetura *software-hardware* responsável pela integração entre os ambientes real e virtual. Lembrando que, sobre a concepção do SB Node, coube-nos apenas o desenvolvimento do *software*, uma vez que a plataforma de *hardware* fora empregada a partir do uso de um mote do tipo TelosB. Nas seções seguintes são apresentados detalhes sobre o desenvolvimento de cada um desses componentes.

4.2.1 O Simulador MSPSim-RX

Para a aquisição da ferramenta de simulação que irá conceber os nós sensores virtuais presentes na infraestrutura do nosso simulador híbrido HS_2N, três condições foram pré-avaliadas:

a O não-desenvolvimento de um simulador: possível a partir da identificação de uma ferramenta já existente, que fosse capaz de atender às nossas necessidade;
b O desenvolvimento completo de um simulador: em que seria elaborado e executado um projeto para o desenvolvimento de uma ferramenta de simulação a partir do zero;
c O desenvolvimento parcial de um simulador: que seria o desenvolvimento de um simulador a partir de uma ferramenta de simulação de código aberto já existente.

Durante o processo de avaliação, prontamente descartamos as condições estipuladas pelas supracitadas opções (a) e (b). Pois, não fora encontrado na literatura científica, nem ao menos entre os fabricantes de *software*, qualquer ferramenta de simulação com características precisas quanto às nossas necessidades. Ao mesmo tempo, concluímos que o desenvolvimento completo de um simulador demandaria uma alta complexidade, além de grande custo temporal e de mão de obra, fatores que favoreceu o descarte, por julgarmos serem inviáveis para o programa de estudo presente.

Como alternativa final, adotamos a condição apresentada na supracitada opção (c), relacionada ao desenvolvimento de um simulador a partir de uma ferramenta de simulação já existente. Essa ferramenta, definida como sendo o simulador MSPSim, e cujo o processo de seleção é descrito a seguir, é tratada no decorrer deste documento com a denominação de “simulador base”.

Para a seleção do simulador base adotado para o nosso projeto, analisamos alguns pré-requisitos que julgamos serem essenciais para o que pleiteamos. A seguir, listamos e detalhamos cada um desses quesitos em ordem de prioridade:

1. Alta-fidelidade de tempos de execução: a ferramenta deveria ser capaz de emular o microcontrolador, de forma a simular o tempo de CPU do nó sensor virtual o

mais próximo possível do tempo de CPU de um nó sensor real. Isso permite rodar experimentos com um alto grau de fidelidade de execução, reforçando a premissa de um ambiente transparente para o usuário.

2. Heterogeneidade de sistema operacional: o simulador deveria suportar simulações de diferentes plataformas de sistemas operacionais para redes de sensores (*e.g.*, TinyOS, Contiki). Os sistema operacionais para redes de sensores são embarcados junto com os *softwares* de gerenciamento do nó sensor.
3. Diversidade de plataformas de nós sensores: o simulador deveria ter diferentes plataformas de nós sensores já desenvolvidas (*e.g.*, TelosB, MICAz, SHIMMER). Isso possibilita uma considerável economia no desenvolvimento de novas plataformas, a fim de tornar a ferramenta mais genérica possível.
4. Comunidade ativa: a ferramenta não deveria estar com o projeto descontinuado e com comunidades de discussão abandonadas. Isso poderia causar transtornos desnecessários, tanto no processo de desenvolvimento do nosso simulador, como em processos futuros de atualização.

Conhecendo as características e peculiaridades de cada simulador e ferramenta de simulação estudada (ver Seções 3.2.1, 3.2.2, 3.2.3 e 3.2.4), e a partir da análise da Tabela 4.1, identificamos o MSPSim como sendo a ferramenta que julgamos ser mais adequada para servir de simulador base para o nosso simulador resultante. Uma descrição mais detalhada sobre o processo de seleção do simulador base é apresentada no Apêndice A.

Tabela 4.1: *Características dos principais simuladores em: nível de sistema operacional, nível de conjunto de instrução de ciclo preciso e integração multinível.*

		Usuários							**Desenvolvedores**	
Simuladores		Simulação	GUI	RTOS	Motes	MCU	Rádio	Entrada	Open Source	Linguagem
SO	TOSSIM	Evento Discreto	Não	TinyOS	MicaZ	AVR	CC2420	NesC	Sim	C
	COOJA	Evento Discreto	Sim	Contiki	Cooja, Minimal-Net, Native	Native	Radio-Drive	C	Sim	Java
Hardware	ATEMU	Evento Discreto	Sim	Genérico	Mica2	AVR	CC1000	ELF, SREC	Sim	C
	Avrora	Evento Discreto	Sim	Genérico	Mica2, MicaZ	AVR	CC1000, CC2420	ASM, ELF, GAS, OD	Sim	Java
	MSPSim	Evento Discreto	Sim	Genérico	ESB, EXP5438, Sentilla-JCreate, Sentilla-USB, Sky, WiSMote, Z1	MSP430, MSP430x	CC2420, CC2520, TR1001	ELF, IHEX	Sim	Java
Integração	COOJA/MSPSim /AvroraZ	Evento Discreto	Sim	Genérico	Cooja, ESB, EXP5438, Mica2, MicaZ, Minimal-Net, Native, Sentilla-JCreate, Sentilla-USB, Sky, WiSMote, Z1	AVR, MSP430, MSP430x, Native	Radio-Drive, CC1000, CC2420, CC2520, TR1001	ASM, C, ELF, GAS, IHEX, OD	Sim	Java

Após consulta a Tabela 4.1, algumas considerações devem ser destacadas:

1. Como pode ser observado, os simuladores que executam em nível de sistema operacional têm como entrada arquivos não compilados, isso acontece porque a compilação é realizada no próprio simulador através de ferramentas de compilação adaptadas. Essa abordagem é uma característica comum a esse tipo de simulador que, como já descrito na Seção 3.2.2, abstrai a interface de programação do *hardware* adequando o *software* ao simulador.
2. Em uma análise de simulações em nível de sistema operacional, o TOSSIM apresenta maior precisão que o COOJA [66], contudo, é bastante limitado quanto à plataforma de nó sensor suportada (apenas MICAz). Já o COOJA, proporciona uma abordagem de mais alto nível, mas com suporte nativo às simulações multiníveis e interface de integração, o que o torna consideravelmente extensível.
3. Os simuladores que executam em nível de conjunto de instrução de ciclo preciso, ou seja, em nível de *hardware*, já não são dependentes do sistema operacional embarcado no binário de entrada, isso possibilita que eles executem qualquer tipo de compilação que seja capaz de gerar os arquivos adequados para redes de sensores, uma vez que esse tipo de simulador é adaptado para executar apenas código de máquina.
4. No simulador integrado, ou seja, composto por mais de um simulador, o universo de plataformas suportadas, sejam elas relacionadas a nó sensor, microcontrolador e *etc.*, aumenta consideravelmente, uma vez que são aproveitadas todas as plataformas já desenvolvidas para ambos os simuladores.
5. Como atualmente a distribuição do COOJA (pacote Contiki) já conta com a integração dos simuladores MSPSim e Avrora (com a modificação AvroraZ), foi considerada neste documento a mesma abordagem.

Após um aprofundado estudo sobre o simulador MSPSim, e aplicada as devidas alterações no código-fonte da ferramenta, concluímos o nosso simulador resultante, cuja versão final denominamos de MSPSim Radio eXtended, ou simplesmente MSPSim-RX. As modificações aplicadas ao MSPSim para o desenvolvimento do nosso simulador MSPSim-RX são apresentadas a seguir.

Dadas as inúmeras alterações realizadas no simulador base MSPSim, preferimos resumir a organização do processo de desenvolvimento do componente MSPSim-RX em 4 atividades básicas: a adequação de comandos, a canalização de tráfego, o tratamento de pacotes e a calibragem de rede.

Adequação de comandos

Nativamente, o simulador base MSPSim fornece uma vasta série de comandos para a gerência do nó sensor simulado. Contudo, entendemos que era necessário adicionar

alguns comandos extras para adequar o simulador, e algumas rotinas desenvolvidas, ao nosso empreendimento.

Assim, algumas alterações pontuais foram realizadas, adequando interfaces de comunicação ao *front-end* Cuke. Dentre elas, podemos destacar a passagem de parâmetros para comunicações externas, poder de encerramento do sistema pela interface gráfica, e manipulação do identificador dos nós sensores. Todas devidamente identificadas no código-fonte do MSPSim-RX.

Canalização de tráfego

Consideramos que essa foi a mais importante alteração realizada no simulador base MSPSim, pois é nesse estágio que os pacotes são extraídos e injetados de/para o meio externo do simulador. A identificação do local e momento de transladação dos pacotes foi possível a partir de exaustivo estudo do código-fonte do MSPSim. Uma vez que a manipulação dos dados em um local e momento inapropriado poderia comprometer a integridade do nó e/ou da rede simulada.

Para efeito de demonstração, apresentamos na Figura 4.5 uma representação da atividade do chip de rádio TI Chipcon CC2420 [35], modelado pelo simulador, identificando o exato momento em que o *frame* de rádio é capturado. Essa etapa é preliminar às funções de simulação do meio, evitando assim atrasos significativos na comunicação entre os transceptores simulados e físico. Igualmente, em um processo de atividade inverso (injeção de *frames*), o meio de rádio simulado também será evitado.

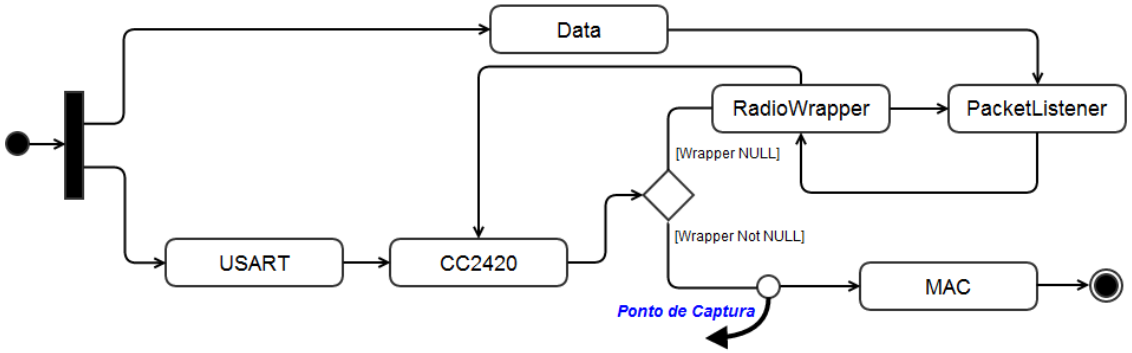


Figura 4.5: *Diagrama de atividades do rádio CC2420 modelado no MSPSim.*

O processo de injeção de dados no simulador obedece às seguintes premissas. Utilizando interfaces do SDK Java do TinyOS para mediações de pacotes, o fluxo de *bytes* originário do meio externo é canalizado para dentro do simulador. Cada pacote é então encapsulado em um invólucro de rádio e inserido na pilha de rádio de cada vNode acessível. A acessibilidade do vNode à comunicações externas é determinada pelo usuário durante a configuração do nó.

No decurso do supracitado processo de injeção de dados, informações que alimentam a rede virtual, como *Received Signal Strength Indicator* (RSSI) e *Link Quality Indicator* (LQI), são transferidas. Assim como são tratados os dados trafegados, adequando o *frame* de origem serial para o rádio. O Código 4.1 mostra a rotina que representa a injeção dos dados no simulador.

Código 4.1 *Rotina de injeção de pacote no simulador.*

```
1  final RadioWrapper radioWrapper = new RadioWrapper(radio);
2  super.sfw.registerPacketListener(new PacketListenerIF() {
3    @Override
4    public void packetReceived(byte[] bytes) {
5      int rssi = bytes[bytes.length - 2] & 0xff;
6      int lqi = bytes[bytes.length - 1] & 0xff;
7      radio.setRSSI(rssi);
8      radio.setLQI(lqi);
9      byte[] packet = PacketConverter.fromSerialToRadio(bytes);
10     radioWrapper.packetReceived(packet);
11   }
12 });
```

O processo de extração dos dados do simulador para o meio externo se dá a partir da canalização do fluxo de *bytes*, proveniente de invólucros de rádio, cujo *frame* embutido é adaptado para comunicações através de porta serial. Veja a rotina de extração de dados representada no Código 4.2.

Código 4.2 *Rotina de extração de pacote do simulador.*

```
1  radioWrapper.addPacketListener(new PacketListener() {
2    @Override
3    public void transmissionStarted() {
4    }
5    @Override
6    public void transmissionEnded(byte[] receivedData) {
7      byte[] packet = PacketConverter.fromRadioToSerial(receivedData);
8      try {
9        sfw.writePacket(packet);
10     } catch (IOException ex) {
11       System.err.println("Writing in serial: bad packet");
12     }
13   }
14 });
```

Tratamento de pacotes

O canal de comunicação entre os ambientes real e virtual apresenta duas interfaces distintas em suas extremidades. A primeira, refere-se a interface de rádio do nó sensor virtual. A segunda, trata-se da interface serial do SB Node, anexada ao PC *host*. Ambas apresentam independentes estruturas de *frames*. Assim, como abordado durante a descrição da atividade de canalização de tráfego, uma ação de tratamento de dados se faz necessária.

Para convertermos o pacote serial em um pacote de rádio, escrevemos um bloco de código, incorporado no método `fromSerialToRadio`, que recebe um vetor de *bytes*, contendo o pacote originado da interface serial do SB Node, e após algumas alterações aplicadas retorna um pacote apto a ser inserido na pilha de rádio do nó sensor virtual. A Figura 4.6 mostra um exemplo de um pacote serial convertido para um pacote de rádio.

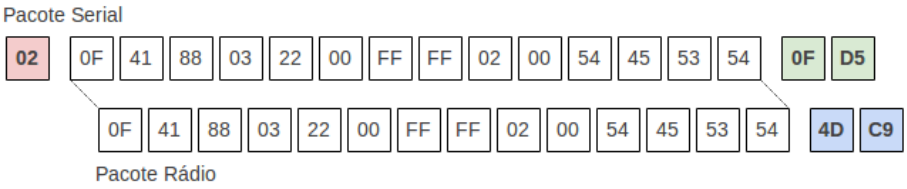


Figura 4.6: *Conversão de um pacote serial para um pacote de rádio.*

A partir da Figura 4.6 é possível observar que o primeiro octeto do *frame* serial é removido. Isso ocorre porque os *frames* originados da interface serial do SB Node, contém um identificado do tipo de mensagem que está sendo encaminhada. Nesse caso específico, o valor hexadecimal encontrado é “02”, que indica que o pacote é estruturado conforme o padrão IEEE 802.15.4 [61].

Outra peculiaridade abordada, trata-se dos dois últimos octetos que são adicionados ao *frame* de rádio resultante. Eles referem-se ao campo *Frame Check Sequence* (FCS), do rodapé do *frame* de rádio (MAC *Footer* - MFR). Esse mecanismo é implementado empregando um *Cyclic Redundancy Check* (CRC) de 16 *bits*, estruturado de acordo o *International Telecommunication Union – Telecommunication Standardization Sector* (ITU-T), e usado para proteger o *frame* a partir de detecções de erros de *bits*. O FCS é calculado usando o seguinte gerador polinomial de grau 16 (Equação 4-1) [61].

$$G_{16}(x) = x^{16} + x^{12} + x^5 + 1 \tag{4-1}$$

Vale ressaltar que cada pacote gera um novo CRC. Em uma analogia as funções *hash*, o cálculo do CRC mapeia dados de comprimento variável em dados de comprimento fixo. Para isso é considerado o cabeçalho (MAC *Header* - MHR) e a carga útil (MAC

payload) do *frame*. A concepção do FCS, a partir do cálculo do CRC, decorre com a aplicação do Algoritmo 4.1 [61], baseado no supracitado polinômio.

Algoritmo 4.1: *FCS field*

– Considerando $M(x) = b_0x^{k-1} + b_1x^{k-2} + ... + b_{k-2}x + b_{k-1}$ o polinômio que representa a sequência de *bits* para o qual o *checksum* deva ser computado.

– Multiplique $M(x)$ por x^{16}, dado o polinômio $x^{16} \times M(x)$.

– Divida $x^{16} \times M(x)$ módulo 2 pelo polinômio gerador, $G_{16}(x)$, para obter o polinômio resultante, $R(x) = r_0x^{15} + r_1x^{14} + ... + r_{14}x + r_{15}$.

– O campo FCS é dado pelo coeficiente resultante.

O processo inverso, ou seja, a conversão dos *frames* de rádio em *frames* seriais também foi desenvolvido. Pertencendo essa rotina ao método `fromRadioToSerial`. Ambos os métodos, `fromSerialToRadio` e `fromRadioToSerial`, fazem parte do conjunto de rotinas da classe `PacketConverter`, que desenvolvemos e incorporamos ao código-fonte do MSPSim-RX.

Na camada física, os pacotes da pilha de comunicação serial contêm alguns preâmbulos que indicam início e fim do pacote, mas esses valores são tratados pelo SDK Java do TinyOS. Assim, consideramos aqui apenas a composição dos *frames* adaptados a subcamada MAC.

Calibragem de rede

A princípio, planejamos o nosso empreendimento para trabalhar de maneira prioritária com comunicações reais. Assim, a ideia de uma rede virtual autônoma seria apenas uma função opcional do nosso simulador híbrido. Dessa forma, esperaríamos que comunicações entre os nós sensores virtuais se desse a partir de um agente físico intermediário, que nesse caso seria o SB Node. Assim, se um nó sensor virtual pretendesse enviar uma mensagem para outro nó sensor virtual, essa mensagem seria primeiramente enviada para o SB Node, e em seguida retransmitida ao segundo nó sensor virtual (ou nó sensor virtual de destino).

Pretenderíamos, a partir da supracitada ação, utilizar apenas o rádio real para perturbações na rede. Procurando assim atender a condições de maior fidelidade na comunicação entres os nós sensores, uma vez que apesar do MSPSim-RX emular uma grande variedade de componentes, a rede interna é completamente simulada. No entanto, devido alguns problemas apresentados durante a execução de *benchmarks*, percebemos que se tornaria inviável abordar toda a comunicação entre nós sensores virtuais dessa forma.

Dentre os problemas encontrados, destaque para incapacidade de obter valores de *hardware* dos metadados nos pacotes que atingiam o rádio do SB Node, sendo dessa forma necessário simular alguns valores, tais como do RSSI e LQI. À vista disso, precisando simular os valores dos metadados, então que esses fossem feitos no simulador. Em contrapartida, diminuiríamos a sobrecarga no porta serial do SB Node, que é o ponto crítico da nossa infraestrutura.

Por outro lado, consideramos que seria importante fornecer comunicações virtuais que atendessem o mínimo possível de variantes encontradas no comportamento de uma comunicação real. Assim, aplicamos alguns testes sobre a rede virtual do nosso simulador híbrido e constatamos que em média uma comunicação entre dois nós sensores virtuais apresentam um *delay* de aproximadamente 0,3 ms (ver Seção 5.1.1). Ao mesmo tempo, realizamos alguns testes com transmissões por comunicação entre nós sensores reais e encontramos valores de *delays* para diferentes distâncias entre os nós (ver Seção 5.1.2), mas todos com *delay* de mais de 10 ms. Baseado nessas informações, desenvolvemos um mecanismo para tratar as comunicações da rede virtual com mais fidelidade do que as apresentadas nativamente pelo simulador.

É conhecido que o comportamento de uma comunicação sem fio entre nós sensores não é baseada apenas no atraso causado na entrega de pacotes. Contudo, devido à complexidade exigida para a modelagem de uma comunicação virtual de alta fidelidade, o que demandaria um trabalho extra para tal atividade, resolvemos considerar apenas algumas poucas variáveis para o comportamento da rede virtual do nosso simulador. Assim, desenvolvemos um mecanismo que a partir de valores definidos pelo o usuário é gerado projeções de *delays* para comunicações entre os nós sensores virtuais. O nosso mecanismo encontra-se na classe que desenvolvemos para o MSPSim-RX, denominada `NetworkCalibrator`.

4.2.2 O *Front-end* Cuke e a Concepção dos Nós Virtuais (vNode)

O MSPSim-RX é uma versão modificada do simulador base MSPSim. Contudo, o MSPSim-RX por si só executa as mesmas ações que nativamente estão estipuladas ao simulador base MSPSim. Sem qualquer função adicional. Portanto, para que os usuários possam utilizar das funções adicionais que desenvolvemos, tais como a canalização de tráfego para dentro e para fora do simulador, implantamos um novo *front-end* anexado ao MSPSim-RX. A essa ferramenta denominamos de Cuke.

O *front-end* Cuke tem a finalidade de fornecer interfaces de controle a cada um dos nós sensores virtuais instanciado com o MSPSim-RX. Apenas com o Cuke é possível criar e gerenciar os nós sensores virtuais. Assim, consideramos cada nó sensor virtual (vNode) do nosso simulador híbrido (HS_2N) uma integração das ferramentas

MSPSim-RX e Cuke. Veja a Figura 4.7 para uma melhor compreensão.

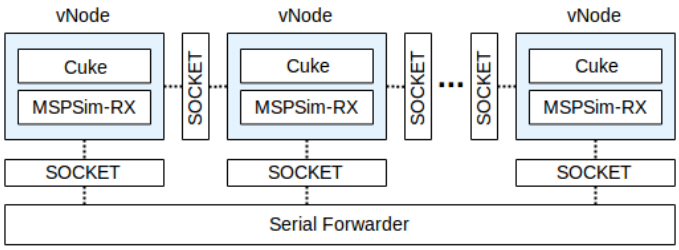


Figura 4.7: *Composição representativa dos nós sensores virtuais no HS_2N.*

Conforme visto na Figura 4.7, cada vNode é instanciado de forma independente, podendo um único nó parar de responder sem que cause pane nos demais vNodes. Também pode ser observado na figura que toda comunicação entre os vNodes, e dos vNodes com o `SerialForwarder` é realizada através de *sockets*. O comportamento da rede é modelado em cada nó sensor, com projeções de atraso nas comunicações entre os nós.

A partir do Cuke é possível gerenciar o nó sensor virtual por scripts ou através do uso de consoles. Com o uso das interfaces de gerenciamento via console é possível instrumentar as ferramentas digitais desenvolvidas para o simulador base MSPSim. Sendo esse, outro grande benefício de se instanciar cada nó sensor de maineira independente.

O *front-end* Cuke é responsável por fornecer as interfaces de controle para cada nó sensor virtual, sendo algumas dessas portadas do nosso simulador MSPSim-RX. Ao mesmo tempo é fornecido também as interfaces de acesso remoto do nó sensor virtual, através da utilização de bibliotecas específicas pertencentes ao SDK Java do TinyOS.

O controle do nó sensor virtual é abordado em duas etapas. A primeira, diz respeito a definição de parâmetros para a composição do nó sensor virtual (*e.g.*, plataforma, id, *firmware*). A segunda, refere-se aos comandos e ferramentas utilizadas para controlar e avaliar o comportamento do nó.

Sobre os parâmetros, passados por argumento, de pré-configuração do nó sensor virtual, destacamos o comando "comm", que possibilita definir o canal de comunicação externo ao nó sensor virtual, que pode ser uma porta USB ou um `SerialForwarder`, ambos local ou remoto; o comando "*network*", que torna possível ao usuário colocar o nó sensor na rede virtual, ou isolá-lo; o comando "platform", usado para definir a plataforma de nó sensor a ser utilizada (*e.g.*, TelosB, TmoteSky); e o comando "*firmware*", que comportará o caminho do binário da aplicação que o usuário deseja carregar no nó sensor.

Outra opção de pré-configuração do nó sensor virtual, refere-se a utilização de scripts com a passagem de comandos nativos do MSPSim. A utilização desses scripts permite, inclusive, controlar o comportamento do nó sensor, até mesmo de maneira

periódica, ou seja, o usuário pode definir que seja executado um determinado comando de tempos em tempos, ou mesmo ligar e desligar o nó sensor, simulando erros na rede.

Para manipulações posteriores do nó sensor virtual, é fornecido uma interface gráfica, junto com um *prompt* de comando, que permite ao usuário carregar ferramentas de monitoramento e acessar qualquer comando nativos do MSPSim. Inclusive, visualizar o comportamento dos LEDs de cada nó sensor virtual. A Figura 4.8 mostra uma impressão de tela do nosso sistema.

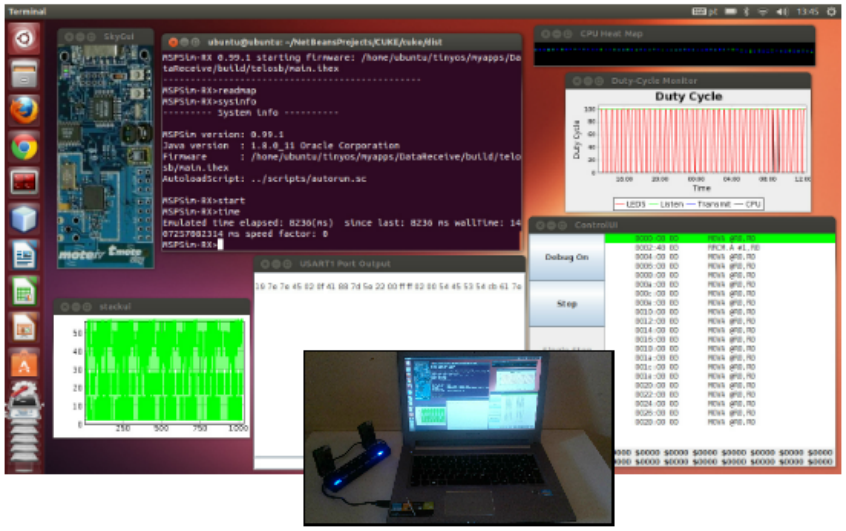


Figura 4.8: *Tela do HS_2N em execução.*

4.2.3 O Dispositivo SB Node

Para permitir a integração entre os dois ambientes (real e virtual) que compõem o nosso simulador híbrido, desenvolvemos um dispositivo para agir em modo promísculo[1], capturando todos os pacotes provenientes da rede física do *testbed*, e repassando-os para a rede virtual do simulador. Simultaneamente é executado o processo inverso, ou seja, do simulador para o *testbed*. A esse dispositivo chamamos de SB Node.

SB Node é uma amistosa denominação concedida ao nosso dispositivo de integração, no qual se trata de uma referência as atribuições e características do equipamento, que é: agir como um farejador (*Sniffer*) de rede, capturando todos os pacotes acessíveis à sua interface de rádio; e servir como uma ponte (*Bridge*) entre o ambiente real do *testbed* físico e o ambiente virtual do simulador.

Para a execução de suas atribuições, prontamente identificamos que o SB Node deveria atender aos seguintes pré-requisitos: (a) disponibilizar uma interface de comuni-

[1] O modo promíscuo capacita a pilha de rádio do SB Node a receber qualquer pacote para tratamento, mesmo o equipamento em questão não ser o destinatário do pacote.

cação via rádio frequência, para permitir comunicações com o *testbed* físico; (b) disponibilizar de uma interface de comunicação via serial, para permitir comunicações com o PC que hospeda os nós virtuais; e (c) disponibilizar mecanismos gerenciáveis, para favorecer o controle do tráfego de pacotes.

Devido à nossa disponibilidade de acesso à aquisição de equipamentos, e considerando que um nó sensor, alternativamente, pode atender aos supracitados pré-requisitos listados, passamos então a cogitar o desenvolvimento de um *software* para embarcarmos em um nó sensor real, com a finalidade de gerenciar toda a comunicação entre o *testbed* físico e o simulador.

Atualmente, disponibilizamos nas unidades do Instituto de Informática da UFG dois tipos de nós sensores, Crossbow Technology TelosB e Crossbow Technology MICAz. Para compor as funcionalidades do SB Node, utilizamos nós sensores do tipo TelosB, exclusivamente por apresentarem maior capacidade de armazenamento (memória RAM) que os nós sensores do tipo MICAz, requisito essencial para o tipo de aplicação que carrega.

A aplicação em questão, ou seja, embarcada no nó sensor TelosB que compõe o SB Node, foi desenvolvida a partir do uso da linguagem de programação orientada a evento, conhecida como nesC [36, 37], que utiliza componentes provenientes do sistema operacional para redes de sensores, TinyOS. Na Figura 4.9 pode ser visto uma representação de como é montado o SB Node.

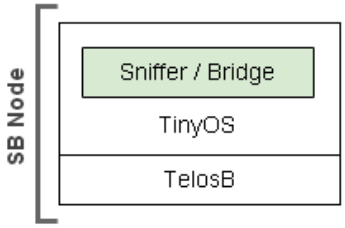


Figura 4.9: *Arquitetura do componente SB Node.*

Apesar dos dispositivos constituintes no TelosB atenderem aos requisitos necessários para o desenvolvimento do SB Node, certamente o equipamento não é o mais adequado, dada algumas limitações identificadas, como por exemplo, do gargalo encontrado no dispositivo serial que embarca o equipamento (ver Seção 5.4). No entanto, uma vez que o desenvolvimento de um *hardware* específico demandaria uma alta complexidade, resultando em uma fuga do custo temporal disponível para esta pesquisa, entendemos que a utilização do *hardware* encontrado no TelosB tenha sido a melhor das alternativas.

Para capturar todos os pacotes que se encontravam no raio de alcance do dispositivo de rádio do SB Node, tivemos que desenvolver uma aplicação para tornar a ferramenta análoga a um *sniffer* de rede, ou seja, ser capaz de capturar até mesmo os pacotes que não tivesse o SB Node como destino final. Para isso, foi preciso configurar o

rádio para funcionar em modo promíscuo, permitindo que a análise de pacote não fosse feito pelo *hardware* do rádio, e sim por nossa aplicação.

Com os pacotes em poder da nossa aplicação, podemos manipulá-los ou mesmo descartá-los, caso identificado que a transmissão para a porta serial seja desnecessária, aliviando assim a sobrecarga na saída serial do dispositivo. Essa seleção de pacote é realizada pela rotina implementada na função `checkPermission`, que lê endereços formatados em *big-endian order*, e cujo suporte é definido em tempo de compilação. A rotina que representa a função `checkPermission` pode ser vista no Código 4.3.

Código 4.3 *Função checkPermission do SB Node.*

```
1  bool checkPermission(message_t* ONE msg) {
2    uint8_t i;
3    uint8_t ARRAY_SIZE;
4    cc2420_header_t* header;
5    uint16_t dest_addr;
6
7    ARRAY_SIZE = sizeof(dest_addr_enabled)/sizeof(dest_addr_enabled[0]);
8    header = getHeader(msg);
9    dest_addr = (uint16_t) header->dest;
10
11   for (i=0; i<ARRAY_SIZE && dest_addr_enabled[i]!=dest_addr; i++);
12
13   return i < ARRAY_SIZE;
14 }
```

A política de descarte aplicada pela função `checkPermission` baseia-se na leitura do *array* `dest_addr_enable`, que comporta os endereços IP dos nós sensores virtuais. Assim, a partir da análise do `dest_addr_enable` é definido se o pacote é descartado ou encaminhado para a rede virtual. Sendo o encaminhamento permitido apenas para os pacotes cujo o endereço de destino esteja presente no `dest_addr_enable`.

Suplementar ao supracitado mecanismo de filtro de pacotes, adicionamos uma função que permite ativar/desativar o filtro em tempo real. Utilizando para isso o *user button* do TelosB. Essa ação fora previamente definida para ser executada por uma interrupção de *hardware*, sinalizada em um evento assíncrono.

A rotina que trata a multiplexação dos pacotes entre o rádio e a porta serial foi implementada com base na aplicação `BaseStation`, presente no pacote TinyOS. A aplicação `BaseStation` utiliza estruturas de dados do tipo fila para controle de fluxo, tratando de maneira harmoniosa os picos de carga. Adicionalmente, aprimoramos a aplicação empregando algumas alterações pontuais, como por exemplo o suporte ao

transporte de mensagens tanto sobre a camada *Active Message* [70] quanto sobre a abstração IEEE 802.15.4 [61], tornando a nossa ferramenta mais genérica possível. Originalmente, a aplicação `BaseStation` apresenta suporte apenas a transmissões que utilizam *active messages*.

Complementar ao nosso *software*, configuramos o rádio CC2420 do TelosB para operar constantemente em potência máxima de 0dBm. Uma vez que SB Node não é alimentado por baterias, entendemos que não existe conflito de escolha entre disponibilidade e consumo. Adicionalmente, definimos a utilização inicial do canal 26 para transmissões de rádio, considerando-o como o canal com melhor desempenho, conforme mostrado em testes realizados em [25]. Mecanismos de aprimoramento de seleção de canal não foi desenvolvido, por abranger estudos que fogem ao escopo deste trabalho. No entanto, indicamos os trabalhos [29, 51, 67], deixando a questão em aberto e condicionada ao usuário, caso deseje melhorar o desempenho da comunicação de rádio.

Demais configurações do rádio CC2420 podem ser adicionadas ao arquivo Makefile para definição em tempo de compilação, como por exemplo, a alteração no valor da constante global `RECEIVE_HISTORY_SIZE`, que adequadamente deve conter a quantidade de nós sensores reais vizinhos ao SB Node. O valor padrão é 4.

CAPÍTULO 5

Avaliação do Simulador Proposto

Apresentamos neste capítulo as avaliações realizadas sobre o conjunto de componentes implementados para a concepção do simulador híbrido HS_2N. O objetivo é analisar o comportamento da infraestrutura, procurando dar consistência ao empreendimento. Assim como, demonstrar o quão válida é a aplicação do conceito *Hardware-In-the-Loop* em ambientes de testes para redes de sensores sem fio.

A primeira série de testes realizada associou-se a etapa de desenvolvimento do simulador. A partir da análise dos resultados obtidos foi possível definir parâmetros para o aprimoramento da ferramenta de simulação. Em seguida, avaliamos a manutenção da integridade dos dados, de modo a garantir a fidelidade das informações trafegadas entre os ambientes real e virtual. O terceiro teste, abordou o levantamento de informações sobre o desempenho e escalabilidade do simulador híbrido, materializando assim o custo operacional do sistema. Por último, foi aplicado uma série de testes para avaliar o comportamento da porta serial do SB Node, uma vez que trata-se de um componente susceptível a alta concentração de carga.

5.1 Avaliando a Rede Virtual

Antes de definirmos a elaboração de um mecanismo de calibragem da rede virtual, comparamos o atraso na entrega de pacotes entre dois nós sensores virtuais, com o atraso na entrega de pacotes entre dois nós sensores reais. Identificado uma considerável disparidade nos resultados obtidos, decidimos desenvolver um mecanismo para calibrar a rede virtual, proporcionando comunicações com maior grau de fidelidade.

Nas subseções seguintes, descrevemos detalhes sobre a abordagem aplicada na realização dos testes de atraso, assim como a análise dos resultados obtidos.

5.1.1 Atraso Entre Nós Sensores Virtuais

O atraso unidirecional entre dois nós sensores virtuais foi calculado a partir de valores obtidos em transmissões sobre o enlace virtual do simulador HS_2N. Durante os

testes foram utilizadas duas aplicações distintas, embarcadas em plataformas virtuais de *motes* do tipo TelosB. A primeira aplicação envia pacotes por *broadcast* a uma frequência de 4 Hz, que equivale a aproximadamente 250 milissegundos. A segunda aplicação apenas está configurada para receber os pacotes, sem gerar qualquer estímulo na rede. Os valores de atrasos são auferidos a partir da diferença entre os tempos de entrada e saída de cada pacote. O resultado obtido do teste é apresentado na Figura 5.1.

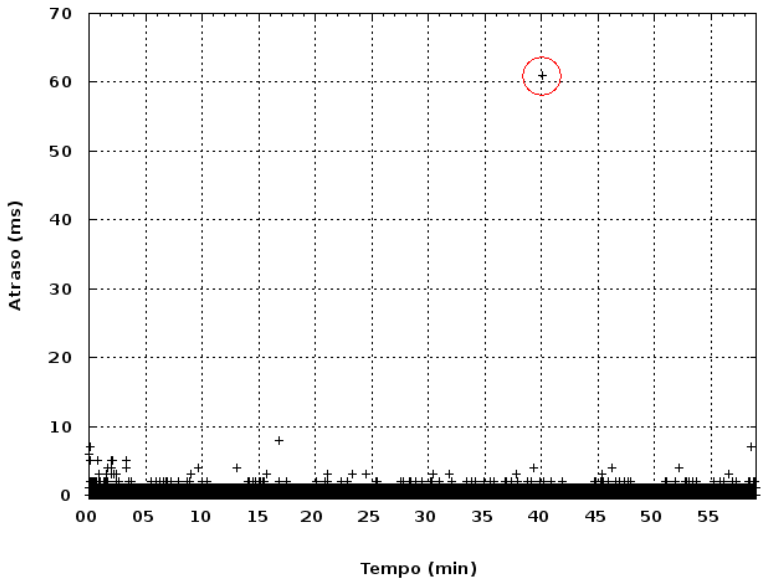


Figura 5.1: *Atraso unidirecional aferido entre dois nós sensores virtuais.*

Com base na análise da Figura 5.1 é possível identificar que os pacotes levam entre 0 a 3 milissegundos para serem entregues a outro nó sensor na rede virtual. Devido às variações apresentadas, algumas provavelmente relacionadas ao comportamento do PC *host*, chegamos ao valor médio de 0,28 milissegundos para o atraso unidirecional entre dois nós sensores virtuais.

5.1.2 Atraso Entre Nós Sensores Reais

Para a realização dos testes de atraso entre nós sensores reais foram utilizados dois *motes* do tipo TelosB. O primeiro é encarregado de enviar pacotes a uma frequência periódica de 4 Hz (*aprox.* 250 ms), com tempos de ocorrência de eventos anexados. O segundo *mote* recebe os pacotes, adiciona o tempo de recebimento, e reenvia-o para uma estação base conectada a um PC. Porém para realizar esse teste alguns parâmetros tiveram que ser considerados. O primeiro, relacionado aos temporizados dos *motes*. O segundo, associado aos tempos carimbados (*timestamps*) nos pacotes.

Diferente dos testes de atraso utilizando o simulador, em que pode-se usar o mesmo temporizador para ambos os *motes*. Com *motes* reais isso já não é possível. Pois

além de ambos estarem fisicamente isolados, torna-se praticamente impossível garantir que os dois temporizadores sejam energizados ao mesmo tempo. Assim, é preciso que haja algum mecanismo que possibilite manter os tempos fornecidos pelos temporizadores dos dois *motes* com valores sincronizados. Para resolver esse problema, utilizamos o componente `TimeSyncMessageC` sugerido pelas *TinyOS Enhancement Proposals* (TEP) número 133 [50]. A TEP 133 fornece a descrição de um mecanismo de sincronização de tempo em nível de pacotes para aplicações desenvolvidas sobre o sistema operacional TinyOS.

Também sugerido pela TEP 133, tivemos que considerar o uso de estratégias de verificação de tempo para validar os *timestamps* presentes nos pacotes. Essa verificação é necessária porque em certas circunstâncias (não descritas na TEP 133) o pacote recebido pode não ter sido devidamente carimbado, passando assim informações irreais. O que pode causar divergências nos resultados obtidos.

Com a infraestrutura de teste devidamente preparada para mensurar os atrasos unidirecionais na comunicação entre dois nós sensores reais, executamos os experimentos considerando duas distâncias entre os *motes*, 1 e 5 metros. Os valores obtidos de atraso descendem da diferença de tempos de entrada e saída de cada pacote. Os resultados dos testes são apresentados nas Figuras 5.2 e 5.3.

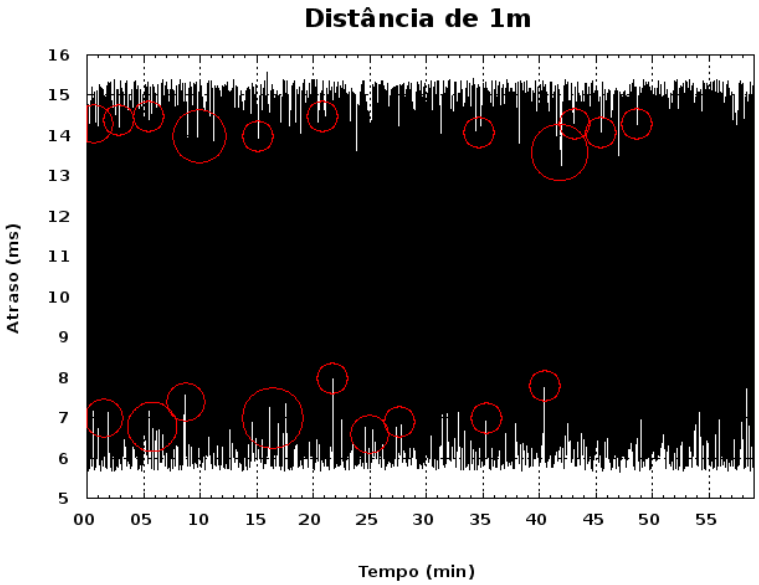


Figura 5.2: *Atraso unidirecional aferido entre dois nós sensores reais, considerando a distância de 1 metro.*

Ao analisarmos os gráficos das Figuras 5.2 e 5.3, onde são plotadas as frequências de tempos de atraso, percebemos que em ambos os casos os pacotes levam aproximadamente entre 6 e 15 milisegundos para serem entregues. Para chegarmos a um resultado mais preciso, utilizamos temporizadores configurados para fornecerem valores de tempo a uma precisão de 32 KHz, que equivale a 32.768 tiques de relógio por segundo do micro-

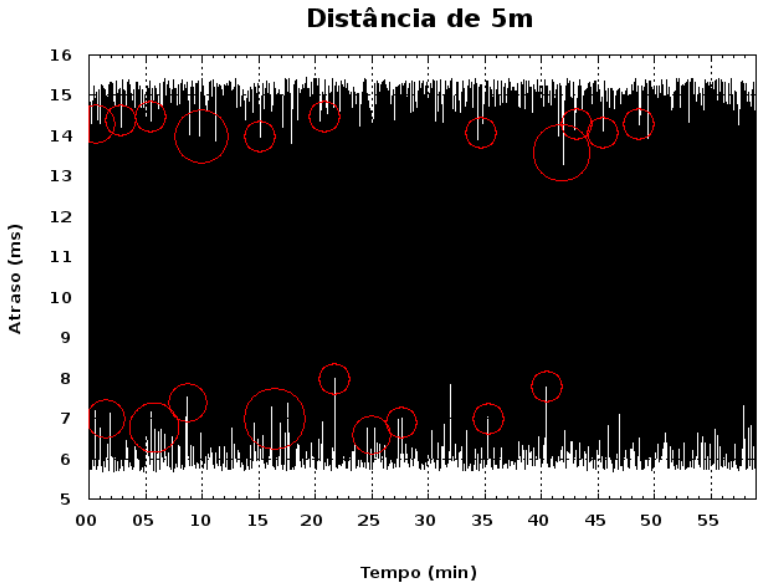


Figura 5.3: *Atraso unidirecional aferido entre dois nós sensores reais, considerando a distância de 5 metros.*

controlador MSP430-F1611 presente nos TelosB. Encontrado assim os valores médios de 10,5567 ms para 1 m, e 10,5772 ms para 5 m.

Outro fator observado refere-se às anomalias presentes nos gráficos. Se observarmos as Figuras 5.2 e 5.3, podemos verificar que as anomalias acontecem quase que em iguais proporções e de forma sincronizada. Infelizmente não conseguimos chegar a um entendimento sobre o que realmente causou esses resultados tão harminiosos. Acreditamos que, dada a tamanha sincronia dos eventos anomálicos, esses distúrbios sejam estimulados por problemas na pilha de comunicação do TinyOS ou no algoritmo de sincronização.

5.2 Integridade dos Dados

Os testes de integridade de dados aplicados ao nosso simulador híbrido HS_2N têm como objetivo verificar se as informações trafegadas entre os ambientes real e virtual se mantêm íntegras em sua composição básica. Devido às alterações realizadas nos pacotes multiplexados entre os ambientes real e virtual (ver Seção 4.2.1), foi considerado como parâmetro de integridade a completude das informações contidas nos cabeçalhos e *payloads* dos *frames* de dados.

Analisando como exemplo o esquema do *frame* de dados apresentado na Figura 5.4, cujas as diretrizes são especificadas em [61], podemos visualizar como as informações são estruturados nos *frames* IEEE 802.15.4. Sabendo que o *Physical layer Service Data Unit* (PSDU) deve ter no máximo 127 *bytes* de tamanho, e o cabeçalho MAC (MHR) dimensões entre 7 e 23 *bytes*, chegamos a uma faixa de valo-

res de trabalho de 102 a 118 *bytes* para o *payload*. Essa definição é importante para a configuração do SB Node.

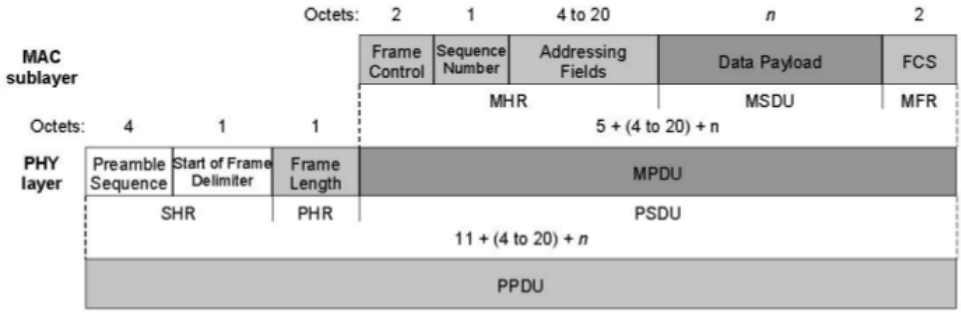


Figura 5.4: *Esquema de especificação do frame de dados de acordo o padrão IEEE 802.15.4 [61].*

Assim, com a definição de informações para verificação de integridade dos dados, desenvolvemos duas aplicações para serem embarcadas em nós sensores. A primeira, com função apenas de gerar dados e enviá-los por *broadcast*, foi configurada para emitir pacotes de tamanho de *payload* entre 1 e 118 *bytes* a uma frequência periódica de 1 Hz (*aprox.* 1s). A segunda aplicação apenas recebe os dados sem gerar qualquer estímulo na rede. Cada nó sensor é posicionado em ambientes mutuamente distintos, ou seja, quando o emissor está localizado em ambiente real, o receptor encontra-se em ambiente virtual, e vice-versa. A partir da execução dos testes, chegamos aos resultados apresentados na Tabela 5.1.

Tabela 5.1: *Verificação de integridade de cabeçalho e payload para frames IEEE 802.15.4.*

Payload (B)	Integridade do Cabeçalho (%)	Integridade do Payload (%)	Erro no Frame (%)
1	100	100	0
2	100	100	0
3	100	100	0
4	100	100	0
...	...	...	...
58	100	100	0
59	100	100	0
60	100	100	0
...	...	...	...
115	100	100	0
116	100	100	0
117	0	0	100
118	0	0	100

Analisando a Tabela 5.1, podemos constatar que ao usarmos tamanhos de *payload* entre 1 e 116 *bytes*, todos os dados que compõem o cabeçalho e o *payload* do *frame* mantiveram-se íntegros. Já para os pacotes com tamanho de *payload* maior que

116 *bytes*, foram todos descartados no SB Node, o que os torna 100% propício a erros. O descarte para *frames* com *payload* de dimensões acima de 116 *bytes* se deve a reserva padrão do TinyOS de 6 *bytes* para campos de endereçamentos curtos. Assim, a carga total do MHR para os *frames* testados foi de 9 *bytes*, sendo 2 *bytes* para controle do *frame*, 1 *byte* para a sequência numérica e 6 *bytes* para o endereçamento.

Apesar das definições de tamanho de *payload* terem sido estipuladas para reconhecer dados com no máximo 118 *bytes*, esse valor pode ser alterado em tempo de compilação no *software* do SB Node. Possibilitando que essa parametrização seja facilmente atualizada.

5.3 Desempenho e Escalabilidade do Simulador Híbrido

Tradicionalmente, quanto maior a fidelidade exigida na execução de uma aplicação, maiores são os recursos e tempos consumidos. Por isso que simulações que emulam dispositivos em nível de *hardware* costumam apresentar menores índices de desempenho e escalabilidade que simulações em níveis de sistema operacional e aplicação. Assim, apesar das inúmeras qualidades apresentadas ao longo deste documento em favor do uso dos simuladores que executam aplicações em nível de *hardware*, há um grande custo operacional associado a essa abordagem. Para mensurar esse custo, realizamos uma série de testes para averiguar o consumo de recursos e a capacidade de escala do nosso simulador híbrido HS_2N.

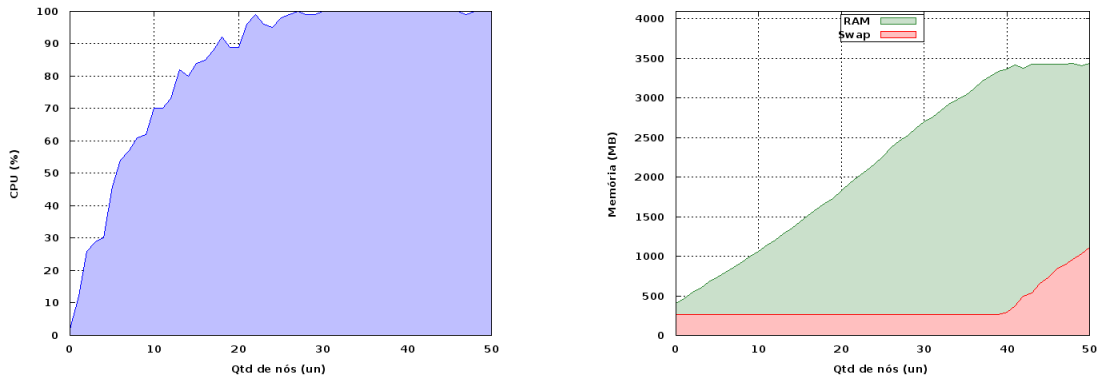
Como grande parte do comportamento dos nós sensores virtuais do HS_2N é herdado do simulador MSPSim, o impacto provocado nas simulações com o aumento do consumo de recursos do PC *host* continua o mesmo. Assim, como o simulador usa recursos do processador para serem consumidos pelos microcontroladores emulados, quanto mais recurso é consumido maior é a necessidade de um balanceamento de carga de processamento. Ou seja, quanto mais nós sensores virtuais são instanciados menores são os recursos disponibilizados para cada nó sensor.

Sendo assim, não é suficiente avaliar, por exemplo, quantos nós sensores virtuais é possível instanciar em um determinado PC, mas devemos também considerar o impacto que é causado nesses nós sensores com o aumento de escala da rede. Por isso, ao avaliarmos o desempenho e a escalabilidade do HS_2N, também verificamos a perturbação que é causada no processamento dos nós.

Para a realização dos testes foram utilizados: um PC com processador Intel Core 2 Quad, modelo Q9550, com velocidade de relógio de 2.83 GHz, capacidade da *cache* L2 de 12 MB, e com memória RAM de 4 GB; uma distribuição do Ubuntu 12.04.4 (x64) instalada e configurada; ferramentas de monitoramento de recursos do PC; e a infraestrutura completa do simulador híbrido HS_2N.

A execução dos testes se deu com a instanciação de forma contínua dos nós sensores virtuais e com o consumo de CPU e memória do PC *host* sendo monitorado constantemente. Cada nó sensor embarcava uma aplicação dedicada a estressar a CPU do microcontrolador, ou seja, manter o consumo em taxas constantes de 100% de uso. Adotamos essa abordagem para permitir analises de desempenho e escalabilidade para o pior caso.

Para monitorar o microcontrolador foram gerados *logs* com preenchimento periódico de informações sobre o consumo de CPU e a sua velocidade de relógio. O esperado nas informações colhidas é que o consumo de CPU se mantenha constante em 100% de uso, e que a velocidade de relógio diminua com o aumento do número de nós sensores virtuais instanciados. Após a conclusão dos experimentos e filtragem das informações, chegamos aos seguintes resultados de consumo de CPU e memória do PC *host*. Como apresentado nas Figuras 5.5(a) e 5.5(b).



(a) (b)

Figura 5.5: *(a) Consumo de CPU e (b) memória do simulador híbrido HS_2N.*

Ao observarmos a Figura 5.5(a), percebemos que o consumo de CPU do PC *host* cresce quase que em um movimento parabólico até se estabilizar em sua plenitude por volta do instanciamento do nó 27. Simultaneamente, observamos na Figura 5.5(b) que o consumo de memória RAM aumenta praticamente em perfeito crescimento linear até o instanciamento do nó 41. Momento em que esse consumo praticamente estabiliza em aproximadamente 3.400 MB (ou 83%) de memória RAM consumida, e passa a utilizar dos recursos da memória virtual (*Swap*) do PC *host*.

Para podermos definir valores mais exatos de escala, avaliamos o impacto causado na execução de CPU dos nós sensores virtuais a medida em que é aumentado o consumo de recursos do PC *host*. Os resultados obtidos podem ser vistos na Figura 5.6. Identificado que o consumo de CPU dos microcontroladores mantiveram-se sempre em 100% de uso, apenas consideramos para análise as velocidades de relógio registradas. Para isso, filtramos as informações definindo valores médios e valores de pico.

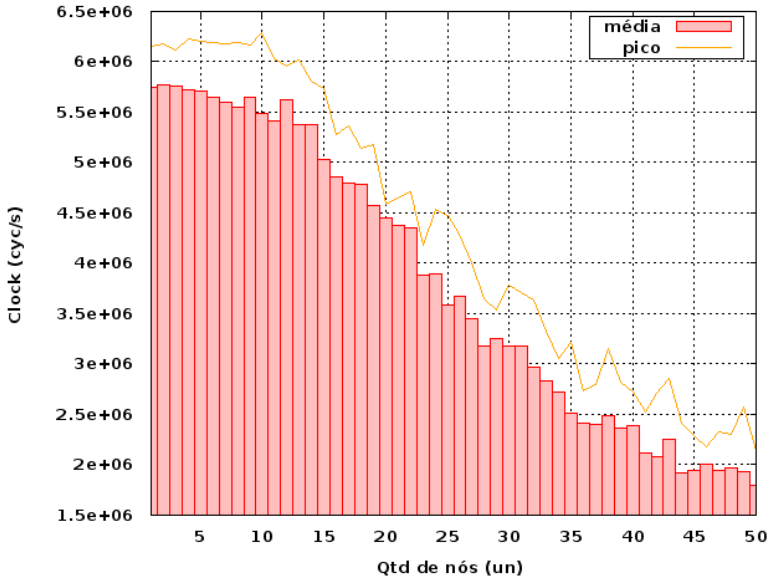


Figura 5.6: *Velocidade de relógio de CPU dos nós sensores virtuais.*

A partir da análise das informações presentes no gráfico da Figura 5.6, percebe-se que o desempenho de CPU dos microcontroladores é reduzido a medida que aumenta a quantidade de nós sensores instanciados no simulador. Apesar de não termos conhecimento da existência de um documento que padronize a velocidade de relógio da CPU, é perceptível que chega um momento em que o desempenho da simulação é consideravelmente prejudicada. Causando em maiores tempos de resposta para os nós sensores se manifestarem a estímulos.

Assim, dada as configurações apresentadas para os testes, inclusive a consideração de pior caso de consumo de CPU, entendemos que é possível trabalhar com até 20 nós sensores instanciados no HS_2N. Contudo, dificilmente o usuário vai se deparar com sistemas que resultem em *overhead* de CPU. Uma vez que dada as limitações inerentes às unidades de alimentação dos nós sensores, compor projetos com sobrecarga de dispositivo torna-se praticamente incompreensíveis. Inclusive, muitas das tradicionais aplicações para redes de sensores, fornecidas com as bibliotecas dos sistemas operacionais, apresentam consumo de menos de 5% de CPU (*e.g.*, `BlinkToRadio`, `BlinkToRadio`, `SyncLog`, `UDPEcho`). Como mostrado na Figura 5.7.

5.4 Avaliação da Porta Serial do SB Node

Conforme informações fornecidas na Seção 4.2.3, o componente SB Node apresenta uma gargalo na porta serial do seu *hardware*, TelosB. Isso porque esses dispositivos são limitados a uma vazão de 115.2 *kbps*. Praticamente metade do valor do dispositivo de rádio CC2420, cuja vazão máxima é de 250 *kbps*. Sendo assim, há um

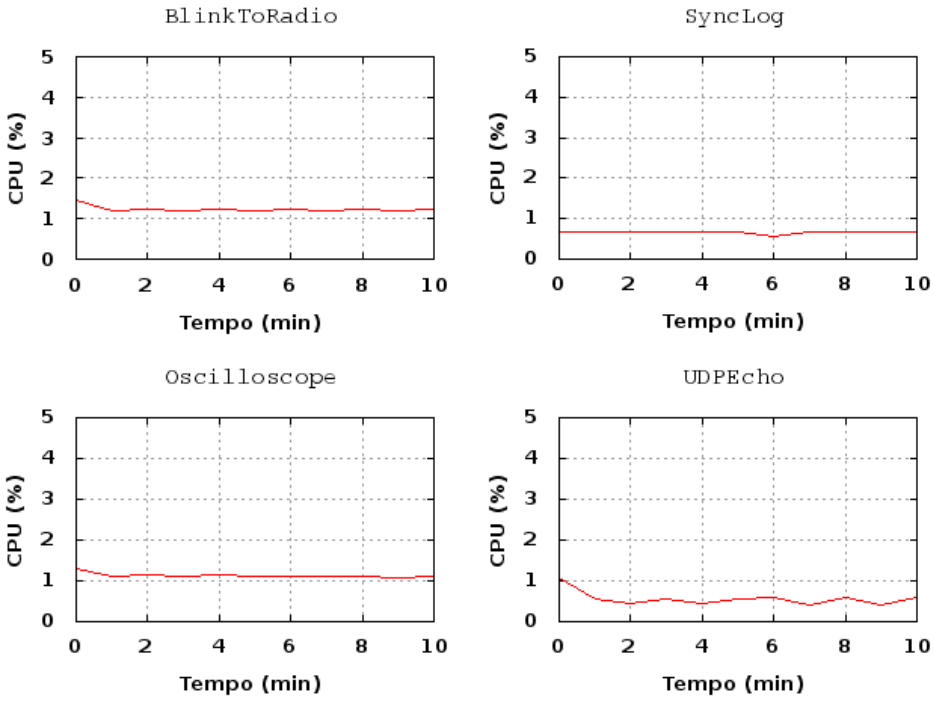


Figura 5.7: *Consumo de CPU no microcontrolador MSP430 para as aplicações BlinkToRadio, BlinkToRadio, SyncLog e UDPEcho.*

afunilamento na via de transmissão entre os nós sensores reais e os nós sensores virtuais, o que pode prejudicar consideravelmente o comportamento da rede, caso haja um tráfego intenso e constante na porta serial do SB Node.

Para comprovar essa limitação, utilizamos o *benchmark USB Serial Receive* [65], que envia pacotes, previamente armazenados em um *buffer*, para um nó sensor real do tipo TelosB. O objetivo é aferir a vazão real da porta serial do mote. Para que não haja nenhuma perturbação que possa comprometer os resultados, o TelosB utilizado no teste foi carregado com a aplicação Null, presente no pacote TinyOS. Os resultados obtidos podem ser verificados na Figura 5.8.

A partir da análise do gráfico plotado na Figura 5.8, podemos observar que a vazão praticamente se mantém constante a um valor de aproximadamente 95 *kbps*. Valor aproximado à taxa máxima de transmissão definida para esses dispositivos.

Também foram executados testes de *Round-trip time* (RTT), em que foi possível observar o aproveitamento do canal de comunicação serial para pacotes de tamanhos variados (Figura 5.9). Para a execução dos testes foram utilizadas duas aplicações desenvolvidas sobre o sistema operacional TinyOS. A primeira aplicação foi embarcada em um nó sensor virtual, que era encarregado de enviar pacotes por *broadcast* com valores crescentes de tamanho de *payload* a uma frequência periódica de 1 Hz (*aprox.* 1s). Sendo que para cada tamanho de *payload* eram enviados 10 pacotes e obtido o valor médio de tempo.

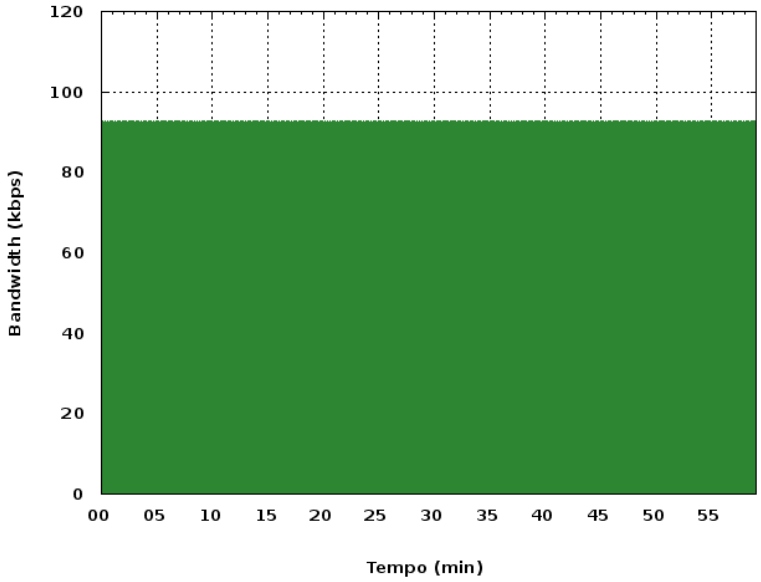


Figura 5.8: *Vazão real da porta serial do TelosB.*

A segunda aplicação foi embarcada em um nó sensor real, e apenas era responsável por devolver a mensagem enviada pelo nó sensor virtual. Entre os nós sensores virtual e real encontrava-se o agente intermediário SB Node.

Analisando os resultados plotados na Figura 5.9 é possível observar que o canal de comunicação serial pode ser melhor aproveitado com *payloads* que co-relacionam com a completude do *frame*. No entanto, os resultados apresentados ainda são inconclusivos, já que a carga gerada é pequena, dada a vazão da porta serial.

Para uma análise mais precisa sobre o impacto causado na nossa infraestrutura devido ao gargalo encontrado no SB Node, refizemos o teste de RTT, sem nenhuma alteração nas aplicações do teste anterior, mas com adição de sobrecarga na transferência de pacotes sobre a porta serial. Para isso, desenvolvemos uma aplicação Java que, paralelamente ao nó sensor virtual, envia pacotes *dummy* de 255 *bytes* (valor máximo permitido no `SerialForwarder`) em intervalos de 1 segundo. O resultado é apresentado na Figura 5.10.

Diferente da tendência observada no primeiro teste de RTT, em que pacotes com valores maiores de *payload* conseguem aproveitar melhor o canal de comunicação serial, o resultado apresentado na Figura 5.10 demonstra que em situações críticas de sobrecarga, as comunicações na porta serial do SB Node tendem a serem seriamente comprometidas. Principalmente quando envolvem trocas de mensagens com altos valores de tamanho de *payload*. Isso se deve ao fato das comunicações na porta serial serem dirigidas por fluxos de bits, resultando em remontagens mais rápidas de pacotes com menores cargas. Esse efeito não teve destaque no primeiro teste de RTT, justamente pela falta de sobrecarga no canal comunicação serial.

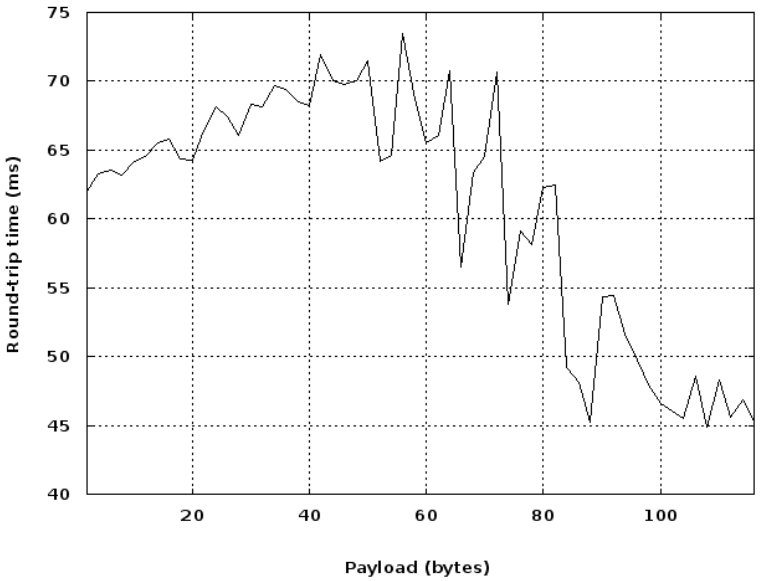


Figura 5.9: *Round-trip time para transmissões entre um nó sensor virtual e um nó sensor real.*

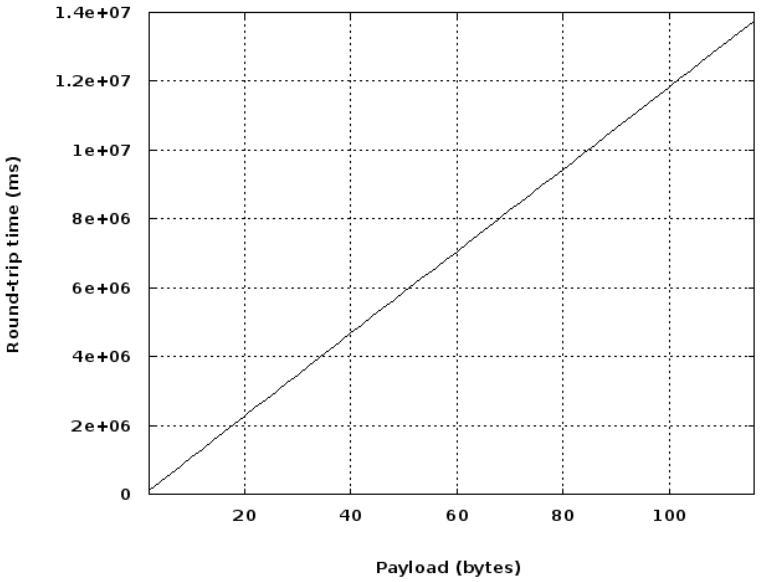


Figura 5.10: *Round-trip time para transmissões entre um nó sensor virtual e um nó sensor real, com sobrecarga na porta serial do SB Node.*

CAPÍTULO 6

Conclusão

A proposta descrita ao longo deste trabalho viabilizou uma alternativa para escalar *testbeds* de redes de sensores sem fio, com o objetivo de aprimorar esses ambientes de testes, possibilitando que o usuário avalie suas aplicações em infraestruturas mais robustas.

A abordagem utilizada baseou-se na recorrência da técnica de simulação *Hardware-In-the-Loop* para co-relacionar nós sensores reais com nós sensores virtuais, compondo um integrado ambiente físico-simulado.

O delineamento utilizado foi motivado pela possibilidade de contribuir com o *testbed* do projeto $[CIA]^2$/GT-TeI, desenvolvido para oferecer acesso público a usuários que desejam testar suas aplicações antes de implantá-las no mundo real. Por sinal, inviabilizado pela falta de uma infraestrutura física própria, o procedimento de avaliação de *softwares* é um dos principais desafios na pesquisa e desenvolvimento de redes de sensores [46, 66].

Para que a proposta apresentada se tornasse viável, entendemos que o empreendimento final deveria atender a um conjunto de predicados que favorecesse a redução do impacto na transferência dos sistemas experimentais para os sistemas reais. A definição desses predicados possibilitou a caracterização de todos os componentes que compõem a implementação do simulador híbrido HS_2N, ou seja, o MSPSim-RX, o Cuke e o SB Node.

O simulador híbrido HS_2N materializou a abordagem que propusemos para integrar os ambientes real e virtual, tornando possível a demonstração do cumprimento de todos os objetivos estipulados para este trabalho.

O primeiro objetivo traçado condicionou-se ao fornecimento de um ambiente híbrido totalmente transparente para o usuário. O segundo objetivo, adicionou a necessidade de alta-fidelidade na execução dos experimentos. Já o terceiro objetivo, adicionou confiabilidade ao empreendimento ao reivindicar a manutenção da integridade dos dados trafegados.

Com o nosso simulador híbrido HS_2N conseguimos oferecer uma ferramenta capaz de conduzir a abordagem apresentada provendo total transparência para o usuário.

Essa transparência decorre a partir do momento em que o HS_2N possibilita ao usuário carregar o mesmo arquivo binário que ele embarcaria em um nó sensor real, sem qualquer recompilação ou programação específica.

Adicionalmente, para tornar a infraestrutura mais transparente, o nosso simulador fornece alta-fidelidade na execução dos experimentos. Isso é possível devido à capacidade de execuções de rotinas em nível *hardware*, com emulação da maioria dos componentes constituintes dos nós sensores virtuais. Uma característica herdada do simulador base MSPSim.

Para garantir a manutenção da integridade dos dados, avaliamos os *frames* IEEE 802.15.4 em diferentes pontos da infraestrutura. Conforme os resultados apresentados na Seção 5.2, o nosso simulador híbrido HS_2N mantém íntegra todas as informações de cabeçalho e *payload* trafegados durante comunicações entre os ambientes real e virtual.

Para dar mais consistência ao nosso simulador híbrido aplicamos também testes de desempenho e escala. O objetivo foi identificar custo operacional do simulador. A partir dos resultados apresentados na Seção 5.3, constatamos que é possível escalar as redes físicas dos *testbeds* em algumas dezenas de nós virtuais usando poucos recursos computacionais. Sendo a escalabilidade fortemente dependente da robustez (CPU e memória) encontrada nos PCs que hospedarão os simuladores. É importante ressaltar que ao avaliar os custos de uma implantação híbrida confrontados aos custos de uma implantação puramente física, o desenvolvedor deve considerar além dos gastos com equipamentos, disponibilidade de espaço físico, capacidade de mobilidade dos nós, competência de injeção de dados de monitoramento, *etc*.

A partir de demais testes realizados foi possível identificar limitações na infraestrutura física do simulador. Considerado o ponto crítico do empreendimento, o componente SB Node, responsável pela integração entre os ambientes real e virtual, apresentou uma vazão em sua porta serial com valores menores do que a metade da vazão estipulada para o transceptor de rádio. Esse gargalo resulta em consideráveis atrasos na transmissão de mensagens, sobretudo em casos de sobrecarga (ver Seção 5.4). Entretanto, esse problema pode ser resolvido, ou pelo menos minimizado, se considerar a utilização de uma outra plataforma de *hardware*, com uma vazão na porta serial superior à identificada no TelosB. Outra opção seria o uso de múltiplos SB Node dividindo a carga trafegada entre os nós sensores reais e os nós sensores virtuais, entretanto, essa alternativa poderia elevar consideravelmente os custos do empreendimento, a ponto de torná-lo inviável.

Vale ressaltar que apesar de se mostrar funcional o nosso simulador híbrido HS_2N ainda não foi integrado a um *testbed* real. Sendo que, dada as características e peculiaridade de cada ferramenta, essa é uma atividade específica, e demanda conhecimento da infraestrutura e do sistema de gerência do *testbed*.

6.1 Trabalhos Relacionados

A recorrência às integrações entre ambientes reais e virtuais no universo das redes sensores é uma prática que já se desenvolve a mais de uma década. Em um dos trabalhos pioneiros, Park *et al*. [55] apresentou uma ferramenta de simulação com funcionalidades adicionais de comunicação com entidades externas. Já Wen *et al*. [73], procurou solucionar o problema de escala das redes físicas, com a adição de nós sensores virtuais.

Independente do objetivo adotado por cada pesquisador, é perceptível que o uso da técnica de simulação *Hardware-In-the-Loop* se dá em duas vias. Ou o trabalho está buscando encontrar meios de suprir algumas deficiências do simulador, ou está fazendo o papel inverso, ou seja, utilizando simuladores para adicionar funcionalidades às redes reais.

Watson *et al*. [71], apresenta uma solução para o problema de comunicação entre nós sensores virtuais através da fidelização da camada de transporte explorada nos nós sensores reais. Mesma abordagem é utilizada por Lo *et al*. [48].

Li et al. [46], encontrou na integração entre ambientes uma solução para resolver problemas pontuais do simulador. Já Girod *et al*. [38], promoveu heterogeneidade ao seu conjunto de ferramentas ao possibilitar interação entre motes, microsevers e nós sensores virtuais.

Song *et al*. [62], integrou o simulador a uma rede real, para permitir que as aplicações feitas para os nós sensores virtuais, as quais são desenvolvidas utilizando scripts, interagissem com as aplicações reais embarcadas nos nós sensores reais. Resultando assim em depuração de experimentos com maior fidelidade.

A busca por fidelidade, por sinal, é um forte indício de recorrência ao uso de nós sensores reais por simuladores. Por outro lado, assim como abordamos neste trabalho, a recorrência por simuladores evidencia na busca por escala, mobilidade ou instrumentação de ferramentas.

Para inserir o nosso simulador híbrido HS_2N ao conjunto de ferramentas descritas ao longo desta seção, elaboramos um comparativo, considerando características e peculiaridade de cada simulador presente nos supracitados trabalhos. Como pode ser visto na Tabela 6.1.

Dentre os tópicos confrontados na Tabela 6.1, destacamos a capacidade do nosso simulador HS_2N em suportar aplicações desenvolvidas sobre qualquer sistema operacional de redes de sensores. Uma vez que o HS_2N executa código de máquina, ele independe da plataforma de sistema operacional sobre a qual foi desenvolvida a aplicação do usuário.

Ao mesmo tempo, execuções em níveis de *hardware* fornecem tempos de execu-

Tabela 6.1: *Comparativo entre os principais simuladores híbridos e o HS_2N.*

Simulador Híbrido	Camada de Operação	Tipo de Enlace	Simulador Base	Sicronizador de Relógio	Executa Binários Reais	Sistemas Operacionais
SensorSim	aplicação	real e virtual	NS-2	sim	não	nenhum
MULE	S.O.	real	TOSSIM	sim	não	TinyOS
EnTOS/EmSim	aplicação	real e virtual	EmStar	sim	não	TinyOS
SEMU	aplicação	real	SEMU	sim	não[3]	nenhum
DiSenS	aplicação[1]	real e virtual	DiSenS	sim	não[2]	TinyOS
H-TOSSIM	S.O.	real e virtual	TOSSIM	sim	não	TinyOS
Hy-Sim	aplicação	real e virtual	Simulink	sim	não	TinyOS
HS_2N	*hardware*	real e virtual	MSPSim	não	sim	todos

1. promete emular componentes do nó sensor utilizando instâncias de máquina virtual.
2. apesar de permitir o carregamento dos binários, aplica alterações para adptar a aplicação ao sistema.
3. apesar de permitir o carregamento dos binários, a execução é simulada.

ções semelhantes aos apresentados por nós sensores reais. Logo, não existe a necessidade de implantar qualquer mecanismo de sincronização de tempo no HS_2N. Em contrapartida, caso os tempos de execução não fossem sincronizados pelos demais simuladores, os transceptores dos nós sensores reais não receberiam as mensagens originadas dos nós sensores virtuais. Prontamente descartando-as pelo *hardware*.

Assim, ao integrarmos com nós sensores reais um simulador que executa aplicações em nível de *hardware*, fornecemos ao HS_2N funcionalidades de via dupla às implementações de *Hardware-In-the-Loop* para as redes de sensores, pois ao mesmo tempo que provemos escalabilidade e instrumentação de ferramentas com o nosso simulador, mantivemos a alta-fidelidade presentes nos nós sensores reais. A questão de mobilidade não é bem definida no HS_2N, ficando para trabalho futuro.

6.2 Trabalhos Futuros

Como trabalhos futuros, propomos:

- Integrar o HS_2N a um *testbed* real.
- O desenvolvimento de uma plataforma dedicada de *hardware* para exercer a função de intercalação das comunicações entre o ambiente real e o ambiente virtual. Recomendamos uma atenção especial com a capacidade de comunicação do dispositivo serial.
- O aprimoramento da rede virtual, objetivando alcançar maior fidelidade nas comunicações entre nós sensores virtuais. Sugerimos a reconstrução do meio de comunicação a partir de simulações baseadas em *traces*.

- Um estudo direcionado à possibilidade de mapear o posicionamento dos nós sensores virtuais em um sistema de diagramação 3D, e simultaneamente prover mobilidade autônoma aos nós.
- Considerar o estudo de integração entre diferentes emuladores de microcontrolador, a fim de aumentar a gama de plataformas de nós sensores suportadas pelo simulador.
- Considerar a alternativa de implantar comunicação concorrente entre os ambientes real e virtual, com a finalidade de aumentar a capacidade de vasão entre o PC e *hardware* dedicado que integra os dois ambientes.

Referências Bibliográficas

[1] ADVANTIC SISTEMAS Y SERVICIOS, S. **AS-XM1000**. Datasheet, Madrid, Spain.

[2] ADVANTIC SISTEMAS Y SERVICIOS, S. **MTM-CM3300-MSP**. Datasheet, Madrid, Spain.

[3] AKYILDIZ, I. F.; WEILIAN, S.; SANKARASUBRAMANIAM, Y.; CAYIRCI, E. E. **A Survey on Sensor Networks**. *IEEE Communications Magazine*, 40(8):102–114, August 2002.

[4] AKYILDIZ, I. F.; VURAN, M. C. **Wireless Sensor Networks**, chapter 1, p. 2. John Wiley & Sons Ltd., 2010.

[5] AL-KARAKI, J. N.; KAMAL, A. E. **Routing techniques in wireless sensor networks: a survey**. *IEEE Wireless Communications*, 11(6):6–28, December 2004.

[6] ARORA, A.; ERTIN, E.; RAMNATH, R.; NESTERENKO, M.; LEAL, W. **Kansei: a high-fidelity sensing testbed**. *IEEE Internet Computing*, 10(2):35–47, March 2006.

[7] ART OF TECHNOLOGY, A. **BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks**. Datasheet, Zurich, Switzerland.

[8] BHATTI, S.; CARLSON, J.; DAI, H.; DENG, J.; ROSE, J.; SHETH, A.; SHUCKER, B.; GRUENWALD, C.; TORGERSON, A.; HAN, R. **MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms**. *Mobile Networks and Applications*, 10(4):563–579, August 2005.

[9] BOULIS, A. **Castalia: Revealing Pitfalls in Designing Distributed Algorithms in WSN**. In: *5^{th} ACM Conference on Embedded Networked Sensor Systems, SenSys '07*, 2007.

[10] BOULIS, A.; HAN, C.-C.; SHEA, R.; SRIVASTAVA, M. B. **SensorWare: Programming sensor networks beyond code update and querying**. *Pervasive and Mobile Computing*, 3(4):386–412, August 2007.

[11] CAO, Q.; ABDELZAHER, T.; STANKOVIC, J.; HE, T. **The LiteOS Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks**. In: *International Conference on Information Processing in Sensor Networks, IPSN '08*, 2008.

[12] CHEN, G.; LI, C.; YE, M.; JIEWU. **An unequal cluster-based routing protocol in wireless sensor networks**. *Wireless Networks*, 15(2):193–207, February 2009.

[13] CHIKHI, Y. **Reducing the Hot Spot Effect in Wireless Sensor Networks with the Use of Mobile Data Sink**. M.s. thesis, University of New Orleans, New Orleans, LA, USA, May 2006.

[14] CHWIF, L.; MEDINA, A. C. **Modelagem e Simulação de Eventos Discretos: Teoria e Aplicações**, chapter 1, p. 4. Ed. do Autor, 2010.

[15] COULSON, G.; PORTER, B.; CHATZIGIANNAKIS, I.; KONINIS, C.; FISCHER, S.; PFISTERER, D.; BIMSCHAS, D.; BRAUN, T.; HURNI, P.; ANWANDER, M.; WAGENKNECHT, G.; FEKETE, S. P.; KRÖLLER, A.; BAUMGARTNER, T. **Flexible Experimentation in Wireless Sensor Networks**. *Communications of the ACM*, 55(1):82–90, January 2012.

[16] CREPALDI, R.; HARRIS, A.; SCARPA, A.; ZANELLA, A.; ZORZI, M. **SignetLab: Deployable Sensor Network Testbed and Management Tool**. In: 4^{th} *International Conference on Embedded Networked Sensor Systems, SenSys '06*, 2006.

[17] CREPALDI, R.; III, A. F. H.; ZANELLA, A.; ZORZI, M. $SignetLab^2$**: A Modular Management Architecture for Wireless Sensor Networks**. In: Pupolin, S., editor, *Wireless Communications 2007 CNIT Thyrrenian Symposium*. Springer US, 2007.

[18] CROSSBOW TECHNOLOGY, I. **MICA2 - Wireless Measurement System**. Datasheet, San Jose, CA, USA.

[19] CROSSBOW TECHNOLOGY, I. **MICAz - Wireless Measurement System**. Datasheet, San Jose, CA, USA.

[20] CROSSBOW TECHNOLOGY, I. **TelosB Mote Platform**. Datasheet, San Jose, CA, USA.

[21] DA SILVA NEVES, P. A. C.; RODRIGUES, J. J. P. C. **Internet Protocol over Wireless Sensor Networks, from Myth to Reality**. *Journal of Communications*, 5(3):189–196, March 2010.

[22] DARGIE, W.; POELLABAUER, C. **Fundamentals of Wireless Sensor Networks - Theory and Practice**, chapter 1, p. 3. John Wiley & Sons Ltd., 2010.

[23] DE PAZ ALBEROLA, R.; PESCH, D. **AvroraZ: Extending Avrora with an IEEE 802.15.4 Compliant Radio Chip Mode**. In: 3^{nd} *ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks, PM2HW2N '08*, 2008.

[24] DES ROSIERS, C. B.; CHELIUS, G.; FLEURY, E.; FRABOULET, A.; GALLAIS, A.; MITTON, N.; NOËL, T. **SensLAB Very Large Scale Open Wireless Sensor Network Testbed**. In: 7^{th} *International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, TridentCOM '11*, 2011.

[25] DODDAVENKATAPPA, M.; CHAN, M. C.; ANANDA, A. L. **Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed**. In: Korakis, T.; Li, H.; Tran-Gia, P.; Park, H.-S., editors, *Testbeds and Research Infrastructure. Development of Networks and Communities*, volume 90. Springer Berlin Heidelberg, 2012.

[26] DOWNARD, I. T. **Simulating Sensor Networks in NS-2**. Technical report, Naval Research Laboratory, 2004.

[27] DUNKELS, A.; GRONVALL, B.; VOIGT, T. **Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors**. In: 29^{th} *Annual IEEE Conference on Local Computer Networks, LCN '04*, 2004.

[28] DUTTA, P.; HUI, J.; JEONG, J.; KIM, S.; SHARP, C.; TANEJA, J.; TOLLE, G.; WHITEHOUSE, K.; CULLER, D. **Trio: Enabling Sustainable and Scalable Outdoor Wireless Sensor Network Deployments**. In: 5^{th} *International Conference on Information Processing in Sensor Networks, IPSN '06*, 2006.

[29] E SILVA, V. F.; MACEDO, D. F.; LEONI, J. L. **Decisão de Espectro em Redes de Sensores Sem Fio Empregando Aprendizado de Máquina**. In: *Anais do* 32^{o} *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, SBRC '14*, 2014.

[30] ERIKSSON, J.; DUNKELS, A.; FINNE, N.; ÖSTERLIND, F.; VOIGT, T. **MSPSim - an Extensible Simulator for MSP430-equipped Sensor Boards**. In: 4^{th} *European Conference on Wireless Sensor Networks, EWSN '07*, 2007.

[31] ERIKSSON, J.; FINNE, N.; TSIFTES, N.; VOIGT, T.; GIELDA, M.; GIELDA, P. **Demo Abstract: Emulink - Heterogeneous Sensor Network Simulation in Cooja**. In: 10^{th} *European Conference on Wireless Sensor Networks, EWSN '13*, 2013.

[32] ERIKSSON, J.; ÖSTERLIND, F.; FINNE, N.; TSIFTES, N.; DUNKELS, A.; VOIGT, T.; SAUTER, R.; MARRÓN, P. J. **COOJA/MSPSim: Interoperability Testing for**

Wireless Sensor Networks. In: 2^{nd} *International Conference on Simulation Tools and Techniques, SIMUTools '09*, 2009.

[33] ESWARAN, A.; ROWE, A.; RAJKUMAR, R. **Nano-RK: an Energy-aware Resource-centric RTOS for Sensor Networks**. In: 26^{th} *IEEE Real-Time Systems Symposium, RTSS '05*, 2005.

[34] FAMBON, O.; ÉRIC FLEURY.; HARTER, G.; PISSARD-GIBOLLET, R.; SAINT-MARCEL, F. **FIT IoT-LAB tutorial: hands-on practice with a very large scale testbed tool for the Internet of Things**. In: *10èmes journées francophones Mobilité et Ubiquité, UbiMob2014*, 2014.

[35] FROM TEXAS INSTRUMENS, C. P. **CC2420 - 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver**. Datasheet, Oslo, Norway, March 2013.

[36] GAY, D.; LEVIS, P.; CULLER, D.; BREWER, E. **nesC 1.3 Language Reference Manual**. TinyOS documentation, July 2009.

[37] GAY, D.; LEVIS, P.; VON BEHREN, R.; WELSH, M.; BREWER, E.; CULLER, D. **The nesC Language: A Holistic Approach to Networked Embedded Systems**. In: *ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, PLDI '03*, 2003.

[38] GIROD, L.; STATHOPOULOS, T.; RAMANATHAN, N.; ELSON, J.; ESTRIN, D.; OSTERWEIL, E.; SCHOELLHAMMER, T. **A System for Simulation, Emulation, and Deployment of Heterogeneous Sensor Networks**. In: 2^{nd} *International Conference on Embedded Networked Sensor Systems, SenSys '04*, 2004.

[39] HANDZISKI, V.; KÖPKE, A.; WILLIG, A.; WOLISZ, A. **TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Networks**. In: 2^{nd} *International Workshop on Multi-hop Ad Hoc Networks: From Theory to Reality, REALMAN '06*, 2006.

[40] HULL, B. **SerialForwarder v1.1**. Viewed on June 14^{th}, 2014 at 15:40, October 2001.

[41] INTEL CORPORATION, C. **SHIMMER - Sensing Health with Intelligence, Modularity, Mobility, and Experimental Reusability**. Datasheet.

[42] LANDSIEDEL, O.; WEHRLE, K.; TITZER, B. L.; PALSBERG, J. **Enabling Detailed Modeling and Analysis of Sensor Networks**. *Praxis der Informationsverarbeitung und Kommunikation*, 28(2):101–106, December 2007.

[43] LCIS.; SYSTEMS, A. **WiSMote - Low-power wireless sensor/actuator module for WSN applications**. Datasheet, France.

[44] LEVIS, P.; MADDEN, S.; POLASTRE, J.; SZEWCZYK, R.; WHITEHOUSE, K.; WOO, A.; GAY, D.; HILL, J.; WELSH, M.; BREWER, E.; CULLER, D. **TinyOS: An Operating System for Sensor Networks**. *Ambient Intelligence*, 2005.

[45] LEVIS, P.; LEE, N.; WELSH, M.; CULLER, D. **TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications**. In: 1^{st} *International Conference on Embedded Networked Sensor Systems, SenSys '03*, 2003.

[46] LI, W.; ZHANG, X.; TAN, W.; ZHOU, X. **H-TOSSIM: Extending TOSSIM with Physical Nodes**. *Wireless Sensor Network*, 1(4):324–333, November 2009.

[47] LI, Y.; PANWAR, S. S.; MAO, S. **A wireless biosensor network using autonomously controlled animals**. *IEEE Network*, 20(3):6–11, May 2006.

[48] LO, S.-H.; DING, J.-H.; HUNG, S.-J.; TANG, J.-W.; TSAI, W.-L.; CHUNG, Y.-C. **SEMU: A Framework of Simulation Environment for Wireless Sensor Networks with Co-simulation Model**. In: *Second International Conference on Advances in Grid and Pervasive Computing, GPC '07*, 2007.

[49] LOPES, C. E. R.; DE MELO, J. C.; DE ASSUNÇÃO, H. P.; SILVA, F. A.; BRAGA, T. R. M.; RUIZ, L. B.; LOUREIRO, A. A.; NOGUEIRA, J. M. S. **MannaSim: Simulando Redes de Sensores Sem Fio**. In: 24^{th} *Simpósio Brasileiro de Redes de Computadores, SBRC '06*, 2006.

[50] MAROTI, M.; SALLAI, J. **TEP 133: Packet-level time synchronization**. Documentary, Core Working Group, 2008.

[51] MASONTA, M. T.; MZYECE, M.; NTLATLAPA, N. **Spectrum Decision in Cognitive Radio Networks: A Survey**. *IEEE Communications Surveys & Tutorials*, 15(3):1088–1107, 2013.

[52] MOTEIV CORPORATION, C. **Telos - Ultra low power IEEE 802.15.4 compliant wireless sensor module**. Datasheet, San Francisco, CA, USA, December 2004. Rev B.

[53] MOTEIV CORPORATION, C. **TmoteSky - Ultra low power IEEE 802.15.4 compliant wireless sensor module**. Datasheet, San Francisco, CA, USA, February 2006.

[54] ÖSTERLIND, F.; DUNKELS, A.; ERIKSSON, J.; FINNE, N.; VOIGT, T. **Cross-Level Sensor Network Simulation with COOJA**. In: 31^{st} *Annual IEEE Conference on Local Computer Networks, LCN '06*, 2006.

[55] PARK, S.; SAVVIDES, A.; SRIVASTAVA, M. B. **SensorSim: A Simulation Framework for Sensor Networks**. In: 3^{rd} *International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems, MSWiM '00*, 2000.

[56] POLLEY, J.; BLAZAKIS, D.; MCGEE, J.; RUSK, D.; BARAS, J. S. **ATEMU: A Fine-grained Sensor Network Simulator**. In: *First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, SECON '04*, 2004.

[57] SCHILLER, J.; LIERS, A.; RITTER, H.; WINTER, R.; VOIGT, T. **ScatterWeb - Low Power Sensor Nodes and Energy Aware Routing**. In: 38^{th} *Hawaii International Conference on System Sciences, HICSS '05*, 2005.

[58] SCHLAGER, M.; ELMENREICH, W.; WENZEL, I. **Interface Design for Hardware-In-the-Loop Simulation**. In: *IEEE International Symposium on Industrial Electronics, ISIE '06*, 2006.

[59] SHEU, J.-P.; CHANG, C.-J.; SUN, C.-Y.; HU, W.-K. **WSNTB: A testbed for heterogeneous wireless sensor networks**. In: *First IEEE International Conference on Ubi-Media Computing, UMEDIA '08*, 2008.

[60] SOBEIH, A.; CHEN, W.-P.; HOU, J. C.; KUNG, L.-C.; LI, N.; LIM, H.; TYAN, H.-Y.; ZHANG, H. **J-Sim: A Simulation Environment for Wireless Sensor Networks**. In: 38^{th} *Annual Symposium on Simulation, ANSS '05*, 2005.

[61] SOCIETY, I. C. **IEEE Std** $802.15.4^{TM}$**-2003**. The Institute of Electrical and Electronics Engineers, Inc., New York, NY, USA, 2003 edition, October 2003.

[62] SONG, Z. Y.; MOSTAFIZUR, M.; MOZUMDAR, R.; TRANCHERO, M.; LAVAGNO, L.; TOMASI, R.; OLIVIERI, S. **Hy-Sim: Model based Hybrid Simulation framework for WSN application development**. In: 3^{rd} *International ICST Conference on Simulation Tools and Techniques, SIMUTools '10*, 2010.

[63] SRIDHARAN, M.; ZENG, W.; LEAL, W.; JU, X.; RAMNATH, R.; ZHANG, H.; ARORA, A. **From Kansei to KanseiGenie: Architecture of Federated, Programmable Wireless Sensor Fabrics**. In: Magedanz, T.; Gavras, A.; Thanh, N.; Chase, J. S., editors, *Testbeds and Research Infrastructures. Development of Networks and Communities*, volume 46. Springer Berlin Heidelberg, 2011.

[64] SRIDHARAN, M.; ZENG, W.; LEAL, W.; JU, X.; RAMNATH, R.; ZHANG, H.; ARORA, A. **KanseiGenie: software infrastructure for resource management and programmability of wireless sensor network fabrics**. In: Ramamurthy, B.; Rouskas,

G. N.; Sivalingam, K. M., editors, *Next-Generation Internet: Architectures and Protocols*. Cambridge University Press, 2011.

[65] STOFFREGEN, P. J.; COON, R. C. **USB Virtual Serial Receive Speed**. Online.

[66] SUNDANI, H.; LI, H.; DEVABHAKTUNI, V. K.; ALAM, M.; BHATTACHARYA, P. **Wireless Sensor Network Simulators: A Survey and Comparisons**. *International Journal of Computer Networks*, 2(6):249–265, February 2011.

[67] THILINA, K. M.; SAQUIB, K. W. C. N.; HOSSAIN, E. **Machine Learning Techniques for Cooperative Spectrum Sensing in Cognitive Radio Networks**. *IEEE Journal on Selected Areas in Communications*, 31(11):2209–2221, November 2013.

[68] TITZER, B. L.; LEE, D. K.; PALSBERG, J. **Avrora: Scalable Sensor Network Simulation with Precise Timing**. In: *Fourth International Symposium on Information Processing in Sensor Networks, IPSN '05*, 2005.

[69] VARGA, A. **The OMNeT++ Discrete Event Simulation System**. In: 15^{th} *European Simulation Multiconference, ESM '01*, 2001.

[70] VON EICKEN, T.; CULLER, D. E.; GOLDSTEIN, S. C.; SCHAUSER, K. E. **Active Messages: A Mechanism for Integrated Communication and Computation**. In: *International Symposium on Computer Architecture, ISCA '98*, 1998.

[71] WATSON, D.; NESTERENKO, M. **MULE: Hybrid Simulator for Testing and Debugging Wireless Sensor Networks**. In: *Second International Workshop on Sensor and Actor Network Protocols and Applications, SANPA '04*, 2004.

[72] WEN, Y.; WOLSKI, R.; MOORE, G. **DiSenS: Scalable Distributed Sensor Network Simulation**. In: 12^{th} *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '07*, 2007.

[73] WEN, Y.; ZHANG, W.; WOLSKI, R.; CHOHAN, N. **Simulation-Based Augmented Reality for Sensor Network Development**. In: 5^{th} *International Conference on Embedded Networked Sensor Systems, SenSys '07*, 2007.

[74] WERNER-ALLEN, G.; SWIESKOWSKI, P.; WELSH, M. **MoteLab: A Wireless Sensor Network Testbed**. In: 4^{th} *International Symposium on Information Processing in Sensor Networks, IPSN '05*, 2005.

[75] YANRUI, L.; HAIBIN, S. **Physical layer designing and simulation for wireless sensor network based on NS-3**. *Electronic Measurement Technology*, 2009.

[76] ZENG, X.; BAGRODIA, R.; GERLA, M. **GloMoSim: A Library for Parallel Simulation of Large-scale Wireless Networks**. In: 12^{th} *Workshop on Parallel and Distributed Simulation, PADS '98*, 1998.

[77] ZHENG, J. **Wireless Sensor Networks - A Networking Perspective**, chapter 2, p. 24–26. John Wiley & Sons, Inc., 2009.

[78] ZOLERTIA. **Z1**. Datasheet, Barcelona, Spain, March 2010. V1.1.

APÊNDICE A

Processo de Seleção do Simulador Base

A.1 Quesito 1: alta-fidelidade de tempos de execução

Considerando que o simulador base deveria atender aos pré-requisitos listados na Seção 4.2.1. Iniciamos o nosso processo de seleção avaliando as ferramentas de simulação que atendesse ao quesito (1), ou seja, identificação dos simuladores capazes de simular tempos de execução com alta-fidelidade.

A partir da análise dos simuladores apresentados na Tabela 4.1 e de informações contidas nas Seções 3.2.1, 3.2.2, 3.2.3 e 3.2.4, pudemos avaliar que os simuladores que executam suas aplicações em nível de conjunto de instrução de ciclo preciso (*hardware*) obtém uma maior fidelidade na execução de tarefas de ciclo preciso. Esse fenômeno se deve à redução da quantidade de ciclos de CPU consumidos por cada processo — onde cada nó sensor virtual tem um processo correspondente -– a um nível equivalente aos dos microcontroladores presentes nos nós sensores reais.

Fundamentado pela supracitada consideração, definimos os simuladores que executam em nível de *hardware* como os únicos capazes de atender ao requisito de alta-fidelidade de tempos de execução. Dentre outras vantagens presentes nesses simuladores, vale ressaltar que em uma infraestrutura híbrida torna-se desnecessário o desenvolvimento de um sistema de sincronização de tempo entre os ambientes real e virtual.

A.2 Quesito 2: heterogeneidade de sistema operacional

Desconsiderando as ferramentas de simulação integrada, já que a avaliação de cada simulador de forma independente é mais conveniente; após a primeira mineração, chegamos aos simuladores que executam aplicações em nível de *hardware*, sendo eles ATEMU, Avrora e MSPSim. Por conseguinte, identificamos quais davam suporte à heterogeneidade de sistema operacional, quesito (2) dos pré-requisitos listados na Seção 4.2.1, tornando a ferramenta mais genérica possível.

Com ATEMU, Avrora e MSPSim executando aplicações em nível de conjunto de instrução (*hardware*), indiretamente anulamos a necessidade de avaliação dos simuladores pelo quesito (2), uma vez que ferramentas de simulação que executam em nível de *hardware* independem da plataforma de sistema operacional utilizada no desenvolvimento da aplicação [23].

A.3 Quesito 3: diversidade de plataformas de nós sensores

Ao avaliarmos o quesito (3), – a partir da análise da Tabela 4.1 – conseguimos observar na coluna "mote" que o MSPSim é o simulador em nível de *hardware* que mais se sobressai quanto à quantidade de plataformas de nós sensores suportadas. Apesar de não ser um requisito primordial, dispor de uma maior quantidade de plataformas de nós sensores já desenvolvidas contribui consideravelmente com o enriquecimento da infraestrutura e com a diminuição dos custos na confecção de novas plataformas.

A.4 Quesito 4: comunidade ativa

Mesmo já sendo possível definir a ferramenta de simulação a ser adotada como simulador base, reforçamos a nossa escolha pelo MSPSim ao considerar a avaliação final, quesito (4). Apesar do ATEMU e do Avrora não serem projetos descontinuados, o MSPSim é o simulador em nível de *hardware* com comunidade online mais ativa, inclusive, fazendo parte do projeto Contiki-OS, que é amplamente apoiado por consagradas instituições de pesquisa.

A.5 Justificativa final para a escolha do MSPSim

Com o MSPSim atendendo aos quatro pré-requisitos solicitados para a definição do simulador base, reforçamos outros pontos favoráveis ao MSPSim ao considerar suas características extensivas (possibilitam a conveniente adição de novas plataformas) e a capacidade de emular de forma completa os microcontroladores da série MSP430 (não suportado por ATEMU e Avrora) e o chip de rádio CC2420 (não suportado por ATEMU), dispositivos comuns aos nós sensores do tipo TelosB, que utilizamos para integrar à nossa infraestrutura híbrida.

Apesar do simulador Avrora (ou mesmo sua versão modificada AvroraZ) também adequar, mesmo que com menos destaque, aos nossos pré-requisitos. Resolvemos, a partir de uma decisão direta e pessoal, adotar o MSPSim como a nossa ferramenta de trabalho.

Motivado, inclusive, pelas supracitadas características adicionais presentes no simulador. Além do mais, o MSPSim fornece, de maneira já constituída, a ferramenta EmuLink [31], que possibilita o fornecimento de interfaces para que outros simuladores sejam facilmente integrados ao MSPSim, como por exemplo o próprio Avrora.