UNIVERSIDADE FEDERAL DE SANTA CATARINA
SISTEMAS DE INFORMAÇÃO

Um Modelo de Simulação para Camada Física de Rádios Cognitivos

Everton Rodrigues Garcia

Florianópolis, maio de 2010

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Um Modelo de Simulação para Camada Física de Rádios Cognitivos

Acadêmico: Everton Rodrigues Garcia
Orientador: Carlos Becker Westphall
Co-Orientador: Rafael de Souza Mendes
Banca examinadora: Carla Merkle Westphall

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Florianópolis – SC – Brasil
maio/2010

Dedicatória

Dedico este trabalho a minha família, aos amigos verdadeiros e a todos que acreditam que nada é por acaso.

“Minha mãe me chama Comanche !
Minha estrela do oriente
Eu sou comanche
Sou batuqueiro
Nasci e vivo contente comigo e
Com minha gente
Pois enquanto existir Deus no céu
Urubu não come folha
Enquanto existir Deus no céu
Eu vou cantando gente boa...”

Jorge Ben Jor

Agradecimentos

Primeiramente a Deus pela saúde, pela clareza de meus pensamentos e por ser o guia de minhas passadas.

Aos meus pais, João Carlos e Claudete, por sua garra, exemplo, apoio, incentivo, inspiração e por acreditarem sempre em seus filhos.

A minha esposa Letícia e filhos, Maysa e Renan, pelo apoio em meus projetos, pelo amparo nos momentos difíceis, e por fazerem parte de minha vida.

Ao meu irmão Vinícius, pela parceria de sempre.

Aos meus sogros, Dionísio e Linda, por terem concebido presente tão maravilhoso, minha esposa, por contribuírem na educação de meus filhos e pela amizade gratificante de sempre.

Ao meu professor orientador Carlos pela paciência, pela troca de conhecimento e pelo acompanhamento de meu trabalho.

Ao meu co-orientador Rafael por ter aceitado minha participação neste desafio, por ter contribuído com todo o seu conhecimento para o melhoramento deste trabalho e por sua paciência nas repetitivas explanações no Laboratório de Redes e Gerência, pois como diz o professor Westphall, “a repetição faz parte do aprendizado !”.

Aos professores que *fazem a diferença* em nosso curso, cuja dedicação em repassar seus conhecimentos é recompensada pela satisfação de seus alunos.

A todos os meus amigos da graduação que estiveram sempre presentes e dispostos a contribuir e a dividir o conhecimento nesta fase de minha caminhada, em especial ao *mano* Cláudio Raso Filho pela camaradagem de sempre.

Ao nosso Brasil, a todos os seus contribuintes e à UFSC por me proporcionar uma educação pública, gratuita e de qualidade.

Resumo

Este trabalho propõe um modelo de camada física para redes sem fio cognitivas (*Ad-hoc*). Apresentamos sua arquitetura, funcionamento, e a implementação deste modelo utilizando técnicas baseadas em simulação discreta de eventos.

O modelo de camada física proposto visa emular o meio físico. Assim, a simulação desconsidera os processos de conversões de sinais de rádio frequência utilizados pelos dispositivos sem fio e o processamento pelo *Digital Signal Processors* (DSP), conforme proposto em [10]. Pretende-se, então, analisar o comportamento de rádios cognitivos, quando do envio de sinal transmitido de um rádio para todos os demais do sistema, considerados problemas de sincronismo, perdas de sinal, e atenuação. Nesse ambiente, os nós são considerados fixos, assim problemas de mobilidade estão fora do contexto da pesquisa. Foram analisados, também, as variações de comportamento considerando diferentes configurações do sistema, tais como cenários onde somente um rádio transmite versus múltiplas transmissões simultâneas. Também, são analisados características do sinal transmitido nessas condições.

O trabalho aqui descrito tenta ser o mais próximo da realidade possível, baseando-se em simuladores de rede sem fio existentes, entretanto, o foco principal em sua especificação é considerar as necessidades de uma camada física (PHY) que atendesse o modelo descrito no trabalho abaixo citado e que pudesse ser passível de simulação.

Por fim, desenvolveu-se um protótipo na linguagem Java [9], cuja simulação e os testes foram efetuados utilizando-se o simulador de eventos discretos – *Java In Simulation Time,* JIST [2] – servindo assim como forma de validação do modelo que, atendendo os requisitos de operação, será um módulo componente do trabalho publicado em [24]: *Um Framework para Operação na Camada de Rádio em Redes Cognitivas, Mendes (2009) - The Sixth International Conference on Networking and Services - ICNS 2010 - March 7-13, 2010 - Cancun, Mexico.*

Palavras-Chave: Redes sem fio, redes cognitivas, camada física, simulação, JIST.

Abstract

This work presents a physical-layer model for wireless (ad hoc) cognitive networks. It introduces an architecture, describes the operations, and provides a simulation environment using a discrete-event simulator.

The physical-layer model aims to emulate the physical environment. It proposes to analyse the digital signals processed in the simulator regardless of the conversion of radio frequency signals used by wireless devices and bypassing the Digital Signal Processor (DSP) processing, as suggested by [10]. The model is intended to meet the requirements of cognitive radios capable of sending a signal transmitted from a radio to all the other radios in the system. It should handle the issues of synchronism, loss, and attenuation. As a limitation, however, the nodes in the simulations are considered to be stationary. Thus, issues of mobility are not addressed.

We related the factors that influence the transmission process while modelling the physical layer. Specifically, we analysed the expected behaviour in relation to variations of transmission such as when only one radio transmits versus simultaneous transmissions. In addition, we analized the characteristics of signal being transmitted in these situations.

We validated our proposal by cross-relating the results with real world data obtained from existing wireless-network simulators. We developed a proof-of-concept simulation environment using the Java programming language [9] and discrete event simulation (i.e. Simulation In Java Time JIST [2]), thus serving as a model validation that, given the requirements of operation, will be a component of the work published in [24]: A Framework for Operation Layer of Cognitive Radio Networks, Mendes (2009) - The Sixth International Conference on Networking and Services - ICNS 2010 - March 7 -13, 2010 - Cancun, Mexico.

Keywords: Wireless networks, cognitive networks, physical layer, simulation, JIST.

Sumário

Lista de Abreviaturas

ANATEL .....Agência Nacional de Telecomunicações

AODV .....Ad hoc On-Demand Distance Vector

AWGN .....Additive White Gaussian Noise

BER .....Bit Error Rate

CRCN .....Cognitive Radio Cognitive Network Simulator

DSP .....Digital Signal Processor

DSA .....Dynamic Spectrum Access

DSR .....Dynamic Source Routing

DSS .....Dynamic Spectrum Sharing

EIRP .....Efective Isotropic Radiated Power

FCC .....Federal Communications Commission

GLOMOSIM .....Global Mobile Information System Simulator

IEEE .....Institute of Electrical and Electronics Engineers

ISM .....Industrial, Scientific and Medical

JVM .....Java Virtual Machine

JIST .....Java in Simulation Time

LOS .....Line Of Sight

MAC .....Media Access Control

MANETs .....Mobile Ad Hoc Networks

NLOS .....No Line Of Sight

NS-2 .....Network Simulator, versão 2

PHY .....Physical Layer

PU .....Primary User

QoS .....Quality of Service

RSL .....Received Signal Level

SIRCIM ….....Simulation of Indoor Radio Channel Impulse Response Models with Impulse Noise

SNR .....Signal Noise Ratio

SWANS .....Scalable Wireless Ad hoc Network Simulator

TV .....Televisão

UHF .....Ultra High Frequency

VHF .....Very High Frequency

WRAN .....Wireless Regional Area Network

Lista de Figuras

Lista de Tabelas

1 Introdução

Cada vez mais as redes sem fio se apresentam como uma solução interessante às necessidades do mundo moderno. O cotidiano reflete um conjunto cada vez maior de necessidades que não se adaptam mais à dependência de infra-estrutura terrestre (cabo ou fibra). Além disso, a relação de coexistência no espectro entre tecnologias distintas (ex.: Wi-Max, UMTS, Bluetooth) apresenta novas formas de comunicação e diferentes formas de utilização dos equipamentos de comunicação. Com o crescente uso de redes sem fio, o espectro eletromagnético, principalmente a banda de 2.450 Ghz, atualmente considerado como não licenciado para este tipo de comunicação, acaba por ficar congestionado e degradado a partir do momento que as crescentes interferências prejudicam a qualidade de transmissão.

Como os enlaces não licenciados utilizam faixas que podem ser utilizadas por qualquer usuário, sem a necessidade de obtenção de autorização prévia do órgão regulador, não há uma garantia de que uma transmissão considerada boa permanecerá assim por todo o tempo, podendo tornar-se degradada a partir do momento que, por exemplo, surja uma ou mais fontes de interferência. Assim, verifica-se que quanto mais fontes de interferência houver num local, menor a probabilidade dos enlaces se manterem disponíveis e, embora os equipamentos sejam projetados para suportar determinados níveis de interferências, eles não suportam as mais severas.

No Brasil, a Agência Nacional de Telecomunicações (ANATEL) é a responsável pelas questões regulatórias que, apesar de não exigir licenciamento para a exploração da banda conhecida nos Estados Unidos como *Industrial, Scientific and Medical* (ISM), ela monitora a potência máxima permitida das empresas provedoras de acesso. Entretanto, nem sempre este controle é feito com a rapidez e eficiência necessária, contribuindo para a degradação dos enlaces em termos de desempenho nos locais onde elas atuam e reduzindo a sua disponibilidade devido a interferências geradas pelas próprias comunicações dos sistemas que incidem umas sobre as outras.

Identificando esta tendência, desde 2004, com a criação do 802.22 *Work Group*, o *Institute of Electrical and Electronics Engineers (*IEEE) vem desenvolvendo padrões para redes sem fio regionais (*Wireless Regional Area Network* - WRAN) no tocante às camadas Física (PHY) e Controle de Acesso ao Meio (MAC) para uso por dispositivos (usuários secundários) que utilizarão o espectro que é alocado para o serviço

de televisão (TV) (usuários primários), sem causar interferências prejudiciais aos receptores de TV, utilizando técnicas de rádios cognitivos. Além disso, o órgão norte americano *Federal Communications Commission* (*FCC*) vem trabalhando no desenvolvimento de novas políticas de gerenciamento, como por exemplo a liberação do acesso oportunista aos canais das bandas *Very High Frequency* (*VHF*) e *Ultra High Frequency (*UHF).

Por definição, um rádio cognitivo deve realizar sensoreamento do espectro, decisão no espectro, compartilhamento do espectro e mobilidade no espectro com o objetivo de mudar seus parâmetros de transmissão, baseado nas interações com o ambiente no qual ele opera, a fim de obter o melhor nível de qualidade de transmissão possível, eficiência do espectro e, principalmente, não interferir nos usuários primários.

Neste contexto, o uso de simuladores é muito importante, pois a adoção de novos padrões só é possível a partir do estudo, simulação e prototipação de um modelo definido. Para as redes sem fio, o desenvolvimento e testes de protocolos de rede, a comparação de desempenho de diferentes protocolos, a necessidade de grande escalabilidade, as restrições de custo nos testes e a necessidade de reproduzir condições ambientais onde o cenário possa ser repetido uma infinidade de vezes, dependem do suporte das simulações.

Como objetivos gerais deste trabalho, pretende-se: 1) Identificar os mecanismos presentes na propagação de um sinal de rádio frequência; 2) Identificar as características que contribuem para a qualidade do sinal; 3) Identificar as funcionalidades existentes nos simuladores mais conhecidos apresentam em seus modelos de camada física; 4) Identificar o que o *framework* definido no trabalho [24] espera de uma camada física em termos de operação e arquitetura; 5) Definir um modelo de camada física que suporte as características necessárias para os rádios cognitivos; 6) E por fim, a implementação deste modelo e sua validação através de simulação, servindo como módulo componente do trabalho [24].

Na seção *Fundamentação Teórica* apresenta-se os principais conceitos envolvidos no escopo deste trabalho, que são: propagação do sinal, redes sem fio cognitivas e simulação em redes sem fio. Na seção *Proposta do Projeto detalha-se* o desenvolvimento da camada física aqui proposta, sua arquitetura, o funcionamento da simulação executada, qual o diferencial em relação aos simuladores de redes sem fio tradicionais e as funcionalidades que foram considerados. Por fim, as seções 4, 5 e 6

tratam da *Análise e Resultados, Conclusão e Trabalhos Futuros*, respectivamente.

1.1 Justificativa / Motivação

Apesar de ser um assunto relativamente novo na comunidade de pesquisa, as Redes Sem Fio Cognitivas prometem ser o futuro das comunicações sem fio.

Desenvolver e aperfeiçoar esta tecnologia para o uso racional e adequado dos recursos disponíveis, e assim contribuir para a evolução do conhecimento humano através do desenvolvimento científico e tecnológico é, com certeza, a maior fonte motivacional deste trabalho.

Outro fator motivacional é a possibilidade de conhecer um assunto que originalmente é de domínio da Engenharia Elétrica, mas que particularmente é bastante desafiador e ao mesmo tempo fascinante, que é como se dá o processo de propagação do sinal em ambientes sem fio, seus diferentes mecanismos de propagação e suas influências na atenuação e qualidade do sinal.

1.2 Definição do Problema

O trabalho [24], apresenta um método de classificação dos sinais de rádio sensoreados no espectro, um método de classificação de ocupação do canal de acordo com a seqüência dos sinais sensoreados, de posse das informações obtidas, possibilita a caracterização, seleção e reconfiguração no espectro. Seu objetivo é atender os requisitos de um rádio cognitivo no tocante aos problemas de reconfigurabilidade dos seus parâmetros, interferência a usuários primários, controle de sensoreamento do espectro, ocupação do espectro, decisões no espectro, compartilhamento do espectro e mobilidade no espectro. Assim, o seu *framework* apresenta um modelo de entrada/saída de dados que envolvem a interação entre a camada de RÁDIO e a camada MAC, entretanto, as questões relacionadas a camada PHY não estão no seu escopo.

O problema a ser tratado aqui é a questão da entrada/saída de sinais/dados da camada de RÁDIO para a camada PHY e sua propagação para os demais rádios, pois o

sinal a ser entregue numa transmissão sofre variações em suas características até a chegada no seu destino, dependendo principalmente da relação sinal-ruído, atenuação e perda de caminho.

1.3 Objetivos

Como objetivo deste trabalho, pretende-se definir um modelo de camada PHY que suporte a comunicação entre rádios com características cognitivas, atender os requisitos do trabalho [24] e validar este modelo através de sua implementação e simulação utilizando para isso o simulador de eventos discretos *JIST* [2], baseando-se para isso no simulador de redes sem fio *Ad Hoc* chamado *SWANS* [1].

1.3.1 Objetivos específicos

A) Estudar e avaliar os fatores que influenciam na transmissão de um sinal de rádio frequência, como potência, distância, modulação, ruído, atenuação e perda de caminho, de modo a compreender os mecanismos de propagação de sinal;

B) Definir e agregar ao modelo de camada PHY as características necessárias a um rádio cognitivo, as quais não são contempladas pelos simuladores de redes sem fio convencionais, mas que são necessárias para atender as necessidades de um rádio cognitivo, conforme a proposta do trabalho [24], e descritas em detalhes a partir do item 3 deste trabalho;

C) Agregar ao modelo de camada PHY as funcionalidades necessárias para o tratamento do sinal, aplicando a ele os fatores físicos que influenciam na sua qualidade durante sua propagação no espectro:

- Aplicando um dos modelos de enfraquecimento do sinal (*fading*);

- Aplicando um dos modelos de perda de caminho (*pathloss*).

D) Definir o ambiente de simulação a ser implementado com os seguintes requisitos:

- Deve simular a camada física conforme as características apresentadas no item anterior;

- Deve definir um modelo de ambiente terrestre com o objetivo de simular uma área física onde os rádios estariam hipoteticamente dispostos e assim poder inferir a distância entre eles e calcular a qualidade do sinal recebido através da aplicação dos fatores que a influenciam;

- Deve simular a transmissão e a recepção de sinais entre rádios através de um modelo de rede *Ad Hoc,* conforme uma interface de rádio cognitivo definida;

- Deve garantir o sincronismo de transmissão de sinais no tempo de simulação, ou seja, se um rádio transmite um sinal no tempo t+1, ele deve ser realmente transmitido aos demais rádios depois que todos os demais rádios tenham recebido as transmissões do instante t;

- Deve permitir a simulação de transmissão de sinais entre usuários primários (*Primary Users* - PU) na rede;

- Deve permitir a gravação de informações de transmissão para posterior análise e estudos;

E) Definir e implementar um gerador de sinais fictícios para fins de simulação, mas que possua as características de sinalização aqui descritas;

F) Implementar um modelo de sinal, seus atributos e a sequência de *bits* que representam os estados UNKNOWN, HIGH, LOW e PU já definida no trabalho [24], de modo a permitir que o rádio cognitivo efetue a classificação do canal;

G) Deixar o modelo o mais aberto possível para inclusão de novas funcionalidades, assim como a substituição das aqui propostas, levando em consideração as futuras necessidades, permitindo a rápida integração de modelos desenvolvidos por

outros pesquisadores.

1.4 Delimitação do Escopo

Embora o tema seja bastante abrangente e novo no cenário de pesquisa, este trabalho está limitado a apresentar os conceitos que dão embasamento à implementação do modelo proposto. Entretanto, como o trabalho [24] é de cunho inovador e o trabalho aqui desenvolvido será parte integrante dele, de certa forma se está contribuindo para a discussão em torno do tema.

O propósito da modelagem da camada PHY e sua simulação, se delimitará inicialmente para servir como como módulo do trabalho [24].

O objetivo na simulação não é ver o funcionamento de um rádio cognitivo, pois para isso, haverá a necessidade de implementação de toda a pilha de protocolos. Entretanto, espera-se dar suporte para a implementação do *framework* descrito no trabalho [24], no tocante a propagação e tratamento do sinal entre os rádios cognitivos.

Na simulação a ser executada, as camadas superiores à camada PHY serão abstraídas, mas considerar-se-ão rádios fictícios que, com capacidades restritas à transmissão e recepção de sinais, enviarão aleatoriamente sinais (conforme a estrutura definida no trabalho [24]) que serão recebidos pela camada PHY e esta aplicará os coeficientes de perda, conforme a localização do rádio e as características de cada sinal (potência, distância entre rádios, etc), e entregues àqueles rádios que estiverem dentro do alcance de cobertura.

A camada física a ser simulada possui arquitetura *ad hoc* [19], ou seja, significa um tipo de rede que não possui um nó ou terminal especial - geralmente designado como ponto de acesso - para o qual todas as comunicações convergem e que as encaminha para os respectivos destinos. Assim, uma rede de computadores *ad hoc* é aquela na qual todos os terminais funcionam como roteadores, encaminhando de forma comunitária as comunicações advindas dos terminais vizinhos.

Embora exista mais de um modelo que represente os fatores físicos que influenciam a propagação do sinal no espectro, este trabalho aplicará pelo menos um

modelo para cada fator (*fading e path loss*), pois para fins de avaliação e validação, o objetivo do *framework* é permitir o envio de um e mais sinais entre os rádios, aplicando a eles os fatores físicos que interferem em sua qualidade, assim como a sua recepção, para aqueles rádios que estiverem dentro do alcance de recepção. O modelos de recepção do sinal (modelo baseado em sinal-ruído - SNR ou o modelo baseado na taxa de erro de bit - BER), que também é um dos fatores a ser considerado na recepção do sinal, não será considerado na camada PHY, pois conforme o simulador de redes *ad-hoc* SWANS, ele é aplicado somente na camada de rádio, excluindo-se do escopo deste trabalho. Isso é compreensível, pois a camada PHY é na verdade um abstração, ou seja, ela não existe de fato, e embora um sinal seja enviado pelo rádio de origem, não há garantias de que ele chegará no(s) rádio(s) de destino, motivo pelo qual esta característica deva ser considerada (calculada) somente na chegada do sinal na camada de rádio do(s) rádio(s) destino(s), após terem sido considerados os fatores de enfraquecimento e perda de caminho do sinal aplicados na camada PHY.

Para isso, utilizar-se-ão os modelos de meio físico já implementados no SWANS, a lógica de processamento dos sinais, copiando os algoritmos e adaptando-os na medida do possível, considerando os termos e condições da licença do *software,* para que seja possível a inclusão de características cognitivas ao *framework*.

2 Fundamentação Teórica

2.1 Redes Sem Fio Cognitivas

Embora, por exemplo, os sinais de televisão (TV) sejam propagados numa faixa do espectro (de rádio-frequências) denominada licenciada, as demais tecnologias (acessos *Wi-Fi*, *bluetooth,* etc) disputam a porção que é considerada não licenciada, cuja disponibilização para aplicações (em geral) ainda não foi regulamentada. Devido ao avanço tecnológico, nota-se, cada vez mais, um aumento do número de aplicações e tecnologias que consequentemente disputam por esta fatia do espectro não licenciado. Esta crescente ocupação do espectro não licenciado levou a FCC, dos Estados Unidos da América, a realizar estudos [8] sobre a medição de ocupação do espectro, os quais demostraram a subutilização de bandas licenciadas, como as bandas de TV.

Atualmente a maioria dos espectros disponibilizados para canais de TV estão subutilizados na maior parte do tempo, enquanto que os usuários de rede sem fio compartilham uma pequena porção de espectro (2,4GHz e 5GHz). Quando num dado momento existem muitos usuários de rede sem fio comunicando, a rede fica congestionada devido ao canal limitado. Isso não acontecerá quando este espectro desocupado puder ser compartilhado, desde que não interfira nos canais de TV.

Este crescimento de serviços e dispositivos sem fio que utilizam faixas do espectro não licenciado em conjunto com a subutilização das faixas licenciadas apresentou-se como uma possibilidade de abertura destas bandas de frequência subutilizadas para acesso dinâmico oportunístico.

Seguindo este foco, conforme em [5], os rádios cognitivos, que são uma evolução natural do rádio definido por *software* que combina aspectos de inteligência artificial e de radiocomunicações, se apresentam como uma opção flexível, confiável e eficiente, pois é através de suas heurísticas que eles efetuarão o sensoreamento e medição do espectro para detectar a presença ou ausência de sinais que competem, adaptando inteligentemente suas características de operação nas condições de tempo real do ambiente de comunicação.

Conforme em [4], o uso de técnicas de cognição no software dos rádios é muito

importante, pois a proliferação de serviços e dispositivos sem fio para uso em diversos segmentos da sociedade nos mostra a dependência que temos do espectro eletromagnético. O uso destas técnicas tem a intenção de contribuir para a solução dos problemas que este novo contexto nos apresenta, a disputa pelo espectro, a interferência e a degradação na qualidade das comunicações.

Desta forma, os rádios cognitivos emergem como uma tecnologia promissora para a solução do problema da injusta utilização do espectro através do seu compartilhamento dinâmico, onde suas principais funcionalidades desejadas são: [4] sensorear periodicamente o espectro, inteligentemente efetuar a detecção de ocupação e uso do espectro, e por fim tomar a decisão de ajustar seus parâmetros para oportunisticamente comunicar entre as brechas que os usuários primários deixam no espectro. Seus principais objetivos são: maximizar a transmissão do sistema secundário enquanto garante sob controle a interferência gerada para o sistema primário. O ajuste de parâmetros como a potência de transmissão é para evitar ultrapassar o limite de interferência da rede primária e manter a conectividade entre os nós da rede, além da possibilidade de troca de canal quando o anterior apresenta-se sobrecarregado. Esta popularidade na área de pesquisa deve-se principalmente porque os rádios cognitivos permitem que o atual espectro fixo, atribuído pela FCC, seja utilizado por novos usuários (rádios cognitivos).

Assim, as redes cognitivas são capazes de adaptar seu funcionamento através da captação de estímulos externos, gerando conhecimento a partir desta interação e tomada de decisão com base no conhecimento adquirido.

Conforme definido em [22], uma rede cognitiva pode ser definida como uma rede que possui um processo cognitivo que pode perceber as condições da rede atual e então planejar, decidir e atuar nessas condições. A rede pode aprender com estas adaptações e usar estes conhecimentos para tomar futuras decisões, levando em consideração os objetivos fim-a-fim, ou seja, todos os elementos envolvidos na transmissão de um fluxo de dados.

Desta forma, as redes cognitivas desafiam a noção atual da escassez de espectro e encaram o problema como sendo um problema de acesso e compartilhamento ao espectro.

Neste contexto pode-se vislumbrar diferentes tipos de sistemas de acesso sem

fio operando na mesma banda de frequência de um sistema primário e outros tantos sistemas secundários, desde que estes também observem o ambiente e reajam inteligentemente sobre suas mudanças a fim de otimizar o comportamento do sistema como um todo.

Conforme citado anteriormente, desde 2004, com a criação do *802.22 Work Group*, a IEEE vem desenvolvendo pesquisas na busca de padrões para a WRAN no tocante a camada PHY e MAC para proporcionar acesso dinâmico ao espectro (*Dynamic Spectrum Access* – DSA) e compartilhamento dinâmico ao espectro (*Dynamic Spectrum Sharing* – DSS).

2.2 Necessidades de um Rádio Cognitivo

Como o objetivo principal de um rádio cognitivo é, através do sensoreamento do espectro, proporcionar mais oportunidades de acesso ao espectro sem interferir com as operações as redes licenciadas, verifica-se no trabalho [24] que o acesso dinâmico ao espectro depende de quatro características básicas, que são:

- Sensoreamento do espectro: O rádio cognitivo deve ser capaz de identificar, com a devida qualidade, os usuários primários, efetuar o controle de sensoreamento, a cooperação (em modo cooperativo) e, a fim de atingir o melhor estado do sensoreamento, definir o tempo de observação, conhecimento da largura de banda, estabelecer um limite para o método de sensoreamento e conhecimento do sinal-ruído;

- Gerenciamento do espectro (ou decisão): O rádio cognitivo deve ser capaz de efetuar a caracterização, seleção e reconfiguração do espectro. Na caracterização espera-se que o rádio cognitivo apresente/defina, através de uma coleção de informações sobre o canal em relação a um (ou mais) aspectos, uma possível classificação do canal que, conforme a literatura, pode ser vago/não-vago, alta-oportunidade/alta-atividade, alta-oportunidade/baixa atividade, baixa-oportunidade/alta-atividade e baixa-oportunidade/baixa-atividade. Na seleção espera-se que o rádio cognitivo efetue a seleção da banda do espectro apropriada, entretanto, alguns fatores devem ser levados em consideração como a qualidade de serviço (QoS) (baseada no nível de interferência sobre a transmissão do usuário primário), a maximização das oportunidades de descoberta e minimização do atraso na localização de um canal desocupado, além dos

requisitos clássicos de rede para maximizar o *throughput* e minimizar o atraso. Na reconfigurabilidade espera-se que o rádio cognitivo efetue, com base nas etapas anteriores, a sua reconfiguração carregando as necessidades de seleção de canal, seleção de modulação, configuração de largura de banda, configuração do tempo de observação, configuração do tempo de transmissão e configuração de potência.

- Compartilhamento do espectro: O rádio deve ser capaz de alocar recursos, alocar canal, alocar potência e efetuar acesso inteligente ao espectro (acesso randômico, tempo intervalado, híbrido).

É fato a possibilidade de ocorrência de duas ou mais redes cognitivas estarem transmitindo numa mesma porção do espectro eletromagnético. Assim, verifica-se a necessidade de coexistência, pois a não detecção do usuário primário não significa que o espectro esteja desocupado, e o compartilhamento consiste em prevenir múltiplas redes cognitivas colidindo em sobreposição numa mesma porção do espectro.

- Mobilidade no espectro: O rádio cognitivo deve ser capaz de efetuar a troca de sua conexão para outra banda não utilizada do espectro, que ocorre sempre que há detecção de usuários primários (*PU*), perda de conexão devido a mobilidade dos usuários envolvidos na comunicação ou quando a banda do espectro não consegue atingir os requisitos de *QoS*.

O presente trabalho, no que for necessário dentro de seu escopo, terá como base os requisitos do *framework* para rádio cognitivo, explicitados no trabalho [24], que são:

- Detecção de PU;

- Classificação do canal;

- Seleção do canal / configuração de largura de banda;

- Seleção de modulação;

- Configuração do tempo de observação;

- Configuração do tempo de transmissão;

- Configuração da potência;

- Estabelecer um limite de sensoreamento;

- Ter ciência do sinal-ruído;

- Qualidade na ciência de detecção ;

- Ciência da largura de banda;

Porque estes são os requisitos necessários para alcançar os seguintes outros requisitos:

- Controle do sensoreamento;

- Cooperação;

- Caracterização do espectro;

- Seleção do espectro;

- Reconfiguração;

- Compartilhamento do espectro intra-rede;

- Compartilhamento do espectro inter-rede;

- Espectro *handoff*;

2.3 Propagação do sinal

O estudo da propagação tem como objetivo entender como a energia é transportada ao longo do meio desde o transmissor até o receptor. Em [18], apesar de todas as facilidades das redes sem fio, elas se deparam com incertezas durante a propagação do sinal, principalmente devido ao comportamento aleatório do meio sujeito a variações em seu estado devido a fatores que são responsáveis por atenuações, como obstáculos e propagação por diferentes direções.

O desempenho de um sistema de comunicação sem fio está diretamente relacionado às limitações do canal de rádio. Neste contexto o que está em jogo é o caminho que o sinal segue do transmissor ao receptor, tenha ele obstáculos ou não, e a força do sinal que chegará no receptor. Em [16] observa-se que em projetos de redes sem fio deve-se levar em consideração o fator quantidade de potência recebida em pontos distintos que pode variar devido a distância ou devido ao meio. Isso porque a potência do

sinal recebido deve ser suficiente para ser detectada (conforme a sensibilidade do receptor) e deve ser superior ao ruído para que possa ser detectada sem erros. A dispersão do sinal é outro fator relevante, pois é ocasionado devido ao enfraquecimento do sinal devido a caminhos múltiplos (*multipath*), onde observa-se que réplicas do sinal chegam ao receptor em diferentes momentos por receberem diferentes atrasos no percurso e por o percorrerem com distâncias diferentes. Isso faz com que o meio/terreno defina a taxa máxima que uma transmissão pode ser operada. Caminhos múltiplos podem aumentar ou diminuir a força do sinal recebido, dependendo se a onda ocorrida por um dos caminhos múltiplos interfere construtivamente ou destrutivamente.

Os modelos de propagação da onda eletromagnética consideram que a potência recebida é o parâmetro mais importante e podem ser descritos pela física de três mecanismos básicos, que são:

- **reflexão**: ocorre quando uma onda eletromagnética em propagação colide com um objeto de dimensões muito grandes em comparação com o comprimento de onda da onda que se propaga, ocorrendo na superfície da terra, nos prédios e paredes;

- **difração**: ocorre quando o caminho entre o transmissor e o receptor é obstruído por uma superfície que possui irregularidades afiadas (arestas);

- **dispersão**: ocorre quando o meio pelo qual a onda trafega possui objetos de dimensões pequenas, em comparação com o comprimento de onda, e o número de obstáculos por volume unitário é grande.

Os mecanismos descritos acima permitem que haja recepção do sinal em lugares onde não há linha de visada, principalmente devido ao espalhamento e difração. A absorção do sinal em obstáculos também contribui com a atenuação do sinal, porém representa uma parcela pequena.

Os modelos de propagação tradicionais consideram a previsão de uma intensidade média do sinal recebido a determinada distância do transmissor, além da variabilidade da intensidade do sinal em determinado local. Desta forma, quando podem caracterizar a intensidade do sinal para grandes distâncias, são úteis na estimativa da área de cobertura do rádio transmissor e são considerados modelos de propagação em larga escala. Quando as distâncias são muito curtas ou para curtas durações e a intensidade dos sinais é caracterizado por flutuações rápidas, estes são considerados

modelos de propagação em pequena escala.

O sinal recebido é uma soma de muitas contribuições vindas de diferentes direções e a sua variabilidade depende da distância entre transmissor e receptor, sendo atenuado em pequena escala quando o transmissor se afasta do receptor por pequenas distâncias e atenuado em larga escala quando esta distância é muito grande. É este nível de sinal médio local que é previsto pelos modelos de propagação aqui descritos.

2.3.1 Sistemas de comunicação

A análise de sistemas de comunicação sem fio é muito importante no desenvolvimento de um projeto para um ambiente desejado e também para predizer, com relativa certeza, o desempenho deste sistema antes de sua fabricação e implantação.

Antes de planejar a implantação de uma rede sem fio é importante entender os parâmetros essenciais que influenciam no desempenho de cada enlace individual que são: a força do sinal recebido, o ruído que o acompanha e quaisquer danos adicionais ao canal, como atenuação, caminhos múltiplos e interferência. Para este planejamento deve-se contabilizar a efetiva potência isotropicamente irradiada (*EIRP - Effective Isotropically Radiated Power*), todas as perdas no enlace antes do recebimento do sinal, assim como o limiar de ruído no receptor para determinar o nível de sinal requerido para sua detecção.

A tolerância de um enlace é obtida comparando-se a força do sinal recebido esperado para a sensibilidade de um receptor, ou seja, a partir de qual valor o receptor poderá o não receber um sinal. Este valor é a medida de quanta tolerância há no enlace de comunicação entre o ponto onde há operação e o ponto onde não é mais possível fechar o enlace, conforme a equação (1) abaixo

$$\text{Link margin} = \text{EIRP} - L_{Path} + G_{Rx} - \text{TH}_{Rx} \quad (1)$$

onde:

- EIRP – É a efetiva potência isotropicamente irradiada, medida em dBW ou em dBm;

- L_{Path} – É a perda de caminho total (*path loss*), medida em dB;

- G_{Rx} – É o ganho de recepção, medida em dB;]

- TH_{Rx} – É o limiar do receptor, ou seja, o nível mínimo de sinal recebido para permitir uma operação confiável (ou taxa de erro de *bit* desejada), medida em dBW ou em dBm.

Esta tolerância de enlace depende de muitos fatores como a modulação utilizada (pois ela altera o SNR necessário), a potência transmitida, o ganho de antena, as perdas físicas relacionadas aos cabos entre o transmissor e a antena, mas dentre eles o principal é a perda de caminho devido a sua magnitude em relação aos outros termos, incluindo perda de propagação geométrica ou perda no espaço livre, assim como fatores ambientais.

2.3.1.1 Perda de Caminho (Path Loss)

A atenuação relacionada ao caminho é a redução na densidade da potência de uma onda eletromagnética enquanto ela se propaga através do espaço. Ela é considerada como sendo o maior componente na análise e desenvolvimento de um projeto de um sistema de telecomunicações.

Comumente associada à propagação da rádio frequência, a perda de caminho está relacionada a muitos fatores como a perda no espaço livre, perda devido a caminhos múltiplos, refração, difração (quando parte da onda eletromagnética é obstruída por um obstáculo opaco), reflexão, absorção (também chamada de perda de penetração – que é quando o sinal passa através de mídia não transparente) e também por efeitos relacionados ao contorno do terreno, do ambiente (rural, urbano, com ou sem vegetação),

à mídia de propagação (ar seco ou úmido), à distância entre transmissor e receptor, assim como à altura e localização da antena. Outros fatores que contribuem para a perda de sinal estão relacionados à limitação de banda, à modulação utilizada, a filtros de transmissão, perdas por fenômenos atmosféricos como a chuva, assim como o material que cobre a antena, o cabo que conecta a antena ao dispositivo do rádio, etc.

O cálculo da perda de caminho é normalmente chamada de previsão e seu valor exato só é possível para casos simples (modelos *free-space* e *flat-earth*), pois na prática ela é calculada utilizando uma variedade de aproximações.

No estudo de comunicações sem fio, a perda de caminho pode ser representada por um expoente, cujo valor está no alcance de 2 a 4 (onde 2 é para propagações no espaço livre e 4 é para ambientes com características de perda e para o caso de total reflexão especular da superfície terrestre – modelos *free-space e flat-earth*).

2.3.1.2 Ruído

Considerado como a chave das características que determinam o desempenho do enlace, em [21] verifica-se que o limiar de recepção ou sensitividade é a força mínima que um sinal deve ter para uma operação aceitável. Se este limiar não é conhecido ou ou especificado, então deve ser fornecida a taxa de sinal-ruído requerida ou a taxa de energia por *bit,* para a densidade espectral de ruído.

Este ruído é definido pelas características do receptor e pela largura de banda, causado pela agitação térmica dos elétrons na parte frontal do receptor, modelado como tendo uma distribuição de amplitude Gaussiniana. Além disso, ele é modelado como tendo uma densidade espectral de potência constante de N0/2, ou seja, a densidade da potência é a mesma para todas as frequências de interesse, considerado assim como *White*. Como o ruído térmico é aditivo em sua essência, ele é modelado como *AWGN (Additive White Gaussian Noise).*

Dado uma carga, a potência do ruído térmico teórico é dada pela equação (2) abaixo,

$$N = kT_0B \tag{2}$$

onde:

- k - 1.38 x 10 -23 J/K é a constante de Boltzman;

- T_0 - 290K é a temperatura padrão do ruído;

- B - é a largura de banda equivalente do ruído.

O resultado desta equação (2) representa a quantidade de potência de ruído na entrada de um receptor perfeito, como uma antena.

Assim, entende-se que como o ruído é definido pelas características do receptor, ele não é reduzido pelas perdas, mas o sinal sim. Consequentemente, a taxa de sinal ruído na saída é reduzida, diminuindo a figura do ruído. Assim, qualquer perda encontrada entre a saída da antena do transmissor e a entrada da antena do receptor irá contribuir para o aumento do ruído total, que em ambientes terrestres é computado pela simples soma das perdas ao ruído do receptor.

2.3.1.3 Interferência e Recepção do Sinal

A computação da interferência e do ruído em cada receptor é um fator crítico na modelagem da comunicação sem fio, pois ela é a base da relação sinal ruído (SNR) e possui forte correlação com a taxa de erro do (*Frame Error Rate* – FER).

Além disso, [21] a potência de interferência e o ruído são calculados como a soma de todos os sinais existentes no canal diferentes daquele que está sendo recebido pelo rádio, mais o ruído térmico no receptor. E assim o resultado desta potência será usado como base do cálculo do SNR, o qual determina a probabilidade de sucesso na recepção do sinal para um dado *frame*.

Um dos tipos de interferência que pode degradar seriamente o desempenho é a interferência *intersymbol* que é quando há sobreposição de pulso no tempo devido ao

transbordamento do canal, como consequência de limitações na largura de banda do transmissor, receptor e do próprio canal.

Como fontes de interferências, pode-se citar:

- Múltiplas portadoras sendo transmitidas do mesmo transmissor;

- Um forte sinal oriundo de um canal adjacente;

- Interferências externas de outros transmissores.

Ainda, se a interferência encontra-se na mesma freqüência de interesse, diz-se que ela é interferência de co-canal, que são causadas por harmônicas de tipos diferentes de sistemas como radiadores não intencionais, ou sinais de um sistema semelhante que está a certa distância onde é possível o compartilhamento do espectro.

Outro tipo de interferência que pode causar sérios problemas é chamada de interferência do canal adjacente, e é aquela onde sua origem está próxima da frequência de interesse.

Normalmente na previsão de enlaces, considera-se um percentual de interferência externa que fornece uma margem para a degradação do ruído. Por exemplo, dada uma margem de interferência de 1dB significa que o ruído terá algo como 1dB a mais do que num ambiente livre de interferência.

Dado um valor de SNR, a recepção do sinal pode seguir dois modelos: limiar de SNR ou o baseado em BER (*Bit Error Rate*). O modelo baseado em limiar de SNR – usa o valor do SNR comparando-o com o limiar de SNR do rádio, aceitando somente sinais cujo SNR que estejam acima do limiar de SNR, a qualquer momento da recepção. O modelo baseado em BER decide probabilisticamente se cada *frame* é recebido com sucesso ou não, baseado no comprimento do *frame* e no BER deduzido pelo SNR e pelo esquema de modulação usado no *transciever*. O modelo avalia cada segmento do *frame* com um valor de BER toda vez que a potência de interferência muda. Este modelo é considerado mais realístico e mais preciso do que o modelo baseado em limiar de SNR. O modelo baseado em limiar de SNR tem um custo computacional menor e pode ser uma boa abstração se o comprimento de cada *frame* é longo. Assim, em cada recepção de sinal, o SNR para um dado sinal é recalculado toda vez que a potência de interferência muda.

2.3.1.4 Modulação (Visão Geral)

Conforme [18], o objetivo final das técnicas de modulação (e demodulação) é transportar o sinal da mensagem por um canal de rádio com a melhor qualidade possível e ocupando a menor quantidade de espectro de rádio, melhorando assim a qualidade geral do enlace. Um esquema de modulação desejável deve oferecer baixas taxas de erro de *bit* em baixas relações sinal-ruído e funcionar bem em condições de caminho múltiplo e atenuação. Isso é bom em dois aspectos: desempenho da taxa de erro de *bit* e eficiência em termos de largura de banda. O desempenho de um modelo de modulação normalmente é medido em termos de sua eficiência de potência e eficiência de largura de banda. A eficiência de potência descreve a capacidade de uma técnica de modulação de preservar a fidelidade da mensagem digital em níveis de potência mais baixos. Em um sistema de comunicação digital, para aumentar a imunidade ao ruído, é necessário aumentar a potência do sinal. Porém, a quantidade pela qual a potência do sinal deve ser aumentada para se obter um certo nível de fidelidade (erro de *bit* aceitável) depende do tipo de modulação empregado. A eficiência de largura de banda descreve a capacidade de um esquema de modulação em adequar uma quantidade de dados dentro de uma largura de banda limitada. A taxa de dados está relacionada a largura de banda pelo fato de que o aumento da taxa de dados diminui a largura do pulso de um símbolo digital, aumentando a sua largura de banda. Num sistema de comunicação digital normalmente existe uma compensação entre eficiência de potência e eficiência de largura de banda.

2.3.2 Previsão de Enlace

A análise e previsão de enlace é a compilação de todos os ganhos e perdas no enlace de comunicação, calculado através da soma da potência de transmissão (em dBm ou dBW) com todos os ganhos e perdas (em dB) relevantes do Enlace e subtraindo este valor do nível de sinal (na mesma unidade do sinal transmitido) requerido na recepção. Quanto maior este valor, mais robusto pode ser considerado o enlace [21] .

Para o desenvolvimento de um sistema de comunicação, deve-se contabilizar

também as perdas geradas por fatores cuja presença seja intermitente, como a chuva, a fim de garantir que o enlace esteja de fato disponível pela porcentagem de tempo requerida.

Fatores a serem computados na previsão de enlace*:*

- EIRP – É a potência transmitida, mais o ganho de antena, menos qualquer outro fator de perda (em dBm ou dBW);

- Perda de Caminho (*Path Loss*) – É a soma de todas as perdas no espaço livre mais os efeitos ambientais, incluindo o multi caminho;

- Ganho do receptor – É o ganho da antena menos as perdas relacionadas a qualquer radome (estrutura para proteção da antena), à polarização e a perdas pontuais. Assim, o nível de sinal do receptor é dado pelo EIRP menos as perdas do caminho mais o ganho do receptor;

- Margem do enlace – O resultado pode ser o nível de sinal recebido esperado (RSL), a taxa de sinal-ruído (SNR), a taxa de portadora-ruído (CNR) ou a taxa de energia-ruído do *bit* (Eb/N0). Se o RSL for determinado, então ele pode ser comparado ao sinal mínimo detectável (MDS) ou ao limiar do sistema;

- Taxa de Sinal-Ruído (SNR) – Deve ser calculado primeiro o ruído conforme a equação (2), em seguida o sinal-ruído e por fim como segue na figura (3):

$$SNR = RSL - N(ruído) \tag{3}$$

Para sistemas de telecomunicações, a *Eb/N0* (taxa de energia por *bit* / densidade de ruído espectral) e o BER (*Bit Error Rate*) do sinal recebido são os fatores determinantes na detecção do sinal. Os valores necessários de *Eb/N0* para um dado BER são fornecidos pelo desenvolvedor do modem.

2.3.3 Perda de Caminho em Larga Escala

O modelo de perda no caminho é um fator importante, pois define a perda de potência média do sinal no caminho. Possui dois modelos: o *Two Ray* e o *Free Sapce*.

O *Two Ray* é adequado para canais com LOS e ambientes urbanos. Seu uso para MANETs é justificado pelas similaridades dos ambientes (potência baixa de transmissão e antenas com altura baixa). Já o *Free Space* é usado como um modelo de referência básica e é também considerado a ser um modelo de propagação ideal, onde mesmo os nós longes do transmissor podem receber pacotes, resultando em poucos *hops* para atingir o destino final numa MANET. Os resultados das simulações com este modelo tendem a apresentar um melhor resultado em relação a outros modelos. Entretanto, uma propagação de um sinal com pouca perda de potência pode causar forte interferência nas transmissões concorrentes, podendo não apresentar um melhor desempenho.

2.3.3.1 **Modelo de Propagação no Espaço Livre (*Free Space*)**

Modelo usado para prever a intensidade do sinal recebido quando o transmissor e o receptor possuem um caminho livre de obstáculos. Normalmente conhecido como “Perda no Espaço Livre”, ele é na verdade o resultado da energia espalhada a partir de um ponto de origem, onde apenas uma parte da energia irradiada estará disponível em um ponto de recepção para ser interceptada por uma antena, sendo considerado um modelo simplista de melhor caso. O modelo *free-space* tem um custo computacional baixo mas não é a forma mais precisa para modelar os efeitos complexos da propagação do sinal, além do fato de não ser realístico. Ele sempre consegue uma recepção perfeita se os dois nós estiverem dentro do alcance de recepção, podendo resultar em resultados irreais em certos cenários. É considerado também como um modelo em grande escala e assim como os demais modelos de propagação, ele prevê que a potência diminui como uma função da distância entre transmissor e receptor, elevada a alguma potência. A potência recebida por uma antena, no espaço livre, que está separada de uma antena transmissora, irradiando, por uma distância “*d”* é dada pela equação (4) do espaço livre de Friis [7] (e suas complementares), a qual considera a

existência de condições ideais de transmissão, e que mostra que a potência recebida cai conforme o quadrado da distância entre o transmissor e o receptor,

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L} \quad (4)$$

$$\lambda = \frac{c}{f} = \frac{2\pi c}{\omega_c}$$

$$G = \frac{4\pi A_e}{\lambda^2}$$

onde:

- P_t - potência transmitida;

- $P_r(d)$ - potência recebida;

- G_t - ganho da antena transmissora (o ganho da antena está relacionado a sua abertura efetiva, Ae);

- G_r - ganho da antena receptora (o ganho da antena está relacionado a sua abertura efetiva, Ae);

- d - distância (em metros) entre Transmissor e Receptor;

- L - fator de perda do sistema não relacionado a propagação (L>=1), e

normalmente são devidas à atenuação da linha de recepção, perdas de filtro e perdas de antena. Um valor igual a 1 significa nenhuma perda no sistema;

- λ - é o comprimento de onda em metros;

- Ae - está relacionada ao tamanho físico da antena;

- f - é a frequência da portadora em Hertz;

- ωc - é a frequência da portadora em radianos por segundo;

- c - velocidade da luz em m/s;

Em relação a perda do caminho (*pathloss*), esta representa a atenuação do sinal como uma quantidade positiva, é medida em dB e é definida como a diferença (em dB) entre a potência transmitida efetiva e a potência recebida, podendo ou não incluir os ganhos da antena. Abaixo as equações (5,6) de perda de caminho considerando e não os ganhos da antena:

$$PL\,(\mathrm{dB}) = 10\log\frac{P_t}{P_r} = -10\log\left[\frac{G_t G_r \lambda^2}{(4\pi)^2 d^2}\right] \tag{5}$$

$$PL\,(\mathrm{dB}) = 10\log\frac{P_t}{P_r} = -10\log\left[\frac{\lambda^2}{(4\pi)^2 d^2}\right] \tag{6}$$

Esta potência transmitida efetiva está relacionada à potência efetiva irradiada, ou seja, ela indica a potência máxima irradiada comparada com uma antena bipolar de meia onda, com um ganho de 1,64 dB.

Deve-se considerar que a equação “1” só é válida quando d>0, por isso, os modelos de propagação em larga escala consideram uma distância próxima (d0) como sendo um ponto de referência para a potência recebida. Assim, usando-se a equação (1),

a potência recebida no espaço livre a uma distância maior que d0 é dada pela equação (7).

$$P_r(d) = P_r(d_0)\left(\frac{d_0}{d}\right)^2 \qquad d \geq d_0 \geq d_f \tag{7}$$

2.3.4 Atenuação em Pequena Escala e Caminhos Múltiplos

A atenuação em pequena escala (*fading*) descreve flutuações rápidas das amplitudes, fases ou atrasos de caminhos múltiplos (*multipath*) de um sinal de rádio em um curto período de tempo ou distância, de modo que os efeitos da perda de caminho em larga escala possam ser ignorados.

Observa-se também a variação da potência do sinal no receptor causada pela mobilidade do nó que cria diversas condições de caminho (*multipath*) a partir do transmissor. O *Fading* possui dois modelos para descrever ambientes MANET: a distribuição Rayleigh e a distribuição Ricean. O modelo *Additive White Gaussian Noise* (AWGN) se refere a uma condição de canal ideal, onde não ocorre fading do sinal.

A atenuação é causada pela interferência de duas versões do sinal transmitido que chegam ao receptor em tempos distintos, cujas ondas se combinam na antena do receptor, resultando num sinal que pode variar muito em amplitude e fase, dependendo da distribuição da intensidade, do tempo relativo de propagação destas ondas e da largura de banda do sinal transmitido.

2.3.4.1 Propagação em Caminhos Múltiplos

Verifica-se que o caminho múltiplo gera os seguintes efeitos:

- Mudanças rápidas na intensidade do sinal em uma pequena distância ou

intervalo de tempo;

Modulação de frequência aleatória por causa de mudanças Doppler variáveis em diferentes sinais de caminho múltiplo;

- Dispersão de tempo (ecos) causada por atrasos de propagação de caminhos múltiplo;

A atenuação ocorre em áreas urbanas porque a altura das antenas móveis é menor que a altura das estruturas ao seu redor, ocasionando mais de um caminho na linha de visão. Entretanto, verifica-se que até mesmo quando se tem, teoricamente, um único caminho para a linha de visão, o efeito do caminho múltiplo é observado devido às reflexões no solo e nas estruturas ao seu redor. Isso faz com que as ondas de rádio cheguem de diferentes direções e com atrasos de propagação distintos.

O sinal se torna atenuado ou distorcido porque ao ser combinado vetorialmente na antena receptora, ele pode estar constituído de um grande número de ondas planas com amplitudes, fases e ângulos de chegada distribuídos aleatoriamente. E até o movimento dos objetos ao redor da antena receptora podem fazer com que o sinal se torne atenuado.

Quando há movimento entre o receptor e o transmissor, observa-se que cada onda do caminho múltiplo apresenta uma alteração na frequência do sinal, chamada de deslocamento Doppler.

2.3.4.2 Fatores que Influenciam a Atenuação em Pequena Escala

São eles:

- Propagação de caminho múltiplo: É o fato do sinal transmitido ser dispersado/refletido a ponto de gerar múltiplas versões, cuja energia foi dissipada em amplitude, fase e tempo. Esta variabilidade de amplitude e fase causam flutuações na intensidade do sinal, ocasionando a atenuação em pequena escala e a distorção do sinal.

- Velocidade da estação móvel: O movimento relativo entre transmissor e receptor resulta em modulação de frequência aleatória devido aos diferentes

deslocamentos Doppler que será positivo se estiver se aproximando e negativo se estiver se afastando.

- Velocidade dos objetos ao redor: Os objetos no canal de rádio que estão em movimento proporcionam um deslocamento Doppler variável no tempo sobre os componentes do caminho múltiplo. A atenuação em pequena escala será conduzida pelo objeto que estiver em maior velocidade, entretanto, se a velocidade dos objetos for menor, a atenuação deles pode ser ignorada, considerando somente a atenuação da estação móvel.

- Largura de banda de transmissão do sinal: A atenuação do sinal em pequena escala (num dado ponto) não será considerada se a largura de banda do sinal de rádio transmitido for maior que a largura de banda do canal de caminho múltiplo, embora o sinal recebido seja distorcido. Agora, se o sinal transmitido tenha largura de banda estreita em comparação como canal, a amplitude será afetada, mas o sinal não será distorcido. Assim, verifica-se que as estatísticas acerca da intensidade do sinal em pequena escala e a probabilidade de distorções no sinal recebido, estão relacionadas às amplitudes e atrasos específicos do canal de caminho múltiplo, além da largura de banda do sinal transmitido.

2.3.4.3 Deslocamento Doppler

Quando uma estação móvel está em movimento em direção à chegada de uma onda, o deslocamento de Doppler é positivo e a frequência recebida é aumentada, entretanto, quando o movimento é de afastamento, este valor é negativo e a frequência recebida é diminuída.

O deslocamento de Doppler está relacionado à velocidade da estação móvel e o ângulo espacial entre a direção do movimento da estação móvel e a direção da chegada da onda.

2.3.4.4 Tipos de Atenuação em Pequena Escala

Ocorre que numa transmissão de rádio, os sinais transmitidos podem ter natureza diferenciada que sofrerão diferentes tipos de atenuação, conforme as

características do canal, ou seja, o que definirá o tipo de atenuação são os parâmetros do sinal (largura de banda, período de símbolo, etc) e os parâmetros do canal (espalhamento de atraso rms e espalhamento Doppler).

Assim, através dos mecanismos de dispersão de tempo, dispersão de frequência, a natureza do sinal transmitido, do canal e da velocidade, observa-se quatro efeitos distintos possíveis que são:

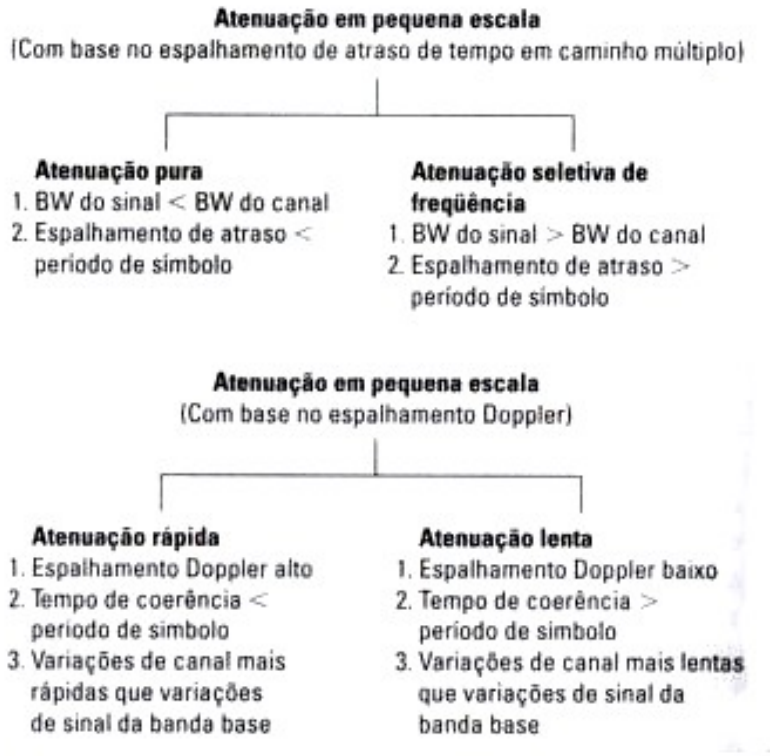

Figura 1: Tipos de atenuação em pequena escala [Rappaport, 2008]

a) Atenuação em pequena escala (com base no Espalhamento de atraso de tempo em caminho múltiplo):

- Dispersão de tempo (atenuação pura/uniforme): Embora a intensidade do sinal recebido mude com o tempo, devido a flutuações no ganho do canal causadas pelo caminho múltiplo, neste modelo a estrutura de caminho múltiplo do canal é tal que as características espectrais do sinal transmitido são preservadas no receptor. A atenuação uniforme é percebida quando um canal de rádio móvel tem um ganho constante e

resposta de fase linear em uma largura de banda maior que a largura de banda do sinal transmitido. Estes canais que apresentam atenuação uniforme, devido a largura de banda do sinal aplicado ser *estreita* quando comparado com a largura de banda de atenuação uniforme do canal, também são conhecidos como canais de variação de amplitude ou canais de banda estreita. Além disso, a atenuação por eles causada são profundas, e durante estes períodos, pode exigir 20 ou 30dB a mais de potência no sinal do transmissor para conseguir baixas taxas de erro de bit;

- Atenuação seletiva de frequência: É causada por atrasos de caminho múltiplo que se aproxima ou excedem o período de símbolo do símbolo transmitido. Ela é percebida no sinal recebido quando um canal de rádio móvel tem um ganho constante e resposta de fase linear em uma largura de banda menor que a largura de banda do sinal transmitido, ou seja, o espectro do sinal transmitido possui uma largura de banda maior que a do canal. Nessas condições, a resposta ao impulso do canal possui um espalhamento com atraso em caminho múltiplo maior que a largura de banda recíproca da forma de onda da mensagem transmitida. E assim, o sinal recebido inclui múltiplas versões da forma de onda transmitida que estão atenuadas e atrasadas no tempo, ficando o sinal distorcido. A atenuação seletiva de frequência deve-se à dispersão no tempo dos símbolos transmitidos dentro do canal, ocasionando interferência entre símbolos. Neste modelo, a sua modelagem é mais difícil porque cada sinal do caminho múltiplo precisa ser modelado e o canal deve ser considerado como um filtro linear.

b) Atenuação em pequena escala (Com base no Espalhamento Doppler):

- Um canal pode ser classificado como de atenuação rápida ou lenta dependendo da velocidade com que o sinal transmitido muda em comparação com a taxa de mudança do canal. Entretanto, seja o canal especificado como de atenuação rápida ou lenta, isso não especifica se o canal tem atenuação uniforme ou se é seletivo de frequência. A velocidade da estação móvel ou a velocidade dos objetos do canal e a sinalização de banda base determina se um sinal sofre atenuação rápida ou lenta. A atenuação rápida ou lenta trata do relacionamento entre taxa de mudança no canal e o sinal transmitido.

- Canal de atenuação rápida: Neste modelo a resposta ao impulso do canal muda rapidamente dentro da duração do símbolo, ou seja, o tempo de coerência do canal

é menor que o período símbolo do sinal transmitido, causando dispersão de frequência devido ao espalhamento Doppler, que leva a distorção do sinal. Sob o domínio da frequência, a distorção do sinal devida à atenuação rápida aumenta com o aumento do espalhamento Doppler relativo à largura de banda do sinal transmitido. A atenuação rápida, só lida com taxa de mudança do canal devido ao movimento, ou seja, um canal com atenuação uniforme e rápida, é um canal em que a amplitude da função delta varia mais rápido que a taxa de mudança do sinal transmitido. Já num canal seletivo de frequência com atenuação rápida as amplitudes, fases e atrasos de tempo de qualquer um dos componentes de caminho múltiplo variam mais rápido que a taxa de mudança do sinal transmitido.

- Canal de atenuação lenta: Neste modelo, a resposta ao impulso do canal muda lentamente em relação ao sinal transmitido, podendo o canal ser considerado estático por um ou vários intervalos de largura de banda. No domínio da frequência, isso significa que o espalhamento de Doppler do canal é muito menor que a largura de banda do sinal.

2.3.4.5 Distribuição de Atenuação de Rayleigh

É um modelo estatístico que descreve os efeitos do ambiente de propagação num sinal de rádio frequência onde o ambiente que espalha o sinal de rádio, antes dele chegar no receptor, possui muitos objetos. Ele descreve a natureza estatística variável no tempo do envelope recebido de um sinal de atenuação uniforme, ou o envelope de um componente do caminho múltiplo individual. Normalmente ele é usado para condições de alta mobilidade e sem linha de visão (No Line Of Sight - NLOS) entre os nós, entretanto, se há pelo menos uma linha de visão (LOS) entre o transmissor e o receptor, então o modelo Ricean é o mais indicado. Também é aceito como um modelo razoável para propagação do sinal troposférico e ionosférico, assim como em ambientes urbanos com muitas construções.

2.3.4.6 Distribuição de Atenuação de Ricean

A distribuição de envelope de atenuação em pequena escala é Ricean quando existe um componente de sinal dominante sem atenuação, como por exemplo um caminho de propagação na linha de visão, ou seja, o efeito de um sinal dominante chegando com muitos sinais de caminho múltiplo mais fracos, faz surgir a distribuição de Ricean. Assim, os demais componentes do caminho múltiplo que chegam em diferentes ângulos são sobrepostos a este sinal dominante e sem atenuação. A medida que este sinal dominante enfraquece, o sinal composto assemelha-se a um sinal de ruído que tem um envelope que é chamado de Rayleigh. O Ricean é usado para condições onde há linha de visão (Line Of Sight - LOS) e o seu parâmetro, chamado de fator K, é que controla a potência do sinal para o caminho com LOS em relação a potência do sinal para o caminho com NLOS.

2.4 Simuladores para Redes Sem Fio

Pessoas em muitas áreas da ciência e da indústria fazem o uso de simuladores desde os primórdios das redes de computadores para descrever de forma fictícia o comportamento de eventos discretos e que muitas vezes procuram reproduzir condições ambientais reais. Por isso, mesmo sendo dispendioso em termos de tempo de processamento e memória requeridos, o desenvolvimento deles tem chamado a atenção de pesquisadores e desenvolvedores. Isso deve-se ao fato de que em testes reais as condições de transmissão sem fio podem variar de experimento para experimento, tornando impossível comparar, por exemplo, o desempenho de diferentes protocolos, sob circunstâncias idênticas. Já nas simulações, um cenário pode ser repetido exatamente quantas vezes for necessário. Outro fator a considerar é que em simulações reais a configuração de equipamentos é uma tarefa que consome muito tempo, o que torna-se impossível para testes de escalabilidade de protocolo com um grande número de nós.

Entende-se por simuladores de eventos discretos [6] o fato da simulação ser executada em passos discretos (tempo) e que os eventos são utilizados como meio de

comunicação entre as entidades de simulação. Os simuladores assim ditos permitem modelar uma rede de computadores arbitrária, especificando o comportamento de ambos os nós e canais de comunicação. Além de permitir a simulação e testes de novos sistemas que muitas vezes sequer foram desenvolvidos em hardware, os simuladores proporcionam condições ambientais que podem ser repetidas uma infinidade de vezes. Por exemplo, para a simulação de redes sem fio móveis *ad hoc* (*Mobile Ad Hoc Networks* - MANETs) muitas são as vantagens, como: reproduzir um ambiente com um grande número de nós, condições de transmissão exatas e precisas, testar desempenho de diferentes protocolos, não perder tempo configurando equipamentos de teste a cada teste, estudar a escalabilidade de protocolos em redes com grande número de nós, não sofrer influências de movimento e do ambiente na transmissão das ondas de rádio, etc.

Ao desenvolver simulações há duas propriedades principais que deve-se levar em consideração: a primeira é a precisão em relação às condições reais e os resultados que precisam ser produzidos e a segunda é o comportamento do ambiente de execução da simulação.

Os modelos existentes de simulação discretizada são codificados como programas orientados a eventos, onde os eventos são processados em sua ordem casual, atualizando o estado da simulação de acordo com o modelo utilizado e possivelmente escalonando mais eventos para simulação.

Devido a sua popularidade, os simuladores de eventos discretos são assunto de muitas pesquisas [11, 12, 13, 14] que focam na sua eficiência, design e forma de execução. Verifica-se também em [2] que dos tipos comumente utilizados, destacam-se os simuladores baseados em *kernels* (núcleo do sistema), *libraries* (bibliotecas) e *languages* (linguagens específicas para simulação), onde o autor apresenta suas semânticas de execução, regras sobre o estado da simulação, vantagens e desvantagens. Cada abordagem destas apresenta uma propriedade diferente e desejada, como:

- padronização – escrever simulações numa linguagem de programação convencional e popular, ao contrário de linguagens específicas para simulações;

- eficiência – otimizar o programa de simulação, seja estática ou dinamicamente;

- transparência – programas de simulação executados de forma eficiente e

correta, sem a inclusão de chamadas a bibliotecas;

- exatidão – computar resultado válidos e úteis, independentemente da forma como foram construídos;

Conforme [15], para simulação de redes móveis ad hoc há a necessidade dos seguintes componentes:

- Um software de simulação – podendo ser um programa desenvolvido pelo pesquisador ou utilizando os programas existentes, como: ns-2, OPNET Modeler, Omnet+ + etc.;

- Uma emulação em software do ambiente físico que engloba o modelo de propagação do sinal de rádio, área de simulação e modelo de mobilidade do nó;

- Uma implementação em software de todos os elementos de rede que estão abaixo da camada de rede como: emulação do rádio, a rede camada-2 (e.g. IEEE 802.11 WLAN), etc;

- Uma implementação dos protocolos e aplicações como os *Ad hoc On-Demand Distance Vector* (AODV) ou o *Dynamic Source Routing* (DSR) para roteamento, TCP / UDP e um gerador de tráfego;

- Um mecanismo para configurar parâmetros para todos os componentes de simulação acima citados e outro para a coleta de resultados, como por exemplo registrando estatísticas num arquivo de log ou base de dados;

Abaixo uma descrição sobre alguns dos simuladores de rede sem fio mais relevantes e quais parâmetros eles permitem simular.

2.4.1 NS-2 – Network Simulator, versão 2

O NS-2 foi desenvolvido na Universidade da Califórnia (Berkeley) e o grupo Monarch, da Carnegie Mellon University (CMU), foi quem o estendeu para redes móveis, suportando WLANs e MANETs. Ele tornou-se o padrão para simulação de redes, sendo um dos primeiros simuladores a apresentarem a aplicação de Simulação de Eventos Discretos. Existe uma infinidade de modelos, por exemplo, de protocolos e geradores de tráfego, disponíveis para esta plataforma. O NS-2 inclui um gerador de cenários, o qual

permite a geração automática de diferentes topologias de redes, padrões de tráfego e de falhas.

Sua principal limitação é sua escalabilidade em temos de memória e tempo de execução, o que é um problema para os pesquisadores e seus novos domínios de pesquisa no campo das redes de computadores, redes de sensores sem fio, redes *peer-to-peer* e arquiteturas em grade, todos utilizando um grande número de nós em seus experimentos. Para contrariar esta limitação, estudos estão sendo dirigidos para desenvolver melhorias como paralelização (PDNS [20]), onde as simulações do NS-2 podem ser distribuídas num *cluster* computacional.

Em relação a propagação do sinal, o NS-2 possui três modelos para predizer a potência do sinal recebido em cada pacote, onde na camada física (PHY) de cada nó há um limite de recepção, ou seja, quando um pacote é recebido, se a potência do sinal está abaixo do limite de recepção, então ele é excluído pela camada MAC.

As simulações para o NS-2 são escritas na linguagem de programação C++ para modelar o comportamento (implementa os protocolos e as primitivas de simulação) dos nós simulados e utilizam *scripts* na linguagem OTcl para controlar a simulação, especificar alguns aspectos como a topologia da rede e descrever as funcionalidades dos elementos de rede simulados. O OTcl também pode ser usado para implementar protocolos. Neste modelo de programação, o NS-2 se mostra como um simulador de protocolo de propósito geral, flexível, e extensível.

O seu sucessor, o NS-3 [17], é um novo simulador, não compatível com o NS-2, que proporciona aos pesquisadores o estudo de protocolos de internet e sistemas de larga escala num ambiente controlado e que promete suprir suas deficiências através de melhorias no software, especificamente no seu core, integração do software, modelos e componentes educacionais. Além disso, ele é um projeto *open source* organizado, desenvolvido e mantido em torno da comunidade de pesquisa, cujo público alvo são os pesquisadores e educadores na área de redes.

2.4.2 Glomosim - Global Mobile Information System Simulator

O GloMoSim [23] foi concebido como um ambiente de simulação escalável para redes sem fio, baseado em bibliotecas, desenvolvido no Laboratório de Computação Paralela da Universidade da Califórnia em Los Angeles (UCLA). Sua arquitetura segue a abordagem de construção de sistemas de rede baseados em camadas, similar ao modelo OSI, onde ele implementa um conjunto de protocolos de rede para redes sem fio. Ele é composto por uma coleção de módulos de biblioteca implementados em *PARallel Simulation Environment for Complex Systems* (PARSEC), uma linguagem baseada em C para descrição de simulações seqüenciais e paralelas, onde cada módulo simula um protocolo de comunicação sem fio da pilha de protocolos e novos módulos e protocolos podem ser desenvolvidos e adicionados à biblioteca. No modelo de programação do PARSEC, os protocolos são implementados em módulos monolíticos chamados *entities* (entidades), o qual permite somente a composição de protocolos em camadas.

2.4.3 JIST / SWANS – Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator

Conforme o seu guia do usuário, o JIST foi desenvolvido por Rimon Barr na Cornell University e vem introduzir uma nova abordagem para simuladores, funcionando como motor de simulação de eventos discretos, usado como ferramenta científica e baseado na linguagem de programação Java [9], onde a transparência, a eficiência e a padronização são a base da proposta.

Transparência implica que as simulações criadas pelo usuário serão automaticamente adaptadas à semântica de tempo de simulação do simulador; eficiência implica que o tempo de execução da simulação como um todo deva ser melhor do que os sistemas existentes; e seguir um padrão implica que as simulações possam ser escritas numa linguagem de programação convencional e padronizada já existente.

Ele foi desenvolvido com o objetivo de suprir algumas deficiências dos simuladores de eventos discretos existentes na época, no que tange à transparência,

eficiência e padronização. Este último como sendo o grande diferencial, pois o código a ser simulado sob o JIST não necessita ser escrito numa linguagem específica para simulações ou numa linguagem com um propósito específico.

Pelo contrário, ele introduz a semântica de tempo de execução para simulação de programas escritos na linguagem Java e roda sob uma máquina virtual (JVM – Java Virtual Machine) não modificada, convertendo-a num sistema escalonador de eventos, modificando a forma de condução das invocações dos métodos das entidades simuladas, de forma flexível , eficiente e escalável.

Ele combina o uso da semântica de simulação, encontrada em linguagens de simulação customizadas e bibliotecas de simulação, com as funcionalidades de uma linguagem moderna, robusta e eficiente. A chave disso tudo é poder executar eficientemente programas de simulações dentro do contexto de uma linguagem moderna e popular.

Tempo de Simulação (Simulation Time)

Em Java, a JVM funciona como um motor de execução baseado em pilha e com semânticas de execução bem definidas. Neste modelo, o andamento do tempo não depende do progresso das aplicações e o relógio do sistema avança indiferentemente da quantidade de instruções processadas.

No JIST o progresso do tempo depende diretamente do progresso das aplicações. O relógio das aplicações, que representam o tempo de simulação, não avançam para o próximo ponto discreto no tempo até que todos os processamentos para o corrente tempo de simulação seja completado. Neste modelo, porções de instruções individuais de *bytecode* da aplicação são processados sequencialmente, seguindo a semântica de controle de fluxo padrão da JVM e o tempo da aplicação continua inalterado. O código da aplicação pode avançar no tempo somente através da chamada do *sleep(n)* (da JistAPI), ou seja, qualquer instrução executa em tempo igual a zero, e só avança para o próximo tempo após o *sleep(n)* ser invocado, onde n identifica a quantidade de tempo a ser avançada.

Benefícios

Incluir a semântica de tempo de simulação dentro da linguagem Java traz consigo alguns benefícios como o reuso de código, o uso de bibliotecas padrão existentes, uso de compiladores existentes, os benefícios da *garbage collection*, a *type-safety* e a reflexão, que são características desejáveis, comprovadamente funcionais e já amadurecidas, sob o ponto de vista de desenvolvimento de aplicações. Assim, a curva de aprendizado necessária será menor, proporcionando o reuso de código na construção de simulações, além do fato de poder rodar dentro de uma máquina virtual padronizada que oferece um ambiente eficiente, bastante otimizado, portável e ao mesmo tempo confiável, facilitando a comunicação entre o *Kernel* de simulação e a simulação em si. Além disso, o fato de rodarem dentro de um mesmo processo reduz a necessidade de serializações e o *overhead* oriundo da mudança de contexto intrínseca das simulações.

Sob o ponto de vista de aplicação, observam-se benefícios da verificação automática de tipagem, da estrutura de um evento sem a necessidade de grandes estruturas de dados para isso e do fato de ter disponível informações como a localização do evento despachado e o estado da entidade de origem.

Sob o ponto de vista de linguagem, a Java proporciona benefícios como: o reuso da linguagem, um ambiente de execução estável através da JVM, um sistema de da *garbage collection* que contribui no gerenciamento da correta alocação de recursos (memória). Além disso, observa-se que a isolação do estado a nível de objeto entre as *Entities*, o gerenciamento de chamadas em tempo de simulação através do Kernel do JIST e o compartilhamento de objetos imutáveis, promovem a segurança.

Sob o ponto de vista de sistema, o fato do *kernel* do JIST e as aplicações rodarem sob o mesmo espaço de processo, a não necessidade de serializações e mudança de contexto, permitem otimizações. Em [2] cita-se que além de não requerer acesso aos fontes na reescrita do *bytecode*, este processo pode utilizar-se das implementações de protocolos e bibliotecas existentes.

Sob o ponto de vista de hardware, o fato do JIST ser puramente Java elimina a necessidade de hardware específico, mesmo rodando em COTS *clusters* [2] ou arquiteturas específicas.

Arquitetura

O JIST possui uma arquitetura composta de um compilador, um reescritor de *bytecode*, um *Kernel* de simulação e uma máquina virtual que é o local onde ocorre o processo de simulação, na Java Virtual Machine (JVM) [9].

Funcionamento e Execução

No modelo de objetos do JIST, qualquer objeto deve estar estar logicamente contido dentro de uma *Entity*, que na prática são objetos regulares da JVM, mas que servem para logicamente encapsular objetos da aplicação e demarcar os componentes da simulação independentemente.

Os programas de simulação que rodarão sob o JIST são escritos em Java e antes de serem executados são compilados para *bytecode* usando o compilador Java padrão. Após serem compiladas, estas classes (**.class*) são então modificadas por um reescritor em nível de *bytecode* para rodarem sob o *Kernel* de simulação e para suportarem a semântica de tempo de simulação, por ele executada. Neste passo, chamadas para métodos públicos são assincronamente dissociados e as chamadas a eles não podem retornar qualquer valor (*void*). Ainda, uma *Entity* representa uma interface (da linguagem) baseada em eventos que possui métodos externos (públicos) e a regra é que somente métodos públicos das *Entities* serão reescritos dentro dos eventos. Os métodos não públicos serão considerados métodos de objetos regulares Java e não métodos de *Entities*, e assim seguem a semântica de invocação normal do Java. Internamente, o JIST substitui todas as referências às *Entities* por objetos *proxy* chamados *Separators* para que as chamadas direcionadas a elas não sejam executadas diretamente.

Assim, as semânticas de tempo de simulação são introduzidas no momento da reescrita do *bytecode (classloader)* e suportadas pelo *Kernel* em tempo de execução.

Todas as funcionalidades do JIST são expostas ao desenvolvedor através da classe *jist.runtime.JistAPI*, que representa a interface da aplicação exposta pelo *Kernel* de simulação. Assim, para que a aplicação execute dentro do interpretador JIST, ou seja, para que o *Kernel* de simulação seja ativado, é necessário passar a classe principal, da aplicação a ser simulada, como parâmetro da classe *jist.runtime.Main*, por exemplo "*jist.runtime.Main HelloWorld*". Ao efetuar esta chamada inicial, um *classloader* é

automaticamente instalado dentro da JVM, o qual dinamicamente reescreve o *bytecode* da aplicação em questão, transformando-a numa *Entity* e não num objeto regular, pois ela passará a implementar a interface *JistAPI.Entity*.

Cada *Entity* possui seu próprio tempo de simulação, o qual determina em que momento as chamadas para outras *Entities* serão escalonadas no *kernel* do JIST.

Na prática, a classe *JistAPI.Entity* funcionará como um marcador do programa fonte, respeitando a sintaxe Java, preservando o processo de compilação e facilitando a execução em *type-safe*, ou seja, prevenindo o comportamento errôneo ou indesejável dos programas. Estas marcas servem para direcionar as transformações do código, que introduzem as semânticas de tempo de simulação dentro do *bytecode*.

Ao executarmos um método (ou aplicação) dentro do JIST as chamadas são transformadas em eventos de simulação, as quais são escalonadas pelo *Kernel* em tempo de simulação e não pela semântica de execução normal da JVM.

Durante a simulação, quando um método de uma *Entity* é chamado, a chamada retorna imediatamente, uma nova entrada é inserida na fila de escalonamento e executada depois, em seu tempo de simulação.

A figura (2) mostra um exemplo básico de uma *Entity* e descreve os seguintes passos:

1)Criação da *Entity hello* no método *main* da classe;

2)No início da execução o tempo de simulação de todas as *Entities* é ajustado para te=0;

3)Método *myEvent()* é chamado, aumenta em 1 (uma) unidade o tempo de simulação local te=1 e em seguida ele efetua uma chamada recursiva sob ele mesmo. Entretanto, este método público é transformado num evento, não havendo recursão;

4)A chamada retorna imediatamente, o método *println()* é chamado e o método *myEvent()* retorna;

5)Como nenhum outro evento foi escalonado para o te=0, o tempo de simulação avança para te=1 e a chamada pendente para o método *myEvent()* é processada, iniciando-se o processo novamente e continuando o *loop* infinitamente.

hello.java

```
1  import jist.runtime.JistAPI;
2
3  class hello implements JistAPI.Entity
4  {
5    public static void main(String[] args)
6    {
7      System.out.println("simulation start");
8      hello h = new hello();
9      h.myEvent();
10   }
11
12   public void myEvent()
13   {
14     JistAPI.sleep(1);
15     myEvent();
16     System.out.println("hello world, t="
17       +JistAPI.getTime());
18   }
19 }
```

Figura 2 – Uma simples *Entity* no JIST [JIST, 2004]

Abaixo a tabela (1) apresenta a saída da execução do código acima. Na coluna da esquerda o sistema utiliza a funcionalidade de “tempo de simulação” e na coluna da direita não.

simulation start	simulation start
hello world, t=1	hello world, t=0
hello world, t=2	hello world, t=0
hello world, t=3	hello world, t=0
hello world, t=4	hello world, t=0
hello world, t=5	hello world, t=0
hello world, t=6	hello world, t=0
hello world, t=7	hello world, t=0
hello world, t=8	hello world, t=0
hello world, t=9	hello world, t=0
hello world, t=10	hello world, t=0
hello world, t=11	hello world, t=0
hello world, t=12	hello world, t=0
hello world, t=13	hello world, t=0

Tabela 1: Tempo de simulação e Tempo de execução

Em cima desta base, os autores criaram o SWANS [1], o qual roda sob o JIST e disponibiliza os mecanismos necessários para simular redes móveis *ad hoc* (MANET). Em sua arquitetura, os nós (*nodes*) são compostos de muitas entidades (*entities*) que estão acopladas, formando os diferentes elementos (camadas) da pilha do protocolo que representa a rede.

Estes elementos implementam diferentes tipos de aplicações, como: protocolos de rede, roteamento, e acesso ao meio; modelos de transmissão, recepção e ruído; modelos de propagação e enfraquecimento do sinal; e modelos de mobilidade dos nós.

Ele possui a vantagem de poder executar aplicações de rede desenvolvidas em Java sobre a rede simulada, como: servidores *web,* aplicações *peer-to-peer* e protocolos *multicast* em nível de aplicação. Durante o funcionamento destas aplicações, elas operam em tempo de simulação dentro do mesmo espaço de processo do JIST, não enviando simplesmente os pacotes dos seus processos para o simulador.

No SWANS, a entidade que proporciona a mobilidade dos nós e a propagação do sinal, para os outros nós dentro do alcance de transmissão sem fio, chama-se *Field*, a qual é comum para todos os nós da rede. Os pacotes transmitidos atravessam as entidades da pilha de protocolos sem custo (virtual) algum, como simples referências, e a duplicação destes pacotes só feita se necessário. Nesta arquitetura, a entidade Radio de cada nó é a última da pilha de protocolos antes da entidade *Field* , que é comum para todos os nós.

Em relação a propagação do sinal, o SWANS implementa na entidade *Field* uma estrutura que agiliza a busca e localização dos nós, chamada *Hierarchical Binning*, favorecendo também na velocidade das simulações. Quando um nó envia um pacote para o *Field*, este pacote deve ser enviado a todos os nós alcançados por este pacote após considerar *fading, gain e pathloss*. Assim, apenas uma porção do conjunto de nós estará dentro do alcance de recepção, os demais serão afetados por interferência acima do limite, mas isso dependerá da área do *Field* e da potência de transmissão. A tabela (2) a seguir, apresenta os simuladores mais conhecidos e os parâmetros por eles simulados:

	NS-2	JIST-SWANS	Glomosim
Roteamento	DSDV, DSR, AODV e TORA	DSR, AODV, ZRP MANET	IP com AODV, Bellman-Ford, DSR, Fisheye, LAR scheme 1, ODMRP, WRP
Mac	IEEE 802.11, 802.16, 802.15.3, 802.15.4. ou TDMA	802.11b e protocolo “Dumb”	CSMA, IEEE 802.11 e MACA
Transporte	TCP e UDP	TCP e UDP	TCP e UDP
Propagação do Sinal	Free Space, two ray e shadowing	Free Space, Two Ray	Free Space, Two Ray
Mobilidade	Random Waypoint, Random Walk, Random Direction	Teleport, Random Walk, Random Waypoint	Random Waypoint, Random Drunken, Trace Based
Modelo de rádio	Comparação de 2 sinais	Additive interference	Noise Accumulating
Modelos de Recepção de Pacotes	SNRT based	Additive Model e Independent Interference Model	Delimitado pelo SNR, baseado em BER com modulação BPSK/QPSK
Aplicação	Telnet e FTP	AppHeartbeat e AppJava	CBR, FTP, HTTP and Telnet
Modelo de Fading	Rayleigh, Ricean	Rayleigh, Ricean	Rayleigh, Ricean

Tabela 2: Simuladores e parâmetros

2.5 Trabalhos Relacionados

De acordo com o levantamento bibliográfico realizado, abaixo cito o único trabalho encontrado que está diretamente ligado a simulação de redes cognitivas. Em sua maioria, os simuladores aqui citados simulam redes sem fio tradicionais com sua respectiva pilha de protocolos, não servindo como um todo para as simulações aqui necessárias.

Abaixo apresentam-se as características do simulador de redes cognitivas

Cognitive Radio Cognitive Network Simulator [3], que foi o único a propor idéias similares às aqui propostas, ou seja, simulação de redes cognitivas, mas bem mais desenvolvido e maduro. Entretanto, devido às particularidades estruturais propostas no trabalho [24], também não pôde ser utilizado como base.

Entretanto, serviu como base de conhecimento, principalmente no tocante à propagação do sinal e na composição do ambiente físico virtual onde os rádios serão dispostos. Assim, viu-se a necessidade de desenvolver um ambiente específico para a simulação.

2.5.1 Cognitive Radio Cognitive Network Simulator

O CRCN foi desenvolvido por Jing Zhong e Jialiang Li na Michigan Tech University (EUA) e seu desenvolvimento foi motivado pela não existência de simuladores que fossem adequados pelas demandas de simulação de um rádio cognitivo, pois os simuladores atuais (NS2, Glomosim, Jist-Swans, Opnet, Qualnet, etc...) foram desenvolvidos para redes sem fios comuns e os pesquisadores não conseguem implementar facilmente os seus algoritmos de rádio cognitivo sobre estes simuladores.

O CRCN é uma extensão do NS2 para a simulação de redes de rádios cognitivos, que suporta avaliação de desempenho para a alocação dinâmica no espectro proposta, que possui algoritmos de controle de potência e protocolos de rede para a adaptação de rádios cognitivos, incluindo os protocolos para a camada MAC e roteamento, e que utiliza o NS2 para gerar tráfego realístico e padrões de topologia.

Em sua arquitetura, os nós compartilham uma camada física (PHY) (multi-radio e multi-canal) que customiza os parâmetros do espectro como potência de transmissão, propagação do sinal, etc. Além disso, como se vê na figura (x) abaixo, esta arquitetura permite o repasse de parâmetros entre as diferentes camadas, onde os desenvolvedores podem substituir elas por suas próprias implementações, desde que sigam os requisitos para desenvolvimento de protocolos do NS-2.

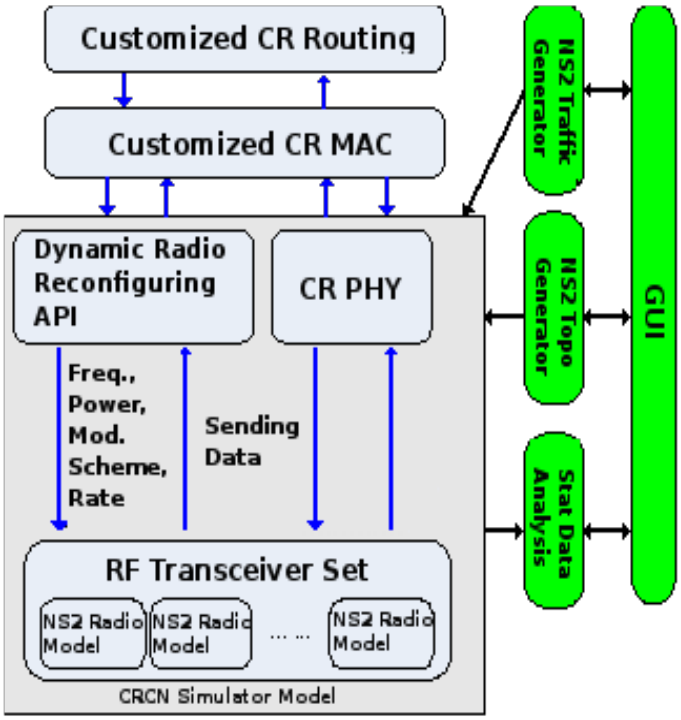

Figura 3: Arquitetura do simulador CRCN [Ijialian, 2009]

Nas suas funcionalidades relacionadas a camada PHY, é definida a reconfiguração dos parâmetros do espectro e dos parâmetros do rádio, a informação de interferência e um modelo físico baseado na taxa de sinal-ruído (SNR) e na taxa de sinal interferência mais ruído (SINR).

Verifica-se também que para proporcionar os algoritmos das camadas MAC e PHY, é necessário que cada rádio possua uma estrutura multi-canal, pois a arquitetura propicia a definição “um rádio x vários canais” e “vários rádios x vários canais”, todos visíveis pelas camadas MAC e PHY, conforme figura (4-5) abaixo.

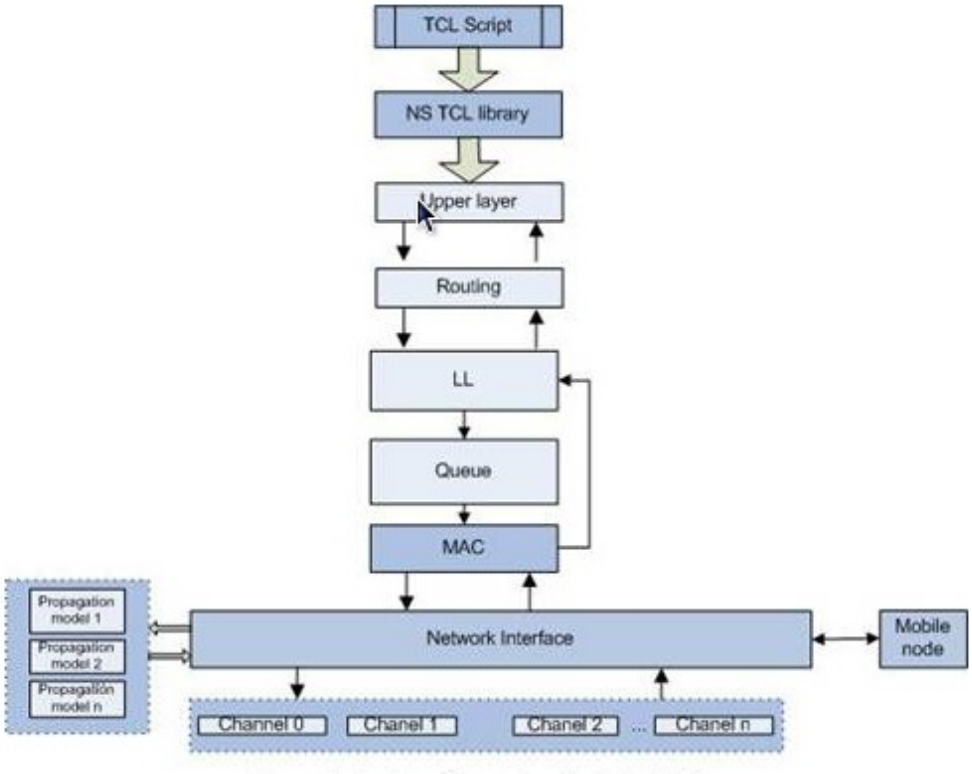

Figura 4: Um rádio x vários canais [ljialian, 2009]

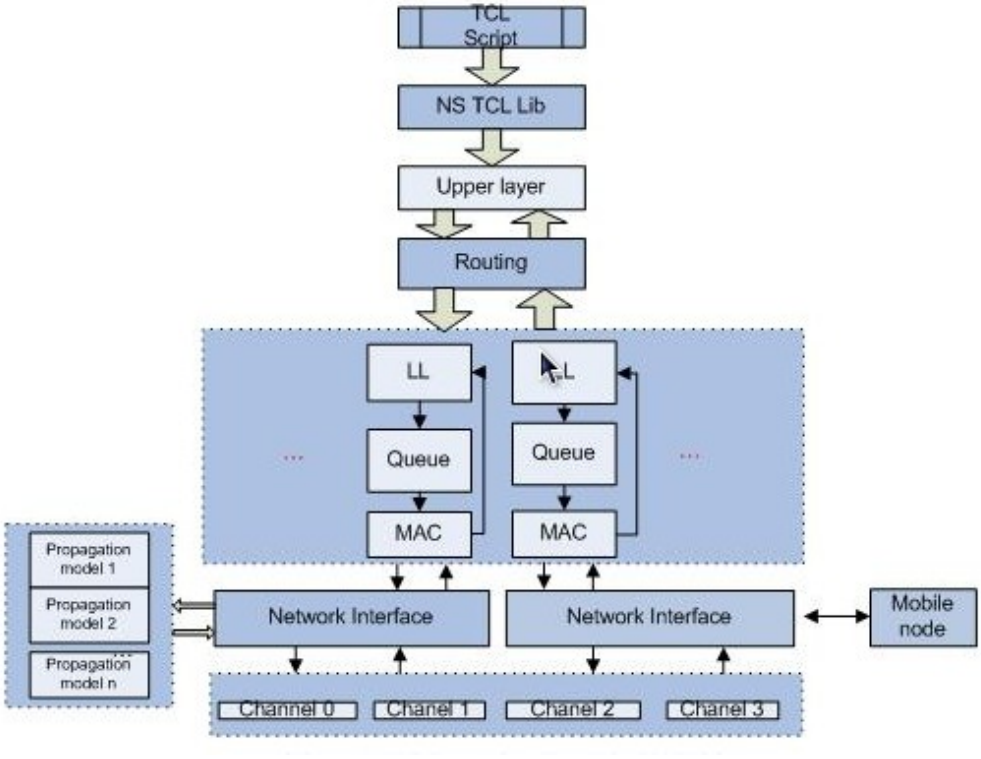

Figura 5: Vários rádios x vários canais [ljialian, 2009]

Independente da quantidade de rádios e canais, o pacote chega num determinado canal e entra na camada PHY, onde é verificada a questão de mobilidade e aplicado o modelo de propagação desejado. A camada física é responsável por providenciar informações sobre o tráfego, sobre interferência e comunicação para as camadas superiores através do bloco de informações.

A nível de camada PHY, para uma simulação, basicamente são configurados o modelo de propagação – *TwoRayGround ou FreeSpace* – e o tipo de antena – *OmniAntenna*.

3 Desenvolvimento do Projeto

3.1 Visão Geral

Nesta seção, apresenta-se, em termos gerais, a proposta do projeto. O modelo de camada física implementado, representado pela entidade *CogPhyField*, prevê a entrega de cada sinal de rádio, transmitido por um rádio cognitivo, para os outros nós com os devidos atrasos de propagação, de modo a garantir a fidelidade dos resultados da simulação. Os componentes deste *framework* estão organizados na forma de classes independentes de software que, operando em conjunto, são responsáveis pela modelagem da propagação do sinal entre os rádios cognitivos e pela entrega dos sinais.

Neste processo de transmissão, espera-se que os rádios cognitivos efetuem transmissões e que estes sinais cheguem à camada física que é comum a todos eles. Portanto, independentemente se o rádio destino encontra-se dentro ou fora do alcance de transmissão de um rádio transmissor, *é a CogPhyField* que verificará se ele está apto a receber o sinal enviado, baseada no nível de sensibilidade do rádio destino e no limiar de intensidade do sinal enviado. Por conseguinte, os demais rádios cognitivos a ela conectados farão a recepção do sinal enviado, desde que estejam dentro deste alcance de recepção e possuam um limiar de sensibilidade compatível com a potência do sinal enviado após sofrer as devidas perdas comuns neste processo. Assim, na recepção dos sinais, antes da *CogPhyField* efetuar a entrega do sinal, ela verificará se o sinal transmitido está dentro do limiar de intensidade, a localização e distância dos rádios que estão dentro da topologia física fictícia, aplicará ao sinal os fatores de perda que influenciam nesta transmissão, os quais afetam profundamente a operação de uma rede e no desempenho do sistema, e analisará caso a caso qual rádio receberá ou não este sinal. Considera-se também que os rádios sejam transceptores que não utilizam técnicas de espalhamento espectral. Espalhamento espectral é uma técnica de modulação em que a largura de banda usada para transmissão é muito maior que a banda mínima necessária para transmitir a informação e desta forma, a energia do sinal transmitido passa a ocupar uma banda muito maior do que a da informação. Assim os rádios podem trocar dados somente na mesma banda de frequência (banda de interesse ou canal).

Para efeito de cálculos, considerar-se-ão as informações dos rádios de origem

e destino, as informações de transmissão enviadas pelo rádio de origem e a distância entre o rádio transmissor e cada rádio receptor que estiver registrado na *CogPhyField.* Além disso, considerou-se que o meio de transmissão é a atmosfera (ao nível do mar) e que a velocidade de propagação das ondas de RF nesse meio é igual a velocidade de propagação da luz no vácuo.

A CogPhyField possui estruturas de dados que gerenciam os rádios e os sinais que estão presentes num dado momento, proporcionando informações do tipo: em qual canal um rádio está operando, quais sinais devem ser transmitidos no tempo X e no tempo Y, assim como quais os rádios estão operando num dado canal.

Em relação à sincronização de entrega de sinais, para que um sinal enviado no tempo “t”, com duração “z” chegue ao seu destino antes de um outro sinal enviado no mesmo tempo “t” mas com duração “z+1”, valeu-se da característica do simulador de eventos discretos JIST chamada *simulation time*, ou tempo de simulação. Desta forma, é possível que mais de um sinal seja enviado num mesmo tempo de simulação, mas deve-se garantir que eles cheguem ao seu destino também no mesmo tempo, desde que tenham a mesma duração.

3.2 Arquitetura

Conforme o *framework* proposto no trabalho [24], a figura (6) abaixo apresenta a sua arquitetura em bloco parcial a ser suportada. Verifica-se a existência de uma camada de controle que é a responsável pela reconfiguração das camadas RADIO, MAC e NET, quando necessário.

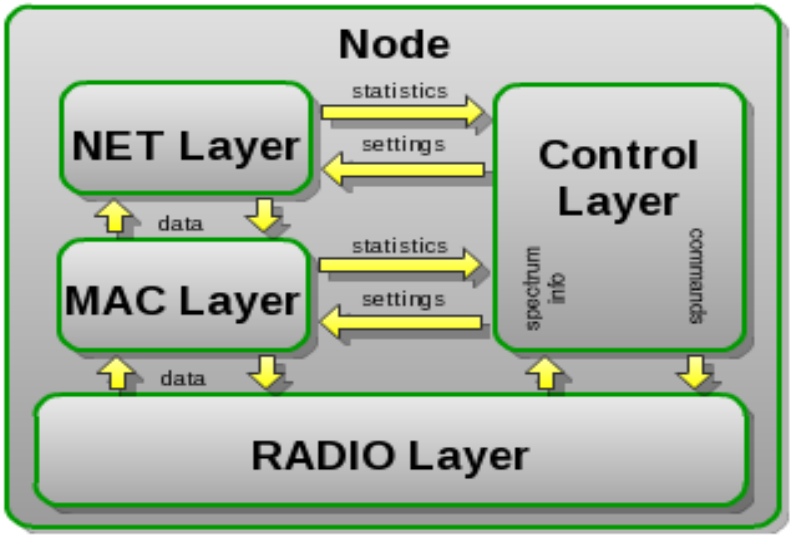

Figura 6 - Arquitetura parcial [Mendes, 2009]

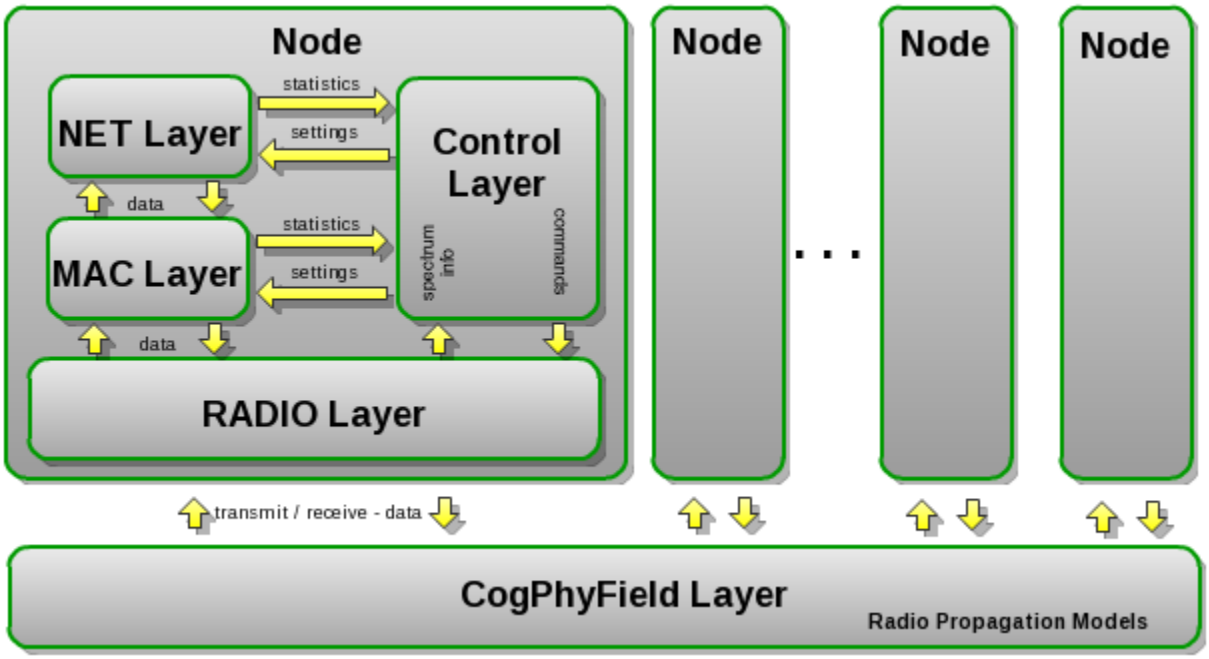

Figura 7 – Arquitetura com a camada CogPhyField

Já na figura (7), apresenta-se a inclusão da camada PHY aqui proposta ao modelo proposto em [24], onde pode ser visto que a comunicação entre os rádios ocorre através da camada CogPhyField. É nesta camada que serão implementadas as funcionalidades para a solução do problema de entrada/saída de sinais/dados da camada de RÁDIO para a camada PHY e sua propagação para os demais rádios.

É sob esta arquitetura que são implementadas as novas características que representam o diferencial do trabalho [24] em relação às novas informações que vão compor a estrutura do sinal, que são:

1) Um rádio cognitivo pode operar em frequências licenciadas de duas formas: através do controle de potência para assim coexistir com os usuários primários, ou oportunisticamente, onde o rádio abandonará a frequência que opera ao identificar a transmissão de usuários primários. Assim sendo, a potência do sinal, que é considerada na estrutura do sinal do modelo aqui proposto, é a soma das potências dos pulsos (Pulse) que o compõe e torna-se uma informação importante para a fase de decisão das heurísticas dos rádios cognitivos, o que não é considerado no modelo de pacotes/mensagens dos simuladores de redes sem fio existentes.

2) Outra informação importante a ser considerada é a modulação. Por este motivo ela também foi considerada neste *framework*, como informação constante da estrutura do sinal, diferentemente dos modelos existentes nos simuladores de redes sem fio convencionais que não a consideram em seu modelo de pacotes/mensagens, mas que serve como subsídio adicional para a fase de decisão das heurísticas dos rádios cognitivos;

3) Para atender a necessidade de um rádio cognitivo, há necessidade de que haja mais informação nas estruturas de dados, além daquelas já existentes para os rádios convencionais. O modelo de dados que é transmitido numa comunicação de rádio frequência não deve ser mais simplesmente o *bit-byte* (01010101), mas sim algo que possa ser trabalhado como fonte de dados articulados pelas heurísticas de um rádio cognitivo, de forma a trazer algum significado e contribuir no processo de aprendizagem e de aquisição de conhecimento. Assim, o modelo tradicional foi estendido pelo trabalho [24], a partir do momento que acrescentou-se mais informação ao sinal, e foi implementado no modelo aqui proposto, a saber:

-> NOSIGNAL: não há sinal presente na banda do espectro selecionada;

-> HIGH: um bit alto foi sensoreado;

-> LOW: um bit baixo foi sensoreado;

-> PU: o mecanismo de detecção detectou a atividade de um PU na porção do espectro sensoreado;

-> UNKNOWN: há algum sinal presente na banda do espectro selecionada mas ele não pode ser interpretado como um HIGH, LOW ou PU.

3.2.1 Componentes

Esta seção enumera estes componentes básicos e fundamentais da implementação da simulação e que estão disponíveis no código fonte do programa na forma de classes. A importância da existência deles no trabalho como um todo é principalmente para dar suporte às camadas superiores e permitir que os rádios cognitivos os utilizem como dados úteis no processo de aprendizado. Conforme o trabalho [24], são dados úteis: a informação da classificação do canal, a frequência/largura de banda sensoreado, o estado do *Signal* (via *Bit*), o intervalo de tempo do *Signal* sensoreado, a potência do *Signal* e a métrica de acurácia associada à classificação.

3.2.1.1 Bit

É a menor unidade de informação e também faz parte da composição do *Signal*. Ele representa um possível estado, está contido dentro da estrutura do Pulse. A possibilidade de classificar o Signal através da informação contida no Bit proporciona, através de fragmentos de *frame,* às camadas superiores uma forma de organizar os usuários da banda selecionada. O Bit pode apresentar as seguintes configurações:

HIGH – *Bit* com sinalização Alta;

LOW– *Bit* com sinalização Baixa;

UNKNOW– Bit com sinalização desconhecida;

PU – *Bit* com sinalização de usuário primário;

NOSIGNAL – *Bit* sem sinalização.

3.2.1.2 Pulse

Um *Signal* poderá ter um ou mais *Pulses,* os quais se apresentam em ordem de tempo baseados em seus atributos *startMoment* e *stopMoment*, ou seja, um *Pulse* com *stopMoment* igual a 0.005 virá antes do *Pulse* com *startMoment* 0.006, sem que haja nenhum outro no intervalo entre os dois. Em sua composição identificam-se quatro atributos, que são:

- startMoment: identifica o tempo inicial deste Pulse;

- stopMoment: identifica o tempo final deste Pulse (a soma dos tempos dos pulses identificam o tempo de duração do *Signal*);

- *Bit*: identifica um *Bit*, o qual representa um estado específico (no intervalo de tempo entre o *startMoment* e o *stopMoment*);

- *power*: identifica a potência relacionada a este Pulse (a soma da(s) potência(s) do(s) Pulse(s) identifica(m) a potência total do *Signal*).

Como o JIST trabalha com unidade de tempo igual a 1 “time”, definiu-se que o intervalo de 1 Pulse poderá ter, por exemplo para o time igual a 5, o startMoment igual a 5.0000 e o stopMoment igual a 5.9999.

3.2.1.3 Signal

É a representação do sinal de rádio frequência propriamente dita e que contém dados. O Signal é composto de um ou mais Pulses, a quantidade destes Pulses e a informação da modulação em que opera (também é útil para o rádio cognitivo). A classificação do *Signal* se dá na forma dos estados dos *Bits* contidos no conjunto de Pulses. Uma forma de manter a organização dos rádios da banda selecionada se dá, por exemplo, quando os rádios estão operando normalmente e ao identificar a existência de um *Signal*, cujo Pulse, no instante i até o f, possui um *Bit* com estado PU, eles têm de efetuar uma tomada de decisão, pois conforme as premissas de uma rede cognitiva e

funcionamento de um rádio cognitivo, quando a operação concomitante com a de um PU não é desejada, tem-se que efetuar a troca do canal, ou quando é permitida, deve-se fazer o controle de potência para não causar interferências à transmissão licenciada. Esta identificação do *Signal* pode ser efetuada através de reconhecimento de padrões, onde o *frame* deve ser considerado válido ou não, e também caracterizando a atividade de um PU na banda do espectro selecionada.

3.2.1.4 Channel

O canal representa a frequência em que o rádio está operando. Para fins práticos esta informação poderia ser abstraída, pois a frequência já é parte constante da entidade *Radio* a partir do momento em que ele inicia sua operação numa dada frequência*,* entretanto, o Channel aqui proposto possui ainda, além do seu número que vai de 1 a 14, as seguintes propriedades e características que estão baseadas na oportunidade que o canal disponibiliza. Estas informações são realmente úteis, a nível de camada rádio e superiores, como informação que será utilizada no modelo de aprendizado do rádio cognitivo, principalmente na fase de classificação do canal. Nesta fase, o foco está em classificar a sua atividade e tornar esta informação disponível às camadas superiores, para que seja possível decidir rapidamente sobre a próxima ação a ser tomada, de acordo com os princípios do gerenciamento do espectro.

Possui um estado, que pode ser:

- EMPTY;
- OCCUPABLE;
- UNOCCUPABLE;
- NONANALYZED.

Ele possui um tipo (ou classificação), que é inferida através da combinação e análise dos dados até aqui apresentados, permitindo ao rádio a identificação da existência de um fluxo de dados compatível ou incompatível com o seu padrão de transmissão, fluxo de dados de um PU, o seu próprio fluxo, *timeout* e fluxo indefinido caso não se encaixe

nos anteriores.

- PRIMARY_USER
- INCOMPATIBLE
- COMPATIBLE
- SAME_ID
- TIMEOUT
- UNDEF

Por exemplo, dado um fluxo de um *Signal*, num intervalo de tempo, cujos *Bits* dos Pulses sejam UNKNOWN ou HIGH e LOW e cujos *frames* sejam classificados como INCOMPATIBLE, pode caracterizar a presença de um usuário primário e resultar numa classificação OCCUPABLE do canal.

Por fim, a figura (8) apresenta a arquitetura sob uma outra perspectiva, incluindo a interação entre os rádios, a visão da camada CogPhyField, a composição dos *Signals* e seu gerador, assim como a entidade (*Logging*) responsável pelo registro da comunicação entre os rádios, que ocorrerá através da execução da simulação.

Observa-se também que as questões relacionadas à existência de um ou mais canais, que rádios operam em que canal, quando aplicar um modelo de propagação e a localização de cada rádio, ficam restritas à camada CogPhyField.

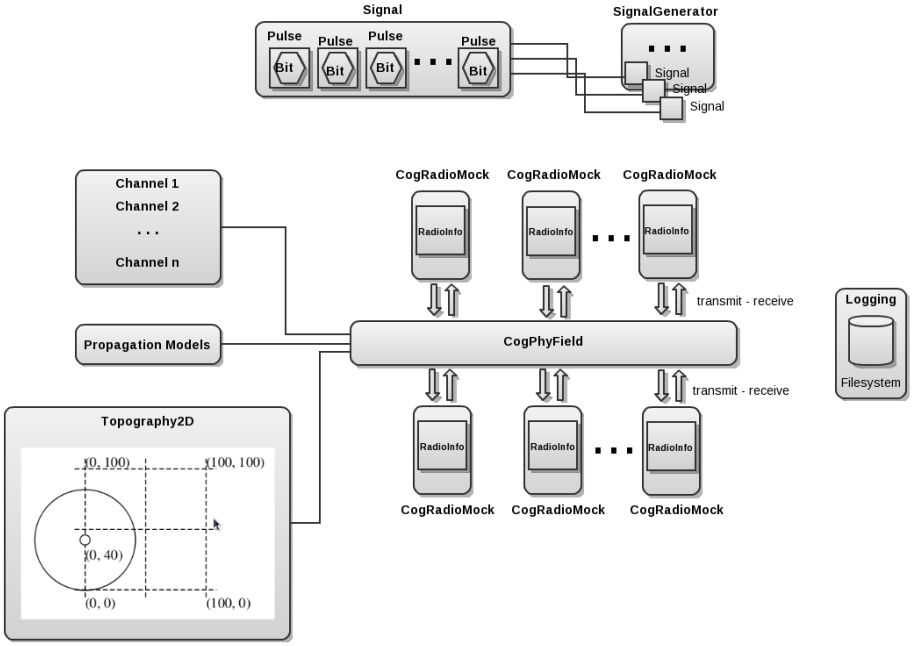

Figura 8: Visão Geral do sistema a ser simulado

3.3 Requisitos

Nesta etapa definiram-se os requisitos principais deste trabalho, os quais serão inseridos no contexto do comportamento do software para que seu objetivo maior fosse possível. Os requisitos da camada física aqui proposta basearam-se principalmente em atender as necessidades de um rádio cognitivo, conforme especificado no trabalho [24], além de basear-se e tentar operar da forma mais próxima possível de um simulador de redes sem fio existente, no caso o SWANS.

Escolheu-se o SWANS porque conforme explana a literatura, ele apresenta todas as características de um simulador de redes sem fio moderno, eficiente, que roda sob o JIST e que foi desenvolvido utilizando a linguagem de programação Java.

A partir da definição dos componentes básicos da composição de um sinal, dos fundamentos e princípios da propagação do sinal, assim como o funcionamento dos simuladores de redes sem fio, pode-se definir em termos gerais os requisitos do modelo a

ser desenvolvido para fins de simulação.

Como este trabalho visa implementar um modelo de camada física com suporte a características cognitivas, definiu-se que esta deva possuir as características de arquitetura e dos componentes acima descritos, além de possuir requisitos básicos de operação, cujo resultado final é a possibilidade de simular este modelo sob o simulador de eventos discretos JIST.

Requisitos básicos de operação:

- Permitir a transmissão simultânea de um ou mais sinais de um rádio a todos os outros do sistema;

- Efetuar o sincronismo no envio de sinais;

- Aplicar pelo menos um modelo de propagação do sinal, considerando perda de caminho e atenuação;

- Permitir a transmissão de sinal de um usuário primário (PU).

Além das funcionalidades básicas de operação, o fato de ter de efetuar uma simulação onde um ou mais rádios cognitivos efetuarão a troca de sinais, sentiu-se a necessidade de agregar requisitos para simulação do modelo e que serão descritos mais adiante:

- Gerar sinais aleatoriamente com configuração distintas que comporte os seguintes tipos de estados já definidos - *Bit HIGH, Bit LOW, Bit NOSIGNAL e Bit PU* - e que estejam acessíveis para que os rádios possam enviá-los aos demais, se comportando como se eles (os sinais) estivessem vindo das camadas superiores para a camada de rádio e assim para a camada física.

- Disponibilização de uma perspectiva de visualização do tráfego de sinais entre os rádios na forma de *log*, efetuando o registro das operações de transmissão e recepção, possibilitando o rastreamento dos eventos e depuração;

- Elaborar um modelo de topologia física cujo objetivo é a simulação da disponibilização dos rádio um ambiente terreno onde possa-se inferir a distância entre eles.

3.4 Funcionalidades

A partir da visão geral do sistema, dos requisitos e da arquitetura proposta, definiu-se funcionalidades de alto nível para o modelo de camada física, cujo objetivo é servir como um guia e manutenção do foco no desenvolvimento.

Abaixo cita-se as funcionalidades gerais consideradas, as quais apresentam-se separadas pelo nome da classe onde estão implementadas.

3.4.1 Funcionalidades - Classe CogPhyField

3.4.1.1 Registrar a entrada de um rádio

Neste modelo de camada física, pressupõe-se que uma de suas premissas seja ter a consciência dos rádios que nela operam. Desta forma, criaram-se estruturas de dados que possibilitassem o controle rápido e eficiente de quem entra e de quem sai, baseadas em tabelas *hash* e que permitem associações do tipo:

- um canal *Vs* um vetor dos sinais que trafegam neste canal, num dado intervalo de tempo;

- um canal *Vs* um vetor dos rádios que estão operando neste canal;

- um identificador do rádio *Vs as* informações deste rádio.

O processo de registrar um rádio é executado pelo método de assinatura *“public void enter(CogRadioInterface entity, RadioInfo info)”* e consiste de associar o rádio nas estruturas de dados citadas, assim como registrá-lo na grade aqui chamada de *Topology2D.*

3.4.1.2 Registrar a saída de um rádio

Opostamente ao método de entrada, o processo de saída, executado pelo método de assinatura *“public void exit(Integer iD)”,* faz a desassociação deste rádio das estruturas de dados citadas, assim como retira-o da grade.

3.4.1.3 Transmitir um sinal para o meio

Basicamente, o processo de transmissão de um sinal para todos os rádios que estiverem operando num dado canal ocorre executando-se o método de assinatura *“public void transmit(RadioInfo srcInfo, Signal signal, long duration, int channel)”*. Primeiramente é verificado se a potência de transmissão do rádio transmissor é maior ou gual ao limite de propagação estabelecido. Este limite, medido em dBm, baseou-se no padrão pré-estabelecido no SWANS e representa o limiar de intensidade do sinal para a camada física, ou padrão mínimo para o sinal a ser propagado.

Em seguida, este sinal é inserido num *buffer* de sinais e transmitido, com sua devida duração, para todos os rádios que estiverem no canal especificado. Entretanto, antes da transmissão propriamente dita ocorrer, serão aplicadas as perdas do meio (*fading e pathloss*) e só então o sinal será entregue aos outros rádios, desde que estejam dentro do alcance de transmissão. Este alcance baseia-se na potência do sinal após sofrer as perdas e na sensibilidade do rádio destino, ou seja, o rádio destino somente efetuará uma recepção se a potência do sinal recebido for maior que a potência de sensibilidade do rádio.

3.4.1.4 Sincronizar entrega de sinais

A sincronização da entrega de sinais é o procedimento que deve garantir que os sinais que forem transmitidos no instante “*t*” cheguem fielmente primeiro que aqueles enviados no instante “*t+1”.* Felizmente esta funcionalidade já é garantida pelo próprio simulador de eventos discretos *JIST,* conforme explanado no item 2.4.3

3.4.2 Funcionalidades – Classe Topography2D

Para simular o ambiente físico em que os rádios serão dispostos utilizou-se a técnica em grade baseada num plano cartesiano que é o modelo mais comumente utilizado nos simuladores de redes sem fio existentes. Sua função limita-se a criar uma grade quadrada (comprimento X comprimento), alocar um ponto (X,Y) aleatório e distinto para cada rádio e, quando for solicitada, calcular a distância entre dois pontos dentro desta grade. Para fins de cálculos, definiu-se um valor padrão para representar a distância em metros de um ponto a outro (item 3.4.2.3). Apenas a entidade *CogPhyField*, que é onde simula-se a propagação do sinal, é que tem consciência de rede em grade. A grade, cujo tamanho possui forma quadrada, deve ser passada como parâmetro na instanciação da *CogPhyField .*

3.4.2.1 Inserir um rádio na grade

Esta funcionalidade é implementada através do método de assinatura "*public void addRadio(Integer radioId)*", o qual atribui randomicamente um ponto dentro da grade para um rádio assim que ele se registra na *CogPhyField,* conforme visto no item 3.4.1.1 . Cada rádio possui um único ponto distinto.

3.4.2.2 Retirar um rádio da grade

É a funcionalidade inversa da anterior, implementada através do método de assinatura *"public void removeRadio(Integer radioId)",* e que deixa o ponto da grade, onde encontrava-se este rádio, vago para outro rádio qualquer.

3.4.2.3 Calcular a distância entre 2 rádios

A classe *Topography2D* possui uma subclasse chamada *Ponto* cuja função é definir um ponto do plano cartesiano. Ela possui um método de assinatura *"public double distancia(Ponto p)"* que calcula a distância entre dois pontos. Considerando que a distância mínima entre dois pontos é 1.4142, definiu-se que os rádios teriam uma distância mínima de 10, então para que a distância de um ponto a outro apresentasse no mínimo o valor 10 (metros), multiplicou-se o valor da distância mínima por 7.0711 .

3.4.3 Funcionalidades – Classe SignalGenerator

Devido a necessidade de realizar uma simulação e não ter uma pilha de protocolos implementada para que, no seu devido momento, um sinal fosse gerado e enviado pilha abaixo até chegar na camada física, implementou-se um gerador de sinais para ser utilizado pelo rádio fictício no processo de transmissão.

3.4.3.1 Gerar e disponibilizar um vetor de sinais (Signal)

Esta classe implementa em sua estrutura os mecanismos necessários para que sejam gerados sinais (Signal) e cujo objetivo principal é disponibilizar um Signal para que o rádio pudesse transmití-lo e os demais rádios recebê-lo.

Os sinais gerados aleatoriamente possuem configurações distintas que comportem os seguintes tipos de estados de Bit já definidos - *Bit HIGH, Bit LOW, Bit NOSIGNAL e Bit PU* -, que tivesse uma quantidade de Pulse aleatória, com startMoment e stopMoment aleatórios e cuja potência representasse a soma de todas as potências dos Pulse. Além disso os sinais gerados deveriam estar acessíveis para que os rádios pudessem enviá-los aos demais, se comportando como se eles (os sinais) estivessem vindo das camadas superiores para a camada de rádio e assim para a camada física.

Para posterior análise, foi implementada a funcionalidade de persistência deste vetor de sinais em um arquivo, no sistema de arquivos.

3.4.3.2 Gerar Signal de PU

Considerou-se também a geração de sinais cujo estado do *Bit* representasse o PU. Desta forma, no intuito de simular a presença de transmissão de um PU pode-se utilizar este conjunto de sinais. Entretanto, as transmissões de um PU normalmente não serão originadas de um rádio, mas sim, por exemplo, de uma emissora de TV que não estará registrada na *CogPhyField*. Na prática os rádios não têm conhecimento da existência de uma emissora de TV e só o terão quando ela efetuar uma transmissão. Da mesma forma, a camada física aqui simulada não tem como saber da existência de um PU, não podendo transmitir sinais oriundos de entidades que não estejam registradas em sua estrutura. Então, para fins de simulação, pode-se utilizar uma entidade rádio qualquer para fazer este papel de PU, desde que este rádio envie somente sinais com estado do *Bit* igual a PU.

3.4.4 Funcionalidades – Classe PathLoss

O cálculo do *PathLoss*, ou perda no caminho, neste modelo, foi adaptado do já implementado no simulador de redes sem fio *Ad hoc, SWANS,* considerando assim que os cálculos possuam acurácia.

3.4.4.1 Calcular FreeSpace / TwoRay

Estes modelos representam o comportamento e cálculo dos dois modelos de PathLoss já explanados. Ambos possuem um único método que retorna um valor *double* que é a potência de perda.

3.4.5 Funcionalidades – Classe Fading

O cálculo do *Fading*, ou enfraquecimento, neste modelo, foi adaptado do já implementado no simulador de redes sem fio *Ad hoc, SWANS,* considerando assim que os cálculos possuam acurácia.

3.4.5.1 Calcular None / Rician

Estes modelos representam o comportamento e cálculo dos dois modelos de Fading já explanados. Ambos possuem um único método que retorna um valor *double* que é a potência de perda. O modelo None identifica que não há perda por Fading e retorna “0.0”. Para ser instanciado, o modelo Rician deve receber um valor para o atributo “fator K”.

3.4.6 Funcionalidades – Classe CogRadioMock

Esta classe implementa os métodos necessários para o rádio efetuar a transmissão e recepção de um sinal. Obviamente o funcionamento de um rádio cognitivo possui diversas funcionalidades e protocolos de funcionamento que não estão sendo consideradas neste modelo por questão de escopo.

3.4.6.1 Transmitir um sinal

Esta funcionalidade, representada pelo método *“public void transmit(Signal signal, long duration, int channel)”*, envia para a camada física o sinal, a duração deste sinal, o canal em que o sinal deve ser transmitido e as informações do próprio rádio. Para isso, o rádio deve ter a referência da entidade que representa a camada física para poder

efetuar a seguinte chamada “*cogPhyFieldEntity.transmit(info, signal, duration, channel)*”. Esta chamada significa “camada física, transmita este sinal, deste rádio, com esta duração, neste canal”.

3.4.6.2 Receber um sinal

Esta funcionalidade é representada pelo método *“public void receive(Signal signal, Double power, Long duration”.* Tanto na transmissão quanto na recepção espera-se que o rádio cognitivo trate a informação e execute alguma ação, que no caso da recepção poderia ser no mínimo enviar o sinal para as camadas superiores.

3.4.6.3 Logging

Com o objetivo de disponibilizar uma perspectiva de visualização do tráfego de sinais entre os rádios na forma de *log*, efetuando o registro das operações de transmissão e recepção, possibilitando o rastreamento dos eventos e depuração, foi definida a classe Logging, a qual implementa o recurso java.util.logging, que permite que aplicações Java registrem o comportamento de suas aplicações de forma eficiente, rápida e prática. Na configuração utilizou-se apenas o nível de atuação INFO, que permite o registro de informações em tempo de execução. Os logs gerados foram configurados para serem registrados no arquivo chamado *“logging.log”* contendo suas respectivas informações (data/hora, classe, nível, mensagem,etc..), no formato XML.

O mais importante a se observar nestes registros de *log* é a informação contida no campo <message>.

Exemplo de saída no arquivo de log, no formato padrão e em seguida no formato XML:

Exemplo 1 - REGISTRO DE UM RÁDIO COGNITIVO

É quando um rádio registra-se na camada CogPhyField:

– Método sendo executado: Topography2D.ADDRADIO;

– Rádio que executou o método: RadioID: 1;

– Localização do rádio na grade: PONTO: (30,85);

– Tempo de simulação (*Simulation Time*) de execução do método: TIME: 0.

- Formato padrão:

May 01, 2010 11:19:24 PM br.com.datacoffee.cognet.tools.Logging log
INFO: Topography2D.ADDRADIO ->, RadioID: 1, PONTO: (30,85), TIME: 0

- Formato XML:

```
<record>
  <date>2010-05-01T23:19:24</date>
  <millis>1274062764520</millis>
  <sequence>0</sequence>
  <logger>CogNetLogging</logger>
  <level>INFO</level>
  <class>br.com.datacoffee.cognet.tools.Logging</class>
  <method>log</method>
  <thread>10</thread>
  <message>Topography2D.ADDRADIO -&gt;, RadioID: 1, PONTO: (30,85), TIME: 0</message>
</record>
```

Exemplo 2 - TRANSMISSÃO DE UM SINAL DE UM RÁDIO COGNITIVO

É quando um rádio transmite um sinal para o meio (canal):

– Método sendo executado: CogRadioMock.TRANSMIT;

– Rádio que executou o método: RadioID: 1;

– Sinal transmitido: SIGNAL: […];

– Pulses deste Signal: [(P1),(P2),(...),(Pn)];

– Informações do Pulse:(startMoment,stopMoment,Bit,potência do Bit);

– Potência de transmissão do Signal: TRANSMIT_POWER: 15.0;

– Tempo de simulação (*Simulation Time*) de execução do método: TIME: 50.

- Formato padrão:

May 01, 2010 11:19:24 PM br.com.datacoffee.cognet.tools.Logging log
INFO: CogRadioMock.TRANSMIT ->, RadioID: 1, SIGNAL: [(1.0,2.1041,HIGH,3.3209),
(2.1042,4.9479,LOW,8.5533),(4.948,5.9507,LOW,3.0159),(5.9508,5.9794,HIGH,0.086),
(5.9795,5.9915,NOSIGNAL,0.0),(5.9916,5.9977,HIGH,0.0183),(5.9978,5.9997,HIGH,0.0057),
(5.9998,5.9999,NOSIGNAL,0.0)], TRANSMIT_POWER: 15.0, TIME: 50

- Formato XML:

<record>
  <date>2010-05-01T23:19:24</date>
  <millis>1274062764590</millis>
  <sequence>2</sequence>
  <logger>CogNetLogging</logger>
  <level>INFO</level>
  <class>br.com.datacoffee.cognet.tools.Logging</class>
  <method>log</method>
  <thread>10</thread>
  **<message>CogRadioMock.TRANSMIT ->, RadioID: 1, SIGNAL: [(1.0,2.1041,HIGH,3.3209),
(2.1042,4.9479,LOW,8.5533),(4.948,5.9507,LOW,3.0159),(5.9508,5.9794,HIGH,0.086),
(5.9795,5.9915,NOSIGNAL,0.0),(5.9916,5.9977,HIGH,0.0183),(5.9978,5.9997,HIGH,0.0057),
(5.9998,5.9999,NOSIGNAL,0.0)], TRANSMIT_POWER: 15.0, TIME: 50</message>**
</record>

Exemplo 2 - RECEPÇÃO DE UM SINAL DE UM RÁDIO COGNITIVO

É quando um rádio recebe um sinal do meio:

- Método sendo executado: CogRadioMock.RECEIVE;
- Rádio que executou o método: RadioID: 2;
- Sinal recebido: SIGNAL: […];
- Pulses do Signal: [(P1),(P2),(...),(Pn)];
- Informações do Pulse:(startMoment,stopMoment,Bit,potência do Bit);
- Potência de recepção do Signal: RECEIVE_POWER: 7.8113900770737E-8;
- Tempo de simulação (*Simulation Time*) de execução do método: TIME: 300.

- Formato padrão:

May 01, 2010 11:19:28 PM br.com.datacoffee.cognet.tools.Logging log
INFO: CogRadioMock.RECEIVE ->, RadioID: 2, SIGNAL: [(1.0,29.1821,PU,14.091),
(29.1822,30.4952,PU,0.6565),(30.4953,30.877,PU,0.1908),(30.8771,30.9271,PU,0.025),
(30.9272,30.9782,PU,0.0255),(30.9783,30.9877,PU,0.0047),(30.9878,30.9892,PU,7.0E-4),
(30.9893,30.9952,PU,0.0030),(30.9953,30.9972,PU,0.0010),(30.9973,30.9975,PU,1.0E-4),
(30.9976,30.9985,PU,4.0E-4),(30.9986,30.9997,PU,6.0E-4),(30.9998,30.9999,PU,0.0)], RECEIVE_POWER:
7.8113900770737E-8, TIME: 300

- Formato XML:

<record>
  <date>2010-05-16T23:19:28</date>
  <millis>1274062768799</millis>
  <sequence>721</sequence>
  <logger>CogNetLogging</logger>

<level>INFO</level>
<class>br.com.datacoffee.cognet.tools.Logging</class>
<method>log</method>
<thread>10</thread>
<message>CogRadioMock.RECEIVE ->, RadioID: 2, SIGNAL: [(1.0,29.1821,PU,14.091),(29.1822,30.4952,PU,0.6565),(30.4953,30.877,PU,0.1908),(30.8771,30.9271,PU,0.025),(30.9272,30.9782,PU,0.0255),(30.9783,30.9877,PU,0.0047),(30.9878,30.9892,PU,7.0E-4),(30.9893,30.9952,PU,0.0030),(30.9953,30.9972,PU,0.0010),(30.9973,30.9975,PU,1.0E-4),(30.9976,30.9985,PU,4.0E-4),(30.9986,30.9997,PU,6.0E-4),(30.9998,30.9999,PU,0.0)], RECEIVE_POWER: 7.8113900770737E-8, TIME: 300</message>
</record>

Neste modelo estão sendo registrados em log as seguintes funcionalidades:

- Em que momento um rádio entra na *CogPhyField* e em que ponto da grade;

- Transmitir um sinal;

- Receber um sinal;

- Inserir um rádio da grade;

- Remover um rádio da grade.

3.5 Simulação

Como objeto de validação do modelo proposto e do código implementado sugeriu-se a realização de uma simulação que apresentasse o funcionamento da camada física. A simulação será executada sob o simulador de eventos discretos *JIST*, respeitará os requisitos propostos e a modelagem utilizará as estruturas definidas na arquitetura e nos componentes.

Como objetivo específico da simulação cita-se a transmissão de sinais, a recepção de sinais, a sincronização da transmissão e da recepção em conformidade com o tempo de simulação, assim como a verificação da aplicação das perdas relacionadas aos modelos de propagação.

Do ponto de vista de implementação em código, a simulação é simples mas deve ser estruturada seguindo as premissas do simulador de eventos discretos JIST.

A classe que implementa a simulação chama-se *MainSim*, cuja inicialização prevê o recebimento de dois parâmetros iniciais, sendo o primeiro representando o

comprimento da grade e o segundo o número de nós (rádios cognitivos). O método inicial da simulação possui a assinatura "*public void init(String comprimento, String nos)*".

Primeiramente, através do método de assinatura *"public CogPhyField createSim(int length, int nodes)"* instancia-se o modelo de grade, define-se o modelo de *fading* e *pathloss*, gera-se o conjunto de sinais, instancia-se a camada física *CogPhyField* , instancia-se o conjunto de nós, e registra-se os nós na camada física. Foram gerados 5 tipos de sinais, cada um com uma combinação de Bit específica, mas que abrangesse todos os tipos de Bits possíveis pelo modelo. Conforme a dinâmica de execução da simulação, cada rádio enviará os cinco sinais aos demais rádios num tempo distinto e aleatório de 1 a 5.

Dinâmica de execução:

-> MainSim (tamanho da grade + número de nós);

-> Instancia-se o modelo de grade;

-> Define-se o modelo de fading e pathloss;

-> Gera-se o conjunto de sinais;

-> Instancia-se a camada física CogPhyField;

-> Instancia-se o conjunto de nós;

-> Registra-se os nós na camada física;

-> Efetua transmissões;

-> Efetua recepções.

Para a simulação também foram gerados sinais pré-definidos com a seguinte configuração, no intuito de abranger todos os possíveis sinais que trafegariam no meio:

a) HIGH / LOW / NOSIGNAL;

b) PU / NOSIGNAL;

c) HIGH / NOSIGNAL;

d) LOW / NOSIGNAL;

e) HIGH / LOW.

O simulador SWAN possui em seu código fonte uma classe chamada *Constants*, que na verdade é uma interface Java, onde estão declarados parâmetros que podem ser utilizados por todo o código do simulador. Visando manter um padrão próximo do que já foi implementado e que pode ser aproveitado, utilizou-se os parâmetros definidos nesta interface. Por exemplo, a instanciação de um rádio requer a definição de diversos parâmetros (largura de banda, frequência de operação, potência de transmissão, ganho da antena, sensibilidade, etc) que já constavam nesta interface. Nem todos os parâmetros lá definidos foram utilizados, mas dos que foram na instanciação dos rádios, cita-se:

```
/** Default radio frequency (units: Hz). */
public static final double FREQUENCY_DEFAULT = 2.4e9; // 2.4 Ghz
/** Default radio bandwidth (units: bits/second). */
public static final int BANDWIDTH_DEFAULT = (int)1e6; // 1Mb/s
/** Default transmission strength (units: dBm). */
public static final double TRANSMIT_DEFAULT = 15.0;
/** Default antenna gain (units: dB). */
public static final double GAIN_DEFAULT = 0.0;
/** Default radio reception sensitivity (units: dBm). */
public static final double SENSITIVITY_DEFAULT = -91.0;
/** Default radio reception threshold (units: dBm). */
public static final double THRESHOLD_DEFAULT = -81.0;
/** Default temperature (units: degrees Kelvin). */
public static final double TEMPERATURE_DEFAULT = 290.0;
/** Default temperature noise factor. */
public static final double TEMPERATURE_FACTOR_DEFAULT = 10.0;
/** Default ambient noise (units: mW). */
public static final double AMBIENT_NOISE_DEFAULT = 0.0;
```

/** Default minimum propagated signal (units: dBm). */

public static final double PROPAGATION_LIMIT_DEFAULT = SENSITIVITY_DEFAULT;

Através de todo este conjunto de passos tem-se pronta uma camada física com um conjunto de rádios cognitivos com poder de transmissão e recepção.

Em seguida, ainda no método *init,* através desta chamada *"data.radioEntity.transmit(signalList.get(j), time, data.info.getChannel())"* é que os rádios iniciarão o processo de transmissão dos sinais anteriormente gerados.

Para avaliar se os rádios estavam transmitindo, recebendo e se os modelos de perda estavam também funcionando, definiu-se uma topologia física básica onde variou-se o comprimento da grade (ver item 3.4.2 – Funcionalidades – Classe Topography2D) e o número de nós, ficando assim:

a) comprimento da grade 100 , número de nós 5;

b) comprimento da grade 500 , número de nós 50;

c) comprimento da grade 1000 , número de nós 100;

Uma outra variação possível neste modelo de simulação seria a troca dos modelos de perdas de potência, *fading e pathloss*, entretanto não está no escopo a avaliação da acurácia destes modelos ou qual a consequência de um ou de outro na potência do sinal recebido. Assim na configuração destes parâmetros optou-se por:

- *Fading* – modelo FreeSpace;

- *PathLoss* – modelo None (nenhum), pois o Rician é um modelo indicado para situações onde haja mobilidade dos nós, o que não é o caso.

Por fim, aguardou-se a finalização da execução de cada uma das 3 configurações acima citadas e a consequente geração dos arquivos de log para análise.

4 Análise e Resultados

Findo o processo de execução das simulações, um fator que chamou a atenção e mereceu análise foi, inicialmente, o tempo de processamento de tais requisições, em cada simulação, e o tamanho dos *logs* gerados.

1) comprimento da grade 100 , número de nós 5;

- BUILD SUCCESSFUL (total time: 3 seconds)

- totalling 692.1 KB

2) comprimento da grade 500 , número de nós 50;

- BUILD SUCCESSFUL (total time: 1 minute 27 seconds)

- totalling 386.1 MB

3) comprimento da grade 1000 , número de nós 100;

- BUILD SUCCESSFUL (total time: 11 minutes 46 seconds)

- totalling 3.4 GB

Diante destes resultados, observou-se que quanto maior o número de nós, maior será a quantidade de informação a ser registrada e, assim sendo, há a necessidade de melhoria em relação a forma de registro do tráfego de sinais entre os rádios.

As métricas de interesse nestas simulações giram em torno da quantidade de sinais transmitidos, quantidade de sinais recebidos, duração da transmissão do sinal, em que tempo os sinais foram recebidos e se a potência de transmissão sofreu perdas. Assim, definiu-se questões a serem respondidas:

a) Quantos sinais foram transmitidos ?

b) Quantos sinais foram recebidos ?

c) Qual a frequência de recebimento destes sinais e em que tempo ?

d) Dado um sinal transmitido no tempo 0, qual sua duração, quantas vezes ele

foi recebido e em que tempo ?

e) Dado este mesmo sinal, cuja potência de transmissão é padrão, houveram perdas nesta potência ? Quais os valores para os casos recebidos do item anterior ?

Então para efeito de análise, escolheu-se a simulação número 1 - comprimento da grade 100 , número de nós 5 – e utilizando o ambiente *shell* do sistema operacional Linux para varrer os arquivos de log desta simulação, obteve-se as seguintes respostas:

a) 25 sinais foram transmitidos, conforme o esperado.

$ cat logging.log | grep CogRadioMock.TRANSMIT | wc -l

Resposta: 25

b) 100 sinais foram recebidos, conforme o esperado.

$ cat logging.log | grep CogRadioMock.RECEIVE | wc -l

Resposta: 100

c) como a duração considerada é um número randômico de 1 a 5 (ver classe MainSim.java), desconsidera-se o tempo 0, e conforme abaixo, tudo dentro do esperado, ou seja 100 Signals recebidos.

$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 1" | wc -l

Resposta: 40

$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 2" | wc -l

Resposta: 0

$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 3" | wc -l

Resposta: 20

$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 4" | wc -l

Resposta: 20

$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 5" | wc -l

Resposta: 20

d) Considerando o Signal abaixo, sua duração é 3, foi recebido 20 vezes, mas no tempo 3 somente 4 vezes, conforme o esperado.

INFO: CogRadioMock.TRANSMIT, RadioID: 3, SIGNAL: [(1.0,2.482,LOW,2.2233), (2.4821,4.9964,LOW,3.772),(4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598), (9.3969,10.5724,LOW,1.7635),(10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456), (10.9611,10.9788,LOW,0.0266),(10.9789,10.9913,LOW,0.0186),(10.9914,10.9928,LOW,0.0021), (10.9929,10.9964,LOW,0.0053),(10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4), (10.9998,10.9999,LOW,2.0E-4)], TRANSMIT_POWER: 15.0, TIME: 0 **DURACAO: 3**

$cat logging.log | grep "(1.0,2.482,LOW,2.2233),(2.4821,4.9964,LOW,3.772), (4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598),(9.3969,10.5724,LOW,1.7635), (10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456), (10.9611,10.9788,LOW,0.0266),(10.9789,10.9913,LOW,0.0186), (10.9914,10.9928,LOW,0.0021),(10.9929,10.9964,LOW,0.0053), (10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4), (10.9998,10.9999,LOW,2.0E-4)" | grep RECEIVE | grep "TIME: 3" | wc -l

Resposta: 4

e) Abaixo as potências recebidas em destaque para cada rádio, tudo conforme o esperado.

$ cat logging.log | grep "(1.0,2.482,LOW,2.2233),(2.4821,4.9964,LOW,3.772), (4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598),(9.3969,10.5724,LOW,1.7635), (10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456), (10.9611,10.9788,LOW,0.0266),(10.9789,10.9913,LOW,0.0186), (10.9914,10.9928,LOW,0.0021),(10.9929,10.9964,LOW,0.0053), (10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4), (10.9998,10.9999,LOW,2.0E-4)" | grep RECEIVE | grep "TIME: 3"

Resposta: 4 registros com este padrão de informação

INFO: CogRadioMock.RECEIVE, RadioID: 1, SIGNAL: [(1.0,2.482,LOW,2.2233),

(2.4821,4.9964,LOW,3.772),(4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598),
(9.3969,10.5724,LOW,1.7635),(10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456),
(10.9611,10.9788,LOW,0.0266),(10.9789,10.9913,LOW,0.0186),(10.9914,10.9928,LOW,0.0021),
(10.9929,10.9964,LOW,0.0053),(10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4),
(10.9998,10.9999,LOW,2.0E-4)], **RECEIVE_POWER: 2.0488896197039424E-7**, TIME: 3

INFO: CogRadioMock.RECEIVE, RadioID: 2, SIGNAL: [(1.0,2.482,LOW,2.2233),
(2.4821,4.9964,LOW,3.772),(4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598),
(9.3969,10.5724,LOW,1.7635),(10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456),
(10.9611,10.9788,LOW,0.0266),(10.9789,10.9913,LOW,0.0186),(10.9914,10.9928,LOW,0.0021),
(10.9929,10.9964,LOW,0.0053),(10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4),
(10.9998,10.9999,LOW,2.0E-4)], **RECEIVE_POWER: 8.168766266273502E-8**, TIME: 3

INFO: CogRadioMock.RECEIVE, RadioID: 4, SIGNAL: [(1.0,2.482,LOW,2.2233),
(2.4821,4.9964,LOW,3.772),(4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598),
(9.3969,10.5724,LOW,1.7635),(10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456),
(10.9611,10.9788,LOW,0.0266),(10.9789,10.9913,LOW,0.0186),(10.9914,10.9928,LOW,0.0021),
(10.9929,10.9964,LOW,0.0053),(10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4),
(10.9998,10.9999,LOW,2.0E-4)], **RECEIVE_POWER: 2.1623205957805312E-8**, TIME: 3

INFO: CogRadioMock.RECEIVE, RadioID: 5, SIGNAL: [(1.0,2.482,LOW,2.2233),
(2.4821,4.9964,LOW,3.772),(4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598),
(9.3969,10.5724,LOW,1.7635),(10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456),
(10.9611,10.9788,LOW,0.0266),(10.9789,10.9913,LOW,0.0186),(10.9914,10.9928,LOW,0.0021),
(10.9929,10.9964,LOW,0.0053),(10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4),
(10.9998,10.9999,LOW,2.0E-4)], **RECEIVE_POWER: 1.8061007606907763E-8**, TIME: 3

5 Conclusão

O desenvolvimento de simulações como ponto de partida para a pesquisa científica é realmente necessário. A cada dia novas idéias surgem e com elas podem surgir novas tecnologias cujo foco é apresentar algo novo ou é a melhoria de algo existente. As redes cognitivas se enquadram neste último, e pelos benefícios que prometem, parece ser este um caminho sem volta, pelo menos até o momento.

Por tratar-se de um assunto ainda em desenvolvimento, e não existirem comercialmente dispositivos para redes sem fio com comportamento cognitivo, ainda há tempo para surgirem novas idéias e novas formas de conduzir a dinâmica existente no processo de coexistência entre usuários primários e secundários dentro do espectro eletromagnético, dando liberdade também para que os pesquisadores invistam seu tempo no desenvolvimento de novos protocolos através das simulações.

Este trabalho objetivou a construção de um modelo com idéias novas e a implementação e simulação deste modelo baseado em outros já existentes, do ponto de vista do funcionamento. A partir da definição de sua arquitetura na seção 3.2, definição de sua composição na seção 3.2.1 e implementação na seção 3.5 para que executasse sob o simulador é que foi possível, a cada passo, avançar na compreensão das características envolvidas no processo de propagação do sinal e o reflexo disso na sua qualidade, conforme objetivo demonstrado na seção 1.3.1. Propor um modelo de camada física que atendesse os requisitos de um rádio cognitivo, conforme objetivo demonstrado na seção 1.3.1, foi com certeza a parte mais difícil deste processo como um todo, principalmente considerando a abrangência do tema e da quantidade de informação necessária para que se esteja seguro ao ponto de partir para a sua implementação.

Ao definir a arquitetura e a composição do sinal, já foi possível agregar ao modelo de camada PHY as características necessárias a um rádio cognitivo, conforme objetivo demonstrado na seção 1.3.1.

As funcionalidades necessárias para o tratamento do sinal, como *fading e pathloss,* já encontravam-se implementadas no SWANS, entretanto, a aplicação delas no modelo aqui proposto só foi possível ao adaptar estes modelos.

Assim, a inclusão das funcionalidades acima citadas, a definição de um ambiente físico fictício através da classe Topography2D, efetuar a transmissão e recepção

de sinais de forma sincronizada entre os rádios com a ajuda do “*simulation time*” do JIST, a inclusão de sinais que representassem os usuários primários através do Bit PU e sua inclusão na simulação, assim como a inclusão da perspectiva de gravação dos logs relacionados a este processo através da API do Java java.util.logging, conforme objetivo demonstrado na seção 1.3.1 , permitiram definir o ambiente de simulação.

Claro que iniciar a simulação deste modelo sem ter toda a pilha de protocolos pronta e sem um único *Signal* para ser transmitido poderia, para alguns casos, inviabilizar a sua execução, entretanto, para o escopo deste trabalho, a definição de um rádio com funcionalidades restritas através da classe *CogRadioMock* (rádio cognitivo falso) e a definição de um gerador de sinais através da classe *SignalGenerator*, atendeu com sucesso esta necessidade, conforme objetivo demonstrado na seção 1.3.1.

Já a classificação do canal, que é uma funcionalidade desejada num rádio cognitivo completo, a nível de camada de rádio, poderá ser inferida a partir do modelo de sinal e seus atributos aqui propostos.

Uma vez que linguagem de programação Java permite, no uso de *interfaces,* a definição de funcionalidades mínimas e a especialização delas ou inclusão de novas nos objetos que as implementam, o modelo aqui proposto seguiu esta tendência de uso e assim tentou deixar o modelo o mais aberto possível para inclusão de novas funcionalidades, conforme objetivo demonstrado na seção 1.3.1.

Com base nos objetivos propostos e na análise dos resultados das simulações, acredita-se ter alcançado a meta deste trabalho.

5.1 Trabalhos Futuros

A camada física aqui proposta é somente a ponta do *iceberg* no desenvolvimento da pilha de protocolos. Existe mais de um forma de executar uma funcionalidade e uma normalmente considera aquilo que a outra não considerou, ou foram feitas para situações distintas.

Conforme verificado no funcionamento do SWANS, o modelo de recepção de sinais/pacotes está implementado na camada de rádio, entretanto, se formos considerar

que, numa situação de recepção de dois ou mais sinais, ao invés de considerar somente aquele de maior potência quisermos degradar este sinal de maior potência efetuando a interferência nele através da mescla de *Bits* cujo *time* seja o mesmo, daí acreditamos que esta funcionalidade talvez tenha que ser executada pela camada física. Se este for o caminho, haverá a necessidade de implementar os modelos analíticos de recepção de sinais, que permitem determinar com sucesso o recebimento de pacotes, baseados no modelo de BER (*Bit Error Rate*), que é uma função do *pathloss*, esquema de modulação e a distribuição de enfraquecimento (*fading*) de *Ricean e Rayleigh*, ou baseado no modelo de SNR.

A acurácia dos modelos de propagação de sinal é um fator relevante e, como em qualquer área, existem os mais precisos e os mais genéricos, por assim dizer. Conforme a literatura pesquisada, o *Simulation of Indoor Radio Channel Impulse Response Models with Impulse Noise – (*SIRCIM) - faz muito mais do que computar a potência do rádio. Ele examina o *payload* do pacote recebido e determina, baseado na porcentagem de *bits* corretamente recebido se o pacote foi ou não recebido corretamente. Embora possua um custo computacional alto, ele ainda fornece resposta ao impulso do canal em nível de sinal, possibilitando contabilizar o *fading* do canal e computar a interferência *inter-symbol* causada por ele, se mostrando como um bom trabalho futuro a ser implementado na *CogPhyField*.

Outra característica desejada e comumente implementada nos simuladores de redes sem fio é a questão da mobilidade dos nós, sendo também um outro tema para trabalho futuro.

Além disso, conforme identificado na análise dos resultados, há a necessidade de implementação de um modelo de *logging* mais adequado, que não interfira no tempo de execução da simulação e nem na quantidade de espaço necessária para armazenamento dos mesmos, considerando o pequeno número de nós utilizados na simulação 3 que foi de 100.

Referências Bibliográficas

[1] R. Barr, "SWANS-Scalable Wireless Ad hoc Network Simulator Users Guide," Retrieved March, 2006, pp. 1-15.

[2] R. Barr, "JiST-Java in Simulation Time Users Guide," Cornell University, March, vol. 19, 2004.

[3] Cognitive Radio Cognitive Network Simulator. Disponível em <http://www.ece.mtu.edu/~ljialian/> . Acesso em: 15/03/2010;

[4] CORDEIRO, Carlos; CHALLAPALI, Kiran; BIRRU, Dagnachew. IEEE 802.22: An Introduction to the First Wireless Standard based on Cognitive Radios. Journal Of Communications, Briarcliff Manor, p. 38-47. abr. 2006.

[5] XU, Fangmin et al. Architecture for Next-Generation Reconfigurable Wireless Networks using Cognitive Radio. Cognitive Radio Oriented Wireless Networks And Communications, 2008. Crowncom 2008. 3rd International Conference On, Singapore, n. , p.1-5, 15-17 maio 2008.

[6] G. S. Fishman. Principles of Discrete Event Simulation. John Wiley & Sons, Inc., New York, NY, USA, 1978.

[7] Fórmula de Friis. Disponível em: <http://pt.wikipedia.org/wiki/Fórmula_de_Friis>. Acesso em: 13/08/2009.

[8] FCC (2002). Spectrum policy task force report, FCC 02-155.

[9] The Source for Java Developers. Disponível em: <http://java.sun.com/> . Acesso em : 01/11/2009.

[10] G. Judd and P. Steenkiste, "A software architecture for physical layer wireless network emulation," Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization - WiNTECH '06, 2006, p. 2.

[11] Misra, J. Distributed discrete event simulation. ACM Computing Surveys 18, 1 (Mar. 1986), 39–65.

[12] Nicol, D. M., and Fujimoto, R. M. Parallel simulation today. Annals of Operations Research (Dec. 1994), 249–285.

[13] Fujimoto, R. M. Parallel discrete event simulation. Communications of the ACM 33, 10 (Oct. 1990), 30–53.

[14] Fujimoto, R. M. Parallel and distributed simulation. In Winter Simulation Conference (Dec. 1995), pp. 118–125.

[15] F. Kargl and E. Schoch, "Simulation of manets: a qualitative comparison between jist/swans and ns-2," Proceedings of the 1st international workshop on System evaluation for mobile platforms, ACM, 2007, p. 46.

[16] Miguel. M. Campista, Propagação e antenas aplicadas ao IEEE 802.11. Disponível em: <http://www.gta.ufrj.br/seminarios/semin2003_1/miguel/index.html> . Acesso em 05/03/2010

[17] The ns-3 network simulator. Disponível em: <http://www.nsnam.org/>. Acesso em: 03/01/2010.

[18] T. Rappaport, Comunicações sem fio, Pearson Prentice Hall, 2008.

[19] Redes ad hoc. Disponível em: <http://pt.wikipedia.org/wiki/Redes_ad_hoc> . Acesso em 17/12/2009.

[20] G. Riley, R. Fujimoto, and M. Ammar. A generic framework for parallelization of network simulations. In Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication, March 1999.

[21] J. Seybold, Introduction to RF propagation, Wiley-Interscience, 2005.

[22] R. W. Thomas, L. A. DaSilva, and A. B. Mackenzie, “Cognitive Networks,” Proc. IEEE DySPAN 2005, Nov. 2005, pp. 352–60.

[23] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a library for parallel simulation of large-scale wireless networks", Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS '98 (Cat. No.98TB100233), pp. 154-161.

[24] R. De S. Mendes, E. R. Garcia e C. B. Westphall, “Um Framework para Operação na Camada de Rádio em Redes Cognitivas”, The Sixth International Conference on Networking and Services - ICNS 2010 - March 7-13, 2010 - Cancun, Mexico.

Anexos

Artigo

Um Modelo de Simulação para Camada Física de Rádios Cognitivos

Everton Rodrigues Garcia

Laboratório de Redes e Gerência, Universidade Federal de Santa Catarina (UFSC) , Florianópolis – SC – Brasil

evertonr@inf.ufsc.br

Abstract. *With the increasing use of unlicensed spectrum bands, the under-utilization of licensed tracks presented as a possible solution from the opening of these frequency bands for dynamic access and opportunistic. In this context, cognitive wireless networks, through cognitive radios, promises to be the future of wireless communications, intended to treat it with greater efficiency, flexibility and reliability of the race problem of spectrum interference and degradation of communications. Since it is a matter still under development, several models of the physical layer for cognitive radios are offered in combination and there are few simulators that allow the simulation of these new features. The use of simulation as a methodology for scientific development is very important because they reproduce so fictitious, with less effort and cost, the behavior of events that occur in actual environmental conditions.*
This paper proposes a model of physical layer for wireless networkscognitive (Ad hoc), presenting its architecture, operation, and explanation on the implementation and simulation of this model using techniques based on discrete event simulation.

Resumo. *Com a crescente utilização das faixas do espectro não licenciado, a subutilização das faixas licenciadas apresentou-se como uma possível solução a partir da abertura destas bandas de frequência para o acesso dinâmico e oportunístico. Neste contexto as redes sem fio cognitivas, através dos rádios cognitivos, prometem ser o futuro das comunicações sem fio, pois objetivam tratar com maior eficiência, flexibilidade e confiabilidade o problema da disputa de espectro, interferência e degradação das comunicações. Por tratar-se de um assunto ainda em desenvolvimento, diversos modelos de camada física para rádios cognitivos são propostos e em associação ainda são poucos os simuladores que permitem a simulação destas novas funcionalidades. O uso de simulações como metodologia para o desenvolvimento científico é muito importante porque reproduzem de forma fictícia, com menor esforço e custo, o comportamento de eventos que ocorrem em condições ambientais reais.*
O presente trabalho propõe um modelo de camada física para redes sem fio cognitivas (Ad-hoc), apresentando sua arquitetura, funcionamento, e a explanação sobre a implementação e simulação deste modelo utilizando técnicas baseadas em simulação discreta de eventos.

1 Introdução

As redes sem fio como forma de comunicação associam-se cada vez mais às necessidades do mundo moderno que não se adapta mais à dependência da infraestrutura de rede

cabeada. A sua popularização pode ser observada nas classes de aplicações que antes só eram disponibilizadas através da rede cabeada, como acesso a internet através de pontos de acesso, soluções de voz sobre IP e até mesmo na interligações de escritórios remotos.

O crescimento do número de usuários que utilizam as diversas tecnologias sem fio dentro do espectro eletromagnético, considerado como não licenciado, pode fazer com que ele torne-se congestionado e as interferências decorrentes deste processo prejudicam a qualidade das transmissões. Identificando esta tendência, desde 2004, com a criação do 802.22 Work Group, o Institute of Electrical and Electronics Engineers (IEEE) vem desenvolvendo padrões para redes sem fio regionais (Wireless Regional Area Network - WRAN) no tocante às camadas Física (PHY) e Controle de Acesso ao Meio (MAC) para uso por dispositivos (usuários secundários) que utilizarão o espectro licenciado aos usuários primários, sem causar interferências prejudiciais a eles, utilizando técnicas de rádios cognitivos.

Por definição, um rádio cognitivo deve realizar sensoreamento do espectro, decisão no espectro, compartilhamento do espectro e mobilidade no espectro com o objetivo de mudar seus parâmetros de transmissão, baseado nas interações com o ambiente no qual ele opera, a fim de obter o melhor nível de qualidade de transmissão possível, eficiência do espectro e, principalmente, não interferir nos usuários primários. Neste contexto, o uso de simuladores é muito importante, pois a adoção de novos padrões só é possível a partir do estudo, simulação e desenvolvimento de um protótipo a partir de um modelo definido. Para as redes sem fio o desenvolvimento e testes de protocolos de rede, a comparação de desempenho de diferentes protocolos, a necessidade de grande escalabilidade, as restrições de custo nos testes e a necessidade de reproduzir condições ambientais onde o cenário possa ser repetido uma infinidade de vezes, dependem do suporte das simulações.

O presente trabalho faz uma abordagem simples de como simular a propagação do sinal em redes cognitivas propondo um modelo de camada física que atenda os requisitos do trabalho [10], baseando-se nos simuladores de redes sem fio existentes e assim proporcionando a estrutura necessária para que as camadas superiores possam ser desenvolvidas e acopladas a este modelo, sendo possível a simulação da pilha de protocolos, e suas responsabilidades, de uma rede cognitiva completa.

O restante deste trabalho está organizado apresentando seções que descrevem os conceitos principais aqui envolvidos, a descrição do modelo especificado, a descrição da simulação executada e a conclusão e trabalhos futuros.

2 Background

2.1 Redes Cognitivas

Embora exista uma faixa do espectro eletromagnético licenciada para somente determinados tipos de sinais, os demais tecnologias disputam uma pequena porção considerada não licenciada, ou seja, sem regulamentação. O uso crescente desta pequena porção aliado ao surgimento de novas tecnologias que também utilizarão este recurso faz com que o espectro torne-se congestionado devido ao canal limitado. Isso não aconteceria se a porção licenciada pudesse ser compartilhada para acesso dinâmico e oportunístico com novos dispositivos que, quando comunicassem, não interferissem nos dispositivos que lá comunicam de direito.

Baseados em técnicas de inteligência artificial, os rádios cognitivos, através de suas heurísticas, tem a intenção de contribuir para a solução dos problemas da disputa pelo espectro através do sensoreamento e detecção da presença ou ausência de sinais que competem, adaptando inteligentemente suas características de operação para condições que permitam no mínimo a coexistência entre estes sinais

Desta forma, os rádios cognitivos emergem como uma tecnologia promissora para a

solução do problema da injusta utilização do espectro através do seu compartilhamento dinâmico, onde suas principais funcionalidades desejadas são: [4] sensorear periodicamente o espectro, inteligentemente efetuar a detecção de ocupação e uso do espectro, e por fim tomar a decisão de ajustar seus parâmetros para oportunisticamente comunicar entre as brechas que os usuários primários deixam no espectro. Seus principais objetivos são: maximizar a transmissão do sistema secundário enquanto garante sob controle a interferência gerada para o sistema primário. O ajuste de parâmetros como a potência de transmissão é para evitar ultrapassar o limite de interferência da rede primária e manter a conectividade entre os nós da rede, além da possibilidade de troca de canal quando o anterior apresenta-se sobrecarregado.

Assim, as redes cognitivas são capazes de adaptar seu funcionamento através da captação de estímulos externos, gerando conhecimento a partir desta interação e tomada de decisão com base no conhecimento adquirido.

Conforme definido em [9], uma rede cognitiva pode ser definida como uma rede que possui um processo cognitivo que pode perceber as condições da rede atual e então planejar, decidir e atuar nessas condições. A rede pode aprender com estas adaptações e usar estes conhecimentos para tomar futuras decisões, levando em consideração os objetivos fim-a-fim, ou seja, todos os elementos envolvidos na transmissão de um fluxo de dados.

Desta forma, as redes cognitivas desafiam a noção atual da escassez de espectro e encaram o problema como sendo um problema de acesso e compartilhamento ao espectro.

2.2 Sensoreamento no Espectro

Os maiores desafios na definição dos modelos para redes cognitivas apresentam-se no contexto da garantia de coexistência, dentro do espectro eletromagnético, entre os usuários licenciados e os não licenciados.

Em [10] apresenta-se uma possível abordagem que baseia-se nas necessidades de um rádio cognitivo e num método bem definido para a classificação do um sinal sensoreado.

Conforme o trabalho [10], as arquiteturas crosslayer introduzem desordem na atribuição de responsabilidades entre as camadas de rede nas propostas de redes cognitivas. Assim, seu foco está em definir as responsabilidades de um framework para o sensoriamento eficaz do espectro:

- Definir os canais em que tem capacidade de operar;
- Definir as modulações com as quais tem capacidade de operar;
- Analisar o espectro;
- Movimentar-se no espectro;
- Transmitir e receber dados com ciência de sua respectiva potência;
- Permitir que um método de sensoriamento mais robusto possa sugerir modulações a serem assumidas;

E a partir destas responsabilidades:

- Fornecer informações sobre o hardware;
- Efetuar operações sobre o espectro;
- Efetuar a classificação do sinal;
- Efetuar a classificação do canal;
- Efetuar recomendações às camadas superiores.

Além disso, verifica-se neste trabalho que o acesso dinâmico ao espectro depende de quatro características básicas:

- Sensoreamento do espectro;
- Gerenciamento do espectro;
- Compartilhamento do espectro;
- Mobilidade no espectro.

O modelo acima conduz à classificação dos sinais de rádio sensoreados no espectro, especifica um método de classificação de ocupação do canal de acordo com a seqüência dos sinais sensoreados, assim como, de posse das informações obtidas, possibilita a caracterização, seleção e reconfiguração no espectro. Seu objetivo é atender os requisitos de um rádio cognitivo no tocante aos problemas de reconfigurabilidade dos seus parâmetros, interferência a usuários primários, controle de sensoreamento do espectro, ocupação do espectro, decisões no espectro, compartilhamento do espectro e mobilidade no espectro. Assim, o seu framework apresenta um modelo de entrada/saída de dados que envolvem a interação entre a camada de RÁDIO e a camada MAC, entretanto, as questões relacionadas a camada PHY não estão no seu escopo.

Conforme visto na figura abaixo, um problema a ser tratado é a questão da entrada/saída de sinais/dados da camada de RÁDIO para a camada PHY e sua propagação para os demais rádios, pois o sinal a ser entregue numa transmissão sofre variações em suas características até a chegada no seu destino, dependendo principalmente da relação sinal-ruído, atenuação e perda de caminho.

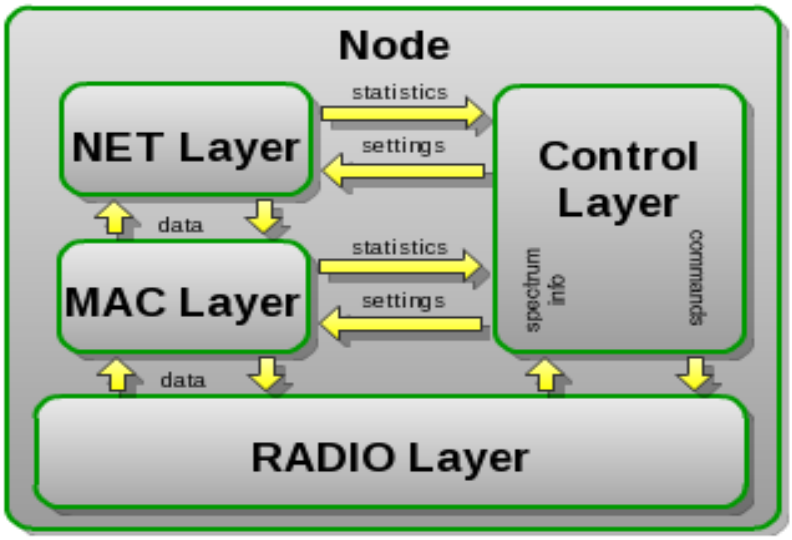

Figura 1 - Arquitetura parcial [Mendes, 2009]

2.3 Simuladores

Entende-se por simuladores de eventos discretos [5] o fato da simulação ser executada em passos discretos (tempo) e que os eventos são utilizados como meio de comunicação entre as entidades de simulação. Os simuladores assim ditos permitem modelar uma rede de computadores arbitrária, especificando o comportamento de ambos os nós e canais de comunicação. Além de permitir a simulação e testes de novos sistemas que muitas vezes sequer foram desenvolvidos em hardware, os simuladores proporcionam condições ambientais que podem ser repetidas uma infinidade de vezes.

Os modelos existentes de simulação discretizada são codificados como programas orientados a eventos, onde os eventos são processados em sua ordem casual, atualizando o estado da simulação de acordo com o modelo utilizado e possivelmente escalonando mais eventos para

simulação.

Conforme [8], para simulação de redes móveis ad hoc há a necessidade dos seguintes componentes:

- Um software de simulação – podendo ser um programa desenvolvido pelo pesquisador ou utilizando os programas existentes, como: ns-2, OPNET Modeler, Omnet+ + etc.;

- Uma emulação em software do ambiente físico que engloba o modelo de propagação do sinal de rádio, área de simulação e modelo de mobilidade do nó; - Uma implementação em software de todos os elementos de rede que estão abaixo da camada de rede como: emulação do rádio, a rede camada-2 (e.g. IEEE 802.11 WLAN), etc;

- Uma implementação dos protocolos e aplicações como os Ad hoc On- Demand Distance Vector (AODV) ou o Dynamic Source Routing (DSR) para roteamento, TCP / UDP e um gerador de tráfego;

- Um mecanismo para configurar parâmetros para todos os componentes de simulação acima citados e outro para a coleta de resultados, como por exemplo registrando estatísticas num arquivo de log ou base de dados;

2.3.1 JIST / SWANS – Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator

Conforme o seu guia do usuário, o JIST foi desenvolvido por Rimon Barr na Cornell University e vem introduzir uma nova abordagem para simuladores, funcionando como motor de simulação de eventos discretos, usado como ferramenta científica e baseado na linguagem de programação Java [6], onde a transparência, a eficiência e a padronização são a base da proposta.

Transparência implica que as simulações criadas pelo usuário serão automaticamente adaptadas à semântica de tempo de simulação do simulador; eficiência implica que o tempo de execução da simulação como um todo deva ser melhor do que os sistemas existentes; e seguir um padrão implica que as simulações possam ser escritas numa linguagem de programação convencional e padronizada já existente.

Ele foi desenvolvido com o objetivo de suprir algumas deficiências dos simuladores de eventos discretos existentes na época, no que tange à transparência, eficiência e padronização. Este último como sendo o grande diferencial, pois o código a ser simulado sob o JIST não necessita ser escrito numa linguagem específica para simulações ou numa linguagem com um propósito específico.

Pelo contrário, ele introduz a semântica de tempo de execução para simulação de programas escritos na linguagem Java e roda sob uma máquina virtual (JVM – Java Virtual Machine) não modificada, convertendo-a num sistema escalonador de eventos, modificando a forma de condução das invocações dos métodos das entidades simuladas, de forma flexível , eficiente e escalável.

Ele combina o uso da semântica de simulação, encontrada em linguagens de simulação customizadas e bibliotecas de simulação, com as funcionalidades de uma linguagem moderna, robusta e eficiente. A chave disso tudo é poder executar eficientemente programas de simulações dentro do contexto de uma linguagem moderna e popular.

Tempo de Simulação (Simulation Time)

Em Java, a JVM funciona como um motor de execução baseado em pilha e com semânticas de execução bem definidas. Neste modelo, o andamento do tempo não depende do progresso das aplicações e o relógio do sistema avança indiferentemente da quantidade de instruções processadas.

No JIST o progresso do tempo depende diretamente do progresso das aplicações. O relógio das aplicações, que representam o tempo de simulação, não avançam para o próximo ponto

discreto no tempo até que todos os processamentos para o corrente tempo de simulação seja completado. Neste modelo, porções de instruções individuais de bytecode da aplicação são processados sequencialmente, seguindo a semântica de controle de fluxo padrão da JVM e o tempo da aplicação continua inalterado. O código da aplicação pode avançar no tempo somente através da chamada do sleep(n) (da JistAPI), ou seja, qualquer instrução executa em tempo igual a zero, e só avança para o próximo tempo após o sleep(n) ser invocado, onde n identifica a quantidade de tempo a ser avançada.

Benefícios

Incluir a semântica de tempo de simulação dentro da linguagem Java traz consigo alguns benefícios como o reuso de código, o uso de bibliotecas padrão existentes, uso de compiladores existentes, os benefícios da garbage collection, a type-safety e a reflexão, que são características desejáveis, comprovadamente funcionais e já amadurecidas, sob o ponto de vista de desenvolvimento de aplicações. Assim, a curva de aprendizado necessária será menor, proporcionando o reuso de código na construção de simulações, além do fato de poder rodar dentro de uma máquina virtual padronizada que oferece um ambiente eficiente, bastante otimizado, portável e ao mesmo tempo confiável, facilitando a comunicação entre o Kernel de simulação e a simulação em si. Além disso, o fato de rodarem dentro de um mesmo processo reduz a necessidade de serializações e o overhead oriundo da mudança de contexto intrínseca das simulações.

Sob o ponto de vista de aplicação, observam-se benefícios da verificação automática de tipagem, da estrutura de um evento sem a necessidade de grandes estruturas de dados para isso e do fato de ter disponível informações como a localização do evento despachado e o estado da entidade de origem.

Sob o ponto de vista de linguagem, a Java proporciona benefícios como: o reuso da linguagem, um ambiente de execução estável através da JVM, um sistema de da garbage collection que contribui no gerenciamento da correta alocação de recursos (memória). Além disso, observa-se que a isolação do estado a nível de objeto entre as Entities, o gerenciamento de chamadas em tempo de simulação através do Kernel do JIST e o compartilhamento de objetos imutáveis, promovem a segurança.

Sob o ponto de vista de sistema, o fato do kernel do JIST e as aplicações rodarem sob o mesmo espaço de processo, a não necessidade de serializações e mudança de contexto, permitem otimizações. Em [2] cita-se que além de não requerer acesso aos fontes na reescrita do bytecode, este processo pode utilizar-se das implementações de protocolos e bibliotecas existentes.

Sob o ponto de vista de hardware, o fato do JIST ser puramente Java elimina a necessidade de hardware específico, mesmo rodando em COTS clusters [2] ou arquiteturas específicas.

Arquitetura

O JIST possui uma arquitetura composta de um compilador, um reescritor de bytecode, um Kernel de simulação e uma máquina virtual que é o local onde ocorre o processo de simulação, na Java Virtual Machine (JVM) [6].

Funcionamento e Execução

No modelo de objetos do JIST, qualquer objeto deve estar estar logicamente contido dentro de uma Entity, que na prática são objetos regulares da JVM, mas que servem para logicamente encapsular objetos da aplicação e demarcar os componentes da simulação independentemente.

Os programas de simulação que rodarão sob o JIST são escritos em Java e antes de serem executados são compilados para bytecode usando o compilador Java padrão. Após serem

compiladas, estas classes (*.class) são então modificadas por um reescritor em nível de bytecode para rodarem sob o Kernel de simulação e para suportarem a semântica de tempo de simulação, por ele executada. Neste passo, chamadas para métodos públicos são assincronamente dissociados e as chamadas a eles não podem retornar qualquer valor (void). Ainda, uma Entity representa uma interface (da linguagem) baseada em eventos que possui métodos externos (públicos) e a regra é que somente métodos públicos das Entities serão reescritos dentro dos eventos. Os métodos não públicos serão considerados métodos de objetos regulares Java e não métodos de Entities, e assim seguem a semântica de invocação normal do Java. Internamente, o JIST substitui todas as referências às Entities por objetos proxy chamados Separators para que as chamadas direcionadas a elas não sejam executadas diretamente.

Assim, as semânticas de tempo de simulação são introduzidas no momento da reescrita do bytecode (classloader) e suportadas pelo Kernel em tempo de execução.

Todas as funcionalidades do JIST são expostas ao desenvolvedor através da classe jist.runtime.JistAPI, que representa a interface da aplicação exposta pelo Kernel de simulação. Assim, para que a aplicação execute dentro do interpretador JIST, ou seja, para que o Kernel de simulação seja ativado, é necessário passar a classe principal, da aplicação a ser simulada, como parâmetro da classe jist.runtime.Main, por exemplo “jist.runtime.Main HelloWorld”. Ao efetuar esta chamada inicial, um classloader é automaticamente instalado dentro da JVM, o qual dinamicamente reescreve o bytecode da aplicação em questão, transformando-a numa Entity e não num objeto regular, pois ela passará a implementar a interface JistAPI.Entity.

Cada Entity possui seu próprio tempo de simulação, o qual determina em que momento as chamadas para outras Entities serão escalonadas no kernel do JIST.

Na prática, a classe JistAPI.Entity funcionará como um marcador do programa fonte, respeitando a sintaxe Java, preservando o processo de compilação e facilitando a execução em type-safe, ou seja, prevenindo o comportamento errôneo ou indesejável dos programas. Estas marcas servem para direcionar as transformações do código, que introduzem as semânticas de tempo de simulação dentro do bytecode.

Ao executarmos um método (ou aplicação) dentro do JIST as chamadas são transformadas em eventos de simulação, as quais são escalonadas pelo Kernel em tempo de simulação e não pela semântica de execução normal da JVM.

Durante a simulação, quando um método de uma Entity é chamado, a chamada retorna imediatamente, uma nova entrada é inserida na fila de escalonamento e executada depois, em seu tempo de simulação.

A figura (2) mostra um exemplo básico de uma Entity e descreve os seguintes passos:

1)Criação da Entity hello no método main da classe;

2)No início da execução o tempo de simulação de todas as Entities é ajustado para te=0;

3)Método myEvent() é chamado, aumenta em 1 (uma) unidade o tempo de simulação local te=1 e em seguida ele efetua uma chamada recursiva sob ele mesmo. Entretanto, este método público é transformado num evento, não havendo recursão;

4)A chamada retorna imediatamente, o método println() é chamado e o método myEvent() retorna;

5)Como nenhum outro evento foi escalonado para o te=0, o tempo de simulação avança para te=1 e a chamada pendente para o método myEvent() é processada, iniciando-se o processo novamente e continuando o loop infinitamente.

hello.java

```
1  import jist.runtime.JistAPI;
2
3  class hello implements JistAPI.Entity
4  {
5    public static void main(String[] args)
6    {
7      System.out.println("simulation start");
8      hello h = new hello();
9      h.myEvent();
10   }
11
12   public void myEvent()
13   {
14     JistAPI.sleep(1);
15     myEvent();
16     System.out.println("hello world, t="
17       +JistAPI.getTime());
18   }
19 }
```

Figura 2 – Uma simples Entity no JIST [JIST, 2004]

Abaixo a tabela (1) apresenta a saída da execução do código acima. Na coluna da esquerda o sistema utiliza a funcionalidade de “tempo de simulação” e na coluna da direita não.

simulation start	simulation start
hello world, t=1	hello world, t=0
hello world, t=2	hello world, t=0
hello world, t=3	hello world, t=0
hello world, t=4	hello world, t=0
hello world, t=5	hello world, t=0
hello world, t=6	hello world, t=0
hello world, t=7	hello world, t=0
hello world, t=8	hello world, t=0
hello world, t=9	hello world, t=0
hello world, t=10	hello world, t=0
hello world, t=11	hello world, t=0
hello world, t=12	hello world, t=0
hello world, t=13	hello world, t=0

Tabela 1: Tempo de simulação e Tempo de execução

Em cima desta base, os autores criaram o SWANS [1], o qual roda sob o JIST e disponibiliza os mecanismos necessários para simular redes móveis ad hoc (MANET). Em sua arquitetura, os nós (nodes) são compostos de muitas entidades (entities) que estão acopladas, formando os diferentes elementos (camadas) da pilha do protocolo que representa a rede.

Estes elementos implementam diferentes tipos de aplicações, como: protocolos de rede, roteamento, e acesso ao meio; modelos de transmissão, recepção e ruído; modelos de propagação e enfraquecimento do sinal; e modelos de mobilidade dos nós.

Ele possui a vantagem de poder executar aplicações de rede desenvolvidas em Java

sobre a rede simulada, como: servidores web, aplicações peer-to-peer e protocolos multicast em nível de aplicação. Durante o funcionamento destas aplicações, elas operam em tempo de simulação dentro do mesmo espaço de processo do JIST, não enviando simplesmente os pacotes dos seus processos para o simulador.

No SWANS, a entidade que proporciona a mobilidade dos nós e a propagação do sinal, para os outros nós dentro do alcance de transmissão sem fio, chama-se Field, a qual é comum para todos os nós da rede. Os pacotes transmitidos atravessam as entidades da pilha de protocolos sem custo (virtual) algum, como simples referências, e a duplicação destes pacotes só feita se necessário. Nesta arquitetura, a entidade Radio de cada nó é a última da pilha de protocolos antes da entidade Field , que é comum para todos os nós.

Em relação a propagação do sinal, o SWANS implementa na entidade Field uma estrutura que agiliza a busca e localização dos nós, chamada Hierarchical Binning, favorecendo também na velocidade das simulações. Quando um nó envia um pacote para o Field, este pacote deve ser enviado a todos os nós alcançados por este pacote após considerar fading, gain e pathloss. Assim, apenas uma porção do conjunto de nós estará dentro do alcance de recepção, os demais serão afetados por interferência acima do limite, mas isso dependerá da área do Field e da potência de transmissão.

2.4 Trabalhos relacionados

A simulação de redes cognitivas, assim como o tema, também é um assunto em desenvolvimento. A dificuldade de utilizar os simuladores de rede sem fio atuais é implementar os algoritmos (que mudam de pesquisador para pesquisador) de rádio cognitivo sobre estes simuladores. O trabalho Cognitive Radio Cognitive Network Simulator [3], possui idéias similares às aqui propostas, ou seja, simulação de redes cognitivas, mas com nível de desenvolvimento bem superior, apresentando e simulando todas as camadas da pilha de protocolos. Entretanto, devido às particularidades estruturais propostas no trabalho [1], a qual baseamo-nos, também não pôde ser utilizado como referência, exceto no tocante às questões de propagação do sinal e à composição do ambiente físico virtual onde os rádios serão dispostos. Assim, viu-se a necessidade de desenvolver uma arquitetura específica para a simulação.

3 Descrição do Modelo

3.1 Visão Geral

O modelo desenvolvido e entitulado CogPhyField é responsável pela entrega de cada sinal de rádio, transmitido por um rádio cognitivo, para os outros nós com os devidos atrasos de propagação. O fator mais relevante é considerar a estrutura de sinal descrita em [1] e disponibilizar esta informação para as camadas superiores.

O modelo de camada física proposto, que visa emular o meio físico, desconsidera os processos de conversões de sinais de rádio frequência utilizados pelos Digital Signal Processors (DSP), conforme proposto em [7]. Pretende-se, então, analisar o comportamento de rádios cognitivos, quando do envio de sinal transmitido de um rádio para todos os demais do sistema, considerados problemas de sincronismo, perdas de sinal, e atenuação. Nesse ambiente, os nós são considerados fixos, assim problemas de mobilidade estão fora do contexto da pesquisa. Foram analisados, também, as variações de comportamento considerando diferentes configurações do sistema, tais como cenários onde somente um rádio transmite versus múltiplas transmissões simultâneas. Também são analisadas as características do sinal transmitido nessas condições.

Neste processo de transmissão, espera-se que os rádios cognitivos efetuem transmissões

e que estes sinais cheguem à camada física que é comum a todos eles. Portanto, independentemente se o rádio destino encontra-se dentro ou fora do alcance de transmissão de um rádio transmissor, é a CogPhyField que verificará se ele está apto a receber o sinal enviado, baseada no nível de sensibilidade do rádio destino e no limiar de intensidade do sinal enviado. Por conseguinte, os demais rádios cognitivos a ela conectados farão a recepção do sinal enviado, desde que estejam dentro deste alcance de recepção e possuam um limiar de sensibilidade compatível com a potência do sinal enviado após sofrer as devidas perdas comuns neste processo. Assim, na recepção dos sinais, antes da CogPhyField efetuar a entrega do sinal, ela verificará se o sinal transmitido está dentro do limiar de intensidade, a localização e distância dos rádios que estão dentro da topologia física fictícia, aplicará ao sinal os fatores de perda que influenciam nesta transmissão, os quais afetam profundamente a operação de uma rede e no desempenho do sistema, e analisará caso a caso qual rádio receberá ou não este sinal. Assim os rádios podem trocar dados somente na mesma banda de frequência (banda de interesse ou canal).

Em relação à sincronização de entrega de sinais, para que um sinal enviado no tempo “t”, com duração “z” chegue ao seu destino antes de um outro sinal enviado no mesmo tempo “t” mas com duração “z+1”, valeu-se da característica do simulador de eventos discretos JIST chamada simulation time, ou tempo de simulação. Desta forma, é possível que mais de um sinal seja enviado num mesmo tempo de simulação, mas deve-se garantir que eles cheguem ao seu destino também no mesmo tempo, desde que tenham a mesma duração.

Os componentes deste framework estão organizados na forma de classes independentes de software que, operando em conjunto, são responsáveis pela modelagem da propagação do sinal entre os rádios cognitivos e pela entrega dos sinais.

A CogPhyField possui estruturas de dados que gerenciam os rádios e os sinais que estão presentes num dado momento, proporcionando informações do tipo: em qual canal um rádio está operando, quais sinais devem ser transmitidos no tempo X e no tempo Y, assim como quais os rádios estão operando num dado canal.

3.2 Arquitetura

A arquitetura baseou-se no modelo descrito em [1] e conforme a figura (3) abaixo pode ser visto que a comunicação entre os rádios ocorre através da camada CogPhyField. É nesta camada que serão implementadas as funcionalidades para a solução do problema de entrada/saída de sinais/dados da camada de RÁDIO para a camada PHY e sua propagação para os demais rádios.

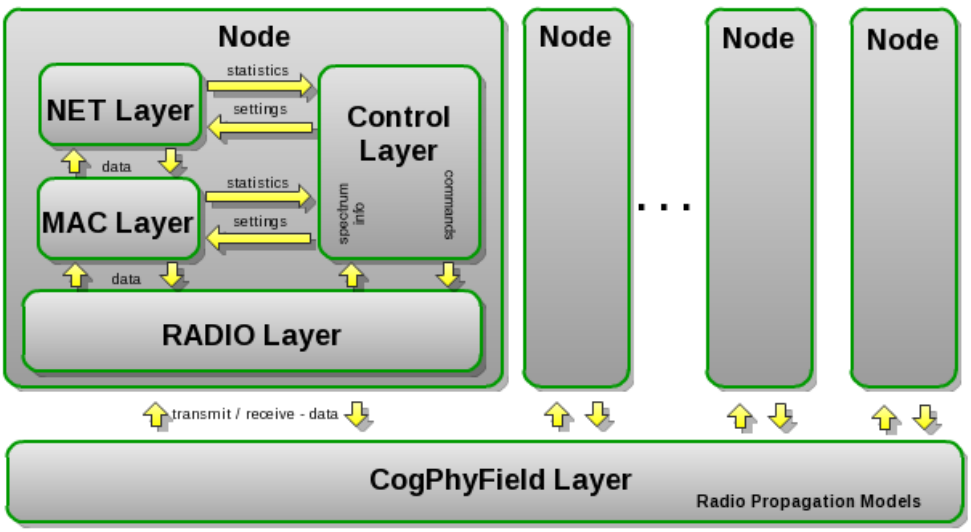

Figura 3 – Arquitetura com a CogPhyField

Também é sob esta arquitetura que são implementadas as novas características que representam o diferencial do trabalho [10] em relação às novas informações que vão compor a estrutura do sinal, que são:

- Considerar a potência na estrutura do sinal, pois é uma informação importante para a fase de decisão das heurísticas dos rádios cognitivos;

- Considerar a modulação na estrutura do sinal;

- Considerar informações que estendem o modelo tradicional , acrescentando-se as informações ao sinal:

-> NOSIGNAL: não há sinal presente na banda do espectro selecionada;

-> HIGH: um bit alto foi sensoreado;

-> LOW: um bit baixo foi sensoreado;

-> PU: o mecanismo de detecção detectou a atividade de um PU na porção do espectro sensoreado;

-> UNKNOWN: há algum sinal presente na banda do espectro selecionada mas ele não pode ser interpretado como um HIGH, LOW ou PU.

3.3 Componentes

Os componentes foram desenvolvidos para dar suporte às camadas superiores através da adição de novas informações e assim permitir que os rádios cognitivos as utilizem como dados úteis no processo de aprendizado.

São eles:

3.3.1 Bit

Representa um estado e está contido dentro da estrutura do Pulse. A possibilidade de classificar o Signal através da informação contida no Bit proporciona, através de fragmentos de frame, às camadas superiores uma forma de organizar os usuários da banda selecionada. O Bit pode apresentar as seguintes configurações:

HIGH – Bit com sinalização Alta;

LOW– Bit com sinalização Baixa;

UNKNOW– Bit com sinalização desconhecida;

PU – Bit com sinalização de usuário primário;

NOSIGNAL – Bit sem sinalização.

3.3.2 Pulse

Um Signal poderá ter um ou mais Pulses, os quais se apresentam em ordem de tempo baseados em seus atributos startMoment e stopMoment, ou seja, um Pulse com stopMoment igual a 0.005 virá antes do Pulse com startMoment 0.006, sem que haja nenhum outro no intervalo entre os dois. Em sua composição identificam-se quatro atributos, que são:

- startMoment: identifica o tempo inicial deste Pulse;

- stopMoment: identifica o tempo final deste Pulse (a soma dos tempos dos pulses identificam o tempo de duração do Signal);

- Bit: identifica um Bit, o qual representa um estado específico (no intervalo de tempo entre o startMoment e o stopMoment);

- power: identifica a potência relacionada a este Pulse (a soma da(s) potência(s) do(s) Pulse(s) identifica(m) a potência total do Signal).

3.3.3 Signal

É a representação do sinal de rádio frequência propriamente dita e que contém dados. O Signal é composto de um ou mais Pulses, a quantidade destes Pulses e a informação da modulação em que opera (também é útil para o rádio cognitivo). A classificação do Signal se dá na forma dos estados dos Bits contidos no conjunto de Pulses. Uma forma de manter a organização dos rádios da banda selecionada se dá, por exemplo, quando os rádios estão operando normalmente e ao identificar a existência de um Signal, cujo Pulse, no instante i até o f, possui um Bit com estado PU, eles têm de efetuar uma tomada de decisão, pois conforme as premissas de uma rede cognitiva e funcionamento de um rádio cognitivo, quando a operação concomitante com a de um PU não é desejada, tem-se que efetuar a troca do canal, ou quando é permitida, deve-se fazer o controle de potência para não causar interferências à transmissão licenciada. Esta identificação do Signal pode ser efetuada através de reconhecimento de padrões, onde o frame deve ser considerado válido ou não, e também caracterizando a atividade de um PU na banda do espectro selecionada.

3.3.4 Channel

O canal representa a frequência em que o rádio está operando. Ele possui algumas características adicionais que estão baseadas na oportunidade que o canal disponibiliza e que são úteis, a nível de camada rádio e superiores, como informação útil para o aprendizado do rádio cognitivo, principalmente na fase de classificação do canal. Nesta fase o foco está em classificar a sua atividade e tornar esta informação disponível às camadas superiores, para que seja possível decidir rapidamente sobre a próxima ação a ser tomada, de acordo com os princípios do gerenciamento do espectro.

Estados possíveis:

- EMPTY;
- OCCUPABLE;
- UNOCCUPABLE;
- NONANALYZED.

Ele possui um tipo de classificação que é inferida através da combinação e análise dos dados até aqui apresentados, permitindo ao rádio a identificação da existência de um fluxo de dados compatível ou incompatível com o seu padrão de transmissão, fluxo de dados de um PU, o seu próprio fluxo, timeout e fluxo indefinido caso não se encaixe nos anteriores.

- PRIMARY_USER
- INCOMPATIBLE
- COMPATIBLE
- SAME_ID
- TIMEOUT
- UNDEF

Por exemplo, dado um fluxo de um Signal, num intervalo de tempo, cujos Bits dos Pulses sejam UNKNOWN ou HIGH e LOW e cujos frames sejam classificados como INCOMPATIBLE, pode caracterizar a presença de um usuário primário e resultar numa classificação OCCUPABLE do canal.

Por fim a figura (4) apresenta a arquitetura sob uma outra perspectiva, incluindo a interação entre os rádios, a visão da camada CogPhyField, a composição dos Signals e seu gerador,

assim como a entidade (Logging) responsável pelo registro da comunicação entre os rádios, que ocorrerá através da execução da simulação.

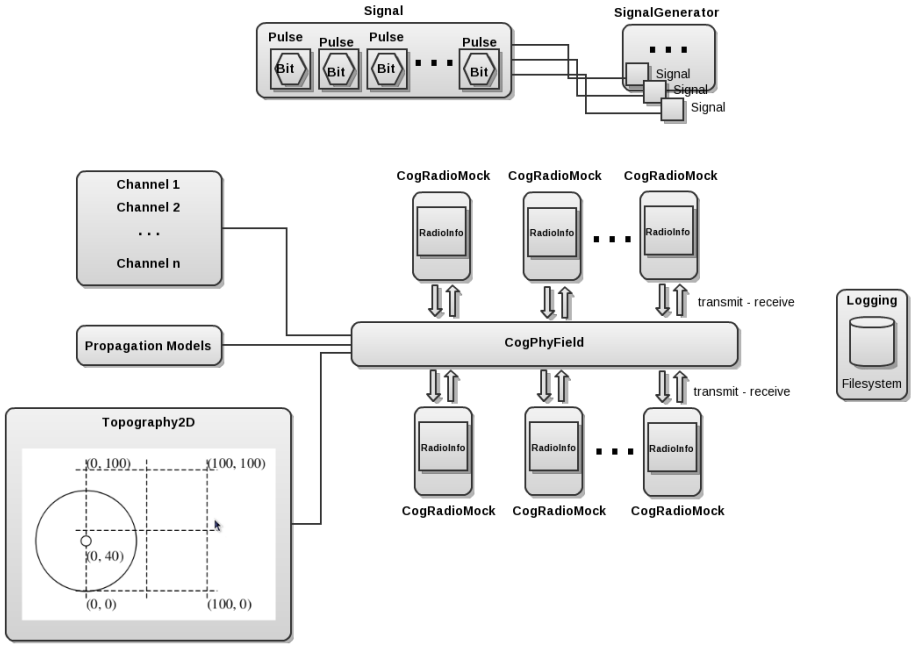

Figura 4 – Visão geral CogPhyField

3.4 Requisitos do Projeto

Os requisitos da camada física aqui proposta basearam-se principalmente em atender as necessidades de um rádio cognitivo, conforme especificado no trabalho [10], além de basear-se e tentar operar da forma mais próxima possível de um simulador de redes sem fio existente, no caso o SWANS. Assim definiu-se funcionalidades que atendessem ao modelos e permitissem a simulação:

Requisitos básicos de operação:

- Permitir a transmissão simultânea de um ou mais sinais de um rádio a todos os outros do sistema;
- Efetuar o sincronismo no envio de sinais;
- Aplicar pelo menos um modelo de propagação do sinal, considerando perda de caminho e atenuação;
- Permitir a transmissão de sinal de um usuário primário (PU).

Requisitos adicionais de simulação:

- Gerar sinais aleatoriamente com configuração distintas que comporte os seguintes tipos de estados já definidos - Bit HIGH, Bit LOW, Bit NOSIGNAL e Bit PU - e que estejam acessíveis para que os rádios possam enviá-los aos demais, se comportando como se eles (os sinais) estivessem vindo das camadas superiores para a camada de rádio e assim para a camada física.
- Disponibilização de uma perspectiva de visualização do tráfego de sinais entre os rádios na forma de log, efetuando o registro das operações de transmissão e recepção, possibilitando o rastreamento dos eventos e depuração;
- Elaborar um modelo de topologia física cujo objetivo é a simulação da disponibilização dos rádio um ambiente terreno onde possa-se inferir a distância entre eles.

3.5 Funcionalidades

A partir da visão geral e dos requisitos, definiu-se funcionalidades. Aqui as descreveremos dentro do escopo de cada classe onde foram implementadas:

Funcionalidades - Classe CogPhyField

- Registrar a entrada e saída de um rádio

Deve efetuar o controle de quem entra e de quem sai do sistema através de estruturas hash, permitindo saber que sinais trafegam num dado canal, que rádio operam num dado canal e as informações de um rádio que opera num dado canal;

- Transmitir um sinal para o meio

Deve executar o o processo de transmissão de um sinal para todos os rádios que estiverem operando num dado canal e aplicar ao sinal as perdas relacionadas ao meio;

- Sincronizar entrega de sinais

Deve garantir que os sinais que forem transmitidos no instante “t” cheguem fielmente primeiro que aqueles enviados no instante “t+1”. Felizmente esta funcionalidade já é garantida pelo próprio simulador de eventos discretos JIST.

Funcionalidades – Classe Topography2D

Os rádios foram dispostos num ponto de um campo virtual (Field) criado utilizando a técnica em grade baseada em plano cartesiano (X,Y).

- Inserir um rádio na grade

Atribui randomicamente ao rádio um ponto no plano, antes do seu registro na CogphyField;

- Retirar um rádio da grade
- Calcular a distância entre 2 rádios

Considerando que a distância mínima entre dois pontos é 1.4142, definiu-se que os rádios teriam uma distância mínima de 10, então para que a distância de um ponto a outro apresentasse no mínimo o valor 10 (metros), multiplicou-se o valor da distância mínima por 7.0711 .

Funcionalidades – Classe SignalGenerator

- Gerar e disponibilizar um vetor de sinais (Signal)
- Gerar Signal de PU

Implementado para gerar sinais para serem utilizados pelo rádio fictício no processo de transmissão. Os sinais, gerados aleatoriamente, possuem configurações distintas e comportam os tipos de estados de Bit já definidos - Bit HIGH, Bit LOW, Bit NOSIGNAL e Bit PU -, com uma quantidade de Pulses aleatória, com startMoment e stopMoment aleatórios e cuja potência representasse a soma de todas as potências dos Pulse. Além disso os sinais gerados deveriam estar acessíveis para que os rádios pudessem enviá-los aos demais, se comportando como se eles (os sinais) estivessem vindo das camadas superiores para a camada de rádio e assim para a camada física.

Funcionalidades – Classe PathLoss

- Calcular FreeSpace / TwoRay

Funcionalidades – Classe Fading

- Calcular None / Rician

Funcionalidades – Classe CogRadioMock

Implementa os métodos necessários para o rádio efetuar estritamente, por questão de escopo, a transmissão e recepção de um sinal. Além disso, foi disponiblizado uma perspectiva de visualização do tráfego de sinais entre os rádios na forma de log, efetuando o registro das operações de transmissão e recepção, possibilitando o rastreamento dos eventos e depuração.

- Transmitir um sinal
- Receber um sinal
- Logging

Exemplo

Registro no arquivo de log na transmissão de um sinal, no formato padrão e em seguida no formato XML:

- Formato padrão:

May 01, 2010 11:19:24 PM br.com.datacoffee.cognet.tools.Logging log
INFO: CogRadioMock.TRANSMIT ->, RadioID: 1, SIGNAL: [(1.0,2.1041,HIGH,3.3209), (2.1042,4.9479,LOW,8.5533),(4.948,5.9507,LOW,3.0159),(5.9508,5.9794,HIGH,0.086), (5.9795,5.9915,NOSIGNAL,0.0),(5.9916,5.9977,HIGH,0.0183),(5.9978,5.9997,HIGH,0.0057), (5.9998,5.9999,NOSIGNAL,0.0)], TRANSMIT_POWER: 15.0, TIME: 50

- Formato XML:

```
<record>
  <date>2010-05-01T23:19:24</date>
  <millis>1274062764590</millis>
  <sequence>2</sequence>
  <logger>CogNetLogging</logger>
  <level>INFO</level>
  <class>br.com.datacoffee.cognet.tools.Logging</class>
  <method>log</method>
  <thread>10</thread>
  <message>CogRadioMock.TRANSMIT -&amp;gt;, RadioID: 1, SIGNAL: [(1.0,2.1041,HIGH,3.3209),
(2.1042,4.9479,LOW,8.5533),(4.948,5.9507,LOW,3.0159),(5.9508,5.9794,HIGH,0.086),
(5.9795,5.9915,NOSIGNAL,0.0),(5.9916,5.9977,HIGH,0.0183),(5.9978,5.9997,HIGH,0.0057),
(5.9998,5.9999,NOSIGNAL,0.0)], TRANSMIT_POWER: 15.0, TIME: 50</message>
</record>
```

- Informações disponíveis:

Método sendo executado: CogRadioMock.TRANSMIT;
Rádio que executou o método: RadioID: 1;
Sinal transmitido: SIGNAL: […];
Pulses deste Signal: [(P1),(P2),(...),(Pn)];
Informações do Pulse:(startMoment,stopMoment,Bit,potência do Bit);
Potência de transmissão do Signal: TRANSMIT_POWER: 15.0;
Tempo de simulação (Simulation Time) de execução do método: TIME: 50.

4 Simulação

A simulação foi executada sob o simulador de eventos discretos JIST, respeitando suas premissas e os requisitos propostos e utilizando as estruturas definidas na arquitetura e nos

componentes.

O Objetivo específico da simulação é a transmissão de sinais, a recepção de sinais, a sincronização da transmissão e da recepção em conformidade com o tempo de simulação, assim como a verificação da aplicação das perdas relacionadas aos modelos de propagação.

Na dinâmica de execução, primeiramente instancia-se o modelo de grade, define-se o modelo de fading e pathloss, gera-se o conjunto de sinais, instancia-se a camada física CogPhyField , instancia-se o conjunto de nós, registra-se os nós na camada física e os rádios estão prontos para transmitir e receber os sinais gerados.

Foram gerados 5 tipos de sinais, cada um com uma combinação de Bit específica, mas que abrangesse todos os tipos de Bits possíveis pelo modelo. Definiu-se na dinâmica de execução da simulação que cada rádio enviaria cinco sinais aos demais rádios num tempo distinto e aleatório de 1 a 5.

Através deste conjunto de passos tem-se pronta uma camada física com um conjunto de rádios cognitivos com poder de transmissão e recepção.

Para avaliar se os rádios estavam transmitindo, recebendo e se os modelos de perda estavam também funcionando, definiu-se uma topologia física básica onde variou-se o comprimento da grade e o número de nós, ficando assim:

a) comprimento da grade 100 , número de nós 5;

b) comprimento da grade 500 , número de nós 50;

c) comprimento da grade 1000 , número de nós 100;

Por questão de restrição de escopo não variamos as configurações de Fading e o PathLoss, ficando assim:

- Fading: None
- PathLoss: Free Space Pathloss

5 Análise dos resultados

As métricas de interesse nestas simulações giram em torno da quantidade de sinais transmitidos, quantidade de sinais recebidos, duração da transmissão do sinal, em que tempo os sinais foram recebidos e se a potência de transmissão sofreu perdas. Assim, definiu-se questões a serem respondidas:

a) Quantos sinais foram transmitidos ?

b) Quantos sinais foram recebidos ?

c) Qual a frequência de recebimento destes sinais e em que tempo ?

d) Dado um sinal transmitido no tempo 0, qual sua duração, quantas vezes ele foi recebido e em que tempo ?

e) Dado este mesmo sinal, cuja potência de transmissão é padrão, houveram perdas nesta potência ? Quais os valores para os casos recebidos do item anterior ?

Após executadas as 3 simulações acima, também analisamos o tempo de execução e o tamanho do arquivo de log gerado, obtendo os seguintes valores:

1) comprimento da grade 100 , número de nós 5;

- total time: 3 seconds
- totalling 692.1 KB

2) comprimento da grade 500 , número de nós 50;

- total time: 1 minute 27 seconds
- totalling 386.1 MB

3) comprimento da grade 1000 , número de nós 100;
- total time: 11 minutes 46 seconds
- totalling 3.4 GB

Então para efeito de análise, escolheu-se a simulação número 1 - comprimento da grade 100 , número de nós 5 – e utilizando o ambiente shell do sistema operacional Linux para varrer os arquivos de log desta simulação, obteve-se as seguintes respostas:

a) 25 sinais foram transmitidos, conforme o esperado.
$ cat logging.log | grep CogRadioMock.TRANSMIT | wc -l
Resposta: 25

b) 100 sinais foram recebidos, conforme o esperado.
$ cat logging.log | grep CogRadioMock.RECEIVE | wc -l
Resposta: 100

c) como a duração considerada é um número randômico de 1 a 5 (ver classe MainSim.java), desconsidera-se o tempo 0, e conforme abaixo, tudo dentro do esperado, ou seja 100 Signals recebidos.
$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 1" | wc -l
Resposta: 40
$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 2" | wc -l
Resposta: 0
$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 3" | wc -l
Resposta: 20
$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 4" | wc -l
Resposta: 20
$ cat logging.log | grep CogRadioMock.RECEIVE | grep "TIME: 5" | wc -l
Resposta: 20

d) Considerando o Signal abaixo, sua duração é 3, foi recebido 20 vezes, mas no tempo 3 somente 4 vezes, conforme o esperado.
INFO: CogRadioMock.TRANSMIT, RadioID: 3, SIGNAL: [(1.0,2.482,LOW,2.2233),
(2.4821,4.9964,LOW,3.772),(4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598),
(9.3969,10.5724,LOW,1.7635),(10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456),
(10.9611,10.9788,LOW,0.0266),(10.9789,10.9913,LOW,0.0186),(10.9914,10.9928,LOW,0.0021),
(10.9929,10.9964,LOW,0.0053),(10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4),
(10.9998,10.9999,LOW,2.0E-4)], TRANSMIT_POWER: 15.0, TIME: 0 DURACAO: 3

$cat logging.log | grep "(1.0,2.482,LOW,2.2233),(2.4821,4.9964,LOW,3.772),
(4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598),(9.3969,10.5724,LOW,1.7635),
(10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456),(10.9611,10.9788,LOW,0.0266),
(10.9789,10.9913,LOW,0.0186),(10.9914,10.9928,LOW,0.0021),(10.9929,10.9964,LOW,0.0053),
(10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4),(10.9998,10.9999,LOW,2.0E-4)" |
grep RECEIVE | grep "TIME: 3" | wc -l
Resposta: 4

e) Abaixo as potências recebidas em destaque para cada rádio, tudo conforme o esperado.
$ cat logging.log | grep "(1.0,2.482,LOW,2.2233),(2.4821,4.9964,LOW,3.772),
(4.9965,5.2907,LOW,0.4414),(5.2908,9.3968,LOW,6.1598),(9.3969,10.5724,LOW,1.7635),

(10.5725,10.7972,LOW,0.3371),(10.7973,10.961,LOW,0.2456),(10.9611,10.9788,LOW,0.0266),
(10.9789,10.9913,LOW,0.0186),(10.9914,10.9928,LOW,0.0021),(10.9929,10.9964,LOW,0.0053),
(10.9965,10.9991,LOW,0.0039),(10.9992,10.9997,LOW,8.0E-4),(10.9998,10.9999,LOW,2.0E-4)" |
grep RECEIVE | grep "TIME: 3" | wc -l
Resposta: 4 registros com este padrão de informação

6 Conclusões e Trabalhos Futuros

O desenvolvimento de simulações como ponto de partida para a pesquisa científica é realmente necessário. A cada dia novas idéias surgem e com elas podem surgir novas tecnologias cujo foco é apresentar algo novo ou é a melhoria de algo existente. As redes cognitivas se enquadram neste último, e pelos benefícios que prometem, parece ser este um caminho sem volta, pelo menos até o momento.

Por tratar-se de um assunto ainda em desenvolvimento, e não existirem comercialmente dispositivos para redes sem fio com comportamento cognitivo, ainda há tempo para surgirem novas idéias e novas formas de conduzir a dinâmica existente no processo de coexistência entre usuários primários e secundários dentro do espectro eletromagnético, dando liberdade também para que os pesquisadores invistam seu tempo no desenvolvimento de novos protocolos através das simulações.

Assim, este trabalho objetivou a construção de um modelo com idéias novas e a implementação e simulação deste modelo baseado em outros já existentes, do ponto de vista do funcionamento.

Como trabalhos futuros a serem implementados na CogphyField, visando atender os requisitos do trabalho [1], citamos a implementação do modelo de recepção de sinais na camada física e não na camada de rádio, pois se formos considerar que, numa situação de recepção de dois ou mais sinais, ao invés de considerar somente aquele de maior potência (como é feito no SWANS) quisermos degradar este sinal de maior potência efetuando a interferência nele através da mescla de Bits cujo time seja o mesmo, daí acreditamos que esta funcionalidade talvez tenha que ser executada na camada física.

Outro fator relevante é aprimorar a acurácia dos modelos de propagação do sinal utilizando, por exemplo, um modelo mais robusto como o Simulation of Indoor Radio Channel Impulse Response Models with Impulse Noise – (SIRCIM).

A questão da mobilidade também é importante para dar uma noção mais realista ao modelo.

Com base na Análise dos Resultados, identifica-se também a necessidade de um modelo de logging mais adequado, que não interfira no tempo de execução da simulação e nem na quantidade de espaço necessária para armazenamento dos mesmos.

7 Referências

[1] R. Barr, "SWANS-Scalable Wireless Ad hoc Network Simulator Users Guide," Retrieved March, 2006, pp. 1-15.

[2] R. Barr, "JiST-Java in Simulation Time Users Guide," Cornell University, March, vol. 19, 2004.

[3] Cognitive Radio Cognitive Network Simulator. Disponível em <http://www.ece.mtu.edu/~ljialian/> . Acesso em: 15/03/2010;

[4] CORDEIRO, Carlos; CHALLAPALI, Kiran; BIRRU, Dagnachew. IEEE 802.22: An

Introduction to the First Wireless Standard based on Cognitive Radios. Journal Of Communications, Briarcliff Manor, p. 38-47. abr. 2006.

[5] G. S. Fishman. Principles of Discrete Event Simulation. John Wiley & Sons, Inc., New York, NY, USA, 1978.

[6] The Source for Java Developers. Disponível em: <http://java.sun.com/> . Acesso em : 01/11/2009.

[7] G. Judd and P. Steenkiste, "A software architecture for physical layer wireless network emulation," Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization - WiNTECH '06, 2006, p. 2.

[8] F. Kargl and E. Schoch, "Simulation of manets: a qualitative comparison between jist/swans and ns-2," Proceedings of the 1st international workshop on System evaluation for mobile platforms, ACM, 2007, p. 46.

[9] R. W. Thomas, L. A. DaSilva, and A. B. Mackenzie, “Cognitive Networks,” Proc. IEEE DySPAN 2005, Nov. 2005, pp. 352–60.

[10] R. De S. Mendes, E. R. Garcia e C. B. Westphall, “Um Framework para Operação na Camada de Rádio em Redes Cognitivas”, The Sixth International Conference on Networking and Services - ICNS 2010 - March 7-13, 2010 - Cancun, Mexico.

Fontes

/***
Este software é uma adaptação baseada nos códigos implementados no simulador Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de 2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar, copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso acompanhe todas as cópias do software. Cópias do software modificadas podem ser distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado; c) a data de todas as mudanças, claramente identificadas no software. "
***/

//
// JIST (Java In Simulation Time) Project
// Timestamp: <Constants.java Wed 2005/02/09 18:18:27 barr rimbase.rimonbarr.com>
//

// Copyright (C) 2005 by Cornell University
// All rights reserved.
// Refer to LICENSE for terms and conditions of use.

package br.com.datacoffee.cognet;

import java.util.Random;

```
/**
 * SWANS constants.
 *
 * @author Rimon Barr &lt;barr+jist@cs.cornell.edu&rt;
 * @version $Id: Constants.java,v 1.62 2005-02-09 23:32:20 barr Exp $
 * @since SWANS1.0
 */
public final class Constants
{

  //////////////////////////////////////////////////
  // Randomness
  //

  /** Global random number generator. */
  public static Random random = new Random(0);

  //////////////////////////////////////////////////
  // Time
  //

  /** zero delay. */
  public static final long PROCESS_IMMEDIATELY = 0;
  /** smallest possible delay. */
  public static final int EPSILON_DELAY = 1;
  /** one nano-second in simulation time units. */
  public static final long NANO_SECOND  = 1;
  /** one micro-second in simulation time units. */
  public static final long MICRO_SECOND = 1000 * NANO_SECOND;
  /** one milli-second in simulation time units. */
  public static final long MILLI_SECOND = 1000 * MICRO_SECOND;
  /** one second in simulation time units. */
  public static final long SECOND       = 1000 * MILLI_SECOND;
  /** one minute in simulation time units. */
```

```
public static final long MINUTE       = 60 * SECOND;
/** one hour in simulation time units. */
public static final long HOUR         = 60 * MINUTE;
/** one day in simulation time units. */
public static final long DAY          = 24 * HOUR;

//////////////////////////////////////////////////
// Nature
//

/** Boltzmann's constant (units: Joules/Kelvin). */
public static final double BOLTZMANN = 1.3807e-23;
/** Speed of light in a vacuum (units: meter/second). */
public static final double SPEED_OF_LIGHT = 2.9979e8;
/** Pre-computed natural logarithm of 10. */
public static final double log10 = Math.log(10);

//////////////////////////////////////////////////
// Field-related constants
//

// constants

/** Default field boundary (units: sim distance, usually meters). */
public static final float FIELD_BOUND_X = (float)200.0, FIELD_BOUND_Y = (float)200.0;

/** node placement choice constant. */
public static final int PLACEMENT_INVALID = -1;
/** node placement choice constant. */
public static final int PLACEMENT_RANDOM  = 1;
/** node placement choice constant. */
public static final int PLACEMENT_GRID    = 2;
/** node placement choice constant. */
public static final int PLACEMENT_MAX     = 2;
```

```
/** node placement choice constant. */
public static final int PLACEMENT_DEFAULT = PLACEMENT_RANDOM;

/** node mobility choice constant. */
public static final int MOBILITY_INVALID  = -1;
/** node mobility choice constant. */
public static final int MOBILITY_STATIC   = 1;
/** node mobility choice constant. */
public static final int MOBILITY_WAYPOINT = 2;
/** node mobility choice constant. */
public static final int MOBILITY_TELEPORT = 3;
/** node mobility choice constant. */
public static final int MOBILITY_WALK     = 4;
/** node mobility choice constant. */
public static final int MOBILITY_DEFAULT  = MOBILITY_STATIC;

/** spatial data structure choice constant. */
public static final int SPATIAL_INVALID = -1;
/** spatial data structure choice constant. */
public static final int SPATIAL_LINEAR  = 0;
/** spatial data structure choice constant. */
public static final int SPATIAL_GRID    = 1;
/** spatial data structure choice constant. */
public static final int SPATIAL_HIER    = 2;
/** spatial data structure choice constant. */
public static final int SPATIAL_WRAP    = 16;

////////////////////////////////////////////////
// packet constants
//

/** packet with zero wire size. */
public static final int ZERO_WIRE_SIZE = Integer.MIN_VALUE;
```

```
//////////////////////////////////////////////
// Radio-related constants
//

// radio modes

/** Radio mode: sleeping. */
public static final byte RADIO_MODE_SLEEP       = -1;
/** Radio mode: idle, no signals. */
public static final byte RADIO_MODE_IDLE        = 0;
/** Radio mode: some signals above sensitivity. */
public static final byte RADIO_MODE_SENSING     = 1;
/** Radio mode: signal locked and receiving packet. */
public static final byte RADIO_MODE_RECEIVING   = 2;
/** Radio mode: transmitting packet. */
public static final byte RADIO_MODE_TRANSMITTING = 3;

// timing constants

/** RX-TX switching delay. */
public static final long RADIO_TURNAROUND_TIME = 5 * MICRO_SECOND;
/** physical layer delay. */
public static final long RADIO_PHY_DELAY = RADIO_TURNAROUND_TIME;
/** Constant used to specify the default "delay to the wire". */
public static final int RADIO_NOUSER_DELAY = -1;

// defaults

/** Default radio frequency (units: Hz). */
public static final double FREQUENCY_DEFAULT = 2.4e9; // 2.4 GHz
/** Default radio bandwidth (units: bits/second). */
public static final int BANDWIDTH_DEFAULT = (int)1e6; // 1Mb/s
/** Default transmission strength (units: dBm). */
public static final double TRANSMIT_DEFAULT = 15.0;
```

```
/** Default antenna gain (units: dB). */
public static final double GAIN_DEFAULT = 0.0;
/** Default radio reception sensitivity (units: dBm). */
public static final double SENSITIVITY_DEFAULT = -91.0;
/** Default radio reception threshold (units: dBm). */
public static final double THRESHOLD_DEFAULT = -81.0;
/** Default temperature (units: degrees Kelvin). */
public static final double TEMPERATURE_DEFAULT = 290.0;
/** Default temperature noise factor. */
public static final double TEMPERATURE_FACTOR_DEFAULT = 10.0;
/** Default ambient noise (units: mW). */
public static final double AMBIENT_NOISE_DEFAULT = 0.0;
/** Default minimum propagated signal (units: dBm). */
//public static final double PROPAGATION_LIMIT_DEFAULT = -111.0;
public static final double PROPAGATION_LIMIT_DEFAULT = SENSITIVITY_DEFAULT;
/** Default radio height (units: sim distance units, usually meters). */
public static final double HEIGHT_DEFAULT = 1.5;
/** Default threshold signal-to-noise ratio. */
public static final double SNR_THRESHOLD_DEFAULT = 10.0;


//////////////////////////////////////////////////
// Mac-related constants
//

// defaults

/** Default mac promiscuous mode. */
public static final boolean MAC_PROMISCUOUS_DEFAULT = false;
/** link layer delay. */
public static final long LINK_DELAY = MICRO_SECOND;

//////////////////////////////////////////////////
// Network-related constants
```

```
//

/** network layer loss model choice constant. */
public static final int NET_LOSS_INVALID       = -1;
/** network layer loss model choice constant. */
public static final int NET_LOSS_NONE          = 0;
/** network layer loss model choice constant. */
public static final int NET_LOSS_UNIFORM       = 1;
/** network layer loss model choice constant. */
public static final int NET_LOSS_DEFAULT       = NET_LOSS_NONE;

/** network packet priority level. */
public static final byte NET_PRIORITY_CONTROL  = 0;
/** network packet priority level. */
public static final byte NET_PRIORITY_REALTIME = 1;
/** network packet priority level. */
public static final byte NET_PRIORITY_NORMAL   = 2;
/** network packet priority level. */
public static final byte NET_PRIORITY_NUM      = 3;
/** network packet priority level. */
public static final byte NET_PRIORITY_INVALID  = -1;


/** network interface constant. */
public static final int NET_INTERFACE_INVALID = -1;
/** network interface constant. */
public static final int NET_INTERFACE_LOOPBACK = 0;
/** network interface constant. */
public static final int NET_INTERFACE_DEFAULT = 1;

/** network layer delay. */
public static final long NET_DELAY = MICRO_SECOND;
/** default time-to-live. */
public static final byte TTL_DEFAULT = 64;
```

```
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_INVALID       = -1;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_TCP           = 6;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_UDP           = 17;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_OSPF          = 87;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_BELLMANFORD   = 520;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_FISHEYE       = 530;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_AODV          = 123;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_DSR           = 135;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_ODMRP         = 145;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_LAR1          = 110;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_ZRP           = 133;
/** zrp-subprotocol number. */
public static final short NET_PROTOCOL_ZRP_NDP_DEFAULT  = 1;
/** zrp-subprotocol number. */
public static final short NET_PROTOCOL_ZRP_IARP_DEFAULT = 2;
/** zrp-subprotocol number. */
public static final short NET_PROTOCOL_ZRP_BRP_DEFAULT  = 3;
/** zrp-subprotocol number. */
public static final short NET_PROTOCOL_ZRP_IERP_DEFAULT = 4;
/** zrp-subprotocol number. */
public static final short NET_PROTOCOL_ZRP_IARP_ZDP     = 5;
/** zrp-subprotocol number. */
```

```
public static final short NET_PROTOCOL_ZRP_BRP_FLOOD    = 6;

/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_NO_NEXT_HEADER = 59;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_HEARTBEAT      = 500;
/** network level (IP) protocol number. */
public static final short NET_PROTOCOL_MAX            = 999;

////////////////////////////////////////////////
// Routing-related constants
//

////////////////////////////////////////////////
// Transport-related constants
//

/** transport layer delay. */
public static final long TRANS_DELAY = MICRO_SECOND;
/** socket delay. */
public static final long TRANS_PROCESSING_DELAY = MICRO_SECOND;

/** transport level (tcp/udp) protocol number. */
public static final short TRANS_PROTOCOL_INVALID = -1;
/** transport level (tcp/udp) protocol number. */
public static final short TCP_PROTOCOL_ECHO      =  7;
/** transport level (tcp/udp) protocol number. */
public static final short TCP_PROTOCOL_FTP       = 21;
/** transport level (tcp/udp) protocol number. */
public static final short TCP_PROTOCOL_TELNET    = 23;
/** transport level (tcp/udp) protocol number. */
public static final short TCP_PROTOCOL_SMTP      = 25;
/** transport level (tcp/udp) protocol number. */
public static final short TCP_PROTOCOL_TIME      = 37;
```

```
/** transport level (tcp/udp) protocol number. */
public static final short TCP_PROTOCOL_HTTP      = 80;

/** TCP States. */
public final class TCPSTATES
{
  /**
   * TCP state: LISTEN - represents waiting for a connection request from
   * any remote TCP and port.
   */
  public static final int LISTEN = 800;
  /**
   * TCP state: SYN-SENT - represents waiting for a matching connection
   * request after having sent a connection request.
   */
  public static final int SYN_SENT = 801;
  /**
   * TCP state: SYN-RECEIVED - represents waiting for a confirming
   * connection request acknowledgment after having both received and sent a
   * connection request.
   */
  public static final int SYN_RECEIVED = 802;
  /**
   * TCP state: ESTABLISHED - represents an open connection, data received
   * can be delivered to the user.  The normal state for the data transfer
   * phase of the connection.
   */
  public static final int ESTABLISHED = 803;
  /**
   * TCP state: FIN-WAIT-1 - represents waiting for a connection
   * termination request from the remote TCP, or an acknowledgment of the
   * connection termination request previously sent.
   */
  public static final int FIN_WAIT_1 = 804;
```

```
  /**
   * TCP state: FIN-WAIT-2 - represents waiting for a connection
   * termination request from the remote TCP.
   */
  public static final int FIN_WAIT_2 = 805;
  /**
   * TCP state: CLOSE-WAIT - represents waiting for a connection termination
   * request from the local user.
   */
  public static final int CLOSE_WAIT = 806;
  /**
   * TCP state: CLOSING - represents waiting for a connection termination
   * request acknowledgment from the remote TCP.
   */
  public static final int CLOSING = 807;
  /**
   * TCP state: LAST-ACK - represents waiting for an acknowledgment of the
   * connection termination request previously sent to the remote TCP (which
   * includes an acknowledgment of its connection termination request).
   */
  public static final int LAST_ACK = 808;
  /**
   * TCP state: TIME-WAIT - represents waiting for enough time to pass to be
   * sure the remote TCP received the acknowledgment of its connection
   * termination request.
   */
  public static final int TIME_WAIT = 809;
  /**
   * TCP state: CLOSED - represents no connection state at all.
   */
  public static final int CLOSED = 810;

} // class: TCPSTATES
```

```
} // class: Constants



//package br.com.datacoffee.cognet;
//
//public final class Constants
//{
//  // Time
//  //
//
//  /** zero delay. */
//  public static final long PROCESS_IMMEDIATELY = 0;
//  /** smallest possible delay. */
//  public static final int EPSILON_DELAY = 1;
//  /** one nano-second in simulation time units. */
//  public static final long NANO_SECOND  = 1;
//  /** one micro-second in simulation time units. */
//  public static final long MICRO_SECOND = 1000 * NANO_SECOND;
//  /** one milli-second in simulation time units. */
//  public static final long MILLI_SECOND = 1000 * MICRO_SECOND;
//  /** one second in simulation time units. */
//  public static final long SECOND       = 1000 * MILLI_SECOND;
//  /** one minute in simulation time units. */
//  public static final long MINUTE       = 60 * SECOND;
//  /** one hour in simulation time units. */
//  public static final long HOUR         = 60 * MINUTE;
//  /** one day in simulation time units. */
//  public static final long DAY          = 24 * HOUR;
//
//  ////////////////////////////////////////////////
//  // Nature
//  //
//
```

```
//  /** Boltzmann's constant (units: Joules/Kelvin). */
//  public static final double BOLTZMANN = 1.3807e-23;
//  /** Speed of light in a vacuum (units: meter/second). */
//  public static final double SPEED_OF_LIGHT = 2.9979e8;
//  /** Pre-computed natural logarithm of 10. */
//  public static final double log10 = Math.log(10);
//
//
//
//  /* defaults
//        public static RadioInfo createInfo(int iD, double frequency, int bandwidth,
//           double transmit, double gain, double sensitivity_mW, double threshold_mW,
//           double temperature, double thermalFactor, double ambientNoise_mW)
//    */
//  /** Default radio frequency (units: Hz). */
//  //public static final double FREQUENCY_DEFAULT = 2.4e9; // 2.4 GHz
//  public static final int FREQUENCY_DEFAULT = 2442; // Canal 13
//  /** Default radio bandwidth (units: bits/second). */
//  public static final int BANDWIDTH_DEFAULT = (int)1e6; // 1Mb/s
//  /** Default transmission strength (units: dBm). */
//  public static final double TRANSMIT_DEFAULT = 15.0;
//  /** Default antenna gain (units: dB). */
//  public static final double GAIN_DEFAULT = 0.0;
//  /** Default radio reception sensitivity (units: dBm). */
//  public static final double SENSITIVITY_DEFAULT = -91.0;
//  /** Default radio reception threshold (units: dBm). */
//  public static final double THRESHOLD_DEFAULT = -81.0;
//  /** Default temperature (units: degrees Kelvin). */
//  public static final double TEMPERATURE_DEFAULT = 290.0;
//  /** Default temperature noise factor. */
//  public static final double TEMPERATURE_FACTOR_DEFAULT = 10.0;
//  /** Default ambient noise (units: mW). */
//  public static final double AMBIENT_NOISE_DEFAULT = 0.0;
//  /** Default minimum propagated signal (units: dBm). */
```

```
//  //public static final double PROPAGATION_LIMIT_DEFAULT = -111.0;
//  public static final double PROPAGATION_LIMIT_DEFAULT = SENSITIVITY_DEFAULT;
//  /** Default radio height (units: sim distance units, usually meters). */
//  public static final double HEIGHT_DEFAULT = 1.5;
//  /** Default threshold signal-to-noise ratio. */
//  public static final double SNR_THRESHOLD_DEFAULT = 10.0;
//
//  /* Default modulation*/
//
//}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.mac;

import br.com.datacoffee.cognet.phy.Signal;

/**
 *
 * @author everton
 */
public class CogMac implements CogMacInterface{

    public void receive(Signal signal) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public void send(Signal signal, MacAddress nextHop) {
        throw new UnsupportedOperationException("Not supported yet.");
```

```
    }

}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.mac;

import br.com.datacoffee.cognet.phy.Signal;
import jist.runtime.JistAPI;

/**
 *
 * @author everton
 */
public interface CogMacInterface extends JistAPI.Proxiable {

    void receive(Signal signal);

      void send(Signal signal, MacAddress nextHop);

}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.mac;

/**
 *
 * @author everton
 */
class MacAddress {

}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/

package br.com.datacoffee.cognet.phy;

import jist.runtime.JistAPI.Timeless;

/**
 *
 * @author everton
 */
public class Bit implements Timeless {

    public static final int NOSIGNAL = 0;
    public static final int HIGH = 1;
    public static final int LOW = 2;
    public static final int UNKNOW = 3;
    public static final int PU = 4;

/*
```

```
 NOSIGNAL – não há sinal presente na banda do espectro selecionada;
 HIGH – um bit alto foi sensoreado;
 LOW – um bit baixo foi sensoreado;
 PU – o mecanismo de detecção detectou a atividade de um PU na porção
    do espectro sensoreado;
 UNKNOWN – há algum sinal presente na banda do espectro selecionada
    mas ele não pode ser interpretado como um HIGH, LOW ou PU.
*/

    public static String getName(int bit) {
        switch(bit){
            case NOSIGNAL: return "NOSIGNAL";
            case HIGH: return "HIGH";
            case LOW: return "LOW";
            case UNKNOW: return "UNKNOW";
            case PU: return "PU";
            default: throw new IllegalArgumentException("Invalid bit value:" + bit );
        }
    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.phy;

import jist.runtime.JistAPI.Timeless;

public class Channel implements Timeless {
    private ChannelInfo info;
    private int number;

    public Channel(ChannelInfo channelInfo, int number) {
        this.info = channelInfo;
        this.number = number;
    }

    /**
     * @return the info
     */
```

```
public ChannelInfo getInfo() {
    return info;
}

/**
 * @param info the info to set
 */
public void setInfo(ChannelInfo info) {
    this.info = info;
}

/**
 * @return the number
 */
public int getNumber() {
    return number;
}

/**
 * @param number the number to set
 */
public void setNumber(int number) {
    this.number = number;
}

public static final class ChannelState {

    public static final int EMPTY = 0;
    public static final int OCCUPABLE = 1;
    public static final int UNOCCUPABLE = 2;
    public static final int NONANALYZED = 3;

    public String getName(int modulation) {
        switch (modulation) {
```

```
            case EMPTY:
                return "EMPTY";
            case OCCUPABLE:
                return "OCCUPABLE";
            case UNOCCUPABLE:
                return "UNOCCUPABLE";
            case NONANALYZED:
                return "NONANALYZED";
            default:
                throw new IllegalArgumentException("Invalid modulation value:" +
modulation);
        }
    }
}

public static final class Type {

    public static final int PRIMARY_USER = 0;
    public static final int INCOMPATIBLE = 1;
    public static final int COMPATIBLE = 2;
    public static final int SAME_ID = 3;
    public static final int TIMEOUT = 4;
    public static final int UNDEF = 5;

    public String getName(int type) {
        switch (type) {
            case PRIMARY_USER:
                return "PRIMARY_USER";
            case INCOMPATIBLE:
                return "INCOMPATIBLE";
            case COMPATIBLE:
                return "COMPATIBLE";
            case SAME_ID:
                return "SAME_ID";
```

```
            case TIMEOUT:
                return "TIMEOUT";
            case UNDEF:
                return "UNDEF";
            default:
                throw new IllegalArgumentException("Invalid type value:" + type);
        }
    }
}

public static class ChannelInfo implements Timeless {

    protected int type;
    protected int state;

    public int getType() {
        return this.type;
    }

    public void setType(int type) {
        this.type = type;
    }

    public int getChannelState() {
        return this.state;
    }

    public void setChannelState(int state) {
        this.state = state;
    }

    public ChannelInfo createInfo(int type, int state) {
        ChannelInfo channelInfo = new ChannelInfo();
        channelInfo.setChannelState(state);
```

```
            channelInfo.setType(type);
            return channelInfo;
        }
    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.phy;

import br.com.datacoffee.cognet.phy.Topography2D.Ponto;
import br.com.datacoffee.cognet.radio.CogRadioInterface;
import br.com.datacoffee.cognet.radio.RadioInfo;
import java.util.Hashtable;
import java.util.Vector;
import jist.runtime.JistAPI;

public class CogPhyField implements CogPhyInterface {
    /*
     * Estrutura de dados que controla os rádios que habitam a simulação
     */
    // aXb, where a=Channel.number, b=existing Signals in a Channel at any given time

    protected Hashtable<Integer, Vector<Signal>> channelBufferSignals;
```

```
    //protected Map<Integer, Vector<Signal>> channelBufferSignals;
    // aXb, where a=Channel.number, b=Vector of existing CogRadios on a Channel
    protected Hashtable<Integer, Vector<RadioData>> channelRadiosMap;
    // aXb, where a=CogRadio ID, b=RadioData that tells you which channel it is
    protected Hashtable<Integer, RadioData> radioIdRadioDataMap;
    // Entidade de referência própria
    protected CogPhyInterface self;
    // Estrutura espacial, que coordena o grid onde os rádio serão postos
    protected Topography2D loc;
    /**
     * Propagation limit (in dBm). This is the signal strength threshold.
     */
    protected double propagationLimit;
    protected PathLoss pathloss;
    protected Fading fading;

    public CogPhyField(Topography2D loc, Fading fading, PathLoss pathloss, double
propagationLimit) {
        this.channelBufferSignals = new Hashtable();
        this.channelRadiosMap = new Hashtable();
        this.radioIdRadioDataMap = new Hashtable();

        this.loc = loc;

        this.fading = fading;

        this.pathloss = pathloss;

        this.propagationLimit = propagationLimit;

        this.self = (CogPhyInterface) JistAPI.proxy(this, CogPhyInterface.class);
    }

    public CogPhyInterface getProxy() {
```

```
        return this.self;
    }

    public void enter(CogRadioInterface entity, RadioInfo info) {
        //public void enter(final Channel channel, CogRadioInterface entity, RadioInfo info) {
        if (!JistAPI.isEntity(entity)) {
            throw new IllegalArgumentException("entity expected");
        }

        RadioData radioData = new RadioData();
        radioData.radioEntity = entity;
        radioData.info = info;

        if (this.getChannelRadiosMap().isEmpty() || !
this.channelRadiosMap.containsKey(info.getChannel())) {
            this.getChannelRadiosMap().put(info.getChannel(), new Vector<RadioData>());
            this.getChannelRadiosMap().get(info.getChannel()).addElement(radioData);
        } else {
            if (this.getChannelRadiosMap().containsKey(info.getChannel())) {
                if (!this.channelRadiosMap.get(info.getChannel()).contains(radioData)) {
                    this.getChannelRadiosMap().get(info.getChannel()).addElement(radioData);
                }
            } else {
                this.getChannelRadiosMap().put(info.getChannel(), new Vector<RadioData>());
                this.getChannelRadiosMap().get(info.getChannel()).addElement(radioData);
            }
        }

        // If radio already exists in some channel, remove it from radioChannelMap and
channelRadiosMap
        if (getRadioIdRadioDataMap().isEmpty() || !
radioIdRadioDataMap.containsKey(radioData.info.getId())) {
            // If the channel does not exist yet in the Phy, add it
            this.getRadioIdRadioDataMap().put(radioData.info.getId(), radioData);
```

```
        } else {
            // Se o radio existir, remover ele primeiro
            if (getRadioIdRadioDataMap().containsKey(radioData.info.getId())) {
                getRadioIdRadioDataMap().remove(radioData.info.getId());
            }
            this.getRadioIdRadioDataMap().put(radioData.info.getId(), radioData);
        }

        if (this.getchannelBufferSignals().isEmpty() || !
this.channelBufferSignals.containsKey(info.getChannel())) {
            // Somente cria a estrutura. Os sinais serão adicionados depois.
            this.getchannelBufferSignals().put(info.getChannel(), new Vector<Signal>());
        }

        this.loc.addRadio(radioData.info.getId());

    }

    public void exit(Integer iD) {
        RadioData data = this.getRadioIdRadioDataMap(iD);

        (this.channelRadiosMap.get(data.oChannel)).removeElement(iD);
        this.radioIdRadioDataMap.remove(iD);

        this.loc.removeRadio(iD);
    }

    /*
     * Transmits a signal to all radios on a channel.
     */
    public void transmit(RadioInfo srcInfo, Signal signal, long duration, int channel) {
        // transmitPower tem que ser maior ou igual ao limiar de potência da PHY - fonte:
SWANS
```

```
        if (srcInfo.getTransmitPower() >= this.propagationLimit) {
            if (!this.channelBufferSignals.containsKey(channel)) {
                this.getchannelBufferSignals().put(channel, new Vector<Signal>());
            }

            this.insertSignalBuffer(signal, channel, duration);

            for (RadioData dstRadio : this.getChannelRadiosMap().get(channel)) {
                if (srcInfo.getId() == dstRadio.info.getId()) {
                    continue;
                }

                this.tansmitToRadio(srcInfo, this.loc.getRadioPontoMap().get(srcInfo.getId()),
dstRadio.info,
                        this.loc.getRadioPontoMap().get(dstRadio.info.getId()),
dstRadio.radioEntity, signal, new Long(duration));
            }

            //Escalona a remoção do Signal do Buffer
            JistAPI.sleep(duration);
            this.removeSignalBuffer(signal, channel);
        }
    }

    public void tansmitToRadio(RadioInfo srcInfo, Ponto srcPonto, RadioInfo dstInfo,
            Ponto dstPonto, CogRadioInterface dstEntity, Signal signal, Long duration) {

        // calcula a intensidade do sinal
        double loss = pathloss.compute(srcInfo, srcPonto, dstInfo, dstPonto);
        double fade = fading.compute();
        double dstPower = srcInfo.getTransmitPower() - loss + fade;
        //Math.pow(10.0, x / 10.0)
        // additional cuttoffs
        double dstPower_mW = Math.pow(10.0, dstPower / 10.0);
```

```
    // if(dstPower_mW < dstInfo.getShared().getBackground_mW()) return;
    if (dstPower_mW < dstInfo.getSensitivity_mW()) {
        System.out.println("Signal descartado");
        return;
    }

    dstEntity.receive(signal, new Double(dstPower_mW), duration);

}

public void insertSignalBuffer(Signal signal, int channel, Long duration) {
    this.getchannelBufferSignals().get(channel).addElement(signal);
}

public void removeSignalBuffer(Signal signal, int channel) {
    this.getchannelBufferSignals().get(channel).removeElement(signal);

}

/* TODO - Review this code
 *
private String calcularDistancias(Hashtable<Integer, RadioData> radios) {
RadioData data1 = null;
RadioData data2 = null;
Ponto p1 = null;
Ponto p2 = null;
double distancia = 0.0;
StringBuilder sb = new StringBuilder();

for (Integer z = 1; z < radios.size(); z++) {
data1 = (RadioData) radios.get(z);
p1 = data1.loc.getRadioPontoMap().get(data1.info.getId());
if (radios.size() > 1) {
for (Integer x = 1; x < radios.size(); x++) {
```

```
data2 = (RadioData) radios.get(x);
if (data1.info.getId() == data2.info.getId()) {
continue;
}
p2 = data2.loc.getRadioPontoMap().get(data2.info.getId());
distancia = p1.distancia(p2);
sb.append("Radio " + data1.info.getId() + " " + data2.info.getId() + " : " + distancia);
}
}
}
sb.append("FIM");
return sb.toString();
 *
}
 */
// GETTERS AND SETTERS
/**
 * @return the channelBufferSignals
 */
public Hashtable<Integer, Vector<Signal>> getchannelBufferSignals() {
    return channelBufferSignals;
}

/**
 * @return the pathloss
 */
public PathLoss getPathloss() {
    return pathloss;
}

/**
 * @param pathloss the pathloss to set
 */
public void setPathloss(PathLoss pathloss) {
```

```
    this.pathloss = pathloss;
}

/**
 * @return the fading
 */
public Fading getFading() {
    return fading;
}

/**
 * @param fading the fading to set
 */
public void setFading(Fading fading) {
    this.fading = fading;
}

/**
 * @return the limite
 */
public double getLimit() {
    return propagationLimit;
}

/**
 * @param limite the limite to set
 */
public void setLimite(double limite) {
    this.propagationLimit = limite;
}

/**
 * @param channelBufferSignals the channelBufferSignals to set
 */
```

```
    public void setchannelBufferSignals(Hashtable<Integer, Vector<Signal>>
channelBufferSignals) {
        this.channelBufferSignals = channelBufferSignals;
    }

    /**
     * @return the channelRadiosMap
     */
    public Hashtable<Integer, Vector<RadioData>> getChannelRadiosMap() {
        return channelRadiosMap;
    }

    /**
     * @param channelRadiosMap the channelRadiosMap to set
     */
    public void setChannelRadiosMap(Hashtable<Integer, Vector<RadioData>>
channelRadiosMap) {
        this.channelRadiosMap = channelRadiosMap;
    }

    /**
     * @return the radioIdRadioDataMap
     */
    public Hashtable<Integer, RadioData> getRadioIdRadioDataMap() {
        return radioIdRadioDataMap;
    }

    /**
     * @return the RadioData
     */
    public RadioData getRadioIdRadioDataMap(Integer iD) {
        return radioIdRadioDataMap.get(iD);
    }
```

```
    /**
     * @param radioIdRadioDataMap the radioIdRadioDataMap to set
     */
    public void setRadioIdRadioDataMap(Hashtable<Integer, RadioData>
radioIdRadioDataMap) {
        this.radioIdRadioDataMap = radioIdRadioDataMap;
    }

    // Similar ao SWANS
    // Utilizada para catalogar os rádios e suas informações ao entrarem na PHY
    public static class RadioData {

        public CogRadioInterface radioEntity;
        public RadioInfo info;
        public Topography2D loc;
        public Integer oChannel;
    }
    // Method that returns the result of the combination of signals that travel in a channel
//    private Signal analyzeSignalsOnChannel(int channel, long duration) {
//        Signal newSignal = null;
//        double intervalStart = JistAPI.getTime() / 1000;
//        double intervalStop = duration / 1000;
    //TODO - verificar se synchronized funciona
    //synchronized (this.getCurrentSignals()) {
//        try {
//        if (this.getchannelBufferSignals().get(channel).size() == 1) {
//            newSignal = this.getchannelBufferSignals().get(channel).get(0);
//        } else {
//            newSignal = this.combineSignalsOnChannel(intervalStart, intervalStop);
//        }
//        } catch (Exception e) {
//            e.printStackTrace();
//        } finally {
//        this.getCurrentSignals().notify();
```

```
//        }
//        return newSignal;
//    }

    /*
    Método que recebe dois instantes de tempo, onde a saída é um Signal cujos Pulses representam
     * aqueles que estiverem no intervalo entre o tempo atual e o tempo de duração da transmissão
     */
//    protected Signal combineSignalsOnChannel(double start, double stop) {
//        Signal combinedSignals = null;
//        // verificar em todos os signals, o de menor início e o de maior fim
//        return combinedSignals;
//    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.phy;

import br.com.datacoffee.cognet.radio.CogRadioInterface;
import br.com.datacoffee.cognet.radio.RadioInfo;
import jist.runtime.JistAPI;

public interface CogPhyInterface extends JistAPI.Proxiable {

    //void enter (Channel channel, CogRadioInterface entity, RadioInfo radio);
    void enter(CogRadioInterface entity, RadioInfo radio);

    void exit(Integer iD);

    void transmit(RadioInfo info, Signal signal, long duration, int channel);
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.phy;

import java.util.Random;

/**
 *
 * @author everton
 */
public interface Fading {

    /**
     * Compute the fading loss.
     *
     * @return fading loss (units: dB)
     */
    double compute();
```

```
/* Computes zero fading.
 *
 * @author Rimon Barr &amp;lt;barr+jist@cs.cornell.edu&amp;gt;
 * @since SWANS1.0
 */
final class None implements Fading {
    // Fading interface

    /** {@inheritDoc} */
    public double compute() {
        return 0.0;
    }
}

/**
 * Computes Rician fading. Equivalent to GloMoSim code.
 *
 * @author Rimon Barr &amp;lt;barr+jist@cs.cornell.edu&amp;gt;
 * @since SWANS1.0
 */
final class Rician implements Fading {

    /** distribution parameters. */
    private final double kFactor,  stddev;
    public static final double log10 = Math.log(10);
    public static Random random = new Random(0);

    /**
     * Create new Rician fading model object.
     *
     * @param kFactor k
     */
    public Rician(double kFactor) {
```

```
        this.kFactor = kFactor;
        this.stddev = computeStandardDeviation(kFactor);
    }

    /**
     * Compute zero-order Bessel function.
     *
     * @param x input
     * @return output of Bessel
     */
    private static double Besseli0(double x) {
        double ax = Math.abs(x);
        if (ax < 3.75) {
            double y = x / 3.75;
            y *= y;
            return 1.0 + y * (3.5156229 + y * (3.0899424 + y * (1.2067492 +
                    y * (0.2659732 + y * (0.360768e-1 + y * 0.45813e-2)))));
        } else {
            double y = 3.75 / ax;
            return (Math.exp(ax) / Math.sqrt(ax)) * (0.39894228 + y * (0.1328592e-1 +
                    y * (0.225319e-2 + y * (-0.157565e-2 + y * (0.916281e-2 + y * (-
0.2057706e-1 +
                    y * (0.2635537e-1 + y * (-0.1647633e-1 + y * 0.392377e-2)))))));
        }
    }

    /**
     * Compute first-order Bessel function.
     *
     * @param x input
     * @return output of Bessel
     */
    private static double Besseli1(double x) {
        double ax = Math.abs(x);
```

```
            if (ax < 3.75) {
                double y = x / 3.75;
                y *= y;
                return x * (0.5 + y * (0.87890494 + y * (0.51498869 + y * (0.15084934 + y *
(0.2658733e-1 +
                        y * (0.301532e-2 + y * 0.32411e-3))))));
            } else {
                double y = 3.75 / ax;
                return Math.abs((Math.exp(ax) / Math.sqrt(ax)) * (0.39894228 + y * (-
0.3988024e-1 +
                        y * (-0.362018e-2 + y * (0.163801e-2 + y * (-0.1031555e-1 + y *
(0.2282967e-1 +
                        y * (-0.2895312e-1 + y * (0.1787654e-1 - y * 0.420059e-2)))))))));
            }
        }

        /**
         * Computes standard deviation for Rician distribution such that mean is 1.
         *
         * @param kFactor k
         * @return Rician standard deviation
         */
        private static double computeStandardDeviation(double kFactor) {
            return 1.0 / (Math.sqrt(Math.PI / 2.0) * Math.exp(-kFactor / 2.0) *
                    ((1 + kFactor) * Besseli0(kFactor / 2.0) + kFactor * Besseli1(kFactor / 2.0)));
        }

        // Fading interface
        /** {@inheritDoc} */
        public double compute() {
            // compute fading_dB; positive values are signal gains
            double a = Math.sqrt(2.0 * kFactor * stddev * stddev), r, v1, v2;
            do {
                v1 = -1.0 + 2.0 * random.nextDouble();
```

```
            v2 = -1.0 + 2.0 * random.nextDouble();
            r = v1 * v1 + v2 * v2;
        } while (r > 1.0);
        r = Math.sqrt(-2.0 * Math.log(r) / r);
        v1 = a + stddev * v1 * r;
        v2 = stddev * v2 * r;
        return 5.0 * Math.log(v1 * v1 + v2 * v2) / log10;
      }
    }
} // class: Fading
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.phy;

import br.com.datacoffee.cognet.Constants;
import br.com.datacoffee.cognet.phy.Topography2D.Ponto;
import br.com.datacoffee.cognet.radio.RadioInfo;

/**
 *
 * @author everton
 */
public interface PathLoss {

    ////////////////////////////////////////////////
    // interface
    //
    /**
```

```
     * Compute the path loss.
     *
     * @param srcRadio source radio information
     * @param srcLocation source location
     * @param dstRadio destination radio information
     * @param dstLocation destination location
     * @return path loss (units: dB)
     */
    double compute(RadioInfo srcInfo, Ponto srcPonto, RadioInfo dstInfo, Ponto dstPonto);

    final class FreeSpace implements PathLoss {
        // PathLoss interface

        /** {@inheritDoc} */
        public double compute(RadioInfo srcInfo, Ponto srcPonto, RadioInfo dstInfo, Ponto
dstPonto) {

            double log10 = Math.log(10);

            double dist = srcPonto.distancia(dstPonto);
            double pathloss = -srcInfo.getGain() - dstInfo.getGain();
            double valueForLog = 4.0 * Math.PI * dist / srcInfo.getWaveLength();
            if (valueForLog > 1.0) {
                pathloss += (float) Math.log((float) valueForLog) / log10 * 20.0;
            }
            return pathloss;
        }
    }

    final class TwoRay implements PathLoss {
        // PathLoss interface

        /** {@inheritDoc} */
        public double compute(RadioInfo srcInfo, Ponto srcPonto, RadioInfo dstInfo, Ponto
```

```
dstPonto) {

        double dist = srcPonto.distancia(dstPonto);
        double pathloss = -srcInfo.getGain() - dstInfo.getGain();
        double planeEarthLoss = (dist * dist) /
            (Topography2D.height * Topography2D.height);
        double freeSpaceLoss = 4.0 * Math.PI * dist / srcInfo.getWaveLength();
        if (planeEarthLoss > freeSpaceLoss) {
          if (planeEarthLoss > 1.0) {
            pathloss += 20.0 * Math.log(planeEarthLoss) / Constants.log10;
          }
        } else {
          if (freeSpaceLoss > 1.0) {
            pathloss += 20.0 * Math.log(freeSpaceLoss) / Constants.log10;
          }
        }
        return pathloss;
      }
    } // class: TwoRay
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/



package br.com.datacoffee.cognet.phy;

public class Pulse implements Comparable<Pulse> {

    /*
     * 1s
     * 1ms = 0,001s - milisegundo
     * 1mcs = 0,001ms - microsegundo
     * 1ns = 0,001mcs - nanosegundo
     * 1ps = 0,001ns - picosegundo
     *
     * Para fins de cálculos, o tempo = JistAPI.getTime() / 1000.
     */
    protected double startMoment;
    protected double stopMoment;
```

```
protected int bit;
protected double power;

public Pulse(double start, double stop, int bit, double power) {
    this.setPower(power);
    this.setStartMoment(start);
    this.setStopMoment(stop);
    this.setBit(bit);
}

public Pulse(double start, double stop, int bit) {
    this.setPower(power);
    this.setStartMoment(start);
    this.setStopMoment(stop);
    this.setBit(bit);
}

public double getStartMoment() {
    return this.startMoment;
}

public void setStartMoment(double startMoment) {
    this.startMoment = startMoment;
}

public double getStopMoment() {
    return this.stopMoment;
}

public void setStopMoment(double stopMoment) {
    this.stopMoment = stopMoment;
}

public int getBit() {
```

```
        return this.bit;
    }

    public void setBit(final int bit) {
        this.bit = bit;
    }

    public double getPower() {
        return this.power;
    }

    public void setPower(double power) {
        this.power = power;
    }

    public int compareTo(Pulse o) {
        return (this.startMoment < o.startMoment && this.stopMoment < o.startMoment) ? -1 :
((this.startMoment > o.stopMoment && this.stopMoment > o.stopMoment) ? +1 : 0);
    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.phy;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.TreeSet;

public class Signal implements jist.runtime.JistAPI.Timeless {

    //Conjunto de Pulses
    protected TreeSet<Pulse> pulsesIntervals = new TreeSet<Pulse>(new
PulseComparator());
    // identifica o comprimento da parte de dados no treeset, que na verdade é a soma da
quantidade de pulses;
    protected int size;
    // identifica a modulação do sinal
```

```
    protected int modulation;


    public Signal(TreeSet<Pulse> pulses, int modulation) {
        if (pulses != null) {
            this.pulsesIntervals = pulses;
        }
        this.size = pulses.size();

        this.modulation = modulation;
    }

    public Signal(TreeSet<Pulse> pulses) {
        this(pulses, Modulation.FSK);
    }

    public int getSize() {
        return this.size;
    }

    public TreeSet<Pulse> getPulses() {
        return this.pulsesIntervals;
    }

    public void putPulse(int start, int stop, int bit, double power) {
        Pulse newPulse = new Pulse(start, stop, bit, power);
        this.pulsesIntervals.add(newPulse);
    }

    /*
    Retorna o Bit referente ao time solicitado, entretanto, num time podem haver mais de
um Pulse
     * e consequentemente mais de um BIT, então retorna-se um ArrayList
     */
```

```
    public ArrayList<Integer> getBit(long time) {
        /*Os Pulse de um Signal não precisãm preencher o signal por completo.
        Se for solicitado o bit para o instante Pn e não houver um pulse que abarque aquele instante,
        então será verificado se pelo menos mais da metade dele está no intervalo, que se tiver retorna o bit dentro do arraylist,
        caso contrário será retornado um null, e deve ser descartado.
         * */

        DecimalFormat df = new DecimalFormat("#.####");
        ArrayList<Integer> bits = new ArrayList();

        double start = Double.valueOf(df.format(time / 1000));
        double stop = Double.valueOf(df.format(time / 1000 + 0.000999));
        double tamanho = Double.valueOf(df.format(stop - start));

        for (final Pulse p : this.pulsesIntervals) {
            double tamanhoPulse = Double.valueOf(df.format(p.getStopMoment() - p.getStartMoment()));
            double intervaloInferior = Double.valueOf(df.format(p.getStopMoment() - start));
            double intervaloSuperior = Double.valueOf(df.format(stop - p.getStartMoment()));
            double porcentagem = 0.0;
            /**
             * if the interval i (start) until i+1 (stop) is out of p.start until p.stop
             * then it continues.
             */
            if (p.getStopMoment() < start) {
                continue;
            }
            // Se estiver depois, não interessa e sai fora.
            if (p.getStartMoment() > stop) {
                break;
            }
```

```
            if (start >= p.getStartMoment() && stop <= p.getStopMoment()) {
                bits.add((Integer)p.getBit());
                continue;

            } else if (p.getStartMoment() < start && (p.getStopMoment() > start &&
p.getStopMoment() < stop)) {
                // intervalo de interesse está pra tras
                porcentagem = Double.valueOf(df.format((intervaloInferior * 100) /
tamanhoPulse));
                if (porcentagem > 50) {
                    bits.add((Integer)p.getBit());
                    continue;
                }

            } else if (p.getStopMoment() > stop && (p.getStartMoment() > start &&
p.getStartMoment() < stop)) {
                // intervalo de interesse está pra frente
                porcentagem = Double.valueOf(df.format((intervaloSuperior * 100) /
tamanhoPulse));
                if (porcentagem > 50) {
                    bits.add((Integer)p.getBit());
                    continue;
                }
            }
        }
        return bits;
    }

    class PulseComparator implements Comparator<Pulse> {

        public int compare(Pulse p1, Pulse p2) {
            return p1.compareTo(p2);
        }
    }
```

```
public static final class Modulation {

    public static final int FSK = 20;
    public static final int PSK = 5;
    public static final int BPSK = 21;
    public static final int QPSK = 33;
    public static final int NOSIGNAL = 1;

    public String getName(int modulation) {
        switch (modulation) {
            case FSK:
                return "FSK";
            case PSK:
                return "PSK";
            case BPSK:
                return "BPSK";
            case QPSK:
                return "QPSK";
            case NOSIGNAL:
                return "NOSIGNAL";
            default:
                throw new IllegalArgumentException("Invalid bit value:" + modulation);
        }
    }
}

@Override
public String toString() {
    DecimalFormat df = new DecimalFormat("#.####");
    StringBuilder sb = new StringBuilder();
    sb.append('[');
    for (Pulse pulse : this.pulsesIntervals) {
        sb.append('(').append(pulse.getStartMoment()).append(',').append(pulse.getStopM
```

```
oment()).append(',').append(Bit.getName(pulse.getBit())).append(',').append(pulse.getPow
er()).append(')').append(',');
        }

        int last = sb.length() - 1;
        if (sb.charAt(last) == ',') {
            sb.deleteCharAt(last);
        }
        sb.append(']');
        return sb.toString();
    }
//    public int getBit(final int i, final int modulation) {
//        final double start = i;
//        final double stop = i + 1;
//
//        double unknowDuration = 0;
//        double noSignalDuration = 0;
//        double highDuration = 0;
//        double lowDuration = 0;
//
//        for (final Pulse p : this.pulsesIntervals) {
//            /**
//             * if the interval i (start) until i+1 (stop) is out of p.start until p.stop
//             * then it continues.
//             */
//            if (p.getStopMoment() < start) {
//                continue;
//            }
//
//            if (p.getStartMoment() > stop) {
//                break;
//            }
    /**
     * if the interval i (start) until i+1(stop) has any part in common with
```

```
     * p.start until p.stop, then calculate what is the length of this part.
     */
//           double d = (p.getStopMoment() > stop) ? stop : p.getStopMoment();
//           d -= (p.getStartMoment() < start) ? start : p.getStartMoment();
    /**
     * if there isn't a identified bit, then assign p.bit to current bit
     * else, if there is a bit then if the bit has the same type of the last bit read it increase the duration
     * else, if there is a other type bit, then the duration will put the difference between last bit duration and the current bit duration
     */
//           switch (p.getBit()) {
//               case Bit.NOSIGNAL:
//                   noSignalDuration += d;
//                   break;
//               case Bit.UNKNOW:
//                   unknowDuration += d;
//                   break;
//               case Bit.HIGH:
//                   highDuration += d;
//                   break;
//               case Bit.LOW:
//                   lowDuration += d;
//                   break;
//           }
//       }
//
//       Integer ret = null;
//       if (noSignalDuration >= Signal.DEFAULT_BIT_THRESHOLD) {
//           ret = Bit.NOSIGNAL;
//       } else if (unknowDuration >= Signal.DEFAULT_BIT_THRESHOLD) {
//           ret = Bit.UNKNOW;
//       } else if (highDuration >= Signal.DEFAULT_BIT_THRESHOLD) {
//           ret = Bit.HIGH;
```

```
//        } else if (lowDuration >= Signal.DEFAULT_BIT_THRESHOLD) {
//            ret = Bit.LOW;
//        }else{
//
//        /**
//             * raffle a bit based on the duration of each duration
//             * TODO review this algorithm
//         */
//            ret = this.pickUpBit(noSignalDuration, unknowDuration, highDuration,
lowDuration);
//        }
//        return ret;
//    }
    /**
     * pick up a Bit with random criteria
     */
//    private int pickUpBit(double noSignalDuration, double unknowDuration, double
highDuration, double lowDuration) {
//        Random r = new Random();
//
//        double factor = (noSignalDuration + unknowDuration + highDuration +
lowDuration) / 4;
//
//        double noSignal = r.nextDouble() * noSignalDuration * factor;
//        double unknow = r.nextDouble() * unknowDuration * factor;
//        double high = r.nextDouble() * highDuration * factor;
//        double low = r.nextDouble() * lowDuration * factor;
//
//        if (noSignal > unknow && noSignal > high && noSignal > low) {
//            return Bit.NOSIGNAL;
//        } else if (unknow > noSignal && unknow > high && noSignal > low) {
//            return Bit.UNKNOW;
//        } else if (high > noSignal && high > unknow && high > low) {
//            return Bit.HIGH;
```

```
//        } else {
//            return Bit.LOW;
//        }
//    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.phy;

import jist.runtime.JistAPI;

/**
 *
 * @author everton
 */
public interface Topography extends JistAPI.Timeless {

    void addRadio(Integer radioId);

    void removeRadio(Integer radioId);
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.phy;

import br.com.datacoffee.cognet.tools.Logging;
import java.util.Hashtable;
import java.util.Random;
import jist.runtime.JistAPI;

/**
 *
 * @author everton
 */
public class Topography2D implements Topography {

    Logging logger;

    //definiu-se um valor padrão para representar a distância em metros de um ponto a
```

outro.

```
    protected static double height = 7.0711;

    // aXb, where a=radioId, b=Ponto
    private Hashtable<Integer, Ponto> radioPontoMap;

    protected Integer length = 0;

    protected Integer[][] grid;




    public Topography2D(Integer length) {
        logger = new Logging();

        this.radioPontoMap = new Hashtable<Integer, Ponto>();

        this.length = length;

        this.grid = new Integer[length][length];
    }

    public void init(Integer inteiro){

    }

    public void addRadio(Integer radioId) {
        boolean livre = false;
        Integer x = null;
        Integer y = null;
        Ponto ponto = new Ponto();
        NumeroRandomico nrandom = new NumeroRandomico();
```

```
        while (livre == false) {
            x = nrandom.getRandomInt(length-1);
            y = nrandom.getRandomInt(length-1);
            if (grid[x][y] == null) {
                ponto.setX(x);
                ponto.setY(y);
                this.grid[x][y] = radioId;
                this.getRadioPontoMap().put(radioId, ponto);
                livre = true;
            }
        }
        logger.log("Topography2D.ADDRADIO, RadioID: " + radioId + ", PONTO:
("+ponto.getX()+","+ponto.getY()+"), TIME: " + JistAPI.getTime());
    }

    public void removeRadio(Integer radioId) {
        for (int y = 0; y < length; y++) {
            for (int x = 0; x < length; x++) {
                if (grid[x][y] == radioId) {
                    grid[x][y] = null;
                    this.getRadioPontoMap().remove(radioId);
                    return;
                }
            }
        }
        logger.log("Topography2D.ADDRADIO, RadioID: " + radioId + ", TIME: " +
JistAPI.getTime());
    }

    public boolean addRadio(Integer radioId, Ponto ponto) {
        if (grid[ponto.x][ponto.y] == null) {
            grid[ponto.x][ponto.y] = radioId;
            this.getRadioPontoMap().put(radioId, ponto);
            return true;
```

```
    }
    return false;
}

public boolean removeRadio(Integer radioId, Ponto onde) {
    if (grid[onde.x][onde.y] == radioId) {
        grid[onde.x][onde.y] = null;
        this.getRadioPontoMap().remove(radioId);
        return true;
    }
    return false;
}

/**
 * @return the radioPontoMap
 */
public Hashtable<Integer, Ponto> getRadioPontoMap() {
    return radioPontoMap;
}

public static class Ponto {

    private int x,  y;

    public Ponto() {
    }

    public Ponto(int a, int b) {
        x = a;
        y = b;
    }

    public void setX(int a) {
        x = a;
```

```
        }

        public void setY(int a) {
            y = a;
        }

        public int getX() {
            return x;
        }

        public int getY() {
            return y;
        }

        public String toString() {
            return x + ";" + y;
        }

        public double distOrigem() {
            return Math.sqrt(getX() * getX() + getY() * getY());
        }

        // Calcula distancia entre dois pontos
        public double distancia(Ponto p) {
            return (Math.sqrt((getX() - p.getX()) * (getX() - p.getX()) + (getY() - p.getY()) *
(getY() - p.getY()))) * Topography2D.height;
        }
    }

    public static class NumeroRandomico {

        Random rand = new Random();

        public Integer getRandomInt() {
```

```
            return rand.nextInt();
        }

        public Integer getRandomInt(Integer limite) {
            return rand.nextInt(limite + 1);
        }

        public boolean getRandomBoolean() {
            return rand.nextBoolean();
        }

        public long getRandomLong() {
            return rand.nextLong();
        }

        public float getRandomFloat() {
            return rand.nextFloat(); // 0.0 <= f < 1.
        }

        public double getRandomDouble() {
            return rand.nextDouble(); // 0.0 <= d < 1.0
        }
    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.radio;

import br.com.datacoffee.cognet.phy.Signal;
import jist.runtime.JistAPI;

public interface CogRadioInterface extends JistAPI.Proxiable {

    /* Inicia o processo de transmissão. Coloca o rádio em modo de transmissão
    e envia o Signal para aqueles rádios que podem recebê-lo.
     *
     * @param signal Signal object to transmit
     * @param delay time to the wire
     * @param duration time on the wire
     */
    void transmit(Signal signal, long duration, int channel);
```

```
    /**
     * End message transmission. Putting the radio back into idle (or possibly
     * receiving) mode. Called from mac entity.
     */
    void endTransmit(int radioId);

    /**
     * Start receiving message. Puts radio into receive or sensing mode depending
     * on the message power and the state of the radio. A radio that is currently
     * transmitting will ignore incoming messages. Called from field entity.
     *
     * @param signal Signal incoming signal
     * @param power signal strength of incoming message (units: mW)
     * @param duration time until end of transmission (units: simtime)
     */
    void receive(Signal signal, Double power, Long duration);

    /**
     * End message reception. Puts the radio back into sensing or idle mode, and
     * sends the received message to upper layers for processing, if no error has
     * occurred during the reception. Called from field entity.
     *
     *
     */
    void endReceive(int radioId);
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.radio;

import br.com.datacoffee.cognet.phy.CogPhyInterface;
import br.com.datacoffee.cognet.phy.Signal;
import br.com.datacoffee.cognet.tools.Logging;
import jist.runtime.JistAPI;

/**
 *
 * @author everton
 */
public class CogRadioMock implements CogRadioInterface {

    /**
     * self-referencing radio entity reference.
     */
```

```
    protected CogRadioInterface self;
    /**
     * field entity downcall reference.
     */
    protected CogPhyInterface cogPhyFieldEntity;
    /**
     * radio informations.
     */
    protected RadioInfo info;

    // Logador
    Logging logger = new Logging();

    public CogRadioMock(RadioInfo info) {
        this.self = (CogRadioInterface) JistAPI.proxy(this, CogRadioInterface.class);
        this.info = info;
    }

    /**
     * Return self-referencing radio entity reference.
     *
     * @return self-referencing radio entity reference
     */
    public CogRadioInterface getProxy() {
        return this.self;
    }

    public void transmit(Signal signal, long duration, int channel) {
        //Loga
        logger.log("CogRadioMock.TRANSMIT, RadioID: " + this.info.getId() + ", SIGNAL: " +
signal.toString() + ", TRANSMIT_POWER: " + this.info.getTransmitPower() + ", TIME: " +
JistAPI.getTime()+" DURACAO: "+duration);
        cogPhyFieldEntity.transmit(info, signal, duration, channel);
        JistAPI.sleep(duration);
```

```
        self.endTransmit(this.info.getId());
    }

    public void endTransmit(int radioId) {
        //logger.log("CogRadioMock.ENDTRANSMIT ->, RadioID: "+radioId+", TIME:
"+JistAPI.getTime());
    }

    public void receive(Signal signal, Double power, Long duration) {
        JistAPI.sleep(duration);
        // Loga
        logger.log("CogRadioMock.RECEIVE, RadioID: " + this.info.getId() + ", SIGNAL: " +
signal.toString() + ", RECEIVE_POWER: " + power + ", TIME: " + JistAPI.getTime());
        self.endReceive(this.info.getId());

    }

    public void endReceive(int radioId) {
        //logger.log("CogRadioMock.ENDRECEIVE ->, RadioID: "+radioId+", TIME:
"+JistAPI.getTime());
    }

    /**
     * Set upcall field entity reference.
     *
     * @param fieldEntity upcall field entity reference
     */
    public void setFieldEntity(CogPhyInterface fieldEntity) {
        if (!JistAPI.isEntity(fieldEntity)) {
            throw new IllegalArgumentException("entity expected");
        }
        this.cogPhyFieldEntity = fieldEntity;
    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.radio;

import br.com.datacoffee.cognet.mac.CogMacInterface;
import br.com.datacoffee.cognet.phy.CogPhyField;
import br.com.datacoffee.cognet.phy.CogPhyInterface;
import br.com.datacoffee.cognet.phy.Signal;
import jist.runtime.JistAPI;
import jist.swans.radio.BERTable;

/**
 *
 * @author everton
 */
public class CogRadioNoiseMock implements CogRadioInterface {

    /*
```

```
Referência própria à entidade CogRadioInterface
 */
protected CogRadioInterface self;
/*
Referência à camada CogPhyField, para receber chamadas
 */
protected CogPhyInterface phy;
/*
CogMac entity upcall reference.
 */
protected CogMacInterface macEntity;
/*
Propriedades
 */
protected RadioInfo info;
/**
 * radio mode: IDLE, SENSING, RECEIVING, SENDING, SLEEP.
 */
protected byte mode;
/**
 * Signal sendo recebido
 */
protected Signal signalBuffer;
/**
 * number of signals being received.
 */
protected int signals;
/**
 * transmission signal strength.
 */
protected double signalPower_mW;
/**
 * end of transmission time.
 */
```

```
    protected long signalFinish;
    /**
     * radio type: SNR or BER.
     */
    protected byte type;
    /**
     * threshold signal-to-noise ratio.
     */
    protected float thresholdSNR;
    /**
     * total signal power.
     */
    protected double totalPower_mW;
    /**
     * bit-error-rate table.
     */
    protected BERTable ber;

    /**
     * Create a new radio with additive noise model.
     *
     * @param id radio identifier
     * @param shared shared radio properties
     * @param snrThreshold_mW threshold signal-to-noise ratio
     */
    public CogRadioNoiseMock(int id, RadioInfo info, float snrThreshold_mW) {
        this.totalPower_mW = info.getBackground_mW();
        this.mode = (totalPower_mW > info.getSensitivity_mW()) ?
RadioMode.RADIO_MODE_SENSING : RadioMode.RADIO_MODE_IDLE;
        this.unlockSignal();
        this.signals = 0;
        this.info = info;
        this.type = RadioConstants.SNR;
        this.thresholdSNR = snrThreshold_mW;
```

```
    this.self = (CogRadioInterface) JistAPI.proxy(this, CogRadioInterface.class);
}

/**
 * Return self-referencing radio entity reference.
 *
 * @return self-referencing radio entity reference
 */
public CogRadioInterface getProxy() {
    return this.self;
}

/**
 * Set upcall field entity reference.
 *
 * @param fieldEntity upcall field entity reference
 */
public void setPhy(CogPhyField PhyFieldEntity) {
    if (!JistAPI.isEntity(PhyFieldEntity)) {
        throw new IllegalArgumentException("entity expected");
    }
    this.phy = PhyFieldEntity;
}

/**
 * Set downcall mac entity reference.
 *
 * @param macEntity downcall mac entity reference
 */
public void setMacEntity(CogMacInterface macEntity) {
    if (!JistAPI.isEntity(macEntity)) {
        throw new IllegalArgumentException("entity expected");
    }
    this.macEntity = macEntity;
```

```
}

// Transmissão
//
// RadioInterface interface
/** {@inheritDoc} */
public void transmit(Signal signal, long duration, int channel) {
    // radio in sleep mode
    if (mode == RadioMode.RADIO_MODE_SLEEP) {
        return;
    }
    // ensure not currently transmitting
    if (mode == RadioMode.RADIO_MODE_TRANSMITTING) {
        throw new RuntimeException("radio already transmitting");
    }
    // clear receive buffer
    signalBuffer = null;
    // set mode to transmitting
    mode = RadioMode.RADIO_MODE_TRANSMITTING;

    phy.transmit(info, signal, duration, channel);
    // schedule end of transmission
    JistAPI.sleep(duration);
    self.endTransmit(this.info.getId());
}

// RadioInterface interface
/** {@inheritDoc} */
public void endTransmit(int radioId) {
    // radio in sleep mode
    if (mode == RadioMode.RADIO_MODE_SLEEP) {
        return;
    }
    // check that we are currently transmitting
```

```
        if (mode != RadioMode.RADIO_MODE_TRANSMITTING) {
            throw new RuntimeException("radio is not transmitting");
        }
        // set mode
        mode = (signals > 0 ? RadioMode.RADIO_MODE_RECEIVING :
RadioMode.RADIO_MODE_IDLE);
    }

    ////////////////////////////////////////////////
    // Recepção
    //
    // RadioInterface interface
    /** {@inheritDoc} */
    public void receive(final Signal signal, final Double powerObj_mW, final Long
durationObj) {

        final double power_mW = powerObj_mW.doubleValue();
        final long duration = durationObj.longValue();

        switch (mode) {
            case RadioMode.RADIO_MODE_IDLE:
                if (power_mW >= info.getThreshold_mW() && power_mW >= totalPower_mW *
thresholdSNR) {
                    lockSignal(signal, power_mW, duration);
                    mode = RadioMode.RADIO_MODE_RECEIVING;
                } else if (totalPower_mW + power_mW > info.getSensitivity_mW()) {
                    mode = RadioMode.RADIO_MODE_SENSING;
                }
                break;
            case RadioMode.RADIO_MODE_SENSING:
                if (power_mW >= info.getThreshold_mW() && power_mW >= totalPower_mW *
thresholdSNR) {
                    lockSignal(signal, power_mW, duration);
                    mode = RadioMode.RADIO_MODE_RECEIVING;
```

```
            }
            break;
        case RadioMode.RADIO_MODE_RECEIVING:
            if (power_mW > signalPower_mW && power_mW >= totalPower_mW *
thresholdSNR) {
                lockSignal(signal, power_mW, duration);
                mode = RadioMode.RADIO_MODE_RECEIVING;
            } else if (type == RadioConstants.SNR && signalPower_mW < (totalPower_mW
- signalPower_mW + power_mW) * thresholdSNR) {
                unlockSignal();
                mode = RadioMode.RADIO_MODE_SENSING;
            }
            break;
        case RadioMode.RADIO_MODE_TRANSMITTING:
            break;
        case RadioMode.RADIO_MODE_SLEEP:
            break;
        default:
            throw new RuntimeException("unknown radio mode");
      }
      // cumulative signal
      signals++;
      totalPower_mW += power_mW;
      // schedule an endReceive
      JistAPI.sleep(duration);
      //self.endReceive(powerObj_mW);
      self.endReceive(this.info.getId());
   } // function: receive

   // RadioInterface interface
   /** {@inheritDoc} */
   public void endReceive(Double powerObj_mW) {
      final double power_mW = powerObj_mW.doubleValue();
      // cumulative signal
```

```
        signals--;
        totalPower_mW = signals == 0 ? info.getBackground_mW() : totalPower_mW -
power_mW;
        switch (mode) {
            case RadioMode.RADIO_MODE_RECEIVING:
                if (JistAPI.getTime() == signalFinish) {
                    // A MAC recebe o Signal
                    this.macEntity.receive(signalBuffer);
                    unlockSignal();
                    mode = (totalPower_mW >= info.getSensitivity_mW() ?
RadioMode.RADIO_MODE_SENSING : RadioMode.RADIO_MODE_IDLE);
                }
                break;
            case RadioMode.RADIO_MODE_SENSING:
                if (totalPower_mW < info.getSensitivity_mW()) {
                    mode = RadioMode.RADIO_MODE_IDLE;
                }
                break;
            case RadioMode.RADIO_MODE_TRANSMITTING:
                break;
            case RadioMode.RADIO_MODE_IDLE:
                break;
            case RadioMode.RADIO_MODE_SLEEP:
                break;
            default:
                throw new RuntimeException("unknown radio mode");
        }
    } // function: endReceive

    /**
     * Lock onto current packet signal.
     *
     * @param msg packet currently on the air
     * @param power_mW signal power (units: mW)
```

```
     * @param duration time to EOT (units: simtime)
     */
    protected void lockSignal(Signal signal, double power_mW, long duration) {
        signalBuffer = signal;
        signalPower_mW = power_mW;
        signalFinish = JistAPI.getTime() + duration;
        this.macEntity.receive(signal);
    }

    /**
     * Unlock from current packet signal.
     */
    protected void unlockSignal() {
        signalBuffer = null;
        signalPower_mW = 0;
        signalFinish = -1;
    }

    public void endReceive(int radioId) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public static class RadioMode {

        /** Radio mode: sleeping. */
        public static final byte RADIO_MODE_SLEEP = -1;
        /** Radio mode: idle, no signals. */
        public static final byte RADIO_MODE_IDLE = 0;
        /** Radio mode: some signals above sensitivity. */
        public static final byte RADIO_MODE_SENSING = 1;
        /** Radio mode: signal locked and receiving packet. */
        public static final byte RADIO_MODE_RECEIVING = 2;
        /** Radio mode: transmitting packet. */
        public static final byte RADIO_MODE_TRANSMITTING = 3;
```

```
    }

    public static class RadioConstants {

        /** Default threshold signal-to-noise ratio. */
        public static final double SNR_THRESHOLD_DEFAULT = 10.0;
        /** signal-to-noise error model constant. */
        public static final byte SNR = 0;
        /** bit-error-rate error model constant. */
        public static final byte BER = 1;
    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.radio;

import jist.runtime.JistAPI;

/**
 *
 * @author everton
 */
public class RadioInfo implements JistAPI.Timeless {

    /*
     * Wavelength of radio (units: meter).
     */
    protected double wavelength;
    /**
     * Bandwidth (units: bits/second).
```

```
 */
protected int bandwidth;
/**
 * Transmission power (units: dBm).
 */
protected double transmitPower;
/**
 * Antenna gain (units: dBm).
 */
protected double gain;
/**
 * Reception sensitivity (units: mW).
 */
protected double sensitivity_mW;
/**
 * Reception threshold (units: mW).
 */
protected double threshold_mW;
/**
 * Background noise, including bandwidth factor
 *   (units: mW * bits/second).
 */
protected double background_mW;

protected Integer id;

protected int channel;

public RadioInfo() {
}

/**
 * @return the channel
 */
```

```
public int getChannel() {
    return channel;
}

/**
 * @return the wavelength
 */
public double getWaveLength() {
    return wavelength;
}

/**
 * @return the bandwidth
 */
public int getBandwidth() {
    return bandwidth;
}

/**
 * @return the transmitPower
 */
public double getTransmitPower() {
    return transmitPower;
}

/**
 * @return the gain
 */
public double getGain() {
    return gain;
}

/**
 * @return the sensitivity_mW
```

```
 */
public double getSensitivity_mW() {
    return sensitivity_mW;
}

/**
 * @return the threshold_mW
 */
public double getThreshold_mW() {
    return threshold_mW;
}

/**
 * @return the background_mW
 */
public double getBackground_mW() {
    return background_mW;
}

/**
 * @return the id
 */
public Integer getId() {
    return id;
}

public static class FrequenciaCanal {
    public static final int getChannel(int frequency){
        int retorno = 0;
        switch (frequency) {
            case 2412: retorno = 1;
            case 2417: retorno = 2;
            case 2422: retorno = 3;
            case 2427: retorno = 4;
```

```
            case 2432: retorno = 5;
            case 2437: retorno = 6;
            case 2442: retorno = 7;
            case 2447: retorno = 8;
            case 2452: retorno = 9;
            case 2457: retorno = 10;
            case 2462: retorno = 11;
            case 2467: retorno = 12;
            case 2472: retorno = 13;
            case 2484: retorno = 14;
        }
        return retorno;
    }
}



/**
 * Create shared radio parameters.
 *
 * @param frequency radio frequency (units: Hertz)
 * @param bandwidth bandwidth (units: bits/second)
 *
 * @param transmit transmission power (units: dBm)
 * @param gain antenna gain (units: dB)
 *
 * @param sensitivity_mW receive sensivity (units: mW)
 * @param threshold_mW receive threshold (units: mW)
 *
 * @param temperature field temperature (units: degrees Kelvin)
 * @param thermalFactor thermal noise
 * @param ambientNoise_mW ambient noise (units: mW)
 *
 * @return shared radio information object
```

```
     */
    public static RadioInfo createInfo(Integer id, double frequency, int frequencia, int
bandwidth,
            double transmit, double gain, double sensitivity_mW, double threshold_mW,
            double temperature, double thermalFactor, double ambientNoise_mW) {
        RadioInfo info = new RadioInfo();

        // iD
        info.id = id;
        // wavelength
        info.wavelength = InfoConstants.SPEED_OF_LIGHT / frequency;
        // bandwidth
        info.bandwidth = bandwidth;
        // channel
        info.channel = FrequenciaCanal.getChannel(frequencia);
        // transmit
        info.transmitPower = transmit;
        info.gain = gain;
        // receive
        info.sensitivity_mW = sensitivity_mW;
        info.threshold_mW = threshold_mW;
        // noise
        double thermalNoise_mW = InfoConstants.BOLTZMANN * temperature *
thermalFactor * 1000.0;
        info.background_mW = (ambientNoise_mW + thermalNoise_mW) * bandwidth;
        return info;
    }

    public static class InfoConstants {

        /** Boltzmann's constant (units: Joules/Kelvin). */
        public static final double BOLTZMANN = 1.3807e-23;
        /** Speed of light in a vacuum (units: meter/second). */
        public static final double SPEED_OF_LIGHT = 2.9979e8;
```

```
    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
 ***/
package br.com.datacoffee.cognet.sim;

import br.com.datacoffee.cognet.tools.Logging;
import br.com.datacoffee.cognet.tools.SignalGenerator;
import br.com.datacoffee.cognet.radio.CogRadioMock;
import br.com.datacoffee.cognet.Constants;
import br.com.datacoffee.cognet.phy.CogPhyField;
import br.com.datacoffee.cognet.phy.CogPhyField.RadioData;
import br.com.datacoffee.cognet.phy.Fading;
import br.com.datacoffee.cognet.phy.PathLoss;
import br.com.datacoffee.cognet.phy.Signal;
import br.com.datacoffee.cognet.phy.Topography2D;
import br.com.datacoffee.cognet.radio.RadioInfo;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Vector;

/**
```

```
 *
 * @author everton
 */
public class MainSim {

    SignalGenerator generator = new SignalGenerator();
    Logging logger = new Logging();
    Topography2D loc = null;

    public void createNode(Integer id, CogPhyField cogPhyField, Topography2D loc,
RadioInfo info) {
        // Create entities
        CogRadioMock mock = new CogRadioMock(info);

        // Add radios to CogPhyField
        cogPhyField.enter(mock.getProxy(), info);
        // Set CogPhyField to radio
        mock.setFieldEntity(cogPhyField.getProxy());
    }

    public CogPhyField createSim(int length, int nodes) {

        loc = new Topography2D(length);
        Fading fading = new Fading.None();
        PathLoss loss = new PathLoss.FreeSpace();
        //PathLoss loss = new PathLoss.TwoRay();
        generator.init();
        // Atributo inserido devido a necessidade de definir o canal,
        int frequencia = 2472;
        CogPhyField cogPhyField = new CogPhyField(loc, fading, loss,
Constants.PROPAGATION_LIMIT_DEFAULT);

        for (int id = 1; id < nodes + 1; id++) {
            RadioInfo info = RadioInfo.createInfo(
```

```
                id, Constants.FREQUENCY_DEFAULT, frequencia,
Constants.BANDWIDTH_DEFAULT, Constants.TRANSMIT_DEFAULT,
Constants.GAIN_DEFAULT, Constants.SENSITIVITY_DEFAULT,
Constants.THRESHOLD_DEFAULT, Constants.TEMPERATURE_DEFAULT,
Constants.TEMPERATURE_FACTOR_DEFAULT,
Constants.AMBIENT_NOISE_DEFAULT);
            createNode((Integer) id, cogPhyField, loc, info);
        }

        return cogPhyField;
    }

    public static void main(String[] args) throws IOException {
        if (args.length < 2) {
            System.out.println("syntax: Main <length_of_grid> <nr_nodes>");
            System.out.println("    eg: Main 100 20");
            return;
        }

        MainSim umaSim = new MainSim();
        umaSim.init(args[0], args[1]);
    }

    public void init(String comprimento, String nos) {

        int tamanho = Integer.parseInt(comprimento);
        int nodes = Integer.parseInt(nos);

//        System.out.println("Tamanho da Grid = " + tamanho + " x " + tamanho);
//        System.out.println("Creating simulation nodes... ");
        final CogPhyField cogPhyField = createSim(tamanho, nodes);

        // Executando a simulação
        Hashtable<Integer, RadioData> radios = cogPhyField.getRadioIdRadioDataMap();
```

```
        Vector<Signal> signalList = new Vector<Signal>();
        RadioData data = new RadioData();

        signalList = generator.getSignalList();
        // Para considerar os tempos entre 1 e 5

        int time = 0;



        // itera os rádios, sendo que o primeiro é o rádio 1, e assim segue...
        for (int z = 1; z <= radios.size(); z++) {
            data = (RadioData) radios.get(z);
            time = (int) (1 + Math.random( ) * 5);
            // manda cada um deles transmitir a cada time os sinais que estiverem no
signalList
            for (int j = 0; j < signalList.size(); j++) {
                data.radioEntity.transmit(signalList.get(j), time, data.info.getChannel());
            }

        }
    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
***/


package br.com.datacoffee.cognet.tools;

import br.com.datacoffee.cognet.phy.Signal;
import java.util.logging.FileHandler;
import java.util.logging.Logger;
import java.util.logging.XMLFormatter;

/**
 *
 * @author everton
 */
public class Logging {

    Logger logger;
    protected int radioId;
    protected Signal sendSignal;
```

```
    protected Signal receiveSignal;
    protected int sendTime;
    protected int receiveTime;
    protected double sendPower;
    protected double receivePower;

    public Logging() {
        logger = Logger.getLogger("CogNetLogging");
        FileHandler handler = null;
        try {
            boolean append = true;
            //handler = new FileHandler("/home/everton/particular/ufsc/tcc/simulacao-
testes/CogNet/src/br/com/datacoffee/cognet/sim/logs/logging.log", append);
            handler = new FileHandler("./src/br/com/datacoffee/cognet/sim/logs/logging.log",
append);
            handler.setFormatter(new XMLFormatter());
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        logger.addHandler(handler);
    }


    public void log(String log) {
        logger.info(log);
    }
}
```

```
/***
Este software é uma adaptação baseada nos códigos implementados no simulador
Scalable Wireless Ad hoc Network Simulator, SWANS - respeitando sua licensa, conforme
trecho abaixo. Os códigos utilizados foram modificados e adaptados conforme a
necessidade explicitada no escopo do trabalho de conclusão de curso intitulado "Um
Modelo de Simulação da Camada Física para Rádios Cognitivos", e estas alterações
ocorreram durante o período do seu desenvolvimento até sua conclusão, em maio de
2010.

" 5. AO LICENCIADO é concedida a permissão para fazer o download, compilar, executar,
copiar e modificar o software para fins acadêmicos e não comerciais desde que este aviso
acompanhe todas as cópias do software. Cópias do software modificadas podem ser
distribuídas para fins não comerciais e acadêmicos desde que: a) Este aviso acompanhe
as todas cópias; b) Que as cópias levem avisos que afirmem que o software foi alterado;
c) a data de todas as mudanças, claramente identificadas no software. "
 ***/
package br.com.datacoffee.cognet.tools;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Random;
import java.text.DecimalFormat;
import java.util.Iterator;
import java.util.TreeSet;
import java.util.logging.Level;
import java.util.logging.Logger;

import br.com.datacoffee.cognet.phy.Bit;
import br.com.datacoffee.cognet.phy.Pulse;
import br.com.datacoffee.cognet.phy.Signal;
import br.com.datacoffee.cognet.Constants;
import java.util.Hashtable;
import java.util.Vector;
```

```
/*
 * Gera uma List com Sinais
 */
public class SignalGenerator {

    private Vector<Signal> signalList;

    public SignalGenerator() {
    }

    /**
     * @param args
     * @throws IOException
     */
    public static void main(final String[] args) throws IOException {
        SignalGenerator generator = new SignalGenerator();
        generator.init();
    }

    public void init() {
        Hashtable<String, String> parameters = new Hashtable<String, String>();

        final File serialSignalFile = new
File("./src/br/com/datacoffee/cognet/sim/logs/signals.txt");
        /* Número de sinais, conforme cada tipo de duração: 5, 10, 15, 20, 25 times */
        final int numberOfSignals = 5;
        /*o signal pode ter bits distintos, definiu-se as seguintes combinações possíveis
        H-L-NS, PU-NS, H-NS, L-NS, H-L */
        final int numberOfSignalTypes = 5;
        //Potência default
        final double power = Constants.TRANSMIT_DEFAULT;
        /*Existem 5 Bits possíveis : L-U-H-NS-PU*/
```

```
        final int numberOfBits = 5;

        setSignalList(this.generateSignals(numberOfSignals, numberOfSignalTypes,
numberOfBits, power));
//DEBUG
//        for (Signal signal : signalList) {
//            Iterator iterator = (signal.getPulses()).iterator();
//            while (iterator.hasNext()) {
//                Pulse p = (Pulse) iterator.next();
//                System.out.println(p.getStartMoment() + "-" + p.getStopMoment() + "-" +
p.getBit() + "-" + p.getPower());
//            }
//
//        }
        try {
            this.saveSignalFile(getSignalList(), serialSignalFile);
        } catch (IOException ex) {
            Logger.getLogger(SignalGenerator.class.getName()).log(Level.SEVERE, null, ex);
        }


    }

    private Vector<Signal> generateSignals(
            int numberOfSignals,
            int numberOfSignalTypes,
            int numberOfBits,
            double power) {

        final Vector<Signal> retorno = new Vector<Signal>();
        Signal signal;
        int duracao = 0;
        double stopTime = 0.0;
```

```
    // para cada número de sinal - 5,10,15,20,25,30
    for (int i = 1; i <= numberOfSignals; i++) {
        duracao += i;
        // cria um novo signal com um conjunto de pulses
        //for (int bit = 1; bit <= numberOfSignalTypes; bit++) {
            signal = new Signal(this.generatePulses(i, duracao), Signal.Modulation.BPSK);
            retorno.add(signal);
        //}
    }

    return retorno;
}

private TreeSet<Pulse> generatePulses(int signalType, int duracao) {
    /*
    public static final int NOSIGNAL = 0;
    public static final int HIGH = 1;
    public static final int LOW = 2;
    public static final int UNKNOW = 3; -> Não considerado no sinal gerado,
    mas sim para sobreposição
    public static final int PU = 4;
     *
     * possibilidades: H-L-NS, PU, H, L, H-L
     */
    TreeSet<Pulse> pulses = new TreeSet<Pulse>();
    int[] bits;
    switch (signalType) {

        case 1:
            bits = new int[]{0, 1, 2};
            pulses = generatePulses(bits, duracao);
            break;
        case 2:
            bits = new int[]{4};
```

```
            pulses = generatePulses(bits, duracao);
            break;
        case 3:
            bits = new int[]{1};
            pulses = generatePulses(bits, duracao);
            break;
        case 4:
            bits = new int[]{2};
            pulses = generatePulses(bits, duracao);
            break;
        case 5:
            bits = new int[]{1, 2};
            pulses = generatePulses(bits, duracao);
            break;
    }
    return pulses;
}

private TreeSet<Pulse> generatePulses(int[] bitTypes, int duracao) {
    DecimalFormat df = new DecimalFormat("#.####");
    TreeSet<Pulse> retorno = new TreeSet<Pulse>();
    Bit umBit = new Bit();
    double start = 1.0;
    //start = Double.valueOf(df.format((double) duracao / 1000));
    double startAux = start + (0.0001);

    double stop = Double.valueOf(df.format((double) (duracao+1) - (0.0001)));

    double range = 0.0;
    double random = 0.0;

    Random rand = new Random();

    while (true) {
```

```
    range = Double.valueOf(df.format((double) (stop - startAux)));

    Pulse umPulse = null;
    random = Double.valueOf(df.format(((rand.nextDouble() * range) + startAux)));
    if (random == stop) {
        umPulse = new Pulse(start, random, bitTypes[rand.nextInt(bitTypes.length)]);
        retorno.add(umPulse);
        break;
    }

    // quando ocorre 57 58 59 e o random deu 58
    if (random == (stop - 0.0001) && random != startAux) {
        random = Double.valueOf(df.format(random - 0.0001));
    } else if (random == (stop - 0.0001) && random == startAux) {
        umPulse = new Pulse(start, stop, bitTypes[rand.nextInt(bitTypes.length)]);
        retorno.add(umPulse);
        break;
    }

    umPulse = new Pulse(start, random, bitTypes[rand.nextInt(bitTypes.length)]);
    retorno.add(umPulse);

    start = Double.valueOf(df.format((double) (random + 0.0001)));
    startAux = Double.valueOf(df.format((double) (start + 0.0001)));

    if (startAux == stop) {
        umPulse = new Pulse(start, startAux, bitTypes[rand.nextInt(bitTypes.length)]);
        retorno.add(umPulse);
        break;
    }
}

// Potência de cada Pulse
if (!retorno.isEmpty()) {
```

```
        //Calculo da potência de cada pulse, proporcional ao tempo dos Pulses do Signal

        Iterator iterator = retorno.iterator();
        double totalTimeOfPulse = 0.0;
        while (iterator.hasNext()) {
            Pulse p = (Pulse) iterator.next();
            if (p.getBit() != 0) {
                totalTimeOfPulse += Double.valueOf(df.format(p.getStopMoment() -
p.getStartMoment()));
            }
        }

        double averagePower =
Double.valueOf(df.format(Constants.TRANSMIT_DEFAULT / totalTimeOfPulse));

        iterator = retorno.iterator();
        while (iterator.hasNext()) {
            Pulse p = (Pulse) iterator.next();
            double pulseIntervalTime = Double.valueOf(df.format(p.getStopMoment() -
p.getStartMoment()));

            double potencia = 0.0;
            // Se o Bit for NOSIGNAL, não há potência
            if (p.getBit() != 0) {
                potencia = Double.valueOf(df.format(pulseIntervalTime * averagePower));
            }
            p.setPower(potencia);
        }
    }
    // se o random = stop, então soma 0.0001, cria o pulse e sai;
    // dividir o power proporcionalmente
    return retorno;
  }
```

```
    private void saveSignalFile(Vector<Signal> signalList, File file) throws IOException {
        final FileOutputStream fos;
        try {
            fos = new FileOutputStream(file);
            final String lineSeparator = System.getProperty("line.separator");

            for (final Signal signal : signalList) {
                fos.write(signal.toString().getBytes());
                fos.write(lineSeparator.getBytes());
            }
            fos.close();
        } catch (IOException ex) {
            Logger.getLogger(SignalGenerator.class.getName()).log(Level.SEVERE, null, ex);
        } catch (Exception ex) {
            Logger.getLogger(SignalGenerator.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * @return the signalList
     */
    public Vector<Signal> getSignalList() {
        return signalList;
    }

    /**
     * @param signalList the signalList to set
     */
    public void setSignalList(Vector<Signal> signalList) {
        this.signalList = signalList;
    }
}
```