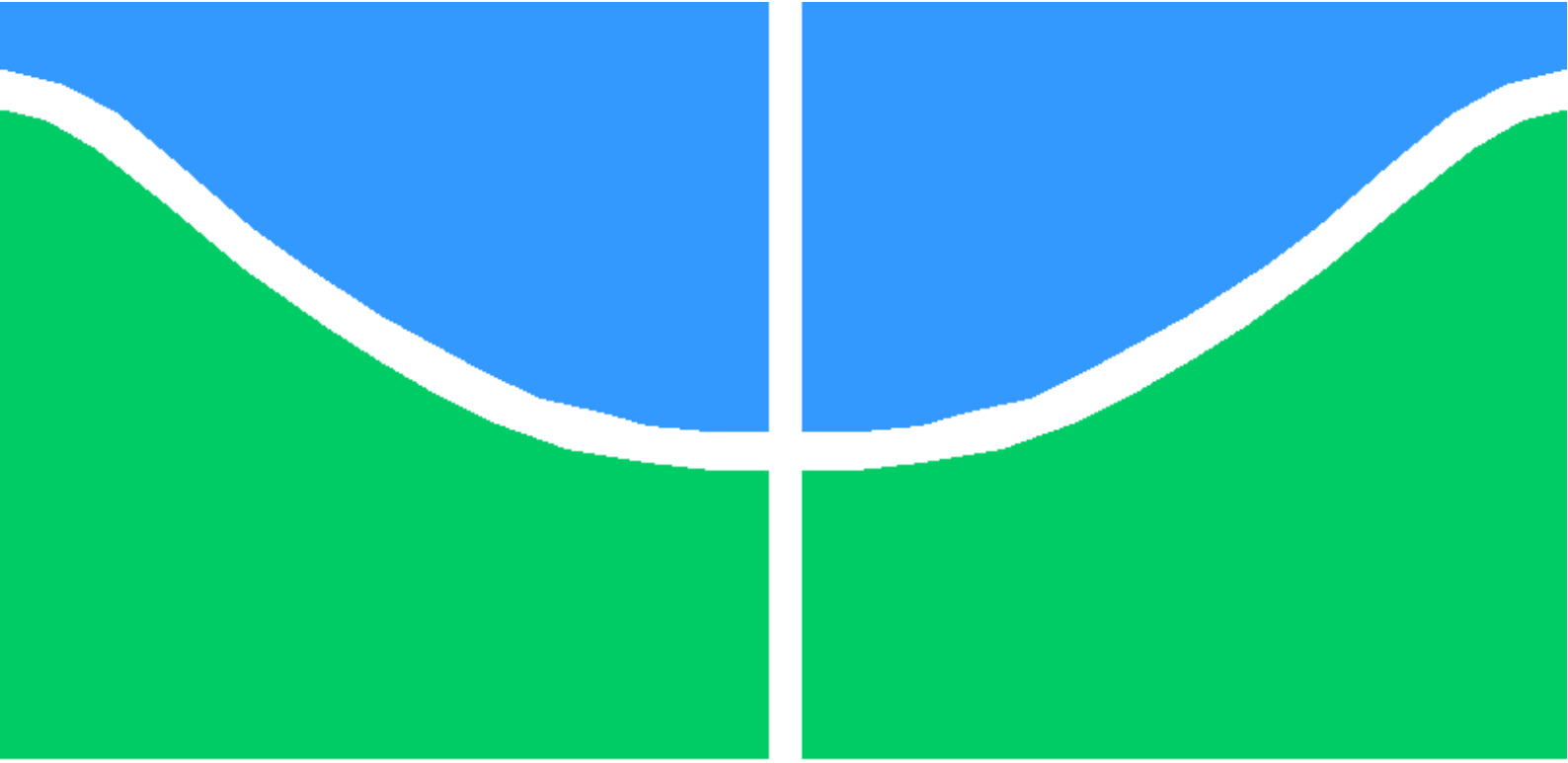

TRABALHO DE GRADUAÇÃO

# IMPLEMENTAÇÃO E SIMULAÇÃO DE UM PROTOCOLO DE CONTROLE DE ACESSO AO MEIO PARA REDES SEM FIO *AD HOC* COGNITIVAS

**Amanda Marques da Silva**

**Pedro Henrique de Mesquita**

**Brasília, fevereiro de 2011**

**UNIVERSIDADE DE BRASÍLIA**

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

# IMPLEMENTAÇÃO E SIMULAÇÃO DE UM PROTOCOLO DE CONTROLE DE ACESSO AO MEIO PARA REDES SEM FIO *AD HOC* COGNITIVAS

**Amanda Marques da Silva**

**Pedro Henrique de Mesquita**

*Relatório submetido ao Departamento de Engenharia*
*Elétrica como requisito parcial para obtenção*
*do grau de Engenheiro de Redes de Comunicação*

Banca Examinadora

Prof. Marcelo Menezes de Carvalho, ENE/UnB _____
*Orientador*

Prof. Renato Mariz de Moraes, ENE/UnB _____
*Examinador interno*

Prof. Luiz A. DaSilva, Virginia Tech _____
*Examinador externo*

**Dedicatórias**

*Àqueles que me deram suporte durante a execução deste trabalho.*

*Pedro Henrique de Mesquita*

*A todos que de alguma forma contribuíram para a realização deste trabalho. Muito obrigada.*

*Amanda Marques da Silva*

## Agradecimentos

*Primeiramente, a Deus, sem Ele chegar até aqui não seria possível. Aos meus pais, Valdir e Susana, que permaneceram ao meu lado quando já não havia mais ninguém. Ao meu irmão Rafael, cuja ajuda chegou nos momentos certos. A minha irmã Flávia, cujo auxílio e paciência sempre estiveram presentes. Valeu, família. Agradeço aos amigos que estiveram (sofreram) comigo nos últimos cinco anos. Um agradecimento especial à Larissa, Pedro e Priscilla, companhias sempre presentes. Obrigada pelos dias inesquecíveis e boas risadas. Finalmente, agradeço ao meu orientador, Prof. Marcelo, por nos aceitar como alunos mesmo sem nos conhecer (espero que não tenha se arrependido!).*

*Amanda Marques da Silva*

*À minha família que me apoiou durante toda esta jornada. Aos meus amigos que estiveram comigo nos últimos anos.*

*Pedro Henrique de Mesquita*

## RESUMO

A demanda pelo uso mais racional do espectro de radiofrequência tornou-se cada vez maior com o aumento significativo de aplicações e tecnologias de comunicação sem fio observado nos últimos anos. Recentemente, com o advento de transceptores de rádio capazes de identificar faixas de frequência ociosas no tempo e no espaço — os chamados rádios cognitivos — tornou-se possível o projeto de técnicas para acesso dinâmico e oportunista ao espectro. A partir dessa técnica, o presente trabalho descreve um novo protocolo para controle de acesso ao meio para redes sem fio do tipo *ad hoc*. O protocolo proposto baseia-se em modificações no mecanismo DCF do padrão IEEE 802.11 para permitir que os dispositivos troquem informações sobre a disponibilidade de faixas do espectro. De posse destas informações, o protocolo permite a transmissão oportunista de quadros em faixas de frequência ociosas, concomitantemente com os quadros transmitidos através dos canais-padrão do próprio IEEE 802.11. Para avaliação do desempenho, apresentamos a implementação do protocolo no simulador de redes *network simulator 3*. Os resultados obtidos mostram que é possível praticamente dobrar a vazão da rede. Além disso, o protocolo é simples de ser implantado no paradigma atual, uma vez que usufrui de grande parte da tecnologia legada (IEEE 802.11).



## ABSTRACT

The demand for a more rational use of the radio frequency spectrum has become increasingly higher with the significant augment of wireless communication applications and technologies in the past few years. Recently, with the emergence of radio transceivers that are capable of sensing idle frequency bands in time and space — the so-called cognitive radios — it has now become possible to design techniques for dynamic and opportunistic spectrum access. Based on these techniques, this work describes a new protocol for medium access control for ad hoc wireless networks. The proposed protocol is based on a modification of the IEEE 802.11 DCF mechanism that allows devices to exchange information about the availability of spectrum bands. Once they have this information, the protocol allows the opportunistic transmission of frames over the idle frequency bands simultaneously with frames transmissions over the IEEE 802.11 standard channels. For performance evaluation, we implemented the protocol in the *network simulator 3*. Our results show that it is possible to practically double the network's throughput. In addition to throughput gains, our protocol is simple to deploy under the current paradigm, since it takes advantage of legacy technology (IEEE 802.11).

# SUMÁRIO


**1 Introdução ..... 1**
1.1 Descrição do trabalho ..... 2
1.2 Objetivos do projeto ..... 2
1.3 Contribuições do Projeto ..... 2
1.4 Apresentação do manuscrito ..... 3

**2 Revisão Bibliográfica ..... 4**

**3 O Protocolo ..... 7**
3.1 Proposta ..... 7
3.2 Descrição do Protocolo ..... 7

**4 Software NS-3 ..... 14**

**5 Implementação no Simulador NS-3 ..... 19**

**6 Avaliação de Desempenho ..... 23**

**7 Conclusões ..... 34**

**REFERÊNCIAS BIBLIOGRÁFICAS ..... 36**

**Anexos ..... 37**

**I Modelo de Simulação ..... 38**

**II Módulo de Envio ..... 44**
II.1 dispatch-module.h ..... 44
II.2 dispatch-module.cc ..... 45

**III Ipv4 L3 Protocol ..... 49**
III.1 ipv4-l3-protocol.h ..... 49
III.2 ipv4-l3-protocol.cc ..... 49

**IV Ipv4 Interface ..... 54**
IV.1 ipv4-interface.h ..... 54
IV.2 ipv4-interface.cc ..... 54

**V Wifi Net Device ..... 55**
V.1 wifi-net-device.h ..... 55
V.2 wifi-net-device.cc ..... 55
V.3 net-device.h ..... 55

**VI Adhoc Wifi Mac ..... 56**
VI.1 adhoc-wifi-mac.h ..... 56
VI.2 adhoc-wifi-mac.cc ..... 56
VI.3 wifi-mac.h ..... 56

**VII Dca Txop ..... 57**
VII.1 dca-txop.h ..... 57
VII.2 dca-txop.cc ..... 57

**VIII Mac Low ..... 58**
VIII.1 mac-low.h ..... 58
VIII.2 mac-low.cc ..... 58

**IX Wifi Mac Header ..... 63**
IX.1 wifi-mac-header.h ..... 63
IX.2 wifi-mac-header.cc ..... 63

**X Dados Coletados ..... 67**
X.1 Topologia de 10 nós ..... 67
X.1.1 Probabilidade de 0% ..... 67
X.1.2 Probabilidade de 10% ..... 67
X.1.3 Probabilidade de 40% ..... 67
X.1.4 Probabilidade de 70% ..... 68
X.2 Topologia de 40 nós ..... 68
X.2.1 Probabilidade de 0% ..... 68
X.2.2 Probabilidade de 10% ..... 68
X.2.3 Probabilidade de 40% ..... 68
X.2.4 Probabilidade de 70% ..... 69

# LISTA DE FIGURAS


3.1 Pré-fila distribui os pacotes entre as interfaces do nó. ..... 8
3.2 Nova estrutura das camadas do dispositivo destacando-se o local de inserção do módulo de envio e a presença de múltiplas interfaces. ..... 9
3.3 Operação do novo protocolo quando a negocição para uso do canal cognitivo é bem-sucedida. Neste caso, assim que o transmissor recebe a confirmação de disponibilidade do canal 6, ocorre a transmissão de dados nos dois canais. ..... 12
3.4 Operação do novo protocolo quando o receptor não concorda quanto ao canal disponível. ..... 12
3.5 Operação do novo protocolo quando a negocição para uso do canal cognitivo é bem-sucedida, mas o pacote não é recebido pela interface cognitiva, conforme indica a linha tracejada. ..... 13
3.6 Fluxograma do funcionamento do protocolo. ..... 13

4.1 Estrutura do NS-3 ..... 14
4.2 Estrutura do Dispositivo de Rede *Ad hoc* ..... 16
4.3 Caminho do pacote no envio ..... 17
4.4 Caminho do pacote na recepção ..... 18

5.1 Manipulações para obter o endereço IP cognitivo do receptor. ..... 21

6.1 Sobreposição dos canais estipulados pelo padrão IEEE 802.11. ..... 23
6.2 Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 10% para uma rede de 10 nós. ..... 26
6.3 Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 40% para uma rede de 10 nós. ..... 27
6.4 Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 70% para uma rede de 10 nós. ..... 28
6.5 Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 10%, 40% e 70% para uma rede de 10 nós. ..... 29
6.6 Vazão média dos nós na rede versus número de canais totais com probabilidade de ausência do UP de 10%, 40% e 70% para uma rede de 10 nós. ..... 30
6.7 Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 10% para uma rede de 40 nós. ..... 31
6.8 Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 40% para uma rede de 40 nós. ..... 31

6.9 Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 70% para uma rede de 40 nós. ..... 32
6.10 Vazão agregada versus número de canais totais com probabilidade de ausência do PU de 10%, 40% e 70% para uma rede de 40 nós. ..... 32
6.11 Vazão média dos nós na rede versus número de canais totais com probabilidade de ausência do PU de 10%, 40% e 70% para uma rede de 40 nós. ..... 33

# LISTA DE TABELAS


6.1 Parâmetros da camada MAC e física ..... 25
6.2 Vazão com e sem o módulo de envio para 10 nós ..... 25
6.3 Vazão com e sem o módulo de envio para 40 nós ..... 26
6.4 Comparação entre as vazões esperada e obtida para um canal cognitivo e topologia de 10 nós. ..... 27
6.5 Comparação do ganho na vazão para diferentes probabilidades e quantidades de canais para um topologia de 10 nós. ..... 28
6.6 Comparação entre as vazões esperada e obtida para um canal cognitivo e topologia de 40 nós. ..... 29
6.7 Comparação do ganho na vazão para diferentes probabilidades e quantidades de canais para um topologia de 40 nós. ..... 30

X.1 Vazões agregadas coletadas para $P_d = 0\%$ e 10 nós ..... 67
X.2 Vazões agregadas coletadas para $P_d = 10\%$ e 10 nós ..... 67
X.3 Vazões agregadas coletadas para $P_d = 40\%$ e 10 nós ..... 67
X.4 Vazões agregadas coletadas para $P_d = 70\%$ e 10 nós ..... 68
X.5 Vazões agregadas coletadas para $P_d = 0\%$ e 40 nós ..... 68
X.6 Vazões agregadas coletadas para $P_d = 10\%$ e 40 nós ..... 68
X.7 Vazões agregadas coletadas para $P_d = 40\%$ e 40 nós ..... 69
X.8 Vazões agregadas coletadas para $P_d = 70\%$ e 40 nós ..... 69

# LISTA DE ABREVIATURAS

### Acrônimos

| | |
|---|---|
| ACK | Acknowledgment |
| AP | Access Point |
| ARP | Address Resolution Protocol |
| API | Application Programming Interface |
| CAC | Confirm Available Channel |
| CMA | Confirm MAC Address |
| CTS | Clear-To-Send |
| CSMA/CA | Carrier Sense Multiple Access/Collision Avoidance |
| DCF | Distributed Control Function |
| EDCAF | Enhanced Distributed Channel Access Funtion |
| FCC | Federal Communication Commission |
| ISM | Industry, Scientific and Medical |
| MAC | Medium Access Control |
| NS-3 | Network Simulator Versão 3 |
| OPACK | Opportunistic Acknowledgment |
| P2P | Point to Point |
| PCF | Point Coordination Function |
| PIFS | PCF Interframe Space |
| QoS | Quality-of-Service |
| RAC | Request Available Channel |
| RTS | Request-To-Send |
| SIFS | Short Interframe Space |
| SN | Sequence Number |
| UP | Usuário Primário |
| US | Usuário Secundário |

### Abreviações

| | |
|---|---|
| rx | Receptor |
| tx | Transmissor |

### Subscritos

| | |
|---|---|
| std | Standard (Padrão) |
| cog | Cognitivo |

# Capítulo 1

# Introdução

A demanda pelo acesso ao espectro está, a cada dia, maior: seja devido à criação de novos serviços ou pelo aumento do número de usuários nos sistemas já estabelecidos. As políticas de alocação fixa de espectro são outro fator limitador do uso eficiente do espectro. No momento, grande parte do espectro já foi atribuído, tornando difícil encontrar uma porção ainda não alocada para novos serviços ou para expansão dos serviços já existentes. Ao mesmo tempo, as partes já licenciadas são subutilizadas, conforme pode ser visto no estudo feito pela *Federal Communications Commission* [1]. Neste estudo, mostra-se que o maior problema não é a indisponibilidade do espectro, e sim a subutilização do espectro já alocado.

Uma proposta que procura resolver tal questão é a técnica dos rádios cognitivos. Esta técnica procura fazer um acesso dinâmico ao espectro usando-o de forma oportunista, evitando ao máximo interferências aos usuários legítimos daquela porção do espectro. Mais especificamente, permite aos seus usuários monitorar o espectro à procura de espaços ociosos de frequência e utilizá-los de forma oportunista. Uma vez encontradas as oportunidades, os dispositivos devem configurar os parâmetros da sua interface, tais como: frequência de transmissão, potência, taxa de transmissão, tipo de modulação, etc., para fazer um uso eficiente do canal encontrado. Caso seja detectado a presença de um usuário primário, o canal deve ser, então, liberado.

No âmbito de redes *ad hoc* [8], nas quais os nós não possuem uma unidade de controle central, é necessário que os nós se organizem de modo a gerenciar o acesso aos recursos. Os protocolos cognitivos de controle de acesso ao meio (MAC, do inglês *Medium Access Control*) são importantes nesta tarefa de gerenciamento do acesso aos recursos. Além disso, eles desempenham um importante papel na detecção de usuários primários e exploração das bandas para acesso oportunista. Sobretudo, o protocolo deve decidir a melhor forma de se fazer o sensoriamento do espectro, de modo a evitar o desperdício de recursos e interferência na comunicação dos usuários primários.

É neste contexto que apresentamos, a seguir, a proposta de um novo protocolo para controle de acesso ao meio para redes cognitivas.

## 1.1 Descrição do trabalho

Este trabalho apresenta um novo protocolo de controle de acesso ao meio para redes *ad hoc* que está baseado em modificações do padrão IEEE 802.11 [10]. O protocolo faz uso das técnicas de rádio cognitivo para acesso oportunista a outras faixas de frequência. Como aproveita os mecanismos de controle do IEEE 802.11, o protocolo não utiliza canais extras para troca de informações de controle.

Os dispositivos que utilizam esse novo protocolo utilizam dois rádios. Um rádio é utilizado para o uso regular da banda ISM, seguindo o mecanismo padrão de disputa pelo acesso ao canal do IEEE 802.11. O segundo rádio é utilizado para transmissões oportunistas em outras faixas de frequência. Por fazerem uso regular da banda ISM, os usuários deste novo protocolo não dependem da disponibilidade de espectro livre em outras faixas de frequência para seu funcionamento, ou seja, mesmo não havendo espaços ociosos nas outras faixas de frequência, ainda há transmissão na banda ISM.

Este protocolo não necessita de um canal extra de controle, pois as negociações de transmissões oportunistas ocorrem durante a troca de quadros RTS/CTS. O nó que adquire o direito de utilizar o canal padrão para sua transmissão, ganha, automaticamente, o direito de tentar uma transmissão pelo módulo cognitivo.

Uma das características do protocolo proposto é a simplicidade em sua implementação, pois o mesmo baseia-se nas tecnologias amplamente utilizadas do padrão IEEE 802.11. Outra característica interessante, é que o protocolo não especifica a banda utilizada para transmissões oportunistas, por isso pode ser utilizado como alternativa para uso dinâmico dos canais extras definidos no próprio padrão IEEE 802.11.

## 1.2 Objetivos do projeto

Este projeto tem como objetivo especificar o protocolo proposto, implementá-lo em um simulador computacional de redes de computadores e avaliar o desempenho desse novo protocolo para redes sem fio do tipo *ad hoc*. Deseja-se analisar, por meio de simulações, o ganho na vazão quando comparamos tal protocolo com o padrão IEEE 802.11.

## 1.3 Contribuições do Projeto

Entre as principais contribuições deste trabalho, destacam-se:

- Especificação de um novo protocolo para controle de acesso ao meio (MAC) para uso oportunista de canais em outras bandas;
- Desenvolvimento de um módulo no simulador de redes NS-3;
- Avaliação de desempenho do protocolo via simulações computacionais.

## 1.4 Apresentação do manuscrito

O trabalho se organiza de maneira a permitir ao leitor acompanhar como se deu a evolução do desenvolvimento do projeto. No Capítulo 2, abordam-se algumas propostas de protocolos MAC que fazem uso da técnica de rádios cognitvos. Posteriormente, no Capítulo 3, descreve-se o funcionamente do novo protocolo que procura aprimorar as propostas anteriormente mencionadas.A seguir, no Capítulo 4, apresenta-se um estudo da arquitetura do software utilizado para a implementação e simulação deste novo protocolo. No Capítulo 5, é explicado, detalhadamente, quais foram as modificações no simulador necessárias para a implementação do protocolo, baseando-se na proposta e nas características e limitações do simulador utilizado. No Capítulo 6, é feita a avaliação de desempenho do protocolo desenvolvido através da execução de simulações e coleta de dados para diferentes cenários e topologia. Finalmente, no Capítulo 7, é feito um breve resumo do que foi apresentado e conclusões a respeito do que foi obtido. Mais ainda, são sugeridas possibilidades de pesquisa para trabalhos futuros.

# Capítulo 2

# Revisão Bibliográfica

Trabalhos sobre o tópico de rádios cognitivos têm sido desenvolvidos recentemente [9]. Para se ter uma ideia, a primeira conferência acerca do tema foi a DySPAN (do inglês, *Dynamic Spectrum Access Networks*) em 2005. Uma das abordagens refere-se a propostas de protocolos para controle de acesso ao meio (MAC) para redes cognitivas.

Bahl et al. [2], propõem um novo sistema para redes sem fio, baseado no *Wi-Fi*, que opera sobre faixas de frequência do espectro UHF. Esse sistema se aproveita das regras estabelecidas pela *Federal Communications Commission* (FCC), que permite o uso de faixas de frequência na banda alocada para TV, desde que não haja interferência nos usuários legítimos dessa banda. O sistema, denominado *WhiteFi*, incorpora um novo algoritmo adaptativo de alocação de espectro, que lida com a variação espacial e fragmentação do espectro, e propõe um protocolo com pouca informação extra que trata a variação temporal do espectro. Os canais de UHF possuem uma banda estreita. Por isso, o sistema procura agregar canais contínuos de forma a aumentar a vazão da rede. Isto causa uma variação da largura de banda dos canais utilizados.

O sistema *WhiteFi* se foca, primariamente, na questão de configuração de uma rede do tipo *Wi-Fi* constituída de um ponto de acesso (AP) e suas múltiplas estações-clientes associadas. A configuração adaptativa opera baseada em três contribuições:

- Um novo algoritmo de alocação de espectro para gerenciar bandas de largura variável.
- Um novo mecanismo de descoberta dos APs. A descoberta é facilitada pelo uso de uma nova técnica, chamada *Signal Interpretation before Fourier Transform* (SIFT), que analiza os sinais no domínio do tempo para detectar transmissões em diferentes larguras de faixa.
- Um novo método para lidar com desconexões, ocasionada pela variação temporal do espectro.

O sistema mostra-se promissor. Entretanto alguns fatos e limitações devem ser melhor discutidos. O sistema apresentado não considera os efeitos de inteferência entre APs, ou seja, durante a procura de canais em potencial, um AP não se preocupa com a sobreposição, parcial ou completa, de canais ocupados por outros APs. Além disso, o sistema faz uso de rádios muito sofisticados e de tamanho incompatível com dispositivos portáteis, tais como, uma antena UHF, um processador, um scanner e um conversor de sinal UHF.

O *WhiteFi* utiliza recursos extras para seu funcionamento. Ele faz uso de um segundo rádio para mapeamento do espectro, aferição do tempo de utilização e detecção de sinais de controle durante conexões ativas. Além disso, usa um canal extra para impedir interfência nos UPs e recuperar conexões perdidas. Esse canal extra é um canal UHF, que é usado pra substituir o canal de controle usual quando este passa a ser usado pelo UP. Entretanto, é difícil garantir sua disponibilidade, pois o mesmo também pode estar ocupado. O *WhiteFi* ainda utiliza uma entidade central para coordenar os nós da rede.

Hsu et al. [3], descrevem o *Statistical Channnel Allocation* MAC (SCA-MAC), novo protocolo para redes *ad hoc* cogntivas no qual o controle da rede é distribuído entre os nós. Diferentemente do *WhiteFi*, o SCA-MAC considera a sobreposição de transmissões entre os usuários secundários como colisão. O SCA-MAC é um protocolo baseado no CSMA/CA que explora estatísticas de uso do espectro para decidir o acesso ao canal. Para cada transmissão, o emissor negocia com o receptor os parâmetros da transmissão em um canal de controle extra.

O SCA-MAC consiste de três operações principais: sensoriamento e aprendizagem do ambiente; troca de quadros de CRTS/CCTS em um canal de controle extra; transmissão de dados e ACKs em um canal de dados. O protocolo ainda permite agregação de canais para aumentar a taxa de transmissão.

Os dispositivos cognitivos são dotados de inteligência para sensoriamento e coleta de estatísticas de uso do espectro. Baseado nessas estatísticas, a probabilidade de sucesso na transmissão pode ser aumentada e a probabilidade de interferência nos usuários licenciados pode ser reduzida. Para controlar a interferência no UP, o SCA-MAC pode avaliar seu impacto, ou seja, ele pode predizer a taxa de sucesso baseado no tamanho do pacote e nas estatísticas coletadas, de forma a decidir entre as alternativas possíveis. Os pacotes de controle carregam a informação de agregação de canal e tamanho de pacote.

Apesar dos bons resultados apresentados, o protocolo necessita de um canal de controle dedicado para seu funcionamento. A existência de um canal de controle dedicado não pode deixar de ser vista como uma falta de eficácia no uso do espectro, e isso também leva ao conhecido problema de saturação do canal de controle. Além disso, o protocolo utiliza bastante informação extra de controle ao fazer várias negociações para acesso ao canal de dados.

O *Dynamic Open Spectrum Sharing* (DOSS) [4] também é um protocolo para a camada MAC para redes sem fio *ad hoc*. O protocolo utiliza três bandas de frequência para seu funcionamento: uma banda para seus canais de controle; uma para transmissão de dados; e outra para uso de um "sinal ocupado". Na banda de controle são trocadas informações dos parâmetros do canal, tais como, frequências centrais e largura de banda. A banda de "sinal ocupado" é utilizada para sinalizar a ocupação de um determinado canal de dados, com o intuito de se evitar o problema do terminal escondido e do terminal exposto.

Alguns fatos devem ser levantados acerca do protocolo em questão. Primeiramente, os nós no DOSS são apenas usuários secundários do espectro. Portanto se nenhuma oportunidade de espectro for encontrada, o sistema deixa de funcionar. O sistema faz uso de múltiplas bandas para seus canais de controle, além de uma banda dedicada para o "sinal ocupado". O protocolo utiliza vários canais de controle para evitar o problema da saturação. Entretanto ele não especifica como um

nó da rede pode descobrir o canal de controle no qual seu par de comunicação está monitorando. O sistema requer o uso de múltiplos rádios, sendo pelo menos dois: um para a transmissão do "sinal ocupado" e outro para dados e controle, sendo que o segundo deve ser mais sofisticado para a transmissão em duas bandas distintas.

O primeiro esforço internacional de padronização para uso de rádios cognitivos é o padrão IEEE 802.22 [5] . A estrutura do IEEE 802.22 é composta por uma estação base para acesso e compartilhamento do espectro. A estação base gerencia sua célula e suas estações-clientes associadas. O padrão opera em intervalos de tempo e utiliza uma estrutura de quadros. No início de cada quadro, há um cabeçalho de controle que é utilizado para informar aos usuários cognitivos a disponibilidade dos canais, as larguras de banda suportadas, o tempo de acesso ao espectro, etc.

Um ponto a se destacar a respeito do IEEE 802.22 é a necessidade da presença de uma entidade de controle central que impede o seu uso em redes *ad hoc*. Mesmo que não houvesse a presença dessa unidade de controle, o protocolo exige sincronismo. Todavia, garantir sincronismo entre as estações-clientes de uma rede *ad hoc* é extremamente complicado.

Apesar dos avanços, várias limitações são ainda identificadas, especialmente quanto ao uso de vários canais para controle, acesso centralizado em muitos protocolos e necessidade de implementação de um novo protocolo sem herdar o legado, por exemplo, do IEEE 802.11. Neste trabalho, apresentaremos uma solução que procura resolver o problema de acesso ao meio para redes sem fio *ad hoc*. A solução baseia-se em uma adaptação do padrão do IEEE 802.11.

# Capítulo 3

# O Protocolo

## 3.1 Proposta

Neste capítulo apresentamos a descrição de um novo protocolo de controle de acesso ao meio (MAC) para uso em redes sem fio do tipo *ad hoc*. A ideia principal do protocolo proposto é aumentar a vazão de redes sem fio do tipo *ad hoc* a partir de modificações simples da função de coordenação distribuída (DCF) da norma IEEE 802.11. Com as modificações propostas, é possível a transmissão oportunista de quadros em faixas de espectro ociosas concomitantemente com os quadros transmitidos através dos canais-padrão do próprio IEEE 802.11. O padrão IEEE 802.11 foi escolhido para ser combinado com a técnica de rádios cognitivos pois se trata de uma tecnologia com grande penetração no mercado. O protocolo proposto pode ser aplicado para redes estruturadas, pois mesmo essas fazem uso do mecanismo DCF do IEEE 802.11.

## 3.2 Descrição do Protocolo

No esquema proposto, os nós devem ser equipados com pelo menos dois rádios. O rádio principal, que usa a tecnologia IEEE 802.11, que também será aqui designado de canal padrão, é responsável pela troca dos quadros regulares de controle e pacotes de dados, enquanto o cognitivo é responsável apenas pelo envio de dados. A não presença de quadros de controle no canal cognitivo procura diminuir a interferência causada aos PUs. A negociação para uso do canal cognitivo ocorre durante a troca usual de RTS/CTS, não sendo necessário um canal de controle extra. Este protocolo proporcionará, ainda, o controle necessário que permitirá a coordenação da distribuição do fluxo de pacotes entre os múltiplos rádios de um dispositivo.

O protocolo baseia-se nas soluções alcançadas com o IEEE 802.11 para compartilhamento de espectro, tais como CSMA/CA, tempos de recuo, RTS/CTS, e acrescenta funcionalidades que procuram aumentar a vazão. Este ganho de desempenho é alcançado através do uso oportunista de outras bandas de frequência, de forma que os quadros são transmitidos concomitantemente no canal padrão designado para o dispositivo e na oportunidade de espectro descoberta.

Uma varredura dos canais é necessária para detectar oportunidades de acesso. Esta varre-

dura pode ser feita de forma sequencial ou aleatória, individualmente ou paralelamente. Para a varredura, considera-se limites de potência para detecção do UP. A avaliação das diferenças de desempenho causadas pelas diferentes formas de varredura não fazem parte do escopo deste trabalho.

No nosso protocolo, os nós competem pelo acesso ao canal de maneira usual, de acordo com os mecanismos regulares do 802.11. Essa disputa ocorre sobre o canal padrão designado, na faixa de frequência de 2.4GHz. Os nós deverão também ser equipados com um segundo rádio que será usado para as transmissões oportunistas na banda de frequência. Nós assumimos que essas frequências alternativas são licenciadas para os usuários primários (UP), enquanto os nós da rede *ad hoc* cognitiva serão tratados como usuários secundários (US).

Em adição ao rádio cognitivo, cada nó ainda será equipado com um módulo de envio, que é um *buffer* usado para armazenar cada pacote gerado pela aplicação. O módulo será inserido entre as camadas de rede e enlace da pilha de protocolo IP. Os pacotes gerados pela camada de aplicação serão encaminhados normalmente para a camada de transporte e, a seguir, para a camada de rede IP. Em seguida, os pacotes serão enfileirados nesse novo módulo para serem encaminhados para a fila da camada de enlace (ver Figura 3.1). Por esse motivo, o módulo de envio será denotado **pré-fila**. A pré-fila é responsável pelo controle da distribuição dos pacotes entre as interfaces MAC principal e cognitiva, o que quer dizer que, dentro do dispositivo, há somente um módulo de envio, porém podem existir múltiplas interfaces. A Figura 3.2 mostra como é a pilha de protocolos dentro de um dispositivo, destacando-se as múltiplas interfaces e o local de inserção do módulo de envio entre as camadas de pilha.

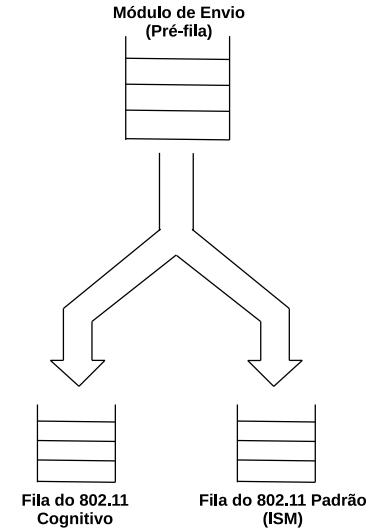


Figura 3.1: Pré-fila distribui os pacotes entre as interfaces do nó.

Além das mudanças já mencionadas, é necessário que se adicione campos nos cabeçalhos dos quadros de RTS, CTS e ACK do IEEE 802.11. Nós denotamos esses campos de *request available channel* (RAC), *confirm available channel* (CAC), *confirm MAC address* (CMA) e *opportunistic acknowledgement* (OPACK). O acréscimo destes campos visa possibilitar que o canal cognitivo seja usado somente para transmissões de dados, evitando comunicações desnecessárias de RTS, CTS,

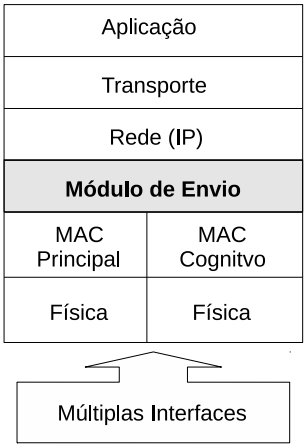


Figura 3.2: Nova estrutura das camadas do dispositivo destacando-se o local de inserção do módulo de envio e a presença de múltiplas interfaces.

ACK e ARP.

Os campos de RAC e CAC são utilizados para armazenar números inteiros que representam um canal de transmissão na faixa de frequência do UP. O protocolo partiu de uma sub-divisão igualitária dos canais. O protocolo pode ainda operar em faixas ou frações de faixa de frequência. Os campos RAC e CAC podem incorporar esta modificação. O campo de RAC do RTS serve para informar o primeiro canal disponível para transmissão de dados que o emissor encontrar na faixa dos UPs. Caso nenhum canal disponível seja descoberto, o campo deve ser preenchido com o valor zero por convenção. Na recepção do RTS, o destinatário lê o campo de RAC e verifica a disponibilidade do canal proposto pelo emissor. Esse verificação é necessária, pois durante a transmissão do RTS, um UP pode ter ocupado o canal proposto ou, em caso de rede com múltiplos saltos, um canal disponível para o transmissor pode não estar disponível na região de cobertura do receptor. Agindo desta maneira, o receptor ainda ajuda a evitar interferências nas comunicações em andamento dos UPs no canal proposto pelo emissor. Após a verificação do canal proposto, o receptor gera o quadro CTS respondendo ao emissor. Se aceitar a transmissão no canal proposto, o destinatário do RTS repete no campo de CAC do CTS o número de canal recebido, confirmando a disponibilidade do canal como percebido por ele. Caso recuse a transmissão no canal proposto, o campo de CAC assume valor zero.

Quando o emissor receber o quadro de CTS, ele verifica o valor do campo CAC. Caso o valor encontrado seja igual a 0, a transmissão no canal alternativo não ocorre. Se a resposta for igual ao número do canal enviado, a pré-fila é acionada para enviar uma cópia do primeiro pacote disponível para a interface cognitiva após um intervalo de tempo SIFS determinado, empiricamente, para que ambos os pacotes sejam enviados exatamente no mesmo instante. Ou seja, são transmitidos dois pacotes distintos, um por cada interface. Caso a pré-fila esteja vazia, nada é transmitido na faixa de frequência alternativa.

O campo OPACK nada mais é que um *bit* de indicação para um valor de verdadeiro ou falso. Se os pacotes em ambas as interfaces do destinatário forem recebidos com sucesso, um pacote de

ACK é enviado de volta ao emissor, com a campo de OPACK habilitado em 1, indicando sucesso na recepção oportunista. Por outro lado, se algum dos quadros não for recebido de maneira bem-sucedida temos três possibilidades.

1. Se o sucesso ocorrer apenas no canal ISM padrão, então o campo de OPACK é preenchido com o valor 0, indicando falha na transmissão oportunista. Nesta situação, o próximo pacote que será transmitido no canal ISM padrão será o mesmo que falhou no canal alternativo, sendo esse um dos motivos para usarmos uma cópia, e não o próprio pacote, em transmissões na interface cognitiva. Outro motivo para usarmos uma cópia do pacote na rede cognitiva é o fato de a interface cognitiva não ser habilitada para realizar retransmissões. Imagine que, na transmissão subsequente à falha desses mesmos nós, eles não concordem quanto a uma oportunidade de uso de espectro. O pacote que seria transmitido pela interface principal não seria aquele que é esperado pelo receptor, ocasionando um descarte desse pacote (o motivo desse descarte será explicado mais à frente). No pior cenário possível, se a tentativa de transmissão ocorrer por sete vezes seguidas, segundo o padrão IEEE 802.11, o destinatário é considerado inalcançável e a comunicação é cessada. Em outra possibilidade, algumas tentativas de transmissão ocorrerão, antes que uma nova oportunidade de canal seja acertada entre os nós, causando degradação do desempenho.

2. Se a falha ocorrer somente na interface principal, escolhemos a abordagem mais simples: ignoramos o pacote recebido na interface cognitiva para que os pacotes sejam recebidos sempre em ordem e repetimos todo o processo de envio. Vale ressaltar que ambos os canais operam na mesma velocidade de transmissão para não causar recepções fora de ordem. A outra abordagem possível seria aceitar pacotes fora de ordem. No entanto, mais armazenamento e controle seria mandatório no receptor, tornando o protocolo mais complexo devido a maior demanda de recursos e controle dos pacotes já recebidos.

3. Por último, temos a possibilidade de ambos os pacotes serem mal sucedidos. Também nesta situação, todo o procedimento de envio é refeito.

Finalmente, se o RAC do RTS já contém o valor 0, o receptor replica este valor no CTS e a transmissão ocorre de modo padrão seguindo o 802.11 usual. Aqui, assumimos que mesmo que o receptor saiba de um canal disponível, o transmissor pode estar percebendo atividade neste canal. Então, não há transmissões oportunistas.

Um importante passo a considerar é a necessidade de se determinar o endereço MAC da interface na faixa de frequência oportunista. De modo a evitar desperdícios das oportunidades encontradas e minimizar interferências na comunicação dos usuários legítimos da rede primária, a solução propõe evitar a realização de ARP no canal alternativo. Para isso, o CMA é um campo de tamanho suficiente para armazenar um endereço MAC. Então, o receptor envia no campo CMA do CTS o endereço da interface cognitiva onde ele deseja receber as transmissões oportunista. Deste modo o emissor já saberá qual o endereço físico associado ao endereço IP da rede secundária.

Dadas as mudanças propostas, a comunicação entre os nós se dará conforme explicação a seguir. Ao receber o primeiro pacote gerado pela aplicação (pacote com número de sequência (SN) igual

a 0), a pré-fila encaminha-o à interface principal, para que o nó entre na disputa pelo acesso do canal e requisite as informações de ARP (ARP *request*). É importante ressaltar que as requisições de ARP ocorrem apenas uma vez, antes da transmissão do primeiro pacote pela interface.

Após receber a resposta do ARP (ARP *reply*), o nó entra efetivamente na disputa pelo canal. Ao ganhar o direito de transmitir no canal, os nós trocarão quadros de RTS/CTS. É nesta etapa de troca de RTS/CTS que os nós informam ao outro par as oportunidades no espectro por meio dos campos de RAC e CAC criados no cabeçalho. Tais informações são obtidas através de varreduras constantes no espectro. Dessa forma, ao receber o RTS, o receptor compara o valor do campo RAC com os valores de canais disponíveis aferidos por ele. Para encerrar a negociação, o receptor envia o CTS com o valor resultante desta comparação no campo CAC. Vale ressaltar que o campo CAC ajuda a outros nós vizinhos ao destinatário a identificarem a trasnmissão eminente no canal proposto, ajudando, assim, no processo de busca do canal.

Essa negociação é feita antes de cada pacote que será transmitido. Se eles tiverem sucesso, o emissor iniciará a transmissão oportunista.

Se o pacote esperado chegar ao receptor, ele responde ao emissor com um quadro de ACK. Ao receber este quadro de controle, o par emissor da comunicação invoca o módulo de envio para que o mesmo encaminhe outro pacote à interface principal, o que significa que o encaminhamento de pacotes para a interface ocorrerá apenas após a confirmação do recebimento do pacote anterior. Portanto a fila padrão do MAC terá um ou nenhum pacote durante toda a transmissão. Para os pacotes subsequentes, repete-se todo o procedimento acima descrito, exceto a requisição de ARP.

Se durante a troca de RTS/CTS, os nós concordarem quanto à disponibilidade de canal alternativo para a transmissão oportunista, uma cópia do primeiro pacote disponível na fila do módulo de envio é encaminhada à interface cognitiva. Se a pré-fila não tiver pacotes disponíveis (a transmissão está mais rápida que a geração de pacotes pela aplicação), a oportunidade é perdida e nada é enviado pela interface cognitiva. Se houver um pacote na pré-fila, corrigimos os endereços IP do seu cabeçalho, conforme os endereços da rede cognitiva, antes de o encaminharmos para a interface cognitiva. Essa mudança se faz necessária, pois a interface cognitiva possui um endereço de rede diferente do endereço da interface principal. Antes de serem efetivamente transmitidos, as interfaces esperam um intervalo de tempo SIFS, selecionados de forma que os pacotes sejam transmitidos simultaneamente pelas múltiplas interfaces. O novo intervalo de tempo SIFS, modificado do IEEE 802.11, deve ser grande o suficiente para acomodar todas as alterações.

Na recepção pelo destinatário, o dispositivo informa ao emissor o recebimento de pacotes pela interface cognitiva através do campo OPACK, sendo que OPACK=1 indica sucesso na transmissão e OPACK=0, indica falha. Em seguida, o dispositivo compara o número de identificação (SN) que ele está esperando com aquele presente no cabeçalho do pacote. Se o pacote recebido for o esperado, o SN é incrementado. Esse controle foi criado para que pacotes fossem recebidos estritamente em ordem, o que quer dizer que pacotes recebidos fora de ordem serão descartados. Esse mecanismo é menos eficiente no sentido de que mesmo que os pacotes cheguem corretamente, em qualquer uma das interfaces, eles são descartados se estiverem fora de ordem, contudo sua implementação é mais simples. Pacotes recebidos pela interface cognitiva têm os IPs do cabeçalho alterados para que sejam encaminhados para a aplicação correta.

Para melhor esclarecimento, apresentamos na Figura 3.3 uma ilustração de como funciona a operação do protocolo. Na ilustração, o emissor sugere o canal 6 para a transmissão oportunista. O receptor confirma o opção de canal secundário e, a seguir, há a troca de dados por ambos os canais. O receptor confirma o recebimento dos dados através do ACK e OPACK.

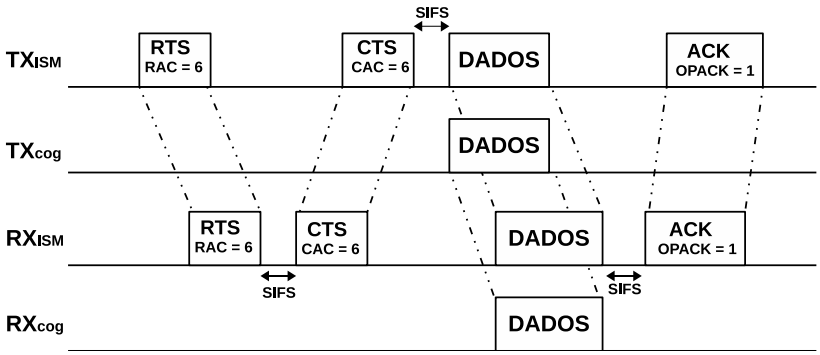


Figura 3.3: Operação do novo protocolo quando a negocição para uso do canal cognitivo é bem-sucedida. Neste caso, assim que o transmissor recebe a confirmação de disponibilidade do canal 6, ocorre a transmissão de dados nos dois canais.

Nas Figuras 3.4 e 3.5, apresentamos duas situações onde ocorrem problemas. Na Figura 3.4, o emissor envia uma sugestão de canal alternativo para a transmissão cognitiva, mas o receptor não aceita a sugestão. Assim, a transmissão ocorre apenas no canal principal. Na Figura 3.5, os nós concordam quanto a uma oportunidade de transmissão, mas por erro de canal ou interferências o pacote não chega à interface cognitiva do receptor. Assim, o receptor confirma a chegada do pacote pela interface principal (ACK), mas não confirma a chegada pela interface cognitiva (OPACK = 0).

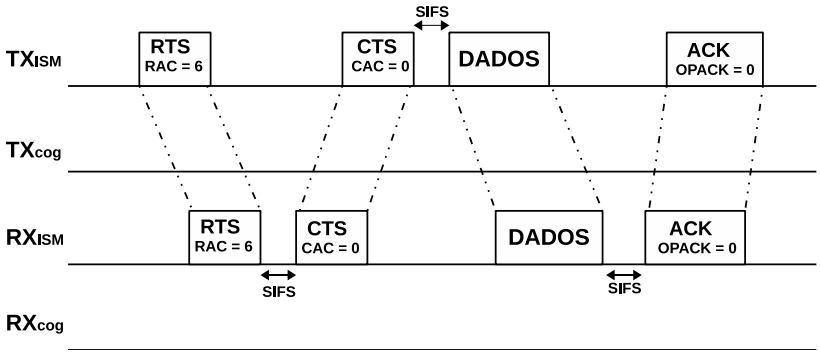


Figura 3.4: Operação do novo protocolo quando o receptor não concorda quanto ao canal disponível.

As Figuras 3.3, 3.4 e 3.5 deixam claro que as oportunidades no espectro encontradas pelo par de comunicação são usadas exclusivamente para troca de dados. Já o controle de toda a comunicação é feito através dos quadros que seriam usualmente trocados, tendo-se a transmissão (ou não) pela interface cognitiva. Desta maneira, destaca-se a utilização de um único canal de controle que evita a troca extra de *bytes* de controle no canal cognitivo. Para uma descrição mais concisa do protocolo, apresentamos um fluxograma de seu funcionamento na Figura 3.6.

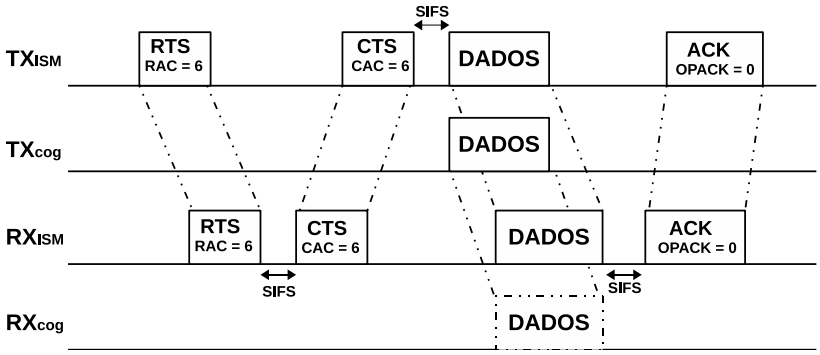


Figura 3.5: Operação do novo protocolo quando a negocição para uso do canal cognitivo é bem-sucedida, mas o pacote não é recebido pela interface cognitiva, conforme indica a linha tracejada.

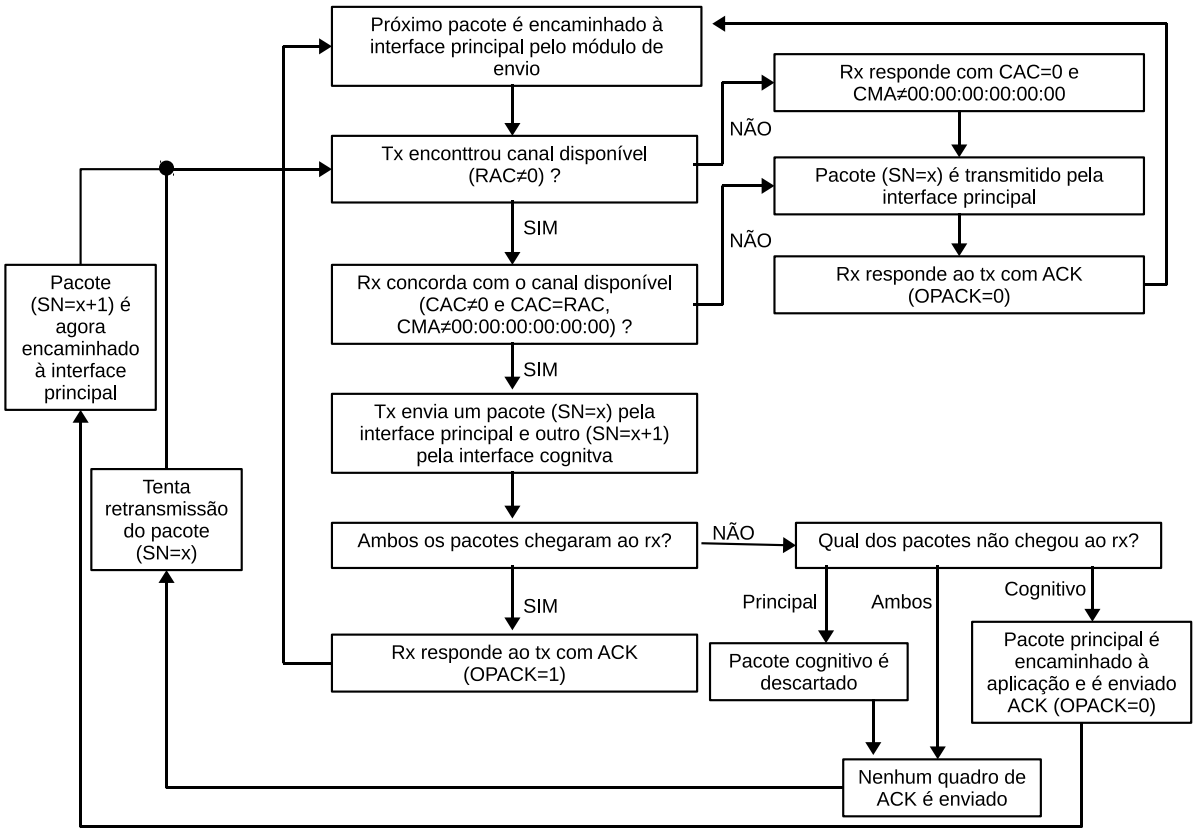


Figura 3.6: Fluxograma do funcionamento do protocolo.

# Capítulo 4

# Software NS-3

Para desenvolvimento do projeto foi utilizado o *software Network Simulator 3* (NS-3), versão 3.8 [6],[7], pois trata-se de um programa com código aberto e que passa por constantes atualizações por parte de seus desenvolvedores. O *Network Simulator 3* é um simulador de rede a eventos discretos no qual o núcleo da simulação e modelos são implementados em C++. O NS-3 é construído como uma biblioteca que pode ser estática ou dinamicamente associada a um programa principal em C++ que define a topologia de simulação e inicia o simulador. O NS-3 também exporta quase toda sua API para o Python, permitindo que os programas em Python importem um módulo "ns3" da mesma maneira que em C++. O código-fonte para o NS-3 fica organizado na pasta `src/` e pode ser descrito pelo diagrama apresentado na Figura 4.1.

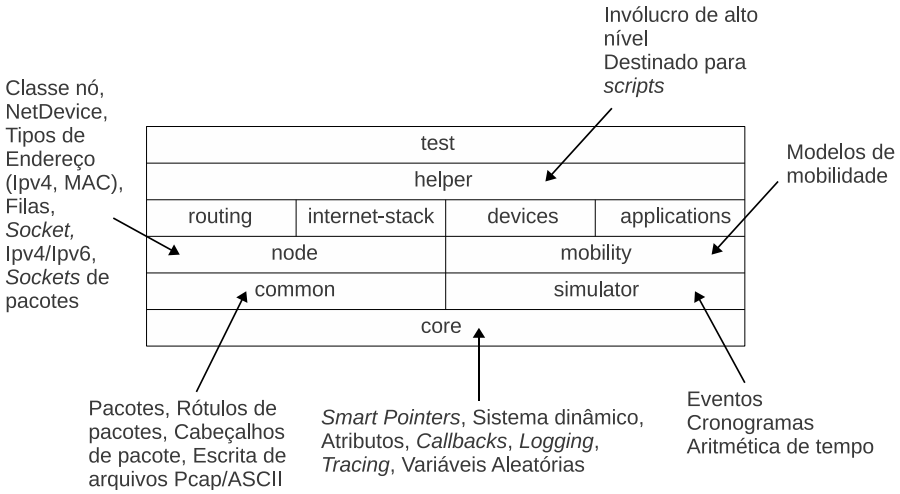


Figura 4.1: Estrutura do NS-3

Em vários casos, usuários podem não se satisfazer com uma mera adaptação dos modelos existentes, eles podem querer estender o núcleo do simulador de maneira inovadora. A estrutura da Figura 4.1 foi pensada de forma a se facilitar a adição de novas funcionalidades. Para tal, é preciso decidir em qual sub-pasta deve-se colocar o novo modelo. A sub-pasta `src/devices` contém os modelos de dispositivos de rede, tais como `Wifi, Wimax, CSMA, P2P`, etc. Já a sub-pasta `src/applications` abrange vários tipos de aplicações, por exemplo `OnOff, UdpEcho`, etc. A

sub-pasta `src/internet-stack` possui as classes que tratam dos protocolos de rede e transporte. Os algoritmos de roteamento podem ser encontrados na sub-pasta `src/routing`.

Nesse projeto, iremos modificar o módulo do dispositivo de rede sem-fio, situado em `src/devices/wifi`, por isso descreveremos com mais detalhes o seu funcionamento e sua organização. O `WifiNetDevice` modela um controlador de interface de rede sem fio baseada no padrão IEEE 802.11. Resumindo, o NS-3 fornece modelos para os seguintes aspectos do IEEE 802.11:

- IEEE 802.11 DCF básico com modos infra-estrutura e *ad hoc*
- As camadas físicas do 802.11a e 802.11b
- Vários modelos de propagação, incluindo TwoRayGround, Friis, etc.
- Dois modelos de atraso de propagação, etc.

O conjunto de modelos do IEEE 802.11 fornecido pelo NS-3 tenta prover uma implementação precisa da especificação do IEEE 802.11. A implementação é modular e fornece quatro níveis de modelos: os modelos para a camada física; os modelos inferiores do MAC que implementam a função DCF e EDCAF; os modelos superiores do MAC que implementam a geração de *beacons* e a associação dos nós; e um conjunto de algoritmos de controle usados pelos modelos inferiores do MAC.

Existem seis modelos superiores do MAC, três com funções de QoS e três sem essas funções. No projeto em questão foi utilizado um modelo sem QoS implementado pela classe `AdhocWifiMac`, um simples modelo *ad hoc* que não implementa nenhum tipo de geração de *beacons* ou associação.

A camada MAC inferior é dividida em três componentes: a classe `MacLow` que trata do envio dos quadros de RTS/CTS/DATA/ACK; as classes `DcfManager` e `DcfState` que implementam a funções DCF e EDCAF, utilizadas para calcular quando será garantido o acesso ao meio de transmissão; as classes `DcaTxop`, para os modelos sem QoS, e `EdcaTxopN`, para os modelos com QoS, que tratam da fila, fragmentação e das retransmissões dos pacotes, se forem necessárias. A camada física é desenvolvida pela classse `WifiPhy`. Esse arranjo das classes pode ser oberservado na figura 4.2.

A modularidade fornecida pela implementação faz com que a configuração das camadas inferiores do `WifiNetDevice` seja complexa, mas eficaz. Por essa razão, é provido algumas classes auxiliares, chamadas de *helpers*, que permitem ao usuário controlar os modelos das camadas inferiores de uma forma simples. A `YansWifiChannelHelper` é utilizada para criar um canal sem fio com modelos padrão de propagação e atraso. A classe `YansWifiPhyHelper` configura uma fonte de objetos que criam instâncias da classe `YansWifiPhy` e adiciona outros objetos a ela, possibilitando também acrescentar um modelo de mobilidade e um modelo de erro de canal.

A subclasse `WifiChannel` é utilizada para conectar um conjunto de interfaces de rede do tipo `WifiNetDevice`. A classe `WifiPhy` é o objeto dentro da classe `WifiNetDevice` que recebe os bits provenientes do canal. Essa classe modela um canal 802.11a, em termos de frequência, modulação e taxa de transmissão de bits com os modelos de propagação e atraso encontrados no canal.

As funções da camada MAC superior são implementadas em várias outras classes que lidam com:

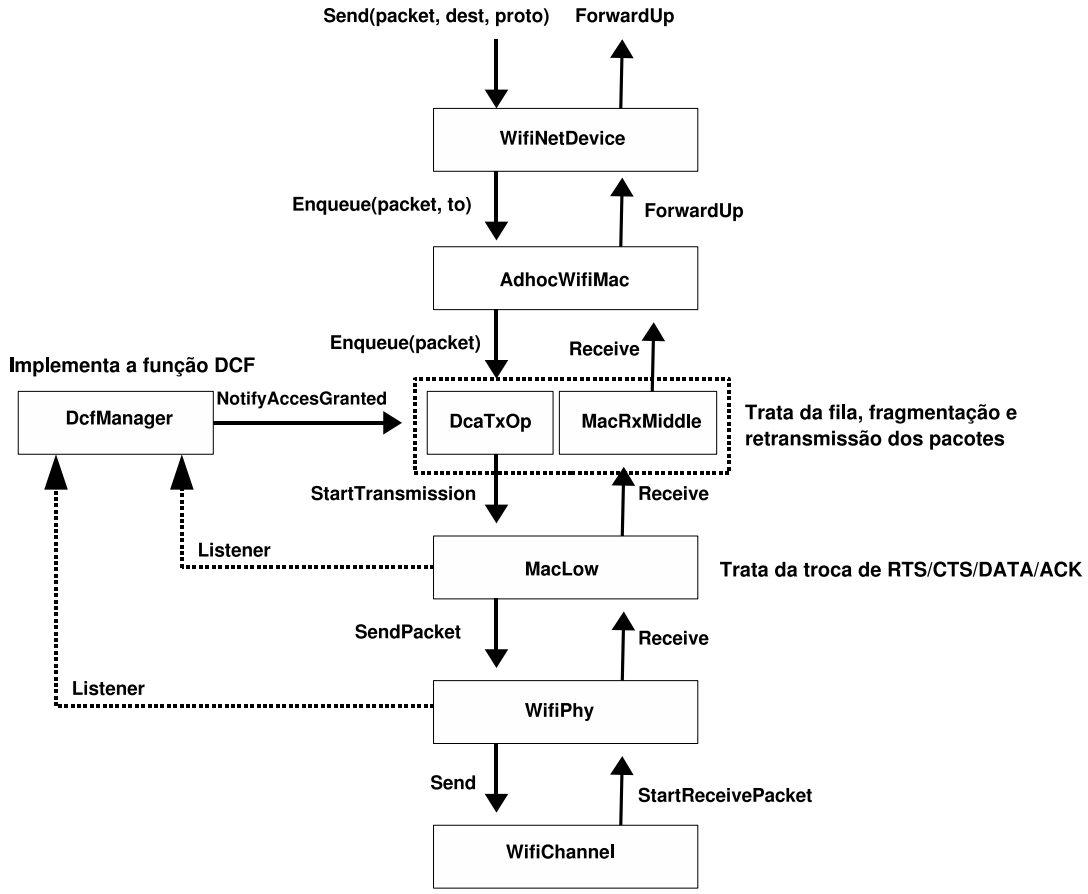


Figura 4.2: Estrutura do Dispositivo de Rede *Ad hoc*

- Fragmentação e desfragmentação dos pacotes
- Uso do RTS/CTS
- A fila de transmissão do MAC
- A geração de beacons, etc.

A classe `WifiNetDevice` faz uso de vários atributos para a configuração de seus valores padrão. Existem, aproximadamente, cem valores que são armazenados no sistema. Por exemplo, a classe `WifiMac` faz uso dos seguintes atributos: CtsTimeout, AckTimetot, Sifs, Pifs, EifsNoDifs, Slot, Ssid, MaxPropagationDelay e MaxMsduSize.

Esta breve descrição mostra como os dados são tratados a partir do dispositivo de rede. Descreveremos agora como os dados são tratados nas camadas superiores, tanto no envio como no recebimento dos mesmos. A figura 4.3 mostra o caminho, descrito abaixo, percorrido pelos pacotes no envio, desde sua geração pela aplicação até chegada ao dispositivo de rede.

1 – A aplicação cria previamente um socket (no caso, um `UdpSocket`) e então chama o método `Socket::Send()`.

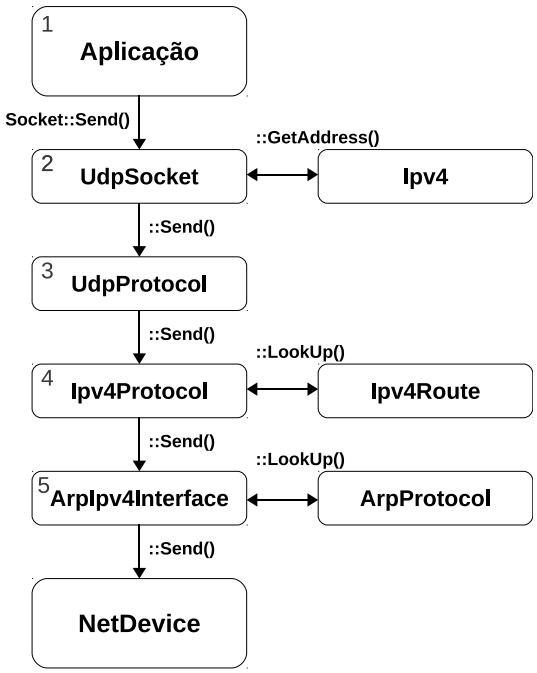


Figura 4.3: Caminho do pacote no envio

2 – O método `Socket::Send()` encaminha os dados para o método `UdpSocket::DoSend()` que posteriormente os encaminha para o método `UdpSocket::DoSendTo()`. Estas funções põem os endereços corretos de fonte e destino, lidam com as chamadas da classe socket, tais como *bind()* e *connect()* e então a função `UdpProtocol::Send()` é chamada. Como numa implementação real, o socket deve requisitar a camada Ipv4 o endereço correto que combine com o endereço de destino.

3 – `UdpProtocol` é onde a lógica de protocolo independente de socket para o UDP é implementada. O método `Send()` adiciona o cabeçalho UDP, inicializa a soma de verificação (*checksum*) e envia o pacote para a camada Ipv4. Nessa classe, uma API privada (`Ipv4Private`) é chamada e o método `Send()` é executado.

4 – A classe `Ipv4Protocol` adiciona o cabeçalho IP, procura por uma rota de envio e envia o pacote para a `Ipv4Interface` apropriada. Neste exemplo, o dispositivo deve ser um que suporta ARP, então o pacote é enviado para o objeto `ArpIpv4Interface`.

5 – `Ipv4Interface` é uma classe base abstrata; neste caso descrevemos a classe concreta `ArpIpv4Interface`. Este objeto verifica se o ARP é suportado pelo dispositivo de rede, e se o endereço MAC procurado já existe no cache do mesmo. Se sim, ele envia o pacote para o `NetDevice`, caso contrário, um requerimento de ARP é iniciado.

Já a figura 4.4 apresenta o encaminhamento, na recepção, desde o dispositivo de rede até a entrega à aplicação.

1 – O `NetDevice` chama a função registrada no `Node::m_receiveCallback`.

2 – Isto é tipicamente a função `Node::ReceiveFromDevice()`. O `Node::ReceiveFromDevice` armazena um conjunto de callbacks que são verificados baseado no número de protocolo e no dis-

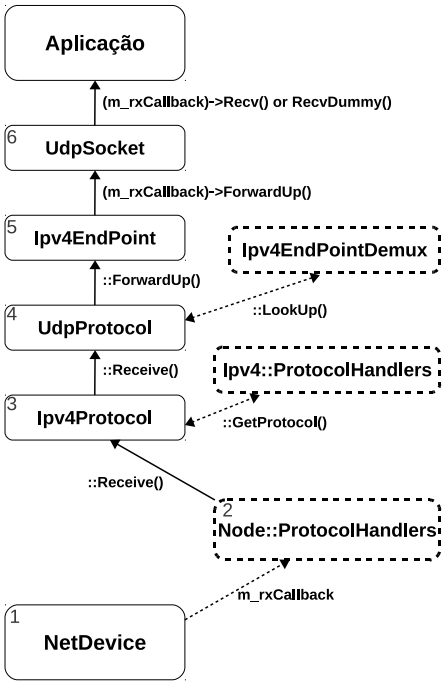


Figura 4.4: Caminho do pacote na recepção

positivo. Neste caso, a verificação irá resultar na chamada do método `Ipv4Protocol::Receive()`;

3 – A classe `Ipv4Protocol` remove o cabeçalho IP, checa a soma de verificação (se implementado) e então encaminha o pacote ou chama o método `ForwardUp()`. O método `ForwardUp()` verifica o número de protocolo IP em um demultiplexador baseado em callbacks e chama o método registrado `Receive()`.

4 – O método `Receive()` da classe `UdpProtocol` remove o cabeçalho UDP e verifica o estado de contexto por fluxo, que são um ou mais objetos `Ipv4EndPoint` armazenados em um `Ipv4EndPointDemux` (identificados por um endereço e porta de origem, e um endereço e porta de destino) . Ele então chama o método `Ipv4EndPoint::ForwardUp()`.

5 – O `Ipv4EndPoint` possui um *callback* no qual um objeto `Socket` pode registrar um método de recepção. Neste exemplo, este callback chama-se `UdpSocket::ForwardUp()`;

6 – O próprio `UdpSocket` chama um dos dois *callbacks* para enviar os dados para a aplicação.

# Capítulo 5

# Implementação no Simulador NS-3

Conforme a descrição do protocolo, fez-se necessário o desenvolvimento de uma nova classe no NS-3 para lidar com a distribuição dos pacotes entre as múltiplas interfaces do dispositivo, o módulo de envio, `DispatchModule`. Além disso, foram necessárias modificações nas classes já existentes na pilha de protocolos do NS-3.

Primeiramente, o módulo de envio foi inserido entre as classes `ipv4-l3-protocol`, identificada pelo números 4 e 3 nas Figuras 3.3 e 3.4, respectivamente, e `ipv4-interface`, representada pelo número 5 na Figura 3.3. Esse módulo consiste de duas filas, uma fila para pacotes e outra para seus cabeçalhos IP (IP *headers*). Esta diferenciação foi escolhida para facilitar a modificação de campos do cabeçalho IP quando determinado pacote for direcionado para transmissão pela interface cognitiva. Devido às características do simulador NS-3, a interface cognitiva se encontra em outra sub-rede. Portanto, necessita de diferente endereçamento para o par de comunicação. Esse módulo recebe os pacotes e cabeçalhos provenientes da classe `ipv4-l3-protocol` para serem armazenados. Posteriormente, estes serão encaminhados para as interfaces conforme as requisições são feitas.

Além disso, foram criadas duas variáveis (referentes às interfaces principal e cognitiva) para armazenar os objetos da classe `ipv4-interface`, de forma a possibilitar o envio do pacote às interfaces. Foram criados dois métodos de envio: `DispatchModule::Send`, para a interface principal, e `DispatchModule::CogSend`, para a cognitiva. Conforme já descrito, ao se encaminhar um pacote para a interface cognitiva, o mesmo não é retirado da fila do módulo de envio, pois a interface cognitiva não retransmite os pacotes quando eles não são recebidos corretamente. Os endereços de origem e destino presentes no cabeçalho também são alterados. Tais diferenças levaram à individualização dos métodos de envio. Além disso, é nestes métodos de envio que os cabeçalhos são adicionados aos pacotes.

A inserção do módulo de envio foi alcançada através de modificações na classe `ipv4-l3-protocol`. O método de envio, `Ipv4L3Protocol::SendRealOut`, que, anteriormente, era responsável pelo encaminhamento dos pacotes para a interface, agora os envia para a pré-fila. Neste método, é realizado o armazenamento, de forma indireta, dos objetos `ipv4-interface` na pré-fila.

A troca de RTS/CTS necessária para iniciar uma transmissão foi alterada na classe `mac-low`. No método de envio do RTS, `MacLow::SendRtsForPacket`, é verificada a presença do usuário primário nos canais escolhidos para o sensoriamento. A não presença de usuários primários no

canal verificado no alcance do transmissor é informada ao receptor através do campo RAC, que indica qual canal está disponível para a transmissão oportunista. Devido às mudanças realizadas, ocorrem atrasos na geração do quadro CTS. Portanto, foi incrementado o tempo de espera do CTS. O incremento escolhido empiricamente foi de $0.1ms$. A escolha de um canal disponível dentre vários possíveis implica na alteração da frequência do rádio cognitivo a cada transmissão. Na recepção do RTS, `MacLow::ReceiveOk`, foi armazenado o valor referente ao campo RAC para posterior acesso durante a criação do quadro CTS.

No método de envio do CTS, `MacLow::SendCtsAfterRts`, é verificada a presença do usuário primário no canal proposto. Diferentemente do emissor que varre vários canais, o receptor verifica a disponibilidade apenas do canal sugerido no RTS. Tendo encontrado este canal ocioso, o receptor responde com o mesmo número de canal no campo de CAC do quadro CTS. Caso contrário, o campo CAC é preenchido com um número de canal inexistente, no nosso caso, o valor zero. Se o campo de RAC do RTS já é recebido com zero, o receptor não realiza a verificação de canal disponível.

O receptor também informa ao emissor qual o endereço físico da interface cognitiva escolhida para a transmissão, evitando dessa forma, o ARP na rede cognitiva. Se uma oportunidade de canal é sugerida e confirmada, o receptor informará o endereço físico da interface cognitiva através do campo CMA do quadro de CTS. Por outro lado, se não houver concordância quanto à disponibilidade de canal, o receptor preenche o campo com um endereço padrão (00:00:00:00:00:00).

Na recepção do CTS pelo emissor, `MacLow::ReceiveOk`, é verificado se o campo CAC do CTS é igual àquele enviado por ele mesmo no campo RAC do RTS, porém diferente de zero. Se as opções de canal sugeridas forem iguais, manipulações são realizadas para se obter o endereço IP cognitivo do nó destinatário, pois o endereço de destino presente no cabeçalho do pacote IP é aquele referente à rede legítima. A Figura 5.1 explica, por meio de um exemplo, quais manipulações foram realizadas. Destaca-se, porém, que elas só são possíveis devido à forma como o NS-3 atribui endereços aos nós.

No exemplo, imagine que queremos uma transmissão do nó 3 para o nó 7 e que, em determinado momento, eles concordem quanto a uma oportunidade de uso de certo canal. Isto levanta a seguinte questão: como o transmissor pode obter o endereço IP cognitivo do receptor, sendo que ele somente possui o endereço IP principal do receptor e tem acesso ao seu próprio endereço IP cognitivo? Como o NS-3 atribui o endereços sequencialmente aos nós do contêiner (e os nós são os mesmos tanto na rede principal quanto cognitiva), podemos resolver a situação da maneira como é mostrado na Figura 5.1 . A figura contém uma tabela que mostra os endereços dos nós em questão tanto na rede principal quanto na rede cognitiva e um diagrama com as manipulações realizadas. Em uma rede mais realista, os endereços IP são atribuídos de outra forma, que não sequencialmente, também seria necessário criar um campo no CTS para informar o endereço IP da interface cognitiva.

De posse do endereço MAC e IP corretos para a interface cognitiva, podemos adicionar uma nova entrada na tabela de cache do ARP, para assim evitar a execução do mesmo na rede cognitiva. O método responsável por essa inserção, `Ipv4Interface::SetCache`, verifica se o endereço IP já existe na tabela de cache do nó. Se não existir, ele insere uma nova entrada na tabela. Após todo este processo, o pacote é, finalmente, encaminhado para a interface cognitiva, onde aguarda

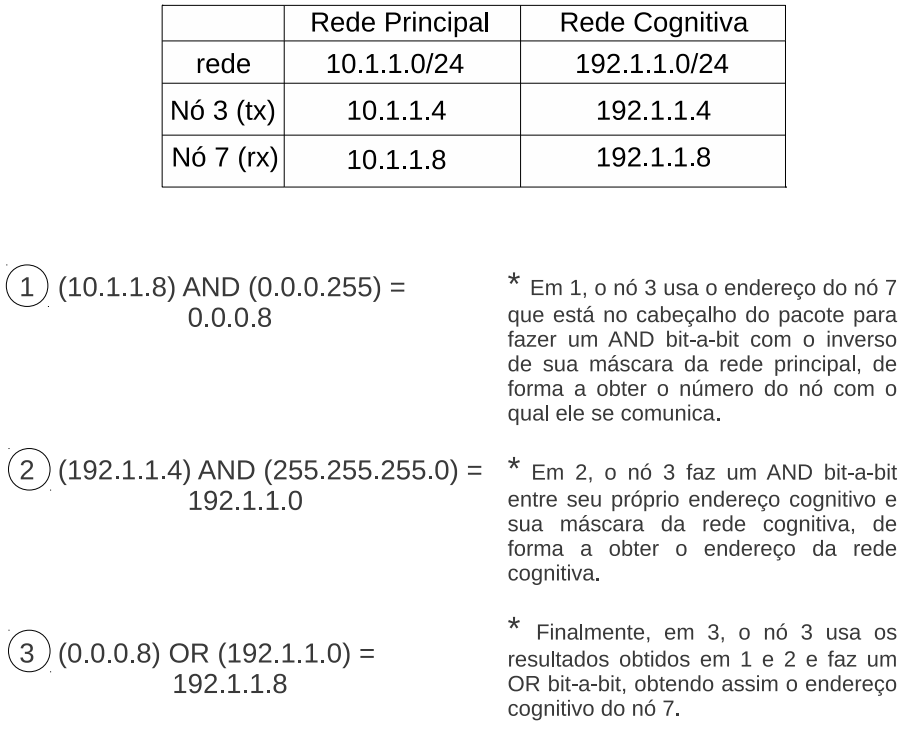


Figura 5.1: Manipulações para obter o endereço IP cognitivo do receptor.

um tempo SIFS específico até a transmissão. O tempo SIFS atribuído para a rede cognitiva foi de $0.026ms$. Tal valor, determinado empiricamente, foi escolhido para que os pacotes fossem transmitidos simultaneamente pelas múltiplas interfaces.

Na chegada dos pacotes ao receptor, `MacLow::ReceiveOk` verifica se o endereço de destino presente no cabeçalho MAC é igual ao endereço físico de alguma de suas interfaces. Sendo igual ao da interface cognitiva, a variável *cogAck*, presente na classe `ipv4-l3-protocol`, assume o valor um para sinalizar a chegada de um pacote pela sub-rede cognitiva. Se for igual ao da interface principal, um quadro de ACK é programado para ser enviado após um tempo SIFS.

No método de envio do ACK, `MacLow::SendAckAfterData`, o valor da variável *cogAck* é atribuído ao campo OPACK do cabeçalho do quadro ACK a ser enviado e, em seguida, tal variável é zerada.

Os pacotes recebidos são encaminhados para cima na pilha de protocolos conforme Figuras 3.2 e 3.4, pois não passam pelo módulo de envio[1] até que chegam à classe `ipv4-l3-protocol`. Em seu método de recepção, `Ipv4L3Protocol::Receive`, remove-se o cabeçalho IP do pacote. Se os endereços IP do cabeçalho forem da sub-rede cognitiva, eles são reajustados, num processo inverso àquele apresentado na Figura 5.1. Desta forma, todos os pacotes são encaminhados para a mesma aplicação. É neste método que é verificado se o SN do pacote recebido combina com aquele que é esperado. Para tal, cada nó possui dois vetores, `IdList` para armazenar o SN esperado e `Ipv4AddressList` para armazenar o endereço de rede do qual os pacotes advém. Tais vetores

[1] Afinal ele é de envio, não recepção.

possuem tantas posições quanto são os nós presentes na simulação (na verdade, quantidade de nós menos uma unidade), pois cada posição representa uma conexão possível.

Dado o recebimento de um pacote, é verificado se o endereço de origem presente no cabeçalho do mesmo já existe no vetor `Ipv4AddressList`. Se não, trata-se do primeiro pacote recebido advindo daquela origem. Neste caso, o endereço de origem é adicionado ao vetor `Ipv4AddressList` na primeira posição disponível. O campo SN recebe o valor 1. Afinal o vetor `IdList` armazena o SN esperado para o próximo pacote, uma vez que o pacote com SN igual a zero acabou de ser recebido. Agora, se o endereço já se encontra presente no vetor `Ipv4AddressList`, compara-se o SN do pacote recebido com o esperado. Se forem iguais, o pacote segue o encaminhamento para as camadas superiores e o valor de SN esperado é incrementado. Senão, o pacote é descartado, pois não possui o SN esperado.

Do recebimento do ACK, `MacLow::ReceiveOk`, é testado se este corresponde ao reconhecimento de uma transmissão de dados bem-sucedida. Caso sim, verifica-se o valor do campo OPACK. Se este campo estiver com o valor 1, indicando que a transmissão do pacote pela interface cognitiva foi bem-sucedida, faz-se a retirada deste pacote da fila do módulo de envio. Então, uma chamada indireta do método `DispatchModule::Send` é executada, o qual encaminha pacotes à interface principal.

Observe que, se a transmissão for mais rápida que a geração de pacotes pela aplicação, eventualmente, a pré-fila ficará vazia. Nesta situação de pré-fila vazia, ao se receber o último quadro de ACK, não haveria mais encaminhamentos de pacotes à interface para transmissão. Sem transmissões, não há mais a recepção de ACKs, e sem a recepção da ACKs não há mais encaminhamentos. Isto é, a comunicação cessaria. Para contornar tal obstáculo, o módulo de envio sinaliza quando sua fila está vazia ao `ipv4-l3-protocol`. Assim, dada a chegada de um novo pacote advindo da aplicação, ele é diretamente encaminhado para o dispositivo de rede, voltando-se, desta forma, ao ciclo de ACKs e encaminhamentos.

Todos os campos criados para os cabeçalhos dos quadros MAC mencionados anteriormente, OPACK, CMA, RAC e CAC, foram adicionados na classe `wifi-mac-header`, mais especificamente em `WifiMacHeader::Serialize`, `WifiMacHeader::Deserialize` e `WifiMacHeader::GetSize`. O método `WifiMacHeader::Serialize` é responsável por fazer a escrita dos parâmetros no *buffer* da memória, enquanto o método `WifiMacHeader::Deserialize` é responsável pela leitura. Já o método `WifiMacHeader::GetSize` é usado para informar a quantidade de memória ocupada pelo cabeçalho, possibilitando sua leitura correta, bit-a-bit, através do método citado acima.

# Capítulo 6

# Avaliação de Desempenho

Para testar o protocolo desenvolvido, foram realizadas simulações em diferentes cenários e topologias. Nesta seção, apresentaremos quais foram as soluções de implementação escolhidas para os cenários de simulação, bem como os resultados obtidos.

A banda que será usada para as transmissões oportunistas serão os próprios canais do IEEE 802.11. É sabido, porém, que no IEEE 802.11 há sobreposição dos canais, conforme pode ser visto na Figura 6.1. Portanto, deve-se respeitar quatro canais de distância para minimizar a interferência co-canal. Por exemplo, se a interface principal de dispositivo opera no canal 3, a interface cognitiva pode procurar uma oportunidade de acesso em qualquer canal do 8 ao 13. Desta forma, os campos RAC e CAC carregam a informação do número do canal do IEEE 802.11, isto é, um número de 1 a 13.

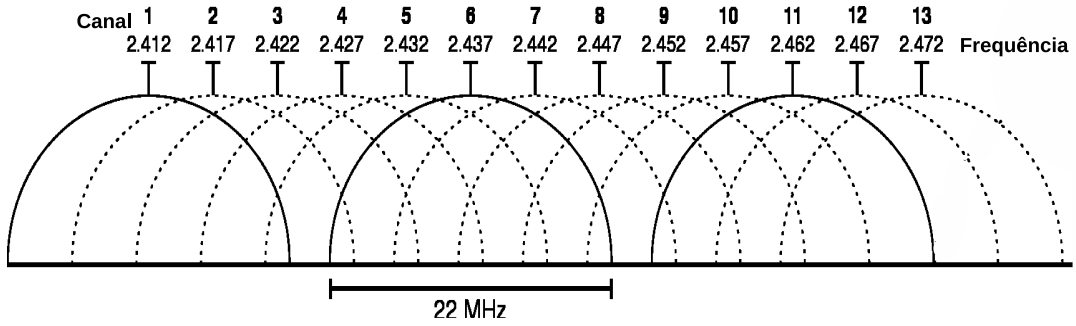


Figura 6.1: Sobreposição dos canais estipulados pelo padrão IEEE 802.11.

A presença dos UPs foi simulada através de uma variável aleatória. Os valores sorteados determinam o estado do canal. Para as simulações realizadas, foram utilizadas as probabilidades de 30%, 60% e 90% para mapear a ocupação do canal pelo UP.

Devido à forma como o simulador funciona, a varredura do espectro não pode ser feita de forma paralela. O simulador executa os comandos sequencialmente, tornando impossível o processamento paralelo. Portanto, a varredura do espectro é realizada no momento de criação dos quadros de

RTS/CTS. Para diminuir os atrasos decorrentes desse processo, o emissor informa ao receptor apenas o primeiro canal disponível encontrado. Já o receptor, somente verifica a disponibilidade do canal sugerido pelo emissor.

Com a busca de oportunidades na faixa do espectro, as mudanças de canal cognitivo deveriam ser acompanhadas por variações na frequência na qual o rádio transmite. Entretanto, por se tratar de uma simulação, tais valores de frequência não foram alterados, visto que foi observado que o simulador não percebe interferência co-canal. Os valores de frequência seriam utilizados apenas no cálculo da perda de percurso, o que não faz parte do escopo deste estudo. Desta forma, apesar de a varredura ser feita em vários canais, a transmissão cognitiva ocorre somente pelo mesmo canal. Assim, como a interface principal foi configurada para transmitir no canal 1, os nós somente fazem a varredura a partir do canal 6, em respeito à distância mínima necessária para evitar o interferência de canal adjacente. A transmissão, entretanto, é feita sempre no canal 6.

Para compor os cenários de simulação, foram definidas diversas outras características. O cenário é composto de uma rede *ad hoc* totalmente conectada, onde os nós estão posicionados de forma aleatória no terreno. As topologias testadas foram para uma rede de 10 nós e outra com 40 nós. Além disso, as topologias têm tantas conexões quantos são os números de nós, de modo que todos os nós transmitem, porém nem todos são necessariamente receptores.

A aplicação utilizada é do tipo *On-Off*, configurada de tal forma que todos os nós gerem pacotes constantemente, de modo a saturar a rede para obter o máximo de seu desempenho quanto à vazão agregada. É utilizado o protocolo UDP na camada de transporte e a camada de enlace é baseada no padrão IEEE 802.11b, com taxa de transmissão de 1Mbps.

O modelo de propagação de Friis é utilizado na simulação, já que não há nenhum obstáculo entre o emissor e o receptor (espaço-livre). As transmissões são modeladas com o atraso de propagação constante, isto é, a velocidade de propagação é constante. Mais ainda, os nós estão em posições contantes, porque, sendo a rede totalmente conectada, não queremos que os nós fiquem fora do alcançe uns dos outros.

Outra configuração importante do modelo testado é que só existem duas interfaces por nó. Uma interface principal que funciona normalmente na banda designada para o nó. Outra interface, denotada cognitiva, cujas transmissões dependem da disponibilidade de espectro livre em outras bandas. Outros parâmetros pertinentes para a realização das simulaçoẽs são apresentados na Tabela 6.1.

Para cada probabilidade testada, foram utilizadas diferentes quantidades de canais cognitivos disponíveis para a varredura. Estas quantidades foram selecionadas de forma a proporcionar uma avaliação do melhor e do pior caso de utilização do espectro.

O melhor caso ocorre quando os dispositivos se encontram nos extremos do espectro disponível. Os canais 1 e 13 podem, ainda que respeitem a distância de quatro canais, fazer a varredura dos oito canais restantes (ver Figura 6.1). Operando no canal 1, o nó pode varrer os canais de 6 a 13, se operar no 13, pode varrer os canais de 1 a 8.

Já o pior caso ocorre da utilização dos canais centrais do espectro, canais 5 a 9. Estes testam, no máximo, quatro canais em busca de oportunidades (ver Figura 6.1). Um nó operando no canal

Tabela 6.1: Parâmetros da camada MAC e física

| MAC | | PHY | |
|---|---|---|---|
| Carga útil | 1000 bytes | Erro de canal | 1% |
| $SIFS_{std}$ | 16$\mu$s | Limiar de detecção | -96 dBm |
| $SIFS_{cog}$ | 26$\mu$s | Figura de ruído | 7 dB |
| DIFS | 60$\mu$s | Ganho de tx | 1 dB |
| Cabeçalho MAC | 24 bytes | Ganho de rx | 1 dB |
| RTS | 44 bytes | Temperatura | 290 Kelvin |
| CTS | 46 bytes | | |
| ACK | 38 bytes | | |

7, por exemplo, pode verificar os canais 1, 2, 12 e 13 sem que haja sobreposição em seu canal principal e os canais verificados.

Mais ainda, as simulações permitem avaliar a vantagem do uso do rádio cognitivo ao se utilizar os recursos mínimos necessários para o funcionamento da técnica, ou seja, apenas um canal está disponível para a varredura.

Antes de apresentarmos os resultados obtidos com as simulaçoẽs descritas acima, mostramos por meio da Tabela 6.2 uma comparação entre o desempenho obtido para a vazão total da rede quando comparamos o padrão 802.11, com e sem o módulo de envio instalado entre as camadas da pilha de protocolo dos nós, para uma topologia com 10 nós. Consideramos como vazão total ou vazão agregada, a soma da carga útil de todos os pacotes de dados enviados dividido pelo tempo total da simulação. Para melhor precisão dos resultados obtidos foram realizadas cinco simulaçoẽs com diferentes sementes[2] e para melhor apreciação dos resultados foram calculados intervalos de confiança de 95% utilizando a função **t** de Student. Aos dispositivos sem a inserção do módulo, chamaremos *padrão*, então usaremos o subscrito *std*. Já os com o módulo, chamaremos *cognitivos*, para tal usaremos o subscrito *cog*.

Tabela 6.2: Vazão com e sem o módulo de envio para 10 nós

| Vazão | Média(Mbps) | Inter. de Confiança |
|---|---|---|
| $Vazão_{std}$ | 0,791333 | 0,006272641 |
| $Vazão_{cog}$ | 0,764613 | 0,005142589 |

Para um topologia com 10 nós, observou-se que, devido ao aumento de processamento (número de instruções executadas) ocasionado pela introdução do módulo de envio, a vazão agregada alcançada diminuiu aproximadamente 3.38% em relação àquela obtida pela rede padrão. Isto pode ser visto como uma limitação do simulador, ou seja, um *hardware* mais rápido reduziria essa diferença de desempenho. A Tabela 6.3 faz a mesma comparação só que para 40 nós.

[2]As sementes utilizadas foram 1, 8, 12, 27 e 33.

Tabela 6.3: Vazão com e sem o módulo de envio para 40 nós

| Vazão | Média(Mbps) | Inter. de Confiança |
|---|---|---|
| $\text{Vazão}_{std}$ | 0,771653 | 0,008128115 |
| $\text{Vazão}_{cog}$ | 0,717973 | 0,030224645 |

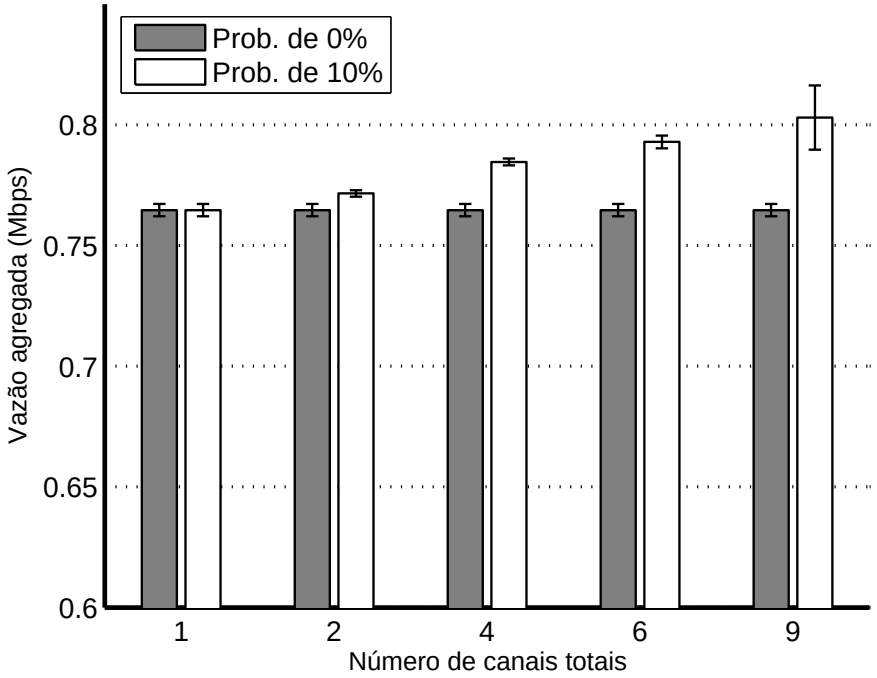


Figura 6.2: Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 10% para uma rede de 10 nós.

Para 40 nós, foi constatada uma queda já esperada na vazão agregada, visto que há uma maior disputa pelo canal e aumento da troca de quadros de controle. Foi observada uma discrepância de 6.96% entre as vazões. Tal discrepância foi maior que a observada para 10 nós pois, como a quantidade de nós é maior, há mais processamento extra devido à inserção do módulo de envio.

Finalmente, os resultados das simulações produzidas para a topologia de 10 nós são apresentados. Nos resultados mostrados, os gráficos ilustrados nas Figuras 6.2, 6.3 e 6.4 possuem como base de comparação as vazões entre a rede cognitiva sem nenhuma oportunidade de espectro (0%) e a rede cognitiva com as probabilidades de ausência do UP de 10%, 40% e 70% conforme descrito anteriormente. Nos gráficos, o eixo das abscissas representa a quantidade de canais totais disponíveis, ou seja, seu canal legítimo mais os canais cognitivos. De forma similar ao descrito anteriormente, foram tomadas cinco realizações do experimento com diferentes sementes para o cálculo do intervalo de confiança de 95%. Nos gráficos, é apresentada a média da vazão obtida nestas realizações juntamente com os intervalos calculados.

Como pode ser observado, houve um ganho significativo na vazão agregada da rede ao se

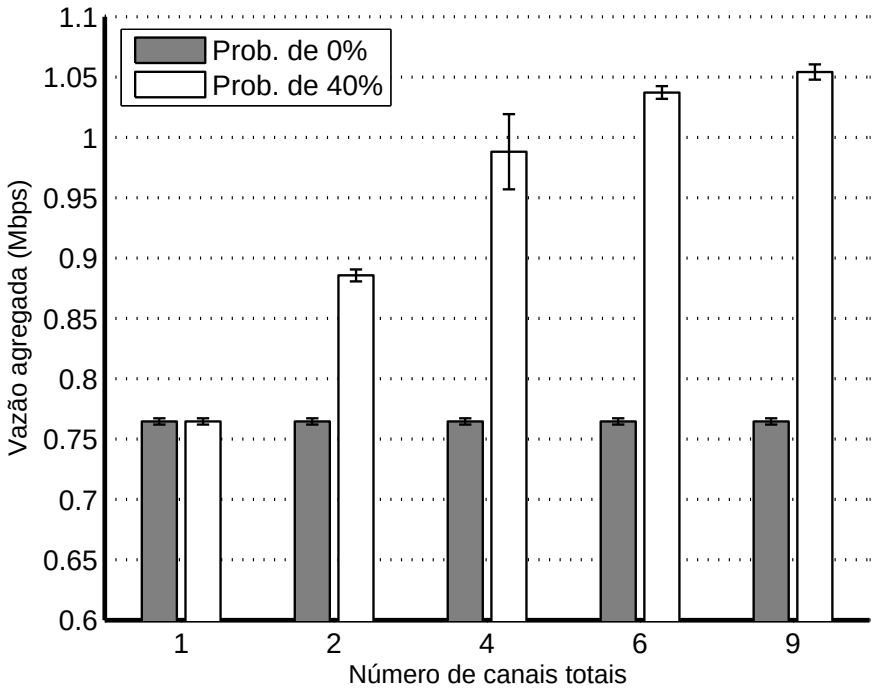


Figura 6.3: Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 40% para uma rede de 10 nós.

comparar o 802.11 com módulo cognitivo mas sem canais cognitivos disponíveis e o 802.11 com probabilidade de canais extras. A vazão obtida com apenas um canal cognitvo é coerente com aquela esperada probabilisticamente, uma apreciação desse resultado pode ser visto na Tabela 6.4. Se denotarmos por $P_d$ a probabilidade do emissor ou do receptor encontrarem um canal disponível, o ganho esperado ($g_e$) no canal cognitivo é dado por $P_d^2$. Logo, a vazão esperada pode ser calculada a partir da vazão obtida sem canais cognitivos disponíveis (Vazão$_{cog_0\%}$) multiplicando-se o valor de ganho esperado acrescido de um.

Tabela 6.4: Comparação entre as vazões esperada e obtida para um canal cognitivo e topologia de 10 nós.

| $P_d$ | Vazão$_{cog_0\%}$(Mbps) | $g_e$ | Vazão esperada(Mbps) | Vazão obtida(Mbps) | Ganho obtido |
|---|---|---|---|---|---|
| 10% | 0,764613 | 1% | 0,772259 | 0,771547 | 0,90% |
| 40% | 0,764613 | 16% | 0,886951 | 0,885627 | 15.8% |
| 70% | 0,764613 | 49% | 1,139273 | 1,125520 | 47.2% |

A tabela 6.5 apresenta os ganhos obtidos para diferentes quantidades de canais cognitivos e probabilidades de ausência do UP, para uma topologia de dez nós.

Naturalmente, quanto maior a probabilidade de se encontrar um canal disponível, maior é o ganho alcançado. A Figura 6.5 permite comparar os três cenários apresentados com melhor clareza.

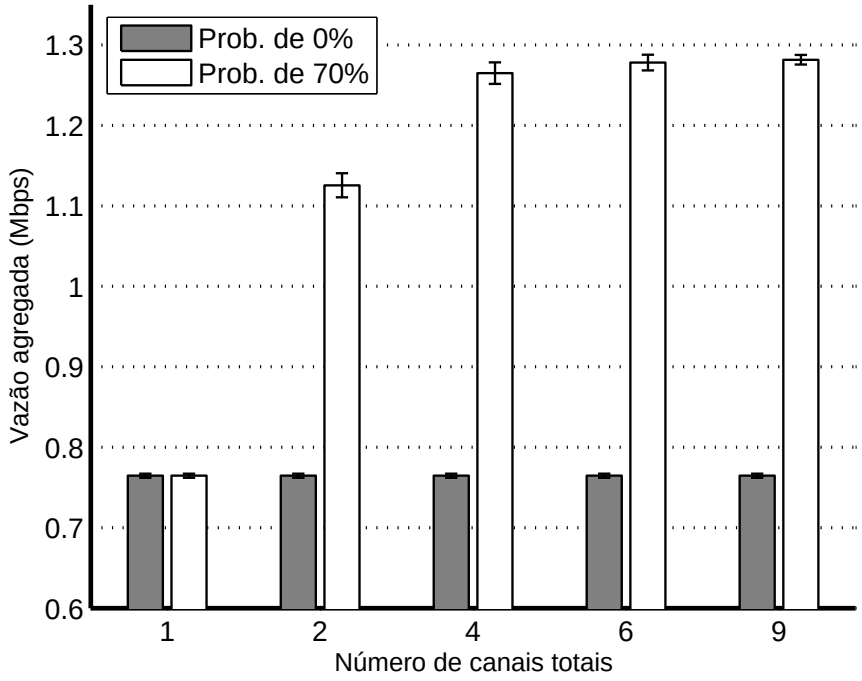


Figura 6.4: Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 70% para uma rede de 10 nós.

No melhor dos casos, com uma probabilidade de 70% de encontrar o canal disponível e oito canais extras para varredura, observou-se que o desempenho praticamente dobrou.

Pode-se verificar que o novo protocolo permite aumento da vazão alcançada por cada nó da rede. Para avaliar o desempenho da rede quanto a vazão média dos nós, a Figura 6.6 apresenta os dados aferidos para as três situações simuladas. Considera-se como vazão média, o resultado da divisão da vazão agregada pelo número de nós da topologia.

Através da análise dos gráficos obtidos, constatou-se que ocorre uma saturação da vazão obtida dado o aumento do número de canais. Nas simulações, tal saturação foi observada em torno de seis canais, um canal principal mais cinco cognitivos, para os gráficos de vazão agregada. Para vazão média por nó, a saturação ocorreu em torno de quatro canais totais. Portanto, pode-se concluir que com a técnica de rádio cognitivos, não se faz necessário uma grande quantidade de canais

Tabela 6.5: Comparação do ganho na vazão para diferentes probabilidades e quantidades de canais para um topologia de 10 nós.

| Número de canais | Prob. de 10% | Prob. de 40% | Prob. de 70% |
|---|---|---|---|
| 1 | 0,9% | 15,8% | 47,2% |
| 3 | 2,6% | 29,2% | 65,4% |
| 5 | 3,7% | 35,6% | 67,1% |
| 8 | 5,0% | 37,9% | 67,6% |

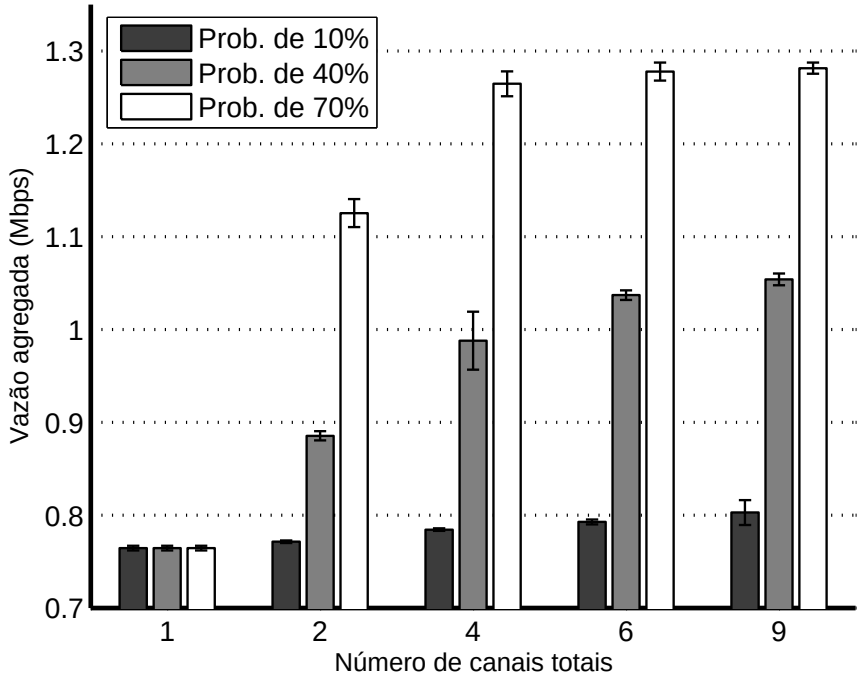


Figura 6.5: Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 10%, 40% e 70% para uma rede de 10 nós.

disponíveis para varredura com o intuito de se obter um ganho significativo na vazão.

Finalmente são apresentados os resultados obtidos para uma topologia de 40 nós. Conforme observado nas Figuras 6.7, 6.8 e 6.9, notou-se que a vazão foi menor quando comparado aos casos com a topologia de 10 nós. Tal resultado, como já descrito, deve-se ao fato da disputa pelo canal ser maior e haver uma maior troca de bytes de controle, tais como o do protocolo ARP.

A Tabela 6.6 mostra, para a topologia de 40 nós, o ganho obtido quando se utiliza apenas um canal cognitivo. Os dados apresentados seguem a mesma análise feita para a topologia de 10 nós.

Tabela 6.6: Comparação entre as vazões esperada e obtida para um canal cognitivo e topologia de 40 nós.

| $P_d$ | Vazão$_{cog_0\%}$(Mbps) | $g_e$ | Vazão esperada(Mbps) | Vazão obtida(Mbps) | Ganho obtido |
|---|---|---|---|---|---|
| 10% | 0,717973 | 1% | 0,725152 | 0,723200 | 0,73% |
| 40% | 0,717973 | 16% | 0,832849 | 0,833653 | 16.11% |
| 70% | 0,717973 | 49% | 1,069780 | 1,041920 | 45.12% |

A tabela 6.7 apresenta os ganhos obtidos para diferentes quantidades de canais cognitivos e probabilidades de ausência do UP, para uma topologia de quarenta nós.

Para melhor apreciação dos resultados alcançados, a Figura 6.10 apresenta um gráfico comparitivo do desempenho para diferentes $P_d$. O gráfico da Figura 6.11 mostra os resultados obtidos

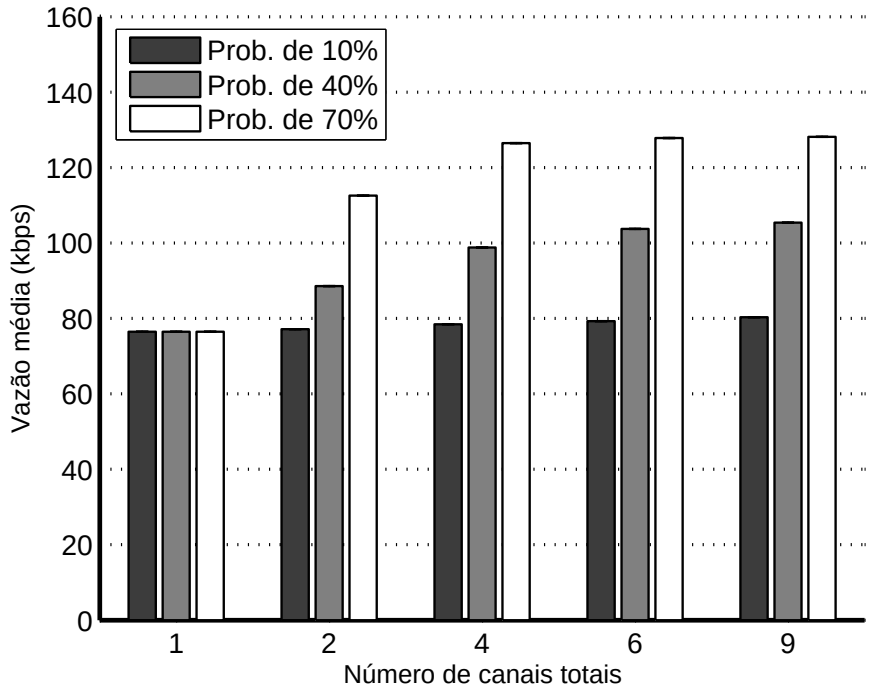


Figura 6.6: Vazão média dos nós na rede versus número de canais totais com probabilidade de ausência do UP de 10%, 40% e 70% para uma rede de 10 nós.

Tabela 6.7: Comparação do ganho na vazão para diferentes probabilidades e quantidades de canais para um topologia de 40 nós.

| Número de canais | Prob. de 10% | Prob. de 40% | Prob. de 70% |
|---|---|---|---|
| 1 | 0,7% | 16,1% | 45,1% |
| 3 | 1,6% | 28,9% | 61,0% |
| 5 | 4,3% | 34,1% | 62,5% |
| 8 | 4,6% | 36,2% | 62,6% |

para a vazão média dos nós.

A mesma análise feita para 10 nós também se aplica para o caso de 40 nós, ou seja, houve uma saturação do ganho da vazão em torno de cinco canais.

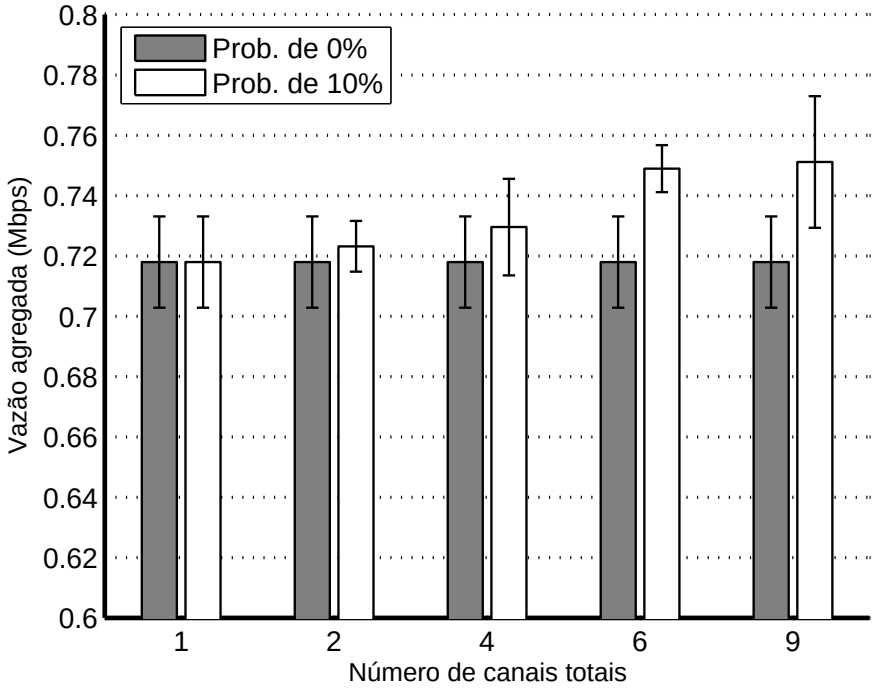


Figura 6.7: Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 10% para uma rede de 40 nós.

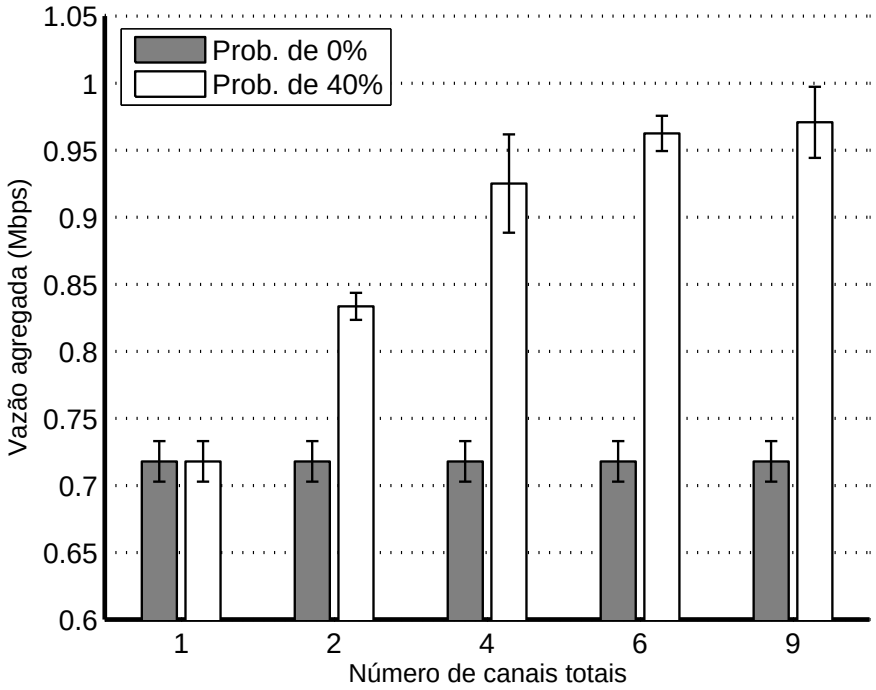


Figura 6.8: Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 40% para uma rede de 40 nós.

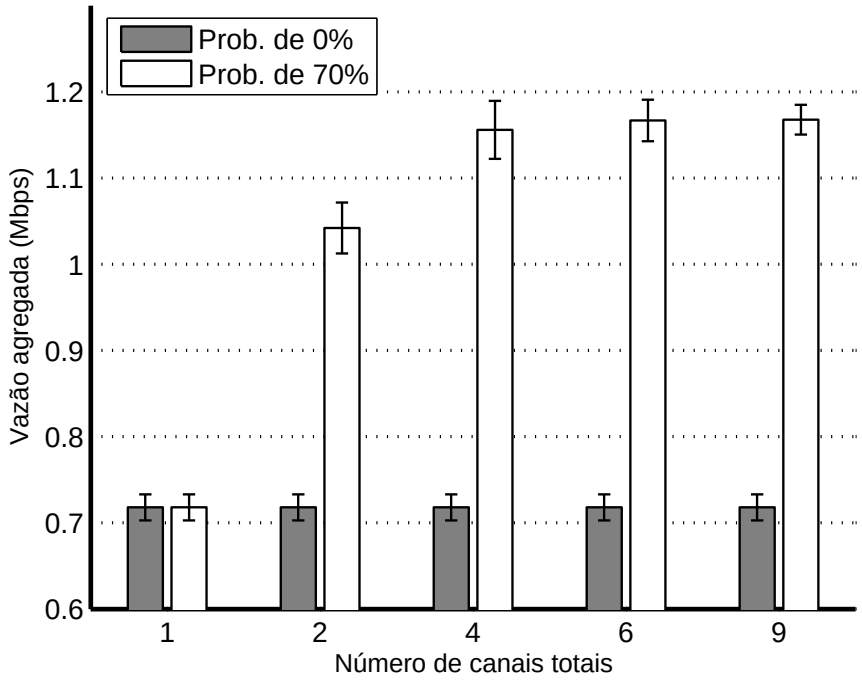


Figura 6.9: Vazão agregada versus número de canais totais com probabilidade de ausência do UP de 0% e 70% para uma rede de 40 nós.

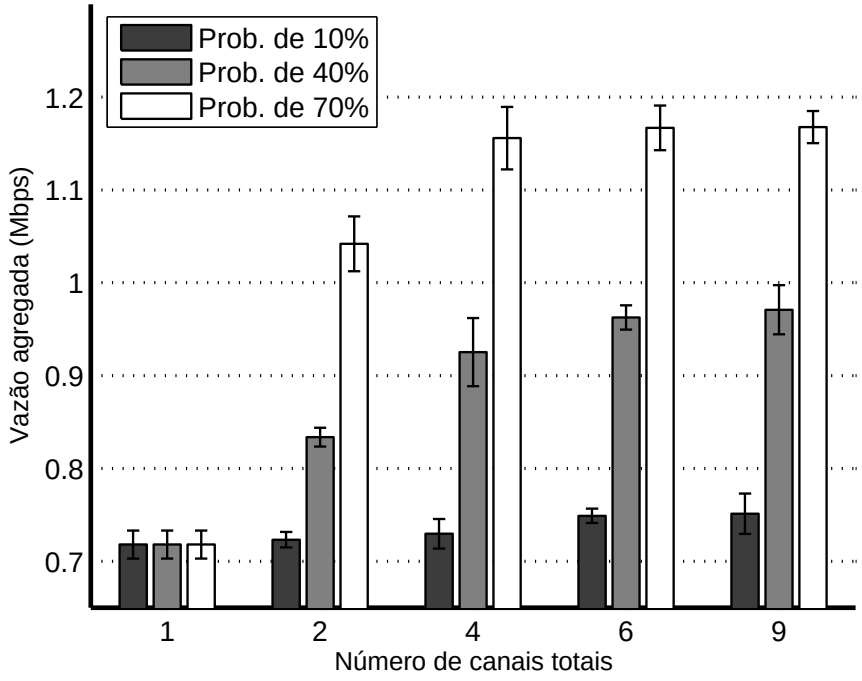


Figura 6.10: Vazão agregada versus número de canais totais com probabilidade de ausência do PU de 10%, 40% e 70% para uma rede de 40 nós.

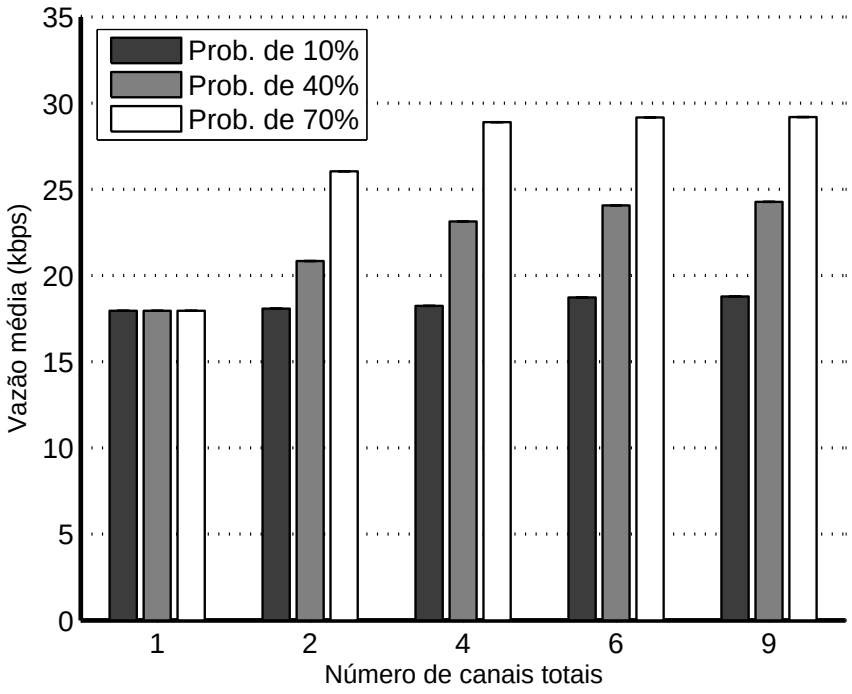


Figura 6.11: Vazão média dos nós na rede versus número de canais totais com probabilidade de ausência do PU de 10%, 40% e 70% para uma rede de 40 nós.

# Capítulo 7

# Conclusões

O presente estudo desenvolveu, implementou e avaliou o desempenho de um novo protocolo de controle de acesso ao meio (MAC) para redes sem fio *ad hoc* através da ferramenta de simulação de redes NS-3. O protocolo baseia-se em modificações no funcionamento do padrão IEEE 802.11 combinado com a técnica de rádios cognitivos. Com essa técnica procurou-se aumentar a vazão do sistema por meio do uso, simultâneo, de canais do padrão IEEE 802.11 (banda ISM) e do uso oportunista de outras faixas de frequência. O protocolo proposto tem duas grandes vantagens em relação as outras propostas apresentadas na literatura atualmente:

1. A facilidade com que ele pode ser implementado, uma vez que faz uso de grande parte da tecnologia do padrão IEEE 802.11;
2. O protocolo não depende exclusivamente da disponibilidade de espectro livre em outras bandas de frequência para o seu funcionamento.

Usando-se os procedimentos usuais do IEEE 802.11, os nós disputam o direito de usufruir do canal principal. Então, utilizando-se dos resultados da varredura nos canais cognitivos, os nós negociam a transmissão oportunista.

Para decidir sobre a ociosidade de um canal, transmissor e receptor trocam informações acerca da presença ou não de usuários primários. Isso foi possível através da inserção de novos campos nos cabeçalhos dos quadros RTS, CTS e ACK. Por decisão de projeto, escolheu-se pela abordagem mais simples quando do recebimento de pacotes, isto é, a recepção de pacotes pela interface cognitiva só é considerada válida se há a chegada bem-sucedida de pacotes pela interface principal.

O protocolo, conforme pretendido, alcançou um aumento na vazão, que pode ser verificado através dos dados coletados. Os resultados obtidos confirmam que a técnica de rádios cognitivos proporciona um uso mais racional e eficiente do espectro eletromagnético. Tal resultado se mostra importante dado que o espectro é um recurso limitado e há, no momento, uma massificação dos dispositivos de comunicação sem fio que compartilham tal recurso. O ganho obtido com a simulação do protocolo aproximou-se dos valores teoricamente esperados.

Por meio dos gráficos elaborados, é possível perceber que, para todos os cenários simulados, houve uma saturação na vazão dado o crescimento do número de canais. Tal fenônemo ocorreu

por volta do uso de seis canais, sendo cinco deles canais cognitivos. Isso leva a concluir que independentemente do tamanho da rede, não se faz necessário uma grande quantidade de canais cognitivos para se obter ganho apreciável.

Da forma como foi simulado, utilizando-se apenas os canais do próprio IEEE 802.11, mesmo no pior dos casos, quando o dispositivo se encontra nos canais intermediários, existem no máximo quatro canais para varredura, um valor próximo do valor de saturação. Assim, sem depender do canal alocado para o sistema, ainda é possível obter ganho próximo do limite de saturação.

Apesar do modelo proposto ter alcançado um ganho de vazão significativo (no melhor dos casos, o desempenho praticamente dobrou), ainda há limitações no modelo de simulação. Para trabalhos futuros, alguns aspectos deverão ser aprimorados.

O protocolo foi desenvolvido apenas para cenários de redes totalmente conectadas com único enlace. Para uso efetivo em uma rede do tipo *ad hoc*, o protocolo deveria ser testado em um cenário com múltiplos saltos, levando-se em consideração as oportunidades de espectro no alcance dos diferentes nós. Uma extensão possível deste trabalho seria negociar os canais disponíveis entre os pares de nós pertencentes a um caminho, e não diretamente entre emissor e receptor.

Para facilitar as negociações acerca do canal cognitivo, outra melhoria em relação ao protocolo apresentado seria o emissor informar um conjunto de canais ociosos e não apenas o primeiro encontrado durante a varredura. Como foi implementado, o protocolo tende a utilizar os primeiros canais verificados. Esse uso majoritário ocasiona um maior prejuízo aos UPs desses canais. Portanto, esta melhoria pode também levar a um balanceamento no uso dos canais.

A busca dos canais foi realizada de forma sequencial devido às limitações do próprio simulador. Este realiza suas operações como uma pilha de execução, não possibilitando uma busca mais eficiente pelos canais, como por exemplo, uma busca paralela, na qual canais são verificados simultaneamente. No futuro, verificar-se-á o impacto que os diferentes modos de varredura podem causar na vazão da rede.

No modelo desenvolvido, usou-se apenas uma interface cognitiva. Porém para trabalhos futuros, tal modelo pode ser aprimorado de forma a conter múltiplas interfaces. Da forma como foi realizado, o campo de CMA, permite ao receptor informar qual interface cognitiva ele deseja utilizar para transmissão. Isto pode facilitar a inserção de mais interfaces, pois basta o receptor informar ao emissor qual endereço físico da interface que será usada para a comunicação oportunista.

Por fim, outro fato não considerado foi o efeito da interferência causada pelos próprios USs. Usuários secundários pertencentes a redes diferentes, que coexistam na mesma região, porém transmitem em canais diferentes, podem descobrir a mesma oportunidade de espectro ocioso ocasionando interferência entre suas comunicações. Outra situação, não avaliada, seria no caso de redes *ad hoc* com múltiplos saltos, onde pode haver a interferência causada por nós distantes, utilizando a mesma banda do par transmissor-receptor.

# REFERÊNCIAS BIBLIOGRÁFICAS


[1] Federal Communications Commission, Spectrum Policy Task Force, Novembro 2002.

[2] BAHL, P.; CHANDRA, R.; MOSCIBRODA, T.; MURTY, R.; WELSH, M. White space networking with wi-fi like connectivity, *ACM SIGCOMM Computer Communication Review*, v. 39, n. 4, p. 27–38, Agosto 2009.

[3] HSU, A.C.-C.; WEI, D.S.L.; KUO, C.-C.J. A Cognitive MAC Protocol Using Statistical Channel Allocation for Wireless Ad-hoc Networks, *IEEE Communications Society*, p. 105–110, 2007.

[4] MA, L.; HAN, X.;SHEN, C.-C. Dynamic open spectrum sharing MAC protocol for wireless ad hoc networks, *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks 2005 DySPAN 2005*, p. 203–213, 2005.

[5] CORDEIRO, C.;CHALLAPALI, K.;BIRRU, D.;SHANKAR N, S. IEEE 802.22: An Introduction to the First Wireless Standard based on Cognitive Radios, *Journal of Communications*, v. 1, n. 1, p. 38–47, 2006.

[6] NS-3 Developers, Reference Manual. Disponível em: http://www.nsnam.org/tutorias.html. Acessado em: Junho de 2010.

[7] NS-3 Developers, Tutorial. Disponível em: http://www.nsnam.org/tutorias.html. Acessado em: Junho de 2010

[8] AKYILDIZ, I. F.;LEE, W.-Y.;CHOWDHURY, K. R. CRAHNs: Cognitive radio ad hoc networks, *Ad Hoc Networks*, v. 7, n. 5, p. 810–836, Elsevier B. V., 2009

[9] CORMIO, C.;CHOWDHURY, K. R. A survey on MAC protocols for cognitive radio networks, *Ad Hoc Networks*, v. 7, n. 7, p. 1315–1329, Elsevier B. V., 2009

[10] IEEE Standard 802.11-2007, Revision of IEEE Standard 802.11-1999. Disponível em http://standards.ieee.org/getieee802/download/802.11-2007.pdf. Acessado em: Setembro de 2010

# ANEXOS

# I. MODELO DE SIMULAÇÃO

```
/*Script que lê topologia e configura a rede wifi.
Amanda Marques da Silva - email: amandams@ymail.com
Pedro Henrique de Mesquita - email: ph.mesquita@yahoo.com.br
Engenharia de Redes - UnB*/

/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */

#include "ns3/core-module.h"
#include "ns3/common-module.h"
#include "ns3/node-module.h"
#include "ns3/helper-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/simulator-module.h"
#include "ns3/projeto-module.h"

#include <iostream>
#include <iomanip>
#include <fstream>
#include <vector>
#include <string>
#include <ctime>
#include <cstdio>
#include <cstdlib>
#include <stdexcept>

NS_LOG_COMPONENT_DEFINE ("WifiSimpleAdhoc");

using namespace ns3;

static uint32_t m_bytesTotal;

/***************************************************************
*              FUNÇÃO QUE CALCULA O THROUGHPUT                 *
***************************************************************/
static void Throughput () // Calculado em Mbps a cada 60s
{
  double mbps;
  Time time;
```

```
  mbps = (((m_bytesTotal * 8.0) / 1000000/60));
  time = Simulator::Now ();
  m_bytesTotal = 0;

  std::cout << time.GetSeconds() << " "<< mbps <<std::endl;
  Simulator::Schedule (Seconds (60), &Throughput);

}
/*************************************************************
*           FUNÇÃO QUE CALCULA O PAYLOAD NO RECEPTOR          *
*************************************************************/
void PhyRxOkTrace (std::string context, Ptr<const Packet> packet,
                   double snr,WifiMode mode, enum WifiPreamble preamble)
{
  Ptr<Packet> m_currentPacket;
  WifiMacHeader hdr;
  m_currentPacket = packet->Copy();
  m_currentPacket->RemoveHeader (hdr);

  if ((hdr.IsData()))
  {
    m_bytesTotal+= m_currentPacket->GetSize ();
  }
}
/*************************************************************
*                          FUNÇÃO MAIN                       *
*************************************************************/
int main (int argc, char *argv[])
{
  FILE *top;
  /*IEEE 802.11b, taxa de transmissão de 1Mbps*/
  std::string phyMode ("wifib-1mbs");
  /*Tamanho do pacote em bytes*/
  uint32_t packetSize = 1000;
  bool verbose = false;
  double px,py;
  int quantNo;
  int quantEnlace;

  /*Uso de argumentos em linha de comando que permitem alterar
    como os "scripts" sem a necessidade de editá-los*/
  CommandLine cmd;
  cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
```

```
cmd.AddValue ("packetSize", "size of application packet sent",
              packetSize);
cmd.AddValue ("verbose", "turn on all WifiNetDevice log components",
              verbose);
cmd.Parse (argc, argv);

/*Leitura do arquivo de topologia. O arquivo deve ter o seguinte formato:
quantidade_de_nós quantidade_de_enlaces
número_do_nó posição_x posição_y
....
número_do_nó_fonte número_do_nó_destino
....

Exemplo de uma toplogia de 3 nós, com 2 enlaces, onde os nós de número 1 e 2
transmitem para o nó número 0. O terreno é 10x10:
      3 2
      0 4.340135 0.724999
      1 4.658563 6.392632
      2 8.777354 9.785316
      1 0
      2 0
*/
top = fopen("projeto/topologia.txt","r");
fscanf(top, "%d %d",&quantNo,&quantEnlace);

/*Desabilitar fragmentação para quadros menores que 2200 bytes*/
Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
                    StringValue ("2200"));
/*Habilitar trocas de RTS/CTS para quadros maiores de 500 bytes*/
Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
                    StringValue ("500"));
/*Fixar taxas de dados não unicast para as mesmas que unicast*/
Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                    StringValue (phyMode));

/*Criação do contêiner de nós de acordo com a quantidade lida do arquivo*/
NodeContainer c;
c.Create (quantNo);

/*O conjunto de helpers abaixo ajuda na criação do núcleo de rede desejado*/
WifiHelper wifi;
if (verbose)
{
```

```
  wifi.EnableLogComponents ();  // Habilitar o modo de registro do Wifi
 }
 wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

 YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
 wifiPhy.Set ("RxGain", DoubleValue (0) );
 wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
 *Número de canal escolhido para a interface principal*/
 wifiPhy.Set("ChannelNumber", UintegerValue(1));

 /*Configuração do modelo de perda do canal*/
 Ptr<FriisPropagationLossModel> lossmodel =
                                CreateObject<FriisPropagationLossModel>();
 lossmodel->SetLambda(2.4e9,3e5);

 /*Configuração do canal*/
 Ptr<YansWifiChannel> wifiChannel = CreateObject <YansWifiChannel> ();
 wifiChannel->SetPropagationLossModel (lossmodel);
 wifiChannel->SetPropagationDelayModel
             (CreateObject<ConstantSpeedPropagationDelayModel>));

 wifiPhy.SetChannel (wifiChannel);

 /*Adiciona um MAC superior sem QoS e desabilita controle de taxa*/
 NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
 wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                               "DataMode",StringValue(phyMode),
                               "ControlMode",StringValue(phyMode));
 /*Configura o MAC para modo Ad Hoc*/
 wifiMac.SetType ("ns3::AdhocWifiMac");
 NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

/*Cria um modelo de mobilidade*/
 MobilityHelper mobility;
 Ptr<ListPositionAllocator> positionAlloc =
                            CreateObject<ListPositionAllocator> ();
 /*Lê a posição dos nós a partir do arquivo*/
 for(int i=0;i<quantNo;i++)
 {
   int no;
   fscanf(top,"%d %lf %lf",&no,&px,&py);
   positionAlloc->Add (Vector (px, py, 0.0));
 }
```

```
mobility.SetPositionAllocator (positionAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (c);

/*Adiciona a pilha de procolos aos nós*/
InternetStackHelper internet;
internet.Install (c);

/*Atibui enderaçamento IP ao nós*/
Ipv4AddressHelper ipv4;
NS_LOG_INFO ("Assign IP Addresses.");
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interface = ipv4.Assign (devices);


/*Configuração da segunda interface de modo similar ao realizado acima*/
ipv4.SetBase("192.1.1.0", "255.255.255.0");
WifiHelper cogwifi;
Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
                    StringValue ("500000"));
Config::SetDefault ("ns3::WifiRemoteStationManager::MaxSlrc",
                    StringValue ("0"));
cogwifi.SetStandard (WIFI_PHY_STANDARD_80211b);
NqosWifiMacHelper cogmac = NqosWifiMacHelper::Default();
wifiPhy.Set("ChannelNumber", UintegerValue(6));
Ptr<YansWifiChannel> CogwifiChannel = CreateObject <YansWifiChannel> ();
CogwifiChannel->SetPropagationLossModel (lossmodel);
CogwifiChannel->SetPropagationDelayModel
               (CreateObject<ConstantSpeedPropagationDelayModel>());
wifiPhy.SetChannel (CogwifiChannel);
cogwifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                 "DataMode",StringValue(phyMode),
                                 "ControlMode",StringValue(phyintervalMode));
cogmac.SetType ("ns3::AdhocWifiMac", "Sifs", TimeValue (NanoSeconds(26000)));
NetDeviceContainer cogdevices = cogwifi.Install (wifiPhy, cogmac, c);
Ipv4InterfaceContainer coginterface = ipv4.Assign (cogdevices);
interface.Add(coginterface);


/*Criação das aplicações*/
float tempo = 0.01;
for(int i=0;i<quantEnlace;i++)
{
```

```
    int from, to;
    fscanf(top,"%d %d %lf",&from,&to);

    PacketSinkHelper sink("ns3::UdpSocketFactory",
                          InetSocketAddress(interface.GetAddress(to), 80));
    ApplicationContainer sinkApp = sink.Install(c.Get(to));
    sinkApp.Start(Seconds(tempo));
    sinkApp.Stop(Seconds(50.0));

    OnOffHelper onOff ("ns3::UdpSocketFactory",
                       InetSocketAddress(interface.GetAddress(from), 80));
    onOff.SetAttribute ("OnTime", RandomVariableValue (ConstantVariable (1.0)));
    onOff.SetAttribute ("OffTime", RandomVariableValue (ConstantVariable (0.0)));
    onOff.SetAttribute ("PacketSize", UintegerValue(packetSize));
    onOff.SetAttribute ("Remote",
                        AddressValue(InetSocketAddress(interface.GetAddress(to),
                        80)));
    ApplicationContainer udpApp = onOff.Install(c.Get(from));
    udpApp.Start(Seconds(tempo));
    udpApp.Stop(Seconds(50.0));
    tempo+=0.2;
  }
  /*Hablita a criação do arquivos de ratreamento do pacotes*/
  AsciiTraceHelper ascii;
  wifiPhy.EnableAsciiAll (ascii.CreateFileStream ("teste11.tr"));

  /*Rastreia os pacotes recebidos no terminal escolhido*/
  Config::Connect ("/NodeList/2/DeviceList/*/Phy/State/RxOk",
                   MakeCallback(&PhyRxOkTrace));
  Throughput ();

  /*Cria os arquivos de captura de pacote (PCAP) para todos os dispositivos*/
  wifiPhy.EnablePcap ("wifi-simple-adhoc", devices);
  wifiPhy.EnablePcap ("cog-wifi-simple-adhoc", cogdevices);

  Simulator::Stop(Seconds(60.1));
  Simulator::Run ();
  Simulator::Destroy ();

  fclose(top);

  return 0;
}
```

# II. MÓDULO DE ENVIO

## II.1 dispatch-module.h

```
// -*- Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*-

#ifndef DISPATCHMODULE_H
#define DISPATCHMODULE_H

#include <string>
#include <list>
#include "ns3/packet.h"
#include "ns3/object.h"
#include "ns3/ptr.h"
#include "ns3/traced-callback.h"
#include "ns3/ipv4-interface.h"
#include "ns3/ipv4-header.h"
#include <queue>

namespace ns3 {

class DispatchModule
{
public:
  static TypeId GetTypeId (void);
  DispatchModule ();

  ~DispatchModule ();

  void Enqueue (Ptr<Packet> p, Ipv4Header iph);
  void Send (void);
  void CognitiveSend (void);
  void SetMainInterface (Ptr<Ipv4Interface> main_out);
  void SetCognitiveInterface(Ptr<Ipv4Interface> cog_out);
  Ptr<Ipv4Interface> DoGetCognitiveInterface(void);
  bool IsEmpty(void);
  Mac48Address DoGetMacAddr (void);
  void DoPopQueue (void);
  Ipv4Address GetMainAddress (void);
  Ipv4Address DoGetCognitiveIp (void);
```

```
  void SetFlagAck(bool flag);
  bool GetFlagAck(void);

  std::queue<Ptr<Packet> > fila_pacote;
  std::queue<Ipv4Header> fila_header;
  Ptr<Ipv4Interface> mainInterface;
  Ptr<Ipv4Interface> cogInterface;
  bool FlagAck;

};

} // namespace ns3

#endif /* DISPATCHMODULE_H */
```

## II.2 dispatch-module.cc

```
// -*- Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*-
//
#include "ns3/log.h"
#include "ns3/trace-source-accessor.h"
#include "dispatch-module.h"
#include "ns3/ipv4-address.h"

NS_LOG_COMPONENT_DEFINE ("DispatchModule");

namespace ns3 {

NS_OBJECT_ENSURE_REGISTERED (DispatchModule);

TypeId
DispatchModule::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::DispatchModule")
    .SetParent<Object> ()
    ;
  return tid;
}

DispatchModule::DispatchModule()
{
  NS_LOG_FUNCTION_NOARGS ();
```

```
}

DispatchModule::~DispatchModule()
{
  NS_LOG_FUNCTION_NOARGS ();
}
/*Enfilera os pacotes na pré-fila*/
void DispatchModule::Enqueue (Ptr<Packet> p, Ipv4Header iph)
{
  fila_pacote.push(p);
  fila_header.push(iph);
}
/*Encaminha os pacotes para a interface principal*/
void DispatchModule::Send (void)
{
  if(!fila_pacote.empty())
  {
    FlagAck = false;
    Ptr<Packet> packet = fila_pacote.front();
    fila_pacote.pop();

    Ipv4Header ipHeader = fila_header.front();
    fila_header.pop();
    packet->AddHeader (ipHeader);

    if (mainInterface->IsUp ())
    {
      mainInterface->Send (packet, ipHeader.GetDestination ());
    }
    else
    {
      NS_LOG_LOGIC ("Dropping -- outgoing interface is down: "
                    << ipHeader.GetDestination ());
    }
  }
  else
  {
    FlagAck = true;
  }
}
/*Encaminha os pacotes para a interface cognitiva*/
void DispatchModule::CognitiveSend (void)
{
```

```
  if(!fila_pacote.empty())
  {
    Ptr<Packet> p = fila_pacote.front();
    Ptr<Packet> packet = p->Copy ();
    Ipv4Header ipHeader = fila_header.front();

    uint32_t addr = ipHeader.GetDestination().Get();
    uint32_t mask = cogInterface->GetIpMask().Get();
    uint32_t cogNet = (cogInterface->GetIpAddr().Get()) & (mask);
    uint32_t node = (addr) & (~mask);
    uint32_t cogAddr = node | cogNet;
    Ipv4Address cogIp;
    cogIp.Set(cogAddr);

    ipHeader.SetDestination (cogIp);
    ipHeader.SetSource(cogInterface->GetIpAddr());
    packet->AddHeader (ipHeader);

    if (cogInterface->IsUp ())
    {
      cogInterface->Send (packet, ipHeader.GetDestination ());
    }
    else
    {
      NS_LOG_LOGIC ("Dropping -- outgoing interface is down: "
                    << ipHeader.GetDestination ());
    }
  }
}

void DispatchModule::SetMainInterface (Ptr<Ipv4Interface> main_out)
{
  mainInterface = main_out;
}

void DispatchModule::SetCognitiveInterface (Ptr<Ipv4Interface> cog_out)
{
  cogInterface = cog_out;
}

Ptr<Ipv4Interface> DispatchModule::DoGetCognitiveInterface(void)
{
  return cogInterface;
```

```
}

bool DispatchModule::IsEmpty(void)
{
  return fila_pacote.empty();
}

Mac48Address DispatchModule::DoGetMacAddr (void)
{
  return cogInterface->GetMacAddr();
}

void DispatchModule::DoPopQueue (void)
{
  fila_pacote.pop();
  fila_header.pop();
}

Ipv4Address DispatchModule::GetMainAddress (void)
{
  return mainInterface->GetIpAddr();
}

Ipv4Address DispatchModule::DoGetCognitiveIp (void)
{
  Ipv4Header hdr = fila_header.front();
  return hdr.GetDestination();
}

void DispatchModule::SetFlagAck(bool flag)
{
  FlagAck = flag;
}

bool DispatchModule::GetFlagAck(void)
{
  return FlagAck;
}

} // namespace ns3
```

# III. IPV4 L3 PROTOCOL

## III.1 ipv4-l3-protocol.h

```
//Novas variáveis e novos métodos

+ public:
+  DispatchModule pre_fila;
+  uint16_t cogAck;
+  int IdList[10];
+  Ipv4Address Ipv4AddressList[10];
+  int position;
+  int found;
+
+  void DoSend (void);
+  void DoCognitiveSend (void);
+  Mac48Address GetMacAddr (void);
+  void SetInterfaces (void);
+  Ptr<Ipv4Interface> GetCognitiveInterface (void);
+  void SetCognitiveAck (uint16_t ack);
+  uint16_t GetCognitiveAck (void);
+  void PopQueue (void);
+  Ipv4Address GetCognitiveIp (void);
```

## III.2 ipv4-l3-protocol.cc

```
@@void Ipv4L3Protocol::Receive(...)
{
  ...
  ...
  uint32_t interface = 0;
  Ptr<Packet> packet = p->Copy ();

+  Ipv4Header ipHeader;
+  if (Node::ChecksumEnabled ())
+    {
+      ipHeader.EnableChecksum ();
+    }
+  packet->RemoveHeader (ipHeader);
+  ipHeader.SetDestination(pre_fila.GetMainAddress());
```

```
+  uint32_t teste = ((255 & ipHeader.GetSource().Get()) | (167837952));
+  Ipv4Address add;
+  add.Set(teste);
+  found = 0;
+  for(int i =0; i<position; i++)
+  {
+
+    if(add==Ipv4AddressList[i])
+    {
+      found = 1;
+      if(ipHeader.GetIdentification()!=IdList[i])
+      {
+        //DROP
+        SetCognitiveAck (0);
+        return;
+      }
+      else
+      {
+        IdList[i]++;
+      }
+    }
+  }
+  if(found==0)
+  {
+    Ipv4AddressList[position] = ipHeader.GetSource();
+    IdList[position] = 1;
+    position ++;
+  }
+  found = 0;
   ...
   ...
- Ipv4Header ipHeader;
- if (Node::ChecksumEnabled ())
-   {
-     ipHeader.EnableChecksum ();
-   }
- packet->RemoveHeader (ipHeader);
   ...
   ...
}

@@ void Ipv4L3Protocol::SendRealOut (...)
{
```

```
  if (route == 0)
    {
      NS_LOG_WARN ("No route to host.  Drop.");
      m_dropTrace (ipHeader, packet, DROP_NO_ROUTE,
                   m_node->GetObject<Ipv4> (), 0);
      return;
    }
-  packet->AddHeader (ipHeader);
-  Ptr<NetDevice> outDev = route->GetOutputDevice ();
-  int32_t interface = GetInterfaceForDevice (outDev);
-
-  NS_ASSERT (interface >= 0);
-  Ptr<Ipv4Interface> outInterface = GetInterface (interface);

+  Ptr<NetDevice> outDev = route->GetOutputDevice ();
+  int32_t interface = GetInterfaceForDevice (outDev);
+  NS_ASSERT (interface >= 0);
+  Ptr<Ipv4Interface> outInterface = GetInterface (interface);
+  pre_fila.SetMainInterface(outInterface);
+  Ptr<Ipv4Interface> outInterface2 = GetInterface (2);
+  pre_fila.SetCognitiveInterface(outInterface2);
+  pre_fila.Enqueue(packet,ipHeader);

if (!route->GetGateway ().IsEqual (Ipv4Address ("0.0.0.0")))
    {
      if (outInterface->IsUp ())
        {
          NS_LOG_LOGIC ("Send to gateway " << route->GetGateway ());
          m_txTrace (packet, m_node->GetObject<Ipv4> (), interface);
          outInterface->Send (packet, route->GetGateway ());
        }
      else
        {
          NS_LOG_LOGIC ("Dropping -- outgoing interface is down: "
                        << route->GetGateway ());
          Ipv4Header ipHeader;
          packet->RemoveHeader (ipHeader);
          m_dropTrace (ipHeader, packet, DROP_INTERFACE_DOWN,
                       m_node->GetObject<Ipv4> (), interface);
        }
    }
  else
    {
```

```
-     if (outInterface->IsUp ())
-       {
-         NS_LOG_LOGIC ("Send to destination " << ipHeader.GetDestination ());
-         m_txTrace (packet, m_node->GetObject<Ipv4> (), interface);
-         outInterface->Send (packet, ipHeader.GetDestination ());
-       }
-     else
-       {
-         NS_LOG_LOGIC ("Dropping -- outgoing interface is down: "
                        << ipHeader.GetDestination ());
-         Ipv4Header ipHeader;
-         packet->RemoveHeader (ipHeader);
-         m_dropTrace (ipHeader, packet, DROP_INTERFACE_DOWN,
                      m_node->GetObject<Ipv4> (), interface);
-       }

+     if(ipHeader.GetIdentification() == 0 )
+     {
+       pre_fila.Send();
+     }
+     else if(pre_fila.GetFlagAck())
+     {
+       pre_fila.Send();
+     }
    }
}

//Novos métodos

+ void Ipv4L3Protocol::DoSend (void)
+ {
+   pre_fila.Send();
+ }
+
+ void Ipv4L3Protocol::DoCognitiveSend (void)
+ {
+   pre_fila.CognitiveSend();
+ }
+
+ Mac48Address Ipv4L3Protocol::GetMacAddr (void)
+ {
+   return pre_fila.DoGetMacAddr();
+ }
```

```
+
+ void Ipv4L3Protocol::SetInterfaces (void)
+ {
+   Ptr<Ipv4Interface> outInterface = GetInterface (1);
+   pre_fila.SetMainInterface(outInterface);
+   outInterface = GetInterface (2);
+   pre_fila.SetCognitiveInterface(outInterface);
+ }
+
+ Ptr<Ipv4Interface> Ipv4L3Protocol::GetCognitiveInterface(void)
+ {
+   return pre_fila.DoGetCognitiveInterface();
+ }
+
+ void Ipv4L3Protocol::SetCognitiveAck (uint16_t ack)
+ {
+   cogAck = ack;
+ }
+
+ uint16_t Ipv4L3Protocol::GetCognitiveAck (void)
+ {
+   return cogAck;
+ }
+
+ void Ipv4L3Protocol::PopQueue (void)
+ {
+   pre_fila.DoPopQueue();
+ }
+
+ Ipv4Address Ipv4L3Protocol::GetCognitiveIp (void)
+ {
+   return pre_fila.DoGetCognitiveIp();
+ }
```

# IV. IPV4 INTERFACE

## IV.1 ipv4-interface.h

```
//Novos métodos
+ public:
+   Ipv4Address GetIpAddr (void);
+   Ipv4Mask GetIpMask (void);
+   Mac48Address GetMacAddr (void);
+   void SetCache (Mac48Address addr, Ptr<Packet> packet, Ipv4Address ipAddr);
```

## IV.2 ipv4-interface.cc

```
//Novos métodos
+ Ipv4Address Ipv4Interface::GetIpAddr (void)
+ {
+   Ipv4InterfaceAddress ip = m_ifaddrs.front();
+   return ip.GetLocal();
+ }
+ Ipv4Mask Ipv4Interface::GetIpMask (void)
+ {
+   Ipv4InterfaceAddress ip = m_ifaddrs.front();
+   return ip.GetMask();
+ }
+ Mac48Address Ipv4Interface::GetMacAddr (void)
+ {
+   return m_device->DoGetMacAddr ();
+ }
+ void Ipv4Interface::SetCache (Mac48Address addr, Ptr<Packet> packet,
                                Ipv4Address ipAddr)
+ {
+   ArpCache::Entry *entry = m_cache->Lookup(ipAddr);
+   if(entry == 0)
+   {
+     entry = m_cache->Add(ipAddr);
+     entry->MarkWaitReply(packet);
+     entry = m_cache->Lookup(ipAddr);
+     entry->MarkAlive(addr);
+   }
+ }
```

# V. WIFI NET DEVICE

## V.1 wifi-net-device.h

```
//Novos métodos
+ public:
+   virtual Mac48Address DoGetMacAddr (void);
```

## V.2 wifi-net-device.cc

```
@@ void WifiNetDevice::DoStart (void)
{
   m_phy->Start ();
   m_mac->Start ();
   m_stationManager->Start ();
   NetDevice::DoStart ();
+  m_mac->SetNode(m_node);
}

//Novos métodos
+ Mac48Address WifiNetDevice::DoGetMacAddr (void)
+ {
+   return m_mac->GetAddress();
+ }
```

## V.3 net-device.h

```
//Novos métodos
+ public:
+   virtual Mac48Address DoGetMacAddr (void) = 0;
```

# VI. ADHOC WIFI MAC

## VI.1 adhoc-wifi-mac.h

```
//Novas variáveis e novos métodos
+ public:
+   Ptr<Node> m_node;
+   virtual void SetNode(Ptr<Node> node);
```

## VI.2 adhoc-wifi-mac.cc

```
//Novos métodos
+ void AdhocWifiMac::SetNode (Ptr<Node> node)
+ {
+   m_dca->SetNode(node);
+ }
```

## VI.3 wifi-mac.h

```
//Novos métodos
+ public:
+   Ptr<Node> m_node;
+   virtual void SetNode (Ptr<Node> node) = 0;
```

# VII. DCA TXOP

## VII.1 dca-txop.h

```
//Novas variávies e novos métodos
+ public:
+   Ptr<Node> m_node;
+   void SetNode (Ptr<Node>);
```

## VII.2 dca-txop.cc

```
//Novos métodos
+ void DcaTxop::SetNode (Ptr<Node> node)
+ {
+   m_low->SetNode(node);
+ }
```

# VIII. MAC LOW

## VIII.1 mac-low.h

```
//Novas variáveis e novos métodos
+ public:
+   void SetNode (Ptr<Node>);
+   Ptr<Node> m_node;
+   int32_t cogChannel;
+   int flag;
```

## VIII.2 mac-low.cc

```
@@ void MacLow::ReceiveOk (...)
 {
 ...
 ...
  if (hdr.IsRts ())
    {
      if (isPrevNavZero &&
          hdr.GetAddr1 () == m_self)
        {
          NS_LOG_DEBUG ("rx RTS from=" << hdr.GetAddr2 () << ",
                        schedule CTS");
          NS_ASSERT (m_sendCtsEvent.IsExpired ());
          m_stationManager->ReportRxOk (hdr.GetAddr2 (), &hdr,
                                        rxSnr, txMode);
+         cogChannel = hdr.GetRtsCognitiveChannel();
          ...
        }
      else
        {
          NS_LOG_DEBUG ("rx RTS from=" << hdr.GetAddr2 () << ",
                        cannot schedule CTS");
        }
    }
 else if (hdr.IsCts () &&
           hdr.GetAddr1 () == m_self &&
           m_ctsTimeoutEvent.IsRunning () &&
           m_currentPacket != 0)
```

```
 {
       ...
       ...
       m_sendDataEvent = Simulator::Schedule (GetSifs (),
                                              &MacLow::SendDataAfterCts, this,
                                              hdr.GetAddr1 (),
                                              hdr.GetDuration (),
                                              txMode);
+      if((cogChannel != 0)&&(hdr.GetCtsCognitiveChannel() == cogChannel))
+      {
+        Ptr<Ipv4> ip = m_node->GetObject<Ipv4>();
+        Ptr<Ipv4L3Protocol> aux = ip->GetObject<Ipv4L3Protocol>();
+        Ipv4Address ipaddr = aux->GetCognitiveIp();
+        Ptr<Ipv4Interface> cogInterface = aux->GetCognitiveInterface();
+
+        uint32_t addr = ipaddr.Get();
+        uint32_t mask = cogInterface->GetIpMask().Get();
+        uint32_t cogNet = (cogInterface->GetIpAddr().Get()) & (mask);
+        uint32_t node = (addr) & (~mask);
+        uint32_t cogAddr = node | cogNet;
+        Ipv4Address cogIp;
+        cogIp.Set(cogAddr);
+
+        cogInterface->SetCache(hdr.GetCognitiveAddress(),packet,cogIp);
+
+        Simulator::Schedule (GetSifs (),&Ipv4L3Protocol::DoCognitiveSend, aux);
+      }
 }
 else if (hdr.IsAck () &&
          hdr.GetAddr1 () == m_self &&
          (m_normalAckTimeoutEvent.IsRunning () ||
          m_fastAckTimeoutEvent.IsRunning () ||
          m_superFastAckTimeoutEvent.IsRunning ()) &&
          m_txParams.MustWaitAck ())
 {
    ...
    ...
    bool gotAck = false;
+    Ptr<Ipv4> ip = m_node->GetObject<Ipv4>();
+    Ptr<Ipv4L3Protocol> aux = ip->GetObject<Ipv4L3Protocol>();
+    if(hdr.GetCognitiveAck()==1)
+    {
+      aux->PopQueue();
```

```
+    }
+    if(flag == 0)
+    {
+      flag = 1;
+    }
+    else
+    {
+      aux->DoSend();
+    }
  }
  ...
  ...
 else if (hdr.IsData () || hdr.IsMgt ())
 {
     NS_LOG_DEBUG ("rx unicast/sendAck from=" << hdr.GetAddr2 ());
     NS_ASSERT (m_sendAckEvent.IsExpired ());
-     m_sendAckEvent = Simulator::Schedule (GetSifs (),
-                                           &MacLow::SendAckAfterData, this,
-                                           hdr.GetAddr2 (),
-                                           hdr.GetDuration (),
-                                           txMode,
-                                           rxSnr);
-
+     Ptr<Ipv4> ip = m_node->GetObject<Ipv4>();
+     Ptr<Ipv4L3Protocol> aux = ip->GetObject<Ipv4L3Protocol>();
+
+     if(hdr.GetAddr1()!=aux->GetMacAddr())
+     {
+       m_sendAckEvent = Simulator::Schedule (GetSifs (),
+                                             &MacLow::SendAckAfterData, this,
+                                             hdr.GetAddr2 (),
+                                             hdr.GetDuration (),
+                                             txMode,
+                                             rxSnr);
+     }
+     else if(hdr.GetAddr1()==aux->GetMacAddr())
+     {
+       aux->SetCognitiveAck (1);
+     }
 }
 ...
 ...
}
```

```
@@ void MacLow::SendRtsForPacket (...)
 {
    ...
    rts.SetAddr1 (m_currentHdr.GetAddr1 ());
    rts.SetAddr2 (m_self);

+    UniformVariable x;
+    for(int i=6;i<=6;i++)
+    {
+      if(x.GetValue()<0)
+      {
+        cogChannel = i;
+        rts.SetRtsCognitiveChannel(cogChannel);
+        break;
+      }
+      else
+      {
+        rts.SetRtsCognitiveChannel(0);
+      }
+    }
 ...
 ...
 Time txDuration = m_phy->CalculateTxDuration (GetRtsSize (),
                                               rtsTxMode, WIFI_PREAMBLE_LONG);
 Time timerDelay = txDuration + GetCtsTimeout ();
+ timerDelay += NanoSeconds(100000);
 NS_ASSERT (m_ctsTimeoutEvent.IsExpired ());
 ...
 }
@@ void MacLow::SendCtsAfterRts (...)
 {
  ...
  cts.SetNoRetry ();
  cts.SetAddr1 (source);

+  UniformVariable x;
+  Ptr<Ipv4> ip = m_node->GetObject<Ipv4>();
+  Ptr<Ipv4L3Protocol> aux = ip->GetObject<Ipv4L3Protocol>();
+  aux->SetInterfaces();
+  if((x.GetValue()<0)&&(cogChannel!=0))
+  {
+    cts.SetCognitiveAddress(aux->GetMacAddr());
+    cts.SetCtsCognitiveChannel(cogChannel);
```

```
+  }
+  else
+  {
+    cts.SetCognitiveAddress("00:00:00:00:00:00");
+    cts.SetCtsCognitiveChannel(0);
+  }
+  cogChannel = 0;

  duration -= GetCtsDuration (source, rtsTxMode);
  duration -= GetSifs ();
  ...
 }

@@ void MacLow::SendAckAfterData (...)
 {
   ...
   ack.SetNoMoreFragments ();
   ack.SetAddr1 (source);

+   Ptr<Ipv4> ip = m_node->GetObject<Ipv4>();
+   Ptr<Ipv4L3Protocol> aux = ip->GetObject<Ipv4L3Protocol>();
+   ack.SetCognitiveAck(aux->GetCognitiveAck());
+   aux->SetCognitiveAck(0);

   duration -= GetSifs ();
   NS_ASSERT (duration >= MicroSeconds (0));
   ...
  }

//Novos métodos
+ void MacLow::SetNode (Ptr<Node> node)
+ {
+   m_node = node;
+ }
```

# IX. WIFI MAC HEADER

## IX.1 wifi-mac-header.h

```
//Novas variáveis e métodos
+ public:
+   void SetRtsCognitiveChannel (uint16_t cogChannel);
+   void SetCtsCognitiveChannel (uint16_t cogChannel);
+   uint16_t GetRtsCognitiveChannel(void);
+   uint16_t GetCtsCognitiveChannel(void);
+   void SetCognitiveAddress (Mac48Address cogAddress);
+   Mac48Address GetCognitiveAddress (void);
+   void SetCognitiveAck (uint16_t ack);
+   uint16_t GetCognitiveAck (void);
+
+   uint16_t RAC;
+   uint16_t CAC;
+   uint16_t OPACK;
+   Mac48Address CMA;
```

## IX.2 wifi-mac-header.cc

```
@@ uint32_t WifiMacHeader::GetSize (void) const
  {
   ...
   case TYPE_CTL:
    switch (m_ctrlSubtype) {
-     case SUBTYPE_CTL_RTS:
-      size = 2+2+6+6;
-      break;
-     case SUBTYPE_CTL_CTS:
-     case SUBTYPE_CTL_ACK:
-      size = 2+2+6;
-      break;
+    case SUBTYPE_CTL_RTS:
+      size = 2+2+6+6+2;
+      break;
+    case SUBTYPE_CTL_CTS:
+      size = 2+2+6+6+2;
+      break;
```

```
+    case SUBTYPE_CTL_ACK:
+      size = 2+2+6+2;
+      break;
     case SUBTYPE_CTL_BACKREQ:
     ...
  }

@@ void WifiMacHeader::Serialize (Buffer::Iterator i) const
  {
    ...
    case TYPE_CTL:
     switch (m_ctrlSubtype) {
-      case SUBTYPE_CTL_RTS:
-       WriteTo (i, m_addr2);
-       break;
-      case SUBTYPE_CTL_CTS:
-      case SUBTYPE_CTL_ACK:
-       break;
+     case SUBTYPE_CTL_RTS:
+       WriteTo (i, m_addr2);
+       i.WriteHtolsbU16 (RAC);
+       break;
+     case SUBTYPE_CTL_CTS:
+       WriteTo (i, CMA);
+       i.WriteHtolsbU16 (CAC);
+       break;
+     case SUBTYPE_CTL_ACK:
+       i.WriteHtolsbU16 (OPACK);
+       break;
     case SUBTYPE_CTL_BACKREQ:
    ...
  }

@@ uint32_t WifiMacHeader::Deserialize (Buffer::Iterator start)
  {
   ...
   case TYPE_CTL:
   switch (m_ctrlSubtype) {
-    case SUBTYPE_CTL_RTS:
-      ReadFrom (i, m_addr2);
-      break;
-    case SUBTYPE_CTL_CTS:
-    case SUBTYPE_CTL_ACK:
```

```
-      break;
+    case SUBTYPE_CTL_RTS:
+      ReadFrom (i, m_addr2);
+      RAC = i.ReadLsbtohU16 ();
+      break;
+    case SUBTYPE_CTL_CTS:
+      ReadFrom (i, CMA);
+      CAC = i.ReadLsbtohU16 ();
+      break;
+    case SUBTYPE_CTL_ACK:
+      OPACK = i.ReadLsbtohU16 ();
+      break;
    case SUBTYPE_CTL_BACKREQ:
    ...
  }

//Novos métodos
+ void WifiMacHeader::SetRtsCognitiveChannel (uint16_t cogchannel)
+ {
+   RAC = cogchannel;
+ }
+ void WifiMacHeader::SetCtsCognitiveChannel (uint16_t cogchannel)
+ {
+   CAC = cogchannel;
+ }
+ uint16_t WifiMacHeader::GetRtsCognitiveChannel (void)
+ {
+   return RAC;
+ }
+ uint16_t WifiMacHeader::GetCtsCognitiveChannel (void)
+ {
+   return CAC;
+ }
+ void WifiMacHeader::SetCognitiveAddress (Mac48Address cogAddress)
+ {
+   CMA = cogAddress;
+ }
+ Mac48Address WifiMacHeader::GetCognitiveAddress (void)
+ {
+   return CMA;
+ }
+ void WifiMacHeader::SetCognitiveAck (uint16_t ack)
+ {
```

```
+   OPACK = ack;
+ }
+ uint16_t WifiMacHeader::GetCognitiveAck (void)
+ {
+   return OPACK;
+ }
```

# X. DADOS COLETADOS

## X.1 Topologia de 10 nós

### X.1.1 Probabilidade de 0%

Tabela X.1: Vazões agregadas coletadas para $P_d = 0\%$ e 10 nós

| Nº de canais | Seed=1 | Seed=8 | Seed=12 | Seed=27 | Seed=33 | Média | Int. de confiança |
|---|---|---|---|---|---|---|---|
| 2 | 0,761200 | 0,766667 | 0,764667 | 0,764800 | 0,765733 | 0,764613 | 0,005143 |
| 4 | 0,761200 | 0,766667 | 0,764667 | 0,764800 | 0,765733 | 0,764613 | 0,005143 |
| 6 | 0,761200 | 0,766667 | 0,764667 | 0,764800 | 0,765733 | 0,764613 | 0,005143 |
| 9 | 0,761200 | 0,766667 | 0,764667 | 0,764800 | 0,765733 | 0,764613 | 0,005143 |

### X.1.2 Probabilidade de 10%

Tabela X.2: Vazões agregadas coletadas para $P_d = 10\%$ e 10 nós

| Nº de canais | Seed=1 | Seed=8 | Seed=12 | Seed=27 | Seed=33 | Média | Int. de confiança |
|---|---|---|---|---|---|---|---|
| 2 | 0,771067 | 0,770267 | 0,771600 | 0,773200 | 0,771600 | 0,771547 | 0,002664 |
| 4 | 0,785467 | 0,782800 | 0,784133 | 0,785467 | 0,785067 | 0,784587 | 0,002825 |
| 6 | 0,789867 | 0,795067 | 0,791600 | 0,794133 | 0,793733 | 0,792880 | 0,005238 |
| 9 | 0,803467 | 0,808933 | 0,784533 | 0,811467 | 0,806400 | 0,802960 | 0,026617 |

### X.1.3 Probabilidade de 40%

Tabela X.3: Vazões agregadas coletadas para $P_d = 40\%$ e 10 nós

| Nº de canais | Seed=1 | Seed=8 | Seed=12 | Seed=27 | Seed=33 | Média | Int. de confiança |
|---|---|---|---|---|---|---|---|
| 2 | 0,882400 | 0,885600 | 0,882133 | 0,892133 | 0,885867 | 0,885627 | 0,010001 |
| 4 | 0,976933 | 1,010400 | 0,949600 | 0,997467 | 1,006267 | 0,988133 | 0,062346 |
| 6 | 1,034133 | 1,043067 | 1,032267 | 1,037200 | 1,039067 | 1,037147 | 0,010207 |
| 9 | 1,054667 | 1,056000 | 1,048133 | 1,050533 | 1,061333 | 1,054133 | 0,012695 |

### X.1.4 Probabilidade de 70%

Tabela X.4: Vazões agregadas coletadas para $P_d = 70\%$ e 10 nós

| Nº de canais | Seed=1 | Seed=8 | Seed=12 | Seed=27 | Seed=33 | Média | Int. de confiança |
|---|---|---|---|---|---|---|---|
| 2 | 1,140667 | 1,124267 | 1,113333 | 1,134800 | 1,114533 | 1,125520 | 0,030061 |
| 4 | 1,272667 | 1,256267 | 1,260400 | 1,279600 | 1,255200 | 1,264827 | 0,026778 |
| 6 | 1,285467 | 1,281200 | 1,269867 | 1,284267 | 1,269467 | 1,278053 | 0,019401 |
| 9 | 1,281200 | 1,281333 | 1,279600 | 1,289467 | 1,276267 | 1,281573 | 0,012072 |

## X.2 Topologia de 40 nós

### X.2.1 Probabilidade de 0%

Tabela X.5: Vazões agregadas coletadas para $P_d = 0\%$ e 40 nós

| Nº de canais | Seed=1 | Seed=8 | Seed=12 | Seed=27 | Seed=33 | Média | Int. de confiança |
|---|---|---|---|---|---|---|---|
| 2 | 0,720267 | 0,713333 | 0,699467 | 0,726667 | 0,730133 | 0,717973 | 0,030225 |
| 4 | 0,720267 | 0,713333 | 0,699467 | 0,726667 | 0,730133 | 0,717973 | 0,030225 |
| 6 | 0,720267 | 0,713333 | 0,699467 | 0,726667 | 0,730133 | 0,717973 | 0,030225 |
| 9 | 0,720267 | 0,713333 | 0,699467 | 0,726667 | 0,730133 | 0,717973 | 0,030225 |

### X.2.2 Probabilidade de 10%

Tabela X.6: Vazões agregadas coletadas para $P_d = 10\%$ e 40 nós

| Nº de canais | Seed=1 | Seed=8 | Seed=12 | Seed=27 | Seed=33 | Média | Int. de confiança |
|---|---|---|---|---|---|---|---|
| 2 | 0,722667 | 0,718133 | 0,728267 | 0,715333 | 0,731600 | 0,723200 | 0,016843 |
| 4 | 0,729467 | 0,742933 | 0,735200 | 0,731867 | 0,708400 | 0,729573 | 0,031983 |
| 6 | 0,756400 | 0,752533 | 0,750800 | 0,743200 | 0,741733 | 0,748933 | 0,015551 |
| 9 | 0,724933 | 0,764000 | 0,760667 | 0,764933 | 0,741200 | 0,751147 | 0,043575 |

### X.2.3 Probabilidade de 40%

Tabela X.7: Vazões agregadas coletadas para $P_d = 40\%$ e 40 nós

| Nº de canais | Seed=1 | Seed=8 | Seed=12 | Seed=27 | Seed=33 | Média | Int. de confiança |
|---|---|---|---|---|---|---|---|
| 2 | 0,882400 | 0,885600 | 0,882133 | 0,892133 | 0,885867 | 0,885627 | 0,020187 |
| 4 | 0,936800 | 0,935067 | 0,874667 | 0,927733 | 0,951733 | 0,925200 | 0,073403 |
| 6 | 0,972933 | 0,962667 | 0,951467 | 0,952533 | 0,973333 | 0,962587 | 0,026256 |
| 9 | 0,981733 | 0,988533 | 0,944133 | 0,974667 | 1,001200 | 0,978053 | 0,052980 |

### X.2.4 Probabilidade de 70%

Tabela X.8: Vazões agregadas coletadas para $P_d = 70\%$ e 40 nós

| Nº de canais | Seed=1 | Seed=8 | Seed=12 | Seed=27 | Seed=33 | Média | Int. de confiança |
|---|---|---|---|---|---|---|---|
| 2 | 1,075733 | 1,058267 | 1,027333 | 1,026133 | 1,022133 | 1,041920 | 0,059071 |
| 4 | 1,174400 | 1,160267 | 1,186933 | 1,136400 | 1,120800 | 1,155760 | 0,067275 |
| 6 | 1,151867 | 1,188933 | 1,144533 | 1,164667 | 1,183600 | 1,166720 | 0,019401 |
| 9 | 1,180000 | 1,160800 | 1,149467 | 1,164667 | 1,183067 | 1,167600 | 0,012072 |