ESCOLA POLITÉCNICA
DE PERNAMBUCO

d.s.c. UPE
Departamento de Sistemas e Computação

Análise do Escalonamento de Redes *Ad Hoc* IEEE 802.11 através de medidas de Vazão e Atraso usando o NS-2

Trabalho de Conclusão de Curso

Engenharia da Computação

Ricardo José Ulisses de Miranda Soares Filho
Orientador: Renato Mariz de Moraes

Recife, junho de 2008

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

d.s.c. UPE
Departamento de Sistemas e Computação

Análise do Escalonamento de Redes *Ad Hoc* IEEE 802.11 através de medidas de Vazão e Atraso usando o NS-2

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Ricardo José Ulisses de Miranda Soares Filho
Orientador: Renato Mariz de Moraes

Recife, junho de 2008

UNIVERSIDADE
DE PERNAMBUCO

Ricardo José Ulisses de Miranda Soares Filho

Análise do Escalonamento de Redes *Ad Hoc* IEEE 802.11 através de medidas de Vazão e Atraso usando o NS-2

Resumo

O estudo do escalonamento de redes *ad hoc* IEEE 802.11 é de fundamental importância para aplicações em larga escala dessas redes. Realizar o escalonamento dessas redes tem o objetivo de aumentar o número de nós que as compõem visando alcançar uma tendência em seu comportamento, possibilitando análises posteriores para uma determinada configuração de tráfego. As simulações realizadas neste trabalho utilizaram o programa de simulação NS-2, desenvolvido especialmente para a pesquisa de redes, o qual provê suporte nativo a simulação de redes *ad hoc*. A análise do escalonamento de redes aqui realizada se baseia em duas métricas comumente usadas para medir o desempenho de redes: vazão e atraso. Com os resultados obtidos a partir de programas desenvolvidos para extração dessas métricas, foi possível verificar que à medida que as redes escalonam, os valores médios das métricas demonstram uma degradação do desempenho dessas redes, apesar de ainda não indicar uma tendência a estabilização desses valores. Neste trabalho também são mostrados os problemas encontrados durante as simulações e o que foi feito para contorná-los.

Abstract

Studying *ad hoc* IEEE 802.11 network scaling has major importance for large scale use of these networks. The scaling of these networks intends to increase the number of nodes which constitutes them aiming to achieve a tendency in its behavior, making it possible to do further analysis or a certain traffic configuration. In this work we have employed the NS-2 simulation program, which is specially targetted at network research, and provides native support to simulating *ad hoc* networks. The network scaling analysis presented here is based on two metrics commonly used to measure network performance: throughput and delay. Using the results obtained from programs developed to extract these metrics, it was possible to check that as long as the network scales, the mean values of those metrics show a performance loss of the networks, although they do not seem to indicate a tendency to the stabilization of these values. This work also shows the problems found throughout the simulations and what was done to circumvent them.

Sumário

Índice de Figuras

Índice de Listagens

Tabela de Símbolos e Siglas

NS-2 – Network Simulator version 2 (Network Simulator versão 2)
Bps – Bytes per second (Bytes por segundo)
bps – bits per second (bits por segundo)
Mbps – Mega bits per second (Mega bits por segundo)
PAN – Personal Area Network (Rede de Área Pessoal)
MANET – Mobile Ad Hoc Network (Rede Ad Hoc Móvel)
IEEE – Institute of Electrical and Electronics Engineers (Instituto de Engenheiros Elétricos e Eletrônicos)
MAC – Media Access Control (Controle de Acesso ao Meio)
TCP/IP – Transmission Control Protocol/Internet Protocol (Protocolo de Controle de Transmissão/Protocolo da Internet)
NAM – Network Animator
TCP – Transmission Control Protocol (Protocolo de Controle de Transmissão)
FTP – File Transfer Protocol (Protocolo de Transferência de Arquivos)
UDP – User Datagram Protocol (Protocolo de Datagrama do Usuário)
CBR – Constant Bit Rate (Taxa de Bit Constante)
LAN – Local Area Network (Rede de Área Local)
MAN – Metropolitan Area Network (Rede de Área Metropolitana)
IBSS – Independent Basic Service Set (Conjunto de Serviço Básico Independente)
BSS/ESS – Basic Service Set/Extended Service Set (Conjunto de Serviço Básico/Conjunto de Serviço Extendido)
SCM – Serviço de Comunicação Multimídia
CSMA/CA – Carrier Sense Multiple Access/Collision Avoidance (Acesso Múltiplo com Detecção de Portadora/Prevenção de Colisão)
CMU – Carnegie Mellon University (Universidade Carnegie Mellon)
GOD – General Operations Director (Diretor de Operações Gerais)
AODV – Ad hoc On-Demand Distance Vector (Vetor de Distância Ad hoc Sob Demanda)
DSR – Dynamic Source Routing (Roteamento Dinâmico pela Origem)
SNS – Staged Network Simulator (Network Simulator com Estágios)
RTS/CTS – Request To Send/Clear To Send (Requisição Para Enviar/Livre Para Enviar)
GHz – Gigahertz
GB – Gigabyte
RAM – Random Access Memory (Memória de Acesso Aleatório)
TTL – Time To Live (Tempo de Vida)
KB – Kilobyte
CPU – Central Processing Unit (Unidade Central de Processamento)
S.O. – Sistema Operacional
CST – Carrier Sense Threshold (Limite de Detecção da Portadora)
RT – Receive Threshold (Limite de Recebimento)
DSDV – Destination-Sequenced Distance Vector (Vetor de Distância Sequenciado por Destino)

Agradecimentos

Agradeço primeiramente a Deus, aos meus pais que foram os principais incentivadores dos meus estudos, à minha esposa por sua enorme compreensão apesar de tantos dias e noites de minha ausência, a meus amigos, companheiros do dia-a-dia acadêmico, que foram fundamentais para o sucesso de tantos projetos e provas (e por terem proporcionado tantas risadas mesmo nos momentos mais difíceis), e a meu orientador que sempre me incentivou e me ajudou com respostas rápidas e observações enriquecedoras.

Capítulo 1

Introdução

Uma rede *ad hoc* é formada por dispositivos computacionais sem fio que podem se mover livremente no espaço e podem atuar tanto como agentes diretos de uma comunicação de rede quanto como roteadores. No papel de agentes, os nós podem ser classificados como geradores ou receptores de tráfego de dados. Já no papel de roteadores, os nós encaminham pacotes entre agentes que não podem se comunicar diretamente devido às limitações de alcance de sinal entre ambos [1].

As redes *ad hoc* são especialmente importantes em ambientes dinâmicos, onde há necessidade de comunicação entre dispositivos de rede sem a configuração prévia de uma infra-estrutura para dar apoio a essa comunicação. Um dos cenários em que a utilidade de uma rede desse tipo é facilmente percebida é um campo de batalha de guerra, onde os combatentes precisam estabelecer comunicação rápida e eficiente entre si sem depender de uma infra-estrutura física de rede. No entanto, existem possibilidades cada vez mais realistas de viabilização em larga escala desse tipo de rede em aplicações comerciais como na combinação entre redes de área pessoal, mais conhecidas pela sigla PAN - *Personal Area Network*, e redes metropolitanas, como as redes celulares [1].

Muitos estudos apontam resultados de diversas análises em redes *ad hoc*, especialmente no que diz respeito ao desempenho de algoritmos de roteamento para estas redes [2]. No entanto, em vários casos, os estudos utilizam-se de uma quantidade bastante limitada de nós, numa faixa de valores igual ou inferior a 100. Para a realização dessas análises, são usadas três abordagens de experimentação: prática, analítica e simulada [3].

Os experimentos práticos são constituídos de equipamentos que realizam testes em campo e obtêm dados relevantes sobre os mesmos. Esse tipo de abordagem geralmente restringe o número de nós participantes da comunicação por ser necessário ter uma grande quantidade de equipamentos disponível para os testes. Os experimentos analíticos são

feitos com o auxílio da matemática e das teorias vigentes sobre como ocorre a comunicação, utilizando-se fortemente a Teoria da Informação. A abordagem analítica tem grande importância no estabelecimento de valores ideais, máximos e mínimos de qualquer aspecto da comunicação de rede. A experimentação simulada é realizada com o auxílio de ferramentas de software que possibilitam gerar modelos e situações semelhantes às existentes nos testes de campo, permitindo uma grande flexibilidade em termos de dimensionamento de modelos e duração do experimento [3].

Um estudo analítico anterior [4] demonstrou que existe um limitante superior para a capacidade (medida em bits/s/Hz) de comunicação entre dois nós numa rede *ad hoc* móvel independente da quantidade de nós causando interferência nessa comunicação. Este estudo restringe-se ao modelo matemático da camada física, indicando que, se este resultado é válido para esta camada, teoricamente é válido para as camadas de rede superiores. Ainda mais, os resultados analíticos obtidos foram calculados para valores de até 10.000 nós.

O problema a ser analisado neste trabalho consiste em verificar se os valores calculados através do estudo analítico [4] podem ser obtidos na camada de rede do modelo TCP/IP, com um número de nós de magnitude semelhante, através do uso do software de simulação Network Simulator-2 (NS-2) [5]. Esta verificação é importante para se ter mais uma validação do NS-2 em termos de corretude da implementação relativa às limitações de capacidade da camada física e para que se possa ter uma idéia de como essas redes se comportarão na camada de rede com uma grande quantidade de usuários ativos quando a tecnologia for amplamente usada.

O simulador NS-2 foi escolhido como software de simulação a ser usado neste trabalho devido a sua larga aceitação pela comunidade acadêmica no estudo de redes de computadores e por ser uma ferramenta de código aberto, possibilitando estudos posteriores mais aprofundados e, até mesmo, implementação de novos trechos de código e alterações no código-fonte original, caso seja necessário ou conveniente.

De acordo com os objetivos aqui definidos, uma parcela de tempo considerável foi investida no estudo do funcionamento do simulador NS-2 antes dos experimentos de interesse serem realmente iniciados. Esse estudo abrangeu desde o funcionamento básico do NS-2 com redes Ethernet convencionais, incluindo sua linguagem de descrição de modelo de simulação OTcl (*Object Tcl*), passando pelo estudo de modelos sem fio 802.11, seus formatos de arquivos de trace usados para posterior extração das métricas escolhidas para análise, assim como estudos para aproximação do modelo sem fio 802.11 usado no simulador com o comportamento padrão dos equipamentos sem fio utilizados no nosso dia-a-dia.

1.1 Métricas Utilizadas

Com o objetivo de produzir resultados relevantes ao estudo do escalonamento de redes *ad hoc*, foram escolhidas duas métricas bastante utilizadas para analisar o desempenho de uma rede: vazão (*throughput*) e atraso fim-a-fim (*end-to-end delay*), respectivamente.

Genericamente, o termo vazão é definido como a razão (requisições por unidade de tempo) em que as requisições podem ser servidas por um sistema. Para análise de rede, a vazão é medida em pacotes por segundo (pps) ou bits por segundo (bps) [3], podendo também ser medida em bytes por segundo (Bps). O estudo da vazão da rede realizado neste trabalho compreende uma análise da razão entre a quantidade de dados que é transferida pela rede entre um nó de origem e um nó de destino e o tempo decorrido até que essa quantidade de dados seja totalmente transferida.

O estudo do atraso fim-a-fim em relação à comunicação de dois nós da rede é realizado através de uma média de tempo calculada através do somatório dos atrasos (tempo decorrido) entre o envio do pacote pelo nó de origem e o recebimento do pacote pelo nó de destino, dividido pela quantidade de pacotes especificada. O atraso é dito fim-a-fim pois consideramos a variação de tempo entre o envio e o recebimento dos nós agentes diretos da comunicação, o que compreende o tempo de transmissão do pacote do nó para a rede (atraso de transmissão), o tempo de propagação no meio sem fio (atraso de propagação) e, freqüentemente, o tempo de encaminhamento do pacote por nós da rede que estejam agindo como roteadores nessa comunicação (atraso de fila e atraso de processamento) [24].

1.2 Estudo de Capacidade das MANETs

Com a evolução da tecnologia e a produção em larga escala dos equipamentos, segue-se uma tendência à diminuição dos preços dos produtos e, cada vez mais, a necessidade de simplificar e agilizar o dia-a-dia são componentes significativos para a adoção desses novos produtos por uma grande parcela da população mundial. Nessa realidade, as redes *ad hoc* possuem um grande potencial de crescimento, dada a praticidade de interligação de equipamentos eletrônicos diversos em rede, sem que haja uma infra-estrutura previamente montada.

Um dos principais interesses no estudo das redes *ad hoc* móveis (da sigla em inglês MANET, significando *Mobile Ad Hoc Network*) é saber como estas redes escalonam. O termo escalonar, no sentido compreendido por este trabalho, significa progressivamente aumentar o número de nós da rede, visando alcançar uma tendência de comportamento desta. Dessa forma, deseja-se saber como um ambiente em que estejam presentes dezenas, centenas, milhares ou milhões de equipamentos afeta a capacidade de comunicação da

rede. O estudo do escalonamento de redes desse tipo, assim como o estudo de qualquer tecnologia em desenvolvimento, acontece inicialmente com o uso de modelos e, posteriormente, com experimentos reais realizados em campo. O uso de modelos é de importância fundamental para que se chegue a resultados plausíveis e de utilização prática, pois servem como uma aproximação do comportamento real da tecnologia que está sendo estudada.

Um estudo analítico sobre a capacidade de comunicação com o escalonamento de MANETs no nível da camada física [4] estabeleceu uma tendência a um limitante inferior para a capacidade de comunicação dessas redes. Este estudo também realiza uma validação do modelo analítico com um modelo simulado. Para isso, foi utilizado o método de Monte-Carlo, com o auxílio do software MATLAB. Os resultados obtidos tanto do modelo quanto da simulação estão exibidos no gráfico da Figura 1, onde "n" representa o número de nós da rede.

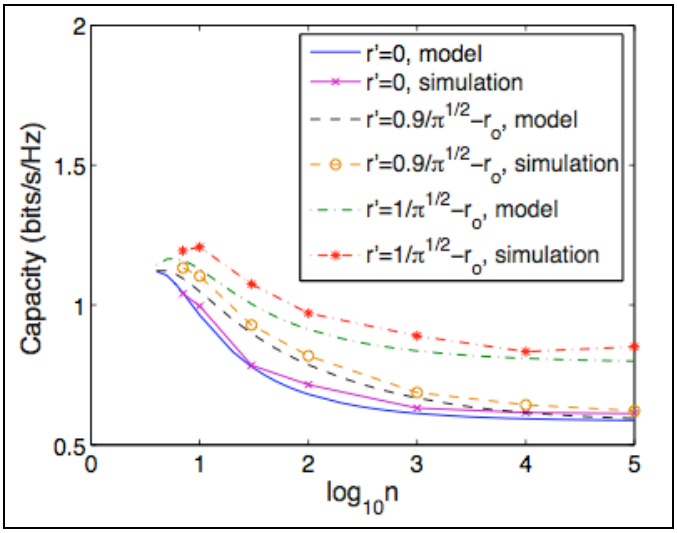

Figura 1. Tendência a um limitante inferior na capacidade (eficiência espectral) de uma MANET pelo modelo analítico e um modelo simulado [4].

O objeto de estudo deste trabalho é, de acordo com os valores estabelecidos em termos de número de nós e configuração geral da rede do estudo [4], realizar um estudo do escalonamento de redes *ad hoc* com a utilização do protocolo de comunicação sem fio IEEE 802.11 (que rege as camadas física e MAC), no nível da camada de rede, nível 3 do modelo TCP/IP, analisando as métricas anunciadas anteriormente: vazão e atraso, por meio de simulação, com auxílio do software NS-2.

Deve-se levar em conta para a posterior análise dos resultados deste trabalho que o NS-2, apesar de ser um software largamente usado na comunidade científica e por entidades afins (privadas, órgãos do governo ou sem fins lucrativos), ainda é um software

em desenvolvimento [5] e seus resultados podem não corresponder fielmente aos aspectos limitantes da camada física e demais camadas.

Capítulo 2

Simulação de Redes com o NS-2

Uma simulação é uma imitação de algum processo por um outro processo. Processos são uma seqüência de estados temporais de um sistema [8]. Uma simulação é um meio bastante eficiente para realização de testes e desenvolvimento de novas tecnologias. Atualmente, os computadores são grandes aliados no ato de simular processos de um sistema. Uma simulação de rede é uma técnica onde dispositivos de uma rede real, como *hosts*, roteadores, *switches*, entre outros, são modelados de maneira a apresentar comportamento semelhante aos dos dispositivos reais correspondentes. Este capítulo é dedicado ao estudo inicial da ferramenta de simulação de redes NS-2.

2.1 Software de Simulação NS-2

O *Network Simulator* é um simulador de eventos discretos, ou seja, no qual a operação do sistema é representada por uma seqüência cronológica de eventos, orientado a objetos e voltado para a pesquisa de redes. Embora há anos em uso por pesquisadores tanto do meio acadêmico quanto fora dele, o NS-2 ainda é um software em constante desenvolvimento. Atualmente ele provê suporte a simulações envolvendo TCP/IP, roteamento e protocolos *multicast* sobre redes com e sem fio (locais e via satélite) [5].

2.1.1 Estrutura Geral

O NS-2 é um simulador que foi desenvolvido utilizando-se duas linguagens de programação para dois propósitos distintos. O núcleo do NS-2 é escrito em C++, o que lhe confere um grande poder de especificação de protocolos no nível de bits/bytes, assim como garante uma rápida velocidade de execução característica da linguagem. Como uma maneira de tornar a descrição dos modelos

mais prática e rápida para o usuário, o NS-2 usa a linguagem de *script* OTcl como linguagem de comandos e de configuração de modelos para simulações. Com a escolha das linguagens citadas, os desenvolvedores do NS-2 souberam adequar de uma maneira prática para os usuários e desenvolvedores da ferramenta tanto o poder de representação e eficiência da linguagem C++ para o componente principal do simulador quanto a praticidade da linguagem de *script* conveniente à execução de várias simulações que necessitam de grandes ou pequenas alterações rapidamente.

De fato, a interface de comando e configuração do NS-2 interpreta comandos OTcl. Tais comandos podem ser fornecidos interativamente pelo usuário, de forma semelhante à linha de comando de um sistema Linux/UNIX (bash, sh, csh, entre outros), ou na forma de *scripts* OTcl, que são constituídos por linhas de código em seqüência, de tal forma a descrever completamente um modelo a ser simulado. Assim, o OTcl funciona como um *front-end* mais amigável e prático para o usuário interagir com o simulador.

Para que seja possível a utilização dessas duas linguagens, OTcl e C++, uma como *front-end* de usuário e outra como *core* da implementação dos componentes de rede, existe, no NS-2, uma correspondência entre objetos na interface OTcl e objetos C++, conforme mostrado na Figura 2. Com o intuito de reduzir o tempo de processamento de pacotes e eventos, o agendador de eventos e os componentes básicos de rede são escritos e compilados em C++. Esses objetos compilados são disponibilizados ao interpretador OTcl através de um "acoplamento" entre a biblioteca OTcl e os componentes compilados C++, criando um objeto correspondente no interpretador OTcl para cada objeto construído em C++. Da mesma maneira isso é feito para métodos e variáveis desses objetos, podendo os mesmos ser referenciados via OTcl ainda que sua especificação real esteja feita em C++.

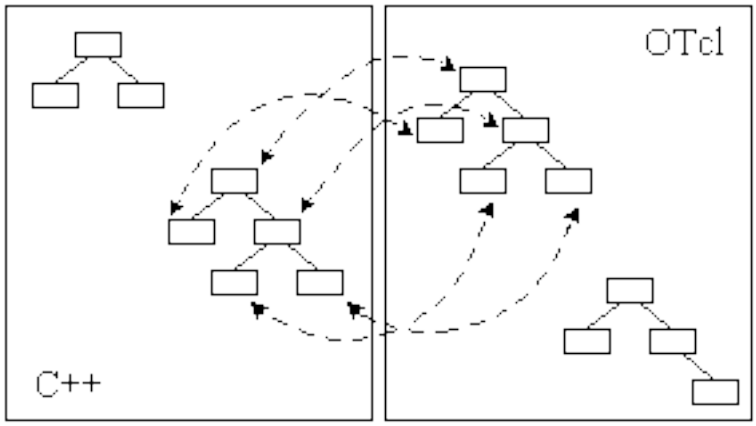

Figura 2. Correspondência entre objetos C++ e Otcl [15].

Uma visão simplificada do passo-a-passo a partir da codificação de um *script* OTcl, sua execução pelo simulador e conseqüente produção de resultados que podem ser analisados ou visualizados, é exibida na Figura 3.

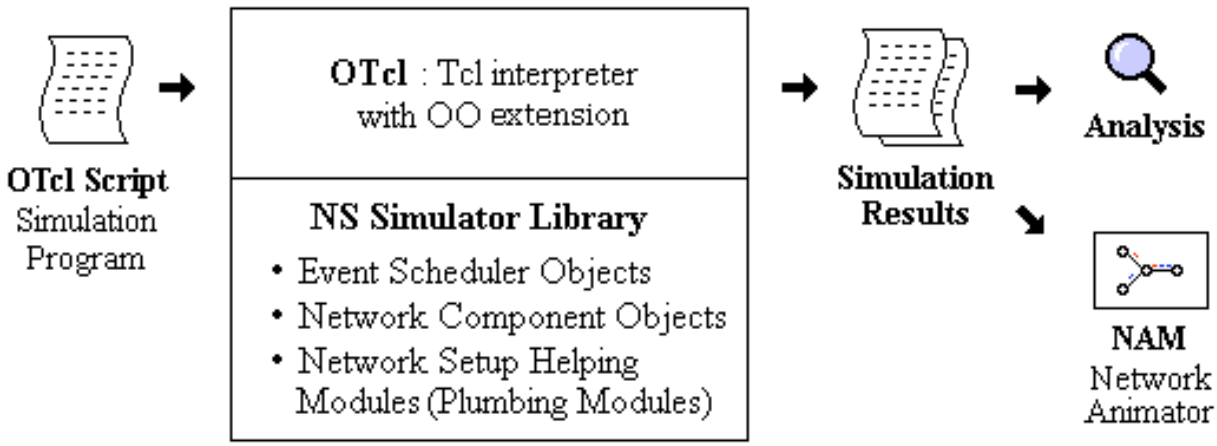

Figura 3. Visão simplificada da execução de uma simulação sob a perspectiva do usuário no NS-2 [15].

Como mostrado na Figura 3, o NS-2 interpreta um *script* OTcl que possui instâncias de objetos do agendador de eventos e instâncias de objetos que são componentes da rede e descrevem o modelo. Ou seja, para configurar e executar uma simulação, o usuário precisa escrever um *script* OTcl que contenha os objetos que representam os componentes da rede (nós, topologia da rede, interconexão dos nós e o tipo de tráfego entre esses nós interconectados), um agendador de eventos e configurar o comportamento da rede por meio do agendador de eventos, em outras palavras, definir quando o agendador de eventos deve disparar uma determinada ação. Uma ação da rede pode ser, por exemplo, a transmissão de dados entre dois nós, ou mesmo a movimentação de um nó na rede num dado valor de tempo da simulação.

Ao longo da execução da simulação, arquivos que armazenam informações relevantes da rede podem estar sendo continuamente escritos à medida que resultados são processados pelo simulador. Para que isto aconteça, é preciso habilitar o *trace* no objeto que executa a simulação. Habilitar *trace* no *script* de simulação é uma tarefa bastante simples e, ao mesmo tempo, possibilitará uma análise do que ocorreu durante toda a simulação assim que esta termine. É com base na análise do arquivo de *trace* que extraímos as métricas de interesse do trabalho.

Além do *trace* voltado à análise do comportamento da rede, também é possível produzir outro tipo interessante de *trace* voltado à análise visual do comportamento da rede. Este *trace* é utilizado pelo software NAM (*Network AniMator*) e possibilita uma visualização gráfica animada do comportamento da rede como: movimentação de nós, transmissão de pacotes de várias fontes de tráfego, estouros de fila em roteadores ocasionando perdas de pacotes e alcance de sinal em redes sem fio. Mais adiante neste

documento será exibida uma visão geral dos formatos de *trace* disponíveis e será dada uma atenção maior ao formato de *trace* escolhido para análise.

2.1.2 Arquitetura

A descrição da estrutura do sistema feita na seção anterior expõe em linhas gerais os componentes básicos do NS-2, o motivo de decisões de projeto em relação às linguagens de programação utilizadas, a correspondência entre objetos das linguagens e um modelo simplificado da execução de uma simulação. Nesta seção é dada uma visão um pouco mais detalhada da arquitetura do NS-2, ou seja, como está estabelecida a hierarquia entre os componentes do sistema e sua interligação. A arquitetura do NS-2 é formada pelos seguintes componentes:

- Agendador de Eventos;
- Componentes de Rede;
- TclCL;
- Biblioteca OTcl;
- Linguagem de *script* Tcl 8.0.

O agendador de eventos e a maioria dos componentes de rede são desenvolvidos em C++, devido ao melhor desempenho do código compilado, conforme mencionado anteriormente. A interligação entre esses componentes com o OTcl é feita através do TclCL (*Tcl with classes*) [6], que é uma interface entre o Tcl e o C++, para o OTcl. A biblioteca OTcl do NS-2 é o resultado da interface entre o OTcl e o C++, via TclCL, contendo algumas classes, métodos e variáveis do sistema.

A linguagem de *script* Tcl 8.0 é a base do OTcl. O próprio OTcl (sigla para *MIT Object Tcl*) é o resultado de um projeto do MIT para extender a linguagem Tcl, que é uma linguagem do paradigma imperativo, para o paradigma orientado a objetos. Devido ao OTcl ter sido construído preservando-se a sintaxe e os conceitos da linguagem de programação Tcl, programa-se em OTcl da mesma forma que em Tcl, apenas com o adicional de ser possível instanciar classes com o OTcl. A Figura 4 mostra um esquema da arquitetura aqui descrita.

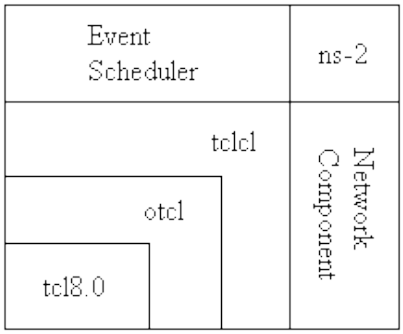

Figura 4. Esquema da arquitetura dos componentes do NS-2 [15].

O esquema de arquitetura do NS-2 mostrado na Figura 4 pode ser entendido da seguinte maneira: um usuário do programa está localizado no canto inferior esquerdo da figura, onde projeta e executa simulações em Tcl, usando os objetos do simulador em OTcl. O agendador de eventos e os componentes de rede, como já foi dito, são implementados em C++. A interface de ambos com a linguagem OTcl é feita através de interligação usando o TclCL. A união desses cinco componentes forma o NS-2, localizado no canto superior direito da figura [15].

2.1.3 Instalação e Configuração

O NS-2 é um software de simulação desenvolvido em vários tipos de sistemas operacionais UNIX ou UNIX-*like*, como FreeBSD, Linux, SunOS e Solaris. Devido a isso, sua instalação e execução é mais conveniente nesses sistemas. No entanto, também é possível instalar e executar o NS-2 no Windows. Como todo este trabalho utilizou duas versões do NS-2 instaladas em sistema operacional Linux, mais especificamente na distribuição Ubuntu Server 7.10 [7], será dada exclusividade ao processo de instalação nesta plataforma e distribuição.

A instalação do NS-2 pode ser feita de duas maneiras: compilando-se cada pacote de software necessário para o sistema como um todo individualmente ou utilizando-se a distribuição *allinone* (ns-allinone) que consiste em um grande pacote contendo todos os pacotes individuais essenciais para compilação do NS-2 e alguns não essenciais, como programas para plotagem de gráficos (xgraph) e o NAM. Abaixo está a lista de pacotes contidos na distribuição allinone mais recente, a ns-allinone-2.33:

- Tcl release 8.4.18 (componente necessário);
- Tk release 8.4.18 (componente necessário);
- Otcl release 1.13 (componente necessário);
- TclCL release 1.19 (componente necessário);
- Ns release 2.33 (componente necessário);
- Nam release 1.13 (componente opcional);

- Xgraph version 12 (componente opcional);
- CWeb version 3.4g (componente opcional);
- SGB version 1.0 (componente opcional);
- Gt-itm gt-itm and sgb2ns 1.1 (componente opcional);
- Zlib version 1.2.3 (opcional, mas necessário se o Nam for utilizado).

O pacote ns-allinone está disponível para *download* na forma de um arquivo compactado ns-allinone-2.33.tar.gz [5]. Para descompactar o arquivo, é preciso ter os programas tar e gzip instalados no sistema. Na distribuição usada neste trabalho, esses programas já estão instalados com o sistema base. Após assegurar-se que os programas estão instalados, basta executar o comando listado na Listagem 1 na linha de comandos do Linux.

Listagem 1. Comando para descompactar o pacote ns-allinone-2.33.tar.gz.

```
$ tar zxvf ns-allinone-2.33.tar.gz
```

Depois de decompactado o pacote ns-allinone-2.33.tar.gz, haverá um diretório de nome ns-allinone-2.33 no diretório atual. Neste diretório estão todos os componentes de software supracitados, necessários e opcionais para a instalação do NS-2.

O próximo passo consiste em instalar alguns pacotes de software necessários para a compilação e instalação do NS-2 e seus componentes. Para isso, é preciso estar logado no sistema como usuário privilegiado. Em sistemas Windows, esta conta do sistema tem o nome Administrator ou Administrador (na versão brasileira). No Linux, o padrão é que esta conta de usuário seja chamada de root. O motivo da instalação de cada software listado abaixo individualmente está fora do escopo deste trabalho, mas, em linhas gerais, eles são necessários como base para compilação dos vários pacotes de software contidos na distribuição ns-allinone, funcionando como bibliotecas, compiladores e programas auxiliares na configuração dos pacotes num estágio pré-compilação. A Listagem 2 mostrada a seguir contém os comandos para alternar de um usuário normal para o usuário root, assim como o comando para instalação dos softwares necessários.

Listagem 2. Comandos para alternar o usuário para root e instalar softwares necessários para a instalação do pacote ns-allinone.

```
$ sudo -s
# apt-get install gcc make libc6-dev libmudflap0-dev xorg-dev g++
```

Os softwares listados acima podem conter dependências, ou seja, podem precisar de outros softwares para sua instalação. Para que as dependências sejam satisfeitas, basta

confirmar as perguntas que o programa apt-get apresentará, fazendo com que os softwares sejam instalados automaticamente.

Após executar os comandos acima, basta mudar o diretório atual para o diretório ns-allinone-2.33 criado anteriormente e executar o *script* "install", que já vem com o pacote *allinone*. Os comandos para realizar essas ações estão listados na Listagem 3.

Listagem 3. Comandos para iniciar a instalação do pacote ns-allinone-2.33.

```
$ cd ns-allinone-2.33
ns-allinone-2.33 $ ./install
```

Depois do sistema estar instalado, é recomendável criar-se um *script* para carregamento das variáveis de ambiente necessárias à execução do NS-2, como mostrado na Listagem 4.

Listagem 4. Definição de variáveis de ambiente usadas pelo NS-2.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/rico/ns-allinone-2.33/otcl-1.13:/home/rico/ns-allinone-2.33/lib
export TCL_LIBRARY=/home/rico/ns-allinone-2.33/tcl8.4.18/library
export PATH=$PATH:/home/rico/ns-allinone-2.33/bin:/home/rico/ns-allinone-2.33/tcl8.4.18/unix:/home/rico/ns-allinone-2.33/tk8.4.18/unix
```

2.1.4 Exemplo de Simulação de Rede Cabeada Simples

Nesta seção é apresentado um exemplo de simulação de uma rede cabeada simples, como uma forma de ilustrar os componentes de configuração de uma simulação com o NS-2 apresentados de maneira exclusivamente teórica até então. A rede a ser simulada é constituída pelos seguintes componentes de rede: 4 nós, 3 enlaces, uma fluxo de tráfego TCP, simulando a aplicação de transferência de arquivos largamente usada na Internet, FTP, e um fluxo de dados UDP, simulando uma aplicação de streaming de vídeo CBR (*Constant Bit Rate*) [24].

Como pode ser visto na Figura 5, os dois *hosts* da esquerda estão conectados ao roteador por meio de enlaces de 10Mbps de capacidade (largura de banda nominal), enquanto o *host* da direita está conectado ao roteador por um enlace de 100Mbps. Os equipamentos denominados *hosts* atuam como agentes de transmissão de dados, enquanto o equipamento denominado roteador atua como um comutador, isto é, apenas encaminha pacotes entre redes distintas.

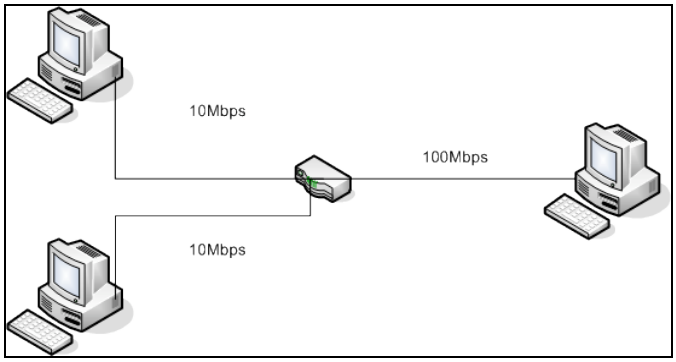

Figura 5. Visão geral da topologia física da rede cabeada de exemplo.

O modelo simulado consiste em duas conexões FTP, cada uma partindo de um dos *hosts* à esquerda e tendo como destino o servidor FTP localizado no *host* à direita, sendo os pacotes encaminhados entre os enlaces pelo roteador. A Figura 6 mostra uma visão mais detalhada da simulação realizada com a topologia exibida na Figura 5.

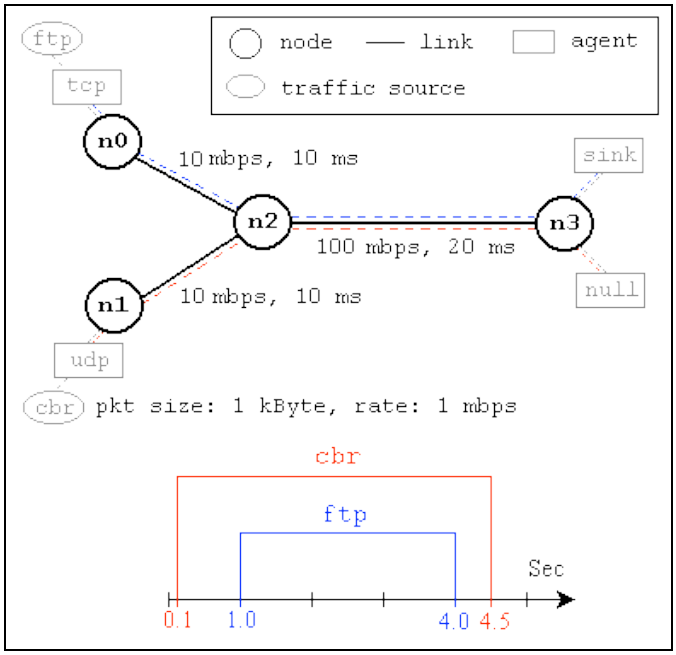

Figura 6. Visão detalhada da rede de exemplo de acordo com os componentes do NS-2 [15].

Este modelo de rede representa o modelo apresentado na Figura 5 com maior riqueza de detalhes. Nele são exibidos os nós, as capacidades (em Mbps) e o atraso de propagação de cada um dos 3 enlaces, o tamanho do pacote (pkt size) e a taxa de envio dos pacotes (rate). Também são exibidas as classes de tráfego (TCP, UDP, Sink e Null) e aplicação (FTP e CBR). Na parte de baixo está localizada uma linha do tempo de simulação, destacando, em vermelho, o intervalo de atuação do tráfego CBR (começando em 0,1 segundo até 4,5 segundos) e o intervalo de atuação do tráfego FTP.

Na Listagem 5 está exposto todo o *script* escrito em OTcl que descreve a simulação para o modelo apresentado. Apesar de se tratar de uma Listagem contendo uma grande quantidade de código, a sua exibição e posterior detalhamento do código são de fundamental importância para o entendimento dos componentes básicos de qualquer simulação usando o NS-2.

Listagem 5. *Script* OTcl simple.tcl de descrição do modelo cabeado de exemplo.

```
#Create a simulator object
set ns_ [new Simulator]

#Open the trace files
set nt [open out.tr w]
$ns_ trace-all $nt
set nf [open out.nam w]
$ns_ namtrace-all $nf

#Define packet color for NAM
$ns_ color 1 Blue
$ns_ color 2 Red

#Define a 'finish' procedure
proc finish {} {
        global ns_ nt nf
        $ns_ flush-trace
        close $nt
        close $nf
        exit 0
}

#Create four nodes
set n0 [$ns_ node]
set n1 [$ns_ node]
set n2 [$ns_ node]
set n3 [$ns_ node]

#Create links between the nodes
$ns_ duplex-link $n0 $n2 10Mb 10ms DropTail
$ns_ duplex-link $n1 $n2 10Mb 10ms DropTail
```

```
$ns_ duplex-link $n2 $n3 100Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns_ queue-limit $n2 $n3 10

#Setup the TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns_ attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns_ attach-agent $n3 $sink
$ns_ connect $tcp $sink
$tcp set fid_ 1

#Setup the FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#Setup UDP communication
set udp [new Agent/UDP]
$ns_ attach-agent $n1 $udp
set null [new Agent/Null]
$ns_ attach-agent $n3 $null
$ns_ connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP communication
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false

#Schedule events for the CBR and FTP agents
$ns_ at 0.1 "$cbr start"
$ns_ at 1.0 "$ftp start"
```

```
$ns_ at 4.0 "$ftp stop"

#Call the finish procedure after 5 seconds of simulation time
$ns_ at 5.0 "finish"

#Print CBR packet size and interval at the beginning of the simulation
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns_ run
```

A seguir são apresentados mais detalhes de cada um dos parâmetros de maior relevância ao modelo apresentado. Embora haja variações de configuração de parâmetros, assim como configurações de parâmetros adicionais dependendo do modelo simulado, é importante enfatizar que a grande maioria dos componentes do *script* de exemplo de modelo de rede cabeada [15] servirão como base para qualquer modelo de simulação no NS-2, conforme já esclarecido anteriormente.

- `set ns_ [new Simulator]`

  Este comando está presente em toda simulação usando o NS-2. É responsável por instanciar um objeto do tipo Simulator e o atribui-lo à variável `ns_`. O objeto `Simulator` é responsável pelo agendador de eventos da simulação, assim como por parâmetros da simulação como o formato dos pacotes e o formato de endereçamento padrão da rede. Ele possui funções de fundamental importância a qualquer simulação, como criação de objetos compostos como nós e enlaces, conexões entre agentes de tráfego da rede e especificação de opções de exibição de animações pelo NAM.

- `$ns_ trace-all $nt`

  A função `trace-all` do objeto `Simulator` realiza o *logging* de todo o comportamento da rede durante a simulação em um formato geral no descritor de arquivos especificado pela variável nt.

- `$ns_ namtrace-all $nf`

  A função `namtrace-all` realiza o *logging* de todo o comportamento da rede no formato apropriado para leitura pelo NAM no descritor de arquivos nf.

- `set n0 [$ns_ node]`

A função `node` cria um objeto composto que é formado por um endereço e uma porta determinados aleatoriamente pelo objeto `Simulator`, não sendo possível especificá-los no comando de criação de nó.

- `$ns_ duplex-link $n0 $n2 10Mb 10ms DropTail`

Assim como a função `node`, a função `duplex-link` também cria um objeto composto, com as propriedades especificadas pelos seus parâmetros. O comando realiza a criação de dois enlaces simplex (formando um enlace duplex) entre os nós `$n0` e `$n2` com largura de banda 10Mbps e atraso de 10ms. Em simulações cabeadas, a fila de saída de um nó é implementada como parte de um enlace, portanto é necessário especificar o tipo de fila a ser usada. No caso acima, optou-se pela fila do tipo `DropTail`, que corresponde a uma política FIFO (*First-In First-Out*) [25].

- `$ns_ queue-limit $n2 $n3 10`

A função `queue-limit`, ao receber como terceiro parâmetro um valor inteiro, configura um limitante superior para a quantidade de pacotes que podem estar presentes na fila de um determinado enlace, especificado pelo primeiro e segundo parâmetros. Também existe a possibilidade de definir a quantidade de bytes na fila, especificando-se o terceiro parâmetro com a letra "B" justaposta imediatamente ao valor inteiro.

- `set tcp [new Agent/TCP]`

Instancia um novo objeto da classe Agent/TCP e o associa à variável `tcp`. O instanciamento de objetos de outros agentes de tráfego, como o Agent/UDP, é idêntico, mudando-se apenas o nome da classe, portanto a linha correspondente ao mesmo comando em relação à classe Agent/UDP não será exibida nesta descrição.

- `set ftp [new Application/FTP]`

Instancia-se um novo objeto da classe Application/FTP e o associa à variável `ftp`. Isto é necessário para a simulação de um determinado protocolo da camada de aplicação que utilizará um agente de tráfego (protocolo da camada de transporte) previamente configurado. No caso, como o FTP é uma aplicação que se baseia no protocolo da camada de transporte TCP, esta aplicação fará uso do agente de tráfego TCP instanciado anteriormente. A escolha do tipo de aplicação

é especialmente importante para modelar determinados tipos de tráfego de maior interesse em certas configurações de rede.

- `$ns_ attach-agent $n0 $tcp`

A função `attach-agent` associa um objeto de uma das classes Agent (`$tcp`) a um objeto nó (`$n0`). Isso significa que, durante a execução da simulação, a ação de tráfego (envio ou recebimento) associada ao agente de tráfego referenciado pela variável `$tcp` será realizada pelo nó `$n0`.

- `$ns_ connect $tcp $sink`

Especifica a ligação entre os agentes de tráfego previamente definidos, ou seja, define uma origem e um destino para o tráfego de dados. Internamente, o NS-2 associa o par endereçoIP:porta de origem ao par endereçoIP:porta de destino.

- `$ns_ at 0.1 "$cbr start"`

A função membro do agendador de eventos `at` configura uma ação a ser realizada pelo simulador e o tempo em que esta ação deve ser executada. No exemplo acima, em que foi usada a primeira linha referente a uma configuração desse tipo no *script* simple.tcl, o agendador de eventos é configurado para iniciar a aplicação $cbr.

- `$ns_ run`

A chamada à função `run` é, assim como a instanciação do objeto agendador de eventos, imprescindível para a realização de qualquer simulação. Deve ser colocada após toda configuração do modelo de rede a ser simulado.

Após a configuração do modelo e parâmetros da simulação conforme exibido na Listagem 5, basta executar o NS-2 passando como primeiro argumento da linha de comando o nome do *script* OTcl, conforme mostra a Listagem 6.

Listagem 6. Execução da simulação de exemplo simple.tcl.

```
$ ns simple.tcl
CBR packet size = 1000
CBR interval = 0.0080000000000000002
```

Como se trata de um modelo e simulação simples, o tempo de execução é bastante reduzido, da ordem de 0,2 segundo. Conforme especificado no arquivo simple.tcl, antes de executar a simulação, o NS-2 exibe dois parâmetros configurados para a simulação: o tamaho do pacote no fluxo CBR e o intervalo entre o envio de dois pacotes neste mesmo

fluxo. Embora não especificado explicitamente no arquivo de simulação, este intervalo corresponde ao inverso do parâmetro "rate_" em relação ao tamanho do pacote.

Como a taxa (*rate*) de transmissão de dados foi especificada para 1Mbps (1mb), com um pacote de 1000 Bytes (8000 bits) tem-se o intervalo mostrado na Listagem 7.

Listagem 7. Cálculo do atraso total de envio de um pacote do fluxo CBR.

```
8000 bits / 1000000 bits/s ~ 0.008 s
```

2.1.5 Arquivos de *Trace* Gerados pelo NS-2

Conforme estabelecido na descrição do *script* simple.tcl, dois arquivos de *trace* são gerados: o arquivo de *trace* genérico da simulação (trace) e o arquivo de *trace* para análise gráfica (namtrace) com o auxílio da ferramenta de software Network Animator (NAM). Toda simulação executada com o NS-2 necessita da especificação de pelo menos o arquivo de *trace* genérico, caso contrário a simulação não será executada e um aviso de erro será exibido, conforme mostra a Listagem 8, e o programa abortará a execução da simulação.

Listagem 8. Execução de simulação sem a definição de um arquivo de *trace* genérico.

```
$ ns example.tcl
num_nodes is set 50
Warning: You have not defined you tracefile yet!
Please use trace-all command to define it.
```

O namtrace serve apenas como entrada para exibição do comportamento da simulação pelo NAM. Para executar o NAM e visualizar a animação da simulação, basta executar o comando mostrado na Listagem 8.

Listagem 9. Comando para visualização gráfica da simulação com o NAM.

```
$ nam arquivo.nam
```

A Figura 7 mostra um *screenshot* da execução do NAM para o namtrace gerado pela simulação simples de exemplo. É possível visualizar e distinguir nitidamente os dois fluxos de pacotes TCP e UDP, gerados pelas aplicações FTP e CBR, nas cores azul e vermelho, respectivamente.

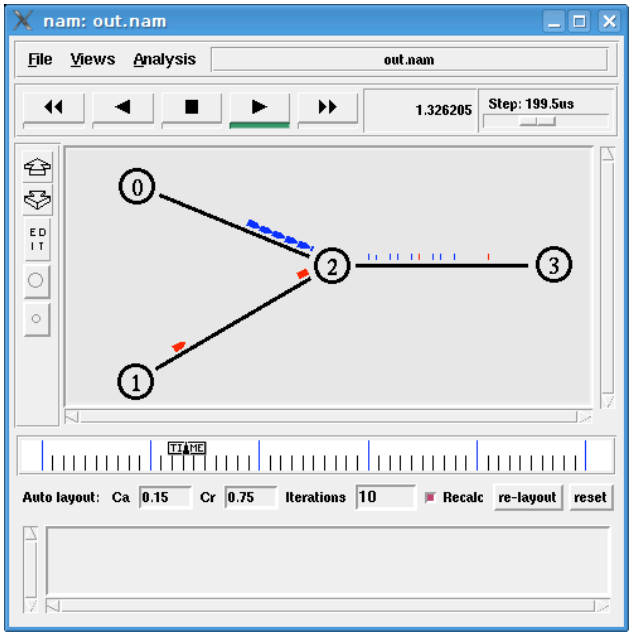

Figura 7. Screenshot da execução do NAM para a simulação simple.tcl.

O *trace* genérico descreve o comportamento da rede no que diz respeito a alguns eventos básicos interessantes para análise. Na simulação cabeada de exemplo, existem cinco eventos de interesse:

- Pacotes Recebidos (r);
- Pacotes Enfileirados (+);
- Pacotes Desenfileirados (-);
- Pacotes Descartados (d);
- Pacotes com Erro (e).

Como não é o foco deste trabalho, o *trace* de uma rede cabeada não será examinado com muitos detalhes. Apenas para ilustrar o que foi dito sobre este arquivo de *trace* até então, a Listagem 10 exibe um trecho do mesmo.

Listagem 10. Trecho do arquivo de *trace* da simulação cabeada de exemplo.

```
+ 0.1 1 2 cbr 1000 ------- 2 1.0 3.1 0 0
- 0.1 1 2 cbr 1000 ------- 2 1.0 3.1 0 0
+ 0.108 1 2 cbr 1000 ------- 2 1.0 3.1 1 1
- 0.108 1 2 cbr 1000 ------- 2 1.0 3.1 1 1
r 0.1108 1 2 cbr 1000 ------- 2 1.0 3.1 0 0
+ 0.1108 2 3 cbr 1000 ------- 2 1.0 3.1 0 0
- 0.1108 2 3 cbr 1000 ------- 2 1.0 3.1 0 0
```

A primeira linha do trecho do arquivo de *trace* mostra que está ocorrendo a inserção de um pacote na fila de transmissão (+) no tempo 0.1 segundo, com origem no *host* de número 1, com destino ao *host* de número 2, o pacote contém tráfego do tipo CBR e tem 1000 bytes de tamanho. Nenhuma *flag* no cabeçalho IP ou UDP foi configurada (-------). A seguir, estão os valores para o fluxo do pacote, identificado pelo inteiro 2, o endereço IP e a porta de origem (1.0) e o endereço IP e porta de destino (3.1). Os dois últimos valores correspondem ao número de seqüência (0), usado por agentes interessados em prover sequenciamento, e o identificador único do pacote (0), o qual identifica unicamente cada novo pacote gerado durante toda a simulação. Todas as demais linhas se referem a outros eventos, mudando apenas o primeiro caractere da linha, o qual define qual evento está ocorrendo.

Capítulo 3

Simulação de Redes Sem Fio *Ad Hoc* IEEE 802.11

O objeto de estudo deste trabalho tem foco principal na simulação de redes sem fio *ad hoc* IEEE 802.11. Este capítulo demonstra alguns princípios básicos da especificação IEEE 802.11. Em seguida, é exibida a estrutura de configuração de uma simulação sem fio *ad hoc* 802.11 no NS-2, assim como trechos de exemplo de um *script* de configuração da rede sem fio em OTcl. Também será abordada a geração de modelos de cenário sem fio (disposição dos nós) e modelos de tráfego de dados, assim como a análise do arquivo de *trace* produzido pela simulação da rede.

3.1 Princípios do Padrão IEEE 802.11

O padrão 802.11, genericamente denominado wi-fi, corresponde ao grupo de trabalho de número 11 do *IEEE LAN/MAN Standards Committee* (IEEE 802) [9]. Este grupo de trabalho possui sub-divisões em vários outros grupos de trabalho que são responsáveis por desenvolver áreas específicas do padrão. Estas sub-divisões são especificadas por uma única letra do alfabeto.

Este padrão é responsável pela especificação das camadas 1 e 2 do modelo OSI (física e enlcace ou MAC, respectivamente) da comunicação sem fio. Nesta especificação, o padrão define duas topologias básicas de operação: os clientes se comunicam diretamente uns com os outros ou eles se conectam a um ponto de acesso central. Estas duas topologias lógicas são definidas pelas siglas IBSS e BSS/ESS [10].

A topologia IBSS (*Independent Basic Service Set*) é comumente conhecida como modo *ad hoc*, a qual é objeto de estudo deste trabalho. Diferentemente de uma rede BSS/ESS (*Basic Service Set/Extended Service Set*), a qual é conhecida como modo infra-estruturado, o modo *ad hoc* não possui uma estrutura previamente montada para

comunicação entre os nós da rede, ou seja, não há um ponto de acesso central que intermedia a comunicação entre dois nós que queiram se comunicar. Como dito anteriormente, os nós se comunicam diretamente uns com os outros. Caso um nó esteja fora do alcance de transmissão direta de outro nó, a comunicação entre eles somente é possível com a transmissão *multi-hop* (múltiplos saltos), na qual nós intermediários (saltos) têm que repassar pacotes da origem ao destino, agindo como roteadores, usando algum protocolo de roteamento apropriado. A Figura 8 põe lado a lado as duas topologias citadas.

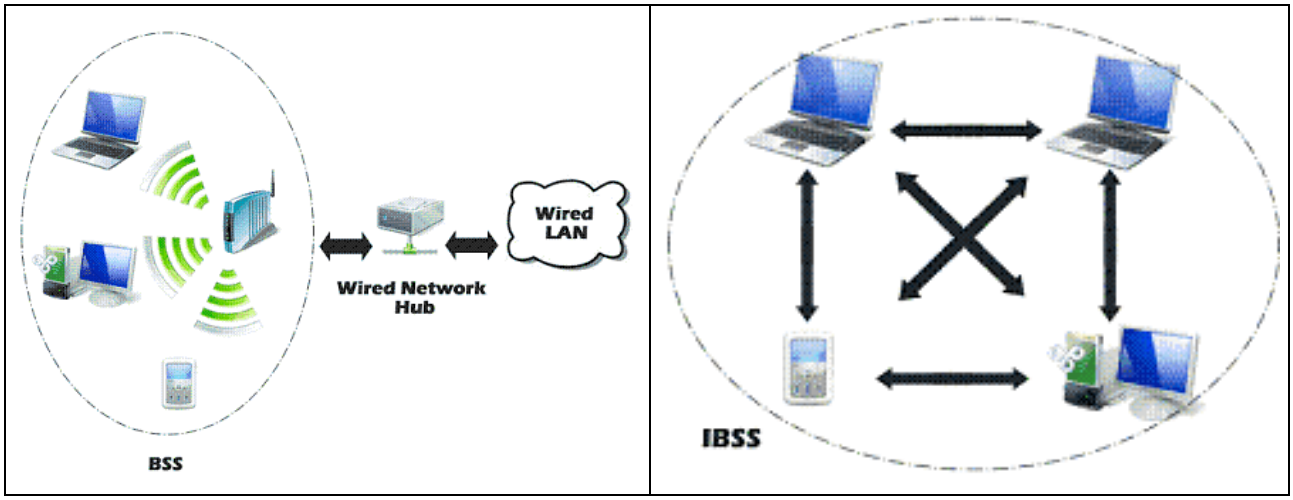

Figura 8. Exemplo de topologia infra-estruturada (BSS) à esquerda e *ad hoc* (IBSS) à direita.

Na especificação da camada física, o padrão 802.11 define duas faixas de freqüência a ser utilizadas: 2,4 GHz e 5GHz. Ambas podem ser usadas livremente, apenas com algumas restrições como não interferir em entidades detentoras de licença de utilização dessas faixas para fins comerciais SCM (Serviços de Comunicação Multimídia) [26] e não utilizar potência de transmissão acima de 100mW [11].

Na camada de enlace (MAC) são definidos esquemas de acesso ao meio físico, como o CSMA/CA (mecanismo de detecção da portadora que evita colisões), mecanismos para recuperação de erros e diferentes constantes de tempo para transmissão de quadros no meio [11].

3.2 Simulação de Redes *Ad Hoc* IEEE 802.11 no NS-2

O modelo de simulações sem fio utilizado pelo NS-2 foi desenvolvido por pesquisadores do Projeto Monarch (*Mobile Networking Architectures*) originalmente pertencente à CMU (*Carnegie Mellon University*) e agora pertencente à *Rice University* [12]. Inicialmente, esse modelo foi adotado apenas como uma extensão do NS-2. Atualmente o modelo é parte integrante

do NS-2 e tem sido continuamente aprimorado ao longo dos anos. Os componentes principais deste modelo são o *MobileNode*, mecanismos de roteamento e componentes de rede que são usados para configurar o *MobileNode*.

Os algoritmos de roteamento *ad hoc* implementados no NS-2 são [5]:

- AODV (*Ad hoc On-demand Distance Vector*);
- DSR (*Dynamic Source Routing*);
- DSDV (*Destination Sequence Distance Vector*);
- TORA (*Temporally Ordered Routing Algorithm*).

O protocolo de roteamento AODV é de interesse especial, pois foi o escolhido para ser usado nas simulações feitas neste trabalho. Este é um protocolo de roteamento *ad hoc* que funciona como uma combinação dos protocolos DSR e DSDV. Ele possui a estrutura básica de descoberta e manutenção de rotas do protocolo DSR e usa roteamento *hop-by-hop*, números de seqüência e *beacons* (pacotes especiais contendo informações genéricas do nó sem fio) do protocolo DSDV. Quando um nó quer saber uma rota para um dado nó de destino, ele gera um *route request*. Este *route request* é encaminhado por nós intermediários que também criam uma rota reversa para estes a partir do nó de destino. Quando o *request* alcança o nó de destino, este nó gera um *route reply* contendo o número de saltos (*hops*) necessários para alcançá-lo. Todos os nós que encaminharam o *route reply* para o nó de origem criam uma rota de encaminhamento para o destino.

Os componentes principais de configuração de um *MobileNode* são [5]:

- Canal de Comunicação (Channel);
- Interface de Rede (NetIf);
- Modelo de Propagação de Ondas de Rádio (Radio propagation model);
- Protocolo MAC (MAC protocol);
- Fila da Interface de Rede (Ifq);
- Camada de Enlace (LL);
- Modelo de Antena (Antenna).

Com essas configurações, o NS-2 permite a simulação de redes sem fio “puras”, ou seja, onde a comunicação acontece ponto-a-ponto no mesmo domínio de colisão, e a simulação de redes *ad hoc multi-hop*. A partir da versão 2.33 (a versão mais recente do NS-2 no momento da escrita deste trabalho e a versão que está sendo utilizada para a execução das simulações do mesmo) foram adicionados novos modelos da camada MAC 802.11, possibilitando a simulação de redes sem fio com infra-estrutura [13], o que não

era possível até a versão 2.32. No entanto, o estudo dessas redes está fora do escopo deste trabalho.

3.2.1 Configuração e Criação de Nós *MobileNode*

Um nó da classe *MobileNode* possui as mesmas características de um nó da classe Node, o qual foi utilizado na primeira simulação de exemplo deste trabalho, com algumas características adicionais como a habilidade de transmitir e receber pacotes (encapsulados em quadros da camada de enlace) de um canal de comunicação sem fio e capacidade de realizar movimento. A Listagem 11 mostra um trecho de um *script* OTcl de exemplo de configuração de um *MobileNode* de acordo com os componentes apresentados até então.

Listagem 11. Configuração dos componentes de um *MobileNode*.

```
set val(chan)      Channel/WirelessChannel  ;# Channel Type
set val(prop)      Propagation/FreeSpace    ;# radio-propagation model
set val(netif)     Phy/WirelessPhy          ;# network interface type
set val(mac)       Mac/802_11               ;# MAC type
set val(ifq)       Queue/DropTail/PriQueue  ;# interface queue type
set val(ll)        LL                       ;# link layer type
set val(ant)       Antenna/OmniAntenna      ;# antenna model
set val(ifqlen)    50                       ;# max packet in ifq
set val(nn)        50                       ;# number of mobilenodes
set val(rp)        DumbAgent                ;# routing protocol
set val(x)         100
set val(y)         100
set val(nn)        6

set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

$ns_ node-config -adhocRouting  $val(rp) \
                 -llType        $val(ll) \
                 -macType       $val(mac) \
                 -ifqType       $val(ifq) \
                 -ifqLen        $val(ifqlen) \
                 -antType       $val(ant)\
                 -propType      $val(prop) \
                 -phyType       $val(netif) \
                 -topoInstance  $topo \
```

```
                -channel        [new $val(chan)] \
                -agentTrace     ON \
                -routerTrace    OFF \
                -macTrace       OFF \
                -movementTrace OFF
```

De forma a facilitar o trabalho de configuração de simulações com parâmetros diferentes, é comum definir as variáveis referentes aos componentes do *MobileNode* no início do *script* que descreve o modelo de rede a ser simulado.

É importante observar que neste trecho de *script*, referente a uma comunicação sem fio simples, foi utilizado o protocolo de roteamento *ad hoc* denominado `DumbAgent` [14]. Este é um caso especial de uma classe utilizada para ignorar a utilização de um protocolo de roteamento, já que no exemplo completo os nós estão no mesmo domínio de colisão, ou seja, eles podem se comunicar diretamente entre si sem a necessidade de comunicação *multi-hop*.

Além do protocolo de roteamento DumbAgent, dois outros componentes necessários à configuração do *MobileNode* que estão presentes na configuração da Listagem 11 e ainda não foram mencionados são: a topografia (ou, em outras palavras, o terreno) onde os nós estão dispostos e os níveis de *trace* a ser produzidos pelos nós. A topografia define um tipo de terreno e uma área limite para a localização dos nós, a qual é especificada em metros quadrados. Sendo assim, as coordenadas x e y que definem suas dimensões devem estar em metros. Este objeto servirá como referência para a verificação dos limites das coordenadas dos nós a ser especificadas posteriormente.

A geração de informações sobre a simulação é representada no NS-2 por níveis de *trace*. Por definição da língua inglesa, *trace* significa “encontrar ou descobrir por investigação; descrever a origem ou desenvolvimento de algo” [30]. É justamente esse o objetivo das informações geradas pelo *trace* da simulação: investigar o comportamento da rede modelada. Para as simulações sem fio feitas com o NS-2 existem 4 níveis de *trace*: *trace* de agentes, *trace* de roteadores, *trace* de camada MAC e *trace* de movimento. Para o estudo realizado e aqui descrito, apenas o *trace* de agentes é relevante, já que se está interessado apenas em métricas de comunicação entre agentes de tráfego (comunicação fim-a-fim).

Após a configuração do *MobileNode*, deve-se realmente criar os nós da rede. Este procedimento está descrito na Listagem 12.

Listagem 12. Criação dos nós da rede sem fio referentes à Listagem 11.

```
for {set i 0} {$i < $val(nn)} {incr i} {
        set node_($i) [$ns_ node]
```

```
            $node_($i) random-motion 0
    }
```

Os nós *MobileNode* previamente configurados são criados com auxílio da função node da classe Simulator. A segunda linha dentro do laço "for" indica que este nó não deve realizar movimento aleatório. No caso dessa simulação de exemplo não é desejável que os nós realizem movimento aleatório, mas sim que a movimentação seja especificada em conjunto com a definição do cenário da simulação.

3.2.2 Configuração da Posição e Movimento dos Nós *MobileNode*

A configuração da posição e movimento dos nós de uma rede sem fio é, em sua essência, a configuração do cenário desejado para a simulação. Os nós *MobileNode* foram projetados e implementados para se movimentar numa topologia tridimensional, no entanto, a terceira dimensão ainda não é usada pelas simulações realizadas com o NS-2 [5].

Existem basicamente duas formas de se especificar posição e movimento de um nó: configurar o nó para realizar movimento aleatório (conforme descrito na seção 3.2.1) e configurar manualmente sua posição inicial e destino numa movimentação com o uso de coordenadas. Este último tipo de configuração está exemplificado na Listagem 13.

Listagem 13. Exemplo de configuração da posição e movimento de um nó sem fio.

```
$node_(0) set X_ 91.731140362117
$node_(0) set Y_ 23.531104229966
$node_(0) set Z_ 0.000000000000

$ns_ at 40.000000000000 "$node_(0) setdest 92.12 24.94 4.0"
```

Conforme anunciado anteriormente, a posição de um nó deve ser configurada apenas através das coordenadas x e y, sendo a coordenada z configurada como zero. A configuração da movimentação acontece com a configuração discreta do agendador de eventos, utilizando a função auxiliar `at`. A *string* de comando faz com que o simulador movimente o nó `node_(0)` para o ponto (92,12;24,94) do plano definido pela topografia da simulação, com uma velocidade de 4,0 m/s.

3.2.3 Configuração do General Operations Director (GOD)

O *General Operations Director* (usualmente tratado pela sigla GOD) é um objeto utilizado pelo NS-2 para armazenar informações globais sobre o estado do cenário constituído pela configuração da rede e os nós que nela estão presentes. Este objeto atua fornecendo uma visão

completa da rede que um observador onisciente teria, sem que este observador fosse parte integrante desta rede. A finalidade do instanciamento deste objeto (Listagem 14) e de sua configuração é acelerar a computação do roteamento e conectividade da rede fornecendo informações sobre esta conectividade para serem usadas em tempo de simulação [14].

Listagem 14. Instanciamento do objeto GOD.

```
set god_ [create-god $val(nn)]
```

Após ter sido instanciado, o objeto guarda o número de nós envolvidos no modelo e pode ser configurado com o menor número de saltos (nós) entre um nó e outro, evitando o cálculo de caminho mais curto (*shortest path*) de rotas entre nós em tempo de execução. A Listagem 15 exibe algumas linhas referentes à configuração do objeto GOD.

Listagem 15. Exemplo de configuração do objeto GOD.

```
$god_ set-dist 0 1 2
$god_ set-dist 0 2 3
$god_ set-dist 0 3 4
```

A primeira linha da Listagem 15 deve ser interpretada como a menor distância em número de saltos entre o nó 0 (zero) e o nó 1 (um) é igual a 2 e as linhas seguintes seguem o mesmo raciocínio.

3.3 Geração Automática de Cenários e Padrões de Tráfego

Com o intuito de auxiliar a realização de simulações com várias configurações de posição, movimento e tráfego entre nós em uma rede sem fio, a equipe do Projeto Monarch desenvolveu dois programas que estão disponíveis no NS-2: o setdest e o cbrgen.tcl.

Ambos os programas estão localizados no diretório ns-2.33/indep-utils/cmu-scen-gen. O setdest é um gerador de cenários, ou seja, posições e movimentos dos nós, desenvolvido em C++ e já estará compilado caso se esteja utilizando a versão allinone mencionada no Capítulo 2. O cbrgen.tcl, apesar do nome indicar apenas o tráfego do UDP do tipo CBR, pode ser usado para gerar padrões de tráfego tanto para fluxos UDP simulando aplicações do tipo CBR quanto TCP simulando a aplicação de transferência de arquivos FTP.

3.3.1 Geração de Cenários com setdest

A geração de cenários com o programa setdest é bastante direta. Basta ter uma idéia de como se deseja modelar a rede a ser simulada e especificar as flags disponibilizadas pelo programa. A Listagem 16 mostra um exemplo de cenário simples a ser criado.

Listagem 16. Execução do programa setdest para geração de um cenário de rede sem fio.

```
./setdest -v 1 -n 50 -p 900 -M 20 -t 900 -x 1500 -y 300 > scenario.tcl
```

Os parâmetros utilizados foram:

- -v 1: versão do cenário a ser gerada (a versão 1 corresponde à versão original desenvolvida pela equipe do Projeto Monarch);
- -n 50: numero de nós do modelo igual a 50;
- -p 900: *pause time* igual a 900s (tempo em que os nós devem ficar parados);
- -M 20: velocidade máxima dos nós igual a 20m/s;
- -t 900: tempo total da simulação igual a 900s;
- -x 1500: limite da coordenada x do terreno usado pela simulação igual a 1500m;
- -y 300: limite da coordenada y do terreno usado pela simulação igual a 300m.

Como pode-se notar, foi utilizado um valor de *pause time* igual ao valor de tempo total da simulação. Isto foi feito propositalmente, para geração de um cenário onde não haverá movimento dos nós. Caso se deseje gerar padrões de movimento em redes sem fio, basta especificar um valor de pause time menor que o tempo total da simulação. Assim, o setdest gerará padrões de movimento usando o algoritmo *Random Waypoint* [15].

Como o setdest cria o cenário e o imprime na saída padrão (stdout) do shell do sistema operacional, é necessário redirecionar a saída para um arquivo para posterior carregamento do mesmo dentro do *script* que define a simulação. Isto é feito através do caractere “>” seguido do nome do arquivo a conter as informações capturadas da saída padrão.

3.3.2 Geração de Padrões de Tráfego com cbrgen.tcl

A geração de padrões de tráfego com o auxilio do *script* cbrgen.tcl é tão simples quanto a geração de cenários descrita na seção passada. O cbrgen.tcl gera padrões de tráfego baseado no número total de nós, o número máximo de conexões entre os nós, a taxa de envio (em pacotes/s) e o tipo de tráfego da camada de aplicação, devendo ser executado de acordo com o exemplo da Listagem 17.

Listagem 17. Geração de padrão de tráfego com o *script* cbrgen.tcl.

```
$ ns cbrgen.tcl -type cbr -nn 50 -seed 2 -mc 30 -rate 4.0 > traffic.tcl
```

Como o cbrgen.tcl está escrito em OTcl, o próprio binário do NS-2 (denominado "ns") deve ser usado como interpretador para execução do *script*. O *script* utiliza uma semente para o gerador de números aleatórios que, por sua vez, é usado para definir o tempo de início de cada um dos fluxos de tráfego. Este tempo é um valor aleatório entre 0 e 180 segundos.

Assim como o setdest, o cbrgen.tcl imprime a configuração de fluxos de tráfego na saída padrão e, por este motivo, a saída deve ser redirecionada para um arquivo para posterior carregamento por um *script* de configuração de uma simulação.

3.4 Análise do Arquivo de *Trace*

O NS-2 disponibiliza dois formatos básicos de *trace* para simulações sem fio: o antigo formato do Projeto Monarch, também conhecido como CMU-*Trace*, e um novo formato de *trace* o qual foi desenvolvido como um esforço para integrar o antigo formato de *trace* com o formato de *trace* genérico, ou seja, para qualquer tipo de simulação, do NS-2. No entanto, no momento da escrita deste trabalho, o suporte ao novo formato está disponível apenas para as simulações sem fio.

Apesar da ainda grande utilização do formato antigo, neste trabalho optou-se por utilizar o novo formato. Esta escolha foi feita devido ao novo formato ter sido escolhido pelos desenvolvedores do NS-2 para ser o formato de *trace* padrão para todos os tipos de simulação disponibilizadas pela ferramenta. Para utilizá-lo nas simulações realizadas neste trabalho, foi feita a configuração do agendador de eventos exibida na Listagem 18.

Listagem 18. Configuração do novo formato de *trace* no NS-2.

```
$ns_ use-newtrace
set tracefd [open "wireless-trace.tr" w]
$ns_ trace-all $tracefd
```

Como mostrado, a configuração do novo formato deve ser feita antes do comando de *trace* universal, `trace-all`, pois a primitiva `use-newtrace` configura uma variável do simulador denominada "newTraceFormat", de forma a possibilitar a produção de *trace* no novo formato.

3.4.1 Exemplo da Saída Produzida pelo Novo Formato de *Trace*

O novo formato de *trace* do NS-2 disponibiliza uma gama de informações sobre as camadas, os nós e o roteamento dos pacotes entre eles. Algumas linhas da saída do novo formato de *trace* podem ser vistas na Listagem 19. Nesta Listagem estão transcritas 6 linhas de *trace* que, dado seu comprimento, estão dispostas com quebras de linha. Para facilitar a identificação, cada nova linha de *trace* começa com o caractere "s" ou "r".

Listagem 19. Exemplo de saída do novo formato de *trace* do NS-2.

```
    s -t 0.086515825 -Hs 3811 -Hd -2 -Ni 3811 -Nx 5821.40 -Ny 1658.53 -Nz
0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 3811.1 -Id
3812.0 -It cbr -Il 64 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 0
    s -t 0.121304076 -Hs 1439 -Hd -2 -Ni 1439 -Nx 9551.09 -Ny 2971.31 -Nz
0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 1439.0 -Id
1440.0 -It cbr -Il 64 -If 0 -Ii 1 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 0
    s -t 0.129406862 -Hs 730 -Hd -2 -Ni 730 -Nx 1782.55 -Ny 3459.60 -Nz
0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 730.1 -Id
731.1 -It cbr -Il 64 -If 0 -Ii 2 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 0
    s -t 0.139662372 -Hs 50 -Hd -2 -Ni 50 -Nx 5148.92 -Ny 426.73 -Nz 0.00
-Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 50.1 -Id 52.0 -
It cbr -Il 64 -If 0 -Ii 3 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 0
    s -t 0.151689062 -Hs 3262 -Hd -2 -Ni 3262 -Nx 511.00 -Ny 2046.89 -Nz
0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 3262.0 -Id
3263.0 -It cbr -Il 64 -If 0 -Ii 4 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 0
    r -t 0.195079601 -Hs 3812 -Hd 3812 -Ni 3812 -Nx 5451.95 -Ny 1095.33 -
Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 13a -Md ee4 -Ms 2257 -Mt 800 -
Is 3811.1 -Id 3812.0 -It cbr -Il 84 -If 0 -Ii 0 -Iv 27 -Pn cbr -Pi 0 -Pf
4 -Po 0
```

3.4.2 Análise da Saída Produzida pelo Novo Formato de *Trace*

Para o estudo das métricas deste trabalho, apenas algumas informações contidas em cada linha de *trace* são relevantes. Cada uma destas linhas descreve um evento realizado pelo objeto Simulator. Cada evento possui um tipo e sua descrição em termos de tempo de ocorrência, informações de posição, nível de *trace*, informações de pacote no nível da camada de rede (IP), entre outros.

Essas informações estão separadas por um caractere de espaço em branco e, com exceção da primeira informação, que é um caractere de indicação do tipo do evento, todas as demais informações são precedidas por *flags* que indicam seu significado.

Existem 4 tipos de eventos, que são:

- s – envio (*send*);
- r – recebimento (*receive*);

- d – descarte (*drop*);
- f – encaminhamento (*forward*).

No nível de nós, a informação mais relevante é o nível de *trace* (-Nl) que pode ser seguido por AGT (agent), RTR (router) ou MAC (quadro da camada de enlace).

No nível de pacotes da camada IP, as informações de interesse são:

- -Is endereçoIPOrigem.portaOrigem;
- -Id endereçoIPDestino.portaDestino;
- -Il tamanho-em-Bytes;
- -Ii id único de identificação do fluxo.

Capítulo 4

Análise da Vazão e Atraso com o Escalonamento das Redes Testadas

Neste capítulo são descritos os estudos e experimentos de interesse principal deste trabalho. Aqui estão descritas as pesquisas e testes iniciais realizados, configurações desses testes, testes de interesse do trabalho e suas configurações, como aproximações para comportamento das simulações com o funcionamento de equipamentos reais e as suposições para a realização das simulações. Finalmente, são mostrados os resultados da análise das simulações estudadas.

4.1 Estudo Inicial de Simulações de Redes *Ad Hoc* com Grande Quantidade de Nós

A simulação é uma forma bastante conveniente de explorar redes *ad hoc*, dada a dificuldade de gerenciar muitos dispositivos e o alto custo necessário para arcar com uma estrutura de testes utilizando equipamentos reais.

Muitos estudos foram feitos a respeito do desempenho de algoritmos de roteamento *ad hoc* [2,17], no entanto, o cenário e a quantidade de nós utilizados nesses estudos possuem dimensões bastante reduzidas. Em um estudo anterior, cujo objetivo era comparar o desempenho de dois algoritmos de roteamento *ad hoc* [17], a rede possuía configuração máxima de acordo com as seguintes especificações: topografia com dimensões de 2200 x 600m, 100 nós e 40 fluxos de tráfego CBR. Este modelo de rede foi usado para sustentar a argumentação principal do estudo, de que os algoritmos AODV e DSR mantém um comportamento semelhante em termos de roteamento eficiente para entregas de pacotes bem sucedidas. Segundo os autores, não havia sido possível simular modelos de rede maiores devido à lentidão do NS-2.

Testes posteriormente realizados [16] com uma continuação do estudo feito em [17], demonstraram que para redes maiores, o algoritmo DSR perde consideravelmente sua eficácia. Este exemplo simples demonstra quão relevante pode ser a consideração do escalonamento para modelos de larga escala de redes *ad hoc* para o estudo dessas redes.

4.1.1 Extensões para Simulações *Ad Hoc* em Larga Escala

Durante a fase inicial de projeto das simulações, foram feitas pesquisas visando a obtenção de conhecimento a respeito de métodos de melhorar o desempenho da execução de simulações *ad hoc* em larga escala (mais de 1000 nós) no NS-2. Essas pesquisas levaram ao conhecimento de duas extensões do NS-2: *Staged Network Simulator* [18] e *Fast*-NS2 [16].

O *Staged Network Simulator* (ou SNS, para simplificar) é uma extensão que visa a eliminação das computações redundantes feitas pelo simulador, como, por exemplo, a posição dos nós e níveis de potência de transmissão/recepção que são recomputados completamente a cada mudança na rede. As técnicas utilizadas pelos autores desta extensão para atingir uma melhora de desempenho nesses casos é o *caching* de funções e reuso de valores resultantes de operações computacionalmente caras. Na tentativa de instalar o SNS para uso com o NS-2, observou-se que esta extensão é baseada numa versão muito antiga do simulador (2.1b9a) e depende de bibliotecas do sistema operacional GNU/Linux de versões ultrapassadas há mais de 4 anos [19]. Devido, em parte, aos requisitos obsoletos, não foi dado prosseguimento à instalação do SNS com o NS-2.

A extensão *Fast*-NS2 foca o problema da lentidão do NS-2 para simulações *ad hoc* basicamente na computação realizada para o cálculo da interferência entre os nós. Para se ter uma idéia da magnitude desses cálculos, um sistema com N pares de transmissores e receptores requer cálculo de interações entre eles com complexidade $O(N^2)$, ou seja, a interferência de cada nó em relação a todos os demais deve ser calculada [16]. Como os sinais de rádio decaem exponencialmente com a distância entre transmissor e receptor, após alguma distância do transmissor, nenhum receptor será afetado, portanto, pode-se desconsiderar os cálculos de interação entre eles. Caso considerados, os cálculos serviriam apenas como computação desnecessária para a simulação e consumidora de tempo de execução.

Dado o sucesso da idéia implementada, a extensão *Fast*-NS2 foi incorporada à versão oficial do simulador a partir da versão 2.27 [20] e está presente até a versão atual utilizada por este trabalho (2.33). Apesar de sugerir e implementar duas abordagens algorítmicas para conseguir a diminuição da computação dos cálculos de interferência entre os nós, uma baseada em *grid* e outra baseada em lista, a implementação escolhida para

incorporação à versão oficial do NS-2 pelos seus desenvolvedores principais foi a baseada em lista.

4.1.2 Configuração das Simulações para Aproximação com Equipamentos Atualmente em Uso

As simulações conduzidas durante o desenvolvimento deste trabalho foram configuradas de forma a se aproximar, sempre que possível, das configurações da maioria dos equipamentos sem fio IEEE 802.11 atualmente em uso. Ao configurar um modelo sem fio apenas com os componentes de um *MobileNode*, o NS-2 utiliza valores *default* para configuração de parâmetros mais específicos dos componentes de uma simulação [21].

Taxa de Transmissão de Dados

Por *default*, o NS-2 utiliza taxa de transmissão de dados de 2Mbps. Atualmente, os equipamentos disponíveis no mercado dispõem de taxas de transmissão maiores, as quais são utilizadas *out-of-the-box*. Nas simulações executadas neste trabalho, optou-se por utilizar uma taxa de transmissão de 11Mbps, utilizando-se como base para tal escolha o grande uso da tecnologia IEEE 802.11b, a qual utiliza esta taxa. Isso é feito configurando-se o objeto Mac/802_11 como mostra a Listagem 20.

Listagem 20. Configuração da taxa de transmissão dos dados utilizada.

```
Mac/802_11 set dataRate_ 11mb
```

RTS Threshold

A maioria dos equipamentos IEEE 802.11 disponíveis possuem o mecanismo RTS/CTS (*Request To Send/Clear To Send*) desabilitado. Para simular as redes evitando o *overhead* gerado pelas requisições RTS/CTS, é preciso configurar o objeto Mac/802_11 com um valor maior que o tamanho em bytes de qualquer pacote utilizado na simulação.

Listagem 21. Configuração do tamanho mínimo de pacote necessário para ativar o mecanismo RTS/CTS.

```
Mac/802_11 set RTSThreshold_ 3000
```

A configuração mostrada na Listagem 21 faz com que o mecanismo RTS/CTS seja usado apenas quando o tamanho do pacote a ser enviado ultrapasse 3000 bytes, um valor que extrapola em muito o tamanho máximo permitido para o *payload* de um pacote IP dentro de um frame IEEE 802.11. Como esse valor não será ultrapassado pelo tamanho dos pacotes trafegando, pode-se considerar que o mecanismo RTS/CTS está desabilitado.

4.1.3 Linguagem de Programação Utilizada pelos *Scripts* de Análise das Métricas de Interesse ao Trabalho

Para o desenvolvimento dos artefatos de software utilizados por este trabalho para analisar os arquivos de *trace* gerados pelas simulações executadas no NS-2, optou-se por utilizar a linguagem de programação Perl [23]. Esta linguagem é bastante conhecida por sua praticidade de programação e grande poder de manipulação de arquivos e padrões através do uso de expressões regulares.

4.1.4 Configuração de Hardware da Máquina Utilizada para Realizar as Simulações

Todas as simulações realizadas durante o desenvolvimento deste trabalho foram feitas numa máquina com processador Intel® Core2Quad Q6600 com quatro núcleos, cada qual com freqüencia de *clock* de 2.4GHz, 8GB de memória RAM e 6GB de *swap*.

4.2 Simulações Iniciais

De acordo com as configurações relatadas em um estudo sobre a análise de desempenho dos algoritmos de roteamento *ad hoc* [2], que utilizou o NS-2, foram feitas simulações iniciais para verificação do resultado das medidas de vazão e atraso médio de todos os nós geradores de tráfego. Estas simulações utilizaram 50 nós, 3 padrões de tráfego e 7 cenários distintos, com mobilidade dos nós e sem mobilidade, totalizando a geração de 42 gráficos, sendo 21 de vazão média e 21 de atraso médio.

4.2.1 Configuração dos Padrões de Tráfego

Para realização das simulações iniciais, foram usados 3 padrões de tráfego conforme a configuração a seguir:

- Tráfego UDP do tipo CBR;
- Tamanho do pacote: 64 bytes;
- Taxa de transmissão dos pacotes: 4 pacotes/segundo;
- Número máximo de conexões de tráfego: 10, 20 e 30.

Devido à variação na quantidade de conexões de tráfego CBR utilizadas, foram gerados 3 arquivos com o seguinte padrão:

tp-cbr-nn-s-mc-tx.tcl

Onde, “cbr” é o tipo de tráfego, “nn” é o numero de nós da rede, “s” é a semente para geração de números aleatórios para definir quais nós serão usados como origem e destino nas conexões, “mc” é o numero máximo de conexões e “tx” é a taxa de transmissão de pacotes. Assim, com o auxílio do programa cbrgen.tcl, foram gerados os arquivos mostrados na Listagem 22.

Listagem 22. Arquivos de tráfego CBR usados nas simulações iniciais.

```
tp-cbr-50-2-10-4.tcl
tp-cbr-50-2-20-4.tcl
tp-cbr-50-2-30-4.tcl
```

Conforme a configuração original do cbrgen.tcl, os fluxos de tráfego são configurados para iniciar em instantes de tempo aleatórios, variando de 0 a 180 segundos.

4.2.2 Configuração dos Cenários

A configuração dos cenários é caracterizada principalmente pelo componente *pause time*. Este componente define quanto tempo um nó deve ficar parado, sem realizar qualquer movimento, desde o início da simulação. Depois do tempo especificado pelo *pause time*, o nó se move na direção de coordenadas especificadas pelo arquivo de cenário. Após atingir o destino, o nó fica parado novamente pelo mesmo período de *pause time* inicial, repetindo esse comportamento durante todo o tempo da simulação.

O tempo total de simulação para os cenários descritos nesta seção é de 900 segundos. Foram selecionados 7 diferentes *pause times*: 0, 30, 60, 120, 300, 600, 900. O primeiro valor corresponde à mobilidade geral da rede durante todo o tempo de simulação, enquanto o último corresponde a um comportamento totalmente estacionário, onde não há movimentação alguma dos nós. A velocidade máxima de deslocamento dos nós foi configurada para 20m/s e também é definida para um valor aleatório no arquivo de cenário. A área definida para topografia da simulação foi de 1500 x 300m.

Devido à variação nos valores de *pause time* utilizados, foram gerados 7 arquivos com o seguinte padrão:

scen-XxY-nn-p-m-ts.tcl

Onde, “XxY” corresponde às dimensões da topografia (1500x300), “nn” ao numero de nós da rede, “p” ao valor de *pause time*, “m” à velocidade máxima de deslocamento de um nó e “ts” ao tempo total da simulação. Com o auxílio do programa setdest, foram gerados os arquivos conforme mostrado na Listagem 23.

Listagem 23. Arquivos de cenário usados nas simulações iniciais.

```
scen-1500x300-50-0-20-900.tcl
scen-1500x300-50-30-20-900.tcl
scen-1500x300-50-60-20-900.tcl
scen-1500x300-50-120-20-900.tcl
scen-1500x300-50-300-20-900.tcl
scen-1500x300-50-600-20-900.tcl
scen-1500x300-50-900-20-900.tcl
```

4.2.3 Resultados da Análise de Vazão

Alguns gráficos referentes aos resultados da análise de vazão das redes simuladas são mostrados abaixo. Pode-se notar que nos casos em que os nós permanecem mais tempo parados, ou seja, há menos movimentação e consequentemente menos perturbação em termos de interferência e necessidade de re-roteamento da rede, a vazão média é um pouco maior. Em termos práticos, não há diferença significativa entre os valores de vazão média calculados para essa configuração de rede.

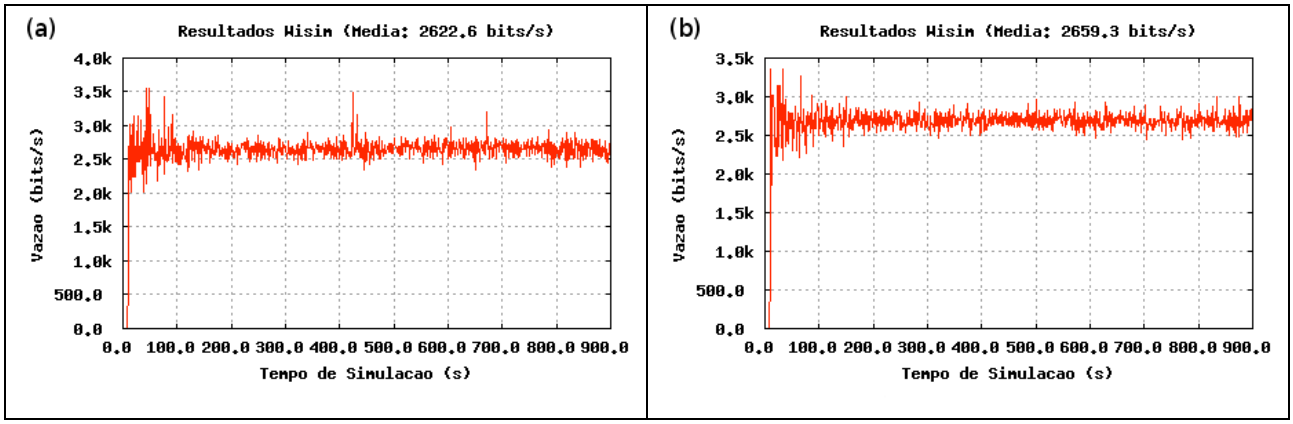

Figura 9. Comparação entre a vazão média de 30 fontes de tráfego com *pause times* de 0 (a) e 900 s (b).

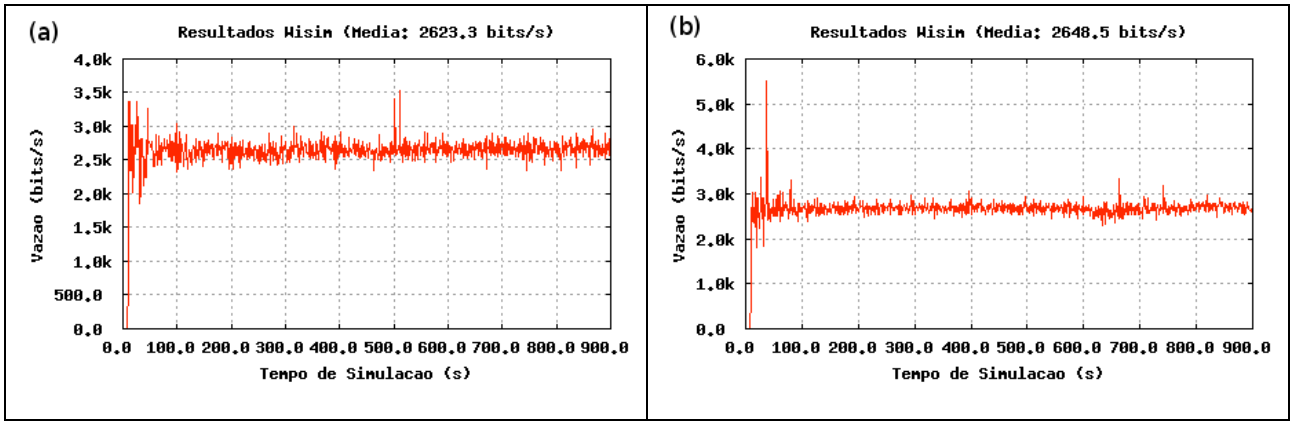

Figura 10. Comparação entre a vazão média de 30 fontes de tráfego com *pause times* de 30 (a) e 600 s (b).

4.2.4 Resultados da Análise de Atraso

Os gráficos da análise de atraso médio de algumas das redes simuladas inicialmente, também mostram que a variação do atraso e o atraso médio de redes com menor movimentação de nós é menor do que o atraso das redes com mais movimento (menor *pause time*). Acredita-se que a diferença média dos valores de atraso tenha sido maior do que a diferença média dos valores de vazão dada a maior necessidade de re-roteamento da rede quando os nós estão em movimento. Esse re-roteamento causa maiores variações no valor médio de atraso dos pacotes, que são perceptíveis pela análise visual dos gráficos.

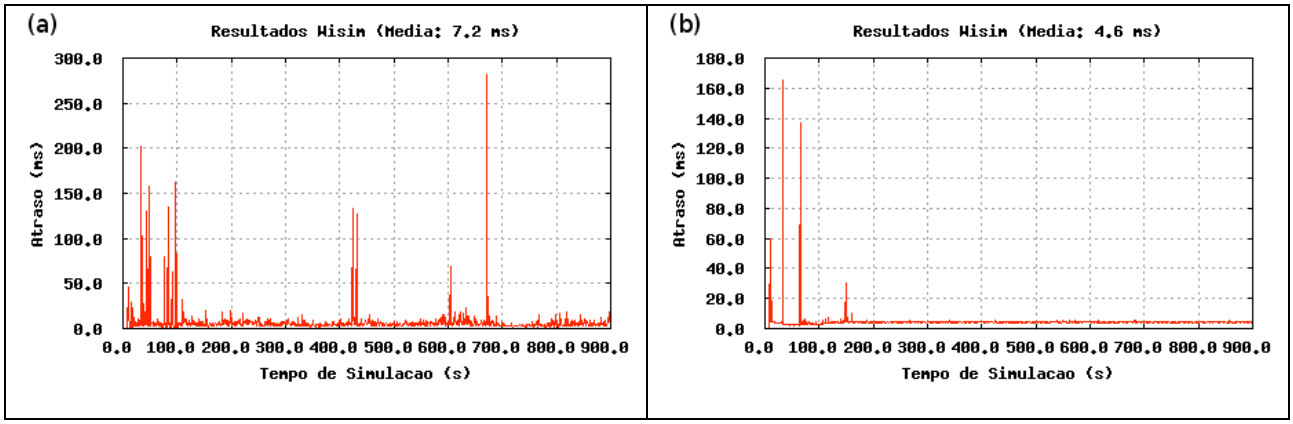

Figura 11. Comparação entre o atraso médio de 30 fontes de tráfego com *pause times* de 0 (a) e 900s (b).

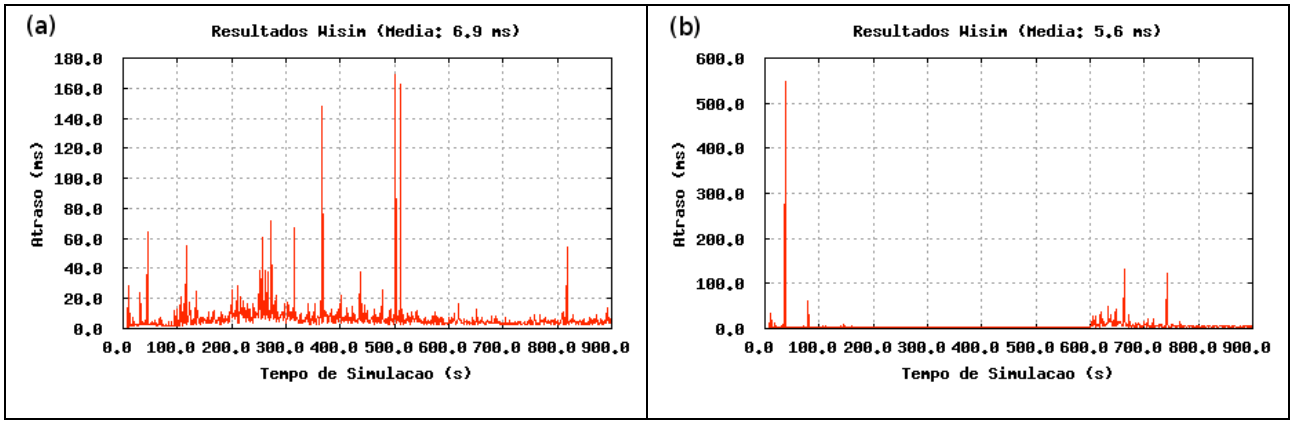

Figura 12. Comparação entre o atraso médio de 30 fontes de tráfego com *pause times* de 30 (a) e 600 s (b).

4.3 *Script* de Análise de Vazão

O *script* de análise de vazão foi desenvolvido em duas versões: tp_calc.pl e tp_calc_avg.pl. A primeira delas realiza o cálculo da vazão de cada um dos fluxos da rede analisada independentemente. Dessa forma, os gráficos produzidos mostram múltiplas linhas, cada qual correspondendo a um fluxo de tráfego. A segunda versão realiza uma média dos valores de vazão

calculados para cada fluxo, tendo sido usada para gerar os valores para plotagem dos gráficos de vazão exibidos na Figura 9 e 10.

O algoritmo do *script* de análise de vazão funciona de uma maneira bastante simples e direta: realiza um somatório dos valores de pacotes recebidos por cada fluxo a cada segundo e guarda esses valores e a janela de tempo (no caso, o segundo) em que cada um desses valores foi recebido. Este somatório foi feito em relação à quantidade de dados transferida apenas entre agentes de tráfego (nó de origem e nó de destino), ignorando dados transmitidos por nós atuando como roteadores numa transmissão *multi-hop*.

Na primeira versão do *script*, o armazenamento da quantidade de dados recebidos por segundo é feita separadamente para cada fluxo. De acordo com a seção 3.4.2, foram utilizados os campos referentes ao endereçamento IP do arquivo de *trace*, mais precisamente os campos –Is e –Id para identificar um fluxo e o campo –Il para computar a quantidade de dados recebida.

Na segunda versão, a qual é simplesmente uma versão pouco modificada da primeira, não era necessário o armazenamento da quantidade de dados recebidos independentemente para cada fluxo. Portanto, é realizado um somatório dos valores recebidos por todos os fluxos num dado segundo. A diferença dessa versão é que, como é preciso realizar uma média, é preciso identificar quantos fluxos estão ativos num dado segundo. Isto é feito com o auxílio dos arrays @traffic_sources e @flow_conn. Ambos os *scripts* estão transcritos na íntregra nos Apêndices A e B, respectivamente.

4.3.1 Validação do *Script* de Análise de Vazão

Foi possível validar o *script* de análise de vazão em fluxos independentes e a configuração da plotagem do gráfico com o programa gnuplot [31] através da comparação com a saída produzida por um dos *scripts* de teste de um artigo que aborda configuração de simulações *ad hoc* [14].

Neste artigo, o *script* ex6sta.tcl corresponde a um modelo de rede *ad hoc* com um total de 6 nós e 3 fluxos de tráfego CBR entre eles. A Figura 13 mostra o gráfico gerado pelo *script* do artigo e a Figura 14 o gráfico gerado pelo *script* desenvolvido neste trabalho.

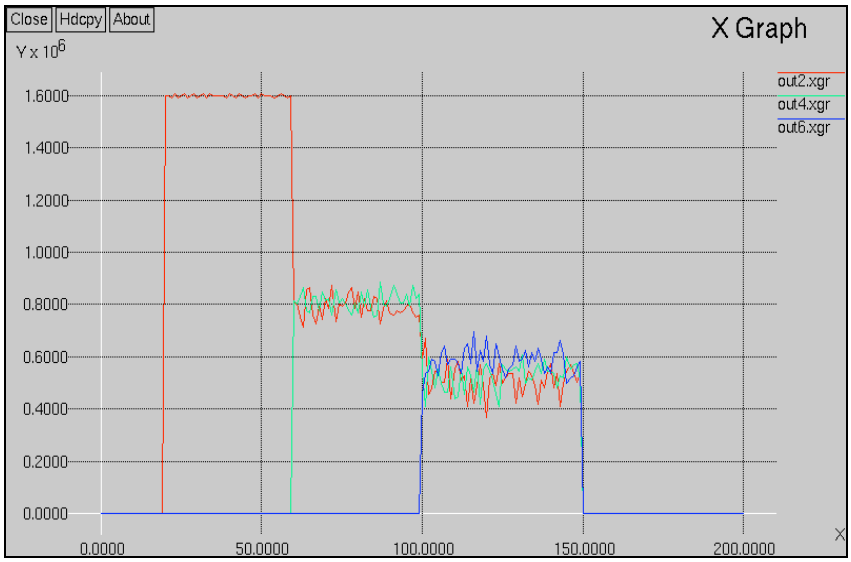

Figura 13. Gráfico de vazão gerado pelo *script* de validação da análise de vazão [22].

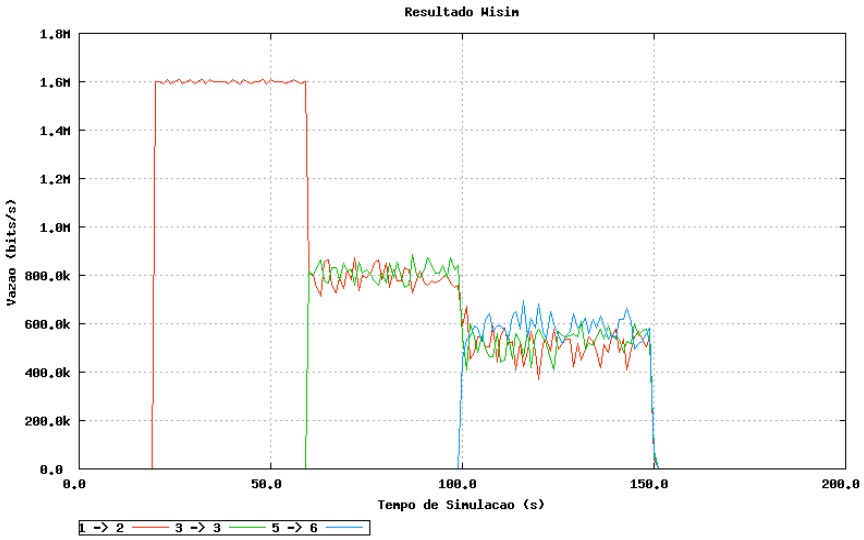

Figura 14. Gráfico gerado pelo *script* tp_calc.pl em conjunto com o gnuplot.

A Figura 13 utiliza os nomes dos arquivos de entrada para o programa de plotagem do gráfico como legenda para as linhas referentes aos fluxos (out2.xgr, out4.xgr, out6.xgr). Já a Figura 14 utiliza os ids dos nós comunicantes como legenda (1 → 2, 3 → 4, 5 → 6), significando a comunicação entre o nó de origem (à esquerda da seta) e o nó de destino (à direita da seta). Através da inspeção visual, pode-se notar que os gráficos gerados são bastante semelhantes, o que comprova a corretude do algoritmo utilizado.

4.4 *Script* de Análise de Atraso

O *script* de análise de atraso foi desenvolvido apenas em uma versão, que corresponde à computação do atraso fim-a-fim médio proveniente da comunicação entre nós da rede.

Descrevendo em linhas gerais, o algoritmo implementado armazena o tempo, em milissegundos, de cada fluxo identificado pelo IP.porta de origem e destino (campos –Is e –Id do arquivo de *trace*), o id único do fluxo (campo –Ii do arquivo de *trace*) e o evento (recebimento ou envio, que corresponde ao primeiro campo de cada linha do arquivo de *trace*). Quando ocorre um evento de recebimento, o tempo deste é guardado e é obtida a diferença resultante da subtração do tempo de envio armazenado anteriormente ao recebimento. Feito isso, é realizada uma média em relação ao número de atrasos computados por fluxo e, posteriormente, é realizada uma média dos atrasos combinados por todos os fluxos pelo número destes.

O *script* com o algoritmo completo para análise de atraso médio está transcrito no Apêndice C deste documento.

4.4.1 Validação do *Script* de Análise de Atraso

Não foi possível validar o *script* de análise de atraso com algum outro programa disponível na Internet. Apesar de, durante as pesquisas, terem sido encontrados alguns programas que se diziam capazes de realizar o cálculo do atraso [25,27], os testes realizados com eles ou não produziram qualquer resultado decorrente a falhas internas do programa ou estavam desatualizados em relação ao novo formato de *trace*. Contudo, foi realizada uma validação da corretude dos valores computados pelo programa através de uma inspeção visual do arquivo de *trace* e dos valores de atraso calculados.

4.5 Simulações com Grande Número de Nós

Vários experimentos foram conduzidos com o intuito de determinar parâmetros adequados para o modelo de rede a ser simulada. As dificuldades nesse sentido são decorrentes principalmente da necessidade de escalonamento do número de nós para o patamar de 10.000 nós.

4.5.1 Configuração do Modelo de Rede

Devido às restrições de tempo, escolheu-se simular apenas cenários estacionários, ou seja, que não possuem movimentação dos nós. Também em relação ao tempo, só foi possível utilizar um modelo de propagação de ondas de rádio, o *Two-Ray Ground*, que consiste na consideração de que um sinal de rádio pode percorrer tanto um caminho direto da antena de origem à antena de destino quanto um caminho decorrente da reflexão das ondas de rádio no solo [5].

Todos os fluxos simulados foram configurados para iniciar entre 0 e 10 segundos, através de uma distribuição aleatória realizada pelo *script* cbrgen.tcl. Foi escolhido o valor de 1/3 do total de nós atuando como fontes de tráfego (origem de um determinado fluxo) [4]. Todos os cenários foram simulados por um período de tempo de 300 segundos,

dado ao tempo necessário para simular 10.000 nós ser inviável para realização deste trabalho, com um período de tempo de simulação significativamente maior do que 300 segundos.

Outro ponto fundamental foi estabelecer as dimensões do terreno. Como o estudo abrange desde uma quantidade ínfima de nós (10) até uma grande quantidade destes (10.000), a utilização de uma dimensão fixa acarretaria em uma área com altíssima densidade de nós para as simulações com maior número destes. A conseqüência de utilizar áreas com grande densidade de nós é que os cálculos de interferência de ondas de rádio feitos pelo NS-2 são muito maiores do que com áreas de menor densidade. Dessa forma, executar as simulações com grande densidade de nós inviabilizaria a realização dessas, dadas as restrições de tempo para realização deste trabalho. Sendo assim, optou-se por utilizar dimensões variáveis, de acordo com uma proporção fixa. Esta proporção foi estabelecida como a densidade dos nós por unidade de área. Com o auxílio dos testes realizados, a proporção mais adequada, que proporcionou uma velocidade de simulação viável de se simular, foi a de 0,0002 nós por metro quadrado. Dessa forma, foram configurados os seguintes cenários:

- 500 x 100m para 10 nós;
- 1000 x 500m para 100 nós;
- 5000 x 1000m para 1000 nós;
- 10000 x 5000m para 10000 nós.

Optou-se pela utilização de tráfego UDP simulando uma aplicação CBR ao invés de tráfego TCP, pois o TCP inerentemente se adapta à carga sobre a qual a rede está submetida. Um exemplo disso é que uma fonte de tráfego TCP pode mudar o tempo em que os pacotes são enviados de acordo com a sua percepção da capacidade da rede de transportar os pacotes, alterando a contabilização das métricas aqui estudadas. Outro problema com o TCP é o *overhead* causado pelo *handshaking* realizado antes de iniciar um fluxo de dados, que também pode alterar os valores das métricas estudadas. Como o UDP é um protocolo não orientado a conexão e não possui mecanismos de adaptação à carga atual da rede, ele foi escolhido. Na simulação da aplicação CBR foi definida para *true* a variável “random_” a qual adiciona um “ruído” nos tempos de partida do agendador da simulação, com o intuito de simular com maior precisão os tempos de partida de fluxos CBR numa rede real.

Uma descrição completa do *script* que configura o modelo de rede pode ser encontrada no Apêndice E.

4.5.2 Alteração no *Script* de Geração de Padrões de Tráfego cbrgen.tcl

Houve a necessidade de alterar o *script* de geração de padrões de tráfego devido ao algoritmo utilizado não ser compatível com as especificações do projeto. O algoritmo implementado no *script* realiza uma contagem do número máximo de conexões entre nós, ao invés de uma contagem do número de nós fonte. Dessa forma, conforme descrito com mais detalhes na Seção 5.3, não é possível, por *default*, gerar padrões de tráfego para a quantidade de nós fonte desejados, que é 1/3 da quantidade total de nós da simulação, pois o algoritmo só leva em consideração o número de conexões de tráfego, independente de quantos nós fonte participarão destas conexões.

O *script* cbrgen.tcl com todas as alterações feitas ao longo deste trabalho está transcrito no Apêndice F.

4.6 Resultados Obtidos

Nesta seção são mostrados os gráficos gerados a partir da análise dos arquivos de *trace* produzidos pelas simulações de interesse principal deste trabalho, as quais visam reproduzir o comportamento de redes *ad hoc* à medida em que estas redes escalonam, ou seja, à medida em que aumentamos muito a quantidade de nós presentes e, proporcionalmente, a quantidade de nós comunicantes.

Os gráficos exibidos em seguida mostram curvas resultantes da obtenção da média aritmética por segundo dos valores de vazão e atraso da rede pela quantidade de nós fonte. Na parte superior dos gráficos está localizada a média total calculada para o tempo de simulação de 300s.

4.6.1 Simulação de 10 nós, 3 nós fonte, 300s, 500 x 100m

Vazão

A Figura 15 mostra que o gráfico da vazão média para uma rede com 10 nós (sendo 3 nós fonte) possui uma certa regularidade ao longo do tempo, apesar de variar de acordo com a introdução de pequenas variações no tempo de partida de pacotes dos fluxos de origem conforme explicado na Seção 4.5.1. A média total calculada para o tempo de simulação de 300s foi de 2684,3 bps.

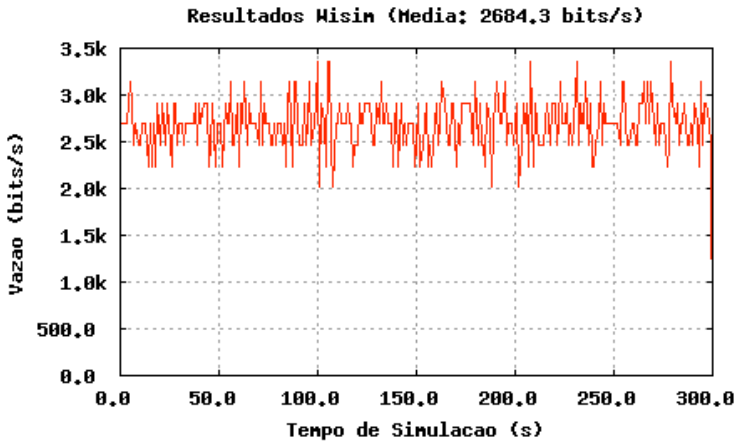

Figura 15. Gráfico da vazão média da rede com 3 nós fonte.

Atraso

O atraso médio obtido na simulação de rede com 10 nós (sendo 3 nós fonte) varia com baixo desvio em relação ao valor médio total já que, devido à quantidade reduzida de nós, ou não há necessidade de roteamento *multi-hop* ou, se houver, os atrasos introduzidos devido ao repasse do pacote por um nó roteador são mínimos. A Figura 16 mostra o gráfico onde a média total calculada para o tempo de simulação de 300s foi de 0,7ms.

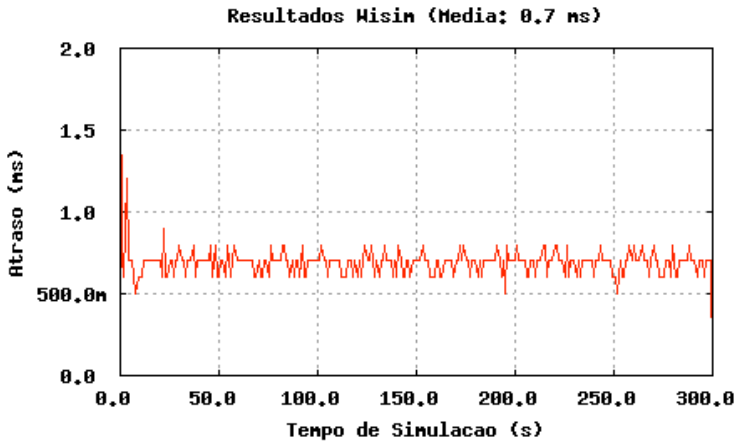

Figura 16. Gráfico do atraso médio da rede com 3 nós fonte.

4.6.2 Simulação de 100 nós, 30 nós fonte, 300s, 1000 x 500m

Vazão

A análise da simulação de 100 nós com 30 deles atuando como fontes de tráfego, resulta numa média bastante próxima da obtida na Seção 4.7.1. O gráfico resultante da análise da simulação, na Figura 17, mostra uma variação ainda menor na vazão média entre os nós ao longo do tempo de

simulação. Atribui-se a isso o fato de o posicionamento dos nós nesta simulação ter favorecido a comunicação ponto-a-ponto ao invés da comunicação *multi-hop,* conforme na Seção 4.6.1.

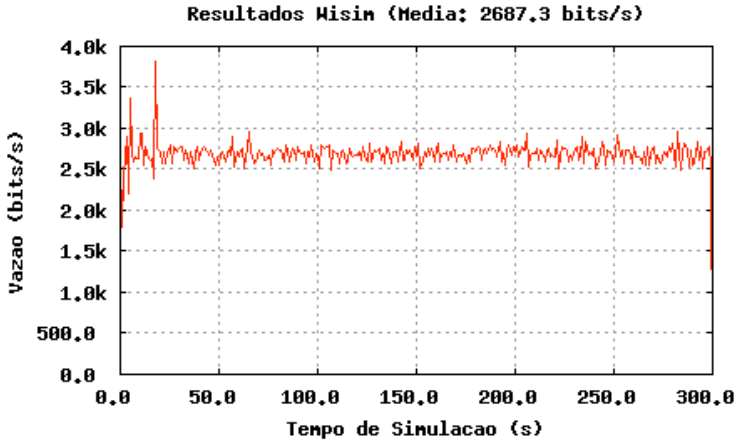

Figura 17. Gráfico da vazão média da rede com 30 nós fonte.

Atraso

Observa-se pela Figura 18 que o atraso médio sofreu um aumento em relação ao valor computado na Seção 4.7.1. Atribui-se esse aumento à grande variância do atraso no início da simulação, causada pela troca de informações de roteamento entre os nós.

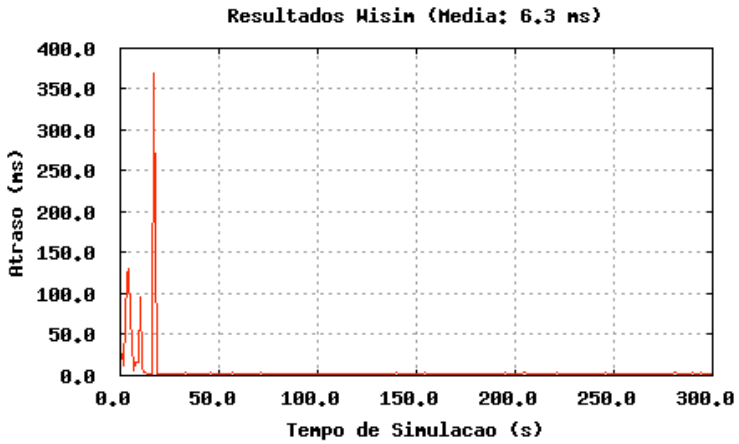

Figura 18. Gráfico do atraso médio da rede com 30 nós fonte.

4.6.3 Simulação de 1000 nós, 300 nós fonte, 300s, 5000 x 1000m

Vazão

A partir do patamar de 1000 nós (sendo 300 nós fonte) foi observada uma diminuição significativa da vazão média devido a perdas de pacote causadas principalmente por grande quantidade de roteadores entre nós fonte e destino, ou seja, perdas por esgotamento do TTL (*time-to-live*) de um pacote na rede e por *overflow* nas filas de pacotes dos nós. A Figura 19

mostra a vazão média total (2291,5 bps) e uma grande variabilidade de valores de vazão média no tempo de simulação.

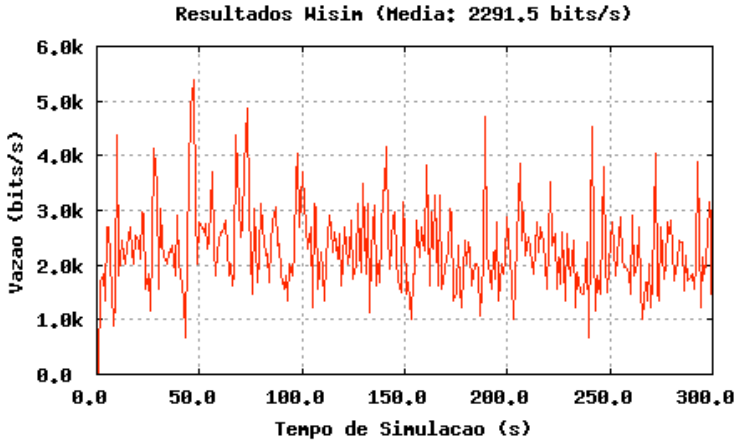

Figura 19. Gráfico da vazão média da rede com 300 nós fonte.

Atraso

Com o maior número de nós da rede e, conseqüentemente, maior número de nós comunicantes, a necessidade da comunicação *multi-hop* aumenta, com isso podemos observar uma grande variação do valor médio do atraso fim-a-fim ao longo de toda a simulação, como evidencia a Figura 20.

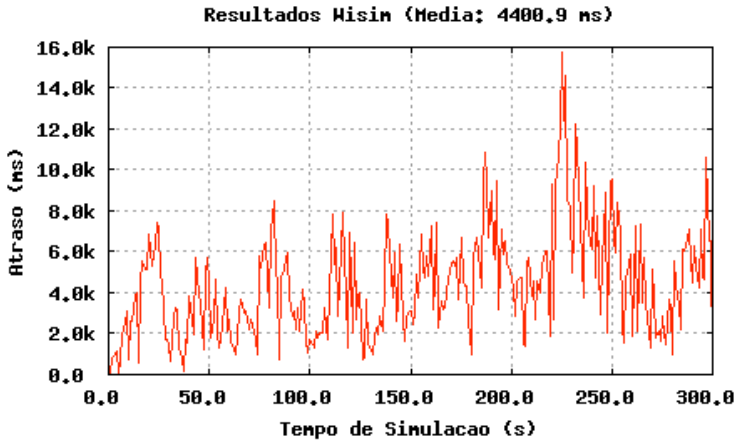

Figura 20. Gráfico do atraso médio da rede com 300 nós fonte.

4.6.4 Simulação de 10000 nós, 3000 nós fonte, 300s, 10000 x 5000m

Vazão

Com a simulação com 10.000 nós (sendo 3000 nós fonte), a vazão cai drasticamente devido à grande quantidade de perdas de pacotes causadas pelos mesmos motivos enunciados na Seção 4.6.3. Como agora a quantidade de nós é ainda maior que na Seção 4.6.3, pode-se notar pela

Figura 21 que, em alguns instantes, a vazão média chega a ser nula, ou seja, não há recebimento de pacotes nesses instantes devido às perdas.

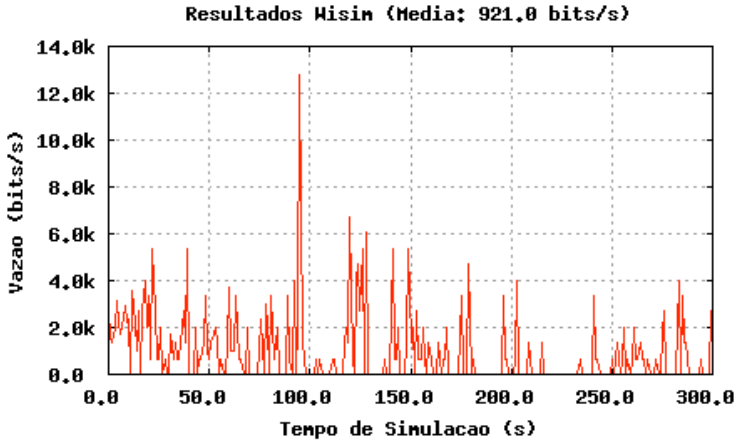

Figura 21. Gráfico da vazão média da rede com 3000 nós fonte.

Atraso

O atraso médio obtido continua a apresentar uma grande variabilidade e, em alguns momentos da simulação, as perdas são tantas que não há efetivamente recebimento de pacote para computação do valor de atraso. A Figura 22 mostra o gráfico obtido como resultado da simulação.

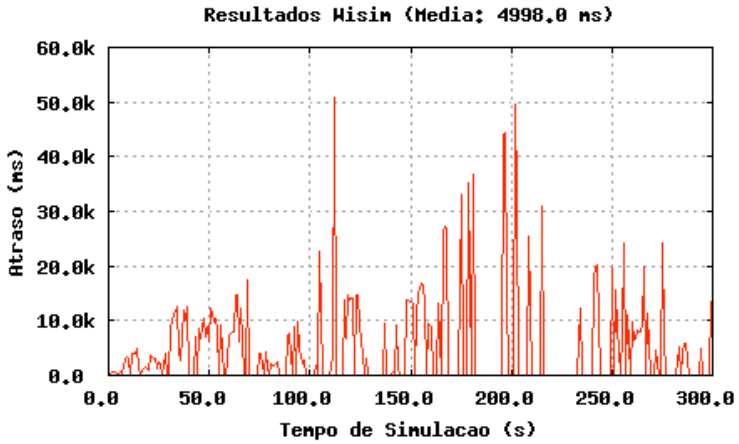

Figura 22. Gráfico do atraso médio da rede com 3000 nós fonte.

4.7 Análise dos Resultados Obtidos

De acordo com o que foi mostrado na Seção 4.6, observa-se que ambas as métricas analisadas convergiram para valores que caracterizam um menor desempenho da rede à medida em que aumentamos o tamanho da rede, ou seja, à medida em que é feito o seu escalonamento. Os

gráficos abaixo mostram as curvas formadas pelos valores médios obtidos tanto da vazão quanto do atraso da rede em escala logarítmica.

4.7.1 Vazão Média

No gráfico dos valores de vazão média computados, como mostra a Figura 23, pode-se observar que, com poucos nós (n), a comunicação entre nós em uma rede *ad hoc* ocorre com valores de vazão média praticamente inalterados (n=10 e n=100 nós). A partir de n=1000 nós, observa-se uma diminuição da vazão média da rede devido a perdas, as quais se acentuam quando a quantidade de nós aumenta para n=10.000. Como pode-se notar pelo gráfico, ainda não é possível observar uma tendência dos valores desta curva semelhante à curva de eficiência espectral exibida na Figura 1.

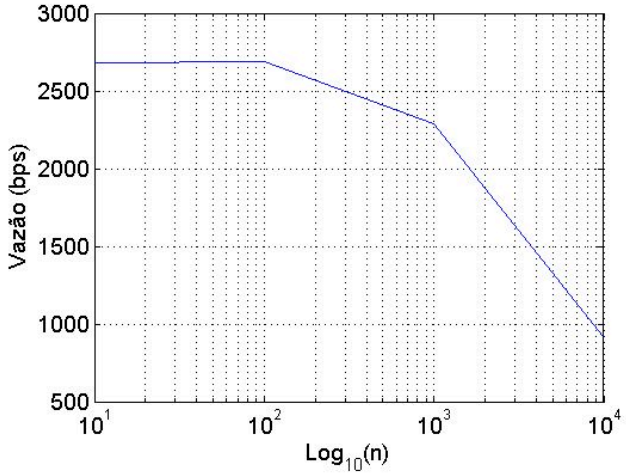

Figura 23. Curva da vazão média obtida pelos dos quatro testes da Seção 4.7 em função do número total de nós (n) da rede.

Se comparado o gráfico da Figura 23 com o gráfico do estudo analítico (Figura 1), observa-se que há uma queda mais rápida da curva deste último. A esse comportamento é atribuído o fato de, naquele trabalho [4], ser utilizada uma área fixa que contém os nós. Assim, a densidade de nós por área é variável, aumentando progressivamente à medida que se aumenta a quantidade de nós da rede. Em contrapartida, no estudo aqui demonstrado, foram utilizadas áreas de dimensões variadas, de acordo com o número de nós, mantendo uma densidade fixa à medida que se aumenta a quantidade de nós da rede. Com a manutenção de valores de densidade de nós por área fixos, obteve-se menor interferência entre os nós comunicantes e, conseqüentemente, a degradação de desempenho foi mais suave do que a apresentada no estudo analítico de capacidade.

4.7.2 Atraso Médio

Por ter-se obtido valores bastante discrepantes de atraso, ou seja, valores muito pequenos nos testes com menor quantidade de nós (n=10 e n=100 nós) e muito grandes nos testes com maior

quantidade de nós (n=1000 e n=10.000 nós), a Figura 24 mostra a curva do atraso médio obtido pelos quatro testes da Seção 4.7 com ambos os eixos em escala logarítmica. Através da observação do gráfico, pode-se notar que há um aumento significativo no valor médio do atraso a partir de n=1000 nós.

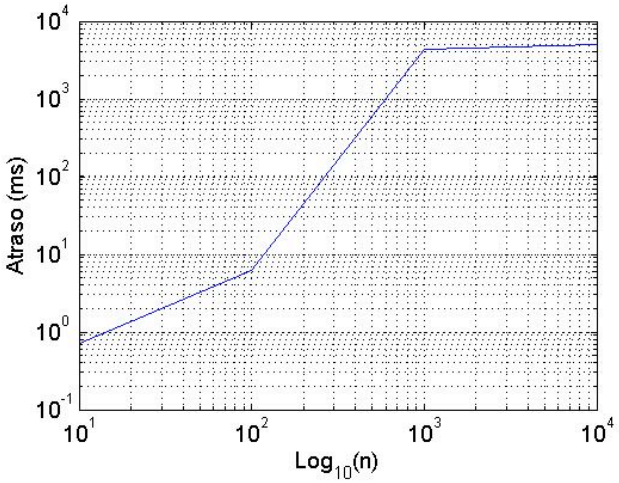

Figura 24. Curva do atraso médio obtido pelos quatro testes da Seção 4.7.

4.8 Medição de Tempo e Memória para Execução das Simulações

Nesta seção estão documentadas as informações de tempo de execução e consumo de recursos de máquina coletadas após (tempo de execução) e durante (consumo de memória) a realização das simulações de interesse do trabalho. Para realizar essas medições, foram utilizados programas utilitários do sistema operacional presentes em qualquer distribuição GNU/Linux: o programa *time* [28] e o programa *top* [29].

O programa *time* exibe 3 valores de tempo referentes a três medidas de tempo da execução de um determinado programa, com precisão de milésimos de segundos. A primeira é o tempo real, ou seja, o tempo decorrido entre o início da execução do programa (no caso, a execução do NS-2 para um determinado modelo de rede) e o seu término (identificado pela palavra "*real*"). Em outras palavras, é o tempo que pode ser percebido pelo usuário. A segunda medida é o tempo total de CPU usado pelo processo em modo usuário (identificado pela palavra "*user*"). A terceira medida é o tempo total de CPU usado pelo sistema em benefício da execução do processo, no modo *kernel* (identificado pela palavra "*sys*") [28].

O programa *top* exibe uma listagem com várias informações sobre o consumo de recursos do sistema e dos processos individualmente. De todos os campos referentes a informações sobre o consumo de recursos por processo, o campo referente à memória virtual (VIRT) é o que interessa. A memória virtual corresponde ao total de memória

usado pelo processo, incluindo código do programa, dados, bibliotecas compartilhadas e páginas de memória que tenham sido transferidas para a área de troca (*swap*), medido em KB, a não ser quando seguido pela letra "m", significando MB [29].

4.8.1 Simulação de 10 nós, 3 nós fonte, 300s, 500 x 100m

Tempo

Pode-se observar pela Listagem 24 que o tempo total de execução da simulação com 10 nós (sendo 3 nós fonte) dura muito pouco tempo, aproximadamente meio segundo para terminar.

Listagem 24. Saída do comando *time* na simulação de 10 nós sendo 3 nós fonte.

```
real    0m0.519s
user    0m0.500s
sys     0m0.010s
```

Memória

O consumo de memória para a mesma simulação, exibido na Listagem 25, é considerado razoável para uma simulação de tão pequena escala (aproximadamente 36MB).

Listagem 25. Saída do comando *top* na simulação de 10 nós sendo 3 nós fonte.

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5766 rico      18   0 36596 9.8m 5044 S    0  0.1   0:00.29 ns
```

4.8.2 Simulação de 100 nós, 30 nós fonte, 300s, 1000 x 500m

Tempo

A Listagem 26 mostra que o tempo total de execução da simulação com 100 nós (sendo 30 nós fonte) foi muito maior que a simulação com 10 nós, mesmo assim a quantidade de tempo usada pela simulação ainda está num patamar muito baixo, de, aproximadamente, 1 minuto.

Listagem 26. Saída do comando *time* na simulação de 100 nós sendo 30 nós fonte.

```
real    1m3.076s
user    1m2.830s
sys     0m0.210s
```

Memória

Apesar do aumento considerável do tempo de simulação, a quantidade de memória virtual cresceu em menor proporção. A Listagem 27 mostra que foram consumidos, aproximadamente, 52MB de memória durante a simulação.

Listagem 27. Saída do comando *top* na simulação de 100 nós sendo 30 nós fonte.

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
```

```
5592 rico      25   0 52924  25m 5028 R  100  0.3   0:47.82 ns
```

4.8.3 Simulação de 1000 nós, 300 nós fonte, 300s, 5000 x 1000m

Tempo

A simulação com 1000 nós (sendo 300 nós fonte) resultou num aumento do tempo da ordem de segundos para dezenas de minutos. Pela Listagem 28 pode-se verificar que a duração da simulação foi de, aproximadamente, 88 minutos.

Listagem 28. Saída do comando *time* na simulação de 1000 nós sendo 300 nós fonte.

```
real    88m51.946s
user    88m38.160s
sys     0m12.740s
```

Memória

O consumo de memória aumentou da ordem de dezenas de MB para centenas de MB. Pela Listagem 29, pode-se verificar que o consumo de memória durante a simulação foi de 471MB, correspondendo a 5,6% de toda memória disponível para o sistema operacional.

Listagem 29. Saída do comando *top* na simulação de 1000 nós sendo 300 nós fonte.

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
5539 rico      25   0  471m 445m 5032 R  100  5.6  31:21.30 ns
```

4.8.4 Simulação de 10000 nós, 3000 nós fonte, 300s, 10000 x 5000m

Tempo

A simulação com 10.000 nós (sendo 3000 nós fonte) durou cerca de dois dias e meio para finalizar. Esse tempo em dias pode ser calculado a partir do valor do campo "*real*" da Listagem 30. Dada a diferença significativa de tempo em relação à simulação com 1000 nós, acredita-se que a parcela mais significativa de tempo responsável por esse aumento foi a maior quantidade de cálculos de interferência realizada pelo NS-2.

Listagem 30. Saída do comando *time* na simulação de 10000 nós sendo 3000 nós fonte.

```
real    3632m46.477s
user    3614m36.620s
sys     3m59.800s
```

Memória

Durante a simulação, observou-se que a quantidade de memória consumida ultrapassou a quantidade de memória física disponível (8GB), provocando o uso da área de troca (*swap*), ou seja, usando espaço em disco para armazenar dados do processo em execução. Isto pode ser

verificado pelo valor da coluna VIRT da Listagem 31. É possível que o uso de *swap* tenha contribuído para a grande quantidade de tempo decorrido para o término dessa simulação.

Listagem 31. Saída do comando *top* na simulação de 10000 nós sendo 3000 nós fonte.

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
20053 rico      25   0 8406m 5.8g 1656 R  100 74.9   3601:00 ns
```

Capítulo 5

Problemas Encontrados

Nesta seção estão listados e brevemente descritos os problemas encontrados durante o desenvolvimento deste trabalho. Acredita-se que alguns desses problemas não possuam documentação que indique sua ocorrência, já que não foram encontradas referências a eles durante as pesquisas, tendo sido observados apenas com a realização das simulações.

5.1 Protocolo de Roteamento

Durante a realização dos testes iniciais, foram usados basicamente dois protocolos de roteamento *ad hoc*: DSR e AODV. O DSR foi o primeiro a ser testado e funcionou bem para uma quantidade de até 100 nós. Com a tentativa de realização de simulações com 1000 nós, o DSR passou a gerar erros de falha de segmentação e estouro de uso de memória, como mostra a Listagem 32. Com isso, partiu-se para a tentativa de utilização de outro protocolo e o escolhido foi o AODV. Com o AODV os problemas de falha de segmentação e estouro de uso de memória não ocorreram, mesmo com as simulações de 10.000 nós. A partir de então, este foi o único protocolo utilizado para realização das simulações e todos os resultados apresentados neste trabalho o utilizam.

Listagem 32. Captura do erro de estouro de memória causado pelo uso do protocolo DSR.

```
rico@adhoc01:~/pfc-ricardo-adhoc01$ time ns wisim-2ray.tcl
num_nodes is set 10000
INITIALIZE THE LIST xListHead
Killed

real    10m1.551s
user    0m43.170s
sys     2m22.820s
```

5.2 Geração de *Traces* Desnecessários

Durante as primeiras simulações de redes maiores, ou seja, a partir de 1000 nós, foi observado um consumo excessivo de espaço em disco pelo arquivo de *trace* da rede. O tamanho do arquivo gerado era de dezenas de gigabytes, o que inviabilizaria o armazenamento de vários arquivos de *trace* das várias simulações que estavam sendo feitas e, até mesmo, inviabilizaria uma única simulação de 10.000 nós.

Na verdade, o problema em questão não era exatamente um problema, mas sim uma característica do NS-2 de gerar informação de acordo com a configuração de *traces* do *MobileNode*. Conforme a seção 3.2.1, a configuração de um *MobileNode* possibilita a geração de *traces* em 4 modalidades: *trace* de agente, *trace* de roteamento, *trace* de MAC e *trace* de movimento. Após um exame cuidadoso do *script* de configuração da rede a ser simulada, a geração da grande quantidade de informações no arquivo de *trace* se deveu à ativação dos *traces* de roteamento e MAC, principalmente este último.

Como não eram necessários os *traces* de roteamento e MAC para o estudo conduzido por este trabalho, estes foram desabilitados, proporcionando uma diminuição por aproximadamente um fator de 100 do tamanho do arquivo gerado.

As Listagens 33 e 34 mostram o uso de espaço em disco consumido por dois arquivos de *trace* (wtest.nam e wtest.tr) que, sozinhos, consumiram todo o espaço em disco disponível na partição (identificada pelo símbolo "/" na coluna "Mounted on" da Listagem 33) quando habilitados os *traces* de MAC, roteamento e agente simultaneamente.

Listagem 33. Visualização do espaço em disco das partições montadas no servidor de simulação.

```
rico@adhoc01:~$ df -h
Filesystem            Size  Used Avail  Use% Mounted on
/dev/sda1             141G  134G     0  100%  /
varrun                3.9G   40K  3.9G    1%  /var/run
varlock               3.9G     0  3.9G    0%  /var/lock
udev                  3.9G   76K  3.9G    1%  /dev
devshm                3.9G     0  3.9G    0%  /dev/shm
```

Listagem 34. Visualização da listagem de arquivos com *trace* de MAC, roteamento e agente habilitados.

```
rico@adhoc01:~$ ls -lh pfc-ricardo-adhoc01
total 133G
drwxr-xr-x 7 rico rico 4.0K 2008-05-01 23:32 graph_v6.3
-rwxr-xr-x 1 rico rico 1.3M 2008-05-01 23:32 graph_v6.3.tar.gz
drwxr-xr-x 2 rico rico 4.0K 2008-05-01 23:32 mm
drwxr-xr-x 2 rico rico 4.0K 2008-05-01 23:32 tp
-rw-r--r-- 1 rico rico  29G 2008-05-02 11:41 wtest.nam
-rwxr-xr-x 1 rico rico 4.3K 2008-05-01 23:32 wtest.tcl
-rw-r--r-- 1 rico rico 104G 2008-05-02 11:41 wtest.tr
```

5.3 Geração de Padrões de Tráfego com Fontes Duplicadas

Originalmente, o *script* de geração de padrões de tráfego cbrgen.tcl gera padrões em relação a um número de conexões entre os nós. Com isso, não é levada em consideração a transmissão de um único fluxo de tráfego por nó fonte, o que não é desejado por este trabalho, visto que cada nó possui uma capacidade de transmissão que será dividida pela metade, caso dele se originem dois fluxos.

Para resolver esse problema, o *script* cbrgen.tcl foi alterado para não utilizar um mesmo nó que já tenha sido usado como origem ou destino de um fluxo de tráfego. Uma versão completa do *script* alterado encontra-se no Apêndice E. Na Listagem 35 está presente o trecho de código adicionado para checagem de utilização dos nós como origem ou destino.

Listagem 35. Trecho de código adicionado ao *script* cbrgen.tcl para evitar uso de fontes de tráfego duplicadas.

```
        if { $node_used == 0 } {
                set node_used [list $dst $i]
                incr src_cnt
        } else {
                if { [lsearch -exact $node_used $dst] == -1 &&
                     [lsearch -exact $node_used $i] == -1 } {
                        lappend node_used $dst $i
                        incr src_cnt
                } else {
                        continue;
                }
        }
```

5.4 Incapacidade de Carregar Arquivos de Cenário Muito Grandes

A partir da simulação de 10.000 nós, ocorreu um problema para carregar o arquivo de cenário da rede. Como já foi mencionado anteriormente, o programa setdest realiza a construção de um cenário com os valores utilizados como referência pelo GOD (*General Operations Director*), descrito na Seção 3.2.3 deste trabalho, com o objetivo de evitar a realização de computações extras de caminho mais curto entre os nós em tempo de execução. No entanto, com uma rede de

10.000 nós, as informações geradas para o GOD resultaram num arquivo de cenário com cerca de 1,3GB. Com um arquivo deste tamanho, não foi possível carregá-lo via OTcl no *script* de descrição do modelo de rede a ser simulado. Todas as tentativas de carregamento resultaram em erro de falha de segmentação, provavelmente algum problema de alocação de memória do interpretador OTcl.

Posteriormente, verificou-se que esta é uma limitação do interpretador TCL, que é utilizado pelo OTcl, como pode ser visto pelos testes transcritos nas Listagens 36 e 37.

Listagem 36. Execução da simulação com arquivo de cenário de 1,3GB para 10.000 nós.

```
rico@adhoc01:~/pfc-ricardo-adhoc01$ ns wisim.tcl
num_nodes is set 10000
INITIALIZE THE LIST xListHead
Loading movement model...
Segmentation fault
```

Listagem 37. Teste de carregamento do arquivo de cenário de 1,3GB com o interpretador tclsh.

```
rico@adhoc01:~/pfc-ricardo-adhoc01$ tclsh a.tcl
Segmentation fault
rico@adhoc01:~/pfc-ricardo-adhoc01$ cat a.tcl
source "mm/movement_model.tcl"
rico@adhoc01:~/pfc-ricardo-adhoc01$
```

Para contornar o problema, foi necessário retirar as linhas referentes às informações geradas para o GOD, resultando numa redução do tamanho do arquivo para cerca de 1,5MB. O resultado disso é uma demora maior para se executar a simulação de 10.000 nós, em contrapartida, é possível executá-las desse modo.

5.5 Consumo Excessivo de Memória

Foi observado a partir da simulação de 10.000 nós que o consumo de memória aumentou consideravelmente. Enquanto simulações de 1000 nós com 1/3 destes como fontes de tráfego consumem em média 500MB de memória, as simulações de 10.000 nós com a mesma proporção de nós fonte consome cerca de 8GB.

Com isso, apesar de o servidor de simulação possuir quatro núcleos, a limitação de memória impossibilita a execução em paralelo de várias simulações de 10.000 ou mais nós, pois isso consumiria toda a memória disponível para os processos do sistema, causando pane ou indisponibilidade temporária. No Apêndice D está transcrita uma mensagem de erro completa sobre o consumo total de memória. Esta mensagem foi gerada pelo kernel do sistema operacional quando se tentou executar duas simulações de 10.000 nós em paralelo. A Listagem 38 mostra o resultado do comando *free* que exibe a quantidade de memória em uso no momento da execução de uma simulação envolvendo

10.000 nós. A Listagem 39 mostra a saída produzida pelo programa no momento em que o S.O. termina um processo da simulação de 10.000 nós em execução.

Listagem 38. Verificação de uso da memória no momento da execução de uma simulação de 10.000 nós.

```
rico@adhoc01:~$ free -m
             total       used       free     shared    buffers     cached
Mem:          7925       7830         94          0         34        507
-/+ buffers/cache:       7288        637
Swap:         6236         20       6215
```

Listagem 39. Saída da execução de uma simulação quando terminada forçadamente pelo S.O.

```
Client 7264: Handoff Attempted
Client 2668: Handoff Attempted
Client 2859: Handoff Attempted
Client 4024: Handoff Attempted
Killed

real    68m58.729s
user    61m1.070s
sys     0m51.420s
```

Capítulo 6

Conclusões e Trabalhos Futuros

Este trabalho procurou realizar uma análise de vazão e atraso na camada de rede do modelo TCP/IP de simulações de redes *ad hoc* IEEE 802.11 com o NS-2 à medida em que as redes simuladas escalonam. Em busca desse objetivo, mas com atenção às restrições de tempo para realização do trabalho, algumas decisões foram tomadas para que fosse possível conduzir experimentos relevantes e concluí-los. Algumas questões ainda estão em aberto, mas certamente o que foi aqui realizado pode servir como base para estudos futuros mais abrangentes.

6.1 Contribuições e Conclusões

O estudo do escalonamento de redes *ad hoc* é de fundamental importância para o futuro da aplicação dessas redes em ambientes diversos, tanto de necessidade de comunicação em casos de desastres, guerras, quanto a necessidade crescente de comunicação que temos com a evolução natural da tecnologia, onde equipamentos antes destinados a uma única finalidade, como aparelhos celulares para telecomunicação, serem atualmente usados para os mais diversos fins, como acesso à Internet. Para que haja uma comunicação eficiente sem a necessidade de uma infra-estrutura física, é importante que sejam feitos testes, inicialmente por meio de simulações, como os que foram realizados neste trabalho, principalmente pelo fator custo e complexidade de gerenciamento ser bem menor que a realização de testes com equipamentos reais.

A realização de simulações com grande número de nós, apesar de ser bastante prática para obtenção de resultados que se aproximam com experiências com equipamentos reais, é computacionalmente bastante custosa, mesmo com a utilização de extensões que possibilitam uma melhora de desempenho dessas simulações no NS-2. Isso pôde ser evidenciado principalmente com a simulação de 10.000 nós, a qual durou cerca de 2 dias e

meio de tempo de execução numa máquina com grande poder de processamento e muita memória. Certamente uma das contribuições relevantes deste trabalho foi a medição do tempo de execução e consumo de recursos da máquina da simulação para as diferentes quantidades de nós nas redes simuladas.

O resultado das análises feitas em relação às métricas estudadas neste trabalho mostram claramente uma degradação da eficiência da rede com o escalonamento do número de nós, o que era esperado devido à necessidade de roteamento *multi-hop* e aumento da interferência com o crescimento do número de transmissores.

Como conclusão final da análise das simulações conduzidas durante o semestre e descritas neste trabalho, ainda não foi possível afirmar se há uma tendência de convergência dos valores das métricas utilizadas como ocorreu no estudo utilizado como base para este trabalho [4]. Para que isso possa ser verificado, é preciso investir tempo para realização de simulações com uma maior densidade de nós por área simulada (considerando uma área de dimensões fixas), um maior tempo de simulação e um maior número de nós, como 100.000 e 1.000.000 de nós, caso seja possível realizar simulações de tal magnitude no NS-2. A execução de tais simulações não foi possível devido às restrições de tempo para a realização deste trabalho e à grande quantidade de tempo necessária para realização das simulações com maior densidade de nós ou com mais de 10.000 nós.

6.2 Trabalhos Futuros

A análise de métricas de desempenho de redes *ad hoc* ainda pode ser bastante aprofundada através da utilização de maior número de nós, utilização de outras configurações de rede disponíveis no NS-2 e análise de *traces* de outras camadas ou comportamento espacial dos nós.

Em relação às outras possíveis configurações de rede, a mais notável acredita-se que seja a utilização de outros modelos de propagação de ondas de rádio além do modelo *Two-Ray Ground*, o qual foi usado para realização das simulações exibidas neste trabalho. O NS-2 atualmente oferece outros modelos que podem ser interessantes para análise, que são o *Free Space* e o *Shadowing*. Assim como a utilização de outros modelos, é possível realizar uma configuração mais detalhada de cada modelo, de acordo com parâmetros como o *path loss* [4], o *carrier sense threshold* (CST) e o *receive threshold* (RT) [16].

Outras alterações de configuração diversas podem ser feitas em relação à taxa de transmissão dos nós da rede, a utilização de mecanismos de RTS/CTS e a utilização de outros protocolos de roteamento, como o DSDV, por exemplo.

Em relação à análise de *traces* de outras camadas, é possível habilitar o *trace* da camada MAC, embora seja necessário uma grande quantidade de espaço em disco (da

ordem de centenas de gigabytes) para armazenamento dessas informações, conforme descrito no Capítulo 5.

Existem ainda possibilidades de expandir os estudos realizados neste trabalho com a utilização de padrões de movimentação dos nós, ao invés da premissa aqui utilizada de que os nós da rede estão fixos durante toda a simulação. Pelos resultados dos testes com simulações iniciais (Seção 4.2.3), foi visto que a movimentação dos nós afeta as métricas de rede analisadas.

Bibliografia

[1] FRODIGH, M., JOHANSSON, P., LARSSON, P., Wireless ad hoc networking—The art of networking without a network, Ericsson Review no. 04, 2000, Disponível em: http://www.ericsson.com/ericsson/corpinfo/publications/review/2000_04/files/2000046.pdf, Acesso em: 20/02/2008.

[2] BROCH, J., MALTZ, D. A., JOHNSON, D. B., HU, Y. , JETCHEVA, J., A performance comparison of multi-hop wireless ad hoc network routing protocols, Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking, p.85-97, October 25-30, 1998, Dallas, Texas, United States.

[3] JAIN R., The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley- Interscience, New York, NY, April 1991, ISBN:0471503361.

[4] MORAES, R. M.; ARAÚJO, F. P., Modeling Interference in Wireless Ad Hoc Networks. In: IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2007, Istanbul, Turkey. Proc. of IEEE/ACM MASCOTS, 2007. p. 54-59.

[5] The Network Simulator - NS-2, Disponível em: http://www.isi.edu/nsnam/ns/, Acesso em: 23/02/2008.

[6] TclCL, Disponível em http://otcl-tclcl.sourceforge.net/tclcl/, Acesso em: 25/03/2008.

[7] Ubuntu Linux, Disponível em http://www.ubuntu.com/, Acesso em: 26/03/2008.

[8] HARTMANN, S., The World as a Process: Simulations in the Natural and Social Sciences, Disponível em: http://philsci-archive.pitt.edu/archive/00002412/01/Simulations.pdf, Acesso em 25/04/2008.

[9] IEEE 802.11, The Working Group Setting the Standards for Wireless LANs, Disponível em: http://ieee802.org/11/, Acesso em: 20/04/2008.

[10] BARKEN, L., How secure is your wireless network? Safeguarding our Wi-Fi LAN, Prentice Hall PTR, Upper Saddle River, New Jersey, 2004, ISBN:0131402064.

[11] MAHRENHOLZ, D., Providing QoS for Publish/Subscribe Communication in Dynamic Ad-Hoc Networks, Disponível em: http://diglib.uni-magdeburg.de/Dissertationen/2006/danmahrenholz.pdf, Acesso em: 17/05/2008.

[12] The Rice University Monarch Project: Mobile Networking Architectures, Disponível em: http://www.monarch.cs.rice.edu/, Acesso em: 19/03/2008.

[13] http://ee.washington.edu/research/funlab/802_11/report_80211_IM.pdf

[14] AAD, I., HUBAUX, J. P., NS-2 for the impatient, Disponível em: http://icapeople.epfl.ch/aad/teaching/ns/ns-2-for-the-impatient.pdf, Acesso em: 14/05/2008.

[15] GREIS, M., Marc Greis' Tutorial for the UCB/LBNL/VINT Network Simulator "ns", Disponível em: http://www.isi.edu/nsnam/ns/tutorial/, Acesso em: 02/03/2008.

[16] NAUMOV, V., GROSS, T., Simulation of Large Ad Hoc Networks, MSWiM'03, September 19, 2003, San Diego, California, USA. Copyright 2003 ACM 1-58113-766-4/03/0009.

[17] DAS, S., PERKINS, C., ROYER, E. Performance comparison of two on-demand routing protocols for ad hoc networks. In INFOCOM'2000 (1), pages 3–12, 2000.

[18] WALSH, K., SIRER, E. G., Staged Simulation: A General Technique for Improving Simulation Scale and Performance. In *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, Special Issue on Scalable Network Modeling and Simulation, April 2004.

[19] SNS Download, Disponível em: http://www.cs.cornell.edu/People/egs/sns/snstarfile.html, Acesso em: 18/03/2008.

[20] NS-2 Change Log, Disponível em: http://www.isi.edu/nsnam/ns/CHANGES.html, Acesso em: 10/04/2008.

[21] ROBINSON, J., Making NS-2 simulate an 802.11b link, Disponível em: http://www.ece.rice.edu/~jpr/ns/docs/ns-802_11b.html, Acesso em: 14/03/2008.

[22] NS-2 for the impatient – Tools, Disponível em: http://icapeople.epfl.ch/aad/teaching/ns/tools.tgz, Acesso em: 05/05/2008.

[23] The Perl Directory, Disponível em: http://www.perl.org/, Acesso em: 15/05/2008.

[24] KUROSE, J. ROSS, K. W., Redes de Computadores e a Internet: uma abordagem *top-down*, 3ª edição, São Paulo, Pearson Addison Wesley, 2006.

[25] COUTINHO, M. M., NS-2 Guia Básico para Iniciantes, Disponível em: http://www.cci.unama.br/margalho/simulacao/livroNS.pdf, Acesso em: 10/05/2008.

[26] Informações Gerais sobre a Regulamentação do Serviço de Comunicação Multimídia (SCM) do Brasil, Disponível em: http://www.teleco.com.br/scm.asp, Acesso em: 20/05/2008.

[27] JTrana: A Java-based NS2 Wireless Trace Analyzer, Disponível em: http://ns2trana.googlepages.com/, Acesso em: 25/05/2008.

[28] time(1) – Linux man page, Disponível em: http://linux.die.net/man/1/time, Acesso em: 20/05/2008.

[29] top(1) – Linux man page, Disponível em: http://linux.die.net/man/1/top, Acesso em: 20/05/2008.

[30] New Oxford American Dictionary, 2ª Edição, USA, Oxford University Press, 2005.

[31] Gnuplot homepage, Disponível em: http://www.gnuplot.info/, Acesso em: 16/03/2008.

Apêndice A

Script para o cálculo da vazão de múltiplos nós sem fio

```
#!/usr/bin/perl
#
# This script makes a calculation on the throughput for each agent traffic independently
#
# Author: Ricardo J. Ulisses Filho
# Universidade de Pernambuco - Recife - Brazil
# Departamento de Sistemas e Computacao
#
use strict;
$| = 1;

# how big is our simulation time range?
my $simulation_time = $ARGV[0] || &usage;

# global variables
my $DEBUG  = 0;
my @events = ('s', 'r', 'f', 'd');
my %events = ('s' => 'start', 'r' => 'end');
my %flow_tp;
my %flow_file;

my $count = 0;

sub print_debug($);

# parsing and flow timing accounting
while (my $line = <STDIN>) {
        $count++;

        if ($line =~ /AGT/) {
                if ($line =~ /^r\s+-t\s+(\S+)/) {
                        my $time = $1;
                        my $time_int = int $time;

                        if ($line =~ /-Is\s+(\S+)\s+-Id\s+(\S+)\s+-It\s+cbr\s+-Il\s+(\d+)/) {
                                my $flow_id = "$1:$2"; # our flow id is made by both sender and
destination ip.port
                                my $bytes_rcvd = $3;
                                my $bits_rcvd = $bytes_rcvd * 8; # converting bytes to bits

                                if (!$flow_file{$flow_id}) {
                                        for (my $i = $simulation_time; $i >= 0; $i--) {
                                                $flow_tp{$flow_id}{$i} = 0; # initialize flow_id
throughput values
                                        }
                                        $flow_file{$flow_id} = "/tmp/flow_${flow_id}";
```

```
                                }

                                print_debug "flow_id => [$flow_id] time_int => [$time_int]\n";
                                $flow_tp{$flow_id}{$time_int} += $bits_rcvd;
                        }
                }
        } else {
                print_debug "skipping meaningless line: $count\n";
        }
}

# generating plot files
print_debug "generating plot files...\n";
for my $flow_id (keys %flow_file) {
        print_debug "generating plot for flow_id = [$flow_id]\n";
        open(PLOTFILE, ">$flow_file{$flow_id}") or die("could not open $flow_file{$flow_id}: $!");
        for (my $i = 0; $i <= $simulation_time; $i++) {
                print PLOTFILE "$i $flow_tp{$flow_id}{$i}\n";
        }
        close(PLOTFILE) or die("could not write file $flow_file{$flow_id}: $!");
}

sub usage {
        print "cat file.tr | ./tp_calc.pl simulation_time_in_seconds\n";
        exit 1;
}

sub print_debug($) {
        if ($DEBUG) {
                print @_;
        }
}
```

Apêndice B

Script para o cálculo da vazão média de múltiplos nós sem fio

```
#!/usr/bin/perl
#
# This script makes calculations on the average throughput for NS-2 new trace # file format
#
# Author: Ricardo J. Ulisses Filho
# Universidade de Pernambuco - Recife - Brazil
# Departamento de Sistemas e Computacao
#
use strict;
$| = 1;

# how big is our simulation time range?
my $simulation_time = $ARGV[0] || &usage;
my $agent_sources   = $ARGV[1] || &usage;

# print "$simulation_time\n";
# exit;

# global variables
my $DEBUG  = 0;
my @flow_tp;
my @flow_conn;
my $flow_file = "/tmp/flow_avg_${agent_sources}";

my $count = 0;
my $time_int_prev = 0;
my @traffic_sources = ();

my $simulation_start;
my $tp_average_sum = 0;

sub print_debug($);

# initialize flow_tp and traffic_sources for each simulation second
for (my $i = 0; $i <= $simulation_time; $i++) {
        $flow_tp[$i] = 0; # how many bits we have per time period (second)
        $flow_conn[$i] = 0; # how many traffic sources we have per time period
}

# parsing and flow throughput accounting
while (my $line = <STDIN>) {
        $count++;

        # considers only agent trace
        if ($line =~ /AGT/) {
                if ($line =~ /^r\s+-t\s+(\S+)/) {
```

```
my $time = $1;
                        my $time_int = int $time;

                        if ($time_int != $time_int_prev) { # verifies if we are in a new time
period of simulation time
                                @traffic_sources = (); # clear array of traffic sources per time
period
                        }
                        $time_int_prev = $time_int;

                        if ($line =~ /-Is\s+(\S+)\s+-Id\s+(\S+)\s+-It\s+cbr\s+-Il\s+(\d+)/) {
                                my $flow_id = "$1:$2"; # our flow id is made by both sender and
destination ip.port
                                my $bytes_rcvd = $3;
                                my $bits_rcvd = $bytes_rcvd * 8; # converting bytes to bits

                                if ( scalar(grep(/^$flow_id$/, @traffic_sources)) == 0 ) {
                                        push @traffic_sources, $flow_id;
                                        $flow_conn[$time_int]++;
                                }

                                print_debug "flow_id => [$flow_id] time_int => [$time_int]
flow_conn => [$flow_conn[$time_int]]\n";
                                $flow_tp[$time_int] += $bits_rcvd;
                        }
                }
        } else {
                print_debug "skipping meaningless line: $count\n";
        }
}

# generating plot file
print_debug "generating plot file...\n";
open(PLOTFILE, ">$flow_file") or die("could not open $flow_file: $!");

for (my $i = 0; $i <= $simulation_time; $i++) {

        if ($flow_conn[$i] == 0) {
                print PLOTFILE "$i 0\n";
                next;
        }

        if (!defined($simulation_start)) {
                $simulation_start = $i;
        }

        my $flow_avg_tmp = $flow_tp[$i] / $flow_conn[$i];
        $flow_avg_tmp = sprintf("%.1f", $flow_avg_tmp); # consider only one decimal point

        print_debug "flow_tp[$i] => $flow_tp[$i] => flow_conn[$i] => $flow_conn[$i] flow_avg_tmp
=> $flow_avg_tmp \n";

        print PLOTFILE "$i " . $flow_avg_tmp . "\n";
        $tp_average_sum += $flow_avg_tmp;
}
close(PLOTFILE) or die("could not write file $flow_file: $!");

my $simulation_range = $simulation_time - $simulation_start;
my $tp_avg_total = $tp_average_sum / $simulation_range;
$tp_avg_total = sprintf("%.1f", $tp_avg_total);

print_debug "result: average throughput in time equals to " . $tp_avg_total . "bits/s\n";
print "$tp_avg_total\n";

exit 0;

# Subroutines

sub usage {
        print "cat file.tr | ./tp_calc_avg-ng.pl simulation_time_in_seconds agent_sources\n";
        exit 1;
}

sub print_debug($) {
if ($DEBUG) {
```

```
                print @_;
        }
}
```

Apêndice C

Script para o cálculo do atraso médio de múltiplos nós sem fio

```
#!/usr/bin/perl
#
# Script to calculate the average end-to-end delay for NS-2 new trace file format with multiple
# transmitting sources
#
# Author: Ricardo J. Ulisses Filho
# Universidade de Pernambuco - Recife - Brazil
# Departamento de Sistemas e Computacao
#
use strict;
$| = 1;

# required parameters
my $simulation_time = $ARGV[0] || &usage;
my $agent_sources   = $ARGV[1] || &usage;

# global variables
my $DEBUG  = 0;
my $flow_file = "/tmp/flow_avg_${agent_sources}";
my $flow_delay_time;
my @flow_conn;
my %flow_delay;
my %flow_time;
my %flow_delay_count;

my $count = 0;
my $time_int_prev = 0;
my @traffic_sources = ();

my $simulation_start;
my $delay_average_sum = 0;

sub print_debug($);

# initialize flow_tp and traffic_sources for each simulation second
for (my $i = 0; $i <= $simulation_time; $i++) {
        $flow_conn[$i]  = 0; # how many traffic sources we have per time period
}

# parsing and delay flow accounting
while (my $line = <STDIN>) {
        $count++;

        # considers only agent trace
        if ($line =~ /AGT/) {
                if ($line =~ /^(s|r)\s+-t\s+(\S+)/) {
```

```
                        my $event = $1;
my $time  = $2;
                        my $time_int = int $time;

                        if ($time_int != $time_int_prev) { # verifies if we are in a new time
period of simulation time
                                @traffic_sources = (); # clear array of traffic sources per time
period
                        }
                        $time_int_prev = $time_int;

                        if ($line =~ /-Is\s+(\S+)\s+-Id\s+(\S+).*?-Ii\s+(\S+)/) {
                                my $flow_id = "$1:$2"; # our flow id is made by both sender and
destination ip.port
                                my $unique_id = $3;

                                $flow_time{$flow_id}{$unique_id}{$event} = $time * 1000; # let's
give results in millisecond

                                # when the packet is being received the actual delay calculation
occurs
                                if ($event eq 'r') {
                                        if ( scalar(grep(/^$flow_id$/, @traffic_sources)) == 0) {
                                                push @traffic_sources, $flow_id;
                                                $flow_conn[$time_int]++;
                                        }

                                        print_debug "flow_delay_time =
$flow_time{$flow_id}{$unique_id}{'r'} - $flow_time{$flow_id}{$unique_id}{'s'};\n";
                                        $flow_delay_time = $flow_time{$flow_id}{$unique_id}{'r'} -
$flow_time{$flow_id}{$unique_id}{'s'};
                                        $flow_delay_time = sprintf("%.1f", $flow_delay_time);

                                        $flow_delay{$time_int}{$flow_id} += $flow_delay_time;

                                        # how many packets were delivered within $time_int?
                                        if (!$flow_delay_count{$time_int}{$flow_id}) {
                                                $flow_delay_count{$time_int}{$flow_id} = 1;
                                        } else {
                                                $flow_delay_count{$time_int}{$flow_id} =
$flow_delay_count{$time_int}{$flow_id} + 1;
                                        }
                                        print_debug "flow_delay_count{$time_int}{$flow_id} =>
$flow_delay_count{$time_int}{$flow_id}\n";
                                        print_debug "flow_delay_time => [$flow_delay_time]\n";
                                        print_debug "flow_id => [$flow_id] time_int => [$time_int]
flow_conn => [$flow_conn[$time_int]]\n";
                                }
                        }
                }
        } else {
                print_debug "skipping meaningless line: $count\n";
        }
}

# generating plot file
print_debug "generating plot file...\n";
open(PLOTFILE, ">$flow_file") or die("could not open $flow_file: $!");

for (my $i = 0; $i <= $simulation_time; $i++) {

        if ($flow_conn[$i] == 0) {
                print PLOTFILE "$i " . 0 . "\n";
                next;
        }

        if (!defined($simulation_start)) {
                $simulation_start = $i;
        }

        my %delay_sum_per_source;
        # average on the number of packets per source (local avg)
        for my $flow_id (keys %{ $flow_delay{$i} }) {
                $delay_sum_per_source{$flow_id} = $flow_delay{$i}{$flow_id} /
$flow_delay_count{$i}{$flow_id};
```

```
        }

        # average on the value of avg_delay per number of sources in time (global avg)
        my $delay_sum_per_sec = 0;
        for my $flow_id (keys %{ $flow_delay{$i} }) {
                $delay_sum_per_sec += $delay_sum_per_source{$flow_id};
        }
        $delay_sum_per_sec /= $flow_conn[$i]; # divided by the number of sources
        $delay_sum_per_sec  = sprintf("%.1f", $delay_sum_per_sec);

        print PLOTFILE "$i " . $delay_sum_per_sec . "\n";

        $delay_average_sum  += $delay_sum_per_sec;
}
close(PLOTFILE) or die("could not write file $flow_file: $!");

my $simulation_range = $simulation_time - $simulation_start;
my $delay_avg_total = $delay_average_sum / $simulation_time;
$delay_avg_total = sprintf("%.1f", $delay_avg_total);

print_debug "result: average delay in time equals to " . $delay_avg_total . "ms\n";
print "$delay_avg_total\n";

exit 0;

# Subroutines

sub usage {
print "cat file.tr | ./delay_calc_avg-ng.pl simulation_time_in_seconds agent_sources\n";
        exit 1;
}

sub print_debug($) {
        if ($DEBUG) {
                print @_;
        }
}
```

Apêndice D

Saída de erro gerada pelo S.O. por sobrecarga de uso de memória pela simulação de 10.000 nós

```
[290718.279917] Out of memory: kill process 12647 (bash) score 608605 or a child
[290718.279950] Killed process 16030 (ns)
[290718.326016] screen invoked oom-killer: gfp_mask=0x201d2, order=0, oomkilladj=0
[290718.326018]
[290718.326018] Call Trace:
[290718.326037]  [<ffffffff80271c56>] out_of_memory+0x2b6/0x330
[290718.326041]  [<ffffffff80273e7c>] __alloc_pages+0x2ec/0x340
[290718.326046]  [<ffffffff80275977>] __do_page_cache_readahead+0x107/0x2a0
[290718.326052]  [<ffffffff804365e8>] io_schedule+0x28/0x40
[290718.326056]  [<ffffffff804368f5>] __wait_on_bit_lock+0x65/0x80
[290718.326060]  [<ffffffff8026e54f>] __lock_page+0x5f/0x70
[290718.326063]  [<ffffffff80270d98>] filemap_nopage+0x238/0x2f0
[290718.326068]  [<ffffffff8027d024>] __handle_mm_fault+0x1c4/0xb60
[290718.326072]  [<ffffffff80242baf>] get_signal_to_deliver+0x8f/0x450
[290718.326076]  [<ffffffff8024027e>] recalc_sigpending+0xe/0x30
[290718.326079]  [<ffffffff8020943e>] do_notify_resume+0x1ce/0x7f0
[290718.326083]  [<ffffffff80439f5c>] do_page_fault+0x1fc/0x870
[290718.326086]  [<ffffffff80241905>] group_send_sig_info+0x25/0xa0
[290718.326089]  [<ffffffff802419d1>] kill_pid_info+0x51/0x90
[290718.326093]  [<ffffffff802426f4>] sys_kill+0x84/0x180
[290718.326098]  [<ffffffff804382ed>] error_exit+0x0/0x84
```

Apêndice E

Script de configuração das redes de interesse do trabalho

```
#############################################################
# wireless simulation related parameters                    #
#############################################################
set val(chan)           Channel/WirelessChannel    ;# Channel Type
set val(prop)           Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)          Phy/WirelessPhy            ;# network interface type
set val(mac)            Mac/802_11                 ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)             LL                         ;# link layer type
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                         ;# max packet in ifq
set val(nn)             10                         ;# number of mobilenodes
set val(rp)             AODV                       ;# routing protocol
set val(x)              500
set val(y)              100
set val(stop)           300
set val(mm)             "sc/scen-500x100-10-300-20-300.tcl"
set val(tp)             "tp/traffic-cbr-10-1-3-4.tcl"

set tracefile    [lindex $argv 0]
set tracefilenam [lindex $argv 1]

Mac/802_11 set RTSThreshold_ 3000
Mac/802_11 set dataRate_ 11Mb

#############################################################
# Initialization                                            #
# 1. create simulator                                       #
# 2. tracing                                                #
# 3. define topography                                      #
#############################################################
set ns_        [new Simulator]

$ns_ use-newtrace
set  tracefd    [open $tracefile w]
$ns_ trace-all  $tracefd

set  namtracefd [open $tracefilenam w]
$ns_ namtrace-all-wireless $namtracefd $val(x) $val(y)

set topo        [new Topography]
$topo load_flatgrid $val(x) $val(y)

#
# Use variable object $god_
#
set god_ [create-god $val(nn)]
```

```
#################################################################
# Define/create/initialize nodes                                #
# 1. define nodes                                               #
# 2. create nodes                                               #
# 3. disable random motion                                      #
# 4. coordinates of wilress nodes                               #
# 5. nam setting, size and position                             #
#################################################################
$ns_ node-config -adhocRouting $val(rp) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -topoInstance $topo \
                 -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace OFF \
                 -movementTrace OFF \
                 -channel [new $val(chan)]

#################################################################
# Scenario                                                      #
#################################################################

#
# Instantiate nodes
#
for {set i 0} {$i < $val(nn)} {incr i} {
        set node_($i) [$ns_ node]
       $node_($i) random-motion 0
}

#
# Movement model
#
puts "Loading movement model..."
source $val(mm)

#
# Define initial node position in NAM
#
for {set i 0} {$i < $val(nn)} {incr i} {
       # This function must be called after mobility model is defined
        $ns_ initial_node_pos $node_($i) 10
}

#
# Traffic pattern
#
puts "Loading traffic pattern..."
source $val(tp)

#
# Print some variables values of this simulation
#
puts "CBR packet size = [$cbr_(0) set packetSize_]"
puts "CBR interval = [$cbr_(0) set interval_]"
puts "CBR random = [$cbr_(0) set random_]"
puts "Starting simulation..."

#
# End of simulation
#
$ns_ at $val(stop).000000000000000 "stop"
$ns_ at $val(stop).000000000000001 "puts \"NS EXITING...\" ; $ns_ halt"

#################################################################
# 'stop' procedure                                              #
#################################################################
proc stop {} {
    global ns_ tracefd namtracefd
```

```
    $ns_ flush-trace
    close $tracefd
    close $namtracefd
}

#####################################################################
# start the simulation                                              #
#####################################################################
$ns_ run
```

Apêndice F

Versão modificada do *script* de geração de padrões de tráfego cbrgen.tcl

```
#
#  Copyright (c) 1999 by the University of Southern California
#  All rights reserved.
#
#  This program is free software; you can redistribute it and/or
#  modify it under the terms of the GNU General Public License,
#  version 2, as published by the Free Software Foundation.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License along
#  with this program; if not, write to the Free Software Foundation, Inc.,
#  59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.
#
#  The copyright of this module includes the following
#  linking-with-specific-other-licenses addition:
#
#  In addition, as a special exception, the copyright holders of
#  this module give you permission to combine (via static or
#  dynamic linking) this module with free software programs or
#  libraries that are released under the GNU LGPL and with code
#  included in the standard release of ns-2 under the Apache 2.0
#  license or under otherwise-compatible licenses with advertising
#  requirements (or modified versions of such code, with unchanged
#  license).  You may copy and distribute such a system following the
#  terms of the GNU GPL for this module and the licenses of the
#  other code concerned, provided that you include the source code of
#  that other code when and as the GNU GPL requires distribution of
#  source code.
#
#  Note that people who make modified versions of this module
#  are not obligated to grant this special exception for their
#  modified versions; it is their choice whether to do so.  The GNU
#  General Public License gives permission to release a modified
#  version without this exception; this exception also makes it
#  possible to release a modified version which carries forward this
#  exception.

# Traffic source generator from CMU's mobile code.
```

```
#
# $Header: /cvsroot/nsnam/ns-2/indep-utils/cmu-scen-gen/cbrgen.tcl,v 1.4 2005/09/16 03:05:39 tomh
# Exp $

# ======================================================================
# Default Script Options
# ======================================================================
set opt(nn)             0               ;# Number of Nodes
set opt(seed)           0.0
set opt(mc)             0
set opt(pktsize)        64

set opt(rate)           0
set opt(interval)       0.0             ;# inverse of rate
set opt(type)             ""

# ======================================================================

proc usage {} {
    global argv0

    puts "\nusage: $argv0 \[-type cbr|tcp\] \[-nn nodes\] \[-seed seed\] \[-mc connections\] \[-
rate rate\]\n"
}

proc getopt {argc argv} {
        global opt
        lappend optlist nn seed mc rate type

        for {set i 0} {$i < $argc} {incr i} {
                set arg [lindex $argv $i]
                if {[string range $arg 0 0] != "-"} continue

                set name [string range $arg 1 end]
                set opt($name) [lindex $argv [expr $i+1]]
                #puts "debug --- opt($name) = $opt($name)"
        }
}

proc create-cbr-connection { src dst } {
        global rng cbr_cnt opt

        set stime [$rng uniform 0.0 10.0]

        puts "#\n# $src connecting to $dst at time $stime\n#"

        puts "set udp_($cbr_cnt) \[new Agent/UDP\]"
        puts "\$ns_ attach-agent \$node_($src) \$udp_($cbr_cnt)"
        puts "set null_($cbr_cnt) \[new Agent/Null\]"
        puts "\$ns_ attach-agent \$node_($dst) \$null_($cbr_cnt)"
        puts "set cbr_($cbr_cnt) \[new Application/Traffic/CBR\]"
        puts "\$cbr_($cbr_cnt) set packetSize_ $opt(pktsize)"
        puts "\$cbr_($cbr_cnt) set interval_ $opt(interval)"
        puts "\$cbr_($cbr_cnt) set random_ 1"
        #puts "\$cbr_($cbr_cnt) set maxpkts_ 10000"
        puts "\$cbr_($cbr_cnt) attach-agent \$udp_($cbr_cnt)"
        puts "\$ns_ connect \$udp_($cbr_cnt) \$null_($cbr_cnt)"

        puts "\$ns_ at $stime \"\$cbr_($cbr_cnt) start\""

        incr cbr_cnt
}

proc create-tcp-connection { src dst } {
        global rng cbr_cnt opt

        set stime [$rng uniform 0.0 180.0]

        puts "#\n# $src connecting to $dst at time $stime\n#"

        puts "set tcp_($cbr_cnt) \[\$ns_ create-connection \
                TCP \$node_($src) TCPSink \$node_($dst) 0\]";
        puts "\$tcp_($cbr_cnt) set window_ 32"
        puts "\$tcp_($cbr_cnt) set packetSize_ $opt(pktsize)"
```

```
        puts "set ftp_($cbr_cnt) \[\$tcp_($cbr_cnt) attach-source FTP\]"


        puts "\$ns_ at $stime \"\$ftp_($cbr_cnt) start\""

        incr cbr_cnt
}

# ======================================================================

getopt $argc $argv

if { $opt(type) == "" } {
    usage
    exit
} elseif { $opt(type) == "cbr" } {
    if { $opt(nn) == 0 || $opt(seed) == 0.0 || $opt(mc) == 0 || $opt(rate) == 0 } {
        usage
        exit
    }

    set opt(interval) [expr 1 / $opt(rate)]
    if { $opt(interval) <= 0.0 } {
        puts "\ninvalid sending rate $opt(rate)\n"
        exit
    }
}

puts "#\n# nodes: $opt(nn), max sources: $opt(mc), send rate: $opt(interval), seed: $opt(seed)\n#"

set rng [new RNG]
$rng seed $opt(seed)

set u [new RandomVariable/Uniform]
$u set min_ 0
$u set max_ 100
$u use-rng $rng

set cbr_cnt 0
set src_cnt 0

set node_used 0

for {set i 0} {$i < $opt(nn)} {incr i} {

        set x [$u value]

        if {$x < 50} {continue;}

        set dst [expr ($i+1) % [expr $opt(nn) + 1] ]

        if { $i == $dst } { continue; }

        if { $node_used == 0 } {
                set node_used [list $dst $i]
                incr src_cnt
        } else {

                if { [lsearch -exact $node_used $dst] == -1 && [lsearch -exact $node_used $i] == -1
} {
                        lappend node_used $dst $i
                        incr src_cnt
                } else {
                        continue;
                }

        }

        if { $opt(type) == "cbr" } {
                create-cbr-connection $i $dst
        } else {
                create-tcp-connection $i $dst
        }

        if { $src_cnt == $opt(mc) } {
```

```
                break
        }

        if {$x < 75} {continue;}

        set dst [expr ($i+2) % [expr $opt(nn) + 1] ]

        if { $i == $dst } { continue; }

        if { [lsearch -exact $node_used $dst] == -1 && [lsearch -exact $node_used $i] == -1 } {
                lappend dst_used $dst $i
                incr src_cnt

        } else {
                continue;
        }

        if { $opt(type) == "cbr" } {
                create-cbr-connection $i $dst
        } else {
                create-tcp-connection $i $dst
        }

        if { $src_cnt == $opt(mc) } {
                break
        }
}

puts "#\n#Total sources/connections: $src_cnt/$cbr_cnt\n#"
```