Universidade Federal do Amazonas
Instituto de Computação
Programa de Pós-Graduação em Informática

DANIEL DA COSTA BITTENCOURT

Avaliação de desempenho de redes tolerantes a atrasos e desconexões utilizando a técnica de virtualização

Manaus
2013

Daniel da Costa Bittencourt

Avaliação de desempenho de redes tolerantes a atrasos e desconexões utilizando a técnica de virtualização

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas, como parte dos requisitos necessários para a obtenção do título de Mestre em Informática.

Orientador: Prof. Dr.-Ing. Edjair de Souza Mota

Manaus

2013

Daniel da Costa Bittencourt

Avaliação de desempenho de redes tolerantes a atrasos e desconexões utilizando a técnica de virtualização

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas, como parte dos requisitos necessários para a obtenção do título de Mestre em Informática.

Banca Examinadora

Prof. Dr.-Ing. Edjair de Souza Mota (Orientador)
Universidade Federal do Amazonas

Prof. Dr. Leandro Silva Galvão de Carvalho
Universidade Federal do Amazonas

Prof. Dr. Edson Nascimento Silva Junior
Universidade Federal do Amazonas

Prof. Dr. Eduardo Coelho Cerqueira
Universidade Federal do Pará

Manaus – 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Bittencourt, Daniel da Costa

Avaliação de desempenho de redes tolerantes a atrasos e desconexões utilizando a técnica de virtualização / Daniel da Costa Bittencourt. – Manaus: PPGI da UFAM, 2013.

111 f.: il.

Dissertação (mestrado) – Universidade Federal do Amazonas. Instituto de Computação, Manaus, BR–AM, 2013. Orientador: Edjair de Souza Mota.

1. DTN. 2. Virtualização. 3. Avaliação de desempenho. I. Mota, Edjair de Souza. II. Título.

A Deus, por ter me dado a vida e me encher de bênçãos.

À minha família, força motriz das minhas vitórias.

Agradecimentos

Em primeiro lugar a Deus, por ter me dado a dádiva da vida e por me conceder, a cada novo dia, forças para enfrentar os desafios.

Aos meus queridos pais, Francisco Fenando Bessa Bittencourt e Maria do Perpetuo Socorro da Costa Bittencourt, por acreditarem em mim, seu filho, e prover conselhos, carinho e apoio incondicional nos momentos mais difíceis. À minha irmã Daniela da Costa Bittencourt pelo incentivo e apoio nas mais diversas tarefas e em todas as horas que precisei.

Ao meu orientador Edjair Mota, pela oportunidade, pela orientação e pelo incentivo, fazendo com que este trabalho pudesse ser realizado.

Aos colegas do GRCM e DCC, pelo convívio e amizade. Em especial à Polianny Almeida, Anderson Rodrigues, Diogo Soares, Bruno Campos e João Batista Pinto Neto pelas ajudas das mais diversas formas, apoio e amizade.

Ao Instituto Nokia de Tecnologia pela cessão dos equipamentos utilizados neste trabalho e pelo apoio durante a realização deste.

Aos amigos Sergio Correa e Willer Davis pelo auxílio em diversos problemas de programação e automatização enfrentados durante este trabalho.

À Secretaria do IComp, pelo apoio administrativo e ao CNPQ, pelo apoio financeiro.

Finalmente, a todos aqueles que tiveram contribuição direta ou indireta para a realização deste trabalho.

Se vi mais longe foi por estar de pé sobre ombros de gigantes.

Isaac Newton

Sumário

Lista de Abreviaturas e Siglas

ADU	Application Data Unit
DOME	Diverse Outdoor Mobile Testbed
DTN	Delay Tolerant Network
EID	Endpoint Identifier
GPX	GPS Exchange Format
JVM	Java Virtual Machine
KVM	Kernel Virtual Machine
NUMA	Non-Uniform Memory Access
ONE	Oportunistic Network Environment
PDU	Protocol Data Units
QEMU	Quick Emulator
SMP	Symmetric Multi-Processor
VDT	Virtual DTN Testbed
VMM	Virtual Machine Monitor

Lista de Figuras

Lista de Tabelas

Resumo

Novos protocolos e tecnologias de redes devem estar embasados em avaliações de desempenho para que seu comportamento e eficácia sejam verificados e comprovados.

Em redes DTN (Delay Tolerant Networks), há uma carência de ferramentas por meio das quais essas avaliações de desempenho são realizadas, recorrendo-se principalmente a soluções de redes reais em pequena escala ou à utilização dos poucos simuladores DTN existentes.

Este trabalho apresenta uma abordagem diferenciada para a realização de avaliações de desempenho em redes DTN que se baseia em um movimento recente no meio científico de utilização de ambientes híbridos, valendo-se das vantagens e do crescimento das tecnologias de virtualização.

Propomos e implementamos um ambente para avaliação de desempenho em DTN baseado em virtualização e totalmente implementado em software, que leva em consideração a mobilidade dos nós e a replicação das caracerísticas estocásticas de redes sem-fio entre os mesmos.

Como resultados obtemos a correta reprodução da ocorrência de contatos observada em uma rede real, a identificação de novos contatos baseados nos *traces* utilizados e a estimativa do erro entre os dois ambientes de teste.

Palavras-Chave: DTN, virtualização, avaliação de desempenho.

Abstract

New protocols and network technologies must be substantiated in performance evaluations so it behavior and efficiency should be veryfied and proven. In Delay Tolerant Netorks - DTN there is a lack of tools where these performance evaluations can be developed. Such lack makes researchers often being using small scale real networks or one of the few DTN simulators available.

This work presents a diferentiated approach on conducing performance evaluations in DNT that is based on a recent movement of the scientific community towards the usage of hybrid enviroments, based on the benefits and on the growth of virtualization techonologies.

We propose and implemente an enviroment for performance evaluation of DTN, based on virtualization and completely implemented in software, that takes into account the node's mobility and the replication of the sthocastic characteristics of wireless networks between them.

As results we obtain a correct reprodution of contats that were observed in a real netwok, the identification of new ones from thre data traces used and the error estmation between the two testbeds.

Keywords: DTN, virtualization, performance evaluation

Capítulo 1

Introdução

1.1 Contextualização

Novas propostas de protocolos ou técnicas de comunicação em redes precisam ser avaliadas através de estudos de desempenho para que sua eficácia seja comprovada. Tradicionalmente utiliza-se as seguintes ferramentas para avaliação de desempenho de redes de computadores: ambientes de testes reais, simulações, análise matemática e, mais recentemente, emulações de rede [1].

Cada uma dessas ferramentas oferece diferentes níveis de possibilidades de abstração das características presentes nas redes a serem estudadas, o que torna os resultados obtidos mais fáceis ou difíceis de serem transportados para a realidade, de acordo com a quantidade de parâmetros de entrada considerados durante a análise de desempenho [2].

A escolha da ferramenta utilizada deve levar em consideração, entre outros fatores, o estágio em que esta pode ser aplicado à pesquisa, o tempo requerido para obtenção dos resultados e o poder de convencimento dos resultados obtidos [3].

Por investigar uma tecnologia nova nova, pesquisadores em DTN (*Delay Tolerant networks*) têm baseado seus estudos principalmente em simulação já que os

custos de criação de cenários reais deste tipo de rede são muitas vezes proibitivos, principalmente quando deseja-se realizar estudos com centenas de nós.

As opções de simuladores DTN ainda são escassas, sendo o ONE (*Opportunistic Network Environment simulator*) [4] a principal referência utilizada até o momento ja que implementa os protocolos de aplicação e roteamento DTN.

Apesar das vantagens de simuladores, ao usar o ONE e alguns outros simuladores semelhantes para redes DTN é comum que as características intrínsecas e aleatórias de canais sem fio sejam abstraídas das simulações já que a conectividade dos nós da rede simulada é definida apenas em relação a parâmetros estáticos.

Em simuladores que levam em consideração estas características e portanto, descrevem o canal sem fio de forma mais realista, ainda persistem problemas como: a diferença de implementações entre os protocolos implementados no simulador e os implementados em uma rede real; a falta de implementação de toda a pilha de protocolos; e a dificuldade de comparação de resultados obtidos entre os dois.

Para solução desses problemas, tem sido cogitado o uso de ambientes de análise híbridos, nos quais são combinados componentes de rede reais e/ou emulados e componentes virtuais, de forma a adicionar um nível mais elevado de realismo ao mesmo tempo em que se mantém a facilidade da repetitividade controlada de testes e modificação das características da rede. Desta forma, os resultados obtidos nesses ambientes podem ser facilmente replicados e/ou transferidos para a cenários reais [5].

Neste trabalho, nós investigamos a praticabilidade da realização de avaliações de desempenho através uso de virtualização e emulação de canal via software como uma alternativa para a realização de estudos em redes DTN, levando-se em consideração: a utilização de sistemas operacionais completos tal qual os utilizados em ambientes reais; a utilização de mobilidade proveniente de diferentes modelos ou de traces coletados em experimentos reais; e a aplicação das características aleatórias

intrínsecas de canais sem fio.

1.2 Motivação

Apesar de não ser nova, apenas recentemente a virtualização de computadores tem se tornado extremamente popular em diversos meios, indo desde grandes *data centers*, até computadores pessoais que são capazes de rodar sistemas operacionais completos ou fornecer um ambiente seguro e isolado para aplicações que oferecem riscos ou para compatibilidade de aplicações antigas [6].

No meio científico, a virtualização vem ganhando interesse por parte dos pesquisadores em redes como uma maneira de possuir múltiplas redes, cada uma modificada para um determinado propósito sobre um ambiente compartilhado de forma a possibilitar a execução de múltiplos experimentos.

Ambientes desse tipo como o GENI [7], VINI [8] e Emulab [9] concentram seu foco no design e na implementação de ambientes de testes que possibilitam estudos diversificados em redes e sistemas distribuídos.

A virtualização de redes DTN já vem sendo experimentada por alguns projetos, com diferentes propósitos, que vão desde a implementação de forma detalhada e precisa do meio sem fio usado *switches* e atenuadores de rádio frequência, onde a movimentação de nós ;e modelada através da alteração dos parâmetros de atenuação entre nós virtuais [10], até a implementação da mobilidade através da modificação de parâmetros determinísticos que bloqueiam ou permitem a passagem de dados [5] em uma rede definida por software.

Ambas as implementações são validas do ponto de vista do comportamento de redes DTN, porém, existem outros fatores que devem ser considerados durante a realização de estudos em DTN como: a criação de agrupamentos de nós pela visita a alguns locais de forma mais frequente do que a outros [11], o que pode facil-

mente extrapolar a quantidade máxima de conexões existentes em um switch de rádio-frequência; as variações existentes em canais sem fio devido ao *path loss* e a interferência, o que confere uma característica estocástica aos dados obtidos em experimentos que utilizam redes sem fio.

Dentro deste cenário, a busca por uma alternativa que una as qualidades dos métodos de virtualização de ambientes de testes em DTN existentes e que melhore os pontos deficientes apresentados, motivou a pesquisa deste trabalho.

1.3 Descrição do problema

No estudo de redes sem fio, algumas características fundamentais precisam ser levadas em consideração para que haja fidelidade entre os dados obtidos e o que acontece na vida real, para que estes possam ser então convertidos em aplicações práticas.

Redes reais levam em consideração estas características, mas os resutados obtidos são mais difícieis de serem produzidos por outros pesquisadores, e a escala na qual o estudo é feita, devido a fatores econômicos, geralmente fica a quem da desejada, para estudos mais amplos em DTN.

No *ONE* e em outros simuladores para DTN semelhantes, a pesar do baixo custo e da possibilidade de repetição e comprovação dos resultados obtidos, estas características são completamente abstraídas das simulações, já que a conectividade nos simuladores é definida apenas em relação a parâmetros de área de cobertura de cada tipo de nó, em tempo de configuração, e pela sua mobilidade ao longo da simulação.

Além disso, devido a não utilização de sistemas operacionais completos nos nós simulados, a utilização da implementação de referência feita pelo DTN-RG [12] do Protocolo de Agregados(*Bundle Protocol*), desenvolvida para incorporar todos os

componentes da arquitetura DTN e prover um *framework* robusto e flexível para experimentação, extensão e implementação, acaba tornando-se impossível.

A utilização de uma mesma implementação do protocolo DTN facilitaria a comparação de resultados obtidos além de ajudar na melhoria e evolução do código, o que beneficiaria toda a comunidade científica.

Por isso, é necessário o desenvolvimento de alternativas de estudo de redes DTN que ofereçam escalabilidade, controlabilidade e confiabilidade ao mesmo tempo levam em consideração fatores essenciais no estudo de redes sem fio e permitam a utilização de implementações e códigos já conhecidos.

Em alguns trabalhos, pesquisadores já tem sido cogitado o uso de ambientes de análise híbridos onde são combinados componentes de rede reais e/ou emulados e componentes virtuais de forma a adicionar um nível elevado de realismo aos dados coletados ao mesmo tendo em que mantem-se a facilidade da repetitividade controlada de testes e a facilidade de modificação das características da rede no âmbito virtual para a realização de testes diversos. Desta forma os resultados obtidos nesses ambientes podem ser facilmente transferidos para a realidade.

Com a utilização de DTNs virtualizadas, onde há um sistema operacional completo em cada nó da rede, é possível executar os mesmos conjuntos de códigos que seriam utilizados em uma implementação real facilitando a implementação destes em cenários reais.

1.4 Objetivo

A virtualização de redes vem sendo utilizada em diversos ambientes de testes e emuladores de rede para expandir as capacidades limitadas de hardware, permitindo assim, a execução de estudos em escalas maiores mas de forma a manter parte da facilidade de gerenciamento proporcionada por simuladores, e o realismo obtido com

ambientes de teste reais.

Um exemplo é o que pode ser visto em [13] onde usando virtualização, realizam-se múltiplos experimentos em redes sem fio usando multiplexação por divisão de tempo.

Como objetivo principal deste trabalho está a verificação da aplicabilidade da virtualização no estudo de redes DTN, de forma a agregar confiabilidade com reprodutibilidade dos resultados.

Para isso, algumas verificações devem ser cumpridas para que se possa afirmar que ambientes de testes virtualizados são adequados para realizar estudos de DTNs. Estas verificações compreendem os objetivos específicos deste trabalho:

1. Identificar soluções de virtualização existentes que possam ser utilizadas para a construção de um ambiente de teste virtualizado que permita a locação de recursos eficiente e justa, sem causar interferências nos resultados.

2. Identificar formas de emulação/virtualização do canal sem fio que permitam a reprodução do comportamento e das características encontradas em ambientes reais.

3. Verificar a conformidade dos resultados obtidos com resultados esperados de um ambiente real.

4. Permitir a execução de diferentes cenários de rede.

1.5 Organização do trabalho

No Capítulo 2 apresentamos os aspectos relevantes para virtualização de máquinas, virtualização de redes e as características estocásticas de canais sem fio 802.11.

No Capítulo 3 descrevemos o ambiente desenvolvido para avaliação da proposta

deste trabalho, o VDT - Virtual DTN Tesbed, bem como o ambiente de testes utilizado como referência para comparação.

No Capítulo 4 especificamos o projeto de experimentos utilizado ao longo do trabalho.

No Capítulo 5 apresentamos e analisamos os resultados. Por fim, no Capítulo 6 apresentamos as conclusões e os trabalhos futuros.

Capítulo 2

Fundamentação teórica

O avanço das redes de comunicação de dados, principalmente as sem fio, e a necessidade de estar sempre conectado, têm trazido aos pesquisadores da área de redes diversos desafios de pesquisa para a construção das redes de comunicação de próxima geração e dos protocolos e técnicas que serão a base destas redes. Um exemplo disso são as redes tolerantes a atrasos e desconexões(DTN).

O conceito de DTN surge como uma solução para cenários de comunicação intermitente com frequentes quedas e desconexões. As DTNs utilizam o conceito de armazena, transporta e repassa para o encaminhamento das mensagens, exigindo que o nó transportador tenha uma implementação que o habilite a executar essas funções.

Segundo Hahn [10], pesquisadores em DTN geralmente usam simuladores de redes como o *Oportunistic Network Environment* (ONE) [4] para a execução de seus experimentos. Ao fazer isso, produzem resultados com pouco realismo, já que diversas características do mundo real são abstraídas para modelos simplificados, pela necessidade de obtenção de resultados em curto tempo.

A afirmação de Hahn, apesar ser uma verdade em determinados casos, não se aplica quando há um cuidado dos parâmetros na escolha para modelar as carac-

terísticas do canal sem fio e quando há tempo suficiente para se realizar a simulação.

Ainda segundo Hahn, ambientes de testes reais, a pesar de poderem produzir resultados mais realísticos, possuem desafios como: dificuldade de gerenciamento, reprodutibilidade dos resultados, custo de implementação (às vezes proibitivo) e a dependência de tecnologias já existentes, o que dificulta o estudo de novas tecnologias.

Segundo Chowdhury [14], a realização de experimentos tem sido um dos principais fatores que motivaram a pesquisa em virtualização de redes. A virtualização de redes surgiu da necessidade de realizar testes que fossem isolados, confiáveis e reprodutíveis, características essas que não eram possíveis de serem alcançadas com os ambientes de testes existentes.

2.1 Trabalhos relacionados

Zimmermann [2] apresenta o UMIC-Mesh, um ambiente de testes híbrido baseado em pico-computadores com interfaces de rede sem fio atuando como a parte física da rede, e com máquinas virtuais interconectadas por meio de funcionalidades avançadas de rede do kernel do linux, na parte virtual, de forma a estender o alcance da rede real.

Na abordagem empregada, o controle de conectividade entre os nós virtuais é simulado pela filtragem de pacotes através do endereço MAC entre os nós da rede que não se comunicam e da emulação de fatores como atraso, perda e largura de banda, entre os nós restantes. São dados alguns conceitos iniciais sobre ambientes de teste híbridos, as vantagens da utilização de virtualização, e os ponto positivos de utilizar esta abordagem.

O objetivo principal do artigo apresentado por Zimmermann é demonstrar como o desenvolvimento de softwares e a validação de protocolos pode ser otimizada ao

utilizar a virtualização para replicar e estender uma rede existente, utilizando-a como geradora de tráfego.

Smith [13] apresenta técnicas de virtualização do canal sem fio de redes sem fio virtuais e os desafios presentes nesta área. Neste trabalho os autores adotam uma técnica onde o núcleo da plataforma de virtualização é modificado de modo a repassar informações de controle de baixo nível entre o hardware real da máquina e o hardware virtual dos roteadores virtuais e então empregam a divisão de tempo TDM para controle de acesso e divisão dos experimentos executados no ambiente de testes.

Staub [15] apresenta o conceito do VirtualMesh, um ambiente de testes de redes sem fio que emprega a virtualização de nós e a emulação em tempo real de uma interface de rede sem fio para cada nó da rede. O tráfego gerado em cada nó da rede é então capturado e redirecionado para um modelo de propagação de redes sem fio no simulador de redes OMNeT++. O simulador então é responsável por determinar a conectividade entre os nós, introduzir a latência adequada e redirecionar o tráfego para o nó de destino, caso haja conectividade.

A abordagem utilizada por Staub possui dois pontos problemáticos: o primeiro vem do fato de que emulação da interface de rede pode trazer diversos problemas de atrasos não desejados durante a execução do experimento o que pode levar a resultados alterados quando o número de nós virtuais é aumentado; o segundo diz respeito ao modelo de propagação simplista utilizado no ambiente de simulação que não retrata as condições estocásticas de canais sem fio.

Morgenroth [5] descreve o HYDRA, um ambiente de testes de redes DTN que emprega a virtualização dos nós para obter um maior realismo de resultados à partir dos dados de movimentação coletados em simulação no ONE. Os dados coletados no ONE servem de marcadores para identificar quando dois ou mais nós realizam contato e trocam informações. Isso é controlado no ambiente de testes utilizando a

ferramenta *iptables* para habilitar o tráfego de informações sobre os nós.

O problema da abordagem proposta reside no fato de que:

1- Os experimentos realizados no ambiente de testes dependem de simulações realizadas anteriormente.;

2- Ao haver o contato dos nós, informações são trocadas entre máquinas virtuais, que são processos em um computador, e desta forma as características do canal sem fio não são levadas em consideração o que diminui a aplicabilidade dos dados coletados.

Kim [16] apresenta o VMT, um ambiente de testes de redes tolerantes a atrasos e desconexões que se baseiam, em traces de mobilidade coletados em outro projeto [17], e em virtualização para oferecer uma forma de construir e executar diversos experimentos de forma controlada e reprodutível.

O trabalho apresentado por Kim possui pontos em comum com os outros trabalhos aqui relacionados no que diz respeito a inclusão das características de redes sem fio na realização dos experimentos, pelo compartilhamento de uma rede sem fio real emulada através de um atenuador programável de rádio frequência (RF) e por placas de rede sem fio em máquinas físicas, e contribui ao adicionar mobilidade ao cenário virtualizado.

Porém, a necessidade de haver uma máquina física para cada nó virtual que está realizando alguma comunicação e o atraso do processo de troca de máquina virtual entre máquinas físicas, quando o nó virtual se movimenta ao longo do ambiente virtualizado, leva a problemas como: perda de oportunidades de contato; perda da possibilidade de existência de partições; e elevado custo para expansão do ambiente de testes já que mais máquinas físicas com interfaces de redes sem fio e atenuadores de RF são necessários.

Duas características principais da virtualização levam a crer que seu emprego no estudo de redes pode contribuir para o sucesso de seu uso no aspecto da avaliação

de desempenho:

1. A primeira é relativa a como os nós da rede a ser estudada são implementados. Este ponto é o que mais se diferencia da simulação já que os nós da rede não são apenas uma abstração simplificada, mas sim, um sistema operacional completo que se comporta como um nó real, adicionando inclusive os atrasos intrínsecos ocasionados pelo hardware ou pelo sistema operacional [18].

2. A segunda é a utilização dos mesmos conjuntos de códigos desenvolvidos para sistemas operacionais e aplicações reais. Desta forma a transição dos códigos desenvolvidos dos ambientes de testes virtualizados para as aplicações reais é facilitada já que não é necessário manter dois tipos de implementações diferentes, o que também previne a coerência de discrepâncias por diferenças de implementação [19].

Na Figura 2.1 está representado um esquema em formato de mapa mental que relaciona todos os trabalhos relacionados aqui descritos em áreas específicas.

Figura 2.1: Mapa mental dos trabalhos relacionados

Para um melhor entendimento da teoria aplicada neste trabalho, este capítulo apresenta explicações sobre as tecnologias empregadas no mesmo.

2.2 Redes Tolerantes a Atrasos e Desconexões

Segundo Carina [20], a arquitetura da Internet, uma das soluções tecnológicas de maior sucesso da atualidade, é utilizada no mundo todo para interconectar diferentes tipos de dispositivos de comunicação, em diferentes cenários, provendo suporte a diversas aplicações. No entanto, para o seu correto funcionamento, algumas características como comunicação fim-a-fim, baixa latência e pequena perda de pacotes devem ser atendidas.

Estas características que se aplicam a redes tradicionais não são encontradas em cenários considerados desafiadores, o que torna o perfil de protocolos da internet inadequado nessas situações. Como exemplo temos: a comunicação entre dispositivos móveis, comunicação entre dispositivos com restrições energéticas, comunicações no âmbito rural, comunicações em cenários de guerra ou sem infraestrutura pre-existente, e comunicações interplanetárias. Redes que atendem a essas característica foram denominadas de Redes Tolerantes a Atrasos e desconexões (*Delay and Disruption Tolerant Networks* - DTN) [20].

Apesar de o termo DTN ser mais o mais comumente encontrado na literatura, pode-se encontrar também outras denominações como: redes esparsas, redes móveis parcialmente conectadas, redes com conectividade eventual e redes com conectividade transiente [21].

As características desta e de outros novos ambientes de rede similares trazem uma série de desafios que precisam ser vencidos, como: frequentes desconexões, atrasos longos e/ou variáveis, conectividade intermitente, entre outros.

Também, segundo Carina [20], as principais características de DTNs são:

- Atrasos longos e/ou variáveis: uma DTN pode ter atrasos de horas, até mesmo, dias. O atraso fim-a-fim é determinado através da soma dos tempos de atraso salto-a-salto.

- Frequentes desconexões: as desconexões podem ocorrer devido a mudanças na topologia da rede ocasionadas pela mobilidade dos nós, por péssimas condições de comunicação, quando há economia de recursos nos nós da rede como em sensores sem fio, entre outros. Estes evento, podem resultar na não existência de um caminho fim-a-fim entre um nó fonte e um nó de destino.

Para superar os problemas associados à conectividade intermitente, as DTNs usam o princípio armazena-carrega-e-envia para o encaminhamento de mensagens.

Em DTNs, como a conectividade entre os nós de origem e destino é eventual ou até inexistente, os nós intermediários devem encaminhar as mensagens aos entre si, repassando as informações da origem, através dos vizinhos adjacentes até que um deles faça a entrega ao nós destinatário.

Para não haver perda de dados na ocorrência de quebras de conexão entre os nós, é necessário que cada mensagem seja armazenada em cada nó intermediário, quando da ocorrência de uma nova oportunidade de conexão, estas possam ser encaminhadas em direção ao destino.

Diferentemente das redes de comutação de pacotes, onde cada pacote corresponde a uma parte de toda a informação, as redes DTN realizam o agrupamento de blocos de informações diferentes, gerados pelas aplicações, em uma única mensagem denominada *agregado* (*bundle*) que então é encaminhado entre os nós da rede seguindo princípio de armazena-carrega-e-envia [22]. Aos agregados é adicionado um cabeçalho para processamento, autenticação e endereçamento.

2.2.1 Arquitetura de Redes Tolerantes a Atrasos e Desconexões

A Arquitetura de redes tolerantes a atrasos e desconexões é descrita pela RFC 4838 [23] que estabelece os conceitos de redes com conexão ocasional com frequentes particionamentos e interrupções na comunicação.

A arquitetura DTN utiliza a técnica de comutação de mensagens e o armazenamento persistente dos dados definindo uma camada intermediaria (*overlay*) abaixo da camada de aplicação. Esta nova camada é denominada camada de agregação (textitBundle Layer) e o protocolo a ela atribuído é executado em todos os nós pertencentes à rede DTN, da origem até o destino, de forma semelhante a que ocorre em redes IP.

A ocorrência de divisões da rede em *subredes* é denominada de rede regional. Nestas redes, a arquitetura em sobrecamada permite a DTN funcionar de forma completamente transparente, permitindo que as aplicações se comuniquem através de múltiplas regiões. Para garantir interoperabilidade com qualquer tipo de tecnologia de rede rede, esta sobrecamada situa-se acima da camada transporte.

As aplicações em DTN enviam mensagens de tamanhos variáveis chamadas de unidades de dados da aplicação (*Application Data Units* - ADUs). As mensagens são transformadas pela camada de agregação em uma ou mais unidades de dados de protocolo (*Protocol Data Units* - PDUs) denominadas agregados (bundles), que são armazenados e encaminhados pelos nós DTN, desta forma, uma página web pode ser enviada em conjunto com dados de imagens e estilos de formatação. Todas as informações são *agregadas* e enviadas de uma única vez, o que evita a realização de inúmeras trocas de mensagens, comuns em redes TCP/IP convencionais.

Os agregados recebem um identificador único que contém além de informações sobre o remetente, informações sobre o destinatário, o tempo de vida, armazenamento e segurança. Também, podem ser eventualmente fragmentados para facilitar o encaminhamento na rede, dependendo das características da rede regional atravessada. As informações sobre remetente e destinatário contém um identificador do nó, na forma de um localizador de recursos universal (URI), por exemplo: dtn://grcm.ufam.dtn/arquivos. Esta URI pode ser dividido em duas partes:

- A primeira, definida por "dtn://grcm.ufam.dtn"é denominada Identificador de

Ponto Terminal (EID) é usado para identificar os nós no envio e recebimento de mensagens entre dois nós (unicast) ou entre grupos de nós (multicast) e dever ser única para cada nó.

- A segunda, definida por "/arquivos", identifica a aplicação dentro do nó a que os dados se destinam ou foram originados.

Segundo [24], além dos dados de origem e destinatário, o encaminhamento de mensagens em uma DTN pode ter as seguintes opções

- *Numero de cópias*: Estabelece quantas cópias devem ser gerada e enviadas pelo remetente.
- *Transferência de custódia*: Transfere a responsabilidade de armazenamento da mensagem. Desta forma o nó de onde a mensagem se origina libera recursos para armazenamento de outras mensagens.
- *Recibo de transferência de custódia*: O nó encaminhador informa ao remetente a aceitação da custódia. Assim o nó remetente pode liberar os recursos antes alocados para quela mensagem e garante que esta foi transmitida corretamente.
- *Recibo de entrega fim-a-fim*: O destinatário informa ao remetente o recebimento da mensagem.
- *Recibo de exclusão*: É informado ao remetente a exclusão da mensagem. Seja por tempo de entrega excedido ou outros problemas.
- *Recibo de encaminhamento*: O nó encaminhador informa ao remetente o encaminhamento da mensagem.
- *Recibo de recepção*: O nó encaminhador informa ao remetente o recebimento da mensagem.

As mensagens presentes em uma DTN são encaminhadas quando há a ocorrência os contatos. Contatos são ocasiões com tempo variável onde é possível haver a troca de mensagens armazenadas entre os nós [24]. Estes contatos podem ser classificados como:

- *Persistentes*: Estão sempre disponíveis, como uma conexão física entre dois nós.
- *Sob Demanda*: Necessitam que alguma ação seja tomada acontecerem. Uma conexão discada, por exemplo.
- *Agendados*: Ocorrem em espaços de tempo previamente definidos e com duração conhecida.
- *Oportunistas*: Ocorrem de forma inesperada e aleatória.
- *Previsíveis*: Podem ter sua ocorrência prevista através de um histórico de contatos.

2.3 Métodos de avaliação de desempenho

2.3.1 Avaliação de desempenho de redes

A proposta de novos protocolos ou técnicas precisa estar embasada em avaliações de desempenho para que seja comprovado que um determinado novo protocolo ou técnica é superior ao atual.

Tradicionalmente, a pesquisa em redes tem utilizado diversas ferramentas como ambientes de testes reais, simulações, análise matemática e mais recentemente, emulações de rede [1]. Segundo Zimmermann [2], cada técnica possui um nível diferente de abstração das características presentes em redes reais e isto torna os resultados obtidos mais fáceis ou difíceis de serem transportados para a realidade.

Quanto mais parâmetros de entrada são considerados em uma análise de desempenho mais precisas podem ser as conclusões obtidas em relação ao mundo real [25]. Além disso, existem diferentes custo-benefícios em relação a cada método, como o estágio em que este pode ser aplicado à pesquisa, o tempo requerido para obtenção dos resultados e o poder de convencimento dos mesmos [3].

2.3.1.1 Ambientes de teste Reais

Ambientes de testes reais geralmente proporcionam o mais alto grau de confiabilidade e convencimento dos dados obtidos quando possuem uma metodologia bem executada, pois geralmente é mais fácil acreditar em resultantes de um sistema implementado de forma muito parecida com as redes existentes na realidade. O fato de um ambiente real estar disponível, nos garante que, quando bem conduzida, a experimentação nos levará a resultados próximos dos observados no mundo real.

Frequentemente, porém, para atingir o nível de confiabilidade desejado são necessárias diversas iterações de coletas de dados além de que deve-se ter a preocupação constante com as adversidades impostas pelo ambiente e não previstas durante o projeto do experimento [26] o que prejudica a reprodutibilidade dos resultados. Outros problemas desta técnica ocorrem quando é necessário fazer análises de alternativas de configurações ou quando deseja-se investigar alguma tecnologia que depende de hardware completamente novo.

Além disso, a interrupção dos serviços à cada nova necessidade de reconfiguração dos equipamentos durante os testes na rede pode deixar frustrados os usuários que fazem uso desse ambiente de testes e com isso uma das parcelas importantes no estudo, a geração de tráfego realista, fica comprometida [27].

2.3.1.2 Simuladores de Rede

Simuladores de rede são as ferramentas geralmente escolhidas para estudo e avaliação de desempenho de redes devido à flexibilidade que elas oferecem na criação de cenários grandes e complexos e o controle durante a execução dos experimentos [5].

Existem hoje alguns de simuladores de rede conceituados como o NS-2 [28] e o OMNeT++ [29] que são oferecidos como software livre, além de alguns comerciais como é o caso do Qualnet [30] e OpNet [31] e de diversos outros simuladores criados para algum tipo de análise específica.

O uso de simuladores garante a reprodutibilidade dos resultados, simplifica o gerenciamento do experimentos e provê flexibilidade na configuração do ambiente de testes, isto aliado a um custo baixo ou até as vezes nulo já que geralmente utiliza-se os recursos computacionais já disponíveis.

Simuladores permitem também o estudo de redes completamente novas que ainda não possuam implementações reais ou equipamentos compatíveis e com a parametrização correta podem se aproximar bastante dos resultados obtidos no mundo real [18]

O problema do uso de simuladores vem do fato de quando estes são erroneamente utilizados, quando por questões de tempo e de simplificação da implementação para obtenção resultados de forma rápida, utilizam implementações aproximadas dos protocolos a serem estudados [2, 5] e abstraem todas as influências do sistema operacional, da pilha de rede e do hardware [32], o que pode deixar elementos importantes para a avaliação de fora do estudo.

Estas simplificações resultam, de um ponto de vista de programação, em um problema ao transferir códigos utilizados em simuladores para aplicações reais e a não conformidade dos resultados obtidos com os observados na realidade devido

a ausência de aspectos como os atrasos ocasionados pelo hardware e pelo sistema operacional.

2.3.1.3 Emulação

Para contornar os problemas de abstração e controlabilidade, diversos trabalhos [33, 34, 35, 36, 32, 9] propõem o uso de emulação de redes na avaliação de desempenho.

Segundo Zheng [33] emuladores de rede são simuladores que funcionam em tempo real e possibilitam testes e avaliações de sistemas de rede, protocolos e aplicações em um ambiente de hardware e software controlável e reconfigurável.

Com base em trabalhos anteriores, neste trabalho definimos duas abordagens para esta técnica:

- *Clássica*: a abordagem clássica consiste em reproduzir no nível de software características de rede reais (perda de pacotes, reordenação, atraso e *jitter*) entre nós físicos [36, 37],

- *Emulação de Ambiente*: a emulação de ambiente [38, 1] consiste em fazer com o que o tráfego gerado em um simulador passe por um ambiente de rede real, como dois nós de uma rede sem fio.

Apesar do nível de fidelidade e controle ser maior, redes emuladas possuem problemas de escalabilidade, no caso da abordagem clássica, já que cada nó corresponde a uma máquina na rede, e a possibilidade de interferência nos resultados devido a sobrecarga imposta pela emulação dessa rede.

2.3.2 Avaliação de desempenho de redes sem-fio

Motivados pelo grande interesse na área e pelos problema envolvidos na construção de ambientes de testes reais para rede sem fio, grande parte dos trabalhos de

avaliação de desempenho tem utilizado a simulação para realização de experimentos e por causa desta tendência, alguns pesquisadores têm questionado a credibilidades destes experimentos [25].

A pesar de não serem necessárias para todo e qualquer avaliação de desempenho em simuladores, a não utilização dos modelos mais completos e complexos de propagação, a não exploração e otimização dos parâmetros presentes nestes modelos e a utilização de modelos extremamente simplistas, impõe riscos principalmente no desenvolvimento de novos protocolos [25].

No estudo de redes sem fio, características fundamentais como a aleatoriedade intrínseca do canal sem fio e a mobilidade dos nós precisam ser levados em consideração para que haja fidelidade entre os dados obtidos nos estudos e o que acontece na vida real [39].

Devido a estas características, muitos algoritmos que apresentam bons resultados em simulações podem ter um desempenho muito a quem do esperado quando submetidos a cenários reais com condições dinâmicas. Modelos mais realistas levam em consideração características como a altura e orientação de antenas, o tipo de terreno, a possibilidade de obstáculos e a ocorrência de reflexões e absorções de sinal [40].

Outra forma de se avaliar o desempenho de redes sem fio é utilizando ambientes de teste reais, que permitem que os experimentos sejam feitos usando o próprio canal sem fio. Como já evidenciado, esta forma de avaliação de desempenho possui limitações, principalmente devido a razões econômicas, quanto a escala do ambiente de teste e da não realização do estudo em um ambiente completamente isolado, o que limita seriamente a reprodutibilidade dos resultados obtidos.

Interferências oriundas de redes existentes são possíveis e dificilmente evitáveis o que torna a tarefa de análise de novos protocolos desafiadora. Além disso, a adição de características de mobilidade, principalmente quando deseja-se estudar grandes

cenários com muitos nós, torna mais inviável, economicamente, a utilização desta técnica em uma grande escala.

A emulação de redes sem fio é algo que vem ganhando destaque pela comunidade científica por reunir os benefícios da simulação com a representatividade dos dados de ambientes de teste reais.

Um método muito utilizado é o da emulação de ambiente. Nela são utilizadas as mesmas plataformas de *Hardware* e *Software* que os ambientes de teste reais, incluindo as interfaces de rede sem fio. Nós são posicionados próximos uns dos outros de forma a ocasionar intencionalmente interferência. Este tipo de emulação é relativamente simples e comumente utilizada. As mudanças de conectividade, devido a mobilidade, são emuladas usando filtros de camada MAC que filtram seletivamente o tráfego entre os nós [41].

Esta forma de implementação possui problemas de escalabilidade já que adicionam um nível de complexidade à emulação [41] quando mobilidade e o canal precisam ser implementados, como descrito por Staub [32].

Ainda que utilizando-se o método clássico de emulação, onde os problemas de escalabilidade são severamente diminuídos, a implementação de mobilidade relacionada à estados de link, baseados em modelos de propagação e distância, pode ocasionar alterações nos resultados, já que este método não leva em consideração a característica estocástica dos canais sem fio.

2.4 Características de redes sem fio 802.11

2.4.1 Interferência em redes 802.11

O padrão IEEE 802.11 é definido sobre a faixa de frequência de uso livre e não protegido de 2.4 Ghz da banda *Industrial, Scientific and Medical* (ISM) para

as versões *B* e *G* e sobre a faixa 5Ghz da banda *Unlicensed National Information Infrastructure* (U-NII) para a versão *A*.

A não necessidade de licenciamento na utilização faixas de frequências ISM e U-NII permite que qualquer usuário com equipamentos homologados pela agência reguladora de telecomunicações, Agência Nacional de Telecomunicações (ANATEL) no caso do Brasil, possa realizar comunicações, porém, exime o órgão regulamentador de garantir a não ocorrência de interferências ao longo da comunicação.

A interferência em comunicações sem fio pode ser descrita como a captação simultânea de sinais por uma mesma interface de rede sem fio, de forma que pelo menos um dos sinais era endereçado ao receptor. Quando isto ocorre há a colisão das ondas eletromagnéticas, o que acaba transformando a onda original em uma somas das ondas, desta forma, distorcendo o sinal original [42]. A interferência pode ser classificada de acordo com o tipo de fonte geradora dos sinais envolvidos na interferência, conforme detalhado nós próximos tópicos.

2.4.1.1 Interferências causadas por outros equipamentos

Além de sistemas de redes sem fio 802.11, outros sistemas utilizam a faixas de frequência das bandas ISM e U-NII, por isso, outros equipamentos operando sob padrão diferente do 802.11 mas que utilizem a mesma faixa de frequências são possíveis geradores de interferência. Alguns exemplos de equipamentos que utilizam a faixa 2.4Ghz são: telefones sem-fio, rádio comunicadores, dispositivos *bluetooth* (padrão IEEE 802.15), entre outros. Além disso ela pode sofrer ruídos de frequências emitidas por dispositivos eletro-eletrônicos como fornos micro-ondas, e de dispositivos de comunicação diversa como mouses sem fio, que compartilham a mesma frequência.

2.4.1.2 Interferências causadas por outros sistemas 802.11

Este tipo de interferência ocorre quando os sinais que ocasionam a colisão são provenientes de outros nós pertencentes da mesma rede ou de outra rede adjacente.

Estratégias de controle de acesso ao meio foram definidos para regularizar a utilização do canal entre dispositivos sem fio, denominados protocolos de controle de acesso ao meio (MAC - Media Access Control). Estes procolos fazem parte dos padrões estabelecidos pelo 802.11 [42]

No protocolo 802.11 é utilizado o mecanismo CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) como estratégia de controle de acesso pela camada MAC. O CSMA/CA faz a escuta do canal para determinar se este está sendo utilizado.

Se o canal estiver ocioso por um espaço de tempo maior ou igual ao *espaçamento inter-quadros distribuído* (*Distributed Inter Frame Space* - DIFS) a estação ainda espera um tempo calculado aleatoriamente, para só então transmitir informações no canal, do contrário ela determina um tempo de espera para realizar nova tentativa e repete o processo em iterações contínuas até conseguir acesso ao canal.

A transmissão é definida como sucesso se uma informação de resposta (`ACK`) for recebido, do contrário, todo o processo se reinicia.

Apesar do sucesso satisfatório na utilização desta estratégia na maioria das situações, há algumas em que CSMA/CA não funciona de maneira satisfatória. A seguir, são descritas duas situações clássicas, amplamente descritas na literatura:

- O problema do terminal escondido: ocorre quando um nó A, envia informações para um nó B sem ter conhecimento de um terceiro nó C, que não está em sua área de cobertura. Assim, apenas B possui conhecimento da existência dos nós A e C e ao realizar comunicação com dos nós o terceiro também tentar se comunicar, ocasionará colisão e perda das informações. Uma representação

gráfica do problema pode ser vista na figura 2.2

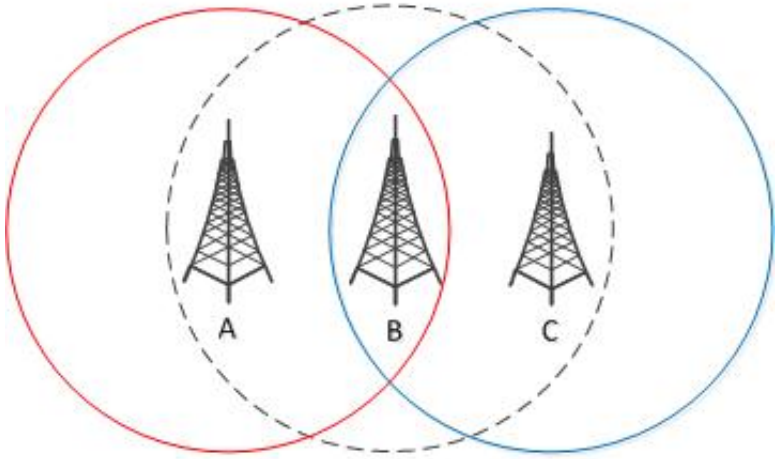


Figura 2.2: Problema do terminal escondido

Uma forma de amenizar problema consiste em utilizar um mecanismo de solicitação e reserva de canal conhecido como RTS/CTS (*Request to Send/Clear to Send*). Assim, quando um nó deseja realizar uma comunicação e o canal está livre ele envia a informação de RTS para que todos de sua área de cobertura saibam de sua intenção de transmitir informações e recebe na resposta um CTS do nó de destino.

- O problema do terminal exposto: ocorre quando pelo uso RTS/CTS dois nós A e B em uma comunicação ativa impedem outros dois nós C e D de realizar comunicação já que C faz parte da área de cobertura de B que requisitou o canal para se comunicar com A, gerando assim uma degradação no desempenho da rede pelo bloqueio momentâneo da comunicação. Uma representação gráfica pode ser vista na figura 2.3

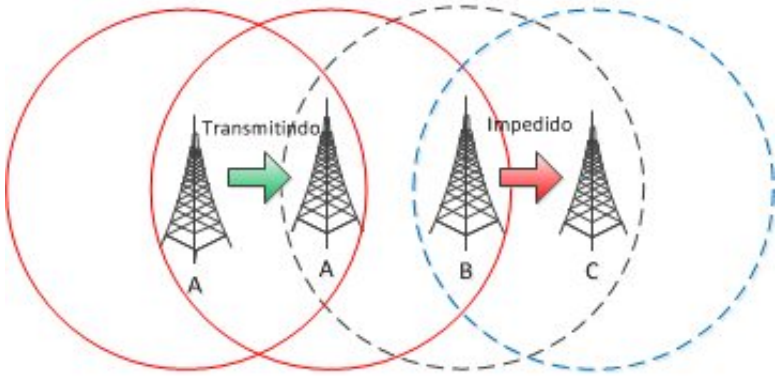


Figura 2.3: Ilustração do problema do terminal exposto

2.4.2 Assimetria de enlaces sem fio

Enlaces de redes sem fio podem sofrer o fenômeno da assimetria. Este fenômeno acontece quando enlaces não possuem as mesmas características em ambas as direções por somente umas das estações estar no raio de transmissão da outra.

A assimetria pode ocorrer principalmente quando o conjunto de elementos atenuadores da potência de transmissão de cada estação é diferente [43]. Desta forma, mesmo quando os enlaces apresentam simetria há uma possibilidade de que força do sinal percebida varie de acordo com o sentido do enlace.

2.4.3 Atenuação do sinal

Para que haja comunicação, a relação sinal/ruido no lado da recepção deve ser maior que um valor mínimo especificado, de acordo com as características do equipamento. Em relação a este quesito, a *distância* entre as estações comunicantes é um fator determinante no enfraquecimento do sinal transmitido. Além da distância, outros fenômenos também podem causar atenuação do sinal. A seguir são apresentados

os conceitos de reflexão, difração e espalhamento.

- *Reflexão* - Esse fenômeno ocorre quando as ondas eletromagnéticas incidem sobre uma superfície *refletora* de dimensão maior do que o comprimento da onda irradiada (por exemplo metais, lâminas d'água, paredes, edifícios). De acordo com tipo de material e o ângulo de incidência, o sinal refletido pode permanecer intacto ou sofrer perda devido a absorção de parte do sinal (geralmente transformado em calor). O que forma dois ângulos iguais: o ângulo de incidência e o ângulo de refração, conforme ilustrado na figura 2.4.

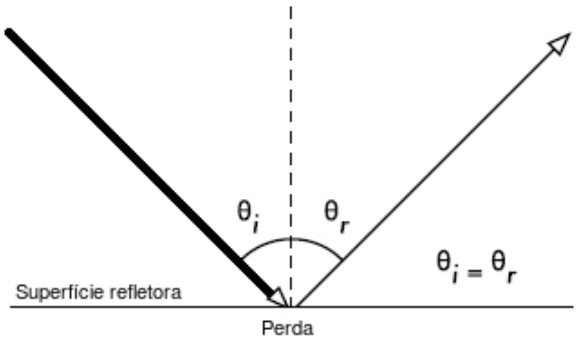


Figura 2.4: Reflexão de ondas eletromagnéticas.
Adaptado de: [44]

Reflexões podem causar muitos problemas em redes sem fio, como a degradação ou cancelamento do sinal original ou buracos em uma área de cobertura.

Em alguns sistemas sem fio a reflexão também pode ocasionar o fenômeno do *enfraquecimento por múltiplos percursos* (*multipath fading*), onde diferentes componentes do mesmo sinal são propagados por diferentes caminhos e alcançam o destino em instantes diferentes e com potências diferentes. Porém, tecnologias de rede sem fio mais recentes que se utilizam de várias antenas para recepção e transmissão, utilizam exatamente esta característica de canais sem fio para proporcionar maior capacidade de transmissão.

- *Difração* - Ocorre quando o sinal sofre um desvio devido a existência de obje-

tos com bordas irregulares no percurso de propagação do sinal (por exemplo: quinas de prédios), conforme ilustrado na figura 2.5. Parte do sinal que circunda a superfície sofre um diminuição na sua velocidade de propagação enquanto que a outra parte mantém a velocidade de propagação original. Neste caso quanto mais acentuada for a curva, mais fraco será o sinal difratado.

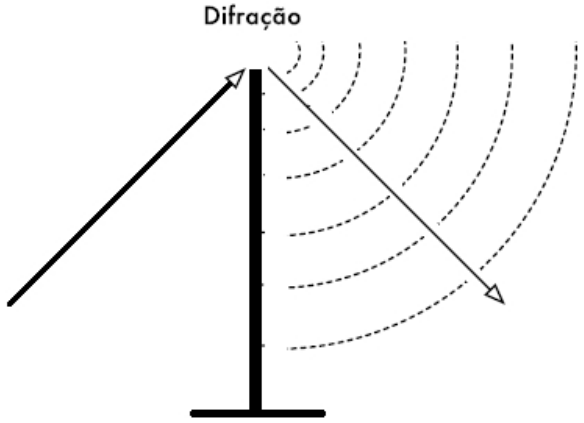


Figura 2.5: Difração de onda eletromagnética.
Adaptado de: [44]

- Dispersão ou espalhamento - Ocorre quando as ondas eletromagnéticas incidem sobre objetos de dimensões menores ou iguais ao comprimento da onda irradiada e o número de obstáculos por unidade de volume é grande (como por exemplo: telhados e pequenos objetos), conforme ilustrado na figura 2.6.

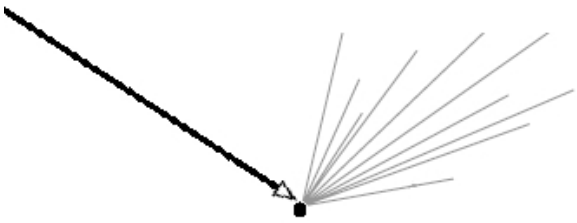


Figura 2.6: Espalhamento de onda eletromagnética.
Adaptado de: [44]

Para uma estudo mais detalhado sobre o impacto destes e outros fenômenos em canais sem fio, consultar [42].

2.5 Modelagem de canais 802.11

De acordo com Kotz em [25], para a avaliação de desempenho de protocolos de rede que utilizem sistemas sem fio, a utilização de modelos de propagação simplistas podem levar a resultados que não são observados na realidade. A seguir apresentamos algumas entre as principais alternativas disponíveis para esses modelos em canais 802.11.

2.5.1 Modelos de propagação

Com a construção de modelos da propagação do sinal eletromagnético estima-se a potência dissipada ao longo de percurso de propagação do sinal.

Segundo Rappaport[42] Os modelos de propagação são classificados em:

- *Larga Escala*: estimam a qualidade média do sinal a uma determinada distância do transmissor, nestes casos considera-se a degradação sofrida ao longo do percurso de propagação da onda (*path loss*).
- *Pequena Escala*: ou modelos de desvanecimento (*fading models*), capturam as rápidas flutuações da força do sinal recebido em curtos espaços de tempo, e quando o receptor é movido em uma fracão do comprimento de onda usado.

Nas próximas subseções seguintes serão apresentados alguns modelos tipicamente utilizados no estudo de redes 802.11.

2.5.1.1 Larga Escala - Determinísticos

Os modelos de propagação determinísticos *Friis*, *Two-Ray ground* e *Log-Distância* são dependes principalmente da distância d, da potência do sinal transmitido P_t, dos ganhos de antena G_t e G_r e de um fator $L \geq 1$, referente a perdas não relacionadas à propagação propriamente dita.

Nesses modelos é assumida a existência de uma linha de visada direta entre o transmissor e o receptor, porém, o *Two-Ray ground*, apresentada em [42], também leva em consideração a ocorrência de um percurso de propagação através de reflexão reflexão sobre uma superfície plana, geralmente o solo.

Essa diferenciação confere ao *Two-Ray ground* uma representação ligeiramente mais precisa que a de *Friis*, porém este modelo não proporciona bons resultados para distâncias muito curtas devido a oscilações pela combinação construtiva e destrutiva dos dois raios.

A estimação da potência P_r no receptor é obtida partir das fórmulas 2.1 para *Friis* e 2.2 para o *Two-Ray ground*. E a perda no caminho em 2.3 para *Log-Distância*, onde λ é o comprimento de onda em metros, h_t, h_r representam as alturas das antenas em relação à superfície plana e L_0 representa a perda no caminho pra a distância de referência (geralmente igual a 1):

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L0} \tag{2.1}$$

$$P_r = \frac{P_t G_t G_r (h_t^2 h_r^2)}{d^4 L0} \tag{2.2}$$

$$L = L_0 + 10_n log10(\frac{d}{d_0}) \tag{2.3}$$

Enquanto o modelo *Friis* é aconselhável apenas para estudos teóricos ou para projeto de protocolos de comunicação devido a utilização de simplificações que pouco descrevem o ambiente real, verificou-se que o modelo *Two-Ray ground* é um modelo geralmente visto como a escolha correta para modelagem de ambientes rurais [45].

2.5.1.2 Pequena Escala - Estatísticos

Segundo Rappaport [42], a atenuação em pequena escala é causada pela interferência entre duas ou mais versões do sinal transmitido que chegam no receptor em tempos ligeiramente diferentes.

Devido a característica estocástica dos canais sem fio, sistemas de transmissão sem fio podem sofrer oscilações em curtos períodos de tempo. Este fenômeno ocorre devido a diversos de fatores como: movimentação dos nós na área de cobertura, largura de banda do canal ou enfraquecimento do sinal por multi-percursos.

A *Distribuição Rayleigh* é usada para descrever a natureza estatística, variável no tempo, do sinal recebido causado pela ocorrência de múltiplos caminhos entre transmissor e receptor. A antena do nó móvel recebe um grande número N de ondas refletidas e dispersas. Devido aos efeitos de cancelamento de ondas, a potência instantânea percebida pelo nó se torna uma variável aleatória, dependente da localização da antena.

Sua função densidade probabilidade é dada por 2.4 onde σ^2 é a média da potência do tempo do sinal recebido antes da detecção do envelope:

$$p(r) = \begin{cases} \frac{r}{\sigma^2} exp(-\frac{r^2}{2\sigma^2}), & \text{se} \quad (0 \leq r \leq (\infty), \\ 0, & \text{se} \quad (r < 0) \end{cases} \tag{2.4}$$

A *Distribuição de Rice* é geralmente empregada para modelar fenômenos de desvanecimento por múltiplos percursos quando há um sinal estacionário dominante (sem atenuação), como uma linha de visada entre os nós [42]. O parâmetro chave desta distribuição é o fator K de Rice. Este valor é definido como a razão entre a potência do sinal de um percurso com visada (*Line-Of-Sight* - LOS) e a potência do sinal de um percurso *sem visada* (*Non-Line-Of-Sight* - NLOS). Assim, a media que

$k \to \infty$ a influência de NLOS tende a zero. E a medida que $k \to 0$ a *Distribuição de Rice* degenera para uma *Distribuição Rayleigh*.

Por fim, a distribuição Nakagami de desvanecimento rápido pode ser aplicada sobre a potência de sinal recebido e serve para transformar um modelo de propagação determinístico em um probabilístico. Nesta distribuição, a potencia de recepção segue uma distribuição gama e sua função probabilidade densidade é dada por 2.5:

$$f(x; m; \Omega) = \frac{2m^m}{\Gamma(m)m^\Omega)} x^{2m-1} exp(-\frac{m}{\Omega}x^2) \tag{2.5}$$

O valor m é definido sobre três campos de distância (0 para d_1 é m_0, d_1 para d_2 é m_1 e d_2 para d_3 é m_2). Quando m é igual a 1, a distribuição Nakagami deriva pra a distribuição Rayleigh e quando $\lim_{m\to\infty} f(x; m; \Omega)$ a distribuição deriva para a distribuição espaço livre.

Para mais detalhes sobre os modelos de propagação em pequena escala, consultar Comunicações sem fio de Rappaport[42].

2.6 Virtualização

Segundo Carissimi [46], virtualização é uma tecnologia que oferece uma camada de abstração e multiplexação dos recursos de uma máquina de forma a repartir os recursos físicos, transparentemente, entre várias aplicações e/ou sistemas denominados máquinas virtuais.

O conceito de máquina virtual permite um mesmo computador ser compartilhado como se fossem vários. A IBM definiu a máquina virtual como uma cópia protegida e isolada do hardware físico, desta forma aplicações e até sistemas operacionais se comportam exatamente como se comportariam no hardware original.

A seguir são definidos os componentes que constituem a estrutura de sistemas de virtualização segundo [46], e sua organização pode ser vista na figura 2.7:

- Máquina Virtual/Sistema Convidado: uma máquina virtual (*Guest*) nada mais é que uma camada de software que oferece um ambiente completo muito similar a uma máquina física. Com isso, cada máquina virtual pode ter seu próprio sistema operacional, bibliotecas e aplicativos.

- O Monitor de Máquina Virtual (VMM - *Virtual Machine Monitor*) é o componente de software que hospeda as máquinas virtuais convidadas e fornece uma abstração dos recursos físicos para as máquinas virtuais, como controle de entrada e saída de dados, armazenamento e gerenciamento de memória. Como o VMM provê uma abstração dos recursos, diferentes sistema operacionais, de diferentes arquiteturas podem ser executados em um mesmo sistema.

- O hospedeiro é o Sistema Operacional (SO) executado diretamente sobre o hardware físico ou servidor.

Entre as décadas de 60 e 70 do século XX, definia-se uma máquina virtual como uma cópia eficiente, isolada e protegida de uma máquina real [46]. Esta definição foi

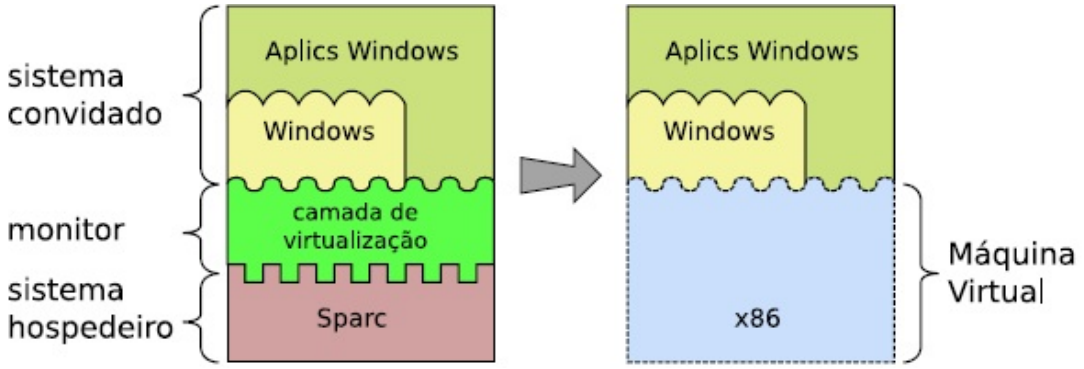


Figura 2.7: Estrutura de uma máquina virtual.
Adaptado de: [6].

constituída com base na visão sistema operacional: uma abstração de software que gerencia um sistema físico (máquina real). Porém, o termo *máquina virtual* evoluiu e englobou um grande número de abstrações como por exemplo a *Java Virtual Machine* (JVM), que não é utilizada para virtualização de sistemas operacionais, mas sim para prover um ambiente comum de execução de código Java [47].

A construção de sistemas de virtualização não é uma tarefa trivial. Quando os conjuntos de instruções do sistema real e do sistema virtualizado são diferentes, é necessário que para cada operação realizada pela máquina virtual, os conjuntos de instruções utilizados sejam traduzidos para um outro conjunto de instruções que possa ser executado pela máquina real. Além disso, é necessário mapear e gerenciar o acesso aos recursos de hardware do sistema hospedeiro entre as máquinas virtuais, e por último, pode ser necessário traduzir as chamadas de sistema emitidas pelas aplicações do sistema convidado em chamadas equivalentes no sistema real, quando os sistemas operacionais virtual e real forem distintos.

Com base nessas características, definiu-se diferentes tipos de virtualização, com objetivos específicos, como veremos a seguir.

2.6.1 Tipos de virtualização

Segundo Laureano & Maziero [6], podemos dividir a virtualização em três tipos básicos:

- *Virtualização do hardware*: a virtualização multiplexa e abstrai o sistema físico em um hardware abstrato (semelhante ao sistema original). Neste modelo, qualquer software escrito para a arquitetura nativa (x86, por exemplo) irá funcionar no sistema convidado. Este era o modelo utilizado nos mainframes IBM e é a tecnologia de virtualização utilizado pelo VMware na plataforma x86.

- *Virtualização do sistema operacional*: a virtualização exporta um sistema operacional como abstração de um sistema específico. A máquina virtual executa aplicações ou um conjunto de aplicações de um sistema operacional específico. O FreeBSD Jails e o *User-Mode Linux* são exemplos desta tecnologia.

- *Virtualização de linguagens de programação*: neste modo, a camada de virtualização cria um ambiente de execução padrão que abstrai as características do sistema operacional permitindo que um mesmo programa seja executado sem diferentes sistemas, sem mudanças na sua estrutura. O Java Virtual Machine é um exemplo deste tipo de máquina virtual.

A forma como a virtualização é implementada também possui diferenciações que impactam diretamente no desempenho do sistema convidado. A Figura 2.8 descreve de forma sucinta estes tipos:

- *Virtualização completa*: um sistema operacional convidado e suas aplicações que foram desenvolvidas para uma plataforma de hardware A, são executadas sobre uma plataforma de hardware distinta B.

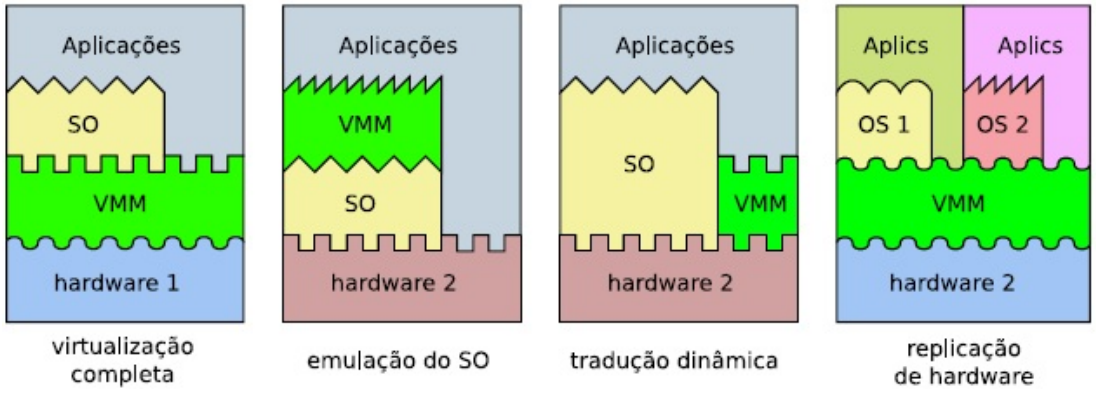


Figura 2.8: Tipos de implementação de máquinas virtuais.
Adaptado de [6]

- *Emulação do sistema operacional*: as aplicações de um sistema operacional X são executadas em outro sistema operacional sistema operacional Y, na mesma plataforma de hardware. Um exemplo que utiliza esta implementação é o projeto Open Source Wine HQ que permite a execução de aplicações desenvolvidas para Windows em sistemas nix (Linux, BSD, Solaris e Mac OS X).

- *Tradução dinâmica*: as instruções de máquina das aplicações são traduzidas durante a execução em outras instruções mais eficientes para a mesma plataforma. Implementação utilizada pelo XEN em paravirtualização.

- *Replicação de hardware*: são criadas várias instâncias virtuais de um mesmo hardware real, cada uma executando seu próprio sistema operacional convidado e suas respectivas aplicações. Implementação utilizada pelo Kernel Virtual Machine e pelo VmWare.

Este trabalho utiliza a virtualização de hardware e de sistema operacional como forma de construção de ambientes de testes que replicam ambientes reais e implementa os métodos de virtualização completa e paravirtualização, explicado em detalhes nas próximas seções.

2.6.1.1 Virtualização completa

Essa técnica consiste na criação de uma cópia virtual completa do hardware que está por trás da plataforma, na qual as aplicações podem rodar como se estivessem em seu sistema de origem. Logo, o sistema hospedeiro não sofre modificações para que o sistema convidado seja executado.

Em contrapartida, há uma variedade de dispositivos que fazem parte de um computador, o que dificulta uma solução virtual que simule exatamente cada tipo de dispositivo básico de um sistema. Em vista disso, o VMM (Monitor de Máquina Virtual), software responsável por fazer a interação entre o hardware real e virtualizado, implementa genericamente uma coleção de dispositivos e por vezes subutiliza o hardware da máquina real que está hospedando o sistema convidado.

Outro detalhe é que pelo fato dos sistemas operacionais convencionais usarem a técnica de paginação na implementação da memória virtual, os VMM precisam traduzir o espaço de endereçamento do sistema da máquina virtual para um real, o que gera uma disputa dos recursos do hospedeiro. Embora não haja implicações mais sérias quanto a isso, há uma perda em relação ao desempenho, que pode ser drasticamente diminuída com o uso de soluções de auxilio de virtualização por hardware.

São exemplos de sistemas que implementam esta técnica o KVM (*Kernel Virtual Machine*), usado neste trabalho, o Qemu e VmWare.

2.6.1.1.1 KVM

O KVM - *Kernel Virtual Machine*, transforma o kernel (núcleo do sistema) Linux um hipervisor através da adição de um módulo ao kernel com uso de tecnologias de virtualização assistidas por hardware (como Intel VT e AMD-SVM). Este módulo exporta um dispositivo chamado /dev/kvm, que permite a execução de outros sistemas com seu próprio espaço de endereço separado do espaço do kernel ou de

qualquer outra VM em execução [48].

2.6.1.2 Paravirtualização

A técnica de paravirtualização consiste na criação de uma cópia virtual completa do hardware que está por trás da plataforma, na qual as aplicações podem rodar como se estivessem em seu sistema de origem. Logo, o sistema hospedeiro não sofre modificações para que o sistema convidado seja executado.

Em contrapartida, há uma variedade de dispositivos que fazem parte de um computador, o que dificulta uma solução virtual que simule exatamente cada tipo de dispositivo básico de um sistema. Em vista disso, o hipervisor implementa genericamente uma coleção de dispositivos e por vezes subutiliza o hardware da máquina real que está hospedando o sistema convidado.

Outro detalhe é que, pelo fato dos sistemas operacionais convencionais usarem a técnica de paginação na implementação da memória virtual, os hipervisores precisam traduzir o espaço de endereçamento do sistema da máquina virtual para um real, o que gera uma disputa dos xrecursos do hospedeiro. Embora não haja implicações mais sérias quanto a isso, há uma perda em relação ao desempenho.

2.6.1.2.1 Linux Containers

O Linux *Containers* permite isolar aplicações com recursos específicos, permitindo que estas executem sem interferir umas com as outras, garantindo isolamento e performance [49].

Isto é feito através da criação de *containers*, com o uso de CGroups (Container Groups), que particionam os recursos gerenciados pelo sistema operacional em grupos isolados de forma que a demanda por recursos possa ser devidamente balanceada. Com a criação desses *containers* as aplicações tem a ilusão de estarem numa máquina totalmente isolada, mas também proporciona o compartilhamento de recursos entre

elas.

2.6.2 Virtualização/Emulação de redes

Com o auxílio de emuladores de rede pode-se adequar as características de uma rede virtual à características esperadas de qualquer rede existente, planejada e/ou não ideal. Desta forma é possível realizar experimentos e obter um alto grau de realismo em um ambiente de testes que utilize redes virtuais para verificação de performance, predição de impactos de mudanças ou para otimizar tomadas de decisão tecnológicas.

A emulação de redes pode ser alcançada pela introdução de um dispositivo na rede que altere o fluxo de pacotes de forma a imitar o comportamento do tráfego de uma aplicação de um ambiente sendo representado. Este dispositivo pode ser ou um computador com um software específico para realizar a emulação ou um dispositivo de emulação dedicado.

O dispositivo utilizado deve incorporar uma variedade de atributos de rede em seu modelo de emulação incluindo a latência, largura de banda disponível, grau de perda de pacotes, duplicação de pacotes, reordenação e *jitter* (variação do atraso). Nós conectados à essa rede poderão experienciar a performance e o comportamento de aplicações para aquele determinado ambiente.

2.6.2.1 Duplicação de características de rede de alto nível

Esta técnica diz respeito à modelagem da rede apenas levando-se em consideração as características de alto nível da rede como conexões e desconexões entre os nós (no caso de um ambiente com mobilidade), latência, largura de banda, entre outros.

Por ser uma simplificação do comportamento observado em nós reais esta forma de modelagem do canal é a que permite a maior escalabilidade. Com o mesmo

hardware é possível montar cenários de tamanho considerável, porém, pelas simplificações feitas, os resultados podem não refletir com exatidão o comportamento observado no mundo real.

Esta técnica é bastante utilizada quando se deseja forcar apenas no desenvolvimento dos algoritmos e implementações como em [5] e [27].

2.6.2.2 Emulação completa de Rede

A emulação completa de rede leva em consideração as características do canal utilizado e recria estas características através de emulação do canal em um simulador de redes como o OMNet++ [29] e o NS-3 [50].

Esta forma proporciona um nível muito maior de detalhamento o que propicia a obtenção de resultados mais próximos da realidade, porém, o custo computacional tende a aumentar conforme aumenta-se o tamanho do cenário a ser virtualizado o que, se não controlado, pode ocasionar desvios e discrepâncias nos resultados obtidos.

Capítulo 3

Descrição da Solução

3.1 Alocação de recursos em ambientes virtualizados

Técnicas de virtualização de sistemas compartilham os recursos entre as instâncias em execução e por isso sistemas virtualizados experimentam uma leve degradação de desempenho quando comparado a sistemas não virtualizados. O objetivo em ambientes de teste virtualizados é que esta degradação seja mínima, evitando que os resultados sofram interferências destas degradações. Por isso a escolha do tipo de virtualização influencia diretamente neste quesito. Na tabela 3.1 podemos ver um resumo das características já apresentadas

Item	Virtualização Completa	ParaVirtualização	Virt. de SO
Modificações	Poucas	Várias	Total
Hypervisor	Sim	Sim	Não
Perf. de SysCalls	Degradada	Intermediária	Melhor
Kernel	Individual	Individual	Compartilhado

Tabela 3.1: Comparação de Técnicas de Virtualização.
Adaptado de: [51]

Um importante requerimento para uma virtualização eficaz é a possibilidade de gerenciar a carga de trabalho, por parte do VMM. Gerenciar a carga de trabalho é designar recursos como CPU, memória, e tempos de operações de IO, da forma mais precisa possível.

Uma das formas mais simples de alocação de recursos de CPU, por exemplo, é a alocação estática. Porém esta técnica se torna ineficiente quando a carga demandada pelas máquinas virtuais é variável. Gerenciadores de carga de trabalho tem o objetivo de alocar dinamicamente os recursos físicos para atender a demanda atual de uma determinada aplicação. Geralmente esta alocação começa com um perfil básico sendo configurado para cada máquina virtual, e de acordo com observações discretas no tempo, modificada para atender à demanda atual de acordo com a quantidade de recursos livres no momento [52].

Se os recursos livres são menores que a demanda, então gera-se uma disputa de recursos que pode interferir no desempenho final de alguma aplicação sendo avaliada.

Algumas das técnicas mencionadas se beneficiam de tecnologias recentes que tem por objetivo melhorar a eficiência da alocação de recursos, como as tecnologias de virtualização por hardware Intel-VT [53] e AMD-V [54] que passam parte da carga diretamente para o hardware, eliminando a necessidade de realizar tradução das instruções emitidas por máquinas virtuais, aliviando a carga sobre o VMM ou o SO hospedeiro que pode se concentrar em outras tarefas.

3.2 Emulação/Virtualização do canal sem fio

A virtualização de redes sem fio apresenta desafios únicos, que não são observados na virtualização de redes cabeadas tradicionais. O maior desafio em redes sem fio é a virtualização do canal.

Basicamente dois tipos de abordagem tem sido adotadas:

- A primeira consiste na utilização do próprio canal sem fio ou através da multiplexação do meio com técnicas de divisão de acesso [13], ou através do compartilhamento de um ambiente de Rádio Frequência, que consiste de máquinas com placas de rede sem fio interligadas por um *switch* de RF com atenuadores programáveis que implementam as características de desvanecimento do canal e perda no caminho.

- A segunda consiste na utilização de simuladores de rede como emuladores de canal.

Apesar do primeiro método utilizar o próprio canal sem fio, a necessidade de atribuição de um equipamento interligado a cada nó comunicante, limita a quantidade de comunicações simultâneas que podem ocorrer. O que é agravado em redes DTN onde pode-se ter diversas sub-redes formadas pelo particionamento da rede principal.

O segundo método, apesar de não oferecer o mesmo nível de detalhes e fidelidade, é escalável o suficiente para permitir a comunicação simultânea de diversos nós que possam formar particionamentos da rede. A desvantagem é que o custo computacional de emular a comunicação de diversos nós ao mesmo tempo pode consumir recursos destinados às máquinas virtuais.

Uma solução para este caso é dividir a carga de processamento da emulação entre diversas instâncias de simuladores/emuladores funcionando em máquinas distintas mas com os mesmos parâmetros de configuração.

3.3 O VDT

Com o objetivo de avaliar o uso da virtualização para o estudo e realização de testes em redes DTN este trabalho propõe e implementa o VDT - *Virtualized DTN*

Testbed ou Ambiente de testes DTN virtual. O VDT compreende um ambiente de testes que utiliza os conceitos de virtualização e máquinas virtuais e a emulação de canal sem fio para prover um ambiente confiável, reprodutível, flexível e escalável para estudo de redes DTN. Uma visão geral do ambiente pode ser visto na figura 3.1.

O VDT é constituído por dois elementos centrais: a ferramenta de virtualização e um emulador de canal. A ferramenta de virtualização é responsável por implementar uma versão virtualizada dos sistemas tipicamente utilizados nos nós que constituem redes DTNs reais. Estes nós foram categorizados em dois tipos e a solução de virtualização escolhida com base no tipo de nó:

- Nós Móveis: Os nós móveis são máquinas virtualizadas com KVM que utilizam OpenWRT [55], na versão *Attitude Adjustment* 12.09, para x86. O OpenWRT é uma distribuição Linux customizada para aplicações de redes sem fio. Ela possui requerimentos de processador e memória bem baixos (comparados com requerimentos de sistemas linux tradicionais) o que possibilita seu uso em sistemas embarcados (roteadores sem fio), e em relação a virtualização, a execução de muitas instancias de máquinas virtuais em um mesmo hospedeiro sem grandes degradações na performance [5]. Por ser baseada em Linux, possui a mesma ampla gama de softwares de rede, e em geral, disponíveis.

  Outra vantagem conferida pelo uso do KVM é que além do OpenWRT, podem ser executados outros sistemas que dificilmente seriam portados para soluções paravirtualizadas, como o sistema operacional móvel Android, o que permitira o estudo de DTNs com dispositivos móveis com restrições de energia

- Nós Estáticos: Os nós estáticos são máquinas virtualizadas com o auxílio do *Linux Containers* [56]. O *Linux Containers* é um sistema de virtualização de SO que proporciona gerenciamento de recursos e isolamento de processos

para aplicações que rodam sobre linux. Desta forma é possível ter centenas de processos em seus respectivos *containers*, isolados, com máxima performance.

Esta escolha se deu pois nós DTN estáticos geralmente são implementados como *gateways* entre uma rede DTN e a internet ou como repositório de armazenamento de informações temporário entre diversos nós móveis DTN, otimizando assim as oportunidades de contato e troca de informações entre os participantes de uma rede DTN, e por isso, possuem recursos mais avançados como o uso de um hardware mais robusto.

Esses tipos de sistemas utilizam geralmente a implementação de referencia do DTN-RG [12], o DTN2, que não é otimizado para sistemas embarcados como o OpenWRT.

A implementação deste tipo de nó com alguma solução de virtualização completa ou paravirtualização seria muito dispendiosa e diminuiria a escalabilidade do sistema.

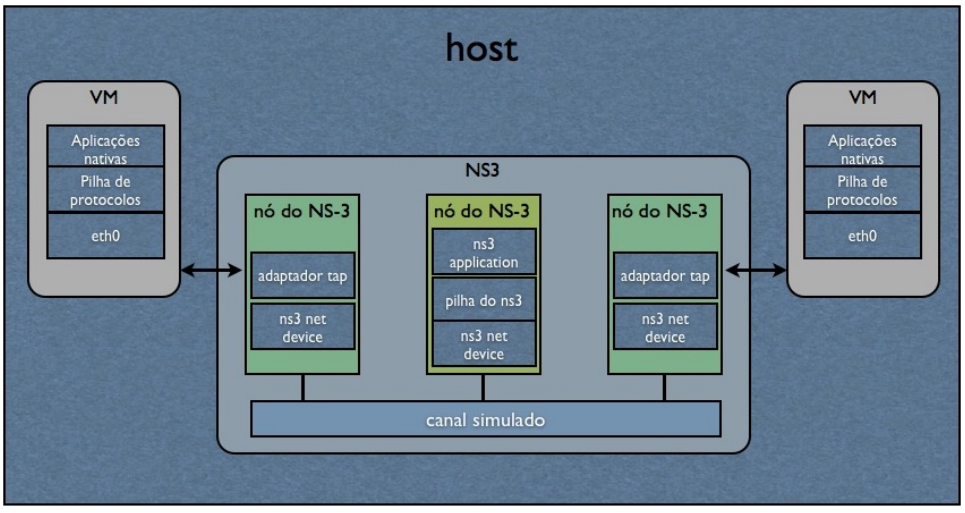


Figura 3.1: Estrutura do VDT

Interligando os nós DTN virtuais está o simulador/emulador de canal sem fio NS-3 [50] na versão de desenvolvimento 3.11. O NS3 é um simulador de redes de

código aberto derivado do NS-2 que, além do modo de simulação discreta no tempo, possui o modo "tempo real"que permite enviar e receber pacotes gerados no NS-3, de outras fontes externas ou a partir de uma rede real (através de interfaces de redes virtuais TAP), como também servir como camada de interconexão entre máquinas virtuais, no modo de emulação de canal, o qual é utilizado neste trabalho.

Uma das principais contribuições e diferenças do NS-3 para o NS-2 foi a preocupação com a validação das implementações dos modelos presentes, como em [57], e o suporte, desde o seu núcleo, à emulação.

O NS-3 suporta um maior número de modelos de propagação e atraso, sendo ao todo 23, em comparação aos suportados pelo NS-2, apenas 5, o que permite a realização de estudos que procuram reproduzir os mais diversos tipos de ambientes existentes, indo desde grande cidades com dezenas de prédios a ambiente de floresta e rurais.

Por ser totalmente baseado em software, o VDT agrega diversas das qualidades de trabalhos relacionados e resolve alguns dos problemas apontados além de adicionar novas funcionalidades.

Ao utilizar máquinas virtuais rodando o mesmo sistema de um nó real, temos a mesma implementação dos softwares e protocolos utilizados em rede reais o que facilita o desenvolvimento, já que ele é feito apenas uma única vez e permite que os resultados obtidos sejam melhor transportados para aplicações práticas.

O emulador provê a emulação do canal sem fio, levando-se em consideração as diversas características intrínsecas deste tipo de rede ao mesmo tempo que dá margem para segurança nos resultados devido a prévia análise e validação dos modelos de propagação. Além disso, como não ha a utilização de hardware real para auxilio na emulação, o número de nós que podem ser utilizados durante um experimento cresce consideravelmente, desde que o hardware onde o ambiente será executado atenda aos requisitos e capacidade necessários. Tal característica permite a ocorrência de

particionamentos na rede DTN a ser virtualizada e a formação de redes regionais de nós.

Como melhorias em relação aos trabalhos anteriores, estão: a possibilidade de criar redes com tecnologias de comunicação sem fio que seriam difíceis de se construir e utilizar na vida real, como por exemplo redes LTE, ou com custo proibitivo, como redes satelitais; a possibilidade de realização de estudos onde há a existência de particionamentos na rede e criação de grupos regionais de nós; a possibilidade de realização de testes em cenários com uma grande quantidade de nós com a agregação de vários servidores rodando, cada um, uma parte de todo o ambiente de testes.

Capítulo 4

Projeto de Experimentos

No capítulo anterior, definiu-se a solução proposta por esse trabalho para resolver o problema da realização de avaliações de desempenho em DTN, utilizando virtualização. Neste capítulo é apresentada o projeto de experimentos utilizado para verificação da solução proposta.

A implementação do ambiente de virtualização VDT pode ser obtida em: `https://github.com/danielcbit/vdt`.

4.1 Cenário de Avaliação

Para avaliação da solução proposta foram utilizadas coletas de informações de mobilidade e conectividade do ambiente de testes denominado DOME (*Diverse Outdoor Mobile Environment*) e sua rede veicular DiselNet, da universidade de Massachusetts Amherst [17].

Os traços de mobilidade e conectividade, coletados ao longo de vários anos foram disponibilizados no repositório de *traces* CRAWDAD [58] para uso por outros pesquisadores em redes em seus trabalhos. Devido às características das frequentes desconexões e comunicação oportunista, diversos estudos em DTN utilizando os da-

dos coletados neste ambiente de teste já foram publicados [16, 59, 60], o que reforça nossa confiança na utilização dos mesmos.

Dentre as informações disponibilizadas nos traces do DOME, foram utilizadas as seguintes:

- gps_logs: Contém informações de posicionamento obtidas por GPS de cada ônibus e para cada dia de coleta.
- mobile-mobile: Contém eventos de contato entre nós móveis (ônibus) para cada dia durante o período de coletas. Cada linha inclui informações sobre o horário do contato, a quantidade de informações transferidas, duração do contato e a posição em que o contato ocorreu.

Apesar de os arquivos mobile-mobile possuírem informações de onde os contatos ocorreram, isto ainda não é suficiente para implementar as características de mobilidade referentes ao posicionamento dos nós a cada instante de tempo. Por isso um programa analisador dos arquivos gps_log foi implementado para converter as informações presentes nesse arquivo em uma saída compatível com o programa GPXImporter da ferramenta BonnMotion [61], que permite criar e analisar cenários de mobilidade em redes.

A partir do BonnMotion, foi possível converter a entrada com as informações de localização dos nós do ambiente de teste, proveniente dos arquivos em gps_log disponibilidados pelo DOME em uma entrada compatível com o NS-3 para implementação da mobilidade dos nós do ambiente virtualizado.

4.2 Metodologia do Ambiente de Testes Virtualizado

4.2.1 Mobilidade

Como dito anteriormente, este trabalho utiliza os registros de mobilidade do projeto DOME. Os arquivos de mobilidade disponibilizados são arquivos de texto organizados pelo ID de cada ônibus que fez parte do experimento e dentro destas, os dias de quando foram realizadas as coletas de posicionamento.

Utilizamos os registros de posição de cada ônibus do dia 23/10/2007 presente na pasta gps_logs, pois esta data é a que apresenta a maior quantidade de informações de posição contíguas dentre as datas disponíveis, para a maioria dos ônibus.

Grande parte dos registros de mobilidade possuem buracos, ou seja, a falta de informações de localização ao longo de um intervalo de tempo, provavelmente causada por mau funcionamento do equipamento de localização, pela falta de sinal de GPS na localidade, ou por desligamentos. Desta forma procuramos maximizar os eventos de contato entre os nós da rede virtualizada selecionando o dia em que os registros de posicionamento dos ônibus tivessem a maior quantidade de pontos contínuos.

Escolhido o dia do qual os registro de posição seriam coletados, a próxima tarefa foi de selecionar um intervalo de hora que seria utilizados nos experimentos. Para isso, utilizamos as informações de log da pasta mobile-mobile que contabilizaram os contatos ocorridos a cada dia entre os ônibus.

O intervalo escolhido procurou maximizar a ocorrência de contatos para que estes então pudessem ser avaliados. Mesmo assim, o intervalo escolhido, de 13:20 às 13:50, possui apenas o registro de sete contatos válidos.

Por contato válido, definimos como sendo os contatos em que os ônibus possuíssem informações de posicionamento registradas no horário apresentado no registro de

contatos mobile_mobile do DOME.

Existiam mais dois contatos, porém um dos ônibus em questão não possuía informações de posicionamento no intervalo de tempo selecionado.

Os registros de posicionamento por GPS dos ônibus foram então convertidos em um padrão de armazenamento de informações geográfias, o GPX (GPS Exchange File). Uma representação gráfica desta etapa pode ser vista na esquematização da figura 4.1. Esse passo foi necessário para atingir dois objetivos:

1. Visualização dos dados: Para que os dados de GPS fossem visualizados em ferramentas comumente utilizada para esse fim, os dados deveriam ser então organizados em um padrão.

2. Entrada do modelo de mobilidade: Para conversão dos dados em entradas válidas para o modelo de mobilidade usado no NS3 foi necessário que estes primeiro estivesse no padrão citado anteriormente para que então pudessem ser transformados.

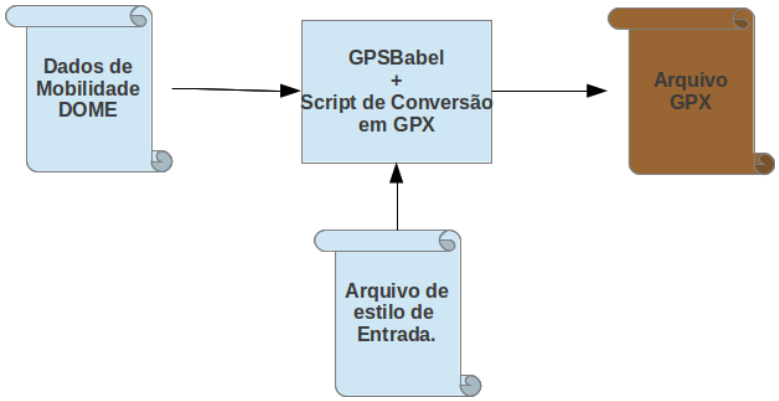


Figura 4.1: Primeira etapa do processo de conversão das informações de mobilidade para o formato NS2.

Para o processo de transformação, foi utilizada a ferramenta de código aberto GPSBabel [62]. Esta ferramenta é amplamente utilizada para realizar a conversão,

alteração e correção de informações geográficas entre os mais variados tipos de formatos e tipos de arquivo. Em especial, foi utilizado o módulo onde, dado um arquivo de estilo que descreve como os dados estão organizados, estes podem ser importados pela ferramenta e então convertidos e salvos em qualquer padrão suportado. Utilizamos um script escrito em Perl para automatização das transformações. O script pode ser visto no Apêndice B.1.

Os arquivos resultantes foram inspecionados com o auxílio da ferramenta GPXViewer [63]. Esta ferramenta também de código livre, permite, além de visualizar as coordenadas e horários do arquivo GPX, a obtenção de informações de velocidade ao longo do tempo e uma visualização do caminho percorrido, como pode ser visto na Figura 4.2.

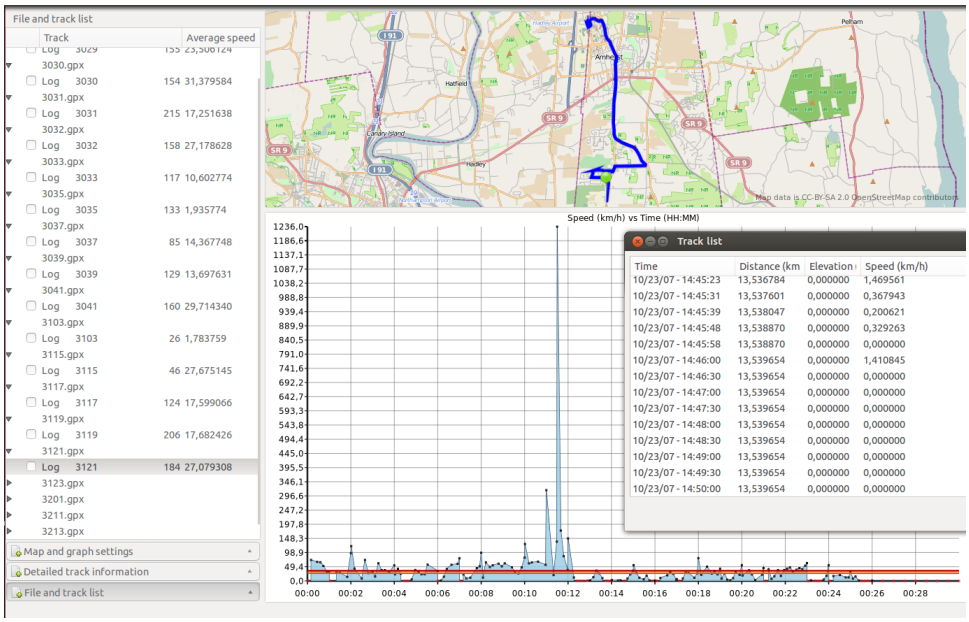


Figura 4.2: GPX Viewer com várias trilhas abertas.

Com os arquivos GPX criados, o próximo passo foi extrair somente o intervalo de horários pretendido para realização do experimento, como pode ser visto na Figura 4.3. Este processo foi feito com a ajuda novamente do GPSBabel, com o seguinte comando:

```
gpsbabel -t -i gpx -f ARQUIVO_GPX.gpx \
-x track,start=200710241320,stop=200710231350 \
-o gpx -F ARQUIVO_FILTRADO.gpx
```

As opções *start* e *stop* definem o início e o fim do intervalo de tempo a ser extraído no formato YYYYMMDDhhmmss, ou seja, ano, mês, dia, hora, minuto e segundo.

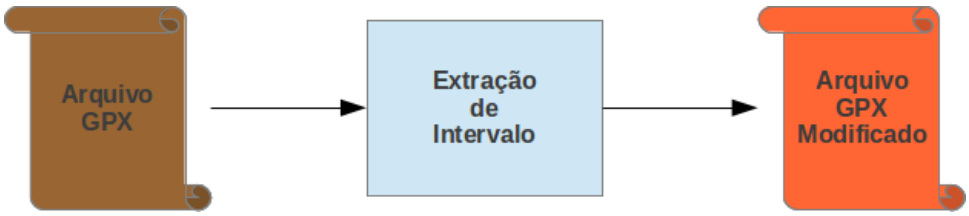


Figura 4.3: Segunda etapa do processo de conversão.

Após rodar o comando para todos os arquivos, apenas 20 deles possuíam algum ponto no intervalo definido. Desses 20, três foram removidos pois, possuíam um número insuficiente de pontos (apenas três minutos de pontos) ou quase não se moviam por estarem em um estacionamento.

Os arquivos estavam então prontos para serem visualizados. Para tal, escolhemos a ferramenta Google Earth para agregá-los em uma única visão e testar se suas movimentações estavam corretas (na maior parte sobre pistas). O resultado da agregação de todos estes pontos pode ser visto na Figura 4.4.

Na visualização das trilhas (disponíveis em `https://dl.dropbox.com/u/6994819/mobilidade_nos_diselnet.kmz`) podemos notar os contatos ocorrendo através do recursos de animação presente no *Google Earth*. Como além do horário é registrada a localização dos nós, pudemos fazer a primeira a análise dos contatos de forma visual, identificando quais ônibus cruzavam entre si e em que horário.

Além de termos corroborado os horários de encontro dos registros originais de contato existentes no DOME, observamos a existência de novos contatos. Atribuímos

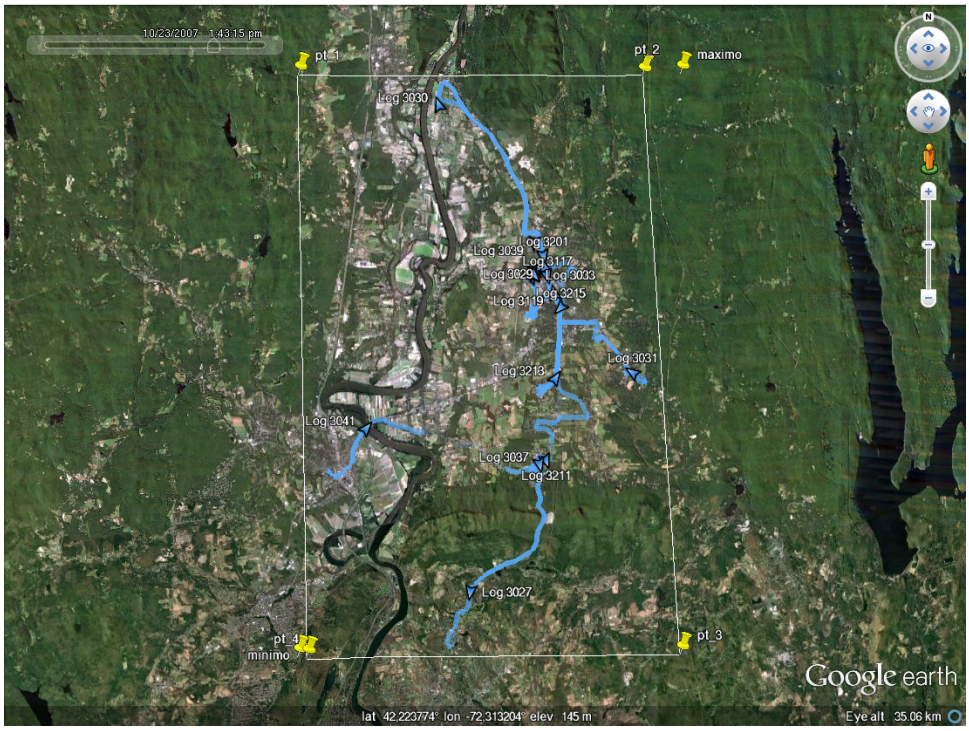


Figura 4.4: Trilhas de GPS obtidas após conversão.

este comportamento às imperfeições nos equipamentos utilizados nos testes originais que não registraram o ocorrido, ou a condições adversas causadas pelo trânsito na cidade.

Por fim, para finalmente converter os arquivos GPX em logs de mobilidade a serem utilizados pelo NS-3, precisamos primeiro modificar o registro *bounds* em todos os arquivos GPX. Este registro armazena as coordenadas X e Y máximas e mínimas entre os pontos existentes que contém todos os pontos e assim define a área em que estes pontos estão contidos.

A modificação foi, necessária pois este registo é utilizado pelo BonnMotion para definir a área de simulação no NS-3. Como todos os arquivos GPX possuem áreas e tamanhos diferentes, definimos um polígono 4.4 que envolvesse todos as trilhas e deste polígono, extraímos os valores de X e Y máximos e mínimos e gravamos em todos os arquivos GPX. A ferramenta BonnMotion, através dos módulos GPXImport

and NSFile, importam o arquivo GPX para o formato padrão do BonnMotion e depois o exporta no formato utilizado pelo NS-3, como mostrado na esquematização da Figura 4.5.

Figura 4.5: Terceira etapa do processo de conversão.

Com os arquivos de mobilidade prontos para serem utilizados no NS-3 nos deparamos com uma limitação de implementação do modelo de mobilidade "ns2-mobility-helper"que implementa a leitura de logs de mobilidade no formato NS2 e os utiliza para definir a mobilidade dos nós.

Na implementação do modelo definiu-se se apenas um único arquivo de log contendo todas as informações de mobilidade dos nós seria utilizado, porém possuíamos arquivos distintos para cada nó (porem todos relativos a mesma área de simulação).

A solução utilizada foi adaptar o modelo para a leitura de apenas um arquivo de mobilidade por nó. Esta adaptação gerou um *patch* que foi aplicado ao código fonte do NS-3 e está descrito no Apêndice C.1.

4.2.2 Emulação canal sem-fio

Para a emulação do canal sem fio procuramos reproduzir o ambiente e as características da rede utilizadas pelo projeto DOME.

No DOME, os nós presentes em cada ônibus possuíam duas interfaces de redes: uma cabeada, ligada a um ponto de acesso sem-fio usando o padrão 11g que fornecia acesso à Internet para os passageiros e era usado para comunicação com outros ônibus; e uma interface de rede sem-fio USB do padrão 11b usada também para comunicação com outros ônibus e à *Throwboxes*, nós sem-fio espalhados em vários

pontos da cidade que atuavam como pontos de troca temporária de informações entre ônibus. Devido à utilização de interfaces 11b definimos que este padrão seria o utilizado como camada física dos enlaces entre os nós virtualizados.

Assumimos que a existência de duas interfaces de redes foi utilizada para simplificar a conexão entre nós móveis, removendo a complexidade da criação de uma rede totalmente ad-hoc entre os mesmos e a utilização concomitante da rede pelos usuários dos ônibus para acesso à internet. Além disso os experimentos executados no DOME utilizavam sempre a interface de rede sem-fio para a realização dos testes de vazão.

Para facilitar a modelagem da camada MAC de alto nível da rede no NS-3, utilizamos o modelo "AdhocWifiMac". Este modelo, apesar do nome, apenas não implementa a geração de *beacons*, varreduras de rede ou associação entre os nós, tal qual seria feita em uma rede infraestruturada.

Esta decisão teve como base o trabalho [59], onde foram analisados os registros e o comportamento em contatos ônibus-ônibus e constatou-se a ocorrência de diversos contatos direcionais, onde ambos os nós participantes, ganhavam ou perdiam acesso ao canal compartilhado.

Também desabilitamos o uso de QoS na camada MAC e definimos o algoritimo de controle de taxa de velocidade da camada baixa do MAC como constante.

4.2.2.1 Modelos de propagação e atraso

Quatro modelos determinísticos básicos estão disponíveis para uso no NS-3, são eles: Log-Distância, Três Log-Distância, Friis e Two-Ray Ground.

Estes modelos, como vistos na Seção 2.5 definem a potência recebida pelo nó de forma determinística em relação a distância d entre os nós.

Com base em estudos feitos em redes veiculares *VANETs* [45, 64] adotamos o modelo de propagação Log-Distancia para computação do sinal recebido pelos nós

com base nos efeitos de propagação em larga escala, e agregado a este, utilizamos o modelo *Nakagami* para contabilização dos efeitos de propagação em pequena escala, variáveis estocasticamente no tempo.

Devido a inexistência de informações do local por onde os ônibus trafegavam, a determinação correta dos melhores parâmetros que descrevessem mais fielmente o ambiente estudado foi impossível, por isso, recorremos às literaturas citadas para a determinação dos parâmetros comumente empregados para realização de estudos em redes veiculares.

Modelamos a rede sem fio com uma área de cobertura de até 150m por nó, distância característica da interface de rede sem fio 802.11b utilizada no DOME durante o período do qual extraímos as informações de mobilidade e de contatos.

Para o atraso, definimos o modelo de propagação em velocidade constante como modelo a ser utilizado. Neste modelo, a velocidade de propagação é constante, dada em relação a velocidade da luz no vácuo, $3 \times 10^8 m/s$.

A utilização modelos de propagação mais realísticos vem da necessidade de se contabilizar os diferentes efeitos comumente encontrados em redes sem fio, como visto na Seção 2.4. A exemplo, dois ônibus podem sofrer interferências na sua comunicação devido a problemas de terminal exposto ou terminal escondido, ao longo de uma via, ou o canal de comunicação pode sofrer os efeitos da assimetria de links.

4.2.3 Ambiente de Virtualização

O hardware de virtualização utilizado neste trabalho consistiu de uma máquina Dell Precision T3500 com processador Intel Xeon W3530 de quatro núcleos, 8 threads e suporte a VT-x e VT-d, 24 GB RAM DDR3 1333 MHz, 500 GB HD e placa de vídeo Nvidia Quadro FX 580, rodando Ubuntu 12.04 x86_64 como sis-

tema operacional host.

Cada nó virtualizado conectava-se ao emulador de canal através de interfaces ponte e interfaces de rede virtuais TAP, como pode ser visto na Figura 4.6.

Uma segunda interface de gerência foi adicionada às maquinas virtuais para que os registros de contatos gerados fossem recuperados e armazenados entre cada execução do experimento e para sincronização dos relógios das maquinas virtuais com o host.

Apesar do suporte a nós virtuais estáticos no VDT, este trabalho utilizou apenas nós móveis durantes os experimentos, para virtualização dos contatos entre ônibus, como os ocorridos no DOME.

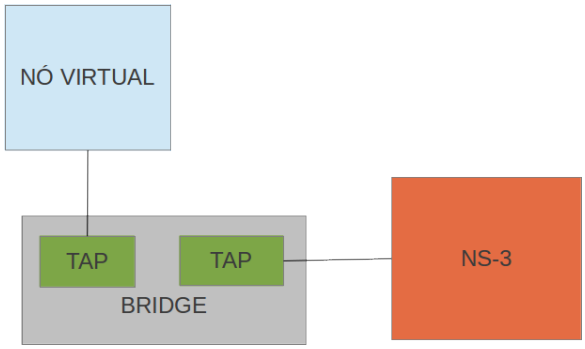


Figura 4.6: Esquematização da conexão entre máquinas virtuais e o emulador de canal.

4.2.4 Coleta de dados

No DOME os dados foram coletados através de uma aplicação escrita em Java que rodava nos nós móveis. A cada ocorrência de contato com outros nós móveis ou com *Throwboxes*, realizava-se um teste de vazão. Neste teste eram registrados: os nós que entraram em contato, a quantidade de bytes transferidos e o tempo de duração de cada contato, além da data, hora e local de ocorrência do contato, em coordenadas de GPS.

Este registro gerado era então armazenado e depois enviado a um repositório central, quando o nó conseguia conectar-se a internet.

Para coleta de dados deste trabalho, utilizamos inicialmente o mesmo aplicativo, porém, devido a um problema de implementação que resultava em um comportamento imprevisto quando algumas condições da rede aconteciam, descoberto durante os experimentos iniciais deste trabalho e melhor explicado na Seção 5.1.1, detectamos que nem todos os contatos que ocorriam eram contabilizados durante a experimentação do ambiente virtualizado, por isso foi necessária a reescrita do código de forma a resolver este problema.

O novo código, como consta no Apêndice D.1, mantém as mesmas características do original porém foi reescrito utilizando a linguagem de programação Python que oferecia os mesmos recursos do Java no tratamento de conexões e provia meios para contornar o problema detectado. Como benefícios extra, tornamos o aplicativo melhor adaptando para a execução nos mais diversos tipos de sistemas.

Além dos dados gerados pelo aplicativo de teste de vazão, utilizamos o recurso no NS-3 de criação de arquivos de animação para registro visual da troca de informações e mobilidade dos nós. Assim poderíamos, após uma execução do experimento, identificar possíveis comportamentos observados nos registros de contato gerados.

Os dados coletados a cada contato, durante o experimento no ambiente virtualizado foram exatamente os mesmos do realizado no DOME, a saber:

- Número de Bytes transferidos
- Tempo de contato

Os resultados obtidos correspondem a um intervalo de confiança de 90% obtido através da execução do experimento em 30 replicações, estatisticamente independentes.

Cada rodada é identificada por um número de execução que serve de parâmetro

de entrada para o pseudo gerador de números aleatório utilizado no NS-3. Assim asseguramos que cada experimento é estatisticamente diferente dos demais.

Para o cálculo do intervalo de confiança, foi utilizada a a fórmula do erro de estimação, com base na tabela t-student. Onde t é igual a 1,6973 para 90%, e n é o número de amostras.

$$e = t\frac{s}{\sqrt{n}} \tag{4.1}$$

Um resumo do projeto de experimentos pode ser visto no Apêndice A.

Capítulo 5

Apresentação e análise dos resultados

Este capítulo é dedicado à apresentação e à análise dos resultados obtidos conforme o projeto de experimento proposto no Capítulo 4.

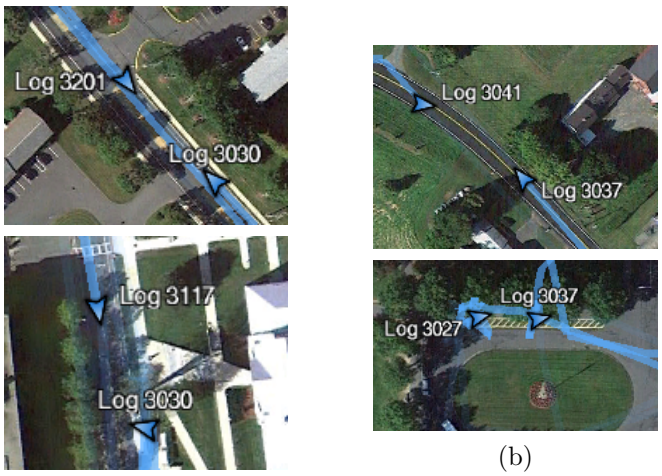
5.1 Mobilidade

A correta reprodução da mobilidade é um dos fatores mais importantes no estudo de uma DTN. Nesta seção apresentamos os resultados referentes a análise dos registros de mobilidade obtidos a partir do DOME de forma gráfica e a representação da mobilidade no VDT.

5.1.1 Análise gráfica dos registros de mobilidade

Com base nos arquivos GPX gerados foi possível validar visualmente a ocorrência dos contatos listados na pasta "mobile-mobile"dos registros do DOME.

Ao todo, sete contatos são listados no intervalo de tempo escolhido para estudo, 13:20 às 13:50. Identificamos que todos os contatos ocorreram na hora exata



(a) (b)

Figura 5.1: Novos contatos identificados.

registrada nos arquivos mobile-mobile.

Além disso identificamos a ocorrência de onze novos contatos que não foram listados pelo DOME, assim como a ocorrência de alguns contatos prolongados dentre os listados, como pode ser visto na Figura 5.1.1. Em (a) podemos ver novos contatos referentes ao nó 3030 e em (b) podemos ver novos contatos referentes ao nó 3037.

Por novos contatos, definimos aqueles onde os ônibus, com base nas informações de posicionamento coletadas, passaram entre si por uma posição válida (ruas e avenidas) e que não estão parados na garagem. E por contatos prologados, definimos aqueles que geralmente ocorriam quando dois ônibus andavam paralelamente em uma mesma avenida durante um intervalo de tempo.

De acordo com Zhang et al. [59], vários fatores podem explicar a ausência destes contatos nos registros originais do DOME, como por exemplo a a velocidade dos ônibus quando estes trafegam em vias expressas pode não proporcionar tempo suficiente para o estabelecimento de uma conexão entre as interfaces de rede e o inicio de uma sessão TCP. Outro fator é que a ocorrência de defeitos nos hardware não era algo tão incomum de ocorrer.

Um terceiro motivo foi descoberto após analisar o programa de medição de vazão originalmente utilizado para coletar os logs de contatos entre ônibus. Descobrimos que por um problema de implementação e uma característica da linguagem utilizada, o envio de dados durante um encontro poderia ocasionar a interrupção desta *thread* quado o nó de destino dos dados saia da área de cobertura do outro, durante uma sessão TCP já aberta. O nó responsável pelo envio dos dados tinha seu *buffer* TCP saturado, bloqueado assim a *thread* de envio e impedindo que o teste entre novos contatos posteriores ocorresse até que sistema desistisse de tentar enviar os dados ou quando o nó de destino voltava para a área de cobertura do primeiro.

5.1.2 Mobilidade no VDT

A mobilidades dos nós no ambiente virtualizado, como descrito na metodologia, corresponde a uma projeção da posição de cada nó em relação ao globo terrestre, em um plano com dimensões iguais às dimensões máximas que contem todas as trajetórias observadas nos registros de mobilidade.

Essa projeção transforma latitudes e longitudes em coordenadas x e y dentro do emulador de canal, que dizem a posição em determinado instante de um nó em relação a outro.

Na Figura 5.2 é possível visualizar uma agregação das trajetórias de todos os nós com o uso da ferramenta, NetAnim, presente no NS-3. Esta trajetórias são um retrato fiel dos caminhos percorridos por cada nó dentro da área de simulação.

A Figura 5.1.2 mostra uma comparação entre as trajetórias de dois nós observadas pelo no ambiente de virtualização (a) e nos registro de mobilidades, aqui vistos com o uso do *Google Earth* (b). Verificamos que as rotas apresentam um casamento perfeito.

Porém, é importante ressaltar que a qualidade da mobilidade é diretamente lig-

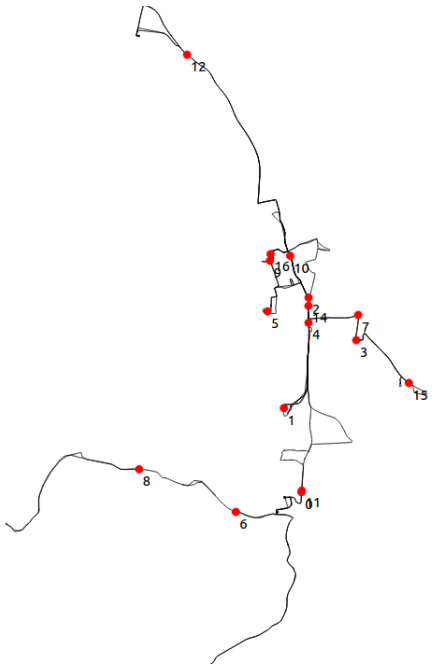
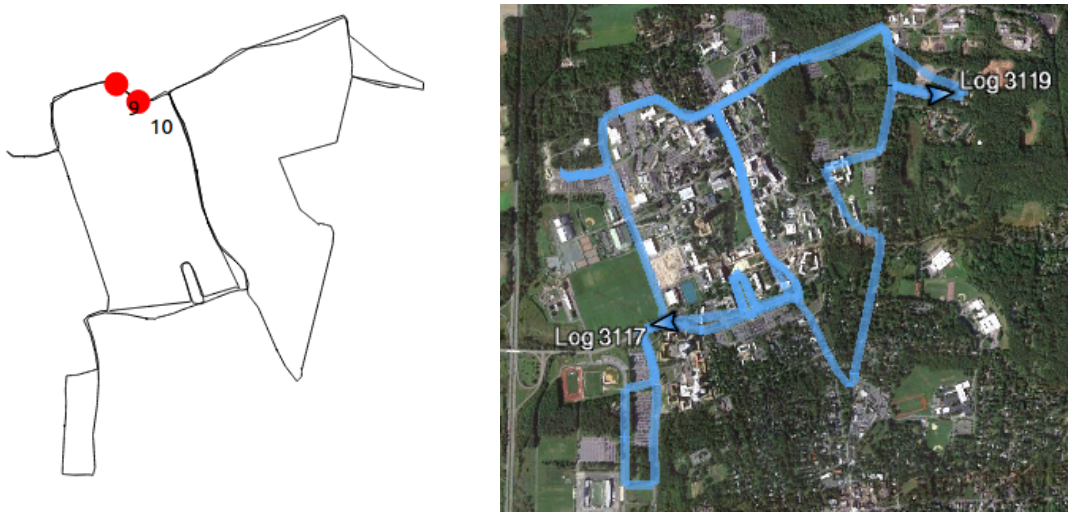


Figura 5.2: Trajetórias agregadas dos nós virtuais.

ada a qualidade dos registros de mobilidade originais. Quanto maior a densidade de pontos de GPS por intervalo de tempo, mais precisas serão as trajetórias representadas durante a virtualização. Problemas como grandes intervalos de tempo sem pontos de GPS podem causar um comportamento anômalo quando virtualizados. O tratamento do que deve ser feito nesses casos depende principalmente do tipo de pesquisa a ser realizada.



(a) (b)

Figura 5.3: Trajetórias dos nós 3117 e 3119.

5.2 Conectividade

Para análise da conectividade entre os nós virtualizados coletamos resultados dos testes de vazão como definidos na metodologia. O que desejou-se mostrar com esta análise é o quão distante estavam os dados obtidos no ambiente virtualizado do observado no experimento real.

Dos dados obtidos, selecionamos o contato entre os nós 3119 e 3117. Eles aparecem como um dos contatos válidos no registro de contatos original do DOME. Para comparação, calculamos o intervalo de 90% de confiança para valores de bytes transferidos e tempo de contato. Com base nesses resultados calculamos separadamente a vazão.

Para estas duas variáveis, consideramos todos os registros de contatos entre os nós citados que ocorressem no intervalo de até 15 segundos entre si, como sendo pertencentes a um único contato e os unimos, gerando assim um único contato. Como visto na metodologia, o compartilhamento do canal usando durante a comunicação ocasionava a ocorrência de múltiplos contatos direcionais a medida que os

nós ganhavam ou perdiam acesso ao canal o que disparava *timeouts* no protocolo TCP/IP e interrompia o teste de vazão. Porém, por ainda estarem dentro do raio de comunicação o teste de vazão iniciava novamente.

Nas Figuras 5.4, 5.5 e 5.6, podemos ver uma comparação de respectivamente, a quantidade média de bytes recebidos durante o contato, o tempo de contato médio de contato e a vazão calculada. Estes respectivos valores medidos no VDT estão sendo comparados ao que foi observado nos logs do DOME, para o mesmo contado.

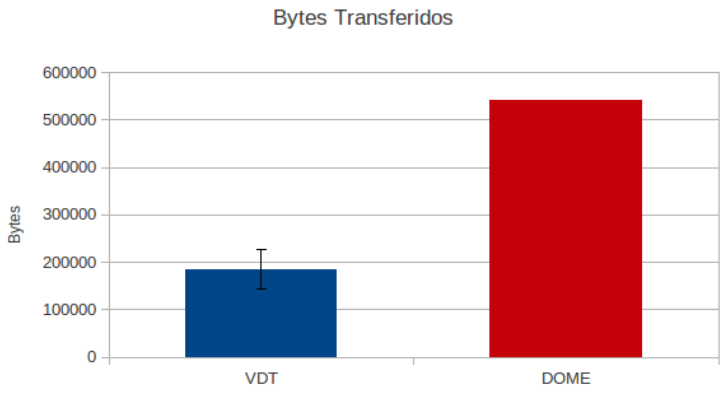


Figura 5.4: Comparação de bytes transferidos.

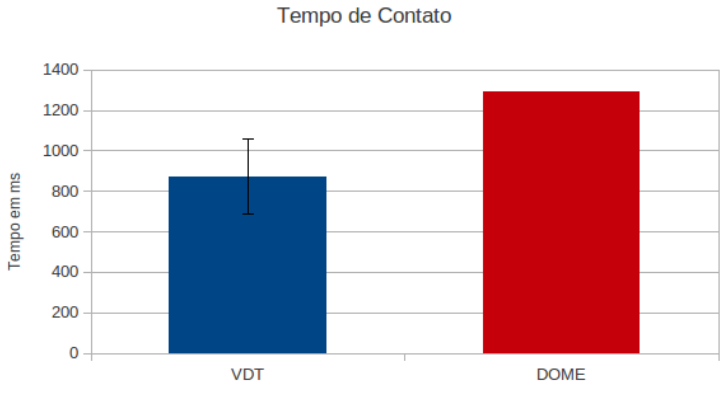


Figura 5.5: Comparação de tempo de contato.

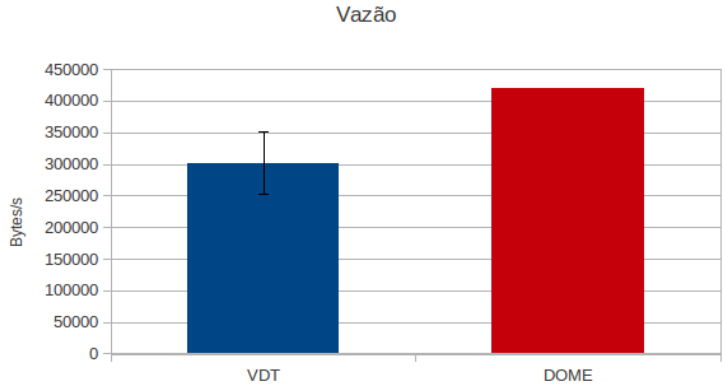


Figura 5.6: Comparação de Vazão.

Comparando-se os resultados na Tabela 5.1, verificamos que, para a vazão e o tempo de contato, o erro observado é de aproximadamente 30%, quando consideramos o valor médio das amostras realizadas em comparação com o valor observado no DOME.

Para a quantidade de bytes transferidos, este valor já sobe para 66%. Esta diferença pode ser atribuída, entre outros fatores, à imperfeições na modelagem do ambiente, dada a falta de informações mais completas a respeito da localidade onde os dados originais foram coletados.

	VDT	DOME	Erro (%)
Numero de Bytes	185.112	541.792	66,00
Tempo de contato (ms)	874	1.292	32,35
Vazão (KB/s)	294	409	28,00

Tabela 5.1: Avaliação de erro do ambiente virtualizado

Capítulo 6

Conclusões e trabalhos futuros

Um dos métodos mais utilizados para estudo de DTNs é a simulação, já que os custos envolvidos para a construção de redes reais, principalmente as de grande porte com centenas de nós, ainda e um fator limitante para diversas universidades e centros de pesquisa.

A limitação no número de simuladores existentes que suportam este tipo de rede e as limitações apontadas neste trabalho que afetam esta técnica de avaliação de desempenho motivam a comunidade científica na busca de alternativas viáveis do ponto de vista econômico e confiáveis do ponto de vista científico.

Como visto, com base no sucesso do uso de virtualização para estudo de outros tipos de rede, diversos trabalhos tem apostado na virtualização como uma alternativa eficiente para a realização de estudos em redes DTN o que mostra que a pesquisa nesta área é de grande interesse da comunidade científica.

Este trabalho apresentou uma nova forma de se realizar avaliações de desempenho em DTNs utilizando virtualização, onde nos apoiamos nos acertos de trabalhos anteriores e propomos melhorias para os problemas levantados.

Vimos que para um estudo mais completo deste tipo de rede é necessário que se leve em consideração fatores como as características inerentemente estocásticas de

canais sem fio e as complexidades impostas pela utilização de sistemas operacionais e pilhas de protocolos completas, de forma que os resultados obtidos a partir destes estudos possam ser transportados para uma rede real com maior facilidade.

Com base nos resultados obtidos, do ponto de vista da mobilidade de nós e da ocorrência de contatos, verificamos que o ambiente de testes virtualizados conseguiu reproduzir de forma correta a ocorrência dos mesmos já que os contatos observados no DOME puderam ser reproduzidos no ambiente.

Verificamos ainda que foi possível validar a ocorrência dos novos contatos observados na análise gráfica dos registros de mobilidade e nos registros de mobilidade obtidos a partir do ambiente virtualizado.

Porém, do ponto de vista da conectividade entre os nós, verificamos que há uma grande variação dos dados coletados. No cenário específico utilizado neste trabalho, que se assemelha a uma VANET, estas variações pode ser atribuídas a diversos fatores apresentados na literaturara, como a velocidade relativa entre os nós, a existência de outros corpos, principalmente metálicos que podem influenciar na propagação do sinal e a constante variação das características do local estudado, o que tornam difícil uma representação correta de todos os fatores que influenciam a comunicação [64, 65].

Com base nestas observações, concluímos que é possível realizar estudos e avaliações de desempenho em redes DTN de forma totalmente virtualizada, o que valida o objetivo (Cap. 1.4) deste trabalho.

Porém, com base no comportamento de conectividade observado verificamos que para cada ambiente, ao ser virtualizado, um estudo detalhado faz-se necessário, de forma a levantar todos os possíveis fatores que interfiram nos resultados obtidos. Portanto, este trabalho não esgota todas as possibilidades de investigação em relação a este tema e propomos como trabalho futuro uma investigação mais detalhista a respeito.

Como contribuições deste trabalho estão a implementação de um ambiente de avaliação de redes tolerantes a atrasos e desconexões baseado em virtualização, a implementação de mobilidade para os nós virtualizados com base em registros reais de mobilidade que permitiram verificar a ocorrência de todos os contatos observados no experimento real, além da ocorrência de novos contatos que não haviam sido registrados.

Também como contribuição deste trabalho temos a identificação do problema no aplicativo de teste de vazão originalmente utilizado no DOME.

Dado um dos objetivos da metodologia proposta, de reutilizar os mesmo conjuntos de código usando em ambientes de teste reais, tal problema só foi possível ser percebido e identificado pois o mesmo código usado no experimento real foi utilizado sobre a infraestrutura provida por um sistema operacional e uma pilha de protocolo, do mesmo tipo que os utilizados por um sistema real. Em um ambiente estritamente simulado, tal comportamento poderia ser perdido e deixar de influenciar os dados coletados da forma como fora observado.

6.1 Trabalhos Futuros

Como trabalhos futuros propomos a realização de estudos mais aprofundados quanto a capacidade máxima de nós a serem virtualizados por *host* de modo que o impacto da virtualização não interfira nos resultados, a utilização de recursos de simulação/emulação distribuída entre diversos nós rodando NS-3 com suporte ao *MPI - Message Passing Interface*, sistema de troca de mensagens padronizado e portável para o uso de computação paralela e distribuída.

Adicionalmente propomos como trabalhos futuros a investigação da utilização de ambientes de avaliação de desempenho virtualizados sobre sistemas de computação em nuvem como o Amazon EC2 (`http://aws.amazon.com/ec2/`). Sistema de Com-

putação em nuvem elástica que faz o uso de maquinas virtuais hospedadas em servidores remotos.

Assim pesquisadores poderiam ter de forma instantânea acesso a uma parque computacional de grande porte para executar seus experimentos sem a necessidade de realizar enormes gastos com hardware que ficarão obsoletos em poucos anos, gerando assim economia e permitindo a realização de estudos de grande porte de forma simples e rápida.

Finalmente, propomos como trabalho futuro a implementação de modelos de propagação estocásticos, como os baseados em traçados de raios (*Raytracing*), em sistemas com arquiteturas de processamento paralelo, como o Parallela, introduzido pela empresa Adapteva (`http://www.adapteva.com/introduction/`).

Este tipo de sistema implementam processadores específicos para realização de processamentos paralelos e tem como grande atrativo o baixo custo e o pequeno consumo de energia e serviriam como suporte para desafogar o processador principal da tarefa de computação das diversas potências recebidas em vários pontos entre dois nós, assim mais nós poderiam ser adicionados na virtualização sem penalizar o desempenho ou a validade dos resultados.

Referências Bibliográficas

[1] J. Liu, S. Mann, N. Van Vorst, and K. Hellman. An Open and Scalable Emulation Infrastructure for Large-Scale Real-Time Network Simulations. *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, May 2007.

[2] A Zimmermann, M Gunes, M Wenig, U Meis, and J Ritzerfeld. How to study wireless mesh networks: A hybrid testbed approach. *in Proc. of AINA '07, Niagara Falls, Canada, May*, 2007.

[3] R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling.* Wiley - Interscience, New York, NY, 1991.

[4] Ari Keränen. The ONE simulator for DTN protocol evaluation. In *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*, Gent, BELGIUM, 2009. Icst.

[5] Johannes Morgenroth, Sebastian Schildt, and Lars Wolf. HYDRA: virtualized distributed testbed for DTN simulations. *Proceedings of the fifth ACM international workshop on Wireless network testbeds, experimental evaluation and characterization - WiNTECH '10*, 2010.

[6] Marcos Aurelio Pchek Laureano and Carlos Alberto Maziero. Virtualização: Conceitos e Aplicações em Segurança. In *Livro-Texto de Minicursos*. 2008.

[7] Global Environment for Network Innovations - GENI.

[8] A Bavier, N Feamster, and M Huang. In VINI veritas: realistic and controlled network experimentation. *ACM SIGCOMM ...*, 2006.

[9] Mike Hibler, Robert Ricci, Leigh Stoller, and Jonathon Duerig. Large-scale virtualization in the emulab network testbed. *USENIX 2008 Annual*, 2008.

[10] Dongwoon Hahn, Ginnah Lee, Youngil Kim, Brenton Walker, Matt Beecher, and Padma Mundur. DTN experiments on the virtual meshtest testbed. In *Proceedings of the 5th ACM workshop on Challenged networks - CHANTS '10*, New York, New York, USA, 2010. ACM Press.

[11] Ha Dang and Hongyi Wu. Clustering and cluster-based routing protocol for delay-tolerant mobile networks. *IEEE Transactions on Wireless Communications*, 9(6), June 2010.

[12] Kevin Fall, Stephen Farrell, and Jörg Ott. DTN Research Group.

[13] Gregory Smith, Anmol Chaturvedi, Arunesh Mishra, and Suman Banerjee. Wireless virtualization on commodity 802.11 hardware. *Proceedings of the the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization - WinTECH '07*, 2007.

[14] N.M.M.K. Chowdhury and Raouf Boutaba. Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, 47(7), 2009.

[15] Thomas Staub, Reto Gantenbein, and Torsten Braun. *VirtualMesh: an emulation framework for wireless mesh and ad hoc networks in OMNeT++*. PhD thesis, Gent, BELGIUM, July 2010.

[16] Youngil Kim, Keith Taylor, and Carson Dunbar. Reality vs emulation: Running real mobility traces on a mobile wireless testbed. *Proceedings of the 3rd ...*, 2011.

[17] Hamed Soroush, Nilanjan Banerjee, Aruna Balasubramanian, Mark D. Corner, Brian Neil Levine, and Brian Lynn. DOME. In *Proceedings of the 1st ACM International Workshop on Hot Topics of Planet-Scale Mobility Measurements - HotPlanet '09*, New York, New York, USA, 2009. ACM Press.

[18] S Ivanov and A Herms. Experimental validation of the ns-2 wireless model using simulation, emulation, and real network. *Communication in Distributed*, 2, 2007.

[19] Carlo Caini, Rosario Firrincieli, Daniele Lacamera, and Marco Livini. Virtualization Technologies for DTN Testbeds. *Personal Satellite Services*, 2010.

[20] Carina Teixeira de Oliveira. *UMA PROPOSTA DE ROTEAMENTO PROBABILÍSTICO PARA REDES TOLERANTES A ATRASOS E DESCONEXÕES*. PhD thesis, COPPE/UFRJ, 2008.

[21] Ling-Jyh Chen, Chen-hung Yu, Tony Sun, Yung-chih Chen, and Hao-hua Chu. A hybrid routing approach for opportunistic networks. In *Proceedings of the 2006 SIGCOMM workshop on Challenged networks - CHANTS '06*, New York, New York, USA, 2006. ACM Press.

[22] CK YEO. Delay Tolerant Networks (DTN). *prius.ist.osaka-u.ac.jp*, (March), 2003.

[23] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and Weiss H. Delay-Tolerant Networking Architecture, 2007.

[24] João Batista Pinto Neto. *Um Modelo para Previsão do Volume de Contato em Redes Tolerantes a Atrasos e Desconexões: Uma Abordagem Quantitativa.* Dissertação de mestrado, UFAM, 2011.

[25] David Kotz, Calvin Newport, Robert S Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental evaluation of wireless simulation assumptions. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems - MSWiM '04*, New York, New York, USA, 2004. ACM Press.

[26] Alexander Zimmermann, Mesut Günes, Martin Wenig, Jan Ritzerfeld, and Ulrich Meis. Architecture of the hybrid MCG-mesh testbed. In *Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization - WiNTECH '06*, number August, New York, New York, USA, 2006. ACM Press.

[27] Alexander Zimmermann, Daniel Schaffrath, Martin Wenig, Arnd Hannemann, Mesut Gunes, and Sadeq Ali Makram. *Performance Evaluation of a Hybrid Testbed for Wireless Mesh Networks.* IEEE, October 2007.

[28] Simulador de Redes NS-2.

[29] András Varga. OMNeT++ - Simulador de Eventos Discretos.

[30] Scalable Network Technologies, QualNet Network Simulator, 2010.

[31] Alain Cohen. Optimized Network Engineering Tools - OPNET.

[32] T Staub, R Gantenbein, and T Braun. VirtualMesh: an emulation framework for wireless mesh networks in OMNeT++. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, volume 6. ICST

(Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[33] P. Zheng and L.M. Ni. Empower: A cluster architecture supporting network emulation. *Parallel and Distributed Systems, IEEE Transactions on*, 15(7), July 2004.

[34] Carlo Caini, R. Firrincieli, R. Davoli, and Daniele Lacamera. Virtual integrated TCP testbed (VITT). In *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[35] Marta Carbone and Luigi Rizzo. Dummynet revisited. *ACM SIGCOMM Computer Communication Review*, 40(2), 2010.

[36] S Hemminger and Others. Network emulation with NetEm. In *Linux Conf Au*. Citeseer, 2005.

[37] M Carson and D Santay. NIST Net-A Linux-based network emulation tool. *Computer Communication Review*, 33(3), 2003.

[38] K. Fall. Network emulation in the VINT/NS simulator. *Proceedings IEEE International Symposium on Computers and Communications (Cat. No.PR00250)*, 1999.

[39] M. Kropff, T. Krop, M. Hollick, P.S. Mogre, and R. Steinmetz. A Survey on RealWorld and Emulation Testbeds for Mobile Ad hoc Networks. *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006.*, 2006.

[40] Mineo Takai, Jay Martin, and Rajive Bagrodia. Effects of wireless physical layer modeling in mobile ad hoc networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing - MobiHoc '01*, New York, New York, USA, 2001. ACM Press.

[41] Erik Nordström, Per Gunningberg, Christian Rohner, and Oskar Wibling. Evaluating wireless multi-hop networks using a combination of simulation, emulation, and real world experiments. *Proceedings of the 1st international workshop on System evaluation for mobile platforms - MobiEval '07*, 2007.

[42] Theodore S. Rappaport. *Comunicacoes Sem Fio - Principios e Praticas.* Pearson/ Prentice Hall, segunda edition, 2009.

[43] Vinay Sridhara and Stephan Bohacek. *Realistic propagation simulation of urban mesh networks.* Computer Networks, New York, NY, USA, vol. 51, n. 12, p. 3392–3412, Jan. 2007.

[44] Saulo Queiroz. *Avaliação de Roteamento Incremental em Redes em Malha Sem Fio Baseadas em 802.11.* PhD thesis, Universidade Federal do Amazonas, 2009.

[45] Joseph Benin, Michael Nowatkowski, and Henry Owen. Vehicular Network simulation propagation loss model parameter standardization in ns-3 and beyond. *2012 Proceedings of IEEE Southeastcon*, March 2012.

[46] Alexandre Carissimi. Virtualização: da teoria a soluções. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2008*, 2008.

[47] Mendel Rosenblum. The Reincarnation of Virtual Machines. *Queue*, 2(5), July 2004.

[48] M. Tim Jones. Descubra a Máquina Virtual do Kernel Linux, 2007.

[49] Carlos Alberto Maziero. Sistemas Operacionais IX - Máquinas Virtuais. In *Sistemas Operacionais*. 2008.

[50] Nicola Baldo, Gustavo Carneiro, George Riley, Mathieu Lacage, Michele Weigle, Tom Goff, and Ruben Merz. Network Simulator 3 - NS3.

[51] Gautam Bhanage, Ivan Seskar, and Y Zhang. Evaluation of openvz based wireless testbed virtualization. Technical Report 732, 2008.

[52] Ludmila Cherkasova and Diwaker Gupta. When virtual is harder than real: Resource allocation challenges in virtual machine based it environments. *Labs, Tech. Rep. HPL-2007-25*, 2007.

[53] Intel. Intel Virtualization Tecnology.

[54] AMD. AMD virtualization Technology.

[55] OpenWRT Wireless Freedom.

[56] LXC - Linux Containers.

[57] Nicola Baldo, M. Requena-Esteso, J. Núñez Martínez, M. Portolès-Comeras, J. Nin-Guerrero, Paolo Dini, and J. Mangues-Bafalluy. Validation of the IEEE 802.11 MAC model in the ns3 simulator using the EXTREME testbed. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[58] David Kotz, Tristan Henderson, and Ilya Abyzov. {CRAWDAD} data set dartmouth/campus (v. 2004-12-18). Downloaded from http://www.crawdad.org/dartmouth/campus, 2004.

[59] Xiaolan Zhang, Jim K. Kurose, Brian Neil Levine, Don Towsley, and Honggang Zhang. Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing. *Proceedings of the 13th annual ACM international conference on Mobile computing and networking - MobiCom '07*, 2007.

[60] Phani Rohit Mullangi, Lakshmish Ramaswamy, and Raga Sowmya Tummalapenta. DATEM: Towards QoS Aware Event Notification Framework for Delay Tolerant Networks. *2012 IEEE 13th International Conference on Mobile Data Management*, July 2012.

[61] Nils Aschenbruck, Raphael Ernst, Elmar Gerhards-Padilla, and Matthias Schwamborn. BonnMotion: a mobility scenario generation and analysis tool. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, Torremolinos, Malaga, Spain, 2010. ICST.

[62] Olaf Klein Robert Lipe, Ron Parker, Alex Mottram. GPSBabel.

[63] Andrew Harvey. GPX Viewer.

[64] H Hartenstein and KP Laberteaux. A tutorial survey on vehicular ad hoc networks. *Communications Magazine, ...*, (June), 2008.

[65] EM Van Eenennaam. A Survey of Propagation Models used in Vehicular Ad hoc Network (VANET) Research. *Paper written for course Mobile Radio ...*, (Spring), 2008.

Apêndice A

Projeto de Experimento

Variáveis de Resposta: Média de Bytes transferidos, média do tempo de contato em ms, vazão média.

Fatores Fixos:

- Tipo de canal Utilizado

  802.11b

- Modelos de propagação do canal sem fio:

  Log-Distância para contabilização dos efeitos de propagação em larga escala do canal sem fio.

  Nakagami para contabilização dos efeitos de propagação em pequena escala, estocásticos no tempo.

- Distância máxima da comunicação: 150m

- Modelo de mobilidade baseado em traces coletados no ambiente de teste real DOME.

- Intervalo de confiança utilizado: 90%

- Número de execuções: 30

Unidades Experimentais:

- Bytes

- ms

- B/s

Apêndice B

Script de conversão em GPX

B.1 Script de conversão em GPX

O *script* a seguir foi implementado para automatização da conversão dos arquivos de mobilidade do DOME para o formato GPX.

Ele aceita como parâmetros de entrada:

- -v: Modo Verboso
- -t: Tempo máximo entre pontos antes de separar-los em tracks diferentes
- -d: Distância máximo entre pontos antes de separa-los em tracks diferentes
- -s: Arquivo de estilo de entrada alternativo. O padrão e o arquivo dieselnet.style existente no mesmo diretório.
- -h: Ajuda e informações adicionais.

```
#!/usr/bin/perl -w

my $VERSION = '0.3';

use strict;
use Cwd;
use Data::Dumper;
use Getopt::Std;

#---------------------------------------------------------------------------
# default values are set here

my $STYLEFILE = 'dieselnet.style';
my $MAXTIME   = '';  # set this value to use a default for -t (e.g. '3m')
my $MAXDIST   = '';  # set this value to use a default for -d (e.g. '.5k')

#---------------------------------------------------------------------------
# ensure writes to stderr and stdout appear in the correct order
# it's a performance hit, but negligible for this script

$| = 1;
open(STDERR,">&STDOUT");
```

```
#-------------------------------------------------------------------------
# process command line agruments

my $VERBOSE = 0;
my @SRCDIRS = ();

my %options = ();
getopts("hvs:t:d:", \%options);

printUsage() if( $options{'h'} );
$VERBOSE = 1 if( $options{'v'} );

if( $options{'s'} )
{
   $STYLEFILE = $options{'s'};
   die "$STYLEFILE is not a valid style file\n" unless( -f $STYLEFILE );
}

if( $options{'t'} )
{
   $MAXTIME = $options{'t'};
   die "-t requires a gpsbabel time parameter (1h, 5m, etc.)\n" unless( $MAXTIME =~ /^(\d+(\.\d+)?[dhms])$/ );
}

if( $options{'d'} )
{
   $MAXDIST = $options{'d'};
   die "-d requires a gpsbabel distance parameter (1k, 0.5m, etc.)\n" unless( $MAXDIST =~ /^(\d+(\.\d+)?[mk])$/ );
}

#-------------------------------------------------------------------------
# turn the MAXTIME and MAXDIST values into proper gpsbabel filter parameters

if( $MAXTIME ne '' )
{
   $MAXTIME = ',split=' . $MAXTIME;
}

if( $MAXDIST ne '' )
{
   $MAXDIST = ',sdistance=' . $MAXDIST;
}

#-------------------------------------------------------------------------
# now that options have been parsed, check for directories

foreach(@ARGV)
{
   if( ! -d $_ )
   {
      die "$_ is not a valid directory\n";
   }
   s/\/*$/\//;
   push(@SRCDIRS,$_);
}

die "No directories given\n" unless(@SRCDIRS);

#-------------------------------------------------------------------------
# process each directory and exit

foreach(@SRCDIRS)
{
   processDirectory($_);
}

exit 0;
```

```
#-------------------------------------------------------------------------
# usage function
#-------------------------------------------------------------------------

sub printUsage
{
   print <<USAGE;
Usage: raw2gpx.pl [-h] [-v] [-t <time>] [-d <distance>] [-s <style>] directories

   -h         print this help
   -v         be verbose about what the script is doing
   -t         set the maximum time between points before splitting tracks
   -d         set the maximum distance between points before splitting tracks
   -s <file>  use a different style instead of the default dieselnet.style

One or more directories can be given and the files in those directories (no
sub-dirs) with a filename of the form YYYY-MM-DD will be processed. Invalid
or uninteresting data points will be ignored: those for 0,0 lat/long and
those that are out of bounds (>180 or <-180). The date from the filename
will be included in each row of data and a new data file written with the
name YYYY-MM-DD.fixed. Each of those new files will be passed to gpsbabel
to generate a .gpx file according to the style file.

The -t and -d arguments must be of the form required by gpsbabel for the
split and sdistance filter options, respectively. For more information see:
http://www.gpsbabel.org/htmldoc-development/filter_track.html

USAGE

exit 0;
}

#-------------------------------------------------------------------------
# process a single directory
#-------------------------------------------------------------------------

sub processDirectory
{
   my $dir = shift;

   print "Processing directory $dir\n" if( $VERBOSE );

   my @src_files;

   #-------------------------------------------------------------------------
   # change to the directory and get a list of globs, adding the files
   # named like YYYY-MM-DD to our to-do list

   my $cwd = cwd();
   chdir($dir) || die "Can't cd to $dir\n";
   foreach( sort glob('*') )
   {
#print "dir: $dir    glob: $_    both: $dir$_    -f: " . (-f "$dir$_") . "\n";
      push(@src_files,$_) if( /^\d\d\d\d-\d\d-\d\d$/ && -f $_ );
#print Dumper(\@src_files);
   }
   chdir($cwd);

my $dirnum = '';
   if( $dir =~ /(\d{4})\/$/ )
   {
      $dirnum = $1;
   }

   foreach my $srcfile (@src_files)
   {
```

```
#print "src_file: $srcfile\n";
      my $date = $srcfile;
      $srcfile = $dir . $srcfile;
      print "Converting $srcfile\n" if( $VERBOSE );

   #-----------------------------------------------------------------
   # the new files created will simply be the source file names with
   # appended extensions

      my $fixfile = $srcfile . '.fixed';
      my $gpxfile = $srcfile . '.gpx';

      my $fh_in;
      my $fh_out;
      open($fh_in,"<$srcfile") || die "Can't open $srcfile\n";
      open($fh_out,">$fixfile") || die "Can't open $fixfile\n";

      my $linenum = 0;

      while(<$fh_in>)
      {
         chomp;
         $linenum++;
         my $line = $_;

   #-----------------------------------------------------------------
   # try to match each line, and skip the ones we don't want
   # abort if we encounter a line that doesn't match the expected format

         if( $line =~ /^(\d\d:\d\d:\d\d) (-?\d+\.\d+(E\d+)?) (-?\d+\.\d+(E\d+)?)$/ )
         {
            my $time = $1;
            my $lat  = $2 + 0;
            my $long = $4 + 0;
            if( $lat == 0 || $long == 0 || $lat < -180 || $lat > 180 || $long < -180 || $long > 180 )
            {
               print "Ignoring line $linenum in $srcfile: $line\n" if( $VERBOSE );
               next;
            }
            print $fh_out "${date}T${time} $lat $long\n";
         }
         else
         {
            die "Could not parse line $linenum in $srcfile: $line\n";
         }
      }
      close($fh_in);
      close($fh_out);

   #-----------------------------------------------------------------
   # run the new file through gpsbabel to generate a gpx
   # abort if gpsbabel fails for any reason

      my $gpscmd = qq(gpsbabel -i xcsv,style=$STYLEFILE -f $fixfile -x track,pack${MAXTIME}${MAXDIST},title="Log\%t$dirnu
      print "--> $gpscmd\n" if( $VERBOSE );
      my $result = system($gpscmd);
      die "gpsbabel command failed\n" if( $result );
      $gpscmd = qq(gpsbabel -i gpx -f $gpxfile -x interpolate,time=30 -o gpx -F $gpxfile);
      print "--> $gpscmd\n" if( $VERBOSE );
      $result = system($gpscmd);
      die "gpsbabel command failed\n" if( $result );
      print "Converted $srcfile to $gpxfile\n" if( $VERBOSE );
   }

   print "Done processing directory $dir\n" if( $VERBOSE );
}
```

B.2 Arquivo diselnet.style

O Código a seguir pertence ao arquivo diselnet.style a ser usado em conjunto com o Script de conversão em GPX. Este arquivo define como os dados de entrada devem ser lidos pelo gpsbabel para então serem transformados em arquivos GPX.

Para suportar outros tipos de arquivos, basta apenas alterar o formato de entrada descrito. Para mais informacoes, consultar manual o manual do gpsbabael em: `http://www.gpsbabel.org/readme.html`

```
# Format: DieselNet gps_log format
# Author Daniel Bittencourt
#  Date: 05/09/2012

DESCRIPTION DieselNet gps_log file format input
EXTENSION txt

#
# FILE LAYOUT DEFINITIONS
#
FIELD_DELIMITER SPACE
RECORD_DELIMITER NEWLINE
ENCODING utf-8
DATUM WGS 84
DATATYPE TRACK

#
# INDIVIDUAL DATA FIELDS, IN ORDER OF APPEARANCE:
#
IFIELD LOCAL_TIME,"","%FT%T"
IFIELD LAT_DECIMAL,"","%f" # LATITUDE
IFIELD LON_DECIMAL,"","%f" # LONGITUDE
```

Apêndice C

Patch NS3 para NS2-Mobility-Helper

C.1 Patch NS3 para NS2-Mobility-Helper

O *patch* a seguir foi implementado para possibilitar a leitura de um único arquivo de mobilidade por nó.

```
From 9df838fe116a574d93b6a19ae8aac9b22ef47ef1 Mon Sep 17 00:00:00 2001
From: Daniel Bittencourt <dcb@dcc.ufam.edu.br>
Date: Wed, 10 Oct 2012 02:13:03 -0400
Subject: [PATCH] Adding support to install NS2MobilityHelper to only one node

Signed-off-by: Daniel Bittencourt <dcb@dcc.ufam.edu.br>
---
 src/mobility/helper/ns2-mobility-helper.cc |  243 ++++++++++++++++++++++++++++
 src/mobility/helper/ns2-mobility-helper.h  |    5 +
 2 files changed, 248 insertions(+)

diff --git a/src/mobility/helper/ns2-mobility-helper.cc b/src/mobility/helper/ns2-mobility-helper.cc
index 268b0a2..45fa000 100644
--- a/src/mobility/helper/ns2-mobility-helper.cc
+++ b/src/mobility/helper/ns2-mobility-helper.cc
@@ -175,6 +175,26 @@ Ns2MobilityHelper::GetMobilityModel (std::string idString, const ObjectStore &st
   return model;
 }

+Ptr<ConstantVelocityMobilityModel>
+Ns2MobilityHelper::GetNodeMobilityModel (std::string idString, const Ptr<Object> object) const
+{
+  std::istringstream iss;
+  iss.str (idString);
+  uint32_t id (0);
+  iss >> id;
+  if (object == 0)
+    {
+      return 0;
+    }
+  Ptr<ConstantVelocityMobilityModel> model = object->GetObject<ConstantVelocityMobilityModel> ();
+  if (model == 0)
+    {
+      model = CreateObject<ConstantVelocityMobilityModel> ();
+      object->AggregateObject (model);
+    }
```

```
+  return model;
+}
+

 void
 Ns2MobilityHelper::ConfigNodesMovements (const ObjectStore &store) const
@@ -394,6 +414,224 @@ Ns2MobilityHelper::ConfigNodesMovements (const ObjectStore &store) const
     }
 }

+void
+Ns2MobilityHelper::ConfigNodeMovements (const Ptr<Object> node) const
+{
+  std::map<int, DestinationPoint> last_pos;    // Stores previous movement scheduled for each node
+
+  //*****************************************************************
+  // Parse the file the first time to get the initial node positions.
+  //*****************************************************************
+
+  // Look through the whole the file for the the initial node
+  // positions to make this helper robust to handle trace files with
+  // the initial node positions at the end.
+  std::ifstream file (m_filename.c_str (), std::ios::in);
+  if (file.is_open ())
+    {
+      while (!file.eof () )
+        {
+          int         iNodeId = 0;
+          std::string nodeId;
+          std::string line;
+
+          getline (file, line);
+
+          // ignore empty lines
+          if (line.empty ())
+            {
+              continue;
+            }
+
+          ParseResult pr = ParseNs2Line (line); // Parse line and obtain tokens
+
+          // Check if the line corresponds with setting the initial
+          // node positions
+          if (pr.tokens.size () != 4)
+            {
+              continue;
+            }
+
+          // Get the node Id
+          nodeId  = GetNodeIdString (pr);
+          iNodeId = GetNodeIdInt (pr);
+          if (iNodeId == -1)
+            {
+              NS_LOG_ERROR ("Node number couldn't be obtained (corrupted file?): " << line << "\n");
+              continue;
+            }
+
+          // get mobility model of node
+          Ptr<ConstantVelocityMobilityModel> model = GetNodeMobilityModel (nodeId,node);
+
+          // if model not exists, continue
+          if (model == 0)
+            {
+              NS_LOG_ERROR ("Unknown node ID (corrupted file?): " << nodeId << "\n");
+              continue;
+            }
+
```

```
+
+         /*
+          * In this case a initial position is being seted
+          * line like $node_(0) set X_ 151.05190721688197
+          */
+         if (IsSetInitialPos (pr))
+           {
+             DestinationPoint point;
+             //                                                  coord         coord value
+             point.m_finalPosition = SetInitialPosition (model, pr.tokens[2], pr.dvals[3]);
+             last_pos[iNodeId] = point;
+
+             // Log new position
+             NS_LOG_DEBUG ("Positions after parse for node " << iNodeId << " " << nodeId <<
+                           " position = " << last_pos[iNodeId].m_finalPosition);
+           }
+       }
+     file.close ();
+   }
+
+ //*****************************************************************
+ // Parse the file a second time to get the rest of its values
+ //*****************************************************************
+
+ // The reason the file is parsed again is to make this helper robust
+ // to handle trace files with the initial node positions at the end.
+ file.open (m_filename.c_str (), std::ios::in);
+ if (file.is_open ())
+   {
+     while (!file.eof () )
+       {
+         int         iNodeId = 0;
+         std::string nodeId;
+         std::string line;
+
+         getline (file, line);
+
+         // ignore empty lines
+         if (line.empty ())
+           {
+             continue;
+           }
+
+         ParseResult pr = ParseNs2Line (line); // Parse line and obtain tokens
+
+         // Check if the line corresponds with one of the three types of line
+         if (pr.tokens.size () != 4 && pr.tokens.size () != 7 && pr.tokens.size () != 8)
+           {
+             NS_LOG_ERROR ("Line has not correct number of parameters (corrupted file?): " << line << "\n");
+             continue;
+           }
+
+         // Get the node Id
+         nodeId  = GetNodeIdString (pr);
+         iNodeId = GetNodeIdInt (pr);
+         if (iNodeId == -1)
+           {
+             NS_LOG_ERROR ("Node number couldn't be obtained (corrupted file?): " << line << "\n");
+             continue;
+           }
+
+         // get mobility model of node
+         Ptr<ConstantVelocityMobilityModel> model = GetNodeMobilityModel (nodeId,node);
+
+         // if model not exists, continue
+         if (model == 0)
+           {
```

```
+              NS_LOG_ERROR ("Unknown node ID (corrupted file?): " << nodeId << "\n");
+              continue;
+            }
+
+
+          /*
+           * In this case a initial position is being seted
+           * line like $node_(0) set X_ 151.05190721688197
+           */
+          if (IsSetInitialPos (pr))
+            {
+              // This is the second time this file has been parsed,
+              // and the initial node positions were already set the
+              // first time.  So, do nothing this time with this line.
+              continue;
+            }
+
+          else // NOW EVENTS TO BE SCHEDULED
+            {
+
+              // This is a scheduled event, so time at should be present
+              double at;
+
+              if (!IsNumber (pr.tokens[2]))
+                {
+                  NS_LOG_WARN ("Time is not a number: " << pr.tokens[2]);
+                  continue;
+                }
+
+              at = pr.dvals[2]; // set time at
+
+              if ( at < 0 )
+                {
+                  NS_LOG_WARN ("Time is less than cero: " << at);
+                  continue;
+                }
+
+
+
+              /*
+               * In this case a new waypoint is added
+               * line like $ns_ at 1 "$node_(0) setdest 2 3 4"
+               */
+              if (IsSchedMobilityPos (pr))
+                {
+                  if (last_pos[iNodeId].m_targetArrivalTime > at)
+                    {
+                      NS_LOG_LOGIC ("Did not reach a destination! stoptime = " << last_pos[iNodeId].m_targetArrivalTime
+                      double actuallytraveled = at - last_pos[iNodeId].m_travelStartTime;
+                      Vector reached = Vector (
+                          last_pos[iNodeId].m_startPosition.x + last_pos[iNodeId].m_speed.x * actuallytraveled,
+                          last_pos[iNodeId].m_startPosition.y + last_pos[iNodeId].m_speed.y * actuallytraveled,
+                          0
+                          );
+                      NS_LOG_LOGIC ("Final point = " << last_pos[iNodeId].m_finalPosition << ", actually reached = " <<
+                      last_pos[iNodeId].m_stopEvent.Cancel ();
+                      last_pos[iNodeId].m_finalPosition = reached;
+                    }
+                  //                                        last position     time  X coord     Y coord      velocity
+                  last_pos[iNodeId] = SetMovement (model, last_pos[iNodeId].m_finalPosition, at, pr.dvals[5], pr.dvals[6
+
+                  // Log new position
+                  NS_LOG_DEBUG ("Positions after parse for node " << iNodeId << " " << nodeId << " position =" << last_p
+                }
+
+
+              /*
```

```
+               * Scheduled set position
+               * line like $ns_ at 4.634906291962 "$node_(0) set X_ 28.675920486450"
+               */
+              else if (IsSchedSetPos (pr))
+                {
+                  //                                        time  coordinate   coord value
+                  last_pos[iNodeId].m_finalPosition = SetSchedPosition (model, at, pr.tokens[5], pr.dvals[6]);
+                  if (last_pos[iNodeId].m_targetArrivalTime > at)
+                    {
+                      last_pos[iNodeId].m_stopEvent.Cancel ();
+                    }
+                  last_pos[iNodeId].m_targetArrivalTime = at;
+                  last_pos[iNodeId].m_travelStartTime = at;
+                  // Log new position
+                  NS_LOG_DEBUG ("Positions after parse for node " << iNodeId << " " << nodeId <<
+                                " position =" << last_pos[iNodeId].m_finalPosition);
+                }
+              else
+                {
+                  NS_LOG_WARN ("Format Line is not correct: " << line << "\n");
+                }
+            }
+        }
+      file.close ();
+    }
+}
+

 ParseResult
 ParseNs2Line (const std::string& str)
@@ -777,5 +1015,10 @@ Ns2MobilityHelper::Install (void) const
 {
   Install (NodeList::Begin (), NodeList::End ());
 }
+void
+Ns2MobilityHelper::Install (Ptr<Object> node) const
+{
+ ConfigNodeMovements (node);
+}

 } // namespace ns3
diff --git a/src/mobility/helper/ns2-mobility-helper.h b/src/mobility/helper/ns2-mobility-helper.h
index a9e5a13..5c1ba8e 100644
--- a/src/mobility/helper/ns2-mobility-helper.h
+++ b/src/mobility/helper/ns2-mobility-helper.h
@@ -90,6 +90,9 @@ public:
    */
   void Install (void) const;

+  //Install to only one node
+  void Install (Ptr<Object> node) const;
+
   /**
    * \param begin an iterator which points to the start of the input
    *        object array.
@@ -111,7 +114,9 @@ public:
     virtual Ptr<Object> Get (uint32_t i) const = 0;
   };
   void ConfigNodesMovements (const ObjectStore &store) const;
+  void ConfigNodeMovements (const Ptr<Object> node) const;
   Ptr<ConstantVelocityMobilityModel> GetMobilityModel (std::string idString, const ObjectStore &store) const;
+  Ptr<ConstantVelocityMobilityModel> GetNodeMobilityModel (std::string idString, const Ptr<Object> object) const;
   std::string m_filename;
 };

--
1.7.10.4
```

Apêndice D

Programa para coleta de Dados VDT

D.1 Programa para coleta de Dados VDT

O *script* a seguir é uma re-implementação do aplicativo originalmente utilizado no DOME, escrito em Java, para coleta de dados e medição de vazão durante contatos. Este programa é escrito em Python e tem como objetivo corrigir os problemas identificados neste trabalho, no programa original além de permitir que ele seja executado em diferentes plataformas.

```
import socket
import thread
import time
import os
import hashlib
from datetime import datetime
import struct
import optparse
import sys

parser = optparse.OptionParser()
parser.add_option('-t', help='ip address for host throughput tests', dest='host', action='store_true')

(opts, args) = parser.parse_args()

if ((opts.host is None) or (len(args) == 0)):
    sys.stderr.write("ERROR: a mandatory option is missing\n")
    sys.exit()
else:
    hostHost = args[0]

def getBroadcastAddress():
    brAddr = hostHost.split('.')
    brAddr[3] = '255'
    brAddr = '.'.join(brAddr)

    return brAddr

class Beacon:

    __slots__ = '__sock', '__addr'
```

```
    def __init__(self, port):
        "Initialize the beacon for sending and receiving data."
        self.__sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.__sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, True)
        self.__sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
        self.__sock.bind(('0.0.0.0', port))
        self.__addr = getBroadcastAddress(), port

    def __del__(self):
        "Shutdown and close the underlying socket."
        self.__sock.shutdown(socket.SHUT_RDWR)
        self.__sock.close()

    def recv(self, size):
        "Receive a broadcast through the underlying soket."
        return self.__sock.recvfrom(size)

    def send(self, data):
        "Send a broadcast through the underlying socket."
        assert self.__sock.sendto(data, self.__addr) == len(data), \
               'Not all data was sent through the socket!'

    def __gettimeout(self):
        return self.__sock.gettimeout()

    def settimeout(self, value):
        self.__sock.settimeout(value)

    def __deltimeout(self):
        self.__sock.setblocking(True)

    timeout = property(__gettimeout, settimeout, __deltimeout,
                       'Timeout on blocking socket operations.')

def getMACAddress():
    macFile = open('/tmp/MAC', 'r')
    return macFile.readlines()[0].strip()

localMAC = getMACAddress()

def run():
    "Test the beacon broadcasting class."
    b = Beacon(50000)
    c = Beacon(50000)
    thread.start_new_thread(greetingSender, (b,))
    thread.start_new_thread(fileReceiver, ())
    greetingListener(c)

def greetingSender(b):
    while True:
        b.send(localMAC)
        time.sleep(0.5)

def greetingListener(b):
    while True:
        data, address = b.recv(1500)
        if (data.decode() != localMAC):
            fileSender(address[0])
            start = datetime.now()
            b.settimeout(0.0)
            while True:
                now = datetime.now()
                diffTime = now - start
                remaning = 1000 - (diffTime.microseconds / 1000)
                if (remaning <= 0):
                    break
                try:
```

```
                    b.recv(1500)
                except socket.error, (value, message):
                    sys.stdout.write("Greeting Listener: " + str(value) + str(message) + "\n")
                    break

            b.settimeout(None)
        else:
            sys.stdout.write("Greeating from self\n")

def fileSender(address):
    TCP_PORT = 5005
    MESSAGE = localMAC + os.urandom(1383)

    sys.stdout.write("Starting fileSender to " + address + "\n")
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(0.7)
    try:
        s.connect((address, TCP_PORT))
    except socket.timeout, (message):
        sys.stdout.write("File Sender: "+  str(message) + "\n")
        return
    except socket.error, (value, message):
        if (value == 107):
            return
        sys.stdout.write("Could not connect. Giving Up" + str(message) + "\n")
        return

    start = int(time.time())
    totalBytesSent = 0
    bytesSent = 0
    while True:
        try:
            bytesSent = s.send(MESSAGE)
            if ((int(time.time()) - start) > 60):
break

            totalBytesSent += bytesSent
            continue
        except socket.timeout, (message):
            sys.stdout.write("File Sender: "+  str(message) + "\n")
            break
        except socket.error, (value, message):
            sys.stdout.write("Data Send Stopped: " + str(value) + str(message) + " Total: " + str(totalBytesSent) + "\n")
            if(value == 11):
                continue
            else:
                break
    try:
        s.shutdown(socket.SHUT_RDWR)
    except socket.error, (value, message):
        if(value == 107):
            pass
    s.close()
    return

def fileReceiver():
    TCP_PORT = 5005
    BUFFER_SIZE = 4096

    namer = hashlib.sha224(str(time.time())).hexdigest()[:10]
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((hostHost, TCP_PORT))
    s.listen(1)
    while 1:
        try:
            conn, addr = s.accept()
        except socket.error, (value,message):
```

```
            sys.stdout.write("fileReceiver Error: " + str(value) + str(message) + "\n")

        startTime = datetime.now()
        bytesReceived = 0
        conn.settimeout(0.7)
        while 1:
            try:
                data = conn.recv(BUFFER_SIZE)
            except socket.timeout, (message):
                sys.stdout.write("File Receiver" + str(message) + "\n")
                break
            except socket.error, (value, message):
                sys.stdout.write("Data Receive Error: " + str(value) + str(message) + "\n")
                break

            endTime = datetime.now()
            if (len(data) > 0):
                if (bytesReceived == 0 and len(data) >= 17):
                    peer = data[:17]
                bytesReceived += len(data)
            else:
                break

        conn.shutdown(socket.SHUT_RDWR)
        conn.close()
        sys.stdout.write("File Receiver going to write log\n")
        logReceiver(bytesReceived, peer, True, startTime, endTime, namer)

def logReceiver(bytesReceived, peer, bus, startTime, endTime, namer):
   logDir = "/tmp/logs/throughput/"

   diffTime = endTime - startTime
   diffTime = diffTime.microseconds / 1000

   date = datetime.now()
   date = '{:%m %d %Y at %H:%M:%S}'.format(date)

   log_entry = "{!s} received {!s} bytes from {!s} in {!s} ms on {!s} at location is not current longitude is 0 latitude

   logFileName = logDir + str(int(time.time())) + namer

   logFile = open(logFileName, 'w+')
   logFile.write(log_entry)
   logFile.close()

if __name__ == '__main__':
    run()
```