

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da
Computação

Avaliação do Consumo de Energia em Protocolos de
Roteamento para Redes Tolerantes a Atrasos e
Interrupções

Thiago Santana Batista

Dissertação submetida à Coordenação do Curso de Pós-Graduação em
Ciência da Computação da Universidade Federal de Campina Grande -
Campus I como parte dos requisitos necessários para obtenção do grau
de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

Marco Aurélio Spohn

(Orientador)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

B333a Batista, Thiago Santana.

Avaliação do consumo de energia em protocolos de roteamento para redes tolerantes a atrasos e interrupções / Thiago Santana Batista. — Campina Grande, 2011.

72 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática. Referências.

Orientador: Prof. Dr. Marco Aurélio Spohn.

1. Rede de Computadores. 2. Desempenho. 3. Consumo de Energia. 4. Tempo de Vida. 5. DTN. I. Título.

CDU – 004.7(043)

UFCG - BIBLIOTECA - CAMPUS I	
30-06-11	3214-11
doação	


**"AVALIAÇÃO DO CONSUMO DE ENERGIA EM PROTOCOLOS DE ROTEAMENTO
PARA REDES TOLERANTES A ATRASOS E INTERRUPÇÕES"**

THIAGO SANTANA BATISTA

DISSERTAÇÃO APROVADA EM 06.05.2011


MARCO AURELIO SPOHN, Ph.D
Orientador(a)


REINALDO CÉZAR DE MORAIS GOMES, Dr.
Examinador(a)


FABIANO SALVADORI, D.Sc
Examinador(a)

CAMPINA GRANDE - PB

Resumo

As Redes Tolerantes a Atrasos e Interrupções caracterizam uma arquitetura completamente inovadora na área das redes de computadores, onde esta aborda o conceito de redes conectadas ocasionalmente e que podem sofrer desconexões frequentes. Por ser uma área nova, ela está em constante desenvolvimento, criando novos algoritmos de roteamento. Porém, quando estes são elaborados e comparados com outros existentes, raramente é realizada a análise do consumo de energia do dispositivo. Este trabalho apresenta uma avaliação de desempenho dos protocolos de roteamento para esse tipo de rede, considerando-se dispositivos móveis com recursos limitados de energia (i.e., alimentados por bateria). Para que fosse realizada uma avaliação mais precisa, utilizou-se o modelo analítico de bateria de Rakhmatov-Vrudhula nas simulações das redes. Além de comparar os protocolos de roteamento, tornou-se possível confrontar o desempenho do modelo linear de bateria (adotado pela maioria dos simuladores de redes móveis) com o modelo de bateria Rakhmatov-Vrudhula e dessa forma pôde-se mais uma vez comprovar a superioridade deste. Os resultados obtidos apontam a superioridade deste modelo por ele considerar o efeito de taxa da capacidade e o efeito da recuperação da bateria. Ainda verificou-se que o protocolo de roteamento *Spray and Wait* possui características escaláveis e menos custoso, considerando o consumo da capacidade da bateria, em sua utilização.

Abstract

Delay and Disruption Tolerant Networks featuring a completely new architecture in the area of computer networks, where it discusses the concept of occasionally connected networks that may suffer frequent disconnections. Being a new area, it is constantly evolving, with new routing algorithms being created every day. However, when these are developed and compared with other existing, is rarely examined the power consumption of the device. This work presents a performance evaluation of routing protocols in this kind of network, considering mobile devices with limited power source (i.e., battery powered). In order to provide a more precise assessment, we used the Rakhmatov-Vrudhula battery model in the network simulations. Besides comparing routing protocols, we were able to compare the performance of the linear battery model (adopted by most mobile network simulators) with the more realistic Rakhmatov-Vrudhula battery model. The results indicate the superiority of the Rakhmatov Vrudhula battery model because it consider the effect of rate capacity and recovery effect from the battery. Although, it was found that the routing protocol Spray and Wait has scalable features and have a very low cost in it's use, considering the consumption of the battery capacity.

Agradecimentos

Primeiramente, à Deus por tudo o que Ele tem me dado em toda a vida. A minha família que deposita esperanças e torce por mim em todos os momentos. A minha namorada, Mitshuia, sempre presente me apoiando e torcendo por mim.

Um muito obrigado ao meu orientador, Marco Spohn, por ter dado essa oportunidade e acreditado em mim. Sempre compartilhando o seu conhecimento e dando suporte nas áreas onde necessitava de lapidações, contribuindo para o meu crescimento profissional.

Agradeço ao REUNI pelo apoio financeiro a este trabalho.

Pelos grandes companheiros ao longo dessa caminhada: José Athayde, Rodrigo Lopes e Alan Cruz. Foi um prazer compartilhar esta experiência com vocês, espero que Deus os abençoe sempre.

Aos meus amigos, que não irei citar para não correr o risco de esquecer alguns, que mesmo não participando diretamente no mestrado comigo, contribuíram e muito com o seus apoios durante esta fase.

Aos companheiros do LATEC: Elmano, Rafael e Ruan. Que sempre estiveram dispostos a discutir, ajudar e compartilhar informações. Foi um ambiente de trabalho muito saudável e que contribuiu muito para este projeto e para o meu crescimento acadêmico.

Conteúdo

1	Introdução	1
1.1	Motivação	2
1.2	Objetivos	3
1.3	Relevância	4
1.4	Estrutura da Dissertação	4
2	Redes Tolerantes a Atrasos e Interrupções	5
2.1	Arquitetura	6
2.2	Aplicações	8
2.3	Estratégias de Roteamento	9
3	Protocolos de Roteamento Analisados	10
3.1	<i>Epidemic Routing</i>	10
3.2	PROPHET	11
3.3	<i>Spray and Wait</i>	12
4	Modelos de Bateria	14
4.1	Modelo Linear	15
4.2	Modelo Rakhmatov-Vrudhula	16
5	Metodologia	18
5.1	Problemas Encontrados	20
5.2	Métricas utilizadas e configurações dos parâmetros	22
6	Cenários e Análises dos Resultados	25

6.1	Tempo de vida da rede	25
6.1.1	Primeiro nó sem recurso	26
6.1.2	35% dos nós sem recurso	28
6.1.3	60% dos nós sem recurso	32
6.2	Quantidade de Mensagens Descartadas	35
6.3	Número de cópias para cada mensagem - <i>Overhead</i>	37
6.4	Quantidade de Mensagens Retransmitidas	38
7	Conclusão	41
A	Script de criação dos cenários	48
B	Código fonte de um dos protocolos de roteamento adaptado	55

Lista de Símbolos

C - Consumo

IC - Intervalo de Confiança

RFC - *Request For Comment*

DTN - *Delay and Disruption Tolerant Network*

NASA - *National Aeronautics and Space Administration*

Epidemic - *Epidemic Routing*

PROPHET - *Probabilistic Routing Protocol using History of Encounters and Transitivity*

Spray and Wait - *Spray and Wait*

Lista de Figuras

2.1	Arquitetura de uma DTN.[16]	6
4.1	Diferentes estados de operação da bateria [32].	15
4.2	Capacidade da Bateria em Função do Perfil de Descarga [32].	16
5.1	Mapa utilizado durante as simulações.	19
5.2	Procedimento (a) errado e (b) correto para cálculo de uma transmissão no simulador	21
6.1	Tempo de vida da rede para 50 nós quando o primeiro nó morre.	27
6.2	Tempo de vida da rede para 100 nós quando o primeiro nó morre.	27
6.3	Tempo de vida da rede para 200 nós quando o primeiro nó morre	29
6.4	Tempo de vida da rede para 50 nós quando 35% dos nós morrem.	29
6.5	Tempo de vida da rede para 100 nós quando 35% dos nós morrem.	30
6.6	Tempo de vida da rede para 200 nós quando 35% dos nós morrem.	30
6.7	Tempo de vida da rede para 50 nós quando 60% dos nós morrem.	33
6.8	Tempo de vida da rede para 100 nós quando 60% dos nós morrem.	33
6.9	Tempo de vida da rede para 200 nós quando 60% dos nós morrem.	35
6.10	Quantidade de mensagens descartadas na rede.	37
6.11	<i>Overhead</i> gerado na rede.	38
6.12	Quantidade de mensagens retransmitidas na rede.	40

Lista de Tabelas

5.1	Parâmetros utilizados	22
5.2	Parâmetros de configuração dos cenários	24
6.1	Tempo de vida para a rede utilizando 50 nós quando o primeiro nó morre. .	28
6.2	Tempo de vida para a rede utilizando 100 nós quando o primeiro nó morre.	28
6.3	Tempo de vida para a rede utilizando 200 nós quando o primeiro nó morre.	31
6.4	Tempo de vida para a rede utilizando 50 nós quando 35% dos nós morrem. .	31
6.5	Tempo de vida para a rede utilizando 100 nós quando 35% dos nós morrem.	32
6.6	Tempo de vida para a rede utilizando 200 nós quando 35% dos nós morrem.	32
6.7	Tempo de vida para a rede utilizando 50 nós quando 60% dos nós morrem. .	34
6.8	Tempo de vida para a rede utilizando 100 nós quando 60% dos nós morrem.	34
6.9	Tempo de vida para a rede utilizando 200 nós quando 60% dos nós morrem.	36
6.10	Quantidade de mensagens descartadas na rede.	36
6.11	<i>Overhead</i> gerado na rede.	39
6.12	Quantidade de mensagens retransmitidas na rede.	39

Capítulo 1

Introdução

A Internet tornou-se um grande sucesso no que diz respeito à comunicação entre dispositivos ao redor do mundo. Utilizando um conjunto de protocolos homogêneo de comunicação, a pilha de protocolos TCP/IP [5], tornou-se possível realizar o roteamento de dados e garantir confiabilidade na troca de pacotes. Entretanto, em seus primórdios, as redes eram em sua maioria cabeadas e, portanto, não apresentavam atrasos muito longos ou desconexões.

Com o passar das décadas, as redes adquiriram mais importância, e, com isso foram surgindo diferentes tecnologias. Como exemplo, podemos citar as redes sem fio que cada vez mais vêm sendo utilizadas no nosso dia-dia. No entanto, existem algumas que têm longos atrasos e desconexões que são determinados pela soma dos atrasos salto-a-salto.

A soma é decomposta em: a) tempo de processamento; b) atraso nas filas; c) atraso de transmissão; d) atraso de propagação. Em diversos cenários, como as redes interplanetárias utilizadas pela NASA [7], essa pode chegar à ordem de horas ou até mesmo dias, tornando a pilha de protocolos TCP/IP ineficaz.

As desconexões podem ser causadas pela mobilidade dos nós em uma rede sem fio, por economia de recursos em redes de sensores que se desligam (i.e., dormem) para poupar energia ou ainda por negação de serviço como, por exemplo, sujar a frequência de um inimigo (*jamming*) em operações militares [39].

As Redes Tolerantes a Atrasos e Interrupções (*Delay and Disruption Tolerant Network* - DTN) [16] possuem uma arquitetura inovadora abordando o conceito de redes conectadas ocasionalmente e que podem sofrer desconexões frequentes. Um ambiente que possui essas desvantagens seria o espaço sideral. Este serviu de estímulo para a criação do projeto Internet

Interplanetária (IPN) [6], que trata de comunicações em um ambiente com grandes atrasos. É necessário destacar, no entanto, que este é um caso específico para esse tipo de rede.

Fall *et al.* [16], constataram que utilizar a solução empregada atualmente para a Internet não resultaria em um bom desempenho, pois o protocolo TCP gera muitos pacotes de controle, tanto para iniciar quanto para terminar uma transmissão. A carga útil, quantidade efetiva de dados, não é favorecida por este protocolo, devido ao fato deste conter diversos cabeçalhos controladores de transmissão. Outro ponto importante é o fato de que os longos atrasos deste cenário podem gerar quedas frequentes nas conexões causando a inviabilidade de utilização da rede.

Nos cenários específicos das DTNs, devido à imprevisibilidade de se determinar o encontro dos nós, fica clara a necessidade de se utilizar o tempo de conexão da forma mais eficiente possível; ou seja, é preferível que se envie a menor quantidade de pacotes de controle, resultando assim num melhor aproveitamento da rede.

1.1 Motivação

É importante ressaltar que em boa parte dos cenários, os nós possuem recursos limitados de energia, devido ao fato de a grande maioria, senão a totalidade, das DTNs ser composta de nós móveis. Ainda assim, a preocupação com o consumo não é um fator constantemente analisado nos estudos destas.

Por se tratar de uma área em ascensão, novos protocolos de roteamento para DTNs estão sendo constantemente elaborados. Quando um novo é proposto, são utilizadas, geralmente, as seguintes métricas como medidas de desempenho para comparação entre os mesmos [27; 37; 34; 8]:

- Quantidade de carga gerada na rede - quantidade de mensagens gerada na rede.
- Atraso da entrega - tempo que uma mensagem leva para chegar ao destinatário.
- Taxa de entrega - quantidade de entrega dos pacotes pelo protocolo por unidade de tempo.

Por isso há a necessidade de um estudo que relacione o consumo da capacidade da bateria ao desempenho dos protocolos. Este, por sua vez, está vinculado ao tempo de vida da rede.

O tempo de vida da rede não possui definição única. Considerando que esta característica é determinada pelo esgotamento de recursos dos dispositivos, alguns autores [14] ponderam que se trata do tempo em que:

- Uma certa porcentagem dos nós permanece conectada.
- Ocorre a desconexão do primeiro nó, resultante do esgotamento da sua bateria.
- Os nós estão conectados, até que ocorra uma fragmentação da rede, devido à desconexão de um ou mais nós. E, assim, causar o particionamento da rede.

Para que uma análise do consumo de energia seja feita de forma mais acurada, é necessário que seja utilizado um modelo de bateria mais realístico do que o modelo linear, geralmente utilizado por diversos simuladores de redes de computadores. Neste modelo, a bateria é representada apenas como um repositório linear de energia. Dessa forma, este modelo não trata de certas propriedades pertencentes às baterias dos dispositivos atuais, como por exemplo, os efeitos de relaxação e de taxa de capacidade [23]. Assim, foi escolhido o modelo analítico de bateria de Rakhmatov-Vrudhula [29] por ser de desenvolvimento simples e flexível além de que, por ser uma modelagem mais realística, gerará resultados mais próximos da realidade.

1.2 Objetivos

Considerando os fatos supracitados, esta dissertação tem como objetivo principal realizar uma avaliação dos principais protocolos de roteamento utilizados nas DTNs, levando em consideração o tempo de vida da rede como a principal medida de desempenho. Além desta, serão analisadas a quantidade de mensagens descartadas, retransmitidas e geradas na rede (*overhead*).

Os objetivos específicos são:

- Estender o simulador com os modelos de bateria linear e Rakhmatov-Vrudhula.
- Analisar o tempo de vida da rede utilizando estes modelos de bateria.
- Mensurar o desempenho dos protocolos de roteamento *Epidemic* [37], *PROPHET* [25] e *Spray And Wait*[34] utilizando os diferentes modelos de bateria e métricas.

- Comparar o desempenho dos diferentes modelos de bateria.

1.3 Relevância

A partir dos dados coletados, abre-se a possibilidade de estudos posteriores para analisar quais protocolos de roteamento se adequam para determinadas situações (e.g, maior tempo de vida da rede, quantidade de operações), visto que sempre existe um *tradeoff* entre estes.

Como o modelo de bateria Rakhmatov-Vrudhula comparativamente é superior em precisão do que o modelo de bateria linear, acarretará em simulações mais acuradas. Estas, por conseguinte, proporcionarão uma maior aproximação da realidade e dados com uma maior relevância.

Com esse estudo, espera-se que futuros pesquisadores possam utilizar os resultados obtidos no desenvolvimento ou aperfeiçoamento dos protocolos de roteamento para DTNs existentes e que ainda serão propostos tomando como base as questões energéticas.

1.4 Estrutura da Dissertação

Este trabalho encontra-se dividido em sete capítulos. O primeiro destina-se a apresentar de forma sucinta a definição do problema, a motivação para realização deste, os objetivos e a relevância para projetos futuros.

O Capítulo 2 apresenta uma descrição detalhada das DTNs, esboçando a sua arquitetura e comparando-a com a atual utilizada na Internet.

Uma revisão bibliográfica dos protocolos de roteamento utilizados é apresentada no Capítulo 3. Pontuam-se as principais características e comportamentos de cada um.

O Capítulo 4 vislumbra os modelos de bateria empregados durante as simulações, mostrando as suas principais propriedades.

A metodologia utilizada durante este trabalho é descrita detalhadamente no Capítulo 5. Nele, são definidos os diversos parâmetros, cenários e métricas utilizadas durante a realização da dissertação.

O Capítulo 6 exhibe e discute os resultados obtidos das simulações. Por fim, no Capítulo 7, são apontadas as conclusões e sugere possíveis trabalhos futuros.

Capítulo 2

Redes Tolerantes a Atrasos e Interrupções

As Redes Tolerantes a Atrasos e Interrupções (*Delay and Disruption Tolerant Network - DTN*) [16], caracterizam uma arquitetura completamente inovadora na área das redes de computadores, onde ela procura solucionar os cenários em que as redes são conectadas ocasionalmente e que podem sofrer desconexões frequentes.

A capacidade de transportar ou rotear dados entre duas entidades é fundamental, e todas as redes de comunicação devem satisfazer essa condição. As DTNs têm como objetivo, solucionar problemas em cenários que são caracterizados pelas desconexões frequentes, caminhos completos fim a fim inexistentes, entre outros problemas semelhantes [39].

À primeira vista, as DTNs são muito parecidas com as redes *ad hoc* [31], porém, a diferença é nítida quando levamos em consideração a abordagem do problema de roteamento. Nelas é necessário que se conheça todo o percurso para que os protocolos funcionem. Há casos em que é possível construir toda uma rota de envio, mas existem diversas situações em que isso não é alcançável. E ainda aquelas cujos caminhos previamente conhecidos se comprometem devido aos longos atrasos, como por exemplo, nas redes interplanetárias [6], os quais chegam a ser da ordem de horas ou até dias.

2.1 Arquitetura

O modelo de arquitetura mais utilizado pela literatura é o baseado no sistema de correios postal [16], o qual funciona através da comutação de mensagens. Neste contexto, é utilizada a técnica de repasse denominada de *store and forward*, na qual um nó pode armazenar uma mensagem, até encontrar o destinatário final ou outro nó que possa repassar a mensagem para este. Essa comunicação entre um par de nós, para um dado instante, só é possível se a distância entre eles for menor que o alcance da transmissão.

Vale lembrar que nas redes usuárias de TCP/IP também é realizado o armazenamento de pacotes para, em seguida, serem encaminhados ao próximo nó. Esta tarefa, normalmente realizada pelos roteadores, é chamada de *Packet Switching*, porém, ela é executada na ordem de milissegundos, enquanto que nas DTNs são da ordem de horas ou dias. Devido à alta latência na entrega das mensagens é necessário o uso de algum armazenamento persistente como, por exemplo, um disco rígido, para preservar as informações diante de eventuais reinicializações do sistema. Neste tipo de rede, os dispositivos de comunicação não se comportam apenas como *transponders* outrossim são roteadores.

Este modelo demonstra que os nós devem ter uma transferência de custódia cuja finalidade será de que, a partir do encontro com o nó intermediário, seja realizada a transferência da responsabilidade da entrega da mensagem para o destinatário ou para outro nó que tenha boas chances de encontrá-lo.

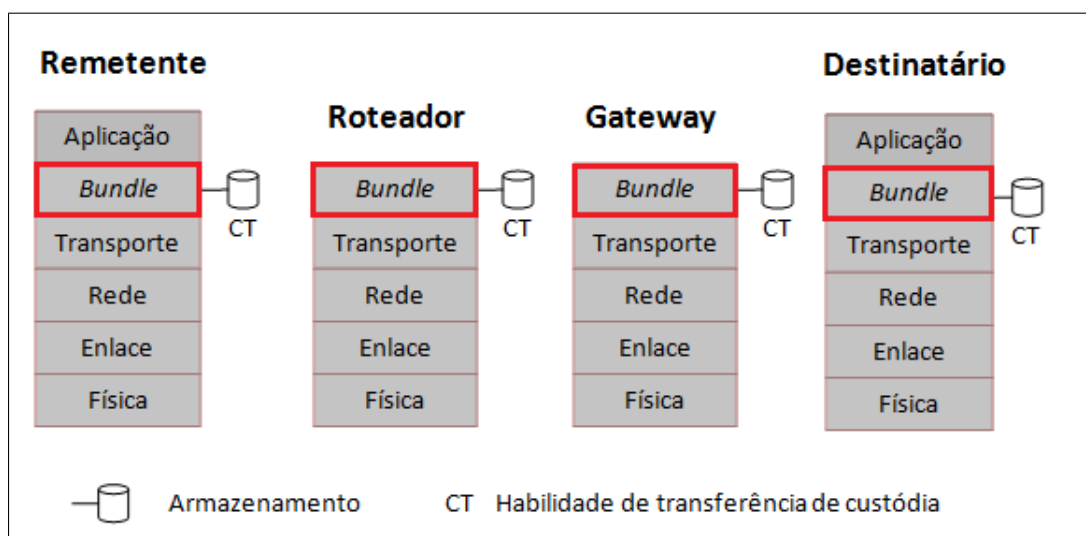


Figura 2.1: Arquitetura de uma DTN.[16]

Observe que a nova arquitetura consiste em adicionar uma camada entre as utilizadas normalmente no padrão da Internet (vide figura 2.1). Também pode-se observar que essa nova camada irá armazenar as mensagens e o controle das transferências de custódia. Os nós intermediários (roteadores e *gateways*) não atingem a camada de aplicação, já que a mensagem não os tem como destinatário.

Visando prover um *framework* compartilhado para o desenvolvimento de algoritmos e aplicações para DTNs, foram publicadas em 2007 as RFC 4838 [3] e RFC 5050 [4], que definem uma abstração comum para este tipo de rede, comumente conhecido como *Bundle Protocol*. Esse protocolo, por sua vez, estabelece uma série de blocos de dados contíguos como um “maço” ou conjunto (*bundle*). Cada um deles contém informação semântica suficiente para permitir que a aplicação faça progresso, onde um bloco individual não faria. Eles são repassados através do método *store and forward* entre os nós participantes (vide Figura 2.1). As camadas de transporte, carregando os *bundles* através de suas redes locais, são chamadas de *bundle convergence layers* (camadas de convergência de conjuntos).

A arquitetura dos *bundles* opera como uma rede sobreposta (*overlay network*), provendo uma nova disposição de nomes baseada em *Endpoint Identifiers* (EIDs), que identificam o remetente e o destino final dos mesmos. Protocolos que utilizam esse método devem influenciar as preferências em nível de aplicação para enviá-los através de uma rede. Devido à natureza *store and forward* dos protocolos tolerantes a atraso, soluções de roteamento para DTNs podem se beneficiar desta exposição da informação na camada de aplicação. Por exemplo, o agendamento ou escalonamento de redes pode ser influenciado se os dados da aplicação devem ser recebidos inteiramente, rapidamente, ou sem variação de atraso dos pacotes. Protocolos deste gênero coletam dados da aplicação em *bundles* e, quando requisitados, são enviados através de configurações de redes heterogêneas com garantias de serviço de alto nível. Estas são definidas pelo nível de aplicação e a especificação da RFC 5050 de *Bundle Protocol* inclui as marcações *bulk*, *normal* e *expedited*.

Na marcação *Expedited*, os *bundles* são sempre transmitidos, remontados e verificados com uma prioridade maior do que qualquer outra classe. *Normal* é enviado depois que todos os pacotes *expedited* tenham sido montados com sucesso em seu destino. Já na marcação *Bulk*, os pacotes não são tratados até que, todos os pacotes das outras classes, de mesma fonte e destino, tenham sido transmitidos e remontados com sucesso.

2.2 Aplicações

Existem diversos casos de aplicação para redes DTN. A seguir são listados alguns exemplos:

- **Redes militares:** os nós estão sujeitos à destruição completa. Neste caso é necessário que a comunicação entre os nós remanescentes continue ativa. É crucial a atualização constante das informações em campo de batalha.
- **Rastreamento de animais selvagens:** geralmente, são colocados transmissores em animais no formato de colares ou pulseiras para realizar a análise do comportamento dos mesmos. Juang *et al.* [19] mostram a utilização prática em um projeto chamado *ZebraNet*. Nele, são utilizados colares rastreadores nas zebras, equipados com GPS e com uma pequena memória para armazenar os dados, que são repassados de forma semelhante a uma rede *peer-to-peer*.
- **Rede interplanetárias (Inter-planetary Network - IPN):** Burleigh *et al.* [10] explanam que neste tipo existem nodos fixos na atmosfera e outros na Terra. Por isso, a comunicação entre os nós ocorre em um certo intervalo de tempo por dia, já que a Terra encontra-se em constante movimento e caso haja necessidade de realizar a comunicação entre ela e outro planeta, a duração pode ser de horas ou dias, já que o espaço para a propagação dos nós é bastante extenso.
- **Desastres Naturais:** com o auxílio das redes DTN, é possível estabelecer uma comunicação sem infraestrutura e em cenários onde não se tem a certeza de uma conexão fim a fim. Uddin *et al.* [36] criaram um modelo para simular este problema, mostrando como seria a utilização das redes DTN para a comunicação entre os veículos de resgate e as bases de apoio.
- **Ambientes públicos:** diversos trabalhos na literatura [24; 12; 25], mencionam uma rede baseada em uma comunidade (por exemplo, uma universidade) ou porção de uma cidade. Daly e Haahr [12] realizaram um experimento no campus da faculdade de *Trinity College Dublin*, distribuindo aparelhos celulares para diversos usuários. Foi instalado em cada aparelho o algoritmo de roteamento proposto pela equipe, chamado de SimBet. Este tipo de ambiente possibilita a formação de rotinas, e, baseado em

um histórico, é possível estabelecer previsões que irão auxiliar os protocolos de roteamento. Foi baseado nisso que Lindgren *et al.* [25] propuseram o protocolo de roteamento *Probabilistic Routing Protocol using History of Encounters and Transitivity* (PROPHET), o qual será melhor descrito no próximo capítulo.

2.3 Estratégias de Roteamento

Desde a formulação das redes DTN, foram propostos vários protocolos de roteamento, os quais utilizam basicamente duas classes de estratégia: a primeira caracteriza-se por armazenar as informações de roteamento sobre os nodos da rede, enquanto a segunda não armazena, tomando suas decisões com base em informações locais dos nodos encontrados e das políticas de roteamento de cada protocolo.

O uso da primeira implica em maiores custos de comunicação e armazenamento, destinados à manutenção de informação do roteamento dos nodos. Esses crescem na proporção do número de nós da rede e, dependendo do tipo de dispositivo, podem vir a ser muito altos. Por outro lado, o tráfego de mensagens dos dados, tende a ser mais econômico utilizando essa estratégia. Exemplos de protocolos de roteamento que utilizam essa tática: *Probabilistic Routing Protocol using History of Encounters and Transitivity* (PROPHET) [25], *Meeting Visit* [11], *Shortest Expected Path Routing* (SEPR)[35], *Context-Aware Adaptive Routing* (CAR) [26].

Já na segunda, o custo de manutenção das informações de roteamento é mínimo, mas a quantidade de mensagens de dados replicados tende a ser bem maior. Generalizando, os protocolos de roteamento dessa classe têm seus algoritmos baseados em restrições de *flooding* utilizando critérios variados. Protocolos de roteamento como *Epidemic* [37], *Spray And Wait* [34], *Adaptive Potential Routing Protocol* (APRP) [27] e o Transmissão Direta [8], se baseiam nessa estratégia.

Capítulo 3

Protocolos de Roteamento Analisados

Dentre os diversos protocolos de roteamento propostos, três se destacam: *Epidemic*, *Spray and Wait* e *PROPHET*. Por serem popularmente utilizados como base de comparação ao se propor um novo protocolo [27; 37; 34; 8; 12], se tornaram objetos de estudo deste trabalho. A seguir, será percorrida uma breve análise sobre cada um.

3.1 *Epidemic Routing*

Vahdat e Becker [37] apresentaram o protocolo *Epidemic* que é baseado na teoria dos algoritmos “epidêmicos” [13; 38]. Tal qual uma epidemia, quando um nó encontra outro que ainda não contém a mensagem que ele possui, o mesmo envia uma cópia da mensagem de forma que o nó destinatário eventualmente ficará “infectado”. Este protocolo foi proposto com o objetivo de maximizar a taxa de entrega, minimizando a latência e a quantidade de recursos consumidos no envio das mensagens que se relaciona com o custo de manutenção de envio das mesmas.

A verificação de que um nó contém ou não determinada mensagem é feita de forma bem simples. Cada nó possui um sumário de vetores, e, quando dois nós se encontram no mesmo alcance de transmissão, este é repassado entre eles, em seguida, é verificada a mensagem que cada nó ainda não possui para então ser solicitado o envio dela.

Cada mensagem possui um identificador único e um contador de saltos, ou seja, uma espécie de *Time to Live* (TTL) que limita o número de recursos utilizados pelo protocolo. Quando esse número chega a um, o nó só poderá repassar a mensagem se for para o desti-

natário final. Assim, enquanto houver espaço de *buffer* disponível ou o número máximo de saltos for maior do que um, as mensagens serão disseminadas na rede. No entanto, se houver o acúmulo destas, eventualmente haverá uma sobreposição onde as mais antigas serão substituídas pelas mais recentes.

3.2 PROPHET

O protocolo de roteamento *Probabilistic Routing Protocol using History of Encounters and Transitivity* (PROPHET) [25] tem como objetivo aproximar-se de situações mais reais, baseando-se em uma probabilidade de encontro. É levada em consideração a existência de uma previsão dos lugares por onde o nó costuma passar, bem como seu tempo de permanência. Então, quando há o encontro dos nós, que não seja o destinatário, o remetente avaliará, por meio de uma função, se o próximo receberá ou não a mensagem.

A métrica probabilística utilizada é chamada de previsibilidade de entrega, onde através de uma função matemática, indica o quão provável o nó encontrado irá transmitir a mensagem para o destinatário.

Cada vez que um nó é encontrado, recalcula-se a previsibilidade de entrega. Assim, quanto maior o número de encontros entre determinados nós, maior será essa medida. Isso significa que, um nó que é frequentemente encontrado tem uma probabilidade de entrega maior do que os outros. A constante de inicialização da previsibilidade de entrega ($P_{encounter}$) serve para estabelecer um valor inicial para esta comparação. Ela é utilizada conforme mostra a equação 3.1, onde $0 \leq P_{encounter} \leq 1$:

$$P(A, B) = P(A, B)_{old} + (1 - P(A, B)_{old}) * P_{encounter} \quad (3.1)$$

Onde:

- $P(A, B)$ é a probabilidade de encontro entre um nó A e um nó B
- $P(A, B)_{old}$ é a probabilidade de encontro antes da atualização
- $(1 - P(A, B)_{old})$ é o complemento da probabilidade de encontro antes da atualização

Se um par de nós não se encontra durante um intervalo, a propensão destes serem bons encaminhadores de mensagens é mínima. Assim, os valores de previsibilidade de entrega

deles são reduzidos. A constante de envelhecimento da previsibilidade de entrega (γ) auxilia neste cálculo. Na equação 3.2, K é o número de unidades de tempo que decorreu desde a última vez que a métrica foi calculada, onde $0 \leq \gamma \leq 1$:

$$P(A, B) = P(A, B)_{old} * \gamma^K \quad (3.2)$$

A previsibilidade de entrega também possui uma propriedade transitiva, ou seja, se um nó A frequentemente encontra o nó B , e B constantemente encontra o nó C , então, C é, provavelmente, um bom nó para transmitir pacotes destinados ao primeiro e vice-versa. Chamada de constante de previsibilidade de entrega da transitividade (β_1), ela controla o quão grande deve ser o impacto da transitividade na previsibilidade de entrega. Na equação 3.3, é mostrada como é realizado o cálculo deste impacto, onde $0 \leq \beta_1 \leq 1$:

$$P(A, C) = P(A, C)_{old} + (1 - P(A, C)_{old}) * P(A, B) * P(B, C) * \beta_1 \quad (3.3)$$

Como um caso especial, o valor P para o próprio nó é sempre definido para ser 1 (i.e., $P(A, A) = 1$). Este protocolo funciona de forma semelhante ao *Epidemic*: Quando dois nós se encontram, eles trocam os seus sumários de vetores que também irão conter as respectivas previsibilidades de entrega dos nós.

É importante ressaltar que este protocolo de roteamento aplica essas diferentes equações dependendo do cenário em que ele se encontra. Dessa forma, em determinadas situações (e.g., quando possuir informações suficientes entre os nós), ele poderá utilizar mais de um cálculo para verificar se deve ou não repassar a mensagem para um dado nó.

3.3 *Spray and Wait*

Spyropoulos *et al.* [34] propõem uma família de protocolos de múltiplas cópias chamada de *Spray*. O objetivo é de gerar um pequeno número de cópias, a fim de assegurar que o número de transmissões seja pequeno e controlado. Dentre os esquemas propostos, um se destaca pela sua ampla utilização: *Spray and Wait*.

O protocolo *Spray and Wait* é uma melhoria do *Epidemic* que consiste em duas fases: na primeira, chamada de *Spray*, ele distribui N vezes a mesma mensagem, e, se não encontrar o destinatário, a segunda, denominada de estado de espera (*Wait*), é acionada. Nela é parada a

infestação dos nós, aguardando assim, que os mesmos nós a quem foram entregues as cópias das mensagens durante a primeira fase, realizem uma entrega diretamente ao destinatário.

Para a fase *Spray* existem dois subtipos de distribuição de mensagens: espalhamento binário e normal. Quando se utiliza o normal, a cada mensagem enviada, é realizado o decremento da quantidade inicial de mensagens N em um, desde que $n > 1$, senão, a mensagem só poderá ser enviada ao destinatário. Já o espalhamento binário funciona da seguinte forma: o nó criador da mensagem inicialmente começa com N cópias dela e à medida que outros nós são encontrados, o mesmo repassa $\lceil n/2 \rceil$ de suas cópias e mantém $\lfloor n/2 \rfloor$ desde que haja $n > 1$ cópias da mensagem, caso contrário, a mensagem só é repassada se encontrar o nó destinatário.

Seus criadores, durante a sua elaboração, observaram que esse novo algoritmo teve um bom desempenho por gerar uma menor carga de controle e por ser escalável. Dessa forma, caso a rede aumente de tamanho (i.e., maior número de dispositivos ativos), o desempenho do algoritmo continua funcionando uniformemente.

Estudos recentes [33] também corroboram que o esquema de infestação é bastante eficaz apenas no início, e que, a partir de certo ponto, aumenta gradativamente o *overhead*¹ na rede, acarretando perda de desempenho.

¹*overhead* é o número de cópias geradas na rede para cada mensagem criada. [27]

Capítulo 4

Modelos de Bateria

Os modelos de bateria são formulados para a captura das características de uma bateria real, considerando suas operações de carga e descarga. Estes são bastante úteis quando se almeja analisar o comportamento de descarga em projetos de sistemas alimentados por baterias.

Existem vários modelos de baterias com diferentes características e complexidades [18; 23]: analíticos, baseados em circuitos elétricos, estocásticos e eletroquímicos. Os do tipo analítico são simples e de implementação flexível, podendo assim, serem facilmente configurados para diferentes tipos de baterias. Por isso, eles foram escolhidos como base para experimentação deste trabalho.

O desempenho de uma bateria em relação ao perfil de corrente de descarga está fortemente atrelado a dois efeitos: (i) **efeito de taxa de capacidade**, que depende da capacidade atual da bateria, e da intensidade da sua corrente de descarga, pois quanto maior a corrente de descarga, menor será a capacidade disponível da bateria (medida em mAh); (ii) **efeito de recuperação**, que depende da recuperação de carga durante períodos ociosos, ocasião em que a corrente solicitada é reduzida significativamente [23]. Esta característica possibilita o aumento da vida útil da bateria. Muitos modelos analíticos conseguem representar esses efeitos.

A Figura 4.1 apresenta os diferentes estados de operação de uma bateria de forma simplificada. Observa-se na Figura 4.1 (A), o estado em que ela encontra-se totalmente carregada, com uma distribuição uniforme das *espécies eletroativas* em toda a região de comprimento w do eletrólito. O processo de descarga ocorre devido ao fluxo externo de elétrons entre os eletrodos, causando redução das suas *espécies eletroativas*, como pode ser observado na Figura

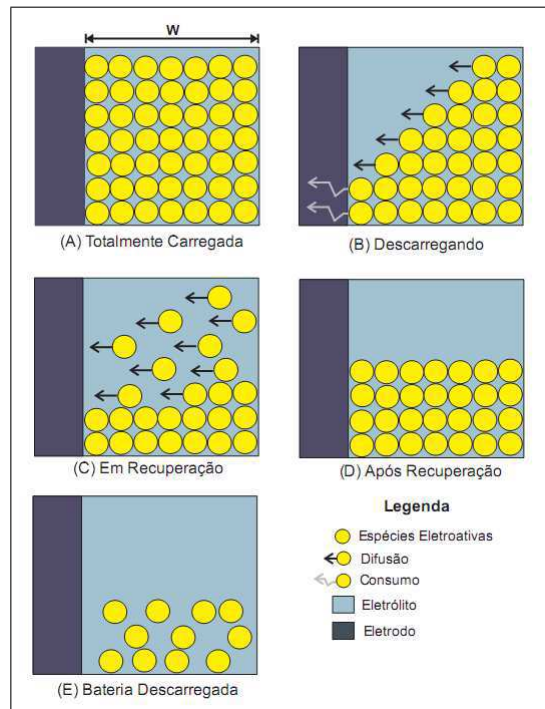


Figura 4.1: Diferentes estados de operação da bateria [32].

4.1 (B). Quando a corrente é reduzida significativamente, ocorre um processo de difusão das espécies, de forma a reequilibrar sua concentração na bateria (Figura 4.1 (C)). A recuperação da capacidade é explicada devido a esse processo de difusão [30]. A Figura 4.1 (D) mostra que, após esse processo, sua capacidade foi recuperada. Quando a concentração das espécies atingir certo limite, a bateria pára de fornecer carga ao sistema, ficando indisponível (Figura 4.1 (E)).

4.1 Modelo Linear

O modelo analítico mais simples é o linear, já mencionado anteriormente. Nele, a bateria é tratada como um recipiente linear de corrente. A equação $C = C' - I.t_d$ permite calcular a capacidade restante, C , de uma bateria. Onde C' é a capacidade no início da operação, I é a corrente constante de descarga durante a operação e t_d é o tempo de duração da operação. A capacidade remanescente é calculada sempre que a taxa de descarga mudar.

A maioria dos simuladores de rede utiliza o modelo linear de descarga de bateria, o qual não leva em consideração o efeito de relaxação desta, podendo induzir a interpretações

equivocadas na análise de protocolos, já que essa característica influencia diretamente no tempo de vida de uma rede.

4.2 Modelo Rakhmatov-Vrudhula

O modelo de bateria Rakhmatov-Vrudhula [30] é um modelo analítico mais realista que o linear, pois consegue capturar o efeito de taxa de capacidade e o efeito de recuperação. Esse modelo tenta estimar o tempo de vida de uma bateria utilizando apenas dois parâmetros específicos, conforme podemos verificar na Equação 4.1. O parâmetro α está relacionado à capacidade da bateria e o parâmetro β_2 ao comportamento não linear durante os períodos de carga e descarga.

$$\alpha = \sum_{k=1}^n 2I_{k-1}A(L, t_k, t_{k-1}, \beta_2). \quad (4.1)$$

A Equação 4.1 descreve o impacto do perfil de descarga no tempo de vida da bateria, onde I_{k-1} é a corrente de descarga durante o período $k - 1$. A função A, calcula o impacto do comportamento não linear na descarga da bateria, onde L é o tempo de vida da bateria, t_k é o tempo de duração do período k e t_{k-1} é o tempo de duração para o período $k - 1$. [29].

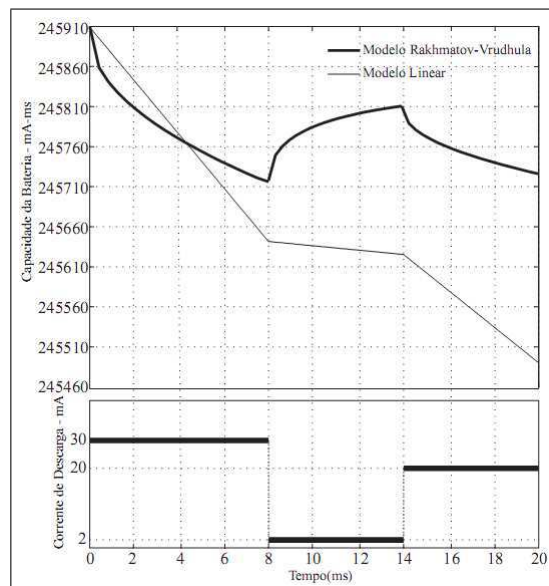


Figura 4.2: Capacidade da Bateria em Função do Perfil de Descarga [32].

O gráfico da Figura 4.2 representa o comportamento de uma bateria quanto ao consumo

de energia. No tempo entre 8 e 14 *ms*, a corrente de descarga foi reduzida de 30 para 2 *mA*, caracterizando um repouso da bateria. Neste intervalo, pode-se observar o efeito de recuperação de carga capturado pelo modelo de Rakhmatov-Vrudhula. Já o modelo linear, para o mesmo intervalo, não consegue capturar este efeito, apresentando um decréscimo linear da capacidade de bateria. Ao final do gráfico, percebe-se claramente a eficácia do modelo Rakhmatov-Vrudhula sobre o modelo linear.

No entanto, por esse modelo de bateria ser mais complexo e mais realístico do que o modelo linear, ele também é computacionalmente mais caro por empregar cálculos complexos (i.e., a função *A*) mensurados a cada operação de descarga.

A utilização deste modelo é justificada pelo fato de se obter uma maior acurácia dos dados durante as simulações, uma vez que seus valores estarão mais próximos dos obtidos num possível *testbed*.

Capítulo 5

Metodologia

Como metodologia seguida para conduzir a avaliação de desempenho dos protocolos de roteamento escolhidos, foram realizadas simulações dos mesmos em cenários pertinentes.

As simulações foram realizadas utilizando a versão 1.4 do *Opportunistic Network Environment* (ONE) [22], um simulador baseado em eventos discretos, específico para redes DTN [15; 20; 28]. Nativamente, ele contém vários protocolos de roteamento implementados (e.g., PROPHET, Transmissão Direta, *Epidemic*, *Spray and Wait*, *MaxProp*), no entanto, apenas o *Epidemic* tem uma versão que utiliza um modelo linear de bateria.

O simulador ONE permite empregar mapas reais em sua execução, através de um conjunto de dados formatados, conhecidos como o formato *Well Known Text* [1]. Por meio dele, é possível importar todas as restrições de movimento das entidades, tais como, estradas, ruas e pontos de interesse. Com esse tipo de restrição, as simulações tendem a obter resultados mais próximos da realidade, enriquecendo assim a pesquisa.

Para utilizar as restrições, o simulador contém um padrão de mobilidade chamado *Shortest Path Map-Based Movement* [21] que é derivado do modelo *Random Waypoint* [9]. Neste, nos pontos de decisões, o nó escolhe um destino aleatoriamente, e, baseando-se nas restrições do mapa, caminha até ele utilizando o menor caminho encontrado, através do algoritmo de *Dijkstra*.

Dentre os mapas do ONE foi escolhido o da porção da cidade de Helsinkí na Finlândia, que contém as restrições mencionadas acima (vide Figura 5.1).

A justificativa para escolha deste simulador baseia-se nos seguintes fatores:

- Ser específico para redes DTN.

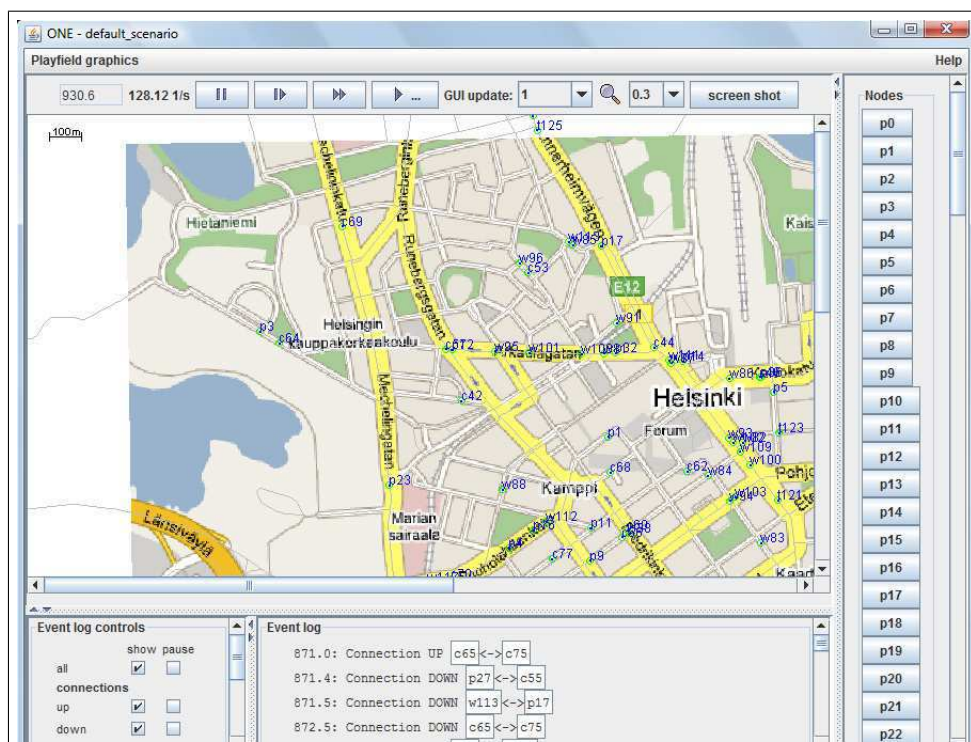


Figura 5.1: Mapa utilizado durante as simulações.

- Conter a maioria dos principais protocolos de roteamento.
- Possuir scripts para facilmente realizar a criação de cenários (vide Apêndice A).
- Ter bastante suporte técnico através de uma comunidade [2].
- Permitir programar a execução de grandes conjuntos de cenários concomitantes.
- O simulador foi desenvolvido na linguagem de programação JAVA. Isso foi um fator positivo devido à familiaridade dos pesquisadores com a mesma.

Como interface de rede das entidades nas simulações, foi utilizada a placa *Lucent Wave-LAN IEEE 802.11*. Sua escolha se deve ao fato de possuir uma gama de testes e investigações realizadas por Lauren e Martin [17], para que fossem obtidos os seus reais valores de consumo de energia em cada operação. Assim, com suas características consolidadas, a precisão dos resultados nos levaria à segurança de dados mais realísticos.

5.1 Problemas Encontrados

Durante o desenvolvimento da metodologia deste trabalho, algumas dificuldades tiveram que ser superadas para que o mesmo pudesse ser concluído.

A seguir, pontuam-se as atividades e melhorias realizadas no simulador com suas respectivas motivações e necessidades:

- Modificar o modelo linear de bateria do simulador para que este utilizasse valores de corrente em *miliampéres* no cálculo da descarga. O padrão antigo do simulador utilizava constantes sem unidade definida, o que dificultava a interpretação dos resultados e tornava difícil a comparação com os dados de outras simulações. Assim, foram trocadas os tipos das variáveis bem como foi ajustado o *script* de configuração dos cenários para recebê-las da forma correta.
- Reescrever a fórmula da descarga, para cada um dos protocolos de roteamento analisados. Anteriormente utilizava-se uma constante sem unidade de corrente definida para qualquer operação. Como mostra a Equação 5.1, a constante (k) era multiplicada pelo intervalo de tempo (δt), desconsiderando assim, a voltagem do dispositivo e a corrente de cada operação (e.g., corrente utilizada na transmissão, recepção). Assim, o protocolo de roteamento foi adaptado para que utilizasse o cálculo do modelo linear de bateria esboçado na Equação 5.2. Note que, leva-se em consideração a voltagem da interface (v) e a corrente de cada operação (i).

$$C = \delta t * k_n \quad (5.1)$$

$$C = v * i_n * \delta t \quad (5.2)$$

- Devido à inexistência do modelo de bateria Rakhmatov-Vrudhula no simulador, o mesmo foi estendido a partir de um código, escrito na linguagem de programação C++, desenvolvido e utilizado por Sausen [32]. Com as modificações e ajustes realizados foram adaptados três tipos de protocolos, variando os seus respectivos modelos de energia (gerando seis novas classes): três com os protocolos de roteamento analisados utilizando o modelo de bateria linear e os outros utilizando o modelo de bateria Rakhmatov-Vrudhula. Um exemplo dessa classe é apresentado no Apêndice B.

- Alterar as atualizações dos eventos do simulador. Para cada uma delas, o cálculo da descarga da bateria era realizado. Se uma transmissão estava sendo realizada no momento em que ocorria essa atualização, ocorria o cálculo como se o envio tivesse chegado ao seu fim, e a próxima atualização computava como se ocorresse outra transmissão produzindo resultados errôneos. Desta forma, era criada a ilusão de várias transmissões para cada intervalo de tempo t_n (vide Figura 5.2 (a)). O problema é que no modelo de bateria Rakhmatov-Vrudhula isto resultava em um grande impacto, pois sucessivas transmissões promovem um rápido esgotamento dos recursos devido à utilização contínua da bateria. O correto é que o cálculo da descarga seja realizado apenas ao término da transferência completa como é mostrado na Figura 5.2 (b).

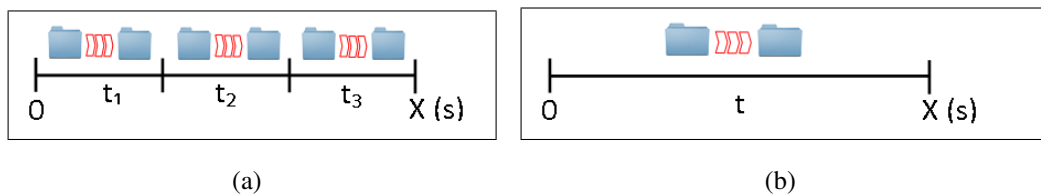


Figura 5.2: Procedimento (a) errado e (b) correto para cálculo de uma transmissão no simulador

Outros aspectos que dificultaram a execução desta pesquisa foram a ausência de padronização das métricas e escassez de informação em artigos e publicações relacionados com o tema proposto. Ao longo da pesquisa bibliográfica não foi encontrada um padrão metodológico [27; 12; 36; 19; 34; 25]. Em alguns estudos foram utilizados simuladores tal qual ONE; em outros mencionam a construção de simuladores próprios e ainda aqueles que realizaram experimentos em campos reais.

Em relação à criação dos cenários, a maioria dos trabalhos [27; 12] não especificavam importantes informações sobre estes. Fica claro, portanto, a dificuldade do embasamento teórico para futuros trabalhos nesta linha de pesquisa. Segue alguns exemplos dessas informações omitidas:

- Número de mensagens na rede;
- Tempo de simulação;

- Tamanho das mensagens;
- Alcance do rádio.

5.2 Métricas utilizadas e configurações dos parâmetros

A análise do desempenho dos protocolos de roteamento pode ser realizada de várias formas, levando em consideração o tempo de vida da rede [14]. Para as simulações executadas, foi utilizada a definição de quantidade de número de nós vivos na rede. Avaliou-se o tempo em três momentos distintos: (a) quando o primeiro nó esgota totalmente a sua bateria; ou seja, ele desliga (i.e., morre); (b) em seguida quando 35% dos nós morrem e, (c) por fim, quando 60% dos nós morrem.

O consumo de energia de cada nó foi calculado cada vez em que este enviava ou recebia uma mensagem e quando realizava a varredura da vizinhança a procura de novos nós (*Scan*). É importante ressaltar que cada cálculo dependia da respectiva função de cada modelo de bateria. O resultado era decrementado da capacidade inicial até que este zerasse ou a simulação terminasse.

Tabela 5.1: Parâmetros utilizados

Número de nós	Quantidade de Mensagens Geradas
50	100
	500
	1000
100	200
	1000
	2000
200	300
	1500
	4000

Para uma análise do modelo de bateria mais realístico, foram computados os números de mensagens retransmitidas e descartadas. Além disto, como houve um prolonga-

mento no tempo de vida da rede, verificou-se o *overhead*¹gerado nela. A taxa de entrega dos pacotes para os protocolos de roteamento escolhidos não foi analisada devido ao fato de que a literatura possui várias referências quanto a mesma (e.g., [27; 12; 36; 34; 25]) e, portanto, não foi o foco deste trabalho ressaltá-la.

A área padrão utilizada na simulação possuía um tamanho de 4500 x 3400 metros e os nós moviam-se de acordo com o padrão de mobilidade *Shortest Path Map-Based Movement*, e a quantidade destes variava entre 50, 100 e 200, divididos igualmente entre pessoas e carros. A diferença básica entre eles relacionava-se com a velocidade e o tamanho do *buffer* de armazenamento das mensagens. Para cada número de nós, escolhidos aleatoriamente, foram variadas as quantidades de mensagens geradas no sistema (vide Tabela 5.1).

Em cada cenário, demonstrado na tabela 5.1, para que os eventos ocorressem de forma mais aleatória possível, foram executadas dez rodadas de simulação utilizando diferentes sementes. Da totalidade desses cenários, calcularam-se seus respectivos desvios-padrões e intervalos de confiança (IC) utilizando o valor de 95%. Como os seus IC's foram pequenos, a quantidade de amostras mostrou-se suficiente para corroborar os resultados.

A tabela 5.2 mostra alguns dos parâmetros mais importantes e comuns para os diferentes protocolos de roteamento e seus cenários.

Como já mencionado no início deste capítulo, os valores de corrente das operações (e.g., transmissão, recepção) foram baseados no estudo utilizando a interface *Lucent WaveLAN IEEE 802.11* [17]. Também foram aplicados os valores de voltagem da interface usados nos modelos de bateria.

O conjunto de parâmetros para o modelo de bateria Rakhmatov-Vrudhula foi baseado no modelo de Sausen [32], portanto, o valor padrão é correspondente ao tipo alcalina, cuja constante de não linearidade é definida como 4074. Este número corresponde ao valor da variável β_2 utilizada no cálculo de suas operações.

Para o protocolo de roteamento *Spray and Wait*, utilizou-se o espalhamento binário, explicado no Capítulo 3. O número inicial de cópias escolhido foi de 6 mensagens, valor padrão para o simulador.

Já para o protocolo PROPHET, foram utilizados os valores sugeridos pelos autores do mesmo [25]. A constante de inicialização da previsibilidade de entrega, a constante de pre-

¹*overhead* é o número de cópias geradas na rede para cada mensagem criada. [27]

visibilidade de entrega da transitividade (β_1) e a constante de envelhecimento da previsibilidade de entrega (γ) foram configurados com os valores 0,75, 0,25 e 0,98 respectivamente.

Tabela 5.2: Parâmetros de configuração dos cenários

Atributo	Valor
Tempo de simulação	21.600 segundos
Número total de amostras	90 amostras
Capacidade inicial das baterias	3240K mAs ¹
Corrente gasta para envio e recebimento de mensagens	280 mA
Corrente gasta no modo de varredura	156 mA
Tamanho das mensagens	500 KB – 1MB ²
Tamanho do buffer	50 MB
Velocidade de transmissão	11 Mpbs ³
Alcance do rádio	250 metros
Intervalo de varredura	3 segundos ⁴
Velocidade de deslocamento	Carros – 20 a 50 Km/h e Pessoas – 1,8 a 5,4 Km/h ⁵
Voltagem da Interface	4,74V ⁶

¹O valor 3240K mAs é equivalente a 900 mAh.

²As mensagens são criadas com tamanhos aleatórios variando de 500KB a 1MB.

³Valor equivalente a 1375K no simulador.

⁴Intervalo no qual cada nó irá verificar quem são seus vizinhos.

⁵Valores padrão do simulador.

⁶Valor de voltagem da interface utilizado tanto no novo modelo linear de bateria quanto no modelo Rakhmatov.

Capítulo 6

Cenários e Análises dos Resultados

Ao longo desse capítulo serão mostrados os resultados, bem como a análise dos diferentes cenários mencionados no Capítulo 5. Os intervalos de confiança dos resultados (i.e., amostras) possuem uma diferença ínfima e, desta forma, torna-se visível a linearidade do gráfico. Seguido de cada gráfico, apresenta-se a sua respectiva tabela de valores. Desta feita, acredita-se que um menor número de discrepâncias nos dados poderá ocorrer.

Como já citado anteriormente no Capítulo 5, não é de interesse deste trabalho a análise da taxa de entrega dos pacotes para os protocolos de roteamento escolhidos, pois esta métrica foi extensamente analisada em outros trabalhos (e.g., [27; 12; 36; 34; 25]).

6.1 Tempo de vida da rede

Este cenário teve como objetivo verificar qual dos protocolos analisados terá um maior tempo de vida da rede. Leva-se em consideração que esta métrica é definida pela quantidade de nós vivos na rede e como estes se comportam ao longo das simulações quando o número de mensagens gerado nela aumenta. Para alguns gráficos não serão mostrados os desempenhos do protocolo de roteamento *Spray and Wait* utilizando o modelo de bateria Rakhmatov-Vrudhula, devido ao fato de que nas execuções dos cenários, todos os nós permaneceram com capacidade na bateria suficiente para realizar operações tanto de envio quanto de recebimento ao término das simulações.

Percebe-se que nos valores das tabelas, mostradas a seguir, o tempo de vida da rede permanece quase constante para todos os protocolos. Isto é causado pelo fato da bateria

não possuir capacidade suficiente para enviar a quantidade de mensagens configuradas neste cenário.

6.1.1 Primeiro nó sem recurso

Nesta simulação, o tempo de vida da rede é definido pelo intervalo de tempo entre o início dela até a primeira ocorrência de esgotamento total do recurso de um nó. Desta forma, analisou-se qual dos protocolos de roteamento consegue manter por mais tempo todos os nós com capacidade de bateria suficiente para realizar operações (i.e, vivo). No gráfico ilustrado na Figura 6.1 fica evidente a superioridade do modelo de bateria Rakhmatov-Vrudhula, onde é possível constatar que o ganho é mais de 100% quando comparado com o modelo linear de bateria.

Considerando apenas o modelo linear, pôde-se perceber que o protocolo de roteamento *Spray and Wait* foi superior aos outros dois protocolos. Os protocolos *Epidemic* e *PROPHET* têm um comportamento bastante similar. No cenário de 50 nós, observa-se um melhor desempenho do protocolo de roteamento *PROPHET*. No entanto, essa melhora ocorre apenas para o primeiro cenário. Nos gráficos das Figuras 6.2 e 6.3 fica evidente a superioridade do protocolo *Epidemic*, demonstrando que o protocolo *PROPHET* é afetado diretamente pelo aumento do número de nós, pois aumenta a complexidade de suas funções. Com este crescimento, se torna mais difícil prever os possíveis encontros entre os nós. Já para o *Epidemic*, o aumento do número de nós é proporcional à chance de entrega da mensagem ao destinatário, pois esta será replicada em diversos nós do cenário.

Constatou-se também que o protocolo *Spray and Wait* provou ser escalável [34], ou seja, não havia mudança de comportamento à medida que a quantidade dos nós e das mensagens aumentavam. Como já mencionado, este protocolo ao utilizar o modelo de bateria Rakhmatov-Vrudhula conseguiu finalizar as simulações com todos os seus nós ativos (i.e., a capacidade na bateria suficiente para realizar operações tanto de envio quanto de recebimento).

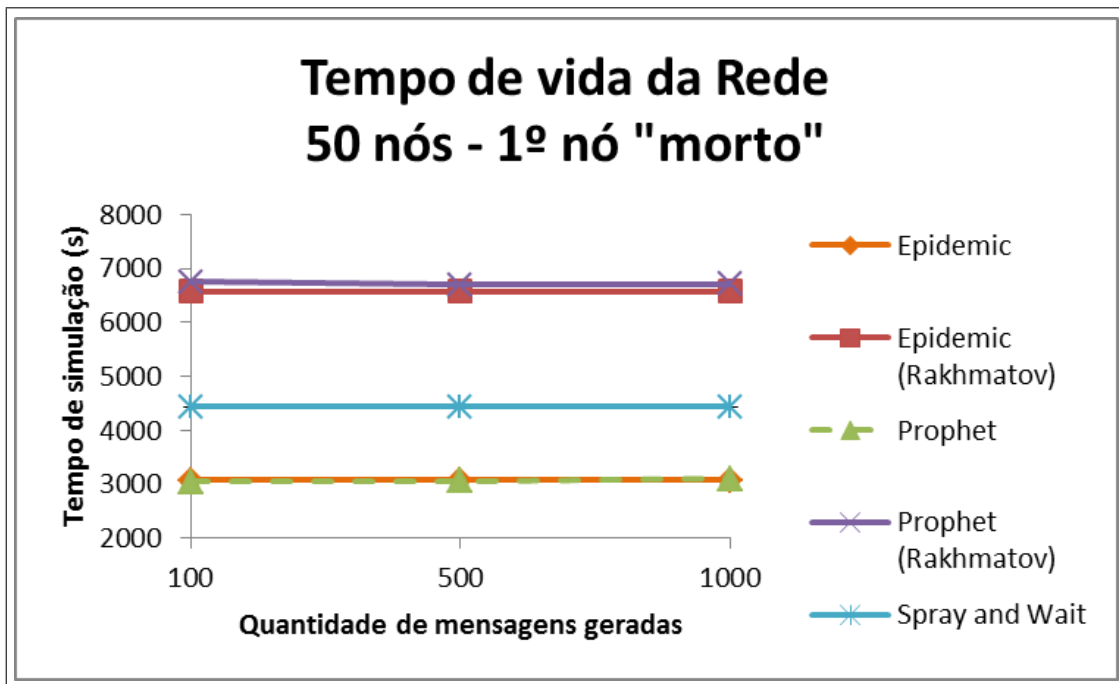


Figura 6.1: Tempo de vida da rede para 50 nós quando o primeiro nó morre.

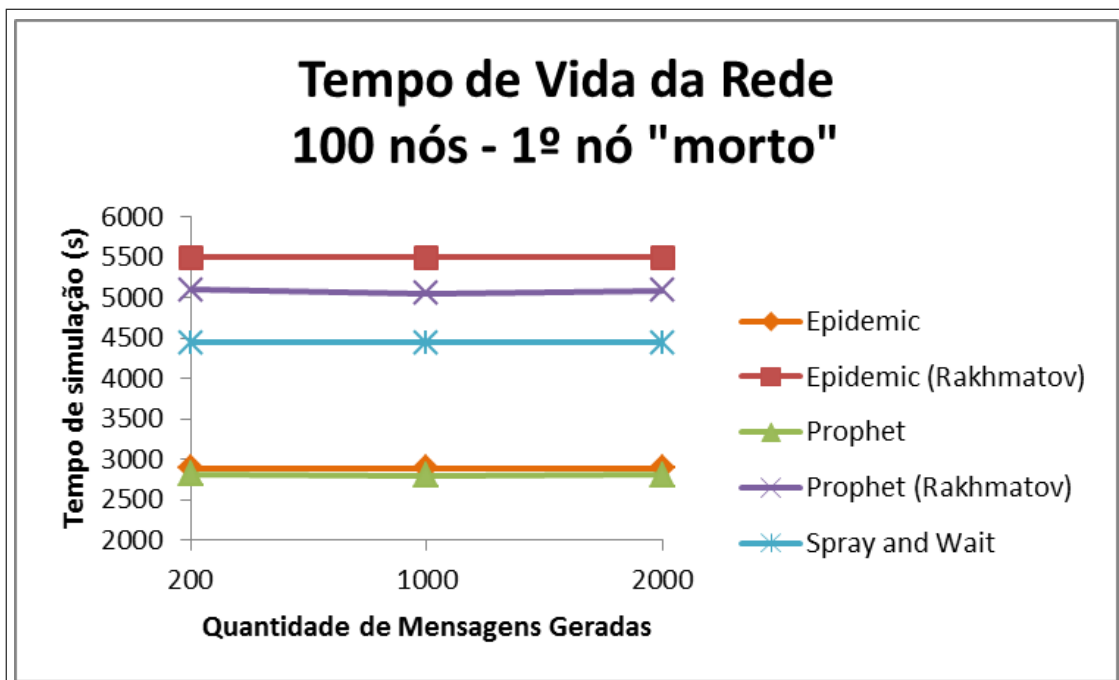


Figura 6.2: Tempo de vida da rede para 100 nós quando o primeiro nó morre.

Tabela 6.1: Tempo de vida para a rede utilizando 50 nós quando o primeiro nó morre.

Protocolos	Tempo de vida da rede (segundos)		
	100 mensagens	500 mensagens	1000 mensagens
Epidemic	3072 ± 2,026	3072 ± 2,026	3072 ± 2,026
Epidemic (Rakhmatov)	6570 ± 9,100	6570 ± 9,100	6570 ± 9,100
Prophet	3096 ± 1,987	3096 ± 2,309	3096 ± 2,204
Prophet (Rakhmatov)	6570 ± 7,037	6570 ± 8,375	6570 ± 7,829
Spray And Wait	4434 ± 0,251	4434 ± 0,251	4434 ± 0,251
Spray And Wait (Rakhmatov)	21600	21600	21600

Tabela 6.2: Tempo de vida para a rede utilizando 100 nós quando o primeiro nó morre.

Protocolos	Tempo de vida da rede (segundos)		
	200 mensagens	1000 mensagens	2000 mensagens
Epidemic	2886 ± 1,041	2886 ± 1,041	2886 ± 1,041
Epidemic (Rakhmatov)	5493 ± 3,787	5493 ± 3,787	5493 ± 3,787
Prophet	2814 ± 1,149	2796 ± 1,581	2805 ± 1,198
Prophet (Rakhmatov)	5097 ± 4,275	5049 ± 3,059	5088 ± 3,679
Spray And Wait	4440 ± 0,251	4440 ± 0,251	4440 ± 0,251
Spray And Wait (Rakhmatov)	21600	21600	21600

6.1.2 35% dos nós sem recurso

Neste segundo cenário, o tempo de vida da rede é definido pelo intervalo entre o início da simulação até o momento em que 35% dos nós tenham esgotado totalmente os recursos da bateria. Desta forma, analisou-se quais dos protocolos conseguiu manter uma maior quantidade de nós vivos em um tempo médio de simulação. No gráfico da Figura 6.4 fica evidente, mais uma vez, que há um ganho de desempenho, sendo que nesse cenário a diferença chega a ser de 200% ao utilizar o modelo de bateria Rakhmatov-Vrudhula.

Verificando apenas o protocolo *Spray and Wait*, corrobora-se sua característica escalonável, pois seus valores se mantiveram quase que constantes ao longo das simulações. Além disso, ele provou ser bastante superior quando comparado com os outros dois protocolos de

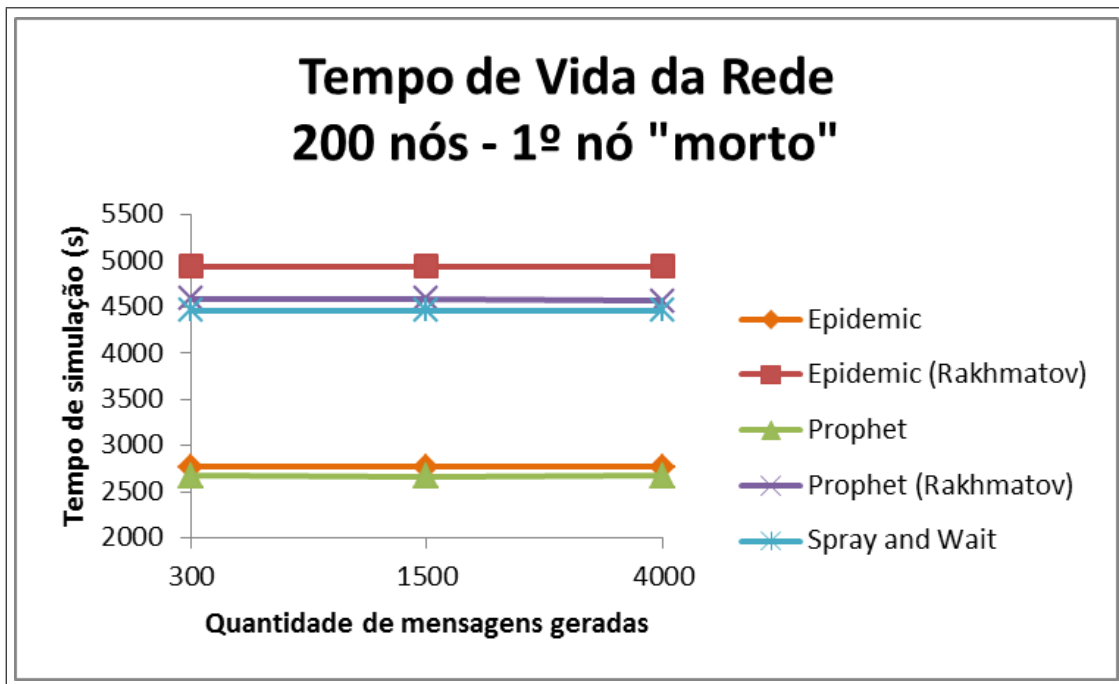


Figura 6.3: Tempo de vida da rede para 200 nós quando o primeiro nó morre

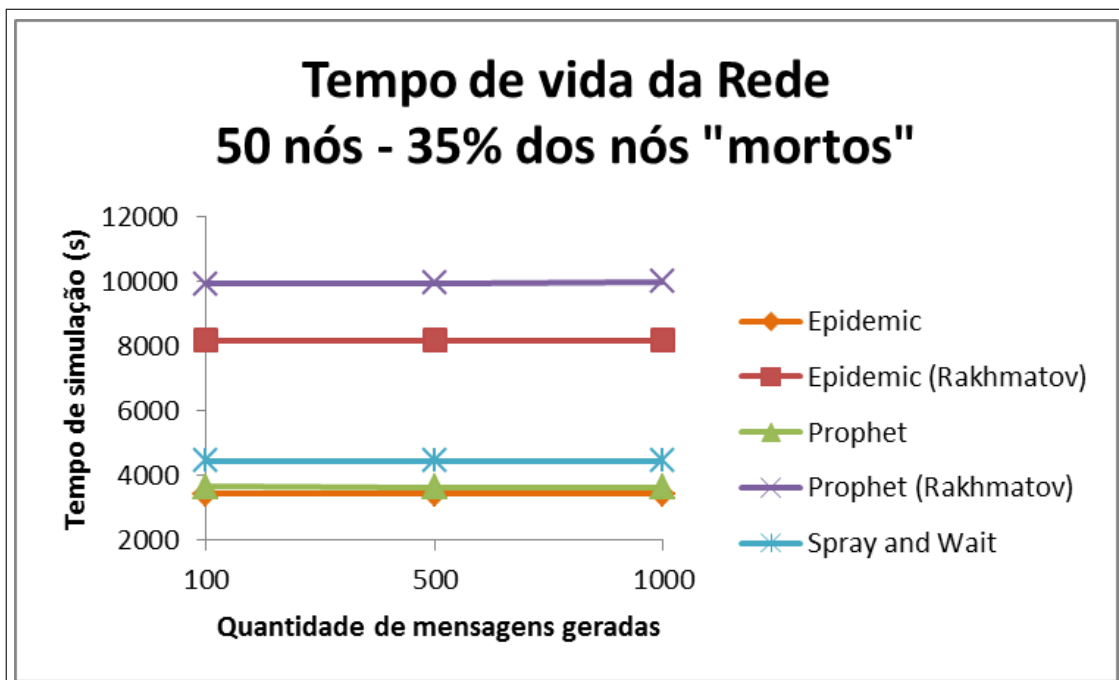


Figura 6.4: Tempo de vida da rede para 50 nós quando 35% dos nós morrem.

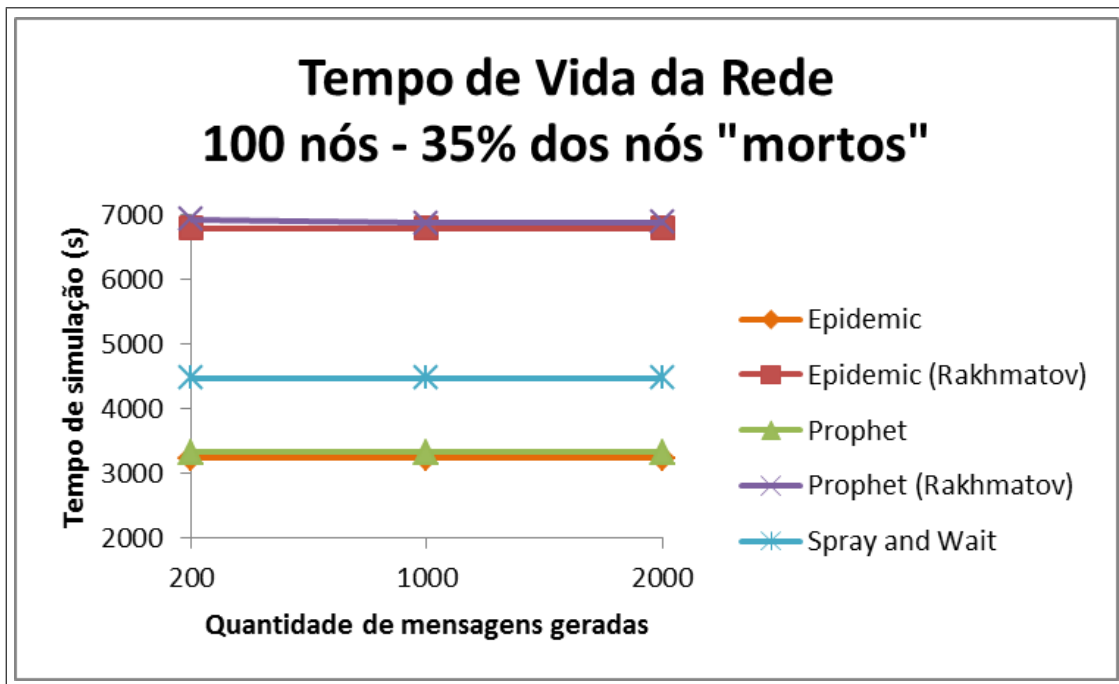


Figura 6.5: Tempo de vida da rede para 100 nós quando 35% dos nós morrem.

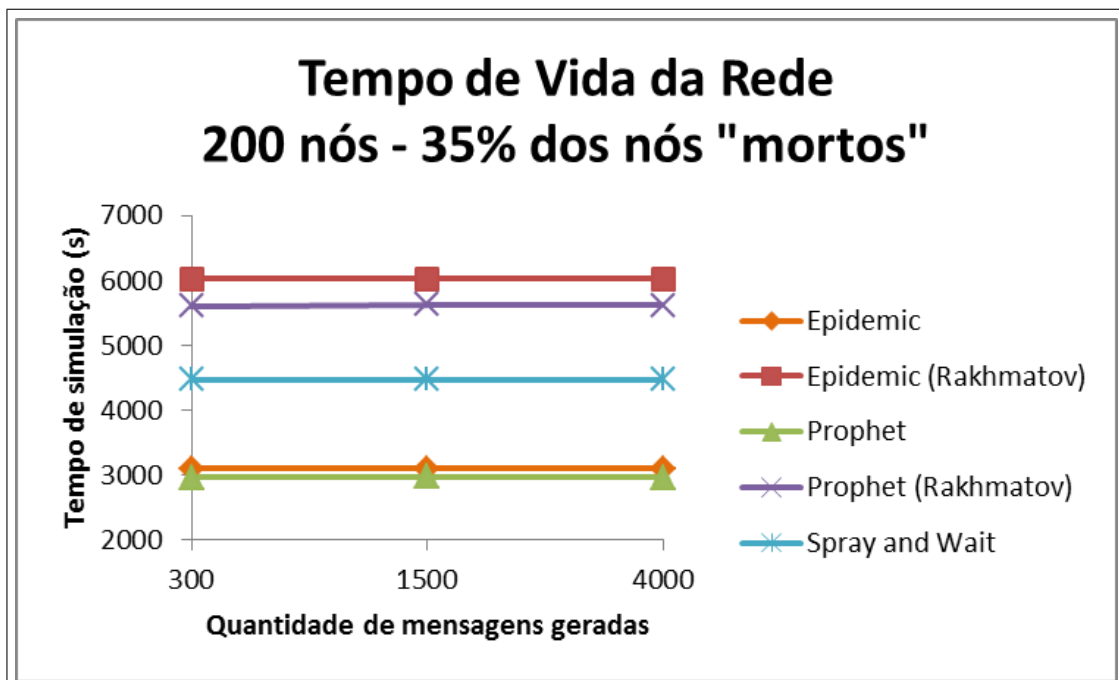


Figura 6.6: Tempo de vida da rede para 200 nós quando 35% dos nós morrem.

Tabela 6.3: Tempo de vida para a rede utilizando 200 nós quando o primeiro nó morre.

Protocolos	Tempo de vida da rede (segundos)		
	300 mensagens	1500 mensagens	4000 mensagens
Epidemic	$2766 \pm 0,963$	$2766 \pm 0,963$	$2766 \pm 0,963$
Epidemic (Rakhmatov)	$4941 \pm 2,313$	$4941 \pm 2,313$	$4941 \pm 2,313$
Prophet	$2676 \pm 0,546$	$2664 \pm 0,546$	$2673 \pm 0,591$
Prophet (Rakhmatov)	$4593 \pm 1,171$	$4587 \pm 1,358$	$4563 \pm 1,547$
Spray And Wait	$4464 \pm 0,251$	$4464 \pm 0,251$	$4464 \pm 0,251$
Spray And Wait (Rakhmatov)	21600	21600	21600

Tabela 6.4: Tempo de vida para a rede utilizando 50 nós quando 35% dos nós morrem.

Protocolos	Tempo de vida da rede (segundos)		
	100 mensagens	500 mensagens	1000 mensagens
Epidemic	$3420 \pm 0,627$	$3420 \pm 0,627$	$3420 \pm 0,627$
Epidemic (Rakhmatov)	$8163 \pm 2,030$	$8163 \pm 2,030$	$8163 \pm 2,030$
Prophet	$3645 \pm 1,380$	$3630 \pm 1,254$	$3630 \pm 1,429$
Prophet (Rakhmatov)	$9930 \pm 5,253$	$9948 \pm 5,631$	$9984 \pm 5,414$
Spray And Wait	$4440 \pm 0,251$	$4440 \pm 0,251$	$4440 \pm 0,251$
Spray And Wait (Rakhmatov)	21600	21600	21600

roteamento.

Para este cenário, foi possível constatar que o protocolo de roteamento *PROPHET* comparando-se com o *Epidemic*, obteve um melhor resultado, tanto com o modelo linear de bateria quanto para o Rakhmatov-Vrudhula. Apenas no gráfico da Figura 6.6, onde há um maior número de nós, é que se consegue melhores resultados para o protocolo *Epidemic*. Deduz-se, portanto, que o *PROPHET* consegue manter por mais tempo os seus nós com recursos de bateria e que, com o crescimento do número de nós e da quantidade de mensagens, haverá também uma diminuição do tempo de vida da rede.

Tabela 6.5: Tempo de vida para a rede utilizando 100 nós quando 35% dos nós morrem.

Protocolos	Tempo de vida da rede (segundos)		
	200 mensagens	1000 mensagens	2000 mensagens
Epidemic	3234 ± 0,469	3234 ± 0,469	3234 ± 0,469
Epidemic (Rakhmatov)	6786 ± 2,037	6786 ± 2,037	6786 ± 2,037
Prophet	3321 ± 0,564	3327 ± 0,906	3330 ± 0,793
Prophet (Rakhmatov)	6921 ± 2,915	6870 ± 2,295	6885 ± 2,233
Spray And Wait	4470 ± 0,251	4470 ± 0,251	4470 ± 0,251
Spray And Wait (Rakhmatov)	21600	21600	21600

Tabela 6.6: Tempo de vida para a rede utilizando 200 nós quando 35% dos nós morrem.

Protocolos	Tempo de vida da rede (segundos)		
	300 mensagens	1500 mensagens	4000 mensagens
Epidemic	3099 ± 0,401	3099 ± 0,401	3099 ± 0,401
Epidemic (Rakhmatov)	6027 ± 1,066	6027 ± 1,066	6027 ± 1,066
Prophet	2967 ± 0,654	2970 ± 0,686	2958 ± 0,698
Prophet (Rakhmatov)	5607 ± 2,125	5628 ± 2,298	5622 ± 1,733
Spray And Wait	4470 ± 0,251	4470 ± 0,251	4470 ± 0,251
Spray And Wait (Rakhmatov)	21600	21600	21600

6.1.3 60% dos nós sem recurso

Neste terceiro cenário, o tempo de vida da rede é definido pelo intervalo de tempo entre o início da simulação até o momento em que 60% dos nós tenham esgotado os recursos da bateria. Assim, nesta etapa verifica-se que protocolos de roteamento conseguem manter uma maior quantidade de nós com recursos de bateria por mais tempo.

Averigua-se nos gráficos das Figuras 6.7,6.8,6.9 que o protocolo *PROPHET* consegue uma melhora significativa se comparado com o *Epidemic* e com os cenários anteriores. Nesta comparação, o *PROPHET* perde inicialmente muitos nós, mas em determinado momento da simulação essa diminuição equilibra-se, mesmo com o aumento do número de nós e da quantidade de mensagens. Outro aspecto é a economia de recursos se comparado ao

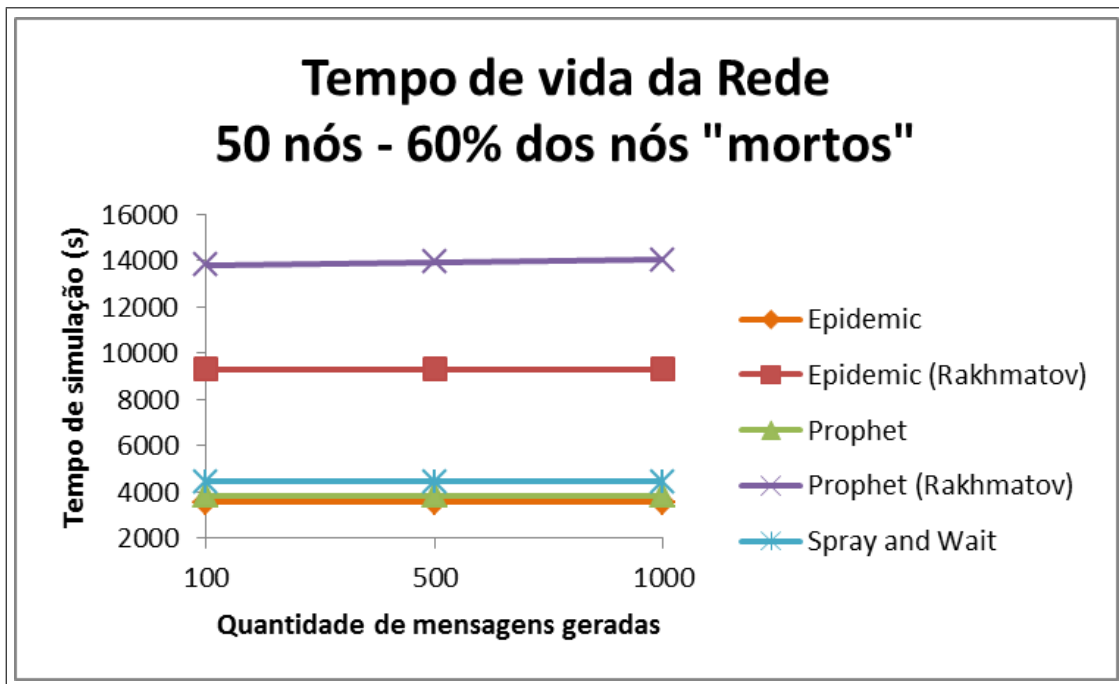


Figura 6.7: Tempo de vida da rede para 50 nós quando 60% dos nós morrem.

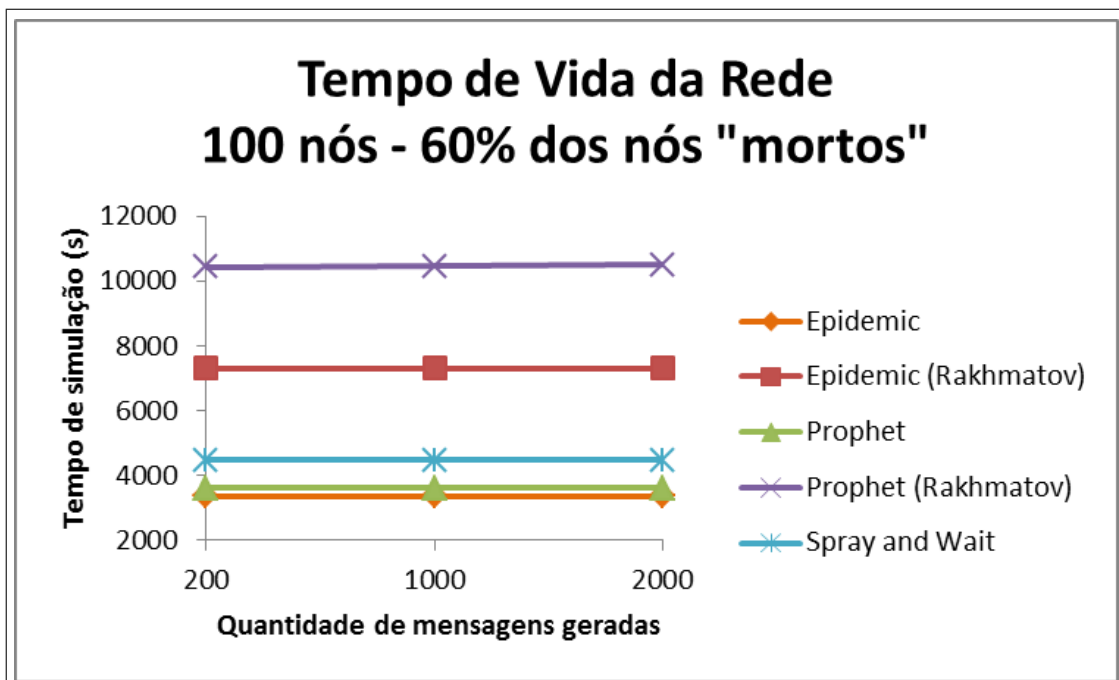


Figura 6.8: Tempo de vida da rede para 100 nós quando 60% dos nós morrem.

Tabela 6.7: Tempo de vida para a rede utilizando 50 nós quando 60% dos nós morrem.

Protocolos	Tempo de vida da rede (segundos)		
	100 mensagens	500 mensagens	1000 mensagens
Epidemic	3558 ± 0,639	3558 ± 0,639	3558 ± 0,639
Epidemic (Rakhmatov)	9312 ± 6,904	9312 ± 6,904	9312 ± 6,904
Prophet	3822 ± 0,574	3828 ± 0,501	3822 ± 0,752
Prophet (Rakhmatov)	13821 ± 7,477	13944 ± 6,473	14028 ± 5,897
Spray And Wait	4440 ± 0,251	4440 ± 0,251	4440 ± 0,251
Spray And Wait (Rakhmatov)	21600	21600	21600

Tabela 6.8: Tempo de vida para a rede utilizando 100 nós quando 60% dos nós morrem.

Protocolos	Tempo de vida da rede (segundos)		
	200 mensagens	1000 mensagens	2000 mensagens
Epidemic	3354 ± 0,546	3354 ± 0,546	3354 ± 0,546
Epidemic (Rakhmatov)	7308 ± 1,350	7308 ± 1,350	7308 ± 1,350
Prophet	3606 ± 0,376	3603 ± 0,520	3606 ± 0,546
Prophet (Rakhmatov)	10452 ± 2,399	10464 ± 3,466	10503 ± 2,268
Spray And Wait	4470 ± 0,251	4470 ± 0,251	4470 ± 0,251
Spray And Wait (Rakhmatov)	21600	21600	21600

Epidemic.

Considerando apenas estes dois protocolos é possível concluir que o *PROPHET* é a melhor escolha quando se busca um maior tempo de vida da rede, enquanto que o *Epidemic* tem um melhor desempenho para redes com tempos de vida menores.

Mais uma vez, é notável a escalabilidade do protocolo de roteamento *Spray and Wait*, que se manteve constante ao longo dos três cenários, provando ser um protocolo extremamente econômico e eficaz como poderemos comparar nas próximas análises.

É possível verificar o fato de que o protocolo *Spray and Wait* mantém praticamente o mesmo tempo de vida da rede para todos os cenários. Isso leva à conclusão de que os nós com esse tipo de protocolo têm um tempo de vida da rede superior aos outros, porém todos os nós esgotam os seus recursos praticamente no mesmo instante.

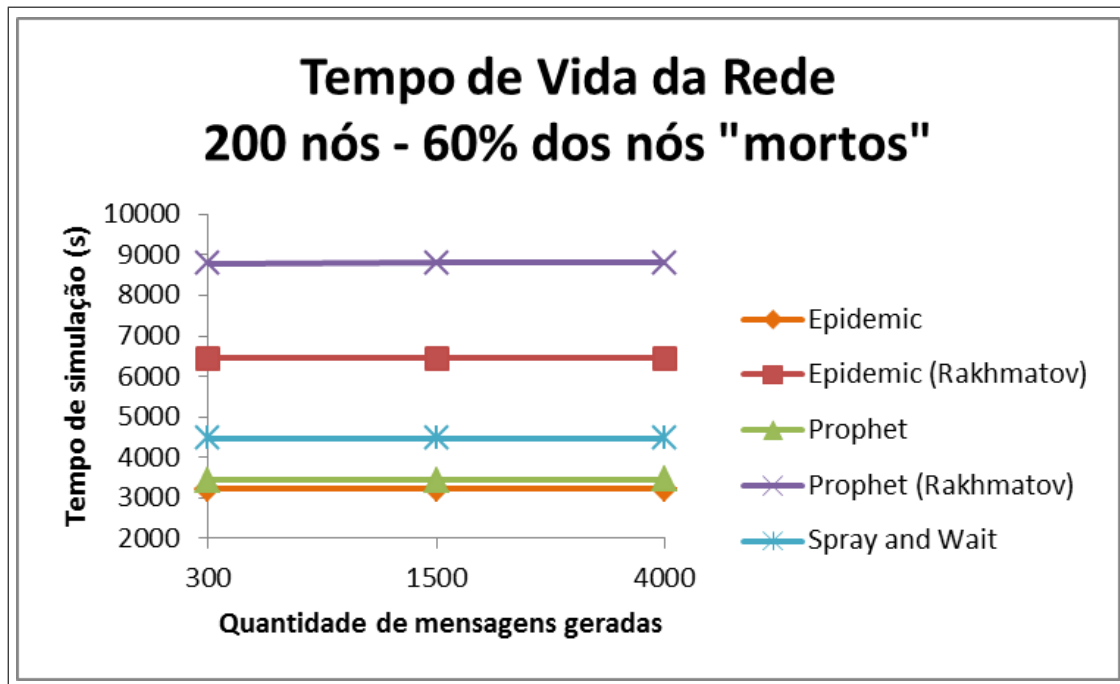


Figura 6.9: Tempo de vida da rede para 200 nós quando 60% dos nós morrem.

6.2 Quantidade de Mensagens Descartadas

O foco desta avaliação teve por objetivo analisar a quantidade de mensagens descartadas ao longo das simulações. Demonstra-se como os protocolos se comportam quando há um aumento tanto no número de nós quanto no número de mensagens.

Pela Figura 6.10, deduz-se que os protocolos de roteamento *Epidemic* e *PROPHET* têm comportamentos bastante similares. Eles possuem um número elevado de mensagens descartadas, devido à utilização completa de seus *buffers* uma vez que há uma grande quantidade de replicação das mensagens ao longo das simulações. A repetição causa grande desperdício da capacidade da bateria por parte desses protocolos. Para o *Epidemic*, esse comportamento já era esperado, já que sua política de roteamento é de replicar a mensagem a todo e qualquer nó que ainda não a tenha. O protocolo de roteamento *PROPHET*, mesmo com a sua política de roteamento de elegibilidade dos nós, provou ser bastante ineficaz em suas escolhas pois manteve o *buffer* preenchido a maior parte do tempo, causando assim, uma maior quantidade de mensagens descartadas.

Evidenciou-se um aumento significativo na quantidade das mensagens descartadas quando se utilizou o modelo de bateria Rakhmatov-Vrudhula. Para o protocolo *PROPHET*

Tabela 6.9: Tempo de vida para a rede utilizando 200 nós quando 60% dos nós morrem.

Protocolos	Tempo de vida da rede (segundos)		
	300 mensagens	1500 mensagens	4000 mensagens
Epidemic	3213 ± 0,438	3213 ± 0,438	3213 ± 0,438
Epidemic (Rakhmatov)	6447 ± 0,906	6447 ± 0,906	6447 ± 0,906
Prophet	3432 ± 0,415	3429 ± 0,401	3444 ± 0,546
Prophet (Rakhmatov)	8793 ± 2,906	8814 ± 2,309	8805 ± 2,464
Spray And Wait	4470 ± 0,251	4470 ± 0,251	4470 ± 0,251
Spray And Wait (Rakhmatov)	21600	21600	21600

Tabela 6.10: Quantidade de mensagens descartadas na rede.

Protocolos	Quantidade de mensagens		
	50 nós	100 nós	200 nós
Epidemic	45074 ± 31,081	118798 ± 34,827	270715 ± 77,187
Epidemic (Rakhmatov)	169041 ± 32,088	361443 ± 41,051	739271 ± 54,844
Prophet	35514 ± 35,439	100590 ± 41,010	246693 ± 81,580
Prophet (Rakhmatov)	199224 ± 17,343	414710 ± 19,800	843749 ± 42,508
Spray And Wait	2300 ± 3,224	2333 ± 3,224	2450 ± 3,224
Spray And Wait (Rakhmatov)	3071 ± 3,532	3100 ± 3,670	3300 ± 3,670

houve um aumento de até 342% enquanto que para o *Epidemic* foi de até 273%. Isso se deve ao fato de que, com um modelo mais realístico de bateria, a quantidade de operações realizadas são bem superiores ao do modelo linear.

Para o protocolo de roteamento *Spray and Wait*, entendeu-se que a quantidade de mensagens descartadas tornou-se bem inferior aos demais. Esse comportamento deve-se a sua política de roteamento, na qual utiliza a replicação das mensagens apenas no início do roteamento seguido do estado de espera para entrega das mensagens.

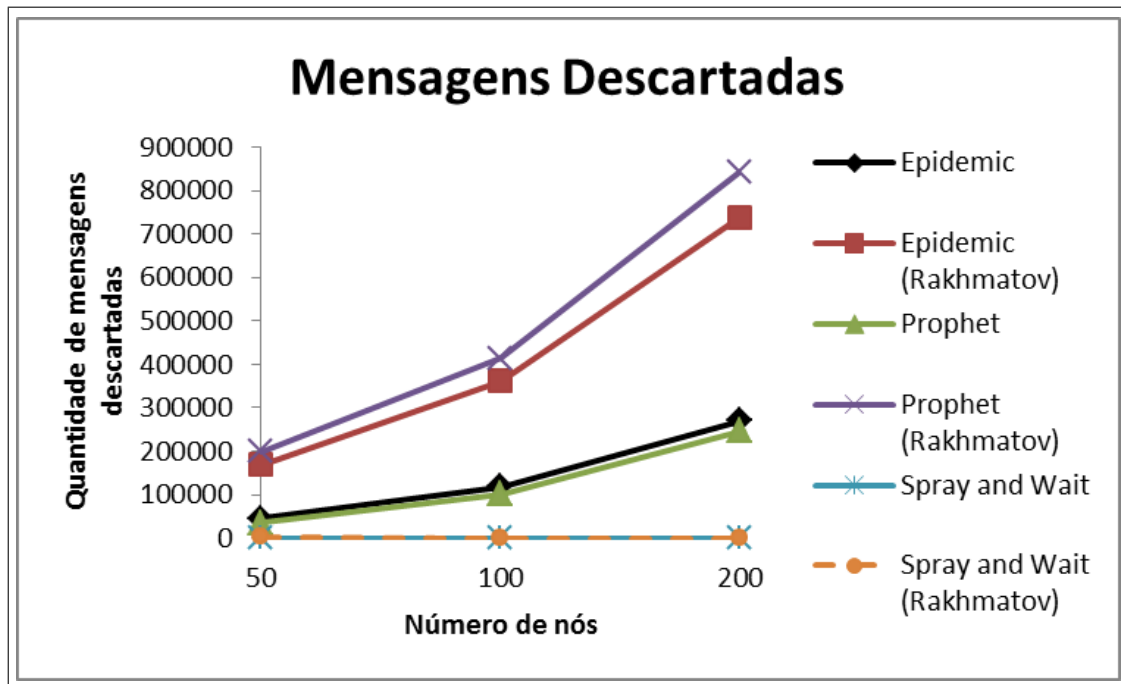


Figura 6.10: Quantidade de mensagens descartadas na rede.

6.3 Número de cópias para cada mensagem - *Overhead*

A meta deste cenário foi mensurar a quantidade de cópias de cada mensagem gerada (*overhead*¹) pelos diferentes protocolos de roteamento.

Devido à análise das mensagens descartadas, realizada no cenário anterior, acreditou-se que esse grande número poderia estar diretamente relacionado com o *overhead*. O gráfico da Figura 6.11 confirma tal especulação, demonstrando que os protocolos de roteamento *Epidemic* e *PROPHET* geraram uma maior quantidade de *overhead* das mensagens devido as suas políticas de roteamento. Ao analisarmos o gráfico da Figura 6.11 é possível notar que eles se comportam de forma similar, com uma sutil diferença demonstrando que o *Epidemic* gera mais *overhead*.

É possível verificar também que ao utilizar o Rakhmatov-Vrudhula ocorre um aumento do *overhead* de até 197% para o protocolo de roteamento *PROPHET* e para o *Epidemic* este é de até 186%. Já para o *Spray and Wait* esse aumento é desprezível.

O protocolo *Spray and Wait*, corroborando sua escalabilidade, tanto durante estes experimentos como em outros já demonstrados na literatura [34], dificilmente gera *overhead* pois

¹*overhead* é o número de cópias geradas na rede para cada mensagem criada. [27]

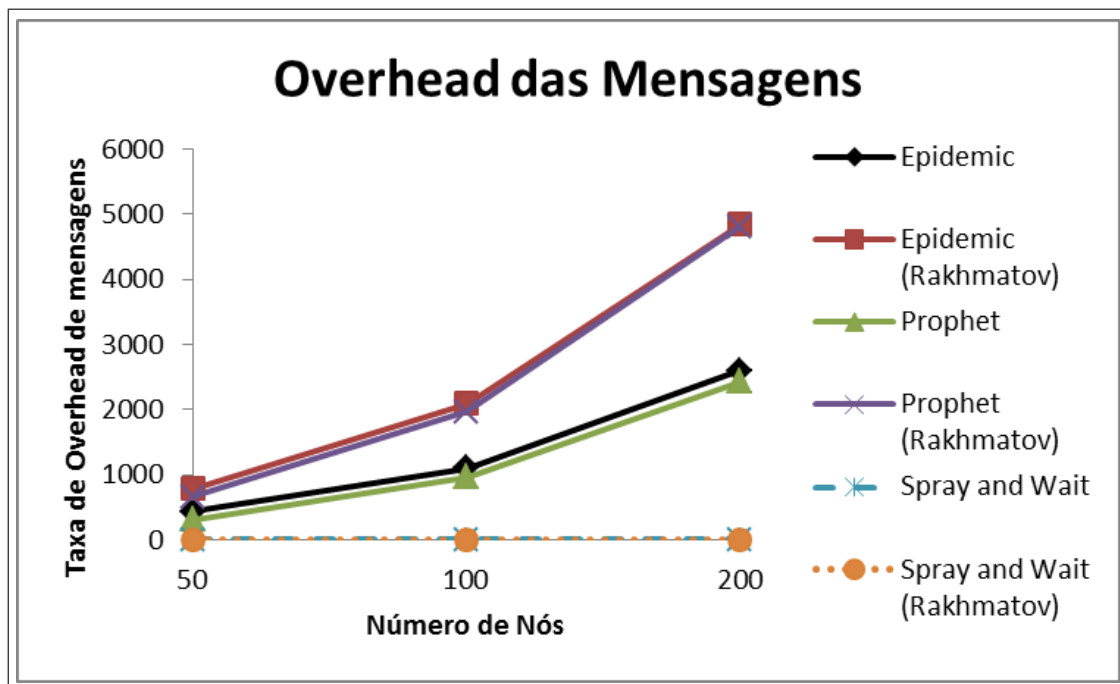


Figura 6.11: *Overhead* gerado na rede.

sua “infestação” é controlada, evitando assim a replicação exorbitante das mensagens.

Analisando o gráfico 6.11 e comparando-o com os demais gráficos de tempo de vida (e.g., Figura 6.6, Figura 6.3), é possível constatar que o *overhead* está diretamente ligado com o consumo da capacidade da bateria. Quanto maior o *overhead*, mais rápido será o esgotamento de recursos dos nós.

6.4 Quantidade de Mensagens Retransmitidas

Este cenário teve como objetivo verificar se a quantidade de mensagens retransmitidas está diretamente ligada com a métrica de *overhead* das mensagens.

Ao analisar o gráfico da Figura 6.12, constata-se que os protocolos *Epidemic* e *PROPHET* tiveram um comportamento bem parecido. À medida que o número de nós e a quantidade de mensagens aumentavam, crescia-se a quantidade de mensagens retransmitidas geradas pelo *overhead* já mostrado no gráfico da Figura 6.11.

Quanto maior a capacidade da bateria, maior será o número das mensagens retransmitidas, o que demonstra, mais uma vez, que os protocolos *Epidemic* e *PROPHET* não utilizam

Tabela 6.11: *Overhead* gerado na rede.

Protocolos	Quantidade de mensagens		
	50 nós	100 nós	200 nós
Epidemic	436 ± 0,459	1106 ± 1,314	2598 ± 2,145
Epidemic (Rakhmatov)	784 ± 0,631	2097 ± 1,171	4844 ± 4,007
Prophet	312 ± 0,429	964 ± 0,908	2438 ± 1,997
Prophet (Rakhmatov)	662 ± 0,313	1972 ± 2,073	4814 ± 4,376
Spray And Wait	6 ± 0,003	6 ± 0,007	6 ± 0,007
Spray And Wait (Rakhmatov)	6 ± 0,001	6 ± 0,001	6 ± 0,001

Tabela 6.12: Quantidade de mensagens retransmitidas na rede.

Protocolos	Quantidade de mensagens		
	50 nós	100 nós	200 nós
Epidemic	47381 ± 31,081	118798 ± 31,081	270715 ± 31,081
Epidemic (Rakhmatov)	169041 ± 32,088	361443 ± 32,088	749340 ± 32,088
Prophet	37809 ± 37,537	106277 ± 42,460	259036 ± 81,580
Prophet (Rakhmatov)	201370 ± 15,395	420566 ± 18,310	856021 ± 44,159
Spray And Wait	1334 ± 0,754	1530 ± 1,507	1793 ± 1,502
Spray And Wait (Rakhmatov)	6327 ± 0,595	6386 ± 0,404	6404 ± 0,360

de forma eficiente os seus recursos de bateria.

É possível verificar também que ao utilizar o Rakhmatov-Vrudhula ocorre um aumento da quantidade de mensagens retransmitidas de até 330% para o protocolo de roteamento *PROPHET* enquanto que para o *Epidemic* este é de até 276%.

Já o protocolo de roteamento *Spray and Wait* conseguiu gerenciar de forma eficiente os seus recursos de bateria devido a sua política de roteamento, e, dessa forma gerou uma menor redundância de mensagens na rede.

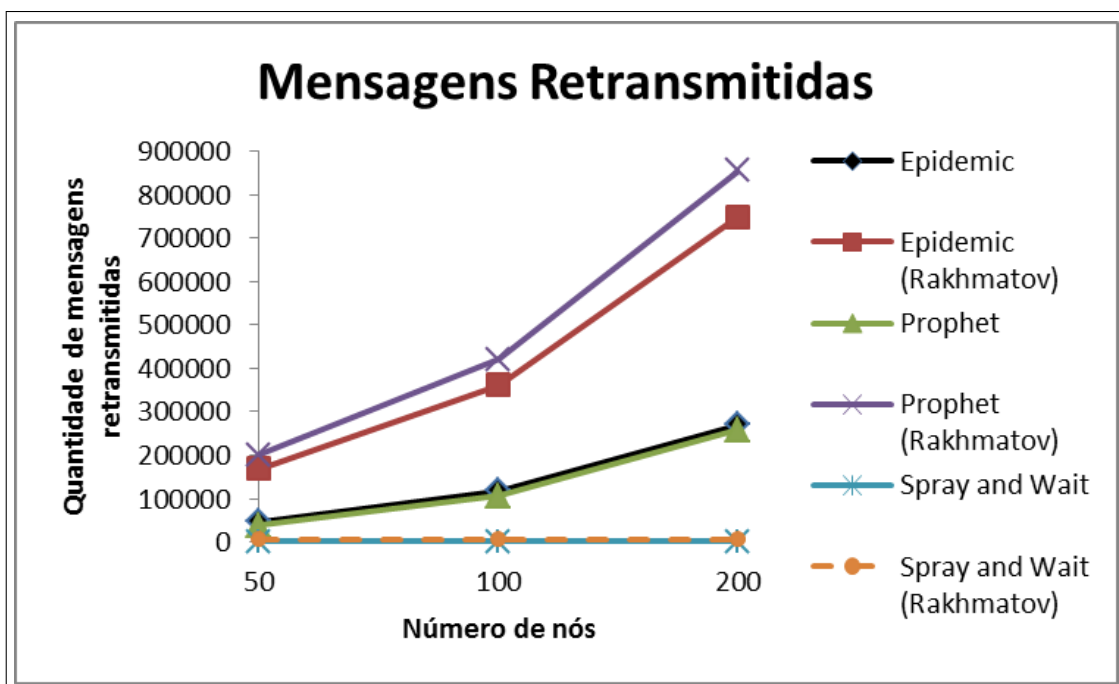


Figura 6.12: Quantidade de mensagens retransmitidas na rede.

Capítulo 7

Conclusão

Neste trabalho, foi realizada uma análise de desempenho dos protocolos de roteamento em redes tolerantes a atrasos e interrupções. Considerou-se uma métrica de desempenho que mensura o tempo de vida da rede com diferentes protocolos. Mesmo sendo aplicada em outras áreas, esta métrica nunca foi utilizada neste tipo de rede. Além desta, também foram utilizadas as métricas de quantidade de mensagens geradas na rede, descartadas e retransmitidas.

Para obter dados mais realísticos e significativos, viu-se a necessidade de usar um modelo de bateria mais realístico do que o linear. Dessa forma, o modelo de bateria Rakhmatov-Vrudhula foi estendido para o simulador (ONE) e utilizado na execução dos experimentos.

Através das simulações, ficou comprovado que o protocolo de roteamento *Spray and Wait* obteve o melhor desempenho dentre os protocolos testados. Além de que, mais uma vez, pôde-se constatar sua escalabilidade ao longo das simulações. Ao utilizar o modelo de bateria Rakhmatov-Vrudhula nos cenários, todos os nós mantiveram-se vivos até ao término da simulação; ou seja, a bateria ainda continha recursos suficientes para manter o dispositivo apto para execução de operações. Logo, espera-se que para situações em que um maior tempo de vida de rede seja necessário, esta solução será a mais adequada entre as analisadas. Utilizando o modelo linear de bateria, verificou-se que ele teve um desempenho superior aos demais protocolos analisados, porém, constatou-se que os nós esgotavam os seus recursos praticamente no mesmo instante. Também evidenciou-se que é importante utilizar um modelo mais realístico de bateria do que o modelo linear, visto que, o primeiro aumenta consideravelmente a quantidade de operações realizadas e considera propriedades importantes

da bateria.

Foi possível constatar que o *Spray and Wait* também é mais econômico do que os demais protocolos analisados, pois ele gerou um menor *overhead* na rede, teve um menor número de mensagens descartadas e retransmitidas.

O protocolo de roteamento *PROPHET* mostrou-se bastante eficaz quando utilizado em cenários em que se necessita ter uma maior quantidade de nós vivos por mais tempo. Já para cenários em que é mais importante haver uma maior troca de informações no início, o protocolo *Epidemic* provou ser uma melhor escolha.

O desempenho do protocolo *PROPHET* se assemelha ao do *Epidemic* nas métricas de quantidade de mensagens geradas na rede, descartadas e retransmitidas devido ao fato de que como ele mantém mais nós vivos ao longo do tempo, logo ele realiza mais operações gerando números maiores ou iguais aos dos gerados pelo *Epidemic*.

Importante lembrar que a métrica de taxa de entrega das mensagens por protocolo de roteamento não foi analisada pelo fato de já ter sido amplamente avaliada em outros trabalhos (e.g., [27; 12; 36; 34; 25]) e, portanto, não iria adicionar nenhum tipo de contribuição para este trabalho. Também deve-se ressaltar que, nos gráficos para o tempo de vida da rede, não houve grandes discrepâncias nos resultados obtidos devido ao fato de que ocorreu o esgotamento dos recursos das baterias dos nós antes que permitissem o envio do número desejado de mensagens do cenário.

Como trabalho futuro, pretende-se encontrar valores mais significativos para a quantidade de mensagens geradas na rede, como também incluir outros protocolos de roteamento na análise. Além disso, almeja-se realizar simulações utilizando valores de corrente e de alcance de transmissão referentes à interface *Bluetooth*TM, visto que ela é bastante utilizada atualmente em redes com características semelhantes as DTNs. Uma outra proposta seria analisar a vazão dos pacotes de forma a verificar a eficácia de cada protocolo mediante as restrições de energia.

Bibliografia

- [1] *Well Known Text*, http://en.wikipedia.org/wiki/Wellknown_text. 2009.
- [2] *User community for the ONE simulator*, <https://www.netlab.tkk.fi/mailman/listinfo/theone>. 2009.
- [3] *Delay-Tolerant Networking Architecture*, <http://tools.ietf.org/html/rfc4838>. 2009.
- [4] *Bundle Protocol Specification*, <http://tools.ietf.org/html/rfc5050>. 2009.
- [5] *TCP/IP stack protocol*, <http://www.faqs.org/rfcs/rfc1180.html>. 2009.
- [6] *InterPlanetary Internet Social Interest Group*, <http://www.ipnsig.org/>. 2009.
- [7] *Station Science - Delay Tolerant Networking (DTN)*, http://www.nasa.gov/mission_pages/station/science/experiments/DTN.html. 2009.
- [8] Muhammad Abdulla and Robert Simon. The impact of the mobility model on delay tolerant networking performance analysis. In *Annual Simulation Symposium*, pages 177–184. IEEE Computer Society, 2007.
- [9] Christian Bettstetter, Giovanni Resta, and Paolo Santi. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Trans. Mob. Comput*, 2(3):257–269, 2003.
- [10] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss. Delay-tolerant networking: an approach to interplanetary internet. *IEEE Communications Magazine*, 41(6):128–136, 2003.
- [11] Brendan Burns, Oliver Brock, and Brian Neil Levine. MV routing and capacity building in disruption tolerant networks. In *INFOCOM*, pages 398–408. IEEE, 2005.

- [12] Elizabeth. M. Daly and Mads. Haahr. Social network analysis for information flow in disconnected delay-tolerant MANETs. *IEEE Transactions on Mobile Computing*, 8(5):606–621, 2009.
- [13] Demers, Greene, Hauser, Irish, Larson, Shenker, Sturgis, Swinehart, and Terry. Epidemic algorithms for replicated database maintenance. In *PODC: 6th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 1987.
- [14] Isabel Dietrich and Falko Dressler. On the lifetime of wireless sensor networks. *ACM Transactions on Sensor Networks*, 5(1):5:1–5:, feb 2009.
- [15] Frans Ekman, Ari Keränen, Jouni Karvo, and Jörg Ott. Working day movement model. In Minkyong Kim, Cecilia Mascolo, and Mirco Musolesi, editors, *MobilityModels*, pages 33–40. ACM, 2008.
- [16] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.
- [17] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM*, pages 1548–1557, 2001.
- [18] Marijn R. Jongerden and Boudewijn R. Haverkort. Which battery model to use? *IET Software*, 3(6):445–457, 2009.
- [19] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *ASPLOS*, pages 96–107, 2002.
- [20] Jouni Karvo and Jörg Ott. Time scales and delay-tolerant routing protocols. In Mostafa H. Ammar and Konstantinos Psounis, editors, *Challenged Networks*, pages 33–40. ACM, 2008.
- [21] A. Keranen and J. Ott. Increasing reality for dtn protocol simulations. *Helsinki University of Technology, Tech. Rep., July, 2007*.

- [22] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. The one simulator for dtn protocol evaluation. In *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, New York, NY, USA, 2009. ICST.
- [23] Kanishka Lahiri, Sujit Dey, Debashis Panigrahi, and Anand Raghunathan. Battery-driven system design: A new frontier in low power design. In *Proceedings of the 2002 Asia and South Pacific Design Automation Conference, ASP-DAC '02*, pages 261–, Washington, DC, USA, 2002. IEEE Computer Society.
- [24] Joohyun Lee, Kyunghan Lee, Jaesung Jung, and Song Chong. Performance evaluation of a dtn as a city-wide infrastructure network. In *Proceedings of the 4th International Conference on Future Internet Technologies, CFI '09*, pages 35–38, New York, NY, USA, 2009. ACM.
- [25] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *Mobile Computing and Communications Review*, 7(3):19–20, 2003.
- [26] Mirco Musolesi, Stephen Hailes, and Cecilia Mascolo. Adaptive routing for intermittently connected mobile ad hoc networks. In *Proceedings of the Sixth IEEE International Symposium on World of Wireless Mobile and Multimedia Networks, WOWMOM '05*, pages 183–189, Washington, DC, USA, 2005. IEEE Computer Society.
- [27] C. M. Nunes and F. L. Dotti. Uma nova estratégia de roteamento para redes tolerantes a atrasos. *27 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 757–770, 2009.
- [28] Mikko Pitkänen, Ari Keränen, and Jörg Ott. Message fragmentation in opportunistic DTNs. In *WOWMOM*, pages 1–7. IEEE, 2008.
- [29] Daler Rakhmatov and Sarma Vrudhula. An analytical high-level battery model for use in energy management of portable electronic systems. In *Proceedings of the 2001 International Conference on Computer-Aided Design (ICCAD-01)*, pages 488–493, Los Alamitos, CA, 2001. IEEE Computer Society.

- [30] Daler Rakhmatov and Sarma Vrudhula. Energy management for battery-powered embedded systems. *ACM Trans. Embed. Comput. Syst.*, 2:277–324, August 2003.
- [31] Subir Kumar Sarkar, T. G. Basavaraju, and C. Puttamadappa. *Ad Hoc Mobile Wireless Networks: Principles, Protocols and Applications*. Auerbach Publications, Boston, MA, USA, 1st edition, 2007.
- [32] Paulo Sérgio Sausen. Gerenciamento integrado de energia e controle de topologia em redes de sensores sem fio. *PhD thesis, Universidade Federal de Campina Grande(UFCG)*, 2008.
- [33] Tara Small and Zygmunt J. Haas. Resource and performance tradeoffs in delay-tolerant wireless networks. In *WDTN '05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 260–267, New York, NY, USA, 2005. ACM Press.
- [34] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking, WDTN '05*, pages 252–259. ACM, 2005.
- [35] Kun Tan, Qian Zhang, and Wenwu Zhu. In *Shortest path routing in partially connected ad hoc networks*, volume 2, pages 1038 – 1042 Vol.2, dec 2003.
- [36] Md. Yusuf Sarwar Uddin, David Nicol, Tarek F. Abdelzaher, and Robin Kravets. A post-disaster mobility model for delay tolerant networking. In Ann Dunkin, Ricki G. Ingalls, Enver Yücesan, Manuel D. Rossetti, Ray Hill, and Björn Johansson, editors, *Winter Simulation Conference*, pages 2785–2796. WSC, 2009.
- [37] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. In *Technical Report, Duke University*, 2000.
- [38] Werner Vogels, Robbert van Renesse, and Kenneth P. Birman. The power of epidemics: robust communication for large-scale distributed systems. *Computer Communication Review*, 33(1):131–135, 2003.

-
- [39] Zhensheng Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges. *IEEE Communications Surveys and Tutorials*, 8(1-4):24–37, 2006.

Apêndice A

Script de criação dos cenários

Para a etapa de execução da análise foram utilizados *scripts* para a criação dos cenários. Estes são fornecidos pelo simulador para a configuração de acordo com as necessidades do usuário.

Em A.1 é apresentado um exemplo de código fonte do script de criação de cenários do usuário. Nele são mostrados os parâmetros necessários para sua compilação utilizando o protocolo de roteamento *Epidemic*, o uso de 50 nós e a geração de 100 mensagens.

O símbolo # representa um comentário no script, já uma variável entre os caracteres %% é uma variável que se modifica a cada execução do mesmo.

Código Fonte A.1: Script de criação de cenário

```
1 #
2 # Default settings for the simulation
3 #
4
5 ## Scenario settings
6 Scenario.name = Cenario_Router-%%Group.router%%_%%Group.nrofHosts%%_%%
   Events.nrof%%_MovementSeed-%%MovementModel.rngSeed%%
7 Scenario.simulateConnections = true
8 Scenario.updateInterval = 0.01
9
10 #OLD
11 # 43k ~= 12h
12 #Scenario.endTime = 100k
13
```

```
14 #NEW
15 # 43200s == 12h
16 # 6h
17 Scenario.endTime = 21600
18 #5h
19 # Scenario.endTime = 18000
20
21 Scenario.nrofHostGroups = 2
22
23 # "Wi-fi" interface for all nodes
24 highspeedInterface.type = SimpleBroadcastInterface
25 # Transmit speed of 2 Mbps = 250kBps *old
26 # Transmit 11Mbps = 1375k
27 highspeedInterface.transmitSpeed = 1375k
28 highspeedInterface.transmitRange = 250
29
30 ## Group-specific settings:
31 # groupID : Group's identifier. Used as the prefix of host names
32 # nrofHosts: number of hosts in the group
33 # transmitRange: range of the hosts' radio devices (meters)
34 # transmitSpeed: transmit speed of the radio devices (bytes per second)
35 # movementModel: movement model of the hosts (valid class name from
    movement package)
36 # waitTime: minimum and maximum wait times (seconds) after reaching
    destination
37 # speed: minimum and maximum speeds (m/s) when moving on a path
38 # bufferSize: size of the message buffer (bytes)
39 # router: router used to route messages (valid class name from routing
    package)
40 # activeTimes: Time intervals when the nodes in the group are active (
    start1, end1, start2, end2, ... )
41 # msgTtl : TTL (minutes) of the messages created by this host group,
    default=infinite
42
43 ## Group and movement model specific settings
44 # pois: Points Of Interest indexes and probabilities (poiIndex1,
    poiProb1, poiIndex2, poiProb2, ... ) - for
```

ShortestPathMapBasedMovement

```
45 # okMaps : which map nodes are OK for the group (map file indexes),
    # default=all - for all MapBasedMovement models
46 # routeFile: route's file path - for MapRouteMovement
47 # routeType: route's type - for MapRouteMovement
48
49
50 # common settings for all groups
51 Group.movementModel = ShortestPathMapBasedMovement
52 Group.router = EpidemicLinearRouter
53 # initial amount of energy units (900 mAh = 3240k mAs)
54 #Group.initialEnergy = 3000k, 3240k
55 Group.initialEnergy = 3240k
56 # the amount of energy taken by each scan
57 #Group.scanEnergy = 38.61
58 # the amount of energy taken each second when transferring
59 #Group.transmitEnergy = 51.47
60 # scan once every 1.5 seconds
61 Group.scanInterval = 3
62 Group.bufferSize = 50M
63 #Group.transmitRange = 250
64 #Group.transmitRange = 10
65 # transmit speed of 2 Mbps = 250kBps
66 #Group.transmitSpeed = 250k
67 # transmit speed of 11 Mbps = 1375kBps
68 #Group.transmitSpeed = 1375k
69 Group.waitTime = 0, 120
70 # walking speeds
71 Group.speed = 0.5, 1.5
72 #Group.msgTtl = 60
73
74 # All nodes have the 802.11 interface
75 Group.nrofInterfaces = 1
76 Group.interface1 = highspeedInterface
77
78 Group.nrofHosts = 50
79
```

```
80 # group1 (pedestrians) specific settings
81 Group1.groupID = p
82 Group1.nrofHosts = 25
83
84 # group2 specific settings
85 Group2.groupID = c
86 # cars can drive only on roads
87 Group2.okMaps = 1
88 # 20–50 km/h values above are in m/s
89 Group2.speed = 5.5, 13.9
90 Group2.nrofHosts = 25
91
92 # another group of pedestrians
93 #Group3.groupID = w
94 #Group3.nrofHosts = 12
95
96 # The Tram groups
97 #Group4.groupID = t
98 #Group4.bufferSize = 50M
99 #Group4.movementModel = MapRouteMovement
100 #Group4.routeFile = data/tram3.wkt
101 #Group4.routeType = 1
102 #Group4.waitTime = 10, 30
103 #Group4.speed = 7, 10
104 #Group4.nrofHosts = 5
105
106 #Group5.groupID = t
107 #Group5.bufferSize = 50M
108 #Group5.movementModel = MapRouteMovement
109 #Group5.routeFile = data/tram4.wkt
110 #Group5.routeType = 2
111 #Group5.waitTime = 10, 30
112 #Group5.speed = 7, 10
113 #Group5.nrofHosts = 5
114
115 #Group6.groupID = t
116 #Group6.bufferSize = 50M
```

```
117 #Group6.movementModel = MapRouteMovement
118 #Group6.routeFile = data/tram10.wkt
119 #Group6.routeType = 2
120 #Group6.waitTime = 10, 30
121 #Group6.speed = 7, 10
122 #Group6.nrofHosts = 5
123
124
125 ## Message creation parameters
126 # How many event generators
127 #Events.nrof = 20
128 Events.nrof = 100
129 # Class of the first event generator
130 Events1.class = MessageEventGenerator
131 # (following settings are specific for the MessageEventGenerator class)
132 # Creation interval in seconds (one new message every 25 to 35 seconds)
133 Events1.interval = 15,25
134 # Message sizes (500kB – 1MB)
135 Events1.size = 500k, 1M
136 # range of message source/destination addresses
137 Events1.hosts = 0,49
138 # Message ID prefix
139 Events1.prefix = M
140
141
142 ## Movement model settings
143 # seed for movement models' pseudo random number generator (default = 0)
144 MovementModel.rngSeed = [1;2;3;4;5;6;7;8;9;10]
145 # World's size for Movement Models without implicit size (width, height;
    meters)
146 MovementModel.worldSize = 4500, 3400
147 # How long time to move hosts in the world before real simulation
148 MovementModel.warmup = 1000
149
150 ## Map based movement –movement model specific settings
151 MapBasedMovement.nrofMapFiles = 4
152
```

```
153 MapBasedMovement.mapFile1 = data/roads.wkt
154 MapBasedMovement.mapFile2 = data/main_roads.wkt
155 MapBasedMovement.mapFile3 = data/pedestrian_paths.wkt
156 MapBasedMovement.mapFile4 = data/shops.wkt
157
158 ## Points Of Interest –specific settings
159 PointsOfInterest.poiFile1 = data/ParkPOIs.wkt
160 PointsOfInterest.poiFile2 = data/CentralPOIs.wkt
161 PointsOfInterest.poiFile3 = data/WestPOIs.wkt
162 PointsOfInterest.poiFile4 = data/shops.wkt
163
164 ## Reports – all report names have to be valid report classes
165
166 # how many reports to load
167 Report.nrofReports = 5
168 # length of the warm up period (simulated seconds)
169 Report.warmup = 0
170 # default directory of reports (can be overridden per Report with output
    setting)
171 Report.reportDir = reports/
172 # Report classes to load
173 Report.report1 = EnergyLevelReport
174 # report once every minute
175 #EnergyLevelReport.granularity = 60
176 EnergyLevelReport.granularity = 30
177 # round to two decimal's precision
178 EnergyLevelReport.precision = 2
179 Report.report2 = MessageStatsReport
180 Report.report3 = ContactTimesReport
181 ContactTimesReport.granularity = 10
182 Report.report4 = TotalContactTimeReport
183 Report.report5 = EventLogReport
184
185 ## Default settings for some routers settings
186 #ProphetRouter.secondsInTimeUnit = 30
187 #SprayAndWaitRouter.nrofCopies = 6
188 #SprayAndWaitRouter.binaryMode = true
```

```
189
190
191 ## Optimization settings — these affect the speed of the simulation
192 ## see World class for details.
193 Optimization.connectionAlg = 2
194 Optimization.cellSizeMult = 5
195 Optimization.randomizeUpdateOrder = true
196
197 ##RahkmatovSettings
198 #Valores das correntes em mA
199 RakhmatovRouter.idleScanLoadValue = 156
200 RakhmatovRouter.txRxLoadValue = 280
201
202 ## GUI settings
203
204 # GUI underlay image settings
205 GUI.UnderlayImage.fileName = data/helsinki_underlay.png
206 # Image offset in pixels (x, y)
207 GUI.UnderlayImage.offset = 64, 20
208 # Scaling factor for the image
209 GUI.UnderlayImage.scale = 4.75
210 # Image rotation (radians)
211 GUI.UnderlayImage.rotate = -0.015
212
213 # how many events to show in the log panel (default = 30)
214 GUI.EventLogPanel.nrofEvents = 30
215 # Regular Expression log filter (see Pattern-class from the Java API for
   RE-matching details)
216 #GUI.EventLogPanel.REfilter = .*p[1-9]<->p[1-9]$
```

Apêndice B

Código fonte de um dos protocolos de roteamento adaptado

Devido ao fato de que o simulador não continha nem o modelo de bateria Rakhmatov-Vrudhula nem o linear, foi realizado o desenvolvimento dos mesmos.

Em B.1 é apresentado o código fonte do protocolo de roteamento *Epidemic* utilizando o modelo de bateria linear. Este foi estendido no simulador ONE para a geração dos resultados deste trabalho.

Código Fonte B.1: Protocolo de roteamento *Epidemic* com o modelo linear de bateria

```
1 package routing ;
2
3 import java.io.BufferedWriter ;
4 import java.io.FileWriter ;
5 import java.io.IOException ;
6 import java.util.Random ;
7
8 import core.Message ;
9 import core.ModuleCommunicationBus ;
10 import core.ModuleCommunicationListener ;
11 import core.NetworkInterface ;
12 import core.Settings ;
13 import core.SettingsError ;
14 import core.SimClock ;
15 import core.SimScenario ;
```

```

16
17 public class EpidemicLinearRouter extends ActiveRouter
18         implements ModuleCommunicationListener {
19
20     private static final double VOLTS = 4.7;
21     private final String IDLE_SCAN_LOAD_ENERGY_VALUE = "
        idleScanLoadValue";
22     private final String TRANSMIT_RECEIVE_LOAD_ENERGY_VALUE = "
        txRxLoadValue";
23     private int idleScanLoadEnergyValue; //em mA
24     private int transmitReceiveLoadEnergy; //em mA
25     /** EpidemicLinear router's setting namespace ({@value})*/
26     public static final String LINEAR_ENERGY= "RakhmatovRouter";
27
28     /** Initial units of energy –setting id ({@value}). Can be either
        a
29     * single value, or a range of two values. In the latter case,
        the used
30     * value is a uniformly distributed random value between the two
        values. */
31     public static final String INIT_ENERGY_S = "initialEnergy";
32     /** Energy usage per scanning –setting id ({@value}). */
33     public static final String SCAN_ENERGY_S = "scanEnergy";
34     /** Energy usage per second when sending –setting id ({@value}).
        */
35     public static final String TRANSMIT_ENERGY_S = "transmitEnergy";
36     /** Energy update warmup period –setting id ({@value}). Defines
        the
37     * simulation time after which the energy level starts to
        decrease due to
38     * scanning, transmissions, etc. Default value = 0. If value of
        "-1" is
39     * defined, uses the value from the report warmup setting
40     * {@link report.Report#WARMUP_S} from the namespace
41     * {@value report.Report#REPORT_NS}. */
42     public static final String WARMUP_S = "energyWarmup";
43

```

```
44     /** {@link ModuleCommunicationBus} identifier for the "current
45         amount of
46         * energy left" variable. Value type: double */
47     public static final String ENERGY_VALUE_ID = "Energy.value";
48
49     private final double[] initEnergy;
50     private double warmupTime;
51     private double currentEnergy;
52     /** energy usage per scan */
53     private double scanEnergy;
54     private double transmitEnergy;
55     private double lastScanUpdate;
56     private double lastUpdate;
57     private double scanInterval;
58     private ModuleCommunicationBus comBus;
59     private static Random rng = null;
60
61     /**
62     * Constructor. Creates a new message router based on the
63     settings in
64     * the given Settings object.
65     * @param s The settings object
66     */
67     public EpidemicLinearRouter(Settings s) {
68         super(s);
69         this.initEnergy = s.getCsvDoubles(INIT_ENERGY_S);
70
71         if (this.initEnergy.length != 1 && this.initEnergy.length
72             != 2) {
73             throw new SettingsError(INIT_ENERGY_S + " setting
74                 must have " +
75                 "either a single value or two
76                 comma separated values");
77         }
78
79         this.scanInterval = s.getDouble(SimScenario.
80             SCAN_INTERVAL_S);
```

```
75
76     Settings linearEnergySettings = new Settings(
77         LINEAR_ENERGY);
78     idleScanLoadEnergyValue = linearEnergySettings.getInt(
79         IDLE_SCAN_LOAD_ENERGY_VALUE);
80     transmitReceiveLoadEnergy = linearEnergySettings.getInt(
81         TRANSMIT_RECEIVE_LOAD_ENERGY_VALUE);
82
83     if (s.contains(WARMUP_S)) {
84         this.warmupTime = s.getInt(WARMUP_S);
85         if (this.warmupTime == -1) {
86             this.warmupTime = new Settings(report.
87                 Report.REPORT_NS).
88                 getInt(report.Report.WARMUP_S);
89         }
90     }
91
92     /**
93      * Sets the current energy level into the given range using
94      * uniform
95      * random distribution.
96      * @param range The min and max values of the range, or if only
97      * one value
98      * is given, that is used as the energy level
99      */
100     protected void setEnergy(double range[]) {
101         if (range.length == 1) {
102             this.currentEnergy = range[0];
103         }
104         else {
105             if (rng == null) {
106                 rng = new Random((int)(range[0] + range
107                     [1]));
```

```
105         }
106         this.currentEnergy = range[0] +
107             rng.nextDouble() * (range[1] - range[0]);
108     }
109 }
110
111 /**
112  * Copy constructor.
113  * @param r The router prototype where setting values are copied
114  * from
115  */
116 protected EpidemicLinearRouter(EpidemicLinearRouter r) {
117     super(r);
118     this.initEnergy = r.initEnergy;
119     setEnergy(this.initEnergy);
120     this.scanEnergy = r.scanEnergy;
121     this.transmitEnergy = r.transmitEnergy;
122     this.scanInterval = r.scanInterval;
123     this.warmupTime = r.warmupTime;
124     this.comBus = null;
125     this.lastScanUpdate = 0;
126     this.lastUpdate = 0;
127     this.idleScanLoadEnergyValue = r.idleScanLoadEnergyValue;
128     this.transmitReceiveLoadEnergy = r.transmitReceiveLoadEnergy;
129 }
130
131 @Override
132 protected int checkReceiving(Message m) {
133     if (this.currentEnergy < 0) {
134         return DENIED_UNSPECIFIED;
135     }
136     else {
137         return super.checkReceiving(m);
138     }
139 }
140 /**
```

```

141     * Updates the current energy so that the given amount is reduced
        from it.
142     * If the energy level goes below zero, sets the level to zero.
143     * Does nothing if the warmup time has not passed.
144     * @param amount The amount of energy to reduce
145     */
146     protected void reduceEnergy(double load, double timer) {
147         if (SimClock.getTime() < this.warmupTime) {
148             return;
149         }
150         //Calculo linear da descarga
151         double amount = (VOLTS*load) * timer;
152         comBus.updateDouble(ENERGY_VALUE_ID, -amount);
153         if (this.currentEnergy < 0) {
154             comBus.updateProperty(ENERGY_VALUE_ID, 0.0);
155         }
156     }
157
158     /**
159     * Reduces the energy reserve for the amount that is used by
        sending data
160     * and scanning for the other nodes.
161     */
162     protected void reduceSendingAndScanningEnergy() {
163         double simTime = SimClock.getTime();
164
165         if (this.comBus == null) {
166             this.comBus = getHost().getComBus();
167             this.comBus.addProperty(ENERGY_VALUE_ID, this.
                currentEnergy);
168             this.comBus.subscribe(ENERGY_VALUE_ID, this);
169         }
170
171         if (this.currentEnergy <= 0) {
172             /* turn radio off */
173             this.comBus.updateProperty(NetworkInterface.
                RANGE_ID, 0.0);

```

```
174         return; /* no more energy to start new transfers
           */
175     }
176
177     if (simTime > this.lastUpdate && sendingConnections.size
178         () > 0) {
179         reduceEnergy(transmitReceiveLoadEnergy, simTime - this
180             .lastUpdate);
181     }
182     this.lastUpdate = simTime;
183
184     if (simTime > this.lastScanUpdate + this.scanInterval) {
185         reduceEnergy(idleScanLoadEnergyValue, this.
186             scanInterval);
187         this.lastScanUpdate = simTime;
188     }
189 }
190
191 @Override
192 public void update () {
193     super.update ();
194     reduceSendingAndScanningEnergy ();
195
196     if (isTransferring () || !canStartTransfer ()) {
197         return; // transferring, don't try other
198             connections yet
199     }
200
201     // Try first the messages that can be delivered to final
202     recipient
203     if (exchangeDeliverableMessages () != null) {
204         return; // started a transfer, don't try others (
205             yet)
206     }
207
208     this.tryAllMessagesToAllConnections ();
209 }
```

```

204
205     @Override
206     public EpidemicLinearRouter replicate () {
207         return new EpidemicLinearRouter(this);
208     }
209
210     /**
211      * Called by the combus is the energy value is changed
212      * @param key The energy ID
213      * @param newValue The new energy value
214      */
215     public void moduleValueChanged(String key, Object newValue) {
216         this.currentEnergy = (Double)newValue;
217     }
218
219
220     @Override
221     public String toString () {
222         return super.toString () + " energy level = " + this.
223             currentEnergy;
224     }
225 }

```

A classe que representa o protocolo de roteamento *Epidemic* em conjunto com o modelo de bateria Rakhmatov-Vrudhula é exibido em B.2, o qual também foi estendido no simulador ONE.

Código Fonte B.2: Protocolo de roteamento *Epidemic* com o modelo de bateria Rakhmatov-Vrudhula

```

1 package routing;
2
3 import java.util.Random;
4
5 import core.Message;
6 import core.ModuleCommunicationBus;
7 import core.ModuleCommunicationListener;

```

```
8 import core.NetworkInterface;
9 import core.Settings;
10 import core.SettingsError;
11 import core.SimClock;
12 import core.SimScenario;
13
14 public class EpidemicRakhmatovRouter extends ActiveRouter
15     implements ModuleCommunicationListener{
16
17     private final String IDLE_SCAN_LOAD_ENERGY_VALUE = "
18         idleScanLoadValue";
19     private final String TRANSMIT_RECEIVE_LOAD_ENERGY_VALUE = "
20         txRxLoadValue";
21     private int idleScanLoadEnergyValue; //em mA
22     private int transmitReceiveLoadEnergy; //em mA
23     public static final String INIT_ENERGY_S = "initialEnergy";
24     public static final String ENERGY_VALUE_ID = "Energy.value";
25     public static final String WARMUP_S = "energyWarmup";
26
27     private final double[] initEnergy;
28     private double warmupTime;
29
30     private double currentEnergy;
31     private double lastScanUpdate;
32     private double lastUpdate;
33     private double scanInterval;
34     private ModuleCommunicationBus comBus;
35     private static Random rng = null;
36
37     //RAHKMATOV
38     /** Prophet router's setting namespace ({@value})*
39     public static final String RAKHMATOV_NS= "RakhmatovRouter";
40     private static final double BETA = 4034; // constante de nao
41         linearidade (valor default beta para o Rakhmatov estimado para uma
42         bateria alcalina)
43     private double corrente; // valor da corrente na chamada da
```

```

    funcao calculaDescargaRahkmatov
41  private double tempo;           // valor do tempo na chamada da
    funcao calculaDescargaRahkmatov
42
43  private int phaseCounter;       // contador de fases do Rakhmatov
    incrementado quando existe uma troca de corrente
44  private double descargaAtual ,t1 ,t2 ,t;    // parametros Rakhmatov
45  private double la []; // [9000000];        // holds load values
46  private double sa []; // = [9000000];     // holds start values
47  private int numConexaoAnterior;
48  private int numScansAnterior;
49  private double updateEnvio;
50  private boolean mudouDeEstado;
51
52  public EpidemicRakhmatovRouter( Settings s ) {
53      super(s);
54
55      this.initEnergy = s.getCsvDoubles(INIT_ENERGY_S);
56      //Verificacao do arquivo de configuracao
57      if (this.initEnergy.length != 1 && this.initEnergy.length != 2) {
58          throw new SettingsError(INIT_ENERGY_S + " setting must have "
59              +
60              "either a single value or two comma separated
61              values");
62      }
63
64      this.scanInterval = s.getDouble( SimScenario.SCAN_INTERVAL_S);
65
66      Settings rahkmatovSettings = new Settings(RAKHMATOV_NS);
67      idleScanLoadEnergyValue = rahkmatovSettings.getInt(
68          IDLE_SCAN_LOAD_ENERGY_VALUE);
69      transmitReceiveLoadEnergy = rahkmatovSettings.getInt(
70          TRANSMIT_RECEIVE_LOAD_ENERGY_VALUE);
71
72      //Verificacao se possui periodo de Warmup e faz as devidas
73      alteracoes
74      if (s.contains(WARMUP_S)) {

```

```
70         this.warmupTime = s.getInt(WARMUP_S);
71         if (this.warmupTime == -1) {
72             this.warmupTime = new Settings(report.Report.
              REPORT_NS).
              getInt(report.Report.WARMUP_S);
73         }
74     }
75 }
76 else {
77     this.warmupTime = 0;
78 }
79 inicializarArrays();
80 }
81
82 private void inicializarArrays() {
83     //inicializar os arrays
84     la = new double[1000000];
85     sa = new double[1000000];
86     this.corrente = 0;
87     this.tempo = 0;
88     this.phaseCounter = 0;
89     numConexaoAnterior = 0;
90     updateEnvio = 0;
91     mudouDeEstado = false;
92     numScansAnterior = 0;
93 }
94
95 protected EpidemicRakhmatovRouter(EpidemicRakhmatovRouter r) {
96     super(r);
97     this.initEnergy = r.initEnergy;
98     setEnergy(this.initEnergy);
99     this.scanInterval = r.scanInterval;
100    this.warmupTime = r.warmupTime;
101    this.comBus = null;
102    this.lastScanUpdate = 0;
103    this.lastUpdate = 0;
104    this.idleScanLoadEnergyValue = r.idleScanLoadEnergyValue;
105    this.transmitReceiveLoadEnergy = r.transmitReceiveLoadEnergy;
```

```
106     inicializarArrays ();
107 }
108
109 /**
110  * Sets the current energy level into the given range using
111     uniform
112     * random distribution .
113     * @param range The min and max values of the range , or if only
114     one value
115     * is given , that is used as the energy level
116     */
117 protected void setEnergy(double range[]) {
118     if (range.length == 1) {
119         this.currentEnergy = range[0];
120     }
121     else {
122         if (rng == null) {
123             rng = new Random((int)(range[0] + range
124                 [1]));
125         }
126         this.currentEnergy = range[0] +
127             rng.nextDouble() * (range[1] - range[0]);
128     }
129 }
130
131 @Override
132 protected int checkReceiving(Message m) {
133     if (this.currentEnergy < 0) {
134         return DENIED_UNSPECIFIED;
135     }
136     else {
137         return super.checkReceiving(m);
138     }
139 }
140
141 /**
142  * Updates the current energy so that the given amount is reduced
```

```

    from it.
140  * If the energy level goes below zero, sets the level to zero.
141  * Does nothing if the warmup time has not passed.
142  * @param amount The amount of energy to reduce
143  */
144  protected void reduceEnergy(double load, double timer) {
145      if (SimClock.getTime() < this.warmupTime) {
146          return;
147      }
148      this.descargaAtual = 0;
149
150      this.tempo = timer*1000;
151      if(phaseCounter==0) {
152          t1 = 0;
153          t2 = this.tempo;
154      } else {
155          t1 = t2;
156          t2 = t1 + this.tempo;
157      }
158      for(int i=0; i<=1 ;i++){
159          if(i==0) {
160              t = t1;
161          } else {
162              t = t2;
163          }
164          this.descargaAtual = calculaDescargaRahkmatov(load ,t ,
              BETA,i); //funcao de descarga
165      }
166      this.descargaAtual = this.descargaAtual/1000;
167      this.currentEnergy = this.currentEnergy - this.descargaAtual;
168      comBus.updateDouble(ENERGY_VALUE_ID, -this.descargaAtual);
169      if (this.currentEnergy < 0) {
170          comBus.updateProperty(ENERGY_VALUE_ID, 0.0);
171      }
172  }
173
174  protected void reduceSendingAndScanningEnergy() {

```

```
175     double simTime = SimClock.getTime();
176
177     if (this.comBus == null) {
178         this.comBus = getHost().getComBus();
179         this.comBus.addProperty(ENERGY_VALUE_ID, this.currentEnergy);
180         this.comBus.subscribe(ENERGY_VALUE_ID, this);
181     }
182
183     if (this.currentEnergy <= 0) {
184         /* turn radio off */
185         this.comBus.updateProperty(NetworkInterface.RANGE_ID, 0.0);
186         return; /* no more energy to start new transfers */
187     }
188
189     if(simTime > this.lastUpdate && sendingConnections.size() !=
        numConexaoAnterior) {
190         if(isTransferring()) {
191             if(sendingConnections.size() > numConexaoAnterior
                ) {
192                 numConexaoAnterior = sendingConnections.
                    size();
193                 if(updateEnvio == 0) {
194                     updateEnvio = simTime;
195                 }
196             }
197         } else {
198             reduceEnergy(transmitReceiveLoadEnergy ,(simTime -
                updateEnvio));
199             numConexaoAnterior = sendingConnections.size();
200             updateEnvio = 0;
201             mudouDeEstado = true;
202         }
203     }
204
205     this.lastUpdate = simTime;
206
207     if (simTime > this.lastScanUpdate + this.scanInterval) {
```

```
208         /* scanning at this update round */
209         if (mudouDeEstado && numScansAnterior != 0) {
210             reduceEnergy (idleScanLoadEnergyValue , numScansAnterior);
211             mudouDeEstado = false;
212             numScansAnterior = 0;
213         } else {
214             numScansAnterior += 3;
215         }
216         this.lastScanUpdate = simTime;
217     }
218 }
219
220 @Override
221 public void update () {
222     super.update ();
223     reduceSendingAndScanningEnergy ();
224
225     if (isTransferring () || !canStartTransfer ()) {
226         return; // transferring , don't try other connections yet
227     }
228
229     // Try first the messages that can be delivered to final
230     recipient
231     if (exchangeDeliverableMessages () != null) {
232         return; // started a transfer , don't try others (yet)
233     }
234
235     this.tryAllMessagesToAllConnections ();
236 }
237
238 @Override
239 public EpidemicRakhmatovRouter replicate () {
240     return new EpidemicRakhmatovRouter (this);
241 }
242
243 @Override
244 public void moduleValueChanged (String key, Object newValue) {
```



```
244     this.currentEnergy = (Double)newValue;
245 }
246
247 @Override
248 public String toString() {
249     return super.toString() + " energy level = " + this.currentEnergy
250     ;
251 }
252 //RAHKMATOV
253
254 private double calculaDescargaRahkmatov(double load , double timer ,
255     double beta , int i) {
256
257     float ret=0;
258
259     // load-change?
260     if(phaseCounter == 0) {
261         la[phaseCounter] = load;
262         sa[phaseCounter] = timer;
263         phaseCounter++;
264     } else {
265         if ( load != la[phaseCounter-1] ) {
266             la[phaseCounter] = load;
267             sa[phaseCounter] = timer;
268             phaseCounter++;
269         }
270     }
271
272     sa[phaseCounter] = timer;
273
274     if(i==0) return (0);
275
276     // 1st period test
277     if(phaseCounter==2) {
278         ret = (float) (2*la[0]* (calculaA(timer , timer , 0, beta)
279             ));
```

```
278         return ret;
279     }
280     double temp = 0;
281     for(int k=1; k<=phaseCounter; k++) {
282         temp += 2*la[k-1] * (calculaA(timer, sa[k], sa[k-1], beta
283             ));
284     }
285     ret = (float) temp;
286     return ret;
287 }
288
289 private double calculaA(double L, double tk, double tk1, double beta)
290 {
291     // the computation of the A-Factor has the following
292     // structure:
293     //  $A = c*(1+2*sum_1)-d*(1+2*sum_2)$ 
294     // first we will compute c, sum1, d, sum_2 and then
295     // we will put it all together.
296     // sum1 and sum2 each contain an exponent value that is
297     // computed before (x1, x2)
298
299     // define local variables
300     double c, d, A, x1, x2;
301     double sum1=0;
302     double sum2=0;
303     double pi = 3.14;
304     double aux1, aux2, aux3;
305
306     // compute c
307     c = Math.sqrt(L-tk1);
308     // compute d
309     d = Math.sqrt(L-tk);
310
311     for(int m=1; m<=10; m++) {
312         aux1 = -beta*beta*m*m;
313         if(tk1==L) {
```

```
312         x1 = aux1;
313     } else {
314         x1 = aux1/(L-tk1);
315     }
316     aux2 = Math.exp(x1);
317     sum1 += aux2 - (pi*aux2)/(pi-1+(Math.sqrt(1+pi*1/-x1)));
318     if(tk==L) {
319         x2 = aux1;
320     } else {
321         x2 = aux1/(L-tk);
322     }
323     aux3 = Math.exp(x2);
324     sum2 += aux3 - (pi*aux3)/(pi-1+(Math.sqrt(1+pi*1/-x2)));
325 }
326
327 // now let's put it all together
328 A = (c*(1+2*sum1))-(d*(1+2*sum2));
329 return A;
330
331 }
332 }
```
