UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

USO DE MECANISMO DE REMOÇÃO DE MENSAGENS OBSOLETAS E GERENCIA DE BUFFER EM REDES TOLERANTES A ATRASOS E INTERRUPÇÕES

Elenilson da Nóbrega Gomes

Orientador:
Carlos Alberto Vieira Campos

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2013

USO DE MECANISMO DE REMOÇÃO DE MENSAGENS OBSOLETAS E GERÊNCIA DE BUFFER EM REDES TOLERANTES A ATRASOS E INTERRUPÇÕES

Elenilson da Nobrega Gomes

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Prof. Carlos Alberto Vieira Campos, D.Sc - UNIRIO

Prof. Igor Monteiro Moraes, D.Sc - UFF

Prof. Sidney Lucena, D.Sc - UNIRIO

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2013

Gomes, Elenilson da Nóbrega
G633 Uso de mecanismo de remoção de mensagens obsoletas e gerência de buffer em redes tolerantes a atrasos e interrupções / Elenilson da Nóbrega Gomes, 2013.
xv, 114f. ; 30 cm

Orientador: Carlos Alberto Vieira Campos.
Dissertação (Mestrado em Informática) – Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

1. Sistemas de transmissão de dados. 2. DTN. 3. Remoção de mensagens. 4. Gerenciamento de buffer. I. Campos, Carlos Alberto Vieira. II. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. III. Título.

CDD - 005.5

Dedico este trabalho a minha esposa e a meu filho, Viviane e Gabriel, pelos incontáveis incentivos dados e a pela compreensão ao longo deste trabalho

Tudo o que um sonho precisa para ser realizado é alguém
que acredite que ele possa ser realizado.

Roberto Shinyashiki

Agradecimentos

Primeiramente agradeço a Deus por me permitir chegar até aqui, depois aos meus pais que sempre me incentivaram na busca do conhecimento. A minha irmã e cunhado, Fabio e Roselene, que sempre me incentivaram. Ao meu Orientador pelo incentivo e paciência, em sempre me colocar no caminho certo. Ao amigo e incentivador Rafael Fernandes que sempre me ajudou quando precisei. Ao meu amigo professor Claudio Mahler, na época meu diretor adnministrativo, que foi o primeiro a me incentivar no mestrado. Aos professores desta instituição que me permitiram a realização deste sonho, em especial aos professores Márcio e Sidney pelo acompanhamento nos seminários e suas valiosas dicas. Ao pessoal da secretaria, em especial a Alessandra. Aos amigos de mestrado, Pablo e sobre tudo ao Carlos Bill que também sempre me ajudou quando precisei, a todos os outros que estiveram juntos nesta jornada. Ao meu atual diretor, Gustavo Castro, que além de apoiar quando precisei, ainda cedeu seu pouco tempo para ajudar na implementação do código. As minhas chefes, Roberta e Sandra, que também me apoioram e a todo os amigos do DESIT. Ao agora aluno de doutorado Juliano Fischer, que além de tirar dúvidas do simulador, ainda disponibilizou seu código da política de gerenciamento de *buffer*.

Resumo

As DTNs *(Delay Tolerant Networks)* surgiram para suprir uma necessidade existente, onde a comunicação entre os dispositivos é intermitente podendo não existir uma conexão fim-a-fim entre origem e o destino. Dependendo do protocolo de roteamento utilizado nas redes tolerantes a atrasos e interrupções, muitas mensagens são replicadas ao longo da rede pelos nós intermediários. Entretanto, depois que uma mensagem chega no seu destino, as suas réplicas continuam ocupando espaço no *buffer* dos nós e sendo repassadas na rede, o que pode prejudicar o encaminhamento ou a entrega de outras mensagens. Dentro desse contexto, neste trabalho é proposto o uso de um mecanismo de remoção de mensagens obsoletas para as DTNs. Este mecanismo visa remover as mensagens que já foram entregues no destino e continuam armazenadas nos nós intermediários. Para avaliação deste mecanismo foram implementados outros dois mecanismos, o Immune e o Immune-TX. Foi realizado uma avaliação de desempenho para verificar seus impactos nos protocolos Epidêmico, *BUBBLE Rap* e *Spray and Wait*. Além do uso de dois traços de mobilidades reais extraídos da base de dados *CRAWDAD*. Também realizou-se um estudo sobre o uso do mecanismo proposto com três políticas de gerenciamento de *buffer*, FIFO, LRF e Aleatória, onde foram realizadas avaliações de desempenho de cada política associada ao mecanismo. Os resultados mostram o bom desempenho do mecanismo de remoção proposto, também quando aplicado em conjunto com as três políticas de gerenciamento de *buffer*.

Palavras-chave: DTN, remoção de mensagens, gerenciamento de *buffer*.

Abstract

DTNs *(Delay Tolerant Networks)* have emerged to fill an existing need, where communication between devices is intermittent and can not be a connection end-to-end between source and destination. Depending on the routing protocol used, many messages are replicated throughout the network by intermediate nodes. However, even after the arrival of messages at their destination, redundant copies of delivered messages continue to circulate in the network and to occupy relevant space in the buffer of network nodes. Within this context, this work proposes the use of a mechanism for removing obsolete messages for DTNs. This mechanism aims to remove messages that have been delivered to the destination and are still stored in intermediate nodes. To evaluate this mechanism two other mechanisms were implemented, the Immune and Immune-TX. We conducted a performance evaluation to verify their impact on Epidemic protocols, Bubble Rap and Spray and Wait, besides the use of two real mobility traces extracted from the *Crawdad* database. Also held a study on the use of the proposed mechanism with three management policies buffer, FIFO, LRF and Random, in which performance reviews of each policy associated with the mechanism. The results show the good performance of the proposed removal mechanism, also when applied in conjunction with the three management policies buffer.

Keywords: DTN, removing message, management buffer

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

DARPA	Defense Advanced Research Projects Agency
DTN	Delay Tolerant Networks
DTNRG	Delay Tolerant Network Research Group
FIFO	First In, First Out
IDE	Integrated Development Environment
IPN	Internet InterPlaNetária
IRTF	Internet Research Task Force
LRF	Least Recently Forwarded
ReMO	Remoção de Mensagens Obsoletas
The ONE	The Opportunistic Network Environment
TTL	Time-To-Live

Sumário

Capítulo 1

Introdução

1.1 Considerações iniciais

Com a crescente utilização de equipamentos portáteis, para simples comunicação ou até mesmo para informações essenciais em áreas de difícil acesso, cada vez mais as redes sem fio tornam-se imprescindíveis. Infelizmente existem situações de uso de uma rede sem fio, nas quais a comunicação entre os dispositivos (nós) é intermitente podendo não existir uma conexão fim-a-fim entre a origem e o destino. Temos como exemplo a falta de conectividade em redes sem fio. Para estas situações surgem as redes tolerantes a atrasos e interrupções (*Delay Tolerant Networks – DTN*) [Fall (2003)] , pois, têm como objetivo manter a comunicação entre os dispositivos mesmo sob grandes atrasos de transmissão, altas taxas de erro e sob interrupções na conectividade da rede. Além disso, o caminho entre origem e destino pode não estar acessível o tempo todo, ou talvez nunca chegue a ficar disponível. Devido a estas características o mecanismo usado é o *store-carry-and-forward* [A. Vahdat (2000)], que faz com que as mensagens sejam

armazenadas no *buffer* para poderem ser enviadas no contato com um outro nó.

Nos casos onde a conexão fim-a-fim entre origem e destino pode não existir, normalmente o roteamento tradicional não pode ser utilizado e no seu lugar, na maioria das vezes, usa-se os protocolos de roteamento replicadores de mensagens. Esses protocolos geram várias cópias da mensagem original com o intuito de aumentar a fração de mensagens entregues e, até mesmo, diminuir o atraso na entrega de mensagens entre a origem e o destino. Como exemplo de um protocolo replicador tem-se o protocolo Epidêmico [A. Vahdat (2000)]. Outros protocolos de roteamento probabilísticos e protocolos de replicação controlada foram propostos na literatura com o intuito de diminuir a replicação de mensagens. Mesmo assim, mensagens podem ter sido replicadas na rede, e quando a mensagem original ou uma de suas réplicas chega ao destino, as outras cópias se tornarão obsoletas, mas continuarão no *buffer* dos nós ocupando espaço desnecessariamente. Assim, quando uma mensagem chegar ao seu destino, a rede pode usar esta informação para comunicar aos nós que esta mensagem foi entregue e que todos os nós que tiverem uma cópia da mesma devem removê-las, pois estas cópias não são mais necessárias e estão obsoletas nos *buffers*.

Dentro desse contexto, neste trabalho é proposto o uso de um mecanismo que se baseia na informação de confirmação de recebimento de uma mensagem pelo destino para a remoção das possíveis cópias desta mensagem que estão, de maneira obsoleta, armazenadas no *buffer* dos nós, juntamente com uma política de gerenciamento de *buffer*. Estas informações são armazenadas numa lista que deve ser trocada a cada contato efetuado entre os nós, possibilitando uma atualização de ambos, similar ao modelo matemático Vaccine [Z. J. Haas (2006)]. Nesse modelo, podemos observar que utilizando protocolos de roteamento que disseminam

mensagens pela rede é possível melhorar o uso do *buffer* em relação aos que não disseminam mensagens. Normalmente, estes mecanismos só funcionam após a entrega de mensagens no destino. Enquanto isto não ocorre, se faz necessário do uso de políticas de gerenciamento de *buffer*.

A principal contribuição desse trabalho é o uso de um mecanismo de remoção de mensagens obsoletas independente do protocolo de roteamento utilizado e de uma avaliação do uso de políticas de gerenciamento de *buffer*, que podem ser empregadas para uma menor ocupação do *buffer*. Sendo assim, este mecanismo foi implementado no simulador *The One* [Keränen et al. (2009)] levando em consideração a característica de independência do funcionamento do roteamento usado na DTN.

1.2 Motivação

Com o uso cada vez mais extenso da comunicação através de mecanismos portáteis, e as limitações impostas pela falta de infraestrutura para atender estas necessidades, surgem como uma solução para estes cenários as DTNs. Embora estas redes imponham grandes desafios, ainda assim, são capazes de atender a lacuna deixada pelas redes TCP/IP.

Em DTN, as mensagens são armazenadas para posteriormente serem encaminhadas a outro nó, a fim de entregar a mensagem com sucesso, várias cópias são encaminhadas, dependendo do roteamento empregado. Estas cópias além de ficarem ocupando espaço nos *buffers* dos nós intermediários, mesmo depois de terem sido entregues no destino, ainda consomem outros recursos como, largura de banda e energia para retransmissão [Bian and Yu (2010)].

1.3 Objetivos

O objetivo desse trabalho é implementar e avaliar um mecanismo que remova as mensagens obsoletas, mensagens que já foram entregues no destino e ainda estão ocupando espaço nos nós intermediários independente do protocolo de roteamento que esteja sendo utilizado. Além disso, é apresentada uma investigação do uso do mecanismo juntamente com três políticas de gerenciamento de *buffer*. Com isso, é esperado um benefício nas DTNs em relação a um aumento na taxa de entrega, redução na sobrecarga de mensagens, redução no atraso médio e na taxa de ocupação do *buffer*.

1.4 Metodologia de pesquisa

O método de pesquisa utilizado pelo presente trabalho é a simulação baseada em dados reais. Experimentos reais envolvendo DTN são muito custosos e complexos, inviabilizando a pesquisa.

Na formulação do problema foi feito um estudo sobre a área de DTN onde foi observado que as cópias das mensagens que permaneciam no *buffer* dos nós intermediários, após a entrega no destino, geravam um transbordo do *buffer*.

A proposta de solução foi dividida em duas partes: remoção de mensagens obsoletas e gerenciamento do *buffer*.

Adicionalmente, é necessário verificar a eficiência da solução proposta, realizando simulações em um cenário com traços reais de contato dos nós.

Para avaliação da proposta de solução são realizados dois tipos de análises. A primeira verifica a influência na rede através de quatro métricas de desempe-

nho, que são: a fração de mensagens entregues, o atraso médio, a sobrecarga de mensagens e a ocupação do *buffer*, com três diferentes protocolos de roteamento; (Epidêmico, *Spray and Wait* e *BUBBLE Rap*), tendo como objetivo investigar a eficiência do mecanismo na remoção de mensagens obsoletas proposto.

Na segunda parte da análise, será feita uma investigação sobre o uso de políticas de gerenciamento de *buffer*, juntamente com o mecanismo de remoção de mensagens obsoletas proposto neste trabalho.

Espera-se que, com uma política de gerenciamento eficaz, diminui-se ainda mais a possibilidade de enchimento do *buffer*.

1.5 Estrutura do texto da dissertação

Os capítulos estão organizados da seguinte maneira. No Capítulo 2 são contextualizadas as Redes Tolerantes a Atrasos e Interrupções. No Capítulo 3, os trabalhos relacionados são apresentados e discutidos. No Capítulo 4, é apresentado o mecanismo de remoção de mensagens obsoletas. No Capítulo 5, o mecanismo é avaliada e os resultados obtidos são mostrados e discutidos. No Capítulo 6, são avaliadas três políticas de gerenciamento de *buffer* juntamente com o mecanismo proposto. Por fim, o Capítulo 7 contém a conclusão e os trabalhos futuros.

Capítulo 2

Redes Tolerantes a Atrasos e Interrupções

Nesse capítulo serão contextualizadas as Redes Tolerantes a Atrasos e Interrupções. Inicialmente será feita uma breve introdução e depois serão abordadas sua arquitetura, alguns protocolos de roteamento, suas aplicações e finalmente é apresentado o simulador utilizado nesta dissertação.

2.1 Introdução

A Internet tornou-se o maior canal de comunicação, onde milhões de dispositivos funcionam em todo o mundo. Esta rede opera sobre a pilha de protocolos TCP/IP, graças a sua flexibilidade, eficiência e robustez, que permitem suportar diversas aplicações em diferentes cenários [Fall (2003)]. Entre estes cenários existem as redes móveis sem fio ad hoc *(MANET)*. Nas redes *MANETs*, os nós podem está em constante movimento. Eles podem se comunicar diretamente

uns com os outros, em diferentes intervalos de tempo. Nestas redes a conectividade entre equipamentos pode ser um grande desafio, principalmente quando não existe uma infraestrutura pré-existente [Krifa et al. (2012)]. Neste cenário surge a necessidade de se trabalhar em redes com conectividade intermitente ou com longos atrasos, onde o maior desafio é encontrar uma rota para encaminhar a informação até o destino. As redes com estas características são chamadas de Redes Tolerantes a Atrasos e Interrupções *(Delay Tolerant Networks - DTNs)*, onde o caminho entre origem e destino pode não estar acessível o tempo todo, ou talvez nunca chegue a estar. [Fall (2005)]

As DTNs foram originalmente desenvolvidas para uso interplanetário, através de um projeto de pesquisa da NASA para desenvolver uma Internet interplanetária, chamado Internet InterPlaNetária (IPN) [INP (2007)] . Em paralelo à estas pesquisas, um grupo investigou que os conceitos utilizados em IPN, poderiam ser aplicados em redes terrestres, em particular as redes de sensores, pois existem situações onde a conexão é intermitente e ou caminho origem e destino podem nunca existir. Assim, uma gama de aplicações de DTN pode ser usada na Terra, tais como: aplicações comerciais, aplicações científicas, militares e de serviço público, entre outras. Com isso, o *Internet Research Task Force (IRTF)* [IRTF (2007)], verificou que o IPN não seria mais o local ideal para estes trabalhos e criou um novo grupo de investigação. Esse grupo foi denominado *(Delay Tolerant Network Research Group - DTNRG)* [DTNRG (2006)], e é atualmente o principal espaço aberto para trabalhar na arquitetura e protocolos DTN [Farrell et al. (2006)].

2.2 Arquitetura DTN

A arquitetura DTN é baseada em uma abstração da comutação de mensagens, onde não existe a necessidade de estabelecer uma conexão fim-a-fim. Assim, pode-se enviar uma mensagem para outro nó e este, deverá reencaminhá-la para o nó de destino, ou para outro nó que posteriormente possa reencaminhá-la também [Fall (2005)]. A arquitetura é uma sobreposição em cima de redes regionais, incluindo a Internet. Na sua concepção, a maior preocupação era não começar do zero, mas sim aproveitar algo já existente que pudesse ser incorporado. As características de comunicação são relativamente homogêneas numa região de comunicação, onde cada região tem um único ID que é conhecido entre regiões de DTN. Os gateways têm filiação em duas ou mais regiões e são o único meio de mover mensagens entre as regiões [Voyiatzis (2012)].

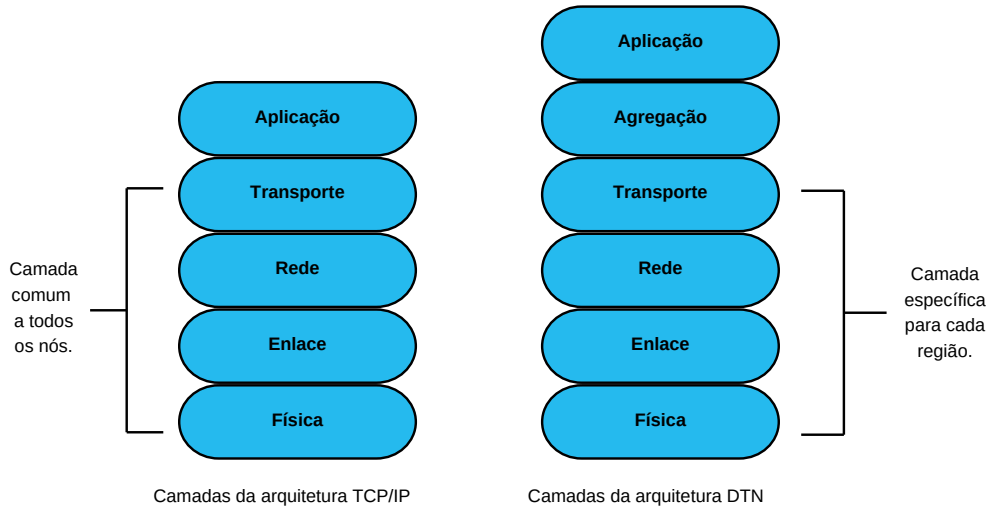


Figura 2.1: Arquitetura DTN

Na Figura 2.1 pode-se observar a existência de uma nova camada, logo abaixo da camada de aplicação, denominada camada de agregação *(Bundle Layer)*.

Nesta camada é feita a convergência entre as redes que utilizam protocolos de comunicação diferentes.

A única característica da camada de agregação é suportar o armazenamento de mensagens em trânsito [Voyiatzis (2012)]. Esta quebra de paradigma é conhecida como *store-carry-and-forward*, que ao contrário de somente armazenar a mensagem e encaminhar, em DTN o nó armazena, carrega e encaminha a mensagem chamado armazenamento persistente.

2.3 Protocolos de Roteamento

Um desafio de DTN é o roteamento, pois é preciso projetar protocolos capazes de superar não somente os atrasos longos e as frequentes desconexões, mas a possibilidade deste caminho origem e destino nunca estar disponível. Em [Moreira et al. (2012a)] é apresentado um estudo sobre a taxonomia de encaminhamento de mensagens em DTNs, onde são identificadas três grandes categorias baseadas em encaminhamento de uma única cópia, inundação de mensagens e replicação controlada. A categoria baseada em encaminhamento de uma única cópia é interessante no controle de mensagens disseminadas na rede, onde reduz o custo em replicação e ocupação de espaço de armazenamento, mas pode sofrer com um alto atraso médio, além de reduzir a fração de mensagens entregues. Já para os encaminhamentos por inundação, disseminam várias cópias na rede buscando aumentar a fração de mensagens entregues e reduzir a taxa de atraso, mas aumenta consideravelmente o espaço de armazenamento. Os encaminhamentos baseados em replicação controlada visam melhorar o desperdício de recursos utilizados na inundação, para isso são subdivididos em três partes: baseado em contatos, uti-

lização de recursos e similaridade social [Moreira et al. (2012b)].

- Baseado em contatos: frequência de contatos e histórico de contatos;
- Utilização de recursos: tempo de vida da mensagem e alocação de recursos;
- Similaridade social: detecção de comunidades, interesses comuns e popularidade do nó.

Um fato que chama a atenção é que 94% das propostas para protocolos de roteamento, estão classificadas na categoria replicadores. Abaixo são definidos alguns protocolos e suas categorias [Moreira et al. (2012a)].

Em [A. Vahdat (2000)] foi definido como um protocolo de inundação, chamado *Epidemic*. O seu funcionamento é semelhante a uma epidemia de gripe por exemplo, quando um indivíduo entra em contato com alguém contaminado pelo vírus da gripe, logo fica contaminado também. Mas, para que não haja duplicações de mensagens, visto que um nó pode encontrar com o mesmo nó no momento seguinte, cada nó tem uma lista das mensagens que estão armazenadas em seu *buffer*. Ao entrarem em contato os dois nós trocam as suas listas, caso existam mensagens que não estejam em seus *buffers*, irão trocá-las. Podem-se observar dois fatores deste protocolo, o primeiro é que como ele aumenta consideravelmente o número de cópias na rede, a probabilidade da mensagem chegar ao destino é maior. O segundo fator que é consequência do primeiro, para os nós que têm sua capacidade de armazenamento *(buffer)* limitada, como o protocolo dissemina várias cópias pela rede, o protocolo esgota mais rapidamente o espaço de armazenamento, prejudicando o funcionamento da rede.

O *Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET)* [A. Lindgren (2003)] é um exemplo de protocolo probabilístico,

classificado como replicador baseado em contatos. Utiliza duas métricas para calcular a probabilidade: história de encontros e a transitividade dos nós. Um nó ao entrar em contato com outro nó é calculada a probabilidade deste nó encontrar com o nó destino. Caso esta probabilidade seja maior que a sua, ele irá encaminhar uma cópia da mensagem para este nó e caso tenha espaço de armazenamento irá manter uma cópia da mensagem, para um possível contato com o nó destino e assim entregá-la. Para o cálculo desta probabilidade também leva-se em conta a transitividade dos nós.

Em [T. Spyropoulos (2008)] é apresentado outro protocolo de roteamento, denominado *Spray and Wait*. Este protocolo é considerado replicador de utilização de recursos. Para este protocolo o processo de encaminhamento pode ser dividido em duas partes: (i) a fase de pulverização , onde L cópias de uma mensagem inicialmente são transmitidas para L vizinhos e (ii) na fase de espera, estes L vizinhos transmitem uma cópia da mensagem apenas quando encontram o destino. Qualquer nó que possui $n > 1$ cópias da mensagem e encontra outro nó sem cópias dessa mensagem, repassa $\lfloor n/2 \rfloor$ de suas cópias e mantém $\lceil n/2 \rceil$; até que haja somente uma cópia. Então ele troca para o roteamento de "transmissão direta", ou seja, irá encaminhar a mensagem apenas para o seu destino.

O protocolo *BUBBLE Rap* em [Hui et al. (2011)] tem na sua concepção usar as relações sociais como melhor decisão no encaminhamento das mensagens. É classificado como replicador com similaridade social. As relações podem variar muito mais lentamente do que a topologia e, portanto, pode ser usado para melhores decisões de encaminhamento. O seu funcionamento é baseado nas métricas de centralidade e comunidade, onde cada nó participa de uma comunidade e sua centralidade é proporcional a sua popularidade. Além disso, também tem uma

centralidade global na rede. Se um nó tem uma mensagem destinada para outro nó, primeiramente é feito uma consulta hierárquica na árvore usando o ranking global, até atingir um nó que está na mesma comunidade, do nó de destino. Depois, o sistema de classificação local é usado em vez do ranking mundial, e a mensagem é encaminhada para o nó de maior ranking local para ser encaminhada até que o destino seja alcançado ou a mensagem expire. Possui duas fases: *Bubble-up* na comunidade global e *Bubble-up* na comunidade local, sempre escolhendo nós com maior centralidade para encaminhamento de mensagens. Para reduzir custos com envio de mensagens, quando uma mensagem é entregue à comunidade que o nó destino pertence, o nó que transportou a mensagem poderá excluí-la, evitando uma maior disseminação.

Em [Spyropoulos et al. (2007)] é descrito o protocolo *Spray And Focus* que segundo seu autor é classificado como um protocolo encaminhador. Se assemelha ao algoritmo do protocolo *Spray-And-Wait* [T. Spyropoulos (2005)] na fase Spray (*spray phase*), onde pulveriza, ou seja, na distribuição de mensagens. Quando uma nova mensagem é gerada em um nó fonte, também cria-se "L" "fichas de encaminhamento" (*token*) para esta mensagem, quantidade de cópias que podem ser geradas. Um *token* de encaminhamento implica que o nó que o possui pode gerar e enviar uma cópia adicional da mensagem, de acordo com as seguintes regras: a) Idêntico ao protocolo Epidêmico em [A. Vahdat (2000)], no qual trocam seus vetores e verificam quais mensagens elas têm em comum, para depois trocarem somente as mensagens incomuns; b) Mantém um contador em cada nó, que é decrementado do número de mensagens, se encontrar com algum nó que ainda não tenha uma cópia, *Binary Spraying* [T. Spyropoulos (2005)]; c) Quando um nó só tem mais um encaminhamento, então ele só pode enviar esta mensagem

de acordo com as regras da fase “Focus”. Ao contrário do *Spray and Wait*, onde na fase *Wait* encaminha apenas ao destino, na fase Focus uma mensagem pode ser encaminhada para outro nó diferente, de acordo com um determinado critério de encaminhamento. Especificamente, estas decisões são tomadas com base em um conjunto de registros, onde cada registro contém o tempo decorrido desde o último encontro da rede. Será encaminhado, caso este outro nó tenha tido um contato mais recente com o nó destinatário.

2.4 As aplicações em DTN

As DTNs surgiram para atender a uma necessidade de comunicação, onde normalmente outras redes não conseguem oferecer seus recursos. Existem várias áreas fazendo uso desta arquitetura.

Em [Silva et al. (2012)], foi proposto à modelagem de um sistema distribuído, denominado Lapras, na área de Educação a Distância. Se utiliza de uma infraestrutura de rede CoDPON *Continuous Displacement Plan Oriented Network* inspirada nas redes DTN. Tal arquitetura tem por objetivo a interligação de localidades desprovidas de inclusão tecnológica, tal como as comunidades ribeirinhas da Amazônia, com grandes centros urbanos. Esta proposta utiliza os barcos que transitam pela malha fluvial local, para conexão com os portos. Uma das vantagens desta rede em relação a tradicional é que, por ser baseada em redes DTN, ela prevê situações de indisponibilidade de nós, que podem ser barcos ou portos.

Outro trabalho desenvolvido com o uso de DTN, capaz de prover conectividade em partes remotas da Índia, Camboja, Ruanda e Costa Rica a um custo bem menor do que o tradicional, oferecido por soluções de telefone fixo. É o

trabalho desenvolvido em DakNet [Pentland et al. (2004)], que utilizam ônibus e motos como canais de comunicação, à medida que se movimentam. Um ônibus público carrega um ponto de acesso móvel, nas vilas existem os *Kiosk*, locais de armazenamento, e na cidade fica um hub com acesso a Internet. Conforme o ônibus se move pelas vilas, recebe e transmite os dados armazenados no *Kiosk*, ao chegar à cidade estes dados são novamente trocados com o hub, que tem acesso a Internet.

O sistema ZebraNet [Juang et al. (2002)] é uma pesquisa científica da área de biologia, que utiliza colares customizados em animais (nós), numa grande área selvagem. Nestes colares incluem sistema de posicionamento global (GPS) , memória *flash*, *transceivers* sem fio e uma pequena CPU. Como não existe cobertura nesta área, nem mesmo de telefonia celular, os colares (nós) são usados para armazenar as informações de localização de cada zebra através do GPS e que são transmitidas salto a salto até a estação base.

No capítulo seguinte são apresentados alguns trabalhos que procuraram resolver o problema do transbordo do *buffer* em redes tolerantes a atrasos e interrupções.

Capítulo 3

Trabalhos Relacionados

Esse capítulo tem como foco o levantamento na literatura sobre a remoção de mensagens em DTNs. Aqui são apresentados trabalhos que procuraram resolver o problema do transbordo do *buffer* usando mecanismos que buscam eliminar mensagens, que já foram entregues no destino e que ainda permanecem ocupando espaço nos nós intermediários. Estes mecanismos podem ser empregados em protocolos de roteamento já existentes, mudando a sua concepção, ou através de novos protocolos que visam diminuir a quantidade de cópias disseminadas na rede.

3.1 Mecanismos de Remoção de Mensagens

Um estudo sobre um novo modelo de redes para aplicações biológicas é o tema dos trabalhos em [Small and Haas (2003)[, [Z. J. Haas (2006)], motivado pelo fato de que estudos sobre mamíferos marinhos, seu ambiente e sua preservação, eram uma das principais preocupações para muitos grupos, tais como a *"Save*

the Whales". Neste trabalho é apresentado o modelo SWIM *(Shared Wireless Infostation Model)* no qual é feita uma avaliação de um novo paradigma da comunicação, que ao contrário de ligações estabelecidas fim-a-fim, como em uma rede de celular. O SWIM cria o que se refere a comunicações virtuais em uma rede DTN. O SWIM explora a mobilidade do sistema, produzindo os links virtuais para descarregar dados quando um nó está perto de um posto de coleta. As relações virtuais são realizadas por ligações físicas dos nós (baleias) que, quando entram em contato, propagam as informações para os destinos através da replicação das informações. Primeiramente, foi apresentado um estudo sobre o tempo de vida dos pacotes (TTL). Paralelamente, foi examinado o uso do armazenamento no sistema, desenvolvendo modelos matemáticos para cinco diferentes métodos de descarte de mensagens tidas como obsoleta: JUST-TTL, FULL-ERASE, IMMUNE, IMMUNE-TX e VACCINE. Estes métodos são progressivos conforme a complexidade.

JUST-TTL: Neste método usa-se o TTL *(Time-To-Live)* , tempo de vida do pacote, para o descarte, que é calculado para todas as cópias, com a probabilidade de $P(\tilde{R}_i, k)$, onde k é o tempo, para todos $1 \leq i \leq N$, e $k > 1$, quando a probabilidade for igual a zero, $P(\tilde{R}_i, k) = 0$ todos os pacotes são descartados.

FULL-ERASE: Este mecanismo usa uma estação base para armazenar as mensagens e reencaminhá-las ao nó destino, ao entrar em contato com a estação base a cópia é descartá-la completamente após a entrega. Neste esquema, uma baleia é capaz de descartar o pacote e, em seguida, receber esse pacote novamente de uma outra baleia.

IMMUNE: Este método descarta o pacote quando ele chega ao destino, mas mantém um identificador do pacote entregue, por isso não vai aceitar o pacote

novamente. Este identificador é referenciado como um *"antipacket"*, uma vez que impede re-infecção com o pacote. Para uma melhor compreensão a Figura 3.1 apresenta uma breve explicação do seu funcionamento.

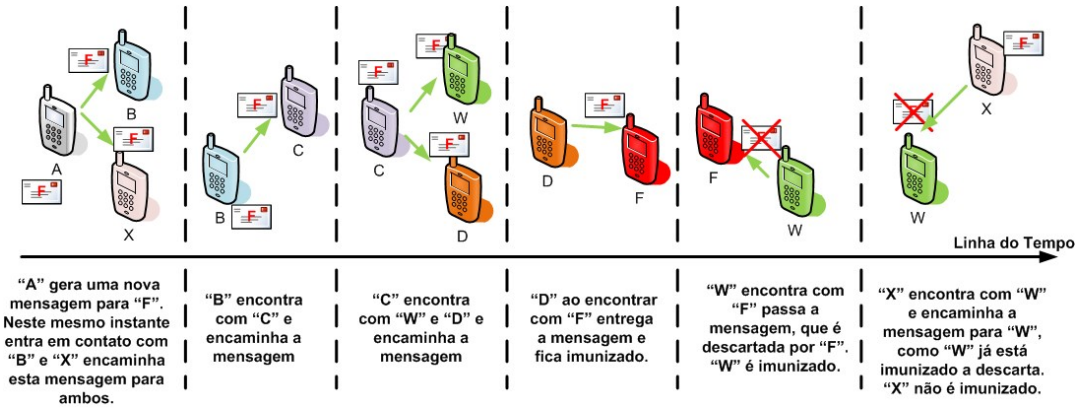


Figura 3.1: Descrição do Mecanismo Immune

IMMUNE-TX: funciona como IMMUNE, mas também compartilha este *"antipacket"* com outras baleias que carregam cópias desse pacote. Isto ocorre quando uma baleia está "infectada" com o pacote, pois, ele pode receber um identificador *"antipacket"* a partir de uma segunda baleia e avisar que uma cópia do pacote já foi entregue. O pacote então seria descartado e a segunda baleia também passaria a ter o *"antipacket"*. Para uma melhor compreensão, veja na Figura 3.2 uma breve explicação do seu funcionamento.

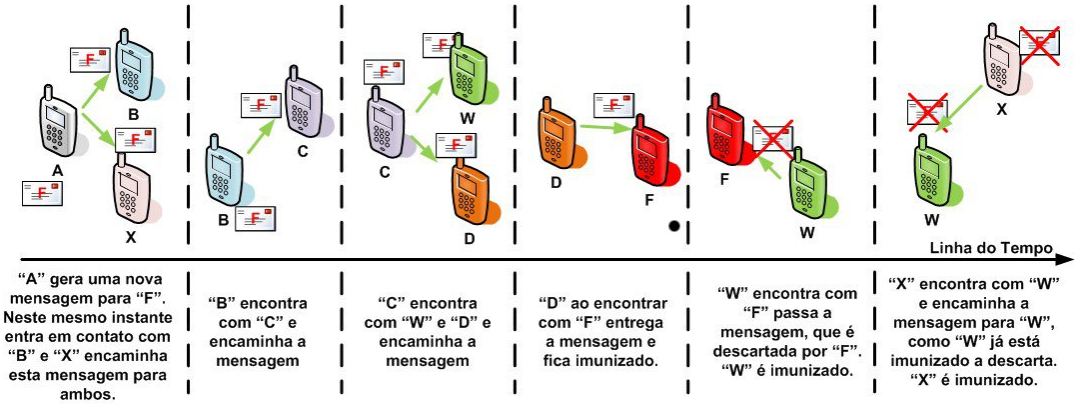


Figura 3.2: Descrição do Mecanismo Immune-TX

VACCINE: Se o pacote já foi recebido no receptor, este método envia a lista, (*antipacket*), quando dentro do alcance de transmissão. Para uma melhor compreensão veja na Figura 3.3 uma breve explicação do seu funcionamento.

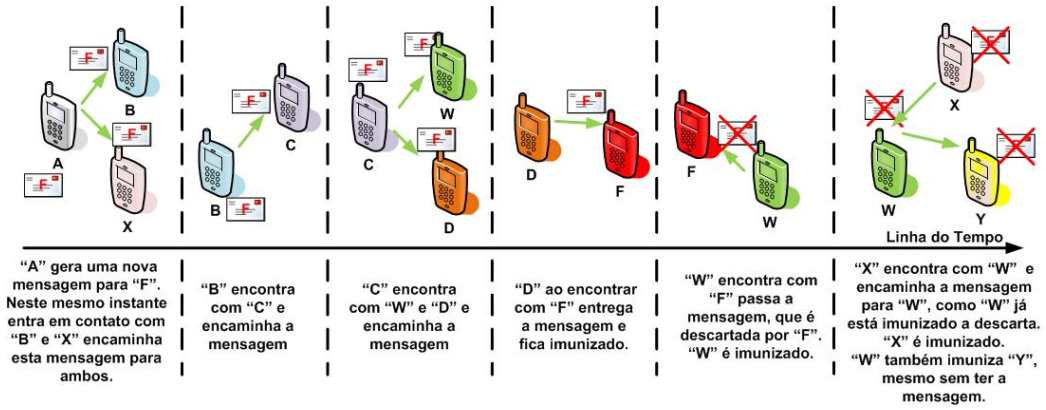


Figura 3.3: Funcionamento do mecanismo Vaccine

O mecanismo proposto na presente dissertação de mestrado utiliza o modelo matemático VACCINE, sendo que o mecanismo proposto pode ser utilizado independente do protocolo de roteamento.

Em [Mundur and Seligman (2008)], foi modificado o protocolo de roteamento Epidêmico [A. Vahdat (2000)]. Como este protocolo dissemina muitas mensagens pela rede, a proposta apresentada foi de, antes de trocarem as mensagens de dados, trocarem uma lista onde são armazenados os IDs de cada mensagem entregue no destino. Esta implementação visa impedir trocas futuras entre mensagens já entregues no destino. O funcionamento é parecido ao modelo apresentado em [Z. J. Haas (2006)], onde são apresentados o VACCINE, o IMMUNE e o IMMUNE-TX. Nesta implementação existe a troca de lista inicial conforme o VACCINE, mas não é acrescentado na lista de imunização do nó que entrega a mensagem no destino. Os resultados da simulação mostram melhoras significativas de desempenho, tanto na taxa de entrega como no atraso médio, quando

comparados o Epidêmico com Imunização e o Epidêmico sem uso do mecanismo. Neste trabalho pode-se observar que somente funciona para o protocolo Epidêmico, não é independente do protocolo de roteamento. Além disso, a lista de imunização no final não estará igual em ambos os nós.

Em [Yuen and Schulzrinne (2010)], é mostrado que em uma DTN, dependendo da mobilidade dos nós, o tempo de entrega pode variar de minutos a horas ou dias. Assim, para agilizar a entrega das mensagens, as mesmas são replicadas durante os encontros dos nós, mesmo quando o nó receptor não for o destino. Para limitar o número de réplicas de mensagens redundantes, as mensagens são eliminadas através do uso do TTL (*time-to-live*). Assim, foi definido um regime baseado no tempo e outro baseado no número de saltos para o TTL. O objetivo é analisar os dois regimes em termos de armazenamento, métricas de custo de energia e tempo de armazenamento, através de simulações. Para o TTL baseado no tempo, inicialmente, o nó de origem tem um valor TTL definido para $TTL_{(max)}$. Este valor é decrementado a cada segundo até chegar a zero, quando a mensagem é apagada. Em cada encontro as mensagens são replicadas com o mesmo valor de TTL. Por conseguinte, todas as réplicas em todos os nós partilham do mesmo TTL, quando o tempo expirar, todas as mensagens serão eliminadas simultaneamente. Neste trabalho, não foi considerado a mobilidade dos nós para cálculo das tuplas, podendo existir a possibilidade de descarte antes da mensagem chegar ao destino, ou até mesmo manter cópias na rede após a sua entrega no destino.

Um mecanismo de remoção de mensagem também é o foco dos trabalhos em [S. Kaveevivitchai (2010b)], [S. Kaveevivitchai (2010a)], onde é apresentada a eficiência no mecanismo para eliminação de mensagem que pode ser aplicado independente do algoritmo de roteamento. O projeto foi separado em duas partes:

(i) métodos para a distribuição de mensagens de confirmação (ACK) e (ii) utilização de nós auxiliares para retransmitir mensagens de ACK. Além disso, os ACKs foram divididos em duas partes, ACKs ativos e ACKs passivos. Os ACKs passivos têm o seguinte comportamento: o ACK de qualquer mensagem 'm' não vai expressar o seu conhecimento para qualquer nó encontrado, a menos que o nó encontrado tente enviar uma cópia da mensagem 'm' para ele. Neste caso, a mensagem de ACK será distribuída lentamente e as cópias da mensagem entregue serão eliminadas em conformidade. Nos ACKs ativos, um nó que tenha ACK de qualquer mensagem tenta expressar seu conhecimento para qualquer nó que faça contato. O caso extremo desta distribuição ativa é a de expressar seu conhecimento para todos os encontros (*broadcast-like*). Podemos observar que o conceito de ACK passivo é igual ao conceito do IMMUNE, e o VACCINE é equivalente para o uso de ACKs ativos descritos em [Z. J. Haas (2006)]. A implementação deste trabalho parece ser similar ao proposto, embora a implementação apresentado no ReMO tenha-se a preocupação com o tamanho da lista que é criada para controle das mensagens entregues no destino.

O controle de mensagens disseminadas na rede é o foco do trabalho de [Yu and Bian (2011)], neste artigo mostra-se a importância de controlar as replicações existentes pelos protocolos disseminadores, onde o uso do TTL e de *"anti-packet"*, não são as melhores soluções. No seu trabalho, é apresentado o método *"THRESHOLD"*, que utiliza um contador para as cópias das mensagens que são disseminadas. Esta variável "i" pode valer de -1,0,1,2,...,$V-1$ onde inicialmente este contador é inicializado com 0, pelo nó que cria a mensagem, e para cada nó que recebe uma cópia é inicializado com 1. Geralmente este valor de "i" pode variar entre $(1 \leq i \leq V-1)$. Quando o valor deste contador chegar a "V", que é

o valor calculado previamente por este método, representa que deve-se eliminar esta mensagem. O outro valor determinístico é quando chega a -1, que significa que a mensagem já foi recebida. Neste trabalho, o mecanismo só foi implementado para o protocolo Epidêmico, onde poderia ser usado para outros protocolos de roteamento, visto que, existem outros que também disseminam cópias na rede.

Em [Wang et al. (2009b)], é proposto um novo esquema de roteamento, chamado *Weighted Epidemic Routing (WER)*. O WER organiza a prioridade de mensagens etiquetando-as para encaminhar ou excluir através da atribuição de um peso para cada uma delas, quando o tamanho do *buffer* dos nós e a duração do contato são limitadas. O peso de uma mensagem é determinada por três fatores: a quantidade de encontros, as quantidades de cópias e o TTL. Enquanto isso, um mecanismo de remoção ACK, é empregado por WER para excluir as mensagens redundantes. Em simulação os resultados mostram que o protocolo WER supera o Epidêmico com *Drop-Tail* (algoritmo de gerenciamento utilizado pelos roteadores de Internet para decidir quando descartar pacotes) e política de descarte FIFO por um fator de cerca de 6,8% e 0,66%, respectivamente, quando o tamanho do *buffer* é pequeno.

Uma proposta de um novo protocolo de roteamento também é o foco do trabalho em [Li et al. (2009)]. Este trabalho propõe um novo protocolo para roteamento em DTNs chamado *Adaptive Priority Routing with Ack mechanism*(APRA). Em primeiro lugar, o APRA encaminha mensagens de acordo com um esquema de probabilidade adaptativo que está relacionado com a densidade de mensagem encontrada nos nós. Em segundo lugar, o APRA atribui uma prioridade à cada mensagem, em função do TTL, previsão de entrega (DP) e densidade de replicação (RD).

Em, [Wang et al. (2009a)], é proposto um novo algoritmo de encaminhamento, chamado *Directional Forward Epidemic Routing (DFER)*. O DFER, em primeiro lugar, armazena as informações da posição dos nós vizinhos, e seleciona os nós apropriados de acordo com a direção e os ângulos formados entre esses vizinhos e o destino. Em segundo lugar, o DFER atribui um peso, que é calculado através de uma função de densidade de replicação, TTL e a previsibilidade de entrega, para cada mensagem. O peso é usado em priorização das mensagens para a transmissão. Uma característica única do DFER é que utiliza um mecanismo para remover mensagens que já foram entregues no destino. Assim como, o utilizado pelo protocolo Epidêmico, que troca vetores, no DFER os vetores trocados são das mensagens que já foram entregues no destino e que ficam armazenadas em cada nó.

Um mecanismo de remoção que se ajusta ao modo de propagação ACK de acordo com taxa de perda é proposto em [An et al. (2012)] é o *Congestion Level based end-to-end ACKnowledgement* (CL-ACK). Neste mecanismo usam-se ACKs para confirmação de recebimento da mensagem no destino. O ACK pode ser definido de duas formas, assim como em [S. Kaveevivitchai (2010b)]: ACKs Passivos (PACK) e ACKs Ativos (AACK). Neste mecanismo pode-se mudar o modo de encaminhamento de ACKs de acordo com o nível de congestionamento. O CL (nível de Congestionamento) é medido pelas métricas “mensagens descartadas” e “quantidade de mensagens replicadas”. Toda vez que um nó entra em contato com o outro ele irá fazer este cálculo para decidir se deve ou não enviar ACKs. Dependendo do congestionamento ele irá transmitir ou não.

O mecanismo ReMO (Remoção de Mensagens Obsoletas) tem a vantagem de ser facilmente implementado, podendo ser usado independente do protocolo

de roteamento empregado. A lista, por armazenar somente o identificador de cada mensagem, não ocupa muito espaço podendo inclusive ser dimensionada. Mesmo protocolos que já façam exclusão de mensagens,podem utilizar o ReMO em conjunto, melhorando ainda mais a performance.

Capítulo 4

O Mecanismo de Remoção de Mensagens Obsoletas - ReMO

Este trabalho tem como finalidade apresentar um mecanismo de remoção de mensagens obsoletas independente do protocolo de roteamento utilizado. Aqui será apresentado o mecanismo ReMO , que foi desenvolvido conforme o modelo *VACCINE*, com uma pequena variação na sua implementação, assim podendo ser utilizado por qualquer protocolo de roteamento de DTNs. Além disso, o mecanismo foi elaborado para se poder dimensionar o tamanho da lista, que deve ser trocada entre os nós com a possibilidade de definir um tamanho fixo.

4.1 Visão Geral

A necessidade de se usar de mecanismos de remoção de cópias de mensagens que já foram entregues nas DTNs deve-se ao fato de que, normalmente, para garantir uma taxa alta de entrega e um atraso menor, os protocolos de roteamento

replicadores ou de inundação, que ao contrário dos encaminhadores que repassam a mensagem salto-a-salto, não gerando cópias de uma mensagem na rede. Os replicadores ou de inundação, normalmente, geram uma grande quantidade de cópias pela rede, como é o caso do protocolo Epidêmico [A. Vahdat (2000)]. Para estes protocolos surge a necessidade de mecanismos que reduzam o número de cópias de mensagens na rede, pois normalmente para estas redes os recursos são escassos, principalmente o tamanho dos *buffers*. Na Figura 4.1 tem-se o fluxograma modelando o instante de contato entre os nós n_{Tx} e n_{Rx}.

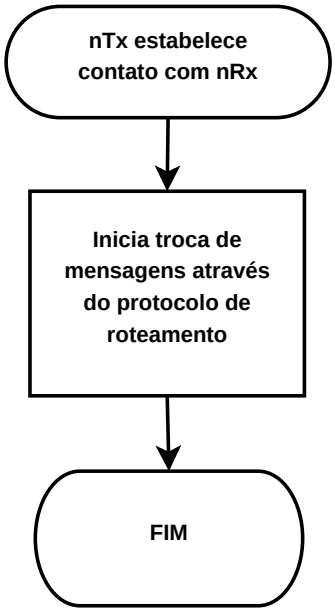


Figura 4.1: Contato entre dois nós sem uso do mecanismo

O mecanismo de Remoção de Mensagens Obsoletas (ReMO), assim como outros encontrados na literatura, utiliza do modelo apresentado em [Small and Haas (2003) e Z. J. Haas (2006)], o *VACCINE*, onde existe um controle da lista que é trocada entre os nós. Dependendo da mobilidade dos nós e da grande

quantidade de mensagens geradas, esta lista pode crescer infinitamente.

Em Gomes et al. (2012) foi apresentado uma versão preliminar deste mecanismo, onde foi realizada uma avaliação de desempenho para verificar sua influência no funcionamento de uma DTN, utilizando os protocolos Epidêmico, *PROPHET* e *Spray and Wait*. Os resultados apresentados neste artigo mostraram que para os protocolos: Epidêmico e *PROPHET*, que geram muitas cópias das mensagens durante sua execução, obtiveram resultados significativos, reduzindo a taxa de ocupação. Já para o protocolo *Spray and Wait*, por controlar mais as cópias que são disseminadas na rede, a sua contribuição não é tão significativa, somente nos casos onde o tamanho dos *buffers* são pequenos.

No mecanismo proposto, procura-se remover as mensagens obsoletas antes de receber novas mensagens, isto faz com que o *buffer* tenha, na sua maioria, cópias de mensagens que ainda não foram entregues no destino, permitindo um número maior de troca de novas mensagens. Além disso, como reduz o número de cópias na rede, melhora a performance. Outro fator importante neste mecanismo é, que ele independe do protocolo de roteamento usado.

A implementação do ReMO é dividida em duas partes, antes do envio das mensagens de dados e após o recebimento das mesmas. Na subseção 4.1.2 será abordado melhor esta implementação.

4.1.1 O algoritmo do mecanismo ReMO

As DTNs trocam mensagens quando ocorre um contato entre dois ou mais nós. Um contato pode ser definido conforme a Figura 4.2 abaixo, onde n_{Tx} e n_{Rx} são nós que estão numa mesma região, com suas respectivas posições no instante t.

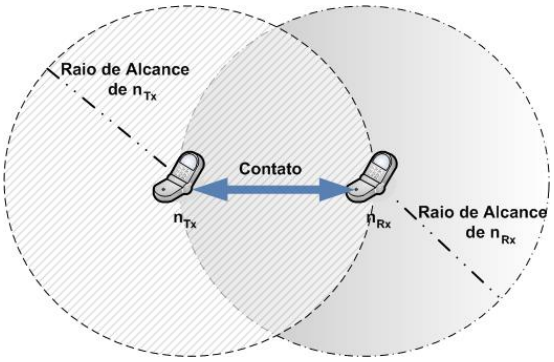


Figura 4.2: Contato entre dois nós

O ReMO utiliza uma lista de controle de mensagens entregues para cada nó, onde são armazenadas as informações das mensagens que já foram entregues no destino. Esta lista é trocada a cada contato e após a sua troca é efetuada a remoção de mensagens obsoletas do *buffer*. Após esta fase os nós transmitirão as mensagens que estejam nos seus respectivos *buffers*, seguindo o protocolo de roteamento que os mesmos estão executando.

A explicação do funcionamento, do mecanismo de remoção de mensagens obsoletas (ReMO), foi divido em duas partes para uma melhor compreensão. Na Figura 4.3 é apresentado o fluxograma do nó n_{Tx}, este nó representa, numa transmissão de dados, o nó transmissor da mensagem.

Descrição do fluxograma de n_{Tx}

- n_{Tx} **estabelece contato com** n_{Rx} - Neste passo os nós entram em contato, estabelecendo um conexão entre eles.

- **Existe registro na lista de** n_{Tx}**?** - Neste passo é verificado se existe algum registro na lista de controle de mensagens entregues no destino, caso não tenha elementos, a lista esteja vazia, não existe necessidade de passar para o

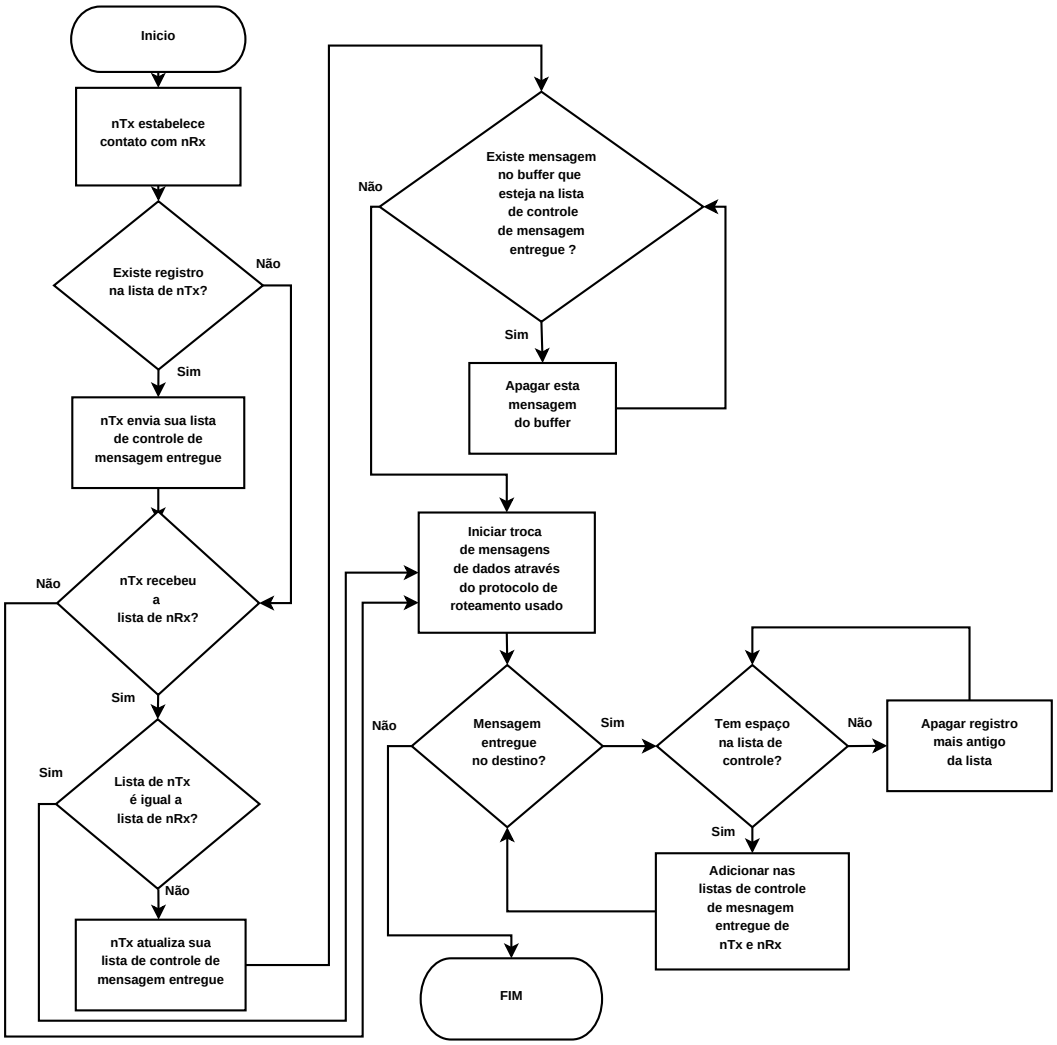


Figura 4.3: Contato de n_{Tx} com o uso do mecanismo ReMO

próximo passo, "n_{Tx} envia sua lista de controle de mensagem entregue", deve saltar para o passo, "n_{Tx} recebeu a lista de n_{Rx}?". Caso exista mensagens na lista de controle de mensagem entregue, deve continuar o algoritmo no próximo passo, "n_{Tx} envia sua lista de controle de mensagem entregue".

- n_{Tx} **envia sua lista de controle de mensagem entregue** - Neste passo o nó n_{Tx} envia sua lista de controle de mensagem entregue, pois já foram

entregues mensagens no destino e este nó já foi informado;

- n_{Tx} **recebeu a lista de** n_{Rx}**?** Aqui é verificado se a lista de n_{Rx} foi enviada para n_{Tx}, caso n_{Tx} a tenha recebido irá passar para o próximo passo que é verificar se a "lista de n_{Tx} é igual a lista de n_{Rx}?". Caso não tenha recebido, também não existirá necessidade de passar para o próximo passo, devendo pular para "iniciar troca de mensagens de dados através do protocolo de roteamento usado".

- **Lista de** n_{Tx} **é igual a lista de** n_{Rx}**?** - Aqui é verificado se existem elementos diferentes da lista de n_{Tx}. Caso não exista, deverá saltar para "iniciar troca de mensagens de dados através do protocolo de roteamento usado". Aqui podemos observar que, caso não tenham sido entregues mensagens no destino, o mecanismo não terá sido acionado, ficando igual a como se não existisse, pois ele terá passado através dos saltos, para "iniciando a troca de mensagens segundo o protocolo de roteamento". Caso exista elementos diferentes ele deverá passar para o passo seguinte "n_{Tx} atualiza sua lista de controle entregue".

- n_{Tx} **atualiza sua lista de controle de mensagem entregue** - Após o recebimento, verificação e atualização da lista de controle n_{Rx}, os dois nós estão idênticos, podendo passar para o passo seguinte.

- **Existe mensagem no** *buffer* **que esteja na lista de controle de mensagem entregue?** - Faz a verificação de mensagens obsoletas no *buffer* caso exista ele irá passar para o passo seguinte que é "apagar esta mensagem do *buffer*", onde deve apagar a mensagem selecionada. Caso não exista

mensagem no *buffer* que esteja na lista de controle entregue, deverá passar para o passo seguinte.

- **Iniciar troca de mensagens de dados através do protocolo de roteamento usado** - Neste passo os responsáveis pela troca das mensagens são os protocolos de roteamento, independente de usar ou não o mecanismo.

- **Mensagem entregue no destino?** - Se faz uma verificação para saber se a mensagem que está sendo entregue é uma mensagem para o destino final ou se será para encaminhamento. Caso seja o destino final ele irá passar para o próximo passo, onde será feita uma outra verificação, "tem espaço na lista de controle?". Caso não exista espaço ela irá passar para o passo seguinte, "apagar registro mais antigo da lista", onde deve apagar na lista de controle o registro mais antigo, abrindo espaço para incluir um novo. Após, ele retorna ao procedimento de verificação anterior para novamente testar. Caso exista espaço ele irá para o seguinte passo, "adicionar nas listas de controle de mensagem entregue de n_{Tx} e n_{Rx}", onde é armazenado os IDs das mensagens nas listas de controle de ambos os nós. Caso a mensagem entregue não seja o destino final, será armazenada no *buffer* para ser reencaminhada num próximo contato.

Na Figura 4.4 é apresentado o fluxograma do nó n_{Rx}, este nó representa o que recebe a mensagem.

Descrição do fluxograma de n_{Rx}

- n_{Rx} **estabelece contato com** n_{Tx} - Neste passo os nós entram em contato,

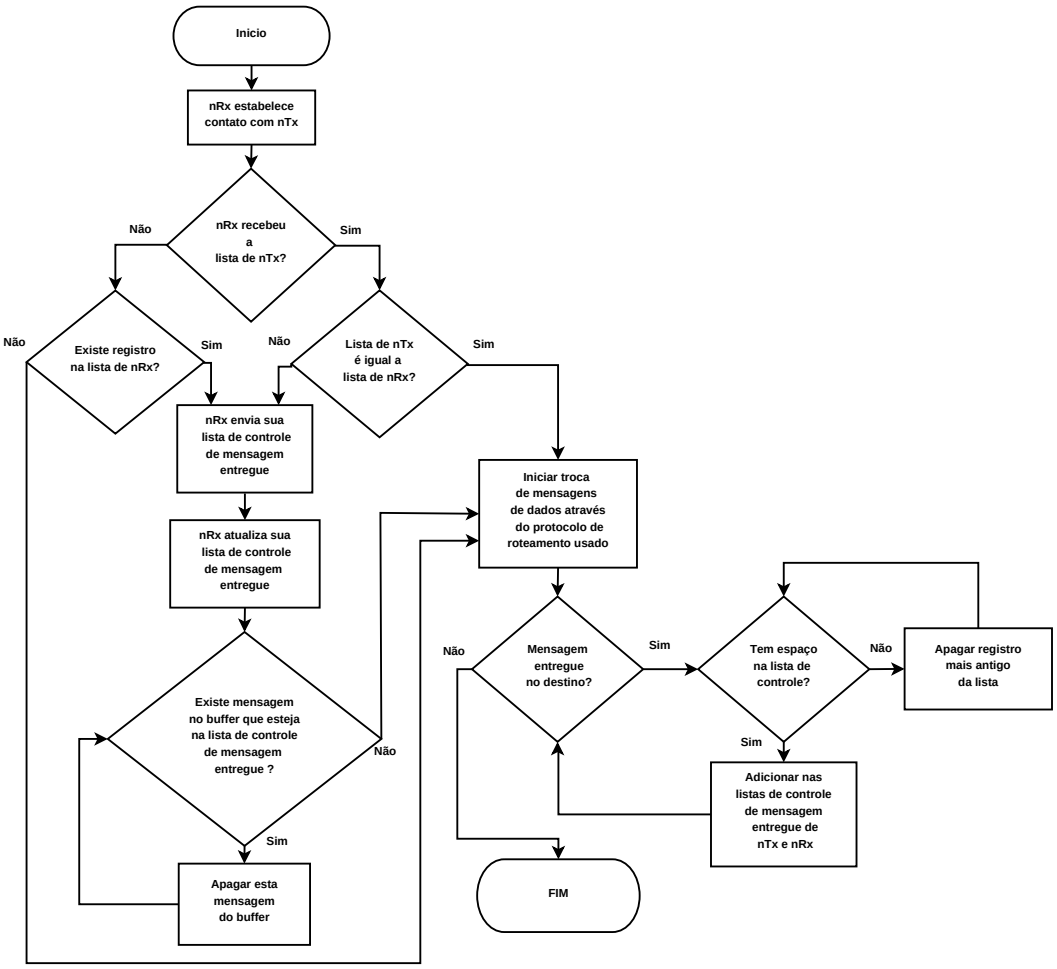


Figura 4.4: Contato de n_{Rx} com o uso do mecanismo ReMO

estabelecendo uma conexão entre eles.

- n_{Rx} **recebeu a lista de** n_{Tx}**?** Aqui é verificado se a lista de n_{Tx} foi enviada para n_{Rx}, caso n_{Rx} a tenha recebido, irá passar para o passo que é, verificar se a "lista de n_{Rx} é igual a lista de n_{Tx}?". Caso não tenha recebido, deverá passar para o passo, "Existe registro na lista de n_{Rx}?".

- **Lista de** n_{Rx} **é igual a lista de** n_{Tx}**?** - Faz a verificação da existência de elementos diferentes da lista de n_{Rx}. Caso não exista, deverá saltar para

"iniciar troca de mensagens de dados através do protocolo de roteamento usado". Pode-se observar que, caso não tenham sido entregues mensagens no destino, o mecanismo não terá sido acionado, ficando igual a como se não existisse. Caso exista elementos diferentes, deverá passar para o passo seguinte "n_{Rx} envia sua lista de controle de mensagem entregue".

- **Existe registro na lista de n_{Rx}?** - Neste passo é verificado se existe algum registro na lista de controle de mensagens entregues no destino, caso não tenha elementos, a lista esteja vazia, não existe necessidade de passar para o próximo passo, "n_{Rx} envia sua lista de controle entregue", deve saltar para o passo, "iniciar troca de mensagens de dados através do protocolo de roteamento usado". Caso exista mensagens na lista de controle de mensagem entregue, deve continuar o algoritmo no próximo passo, "n_{Rx} envia sua lista de controle de mensagem entregue".

- n_{Rx} **envia sua lista de controle entregue** - Neste passo o nó n_{Rx} envia sua lista de controle de mensagem entregue, pois já foram entregues mensagens no destino e este nó já foi informado;

- n_{Rx} **atualiza sua lista de controle entregue** - Após o recebimento, verificação e atualização da lista de controle de n_{Tx}, os dois nós estão idênticos, podendo passar para o passo seguinte.

- **Existe mensagem no *buffer* que esteja na lista de controle de mensagem entregue?** - Se faz uma verificação de mensagens obsoletas no *buffer* caso exista ele irá passar para o passo seguinte que é "apagar esta mensagem do *buffer*", onde deve apagar a mensagem selecionada. Caso não

exista mensagem no *buffer* que esteja na lista de controle de mensagem entregue, deverá passar para o passo seguinte.

- **Iniciar troca de mensagens de dados através do protocolo de roteamento usado** - Neste passo os responsáveis pela troca das mensagens são os protocolos de roteamento, independente de usar ou não o mecanismo.

- **Mensagem entregue no destino?** - Se faz uma verificação para saber se a mensagem que está sendo entregue é uma mensagem para o destino final ou se será para encaminhamento. Caso seja o destino final ele irá passar para o próximo passo, onde será feita uma outra verificação, "tem espaço na lista de controle?". Caso não exista espaço ela irá passar para o passo seguinte, "apagar registro mais antigo da lista", onde deve apagar na lista de controle o registro mais antigo, abrindo espaço para incluir um novo. Após, ele retorna ao procedimento de verificação anterior para novamente testar. Caso exista espaço ele irá para o seguinte passo, "adicionar nas listas de controle de mensagem entregue de n_{Tx} e n_{Rx}", onde é armazenado os IDs das mensagens nas listas de controle de ambos os nós. Caso a mensagem entregue não seja o destino final, será armazenada no *buffer* para ser reencaminhada num próximo contato.

4.1.2 Detalhes da implementação do mecanismo ReMO no simulador ONE

O ReMO foi implementado no simulador *The ONE* [Keränen et al. (2009)], que será descrito melhor no Apêndice A. É um dos simuladores mais utilizados em

DTN. O *The ONE* é um simulador de eventos discretos desenvolvido na linguagem de programação JAVA e de código aberto.

Primeiramente foi incorporado o código do simulador ao Eclipse, que é um IDE desenvolvido em Java, seguindo o modelo *open source* de desenvolvimento de software, as informações de compilação do código no Eclipse pode ser obtido no arquivo "README.txt", que vem junto com o código do simulador.

Para implementação dos mecanismos de remoção foi elaborado um cenário de teste que pudesse verificar as operações realizadas no simulador, este cenário será chamado de "cenário teste", onde foi configurado da seguinte forma: tempo de simulação de 1500s; raio de transmissão de 10m; um grupo de pedestres; com três nós, um *buffer* de 100MB; TTL com o tempo total da simulação, as mensagens foram geradas entre 10 e 20s; com um tamanho entre 500K e 1M; movimento dos nós RandomWaypoint; uma área de 25 x 25 metros.

Para a troca de informações entre os nós no mecanismo ReMO, referentes as mensagens que já chegaram nos seus respectivos destinos, foi adicionada uma lista de mensagens entregues em cada nó, esta lista foi configurada para armazenar (N) registros e ocupa (N) bytes do *buffer*, cada registro contém um identificador único da mensagem, gerado por cada nó. Além disso, no arquivo de configuração inicial do simulador *The One* o *default-settings* é possível configurar o tamanho da lista, caso não seja configurado ela irá assumir o tamanho máximo, que será o tamanho total do *buffer* menos tamanho das mensagens que existam. Outra alteração complementar efetuado no código do simulador para uma melhor verificação do funcionamento dos mecanismos, foi a inclusão de dois novo resultado no relatório *MessageStatsReport* a quantidade de mensagens criadas para envio da lista e o número de mensagens removidas pelo mecanismo. Inclusão de um relatório de

taxa de ocupação do *buffer*, que pode ser configurado para armazenar em tempos em tempos a informação da taxa de ocupação.

O simulador *The One* apresenta em sua composição uma estrutura de módulos, na qual cada módulo apresenta uma funcionalidade dentro do simulador, para a implementação do ReMO foram utilizadas os seguintes módulos: *Core, Input e Routing.*

No módulo *Core* (Núcleo) foram alterados as seguintes "Classes":

- *CBRConnection:* Esta classe é responsável por gerenciar o contato entre dois nós. Aqui foi implementado a preparação das mensagens que deverão ser trocadas entre os nós, para remoção de mensagens obsoletas; além da transmissão das mensagens com a lista de remoção entre os nós;

- *DTNHost:* Classe responsável pela criação dos nós. Foi criada a lista que será usada por todos os nós; também é implementado a inserção na lista do nó que está entregando uma mensagem no destino e no que está recebendo a mensagem no destino; verifica se as mensagem que estão no seu *buffer* coincide com alguma mensagem que recebeu da lista nova, caso contenha deverá ser removida;

- *Message:* Onde são criadas as mensagens que deverão ser trocadas entre os nós; Aqui foi criado um controle para identificar que tipo de mensagem está sendo passada, se uma mensagem de dados ou um mensagem de controle; também foi implementado um contador separado para investigação da quantidade de mensagens excluídas pelo mecanismo ReMO;

No módulo *Input* (Entrada) foram alterados as seguintes "Classes":

- MessageEventGenerator: Gera um evento para criação de mensagens externas; aqui foi implementado a criação das mensagens que deverão ser trocadas entre os nós (mensagem de controle), esta mensagem na sua constituição é idêntica a mensagem de dados, somente foi acrescentado em ambas um identificador booleano, quando a mensagem é de controle ela recebe "verdadeiro";

No módulo *Routing* (Roteamento) foram alterados as seguintes "Classes":

- *MessageRouter:* É uma superclasse para os roteamentos de mensagens; foi criado a variável "listaRemoSize" que serve para definir o tamanho da lista de mensagem removível. Além disso, ao receber uma mensagem onde o nó é o destino final, deverá chamar o procedimento de inclusão de mensagens na lista de mensagens removíveis de ambos os nós.

No capítulo seguinte será apresentada uma avaliação do mecanismo ReMO contendo os resultados obtidos por simulação.

Capítulo 5

Avaliação de Desempenho

Nesse capítulo é apresentada a avaliação de desempenho de simulações com e sem o uso dos mecanismos de remoção de mensagens obsoletas, para efeitos de comparação, onde são usados os três mecanismo, Immune, Immune-TX e ReMO, mais o uso do TTL de 50%, todos comparados ao original com o TTL igual ao tempo total de simulação, igual a infinito. Sendo assim, na Seção 5.1 são apresentados os cenários de simulação com uso de traços de mobilidade real e suas descrições. Na Seção 5.2, é apresentado os parâmetros de simulação utilizados na obtenção dos resultados, tais como, tamanho do *buffer*, tempo de simulação, entre outros. Na Seção 5.3 é descrito o funcionamento dos mecanismos que são comparados ao mecanismo ReMO. Logo depois, na Seção 5.5 são apresentados os resultados das simulações por cenários e protocolos de roteamento.

5.1 Cenário de Simulação

A escolha de cenários que consigam descrever situações que se aproximem da realidade é o fator primordial para a análise de resultados em DTNs. Estas redes normalmente são influenciadas diretamente pela mobilidade, quantidade dos nós e tempo de simulação. Desta forma procurou-se usar traços de mobilidade reais extraídos de experimentos disponibilizados no Crawdad [1], que pudessem contribuir diretamente na obtenção dos resultados. Para isso foram escolhidos dois traços de mobilidade real que tivessem diferenças significativas.

- UCL1 [Abdesslem et al. (2011)] - Na University College London foram recrutados 20 alunos para usarem iMotes e smartphones num experimento para coleta de dados, na qual os participantes podiam responder a questões enviadas. O smartphone escolhido foi o Nokia N95 8GB, com GPS, Wi-Fi, rede de celular 3G, uma câmera e um acelerômetro. Neste celular funciona o Symbian, sistema operacional, para o qual foi desenvolvido um aplicativo de compartilhamento de localização em Python, o LocShare. Este foi instalado nos celulares antes da distribuição para os participantes, projetado para ser executado automaticamente na inicialização e, em seguida, permanecer rodando em segundo plano.

- Rollernet [Leguay and Benbadis (2009)]- Passeio realizado por um grupo de patinadores em Paris, no qual foram coletados dados de mobilidade com iMotes por 62 participantes. Toda sexta-feira à noite e todas as tardes de domingo, em Paris, grupos de entre 5.000 e 15.000 pessoas vão andar de patins. Durante o percurso de três horas, os patinadores geralmente

[1] Recurso comunitário usado para armazenamento de dados de redes sem fio

percorrem 30 quilômetros, atravessando uma grande parte da cidade. Eles são guiados por membros da equipe e ajudados pelas forças de segurança pública. A fim de analisar a mobilidade dos participantes, foram realizados experimentos com sensores (iMotes), com cerca de uma centena de voluntários, que poderiam ser amigos dos organizadores, membros de associações de patins ou conhecidos. O conjunto de dados foi coletado em 20 de agosto de 2006. Segundo os organizadores e as informações da polícia, cerca de 2.500 pessoas participaram da patinação turística (algumas pancadas de chuva, pouco antes da turnê resultou em um número de participantes abaixo da média). A duração total da viagem foi de cerca de três horas, composto por duas sessões de 80 minutos, intercalados com uma pausa de 20 minutos. Os dados coletados permitem medir e caracterizar as interações entre as pessoas ao longo da duração dos passeios. Nesses traços existe uma peculiaridade, que é o efeito sanfona no deslocamento dos usuários ao longo do tempo, devido a dois padrões de mobilidade existentes: um quando os patinadores se aglomeram, aguardando a liberação de algum cruzamento, e o outro quando estão patinando normalmente ao longo de uma via. Essa peculiaridade, gera uma alta conectividade entre os usuários periodicamente que pode influenciar o funcionamento da DTN diferentemente de outros cenários.

5.2 Descrição dos parâmetros da simulação

A escolha dos valores dos parâmetros dos componentes da simulação é outro fator primordial para obtenção e análise dos resultados. Para este trabalho o foco da

investigação é observar o comportamento do *buffer* ao longo do tempo de simulação com relação aos mecanismos de remoções e às políticas de gerenciamento de *buffer*, aumentando a taxa de entrega, reduzindo o atraso médio e a sobrecarga.

Tamanho do *buffer:* Para a escolha deste parâmetro considerou-se principalmente o cenário usado, onde o tempo de simulação, quantidade e mobilidade dos nós, influenciam diretamente no dimensionamento. Para a análise dos resultados precisamos observar o comportamento quando ocorre o transbordo, pois os mecanismos deverão atuar de forma a diminuir este fato. Sendo assim, foram escolhidos *buffers*, através do padrão de tráfego, onde se considerou tamanho e frequência de envio das mensagens, que pudessem atender as avaliações dos seguintes casos: *buffer* pequeno, onde em pouco tempo de simulação ocorresse o transbordo; *buffer* intermediário e *buffer* que durante o tempo total de simulação não ocorresse o transbordo por falta de espaço.

Os protocolos utilizados na simulação foram escolhidos através de critérios simples, mas com grande influência na literatura:

1. Protocolo Epidêmico: por muito tempo considerado mais rápido na entrega de mensagens ao destino, porém vilão em ocupação de *buffer*;

2. Protocolo *Bubble Rap*: este protocolo usa o contexto social para encaminhamento das mensagens reduzindo o número de cópias na rede. A intensão é verificar se mesmo um protocolo recente, baseado em outras métricas, tem sua performance melhorada;

3. Protocolo *Spray and Wait*: este protocolo tem a disseminação das cópias das mensagens controladas, melhorando a utilização do espaço em *buffer*.

Para cada cenário de simulação foram realizados 10 rodadas distintas, para garantir a aleatoriedade estas rodadas foram obtidas através de um *script* chamando "createCreates.pl", que vem nas ferramentas do próprio simulador. Ao ser executado, ele gera um arquivo que é padrão de criação de mensagem para a simulação, os valores extraídos recebem uma aleatoriedade de uma distribuição uniforme com um mínimo incluído e valor máximo exclusivo. O arquivo gerado apresenta 6 colunas, conforme a Figura 5.1: a primeira coluna representa o instante em que os nós estarão em contato; na segunda representa o contato "C"; na terceira é descrito o número da mensagem que está sendo criada; a quarta coluna representa o nó que está criando a mensagem; na quinta coluna para qual nó esta mensagem deve ser entregue; e finalmente a sexta coluna, onde é descrito o tamanho em bytes da mensagem que foi criada. Após a geração destes 10 arquivos, eles são incorporados ao simulador para execução e posterior resultados, após os resultados é feita uma média para apresentação em gráficos dos resultados obtidos.

```
5.2   C     M1    57    16    10000
12.1  C     M2    24    48    10000
28.3  C     M3    46    12    10000
```

Figura 5.1: Exemplo do arquivo gerado pelo script createCreate.pl

Para o protocolo *Bubble Rap* todos os cenários tiveram as configurações descritas na Tabela 5.1. Valores usados conforme descritos em [Moreira et al. (2012b)], e sugerido por "PJ Dillon, *University of Pittsburgh*", autor da implementação do protocolo *Buuble Rap* no simulador *The One*.

Os cenário de simulação foram divididos conforme os traços de mobilidade real

Group.router = DecisionEngineRouter DecisionEngineRouter.decisionEngine = community.DistributedBubbleRap DecisionEngineRouter.communityDetectAlg = KCliqueCommunityDetection DecisionEngineRouter.K = 5 DecisionEngineRouter.familiarThreshold = 700 DecisionEngineRouter.centralityAlg = routing.community.CWindowCentrality

Tabela 5.1: Valores dos parâmetros usados para o protocolo Bubble Rap

escolhidos. Seus parâmetros foram configurados conforme a Tabela 5.2 abaixo.

Parâmetros	UCL1	Rollernet
Tempo de simulação	±6 dias (549019s)	±3 horas (10800s)
Grupo	Pedestres	
Número de Nós	20 nós	62 nós
Tamanho de *buffer*	1M; 2M; 5M; 10M; 15M; 20M; 25M	1M; 2M; 5M; 8M; 10M
Tempo de vida da mensagem (TTL)	6 dias	3 horas
Algoritmos de Roteamento	Epidêmico, BubbleRap e SprayAndWait	
Geração das Mensagens	8640 mensagens	2160 mensagens
Tamanho das mensagens	10KB	
Tempo para encaminhar cópia da mensagem	60s	
Número Máx. de cópias do SprayandWait	6 cópias	

Tabela 5.2: Valores dos parâmetros componentes/protocolos da simulação

5.3 Mecanismos a serem comparados

Os mecanismos de remoção visam informar aos nós intermediários que uma mensagem foi entregue ao nó de destino, possibilitando a remoção de suas réplicas da rede. São também conhecidos como *Antipacket*, pois impedem que o nó receba novamente a mensagem já descartada, tornando-o "imune" e, em alguns casos, podem inclusive informar aos nós intermediários possibilitando a remoção, 'vacinando-os'. Outro mecanismo avaliado foi o TTL (Time To Live), que neste trabalho utilizou a metade do tempo total de simulação.

Conforme o funcionamento de alguns protocolos de roteamento, temos como premissa que quando um nó entra em contato com outro, é feita uma breve verificação das mensagens em sua fila. Se possuírem mensagens diferentes então irão trocá-las.

Os mecanismos foram implementados conforme os modelos matemáticos citados em [Z. J. Haas (2006)], podendo ser empregados em qualquer protocolo de roteamento.

Para uma melhor explicação dos mecanismos é desconsiderado o descarte de mensagens, por tempo de vida(TTL), tempo de contato, ou *buffer* cheio. Também utilizaremos a linha tempo (Δt) para representar o deslocamento dos nós e o encaminhamento ou entrega das mensagens no destino.

5.3.1 O mecanismo Immune

O mecanismo Immune tem a função de tornar imune o nó de uma mensagem obsoleta, isto significa que ao receber uma mensagem que já tenha sido entregue no destino este nó não aceite mais uma cópia desta mensagem, evitando inclusive

as retransmissões desnecessárias.

Implementação do Immune

A implementação deste mecanismo é similar a segunda parte da implementação do ReMO, pois como ele só armazena as mensagens que foram entregues no destino para verificação posterior, caso receba uma nova mensagem deve verificá-la nesta lista.

No módulo *Core* (Núcleo) foram alterados as seguintes "Classes":

- *DTNHost:* Classe responsável pela criação dos nós. Foi criada a lista que será usada por todos os nós; também é implementado a inserção na lista do nó que está entregando uma mensagem no destino e no que está recebendo a mensagem no destino; verifica se as mensagem que estão no seu *buffer* coincide com alguma mensagem que recebeu, caso contenha deverá ser removida;

No módulo *Routing* (Roteamento) foram alterados as seguintes "Classes":

- *MessageRouter:* É uma superclasse para os roteamentos de mensagens; foi criado a variável "listaImmuneSize" que serve para definir o tamanho da lista de mensagem removível. Além disso, ao receber um mensagem onde o nó é o destino final, deverá chamar o procedimento de inclusão de mensagens na lista de mensagens removíveis de ambos os nós;

5.3.2 O mecanismo Immune-TX

O mecanismo Immune-TX também tem a função de tornar imune o nó de uma mensagem obsoleta, mas ele vai além imunizado o nó que tentou passar a men-

sagem obsoleta. Na Figura 3.2 uma breve explicação do seu funcionamento.

Implementação do Immune-TX

O mecanismo Immune-TX foi implementado com partes do mecanismo Immune e também do ReMO, o Immune-TX ao contrário do Immune envia uma mensagem com o identificador de cada mensagem que venha a receber e que já foram entregues no destino. Ele armazena em uma outra lista estes identificadores para no final transmití-las ao nó de contato. Este nó receberá esta lista e irá remover as mensagens do seu *buffer*, além de completar a sua lista com esta nova informação.

No módulo *Core* (Núcleo) foram alterados as seguintes "Classes":

- *CBRConnection:* Esta classe é responsável por gerenciar o contato entre dois nós. Aqui foi implementado a preparação das mensagens que deverão ser trocadas entre os nós, para remoção de mensagens obsoletas; além da transmissão da mensagens com a lista de remoção entre os nós;

- *DTNHost:* Classe responsável pela criação dos nós. Foi criada as duas listas que serão usadas por todos os nós; também é implementado a inserção na lista do nó que está entregando uma mensagem no destino e no que está recebendo a mensagem no destino; verifica se as mensagem que estão no seu *buffer* coincide com alguma mensagem que recebeu da lista nova, caso contenha deverá ser removida;

- *Message:* Onde são criadas as mensagens que deverão ser trocadas entre os nós; Aqui foi criado um controle para identificar que tipo de mensagem

está sendo passada, se uma mensagem de dados ou um mensagem de controle; também foi implementado um contador separado para investigação da quantidade de mensagens excluídas pelo mecanismo Immune-TX;

No módulo *Input* (Entrada) foram alterados as seguintes "Classes":

- MessageEventGenerator: Gera um evento para criação de mensagens externas; aqui foi implementado a criação das mensagens que deverão ser trocadas entre os nós (mensagem de controle), esta mensagem na sua constituição é idêntica a mensagem de dados, somente foi acrescentado em ambas um identificador booleano, quando a mensagem é de controle ela recebe "verdadeiro";

No módulo *Routing* (Roteamento) foram alterados as seguintes "Classes":

- *MessageRouter:* É uma superclasse para os roteamentos de mensagens; foi criado a variável "listaImmuneSize" que serve para definir o tamanho da lista de mensagem removível. Além disso, ao receber um mensagem onde o nó é o destino final, deverá chamar o procedimento de inclusão de mensagens na lista de mensagens removíveis de ambos os nós;

A implementação do Immune-TX é bem parecida com a implementação do ReMO, o que os torna diferentes é que o ReMO troca a lista inteira no início da operação, enquanto que o Immune-TX troca no final e somente as mensagens que recebeu e que estão na sua lista.

5.3.3 TTL - Time To Live

O TTL pode ser considerado um mecanismo de descarte de mensagens, é através dele que podemos dizer o tempo de vida de uma mensagem na rede, isto é um fator importante para as DTNs, pois evita que as mensagens ocupem espaço em *buffer* dos nós por mais tempo do que o necessário. Infelizmente calcular o tempo ideal para remover uma mensagem em DTN não é muito fácil. Em [Small and Haas (2003)], [Z. J. Haas (2006)], é apresentado um estudo sobre o "JUST-TTL", que mostra que mesmo estabelecendo cálculos probabilísticos, ainda assim, os resultados não são satisfatórios.

As mensagens em DTN, assim como as mensagens TCP/IP, também apresentam um campo que pode ser configurado para determinar a quantidade de tempo que uma mensagem pode permanecer na rede.

Para este trabalho são feitas duas variações no TTL. Uma que utiliza o tempo total de simulação, o que significa dizer que o TTL é infinito, que chamamos "sem TTL", pois utiliza o tempo total de simulação, o que faz com que as mensagens não sejam descartadas durante o tempo total. E outro, que utiliza a metade do tempo de simulação, o que significa dizer que se uma simulação leva três horas, o TTL será de uma hora e meia, todas as mensagens criadas terão o seu TTL em uma hora e meia. Em [Yuen and Schulzrinne (2010)], mostra a dificuldade em calcular um TTL que atenda as necessidades de uma DTN, os fatores, movimento dos nós, espaço em *buffer*, tempo que um nó permanece em contato com outro, influenciam diretamente neste calculo. A escolha de um TTL que utiliza metade do tempo de simulação dar-se ao fato desta dificuldade em calculá-lo. A metade do tempo total de simulação, serviria como uma referência, para analisarmos o

comportamento sem a necessidade de buscarmos fracioná-lo em tempos distintos. Com isso, espera-se resultados que sejam satisfatórios ou não.

5.4 Métricas de Desempenho Utilizadas

Neste trabalho serão utilizadas três métricas de desempenho (fração de mensagens entregues, atraso médio e sobrecarga de mensagens) para avaliar a influência do mecanismo proposto no funcionamento da rede sob diferentes protocolos de roteamento e a métrica ocupação do *buffer* para investigar a eficiência do mecanismo na remoção de mensagens obsoletas.

Para as DTNs a fração de mensagens entregues torna-se uma das métricas mais importantes para avaliação. Como neste trabalho está sendo usado protocolos de roteamento disseminadores, a tendência é que os *buffers* sejam preenchidos mais rapidamente diminuindo a taxa de entrega, com o uso do mecanismo espera-se um aumento na taxa de entrega. Pois, como serão eliminadas mensagens obsoletas, o *buffer* deverá ter mais espaço para armazenar novas mensagens, inclusive de mensagens que antes não chegavam ao destino, pois eram descartadas através da política de gerenciamento de *buffer*.

fração de mensagens entregues: número de mensagens entregues dividido pelo número de mensagens criadas;

O atraso médio também foi outra métrica utilizada na investigação deste trabalho, visto que, normalmente uma mensagem é excluída do *buffer* se levar muito tempo para chegar ao destino, pois o simulador *The One* utiliza a política de gerenciamento de *buffer* FIFO, onde as mensagens mais antigas são descartadas. A definição para esta métrica é:

Atraso médio: tempo médio que uma mensagem leva para ser entregue, desde quando é gerada até o seu destino;

A sobrecarga de mensagens ou *overhead* é outra métrica que foi utilizada para investigação dos resultados obtidos, pois dependendo do número de saltos e de seus repasses até o destino existirá um consumo tanto de energia quanto de utilização de *buffer*. Se as mensagens que foram entregues no destino forem eliminadas, consequentemente deixaram de ser reencaminhadas, diminuindo o consumo de energia, como de utilização de *buffer*, em contra partida aumentaremos a chance de mensagens mais antigas chegarem no destino. A sua definição é:

Sobrecarga de mensagens: quantidade de mensagens repassadas dividido pela quantidade de mensagens entregues;

Neste trabalho estamos preocupados em eliminar as mensagens obsoletas dos *buffers*. Se conseguirmos reduzir o número de mensagens que já chegaram no destino e que ainda continuam ocupando espaço nos *buffers* dos nós intermediários, então poderemos usar os *buffers* para armazenarem mensagens que ainda não chegaram e precisam chegar no destino. Por este motivo a métrica Ocupação de *buffer* foi outra escolhida neste trabalho, onde é possível ver em alguns casos uma melhora generalizada de toda a rede. A ocupação de *buffer* tem a sua definição como sendo:

Ocupação do *buffer*: quantidade de dados armazenados dividido pela capacidade total do *buffer*, em um determinado instante de tempo.

Na seção seguinte iremos apresentar e analisar os resultados obtidos com as métricas apresentadas.

5.5 Resultados obtidos da comparação com outros mecanismos de remoção

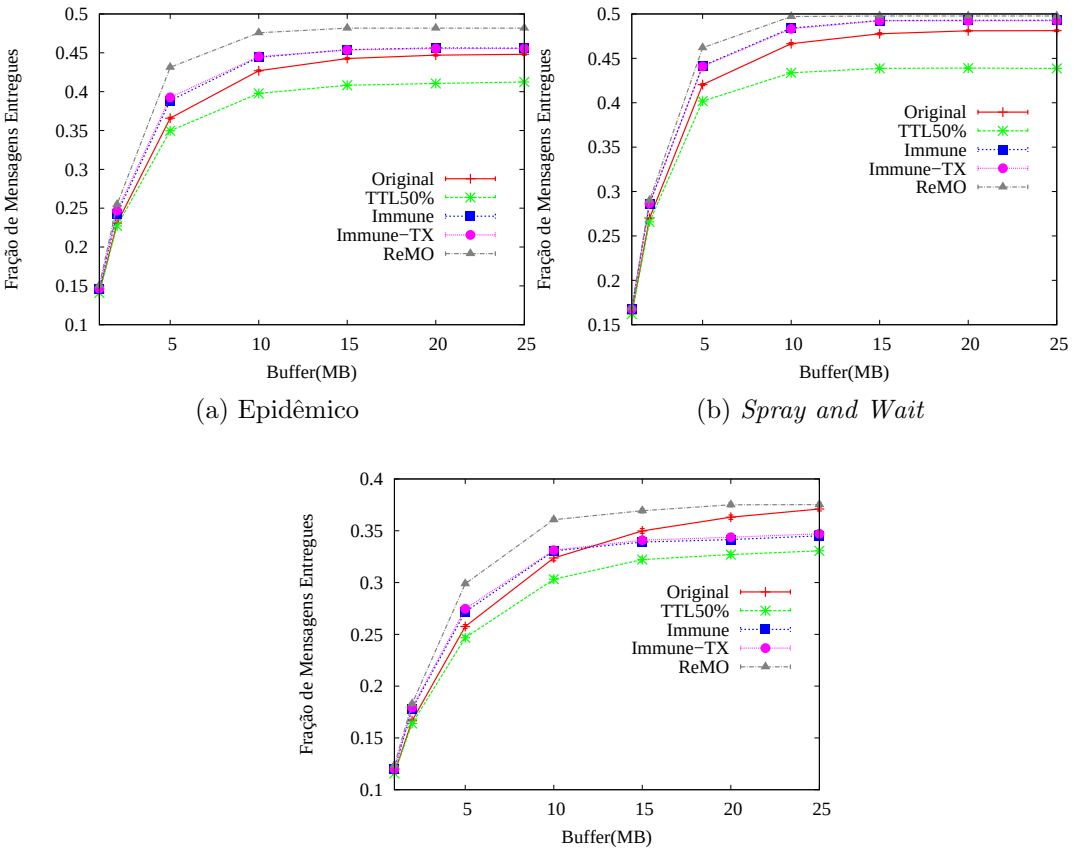
Nesta seção são apresentados os resultados primeiramente obtidos na investigação do uso de mecanismo de Remoção de Mensagens Obsoletas (ReMO) com outros mecanismo de remoção, através das métricas e dos parâmetros configurados no simulador *The One*.

5.5.1 Fração de mensagens entregues para os mecanismos de remoção

Como definido na Seção 5.4, espera-se que com o uso dos mecanismo de remoção, em especial o mecanismo de remoção de mensagens obsoletas (ReMO), aumente a fração de mensagens entregues com o uso dos protocolos analisados, tanto no cenário UCL1 como no cenário Rollernet.

Na Figura 5.2a é apresentado o gráfico fração de mensagens entregues com o uso do protocolo de roteamento Epidêmico, onde o *buffer* é variado, com o tamanho das mensagens definido em 10KB. O protocolo Epidêmico tem como premissa disseminar um maior número de mensagens na rede para aumentar a chance de entrega num menor tempo. Para este traço de mobilidade real os mecanismos apresentaram uma melhora em relação ao sem o uso do mecanismo de remoção, com a remoção de mensagens obsoletas o descarte foi menor, aumentando a chance da entrega no destino. O mecanismo ReMO obteve um melhor desempenho também para este protocolo, isto se deve à remoção de mais mensagens obsoletas. Com isso, o *buffer* teve mais espaço para armazenar outras mensagens

aumentado esta probabilidade. Como os outros mecanismo não removem tantas mensagens como o ReMO, o desempenho ficou abaixo, mas ficou um pouco melhor que o sem mecanismo. Já o TTL de 50% entre os mecanismos analisados foi o que obteve o pior resultado, isto deve-se ao fato, de que muitas mensagens foram criadas no intervalo de tempo inicial da simulação e não tiveram o tempo suficiente para chegar ao destino, sendo descartadas.



(a) Epidêmico

(b) *Spray and Wait*

(c) *Bubble Rap*

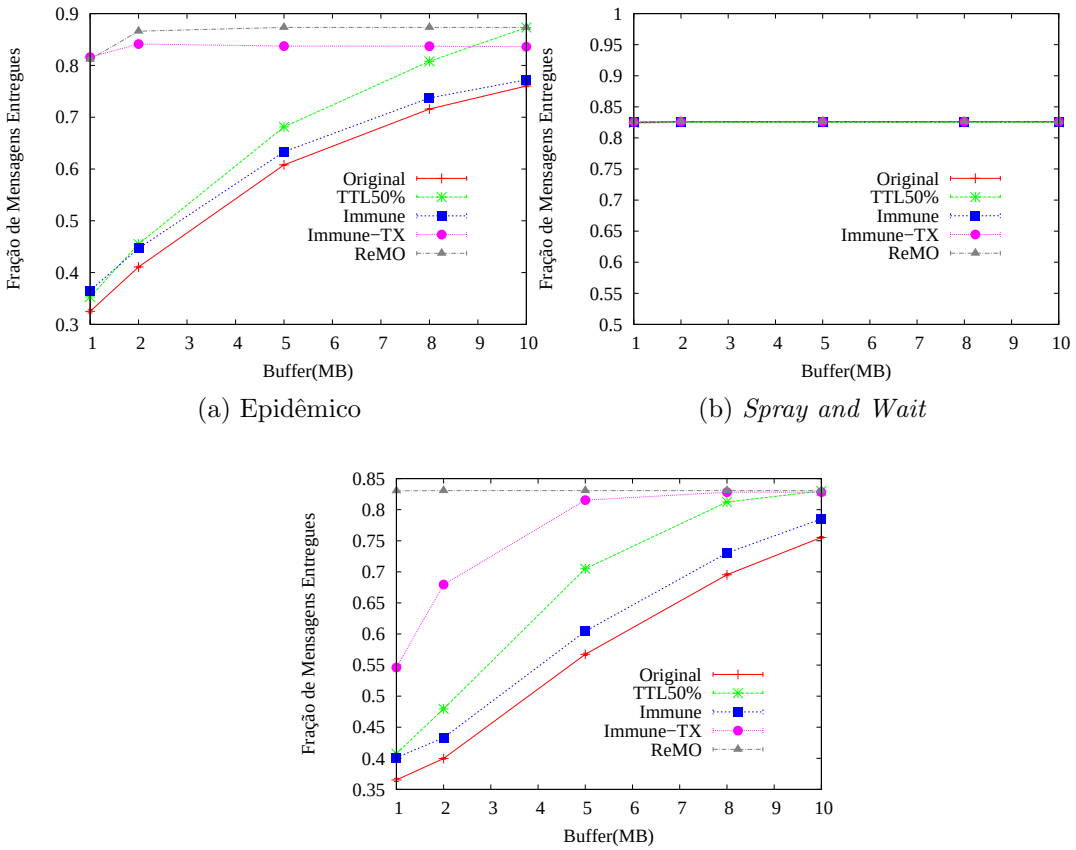
Figura 5.2: fração de mensagens entregues no cenário UCL1 para os mecanismos de remoção

A métrica fração de mensagens entregues também é analisada na Figura 5.2b, onde temos o gráfico com o uso do protocolo de roteamento *Spray and Wait*. Este

protocolo de roteamento tem em sua implementação um controle na disseminação das mensagens na rede, com isso, o número de cópias na rede é controlado. Mas, mesmo tendo este controle, o uso dos mecanismo de remoção mostrou favorável, aumentando a fração de mensagens entregues em relação ao sem mecanismo. O ReMO apresentou uma melhor fração de mensagens entregues entre todos os mecanismo, isto deve-se ao fato de remover mais mensagens obsoletas que os outros mecanismo, favorecendo na entrega de mensagens que antes eram descartadas. Já com o uso do TTL em 50% não obteve também um bom resultado para este protocolo, pois, como já mencionando anteriormente, o descarte das mensagens no meio da simulação impediram a entrega de mensagens, influenciando diretamente no resultado.

Na Figura 5.2c também é analisada a métrica fração de mensagens entregues no cenário UCL1, onde o *buffer* é variado, com o tamanho das mensagens definido em 10KB, e protocolo de roteamento *Bubble Rap*. É possível observar que para o *buffer* pequeno, entre 1M e 10M os mecanismos apresentaram um desempenho melhor e à medida que o *buffer* vai aumentando este desempenho vai igualando ao sem mecanismo e até mesmo piorando. Para esse traço de mobilidade com poucos nós, observamos que os mecanismos Immune e Immune-TX pioram o desempenho a partir de 12M, nesta fase o *buffer* não transborda mais, a sua capacidade fica abaixo de 80%, conforme Figura 5.10c, como estes mecanismos não disseminam a lista de mensagens entregues para todos os nós, trocam um número maior de mensagens obsoletas, influenciando de forma negativa nos resultados. Já o ReMO mantém um melhor desempenho até o 25M. O resultado para o TTL de 50% é o pior, pois como ele descarta as mensagens com o tempo médio de simulação, as chances reduzem bastante, visto também que o protocolo *Bubble*

Rap utiliza o contexto social para encaminhar as mensagens até o destino, selecionando os nós com mais popularidade, como o tempo de vida das mensagens reduzem influenciam diretamente neste mecanismos piorando o seu desempenho.



(a) Epidêmico

(b) *Spray and Wait*

(c) *Bubble Rap*

Figura 5.3: fração de mensagens entregues no cenário Rollernet para os mecanismos de remoção

A Figura 5.3a é feita uma analise da métrica fração de mensagens entregues no cenário Rollernet, onde variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Observa-se novamente uma melhora na métrica fração de mensagens entregues com o uso do mecanismo ReMO, pois mantém o *buffer* abaixo de 80% da sua ocupação, mesmo quando

o tamanho do mesmo é de apenas 1MB, favorecendo o recebimento de novas mensagens, por conseguinte aumentando a fração de entregar mais mensagem no destino. Esta melhora também ocorre com o Immune-TX, devido a movimentação dos nós, aumenta a funcionalidade deste mecanismo, chegando bem próximo ao funcionamento do ReMO. Já o Immune que não avisa aos nós intermediários mantém uma média melhor que o sem mecanismo, mas inferior até mesmo ao TTL de 50%, que como mencionado anteriormente tem uma probabilidade melhor por ser favorecido pelo traço de mobilidade real.

Na Figura 5.3b também é analisada a métrica fração de mensagens entregues no cenário Rollernet, onde variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. É possível observar que para este cenário e com o uso deste protocolo que tem como característica a baixa taxa de ocupação do *buffer*, o uso dos mecanismos de remoção se mostraram pouco efetivo. É possível também observar que devido ao traço de mobilidade o uso do TTL de 50% também teve uma fração de mensagens entregues igual.

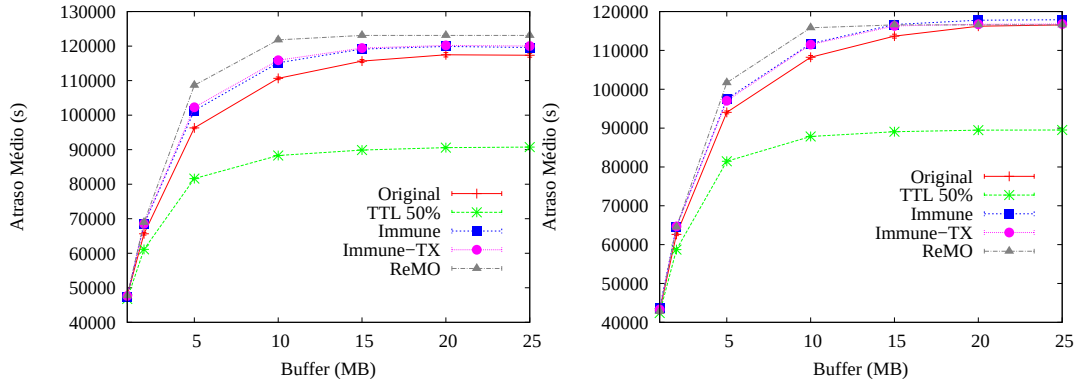
Na Figura 5.3c é analisada a métrica fração de mensagens entregues no cenário Rollernet, onde variando o *buffer*, com tamanho das mensagens definido em 10KB e protocolo de roteamento *Bubble Rap*. Observa-se que a fração de mensagens entregues para o protocolo *Bubble Rap* com o mecanismo de remoção ReMO mantém-se estável numa fração de mensagens entregues de aproximadamente em 83%, isto deve-se ao fato do mecanismo remover as mensagens obsoletas, aumentando o número de novas mensagens recebidas. Se observarmos as taxas de ocupação com o uso deste protocolo, desde o *buffer* de 1MB, Figura 5.13a, veremos que a ocupação do *buffer* não chegou a 10%. O mecanismo Immune-TX também obteve um bom resultado, não superior ao ReMO, mas como também

removeu mais mensagens que o outro mecanismo, obteve um resultado melhor. Já o TTL de 50% surpreende pelo seu desempenho, mas ao observarmos os contatos existentes neste traço de mobilidade real, verifica-se que na metade da simulação foram feitos 1851 contatos entre os nós, e na segunda metade foram feitos mais 3080, logo a fração de mensagens entregues serem entregues na segunda parte da simulação aumenta, explicando o resultado.

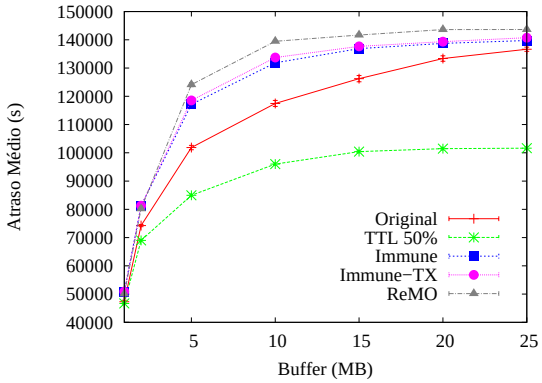
5.5.2 Atraso Médio para os mecanismos de remoção

O atraso médio é outra métrica importante na análise de desempenho em DTN, definido na Seção 5.4. Espera-se que com o uso do mecanismo de remoção de mensagens obsoletas (ReMO) esta métrica sofra uma redução na sua taxa, principalmente nos casos onde o *buffer* tem os seus tamanhos reduzidos e o descarte de mensagens se faz mais necessário.

Na Figura 5.4a também é analisada a métrica atraso médio no cenário UCL1, onde variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Ao analisarmos os resultados apresentados, observamos que o mecanismo ReMO apresenta o pior resultado. Isto ocorre porque o ReMO, ao remover mensagens que já foram entregues no destino, aumenta o espaço no *buffer* evitando que a política de gerenciamento de *buffer*aja. Como a política configurada é a FIFO, que remove a mensagem mais antiga, se ela não atua, mensagens com mais tempo no *buffer* permanecem, aumentado à chance de chegar ao destino. Desta forma, a métrica atraso médio, que é definida como, tempo médio que uma mensagem leva para chegar ao destino, termina apresentando um resultado ruim. Para os outros dois mecanismos ocorreu a mesma



(a) Epidêmico (b) *Spray and Wait*



(c) *Bubble Rap*

Figura 5.4: Atraso médio no cenário UCL1 para os mecanismos de remoção

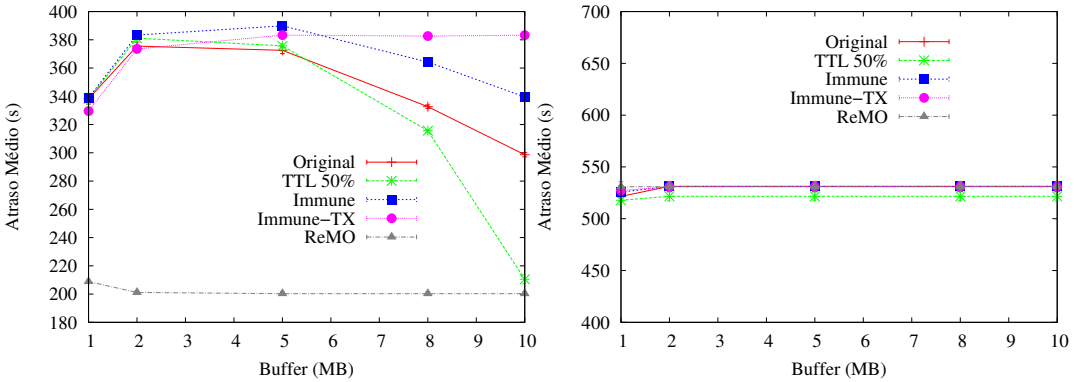
situação, como removem um número menor, o atraso é menor em relação ao ReMO. O TTL de 50% obteve um resultado melhor em relação a todos os outros analisados, isto também se deve a definição desta métrica, pois como o tempo foi dividido em duas partes, logo as mensagens que chegaram antes desta metade obtiveram um tempo de criação e entrega pequeno, na segunda metade o tempo também será pequeno entre criação e entrega resultado neste resultado melhor.

Na Figura 5.4b também é analisada a métrica atraso médio no cenário UCL1, onde variando o *buffer*, com tamanho das mensagens definido em 10KB e o proto-

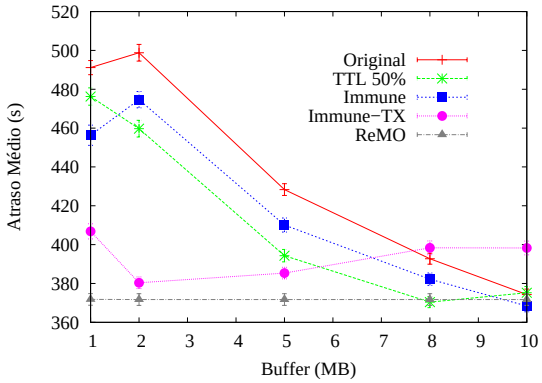
colo de roteamento *Spray and Wait*. Nesta figura observamos que os mecanismos tiveram um resultado não muito significativo, isto se deve a própria definição do protocolo de roteamento. Como este protocolo não dissemina muitas mensagens na rede, faz com que os resultados fiquem bem próximos tanto dos que utilizaram mecanismo como o sem mecanismo. Já para o caso do TTL de 50%, também ocorre a mesma coisa já analisando anteriormente, onde o tempo é dividido em duas partes, fazendo com que o tempo de criação e entrega sejam bem reduzidos.

Na Figura 5.4c é analisada a métrica atraso médio no cenário UCL1, onde variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Pode-se observar que o desempenho dos mecanismo são aparentemente piores, principalmente o ReMO, isto se deve à própria definição desta métrica, que contabiliza o tempo médio que uma mensagem leva para ser entregue, desde quando é gerada até o seu destino. Como os mecanismo descartam as mensagens já entregues no destino e ficam mais tempo com mensagens mais antigas que normalmente são descartadas quando o *buffer* está cheio, por este motivo até mesmo o desempenho de TTL de 50% foi melhor, pois como entregou mensagens novas, onde o tempo de criação e entrega eram menores, obteve um melhor resultado.

Na Figura 5.5a também é analisada a métrica atraso médio no cenário Rollernet, variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Pode-se observar que o uso do ReMO melhorou a métrica atraso médio em relação aos outros mecanismo e também ao original, neste cenário a mobilidade entre o nós é grande, aumentando os contatos e consequentemente aumentando a troca de mensagens. Como o ReMO usa os contatos para atualizar a sua lista de mensagens entregues no destino para em



(a) Epidêmico (b) *Spray and Wait*



(c) *Bubble Rap*

Figura 5.5: Atraso médio no cenário Rollernet para os mecanismos de remoção

seguida removê-las, por isto teve um resultado melhor que os outros mecanismo. O TTL de 50% também apresentou um resultado favorável para este cenário, perdendo somente para o ReMO.

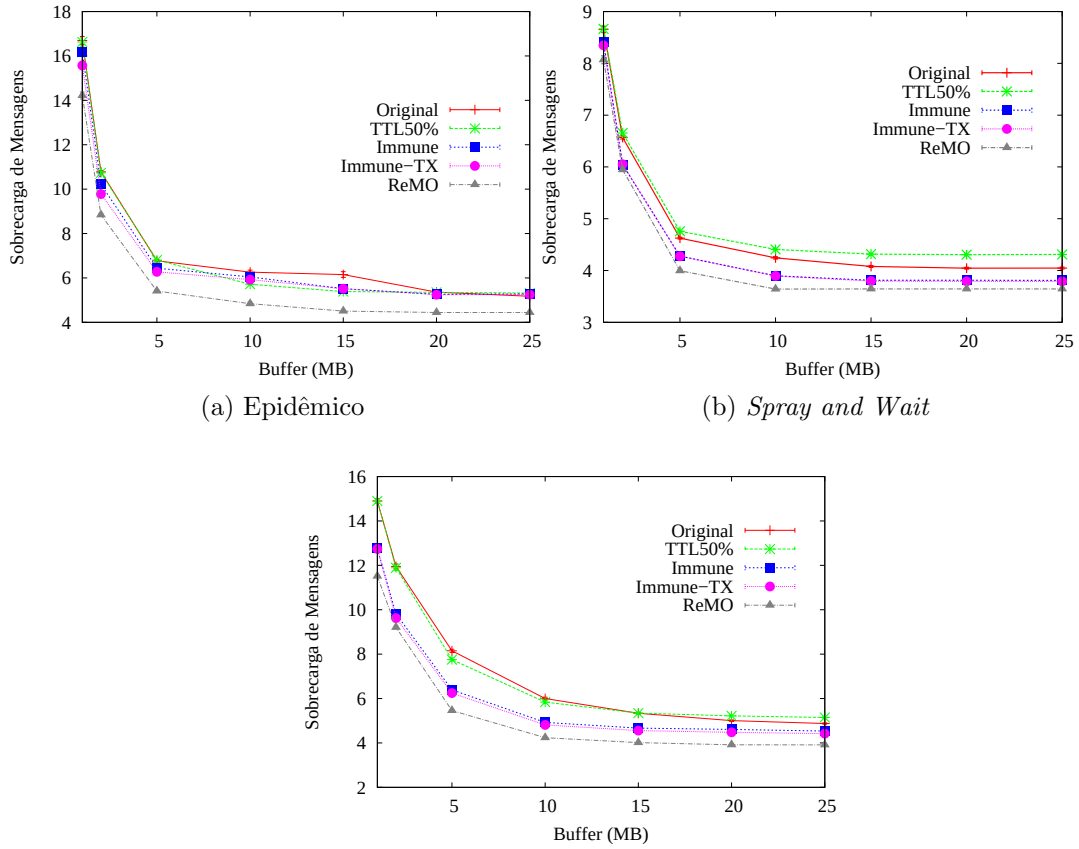
Na Figura 5.5b também é analisada a métrica atraso médio no cenário Rollernet, onde variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Neste cenário com o uso do protocolo *Spray and Wait* o uso dos mecanismo não apresentaram uma melhora na métrica atraso médio, pois este protocolo não dissemina muitas mensagens na rede.

Na Figura 5.5c também é analisada a métrica atraso médio no cenário Rollernet, onde variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Verifica-se que neste cenário o ReMO se mantém constante para esta métrica. Ao observar a métrica ocupação de *buffer* para toda variação do *buffer*, é verificado que a sua taxa de ocupação não chegou a 10%, fazendo com que o resultado se mantivesse constante. Já para os outros mecanismo ocorreu uma variação maior na taxa de ocupação dos *buffer*, produzindo resultados interessantes, onde o Immune apresenta um resultado melhor que o Immune-TX quando os *buffers* são aumentados, principalmente para o de 8M onde as métricas fração de mensagens entregues foram iguais e para atraso médio são bem diferentes. Isto se deve à implementação do protocolo de roteamento. Conforme o espaço nos *buffers* vão aumentando menos os mecanismos vão tendo influência.

5.5.3 Sobrecarga de mensagens para os mecanismos de remoção

Como definido na Seção 5.4, espera-se que com o uso do mecanismo de remoção de mensagens obsoletas (ReMO), a métrica sobrecarga sofra uma redução na sua taxa, principalmente nos casos onde o *buffer* tem os seus tamanhos reduzidos e o descarte de mensagens é maior.

Na Figura 5.6a também é analisada a métrica sobrecarga de mensagens no cenário UCL1, variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Verifica-se que para este cenário o mecanismo ReMO também apresenta uma melhora de performance para a métrica



(a) Epidêmico

(b) *Spray and Wait*

(c) *Bubble Rap*

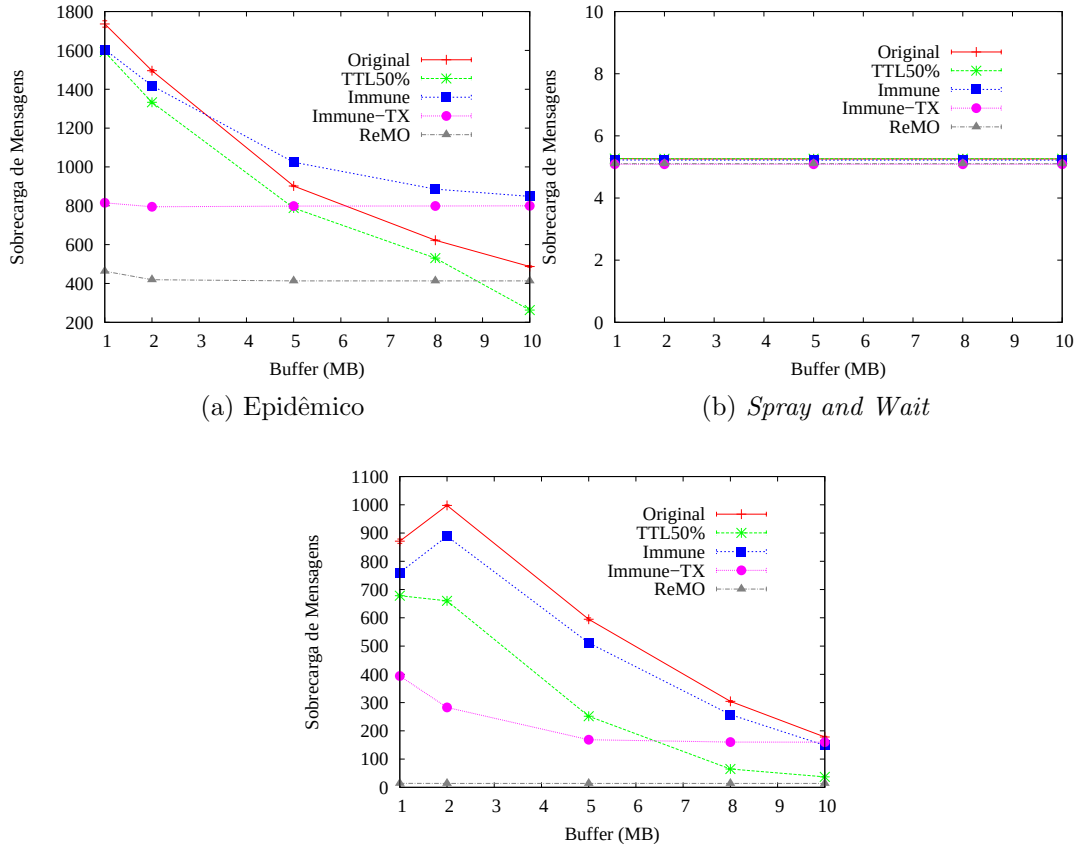
Figura 5.6: Sobrecarga de mensagens no cenário UCL1 para os mecanismos de remoção

sobrecarga de mensagens. Neste caso o protocolo de roteamento Epidêmico dissemina muitas mensagens pela rede, como o mecanismo remove as mensagens obsoletas, reduzindo a quantidade de mensagens repassadas, consequentemente melhorando esta métrica. Os outros mecanismos também removem as mensagens obsoletas, mas como removem uma quantidade menor, por exemplo a mecanismo ReMO removeu para este cenário, com *buffer* de 1MB, em média 1648, o Immune-TX 1454 e o Immune 1419. Observamos que a quantidade de mensagens removidas é diretamente proporcional ao resultado obtido.

Na Figura 5.6b também é analisada a métrica sobrecarga de mensagens no cenário UCL1, variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Mesmo para este protocolo que repassa um número menor de mensagens, ainda assim houve uma melhora na performance com a utilização do mecanismo ReMO e de outros mecanismos. Isto se deve ao fato da remoção das mensagens obsoletas, que evitam que mensagens já entregues no destino sejam reenviadas pela rede.

Na Figura 5.6c também é analisada a métrica sobrecarga de mensagens no cenário UCL1, variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Neste cenário podemos observar que o mecanismo ReMO teve um resultado melhor que os outros, pois o esta métrica tem como definição, calcula através da quantidade de mensagens repassadas dividido pela quantidade de mensagens entregues. Isto favorece o mecanismo ReMO em relação aos outros, pois como remove mais mensagens obsoletas que os outros, reduz a quantidade de mensagens repassadas na rede tendo um melhor desempenho.

Na Figura 5.7a também é analisada a métrica sobrecarga de mensagens no cenário Rollernet, variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Neste cenário temos uma grande mobilidade dos nós, isto favorece a entrega das mensagens e também a quantidade de repasses pelo uso do protocolo Epidêmico. Como os mecanismos removem mensagens já entregues no destino, possibilitando que mensagens que não são normalmente descartadas fiquem mais tempo na rede, isto faz com que estas mensagens sejam repassadas mais vezes pela rede. É possível observar que para o TTL de 50% que descarta as mensagens na metade da simulação, tem uma per-



(a) Epidêmico

(b) *Spray and Wait*

(c) *Bubble Rap*

Figura 5.7: Sobrecarga de mensagens no cenário Rollernet para os mecanismos de remoção

formance até melhor que o próprio ReMO para o *buffer* de 10MB, onde o número de mensagens repassadas é dividida pela quantidade de mensagens entregues.

Na Figura 5.7b também é analisada a métrica sobrecarga de mensagens no cenário Rollernet, variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Como se pode observar o uso dos mecanismos para este protocolo quase não sofre influencia no resultado para este traço de mobilidade real. Isto se deve ao fato do protocolo *Spray and Wait* ser um protocolo replicador controlado, onde dissemina um número controlado

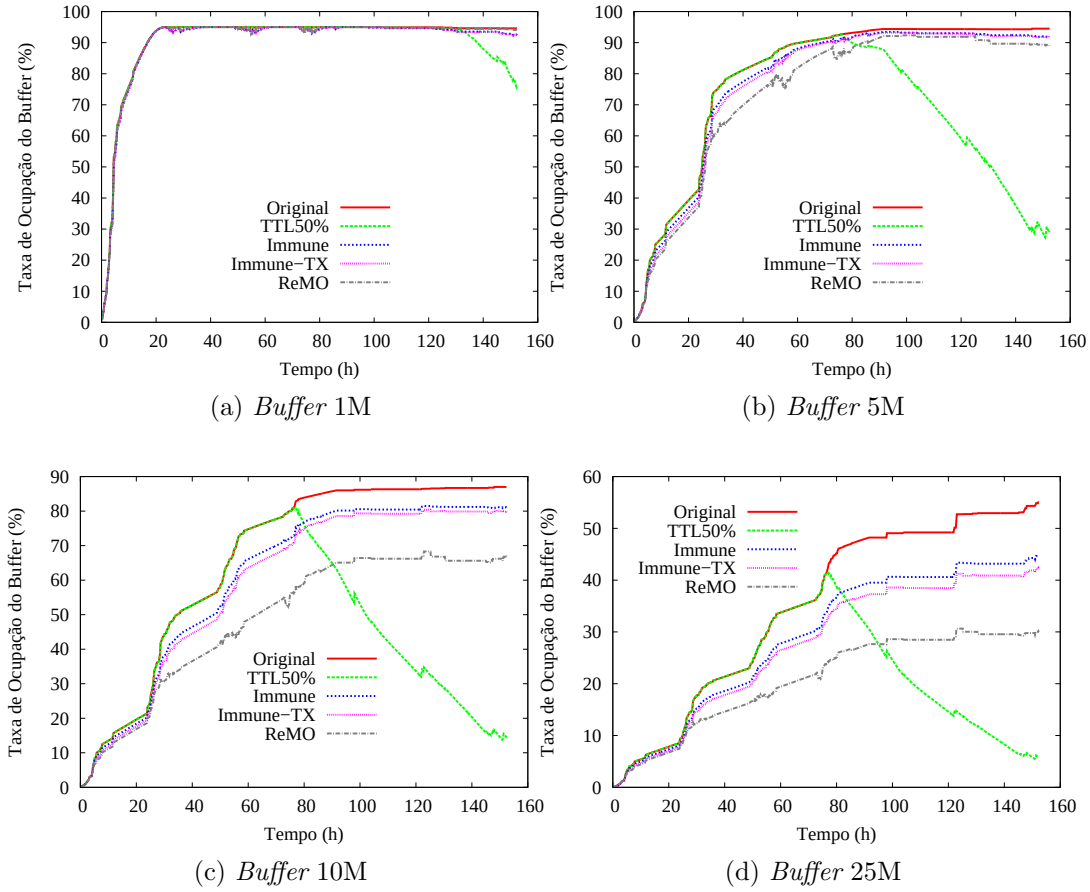
de mensagens na rede, como este traço de mobilidade tem um grau elevado de movimentação entre os nós, fazendo com que as mensagens sejam trocadas e entregues no destino mais rapidamente. Como a sobrecarga, em sua definição, é a quantidade de mensagens repassadas dividida pelo número de mensagens entregues, por existir um controle maior na replicação das mensagens, faz com que os mecanismo tenham pouca influência na remoção, tornando-os inexpressíveis.

Na Figura 5.7c também é analisada a métrica sobrecarga de mensagens no cenário Rollernet, variando o *buffer*, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Pode-se observar que o mecanismo ReMO mantém constante a sobrecarga de mensagens em relação a todas as variações de *buffer*, isto se deve a dois fatos importantes, o primeiro é que este protocolo já remove as mensagens dos nós que não têm chance de chegar ao destino, sendo que o uso dos mecanismos diminui mais ainda o número de mensagens que são repassadas pela rede, reduzindo consideravelmente a sobrecarga de mensagens.

5.5.4 Ocupação do *Buffer*

A métrica Ocupação do *Buffer* serve para mostrar o desempenho dos mecanismos em relação a remoção de mensagens obsoletas, isto faz com que o *buffer* tenha mais espaço para ser ocupado por outras mensagens que ainda não chegaram ao destino. Para esta métrica foi configurado o tempo de 600s, no qual é capturado a taxa média de ocupação nos *buffers*.

Na Figura 5.8a é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 1M, com tamanho das mensagens definido em 10KB e o protocolo



(a) *Buffer* 1M (b) *Buffer* 5M

(c) *Buffer* 10M (d) *Buffer* 25M

Figura 5.8: Ocupação do *Buffer* no cenário UCL1 com o protocolo de roteamento Epidêmico

de roteamento Epidêmico. Nesta figura observa-se que o *buffer* enche com 20 horas de simulação, para os mecanismos funcionarem as mensagens devem chegar ao destino, como este traço de mobilidade não tem uma grande mobilidade, onde temos aproximadamente seis dias de experimento para um total 512 contatos e que este traço de mobilidade apresenta os seus contatos bem distribuídos ao longo da simulação, mas mesmo assim o *buffer* enche com este tempo. O protocolo Epidêmico repassa todas as mensagens para todos os nós, buscando diminuir o tempo e aumentar a fração de mensagens entregues, os mecanismos parecem ter

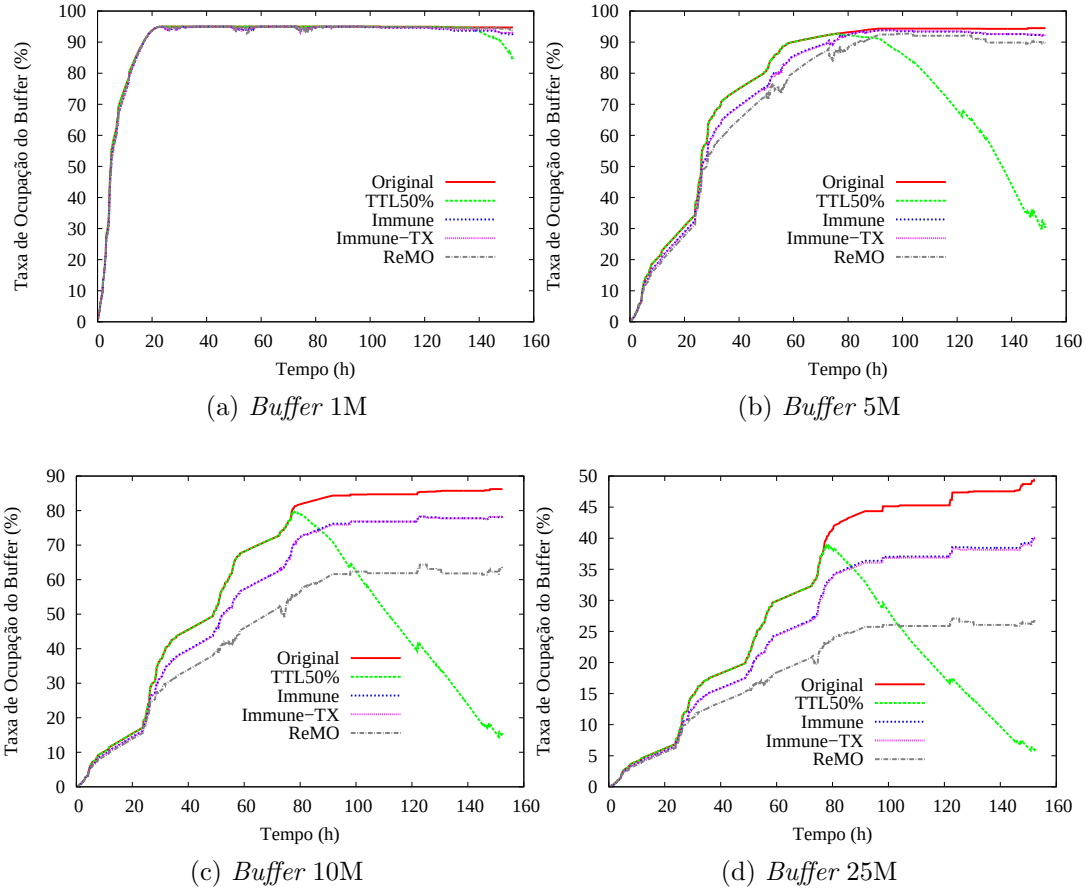
uma atuação pequena na melhoria do *buffer* na ordem de 1% do Original para o ReMO.

Na Figura 5.8b também é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Pode-se observar que o *buffer* enche com mais de 80 horas de simulação, enquanto que com os mecanismos fica próximo a este valor. Com esta capacidade de *buffer* os mecanismos atuam elemininando mais mensagens obsoletas. Pode-se observar que o mecanismo ReMO tem uma melhor performance, tendo sua taxa de ocupação máxima próximo de 92%. Outro fator que chama a atenção é a linha do TTL de 50%, que ao chegar na metade do tempo total de simulação ele descarta todas as mensagens melhorando a sua taxa de ocupação.

Na Figura 5.8c também é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 10M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Neste gráfico pode-se observar que nenhum dos mecanismos descarta mensagens por falta de espaço, ao contrário do uso do sem mecanismo que a aproximadamente 90 horas começa descartar mensagens. O mecanismo ReMO atua deixando um espaço no *buffer* de aproximadamente 30%, e os outros mecanismos em torno de 80%. É possível ver que mesmo os mecanismos Immune e Immune-TX serem diferentes, neste traço de mobilidade, quase não parecem existir devido as suas implementações.

Na Figura 5.8d é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 25M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Pode-se observar um fato interessante deste cenário, é que somente depois de 30 horas de simulação que vemos os mecanis-

mos agindo, até então a diferença entre eles são quase a mesma.



(a) *Buffer* 1M (b) *Buffer* 5M

(c) *Buffer* 10M (d) *Buffer* 25M

Figura 5.9: Ocupação do *Buffer* no cenário UCL1 com o protocolo de roteamento *Spray and Wait*

Na Figura 5.9a é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 1M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Neste cenário temos o protocolo it Spray and Wait que dissemina "$L = 6$" mensagens na Rede, mesmo assim, ele enche o *buffer* com pouco mais de 20 horas. Os mecanismos de remoção parecem quase não funcionarem pois mantém o espaço de ocupação quase igual ao sem mecanismo, neste *buffer* o ReMO melhora a taxa de entrega em aproximadamente 11%, isto

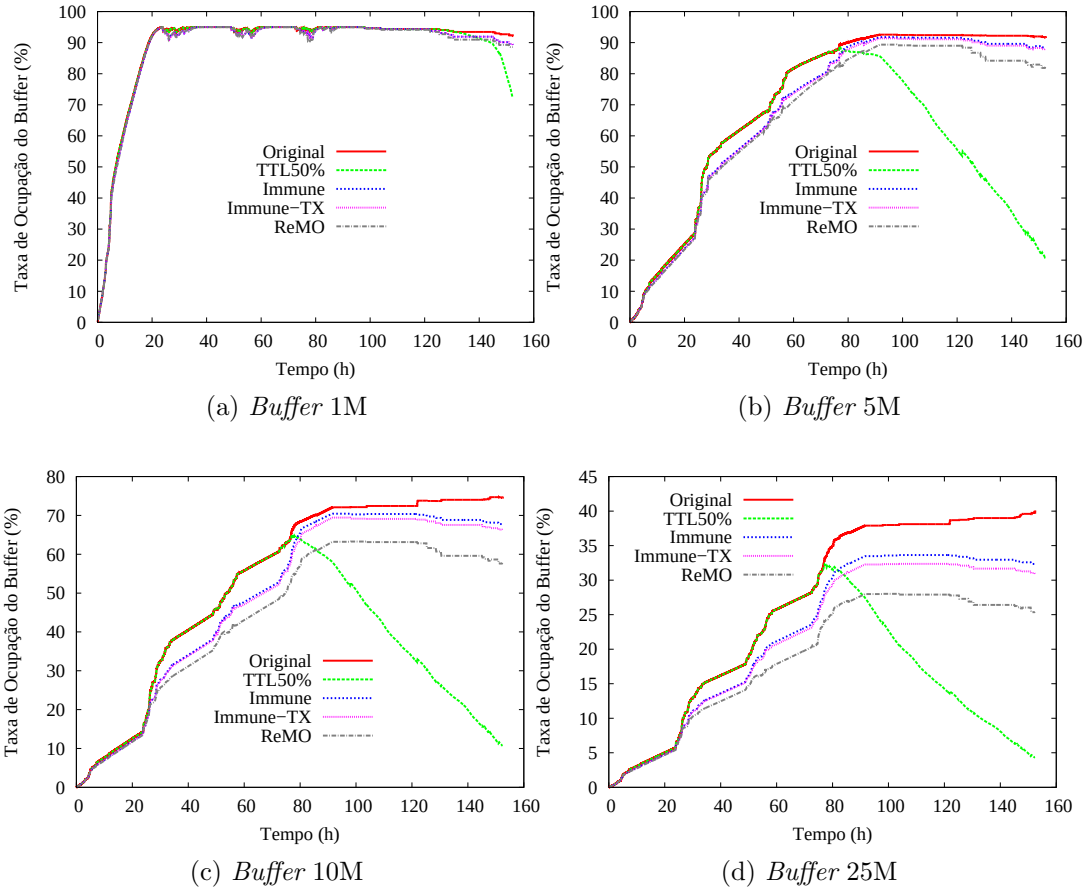
significa que mensagens estão sendo removidas e outras estão chegando.

Na Figura 5.9b também é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Nesta figura podemos observar que já existe uma maior remoção de mensagens obsoletas a partir de aproximadamente 30 horas, pois neste momento vemos que a taxa de ocupação diminui aumentando novamente em 80 horas para os mecanismos Immune e Immune-TX e somente as 100 horas para o ReMO. Isto deve-se ao fato de reduzir a taxa entrega neste momento aumentado assim a taxa de ocupação.

Na Figura 5.9c também é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 10M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Nesta figura observa-se que os mecanismos Immune e Immune-TX têm o funcionamento igual, isto ocorre devido a diferença entre os dois mecanismos, como o Immune-TX é igual ao Immune até que um nó tente passar uma mensagem que ele já tenha o conhecimento dela ter sido entregue no destino, e como neste intervalo de tempo isto não ocorre, o funcionamento dos dois se confundem.

Na Figura 5.9d é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 25M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Pode-se observar que conforme aumenta a capacidade, também aumenta o espaço de ocupação. Isto ocorre devido ao funcionamento dos mecanismos e do aumento da capacidade em cada *buffer*.

Na Figura 5.10a é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 1M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Neste gráfico podemos observar que o *buffer* enche



(a) *Buffer* 1M (b) *Buffer* 5M

(c) *Buffer* 10M (d) *Buffer* 25M

Figura 5.10: Ocupação do *Buffer* no cenário UCL1 com o protocolo de roteamento *BUBBLE Rap*

com pouco mais 20 horas de simulação, onde os mecanismos parecem ter uma atuação pequena na melhoria do *buffer*, mas se analisarmos que novas mensagens estão chegando e não estão sendo descartadas, aumentando a taxa de entrega.

Na Figura 5.10b é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Neste gráfico vemos que o *buffer* enche com aproximadamente 90 horas. Os mecanismos continuam atuando para reduzir o descarte de mensagens, sendo que o mecanismo ReMO não existe descarte de

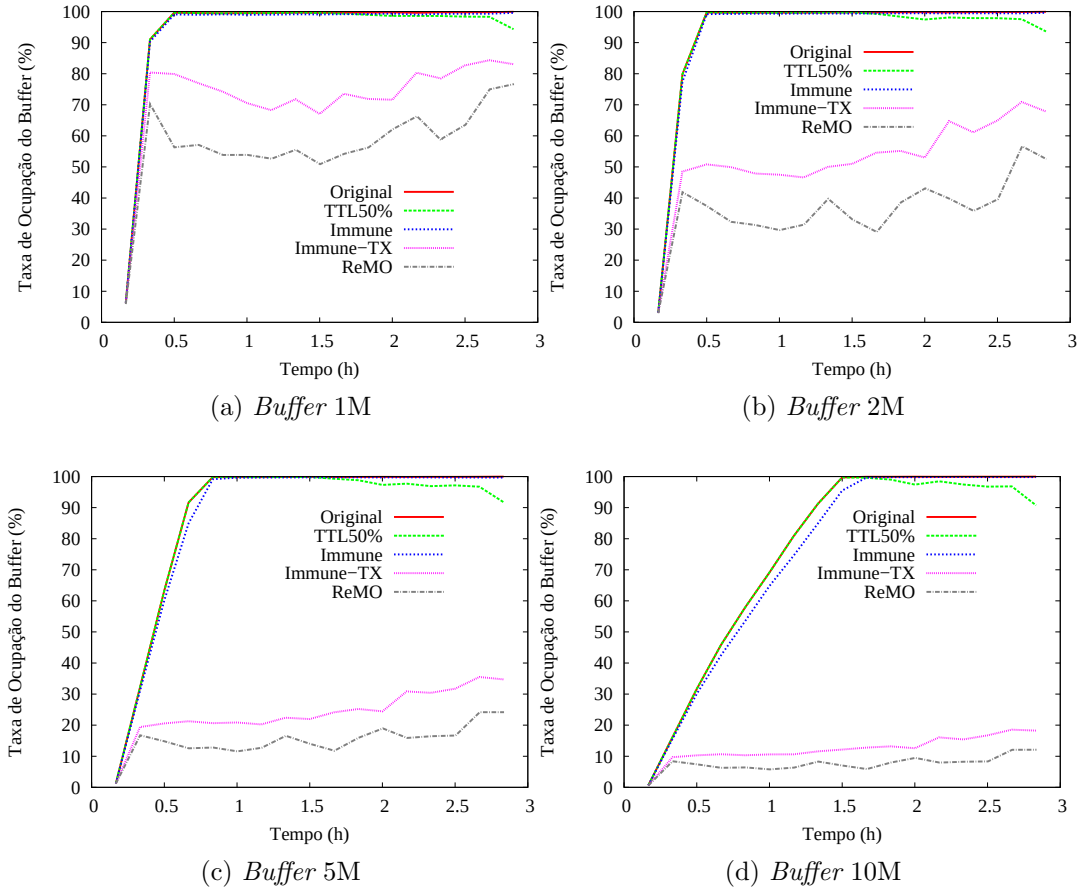
mensagens por transbordo.

Na Figura 5.10c é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 10M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Neste gráfico observamos que o *buffer* não transborda mais, e também conseguimos ver melhor como atua os mecanismos a partir de 30 horas de simulação. O ReMO continua sendo o melhor na parte de remoção de mensagens obsoletas.

Na Figura 5.10d é analisada a métrica ocupação do *buffer* no cenário UCL1, para o *buffer* de 25M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Neste gráfico vemos que o *buffer* não ocupa nem 50% da sua capacidade total, os mecanismos podem parecer desnecessários quanto a necessidade de armazenamento, mas com relação a outras métricas em que o repasse de mensagens é contabilizado (sobrecarga de mensagens).

Na Figura 5.11a é analisada a métrica ocupação do *buffer* no cenário Rollernet, para o *buffer* de 1M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Neste cenário temos uma grande mobilidade entre os nós, são 3081 contatos em três horas de simulação, fazendo com que as mensagens sejam trocadas num curto espaço de tempo. O protocolo Epidêmico dissemina muitas mensagens na rede, como isso observa-se que com 30 minutos de simulação o *buffer* transborda para o mecanismo Immune, TTL de 50% e o Original. Para os mecanismos Immune-TX e ReMO a ocupação não chega a transbordar o *buffer*, isto ocorre devido a grande mobilidade existente neste cenário, onde os dois mecanismos informam aos nós intermediários que as mensagens já chegaram ao destino, removendo-as.

Na Figura 5.11b é analisada a métrica ocupação do *buffer* no cenário Roller-



(a) *Buffer* 1M (b) *Buffer* 2M

(c) *Buffer* 5M (d) *Buffer* 10M

Figura 5.11: Ocupação do *Buffer* no cenário Rollernet com o protocolo de roteamento Epidêmico

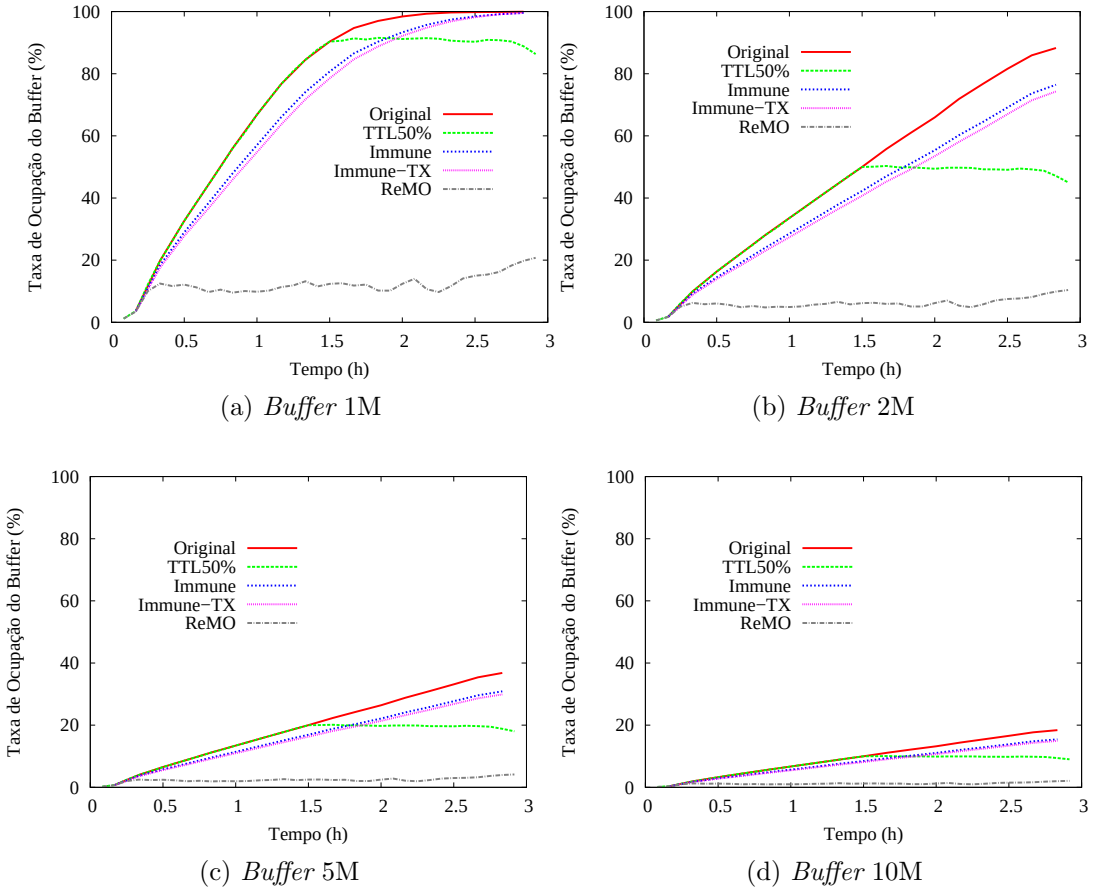
net, para o *buffer* de 2M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Neste gráfico podemos observar que mesmo aumentando a capacidade do *buffer* ele ainda continua transbordando em 30 minutos de simulação para o Immune, Original e TTL de 50%. Para os mecanismos Immune-TX e ReMO os dois diminuem o espaço ocupação quase pela metade do *buffer* de 1M. Os dois casos estão relacionados a mobilidade dos nós e a quantidade de contatos que chega a 4931 durante toda a simulação.

Na Figura 5.11c é analisada a métrica ocupação do *buffer* no cenário Rollernet,

para o *buffer* de 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Pode-se observar que a ocupação do *buffer* para o Immune-TX nestas condições só chega a no máximo de 35% de sua capacidade, para o ReMO ainda fica menos ainda ocupando 25%. O Immune, o TTL de 50% e o Original transbordam com quase 1 hora de simulação, onde a curva do Immune se desta um pouco dos outros dois em 30 minutos de simulação, igualando próximo a 1 hora, sendo que o TTL de 50%, por eliminar as mensagens na metade do tempo total de simulação, consegue após este tempo não mais descartar mensagens.

Na Figura 5.11d é analisada a métrica ocupação do *buffer* no cenário Rollernet, para o *buffer* de 10M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Pode-se observar a curva do Original e do TTL de 50% permanecem juntas até a metade da simulação. O Immune altera em relação ao tempo também, mas não deixa de transbordar, assim como o Original e o TTL de 50%. Já o Immune-TX varia entre 20% da sua ocupação total, e o no caso do ReMO fica entre 15% e 10% da ocupação total.

Na Figura 5.12a é analisada a métrica ocupação do *buffer* no cenário Rollernet, para o *buffer* de 1M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Pode-se observar que para este protocolo o desempenho do mecanismo ReMO ocupa no máximo 20% em toda a simulação. Isto se deve ao fato que o protocolo *Spray and Wait* tem um controle maior na disseminação de mensagens na rede, para esta simulação o número de cópias permitidas pelo protocolo são de $(L = 6)$. Com o uso do mecanismo ReMO eliminando as poucas cópias disseminadas na rede, a taxa de ocupação também diminui melhorando toda performance da rede. Para este protocolo a performance



(a) *Buffer* 1M (b) *Buffer* 2M

(c) *Buffer* 5M (d) *Buffer* 10M

Figura 5.12: Ocupação do *Buffer* no cenário Rollernet com o protocolo de roteamento *Spray and Wait*

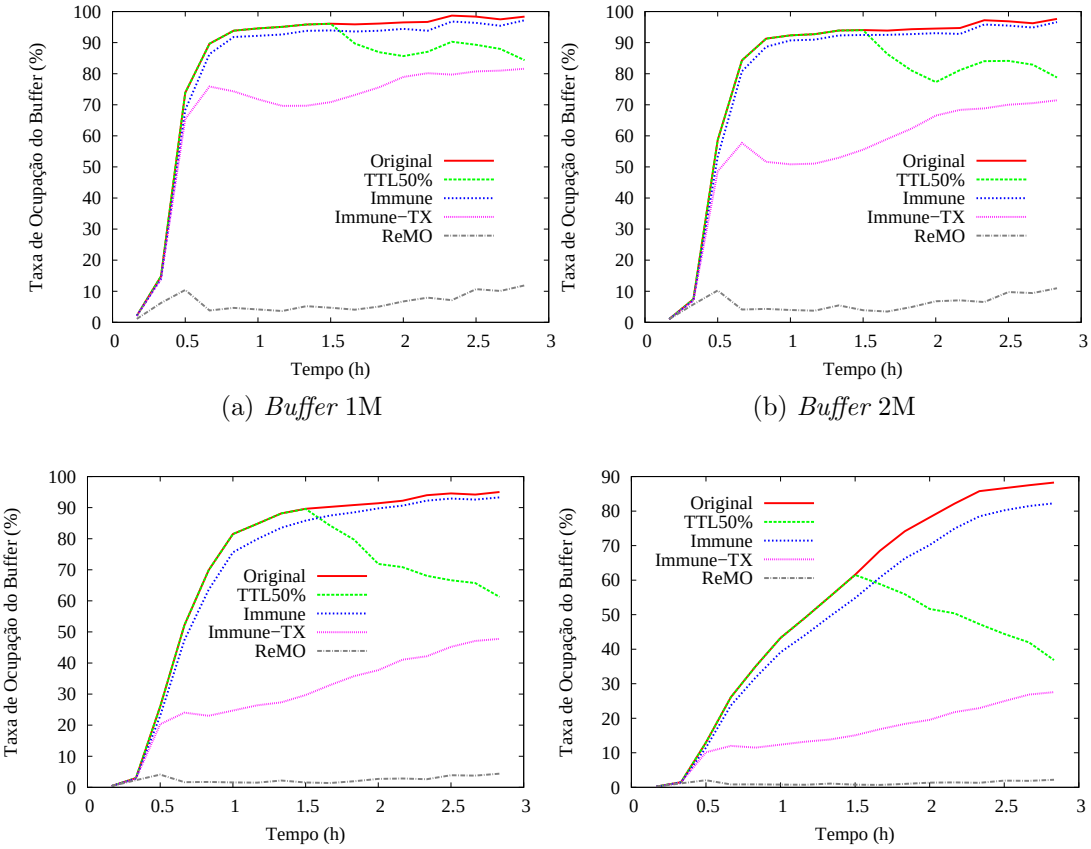
do Immune-TX é bem parecida com a do Immune, devido a pouca quantidade de mensagens que chegam sinalizadas de que já foram entregues no destino. O Original transborda o *buffer* com duas hora e meia de simulação e o TTL de 50% não chega a transbordar porque na metade do tempo as mensagens criadas inicialmente são descartadas.

Nas Figura 5.12b e 5.12c são analisadas a métrica ocupação do *buffer* no cenário Rollernet, para o *buffer* de 2M e de 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Observa-se que

com estas capacidades os *buffers* não transbordam mais, inclusive sem o uso de mecanismos, o comportamento dos dois gráficos são bem parecidos. O Original e o TTL de 50% se misturam até a metade da simulação devido a atuação do TTL; o Immune e o Immune-TX ficam bem próximos, como a diferença do dois mecanismo está em um enviar ao outro nó uma mensagem de controle, isto quase não acontece devido a particularidade do cenário e do protocolo de roteamento. O ReMO se mantém como o melhor mecanismo de remoção, tendo uma taxa de ocupação bem pequena comparada a Original, isto deve-se a grande quantidade de mensagens de controle disseminadas, fazendo com que sejam removidas mais mensagens, em todo o tempo de simulação são removidos para o *buffer* de 5M aproximadamente 105.000 mensagens.

Na Figura 5.12d é analisada a métrica ocupação do *buffer* no cenário Rollernet, para o *buffer* de 10M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Spray and Wait*. Pode-se observar que conforme aumenta o tamanho do *buffer* a taxa de ocupação diminui. Como mencionado anteriormente, isto deve-se a dois fatores importantes, o protocolo de roteamento empregado e o cenário utilizado.

Na Figura 5.13a é analisada a métrica ocupação do *buffer* no cenário Rollernet, para o *buffer* de 1M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Pode-se observar que o uso do mecanismo ReMO neste cenário apresentou uma melhoria de aproximadamente 90% no uso do *buffer*. Isto deve-se a grande mobilidade existente neste cenário, onde as mensagens são trocadas mais rapidamente, além do próprio funcionamento do protocolo de roteamento que escolhe o nó com mais "popularidade" entre os outros nós, proporcionando uma maior remoção das mensagens obsoletas. Além



(a) *Buffer* 1M (b) *Buffer* 2M

(c) *Buffer* 5M (d) *Buffer* 10M

Figura 5.13: Ocupação do *Buffer* no cenário Rollernet com o protocolo de roteamento *BUBBLE Rap*

disso, deve-se considerar que esta média é de todos os nós, não sendo analisado, em especial, para o nó de maior "popularidade".

Na Figura 5.13b é analisada a métrica ocupação do *buffer* no cenário Rollernet, para o *buffer* de 2M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Neste gráfico vemos uma melhoria no mecanismo Immune-TX e o ReMO manteve-se igual ao *buffer* de 1M. Isto deve-se a funcionalidade do protocolo de roteamento *Bubble Rap*. Outro fator importante que pode-se observar é com o mecanismo Immune-TX em relação ao Immune, os

dois apresentam comportamento bem diferentes quando usado o protocolo *Spray and Wait*, neste cenário o Immune-TX fica mais parecido com o ReMO. Como este protocolo escolhe o nó preterido para encaminhar as mensagens ao destino, isto auxilia também na informação e divulgação das mensagens que são entregues no destino, favorecendo a eliminação das mensagens obsoletas da reda.

Na Figura 5.13c é analisada a métrica ocupação do *buffer* no cenário Rollernet, para o *buffer* de 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Neste gráfico vemos que a taxa de ocupação do *buffer* para o mecanismo ReMO é de aproximadamente 2% da capacidade total. O Immune-TX tem sua taxa de ocupação máxima em 50%. Enquanto que o Original e o Immune ficam com taxas de ocupação bem elevadas. No caso do Immune o seu funcionamento para este protocolo não é muito favorável, pois ele só remove a mensagem quando ele é o nó destino ou ele quem entregou a mensagem no destino, como neste protocolo a escolha é de um nó preterido, isto impede a sua melhor funcionalidade, favorecendo aos outros dois que enviam a informação pela rede, no caso do Immune-TX e do ReMO.

Na Figura 5.13d é analisada a métrica ocupação do *buffer* no cenário Rollernet, para o *buffer* 10M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento *Bubble Rap*. Conforme aumenta a capacidade do *buffer*, observa-se que tanto o Immune-TX como o ReMO tendem a ficar bem próximo do mínimo de ocupação. A métrica ocupação do *buffer* é uma média de todos os nós, como este protocolo procura encaminhar a mensagem para um nó preterido, que tem mais popularidade, e como não foi realizada uma avaliação somente para estes nós, acredita-se que para estes nós o comportamento da ocupação do *buffer* seja diferente.

Capítulo 6

Avaliação do mecanismo ReMO em conjunto com Políticas de Gerenciamento de Buffer

Nesse capítulo é feito uma investigação sobre o uso das políticas de gerenciamento de *buffer* em conjunto com o mecanismo de remoção ReMO, onde foi escolhido o protocolo Epidêmico por ser um protocolo que dissemina um grande número de mensagens na rede, aumentado a chance de recebimento e entrega de mensagens, juntamente com o mecanismo que obteve os melhores resultados nas simulações, o ReMO. Além disso, espera-se que a junção de uma política com o mecanismo ReMO melhore o desempenho da rede num todo, tanto na taxa de entrega, na sobrecarga e no atraso médio.

6.1 Políticas de Gerenciamento de *Buffer*

Em DTNs o controle do espaço de armazenamento é fator primordial para o seu funcionamento. Sem espaço não existe troca de mensagens, consequentemente não existe uma rede. Um fator utilizado para evitar este fato são as políticas de gerenciamento de *buffer*. Na literatura encontramos várias sugestões de uso destas políticas. Os mecanismos de remoção de mensagens só passam a funcionar depois que mensagens são entregues no destino, este tempo pode ser insuficiente para evitar de encher o *buffer* de alguns nós, pois os mecanismo só começam a atuar depois que as mensagens começam a ser entregues no destino. O uso do TTL - *(Time-To-Live)*, tempo de vida de uma mensagem, pode também não ser suficiente para que mensagens não sejam descartadas desnecessariamente, como é apresentado neste trabalho. Por isso, neste trabalho também é apresentado um estudo de algumas políticas de gerenciamento de *buffer* que, associadas à mecanismos de remoção de mensagens, podem melhorar o desempenho da rede.

A preocupação em remoção de mensagens é abordada em [S. Rashid (2010)], onde é proposta uma política de gerência de *buffer* em DTN. Quando o *buffer* do nó está cheio e o mesmo precisa de espaço para armazenar uma nova mensagem, a maior mensagem no *buffer* será descartada. Esta estratégia foi chamada de *droplargest(DLA)*. Entretanto, o mecanismo prejudica o funcionamento das aplicações que geram mensagens grandes.

No trabalho [Krifa et al. (2008)], são propostas políticas de gerenciamento de *buffer* eficientes para redes tolerantes a atrasos. É apresentado o drop-tail drop-front. Usando todas as informações que são relevantes à disseminação de mensagem, é proposta uma política de gerenciamento de *buffer* ideal, com base

no conhecimento global da rede. Essa política pode ser ajustada tanto para minimizar o atraso médio de entrega ou para maximizar a taxa média de entrega.

[Naves et al. (2012)] descreve duas políticas de gerenciamento de *buffer* para redes DTNs. A LRF, que descarta a mensagem que foi encaminhada há mais tempo, e a LPS, que utiliza a estimativa da quantidade de réplicas da mensagem que foram disseminadas na rede e a probabilidade de encontro do nó de destino, associado ao protocolo de roteamento. Foram avaliadas a taxa de entrega, a sobrecarga e o atraso de entrega. Além disso, utilizaram-se traços reais de mobilidade e dois protocolos de roteamento, Epidêmico e Prophet.

Em [Leela-Amornsin and Esaki (2010)] é apresentada a preocupação com políticas de gerenciamento de *buffer* proativas e reativas, e principalmente relacionadas às mensagens que podem ser descartadas previamentes e depois, num próximo contato, serem novamente recebidas. Num primeiro momento são mostrados os tipos de políticas de gerenciamento de *buffer*:

Políticas Reativas: a) Last-In First-Drop (LIFD): Rejeita a mensagem se o *buffer* está cheio; b) First-In First-Drop (FIFD):Exclui as mensagens armazenadas no *buffer* primeiro; c) Oldest-Drop or Least Remaining Life First-Drop: Usa o TTL para decidir qual será descartada, a que tiver o menor TTL; d) Youngest-Drop or Most Remaining Life First-Drop: Usa o TTL também e descarta a que tiver o maior TTL; e) Most-Forwarded First-Drop (MOFD): A mensagem que foi enviada o maior número de vezes que é a primeira a ser excluída, tornando as mensagens menos enviadas com mais chances; f) Least-Forwarded First-Drop (LEFD): A mensagem que tem baixa fração de mensagens entregues e foi encaminhada poucas vezes, é a primeira a ser excluída. Ela terá uma pequena chance de chegar ao destino.

Políticas Proativas: a) Time-Based TTL: Inicializa o TTL com um valor(tempo) e vai diminuindo de uma unidade com o passar do tempo, quando chega a zero a mensagem é apagada; b) Hop-Based TTL: Inicializa o TTL com um valor máximo de saltos relacionado a duas tuplas (b,h), onde b é igual ao número máximo de cópias de uma mensagem e h representa o número de saltos permitidos até que seja excluída, $TTL_{(b,h)}$, quando b = 0 a mensagem é excluída. O próximo nó recebe um $TTL_{(b,h-1)}$; c) Feedback Policy: O nó de destino reconhece a mensagem com uma política de *feedback*, então cada nó intermediário exclui a mensagem que transporta correspondente à mensagem de confirmação.

Para apagar mensagens de forma eficaz, geralmente implementa-se a política proativa com um TTL global como um controle final para garantir que as mensagens serão excluídas do sistema eventualmente. Em paralelo, usa-se política reativa para controlar a eliminação de mensagens quando o *buffer* DTN fica congestionado. Todas as políticas reativas (exceto LIFD) geralmente excluem as mensagens antes que elas tenham o TTL expirando, como resultado podem receber a mensagem de outros nós gerando o retorno de mensagem já descartada. Propõe-se então o Crédito baseado em Controle de Congestionamento (CCC), que tem por finalidade manter a alta fração de mensagens entregues e reduzir o número de réplicas. Este mecanismo se mostrou eficaz quando a mensagem é descartada por um nó e que venha posteriormente a recebê-la. Com isso, este mecanismo não só tenta evitar o recebimento de mensagens que já foram descartadas novamente, mas através de informações recebidas pela rede, tenta definir o tempo no parâmetro “lixo” (que é uma lista). O seu funcionamento é simples: Uma vez que qualquer nó tenha apagado as mensagens, que é distinguida pelo seu ID, cada entrada da mensagem excluída será listado no “lixo” até o momento

de expirar por TTL. Quando uma mensagem tenta retornar para qualquer nó, é verificado se seu ID está listado, caso esteja, esse nó irá ignorar esta mensagem.

Neste trabalho é apresentado um estudo de políticas de gerenciamento de *buffer* em conjunto a mecanismos de remoção de mensagens obsoletas que podem melhorar o desempenho da rede. Foram testadas três políticas distintas, duas que descartam a mensagem conforme um critério pré estabelecido e outra que descarta sem nenhum fator estabelecido.

As políticas de gerenciamento de *buffer* agem quando não existe mais espaço suficiente para armazenamento de mensagens, evitando que a rede deixe de funcionar. As políticas testadas foram:

- **FIFO -** *(First In, First Out)*
  - Nesta política a mensagem que está a mais tempo no *buffer* é descartada.
- **Aleatório**
  - Nesta política, a eliminação é feita sem nenhum fator estabelecido. Assim, é realizado um sorteio de todas as mensagens que estão no *buffer* e a escolhida é descartada.
- **LRF -** *(Least Recently Forwarded)*
  - A política LRF visa descartar a mensagem que já foi disseminada e que já atingiu todos ou grande número de nós. Se nenhuma mensagem armazenada no *buffer* do nó foi previamente encaminhada por ele, a LRF descarta a mensagem que está há mais tempo no *buffer*.

6.2 Resultados obtidos da comparação entre as políticas de gerenciamento de *buffer* com e sem o uso do mecanismo ReMO

Nessa seção são apresentados os resultados da comparação entre as políticas de gerenciamento de *buffer* com o mecanismo ReMO. Foram utilizados quase todos os parâmetros utilizados nas simulações com os mecanismos, porém uma das diferenças foi no tamanho dos *buffers*, que só foram utilizados os tamanhos de 1M, 2M e 5M e a outra está no uso apenas do protocolo Epidêmico. Estas mudanças foram feitas para uma melhor visualização da necessidade de uma política de gerenciamento de *buffer* com um mecanismo de remoção de mensagens obsoletas.

6.2.1 Fração de mensagens entregues

Como definido na seção 5.4, espera-se que as políticas de gerenciamento juntamente com o uso do mecanismo de remoção obsoletas (ReMO), aumente a fração de mensagens entregues para os cenário UCL1 e Rollernet.

Na Figura 6.1 é analisada a métrica fração de mensagens entregues no cenário UCL1 com uso de políticas de gerenciamento de *buffer* e o mecanismo de ReMO, onde variando o *buffer* em 1M, 2M e 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Para uma melhor análise tanto das políticas de gerenciamento de *buffer* como do mecanismo ReMO foram selecionados três tamanhos de *buffer* para que as políticas pudessem agir. O uso do protocolo Epidêmico deve-se a dois fatos, o primeiro é que na maioria dos casos obsteve um melhor resultado em relação aos outros protocolos de roteamento

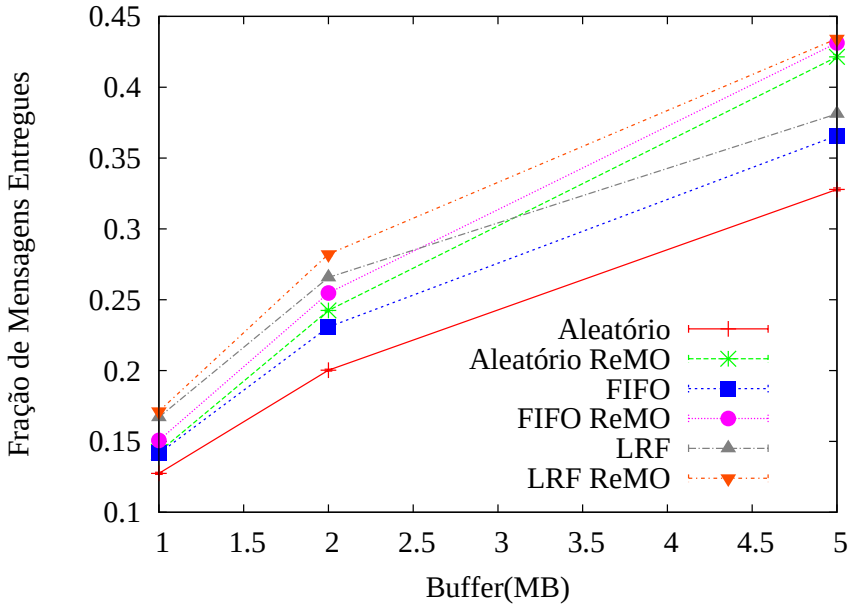


Figura 6.1: fração de mensagens entregues comparada as políticas de gerenciamento de *buffer* com e sem o mecanismo ReMO no cenário UCL1

analisados anteriormente, e o outro é que gera um número maior de cópias na rede. Analisando os resultados pode-se observar que todas as políticas de gerenciamento de *buffer* sofrem uma melhora em relação a sem o uso do mecanismo de remoção de mensagens obsoletas. O que chama bastante a atenção é a política Aleatória que obteve uma melhora de aproximadamente 10%, para o *buffer* de 5M, em relação ao sem mecanismo. A política LRF que descarta a mensagem que está há mais tempo no *buffer* associada ao mecanismo ReMO apresentou para o *buffer* de 2M em relação a política FIFO, que é usada como padrão pelo simulador *The One*, uma melhora de aproximadamente 7%, e para o sem mecanismo para este *buffer* foi pequena de aproximadamente 2%, mas para o *buffer* de 5M esta diferença chega a aproximadamente 7%.

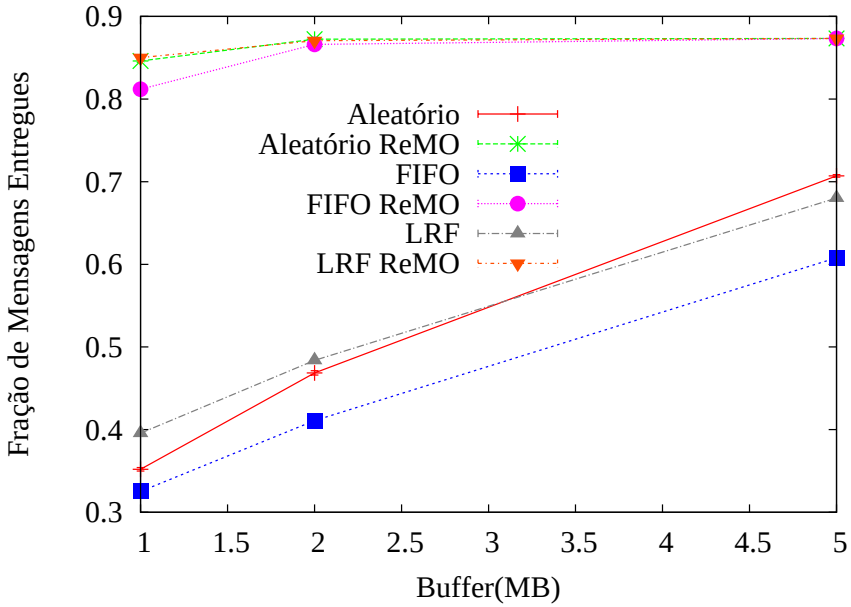


Figura 6.2: fração de mensagens entregues comparada as políticas de gerenciamento de *buffer* com e sem o mecanismo ReMO no cenário Rollernet

Na Figura 6.2 é analisada a métrica fração de mensagens entregues no cenário Rolernet com uso de políticas de gerenciamento de *buffer* e o mecanismo de ReMO, onde variando o *buffer* em 1M, 2M e 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Como mencionado anteriormente, os *buffers* foram alterados para uma melhor análise dos resultados e o uso do protocolo de roteamento Epidêmico. Pode-se observar que o uso do mecanismo ReMO para as três políticas de gerenciamento de *buffer* foram significativas. Ao analisarmos a política FIFO que descarta a mensagem mais antiga, o uso do ReMO melhorou a sua fração de mensagens entregues em aproximadamente 50%. A política Aleatória, onde descarta a mensagem sem nenhum fator estabelecido, se manteve até melhor que a FIFO para este cenário, onde a mobilidade dos nós é alta, também obteve uma melhora de aproximadamente 50%.

A política que obteve um melhor desempenho foi a política LRF, esta obteve uma melhora de 40% no seu desempenho, em relação a sem o uso do mecanismo ReMO. Isto se deve ao fato do mecanismo ReMO ter removido mensagens que poderiam ser escolhidas para descarte pela política LRF, por terem atingido um maior número de nós. Com isso, as escolhas foram favorecidas pela política LRF melhorando o seu desempenho.

6.2.2 Atraso Médio

O atraso médio é definido na Seção 5.4, espera-se que com o uso do mecanismo de remoção de mensagens obsoletas (ReMO) juntamente com as políticas de gerenciamento de *buffer*, ocorra uma redução na sua taxa.

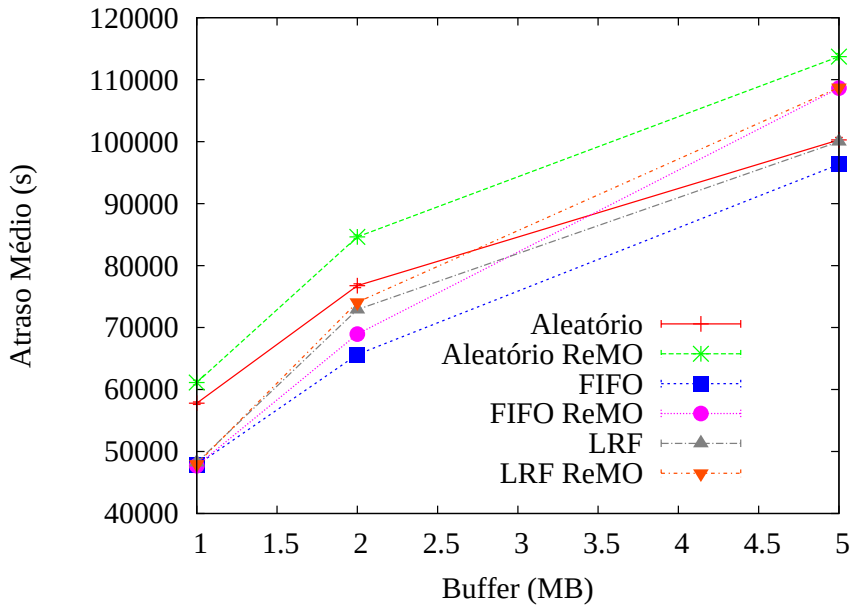


Figura 6.3: Atraso Médio comparadas as políticas de gerenciamento de *buffer* e o mecanismo de remoção ReMO no cenário UCL1

Na Figura 6.3 é analisada a métrica atraso médio no cenário UCL1 com uso

de políticas de gerenciamento de *buffer* e o mecanismo de ReMO, onde variando o it buffer em 1M, 2M e 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Pode-se observar que todos as políticas obtiveram um atraso médio maior em relação as que não usaram o mecanismo de remoção, isto deve-se ao fato da própria definição desta métrica que contabiliza o tempo médio que uma mensagem leva para ser entregue, desde quando é gerada até o seu destino, como melhorou o descarte destas mensagens, fez com que um número maior de mensagens pudessem chegar ao destino, onde o tempo de geração e entrega ficassem altos, fazendo com que aparentemente o resultado mostra-se insatisfatório.

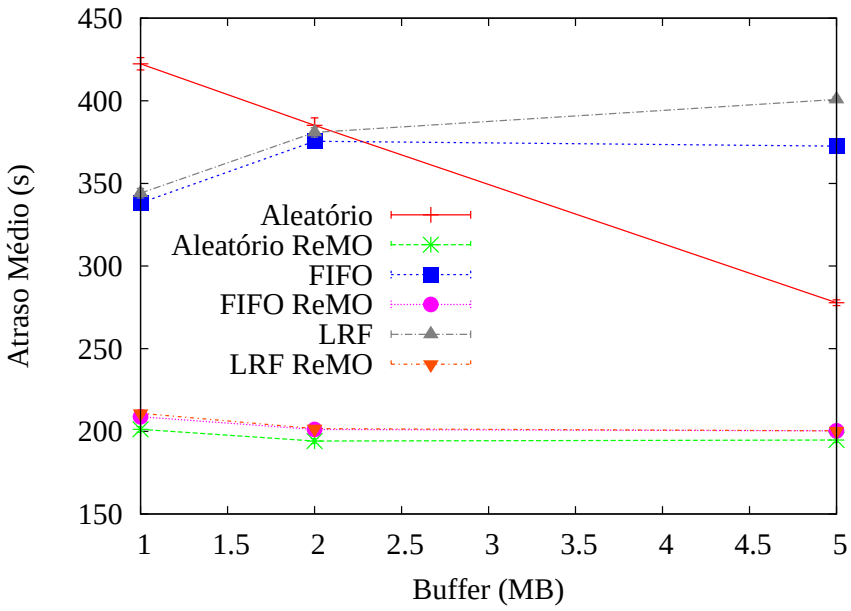


Figura 6.4: Atraso Médio comparadas as políticas de gerenciamento de *buffer* e o mecanismo de remoção ReMO no cenário Rollernet

Na Figura 6.4 é analisada a métrica atraso médio no cenário Rollernet com uso de políticas de gerenciamento de *buffer* e o mecanismo de ReMO, variando

o *buffer* em 1M, 2M e 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Observa-se que mesmo entregando mais mensagens, esta métrica teve uma melhora em todas as políticas que utilizaram o ReMO, isto deve-se ao fato que como este cenário a mobilidade é maior, favorece a troca das mensagens, e da lista de controle entregue, onde aumenta a remoção das mensagens obsoletas. Mesmo as mensagens que poderiam ficar mais tempo na rede, onde poderiam afetar o resultado, são entregues num curto espaço de tempo, melhorando o desempenho. Observamos que a política Aleatória tem um melhor desempenho em relação as outras, tanto a FIFO como a LRF as duas através do mecanismo de remoção sofrem uma melhora considerável de quase 200s.

6.2.3 Sobrecarga de mensagens

Definido na Seção 5.4, espera-se que com o uso do mecanismo de remoção de mensagens obsoletas (ReMO) juntamente com as políticas de gerenciamento de *buffer*, essa métrica sofra uma redução na sua taxa.

Na Figura 6.5 é analisada a métrica sobrecarga de mensagens no cenário Rollernet com uso de políticas de gerenciamento de *buffer* e o mecanismo de ReMO, onde variando o it buffer em 1M, 2M e 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Pode-se observar que o mecanismo melhorou a métrica sobrecarga de mensagens em relação as políticas de gerenciamento de *buffer*, isto deve-se a remoção de cópias de mensagens que já foram entregues no destino e que ainda estão nos nós intermediários.

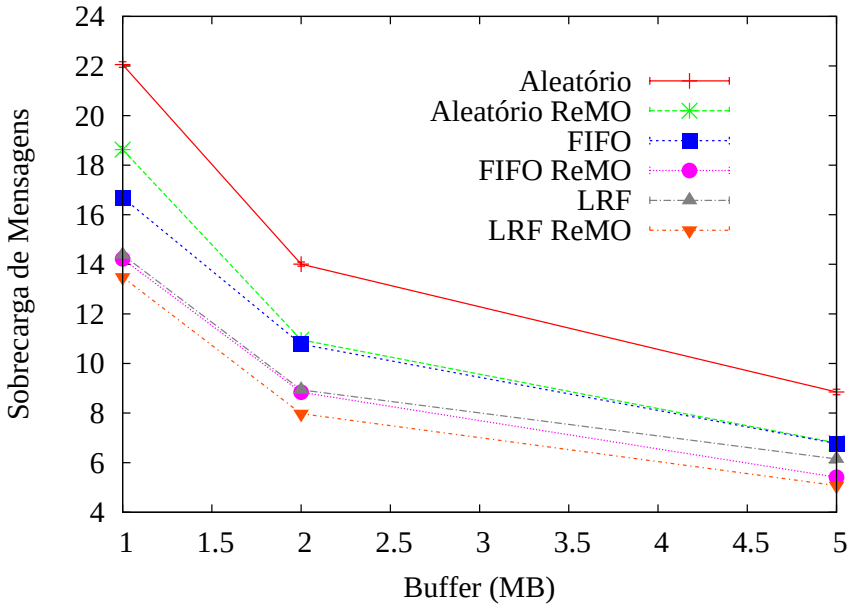


Figura 6.5: Sobrecarga de Mensagens comparadas as políticas de gerenciamento de *buffer* e o mecanismo de remoção ReMO no cenário UCL1

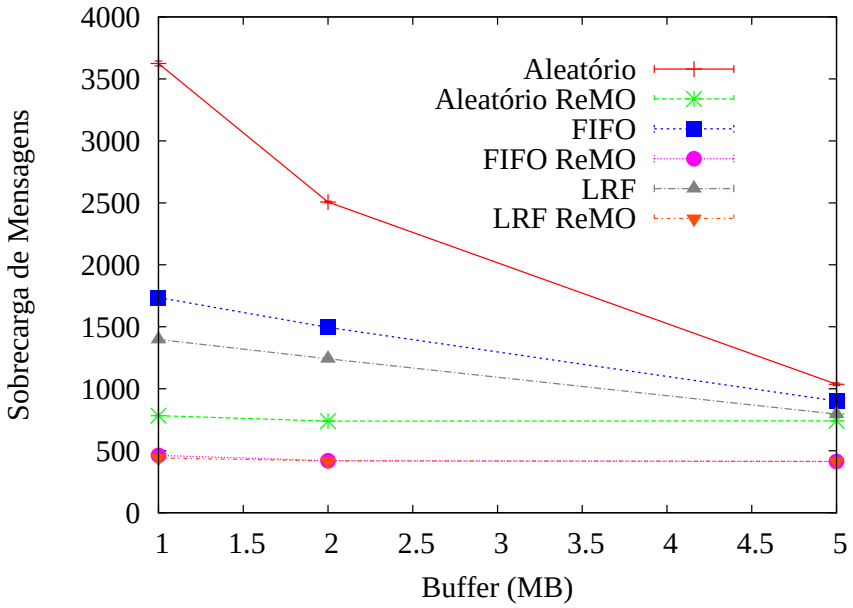


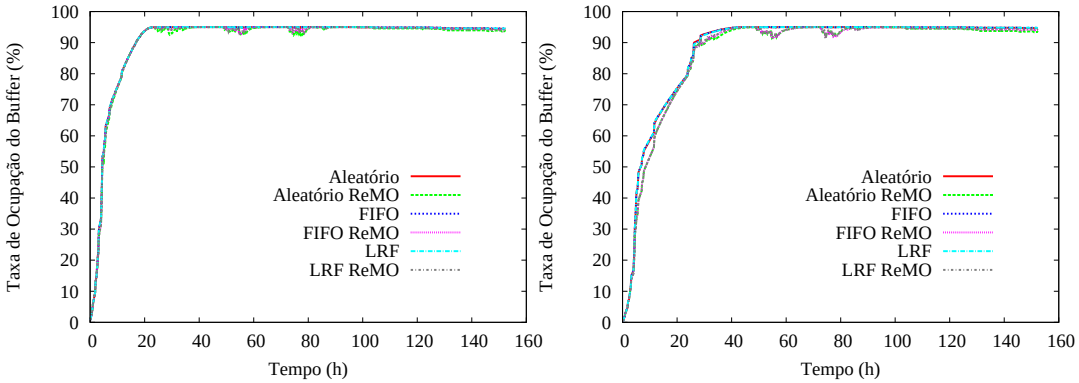
Figura 6.6: Sobrecarga de Mensagens comparadas as políticas de gerenciamento de *buffer* e o mecanismo de remoção ReMO no cenário Rollernet

Na Figura 6.6 é analisada a métrica sobrecarga de mensagens no cenário Rollernet com uso de políticas de gerenciamento de *buffer* e o mecanismo de ReMO, onde variando o it buffer em 1M, 2M e 5M, com tamanho das mensagens definido em 10KB e o protocolo de roteamento Epidêmico. Num cenário onde a mobilidade é alta, observa-se que o mecanismo ReMO diminui em mais de 100% o repasse das mensagens em relação a todas as políticas de gerenciamento de *buffer*. Pode-se observar que para a política FIFO a melhora foi de mais de 200% para o *buffer* de 1M. Outro fator interessante, são que as políticas FIFO e LRF, com o uso do mecanismo ReMO, passam a ter o mesmo comportamento no gráfico.

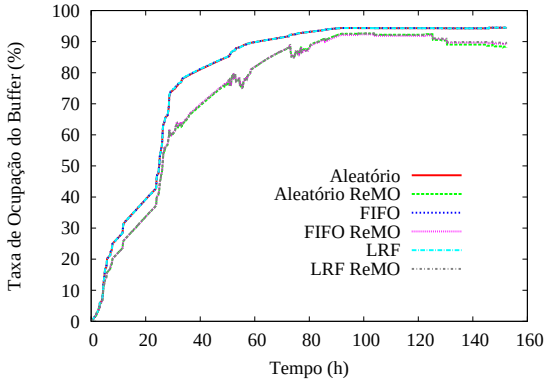
6.2.4 Ocupação do *Buffer*

A métrica Ocupação do *Buffer* é definida na Seção 5.4, para os resultados do uso das políticas de gerenciamento de *buffer* juntamente com o mecanismo ReMO. O uso dessa métrica serve para analisarmos o quanto a remoção de mensagens obsoletas ou não podem favorecer a taxa de ocupação do *buffer*. Como as políticas de gerenciamento começam a agir assim que o *buffer* tem a sua capacidade esgotada, este métrica pode não ser favoravel para uma análise dos resultados. Para esta métrica foi configurado o tempo de 600s, na qual é capturada a taxa de ocupação do *buffer*.

Nas Figuras 6.7a, 6.7b e 6.7c é analisada a métrica ocupação do *buffer* na cenário UCL1, com tamanho de mensagens definidos em 10KB, para o *buffer* de 1M, protocolo de roteamento Epidêmico e as políticas de gerenciamento de *buffer* com e sem o mecanismo ReMO. Na Figura 6.7a pode-se constatar que, mesmo sendo usado o mecanismo ReMO, para este cenário o mecanismo não



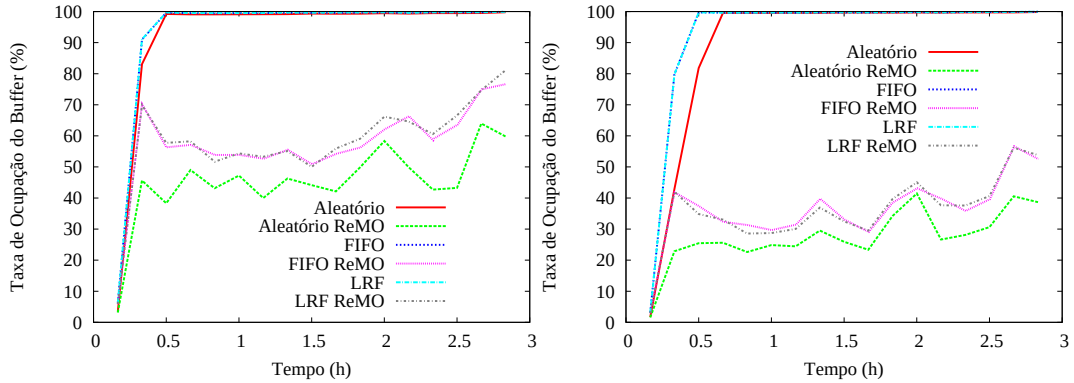
(a) *Buffer* 1M (b) *Buffer* 2M



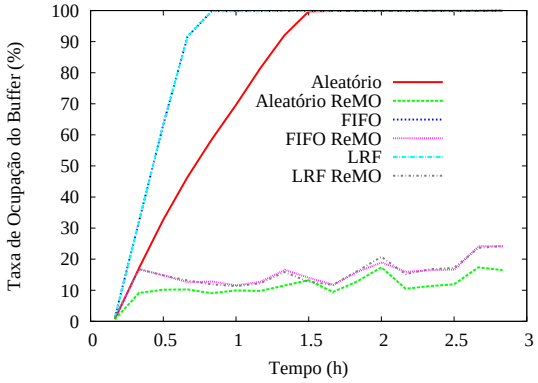
(c) *Buffer* 5M

Figura 6.7: Ocupação do *Buffer* no cenário UCL1 com o protocolo Epidêmico e as políticas de gerenciamento com e sem o mecanismo ReMO

consegue agir de modo satisfatório, tendo a necessidade de usar uma política de gerenciamento, pois rapidamente o *buffer* transborda. Infelizmente esta métrica não permite uma boa avaliação, pois ela só funciona após a ocupação do *buffer*. Na Figura 6.7c podemos constatar isto melhor, onde parecem existir dois grupos de linhas, os que estão na parte inferior do gráfico, são as políticas juntamente com o mecanismo ReMO e o outro somente com as políticas de gerenciamento de *buffer*.



(a) *Buffer* 1M (b) *Buffer* 2M



(c) *Buffer* 5M

Figura 6.8: Ocupação do *Buffer* no cenário Rollernet com o protocolo Epidêmico e as políticas de gerenciamento com e sem o mecanismo ReMO

Nas Figuras 6.8a, 6.8b e 6.8c é analisada a métrica ocupação do *buffer* na cenário Rollernet, com tamanho de mensagens definidos em 10KB, para o *buffer* de 1M, 2M e 5M, protocolo de roteamento Epidêmico e as políticas de gerenciamento de *buffer* com e sem o mecanismo ReMO. Para este cenário podemos observar que a política Aleatória tem a sua curva diferente das outras duas políticas sem o mecanismo, isto deve-se ao fato da política aleatória descartar mensagens sem nenhum fator estabelecido, mensagens estas que não serão replicadas pela rede. Para uma melhor compreensão, analisaremos tanto o funcionamento do protocolo

Epidêmico como da mobilidade existente neste traço real. O protocolo de roteamento Epidêmico conforme vai encontrando com outros nós, vai replicando as suas mensagens. Este traço de mobilidade real, Rollernet, tem um alto grau de mobilidade entre os nós, aumentando a troca de mensagens na rede. A política Aleatória descarta qualquer mensagem em qualquer tempo. Se inicialmente ele descartar uma mensagem que não tenha sido replicada e que levaria um tempo maior para ser entregue, isto significa que esta mensagem deverá trafegar na rede por um tempo maior. Uma quantidade maior de nós receberá esta mensagem, ocupando espaço em todos os nós que venha a receber. Desta forma, em algumas situações, tenha um desempenho melhor que as outras políticas.

Capítulo 7

Conclusão

Neste capítulo são apresentadas as conclusões do trabalho, onde foi proposto um mecanismo de remoção de mensagens obsoletas (ReMO), além de uma investigação sobre o uso de mecanismos de remoção juntamente com três políticas de gerenciamento de *buffer*. Por fim, algumas perspectivas para trabalhos futuros são descritas.

7.1 Conclusões

Em DTNs o armazenamento de mensagens é primordial para o seu funcionamento. Na literatura encontramos vários trabalhos voltados para este assunto, sejam eles através de protocolos de roteamento que procuram controlar a disseminação das mensagens na rede, ou através de mecanismos proativos ou reativos que procuram remover mensagens nos nós.

Neste trabalho é apresentado um mecanismo de remoção de mensagens obsoletas chamado ReMO, que foi comparado a outros três mecanismo de remoção,

o Immune, Immune-TX e o TTL de 50%, que em muitos casos tem a finalidade de remover as mensagens em um determinado tempo. Além disso, também investigou-se o uso de políticas de gerenciamento de *buffer* juntamente com o mecanismo de remoção de mensagens obsoletas apresentado.

O mecanismo de remoção de mensagens obsoletas (ReMO) pode ser usado independente do protocolo de roteamento, em alguns casos pode apresentar melhorias em todo o conjunto da rede.

É possível concluir, segundo análises realizadas através das simulações, que o mecanismo ReMO apresentou um resultado superior aos outros mecanismos aqui comparados. Pode-se observar, que mesmo usando protocolos onde o número de mensagens disseminadas é controlado, caso do *Spray and Wait*, ou onde o protocolo utiliza característica específica para disseminar as mensagens, caso do protocolo *BUBBLE Rap* que utiliza o contexto social, ele apresentou para estes dois protocolos uma melhora nas métricas avaliadas. Para o protocolo Epidêmico, que dissemina mensagens a todos os nós que tiver contato, para aumentar a chance de entrega e diminuir o atraso, também teve bons resultados o uso do ReMO.

O uso de traços de mobilidade real no simulador *The One* também contribui para termos um lado mais realístico na simulação, onde podemos observar que, dependendo da mobilidade dos nós, o uso do mecanismo pode sofrer alterações em relação ao protocolo de roteamento, como ocorreu na probabilidade de entrega que, para o cenário Rollernet, o protocolo Epidêmico teve um resultado melhor que o *Spray and Wait* para o cenário UCL1.

O uso de mecanismo de remoção de mensagens obsoletas juntamente com políticas de gerenciamento de *buffer* provou melhorar as métricas analisadas. No

caso da métrica pobabilidade de entrega para a política de gerenciamento de *buffer* LRF, esta política mostrou um bom desempenho quando utilizada juntamente com o mecanismo de remoção obtendo uma melhora superior a 100%.

7.2 Trabalhos Futuros

Como sugestão de trabalhos futuros, seria interessante fazer um estudo sobre a influência no tamanho da lista de remoção de mensagens obsoletas quando esta lista cresce e fica igual ou maior que o tamanho de uma mensagem.

Para este trabalho foram feitas simulações com três protocolos de roteamento, Epidêmico, *Spray and Wait* e *BUBBLE Rap*, seria importante verificar com outros protocolos existentes na literatura. Além de outros protocolos de roteamento, também seria importante fazer novas simulações com outros traços de mobilidade reais.

Como sugerido anteriormente, para verificar a influência do tamanho da mensagem neste contexto, poderia-se aumentar o tamanho das mensagens e observar o comportamento do mecanismo.

Outro trabalho seria excluir ou zerar todos os elementos da lista de controle de mensagens obsoletas de tempos em tempos, e verificar se vai exister uma melhora no desempenho.

Por fim, implementar em equipamentos reais e verificar se a eficiência se mantém em ambiente reais.

Referências Bibliográficas

A. Lindgren, A. Doria, O. S. (2003). Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20.

A. Vahdat, D. B. (2000). Epidemic routing for partially-connected ad hoc networks. *Technical Report CS-200006*, page 18. Duke University.

Abdesslem, F. B., Henderson, T., and Parris, I. (2011). CRAWDAD trace st_andrews/locshare/2010/ucl1 (v. 2011-10-12). Downloaded from http://crawdad.cs.dartmouth.edu/st_andrews/locshare/2010/UCL1.

An, Y., Huang, J., Song, H., and Wang, J. (2012). A congestion level based end-to-end acknowledgement mechanism for delay tolerant networks. *In: Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 1574–1579. IEEE.

Bian, H. and Yu, H. (2010). An efficient control method of multi-copy routing in dtn. *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, 1:153–156. IEEE.

DTNRG, . (2006). *DTN Research Group*. Grupo de Pesquisa em DTN, http://www.dtnrg.org/.

Fall, K. (2003). A delay-tolerant network architecture for challenged internets. *SIGCOMM '03*, Vol. 10(863960):27–34. 10.1145/863955.863960.

Fall, K. (2005). Disruption tolerant networking for heterogeneous ad-hoc networks. *Military Communications Conference, 2005. MILCOM 2005. IEEE*, Vol. 4(9017714):2195 – 2201. 10.1109/MILCOM.2005.1605995.

Farrell, S., Cahill, V., Geraghty, D., Humphreys, I., and McDonald, P. (2006). When tcp breaks: Delay- and disruption- tolerant networking. *IEEE Internet Computing Magazine*, Vol. 10(9008865):72 – 78. 10.1109/MIC.2006.91.

Gomes, E. d. N., Fernandes, R. M., Campos, C. A. V., and Viana, A. C. (2012). Um mecanismo de remoção de mensagens obsoletas para as redes tolerantes a atrasos e interrupções. *XI Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)*, pages 1–14. CSBC.

Hui, P., Crowcroft, J., and Yoneki, E. (2011). Bubble rap: social-based forwarding in delay tolerant networks. *Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, 10(11):1576–1589. IEEE.

INP, . (2007). *InterPlanetary Networking Special Interest Group (IPNSIG)*. Projeto Internet Interplanetária, http://www.ipnsig.org/.

IRTF, . (2007). *Internet Research Task Force*. Força-Tarefa de Pesquisa da Internet, http://www.irtf.org/.

Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L. S., and Rubenstein, D. (2002). Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. *In: ACM Sigplan Notices*, 37(10):96–107. ACM.

Keränen, A., Ott, J., and Kärkkäinen, T. (2009). The one simulator for dtn protocol evaluation. *In: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, page 55. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Krifa, A., Barakat, C., and Spyropoulos, T. (2008). Optimal buffer management policies for delay tolerant networks. *In: Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON'08. 5th Annual IEEE Communications Society Conference on*, pages 260–268. IEEE.

Krifa, A., Barakat, C., and Spyropoulos, T. (2012). Message drop and scheduling in dtns: Theory and practice. *Mobile Computing, IEEE Transactions on*, 11(9):1470–1483. IEEE.

Leela-Amornsin, L. and Esaki, H. (2010). Heuristic congestion control for message deletion in delay tolerant network. *In: Smart Spaces and Next Generation Wired/Wireless Networking*, pages 287–298. Springer Berlin Heidelberg.

Leguay, J. and Benbadis, F. (2009). CRAWDAD data set upmc/rollernet (v. 2009-02-02). Downloaded from http://crawdad.cs.dartmouth.edu/upmc/rollernet.

Li, Y., Qian, M., Jin, D., Su, L., and Zeng, L. (2009). Adaptive optimal buffer

management policies for realistic dtn. *In: Global Telecommunications Conference, 2009. GLOBECOM 2009*, pages 1–5. IEEE.

Moreira, W., Mendes, P., and Sargento, S. (2012a). Assessment model for opportunistic routing. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 10(3):1785–1790. IEEE.

Moreira, W., Mendes, P., and Sargento, S. (2012b). Opportunistic routing based on daily routines. *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pages 1–6. IEEE.

Mundur, P. and Seligman, M. (2008). Epidemic routing with immunity in delay tolerant networks. *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7.

Naves, J., Moraes, I., and Albuquerque, C. (2012). Lps and lrf: Efficient buffer management policies for delay and disruption tolerant networks. *In: Local Computer Networks (LCN), 2012 IEEE 37th Conference on*, pages 368–375. IEEE.

Pentland, A., Fletcher, R., and Hasson, A. (2004). Daknet: rethinking connectivity in developing nations. *Computer*, 37(1):78–83.

S. Kaveevivitchai, H. Ochiai, H. E. (2010a). Independent dtns message deletion mechanism for multi-copy routing scheme. *Proceedings of the Sixth Asian Internet Engineering Conference*, pages 48–55. ACM.

S. Kaveevivitchai, H. Ochiai, H. E. (2010b). Message deletion and mobility patterns for efficient message delivery in dtns. *Pervasive Computing and Com-*

munications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on, pages 760–763.

S. Rashid, Q. A. (2010). Efficient buffer management policy dla for dtn routing protocols under congestion. *(IJCNS) International Journal of Computer and Network Security*, Vol. 2(No. 9):118–121.

Silva, M. P., Ponciano, R. d. L., and Coelho, R. R. (2012). Modelagem de aplicação ead para viabilização de cursos extracurriculares em comunidades remotas utilizando uma estrutura de redes orientadas a plano de deslocamento contínuo. *VIII Simpósio Brasileiro de Sistemas de Informação (SBSI 2012)*, 5:9–16. SBSI.

Small, T. and Haas, Z. J. (2003). The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way). *In: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, 1:233–244. ACM.

Spyropoulos, T., Psounis, K., and Raghavendra, C. S. (2007). Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility. *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 79–85. PERCOMW '07.

T. Spyropoulos, K. Psounis, C. R. (2005). Spray and wait: An efficient routing scheme for intermittently connected mobile networks. *Proc. ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, pages 252–259. ACM.

T. Spyropoulos, K. Psounis, C. R. (2008). Efficient routing in intermittently

connected mobile networks: the multiple-copy case. *To appear in ACM/IEEE Transactions on Networking.*, 16(1):77–90. IEEE.

Voyiatzis, A. G. (2012). A survey of delay - disruption tolerant networking applications. *Journal of Internet engineering*, 5(1):331–343. IEEE.

Wang, X., Shu, Y., Jin, Z., and Chen, H. (2009a). Directional forward epidemic routing for disruption tolerant networks. *In: Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09. 5th International Conference on. IEEE*, pages 1–4. IEEE.

Wang, X., Shu, Y., Jin, Z., and Chen, H. (2009b). Weighted epidemic routing for disruption tolerant networks. *In: Communications and Networking in China, 2009. ChinaCOM 2009. Fourth International Conference on. IEEE, 2009*, 1:1–5. IEEE.

Yu, H. and Bian, H. (2011). Threshold-based message copies control in delay tolerant networks. *Journal of Networks*, 6(1):96–103.

Yuen, W. H. and Schulzrinne, H. (2010). Message replication and deletion in delay tolerant networks under hop-based and time-based ttl schemes. *Columbia University*, pages 1–12. Citeseer.

Z. J. Haas, T. S. (2006). A new networking model for biological applications of ad hoc sensor networks. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 14(1):27–40.

Apêndice A

O Simulador *The ONE*

O *The Opportunistic Network Environment - The ONE* é um simulador de redes oportunista, um dos mais usados na literatura em DTNs, que foi desenvolvido inicialmente como um gerador de mobilidade e que também pode fazer interface com outros simuladores existentes e os traços de mobilidade coletados de diferentes dispositivos do mundo real [Keränen et al. (2009)]. O gerador de mobilidade permite a construção de cenários passo a passo, adicionando realidade, utilizando mapas do mundo real, diferentes classes de nós (pedestres, veículos, ônibus), com diferentes horários, rotas, preferências de movimento, entre outros, e permitindo a importação de dados de mobilidade de outras simulações. Além disso, pode produzir uma variedade de relatórios: de movimento do nó, passagem de mensagens e estatísticas gerais.

Este simulador apresenta duas versões, uma para sistema operacional Linux e outra para Windows. Além disso, é divido em módulos o que se torna um facilitador nas alterações e manutenção. Os módulos que foram utilizados e que têm um grau de importância estão descritos abaixo:

- Core (Núcleo): Núcleo principal, onde estão as informações principais dos nós. Módulo responsável por gerenciar as informações dos nós, das mensagens, das conexões e do tempo de simulação;

- Input (Entrada): Tem a finalidade de gerenciar a entrada de dados para a simulação. Módulo responsável por repassar para a simulação, as informações de mobilidade externa, por exemplo os traços de mobilidade real e a geração de eventos externos.

- Report (Relatórios): Os resultados que são gerados na simulação estão implementados neste módulo.

- Routing (Roteamento): Responsável por gerenciar as informações de roteamento das mensagens na rede. É o local onde ficam as implementações de alguns protocolos de roteamento, como o Epidêmico, Spray And Focus e Spray and Wait.

O simulador possui dois modos de execução, modo gráfico e modo *batch*. No modo gráfico é possível visualizar, a troca de mensagens de cada nó, a movimentação, o tempo de simulação, o que cada nó está fazendo e é possível até mesmo observar anomalias no movimento dos nós. As desvantagens deste modo são: só permite a execução de uma rodada de simulação por vez, não permitindo mesclar os valores de parâmetros para várias rodadas de simulação. A outra desvantagem é que utiliza muito mais recurso da máquina, por ser executado em modo gráfico. No modo *batch* é possível combinar diferentes parâmetros para obtenção de várias rodadas de simulação. No arquivo de configuração básica do simulador, *default_settings.txt*, é possível informar ao simulador alguns parâmetros de

entrada, tais como:

- o número de nós da simulação;
- indicar arquivos externos de mobilidade;
- o protocolo de roteamento;
- arquivos externos de traços de mobilidade;
- os tipos de relatórios a serem gerados.

Além disso, na execução em modo *batch*, é possível informar quantas rodadas de simulação será executado, qual o arquivo de configuração será usado. Isto facilita a obtenção de várias rodadas de simulação, como normalmente avaliamos mais de um resultado, por parâmetro configurado, neste modo temos a facilidade de reduzir o tempo e o trabalho na execução de várias rodadas.

Apêndice B

Exemplos de arquivos de configuração usados na simulação

Exemplos de arquivos que foram utilizados para obtenção de resultados no simulador *The ONE*. Incluse, um arquivo de exemplo do autor da implementação do *Bubble Rap*, que foi usado para sua configuração.

Arquivos que foram usados para configurar o simulador:

1. Traço de mobilidade real UCL1.
2. Traço de mobilidade real Rollernet.
3. Trecho do arquivo criado pelo *script, createCreate.pl para o traço de mobilidade real UCL1*
4. Trecho do arquivo criado pelo *script, createCreate.pl para o traço de mobilidade real Rollernet*

1) - Exemplo de um trecho do arquivo gerado pelo *script createCreate.pl*, para o traço de mobilidade real - UCL1

```
107.8 C     M1    13    10    10000
137.3 C     M2    12    10    10000
207.0 C     M3    10    3     10000
241.1 C     M4    14    8     10000
291.5 C     M5    12    17    10000
343.3 C     M6    14    13    10000
359.7 C     M7    17    4     10000
383.0 C     M8    4     18    10000
395.6 C     M9    17    6     10000
398.6 C     M10   12    10    10000
429.9 C     M11   18    1     10000
430.1 C     M12   11    16    10000
448.9 C     M13   5     6     10000
461.2 C     M14   9     1     10000
499.4 C     M15   11    15    10000
571.3 C     M16   13    2     10000
590.8 C     M17   17    3     10000
678.4 C     M18   4     1     10000
698.8 C     M19   15    18    10000
743.2 C     M20   7     12    10000
770.4 C     M21   5     16    10000
808.6 C     M22   15    16    10000
839.5 C     M23   3     8     10000
844.6 C     M24   8     1     10000
972.9 C     M25   14    8     10000
996.2 C     M26   1     4     10000
1084.9      C     M27   11    4     10000
1093.4      C     M28   14    0     10000
1120.3      C     M29   6     7     10000
1121.2      C     M30   18    2     10000
1127.8      C     M31   17    4     10000
1199.6      C     M32   18    7     10000
1235.1      C     M33   18    0     10000
1259.1      C     M34   10    1     10000
1264.5      C     M35   7     2     10000
1379.1      C     M36   6     10    10000
1395.9      C     M37   10    9     10000
1527.2      C     M38   15    10    10000
1532.5      C     M39   10    4     10000
1620.2      C     M40   7     0     10000
1755.9      C     M41   4     2     10000
1787.9      C     M42   16    6     10000
1804.6      C     M43   11    15    10000
1833.3      C     M44   10    3     10000
1835.1      C     M45   8     7     10000
1843.2      C     M46   12    5     10000
1880.8      C     M47   9     3     10000
1937.9      C     M48   16    4     10000
1989.3      C     M49   12    6     10000
1989.4      C     M50   16    12    10000
2018.9      C     M51   8     15    10000
```

2) - Exemplo de um trecho do arquivo gerado pelo *script createCreate.pl*, para o traço de mobilidade real - Rollernet

```
5.5   C     M1    12    60    10000
5.6   C     M2    52    8     10000
10.2  C     M3    20    25    10000
18.0  C     M4    60    6     10000
47.7  C     M5    54    25    10000
50.8  C     M6    27    10    10000
77.7  C     M7    25    28    10000
90.0  C     M8    51    7     10000
91.0  C     M9    11    0     10000
93.2  C     M10   33    36    10000
103.5 C     M11   24    60    10000
122.6 C     M12   44    40    10000
128.1 C     M13   21    36    10000
143.3 C     M14   22    10    10000
149.8 C     M15   34    17    10000
151.9 C     M16   55    28    10000
155.0 C     M17   47    34    10000
161.6 C     M18   19    44    10000
174.1 C     M19   53    9     10000
215.9 C     M20   19    60    10000
234.2 C     M21   55    41    10000
241.6 C     M22   15    12    10000
305.6 C     M23   47    16    10000
306.2 C     M24   18    13    10000
306.7 C     M25   60    2     10000
312.5 C     M26   50    17    10000
312.9 C     M27   43    12    10000
324.0 C     M28   28    57    10000
326.1 C     M29   53    57    10000
329.9 C     M30   35    31    10000
345.7 C     M31   20    54    10000
350.2 C     M32   5     59    10000
350.7 C     M33   5     51    10000
354.7 C     M34   31    42    10000
355.0 C     M35   42    36    10000
364.2 C     M36   57    43    10000
387.2 C     M37   20    39    10000
393.1 C     M38   50    2     10000
416.1 C     M39   8     37    10000
439.0 C     M40   59    57    10000
439.3 C     M41   19    47    10000
451.2 C     M42   6     15    10000
490.1 C     M43   52    27    10000
493.2 C     M44   30    32    10000
497.1 C     M45   23    6     10000
502.2 C     M46   11    14    10000
512.8 C     M47   48    40    10000
525.7 C     M48   53    32    10000
528.1 C     M49   8     1     10000
536.6 C     M50   41    21    10000
552.0 C     M51   50    9     10000
553.3 C     M52   30    22    10000
```

3) - Descrição das configurações utilizadas na simulação com o traço de mobilidade real - UCL1

```
#
# Default settings for the simulation
#

## Scenario settings
Scenario.name = default_Ped_ReMO_%%Group.bufferSize%%_%%
MovementModel.rngSeed%%_%%Group.router
Scenario.simulateConnections = false
Scenario.updateInterval = 0.1
# 43200s == 12h
Scenario.endTime = 549019


## Interface-specific settings:
# type : which interface class the interface belongs to
# For different types, the sub-parameters are interface-
specific
# For SimpleBroadcastInterface, the parameters are:
# transmitSpeed : transmit speed of the interface (bytes per
second)
# transmitRange : range of the interface (meters)

# "Bluetooth" interface for all nodes
btInterface.type = SimpleBroadcastInterface
# Transmit speed of 2 Mbps = 250kBps
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 30

# High speed, long range, interface for group 4
highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 1000

# Define 6 different node groups
Scenario.nrofHostGroups = 1

## Group-specific settings:
# groupID : Group's identifier. Used as the prefix of host
names
# nrofHosts: number of hosts in the group
# movementModel: movement model of the hosts (valid class name
from movement package)
# waitTime: minimum and maximum wait times (seconds) after
reaching destination
# speed: minimum and maximum speeds (m/s) when moving on a
path
# bufferSize: size of the message buffer (bytes)
# router: router used to route messages (valid class name from
routing package)
# activeTimes: Time intervals when the nodes in the group are
active (start1, end1, start2, end2, ...)
# msgTtl : TTL (minutes) of the messages created by this host
```

```
group, default=infinite

## Group and movement model specific settings
# pois: Points Of Interest indexes and probabilities
(poiIndex1, poiProb1, poiIndex2, poiProb2, ... )
#       for ShortestPathMapBasedMovement
# okMaps : which map nodes are OK for the group (map file
indexes), default=all
#          for all MapBasedMovent models
# routeFile: route's file path - for MapRouteMovement
# routeType: route's type - for MapRouteMovement


# Common settings for all groups
Group.movementModel = StationaryMovement
Group.router = EpidemicRouter
Group.bufferSize = [40M;60M;80M;120M]
Group.listImmuneSize = 2k
#Group.listRemoSize = 2k
Group.waitTime = 0, 120
# All nodes have the bluetooth interface
Group.nrofInterfaces = 1
Group.interface1 = btInterface
# Walking speeds
Group.speed = 0.5, 1.5
Group.nodeLocation= 10,10

# Message TTL of 300 minutes (5 hours)
Group.msgTtl = 9151

Group.nrofHosts = 20

# group1 (pedestrians) specific settings
Group1.groupID = p

# group2 specific settings
#Group2.groupID = c
# cars can drive only on roads
#Group2.okMaps = 1
# 10-50 km/h
#Group2.speed = 2.7, 13.9

# another group of pedestrians
#Group3.groupID = w

# The Tram groups

#remoção dos grupos 4, 5 e 6;


## Message creation parameters
# How many event generators
Events.nrof = 2
# Class of the first event generator
Events1.class = MessageEventGenerator
# (following settings are specific for the
MessageEventGenerator class)
# Creation interval in seconds (one new message every 25 to 35
```

```
seconds)
Events1.interval = 60
# Message sizes (500kB - 1MB)
Events1.size = 10k
# range of message source/destination addresses
Events1.hosts = 0,19
# Message ID prefix
Events1.prefix = M
Events1.class = MessageEventGenerator
Events2.class = ExternalEventsQueue
Events2.filePath = mobi/ucl1.txt


## Movement model settings
# seed for movement models' pseudo random number generator
(default = 0)
MovementModel.rngSeed =
[2;63;342;564;777;841;900;8372;98092;18293;54]
# World's size for Movement Models without implicit size
(width, height; meters)
MovementModel.worldSize = 4500, 3400
# How long time to move hosts in the world before real
simulation
MovementModel.warmup = 1000

## Map based movement -movement model specific settings
#MapBasedMovement.nrofMapFiles = 4

#MapBasedMovement.mapFile1 = data/roads.wkt
#MapBasedMovement.mapFile2 = data/main_roads.wkt
#MapBasedMovement.mapFile3 = data/pedestrian_paths.wkt
#MapBasedMovement.mapFile4 = data/shops.wkt

## Reports - all report names have to be valid report classes

# how many reports to load
Report.nrofReports = 2
# length of the warm up period (simulated seconds)
Report.warmup = 0
# default directory of reports (can be overridden per Report
with output setting)
Report.reportDir = reports/SprayAndFocus/Infocom
# Report classes to load
Report.report1 = MessageStatsReport
Report.report2 = BufferOccupancyReport

## Default settings for some routers settings
ProphetRouter.secondsInTimeUnit = 30
SprayAndWaitRouter.nrofCopies = 6
SprayAndWaitRouter.binaryMode = true
SprayAndFocusRouter.nrofCopies = 6
SprayAndFocusRouter.transitivityTimerThreshold = 60

## Optimization settings -- these affect the speed of the
simulation
## see World class for details.
Optimization.cellSizeMult = 5
Optimization.randomizeUpdateOrder = true
```

```
## GUI settings

# GUI underlay image settings
GUI.UnderlayImage.fileName = data/helsinki_underlay.png
# Image offset in pixels (x, y)
GUI.UnderlayImage.offset = 64, 20
# Scaling factor for the image
GUI.UnderlayImage.scale = 4.75
# Image rotation (radians)
GUI.UnderlayImage.rotate = -0.015

# how many events to show in the log panel (default = 30)
GUI.EventLogPanel.nrofEvents = 100
# Regular Expression log filter (see Pattern-class from the
Java API for RE-matching details)
#GUI.EventLogPanel.REfilter = .*p[1-9]<->p[1-9]$
```

4) - Descrição das configurações utilizadas na simulação com o traço de mobilidade real - Rollernet

```
#
# Default settings for the simulation
#

## Scenario settings
Scenario.name = default_Ped_ReMO_%%Group.bufferSize%%_%%
Events2.filePath%%_%%Group.router
Scenario.simulateConnections = false
Scenario.updateInterval = 0.1
# 43200s == 12h
Scenario.endTime = 10800


## Interface-specific settings:
# type : which interface class the interface belongs to
# For different types, the sub-parameters are interface-
specific
# For SimpleBroadcastInterface, the parameters are:
# transmitSpeed : transmit speed of the interface (bytes per
second)
# transmitRange : range of the interface (meters)

# "Bluetooth" interface for all nodes
btInterface.type = SimpleBroadcastInterface
# Transmit speed of 2 Mbps = 250kBps
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 10

# High speed, long range, interface for group 4
highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 1000

# Define 6 different node groups
Scenario.nrofHostGroups = 1

## Group-specific settings:
# groupID : Group's identifier. Used as the prefix of host
names
# nrofHosts: number of hosts in the group
# movementModel: movement model of the hosts (valid class name
from movement package)
# waitTime: minimum and maximum wait times (seconds) after
reaching destination
# speed: minimum and maximum speeds (m/s) when moving on a
path
# bufferSize: size of the message buffer (bytes)
# router: router used to route messages (valid class name from
routing package)
# activeTimes: Time intervals when the nodes in the group are
active (start1, end1, start2, end2, ...)
# msgTtl : TTL (minutes) of the messages created by this host
```

```
group, default=infinite

## Group and movement model specific settings
# pois: Points Of Interest indexes and probabilities
(poiIndex1, poiProb1, poiIndex2, poiProb2, ... )
#       for ShortestPathMapBasedMovement
# okMaps : which map nodes are OK for the group (map file
indexes), default=all
#          for all MapBasedMovent models
# routeFile: route's file path - for MapRouteMovement
# routeType: route's type - for MapRouteMovement


# Common settings for all groups
Group.movementModel = StationaryMovement
Group.router = DecisionEngineRouter

DecisionEngineRouter.decisionEngine =
community.DistributedBubbleRap
DecisionEngineRouter.communityDetectAlg =
routing.community.KCliqueCommunityDetection
DecisionEngineRouter.K = 5
DecisionEngineRouter.familiarThreshold = 700
DecisionEngineRouter.centralityAlg =
routing.community.CWindowCentrality

Group.bufferSize = [1M;2M;5M;8M;10M]
Group.listImmuneSize = 2k
#Group.listRemoSize = 2k
Group.waitTime = 0, 120
# All nodes have the bluetooth interface
Group.nrofInterfaces = 1
Group.interface1 = btInterface
# Walking speeds
Group.speed = 0.5, 1.5
Group.nodeLocation= 10,10

# Message TTL of 300 minutes (5 hours)
Group.msgTtl = 1800

Group.nrofHosts = 62

# group1 (pedestrians) specific settings
Group1.groupID = p

# group2 specific settings
#Group2.groupID = c
# cars can drive only on roads
#Group2.okMaps = 1
# 10-50 km/h
#Group2.speed = 2.7, 13.9

# another group of pedestrians
#Group3.groupID = w

# The Tram groups

#remoção dos grupos 4, 5 e 6;
```

```
## Message creation parameters
# How many event generators
Events.nrof = 2
# Class of the first event generator
Events1.class = ExternalEventsQueue
# (following settings are specific for the
MessageEventGenerator class)
# Creation interval in seconds (one new message every 25 to 35
seconds)
#Events1.interval = 60
# Message sizes (500kB - 1MB)
#Events1.size = 10k
# range of message source/destination addresses
#Events1.hosts = 0,61
# Message ID prefix
#Events1.prefix = M
#Events1.class = MessageEventGenerator
Events2.class = ExternalEventsQueue
Events1.filePath = mobi/imote_rollernet_final.txt
Events2.filePath =
[message/Rollernet/message10kRollernet1.txt;message/Rollernet/
message10kRollernet2.txt;message/Rollernet/message10kRollernet
3.txt;message/Rollernet/message10kRollernet4.txt;message/Rolle
rnet/message10kRollernet5.txt;message/Rollernet/message10kRoll
ernet6.txt]



## Movement model settings
# seed for movement models' pseudo random number generator
(default = 0)
#MovementModel.rngSeed =
[2;63;342;564;777;841;900;8372;98092;18293;54]
# World's size for Movement Models without implicit size
(width, height; meters)
#MovementModel.worldSize = 4500, 3400
# How long time to move hosts in the world before real
simulation
#MovementModel.warmup = 1000

## Map based movement -movement model specific settings
#MapBasedMovement.nrofMapFiles = 4

#MapBasedMovement.mapFile1 = data/roads.wkt
#MapBasedMovement.mapFile2 = data/main_roads.wkt
#MapBasedMovement.mapFile3 = data/pedestrian_paths.wkt
#MapBasedMovement.mapFile4 = data/shops.wkt

## Reports - all report names have to be valid report classes

# how many reports to load
Report.nrofReports = 2
# length of the warm up period (simulated seconds)
Report.warmup = 0
# default directory of reports (can be overridden per Report
with output setting)
```

```
Report.reportDir = reports/Spray/Rollernet
# Report classes to load
Report.report1 = MessageStatsReport
Report.report2 = BufferOccupancyReport

## Default settings for some routers settings
ProphetRouter.secondsInTimeUnit = 30
SprayAndWaitRouter.nrofCopies = 6
SprayAndWaitRouter.binaryMode = true
SprayAndFocusRouter.nrofCopies = 6
SprayAndFocusRouter.transitivityTimerThreshold = 60

## Optimization settings -- these affect the speed of the
simulation
## see World class for details.
Optimization.cellSizeMult = 5
Optimization.randomizeUpdateOrder = true


## GUI settings

# GUI underlay image settings
GUI.UnderlayImage.fileName = data/helsinki_underlay.png
# Image offset in pixels (x, y)
GUI.UnderlayImage.offset = 64, 20
# Scaling factor for the image
GUI.UnderlayImage.scale = 4.75
# Image rotation (radians)
GUI.UnderlayImage.rotate = -0.015

# how many events to show in the log panel (default = 30)
GUI.EventLogPanel.nrofEvents = 100
# Regular Expression log filter (see Pattern-class from the
Java API for RE-matching details)
#GUI.EventLogPanel.REfilter = .*p[1-9]<->p[1-9]$
```