DAVI RIBEIRO MILITANI

UM ALGORITMO PARA AUTO-CONFIGURAÇÃO DO PROTOCOLO DE ROTEAMENTO OLSR

LAVRAS – MG

2014

DAVI RIBEIRO MILITANI

UM ALGORITMO PARA AUTO-CONFIGURAÇÃO DO PROTOCOLO DE ROTEAMENTO OLSR

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação

Orientador
Prof. Neumar Malheiros

LAVRAS – MG
2014

DAVI RIBEIRO MILITANI

UM ALGORITMO PARA AUTO-CONFIGURAÇÃO DO PROTOCOLO DE ROTEAMENTO OLSR

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação

APROVADA em 14 de Fevereiro de 2014.

Prof. Hermes Pimenta de Moraes Júnior UFLA

Prof. Tales Heimfartha UFLA

Prof. Neumar Malheiros

(Orientador)

LAVRAS – MG

2014

Dedico ao meu avô materno José Ribeiro, e ao meu orientador Neumar.

AGRADECIMENTOS

Agradeço primeiramente a Deus. Agradeço a minha namorada Ana Carolina pelo apoio, carinho e amor. Agradeço ao meu pai pelo apoio e o exemplo de caráter, a minha mãe pelo amor e pelo apoio, e ao meu irmão por seu incentivo. Agradeço meu orientador Neumar pela paciência e dedicação em me ajudar.

RESUMO

O número de usuários de redes sem fio vem crescendo muito. É comum encontrar redes sem fio em muitas residências e em diversos espaços públicos, como universidades, praças de alimentação, entre outros. Ultimamente, a arquitetura de redes Ad hoc vem sendo utilizada para prover acesso sem fio. Porém, a crescente demanda por mobilidade e aplicações avançadas requer mais esforços no desenvolvimento de redes sem fio de alto desempenho. Este trabalho visa desenvolver um algoritmo para reconfiguração dinâmica do protocolo de roteamento *Optimized Link State Routing* (OLSR), com o objetivo de evitar degradação de desempenho da rede resultante da mobilidade dos dispositivos. Em particular, o objetivo do projeto é a reconfiguração do intervalo de envio de mensagens de controle do OLSR, afim de diminuir a sobrecarga de controle e aumentar o desempenho da rede.

Palavras-Chave: Redes adhoc, protocolos de roteamento, auto-configuração, OLSR

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

Cada vez mais, as pessoas utilizam redes sem fio com seus celulares, tablets, notebooks e outros dispositivos portáteis. Atualmente, encontrar uma rede sem fio é algo comum. Elas estão presentes em diversos espaços públicos, como universidades, praças de alimentação, entre outros. Entretanto, a demanda por mobilidade e por aplicações avançadas, como videoconferência, ainda exigem mais esforços no desenvolvimento de redes de acesso sem fio mais eficientes.

Existem dois tipos de arquiteturas de redes sem fio: redes infraestruturadas e redes Ad hoc (KUROSE; ROSS, 2010). As redes infraestruturadas necessitam de uma estação base, por onde passa toda a comunicação feita na rede. Elas são as mais utilizadas atualmente, porém, o custo de instalação e operação dessas redes é alto. Por essa razão, a comunidade de pesquisa vem tentando desenvolver melhorias para redes Ad hoc; elas dispensam a necessidade de estações base, dessa forma, não existe um ponto único de falha, tendo assim, a vantagem de ser tolerante a falhas. Mas, a implementação de redes Ad hoc eficientes tem um grau de dificuldade maior. Estas redes não tem infraestrutura, assim, os nós da rede necessitam comunicar entre si. No entanto, eles podem se mover arbitrariamente, assim, a conectividade entre os nós muda constantemente. Dessa forma, o protocolo de roteamento para esse tipo de rede requer uma constante troca de informações entre os nós afim de atualizar seu conhecimento sobre a topologia da rede.

Um protocolo muito utilizado em redes Ad hoc é o *Optimized Link State Routing Protocol* (OLSR) (CLAUSEN; JACQUET, 2003). Para que cada nó da rede possa descobrir quem são os seus vizinho, o OLSR provê mensagens de controle chamadas *Hello*. Estas mensagens são trocadas em um intervalo definido pelo o parâmetro *Hello Interval*. Porém, as mensagens acabam aumentando o tráfego de controle, e diminuindo os recursos da rede que poderiam ser utilizados para o tráfego de dados. Portanto, existe a necessidade de protocolos de roteamento

mais eficientes, que minimizem o problema de excesso de tráfego de mensagens de controle. O volume de tráfego de controle é denominado *overhead*.

Foram propostas algumas soluções para minimizar o *overhead* do protocolo OLSR. No trabalho apresentado em (BENZAID; MINET; AGHA, 2002), foi proposta uma estratégia que define dois modos de operação para o OLSR. Um modo padrão no qual não há nenhuma alteração na operação normal do OLSR. No segundo modo, chamado *Fast-OLSR*, o algoritmo limita o número de nós da rede que retrasmitem as mensagens de controle e a frequência com que estas mensagens são trocadas. O algoritmo intercala entre o modo normal e o modo *Fast-OLSR* de acordo com o grau de mobilidade da rede. Porém não foi descrita de forma clara como é medida essa mobilidade. Em (STANZE; ZITTERBART; KOCH, 2006), foi proposta uma técnica para medir o grau de mobilidade da rede. O indicador do grau de mobilidade de rede é chamado *Relative Velocity Indicator* (MARVIN). De acordo com o valor do MARVIN, o algoritmo modifica o intervalo de envio das mensagens de controle. Uma das limitações dessa técnica é que os valores do intervalo são pré-estabelecidos através de simulações. Neste caso um dado valor de MARVIN corresponde a um valor de intervalo. Esse mapeamento pré-estabelicido (estático) é uma limitação para o uso da técnica. Outra técnica para medir o grau de mobilidade foi proposta em (HERNANDEZ-CONS; KASAHARA; TAKAHASHI, 2010). Neste trabalho, o intervalo de tempo de envio das mensagens de controle é auto-configurado de acordo com a LCR (*Link Change Rate*), que é a taxa de mobilidade da rede. Dessa forma, o intervalo se ajusta de acordo com o valor da LCR. Os resultados apresentados no trabalho mostram que o algoritmo teve um menor *overhead* que o OLSR padrão, porém a taxa de entrega de pacotes teve um resultado pior. Ou seja, o algoritmo não contribui muito para o melhoramento do desempenho das redes Ad hoc, pois um dos motivos para diminuir *overhead* é que sobre mais recursos da rede para envio de mensagens de dados, aumentando assim a taxa de entrega de pacotes.

O problema de *overhead* gerado pelas mensagens de controle do protocolo OLSR carece de solução eficaz. Neste contexto o objetivo deste trabalho é propor um algoritmo que auto-configure o *Hello Interval* de acordo com as necessidades reais da rede, afim de diminuir o *overhead* do protocolo OLSR. Outro objetivo é que o algoritmo proposto possa ser implementado nas atuais redes Ad hoc, viabilizando o desenvolvimento de aplicações avançadas. Além disso, melhorando a qualidade das redes Ad hoc, a solução proposta incentiva o uso das mesmas. Contribuindo para diminuir o custo da implantação de rede de acesso sem fio, pois as redes Ad hoc dispensam estações base.

Este texto está organizado como explicado a seguir. No capítulo 2, será apresentado um referencial teórico sobre os conceitos de redes sem fio, de protocolos de roteamento e dos tipos de protocolos. O protocolo OLSR, será abordado de forma mais abrangente, visando explicar os parâmetros de configuração, principalmente o Hello Interval, o funcionamento e as particularidades do protocolo OLSR. No capítulo 3, serão apresentados os trabalhos relacionados, ou seja, as soluções propostas para a auto-configuração do protocolo OLSR. No capítulo 4, será abordada a metodologia usada para o desenvolvimento do projeto. No capítulo 5, será abordada a proposta de solução para este projeto. No capítulo 6, será abordada a avaliação do desempenho do algoritmo proposto, em relação ao protocolo OLSR padrão. No capítulo 7, apresenta-se a conclusão deste trabalho.

2 ROTEAMENTO EM REDES AD HOC

Uma rede de computadores é formada por dispositivos/sistemas computacionais que se comunicam entre si, compartilhando dados e recursos. Cada dispositivo da rede é denominado nó ou elemento. Para que dois nós da rede se comuniquem, é necessário estabelecer um caminho entre eles. O roteamento, determina o caminho ou rota tomado pelos pacotes ao serem transmitidos do remetente ao destinatário. Um protocolo de roteamento, é formado por: (i) Um conjunto de regras que definem a forma e a ordem de como as mensagens de controle são trocadas entre os nós, e como é feita a descoberta da topologia da rede; (ii) Um algoritmo de roteamento que calcula caminho ótimo entre a fonte e o destino (KUROSE; ROSS, 2010).

Cada nó de uma rede de computadores possui uma tabela de roteamento. A tabela de roteamento, contém as informações sobre a topologia da rede, ou seja, dado um pacote com um endereço de destino, a tabela consta qual caminho esse pacote deve seguir, ou pelo menos para qual nó vizinho ele deve ser encaminhado. A tabela é construída através da troca de mensagens de controle entre os nós. Estas mensagens contêm informações sobre a topologia. A forma como estas mensagens são disseminadas, é definida pelo protocolo de roteamento. Existem vários protocolos de roteamento, porém, todos tem como principal objetivo determinar um caminho com menor custo, de acordo com as métricas específicas que serão abordadas mais a frente.

Os dois principais tipos de protocolos utilizados são, o *Link State* e o *Distance Vector* (KUROSE; ROSS, 2010). No algoritmo *Link State*, toda topologia da rede é conhecida. Quando um nó reconhece uma modificação na rede ele envia para todos os outros nós sem distinção sua tabela de vizinhança atualizada com a modificação. Dessa forma, todos os nós têm conhecimento geral da rede. No algoritmo *Distance Vector*, quando uma mudança é identificada, cada nó passa toda sua tabela de roteamento, mas somente para seus vizinhos. Cada vizinho atualiza

sua tabela e, por sua vez, repassa a seus vizinhos. Isto é feito sucessivamente, até que todos os nós atualizem suas informações.

As redes sem fio Ad hoc não necessitam de infrasestrutura, ou seja, não existe um ponto de acesso central em que todos os dados trocados na rede passam para ir de uma origem ao um destino. Dessa forma, um nó da rede pode ter que comunicar com vários outros para alcançar o nó de destino.

Como muitos dos nós em redes Ad hoc são móveis o caminho para alcançar o destino pode sofrer mudanças. Eles podem se deslocar ou desconectar, mudando assim a topologia da rede. A figura 2.1 ilustra essa situação. Quando o nó 1 envia um pacote para o nó 2 através de um caminho conhecido que passa pelas arestas A, B e C, pode acontecer de um nó que faz parte desse caminho desconectar-se, impossibilitando a passagem do pacote por esse caminho. Ao acontecer isso, a rede tem que atualizar as informações de roteamento encontrando se possível um novo caminho de 1 até 2, no caso ilustrado, o caminho alternativo será o que passa pelas aresta D, E e F.

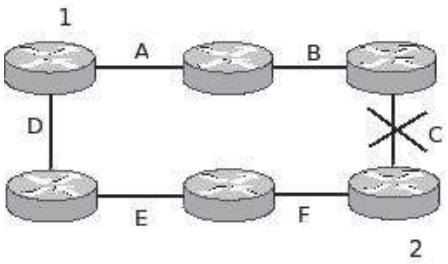

Figura 2.1: Perda de uma enlace.

Duas principais classes de protocolos de roteamento utilizados em redes Ad hoc são os pró-ativos e os reativos. Nos pró-ativos, as informações sobre a tabela de roteamento são trocadas entre os nós, de forma constante em intervalos fixos. Portanto, independente do tráfego de dados na rede, os nós sempre estão trocando informações. Dessa forma, quando um nó resolver enviar um pacote, a rota até o destino já é conhecida.

Os protocolos reativos trocam informações, somente quando um nó necessita enviar um pacote. Dessa forma, o tráfego de controle na rede diminui quando os nós estão em estado ocioso. Porém, quando um nó envia um pacote para um novo destino, é necessário trocar informação para conhecer toda a rota, o que acaba gerando um atraso antes que o nó possa começar a enviar o primeiro pacote.

2.1 Métricas

Uma métrica é utilizada para construir a tabela de roteamento de cada nó. Sua escolha tem como objetivo promover o melhor caminho entre cada nó de acordo com a situação da rede. A escolha têm influência direta no desempenho da rede. Existem várias métricas, duas delas são: *número de saltos* e a *Expected Transmission Count* (ETX) (COUTO *et al.*, 2005).

A métrica *número de saltos* atribui a todos enlaces da rede o mesmo peso, dessa forma, o melhor caminho entre dois nós é o caminho com menor número de enlaces. Porém, a distância física entre os nós, a interferência do ambiente, e o volume de tráfego, tem influência no desempenho de cada enlace. Dessa forma, considerar todos enlaces iguais pode ocasionar a escolha de rotas com baixo desempenho. Por exemplo, quando um nó elege um caminho de menor custo até o destino, todo tráfego passa por esse caminho, podendo sobrecarregar o mesmo. Um caminho alternativo com custo maior, porém, com menor tráfego, caso existisse, poderia ser usado também, dividindo assim o tráfego. A figura 2.2 ilustra essa situação. O nó 1 tem dois fluxo de dados para o nó 3. O caminho de menor custo é o que passa pelas arestas A e B pois tem 2 saltos. Segundo essa métrica, os dois fluxos passariam pelo caminho citando anteriormente, possivelmente congestionando o roteador 2. Porém, o caminho alternativo que passa pelas arestas D, E e C, mesmo sendo mais longo poderia ser usado, balanceando o tráfego de dados nos enlaces, evitando assim perdas de pacotes.

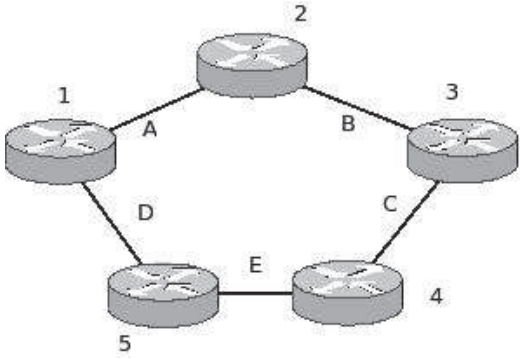

Figura 2.2: Métrica Número de saltos

A métrica *Expected Transmission Count* (ETX) estima o número de transmissões necessárias para entregar um pacote entre dois nós vizinhos ligados por um enlace. Utiliza-se a seguinte fórmula para calcular a estimativa: $ETX = \frac{1}{dfXdr}$

Onde df é a taxa de entrega de pacotes de ida, e dr é a taxa de recebimento de confirmação, os *ACKs*. Para se calcular o peso de uma rota soma-se os valores dos ETXs de todos os enlaces pertencentes a essa rota. Por exemplo, para se calcular o valor da rota que vai do nó A até o nó C passando pelo nó B. Soma-se o valor do ETX do enlace que liga o nó A até B com o ETX do enlace que liga o nó B até o nó C. A métrica escolhe rotas em que o ETX é menor, ou seja, rotas que tem um menor número de retransmissões.

2.2 Protocolos de Roteamento em rede Ad hoc

Existem vários protocolos de roteamento específicos para redes Ad hoc. Dentre eles serão abordados o *Destination Sequenced Distance-Vector* (DSDV), *Ad hoc on-demand Distance Vector* (AODV), *Dynamic Source Routing* (DSR) e o *Optimized Link State Routing* (OLSR).

Destination Sequenced Distance-Vector

O DSDV (PERKINS; BHAGWAT, 1994) é um protocolo de roteamento da classe dos pró-ativos e do tipo *Distance Vector*. As informações trocadas periodicamente entre os nós sobre a tabela de roteamento contém um campo chamado número de sequência. Este campo foi criado com intuito de manter a sincronização das informações. Desta forma, quando um nó recebe informações sobre a topologia da rede de seus vizinhos, ele compara o número de sequência recebido com o que possui. Caso o número recebido seja menor a informação então é descartada, evitando assim informações defasadas que podem gerar laços. Se o número for maior então as informações são atualizadas juntamente com o número de sequência novo.

Com o intuito de diminuir o tráfego de controle na rede o DSDV possui uma maneira diferente de atualizar os dados de cada nó. Quando uma mudança ocorre na rede, um pacote é enviado contendo apenas essa modificação específica, e não toda tabela de roteamento. Desta forma, pacotes menores são enviados diminuindo assim o tráfego na rede. Quando a mudança na rede é grande, de forma que essa informação não consiga mais ser armazenada neste pacote, então, toda a tabela de roteamento é enviada.

Ad hoc on-demand Distance Vector

O AODV (PERKINS; ROYER, 1999) é um protocolo do tipo *Distance Vector*. Ele é da classe reativo, dessa forma, quando um nó deseja enviar um pacote para um destino que não consta em sua tabela de roteamento, todo o processo de descoberta sobre esse caminho é feito. Primeiramente, um pacote chamado *route request* é enviado pelo nó origem em *broadcast*. Quando o nó destino recebe o *route request*, ele responde com um pacote chamado *route replay*. Ao receber *route replay* o nó origem insere em sua tabela de roteamento o caminho até o destino, podendo agora enviar o pacote.

Após o nó origem receber a resposta é possível que outros nós ainda estejam recebendo o pacote *route request* enviado por ele. Este tráfego na rede é desnecessário pois o nó origem já possui o caminho até destino. Para que isto não ocorra, o pacote enviado tem um campo com o número máximo de saltos, ou seja, a quantidade máxima de nós pelos quais o pacote pode passar. O pacote é enviado primeiramente para os nós mais próximos, isto é feito limitando-se a quantidade de saltos. Desta forma, o pacote não passa para os nós com número maior de saltos do que o limite. Caso nenhum nó responda, é enviado novamente o pacote em broadcast porém com um limite maior para o número de saltos.

O AODV é um protocolo feito para se adaptar a cenários nos quais ocorrem muitas mudanças na topologia da rede, como é o caso de redes Ad hoc. Nessas redes os nós se desconectam da rede com maior frequência. Para evitar problemas, como, por exemplo, a quebra de um caminho, o AODV faz manutenção de rotas. Para fazer isto, pacotes são enviados periodicamente para se verificar a modificação de uma rota. Chamados de *Hello*, estes pacotes são enviados por todos os nós. Cada nó espera dos outros nós, um pacote *Hello*. Caso um pacote esperado não chegue a um nó, imediatamente esse nó envia um pacote que informa o erro, para todos os nós que forem necessários. Quando os nós da rede recebem esse pacote informando o erro, eles atualizam sua tabela roteamento. Caso, por exemplo, ocorra uma ruptura de um nó, as tabelas serão atualizadas removendo o caminho até esse nó e modificando qualquer caminho que passe por esse nó. Evitando assim que um pacote seja enviado para um caminho que não existe mais.

Dynamic Source Routing

O DSR (JOHNSON; MALTZ, 1996) é um protocolo do tipo *Link State*, assim como AODV, ele é da classe reativo. A diferença entre eles está em que no DSR cada nó possui informação da rota inteira, e não somente até seus vizinhos. Desta forma, o DSR tem melhor desempenho quando os nós da rede têm menor mobi-

lidade. Pois, nesta situação os nós atualizam a tabela de roteamento em menor frequência. Caso contrário, as tabelas deveriam ser atualizadas constantemente o que geraria um grande tráfego na rede, pois seria necessário a troca de informações com todos os nós, e não somente com os vizinhos.

O DSR utiliza o mecanismo de roteamento *source routing*, em que o remetente conhece a rota completamente, nó a nó, até o destino. Os pacotes do protocolo DSR, carregam no cabeçalho a rota completa. Quando um nó precisa enviar um pacote para o destino para qual ele ainda não conhece a rota, ele utiliza o processo de descoberta de rota. Primeiramente ele envia um pacote *Router Request* (RREQ). Cada nó que recebe um RREQ, reenvia em broadcast até que o RREQ chegue ao destino. Cada nó responde o RREQ com o *router reply* (RREP) que é enviado ao nó de origem. O nó de origem recebe o RREQ e atualiza sua tabela de roteamento. Se algum enlace deste caminho que foi adicionado a tabela de roteamento cair, o nó de origem recebe um *router error* (RERR), e remove todas as rotas que passam pelo enlace que caiu, e, caso seja necessário, um novo processo de descobrimento de rota é realizada.

2.3 Optimized Link State Routing

O OLSR é um protocolo que foi desenvolvido para redes Ad hoc. Ele é da classe pró-ativo, dessa forma, as mensagens de controle são trocadas de forma constante independente da ocorrência de tráfego de dados. A RFC 3626 (CLAUSEN; JACQUET, 2003) divide o OLSR em funcionalidade e núcleo. O núcleo são as requisições necessárias para o protocolo operar, já a funcionalidade consiste em um conjunto de funções auxiliares.

O protocolo OLSR mantém localmente em cada nó uma variedade de bases de informação. Esses Repositórios de Informações são atualizados na medida em que as mensagens de controle são recebidas, e a informação armazenada é usada para gerar essas mensagens. Os principais repositórios do OLSR são:

Conjunto de enlaces, conjunto de vizinhos e o conjunto dos nós *MultiPoint Relay* (MPR) que são os nós que podem retransmitir as mensagens de um dado nó.

O Formato de Pacote do OLSR

A RFC 3626 define como é feita a comunicação no OLSR e a formatação dos pacotes. Para facilitar a extensão do protocolo, o OLSR utiliza um formato unificado de pacote. A figura 2.3 retirada de (CLAUSEN; JACQUET, 2003) ilustra o modelo de qualquer pacote do protocolo OLSR.

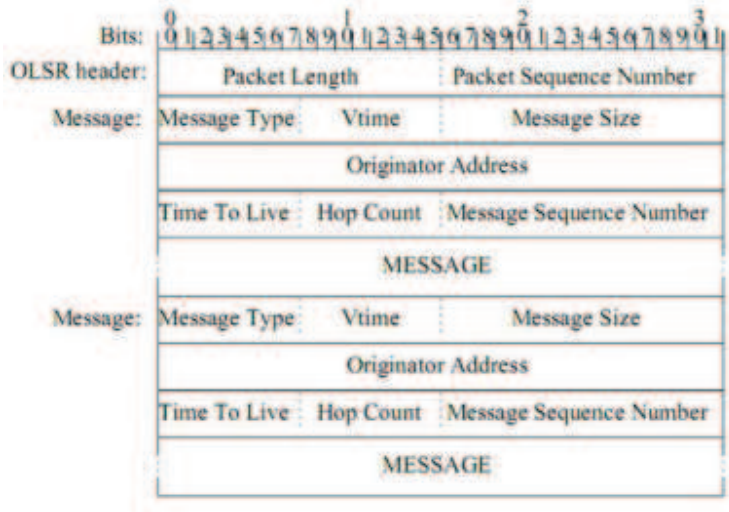

Figura 2.3: Formato do pacote OLSR.

O cabeçalho do pacote contém os campos: (i) *Packet Length*, que representa o comprimento em bytes do pacote. (ii) *Packet Sequence Number* é um contador que toda vez que o pacote é enviado por algum nó deve ser incrementado em 1.

A mensagem contém 8 campos, são eles:

- *Message Type* indica o tipo de mensagem. O tipo é definido de acordo com o valor que variar entre 0 e 127. O protocolo OLSR reserva alguns valores, entre eles o valor 1 que é utilizado para a mensagem *Hello*.

- *Vtime* serve para validadar as informações da mensagem.
- *Message Size* é o campo que possui a contagem total em bytes do tamanho da mensagem.
- *Originator Address* indica IP do remetente da mensagem.
- *Time To Live* é o campo que indica o número máximo de saltos que a mensagem pode percorrer.
- *Hop Count* indica por quantos nós o pacote já passou.
- *Message Sequence Number* possuim um número sequencial único da mensagem, para evitar mensagens repetidas.
- *Message* é a mensagem que o nó enviou.

A RFC 3626 especifica três tipos de mensagens para o OLSR, são elas: (i) A mensagem HELLO: Responsável pela detecção de enlaces, vizinhos e controle MPR. (ii) A mensagem TC: Responsável pela declaração da topologia de rede. (iii) A mensagem MID: Responsável pela declaração de múltiplas interfaces em um determinado nó.

A mensagem de controle *Hello* é responsável pela detecção de enlaces, detecção de vizinhos e controle MPR. As informações sobre todos os enlaces, quais MPRs um nó elegeu e quais são os vizinhos conhecidos, são trocadas através das mensagens de controle *Hello*. Cada nó da rede possui informações sobre qual é seu conjunto de vizinhos, de enlace e MPRs. As mensagens Hello são construídas com base nessas informações e enviadas aos nós vizinhos.

Um dos parâmetros de configuração do OLSR é o *Hello Interval*, ele é responsável pelo intervalo de envio das mensagem de controle *Hello*. Por padrão o valor do parâmetro é 2 segundos. Sendo assim, a cada dois segundos o nós da rede trocam mensagens *Hello* para a descoberta de vizinhança.

Informação da topologia

Considera-se neste trabalho que quando dois nós estão no mesmo raio de comunicação, existe um enlace entre eles. O conjunto de enlaces de um nó é atualizado através das mensagens de controle. O nó troca mensagens Hello, e detecta o enlace entre ele e seus vizinhos. Um enlance pode ser simétrico ou assimétricos. Em um enlace assimétrico o nó recebe mensagens *Hello* do seu vizinho, porém ele não recebe confirmação de que o seu vizinho está recebendo as suas mensagens. E um enlance simétricos existe a comunicação bidirecional, todos dois enviam e recebem confirmação. A figura 2.4 ilustra como ocorre a detecção de enlace. As trocas de mensagens são feitas como descrito em (CLAUSEN; JACQUET, 2003).

- Passo 1: um dado nó X envia uma mensagem Hello vazia. Um dado nó Y recebe a mensagem e guarda a informação do nó X como assimétrico.
- Passo 2: o nó Y manda um mensagem Hello constando que X é seu vizinho assimétrico.
- Passo 3: quando o nó X recebe a mensagem enviada por Y, ele registra que Y é seu vizinho simétrico e envia uma mensagem de volta para Y.
- Passo 4: quando Y recebe essa informação ele registra X como vizinho simétrico e envia uma mensagem de volta informando que o nó x é seu simétrico

O conjunto de vizinhos de um nó é atualizado através das mensagens de controle *Hello*. O conjunto de enlaces está ligado ao conjunto de vizinhos, pois um nó é vizinho do outro se e somente se existir um enlace entre dois. Para detecção de vizinhos de dois saltos existem um campo na mensagem Hello chamado *Neighbor Interface Address*. Quando um dado nó A envia uma mensagem para um dado vizinho simétrico B, o vizinho adiciona no campo *Neighbor Interface Address*

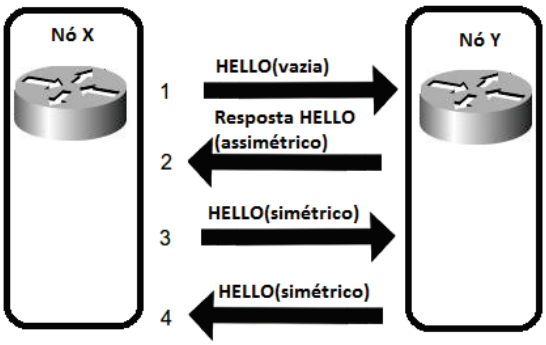

Figura 2.4: Detecção de Enlace no protocolo OLSR

a existência do nó A, assim quando o nó B enviar uma mensagem para algum vizinho, na mensagem constara a existência do nó A.

MultiPoint Relay

OLSR utiliza uma estratégia chamado *MultiPoint Relay* (MPR), que tem função de eliminar mensagens de controle redundantes na rede. Em uma rede Ad hoc que não utiliza o conceito MPR, quando um nó recebe uma mensagem de controle, ele retransmite essa mensagem para todos os vizinhos, ou até mesmo para todos os nós da rede. Porém, pode acontecer que um vizinho já tenha recebido essa mensagem de outro vizinho, ou do próprio nó que enviou a mensagem, gerando assim mensagens redundantes.

Utilizando o conceito *MultiPoint Relay*, são selecionados dentre os nós da rede quais podem retransmitir as mensagens de controle, estes nós são chamados de MPR. Dessa forma, o número de nós que retransmitirá os pacotes será limitado. Cada nó da rede escolhe seu conjunto de MPRs satisfazendo as seguintes regras: (i) O conjunto de MPRs de um nó é um subconjunto de vizinhos simétricos de um salto. (ii) O conjunto de MPRs pode alcançar com um salto os vizinhos de dois

saltos do nó. Com o conjunto de MPRs já definido quando o nó envia as mensagens de controle, todos os seus vizinhos de um salto recebem. Porém, somente os vizinhos do conjunto MPRs podem retransmitir, reduzindo assim o número de mensagens redundantes na rede. Quanto menor for o conjunto de MPRs de um nó menor será o tráfego de controle (CLAUSEN; JACQUET, 2003). Na figura 2.5, é ilustrada uma topologia com 11 nós, no qual o nó F deseja enviar mensagens de controle. Neste caso, o conjunto dos nós MPR para o nó F seriam aqueles identificados com a letra M.

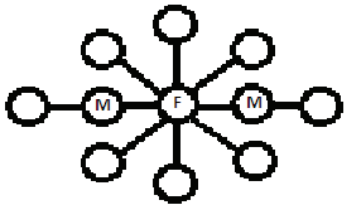

Figura 2.5: O conjuto de nós MPR para o nó F.

Os nós da rede mantêm informações sobre o seu conjunto de MPRs. Cada nó constrói independentemente seu próprio conjunto de MPRs. Essa contrução é feita utilizando as mensagens *Hello*. Outra informação importante fornecida pelas mensagens Hello é o conjunto de nós que escolheram um dado nó como seu MPR. A cada mensagem *Hello* recebida de um vizinho, o nó verifica quais são os MPRs do seu vizinho. Se o enderenço do próprio nó que recebeu a mensagem, estiver registrado como MPR do seu vizinho. O nó registra essa informação, pois toda vez que ele receber um pacote desse vizinho, ele saberá que é um MPR desse vizinho, e então repassara o pacote, caso o destino não seja ele mesmo.

3 TRABALHOS RELACIONADOS

A estratégia necessária para minimizar o problema de excesso de tráfego de controle possui duas etapas: (i) Análise da rede; (ii) Ajuste do protocolo de acordo com a situação analisada. Na literatura já existem algumas soluções que utilizam diferentes estratégias para minimizar o *overhead* do protocolo OLSR.

No trabalho proposto em (BENZAID; MINET; AGHA, 2002), o autor desenvolveu uma extensão do *OLSR* chamado de *fast-OLSR*. Ele foi feito baseado no *OLSR*, porém, a descoberta de vizinhança foi adaptada para lidar melhor com a alta mobilidade. O *fast-OLSR* pode ser utilizado em uma rede em conjunto com nós que utilizam o *OLSR* padrão.

O *fast-OLSR* utiliza três mecanismos para se adaptar a mudanças nas condições da rede, são eles:

1. *Switching to the Fast-moving / Default mode*: Quando um nó detecta grande mobilidade na vizinhaça, ele entra em um modo chamado *Fast-moving* e começa a enviar *Fast-hellos*. Essas mensagens são parecidas com a *Hello* padrão, porém, tem tamanho reduzido, pois o número de vizinho é menor. Ao detectar baixa mobilidade, o algoritmo retorna para o modo *default*.

2. *Establishing Fast links*: Um nó X no modo *Fast-moving* envia *Fast-hellos*. As mensagens *Fast-hellos* são utilizadas para estabilizar os *Fast-links*. Quando os nós que estão em modo *default* recebem estas mensagens, eles respondem com uma outra mensagem *Fast-hello*. Entre as respostas recebidas o nó X seleciona um número pequeno de nós que serão os seus MPRs. As próximas mensagens *Fast-hellos* do nó X são atualizados com os novos MPRs. Os nós declarados MPRs trasmitiram mensagens *Topology Control* a todos os nós da rede, declarando que são MPRs de X. Apenas os nós no modo padrão podem ser selecionados como MPRs. Os nós em *Fast-moving* não conhe-

cem seus vizinhos de dois saltos, a comunicação com eles é feita através dos MPRs.

3. *Refreshing Fast links and Detecting broken links*: Um nó X em modo *Fast-moving* envia mensagens *Fast-hellos* contendo uma tabela com a lista dos seus MPRs. Quando um MPRs recebe esta mensagem, ele responde com uma *Fast-hellos* sem nenhum endereço. O *Fast-hello* vazio contribui para diminuir o excesso de tráfego de controle e permite ao nó X saber que ele sempre pode ser alcançado por esses MPRs que responderam. Se um MPR de X não recebeu o *Fast-hello* de X, então, ele envia uma mensagem *Topology Control* para informar a todos os nós da rede que ele não é mais um MPR para X. Por meio de *Fast-hellos*, o *Fast-OLSR* permite detectar rapidamente um link quebrado. Desta forma, uma nova rota é encontrada antes que algum nó tente enviar algum pacote por esse link quebrado. Além disso, como o número de ligações *Fast-Link* de um nó em modo *Fast-moving* é reduzido, o excesso de tráfego de controle diminui. Assim quando um nó no modo *Fast-OLSR* não tem mais *Fast Links*, ele para de enviar *Fast-hellos* e retorna ao modo padrão.

A solução proposta no trabalho citado não específica como o algoritmo consegue analisar a rede. O algoritmo possui um ajuste de acordo com a mobilidade da vizinhança, porém a forma como essa mobilidade é detectada não está clara. Outro ponto a destacar, é que a solução propõe diminuir o *overhead* de uma outra forma, diferentemente de diminuir o intervalo de envio da mensagem de controle *Hello*, ela diminui o tamanho da mensagem de controle.

Em outro importante trabalho (STANZE; ZITTERBART; KOCH, 2006), foi proposta uma técnica para medir o grau de mobilidade da rede. O indicador do grau da mobilidade de rede é chamado *Relative Velocity Indicator* (MARVIN). De acordo com o valor do MARVIN, o algoritmo modifica o intervalo de envio das mensagens de controle. Para o cálculo do MARVIN é levado em consideração

todos os pacotes recebidos, seja um pacote de dados ou uma mensagem de controle do protocolo. O valor do MARVIN é influenciado pelo peso de cada nó vizinho. Utiliza-se a seguinte fórmula para calcular o peso W de cada vizinho:

$$W = menor(\frac{epi}{tp}, 1)$$

O indicador chamado *epi* define o intervalo atual em que um nó S estima receber um novo pacote de um vizinho. A variável *tp* é o tempo decorrido desde o último pacote recebido do nó S pelo mesmo vizinho. De acordo com a fórmula o peso de um vizinho depende do intervalo de tempo desde do último pacote recebido desse vizinho. Desta forma, se o tempo esperado por um nó para receber o próximo pacote do seu vizinho é igual ao valor atual do *epi*, esse vizinho é ponderada por um. Se nenhum pacote foi recebido anteriormente do vizinho, então o *epi* é zero, e o peso do vizinho é definido como menor do que 1. O valor de MARVIN é calculado com a seguinte fórmula:

$$MARVIN_t = \frac{C_t}{\sum_{i \in N} w_i}$$

Na fórmula o t é o intervalo constante em que um novo valor para MARVIN é determinado, e C_t é o número de alterações ocorridas nos últimos t segundos na vizinhaça do nó. O valor de C_t é calculado pela adição do número de vizinhos que enviaram ao menos um pacote no intervalo t segundos com número de vizinhos perdidos, ou seja, vizinhos que não enviaram mais nenhum pacote nesse intervalo. N é o conjunto de todos os nós que tenha enviado ao menos um pacote no intervalo t. O w_i é o peso correspondente de cada vizinho. Com esta fórmula, o valor da métrica é suavizado com os valores anteriores. O protocolo de roteamento solicita periodicamente o valor atual do MARVIN. Uma função chamada MASP é usada para mapear o valor atual do MARVIN para um valor correspondente do *Hello Interval*. Desta forma, o algoritmo é configurado para utilizar o valor mapeado. O MARVIN pode ser introduzido no OLSR para melhorar seu desempenho. Para fazer isso a função que faz o mapeamento do MARVIN foi determinada de acordo com os três passos descritos a seguir: (i) Em um cenário de alta mobilidade um

valor considerado ideal para o parâmetro é definido; (ii) Um valor para o MARVIN é escolhido, ele é mapeado para o parâmetro definido no passo anterior; (iii) Os dois passos anteriores são repetidos para cenários com diferentes padrões de mobilidade. Por meio de interpolação, os valores são mapeados em uma função. A seguinte função foi obtida:

$$F(MASP) = \frac{1}{6{,}61659 \cdot (MASP - 0{,}0103149)}$$

De todas as funções testadas esta teve o menor desvio padrão por isso ela foi escolhida. O valor do parâmetro *Hello Interval* é configurado através dessa função. O valor é definido entre 0,5 segundo e 5 segundos. Se o valor de MASP aplicado a função retorna valor menor do que 0,5, o *Hello Interval* é configurado para 0,5 segundos. Se o valor obtido foi maior que 5 o parâmetro é configurado em 5 segundos. Desta forma, o OLSR é configurado para atender melhor as reais condições da rede.

A solução proposta neste trabalho tem uma limitação, pois os parâmetros de ajuste da rede são pré-estabelecidos através de simulações. Neste caso um dado valor de MARVIN corresponde a um valor do intervalo. Esse mapeamento pré-estabelicido (estático) é uma limitação para o uso da técnica.

Em uma outra solução proposta (HERNANDEZ-CONS; KASAHARA; TAKAHASHI, 2010), o intervalo de tempo de envio das mensagens de controle é auto-configurado de acordo com a LCR, que é o cálculo da taxa de mobilidade da rede. Dessa forma, o intervalo se ajusta de acordo com o valor da LCR. O valor instantâneo da LCR é calculado através da seguinte fórmula:

$$LCR = \frac{nl+lp}{t}$$

Dado um nó A, sempre que um novo nó entra na sua vizinha o contador de links novos *nl* é incrementado, e sempre que um nó não envia mais pacotes para o nó A dentro de um tempo limite o contador de links perdidos *lp* é incrementado. A variável *t* é o tempo decorrido desde da última medição da LCR. Considera-se

a média ponderada dos valores da LCR. Para fazer o cálculo utiliza-se a seguinte fórmula:

$$LCR_e(k) = LCR_e(k-1)\alpha_E + LCR_s(k)(1-\alpha_E).$$

Onde $LCR_e(k)$ é a média ao longo do tempo, $LCR_e(k-1)$ é a média anterior ao valor atual e $LCR_s(k)$ é o valor absoluto mais atual amostrado, calculado através da fórmula: $LCR = \frac{nl+lp}{t}$, que foi explicada anteriormente.

E α_E é o parâmetro do filtro. Normalmente, o parâmetro α_E é ajustado com valor fixo, porém, o autor do trabalho utilizou um valor dinâmico devido a grande mobilidade da rede.

O parâmetro *Hello Interval* é ajustado de acordo com o algoritmo ilustrado na figura 4.1.

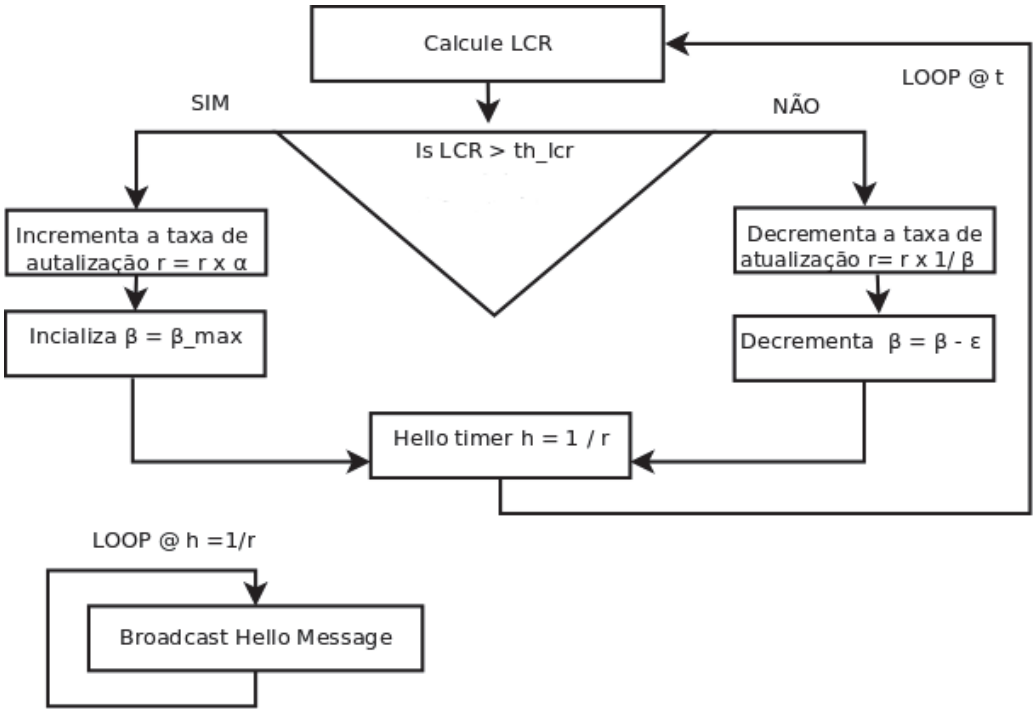

Figura 3.1: Algoritmo para ajuste do parâmetro *Hello Interval*

Esse algoritmo funciona como o descrito a seguir. Um nó com uma taxa de atualização maior recebe mensagens *Hello* com mais frequência. Um coeficiente

de aumento foi definido, sendo $\alpha > 1$ e o coeficiente de redução é de $0 <$ ß <1, os valores de α e ß são pré-definidos. O fator limiar thlcr inicial é definido. Calcula-se o valor da LCR, se o valor calculado for maior do que o thlcr, a taxa de atualização *r* é aumentada, o coeficiente de redução máximo (ßmax) é definido como o valor atual do ß, e o valor do *Hello Interval* é ajustado. Então o loop recomeça. Se não houver nenhuma mudança, ou seja LCR menor que thlcr, então a taxa de atualização é decrementada para que seja enviada um número menor de mensagens *Hello*. Quando uma mudança é detectada, o valor da LCR calculado nesse instante é maior do que o valor do thlcr, então o parâmetro ß é definido para ßmax. Da próxima vez em que se detectar que não existem alterações na rede, ou seja, valor da LCR é menor do que o valor do thlcr, então a diminuição da taxa de atualização é feita de forma agressiva, pois o valor de ß nesse momento é máximo. Se a rede permanece sem alterações, a taxa de atualização continua diminuindo, e o valor de ß também diminui. Desta forma, a primeira queda é agressiva e a as próximas diminuições são menores porque ß também é diminuído por ε. O coeficiente de aumento máximo α limita o crescimento da taxa de envio de mensagens *Hello*, pois o valor não pode crescer sem limites. Na implementação do algoritmo, o valor limite mínimo foi estabelecido, de modo que mesmo em uma rede completamente estática tenha sempre um número mínimo de mensagens *Hello* enviadas.

Os resultados apresentados no trabalho citado mostram que o algoritmo teve um menor *overhead* que o OLSR padrão, porém a taxa de entrega de pacotes teve um resultado pior. Ou seja, o algoritmo não contribui significativamente para o melhoramento do desempenho das redes Ad hoc, pois um dos motivos para diminuir *overhead* é que sobre mais recursos da rede para envio de mensagens de dados, aumentando assim a taxa de entrega de pacotes.

4 METODOLOGIA

O objetivo deste trabalho é propor um algoritmo para auto-configuração do intervalo de tempo de envio da mensagem de controle do protocolo OLSR, chamada *Hello*. O ajuste dinâmico será feito de acordo com as mudanças na topologia da rede. A auto-configuração diminui a sobrecarga de controle aumentando assim, o desempenho da rede.

A execução do projeto foi feita de acordo com as atividades relacionadas a seguir:

1. *Proposta de um novo algoritmo*. Nessa etapa, foi desenvolvido um novo mecanismo para reconfiguração do OLSR. Por padrão, uma mensagem *Hello* é enviada no intervalo de 2 segundos. O algoritmo proposto ajustará esse intervalo de acordo com a mobilidade dos nós. Para construir o novo algoritmo, foram necessárias duas abordagens: (i) Definir uma forma de analisar e medir a mobilidade dos nós. (ii) Criar uma abordagem para ajustar o valor do parâmentro *Hello Interval* de acordo com a mobilidade medida.

2. *Avaliação quantitativa*. Nessa etapa, foi avaliado o algoritmo proposto. Foi realizada uma comparação de desempenho entre a solução proposta neste projeto e o OLSR padrão. A avaliação foi feita por meio de simulações, em diversos cenários. Eles foram diferentes em termos do número de nós na rede, do grau de mobilidade dos nós e da carga de tráfego. Foi utilizado o simulador NS-3. Foi feita uma análise quantitativa dos resultados. As métricas utilizadas para avaliação foram: Vazão de dados da rede em *megabits* por segundo, a porcentagem de perda de pacotes e a vazão de dados de controle em *megabits* por segundo, denominado *overhead*. Os resultados são apresentados, como média de várias repetições das simulações, considerando um intervalo de confiança.

5 SOLUÇÃO PROPOSTA

O parâmetro *Hello Interval* define o intervalo de tempo entre o envio de mensagens *Hello*. Essa mensagem é responsável pela detecção de vizinhos. No protocolo OLSR, por padrão, o valor do parâmetro é 2 segundos. Em uma rede estática o valor 2 segundos pode gerar mensagens *Hellos* desnecessárias. Pois, em um intervalo de tempo em que a rede não sofreu nenhuma alteração não é preciso disseminar informações sobre a rede entre os nós. Nesta situação um valor maior para o parâmetro diminuiria o tráfego de controle aumentando a quantidade de recurso disponível para o tráfego de dados.

Em uma rede com mobilidade, o número de alterações na rede é alto, dessa forma, as informações tem que ser disseminadas rapidamente para que os nós tenham suas tabelas de roteamento atualizadas. Dessa forma, um valor mais alto para o *Hello Interval* pode atrasar as atualizações das tabelas, fazendo com que um nó envie um pacote por um caminho que não existe mais, e/ou necessite de um tempo maior para atualizar em sua tabela de roteamento, uma nova rota que passe por um novo vizinho. Para este cenário um valor menor do *Hello Interval* disseminaria de forma mais rápida as informações, possibilitando que o nó utilize os caminhos mais eficientes para enviar um pacote e evitando que um nó envie um pacote por um caminho que não existe mais, contribuindo assim para o aumento da vazão da rede.

Dessa forma, o valor estático do parâmetro *Hello Interval* reduz a vazão da rede. Para evitar isso, o algoritmo proposto modifica o valor de acordo com as condições reais da rede. Em situações nas quais a rede se encontra estática, ele diminui a frequência do envio de mensagens *Hello*, ou seja, aumenta o valor do parâmetro *Hello Interval*. Em situações nas quais rede se encontra dinâmica ele diminui o valor do parâmetro.

Para fazer o ajuste dinâmico um algoritmo precisa passar por duas etapas. A primeira consiste em analisar a rede, e a segunda em ajustar o *Hello Interval* de acordo com as condições analisadas. As duas etapas são ilustradas pela figura 5.1.

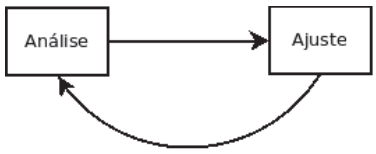

Figura 5.1: Etapas do algoritmo

No algoritmo proposto a etapa de análise consiste em verificar o grau de mobilidade da rede. Ele funciona localmente em cada nó. A mobilidade é identificada da seguinte maneira: Se um dado nó verificar em um dado instante T que um link entre ele um vizinho foi perdido, ou se um novo link for criado entre o nó e um vizinho novo. Então, nessas duas situações ocorreram mudanças na rede no instante T, dessa forma, considera-se que existe mobilidade na rede no instante T. Caso não aconteça nenhuma das situações anteriores a rede é considerada estática no instante T, ou seja, sem mobilidade.

Na etapa de ajuste, o algoritmo modifica o *Hello Interval* de acordo com análise feita em um dado instante T. Se a rede é considerada estática, então, o valor do *Hello Interval* é incrementado com o valor de α no instante T, esse incremento é feito respeitando o valor do limitante superior. Caso seja constatado que existe mobilidade na rede, uma mensagem *Hello* é enviada, o valor do *Hello Interval* é reduzido ao limitante inferior e o ciclo recomeça. O algoritmo pode ser ilustrado pela figura 5.2.

Na fase de análise em (BENZAID; MINET; AGHA, 2002) o autor não descreve como o algoritmo analisa a rede. Em (STANZE; ZITTERBART; KOCH, 2006) a mobilidade da rede é influenciada pelo tempo decorrido desde que um nó recebeu um pacote de um dado vizinho e o tempo estimado para receber próximo. Já em (HERNANDEZ-CONS; KASAHARA; TAKAHASHI, 2010) a mobilidade da rede é influenciada pela entrada de um novo vizinho na tabela de roteamento

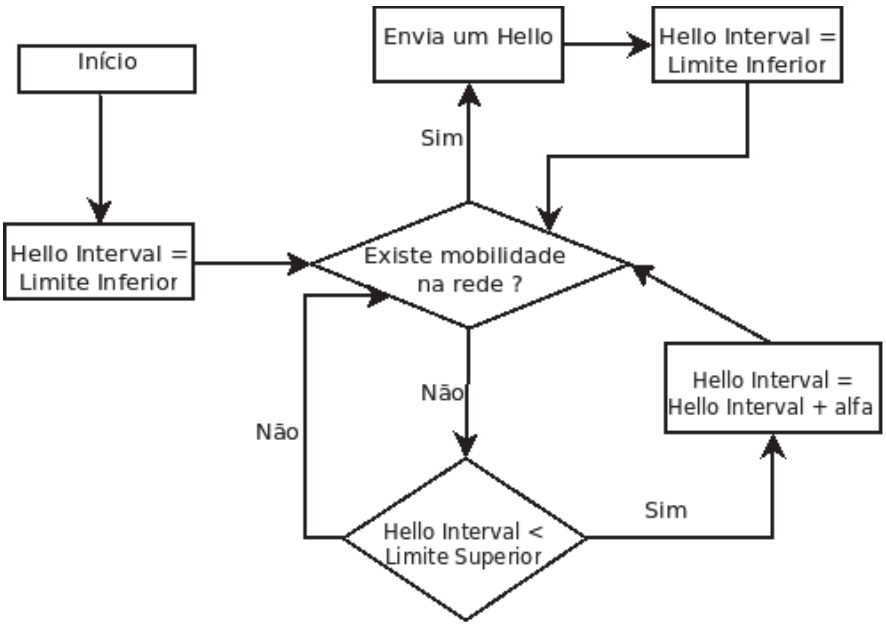

Figura 5.2: Algoritmo Proposto

de um nó e também pelo tempo decorrido desde que um nó recebeu um pacote do seu vizinho. Diferentemente, no algoritmo proposto nesse trabalho, a mobilidade é influenciada pela criação de um link com um vizinho, ou pela perda de um link existente. Essa análise é feita utilizando uma função nativa do protocolo OLSR, que é invocada toda vez que se tem uma alteração no repositório de links.

Na fase de ajuste em (BENZAID; MINET; AGHA, 2002) o algoritmo atua diminuindo o tamanho das mensagens *Hello*, enviando mensagens menores denominadas *Fast-Hellos*. Já em (STANZE; ZITTERBART; KOCH, 2006) e (HERNANDEZ-CONS; KASAHARA; TAKAHASHI, 2010) o algoritmo atua no ajuste do parâmetro *Hello Interval*, assim como no algoritmo proposto. Porém, o ajuste é feito de forma singular. Em (STANZE; ZITTERBART; KOCH, 2006) o valor do parâmetro altera entre os valores 0,5 segundos e 5 segundos de acordo com a fórmula citada no capítulo Trabalhos Relacionados. Em (HERNANDEZ-CONS; KASAHARA; TAKAHASHI, 2010) o ajuste é feito incrementando o valor do *Hello Interval* em uma rede sem mobilidade, e decrementando em uma rede

com mobilidade, respeitando o limite superior e inferior. O valor do incremento e do decremento é calculado de acordo com a fórmula citada no capítulo Trabalhos Relacionados. No algoritmo proposto existe um limite inferior, valor pelo qual o algoritmo é iniciado. Em um cenário sem mobilidade o valor do parâmetro *Hello Interval* é incrementado pelo valor de α, respeitando o limite superior. Porém, diferentemente dos outros trabalhos citados, quando se é detectada mobilidade, o algoritmo imediatamente envia uma mensagem *Hello* e o valor atual do parâmetro sofre uma queda agressiva para o valor do limitante inferior. Essa mensagem *Hello* é enviada a fim de disseminar mais rápido a informação.

No algoritmo implementado o limite superior foi definido para que o laço do algoritmo não cresça sem limites. E o limite inferior foi definido para que o valor do *Hello Interval* não se torne tão pequeno, fazendo com que o protocolo OLSR comprometa todo o recurso da rede, enviando mensagens de controle a todo instante. Os valores dos limitantes superior e inferior e o valor do incremento α foram obtidos através de simulações, o valores definidos serão abordados no capítulo seguinte.

6 AVALIAÇÃO DE DESEMPENHO

Para avaliar os diferentes valores do *Hello Interval* utilizando implementação do OLSR padrão, para definir o valor do incremento α e para avaliar o algoritmo proposto, foi utilizado o simulador de redes Network Simulator (NS-3) [1]. O algoritmo proposto foi implementado utilizando também o NS-3, foi feita uma extensão do código fonte que está disponível no Apêndice A deste texto. A utilizaçao do simulador demandou tempo para construir um cenário que pudesse simular da melhor forma as condições necessárias para avaliar os parâmetros e o algoritmo final.

Avaliação de um cenário sem mobilidade

Foi simulado um cenário de rede sem fio Ad hoc com 25 nós organizados em uma grade 5x5, com uma distância de 500 metros entre cada nó. Todos os nós utilizaram o protocolo OLSR. Os valores testados para o parâmetro *Hello Interval* foram 1, 2, 3, 4, 5 e 6 segundos. Cada simulação tem duração de 200 segundos. Foram feitas 20 repetições para cada valor do *Hello Interval* em um cenário sem mobilidade. Foi sorteado a cada simulação a origem e o destino do fluxo de 6 Mbit/s de pacotes UDP.

A semente foi alterada a cada repetição, garantindo assim, que cada simulação fosse independente. Os resultados obtidos estão apresentados como média de várias repetições das simulações, considerando um intervalo de confiança de 90%. As figuras 6.1 até 6.4 demonstram os resultados obtidos.

No gráfico apresentado pela figura 6.1 verificou-se que o valor de 6 segundos para o parâmetro *Hello Interval* corresponde ao menor valor de *overhead*. Pois, um valor maior para o parâmetro faz o protocolo gerar um número menor de mensagens *Hello*, diminuindo assim o *overhead* da rede.

Pelo resultado demonstrado na figura 6.2 foi possível constatar que em um cenário sem mobilidade o valor de 6 segundos do *Hello Interval* obtém uma vazão

[1] www.nsnam.org

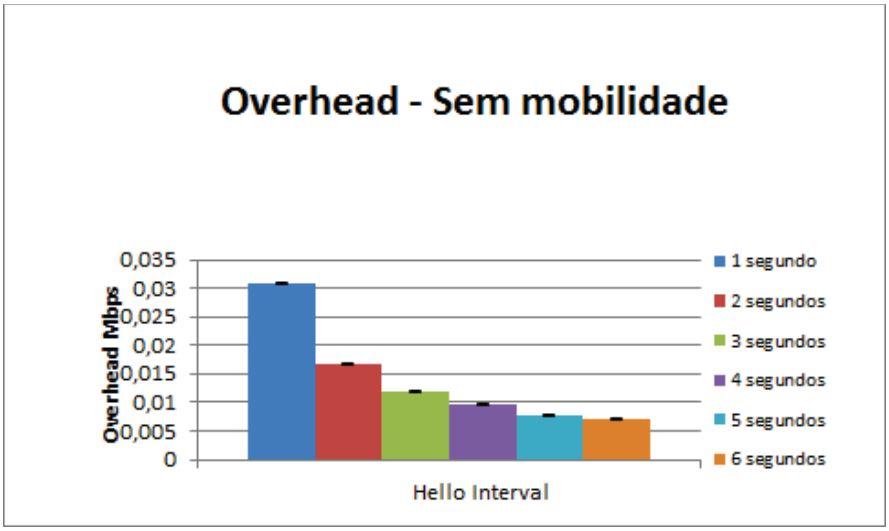

Figura 6.1: Overhead sem mobilidade

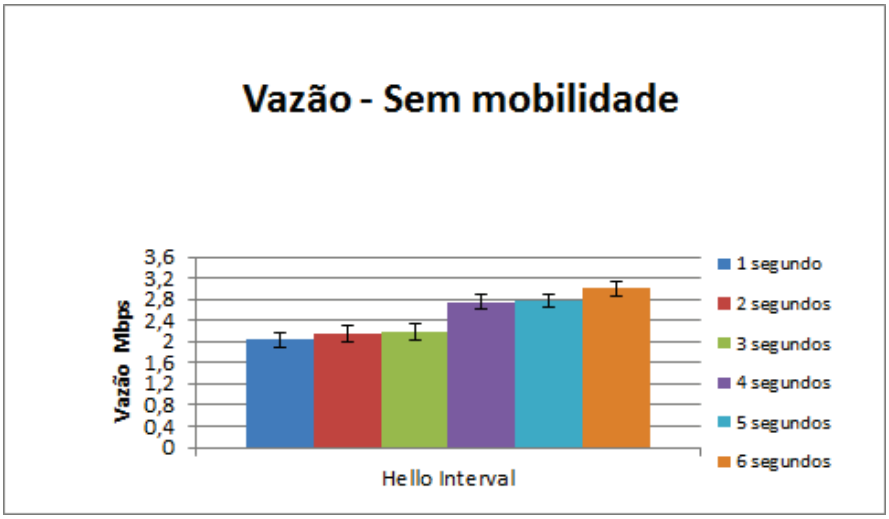

Figura 6.2: Vazão sem mobilidade

maior, pois como citado anteriormente o *overhead* é menor, sobrando assim mais recursos para enviar pacotes de dados.

De acordo com os resultados apresentados na figura 6.3 o valor de 6 segundos obteve a menor taxa de perda de pacotes.

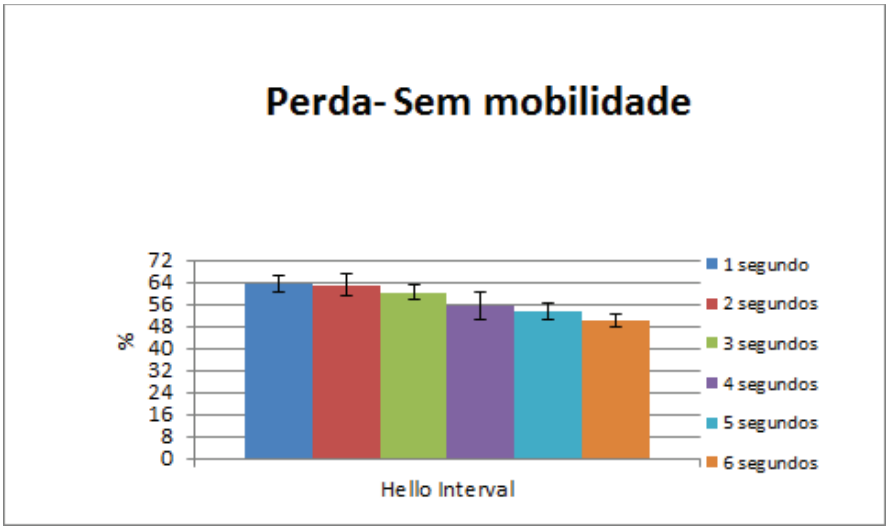

Figura 6.3: Perda sem mobilidade

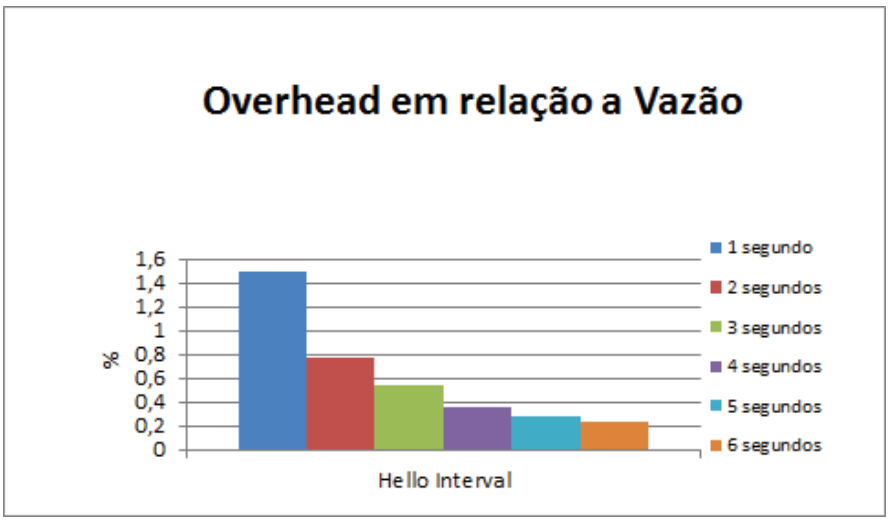

Figura 6.4: Porcentagem do overhead em relação a vazão total em um cenário sem mobilidade

A figura 6.4 demonstra a porcentagem do *overhead* em relação a vazão total. O valor de 6 segundos tem uma porcentagem menor de overhead em relação ao tráfego total.

Avaliação em um cenário com mobilidade

Foi utilizado os mesmos parâmetros de simulação que foram usados na avaliação de um cenário sem mobilidade. Porém, agora os nós se movem de forma aleatória a uma velocidade que pode variar de 2m/s a 4m/s. As figuras 6.5 até 6.8 demostram os resultados.

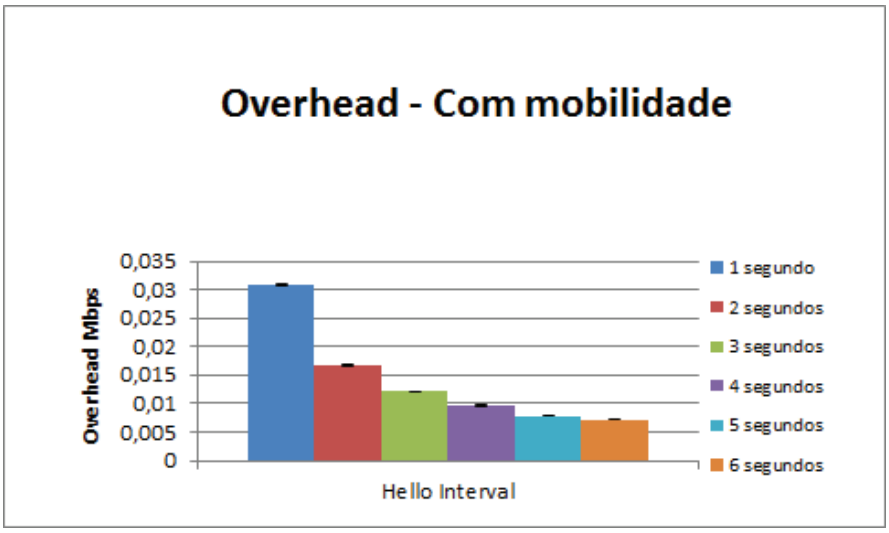

Figura 6.5: Overhead com mobilidade

O resultado obtido na figura 6.5 confirma que o valor de 6 segundos para o parâmetro *Hello Interval* gera também neste caso um *overhead* menor, pois o número gerado de mensagens *Hello* é menor.

Na figura 6.6 foi possível constatar que o valor de 6 segundos não faz o protocolo gerar uma vazão maior. Apesar de obter um *overhead* menor como demonstrado na figura 6.5, o valor de 6 segundos aumenta a perda de pacotes. Pois um valor mais alto para o parâmetro em um cenário com mobilidade faz com que as informações da tabela de vizinhaça fiquem defasadas. Dessa forma, o nó poderá enviar pacotes por uma rota que passa por um vizinho que não existe mais. O valor de 1 segundo corresponde a melhor vazão, pois um valor menor para o parâmetro *Hello Interval* em cenários de mobilidade faz com que as informações

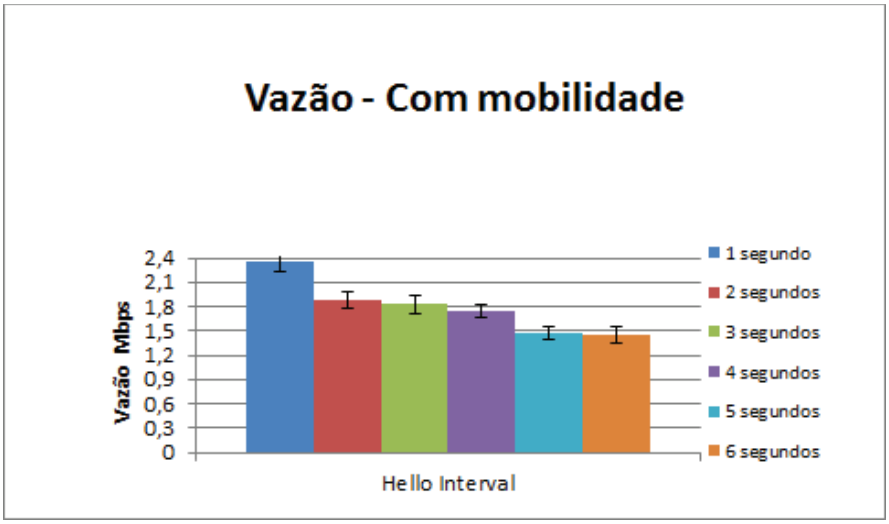

Figura 6.6: Vazão com mobilidade

sejam disseminadas de forma mais rápida, evitando que os nós tenham em suas tabelas de roteamento informações defasadas, melhorando assim a vazão da rede.

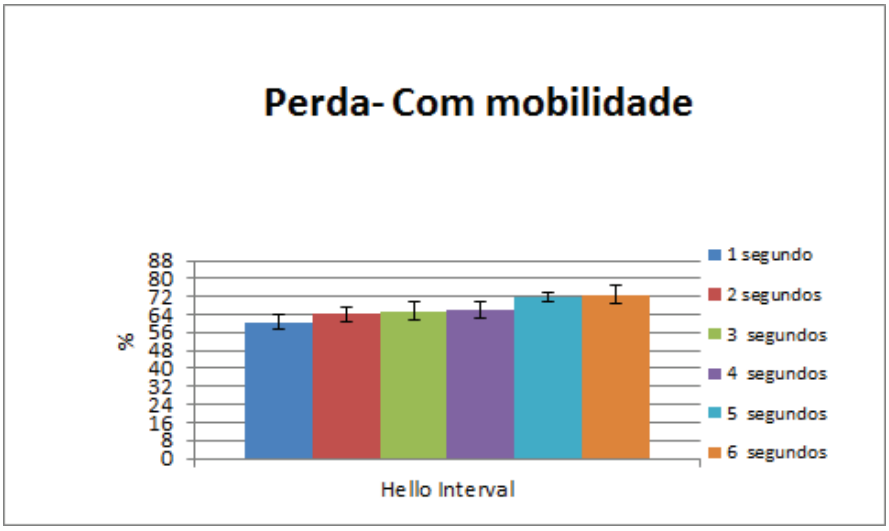

Figura 6.7: Perda com mobilidade

O gráfico da figura 6.7 demonstra que em um cenário de mobilidade, a perda é maior para o valor mais alto do parâmetro, devido ao intervalo de tempo maior para atualizar as informações da topologia, fazendo com as informações

fiquem defasadas. O valor de 1 segundo obteve o melhor resultado, pois ele faz com que se atualize de forma mais rápida as informações.

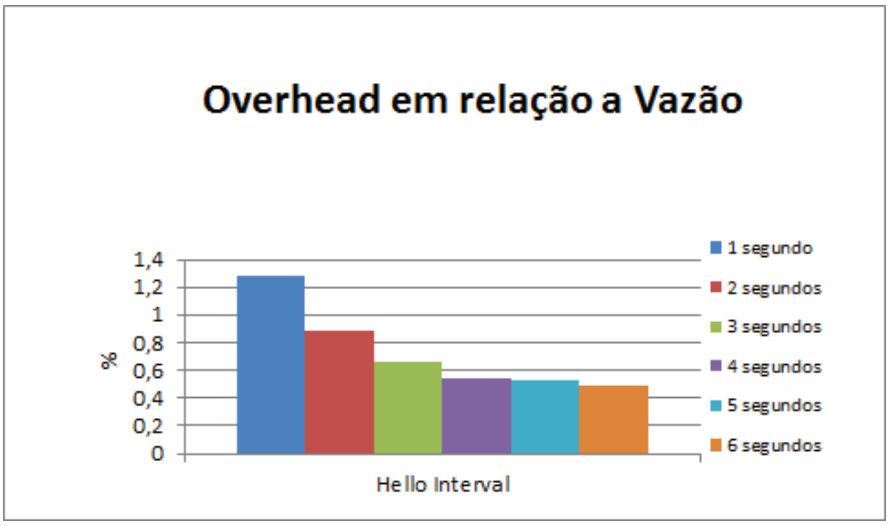

Figura 6.8: Porcentagem do overhead em relação a vazão total em um cenário com mobilidade

Os resultados do gráfico da figura 6.8 demostram a porcentagem do *overhead* em relação a vazão total.

Avaliação do ajuste α

O algoritmo foi implementado e testado. O valor do parâmetro de ajuste α foi obtivo através de simulações. Foram feitas novas simulações utilizando os mesmos parâmetros de simulação anterior. Porém, nesta situação se utilizou um cenário misto. Neste cenário misto a simulação tem um estágio em que os nós se encontram estáticos, esse estágio dura 100 segundos. E um outro estágio que também tem a duração de 100 segundos em que o nós estão movendo de acordo com a mobilidade definida na avaliação do cenário com mobilidade do OLSR padrão. Os resultados obtidos estão descritos pelas figuras 6.9.

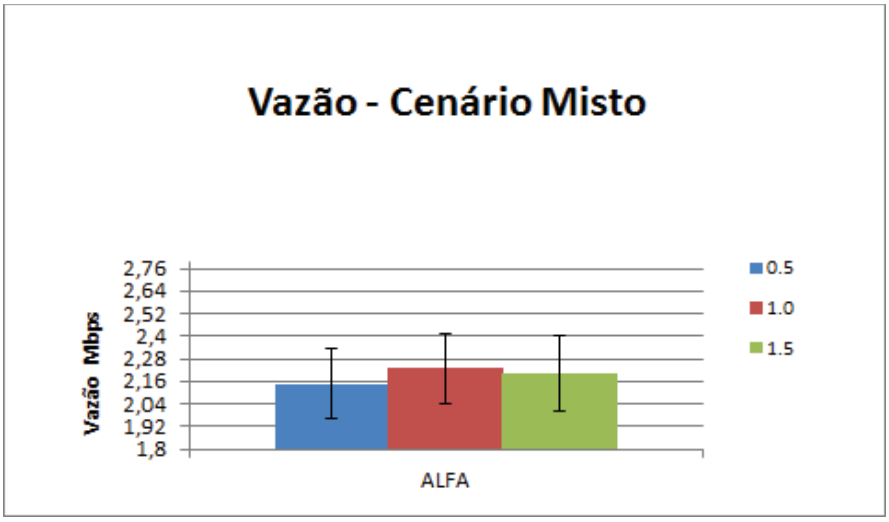

Figura 6.9: Vazão de cada alfa testado

Verificou-se na gráfico demonstrado pela figura 6.9 que o valor de 1 segundo para o parâmetro de incremento α obteve uma vazão melhor, porém, bem próxima ao valor de 1,5 segundos. O valor de 1 segundo foi definido para α.

Avaliação do algoritmo proposto

Para análise do algoritmo implementado foi utilizado a mesma simulação citada na avaliação do parâmetro de incremento α. O algoritmo desse trabalho foi comparado com o OLSR padrão, utilizando o valor padrão 2 segundos, no mesmo cenário de simulação. Os resultados obtidos estão apresentados como média de várias repetições das simulações, considerando um intervalo de confiança de 68% . As figuras 6.10 até a 6.13 apresentam os resultados obtidos.

No gráfico da figura 6.10 verificou-se que o algoritmo proposto obteve melhor vazão média do que o OLSR padrão, pois ele se adapta a cenários com ou sem mobilidade. Intercalando entre os valores ótimos. Diminuindo a perda em cenários com mobilidade, e diminuindo o *overhead* em cenários sem mobilidade.

O gráfico da figura 6.11 demonstra o que foi dito anteriormente, o algortimo proposto tem uma perda menor em relação ao OLSR padrão.

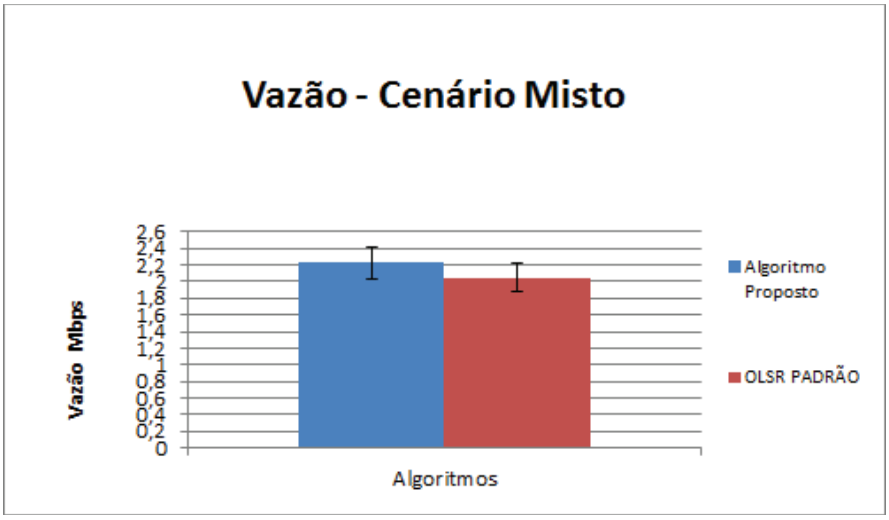

Figura 6.10: Vazão do algoritmo proposto comparado ao OLSR padrão

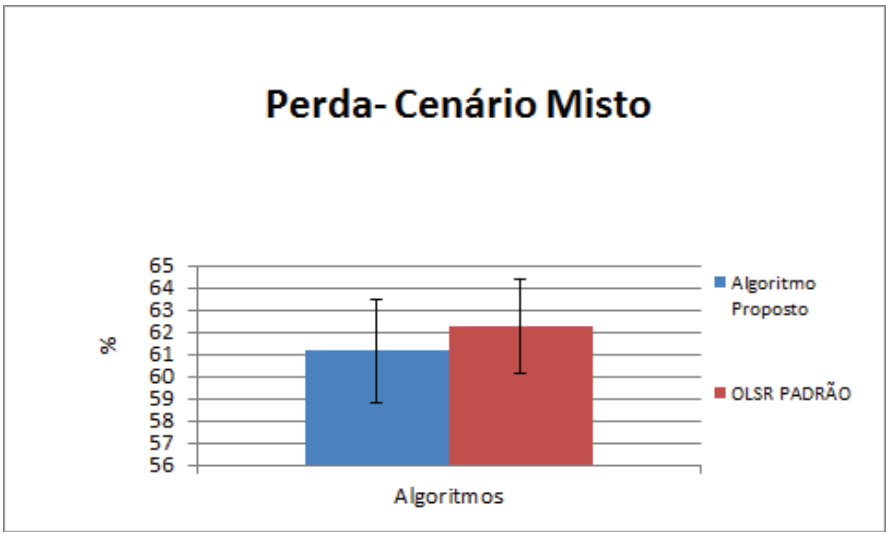

Figura 6.11: Perda do algoritmo proposto comparado ao OLSR padrão

A figura 6.12 demonstra que o *overhead* do algoritmo implementado é bem inferior ao do OLSR padrão, pois, em situações que a rede se encontra estática o algoritmo implementado aumenta o valor do *Hello Interval*, diminuindo o número de mensagens *Hello* enviadas.

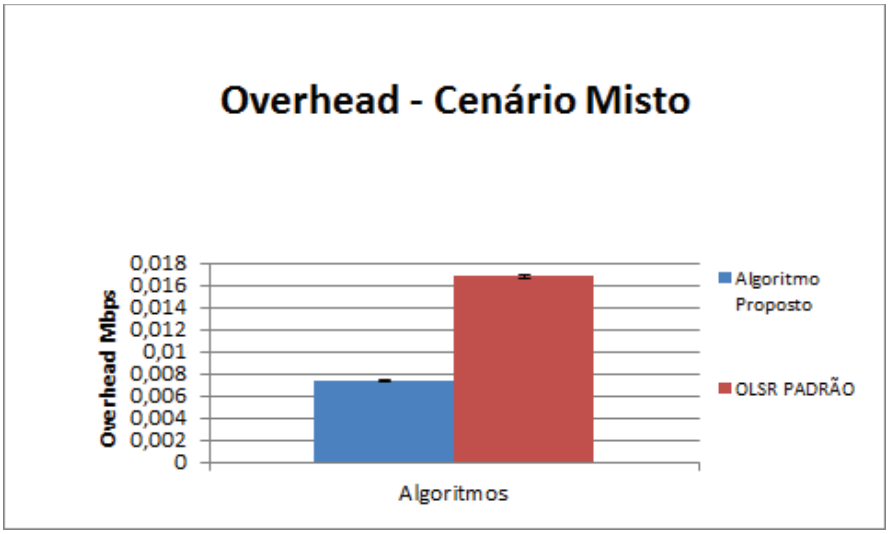

Figura 6.12: Overhead do algoritmo proposto comparado ao OLSR padrão

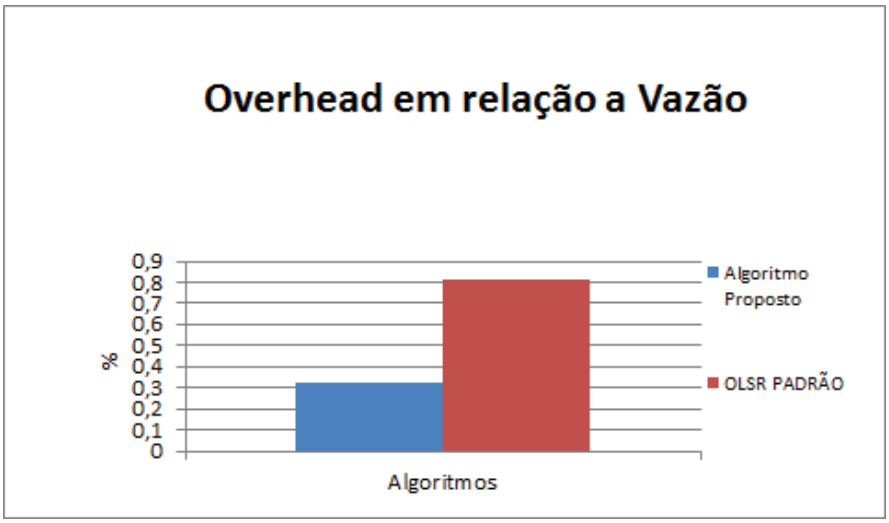

Figura 6.13: Porcentagem de overhead em relação a vazão total do algoritmo proposto comparado ao OLSR padrão

A figura 6.13 demostra o quanto o *overhead* do algoritmo proposto é menor do que o OLSR padrão em relação a vazão total.

Através dos resultados obtidos verifica-se que o OLSR estendido com algoritmo proposto tem uma vazão superior. Porém, não se pode dizer que o algo-

ritmo proposto é melhor que o padrão em qualquer situação, pois existe uma faixa em que o intervalo de confiança dos dois gráficos se encontram.

7 CONCLUSÃO

O protocolo OLSR é muito utilizado em redes Ad hoc. Neste protocolo para que cada nó da rede possa descobrir quem são os seus vizinhos, o OLSR envia mensagens *Hello*. Estas mensagens são trocadas em um intervalo definido pelo o parâmetro *Hello Interval*. Porém, as mensagens acabam aumentando o tráfego de controle. Para solucionar este problema um algoritmo novo foi proposto, a fim de ajustar o parâmetro *Hello Interval*. Alguns trabalhos propostos, já existentes para melhorar o protocolo, foram estudados a fim de entender o mecanismo de ajuste de cada trabalho. Um novo algoritmo foi proposto a partir do problema inicial. Esse algoritmo foi implementado e avaliado utilizando o simulador NS-3, para fazer a avaliação foram implementadas funções para medir a vazão e a taxa da perda de pacotes da rede.

Os resultados obtidos com o OLSR padrão em cenários com ou se mobilidade, se demostrou dentro do esperado. Pois, para um cenário sem mobilidade, espera-se que um valor maior para o parâmetro *Hello Interval* diminua o trafégo de controle, aumentando assim a vazão. Em um cenário com mobilidade, espera-se que um valor baixo para o *Hello Interval* atualize de forma mais rápida as informações da rede entre os nós, aumentando a vazão da rede. Diferentemente, um valor alto para o parâmetro nessa situação, diminuiria também o trafégo de controle. Porém, aumentaria muito a perda de pacotes, o que diminuiria a vazão da rede. Dessa forma, encontrou-se coerência nos resultados, o que justificou a possibilidade de implementar o algoritmo. Pois, existe um cenário em que o valor alto ganha e um outro cenário em que o valor baixo ganha. Dessa forma, se o algoritmo intercalar entre as duas situações a vazão da rede será maior.

A partir dessas informações o algoritmo foi implementado no simulador NS-3. O algoritmo obteve um resultado satisfatório porém, não dentro do esperado. Pois, se estimava um ganho maior na vazão final.

Nos resultados obtidos, a diferença da vazão do tráfego de controle do algoritmo implementado em relação ao OLSR padrão foi alta. Porém, na vazão de dados da rede não foi encontrado em termos absolutos um ganho muito acentuado.

O algoritmo proposto em cenários sem mobilidade diminui o tráfego de controle pois, aumenta o intervalo de envio de mensagens de controle, gerando assim um número menor de mensagens *Hello*. Ao diminuir o tráfego de controle sobra mais recurso para que a rede possa enviar mensagens de dado. Porém, em termos absolutos a vazão não apresentou um ganho muito acentuado, pois a diferença entre o valor do *overhead* do algoritmo proposto em relação ao padrão é de aproximadamente 0,5% da vazão total, esse resultado é demonstrado na figura 6.14 no capítulo Avaliação de Desempenho. Em um cenário em que a vazão medida é de aproximadamente 2,2Mbps, esta porcentagem equivale a uma diferença muito pequena.

Para trabalhos futuros, mudanças podem ser implementadas no algoritmo proposto, a fim de melhorar o ajuste do *Hello Interval*, aumentando assim o ganho. Outro trabalho futuro para ser desenvolvido seria comparar o algoritmo proposto em relação aos trabalhos citados no capítulo de Trabalhos Relacionados. Um outro próximo trabalho seria avaliar o algoritmo em cenários diferentes, usando outras métricas de avaliação. E avaliar o algoritmo utilizando mais de um fluxo, e em um cenário com número maior de nós.

REFERÊNCIAS BIBLIOGRÁFICAS

BENZAID, M.; MINET, P.; AGHA, K. A. Integrating fast mobility in the olsr routing protocol. In: IEEE. *Mobile and Wireless Communications Network, 2002. 4th International Workshop on*. [S.l.], 2002. p. 217–221.

CLAUSEN, T.; JACQUET, P. Rfc 3626-optimized link state routing protocol (olsr). *IETF RFC3626*, IETF, 2003.

COUTO, D.; AGUAYO, D.; BICKET, J.; MORRIS, R. A high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, Springer, v. 11, n. 4, p. 419–434, 2005.

HERNANDEZ-CONS, N.; KASAHARA, S.; TAKAHASHI, Y. Dynamic hello/timeout timer adjustment in routing protocols for reducing overhead in manets. *Computer Communications*, Elsevier, v. 33, n. 15, p. 1864–1878, 2010.

JOHNSON, D.; MALTZ, D. Dynamic source routing in ad hoc wireless networks. *Mobile computing*, Springer, p. 153–181, 1996.

KUROSE, J.; ROSS, K. Redes de computadores e internet. *São Paulo: Person*, 2010.

PERKINS, C.; BHAGWAT, P. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM Computer Communication Review*, ACM, 1994.

PERKINS, C.; ROYER, E. Ad-hoc on-demand distance vector routing. In: IEEE. *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*. [S.l.], 1999. p. 90–100.

STANZE, O.; ZITTERBART, M.; KOCH, C. Mobility adaptive self-parameterization of routing protocols for mobile ad hoc networks. In: IEEE. *Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE*. [S.l.], 2006. v. 1, p. 276–281.

A ALGORITMO PROPOSTO

Para fase de análise do algoritmo utilizou-se duas funções próprias do OLSR, uma chamada *LinkSensing* e outra chamada *LinkTupleTimerExpire*. Foi adicionado nessas duas funções uma variável chamada *mobilidade* que verifica se um link foi adicionado ou removido.

Para fase de ajuste foi feito uma modificação na função *HelloTimerExpire* que é responsável por chamar a função *SendHello*.

Foi feito também para a fase de ajuste a função chamada *AjusteHello* que é responsável por ajustar o valor do *Hello Interval*.

As funções estão dispostas abaixo. Em ordem, a função *AjusteHello*, *HelloTimerExpire*, *LinkTupleTimerExpire* e *LinkSensing*.

```
void
RoutingProtocol::AjusteHello ()
{
 if ( incre <= 5)
  {
        incre = incre + 1;
        novohello = (Seconds(incre));
  }
  if (mobilidade == true)
  {

        incre=1;
        novohello = (Seconds(incre));
        mobilidade = false;
        SendHello ();

  }

  m_AjusteHello.Schedule (Seconds(1));
```

```
}


void
RoutingProtocol::HelloTimerExpire ()
{
  SendHello ();


  m_helloTimer.Schedule (novohello);


}




void
RoutingProtocol::LinkTupleTimerExpire (Ipv4Address neighborIfaceAddr)
{
  Time now = Simulator::Now ();

  // the tuple parameter may be a stale copy; get a newer version from m_state
  LinkTuple *tuple = m_state.FindLinkTuple (neighborIfaceAddr);
  if (tuple == NULL)
    {
      return;
    }
  if (tuple->time < now)
    {
      RemoveLinkTuple (*tuple);
      mobilidade= true;
    }
  else if (tuple->symTime < now)
    {
```

```
      if (m_linkTupleTimerFirstTime)
        m_linkTupleTimerFirstTime = false;
      else
        NeighborLoss (*tuple);

      m_events.Track (Simulator::Schedule (DELAY (tuple->time),
                                           &RoutingProtocol::LinkTupleTimerExpire, this,
                                           neighborIfaceAddr));
    }
  else
    {
      m_events.Track (Simulator::Schedule (DELAY (std::min (tuple->time, tuple->symTime)),
                                           &RoutingProtocol::LinkTupleTimerExpire, this,
                                           neighborIfaceAddr));
    }
}

///
///




void
RoutingProtocol::LinkSensing (const olsr::MessageHeader &msg,
                              const olsr::MessageHeader::Hello &hello,
                              const Ipv4Address &receiverIface,
                              const Ipv4Address &senderIface)
{
  Time now = Simulator::Now ();
  bool updated = false;
  bool created = false;
  NS_LOG_DEBUG ("@" << now.GetSeconds () << ": Olsr node " << m_mainAddress
                    << ": LinkSensing(receiverIface=" << receiverIface
```

```
                    << ", senderIface=" << senderIface << ") BEGIN");

  NS_ASSERT (msg.GetVTime () > Seconds (0));
  LinkTuple *link_tuple = m_state.FindLinkTuple (senderIface);
  if (link_tuple == NULL)
    {
      LinkTuple newLinkTuple;
      // We have to create a new tuple
      newLinkTuple.neighborIfaceAddr = senderIface;
      newLinkTuple.localIfaceAddr = receiverIface;
      newLinkTuple.symTime = now - Seconds (1);
      newLinkTuple.time = now + msg.GetVTime ();
      link_tuple = &m_state.InsertLinkTuple (newLinkTuple);
      created = true;
      mobilidade = true;
      NS_LOG_LOGIC ("Existing link tuple did not exist => creating new one");
    }
  else
    {
      NS_LOG_LOGIC ("Existing link tuple already exists => will update it");
      updated = true;
    }



  link_tuple->asymTime = now + msg.GetVTime ();
  for (std::vector<olsr::MessageHeader::Hello::LinkMessage>::const_iterator linkMessage =
         hello.linkMessages.begin ();
       linkMessage != hello.linkMessages.end ();
       linkMessage++)
    {
      int lt = linkMessage->linkCode & 0x03; // Link Type
      int nt = (linkMessage->linkCode >> 2) & 0x03; // Neighbor Type

#ifdef NS3_LOG_ENABLE
```

```
      const char *linkTypeName;
      switch (lt)
        {
        case OLSR_UNSPEC_LINK: linkTypeName = "UNSPEC_LINK"; break;
        case OLSR_ASYM_LINK: linkTypeName = "ASYM_LINK"; break;
        case OLSR_SYM_LINK: linkTypeName = "SYM_LINK"; break;
        case OLSR_LOST_LINK: linkTypeName = "LOST_LINK"; break;
        default: linkTypeName = "(invalid value!)";
        }

      const char *neighborTypeName;
      switch (nt)
        {
        case OLSR_NOT_NEIGH: neighborTypeName = "NOT_NEIGH"; break;
        case OLSR_SYM_NEIGH: neighborTypeName = "SYM_NEIGH"; break;
        case OLSR_MPR_NEIGH: neighborTypeName = "MPR_NEIGH"; break;
        default: neighborTypeName = "(invalid value!)";
        }

      NS_LOG_DEBUG ("Looking at HELLO link messages with Link Type "
                    << lt << " (" << linkTypeName
                    << ") and Neighbor Type " << nt
                    << " (" << neighborTypeName << ")");
#endif // NS3_LOG_ENABLE

      // We must not process invalid advertised links
      if ((lt == OLSR_SYM_LINK && nt == OLSR_NOT_NEIGH) ||
          (nt != OLSR_SYM_NEIGH && nt != OLSR_MPR_NEIGH
           && nt != OLSR_NOT_NEIGH))
        {
          NS_LOG_LOGIC ("HELLO link code is invalid => IGNORING");
          continue;
        }

      for (std::vector<Ipv4Address>::const_iterator neighIfaceAddr =
```

```
          linkMessage->neighborInterfaceAddresses.begin ();
         neighIfaceAddr != linkMessage->neighborInterfaceAddresses.end ();
         neighIfaceAddr++)
       {
         NS_LOG_DEBUG ("   -> Neighbor: " << *neighIfaceAddr);
         if (*neighIfaceAddr == receiverIface)
           {
             if (lt == OLSR_LOST_LINK)
               {
                NS_LOG_LOGIC ("link is LOST => expiring it");
                link_tuple->symTime = now - Seconds (1);
                updated = true;
               }
             else if (lt == OLSR_SYM_LINK || lt == OLSR_ASYM_LINK)
               {
                NS_LOG_DEBUG (*link_tuple << ": link is SYM or ASYM => should becomeSYM"
                              " (symTime being increased to " << now + msg.GetVTime ());
                link_tuple->symTime = now + msg.GetVTime ();
                link_tuple->time = link_tuple->symTime + OLSR_NEIGHB_HOLD_TIME;
                updated = true;
               }
             else
               {
                 NS_FATAL_ERROR ("bad link type");
               }
             break;
           }
         else
           {
             NS_LOG_DEBUG ("     \\-> *neighIfaceAddr (" << *neighIfaceAddr);
           }
       }
     NS_LOG_DEBUG ("Link tuple updated: " << int (updated));
   }
link_tuple->time = std::max (link_tuple->time, link_tuple->asymTime);
```

```
  if (updated)
    {
      LinkTupleUpdated (*link_tuple, hello.willingness);
    }

  // Schedules link tuple deletion
  if (created && link_tuple != NULL)
    {
    LinkTupleAdded (*link_tuple, hello.willingness);
    m_events.Track (Simulator::Schedule (DELAY (std::min (link_tuple->time,
link_tuple->symTime)),
                                          &RoutingProtocol::LinkTupleTimerExpire, this,
                                          link_tuple->neighborIfaceAddr));
    }
  NS_LOG_DEBUG ("@" << now.GetSeconds () << ": Olsr node " << m_mainAddress
                    << ": LinkSensing END");
}

///
/// \brief Updates the Neighbor Set according to the information contained in
/// a new received HELLO message (following RFC 3626).
```