Algoritmos Distribuídos para Roteamento em Redes Tolerantes a Atrasos e Desconexões

Anna Dolejsi Santos Gabriel Argolo M. Rocha Lúcia M. A. Drummond
Melba Lima Gorza
Departamento de Ciência da Computação - Universidade Federal Fluminense
Niterói-RJ, Brasil
{annads,grocha,lucia,mgorza}@ic.uff.br

Abstract

Redes tolerantes a atrasos e desconexões (DTNs) são uma classe de redes que apresentam frequentes partições e longos atrasos. Redes com estas características possuem uma variedade de aplicações como comunicações entre dispositivos com restrições de energia, comunicações rurais, submarinas e interplanetárias. Neste trabalho, nós propomos dois algoritmos distribuídos para roteamento em redes DTN previsíveis, que consideram o menor número de saltos e o tempo de chegada mais cedo ao destino. Eles produzem como saída uma tabela de roteamento para cada nó da rede. Durante a fase de construção da tabela, são realizadas críticas nos intervalos de tempo dos enlaces adjacentes visando minimizar a quantidade de mensagens e bits enviados na rede. Os algoritmos foram avaliados experimentalmente para verificar a redução do número de mensagens trocadas na versão com crítica quando comparada à versão que não realiza crítica nos intervalos. Os resultados mostraram que a crítica reduz significativamente a quantidade de mensagens trafegadas, obtendo um ganho de até 88% para determinadas topologias de rede.

1. Introdução

Redes de transmissão de dados sem fio tem se popularizado e tornado acessíveis ambientes inicialmente desprovidos de comunicação como, por exemplo, áreas rurais e campos de batalha. Essas redes podem dispor de nós móveis que se comunicam diretamente ou através de nós intermediários que atuam como roteadores. Desconexões frequentes causadas pelo deslocamento desses dispositivos, atrasos longos e variáveis originados pelo tempo dos pacotes nas filas e a possível inexistência de um caminho fim-a-fim são problemas encontrados nesses sistemas móveis. Redes que consideram estas características são denominadas redes tolerantes a atrasos e desconexões - DTN (*Delay-Disruption Tolerant Network*).

As redes DTN diferem da tradicional Internet pois nesta assume-se que a conectividade é ininterrupta e a taxa de perda de pacotes e o retardo de propagação são relativamente baixos. Consequentemente os protocolos desenvolvidos para Internet cabeada não são eficientes para transmissão de dados em redes DTN. Como encontrar o destino, como rotear para o destino e como assegurar uma comunicação robusta diante da constante mudança da topologia são desafios que se apresentam nessas redes.

Redes DTN podem ser classificadas como previsíveis, também conhecida como redes com contatos programados, e imprevisíveis. Na primeira, a variação da topologia ao longo do tempo é conhecida antecipadamente. As redes de satélites são um exemplo, onde as trajetórias dos satélites são previamente programadas. Já nas imprevisíveis, não se conhece de antemão as alterações que podem ocorrer na topologia como, por exemplo, nas redes ad-hoc onde os nós podem se mover arbitrariamente ao longo do tempo [11].

Vrios trabalhos relacionados à modelagem de redes DTN previsíveis podem ser encontrados na literatura. Em [3], Ferreira descreve que essas redes podem ser representadas por grafos onde os intervalos de tempo de disponibilidade dos enlaces previamente definidos são associados a cada aresta. Essas estruturas de dados são denominadas pelo autor como grafos temporais ou evolutivos.

Algoritmos centralizados para roteamento em redes DTN previsíveis foram propostos em [1, 2, 7]. Três algoritmos denominados *foremost journey*, *shortest journey* e *fastest journey* foram apresentados por Bui et al. em [1]. Eles objetivam encontrar, respectivamente, a "jornada mais cedo", ou seja, a jornada onde o instante de tempo de chegada da mensagem no nó de destino é o menor possível, a "jornada com menor número de saltos" e a "jornada mais rápida", ou seja, a que apresenta a menor diferença entre o instante de tempo de chegada da mensagem no destino e

o instante de envio da mesma. Os três algoritmos utilizam um grafo temporal como entrada para suas execuções. Conforme apresentado pelos autores, o conceito de jornada em um grafo temporal é o mesmo que o de um caminho entre uma origem e um destino num grafo convencional. A diferença é que uma jornada considera os instantes de tempo de disponibilidade das arestas, com o intuito de nunca permitir a determinação de rotas utilizando instantes de tempo que existiram apenas no passado.

Chen em [2] propõe um algoritmo para roteamento em redes de satélites executado em duas etapas. Inicialmente, a coleta da informação de um grupo de satélites de baixa altitude (LEOS) é feita por um satélite de médio alcance (MEO). Em seguida, os diversos MEOS trocam as informações obtidas entre si e com a informação global disponível calculam as tabelas de roteamento e as redistribuem para os LEOS.

Outros algoritmos de roteamento propostos em [7] levam em consideração informações que são conhecidas sobre o estado da rede previamente. Este trabalho relata que o desempenho no roteamento das mensagens apresenta melhor resultado nos algoritmos que dispõem de mais conhecimento da rede. Exemplos de informações relevantes aos algoritmos descritos pelos autores são a demanda de tráfego de cada nó da rede e a capacidade de armazenamento das mensagens nos nós intermediários.

Mais experimentos para avaliar o roteamento em redes DTN previsíveis foram realizados em [5]. Para isso, o algoritmo *foremost journey* [1] foi comparado com tradicionais protocolos de roteamento utilizados em redes ad-hoc como, por exemplo, o AODV [13]. Questões relacionadas à quantidade de tempo necessária para entrega das mensagens e a proporção de pacotes perdidos em relação aos enviados foram analisadas.

Embora as tabelas de roteamento possam ser construídas de forma centralizada, essa abordagem apresenta inconvenientes. A necessidade de manter nos nós a informação global sobre o estado da rede torna-se difícil à medida que o tamanho da rede aumenta [12]. Logo, o projeto de algoritmos distribuídos para construção de tabelas de roteamento torna-se uma alternativa interessante.

Apresentamos neste trabalho dois algoritmos distribuídos para roteamento em redes DTN previsíveis. Em ambos, cada nó conhece apenas o custo e os intervalos de disponibilidade dos enlaces adjacentes a ele, ou seja, cada nó tem informações apenas da vizinhança e não da rede toda, fazendo com que um algoritmo centralizado necessite que todas as informações da rede sejam enviadas para um nó centralizador antes da execução do mesmo. Em um dos algoritmos, o *Distributed Shortest Journey*, cada nó calcula a tabela de roteamento dele para todos os outros considerando o menor número de saltos. No outro, denominado *Distributed Foremost Journey*, as tabelas são construídas objetivando a chegada mais cedo da informação ao nó de destino. Em ambos os algoritmos a construção da tabela em cada nó é realizada à medida que os enlaces para os nós vizinhos tornam-se disponíveis e novas informações sobre o estado da rede são obtidas. O projeto desses dois algoritmos distribuídos foram baseados, respectivamente, nos algoritmos sequenciais *shortest journey* e *foremost journey* propostos em [1]. No entanto, filtros para realizar a crítica dos intervalos adjacentes com intuito de reduzir a quantidade de informação trocada na rede foram incluídos, e uma estrutura otimizada da tabela de roteamento foi desenvolvida.

Neste trabalho uma tabela de roteamento é construída de forma distribuída em cada nó da rede. Esta tabela mantém, para cada nó de destino, uma lista ordenada dos intervalos de tempo de disponibilidade dos enlaces adjacentes. Cada intervalo é único na lista e determina qual vizinho deve receber a mensagem para posterior encaminhamento para cada nó de destino. Dessa forma, a melhor jornada, caso exista ao menos uma, é encontrada na tabela para qualquer instante de tempo que seja necessário enviar uma mensagem para um destino.

Dois algoritmos foram projetados de modo que os nós realizem críticas nos intervalos de tempo dos enlaces adjacentes visando minimizar a quantidade de mensagens e bits enviados durante a fase de construção da tabela de roteamento. Duas versões desses algoritmos sem a utilização dessas críticas também foram implementadas para efeito de avaliação de desempenho. Nota-se que para esses algoritmos que não aplicam nenhum tipo de filtro no envio de mensagens tem-se na realidade o envio de mensagens que poderiam ser descartadas pela origem. Os tempos necessários para obter as tabelas de roteamento estáveis também foram analisados para todos os algoritmos.

O restante deste trabalho está organizado da seguinte forma. Na Seção 2 apresentamos o modelo e as definições utilizadas para o projeto e avaliação dos algoritmos. Logo após, na Seção 3, os algoritmos *Distributed Foremost Journey* e *Distributed Shortest Journey* para roteamento em redes DTN são descritos e analisados. Os resultados experimentais para análise do comportamento desses algoritmos são apresentados e criticados na Seção 4. Terminamos este trabalho na Seção 5 com as conclusões encontradas e propostas para trabalhos futuros.

2. Modelo e Definições

Neste trabalho foi adotado o modelo de grafos temporais para representação de redes DTN previsíveis proposto em [3]. Seja uma sequência ordenada de subgrafos $\zeta_G = G_1, G_2, \ldots, G_T$ de $G = (V, E)$, tal que $\bigcup_{i=1}^{T} G_i = G$. Seja $\zeta_T = t_1, t_2, \ldots, t_T$ uma sequência ordenada de instantes de tempo. Então o sistema $\delta = (G, \zeta_G, \zeta_T)$ onde cada G_i é o subgrafo existente no período $[t_i, t_{i+1}[$ é denominado

um grafo temporal. Seja $E_\delta = \bigcup E_i$ e $V_\delta = \bigcup V_i$. Denota-se $|V_\delta| = N$ e $|E_\delta| = M$.

Pode-se representar um grafo temporal através de um conjunto de vértices e arestas, como em um grafo usual, adicionando-se às arestas etiquetas com os índices correspondentes aos instantes de tempo em que o enlace é válido. Os instantes de tempo de disponibilidade da aresta que estiverem dispostos em sequências ordenadamente crescentes e contínuas podem ser agrupados, gerando assim intervalos de tempo. Dessa forma, evita-se a discriminação de todos os instantes de tempo em que a aresta está válida, o que pode ser relativamente grande para determinados sistemas. Chamamos de *up* o instante de tempo de início do intervalo, e *down* quando o mesmo termina.

Em redes DTN as desconexões podem fazer com que uma rota fim-a-fim entre dois nós nunca seja encontrada. No entanto, se os nós intermediários armazenarem as mensagens e as encaminharem em momentos adequados, estas podem ser entregues no destino. Para isso, podem ser utilizadas jornadas, que são rotas construídas levando-se em consideração as restrições de tempo de existência dos enlaces. Define-se jornada como uma tupla $J = (R, R_T)$ onde R é uma rota $R = e_1, e_2, \ldots, e_k$ com $e_i \in E_{G_i}$, sendo G_i um subgrafo de G, mapeando todos os enlaces que a mensagem deve percorrer para chegar ao destino, e $R_T = t_1, t_2, \ldots, t_k$, onde $t_i > 0$, é a sequência de instantes de tempo indicando quando cada enlace da rota deve ser percorrido. Para cada aresta e_i de uma jornada tem-se um custo $c(v_i, v_{i+1})$ associado para transmitir a mensagem do vértice v_i para o v_{i+1}. Jornadas representam para grafos temporais o mesmo que rotas para um grafo tradicional. No entanto, em uma jornada é levada em consideração a restrição de que o próximo enlace não seja um enlace que só tenha existido em um subgrafo passado. A partir dos conceitos de grafo temporal e jornadas podem ser construídos algoritmos que utilizam diferentes métricas como objetivo.

Uma arquitetura para DTN com contatos programados ou previsíveis deve considerar sincronismo de tempo entre os nós da rede [14]. Em nosso trabalho, denominamos um instante de tempo como pulso. Então, a cada pulso p_i existe um subgrafo G_i associado. Conforme será verificado na próxima seção, os nós da rede consideram essa informação para controle do tempo e verificação de quando cada enlace adjacente está ativo para comunicação.

Consideramos neste trabalho, para efeito de construção das tabelas de roteamento, redes DTN com propriedades cíclicas, ou seja, após o último instante t_T, a contagem do tempo é reiniciada e figura-se novamente a representação da rede através do grafo G_1 relativo ao instante t_1. Definimos então que um ciclo corresponde ao processamento de todos os instantes de tempo entre $[t_1, t_T[$. Temos então que a quantidade de ciclos necessária para determinação das tabelas de roteamento finais é igual a quantidade de reinicializações da contagem do tempo mais um. Como será visto nos experimentos descritos na Seção 4 a construção das tabelas de roteamento pode necessitar de mais ciclos para ser concluída dependendo da topologia de rede.

Apesar das tabelas de roteamento necessitarem, em certas ocasiões, de mais de um ciclo para serem determinadas, assumimos no projeto dos algoritmos que o instante t_1 de um ciclo cic_k está no passado em relação ao instante t_T do ciclo cic_{k-1}.

Nos algoritmos distribuídos propostos cada nó conhece apenas os instantes de tempo em que os enlaces com o seus vizinhos estão ativos. Cada nó da rede executa o mesmo algoritmo que envia mensagens nos canais adjacentes, espera por mensagens entrantes e realiza o processamento. Como o objetivo desse trabalho não é avaliar o comportamento do congestionamento das filas nos nós intermediários, não foram levados em consideração os tempos de transmissão e processamento, assim como a capacidade de armazenamento em cada nó. Assumimos que todos os nós possuem identificações distintas e que os canais de comunicação são FIFO.

3. Descrição dos Algoritmos Distribuídos

Nesta seção apresentamos o algoritmo síncrono *Distributed Shortest Journey* que determina as menores jornadas entre todos os nós no grafo temporal associado a $G = (V, E)$. A distância é medida em número de saltos. Um algoritmo síncrono é aquele que possui uma base de tempo global, ou seja, o tempo é dividido em instantes de tempo. Em nosso sistema um ciclo é dividido em instantes de tempo denominados pulsos. Em cada pulso, quando necessário, os nós enviam mensagens para os vizinhos que estão com enlace disponível e processam as mensagens recebidas. Consideramos que a comunicação é realizada dentro de um pulso e que os atrasos de transmissão e as perdas das mensagens são desprezíveis.

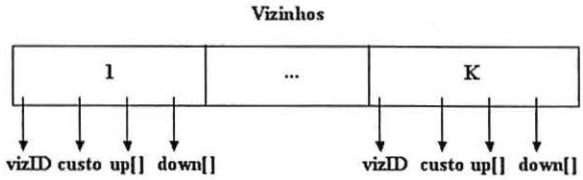

Figura 1. Vetor de Vizinhos

As estruturas de dados usadas pelo algoritmo são o vetor de vizinhos (Figura 1), a tabela de roteamento (Figura 2), e a estrutura das mensagens de atualização (Figura 3).

O vetor de vizinhos, que denominamos de vecInter, armazena as informações iniciais conhecidas pelo nó. Cada posição do vetor contém os dados de um nó vizinho, ou

seja, sua identificação, o custo e os intervalos de disponibilidade do enlace adjacente. Essas informações são representadas pelos campos {*vizID, custo, up[1], up[2], ..., up[max], down[1], down[2], ..., down[max]*}, onde *max* é o número máximo de intervalos de tempo disponíveis.

Cada nó $n_i \in N$ troca mensagens com seus vizinhos anunciando o conjunto de nós que ele alcança. As informações enviadas são as identificações dos nós alcançáveis, o pulso corrente, os intervalos de tempo possíveis para o envio de mensagens para cada nó, o custo, o número de saltos associados a cada intervalo de tempo e o número do ciclo corrente (level). Essas informações são armazenadas na estrutura da mensagem, que chamamos de $setType$. Veja Figura 3.

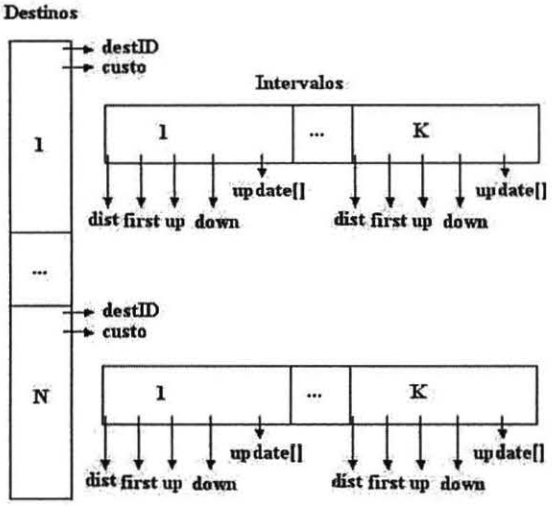

Figura 2. Tabela de Roteamento

Ao receber mensagens dos vizinhos cada nó atualiza suas rotas na tabela de roteamento marcando no campo *update* quais atualizações foram feitas e ainda não encaminhadas para os demais vizinhos. Esta tabela armazena, para cada nó alcançável, um vetor com intervalos de tempo. Cada posição do vetor de intervalos indica o início e o fim do intervalo, assim como o a distância até o destino e para qual vizinho encaminhar a mensagem. Veja Figura 2.

O algoritmo foi dividido em três partes. A primeira (Algoritmo 1), descreve as variáveis utilizadas e suas inicializações. Em seguida, é apresentada a função vecInter no Algoritmo 2, que tem como objetivo comparar as interseções dos intervalos adjacentes e, consequentemente, não enviar informações desnecessárias para os vizinhos, que será referenciado no decorrer do texto como *filtro*. Por último, é apresentado o Algoritmo 3, que possui dois tipos de eventos dependendo da entrada recebida. Neste procedimento, é realizada a crítica dos intervalos adjacentes e envio

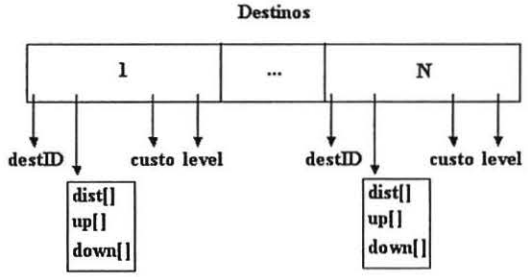

Figura 3. Estrutura das mensagens

de mensagens, quando necessário para os vizinhos. Durante a execução, os nós recebem mensagens dos vizinhos, atualizam a tabela de roteamento e enviam as atualizações para os demais vizinhos. As mensagens trocadas são denominadas SET e seu formato é do tipo $setType$ apresentado anteriormente.

O filtro também verifica para cada nó n_i se todos os intervalos de disponibilidade do enlace de um vizinho n_j estão no passado em relação a todos os intervalos do enlace de um outro vizinho n_k. Em caso positivo, isso significa que qualquer mensagem de atualização de rotas enviada de n_j para n_i não precisa ser encaminhada para n_k, pois o enlace (n_i, n_j) nunca estará ativo para o nó n_k e não poderá fazer parte de nenhuma jornada em n_k. Com o intuito de reduzir o processamento da crítica dos intervalos definimos a matriz $permut[x][y]$, que indica se uma mensagem recebida do vizinho x é possível de ser encaminhada para o vizinho y (linha 9 do Algoritmo 2). Cada nó mantém então nessa estrutura de dados as combinações de vizinhança possíveis.

Algoritmo 1 : Variáveis

1 $p_i = 0$;
2 $level_i = 0$;
3 $initiated_i = false$;
4 $SET_i[j] = nil$ para todo $j \in Neig_i$;
5 $valida_i$;
6 $found_i$;
7 $permut_i[j][k] = false$ para todo $j, k \in Neig_i$;
8 $vecInter_i[j] = vizID, custo, up[], down[]$ para todo $n_j \in Neig_i$;
9 $routeTable_i[j][k] = nil$ para todo $j \in N$ e todo intervalo k

Como descrito no Algoritmo 3, em cada pulso p ($p > 0$), os nós pertencentes ao subgrafo G_p verificam se os enlaces adjacentes estão ativos. Quando um enlace acabou de ficar disponível, o nó constrói a sua mensagem SET, que contém as informações de todos os seus vizinhos, e a envia através do enlace adjacente (linhas 1-9). Nos demais instantes de tempo do intervalo de disponibilidade,

o nó somente envia mensagem ao vizinho caso uma rota seja descoberta ou modificada. Após enviar suas mensagens, o nó processa as mensagens recebidas, podendo gerar modificações na tabela de roteamento. Uma nova rota pode ser inserida, ou uma rota existente pode ser modificada se o número de saltos existente na tabela para alcançar um destino for maior do que o valor recebido. Sempre que a tabela de roteamento é modificada o nó marca a posição que foi atualizada (linhas 10-29).

Algoritmo 2 : Testa_Inter($SET_i, j, level$)

```
1  Para cada (up1, down1) ϵ vecInter_i[j]
   tal que down1 − up1 ≥ vecInter_i[j].custo
2    found_i = false;
3    Para cada (up2, down2) ϵ vecInter_i[k]
     tal que k ϵ Neig_i e k ! = j
4      Se( (notfound) and (up2 +
       vecInter_i[k].custo ≤ up1) or (down1
       ≥ up2 + vecInter_i[j].custo +
       vecInter_i[k].custo) )
5        found_i = true;
6    Se ( found_i == true )
7      SET_i[k] = SET_i[k] ∪ {
       vecInter_i[j], level }
8    Senão
9      permut[j][k] = true
```

Essas ações são executadas até que os nós completem o ciclo, ou seja, não exista mais nenhum subgrafo em G para ser processado. Um novo ciclo pode ser iniciado caso exista algum nó que possua alguma atualização para ser enviada. Desta forma, o algoritmo reinicia o processamento dos subgrafos ζ_G, mantendo as alterações da tabela de roteamento dos nós. O algoritmo finaliza quando não existe mais nenhuma atualização a ser enviada. O resultado obtido é a tabela de roteamento em cada nó do grafo.

A sequência de subgrafos $G_1, G_2, \ldots, G_T$ associada ao grafo evolutivo $G = (V, E)$ pode ser executada várias vezes, gerando assim vários ciclos de execução. O número máximo de ciclos é da ordem do diâmetro do grafo $G = (V, E)$. Isso acontece devido à disposição e desconexão dos intervalos de tempo nos enlaces. Para uma topologia de barramento, por exemplo, onde os nós estão dispostos em linha, temos que os nós das extremidades possuem apenas um vizinho e os nós do meio possuem dois. Assumindo que os intervalos das arestas são totalmente desconexos e dispostos de forma crescente no barramento, temos que cada nó n_k conhecerá no primeiro ciclo as jornadas de no máximo dois saltos, ou seja, as jornadas que o conectam ao vizinho n_{k+1} e ao n_{k+2}. Os intervalos entre n_{k+1} e n_{k+2} são enviados para o n_k pelo nó n_{k+1}. Analogamente, o nó n_k conhecerá no segundo ciclo as jornadas para alcançar o nó n_{k+3}, e assim por diante. Consequentemente, serão necessários então $N - 2$ ciclos para que o primeiro nó (n_0) do barramento conheça o último nó (n_N), para $N > 3$.

Algoritmo 3 : Distributed Shortest Journey

```
1  Entrada:
2    p > 0 , MSG_i(p) = 0;
3  Ação:
4    Para todo enlace(i, j) disponível no
     pulso p
5      Se( initiated == false )
6        initiated = true;
7        Para todo j ϵ Neig_i
8          Testa_Inter( SET_i, j, level_i );
9          Envia SET_i[k] para n_k para todo
           k ϵ Neig_i;
10 Entrada:
11   p > 0 , MSG_i(p) tal que
     origem_i(SET_j) = (n_i, n_j) para
     SET_j ϵ MSG_i(p) );
12 Ação:
13   Para todo SET_j ϵ MSG_i(p)
14     Para todo k ϵ SET_j
15       Para todo (custo, dist, up, down) ϵ
         SET_j[k]
16         Para todo t ϵ routeTable_i[k] tal
           que up ≤ t ≤ down
17           Se ( routeTable_i[k][t].dist > dist
             + 1
18             routeTable_i[k][t].dist = dist +
               1;
19             routeTable_i[k][t].first = n_j;
20             Para todo j ϵ Neig_i
21               routeTable_i[k][t].update[j] =
                 true;
22   Para todo n_k ϵ N
23     Para todo t ϵ routeTable_i[k][t]
24       Para todo n_j ϵ Neig_i
25         Se( routeTable_i[k][t].update[j]
           == true )
26           Se( (n_i, n_j) está disponível )
27             SET_i[j] = SET_i[j] ∪
               {routeTable_i[k][t], level};
28             routeTable_i[k][t].update[j] =
               false;
29   Envia SET_i[j] para n_j;
```

O algoritmo executa vários ciclos até que as tabelas de roteamento estejam estáveis, ou seja, até que não haja novas atualizações a serem enviadas. Para diminuir o tamanho e o número de mensagens trocadas, o algoritmo faz uma crítica entre os intervalos de tempo de seus enlaces adjacentes. Um nó n_i só envia para seu vizinho n_k a informação de que alcança um outro vizinho n_j se o *down* do intervalo de tempo do enlace (n_i, n_j) for maior ou igual ao *up* do intervalo de tempo do enlace (n_i, n_k) mais os custos dos enlaces (n_i, n_k) e (n_i, n_j). Veja Algoritmo 2. Desta forma, intervalos de tempo que só existiram no passado não são enviados aos vizinhos, ou seja, os enlaces pertencentes a subgrafos que já foram processados anteriormente não são considerados.

Ao final da execução cada nó deve ter recebido a identificação de todos os outros nós da rede. Temos ainda que a identificação de cada nó é passada duas vezes em cada aresta. Logo, a quantidade de mensagens enviadas é de $2MN$ e a complexidade é de $O(MN)$. Considerando que cada enlace possua I intervalos, que cada intervalo seja mapeado em L bits, e que a identificação dos nós seja expressa em $log(N)$ bits, a complexidade de bits trafegados é de $O(MNILlog(N))$.

Nota-se que a quantidade de intervalos em cada enlace não altera a complexidade de mensagem, mas influencia no volume de bits trafegados na rede. Como será verificado na avaliação experimental do algoritmo, os filtros aplicados mostram uma redução significativa na quantidade de mensagens enviadas e no volume de bits trafegados dependendo da interseção e disposição dos intervalos nos enlaces, assim como da topologia da rede.

Como a quantidade de ciclos é da ordem do diâmetro do grafo e cada ciclo processa T pulsos, temos que a complexidade de tempo é da ordem de $O(NT)$.

Além do algoritmo *Distributed Shortest Journey*, implementamos também o algoritmo *Distributed Foremost Journey*. Esse considera, como métrica para construção da tabela de roteamento, que o tempo de chegada da mensagem no destino deve ser o mais cedo possível. Utilizamos os mesmos eventos associados a cada pulso na construção do programa principal e a mesma estrutura de dados do algoritmo *Distributed Shortest Journey*. Porém, ao invés da tabela de roteamento armazenar a distância em saltos, ela guarda o tempo de chegada mais cedo para alcançar o destino.

Temos, então, que a linha 17 do Algoritmo 3 pode ser alterada para que o campo *dist* da tabela de roteamento considere como distância entre a origem o destino o tempo de chegada e não o número de saltos. No entanto, nota-se que o tempo de chegada está diretamente ligado ao instante de tempo em que a mensagem é enviada pela origem. Considere que uma mensagem é enviada de um nó n_i para seu vizinho n_j em um instante t_x. O nó n_j encaminha posteriormente essa mensagem em um instante t_y para seu vizinho n_k. Nesse caso, temos então duas situações. Se $t_y = t_x + c(n_i, n_j)$, quer dizer que a mensagem chegou no nó intermediário n_j e já foi encaminhada para n_k, pois o enlace (n_j, n_k) já estava ativo naquele momento. Por outro lado, se $t_y > t_x + c(n_i, n_j)$, temos então que a mensagem chegou no nó n_j e precisou ser armazenada para posterior encaminhamento. Assumindo ainda que existe um intervalo entre $t_x + c(n_i, n_j)$ e t_y, temos que a mensagem poderá ser enviada em qualquer instante de tempo desse intervalo sem que o tempo de chegada no nó seguinte seja alterado. Ou seja, um controle de quando a mensagem chega em cada nó intermediário também precisa ser realizado pelo algoritmo *Distributed Foremost Journey*.

4. Avaliação Experimental

Para avaliar os algoritmos foram utilizados computadores conectados em uma LAN. Cada máquina possui processador Intel Pentium IV 2.8GHz e memória RAM de 512MB, rodando o sistema operacional Ubuntu V3. Os algoritmos foram implementados em linguagem C utilizando a biblioteca MPI para troca de mensagens.

Algumas topologias de rede foram pesquisadas com o intuito de identificar um padrão comportamental na troca de mensagens entre os dispositivos móveis. Para cada uma delas também foi analisado o número de ciclos necessários para determinação da tabela de roteamento final. Elas foram representadas por grafos com 8, 16, 32 e 64 vértices. Para cada vértice do grafo foi criado um processo, o qual representa um nó da rede. Para cada aresta do grafo foi atribuído um custo unitário.

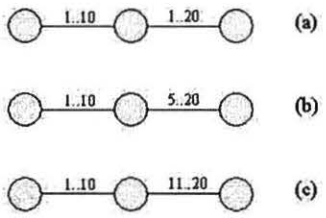

Figura 4. Interseções entre intervalos adjacentes

Variações relativas à disposição dos intervalos de tempo nos enlaces foram realizadas para verificar em quais configurações os filtros para redução do tráfego de mensagens trazem mais benefícios. Para isso, foram utilizados diferentes níveis de interseção entre os intervalos de enlaces adjacentes a um mesmo nó. Primeiro, cada intervalo continha ou estava contido em um outro intervalo adjacente (Figura 4(a)). Em seguida, os intervalos foram dispostos tal que a interseção com os intervalos adjacentes fosse apenas parcial (Figura 4(b)). Por último, os intervalos adja-

centes estavam totalmente desconexos, ou seja, sem nenhuma interseção (Figura 4(c)).

Inicialmente, foi analisada uma topologia de barramento, onde os nós foram colocados em série. Os intervalos foram então dispostos considerando as combinações de níveis de interseção descritos acima. Constatou-se então que o padrão que apresenta maior redução de tráfego nos enlaces quando o filtro é aplicado é quando os intervalos adjacentes estão totalmente desconexos e, alternadamente, com valores maiores e menores. Isso acontece porque cada nó enviará no enlace que possui o menor intervalo de tempo uma mensagem com a informação do intervalo de maior tempo. No enlace que possui o maior intervalo tempo, nenhuma mensagem será enviada, dado que o intervalo de menor tempo estará no passado em relação ao intervalo adjacente. Ou seja, até $(M/2) + 1$ enlaces podem ficar sem receber mensagens, onde M é o número de enlaces.

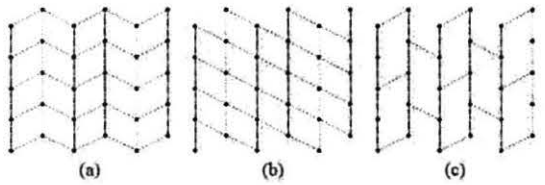

Figura 5. Topologia de redes de satélites

Com relação ao tempo para obtenção da tabela de roteamento final, a configuração que apresentou o maior número de ciclos nessa topologia foi com os intervalos também desconexos, mas com valores em sentido crescente.

O mesmo procedimento foi realizado com estruturas de árvore e estrela. Novamente, os maiores ganhos percebidos com a aplicação dos filtros foram encontrados quando tem-se intervalos desconexos. No entanto, o resultado obtido com essas duas topologias foi inferior ao encontrado na topologia de barramento. A explicação para isso é que na topologia de barramento existem no máximo dois enlaces adjacentes a um nó e, dessa forma, até metade dos enlaces adjacentes pode ficar sem receber mensagens quando utiliza-se o filtro.

Por último, foram analisadas topologias que representam padrões de conectividade encontrados em redes de satélites como, por exemplo, o sistema Iridium [6, 9]. Uma análise sobre a modelagem e roteamento nessas redes pode ser encontrada em [4]. Neste trabalho são apresentados três padrões que podem ser utilizados para conectar satélites alocados em órbitas adjacentes. Nos padrões "W" (Figura 5(a)) e "inclinado" (Figura 5(b)), cada satélite dispõe de quatro enlaces para se conectar aos outros, enquanto que o padrão apresentado na Figura 5(c) utiliza apenas três enlaces para comunicação.

A Tabela 1 apresenta o percentual de redução no número de mensagens enviadas resultante da aplicação da crítica nos intervalos dos enlaces adjacentes. Esses valores foram obtidos através da execução do algoritmo *Distributed Shortest Journey* utilizando grafos onde os intervalos foram dispostos, alternadamente, com valores maiores e menores. Para cada enlace foi associado apenas 1 intervalo de tempo de disponibilidade. Nota-se que o ganho permanece praticamente constante à medida que aumenta-se o número de nós da rede. A topologia de barramento chegou a reduzir 66.3% no número de mensagens para uma rede com 64 nós. O resultado mais tímido encontrado foi com a topologia de estrela composta de 8 nós, que diminuiu em 42.9% o número de mensagens.

Topologia	Nº de Nós			
	8	16	32	64
Barramento	62.5%	65.0%	65.9%	66.3%
W4	60.0%	58.1%	58.7%	59.2%
Inclinado	58.8%	55.4%	58.1%	58.7%
W3	54.0%	58.1%	59.2%	59.1%
Árvore	52.9%	52.4%	55.4%	56.3%
Estrela	42.9%	46.7%	48.4%	49.2%

Tabela 1. Avaliação do Filtro em relação ao Número de Vértices

Topologia	Nº de Intervalos				
	1	2	4	8	16
Barramento	66.3%	70.2%	75.9%	82.5%	88.8%
W4	70.4%	71.8%	73.8%	75.7%	77.0%
Inclinado	69.9%	71.7%	73.9%	75.9%	77.4%
W3	64.8%	66.7%	70.1%	73.5%	76.3%
Árvore	63.7%	64.4%	66.2%	68.0%	69.6%
Estrela	74.3%	56.5%	55.0%	53.5%	52.2%

Tabela 2. Avaliação do Filtro em relação ao Número de Intervalos

Além da redução na quantidade de mensagens trocadas, é importante também levar em consideração que o tamanho das mensagens pode variar, dado que um enlace pode apresentar mais de um intervalo de disponibilidade. Nota-se, no entanto, que se simplesmente aumentarmos a quantidade de intervalos em todos os enlaces adjacentes, temos que o percentual de redução da quantidade de bits trafegados não se modifica. Contudo, se mantivermos reduzida a quantidade de intervalos com instantes de tempo maiores e aumentarmos a quantidade de intervalos com instantes de tempo menores, temos que o filtro apresenta um desempenho melhor, pois impedirá que os intervalos menores que estão no passado em relação aos adjacentes sejam enviados adiante.

Para verificar o comportamento do filtro nessa situação, variamos em 1, 2, 4, 8 e 16 a quantidade de intervalos por enlace para as topologias já mencionadas. As execuções foram realizadas utilizando 64 nós. Os valores constantes na Tabela 2 mostram que o percentual de redução de bits trafegados chega a quase 90% para a topologia de barramento quando utilizados 16 intervalos. Já para as topologias que representam padrões de redes de satélites o ganho é inferior, mas atingindo até 76%.

5. Conclusões

Neste trabalho apresentamos como contribuição dois novos algoritmos distribuídos para roteamento em redes DTN previsíveis, denominados *Distributed Shortest Journey* e *Distributed Foremost Journey*. O primeiro calcula a tabela de roteamento de cada nó considerando o menor número de saltos e no último as tabelas são construídas objetivando a chegada mais cedo da informação ao nó de destino. Em ambos, a construção da tabela em cada nó é realizada à medida que os enlaces para os nós vizinhos tornam-se disponíveis. Para minimizar a quantidade de mensagens e bits enviados durante a fase de construção da tabela de roteamento, eles realizam críticas nos intervalos de tempo dos enlaces adjacentes.

Foi realizada uma análise de conectividade de algumas tolopogias, assim como da disposição dos intervalos de tempo nos enlaces. Para mensurar os ganhos obtidos pela aplicação do filtro, executamos os mesmo algoritmos sem sua utilização. A avaliação mostrou que filtros baseados na comparação dos intervalos podem propiciar ganhos significativos na redução do tráfego de informações na rede.

Como continuação desse trabalho, planejamos desenvolver algoritmos distribuídos que consideram os problemas de *multicast* e *broadcast* em redes DTN previsíveis, assim como estudar problemas de capacidade de armazenamento e sincronismo das jornadas. Outros pontos importantes a serem pesquisados também são a avaliação do desempenho dos algoritmos propostos neste trabalho com outras topologias, além da comparação deles com algoritmos já existentes.

References

[1] Bui-Xuan, B., Ferreira, A., Jarry, A., "Evolving Graphs and Least Cost Journeys in Dynamic Networks", in Proceedings of WiOpt - Modeling and Optimization in Mobile, Ad-Hoc, and Wireless Networks. INRIA Press, March 2003, pp. 141-150.

[2] Chen, C., "Advanced Routing Protocol for Satellite and Space Networks", Master's Thesis, Georgia Institute of Technology, 2005.

[3] Ferreira, A., "On Models and Algorithms for Dynamic Communication Networks: The Case for Evolving Graphs". In Proceedings of 4o recontres francophones sur les Aspects Algorithmiques des Télécommunications (ALGOTEL2002), pages 155-161, Mèze, France, 2002. INRIA Press.

[4] Ferreira, A., Galtier, J., Penna, P., "Topological Design, Routing and Hand-over in Satellite Networks", in Handbook of Wireless Networks and Mobile Computing, I. Stojmenovic, Ed. John Wiley and Sons, 2002, pp. 473-507.

[5] Goldman, A., Monteiro, J., Ferreira, A., "Performance Evaluation of Dynamic Networks using an Evolving Graph Combinatorial Model", 2006.

[6] Hubbel, Y. C., Sanders, L. M., "A Comparison of the IRIDIUM and AMPS systems", IEEE Network, 12(2):52-59, 1997.

[7] Jain, S., Fall, K., Patra, S., "Routing in a Delay Tolerant Network". In Proceedings of ACM SIGCOMM, 2004.

[8] Jones, E.P.C, Li, L., and Ward, A.A.S., "Practical Routing in Delay Tolerant Networks". In WDTN 2005: Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking. New York, NY, USA: ACM Press, 2005, pp. 237-243.

[9] Leopold, R. J., "Low-earth Orbit Global Celular Communications Network". In Proceedings of IEEE International Conference on Communications (ICC), pages 1108-1111, 1991.

[10] Merugu, S., Ammar, M., Zegura, E., "Routing in Space and Time in Networks with Predictable Mobility". Technical Report GIT-CC-04-7, Georgia Institute of Technology, 2004.

[11] Oliveira, C. T. and Duarte, O. C. M. B. "Uma Anlise da Probabilidade de Entrega de Mensagens em Redes Tolerantes a Atrasos e Desconexes", in XXV Simpsio Brasileiro de Redes de Computadores - SBRC'2007, pp. 293-305, Belm, PA, Brasil.

[12] Peleg, D., "Distributed Computing: a locality-sensitive approach", SIAM Monographs on Discrete Mathematics and Applications 5, 2000.

[13] Perkins, C. E., and Royer, E. M., "Ad-hoc On-demand Distance Vector Routing", in Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), February 1999, pp. 90100.

[14] "Delay-Tolerant Networking Architecture", RFC4838, 2007.