Universidade Federal Fluminense

GABRIEL ARGOLO MATOS ROCHA

Roteamento em Redes Tolerantes a Atrasos e Desconexões Previsíveis com Restrições de *Buffer* e Largura de Banda

NITERÓI

2009

GABRIEL ARGOLO MATOS ROCHA

Roteamento em Redes Tolerantes a Atrasos e Desconexões Previsíveis com Restrições de *Buffer* e Largura de Banda

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Processamento Paralelo e Distribuído.

Orientador:

Lúcia Maria de Assumpção Drummond

Coorientador:

Anna Dolejsi Santos

Universidade Federal Fluminense

NITERÓI

2009

Roteamento em Redes Tolerantes a Atrasos e Desconexões Previsíveis com Restrições de *Buffer* e Largura de Banda

GABRIEL ARGOLO MATOS ROCHA

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Aprovada por:

Prof. Lúcia Maria de Assumpção Drummond / Instituto de Computação - UFF (Presidente)

Prof. Anna Dolejsi Santos / Instituto de Computação - UFF

Prof. Celio Vinicius Neves de Albuquerque / Instituto de Computação - UFF

Prof. Eduardo Uchoa Barboza / Departamento de Engenharia de Produção - UFF

Prof. Alfredo Goldman vel Lejbman / Departamento de Ciência da Computação - USP

Niterói, setembro de 2009.

Para a família e os amigos, com todo o carinho de sempre.

Agradecimentos

Meus agradecimentos ao Jacques, que sempre esteve prestativo a resolver grandes problemas no laboratório, ao Prof Eduardo Uchoa, pelos ensinamentos sobre modelagem matemática, às Profs Lucia Drummond e Anna Dolejsi, pelos ensinamentos e por acreditar no potencial deste trabalho. Agradeço também à família e aos amigos que nunca desistiram de convidar-me para churrascos, festas e praias, mesmo depois de ouvir inúmeras vezes que eu não poderia devido aos testes restantes para conclusão desta dissertação de Mestrado.

Resumo

As Redes Tolerantes a Atrasos e Desconexões (DTN - *Delay-Disruption Tolerant Networks*) são caracterizadas pelas alterações na topologia da rede ao longo do tempo. Estas mudanças podem ocorrer devido ao deslocamento dos nós, à potência do sinal ou até mesmo à economia de energia. Em virtude das desconexões presentes nestas redes, é possível que não exista um caminho fim-a-fim entre os nós, o que torna o roteamento em DTNs um desafio.

O objetivo deste trabalho é atacar o problema de roteamento em DTNs. Isto é feito primeiramente através da elaboração de um modelo para estas redes. EM seguida, projetando algoritmos para construção de tabelas de roteamento e desenhando um algoritmo para encaminhamento das mensagens até os destinos. Por último, é elaborada uma formulação matemática em Programação Linear Inteira para obtenção da solução ótima de encaminhamento e realizados experimentos para avaliar o desempenho dos algoritmos propostos.

Três algoritmos para construção das tabelas de roteamento são propostos. O primeiro, denominado Distributed Shortest Journey (DSJ), gera as tabelas levando em consideração as jornadas com menor número de saltos até os destinos. O segundo, o Distributed Earliest Journey (DEJ), constrói as tabelas com jornadas onde o encaminhamento das mensagens é feito o mais cedo possível. O terceiro, o Distributed Fastest Journey (DFJ), considera as jornadas onde que o tempo de duração entre o envio da mensagem pela origem e o recebimento da mesma no destino é o menor possível. Estes algoritmos realizam um filtro nos instantes de tempo de disponibilidade dos enlaces adjacentes para evitar o envio desnecessário de mensagens de controle pelos canais de comunicação. Os experimentos mostraram que a aplicação deste filtro reduziu em torno de 67% a quantidade de mensagens de controle trocadas pelos nós para as topologias de rede avaliadas.

O algoritmo de encaminhamento proposto, o *Alternative Journey Routing Protocol* (AJRP), foi projetado para utilizar as tabelas de roteamento e entregar o maior número de mensagens possível até os destinos sem realizar nenhum tipo de replicação. Além disso, são levadas em consideração as restrições de largura de banda dos enlaces e de capacidade de armazenamento dos nós. Para reduzir a perda de mensagens, é proposto um mecanismo de escolha de jornadas alternativas que estejam menos congestionadas.

Para avaliar o AJRP, foram realizados experimentos submetendo uma baixa e alta carga na rede, onde o AJRP conseguiu entregar, respectivamente, cerca de 96% e 83% das mensagens, quando comparado com a solução ótima determinada. O AJRP também foi avaliado utilizando *traces* coletados através dos contatos entre ônibus da rede veicular *DieselNet*, onde os resultados foram comparados com os de outros algoritmos de roteamento de DTNs encontrados na literatura. Os resultados mostraram que o AJRP entregou em torno de 76% das mensagens e obteve desempenho superior a determinados algoritmos que consideram a replicação de mensagens.

Abstract

Delay-Disruption Tolerant Networks are characterized by frequent topology changes. These modifications can occur due to mobility, wireless range or even power management. Accordingly, there may not exist an end-to-end route between nodes, which makes routing in DTNs a challenge.

The goal of this work is to tackle the DTNs routing problem. This is done by designing a network model, proposing algorithms for building routing tables, implementing an algorithm in order to forward the messages to destinations, developing an Integer Linear Programming formulation to obtain the optimum routing solution, and evaluating these approaches.

Three algorithms for building routing tables are proposed. The first one, called Distributed Shortest Journey (DSJ), generates the tables with the journeys with minimum number of hops to reach the destinations. The second, named Distributed Earliest Journey (DEJ), builds the tables with journeys which the forwarding of messages can be done sooner. The last, called Distributed Fastest Journey (DFJ), takes into account only the journeys in which the elapsed times for delivering a message to the destination are the lowest. These algorithms perform a filtering process in the time intervals assigned to the adjacent links to avoid messages to be sent unnecessarily. For the network topologies used, the experiments showed that the application of the filter reduces the number of sent messages in about 67%.

The proposed algorithm *Alternative Journey Routing Protocol* (AJRP) was designed to use the routing tables and to deliver, using no replication mecanism, the maximum amount of messages to the destinations. In addition, the nodes' storage and edges' bandwidth constraints are considered. In order to reduce the message loss a mecanism for choosing alternative journeys in which the overload are lower is proposed.

AJRP was evaluated under low and high loads of messages in the network and delivered, respectively, 96% and 83% of the messages when compared to the optimum solution. The algorithm was also tested using *traces* of the vehicular network *DieselNet* and the results were compared with the ones obtained by other DTN routing algorithms found in literature. In this case the AJRP delivered 76% of messages and had a higher performance than some algorithms that use messages replication.

Palavras-chave

1. Redes Tolerantes a Atrasos e Desconexões - DTNs
2. Algoritmo Distribuído
3. Roteamento
4. Programação Linear Inteira

Glossário

DTN	:	Delay-disruption Tolerant Network;
FIFO	:	First-In-First-Out;
PL	:	Programação Linear;
PLI	:	Programação Linear Inteira;
DSJ	:	Distributed Shortest Journey;
DEJ	:	Distributed Earliest Journey;
DFJ	:	Distributed Fastest Journey;
AJRP	:	Alternative Journey Routing Protocol;
LEO	:	Low Earth Orbit;

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Redes Tolerantes a Atrasos e Desconexões (DTN - *Delay-disruption Tolerant Network*) possibilitam a transmissão de dados quando dispositivos móveis estão conectados intermitentemente. Neste tipo de rede, a comunicação entre os nós é feita diretamente ou através de nós intermediários que atuam como roteadores. A conectividade intermitente pode ser resultado da mobilidade dos nós, da potência de sinal ou até mesmo do gerenciamento de energia. DTNs são encontradas, por exemplo, em redes de sensores para monitoramento ecológico, na comunicação entre sistemas de satélites e em redes veiculares. Estas redes diferem da tradicional *Internet*, pois é assumido que esta última possui conectividade ininterrupta, além da baixa taxa de perda de pacotes e do baixo retardo de propagação. Partindo deste pressuposto, os protocolos desenvolvidos para *Internet* cabeada são ineficientes para transmissão de dados em DTNs.

DTNs podem ser classificadas como previsíveis, também conhecidas como redes com contatos programados, imprevisíveis e probabilísticas. Na primeira, a variação da topologia ao longo do tempo é conhecida antecipadamente. As redes de satélites do tipo LEO (*Low Earth Orbit*) são um exemplo, onde as trajetórias dos satélites são previamente programadas. Na segunda, 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 [Merugu et al. 2004, Oliveira e Duarte 2007]. Já nas probabilísticas, apesar de não conhecer *a priori* exatamente quando ocorrerá futuros contatos e a duração dos mesmos, é possível obter uma aproximação destes valores através da aplicação de métodos probabilísticos.

A variação da topologia da rede pode dificultar o roteamento em DTNs ao longo do tempo. A necessidade de rotear mensagens até os destinos, assim como assegurar uma comunicação eficiente, são desafios encontrados nestes tipos de rede. As desconexões

frequentes causadas pelo deslocamento dos nós, o elevado tempo de permanência de mensagens nas filas e a possível inexistência de um caminho fim-a-fim são alguns dos problemas existentes.

O encaminhamento das mensagens até os destinos pode ser realizado através de diversos mecanismos como a duplicação das mensagens em nós intermediários, o encaminhamento das mensagens pelo primeiro enlace disponível ou a utilização de tabelas de roteamento [Oliveira e Duarte 2007]. Neste último caso, a construção das tabelas pode ser feita adotando-se uma abordagem centralizada, ou seja, cada nó dispõe previamente de toda informação relevante para o roteamento, ou distribuída, onde os nós não conhecem de antemão o estado global da rede. Apesar das tabelas poderem ser construídas de forma centralizada, esta 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 [Peleg 2000]. Logo, a construção de tabelas de roteamento através de algoritmos distribuídos apresenta-se como uma solução mais apropriada.

Embora algumas estratégias de encaminhamento valham da duplicação de mensagens na rede, esta aborgadem pode acarretar em uma utilização ineficaz de recursos quando aplicado em redes com largura de banda limitada e reduzida capacidade de armazenamento dos nós. Outras formas de encaminhamento necessitam do conhecimento prévio de toda a rede, apesar disto não ser simples de obter na prática [Peleg 2000]. Além disso, até nestes casos não há garantia da entrega das mensagens, mesmo havendo uma rota fim-a-fim, dado que as mensagens podem ser perdidas devido às limitações de capacidade de armazenamento dos nós e da largura de banda dos enlaces. Portanto, o projeto de algoritmos que utilizem mecanismos de encaminhamento sem duplicação de mensagens torna-se uma alternativa interessante.

O presente trabalho propõe uma estratégia de roteamento composta de duas fases. Primeiramente, constrói-se em cada nó da rede uma tabela de roteamento com múltiplas jornadas até os destinos. Estas jornadas são definidas como rotas construídas considerando os instantes de tempo de existência dos enlaces. Em seguida, realiza-se o envio e recebimento das mensagens empregando um mecanismo de escolha de rotas alternativas evitando, assim, a sobrecarga de mensagens em determinados nós da rede

Apresenta-se neste trabalho três algoritmos distribuídos para construção de tabelas de roteamento em DTNs previsíveis. Considera-se que cada nó conhece apenas os intervalos de disponibilidade dos enlaces adjacentes, ou seja, cada nó tem informações apenas sobre a vizinhança e não de toda a rede. Em um dos algoritmos, o *Distributed Shortest Journey*

(DSJ), cada nó calcula a tabela de roteamento dele para todos os outros nós considerando o menor número de saltos. No outro, denominado *Distributed Earliest Journey* (DEJ), as tabelas são construídas objetivando a chegada mais cedo da informação ao nó de destino. Por último, o algoritmo *Distributed Fastest Journey* (DFJ), realiza a construção das tabelas levando-se em consideração as jornadas mais rápidas para chegar até os destinos. Em todos 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 destes três algoritmos distribuídos foi baseado, respectivamente, nos algoritmos sequenciais *shortest journey*, *foremost journey* e *fastest journey* propostos em [Bui-Xuan et al. 2003]. Adicionalmente, filtros para verificar a interseção entre os intervalos adjacentes com intuito de reduzir a quantidade de informação de controle trocada na rede foram incluídos, e uma estrutura de tabela de roteamento para DTNs foi projetada.

As tabelas geradas mantêm, 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. Desta 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. Os algoritmos realizam um filtro nos instantes de tempo de disponibilidade dos enlaces adjacentes para evitar o envio desnecessário de mensagens de controle pelos canais de comunicação. Os experimentos mostraram que a aplicação deste filtro reduziu entre 43% de 67% a quantidade de mensagens de controle trocadas pelos nós, e entre 76% e 90% a quantidade de bits trafegados, para as topologias de rede avaliadas.

Uma vez geradas as tabelas de roteamento, o nós podem enviar e receber mensagens de aplicação ao longo do tempo. Para isto, é proposto neste trabalho o algoritmo distribuído *Alternative Journey and Routing Protocol* (AJRP), para encaminhamento de mensagens em DTNs previsíveis cujo objetivo é entregar o maior número de mensagens aos destinos, executando um mecanismo de escolha de rotas alternativas para evitar o roteamento das mensagens por nós com *buffer* saturado. O encaminhamento das mensagens é feito sem duplicá-las em nós intermediários e cada nó conhece apenas os intervalos de disponibilidade para comunicação com seus vizinhos e a tabela de roteamento. Três mecanismos distintos para seleção de mensagens foram implementados com o intuito de avaliar o comportamento do roteamento na rede. Em um deles, a escolha da mensagem no *buffer* é realizada seguindo a ordem FIFO. No outro, denominado de seleção por HOP, as men-

sagens são ordenadas priorizando as que são destinadas a nós que demandem o menor número de saltos. Por último, a ordenação das mensagens é feita beneficiando aquelas destinadas a nós que possuem o menor número de intervalos de tempo de disponibilidade durante sua jornada, o qual denomina-se seleção por MEET.

Para avaliação desta heurística de encaminhamento foram comparados seus resultados com uma solução ótima de roteamento obtida através de uma formulação matemática em Programação Linear Inteira (PLI). Conforme encontrado na literatura [Jain et al. 2004, Balasubramanian et al. 2007], modelos de Programação Linear (PL) também foram desenvolvidos e utilizados para este fim. No entanto, suas formulações foram desenvolvidas de maneira que a complexidade para obtenção do resultado ótimo aumenta significativamente com o número de mensagens a serem entregues aos destinos, resultando na impossibilidade da avaliação de algoritmos de encaminhamento utilizando alta carga de mensagens na rede. Assim sendo, é necessária a elaboração de um modelo que apresente uma maior escalabilidade frente à carga de mensagens. Neste contexto, o presente trabalho propõe um modelo de PLI para DTNs que, baseado na abordagem *multi-commodities flow* [Ahuja e Orlin 1993], permite resolver instâncias de dimensões elevadas. Esta formulação difere das demais, pois as variáveis são indexadas pelo fluxo de mensagens entre a origem e o destino, ao invés de referenciar cada mensagem gerada. Desta forma, a complexidade para encontrar a solução ótima é diretamente proporcional ao número de nós e não ao número de mensagens da rede.

A comparação dos resultados do AJRP com a solução ótima foi realizada considerando as limitações de largura de banda dos enlaces e de capacidade de armazenamento dos nós. A avaliação mostrou que o AJRP entregou cerca de 96% da demanda de mensagens quando atribuída baixa carga de mensagens na rede e cerca de 83% quando submetida a uma alta carga. O AJRP também foi avaliado utilizando os *traces* gerados entre 14 de fevereiro e 15 de maio de 2007 em experimentos realizados por [Balasubramanian et al. 2007] com a rede veicular *DieselNet*. Os resultados obtidos foram comparados com algoritmos que consideram a replicação de mensagens, tais como: RAPID [Balasubramanian et al. 2007], MaxProp [Burgess et al. 2006], Random [Balasubramanian et al. 2007], Spray and Wait [Spyropoulos et al. 2005] e PROPHET [Lindgren et al. 2004].

Portanto, as contribuições deste trabalho são:

- Elaboração de um modelo para DTNs considerando a intermitência dos contatos e as restrições de capacidade de armazenamento dos nós e de largura de banda dos enlaces

- Projeto dos algoritmos distribuídos DSJ, DEJ e DFJ para construção de tabelas de roteamento
- Desenho do algoritmo AJRP para encaminhamento de mensagens até os destinos considerando jornadas alternativas
- Formulação matemática em Programação Linear Inteira utilizando a abordagem por fluxo de mensagens para obtenção da solução ótima de encaminhamento de mensagens
- Realização de experimentos para avaliar o desempenho dos algoritmos propostos considerando diversas topologias de rede

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 apresenta trabalhos relacionados ao problema de roteamento em DTNs. O Capítulo 3 descreve o modelo de Programação Linear Inteira proposto. O Capítulo 4 apresenta e analisa os algoritmos distribuídos propostos. Os resultados experimentais para análise do desempenho dos algoritmos são discutidos no Capítulo 5. Terminamos este trabalho no Capítulo 6 com as conclusões encontradas e propostas para trabalhos futuros.

Capítulo 2

Trabalhos relacionados

Neste capítulo são descritos diversos trabalhos encontrados na literatura que exploram o roteamento em DTNs. A Seção 2.1, a seguir, descreve alguns modelos propostos para DTNs. Porteriormente, na Seção 2.2, são apresentados trabalhos que propõem algoritmos de roteamento para estas redes.

2.1 Modelos para DTNs

São apresentados nesta seção alguns trabalhos relacionados à elaboração de modelos para DTNs, além do desenvolvimento de formulações matemáticas para encontrar a solução ótima de roteamento em tais redes.

Um dos primeiros trabalhos sobre modelagem de redes onde a topologia sofre frequentes alterações ao longo do tempo pode ser encontrado em [Ferreira 2002]. Este trabalho descreve que estas redes podem ser representadas por uma sequência de grafos direcionados, denominados temporais ou evolutivos, onde cada um deles referencia a topologia da rede para um determinado instante de tempo. É assumido que os instantes de disponibilidade dos enlaces são previsíveis e os mesmos são associados aos arcos dos grafos. O autor introduz um novo conceito sobre a definição de caminhos entre origens e destinos de uma rede. Segundo o autor, um caminho entre dois vértices de um grafo temporal é definido através de uma jornada, que tem como objetivo mapear para cada arco do caminho entre a origem e o destino os intervalos de tempo em que as mensagens podem efetivamente ser enviadas. Desta forma, evita-se que sejam utilizadas no trajeto entre a origem e o destino arcos que ocorreram apenas no passado com relação aos arcos já percorridos pelas mensagens. Apesar de levar em consideração as restrições de conectividade dos nós, o modelo não aborda as limitações de largura de bandas dos enlaces e da

capacidade de armazenamento (*buffer*) dos nós.

Em [Jain et al. 2004], os autores propuseram um modelo para DTNs baseado em grafos direcionados, onde cada intervalo de tempo de contato entre dois nós é representado por um arco entre eles, indicando a capacidade de transmissão de mensagens e o tempo de propagação da mesma. O modelo considera ainda que os nós possuem limitação de *buffer*. Uma formulação em Programação Linear (PL) foi elaborada com o objetivo de encontrar uma solução ótima de roteamento das mensagens até os destinos minimizando o tempo médio de entrega. A estratégia de encaminhamento de mensagens utilizado não considera a duplicação das mesmas em nós vizinhos. A formulação admite ainda que as mensagens podem ser fragmentadas para serem entregues aos destinos. No entanto, este conceito não é usualmente aplicado a DTNs, dado que o encaminhamento nestas redes é realizado por mensagens e não por pacotes [Cerf 2007]. Embora os autores tenham abordado no modelo as limitações encontradas em DTNs, não foi possível avaliar o roteamento utilizando altas cargas de mensagens. A explicação para isto é que foi realizado o mapeamento de cada mensagem como sendo um índice nas variáveis da formulação, o que aumenta consideravelmente a quantidade de variáveis e restrições a serem consideradas pela ferramenta de resolução e demandando, portanto, mais tempo de processamento para obtenção da solução ótima. O autor utiliza a seguinte notação:

Conjuntos	
V	Conjunto de vértices
E	Conjunto de arestas
I^v	Conjunto de arestas entrantes no nó $v \in V$
O^v	Conjunto de arestas saintes do nó $v \in V$
K^v	Conjunto de mensagens com destino o nó $v \in V$
Dados	
$c_{e,j}$	Capacidade da aresta $e \in E$ no instante $t \in T_E$
$d_{e,t}$	Tempo de propagação da aresta $e \in E$ no instante $t \in T_E$
b_v	Capacidade do *buffer* de $i \in V$
$s(k), d(k), w(k), m(k)$	Nó origem, nó de destino, instante de criação e tamanho da mensagem $k \in K$
Variáveis	
$N^k_{v,t}$	Quantidade da mensagem $k \in K$ ocupando o buffer do nó $v \in V$ no instante $t \in T_E$
$X^k_{e,I}$	Quantidade da mensagem $k \in K$ transmitida em $e \in E$ no intervalo $I \in I_E$
$R^k_{e,I}$	Quantidade da mensagem $k \in K$ recebida pelo destino da aresta $e \in E$ no intervalo $I \in I_E$

A seguir é descrita a formulação matemática proposta pelo autor para o problema em questão:

$$min \sum_{v \in V} \sum_{k \in K^v} \sum_{I_q \in T_E} (t_{q-1} - w(k)).\Big(\sum_{e \in I^v} R^k_{e,I_q} + \sum_{e \in O^v} X^k_{e,I_q} \Big) \tag{2.1}$$

considerando:

$$\sum_{e \in I^v} R^k_{e,I_q} + \sum_{e \in O^v} X^k_{e,I_q} = N^k_{v,t_q} - N^k_{v,t_{q-1}} \quad \forall s(k) \neq v, w(k) \neq t_q, v \in V, k \in K, I_q \in T \tag{2.2}$$

$$\sum_{e \in I^v} R^k_{e,I_q} + \sum_{e \in O^v} X^k_{e,I_q} = \ N^k_{v,t_q} - N^k_{v,t_{q-1}} + m(k) \quad \forall s(k) = v, w(k) = t_q, v \in V, k \in K, I_q \in T \tag{2.3}$$

$$R^k_{e,I_q|d_{e,t_{q-1}}} = X^k_{e,I_q} \quad \forall e \in E, k \in K, I_q \in T \tag{2.4}$$

$$\sum_{k \in K} N^k_{v,t_{q-1}} \leq b_v \quad \forall v \in V, I_q \in T \tag{2.5}$$

$$\sum_{k \in K} X^k_{e,I_q} \leq c_{e,t_{q-1}}.|I_q| \quad \forall e \in E, I_q \in T \tag{2.6}$$

$$\sum k \in K N^k_{v,t_0} = m(k) \quad \forall s(k) = v, w(k) = t_0, v \in V, k \in K \tag{2.7}$$

$$\sum k \in K N^k_{v,t_0} = 0 \quad \forall s(k) \neq v, w(k) \neq t_0, v \in V, k \in K \tag{2.8}$$

$$\sum k \in K N^k_{v,t_h} = m(k) \quad \forall d(k) = v, v \in V, k \in K \tag{2.9}$$

$$\sum k \in K N^k_{v,t_h} = 0 \quad \forall d(k) \neq v, v \in V, k \in K \tag{2.10}$$

Um modelo de Programação Linear Inteira (PLI) foi posteriormente desenvolvido em [Balasubramanian et al. 2007]. Neste trabalho, é assumido que cada nó possui contato apenas com um outro nó a cada instante de tempo. Apesar de considerar as limitações de largura de banda dos enlaces, não foram incluídas na formulação restrições quanto à capacidade de armazenamento dos nós. A formulação considera ainda que as mensagens

são encaminhadas até os destinos sem duplicá-las em nós intermediários . Assim como em [Jain et al. 2004], este modelo tem como objetivo minimizar o tempo médio de entrega de todas as mensagens aos destinos. No entanto, não é permitida a fragmentação das mensagens. As variáveis da formulação matemática proposta também referenciam as mensagens a serem roteadas, ou seja, a avaliação do roteamento utilizando altas cargas de mensagens também é limitada neste modelo.

2.2 Algoritmos para encaminhamento de mensagens em DTNs

Nesta seção são abordados diversos trabalhos encontrados na literatura com relação a propostas de algoritmos para roteamento em DTNs.

Em [Vahdat e Becker 2000] é proposto o algoritmo Epidemic Routing, que baseia-se na teoria de algoritmos epidêmicos para compartilhar mensagens com nós vizinhos. Para isso, cada nó envia para seus vizinhos uma lista com as identificações das mensagens que possui. Ao receber esta lista, o nó compara com a lista que detém para identificar quais mensagens não estão contidas. Estas mensagens faltantes são então solicitadas ao vizinho, que por sua vez envia as respectivas cópias para o nó solicitante. Ressalta-se que não é utilizado nenhum tipo de informação com relação às futuras alterações na topologia da rede. Uma avaliação foi realizada para verificar o desempenho do algoritmo quando submetido a variadas cargas de mensagens e limitando o número máximo de saltos que as mensagens podem realizar até alcançarem os destinos. Experimentos também foram realizados para aferir a utilização dos canais de comunicação e dos *buffers* dos nós.

Em [Bui-Xuan et al. 2003], três algoritmos centralizados denominados *foremost journey*, *shortest journey* e *fastest journey* foram desenvolvidos com o objetivo de encontrar, respectivamente, as jornadas mais cedo, ou seja, as jornadas onde o instante de tempo de chegada da mensagem nos nós de destino é o menor possível, as jornadas com menor número de saltos e as jornadas mais rápidas, isto é, as que apresentam as menores diferenças entre o instante de tempo de chegada da mensagem no destino e o instante de envio da mesma. Estes algoritmos utilizam um grafo temporal como entrada para suas execuções, ou seja, todas as alterações de topologia da rede ao longo do tempo devem ser conhecidas antecipadamente. O algoritmo *foremost journey* foi avaliado por [Goldman et al. 2006] comparando-o com tradicionais protocolos de roteamento utilizados em redes ad-hoc como o AODV [Perkins e Royer 1999], o DSDV [Perkins e Bhagwat 1994]

e o DSR [Johnson e Maltz 1996]. Questões relacionadas à quantidade de tempo necessária para entrega das mensagens e a proporção de pacotes perdidos em relação aos enviados foram analisadas.

O algoritmo PROPHET é proposto em [Lindgren et al. 2004] para realizar do roteamento das mensagens utilizando como base a probabilidade que os nós possuem de encontrar uns aos outros. Esta probabilidade é obtida através de cálculos realizados levando em consideração o histórico de contatos dos mesmos ao longo do tempo. Nenhuma informação quanto às alterações futuras na topologia da rede é conhecida antecipadamente. Apesar de considerar restrições na capacidade de armazenamento dos nós, o algoritmo não leva em conta as limitações na largura de banda do enlaces. Um mecanismo de replicação de mensagens é utilizado, onde as cópias das mensagens são enviadas para os nós que possuem as maiores probabilidades de encontrar os respectivos destinos. A avaliação realizada mostrou que o PROPHET conseguiu entregar mais mensagens que o algoritmo Epidemic Routing [Vahdat e Becker 2000], mesmo utilizando um número reduzido de informações de controle e de réplicas de mensagens.

Em [Jain et al. 2004], os autores implementam algoritmos de roteamento que, baseado em oráculos, utilizam informações sobre o estado atual e futuro da rede como os contatos entre os nós ao longo do tempo, a demanda de mensagens e a ocupação dos *buffers*. A estratégia de encaminhamento utilizada envia as mensagens para os nós intermediários até atingir os destinos sem que as mensagens sejam duplicadas. Os experimentos realizados com baixa e alta carga de mensagens na rede, mostram que os algoritmos que atingem o melhor desempenho com relação ao roteamento das mesmas são os que possuem mais informações acerca das características da rede. Os algoritmos também foram avaliados variando-se a capacidade de armazenamento dos nós e a largura de banda dos enlaces.

Chen em [Chen 2005] propõe o algoritmo SGRP para roteamento em redes de satélites com trajetórias previsíveis 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. A métrica utilizada na construção das tabelas considera os menores tempos obtidos através do somatório dos tempos médios de processamento, de fila de transmissão e de propagação das mensagens, que são monitorados e coletados por cada LEO. Uma vez geradas as tabelas de roteamento, as mensagens podem ser encaminhadas até seus destinos sem que sejam replicadas por nós intermediários. Os experimentos realizados

compararam o SGRP com o algoritmo DRA [Ekici et al. 2001], proposto anteriormente para roteamento em redes de satélites com tolopogias constantes, utilizando diversas cargas nos canais de comunicação.

O algoritmo Spray and Wait, elaborado em [Spyropoulos et al. 2005], considera um mecanismo de replicação que gera L cópias de cada mensagem e as distribui entre os contatos esperando que algum deles por ventura encontre o nó de destino. São analisadas algumas estratégias de priorização dos contatos que devem receber as cópias das mensagens, assim como realizada uma avaliação com relação ao número L de cópias a serem geradas considerando o tamanho da rede e a demanda de mensagens. O algoritmo não necessita de nenhuma informação prévia sobre a topologia da rede e não realiza nenhum tipo de verificação quanto à capacidade de *buffer* dos nós e largura de banda dos enlaces. Uma comparação entre o Spray and Wait e o algoritmo Epidemic Rounting [Vahdat e Becker 2000] foi realizada variando a quantidade de nós da rede e a carga de mensagens.

Outro algoritmo, denominado MaxProp, foi proposto em [Burgess et al. 2006] e utiliza informações de histórico de contatos para determinar a prioridade das mensagens a serem transmitidas. Um esquema de propagação de mensagens de controle para confirmação de recebimento também é implementado juntamente com uma política de replicação de mensagens e de exclusão de réplicas. As restrições de capacidade de *buffer* dos nós e largura de banda dos enlaces são consideradas, mas nenhuma informação sobre o estado da rede é conhecida antecipadamente. Foram realizados experimentos utilizando *traces* coletados da rede veicular *UMassDieselNet* e comparando os resultados do MaxProp com uma versão do algoritmo baseado em óraculos proposto em [Jain et al. 2004]. A avaliação mostrou que o MaxProp obteve o melhor desempenho com relação à porcentagem de entrega para distintas cargas de mensagens e variados tamanhos de *buffers*.

O algoritmo Rapid foi desenvolvido e avaliado em [Balasubramanian et al. 2007] e tem o objetivo de rotear as mensagens até o destino por meio da replicação destas nos nós intermediários. Para evitar a sobrecarga de mensagens na rede implementou-se um mecanismo que determina se uma mensagem deve ser replicada ou removida em determinados nós intermediários. Este algoritmo não necessita de nenhuma informação prévia sobre o estado da rede, porém, utiliza informações relativas ao histórico desta para estimar novas alterações na topologia. Uma avaliação foi realizada utilizando os *traces* da rede veicular *DieselNet*, onde o Rapid foi comparado com os algoritmos MaxProp [Burgess et al. 2006], Spray and Wait [Spyropoulos et al. 2005] e PROPHET [Lindgren et al. 2004] utilizando

distintas cargas de mensagens. Outro algoritmo, denominado Random, também foi implementado pelo o autor com o intuito de avaliar o desempenho da entrega de mensagens quando aplicado um mecanismo de duplicação de mensagens de forma randômica entre os vizinhos. Foi verificado também o desempenho do Rapid comparado a uma solução ótima obtida através da formulação em programação linear elaborada também neste trabalho.

O algoritmo NECTAR proposto em [Oliveira e Albuquerque 2009] utiliza o conceito de índice de vizinhança, considerando que os nós movimentam-se de forma que existe certa probabilidade que vizinhos possam ser reencontrados. Políticas de escalonamento e descarte de mensagens são desenvolvidas e um mecanismo de replicação de mensagens é implementado. O algoritmo não considera para encaminhamento das mensagens nenhuma informação sobre o estado futuro da rede. Uma avaliação é realizada comparando os resultados do NECTAR com os obtidos pelos algoritmos Epidemic Routing [Vahdat e Becker 2000] e PROPHET [Lindgren et al. 2004] considerando tamanhos distintos de *buffer* e verificando a quantidade de mensagens entregues e o número de saltos realizados pelas mesmas.

Capítulo 3

Descrição do problema e formulação matemática

DTNs previsíveis podem considerar várias informações conhecidas antecipadamente, tais como: a disponibilidade de contato entre os nós da rede ao longo do tempo; as demandas de mensagens de cada nó; e a capacidade de armazenamento destes. No entanto, a obtenção a *priori* de todo este conhecimento é praticamente inviável devido ao tamanho da rede ou até mesmo a própria dinâmica de geração das mensagens. O presente trabalho considera que apenas os períodos de disponibilidade dos enlaces adjacentes são conhecidos previamente por cada nó da rede.

O problema de roteamento em DTNs consiste em entregar as mensagens aos destinos de acordo com uma métrica pré-determinada, levando-se em conta as restrições de disponibilidade e capacidade dos enlaces para transmissão das mensagens, assim como as limitações de armazenamento destas pelos nós da rede. As desconexões presentes nestas redes podem implicar na inexistência de uma rota fim-a-fim entre a origem e o destino.

Para contornar tais limitações de conectividade, utiliza-se o conceito de jornada que, conforme descrito no capítulo anterior, são definidas como rotas construídas considerando os instantes de tempo de existência dos enlaces. Desta forma, os nós intermediários armazenam em seus *buffers* as mensagens recebidas e as encaminham apenas em momentos adequados, evitando que estas sejam roteadas para enlaces que estavam disponíveis apenas no passado. Por exemplo, na Figura 3.1, as jornadas válidas para envio de mensagens do nó A para o nó C, devem considerar o nó B como intermediário. Ou seja, o nó A deverá enviar as mensagens para B entre os instantes 1 e 10, o nó B armazenará as mensagens em seu *buffer* e poderá encaminhá-las para o nó C a partir do instante 20.

O roteamento das mensagens na rede segue uma métrica específica cujo objetivo é

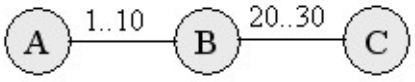

Figura 3.1: Exemplo de jornada

mensurar a qualidade da solução obtida. Exemplos de métricas que podem ser adotadas são: tempo médio ou máximo para entrega das mensagens; percentual de mensagens entregues dentro de um prazo pré-estabelecido; e número máximo de mensagens entregues [Ramanathan et al. 2007, Balasubramanian et al. 2007]. O presente trabalho optou por explorar este último critério, pois este representa um dos objetivos mais básicos e importantes com relação à entrega de mensagens em uma DTN.

Uma arquitetura para DTN com contatos programados ou previsíveis deve considerar sincronismo de tempo entre os nós da rede, conforme descrito na *Request For Comments* [Cerf 2007] publicada no *Internet Engineering Task Force* (IETF). Neste trabalho, foi definido que cada período de tempo é denominado pulso. Portanto, a cada pulso p os nós podem criar, receber, processar e enviar mensagens. Considera-se que a comunicação entre nós adjacentes é realizada dentro de um pulso, ou seja, o envio e recebimento de mensagens de controle e de aplicação não podem ultrapassar o término do pulso em que estas tiveram sua transmissão iniciada. Assume-se ainda que a fragmentação das mensagens não é admitida e que os canais de comunicação obedecem a ordem FIFO (*first-in-first-out*). Ressalta-se que os nós da rede usam esta informação para controle síncrono do tempo e verificação de quando cada enlace adjacente está ativo para comunicação.

DTNs podem apresentar propriedades cíclicas, isto é, após o último pulso a contagem é reiniciada e a execução retorna para o primeiro pulso, onde volta-se novamente a topologia de rede inicial. Para efeito de avaliação admitiu-se apenas a execução do primeiro ciclo e, por conseguinte, as mensagens não entregues neste período são descartadas.

Neste trabalho, propomos um modelo onde uma DTN previsível é representada por um grafo orientado e subdividido em pulsos, isto é, períodos de tempo de mesma dimensão. Cada nó da rede possui identificação distinta e é representado por vértices em diferentes pulsos. Cada enlace é simbolizado por dois arcos de capacidade finita, onde um deles conecta um vértice v_1 no pulso t_x ao v_2 em t_{x+1}, enquanto o outro arco conecta o vértice v_2 no pulso t_x ao v_1 em t_{x+1}. No caso do *buffer*, os vértices de origem e destino referenciam o mesmo nó da rede. Considera-se ainda que a largura de banda disponível é independente para cada um dos contatos, isto é, não é admitido o compartilhamento dos canais de

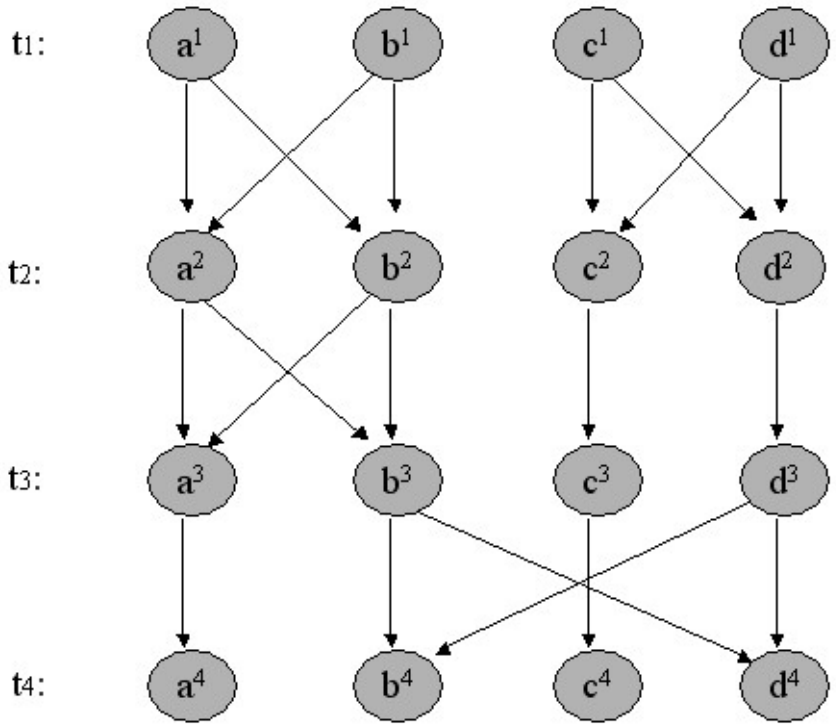

Figura 3.2: Exemplo de representação de uma DTN

comunicação estabelecidos simultaneamente com nós vizinhos.

A Figura 3.2 mostra um exemplo do modelo proposto. O grafo ilustrado contém 4 nós (a, b, c e d) indexados pelos instantes de tempo e seu ciclo de execução ocorre a cada 4 pulsos. Observa-se que entre os pulsos 1 e 2 os enlaces entre os nós a e b e entre os nós c e d estão disponíveis. No entanto, entre os instantes 2 e 3 apenas a comunicação entre a e b permanece disponível. Como pode ser notado, os vértices a e d não possuem conectividade, mas o d é alcançável por a através do vértice intermediário b. Para tanto, basta que, no pulso 1, a encaminhe a mensagem para b, que por sua vez deverá armazenar a mensagem no pulso 2 e reencaminhá-la para d no pulso 3. Verifica-se que a recíproca não é verdadeira, pois não existe jornada partindo de d até a.

Um modelo de PLI é proposto para o problema de roteamento em DTNs. Apesar das formulações matemáticas não serem adequadas para a resolução de problemas *online*, ou seja, problemas onde determinados parâmetros relevantes à solução são conhecidos apenas ao longo da execução, seu uso é importante para obtenção de limites, os quais são usados como valores de referência para avaliação de algoritmos *online* não exatos. Isto posto, a formulação descrita neste capítulo é empregada como um método para resolução *offline*, isto é, considera-se que todas as informações relevantes são conhecidas previamente, do roteamento em DTNs. Salienta-se que, nesta versão, todas as mudanças são previamente

conhecidas, o que possibilita o aproveitamento mais eficiente da largura de banda dos enlaces e dos *buffers* dos nós. O modelo proposto explora a abordagem de Multi-Fluxos [Ahuja e Orlin 1993], onde são mapeados os fluxos de mensagens de cada nó origem.

Foi utilizada a seguinte notação:

Conjuntos	
V	Conjunto de vértices
T	Conjunto de instantes de tempo, numerados de 0 a t_f
A^t	Conjunto de arcos presentes no instante $t \in T$, $A^0 = \emptyset$
A	Conjunto de arcos presentes em algum instante, $A = \bigcup_{t \in T} A^t$
K	Conjunto de vértices origens de mensagens, $K \subseteq V$
Dados	
u_{ij}	Capacidade do arco $(i, j) \in A$
b_i	Capacidade do *buffer* de $i \in V$
c_{ij}	Custo para transmitir uma mensagem pelo arco $(i, j) \in A$
r_i	Custo para armazenar uma mensagem no *buffer* de $i \in V$
d_i^k	Quantidade de mensagens que $i \in V$ deve receber da origem $k \in K$ até o instante t_f
o_k^t	Quantidade de mensagens geradas por $k \in K$ em $t \in T$
Variáveis	
x_{ij}^{kt}	Quantidade de mensagens com origem em $k \in K$ transmitida em $t \in T$ pelo arco $(i, j) \in A^t$
y_i^{kt}	Quantidade de mensagens com origem em $k \in K$ ocupando o *buffer* de $i \in V$ em $t \in T$
z_i^{kt}	Quantidade de mensagens com origem em $k \in K$ recebida por $i \in V$ em $t \neq 0 \in T$

A seguir é descrita a formulação matemática proposta para o problema em questão:

$$min \sum_{t=1}^{T} \sum_{k=1}^{K} \Big(\sum_{i \in V} r_i y_i^{kt} + \sum_{i,j \in A^t} c_{ij} x_{ij}^{kt} \Big) \tag{3.1}$$

considerando:

$$\sum_{j:(j,i) \in A^{t-1}} x_{ji}^{k(t-1)} - \sum_{j:(i,j) \in A^t} x_{ij}^{kt} + y_i^{k(t-1)} - y_i^{kt} - z_i^{kt} = 0 \quad \forall i \in V, k \neq i \in K, t \neq 0 \in T \tag{3.2}$$

$$\sum_{j:(j,k)\in A^{t-1}} x_{jk}^{k(t-1)} - \sum_{j:(k,j)\in A^{t}} x_{kj}^{kt} + y_k^{k(t-1)} - y_k^{kt} = -o_k^t \quad \forall k \in K, t \neq 0 \in T \tag{3.3}$$

$$\sum_{k=1}^{K} x_{ij}^{kt} \leq u_{ij} \quad \forall i, j \in V, t \in T \tag{3.4}$$

$$\sum_{k=1}^{K} y_i^{kt} \leq b_i \quad \forall i \in V, t \in T \tag{3.5}$$

$$\sum_{t=1}^{T} z_i^{kt} = d_i^k \quad \forall i \in V, k \in K \tag{3.6}$$

$$y_i^{k0} = 0 \quad \forall i \in V, k \in K \tag{3.7}$$

$$y_i^{kt}, z_i^{kt} \in Z^+ \quad \forall i \in V, k \in K, t \neq 0 \in T \tag{3.8}$$

$$x_{ij}^{kt} \in Z^+ \quad \forall k \in K, t \in T, (i,j) \in A^t \tag{3.9}$$

A função objetivo (3.1) minimiza a soma dos custos gerados pela transmissão das mensagens nos enlaces e pela utilização do *buffer* dos nós intermediários. Com isso, prioriza-se o encaminhamento das mensagens por jornadas mais curtas, reduzindo a utilização dos recursos da rede. As restrições (3.3) e (3.2) estão relacionadas, respectivamente, à conservação do fluxo de mensagens nos nós que geram e que não geram mensagens. No primeiro conjunto de restrições (3.3), é verificado através dos dados o_k^t a quantidade de mensagens da origem k que devem ser geradas no pulso t. No segundo conjunto de restrições (3.2), é verificado pelas variáveis z_i^{kt} a quantidade de mensagens da origem k que são recebidas pelo nó de destino i no pulso t. Em ambas restrições tem-se também o mapeamento das mensagens entrantes e saintes dos enlaces adjacentes e das mensagens que foram armazenadas no *buffer* do nó no pulso anterior e no pulso corrente. As restrições (3.4) e (3.5) garantem, respectivamente, que a capacidade dos enlaces e dos *buffers* não seja extrapolada. As restrições (3.6) asseguram que todas as mensagens enviadas sejam devidamente entregues aos destinos. Finalmente, as restrições (3.7), (3.8) e (3.9) garantem que as variáveis devem assumir valores inteiros e não-negativos.

Vale ressaltar que a formulação matemática foi elaborada com o intuito de não elevar a complexidade da solução à medida que o número de mensagens aumenta, ou seja, ao invés de indexar cada mensagem nas variáveis das equações, foi indexado apenas o fluxo de mensagens entre uma origem e um destino, o que significa que a complexidade desta abordagem é diretamente proporcional à quantidade de vértices e não à quantidade de mensagens. Assim sendo, é possível avaliar redes cuja quantidade de mensagens a serem trafegadas é relativamente elevada. Cabe ressaltar ainda que o modelo não prevê perda de mensagens. Em vista disso, nos experimentos apresentados no Capítulo 5 são avaliadas apenas instâncias que possuem solução viável admitindo-se que todas as mensagens serão entregues.

Apesar da formulação em PLI proposta considerar as particularidades encontradas em DTNs e possuir menor complexidade para determinar a solução ótima para altas cargas de mensagens, algumas limitações podem ser destacadas. Primeiro, o mapeamento do tempo de execução em períodos de mesma duração pode implicar na geração de muitos pulsos para determinados tipos de rede, o que aumenta a complexidade da solução. Segundo, o dimensionamento do tamanho do pulso deve levar em consideração a quantidade de tempo suficiente para que as mensagens de controle e de aplicação sejam transmitidas completamente. Por último, a número de nós da rede influencia diretamente no tamanho do grafo para representá-la, dado que cada nó é representado por T vértices, onde T é a quantidade de pulsos.

Capítulo 4

Roteamento em DTNs

Neste capítulo são apresentados os algoritmos distribuídos Distributed Shortest Journey (DSJ), Distributed Earliest Journey (DEJ) e Distributed Fastest Journey (DFJ), que controem tabelas de roteamento em DTNs. O algoritmo distribuído *Alternative Journey Routing Protocol* (AJRP) também é apresentado e tem como objetivo realizar o roteamento de mensagens em DTNs considerando as restrições de largura de banda dos enlaces e de capacidade de armazenamento dos nós. A estratégia de roteamento distribuído proposta é composta de duas etapas. Primeiramente, executa-se um algoritmo para construção de tabelas de roteamento em cada nó da rede, mapeando as jornadas viáveis até os destinos. A partir destas tabelas, realiza-se o envio e recebimento das mensagens empregando um mecanismo de escolha de jornadas alternativas evitando, assim, a sobrecarga de mensagens em determinados nós da rede.

Conforme mencionado no Capítulo 2, um ciclo é dividido em instantes de tempo denominados pulsos. Em cada pulso, quando necessário, os nós enviam mensagens de controle e aplicação para os vizinhos que estão com enlace ativo e processam as mensagens recebidas. Assume-se que a transmissão é realizada dentro de um pulso e que os tempos de processamento das mensagens são desprezíveis. Classifica-se as mensagens como referentes à aplicação, ou seja, advindas de uma camada de *software* de nível superior ao roteamento, ou como mensagens de controle, que servem como auxílio para algoritmos executarem o roteamento. Define-se ainda que mensagens são perdidas apenas quando o nó receptor atingiu sua capacidade máxima de armazenamento ou quando as mesmas não alcançaram o destino até o último pulso t_f, ou seja, permaneceram no *buffer* de algum nó intermediário.

4.1 Construção das tabelas de roteamento

Apresentamos a seguir o algoritmo DSJ que, de forma distribuída, constrói em cada nó uma tabela de roteamento com os intervalos de tempo factíveis para envio de mensagens para os destinos. Considera-se que os intervalos de tempo de disponibilidade de contato com os vizinhos são conhecidos previamente. O critério adotado para construção das tabelas por este algoritmo é minimizar o número de saltos que as mensagens de aplicação fazem em nós intermediários até atingir o destino. Cada nó da rede troca mensagens de controle com seus vizinhos anunciando o conjunto de nós conhecidos até o momento. As informações enviadas para os nós vizinhos são as identificações dos nós alcançáveis, o pulso corrente, os intervalos de tempo possíveis para o envio de mensagens, o número de saltos associados a cada intervalo de tempo e o número do ciclo de execução corrente. Ao receber mensagens de controle de determinado vizinho os nós atualizam suas jornadas na tabela de roteamento e as enviam para os demais vizinhos.

As tabelas de roteamento foram projetadas para manter múltiplas jornadas para cada nó destino. Ao receber mensagens de atualização das jornadas, os nós armazenam não apenas a jornada que oferece um roteamento com menor número de saltos, mas sim as R melhores jornadas para cada intervalo de tempo, onde R é um parâmetro do algoritmo. Desta forma, pode-se escolher a melhor jornada a ser empregada no momento de envio das mensagens de aplicação, ou seja, é possível impedir o envio de mensagens para os enlaces cujos vizinhos não dispõem de *buffer* para recebê-las.

As estruturas de dados empregadas pelo algoritmo DSJ são o vetor de vizinhos, a estrutura da mensagem e a tabela de roteamento. O vetor de vizinhos, denominado de vecInter e apresentado na Figura 4.1, armazena as informações iniciais conhecidas pelo nó. Cada posição do vetor contém os dados de um nó vizinho, isto é, sua identificação, a capacidade do enlace e os intervalos de disponibilidade para comunicação. Estas informações são representadas pelos campos *{vizID, capacidade, up[1], up[2], ..., up[max], down[1], down[2], ..., down[max]}*, onde *max* é o número máximo de intervalos de tempo disponíveis. Os campos *up[i]* e *down[i]* representam, respectivamente, o pulso de início e de fim do intervalo i.

A estrutura da mensagem, que chamamos de *setType* e é apresentada na Figura 4.2 é utilizada por cada nó $n_i \in N$ para trocar 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.

A tabela de roteamento, Figura 4.3, contém a lista dos nós destinos da rede e, para cada um deles, são armazenadas R jornadas para cada intervalo de tempo. Estas encontram-se em ordem de relevância, cuja determinação é feita de forma crescente de acordo com a quantidade de saltos para alcançar o destino. Cada jornada contém um conjunto ordenado de intervalos, onde para cada um é mantido o início e fim do mesmo, a distância até o destino, o identificador do vizinho para encaminhar a mensagem e o campo *update*[], que é um vetor que indica quais vizinhos já receberam ou não as atualizações realizadas na respectiva jornada da tabela de roteamento.

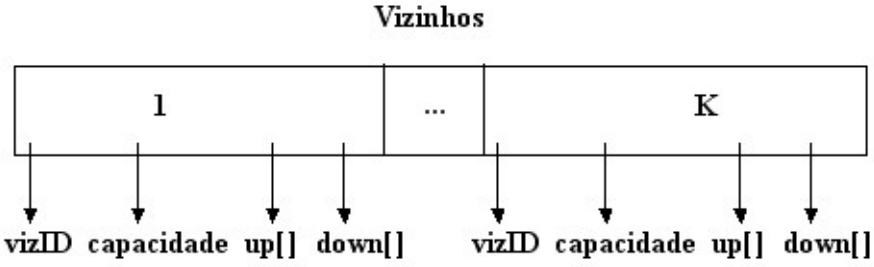

Figura 4.1: Vetor de vizinhos

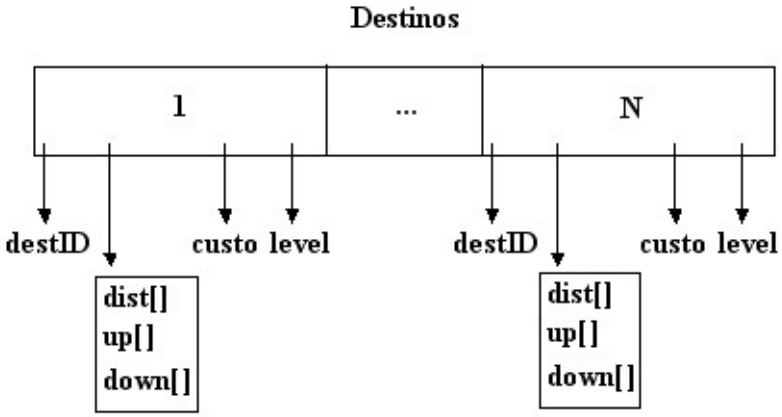

Figura 4.2: Estrutura das mensagens

O algoritmo foi dividido em duas partes. A primeira, descreve a função vecInter no Algoritmo 1, 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 2, que possui dois tipos de eventos dependendo da entrada recebida. Neste procedimento, é realizada a crítica dos intervalos adjacentes e envio de mensagens, quando necessário, para os vizinhos. Durante a construção das tabelas de roteamento, os nós recebem mensagens

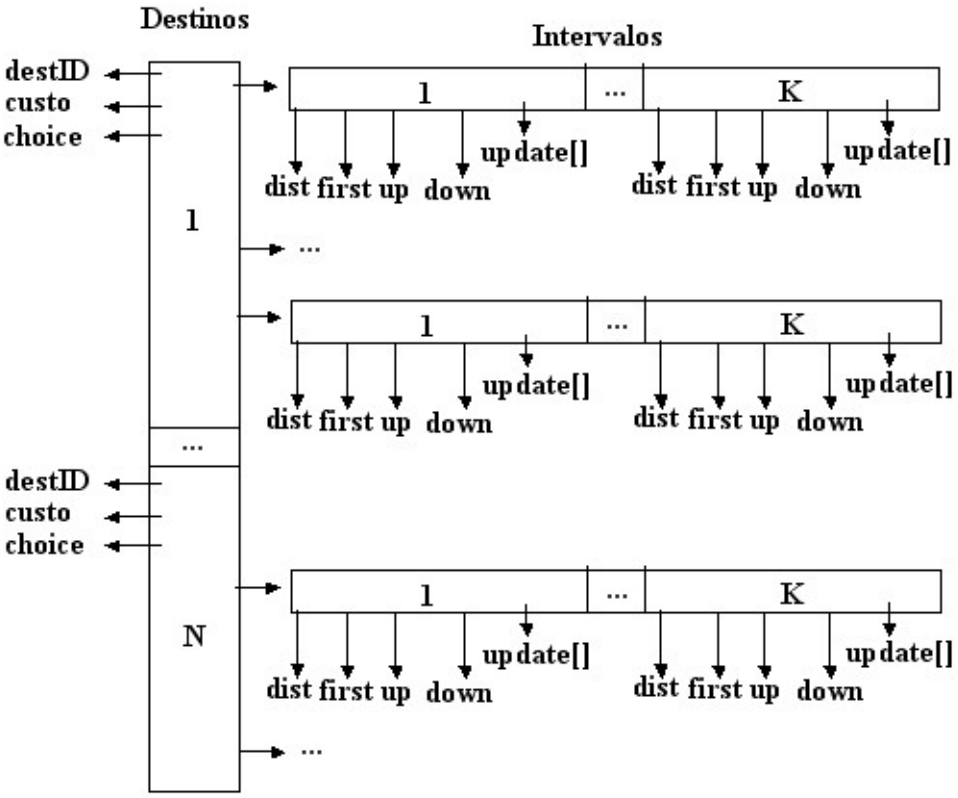

Figura 4.3: Tabela de roteamento

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 jornadas enviadas 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 verificação de interseção entre os intervalos adjacentes, 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 13 do Algoritmo 1). Cada nó mantém então nesta estrutura de dados as combinações de vizinhança possíveis.

Como descrito no Algoritmo 2, 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 5-13 e 32-43). Nos demais instantes de tempo do intervalo de disponibilidade, o nó somente envia mensagem ao

Dados

N	=	conjunto de nós da rede
M	=	conjunto de enlaces da rede
R	=	conjunto de jornadas para cada pulso para cada destino
Y	=	conjunto de intervalos de tempo
$Neig_i$	=	conjunto de vizinhos do nó i
$myRank_i$	=	identificação do nó i

Variáveis

p	=	0
$level_i$	=	0
SET_i	=	$nulo$
$found_i$	=	$false$
$buffer_i$	=	$nulo$
$initiated_i$	=	$false$
$bufferUsage_i$	=	0
$bufferSize_i$	=	tamanho do *buffer* do nó i
$status_i[j]$	=	$0,\ \forall\ j \in Neig_i$
$vecInter_i[j]$	=	(vizID, capacidade, up[], down[]), $\forall\ n_j \in Neig_i$
$permut_i[j][k]$	=	$false,\ \forall\ j,k \in Neig_i$
$routeTable_i[j][r][y]$	=	$nulo,\ \forall\ j \in N$, $r \in R$ e $y \in Y$

vizinho caso uma jornada seja descoberta ou modificada. Após enviar suas mensagens, o nó processa as mensagens recebidas, podendo gerar modificações na tabela de roteamento. Uma nova jornada pode ser inserida, ou uma jornada 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 17-31).

Estas ações são executadas até que os nós completem o ciclo, ou seja, até alcançar o pulso t_f. Um novo ciclo pode ser iniciado caso exista algum nó que possua alguma atualização para ser enviada. Desta forma, o algoritmo reinicia o processamento no pulso t_0 novamente, 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.

A sequência de pulsos de t_0 a t_f pode ser executada várias vezes, gerando assim vários ciclos de execução. O número máximo de ciclos é da ordem do diâmetro da rede. Isso acontece devido à disposição e desconexão dos intervalos de tempo nos enlaces. Para uma topologia linear onde os nós estão em série, 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 dos arcos são totalmente desconexos e

```
1  Algorithm TestIntervals(vecInter, SET_i, j, level)
2  forall (up1, down1) ∈ vecInter_i[j], (down1 − up1) ≥ vecInter_i[j].cost do
3  |  found_i ← false;
4  |  forall (up2, down2) ∈ vecInter_i[k], k ∈ Neig_i, k ≠ j do
5  |  |  if (NOT found) AND (up2 + vecInter_i[k].cost ≤ up1) OR
   |  |  (down1 ≥ up2 + vecInter_i[j].cost + vecInter_i[k].cost) then
6  |  |  |  found_i ← true;
7  |  |  end
8  |  end
9  |  if found_i = true then
10 |  |  SET_i[k] ← SET_i[k] ∪ vecInter_i[j], level;
11 |  end
12 |  if found_i = false then
13 |  |  permut[j][k] ← true;
14 |  end
15 end
```

Algorithm 1: Algoritmo TestIntervals

dispostos de forma crescente, 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) da série conheça o último nó (n_N), para $N > 3$.

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 o tempo de transmissão pelos enlaces (n_i, n_k) e (n_i, n_j). Veja Algoritmo 1. Desta forma, intervalos de tempo que só existiram no passado não são enviados aos vizinhos.

Com relação aos algoritmos DEJ e DFJ, foram utilizados os mesmos eventos associados a cada pulso na elaboração dos algoritmos e a mesma estrutura de dados do algoritmo DSJ. Porém, ao invés da tabela de roteamento armazenar a distância em saltos, ela guarda, respectivamente, o tempo de chegada mais cedo e o tempo de duração da jornada para alcançar o destino.

Para o algoritmo DEJ verifica-se, então, que as linhas 21 e 22 do Algoritmo 2 devem

```
1  Algorithm DSJ(verInter, routeTable)
2  INPUT:
3     p > 0, MSG_i(p) = 0;
4  ACTION:
5  forall active (i, j) in p, j ∈ Neig_i do
6     if initiated = false then
7        initiated ← true;
8        forall j ∈ Neig_i do
9           TestIntervals(SET_i, j, level_i);
10          SEND SET_i[k] to n_k, ∀ k ∈ Neig_i;
11       end
12    end
13 end
14 INPUT:
15    p > 0, SET_j ∈ MSG_i(p), origem_i(SET_j) = (n_i, n_j);
16 ACTION:
17 forall SET_j ∈ MSG_i(p) do
18    forall k ∈ SET_j do
19       forall (dist, up, down) ∈ SET_j[k] do
20          forall t ∈ routeTable_i[k], up ≤ t ≤ down do
21             if routeTable_i[k][t].dist > dist + 1 then
22                routeTable_i[k][t].dist ← dist + 1;
23                routeTable_i[k][t].first ← n_j;
24                forall j ∈ Neig_i do
25                   routeTable_i[k][t].update[j] ← true;
26                end
27             end
28          end
29       end
30    end
31 end
32 forall n_k ∈ N do
33    forall t ∈ routeTable_i[k][t] do
34       forall n_j ∈ Neig_i do
35          if routeTable_i[k][t].update[j] = true then
36             if active (i, j) then
37                SET_i[j] ← SET_i[j] ∪ routeTable_i[k][t];
38                routeTable_i[k][t].update[j] ← false;
39             end
40          end
41       end
42    end
43 end
44 SEND SET_i[j] to n_j;
```

Algorithm 2: Algoritmo DSJ

ser alteradas 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 esta mensagem em um instante t_y para seu vizinho n_k. Nesse caso, tem-se 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 é realizado pelo algoritmo DEJ.

Analogamente, para o algoritmo DFJ as linhas 21 e 22 do Algoritmo 2 também devem ser alteradas para que o campo *dist* considere como distância a duração da jornada. Da mesma forma, é necessário saber quando a mensagem pode chegar em cada nó para que seja feito o controle da duração do jornada à medida que os tempos de conectividade até os nós de destinos são conhecidos.

4.1.1 Análise de complexidade

Ao final da execução dos algoritmos de construção de tabelas de roteamento 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 arco. Logo, a quantidade de mensagens enviadas é de $2MN$ e a complexidade é de $O(MN)$, onde N é o número de nós e M é o número de enlaces. 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 $O(MNILlog(V))$.

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 da rede e cada ciclo processa T

pulsos, temos que a complexidade de tempo, ou seja, a quantidade de pulsos executados pelo algoritmo, é da ordem de $O(NT)$.

A manutenção das R jornadas por intervalo não influencia nas complexidades de mensagem e tempo, dado que são utilizadas as mensagens que já são recebidas dos vizinhos. No entanto, a quantidade de informação a ser armazenada é acrescida, sendo da ordem de $O(NRY)$, onde R é a quantidade de jornadas por intervalo de tempo para cada destino e Y é a quantidade de intervalos disponíveis para cada jornada.

4.1.2 Exemplo de execução

A Figura 4.4 apresenta um exemplo de execução do algoritmo DSJ, monstrando a troca de mensagens entre os nós A, B, C e D da rede ilustrada. Cada enlace está associado um intervalo de tempo de disponibilidade e cada mensagem é representada por uma seta partindo da origem para o destino. A Figura 4.5 apresenta as tabelas de roteamento que cada nó obtém inicialmente com base nos intervalos de disponibilidade dos enlaces adjacentes. Foi considerada apenas uma jornada ($R = 1$) por intervalo de tempo para cada destino.

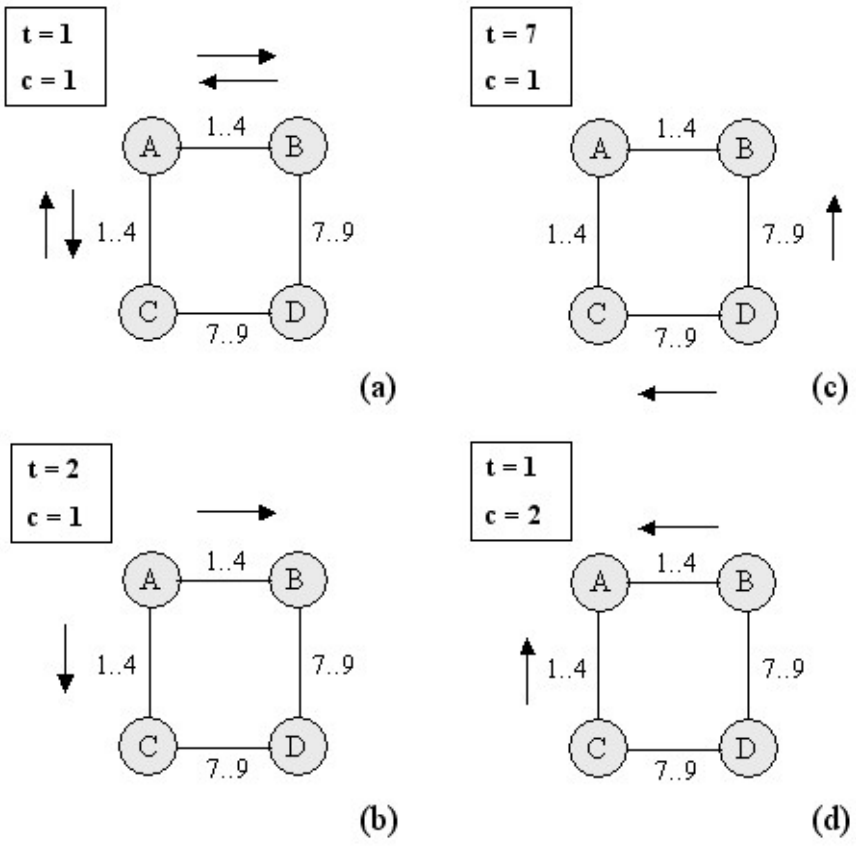

Figura 4.4: Exemplo de execução do DSJ: troca de mensagens

Na Figura 4.4(a), referente às mensagens trocadas no pulso 1 do ciclo 1, o nó A envia uma mensagem de controle para o nó B informando que consegue enviar mensagens para o nó C nos pulsos 1 a 4. Analogamente, o nó A também informa ao nó C que consegue enviar mensagens para o nó B nos pulsos 1 a 4. Ainda no pulso 1, os nós B e C informam ao nó A que conseguem enviar mensagens para o nó D no pulso 7. No pulso 2 do ciclo 1 (Figura 4.4(b)), o nó A possui atualizações na tabela de roteamento e deve propagar para os vizinhos que ainda não as conhece. Desta forma, o nó A envia para o nó B as atualizações referentes ao nó C, ou seja, que alcança o nó D via nó C e, de forma análoga, envia para o nó C as atualizações referentes ao nó B.

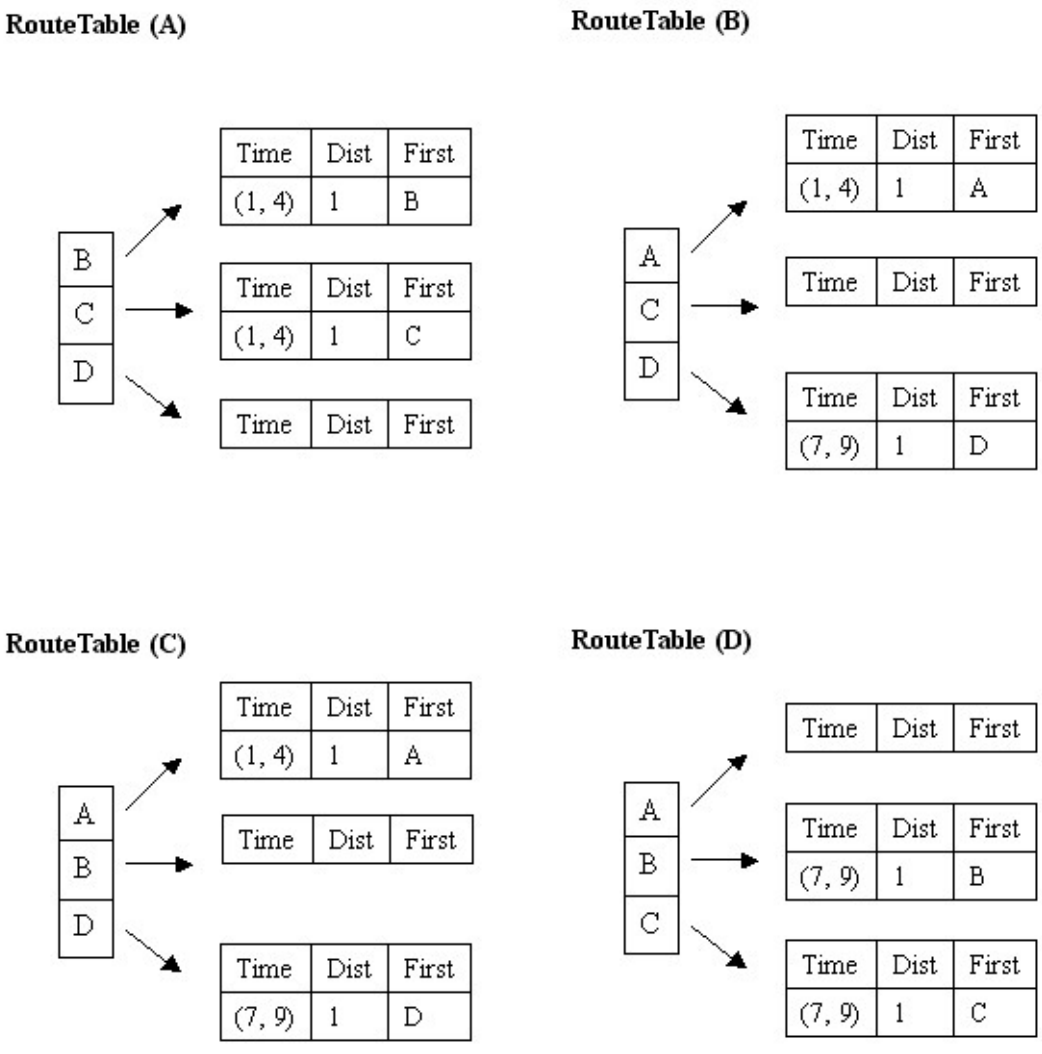

Figura 4.5: Exemplo de execução do DSJ: Tabelas de roteamento iniciais

Entre os pulsos 3 e 6 do ciclo 1 não é enviada nenhuma mensagem, dado que não existem mais atualizações a serem propagadas. No pulso 7 do ciclo 1 (Figura 4.4(c)), quando os enlaces entre os nós B e D e entre os nós C e D tornam-se ativos, o nó D informa ao nó B que alcança o nó C nos pulsos 7 a 9 e informa ao nó C que alcança o nó

B também nos mesmos pulsos. Como pode ser observado, os nós B e C não propagam para o nó D seus conhecimentos sobre o nó A, dado que o enlace para este está no passado com relação ao enlace adjacente a D. Em seguida, para os pulsos 8 e 9, nenhuma informação é trocada.

Após o último pulso do ciclo 1, os nós B e C ainda possuem atualizações na tabela de roteamento que precisam ser propagadas. Então, no pulso 1 do ciclo 2 (Figura 4.4(c)), estes nós informam o nó A sobre suas atualizações.

Ao término do ciclo 2, quando nenhum nó possui mais atualizações para serem enviadas, têm-se as tabelas de roteamento finais, conforme pode ser visualizado na Figura 4.6.

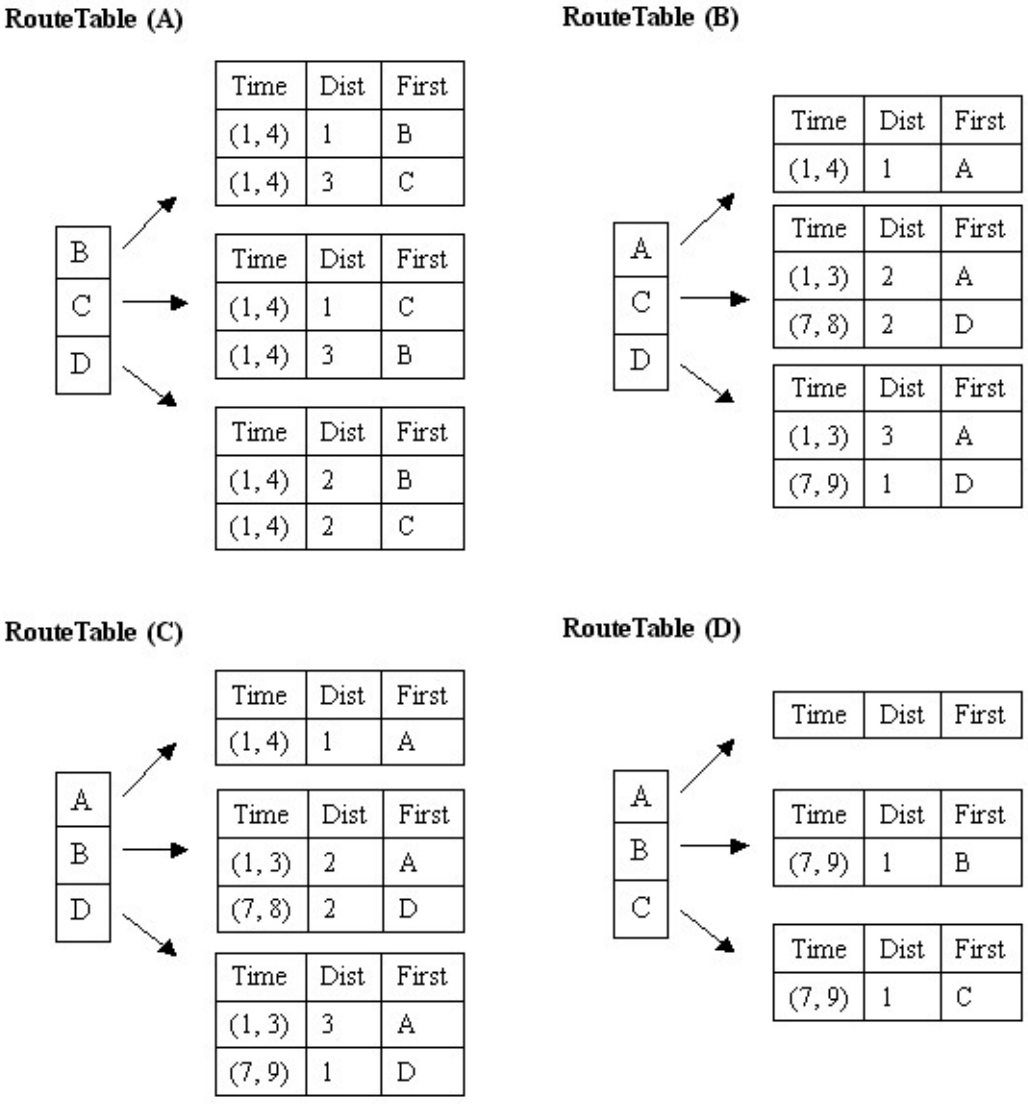

Figura 4.6: Exemplo de execução do DSJ: tabelas de roteamento finais

4.2 Encaminhamento de mensagens através do AJRP

Uma vez construídas as tabelas de roteamento, cada nó tem conhecimento dos momentos adequados para envio de mensagens de aplicação para os demais nós da rede. No entanto, à medida que mais mensagens são geradas, tanto a utilização dos canais de comunicação como a dos *buffers* dos nós intermediários aumentam, congestionando as jornadas e provocando o atraso na entrega ou até a perda de mensagens.

Em virtude disto, o AJRP faz uso de mensagens de controle com o intuito de balancear a carga de mensagens entre as jornadas disponíveis para cada nó da rede. Isto é feito através do envio do percentual de ocupação do *buffer* dos nós para seus vizinhos. A partir desta informação, os nós escolhem como preferencial uma jornada que passa pelo vizinho com maior disponibilidade de armazenamento de mensagens no momento. Isto posto, as mensagens de aplicação são encaminhadas para jornadas que apresentam menos congestionamento, previnindo eventuais gargalos na rede. Além disso, o roteamento é realizado minimizando o número de saltos que a mensagem de aplicação realiza até chegar ao destino, reduzindo a utilização de recursos da rede e, consequentemente, a perda de mensagens devido às limitações de armazenamento dos nós e à capacidade de trasmissão dos enlaces da rede.

Para avaliação do AJRP foi empregada uma estrutura de dados para mapear os instantes de tempo de geração das mensagens pelos nós. Embora o algoritmo não faça uso desta informação para tomada de decisões de roteamento, ou seja, os nós não detêm conhecimento *a priori* da lista de demandas futuras, esta informação é necessária para simulação da geração das mensagens nos momentos pré-estabelecidos nos experimentos.

Os dados utilizados pelo Algoritmo 3, assim como as principais variáveis e suas inicializações, estão descritos a seguir.

O algoritmo AJRP é síncrono e sua apresentação é feita através de dois eventos que, por sua vez, demandam uma ação específica. O primeiro, refere-se à contagem do tempo, onde, a cada pulso p, mensagens de aplicação podem ser recebidas pelos nós e armazenadas na variável $buffer_i$ (linhas 4 a 8). Mensagens também podem ser geradas neste momento de acordo com a variável $demandList_i$ (linhas 9 a 12). O envio destas, e de outras que estiverem no *buffer* do nó, é feito caso o enlace com o vizinho apropriado esteja disponível e sua capacidade ainda não tenha sido esgotada (linhas 13 a 19). As funções $enqueue(message, pol)$ (linha 7 e 11) e $dequeue(j, pol)$ (linha 15) são empregadas, respectivamente, para incluir uma mensagem no *buffer* do nó e para extrair uma mensagem que

Dados

N	=	conjunto de nós da rede
M	=	conjunto de enlaces da rede
R	=	conjunto de jornadas para cada destino
Y	=	conjunto de intervalos de tempo
pol	=	política de seleção das mensagens do *buffer* - FIFO, HOP ou MEET
$Neig_i$	=	conjunto de vizinhos do nó i
$myRank_i$	=	identificação do nó i

Variáveis

p	=	0
SET_i	=	$nulo$
$message$	=	$nulo$
$setSize$	=	0
$buffer_i$	=	$nulo$
$bufferUsage_i$	=	0
$bufferSize_i$	=	tamanho do *buffer* do nó i
$status_i[j]$	=	0, $\forall\ j \in Neig_i$
$vecInter_i[j]$	=	(vizID, capacidade, up[], down[]), $\forall\ n_j \in Neig_i$
$vecUsage_i[j]$	=	$nulo$, $\forall\ j \in Neig_i$
$routeTable_i[j][r][y]$	=	$nulo$, $\forall\ j \in N$, $r \in R$ e $y \in Y$
$demandList_i[j][y]$	=	quantidade de mensagens para envio, tal que $j \in N$ e $y \in Y$

tenha um destino tal que a jornada para este passe pelo vizinho j, ambas utilizando uma determinada política *pol*. Esta política pode ser a FIFO, onde a escolha da mensagem no *buffer* é realizada seguindo a ordem em que foram inseridas, a HOP, onde as mensagens são ordenadas priorizando as que são destinadas a nós que demandem o menor número de saltos, e a MEET, onde a ordenação das mensagens é feita beneficiando aquelas destinadas a nós que possuem o menor número de intervalos de tempo de disponibilidade durante sua jornada.

O segundo evento trata das mensagens de controle recebidas. A ação tomada neste caso é verificar para cada destino da tabela de roteamento, representada pela variável $routeTable_i$, qual jornada está menos congestionada e considerar esta como preferencial para o roteamento das próximas mensagens (linhas 25 a 31).

4.2.1 Análise de complexidade

O algoritmo AJRP termina sua execução após processamento do último pulso $t_f \in T$, isto é, a complexidade de tempo é da ordem de $O(T)$. A quantidade total de pulsos (T) depende da DTN em questão. Ou seja, de acordo com os tempos de disponibilidade dos enlaces da rede será identificada a quantidade de pulsos de execução.

```
1  Algorithm AJRP(verInter, routeTable, demList)
2  INPUT:
3     p > 0, SET_j ∈ MSG_i(p), origem_i(SET_j) = (n_i, n_j);
4  ACTION:
5  forall message ∈ SET_j do
6  |  if (message.destId ≠ myRank_i) then
7  |  |  if (bufferUsage_i < bufferSize_i) then
8  |  |  |  enqueue(message, pol);
9  |  |  |  bufferUsage_i ← bufferUsage_i + 1;
10 |  |  end
11 |  end
12 end
13 forall message ∈ demandList_i[j][y], j ∈ N, p ⊂ y, y ∈ Y do
14 |  if (bufferUsage_i < bufferSize_i) then
15 |  |  enqueue(message, pol);
16 |  |  bufferUsage_i ← bufferUsage_i + 1;
17 |  end
18 end
19 forall active (i, j) in p, j ∈ Neig_i do
20 |  SET_i[k] ← nulo, ∀ k < setSize;
21 |  setSize ← 0;
22 |  while ((message = dequeue(j, pol)) ≠ nulo) AND
   |  (setSize ≤ vecInter_i(j).capacidade) do
23 |  |  bufferUsage_i ← bufferUsage_i − 1;
24 |  |  setSize ← setSize + 1;
25 |  |  SET_i[setSize] ← message;
26 |  end
27 |  SEND SET_i to n_j;
28 |  percent ← (bufferUsage_i/bufferSize_i) ∗ 100;
29 |  SEND Control(percent) to n_j;
30 end
31 INPUT:
32    p > 0, Control_i(percent), origem_i = n_j;
33 ACTION:
34 vecUsage_i[j] ← percent;
35 choice ← vecUsage_i[0];
36 forall k ∈ N do
37 |  forall r ∈ R, y ∈ Y, y > p do
38 |  |  if vecUsage_i[j] < choice then
39 |  |  |  choice ← vecUsage_i[j];
40 |  |  end
41 |  end
42 |  routeTable_i[k].choice ← choice;
43 end
```

Algorithm 3: Algoritmo AJRP

Com relação a quantidade de mensagens de controle enviadas pelo AJRP, observa-se que os N nós da rede enviam mensagens para seus vizinhos. Assumindo um grau d de vizinhança, tem-se então uma complexidade de mensagens de controle da ordem de $O(Nd)$. Considerando ainda que a informação possui tamanho total de b bytes, a complexidade de bits é, portanto, $O(Ndb)$.

Considerando cada enlace $e \in M$ possua y_e ativações, e que cada período iniciado por um y possua p_{ey} pulsos, o número de mensagens de controle enviado é $\sum_{e=1}^{|M|} \sum_{y=1}^{|y_e|} p_{ey}$

4.2.2 Exemplo de execução

A Figura 4.7 apresenta um exemplo de execução do algoritmo AJRP, demonstrando a seleção de jornadas alternativas e a troca de mensagens entre os nós A, B, C e D da rede ilustrada. Cada enlace possui uma capacidade máxima para transmitir mensagens de aplicação e está associado um intervalo de tempo de disponibilidade. As mensagens de controle são representadas por setas tracejadas, enquanto que as mensagens de aplicação são representadas por setas contínuas, ambas partindo da origem para o destino. As tabelas de roteamento da Figura 4.6 são utilizadas como entrada para execução do AJRP. Seja A um nó gerador de mensagens e que precisa enviar 3 mensagens para o nó D. Considere ainda que os nós B e C possuem capacidade para armazenamento de 2 mensagens em seus *buffers*.

Na Figura 4.7(a), referente às mensagens trocadas no pulso 1, os nós B e C enviam mensagens de controle para o nó A informando que seus *buffers* estão com 0% de utilização. Neste caso, tanto a jornada via nó B quanto a jornada via nó C podem ser escolhidas, dado que também possuem o mesmo número de saltos para alcançar o nó de destino D. Neste exemplo, o nó A seleciona a jornada via C e envia uma mensagem de aplicação para este nó. No pulso 2 (Figura 4.7(b)), os nós B e C novamente informam ao nó A sobre a utilização de seus *buffers*. No entanto, agora o nó C está com 50% de utilização, enquanto o nó B permanece com a capacidade total disponível. O nó A seleciona então a jornada via nó B e envia uma mensagem de aplicação. No pulso 3 (Figura 4.7(c)), os nós B e C voltam a ter a mesma utilização *buffer*. O nó A, então, continua com a mesma jornada via B e envia a última mensagem de aplicação que possui.

As mensagens de aplicação do nó A ficam armazenadas nos nós B e C até o pulso 7, quando os enlaces (B, D) e (C, D) tornam-se ativos e as mensagens de aplicação podem finalmente chegar ao destino D.

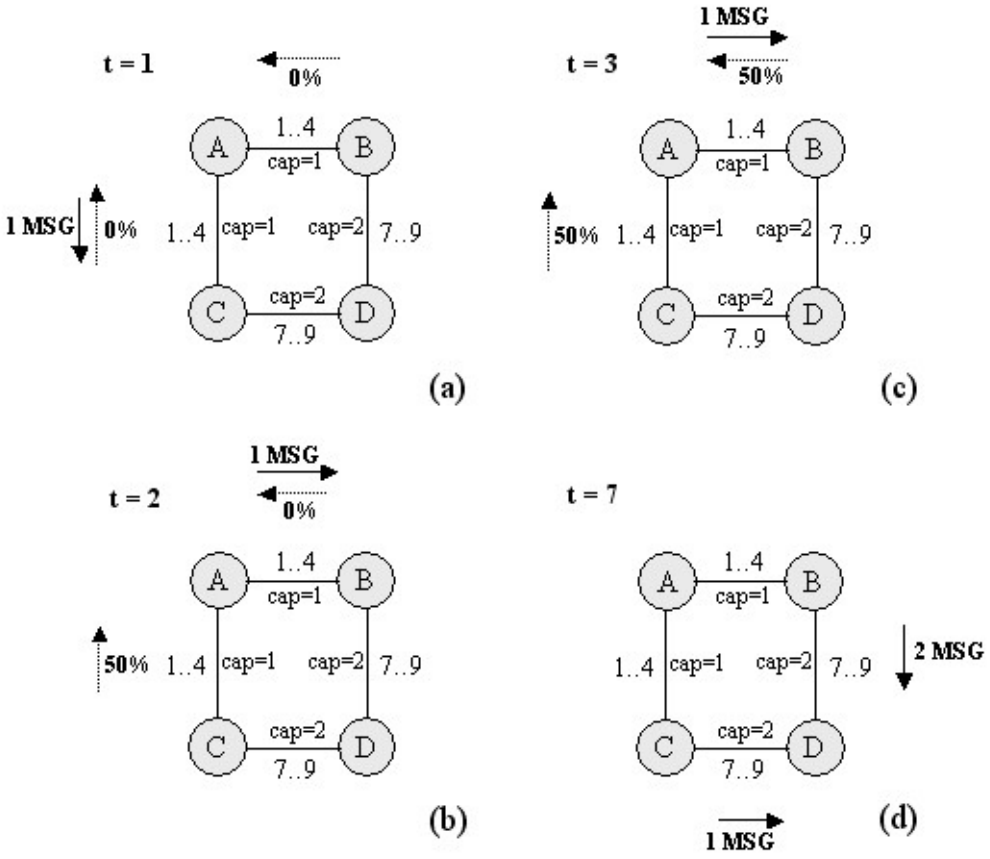

Figura 4.7: Exemplo de execução do AJRP: troca de mensagens

Ressalta-se que neste exemplo foram exibidas apenas as mensagens de controle relevantes para o encaminhamento das mensagens de A até D, ou seja, as mensagens de controle dos nós A e D informando sua utilização de *buffer* foram omitidas.

Capítulo 5

Resultados obtidos

Este capítulo apresenta os experimentos realizados para avaliação do modelo de programação linear inteira, do desempenho do algoritmo DSJ para construção de tabelas de roteamento e do desempenho do algoritmo AJRP para encaminhamento das mensagens até os destinos.

Para obter as soluções ótimas através da formulação matemática em PLI foi utilizado o *solver* CPLEX versão 10.2 [ILOG 2006].

A avaliação dos algoritmos foi realizada através de simulações utilizando um ambiente com 5 computadores conectados em LAN. Cada máquina possui processador Intel Pentium IV 2.8GHz e memória RAM de 512MB, rodando o sistema operacional Ubuntu V3. A implementação foi feita em linguagem C utilizando biblioteca MPI para troca de mensagens. Cada nó das instâncias de rede utilizadas nos testes foi simulado por um processo MPI e alocado em uma das máquinas da LAN.

5.1 Avaliação da construção das tabelas de roteamento

Determinadas topologias de rede foram usadas 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.

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. Na Figura 5.1(a), observa-se que o intervalo (1,10), associado ao enlace (A,B), é um subconjunto do intervalo (1,20) associado ao enlace (B,C). Em seguida, os intervalos foram dispostos tal que a interseção com os intervalos adjacentes fosse apenas parcial. Conforme pode ser visualizado na Figura 5.1(b), a interseção entre os intervalos associados aos enlaces (A,B) e (B,C) resulta no intervalo (5,10). Por último, os intervalos adjacentes estavam totalmente desconexos, ou seja, sem nenhuma interseção. A Figura 5.1(c) mostra que o intervalo (11,20) de disponibilidade do enlace (B,C) inicia apenas após o término do intervalo (1,10) do enlace (A,B).

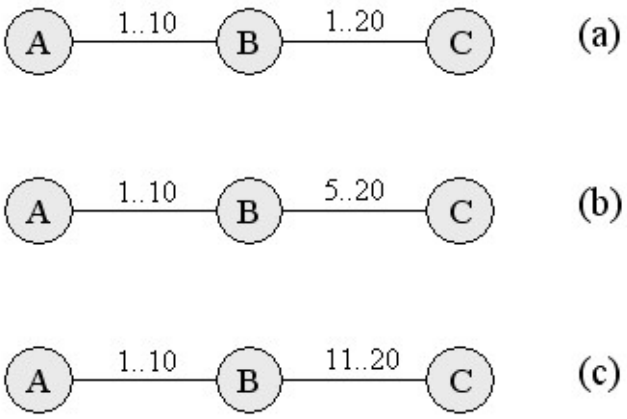

Figura 5.1: Interseções entre intervalos adjacentes

Inicialmente, foi analisada uma topologia linear, 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, conforme pode ser visto na Figura 5.2(a), onde o intervalo (20,30) do enlace (B,C) inicia apenas após o término do intervalo (1,10) dos enlaces (A,B) e (C,D). 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.

Com relação ao tempo para obtenção da tabela de roteamento final, a configuração que apresentou o maior número de ciclos nesta topologia foi com os intervalos também desconexos, mas com os mesmos em sentido crescente. A Figura 5.2(b) apresenta este padrão de conectividade, onde o intervalo (40,50) do enlace (C,D) inicia após o término

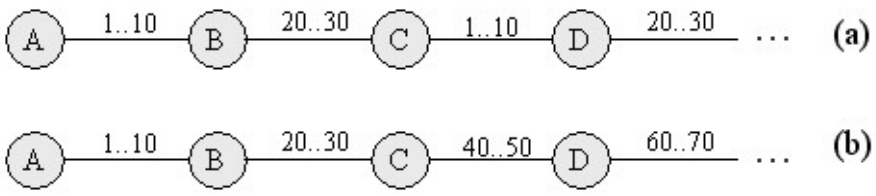

Figura 5.2: Interseções entre intervalos na topologia linear

do intervalo (20,30) do enlace (B,C), que por sua vez apenas inicia após o intervalo (1,10) do enlace (A,B).

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 estas duas topologias foi inferior ao encontrado na topologia linear. A explicação para isso é que na topologia linear existem no máximo dois enlaces adjacentes a um nó e, desta forma, até metade dos enlaces adjacentes pode ficar sem receber mensagens quando utiliza-se o filtro. Quando a quantidade de enlaces adjacentes cresce, a porcentagem de enlaces não utilizados devido ao filtro é reduzida, pois aumenta-se o número de combinações possíveis de intervalos que não estão no passado em relação aos adjacentes. A Figura 5.3 exemplifica, através do mapeamento das mensagens enviadas em cada enlace, o que acontece na aplicação do filtro com combinações distintas para intervalos adjacentes. Cada seta presente na figura representa uma mensagem. O número de nós e o número de enlaces nas Figuras 5.3(a) e (b) são equivalentes, assim como os valores dos intervalos. Como na topologia ilustrada na Figura 5.3(b) os nós dispõem de mais enlaces adjacentes então a quantidade de ciclos para obter a tabela final também é relativamente menor quando comparado com a topologia linear (Figura 5.3(a)), pois o número de nós alcançados a cada ciclo é maior. Devido a esta diferença nas combinações dos intervalos adjacentes, nota-se que na Figura 5.3(a) são enviadas ao todo 3 mensagens, enquanto que na Figura 5.3(b) são necessárias 8 mensagens para atualização das tabelas de roteamento.

Por último, foram analisadas determinadas topologias que representam padrões de conectividade encontrados em sistemas de satélites como, por exemplo, o sistema Iridium [Hubbel e Sanders 1997] [Leopold 1991]. Uma análise sobre a modelagem e roteamento nestas redes pode ser encontrada em [Ferreira 2002] [Ferreira et al. 2002]. Neste trabalho são descritos três padrões que podem ser utilizados para conectar satélites alocados em órbitas adjacentes. Na Figura 5.4 são apresentadas estas três topologias, onde cada satélite é representado por um ponto e os canais de comunicação entre os mesmos são representados

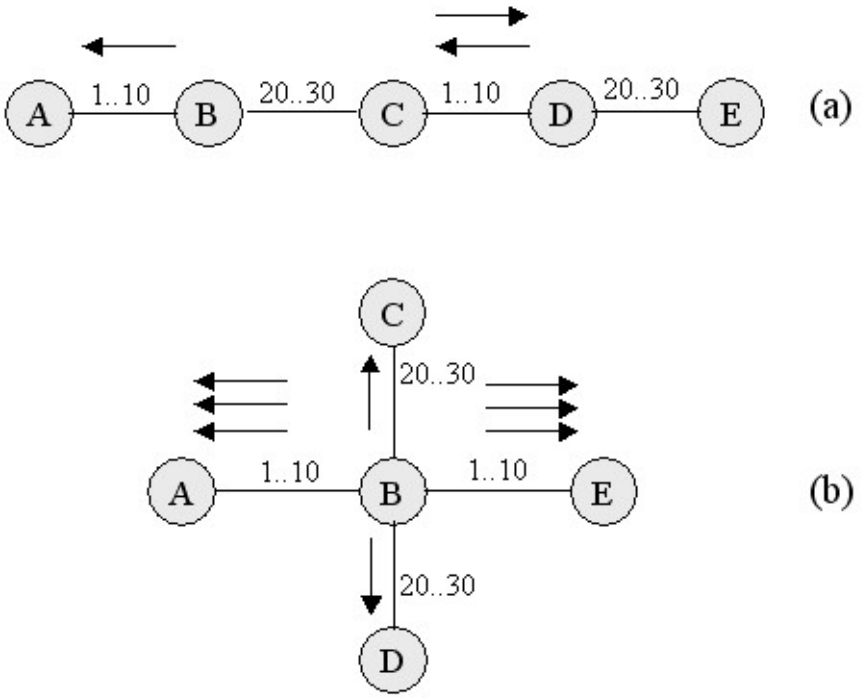

Figura 5.3: Combinações entre intervalos adjacentes

por linhas tracejadas. Nos padrões "W4" (Figura 5.4(a)) e "inclinado" (Figura 5.4(b)), cada satélite dispõe de até quatro enlaces para se conectar aos outros, enquanto que o padrão W3, apresentado na Figura 5.4(c), tem-se no máximo três enlaces para comunicação. Foram considerados nos experimentos que os intervalos associados aos enlaces adjacentes a cada satélite são desconexos, isto é, possuem interseção nula.

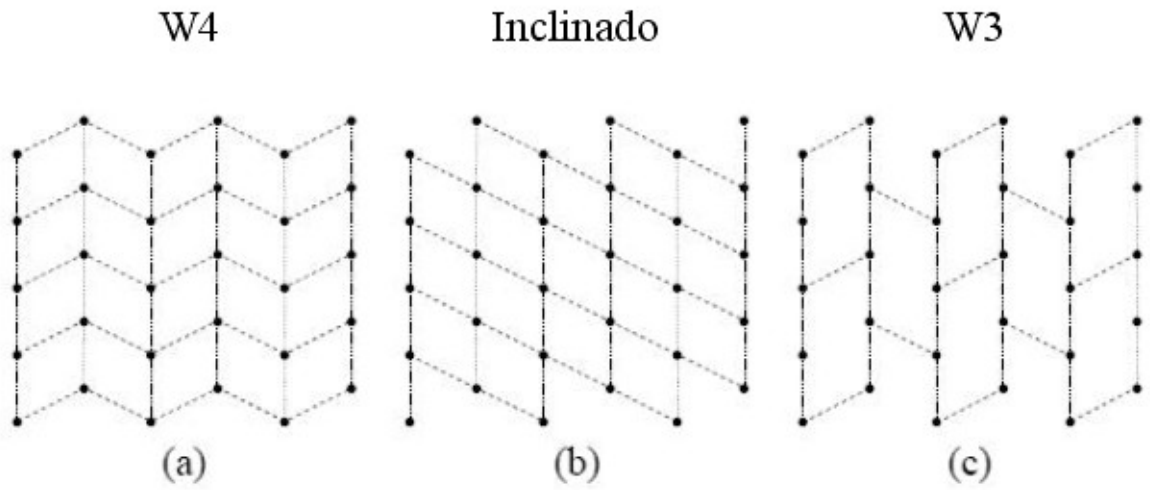

Figura 5.4: Topologia de redes de satélites: W4, inclinado e W3

A Tabela 5.1 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 linear 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
Linear	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 5.1: Redução do número de mensagens em relação à topologia e a quantidade de vértices

Além da redução na quantidade de mensagens trafegadas nos enlaces, 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 quando simplesmente aumenta-se a quantidade de intervalos em todos os enlaces adjacentes, o percentual de redução da quantidade de bits trafegados não se modifica. As Figuras 5.5(a) e (c) mostram, respectivamente, quantas mensagens são enviadas sem e com a aplicação do filtro para uma rede de 3 nós com topologia linear e com apenas um intervalo de disponibilidade associado aos enlaces (A,B) e (B,C). Observa-se que quando o filtro não é aplicado, o nó B envia uma mensagem contendo um intervalo de tempo para o nó A e uma para o nó C. Neste caso, a redução do número de mensagens devido à aplicação do filtro foi de 50%. Nota-se que este mesmo percentual de ganho é obtido com relação à quantidade de bits trafegados, ou seja, a quantidade de intervalos enviados numa mesma mensagem. Ao aumentarmos na mesma proporção a quantidade de intervalos nos enlaces (A,B) e (B,C) (Figuras 5.5(b) e (d)), observa-se que o percentual de ganho permanece em 50%.

Contudo, se for reduzida a quantidade de intervalos com instantes de tempo maiores e acrescida a quantidade de intervalos com instantes de tempo menores, verifica-se que o filtro apresenta um desempenho melhor, pois impedirá que os intervalos menores, que estão no passado em relação aos adjacentes, sejam enviados adiante. Conforme apresentado na Figura 5.6(c) e (d), a redução da quantidade de mensagens para a rede deste exemplo

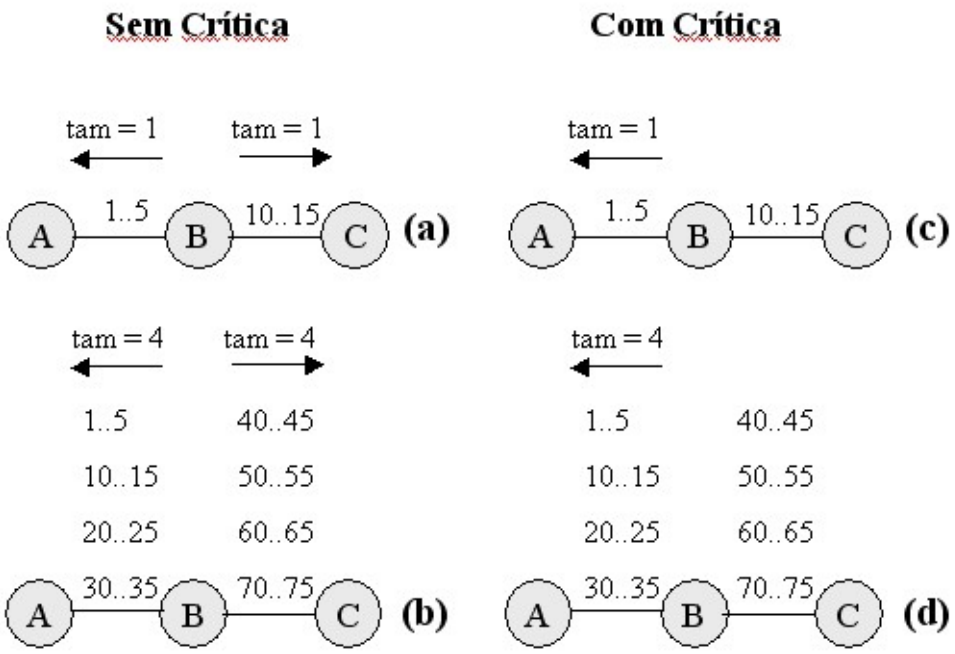

Figura 5.5: Enlaces adjacentes com mesma quantidade de intervalos

permanece em 50%. No entanto, nota-se que sem o filtro são transmitidos ao todo 5 intervalos nas 2 mensagens. Mas, quando o filtro é aplicado, apenas um intervalo é enviado. Ou seja, para este exemplo a redução na quantidade de mensagens foi de 50% e a redução na quantidade de bits foi de 80%.

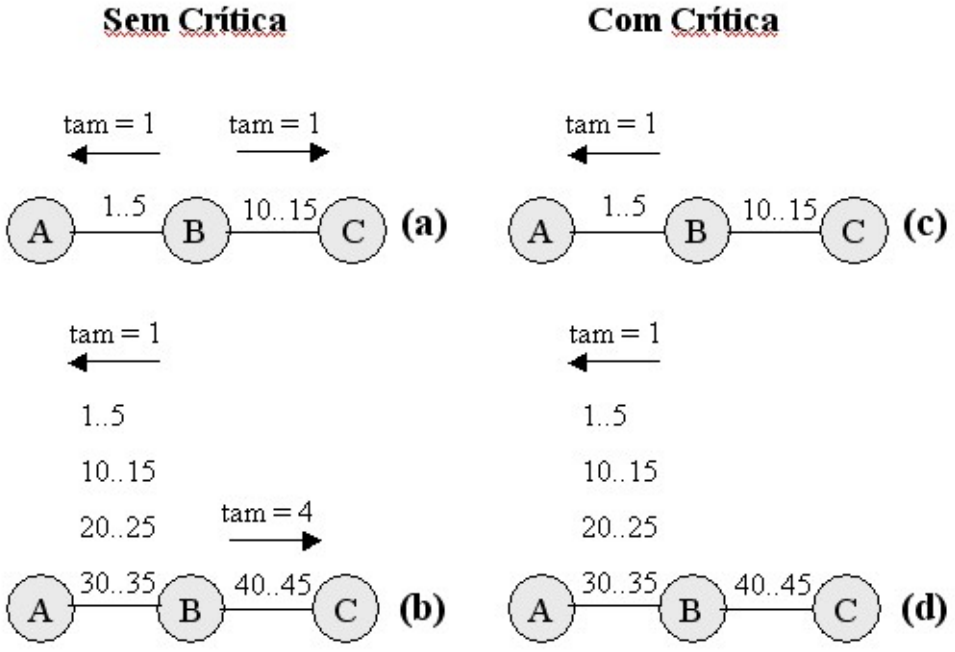

Figura 5.6: Enlaces adjacentes com quantidade de intervalos distinta

Para verificar o comportamento do filtro nesta situação, utilizou-se 1, 2, 4, 8 e 16 intervalos por enlace para as topologias já mencionadas. As execuções foram realizadas com 64 nós. Os valores constantes na Tabela 5.2 mostram que o percentual de redução de bits trafegados chega a quase 90% para a topologia linear quando utilizados 16 intervalos. Já para as topologias que representam padrões de redes de satélites o ganho é inferior, mas atingindo até 76%.

Topologia	Nº de Intervalos				
	1	2	4	8	16
Linear	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 5.2: Redução do número de bits com relação à topologia e a quantidade de intervalos

5.2 Avaliação do encaminhamento de mensagens

Nesta seção são apresentados os resultados com relação ao encaminhamento das mensagens até os destinos, considerando as restrições de largura de banda dos enlaces e da capacidade de armazenamento dos nós.

A seguir, na Seção5.2.1, o modelo de PLI por fluxo proposto neste trabalho é comparado com o modelo de PL proposto em [Jain et al. 2004]. A Seção5.2.2, compara os resultados do algoritmo AJRP com os obtidos pela solução ótima. Na Seção5.2.3, são descritos os experimentos realizados com o algoritmo AJRP utilizando distintas políticas de escalonamento de mensagens. Por último, na Seção5.2.4, são apresentados os resultados do AJRP utilizando *traces* da rede veicular *DieselNet* e comparando-os com outros algoritmos de roteamento para DTNs.

5.2.1 Comparação dos métodos exatos

A rede utilizada para avaliação do modelo de PLI possui 20 nós, cada um está conectado a outros três. O tempo de execução foi dividido em 144 instantes ($T = 144$) e os enlaces foram associados a 1, 3 e 5 intervalos de disponibilidade com duração respectiva de 30, 10 e 5 instantes de tempo em cada intervalo.

Para definir a carga de mensagens na rede adotou-se como medida a quantidade de mensagens criadas em cada nó em determinado instante de tempo para serem entregues a um destino específico. As mensagens foram geradas a cada seis instantes de tempo e com tamanho fixo de 1KB. Ao todo, cada nó envia mensagens para onze diferentes destinos em pulsos distintos. Define-se então que uma carga de *50* mensagens significa que cada nó criará *50* mensagens a cada 6 pulsos para serem entregues a um destino. Em outras palavras, cada nó envia um total de 11 x 50 mensagens durante 66 pulsos.

A solução ótima foi obtida através da execução do PLI apresentado no Capítulo 3, onde foram criadas mais de 150 mil variáveis e foram aplicadas em torno de 70 mil restrições. O tempo necessário para encontrar a solução ótima para as cargas de mensagens utilizadas foi de 1 a 2 segundos.

Para verificar a escalabilidade do modelo de PLI foram realizados testes com carga de 90000 mensagens, ou seja, cada nó gerou 90000 mensagens para 11 nós de destino a cada 6 pulsos. Isto resultou na geração de cerca de 200 milhões de mensagens ao todo. Esta quantidade representa mais de 1000 vezes a carga máxima viável utilizada por Jain *et al.* (2004) para obtenção da solução ótima.

A Tabela 5.3 apresenta a diferença entre a quantidade de variáveis e restrições geradas por cada modelo para distintas cargas de mensagem. Conforme descrito no Capítulo 3, observa-se que, no modelo por fluxo, o número de variáveis e restrições independe da quantidade de mensagens na rede. Já no modelo de Jain *et al.* (2004) o número de variáveis e restrições cresce em função do número de mensagens, implicando em uma maior dificuldade para determinar a solução ótima. Vale ressaltar que uma comparação direta entre as soluções obtidas por ambos os modelos não pode ser realizada, um vez que a funçao objetivo do modelo de Jain *et al.* (2004) está em função do tempo, enquanto a do modelo proposto está em função do número de saltos das mensagens na rede.

Nº Msgs	Nº de Variáveis		Nº de Restrições	
	Formulação por fluxo	Formulação por mensagem (Jain *et al.*)	Formulação por fluxo	Formulação por mensagem (Jain *et al.*)
50	152400	330000	62740	241460
75	152400	495000	62740	360960
100	152400	660000	62740	480460
125	152400	825000	62740	599960
150	152400	990000	62740	719460

Tabela 5.3: Variáveis e restrições geradas pelos modelos

5.2.2 Comparação do AJRP com o ótimo

O algoritmo AJRP foi avaliado utilizando a mesma rede descrita na seção anterior e comparado com a solução ótima. Para isso, foram realizadas reduções na largura de banda dos enlaces, assim como na capacidade de armazenamento dos nós, com o objetivo de verificar um limite superior, ou seja, identificar uma configuração de rede onde não existe solução viável para roteamento sem perda de mensagens. Para tanto, estipulou-se uma carga de 150 mensagens, atribuído um *buffer* de 10MB a cada nó e variada a capacidade do enlace durante cada pulso p em 512KB/t, 384KB/t, 300KB/t e 256KB/t. Verificou-se, então, a inexistência de solução viável quando figura-se uma largura de banda inferior a 300KB/t. A partir desta informação, fixou-se a largura de banda em 384KB/t e, usando a mesma carga de mensagens na rede, variou-se a capacidade de armazenamento dos nós em 10MB, 5MB, 2,5MB, 1,25MB e 640KB. O resultado obtido mostrou que o roteamento é inviável quando os nós possuem *buffer* com tamanho de 640KB. Finalmente, reduziu-se a carga de mensagens para 125, 100, 75 e 50 com o objetivo de avaliar o comportamento do algoritmo quando aplicadas cargas moderadas e baixas na rede.

Para as execuções do AJRP foram utilizadas as tabelas de roteamento geradas pelo algoritmo DSJ, onde foram mantidas até três jornadas por intervalo ($R = 3$) para cada destino. Foi considerado ainda o gerenciamento das mensagens do *buffer* utilizando a ordem FIFO.

A Figura 5.7 apresenta o percentual de mensagens entregues pelo algoritmo AJRP quando variou-se tanto a largura de banda dos enlaces como as cargas na rede. A redução na capacidade dos enlaces demonstra o impacto significativo que ocorre no roteamento das mensagens, onde o percentual de entrega fica em torno de 90% para uma banda de 300KB/t, mesmo com carga baixa na rede. Verifica-se ainda que, para uma carga de 150 mensagens, o percentual de entrega é reduzido. Contudo, este valor ainda é relativamente alto, ficando em torno de 86% na execução com enlaces com capacidade de 512KB/t e 83% com largura de banda de 300KB/t.

Como pode ser visualizado na Figura 5.8, foram utilizados tamanhos de *buffer* distintos e, ainda assim, mais de 95% das mensagens são entregues pelo AJRP quando uma baixa carga é considerada. Além disso, com uma carga moderada de 75 a 100 mensagens, observa-se que o recebimento de mensagens no destino fica em torno de 90%. Por fim, verifica-se que para uma carga de 150 mensagens, o percentual de entrega é inferior chegando a 83% na execução com *buffer* de 10MB e 71% com *buffer* de 1,25MB.

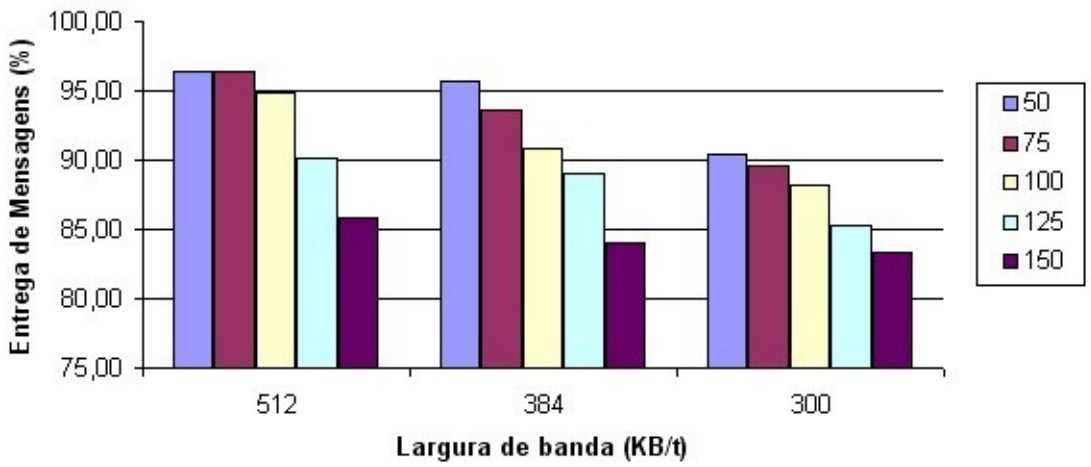

Figura 5.7: Entrega de mensagens variando a demanda e a largura de banda

O impacto das mensagens de controle sobre a lagura de banda foi desconsiderado, uma vez que estas representam apenas 0,004% do tamanho de uma mensagem de aplicação, isto é, um valor ínfimo comparado com um enlace de capacidade de 300KB/t.

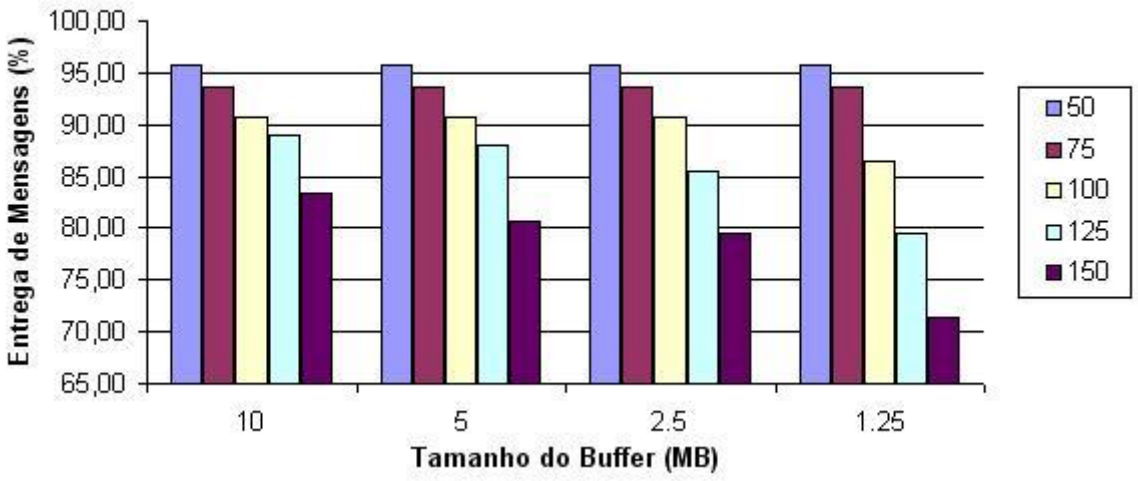

Figura 5.8: Entrega de mensagens variando a Demanda e o tamanho do *Buffer*

5.2.3 Comparação entre as políticas do AJRP

Experimentos foram realizados com o AJRP utilizando os *traces* gerados pela rede veicular *DieselNet* entre 14 de fevereiro e 15 de maio de 2007, onde foram consideradas as mesmas capacidades de *buffer* dos nós, larguras de banda dos enlaces e demandas de mensagens utilizadas em [Balasubramanian et al. 2007]. A Tabela 5.4 apresenta estas

informações. Para evitar a geração de mensagens para destinos que nunca seriam alcançados foram considerados como potenciais destinos apenas os ônibus que circulantes no dia corrente. Definiu-se como carga de mensagem a quantidade de mensagens geradas a cada hora para todos os destinos disponíveis no *trace* do dia corrente.

Primeiramente, foi avaliado apenas o algoritmo AJRP utilizando três políticas distintas para seleção de mensagens no *buffer*. São elas: FIFO, HOP e MEET. Na primeira, a escolha da mensagem no *buffer* é realizada seguindo a ordem em que foram inseridas. Na segunda, as mensagens são ordenadas priorizando as que são destinadas a nós que demandem o menor número de saltos. Na última, a ordenação das mensagens é feita beneficiando aquelas destinadas a nós que possuem o menor número de intervalos de tempo de disponibilidade durante sua jornada.

Para estas execuções do AJRP foram também utilizadas as tabelas de roteamento geradas pelo algoritmo DSJ, mantendo até três jornadas por intervalo ($R = 3$) para cada destino.

Capacidade de armazenamento	40GB
Quantidade total de ônibus	40
Quantidade média de ônibus por dia	20
Tempo entre a geração das cargas de mensagens	1 hora
Tempo considerado para geração mensagens	até 75% do tempo total

Tabela 5.4: Informações sobre o ambiente de teste com os *traces* da DieselNet

A Figura 5.9 ilustra a porcentagem de entrega de mensagens para cada um deles. Como pode ser observado, a seleção através do menor número de saltos conseguiu entregar, para todas as cargas de mensagem avaliadas, o maior número de mensagens aos destinos, seguido pela política MEET e por último a FIFO. Observa-se também que estas duas últimas políticas tiveram desempenhos muito próximos. A explicação para o melhor resultado obtido pela política HOP é exatamente a priorização das mensagens reduzindo a utilização dos recursos da rede. Ou seja, primeiro envia-se as mensagens cujos destinos são os próprios vizinhos. Em seguida, as mensagens cujos destinos são os vizinhos dos vizinhos são selecionadas, e assim sucessivamente.

5.2.4 Comparação entre o AJRP e outros algoritmos

Os resultados obtidos pelo AJRP utilizando a política HOP foram comparados com os obtidos pelos algoritmos RAPID, MaxProp, Spray and Wait, Random e PROPHET, que são algoritmos que foram propostos nos últimos anos com o intuito de otimizar o

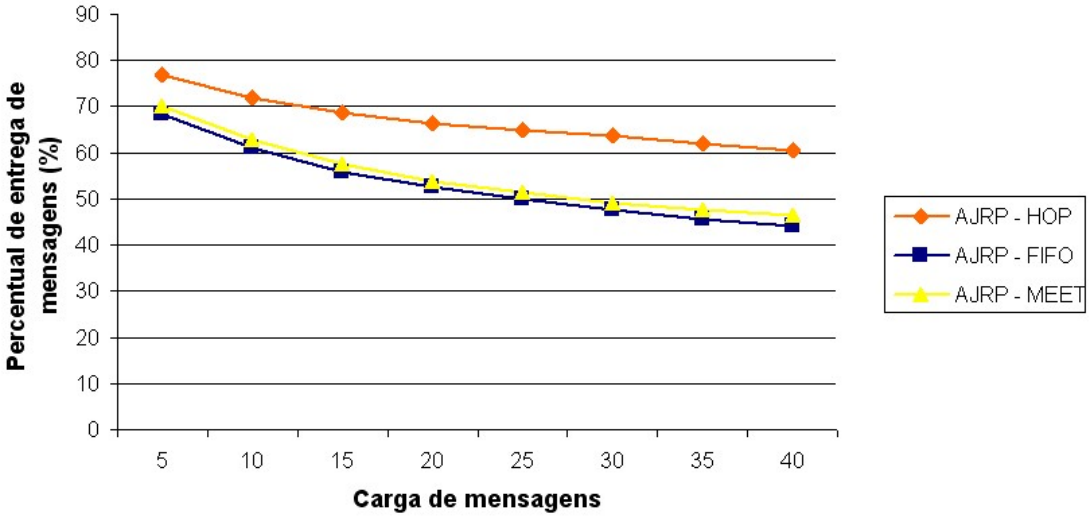

Figura 5.9: Entrega de mensagens variando o algoritmo de seleção de roteamento

roteamento de mensagens em DTNs e foram recentemente implementados e avaliados em [Balasubramanian et al. 2007].

A Figura 5.10 demonstra que o AJRP, mesmo sem realizar nenhum tipo de replicação de mensagens, ou seja, apenas encaminhando cada mensagem a um nó intermediário até alcançar o destino, obteve desempenho superior ao algoritmo Random para todas as cargas de mensagem consideradas. Além disso, observa-se que a diferença na porcentagem de entrega de mensagens pelo AJRP e pelo algoritmo Spray and Wait é relativamente pequena, sendo o AJRP melhor com as cargas de 5 e 40 mensagens, e pior nas demais. Nota-se, contudo, que a curva apresentada pelo algoritmo Spray and Wait apresenta tendência de queda mais acentuada a partir da carga de 35 mensagens, enquanto que a tendência do AJRP é de decréscimo mais suave. Vale ressaltar também que o algoritmo PROPHET não aparece neste gráfico devido ao seu desempenho muito inferior comparado ao algoritmo Random, conforme já descrito em [Balasubramanian et al. 2007].

Como pode ser observado, o AJRP não conseguiu superar os algoritmos Rapid e MaxProp para as instâncias utilizadas. Um dos motivos para isso é que estes dois algoritmos fazem duplicação de mensagens e usam mecanismos para gerenciamento das réplicas. Apesar desta abordagem demandar mais recursos da rede, estas estratégias obtiveram melhor desempenho para a topologia de rede avaliada, favorecendo o percentual de entrega das mensagens.

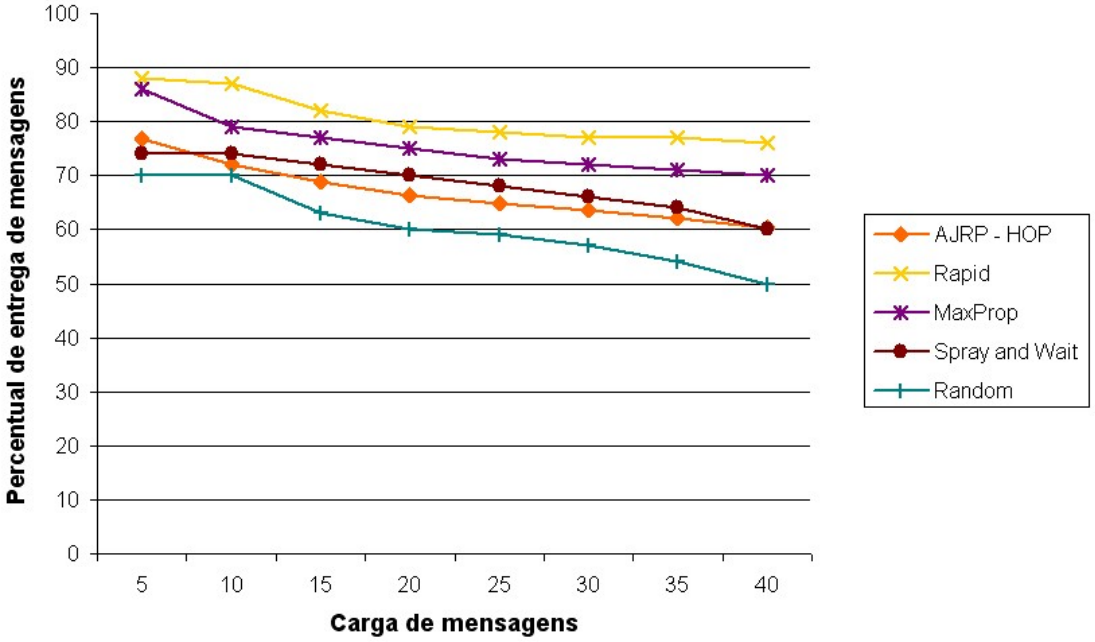

Figura 5.10: Entrega de mensagens de variados algoritmos

Capítulo 6

Conclusões e trabalhos futuros

Neste trabalho foi estudado o problema de roteamento em DTNs previsíveis. Inicialmente, foram elaborados os algoritmos distribuídos DSJ, DEJ e DFJ para construção de tabelas de roteamento nestas redes, que consideram, respectivamente, as jornadas com menor número de saltos, as jornada com entrega mais cedo das mensagens e as jornadas com menor duração de entrega. Nestes algoritmos, 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. Uma análise de conectividade de algumas tolopogias foi realizada, assim como da disposição dos intervalos de tempo nos enlaces. Para mensurar os ganhos obtidos pela aplicação do filtro, executou-se 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, reduzindo em torno de 67% das mensagens e cerca de 90% da quantidade de bits trafegados.

Também foi proposto um modelo para o problema de encaminhamento de mensagens, baseado em grafos direcionados, considerando a intermitência dos contatos entre os nós, além das restrições de *buffer* dos nós e largura de banda dos enlaces. Uma formulação matemática em Programação Linear Inteira foi elaborada com o objetivo de, tendo uma solução ótima de roteamento das mensagens até os destinos, obter um parâmetro para avaliação da heurística AJRP proposta. Por utilizar uma abordagem por fluxos, a formulação matemática elaborada foi capaz de obter soluções ótimas utilizando altas cargas de mensagens, mostrando-se ,por esta razão, mais vantajosa do que as demais formulações encontradas na literatura para o problema.

O algoritmo distribuído AJRP foi desenvolvido com o objetivo de maximizar a entrega

de mensagens aos destinos considerando as restrições de largura de banda dos enlaces e as limitações de capacidade de armazenamento dos nós, utilizando um mecanismo de seleção de jornadas alternativas menos congestionadas. Além disso, diferentes políticas de priorização de envio das mensagens do *buffer* foram implementadas e avaliadas. Este algoritmo foi avaliado comparando os resultados obtidos com a solução ótima para o modelo *offline*, onde foram exploradas variações da capacidade de transmissão dos enlaces e tamanho dos *buffers* dos nós com o objetivo de identificar as configurações extremas para realização do roteamento. Nas execuções realizadas, o AJRP conseguiu entregar mais de 95% das mensagens considerando baixa carga na rede e em torno de 71% com alta carga. Uma avaliação deste algoritmo também foi realizada utilizando *traces* gerados pela rede veicular *DieselNet* e verificou-se que, embora não realize nenhum tipo de replicação de mensagens, seus resultados superaram os de alguns algoritmos de roteamento em DTNs que consideram tal funcionalidade.

Como continuação deste trabalho, pretende-se desenvolver algoritmos de encaminhamento que adotam outras métricas de roteamento como, por exemplo, a de entrega das mensagens considerando seus *deadlines*. Além disso, pretende-se desenvolver mecanismos para melhorar o desempenho do algoritmo de encaminhamento usando, para isso, possivelmente, replicação de mensagens de aplicação e incorporação de mais informações relevantes nas mensagens de controle. Outros pontos importantes a serem pesquisados são a avaliação do desempenho do algoritmo proposto neste trabalho em outras topologias e também a comparação deste com outros algoritmos existentes na literatura.

Referências

[Ahuja e Orlin 1993] Ahuja, R. K., M. T. L. e Orlin, J. B. **Network Flows: Theory, Algorithms, and Applications**. : Prentice Hall, 1993.

[Balasubramanian et al. 2007] Balasubramanian, A.; Levine, B. e Venkataramani, A. **DTN Routing as a Resource Allocation Problem**. In: *ACM SIGCOMM*, p. 373–384, 2007.

[Bui-Xuan et al. 2003] Bui-Xuan, B.; Ferreira, A. e Jarry, A. **Evolving Graphs and Least Cost Journeys in Dynamic Networks**. In: *Modeling and Optimization in Mobile, Ad-Hoc, and Wireless Networks*, p. 141–150, 2003.

[Burgess et al. 2006] Burgess, J.; Gallagher, B.; Jensen, D. e Levine, B. N. **MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks**. In: *IEEE Infocom*, p. 1–11, 2006.

[Cerf 2007] Cerf, V. **RFC4838: Delay-Tolerant Networking Architecture**. IETF, 2007.

[Chen 2005] Chen, C. **Advanced Routing Protocol for Satellite and Space Networks**. Dissertação (Mestrado) — Georgia Institute of Technology, 2005.

[Ekici et al. 2001] Ekici, E.; Akyildiz, I. F. e Bender, M. D. **A Distributed Routing Algorithm for Datagram Traffic in LEO Satellite Networks**. In: *IEEE/ACM Transaction on Networking*, p. 137–147, 2001.

[Ferreira 2002] Ferreira, A. **On Models and Algorithms for Dynamic Communication Networks: The Case for Evolving Graphs**. In: *4o Recontres Francophones sur les Aspects Algorithmiques des Télécommunications*, p. 155–161, 2002.

[Ferreira et al. 2002] Ferreira, A.; Galtier, J. e Penna, P. **Topological Design, Routing and Hand-over in Satellite Networks**. : John Wiley and Sons, 2002.

[Goldman et al. 2006] Goldman, A.; Monteiro, J. e Ferreira, A. **Performance Evaluation of Dynamic Networks using an Evolving Graph Combinatorial Model**. In: *2nd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'06)*, p. 173–180, 2006.

[Hubbel e Sanders 1997] Hubbel, Y. C. e Sanders, L. M. **A Comparison of the IRIDIUM and AMPS systems**. In: *IEEE Network*, p. 12(2):52–59, 1997.

[ILOG 2006] ILOG. **CPLEX 10 user's manual**. 2006.

[Jain et al. 2004] Jain, S.; Fall, K. e Patra, S. **Routing in a Delay Tolerant Network**. In: *ACM SIGCOMM*, p. 145–158, 2004.

[Johnson e Maltz 1996] Johnson, D. B. e Maltz, D. A. **Dynamic source routing in ad hoc wireless networks**. : Kluwer Academic Publishers, 1996.

[Leopold 1991] Leopold, R. J. **Low-earth Orbit Global Celular Communications Network**. In: *IEEE International Conference on Communications*, p. 1108–1111, 1991.

[Lindgren et al. 2004] Lindgren, A.; Doria, A. e Schelén, O. **Probabilistic Routing in Intermittently Connected Networks**. In: *SAPIR Workshop*, p. 239–254, 2004.

[Merugu et al. 2004] Merugu, S.; Ammar, M. e Zegura, E. **Routing in Space and Time in Networks with Predictable Mobility**. Georgia Institute of Technology, 2004.

[Oliveira e Duarte 2007] Oliveira, C. T. e Duarte, O. C. M. B. **Uma Análise da Probabilidade de Entrega de Mensagens em Redes Tolerantes a Atrasos e Desconexões**. In: *XXV Simpósio Brasileiro de Redes de Computadores*, p. 293–305, 2007.

[Oliveira e Albuquerque 2009] Oliveira, E. C. R. e Albuquerque, C. V. N. **NECTAR: A DTN routing protocol based on neighborhood contact history**. In: *24th ACM Symposium on Applied Computing*, p. 359–365, 2009.

[Peleg 2000] Peleg, D. **Distributed Computing: a locality-sensitive approach (Monographs on Discrete Mathematics and Applications)**. : Society for Industrial Mathematics, 2000. ISBN 0898714648.

[Perkins e Bhagwat 1994] Perkins, C. e Bhagwat, P. **Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers**. In: *ACM SIGCOMM Conference on Communications Architectures, Protocols and Applications*, p. 234–244, 1994.

[Perkins e Royer 1999] Perkins, C. E. e Royer, E. M. **Ad-hoc On-demand Distance Vector Routing**. In: *2nd IEEE Workshop on Mobile Computing Systems and Applications*, p. 90–100, 1999.

[Ramanathan et al. 2007] Ramanathan, R.; Basu, P. e Krishnan, R. **Towards a formalism for routing in challenged networks**. In: *ACM MobiCom workshop on Challenged Networks*, p. 3–10, 2007.

[Spyropoulos et al. 2005] Spyropoulos, T.; Psounis, K. e Raghavendra, C. S. **Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks**. In: *ACM WDTN*, p. 252–259, 2005.

[Vahdat e Becker 2000] Vahdat, A. e Becker, D. **Epidemic routing for partially connected ad hoc networks**. Duke University, 2000.