UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TVFP: UM ALGORITMO DE ENTREGA DE MENSAGENS PARA REDES DTN BASEADO EM GRAFOS VARIANTES NO TEMPO

ÁLVARO SHIOKAWA ALVAREZ

ORIENTADOR: PROF. DR. CESAR AUGUSTO CAVALHEIRO MARCONDES

São Carlos – SP

Janeiro / 2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TVFP: UM ALGORITMO DE ENTREGA DE MENSAGENS PARA REDES DTN BASEADO EM GRAFOS VARIANTES NO TEMPO

ÁLVARO SHIOKAWA ALVAREZ

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas Distribuídos e Redes de Computadores
Orientador: Prof. Dr. Cesar Augusto Cavalheiro Marcondes

São Carlos – SP

Janeiro / 2016

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

A473t

Alvarez, Álvaro Shiokawa
TVFP : um algoritmo de entrega de mensagens para redes DTN baseado em grafos variantes no tempo / Álvaro Shiokawa Alvarez. -- São Carlos : UFSCar, 2016.
107 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2016.

1. DTN. 2. TVG. 3. VANET. 4. TVFP. 5. ONE. I. Título.

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Alvaro Shiokawa Alvarez, realizada em 29/01/2016:

Prof. Dr. Cesar Augusto Cavalheiro Marcondes
UFSCar

Prof. Dr. Helio Crestana Guardia
UFSCar

Prof. Dr. Alfredo Goldman Vel Lejbman
USP

À Deus,
a meus pais,
e ao meu orientador Cesar.

AGRADECIMENTOS

Agradeço a meus pais pela minha vida, saúde, força, vigor, inteligência e oportunidades dadas.

Agradeço ao meu professor orientador Cesar Augusto Cavalheiro Marcondes pelos ensinamentos, ajudas e oportunidades dadas, bem como o inestimável auxílio e apoio dado na confecção deste e outros trabalhos.

Agradeço aos professores Hermes Senger e Alexandre Luis Magalhães Levada pelas importantes contribuições dadas para a elaboração deste trabalho durante sua fase de qualificação.

Agradeço a Lourdes Patricia Portugal Poma Costa pelas importantes contribuições e ajudas com a parte técnica deste trabalho, sobretudo pelo conhecimento compartilhado acerca do manuseio e tratamento dos *traces* utilizados nas simulações deste trabalho.

"Perder dinheiro é perder pouco, perder confiança é perder muito, mas perder a coragem é perder tudo, porque perderá a si mesmo. Portanto, mantenha a coragem como o bem mais precioso da vida."

MASUTATSU OYAMA

RESUMO

A perspectiva de um crescimento acelerado da Internet das Coisas tem aumentado a preocupação em tornar as redes sem infraestrutura mais robustas e eficientes energeticamente. Nesse sentido, redes oportunistas e dinâmicas, tais como as redes tolerantes a atrasos e desconexões (DTN) e as redes veiculares ad-hoc (VANET), precisam ser compatíveis com algoritmos de roteamento de mensagens mais eficientes. Exemplos de algoritmos de roteamento em redes DTN incluem o *Epidemic* e o *Spray-and-Wait*, que possuem alto custo de transmissão de mensagens. Com o objetivo de se obter uma comunicação mais eficiente, este trabalho apresenta uma nova abordagem de roteamento de mensagens baseada no uso de grafos dinâmicos, conhecidos no inglês como Time-Varying Graphs (TVG). Na literatura, existem algoritmos que calculam caminhos ótimos em TVGs, como por exemplo, busca do caminho que permite o menor tempo de viagem. Este trabalho propõe a criação de um algoritmo distribuído de roteamento e encaminhamento de mensagens em redes DTN, denominado *Time-Varying Fastest Path* (TVFP), que calcula o caminho a partir do TVG para se entregar a mensagem ao destino no menor tempo possível, utilizando uma única instância da mensagem. O algoritmo foi implementado no simulador ONE e experimentos foram conduzidos para avaliar seu desempenho comparativamente com outros algorimos para DTNs. Os resultados mostram que o TVFP obtem melhor desempenho que os demais algoritmos, fazendo com que a mensagem chegue mais cedo ao destino, e causando uma menor sobrecarga de geração de mensagens redundantes e intermediárias.

Palavras-chave: DTN, TVG, VANET, TVFP, ONE

ABSTRACT

The perspective of an accelerated growth of the Internet of Things, has been increasing the concern on how to make the infrastructure of networks more robust and energetically efficient. In this sense, opportunistic and dynamic networks, such as delay and disruption tolerant networks (DTN) and vehicular ad-hoc networks (VANET), need to be compatible with extremely efficient message routing algorithms. Examples of routing algorithms for opportunistic networks include the Epidemic and the Spray-and-Wait, which have high message transmission cost. With the objective of obtaining a more efficient communication, this work presents a new message routing approach based on dynamic graphs, known as Time-Varying Graphs (TVG). In the literature, there are algorithms which calculate optimal paths within a TVG, for example, the path which allows the minimum travel time for a message. This work proposes a distributed routing algorithm, named Time-Varying Shortest Path (TVFP), which calculates the optimal path in a TVG that allows message delivery within the shortest possible time. The algorithm was implemented in the ONE simulator and experiments were conducted in order to comparatively evaluate its performance against other DTN algorithms. The results show that TVFP has better performance than the other algorithms, delivering messages in shorter times and causing a smaller overhead due to generation of redundant and intermediate messages.

Keywords: DTN, TVG, VANET, TVFP, ONE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

É evidente a importância e presença das redes de computadores no cotidiano das pessoas, assim como tambem na área acadêmica onde há forte investimento e esforço em pesquisa nessa área. Alguns exemplos atuais de pesquisas incluem redes oportunistas, como as redes tolerantes a atraso ou DTNs, que são de interesse para este trabalho. Além disso, a perspectiva de um crescimento acelerado da Internet das Coisas tem aumentado a preocupação em tornar as redes sem infraestruturas, mais robustas e eficientes energeticamente. Nesse sentido, redes oportunistas e dinâmicas precisam ser compatíveis com algoritmos de roteamento de mensagens mais eficientes.

É conhecido o fato de que, por possuírem uma estrutura estática, redes de computadores em geral são representadas e modeladas por estruturas de grafos estáticos, utilizando-se da abstração de vértices mapeados em roteadores ou nós, e arestas mapeadas como enlaces e conexões. Porém, em casos extremos, como as **redes tolerantes a atraso e desconexões** (**DTN**), a representação usual de grafos pode ser inadequada, sendo necessária uma nova abordagem que acompanhe o dinamismo destas redes.

O estudo das redes dinâmicas, como as **redes tolerantes a atraso e desconexões** (***Delay and Disruption Tolerant Networks*** - **DTN**), é de grande importância pois, diante de situações envolvendo conflito, por exemplo guerra em território inimigo, ou emergências provenientes da ocorrência de acidentes, compreende-se a importância da transmissão efetiva e correta de informação, uma vez que a mesma pode salvar vidas. Nessas situações a infraestrutura de comunicação usual, e.g. redes cabeadas, wi-fi, etc., pode estar danificada, comprometida ou até mesmo indisponível. Portanto, utiliza-ser de DTNs pode ajudar nesse aspecto.

Outras redes dinâmicas de interesse a se estudar são as **redes veiculares ad hoc** (***Vehicular Ad-hoc Network*** - **VANET**), que possuem diversas questões a se considerar, como o fato de

conseguir enviar um dado rapidamente de um veículo para outro veículo destino, considerando que, em geral, não existe infraestrutura na estrada, e portanto a rede formada entre veículos não é fixa.

Assim, um tipo de representação que tem sido usada mais recentemente para modelar as redes dinâmicas são os **grafos dinâmicos que variam no tempo**, ou **grafos variantes no tempo** (***Time-Varying Graph*** - **TVG**). Isso possibilita o desenvolvimento de aplicações mais efetivas usando algoritmos cientes de redes dinâmicas, propiciando melhores algoritmos de roteamento em VANETs e DTNs do que algoritmos como de Epidemic e Spray-and-Wait. Esse dinamismo no tempo permite usar a informação temporal, de que no futuro, uma determinada aresta vai estar disponível. Isso garante uma entrega da mensagem, sem o custo de gerar *overhead* extra com múltiplas cópias, só pelo fato de aguardar o momento oportuno.

Neste trabalho, o formalismo dos TVGs será empregado para criar um novo algoritmo distribuído de roteamento *single-copy* para DTNs, denominado ***Time-Varying Fastest Path*** (**TVFP**). Esse algoritmo inicia a partir de um TVG, que representa as interconexões temporais de uma rede DTN formada de ônibus, cuja mobilidade é descrita em um *trace* externo, calcula o **melhor caminho** ao longo da linha do tempo, e garante a entrega da unica mensagem ao destino no **menor tempo possível**. Para avaliá-lo, implementamos o algoritmo no simulador de redes DTN, ***Opportunistic Network Environment*** (**ONE**), realizando simulações comparando seu desempenho com dois algoritmos amplamente usados: ***Epidemic*** e ***Spray-and-Wait***. A nossa solução proposta com o TVFP requer cenários de mobilidade determinísticos e uma mensagem de tamanho pequeno para o seu funcionamento. Os resultados de nossos testes com o TVFP mostram que o mesmo consegue entregar a mensagem ao destino no menor tempo possível, igualando-se aos outros dois algoritmos em tempo de entrega, porém com *overhead* de geração de mensagens reduzido a uma única mensagem na rede.

1.1 Objetivos do trabalho

Em resumo, o objetivo desse trabalho é combinar a técnica teórica de grafos que variam no tempo com roteamento em redes oportunistas. Essa combinação se dá pelo do uso de estruturas de dados que modelam os TVGs. Entre os resultados obtidos, estudamos a partir de simulação, uma maneira adequada de estruturar e armazenar os TVGs, de modo que eles representem fielmente as intercomunicações típicas de uma rede VANET. Adicionalmente, propusemos, e implementamos em um simulador, um algoritmo novo, o qual chamamos de ***Time-Varying Fastest Path*** (**TVFP**), para encontrar o melhor caminho, sendo este o caminho que leva a mensagem

ao destino no menor tempo possível, em redes dinâmicas usando-se essa estrutura de dados proposta. Vale ressaltar que este caminho não necessariamente é o caminho mais curto, em termos de *hops*. Validamos e avaliamos este trabalho, do ponto de vista de desempenho, comparando o TVFP com outros algoritmos conhecidos de roteamento DTN em redes oportunistas, por exemplo, ***Epidemic*** e ***Spray-and-Wait***.

1.2 Organização do trabalho

Este trabalho está organizado em 6 capítulos mais Anexos.

O capítulo 1 introduziu o leitor ao contexto do trabalho, bem como apresentou a proposta desenvolvida.

O capítulo 2 discute e apresenta conceitos teóricos relevantes à nossa proposta.

O capítulo 3 discute sobre trabalhos relacionados e relevantes à nossa proposta.

O capítulo 4 descreve o nosso algoritmo TVFP, o funcionamento do simulador ONE, e como ambos são amplicados dentro de um processo maior para minerar dados a fim de gerar TVGs, que posteriormente são utilizados em roteamento.

O capítulo 5 discute sobre resultados obtidos mediante os experimentos feitos com o TVFP no simulador ONE.

O capítulo 6 conclui este trabalho, bem como apresenta sugestões para melhorias como trabalhos futuros.

Nos Anexos, discutimos detalhes técnicos, desde a sintaxe dos *traces* e grafos TVG utilizados, até parâmetros e detalhes de implementação do simulador ONE, úteis para nosso trabalho. Por fim, discutimos alguns detalhes de implementação do TVFP relevantes ao roteamento no ONE.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo introduzir, por meio da revisão bibliográfica conduzida neste trabalho, conhecimentos específicos que servem para atualizar o leitor com relação à proposta. Para tanto, descreve-se resumidamente o contexto de **redes tolerantes a atrasos e desconexões** (***Delay and Disruption Tolerant networks***) ou simplesmente **DTN**. Em seguida, explica-se com detalhes a fundamentação teórica referente aos **grafos que variam no tempo**, conhecidos como ***time-varying graphs*** ou apenas **TVG**. E conclui-se descrevendo um algoritmo, recente na literatura, que faz uso desse TVG para encontrar o caminho que garanta a entrega da mensagem ao destino, partindo de uma origem, de modo que a mesma possua o menor tempo de viagem possível.

2.1 Redes tolerantes a atrasos e desconexões (DTN)

Ao longo da evolução de redes móveis, pode-se notar que as **redes móveis ad hoc** (***Mobile Ad hoc Nework*** - **MANET**), foram estudadas na literatura em primeiro, e englobavam veículos aéreos e terrestres que poderiam sofrer de desconexão temporária, portando tendo sua comunicação interrompida temporariamente. Entretanto, seus algoritmos não toleravam interrupções longas, por exemplo de minutos, horas ou dias.

As **redes veiculares ad hoc** (***Vehicular Ad hoc Network*** - **VANET**) são uma especialização das MANETS, ou seja, são redes ad hoc, independentes de infraestrutura, porém utilizando-se somente de veículos terrestres. Essa especialização é diferente de outros tipos de MANETs envolvendo desde pessoas até comunicações interplanetárias.

Em geral, conceitua-se os nós em uma VANET como sendo constituidos de veículos equipados com dispositivos sem fio ou tecnologias de interface de rádio para que haja comunicação

sem fio entre os veículos.

Por serem uma especialização das MANETs, a topologia da rede em VANETS, tende a ser mais dinâmica, com a velocidade dos veículos sendo um fator preponderante. De tal modo que, dependendo da interface sem fio, se os veículos estiverem no alcance um do outro, é possível transferir e receber dados. Em outras palavras, os veículos, ou nós móveis, atuam como encaminhadores de dados, atuando simultaneamente como cliente e servidor.

As VANETs executam algoritmos de roteamento de ambiente distribuído, dada a natureza da topologia da sua rede, sendo possível aplicar algoritmos de roteamento de DTNs sobre as mesmas.

Finalmente, foram concebidas as redes DTN que trabalham, que de acordo com (CASTEIGTS, 2011) **redes tolerantes a atrasos e desconexões** (***Delay and Disruption Tolerant Networks*** - **DTN**), são redes de comunicação dinâmicas, oportunistas, sem infraestrutura, e que trabalham com o extremo de tempo de desconexão, realmente tolerando atrasos, e tendo como característica essencial o fato de não possuem rota fim-a-fim fixas, diferindo assim, de uma comunicação típica, onde uma mensagem vai de um ponto a outro passando por um número fixo, estável e conhecido de saltos (os chamados *hops*).

Em uma DTN, um nó pode permanecer com a mensagem a ser transmitida por longo tempo e, eventualmente, após uma longa movimentação, ao conectar-se a outro nó pode transmitir a mensagem. Este nó receptor pode fazer o mesmo, propagando a mesma mensagem para outro nó, e assim por diante. Ou seja, a mensagem original eventualmente chegará a seu destino, mas sem nenhuma garantia. Esse paradigma de roteamento (às vezes também referenciado como encaminhamento) que é baseado no armazenamento, transporte e encaminhamento de mensagens é conhecido como ***store-carry-and-forward***. No trabalho de (PORTUGAL-POMA, 2013), este paradigma é referido como **paradigma baseado em armazenamento de mensagens**.

De acordo (PORTUGAL-POMA, 2013), o paradigma ***store-carry-and-forward*** não usa tabelas de rotas para encaminhar suas mensagens, realizando, ao invés disso, um encaminhamento da mensagem de nó em nó, ou de *hop* em *hop*, conhecido como **encaminhamento progressivo (*hop-by-hop*)** da mensagem. Cada nó repassa a mensagem a nós vizinhos dentro do seu raio de alcance, e estes mesmos nós continuam repassando a mensagem a outros nós em seu alcance, sucessivamente, até que a mesma chegue nó destino e sendo entregue ao mesmo. Se, por exemplo, não há algum nó vizinho para o qual possa ser repassada a mensagem, ou seja, quando não há algum nó vizinho que possa ser usado como nó retransmissor, então a mensagem fica armazenada até que haja uma nova oportunidade de contato, e consequentemente, retransmissão.

Há duas estratégias comumente utilizadas por estes nós para o roteamento das mensagens, em termos de algoritmos de roteamento, ambas descritas em (PORTUGAL-POMA, 2013, 2014):

- ***Single-copy***: Conhecidos como algoritmos de roteamento por encaminhamento ou *forwarding*, mantem uma **única** cópia da mensagem em toda a rede. Durante o encaminhamento da mensagem, cada nó repassa a mesma para um nó vizinho, excluindo logo em seguida a sua cópia da mensagem que se encontra em sua memória, sucessivamente, até que a mensagem chegue ao nó destino.

  Como exemplos de algoritmos ***single-copy*** temos o algoritmo ***FirstContact***, implementado no simulador ONE, e utilizado em parte dos experimentos feitos neste trabalho. Tal qual os algoritmos que tentam enviar de uma origem para um destino, o *FirstContact* procura repassar uma única mensagem, sem cópias e sem heurísticas de escolha do próximo nó de repasse.

- ***Multi-copy***: Conhecidos como algoritmos de roteamento por **replicação** de mensagens, estes algoritmos repassam múltiplas cópias da mensagem em cada transmissão para a ou mais nós vizinhos, sende estes nós intermediários no caminho até o nó destino aonde se deseja levar a mensagem. É um processo que se repete sucessivamente, visando disseminar cópias ao longo da rede, aumentado assim as chances da mensagem ser entregue ao nó destino, e reduzindo o atraso de envio.

  O exemplo fundamental de algoritmos *multi-copy* é o algoritmo de disseminação ***Epidemic*** (VAHDAT; BECKER, 2000), onde o envio das mensagens é feito repassando cópias das mensagens de um nó para seus vizinhos conforme as oportunidades de contato deste nó com seus vizinhos, tal qual ocorre na disseminação de um vírus em uma epidemia. Este processo de repasse continua indefinidamente, até que a mensagem seja entregue ao nó destino. Devido a este comportamento epidêmico, o *overhead* de geração de mensagens decorrente da inundação das cópias na rede é significativo. Assim, o Epidemic é mais adequado para redes esparsas e menos densas em quantidade de nós. Além disso, dada a inundação, em geral, uma das mensagens, chega no menor tempo possível ao destino.

  Outro exemplo é o algoritmo de disseminação ***Spray-and-Wait*** (**SaW**) (SPYROPOULOS; PSOUNIS; RAGHAVENDRA, 2005), cujo comportamento de envio de mensagens é idêntico ao Epidemic, ou seja, cópias das mensagens de um nó são repassadas conforme aparecem oportunidades de contato com este nó. A diferença em relação ao Epidemic se dá através de um mecanismo de controle que estabelece um limite máximo, ou limitante máximo, de cópias da mensagem que podem existir na rede, ou seja, ele dita a quantidade máxima

de cópias da mensagem que podem existir na rede, de modo que nunca existirá uma quantidade de cópias na rede maior que o valor estabelecido pelo limite máximo mencionado. Portanto, para que haja um controle do espalhamento das mensagens na rede, e consequentemente, um controle de inundação, isso é feito para reduzir o *overhead* produzido pela geração destas. Em resumo, o SaW promove um espalhamento controlado de cópias de mensagens na rede. O seu funcionamento se dá ao estabelecer um limite máximo de cópias da mensagem original que podem existir na rede, onde este valor de número de cópias vai sendo reduzido em cada transmissão de uma cópia da mensagem. Quando este valor é igual a um, os nós apenas transmitem a mensagem diretamente para o nó destino, gerando uma cópia da sua mensagem e entregando-a apenas ao nó destino. Na transmissão *direct delivery* o nó com uma cópia da mensagem, e contador de cópias igual a 1, ele faz Este valor de limitante do número máximo de cópias se aplica apenas às cópias geradas pelo SaW durante a simulação, não contabilizando no seu contador de cópias a instância inicial da mensagem que é gerada na origem.

Em relação a este controle do número de cópias, Spray-and-Wait pode operar em dois modos, **normal** e **binário**. No primeiro a redução do número de cópias é sempre feita em unidades (um a um), enquanto que no segundo, o número de cópias é divido pela metade. Ou seja, metade do valor de cópias fica com nó transmissor da mensagem, e a outra metade do valor fica com o nó que recebeu a cópia. Este modo binário promove um menor *overhead* de geração de mensagens na rede. Desse modo, nos experimentos deste trabalho optamos por utilizar o SaW em modo binário.

As figuras 2.1 e 2.2 mostram exemplos do funcionamento dos algoritmos *Epidemic* e SaW.

Na figura 2.1, ocorre o processo de disseminação de cópias partindo do nó A, que é o nó origem, com destino para os nós F e G. A rede se encontra em estados diferentes no decorrer do tempo, por exemplo, no início, A se conecta com os nós B e C, no instante $t1$ B e C disseminam suas cópias para E e D, respectivamente, e no instante $t3$ F e G já se encontram com suas cópias da mensagem.

Na figura 2.2 são exemplificados os funcionamentos dos modos básico e binário do SaW, bem como a transmissão *direct delivery*. O nó origem é A e o nó B é o destino. O nó A, no tempo inicial $t0$ contém um total de 5 mensagens e, conforme vai se conectando com demais nós, esse valor vai sendo reduzido. Quando um nó só possui uma mensagem, como é o caso do nó d, ele só transmite ao destino B por *direct delivery*.

Considerando o acima exposto, o algoritmo TVFP que desenvolvemos para este projeto apresenta vantagens em relação aos algoritmos de roteamento em DTN expostos nesta seção.

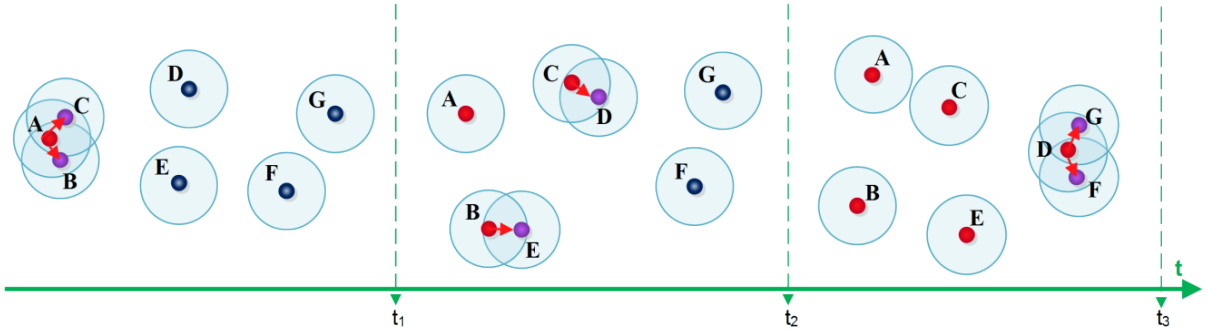


Figura 2.1: Mensagem com origem no nó A, disseminando-se a partir deste - Figura extraída de (PORTUGAL-POMA, 2013)

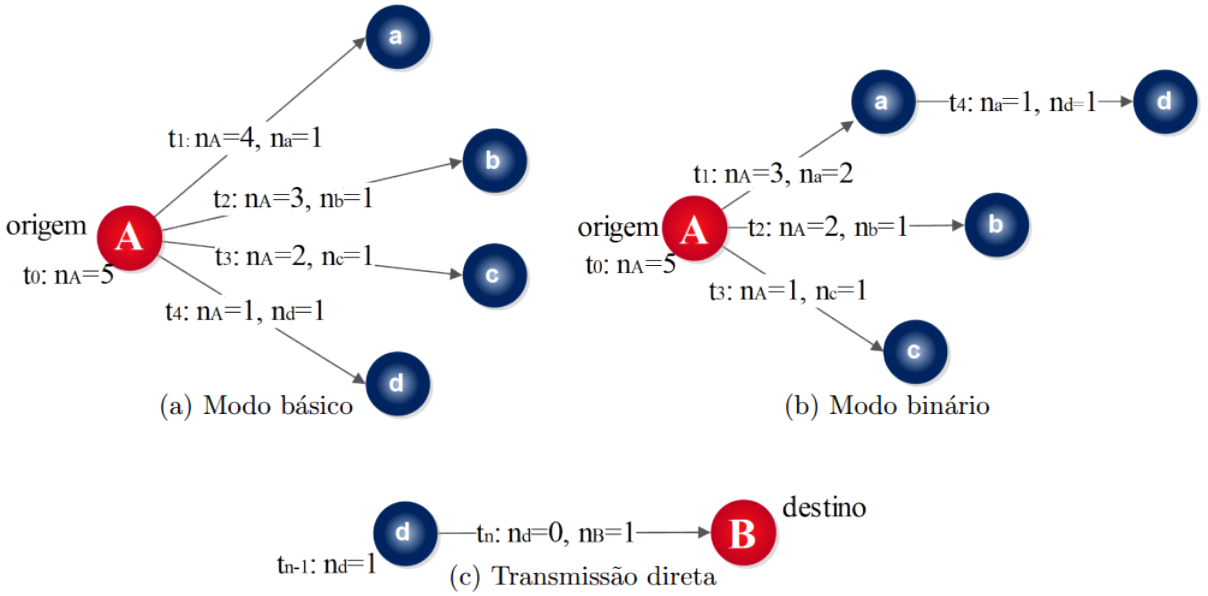


Figura 2.2: Modos de funcionamento do *Spray-and-Wait* - Figura extraída de (PORTUGAL-POMA, 2013)

Devido ao fato do TVFP ser um algoritmo *single-copy* que, portanto, utiliza apenas uma única instância da mensagem e, consegue reduzir o *overhead* causado pela geração das mensagenm. Além disso, evita que a mensagem seja perdida, uma vez que ela, percorrendo um caminho extraído de um TVG, tem a sua entrega ao destino garantido, e no menor tempo possível. Discutimos mais acerca dos experimentos e resultados feitos comparando o desempenho do TVFP com demais algoritmos no capítulo 5.

Por fim, dada a dinamicidade das DTNs e VANETs, com suas topologias com estrutura variável, argumenta-se que podem ser modeladas como um grafo dinâmico de estrutura de vértices e arestas variável ao longo do tempo. Além disso, os algoritmos de roteamento em DTNs podem ser aplicados a VANETS, pois a tambem pode-se aplicar à topologia ad mesma o paradigma *store-carry-forward*. Existem algoritmos que buscam alguma otimização em TVGs, como o do trabalho relacionado à nossa proposta, e se redes dinâmicas podem ser modeladas

como TVGs, esses algoritmos tambem são aplicáveis a estas redes. Na sequência, na seção 2.2, explicamos em detalhes conceitos teóricos sobre grafos dinâmicos TVGs, bem como uma forma de representá-los. Tais conceitos posteriormente poderão ser aplicados diretamente ao problema de encaminhamento em DTNs, apresentando melhorias aos trabalhos relacionados citados nesta seção.

2.2 Grafos variantes no tempo (TVG)

O conhecimento das propriedades de conectividade das redes, e dos nós de rede, permite o projeto de protocolos de roteamento mais eficazes para ambiente específicos (VENDRAMIN, 2012). No contexto de redes de topologia dinâmica, técnicas de teoria dos grafos aplicadas para modelar redes estáticas não são suficientes.

Uma abordagem básica para o estudo das propriedades de conectividade das redes dinâmicas é a amostragem do estado do grafo de conectividade, capturando instantes de cada intervalo fixo de tempo (PALLIS, 2009). Mesmo quando a amostragem do grafo de conectividade permite a análise da variabilidade de métricas de conectividade no tempo, esta metodologia tem as suas limitações quando o problema sendo tratado for a descoberta e otimização de rotas entre nós origem e destino. Ou seja, embora seja possível capturar o estado do grafo de conectividade em algum instante no tempo, e do mesmo extrair alguma métrica referente às conexões entre nós no tempo, é uma metologia limitada para se descobrir e otimizar rotas entre nós origem e destino em um grafo variante no tempo.

Em razão disso, técnicas foram projetadas para adicionar informações da conectividade entre pares de nós, tais como: i) intervalos de presença das arestas e ii) tempos gastos para atravessar as arestas. Estas informações são valiosas, especialmente em redes com um comportamento determinístico (por exemplo, redes de satélites LEO (FERREIRA; GALTIER; PENNA, 2002)), onde o processo de descoberta de rotas poderia ser otimizado para encontrar rotas com o número mínimo de *hops*, tempo de chegada e atraso na entrega de mensagens (BUI-XUAN; FERREIRA; JARRY, 2003).

Para aproveitar as informações temporais de conectividade, diferentes abordagens foram propostas (KEMPE; KLEINBERG; KUMAR, 2002; FERREIRA, 2002, 2004; HOSSMANN; LEGENDRE; SPYROPOULOS, 2009). A maioria delas aplicadas em duas etapas principais (BASAGNI, 2013): i) criar uma representação gráfica (e.g. usando sequências de grafos instantâneos ou agregação de grafos) com informação temporal e/ou dos atributos das entidades e, ii) abstrair as propriedades de conectividades a partir da representação gráfica conceitual resultante.

Grafos variáveis no tempo (***Time Varying Graph*** - **TVG**) (CASTEIGTS, 2011) foram introduzidos como uma forma simples para modelar a complexidade da evolução, ao longo do tempo, das propriedades de conectividades de redes dinâmicas que apresentam algum determinismo. Assim, um TVG consiste na unificação de conceitos e definições matemáticas acerca de uma modelagem de grafos de estrutura de vértices e arestas variável no tempo, que podem ser aplicados em DTNs, redes oportunistas e redes complexas do mundo real.

Portanto, enquanto um grafo estático é utilizado para representar uma rede que não varia com o tempo, um grafo dinâmico é a forma natural de se representar uma rede dinâmica, que varia com o tempo.

A seguir, serão apresentados alguns conceitos relacionados a TVGs, englobando a sua modelagem matemática, uma forma para representar, visualmente, um TVG, e por fim, conceitos sobre as jornadas, que são caminhos percorridos ao longo do tempo em um TVG.

2.2.1 Modelagem matemática de um TVG

No tocante a representação de um sistema dinâmico, este pode ser estudado durante um determinado período de tempo ou tempo de vida τ o qual pertence ao domínio do tempo $\mathbb{T}$ ($\mathbb{N}$ ou $\mathbb{R}^+$ para sistemas discretos e contínuos no tempo, respectivamente).

A representação de um TVG é formada por: um conjunto de entidades (vértices) V e um conjunto de relações (arestas) entre entidades E, as quais podem apresentar rótulos pertencentes a um alfabeto L ($E \subseteq V \times V \times L$). O conjunto E permite múltiplas relações entre um par de entidades, desde que as relações tenham rótulos distintos. A utilização de um alfabeto é opcional e é útil para expressar as propriedades ou attributos, os quais podem ser multivalorados ou tuplas de valores, das arestas entre vértices. Assim, por exemplo, um elemento de um alfabeto pode ser <link de satélite; 44 MHz>, significando respectivamente, essa aresta como o meio de comunicação que conecta os vértices e a largura de banda da comunicação. A figura 2.3 exemplifica a definição de uma aresta de um TVG, onde o rótulo do alfabeto L é *car*.

Figura 2.3: Aresta de rótulo *car* entre nós u e v

Em resumo, um TVG de um sistema pode ser definido como uma quíntupla $\mathcal{G} = (V, E, \tau, \rho, \zeta)$, onde:

- V**:** Conjunto de entidades (vértices), como descrito anteriormente.
- E**:** Conjunto de relações (arestas) entre entidades, como descrito anteriormente.
- τ**:** Tempo de vida, ou evolução do sistema, como descrito anteriormente.
  - Do ponto de vista do tempo de uma simulação feita no ONE, se modelarmos a dinâmica de uma rede DTN, presente em uma simulação no ONE, pode-se considerar o tempo da simulação como equivalente ao tempo de vida τ do TVG.
- ρ**:** É uma **função de presença** que indica se uma aresta existe ou está disponível em determinado instante ou intervalo de tempo do ciclo de vida do sistema τ. Formalmente ρ está definido como: $E \times \tau \rightarrow \{0,1\}$, onde **0** indica aresta **indisponível** e **1 disponível**.
- ζ**:** é uma **função de latência** que indica o tempo gasto para atravessar uma aresta de E, começando a travessia em algum momento do tempo de vida do sistema τ e considerando que a latência de uma aresta pode variar no tempo.

Desse modo, considerando o exposto acima e no contexto da transmissão de uma mensagem, **a entrega efeitiva de uma mensagem** está sujeita às funções de presença e de latência das arestas, que compõem o caminho de nós intermediários entre a origem e o destino. Assim, por exemplo, a entrega efetiva de uma mensagem enviada em um tempo t, passando por uma aresta e, está sujeita a restrições impostas pela função de presença ρ, que exige que $\rho(e)$ retorne 1, ou seja, que a aresta e esteja disponível (pois o valor da função ρ aplicado à aresta e retorna 1, ou seja, está disponível) durante o intervalo de tempo $[t, t+\zeta(e,t)]$, o que quer dizer que a aresta e deve estar disponível como mínimo desde o início da transmissão e durante o tempo que demora para transmitir a mensagem (determinado pela função de latência ζ), de modo contrário, a transmissão é interrompida.

Um TVG pode ter suas relações representadas como na figura 2.4, onde os intervalos de tempo sobre cada aresta representam os períodos de tempo quando a mesmas estão disponíveis, isto é $\cup(t \in \tau : \rho(e,t) = 1)$.

O formalismo do TVG pode representar diversos cenários, desde redes de transporte até redes de comunicação, sistemas complexos, ou redes sociais. Os exemplos nas figuras 2.5 e 2.6 representam cenários possíveis de serem modelados como TVGs, tendo cada um o seu alfabeto de rótulos L com significados específicos para o domínio que o TVG foi modelado.

No TVG da figura 2.5 uma aresta ligando um nó u a outro nó v representa a **possibilidade** de um agente se deslocar de u para v. Neste, caso cada nó representa uma localidade (Ottawa,

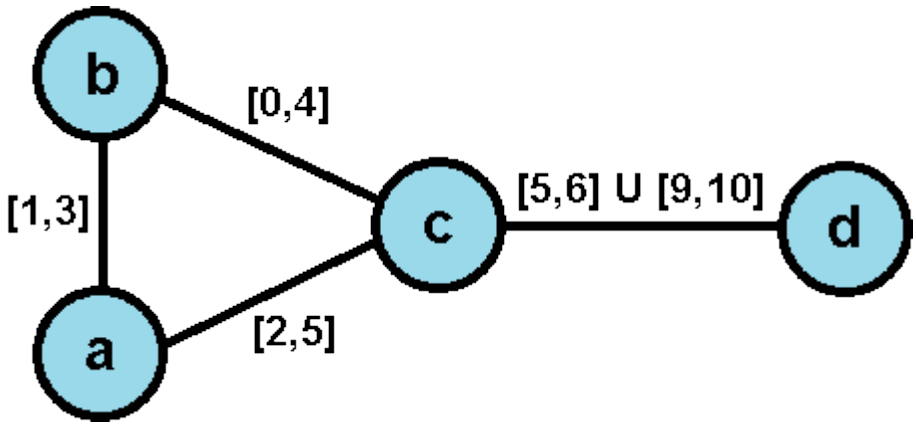


Figura 2.4: TVG com intervalos de tempo sobre as arestas

Montreal, Lisboa). Neste exemplo assume-se que as arestas são direcionadas, e possivelmente múltiplos rótulos (Repare nos múltiplos trajetos de Ottawa para Montreal).

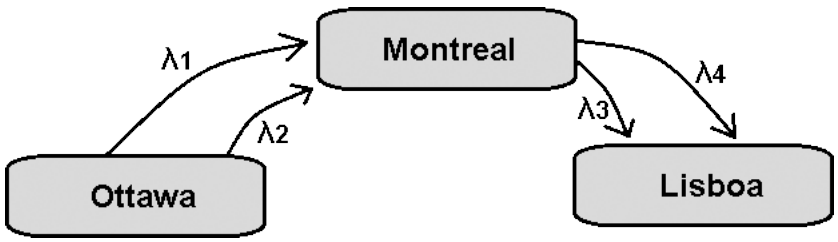


Figura 2.5: Rede de transporte

Neste exemplo os rótulos λ_1 até λ_4 representam **meios de transporte** utilizados para percorrer o trajeto de uma localidade a outra, por exemplo, "ônibus", "carro", "avião", "barco", respectivamente. Exceto por alguma viagem feita de carro de algum local para outro, digamos de Ottawa até Montreal (Ou seja, o rótulo da aresta neste caso é "carro"), e que pode ser iniciada a qualquer instante, as arestas típicas neste cenário estão disponíveis **pontualmente**, isto é, a função de presença ρ para estas arestas retorna 1 apenas em datas particulares quando a viagem pode ser iniciada (Por exemplo, um vôo de um local para outro possui horário fixo de partida).

A função de latência ζ pode variar de uma aresta para outra, ou seja, o tempo para atravessar cada aresta pode diferir entre arestas. Além disso, para uma mesma aresta, a função de latência ζ também pode retornar valores diferentes, por exemplo, trânsito variável na estrada numa viagem de carro de Ottawa para Montreal, dependendo do tempo de partida.

O TVG na figura 2.6 representa o histórico de **conectividade** entre um conjunto de nós em movimento, onde as possibilidades de comunicação aparecem como uma função de suas distâncias respectivas. Os dois rótulos λ_1 até λ_2 podem significar **meios de comunicação**, tais como wi-fi e Satélite, através dos quais há conexão entre os nós, portanto, tendo várias propriedades em termos de alcance, largura de banda, latência, ou consumo de energia. Lembrando que essas

várias propriedades representam a ideia de elementos multivalorados em um rótulo.

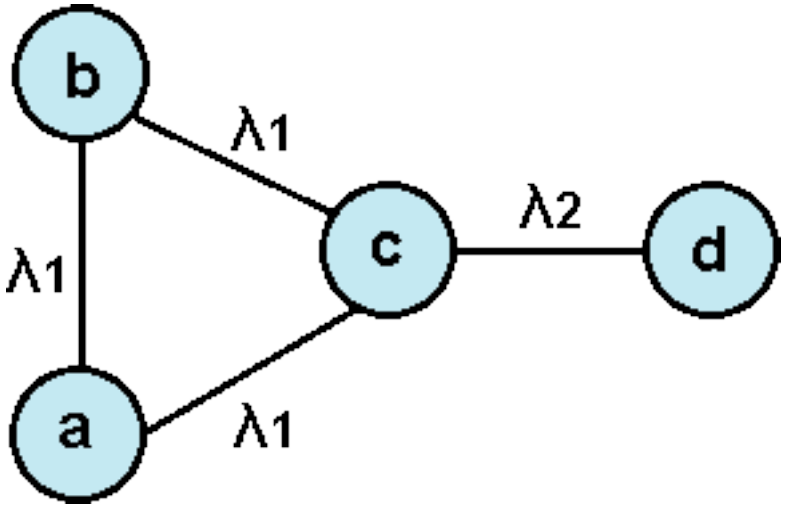


Figura 2.6: TVG representando uma rede de comunicação

Na figura 2.6, as arestas não são direcionadas e não há mais de uma aresta entre nós. Assim sendo, a aresta entre dois nós significa que qualquer um deles, ou ambos, podem enviar uma mensagem para o outro.

Uma função de presença para este tipo de aresta devolve 1 para alguns intervalos de tempo, uma vez que os nós estão geralmente no alcance um do outro durante um período não pontual de tempo, o que significa que as arestas ficam disponíveis nestes determinados intervalos de tempo.

A entrega efetiva de uma mensagem enviada em um tempo t em uma aresta e poderia estar sujeita a restrições adicionais relativas à função de latência, tais como a condição de que ρ(e) retorna 1 para o intervalo de tempo completo $[t, t+\zeta(e,t)]$. Isso quer dizer que a aresta e tem de estar disponível durante a travessia completa da mesma, que se iniciou a partir de uma data t, por parte de uma mensagem sendo enviada, ou em outras palavras, o valor retornado pela função ρ quando aplicado à aresta e retorna 1 durante todo o intervalo $[t, t+\zeta(e,t)]$.

2.2.2 O grafo subjacente G

Seja um TVG $\mathscr{G} = (V, E, \tau, \rho, \zeta)$, o grafo $G = (V, E)$ é chamado de **grafo subjacente de** $\mathscr{G}$, que indica apenas pares de entidades de nós com alguma relação em instante de tempo τ. Este grafo subjacente $\mathscr{G}$ é a representação visual do TVG.

O grafo subjacente G pode ser considerado como a união de todos grafos estáticos G_i de $\mathscr{G}$, que funcionam como *snapshots* do TVG $\mathscr{G}$, ao longo do tempo de vida τ do TVG. Para compreender, imagine o TVG como fosse uma animação de vídeo, ou seja uma sequencia de

quadros, onde conforme o tempo passa, a cada instante de tempo os quadros vão mudando. Neste caso, cada quadro é uma *snapshot* e representaria o estado do grafo em um dado instante de tempo.

Na prática, G resume em um único grafo todos os estados do TVG $\mathscr{G}$ ao longo de seu tempo de vida τ, mostrando todos os intervalos de tempo quando cada aresta esteve disponível. Ou, no contexto de redes, os intervalos de tempo em que pares de nós mantiveram-se conectados ou pareados.

Visualmente, o grafo subjacente G representa um grafo convencional completamente conectado. Porém, temporalmente, o seu TVG $\mathscr{G}$ não necessariamente se mantém sempre conectado ao longo do tempo, jstamente pela ideia de que este mesmo TVG, vai assumindo diversos estados ao longo do tempo, em outras palavras, arestas aparecem e desaparecem. Ou, no contexto de redes, aconteceram conexões e desconexões entre pares de nós ao longo do tempo.

Em nosso trabalho, representamos os TVGs segundo sua forma de grafo subjacente, e o tempo utilizado em novos grafos, seja para o tempo de vida, como para os intervalos de tempos de conectividade, é o tempo contínuo, ou seja, pertencente ao domínio dos números reais. Para gerar TVGs para nosso trabalho, desenvolvemos uma aplicação que converte (por meio de *parsing* de arquivos de texto) relatórios gerados a partir de simulação feita no ONE, sendo que estes relatórios contem todos os eventos decorridos da simulação, dentre eles os eventos de conectividade ocorridos entre pares de nós. Assim, esses relatórios são convertidos para formato compatível com a linguagem do *software* de geração e visualização de grafos ***GraphViz***[1], gerando, assim, um TVG descrito em linguagem reconhecida pelo *software Graphviz*. Posteriormente, para utilizar esse TVG aprendido da mobilidade no simulador ONE, foi desenvolvido em linguagem Java, um segundo trabalho de *parsing* do TVG já processado em GraphViz, de volta para uma representação em estruturas de dados em Java adequadas, de modo a fazer o seu uso, em roteamento.

Segue um exemplo um grafo subjacente com 5 nós na figura 2.7 usando o *Graphviz*. No exemplo temos os nós $n1$, $n2$, $n3$, $n4$ e $n5$, que em algum momento conectaram-se entre si por mobilidade, sendo os intervalos de tempos de conectividade representados pelos rótulos nas arestas. Além disso, neste grafo estamos considerando que o tempo de vida do mesmo, bem com o tempo nos intervalos, é um tempo discreto. O par de nós $n1 - n2$ apresenta um evento de reconexão entre seus nós, pois a princípio $n1$ e $n2$ mantiveram-se conectados durante 4 segundos (do segundo 2 ao segundo 6), desconectando-se a partir do sétimo segundo, reconectando-se a partir do segundo 10, permanecendo assim até o segundo 20. O mesmo raciocínio pode ser

[1] http://www.graphviz.org/

aplicado para os demais nós e arestas.

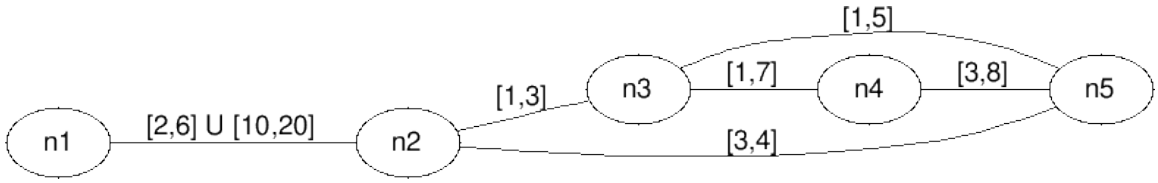


Figura 2.7: Exemplo de grafo subjacente

2.2.3 Jornadas

Segundo (CASTEIGTS, 2011; GOLDMAN; FLORIANO; FERREIRA, 2012), uma **jornada** é um **caminho ao longo do tempo** em um TVG, partindo de um nó origem e até um nó destino deste mesmo TVG. O conceito de jornadas é interessante ao nosso trabalho pois no mesmo tratamos do roteamento em uma rede DTN modelada como um grafo TVG, tendo a preocupação com o caminho que será feito, ao longo do tempo, pela mensagem, enviada de um nó origem a um nó destino.

Uma sequência de pares $\mathscr{J} = \{(e_1,t_1),(e_2,t_2),...,(e_k,t_k)\}$, tal que $\{e_1,e_2,...,e_k\}$ é um passeio em $\mathscr{G}$, e é considerada como uma **jornada em** $\mathscr{G}$ se e somente se:

- $\rho(e_i,t_i) = 1$: A aresta e_i deve estar disponível durante o instante t_i.
- $t_{i+1} \geq t_i + \zeta(e_i,t_i)$ para todo $i < k$: Um instante t_{i+1} deve começar no momento que se terminou a travessia da aresta e_i, começando a travessia da mesma no tempo t_i, ou seja, é necessário atravessar uma aresta e_i por completo antes de poder iniciar, no tempo t_{i+1}, a travessia da próxima aresta e_{i+1}.

A aresta e_1 é a primeira aresta a ser atravessada em uma jornada, sendo t_1 o tempo que se inicia a travessia da mesma.

Restrições adicionais podem ser necessárias dependendo do domínio da aplicação, por exemplo, redes de comunicação, onde as arestas devem se manter presentes até que a mensagem seja entregue, ou seja, $\rho_{[t_i,t_{i+1}+\zeta(e_i,t_i)]}(e_i) = 1$.

A seguir, são apresentados dois novos conceitos referentes à jornadas:

- **Partida**($\mathscr{J}$) ou **departure**($\mathscr{J}$): Refere-se à data de início, t_1, da jornada $\mathscr{J}$.
- **Chegada**($\mathscr{J}$) ou ***arrival***($\mathscr{J}$): Refere-se à data no fim da jornada $\mathscr{J}$, $t_k + \zeta(e_k,t_k)$, ou seja, o tempo t_k quando se inicia a travessia da última aresta da jornada, somado ao tempo $\zeta(e_k,t_k)$ gasto para atravessá-la.

Para entender melhor os conceitos de *departure* e *arrival*, observe o exemplo na figura 2.8, onde há uma jornada qualquer que corresponde ao envio de uma mensagem do vértice a até o vértice k.

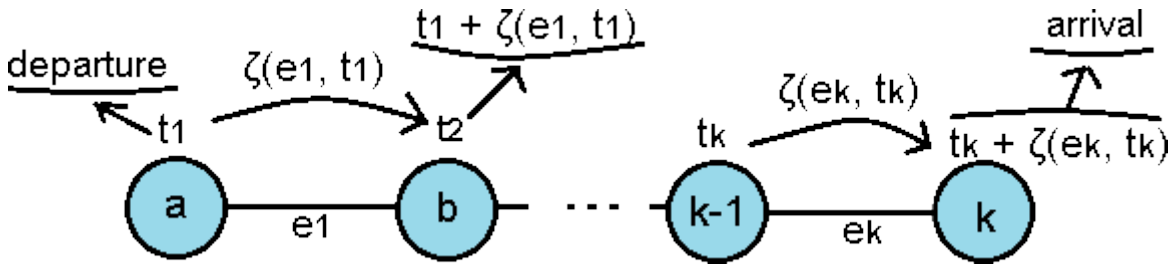


Figura 2.8: Exemplo de jornada

Observe o instante t_1, que é quando se inicia jornada e, por consequência, a travessia da aresta e_1, levando a mensagem de a para b. Para se chegar de a e b, há um tempo gasto para atravessar e_1, representado pela função de latência $\zeta(e_1,t_1)$. A mensagem enviada de a chega em b no tempo t_2, que corresponde a $t_1+\zeta(e_1,t_1)$.

Quando a jornada termina, com a mensagem entregue ao vértice destino k, o tempo na chegada é $t_k+\zeta(e_k,t_k)$, ou seja, o tempo t_k no nó $k-1$ imediatamente anterior ao nó k, que é quando se inicia a travessia da aresta e_k, somado ao tempo $\zeta(e_k,t_k)$ para atravessá-la e_k que os conecta.

Jornadas, por serem consideradas **caminhos ao longo do tempo**, possuem **comprimento topológico** e **comprimento temporal**:

- **Comprimento topológico**($\mathscr{J}$) ou ***topological lenght***($\mathscr{J}$): Número $|\mathscr{J}| = k$ de pares (e_i,t_i) presentes na jornada $\mathscr{J}$.
  - Exemplo: Número de saltos numa rede
- **Comprimento temporal**($\mathscr{J}$) ou ***temporal lenght***($\mathscr{J}$): Duração fim-a-fim da jornada $\mathscr{J}$, isto é, $arrival(\mathscr{J}) - departure(\mathscr{J})$.
  - Uma analogia a se pensar é o Δ_t da Física, com *tempo final* $-$ *tempo inicial*.

Além disso, denotam-se estes conceitos:

- $\mathscr{J}^*_{\mathscr{G}}$: Conjunto de todas possíveis jornadas em um TVG $\mathscr{G}$.
- $\mathscr{J}_{(u,v)} \subseteq \mathscr{J}^*$: Conjunto de todas as possíveis jornadas em um TVG $\mathscr{G}$ que começam em um nó u e terminam em um nó v.

  - Se existe uma jornada de um nó u até um nó v, isto é, se $\mathscr{J}^* \neq 0$, pode-se dizer que u pode **alcançar** v, para o qual usamos uma notação $u \rightsquigarrow v$.
    * A existência de uma jornada de u a v não é simétrica, isto é, $u \rightsquigarrow v \nLeftrightarrow v \rightsquigarrow u$, o que se mantem válido independente das arestas serem direcionadas ou não.
      · Para entender, imagine que há uma jornada do nó u até v, ou seja, uma mensagem enviada por u eventualmente chegará até v. Mas ao término do envio da mensagem, ou até mesmo durante o caminho percorrido para enviá-la, pode ocorrer de alguma aresta, que direta ou indiretamente conecta u até v, ficar indisponível, impossibilitando uma jornada de volta, de v até u.

- **Horizonte de** u: Denotado como $\{v \in V : u \rightsquigarrow v\}$, refere-se a todos os vértices v de V que u pode alcançar ao longo do tempo, durante alguma jornada.

Como comentado anteriormente nesta subseção, o comprimento de uma jornada pode ser medido em termos de saltos ou tempo, o que traz dois conceitos distintos de distância em um TVG $\mathscr{G}$:

- **Distância topológica** ou ***Topological distance***: A **distância topológica** de um nó u até um nó v, em um tempo t, denotada como $d_{u,t}(v)$, e definida como $Min\{|\mathscr{J}| : \mathscr{J} \in \mathscr{J}^*_{(u,v)}, departure(\mathscr{J}) \geq t\}$.

  Para uma data t, uma jornada com *departure* t', com $t' \geq t$, e *topological lenght* $= d_{u,v}(v)$ é qualificada como ***shortest*** ou **mais curta**.

  - A **distância topológica** considera todas as possíveis jornadas de u até v, que começam num tempo t ou posterior, e procura entre elas, as jornadas que contem o menor número de saltos. O *topological lenght* desta jornada é dado por $d_{u,t}(v)$.
  - Assim, é possível existir várias jornadas cujo *topological lenght* é igual a $d_{u,t}(v)$, e que por consequência são classificadas como mais curtas ou *shortest*.

  Para exemplificar este conceito considere um TVG $\mathscr{G}$, com o grafo subjacente G que o representa na figura 2.9, e com as seguintes características:

  - G é feito da união de três grafos estáticos, G_1, G_2 e G_3, na figura 2.9, cada um, respectivamente, para os tempos $t = 1$, $t = 2$ e $t = 3$.
  - O domínio temporal considerado é discreto, utilizando números naturais $\mathbb{N}$.
  - O sistema representado pelo TVG permite *self-loops*, que são representados por arestas que saem e chegam no mesmo vértice, mas aqui não são considerados saltos.

– Possui tempo de vida $\tau = 3$, abrangendo o intervalo $t = [1,3]$.

– A função de latência ζ de cada aresta do TVG sempre retorna 1.

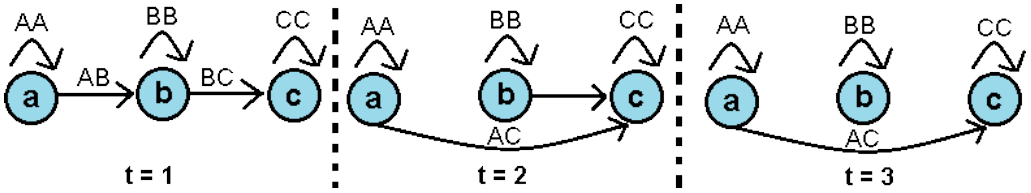


Figura 2.9: TVG exemplificando distância topológica

Suponha uma mensagem seja enviada do nó a para c. De todas as jornadas possíveis que a mensagem pode fazer de a para c, iniciando em algum tempo t ou maior, ou seja, $\mathscr{J}^*_{a,t}(c)$, só interessa a de menor *topological lenght*, ou seja, $d_{a,t}(c)$.

Como explicado anteriormente, uma jornada $\mathscr{J} = \{(e_1,t_1),(e_2,t_2),...,(e_k,t_k)\}$, sendo e_1 a primeira aresta a ser atravessada na jornada, e t_1 o tempo que se inicia a sua travessia.

A tabela 2.1 mostra as possíveis jornadas de a a c, começando num tempo $t = 1$ ou $t = 2$.

Tempo de partida t	**Jornada** $\mathscr{J}_{a,t}(c)$	*Topological lenght* ($\mathscr{J}_{a,t}(c)$)
1	$\mathscr{J}_{a,1}(c) = \{(AB,1),(BC,2)\}$	2
1	$\mathscr{J}_{a,1}(c) = \{(AA,1),(AC,2)\}$	1
2	$\mathscr{J}_{a,2}(c) = \{(AC,2)\}$	1

Tabela 2.1: Possíveis jornadas de a a c iniciando num tempo $\geq t$

Se a enviar uma mensagem para c começando em:

– $t = 1$: A mensagem parte de a para b, atravessando a aresta AB, gastando-se 1 segundo. Quando a mensagem está em b, $t = 2$, a mensagem é então enviada para c, atravessando a aresta BC, gastando-se mais 1 segundo. A mensagem chegará em c quando $t = 3$. Assim foram feitos 2 saltos ao longo do tempo neste jornada.

  Por outro lado, a mensagem pode permanecer em a de $t = 1$ até $t = 2$, sendo que a partir de $t = 2$ inicia-se a travessia da aresta AC, fazendo com que a mensagem chegue a c quando $t = 3$, gastando-se apenas 1 segundo.

– $t = 2$: A mensagem parte de a para c, atravessando a aresta AC, chegando em c em $t = 3$, gastando-se apenas 1 segundo.

Aqui a distância topológica $d_{a,t}(c) = 1$, e duas jornadas, uma começando em $t = 1$ e outra em $t = 2$, possuem *topological lenght* $= 1$, e portanto são *shortest*.

Note que não foram consideradas jornadas de a até c iniciando em $t = 3$, pois ainda que a aresta AC esteja disponível para travessia no instante $t = 3$, a latência de 1 segundo para

atravessá-la extrapola o tempo de vida $\tau = 3$ do TVG, pois a mensagem só seria entregue a c num instante $t = 4$, o qual inexiste no tempo de vida τ considerado neste exemplo.

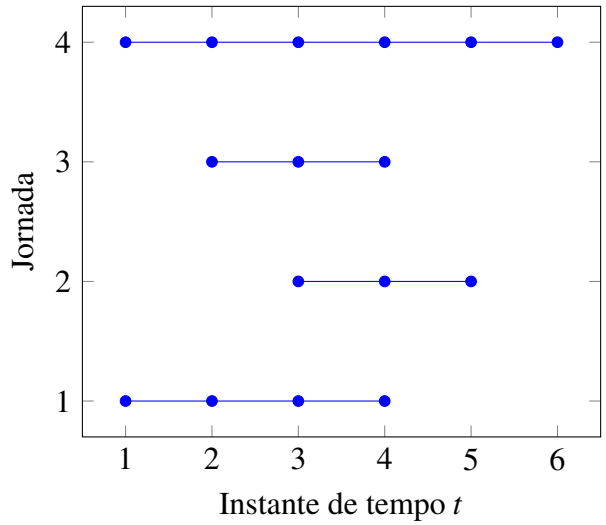
- **Distância temporal** ou ***Temporal distance***: A **distância temporal** de um nó u até um nó v, em um tempo t, denotada como $\hat{d}_{u,t}(v)$, e definida como $Min\{arrival(\mathscr{J}) : \mathscr{J} \in \mathscr{J}^{*}_{(u,v)}, departure(\mathscr{J}) \geq t\} - t$.

  Para uma data t, uma jornada com *departure* t', com $t' \geq t$, e $arrival = t + \hat{d}_{u,t}(v)$ é qualificada como ***foremost*** ou **adiantada**.

  Finalmente, para qualquer data t, uma jornada com $departure \geq t$ e *temporal lenght* $= Min\{\hat{d}_{u,t'}(v) : t' \in \tau \cap [t, +\infty)]\}$ é qualificada como ***fastest*** ou **mais rápida**.

  - A **distância temporal** considera todas as posséveis jornadas de u até v, que começam num tempo t ou posterior, e procura entre elas, as jornadas de menor *arrival*, ou seja, que terminam mais cedo. O *temporal lenght* desta jornada é dado por $\hat{d}_{u,t}(v)$.
  - As jornadas qualificadas como ***foremost*** ou **adiantadas** são as que, começando em um t ou posterior, terminam **mais cedo**, ou seja, o seu tempo de *arrival* é o menor, independente da duração da jornada.
  - As jornadas qualificadas como ***fastest*** ou **adiantadas**, são as que, começando em um tempo t ou posterior, tem **duração menor**, ou seja, o seu *temporal lenght* é o menor, independente de quão cedo ou tarde foi iniciada ou terminada a jornada.

  Para entender estes conceitos observe o exemplo no gráfico a seguir, que considera um TVG com tempo de vida $\tau = 7$, indo de 0 a 7, e algumas jornadas de u a v iniciando em algum tempo t ou posterior.



Aqui há 4 jornadas que levam de u a v, as jornadas 1, 2, 3 e 4, com algumas informações sobre elas exibidas na tabela 2.2.

Jornada	***Tempo de departure***	***Tempo de arrival***	***Temporal lenght***
1	1	4	3
2	3	5	2
3	2	4	2
4	1	6	5

Tabela 2.2: Tabela com dados das jornadas 1, 2, 3 e 4

Com os dados da tabela 2.2 é possível fazer considerações sobre os valores de *temporal distance* e quais são as jornadas *foremost* e *fastest*, que podem diferir dependendo de quando se inicia a jornada de u a v. Para este exemplo analisamos, no intuito de simplificar, apenas os casos de $\hat{d}_{u,0}(v)$ e $\hat{d}_{u,2}(v)$:

- Para $\hat{d}_{u,0}(v)$:
  - Considera as jornadas que iniciaram em um tempo $t \geq 0$, neste caso, 1, 2, 3 e 4.
  - Entre estas jornadas, o menor tempo de *arrival* vale 4, portanto o menor *temporal lenght* $= arrival - departure = 4 - 0 = 4$.
    - Assim a *temporal distance* $\hat{d}_{u,0}(v) = 4$.
  - As jornadas *foremost* terminam mais **cedo**, possuindo tempo de $arrival = t + \hat{d}_{u,0}(v) = 0 + 4 = 4$.
    - Assim as jornadas *foremost* são as jornadas 1 e 3.
  - As jornadas *fastest* duram **menos**, possuindo $departure \geq t$ e o menor *temporal lenght*.
    - Assim a jornadas *fastest* são as jornadas 2 e 3, pois, para a jornada 2, $arrival - departure = 5 - 3 = 2$, e para a jornada 3, $arrival - departure = 4 - 2 = 2$.
- Para $\hat{d}_{u,2}(v)$:
  - Considera as jornadas que iniciaram em um tempo $t \geq 2$, neste caso, 2 e 3.
  - Entre estas jornadas, o menor tempo de *arrival* vale 4, portanto o menor *temporal lenght* $= arrival - departure = 4 - 2 = 2$.
    - Assim a *temporal distance* $\hat{d}_{u,2}(v) = 2$.
  - As jornadas *foremost* terminam mais **cedo**, possuindo tempo de $arrival = t + \hat{d}_{u,2}(v) = 2 + 2 = 4$.
    - Assim a jornada *foremost* é a jornada 3.
  - As jornadas *fastest* duram **menos**, possuindo $departure \geq t$ e o menor *temporal lenght*.

· Assim a jornadas *fastest* são as jornadas 2 e 3, pois, para a jornada 2, $arrival - departure = 5 - 3 = 2$, e para a jornada 3, $arrival - departure = 4 - 2 = 2$.

Além disso, em um TVG, segundo (GOLDMAN; FLORIANO; FERREIRA, 2012), diferentes jornadas ótimas podem existir, a depender da métrica otimizada na jornada. Existem três tipos de jornadas ótimas:

- **Jornada adiantada (*Foremost journey*):** Jornada onde busca-se otimizar o tempo de chegada, de modo que a mesma tenha o menor tempo de chegada possível. Nosso algoritmo TVFP, por buscar o caminho em um TVG que permita a mensagem chegar ao destino no menor tempo possível, encontra a jornada mais adiantada entre origem e destino.

- **Jornada mais curta (*Shortest journey*):** Jornada onde busca-se otimizar o total de *hops* gastos na jornada, de modo a minimizar o número de *hops* no caminho percorrido na jornada.

- **Jornada mais rápida (*Fastest journey*):** Jornada onde busca-se otimizar o tempo de viagem da jornada, ou seja, a diferença entre o tempo de chegada e o tempo de partida, resultando em uma viagem de duração o mais curta possível. No trabalho relacionado de (DING; YU; QIN, 2008), foi desenvolvido um algoritmo que encontra o caminho em um TVG que propicie o menor tempo de viagem possível para uma mensagem partindo de uma origem até um destino.

Capítulo 3

TRABALHOS RELACIONADOS

3.1 Algoritmo *Two-Step-LTT* para encontrar as jornadas mais rápidas em TVGs

Na literatura existem algumas soluções envolvendo algoritmos de roteamento em grafos dinâmicos, por exemplo o algoritmo ***Two-Step-LTT*** do trabalho de (DING; YU; QIN, 2008). Esse algoritmo, conforme apresentaremos, é baseado em Dijkstra. Ele pode ser considerado relevante pois tem relação com nosso trabalho e é referência no assunto.

O algoritmo é chamado ***Two-Step Least Time Travel*** **(Two-Step-LTT)**, e busca encontrar o **caminho ótimo** em um grafo dinâmico TVG, de um nó origem até um nó destino, que proporcione o menor **tempo de viagem** possível até o destino, portanto o caminho ótimo é aquele que, quando percorrido pela mensagem, fará com que a mesma tenha o menor tempo de viagem possível até o destino. Em outras palavras, este algoritmo calcula o caminho em um TVG, que proporcione a **jornada mais rápida**, ou ***fastest journey***, entre um nó origem e um nó destino deste TVG. Esta proposta difere da nossa, pois, com o TVFP (a ser apresentado na seção seguinte) e um TVG, calculamos o caminho no qual garantimos que a mensagem será entregue ao destino, e no **menor tempo** possível no *snapshot*, mas não necessariamente será o caminho mais curto em número de *hops*, ou com o menor tempo possível total de viagem. Ambas as propostas, tanto a do trabalho relacionado como a nossa, se assemelham no fato que requerem conhecimento do TVG a priori para funcionar, uma vez que por meio do TVG que serão calculados os caminhos entre origem e destino. Em resumo, no trabalho relacionado, busca-se encontrar a **jornada mais rápida**, ou ***fastest journey***, entre um nó origem e um nó destino, enquanto que no nosso trabalho, encontramos a **jornada mais adiantada**, ou ***foremost journey***, entre um nó origem e um nó destino, porém utilizando uma única instância da mensagem.

O *Two-Step-LTT* recebe este nome pois o processo de cálculo do menor tempo de viagem é feito em dois passos, ou fases, como sugere o seu nome:

- ***time-refinement:*** Primeira fase do processo. O algoritmo recebe um TVG como entrada e calcula para todo nó pertencente ao grafo, o seu tempo de chegada mais cedo até o nó destino, de modo que possa ser identificado, dentre os possíveis tempos de partida, o tempo de partida **ótimo** a partir da origem, sende este o tempo para se iniciar a transferência de uma mensagem que permite o **menor** tempo de viagem possível para a mensagem.
- ***path-selection:*** Segundo passo. Com o tempo de partida ótimo obtido na primeira fase do processo, descrita anteriormente, escolhe-se o caminho, dentre os possíveis caminhos calculados pelo algoritmo, que permita o tempo de viagem ótimo, ou seja, mínimo, de um nó origem até um nó destino, sendo esta viagem iniciada pelo tempo de partida ótimo calculado anteriormente.

Nosso algoritmo TVFP também trabalha recebendo um TVG como entrada, com o diferencial sendo que o caminho no *snapshot* encontrado permite que a mensagem seja entregue ao destino, e no menor tempo possível, independente do tempo de viagem ou número de *hops*.

Na sequência descrevemos o funcionamento do *Two-Step-LTT*. O **Two-Step-LTT** visa encontrar o tempo de viagem (*travel time*) mínimo entre um vértice origem v_s, até um vértice destino v_e, começando a viagem em um tempo de partida (*starting time*) t, sendo que este t é escolhido de um intervalo de tempos de partida (*starting time interval*) $T = [t_s, t_e] \subseteq \tau$, onde τ é um domínio temporal. O caminho de v_s até v_e encontrado, que minimiza o *travel time*, é o caminho otimizado p^*, e iniciou a jornada num tempo de partida ótimo t^*.

O algoritmo é dividido em três partes: o algoritmo de *timeRefinement*, o algoritmo *path-Selection* e um algoritmo, chamado de *Two-Step-LTT* que gerencia os demais algoritmos para cumprir o objetivo.

Alguns termos importantes utilizados nos algoritmos do *Two-Step-LTT* são:

- $G_T(V,E,W)$: Um grafo dependente no tempo, com um conjunto de vértices v, um conjunto de arestas E, e W um conjunto de funções que retornam valores positivos.
- $g_e(t) - t$: É o tempo total de viagem (*travel time*) iniciando num tempo t, e terminando no *arrival time* $g_e(t)$.
- $g_i(t)$: Trata-se da função que calcula o menor tempo de chegada (*earliest arrival time*

function) no vértice v_i, partindo do vértice de origem v_s, com tempo de partida da viagem t.

- $g_p(t)$: *Arrival time function* ao longo do caminho p, se começada a viagem num tempo t.
- τ_i: É o tempo ideal para começar a viagem, sendo relativo ao vértice i.
- t^*: É o tempo de partida (*starting time*) ótimo.
- p^*: É o caminho ótimo entre um vértice de origem v_s e um vértice destino v_e.
- $\omega_{i,j}(t)$: Função *edge-delay* (*edge-delay function*), que especifica quanto tempo se leva para viajar do vértice v_i até o vértice v_j, com a viagem começando no tempo t, ou em outras palavras, com o tempo de partida de v_i sendo t. No lugar de t pode-se usar $g_i(t)$, que é o tempo de viagem até i.

O algoritmo *Two-Step-LTT* chama os outros dois algoritmos *timeRefinement* e *pathSelection*. O primeiro calculará cada *earliest arrival time function* dos vértices do grafo. A partir disto, o algoritmo calcula o *starting time* ótimo (t^*). Em seguida o algoritmo envia este valor, bem como os outros valores necessários, para o algoritmo *pathSelection*, que achará o caminho ótimo p^*. Finalmente, este caminho p^* e o *starting time* ótimo t^* são retornados.

Em seguida são descritos em alguns passos sobre o que ocorre em cada algoritmo. Mais detalhes de cada um dos algoritmos encontram-se em (DING; YU; QIN, 2008).

3.1.1 Algoritmo *Two-Step-LTT*

O algoritmo *Two-Step-LTT* pode ser descrito da seguinte maneira:

1. O algoritmo recebe como entrada os seguintes argumentos:
   - O grafo dependente do tempo G_T
   - O vértice de origem v_s
   - O vértice de destino v_e
   - O intervalo de possíveis tempos de partida $T = [t_s, t_e]$
2. O algoritmo *timeRefinement*, da subseção 3.1.2, é chamado, recebendo como entrada o grafo G_T, v_s, V_e e T, para que sejam computadas as *earliest arrival time functions* de cada vértice v_i do grafo.

3. Se o valor da *earliest arrival time function* do vértice v_e for infinito não foi possível calcular um caminho de v_s até v_e. Neste caso, o algoritmo retorna vazio, caso contrário o algoritmo prossegue para achar o caminho.

4. Aqui é determinado o *starting time* ótimo. Para tanto é feito o cálculo do *travel time* $(g_e(t) - t)$ mínimo para o *starting time* t no intervalo T, de modo a calcular qual é o t que minimiza o *travel time* (t^*).

5. Chama-se o algoritmo *pathSelection*, que recebe como parâmetros o grafo G_T, as *earliest arrival time functions*, v_s, v_e e t^*, que foram obtidos no passo anterior, para que possa ser calculado o caminho ótimo p^*.

6. O algoritmo retorna o caminho ótimo p^* junto com o *starting time* ótimo t^*.

3.1.2 Algoritmo *timeRefinement*

O algoritmo de *timeRefinement* pode ser descrito da seguinte maneira:

1. O algoritmo recebe como entrada os seguintes argumentos:

   - O grafo dependente do tempo G_T
   - O vértice de origem v_s
   - O vértice de destino v_e
   - O intervalo de possíveis tempos de partida $T = [t_s, t_e]$

2. Primeiramente, a *arrival-time function* $g_s(t)$ do nó origem, é iniciada com t, para $t \in T$, de possíveis tempos de partida. Ou seja, a função de *arrival-time* para o nó v_s é definida como: $g_s(t) = t$, para $t \in T$. Além disso τ_s é inicializado com t_s, que é o menor instante de tempo em T.

3. Como está sendo analisado o vértice v_i, que inicialmente corresponde ao v_s, verifica-se todos os vértices v_f vizinhos que incidem em v_i, ou seja, para cada vértice v_i, que não o origem, inicializa-se a sua respectiva *earliest time function* $g_i(t)$ com valor infinito $(g_i(t) = \infty)$, para $t \in T$. Cada valor τ_i é inicializado também com o valor t_s.

4. Cria-se uma fila de prioridade Q, contendo pares $(\tau_i, g_i(t))$ para cada vértice v_i, sendo que os elementos da fila são ordenados por ordem crescente de valores $g_i(\tau_i)$. Ou seja, trata-se de uma fila de prioridade, na forma de uma lista de nós candidatos a serem os próximos

hops, ordenadas pelo tempo de disponibiliadade, que garante que a primeira posição é sempre o par $(\tau_i, g_i(t))$ que apresentar o menor de valor de $g_i(\tau_i)$.

5. Se a fila tiver apenas um elemento o algoritmo retorna $g_i(v_i)$ porque este é obrigatóriamente o vértice final v_e.

6. Retira o elemento da frente da fila e o coloca no par $(\tau_i, g_i(t))$, ou seja, $(\tau_i, g_i(t)) \leftarrow dequeue(Q)$. Após isso, a fila Q tem um elemento a menos.

7. Copia o elemento atual na frente da fila e o coloca no par $(\tau_k, g_k(t))$, ou seja, $(\tau_i, g_i(t)) \leftarrow head(Q)$. Note que não foram removidos elementos da fila.

8. Agora para cada aresta que chega em v_i, ou seja, cada par $(v_f, v_i) \in E$, verifica-se qual delas possui o menor valor de *edge-delay function*, para $g_k(\tau_k)$, dado por, $\omega_{f,i}(g_l(\tau_k))$, e o coloca na variável Δ.

9. Atualiza-se τ_i', que será atribuída a τ_i no passo 11. O valor que τ_i' recebe é o maior t no qual o *arrival time* de i para este t ainda é menor ou igual que o *arrival time* de k em $\tau_k + \Delta$ $(g_i(t) \leq g_k(\tau_k))$.

10. Para cada aresta que sai de v_i, ou seja, $(v_i, v_f) \in E$, e por consequência, cada vértice v_j alcançado por v_i, atualiza-se o valor de $g_j'(t)$, que recebe $g_i(t) + \omega_{i,j}(g_i(t)$ para $t \in [\tau_i, \tau_i']$. O valor é então passado para $g_j(t)$ se for menor que o valor antigo. Este passo é similar ao Dijkstra porque ele atualiza o acumulador de v_j com o de v_i mais o peso da aresta (dado por $\omega_{i,j}$) caso seja menor do que o já presente. Com a atualização do valor de $g_j(t)$ torna-se necessário atualizar a fila com o novo valor, recalculando as prioridades.

11. Atualiza-se o τ_i com o valor previamente calculado de τ_i'.

12. A partir deste passo dá-se a checagem do final do algoritmo ou do que fazer com o vértice v_i. Se o τ_i for maior ou igual t_e e se o vértice v_i sendo analisando for o vértice destino v_e, então a *earliest arrival function* $g_i(t)$ para o vértice destino está completamente refinada e é retornada. Se o vértice não v_i não for igual a v_e, no entanto, o *earliest arrival function* vértice v_i já está refinado e o algoritmo deve prosseguir para para outro vértice, voltando para o passo 5. Caso τ_i não seja maior ou igual a t_e, significa que não foi possível refinar a *earliest arrival time function* $g_i(t)$ do vértice v_i sendo analisado ainda e, portanto, o par $(\tau_i, g_i(t))$ é recolocado na fila, e o algoritmo deve voltar ao passo 5 para continuar o processo de refinamento.

3.1.3 Algoritmo *pathSelection*

Este algoritmo faz a descoberta do caminho ótimo p^* através de *backtracking*, começando a descoberta do caminho pelo vértice destino v_e até encontrar o vértice origem v_s. O algoritmo de *pathSelection* pode ser descrito da seguinte maneira:

1. O algoritmo recebe como entrada os seguintes argumentos:
   - O grafo dependente do tempo G_T
   - O vértice de origem v_s
   - O vértice de destino v_e
   - O *starting time* ótimo t^*
2. Atribui-se o vértice v_e ao vértice auxiliar v_j, sendo que v_j sempre recebe o vértice a ser analisado. Inicialmente analisa-se o v_e.
3. A variável p^*, que representa o caminho ótimo, inicia vazio.
4. Se o vértice v_j atual for o vértice origem v_s, o algoritmo retorna o caminho p^*.
5. Procura-se qual vértice v_i, que permite chegar em v_j com o menor *earliest arrival time* $g_e(t^*)$, testando se o *earliest arrival time* em v_j é igual ao *earliest arrival time* de v_i acrescido do tempo de travessira entre v_i e v_j no instante do $g_i(t^*)$ de v_i.
6. A aresta (v_i, v_j), calculada no passo anterior, é adicionada ao início do caminho p^* e retorna para o passo 4, com v_j passando a apontar para o v_i.

3.2 Algoritmos de roteamento em DTNs com classificadores

Além de algoritmos de roteamento em DTN baseados em TVGs, ou seja, algoritmos que necessitam de uma fase de coleta de dados da mobilidade para extrair informação útil para determinar uma seleção do próximo nó para o roteamento, também podemos considerar pertinentes trabalhos relacionados como o trabalho de (PORTUGAL-POMA, 2013, 2014), onde são usados classificadores de *big data* implementados no simulador ONE, e que realizam uma simulação para coleta de dados para posterior decisão de roteamento. Inicialmente é feita uma simulação para coletar de dados, de modo a treinar os classificadores. Com os dados coletados, e consequentemente, os classificadores treinados, é possível agora fazer uma simulação utilizando algoritmos que façam uso destes classificadores e, portanto, agora utilizando os dados

coletados, que permitirá aos nós de uma rede na simulação decidirem quais nós vizinhos são os melhores candidatos para repasse de uma mensagem. Assim, a fim de reduzir *overheads* de geração de mensagem e entregar a mensagem ao destino mais rapidamente.

Nosso trabalho segue uma abordagem similar, na qual inicialmente procuramos obter dados para poder gerar um TVG, através de uma simulação no ONE, podendo-se comparar esta fase com a fase de coleta do trabalho relacionado (PORTUGAL-POMA, 2013, 2014). Posteriormente utilizamos estes mesmos dados (TVG) para futuras simulações, permitindo o cálculo do caminho que entrega a mensagem ao destino no menor tempo possível.

3.3 DTNTES: Uma ferramenta utilizada para extrair informação útil de *traces* de redes DTN

No trabalho de (GOLDMAN; FLORIANO; FERREIRA, 2012) foi desenvolvida uma ferramenta, na forma de uma interface web, chamada ***DTN Trace Evaluator System*** (**DTNTES**) que, por meio de um elaborado processo, extrai informações úteis de um dado *trace* de mobilidade passado como entrada para a aplicação, sendo que aceita *traces* em diversos formatos, incluindo o formato aceito pelo simulador ONE.

Para se obter a informação de um dado *trace*, a ferramenta DTNTES constrói um grafo TVG correspondente ao *trace* passado como entrada, tornando disponíveis ao usuário alguns serviços baseados nestas informações obtidasa, como por exemplo serviços de conectividade e serviços de jornadas, onde o primeiro possui informações acerca da conectividade entre nós do grafo, e o segundo informações tais como encontrar a jornada ótima entre um par de nós do grafo.

Nosso trabalho segue uma abordagem similar, no sentido de que é gerado um grafo TVG a partir de um *trace*, de forma indireta, pois não geramos o grafo TVG processando o *trace*, e sim processando um relatório gerado ao final de uma simulação que faz uso deste *trace*. Assim, tal qual o trabalho relacionado, extraímos informação útil do TVG, pois, utilizando nosso algoritmo TVFP, buscamos encontrar o caminho entre um par de nós origem e destino do TVG, que leve a mensagem ao destino no menor tempo possível, ou em outras palavras, buscamos encontar a jornada mais adiantada entre um par de nós origem e destino do TVG.

Capítulo 4

PROPOSTA DESTE TRABALHO

Nesse capítulo abordaremos a principal contribuição do trabalho, que é a criação de um algoritmo distribuído de roteamento em redes DTN, e que usa uma lógica simples, denominado ***Time-Varying Fastest Path*** **(TVFP)**, que encontra o caminho em um grafo TVG, entre um par de nós origem e destino, que permita com que a mensagem chegue ao destino no menor tempo possível, sem que sejam feitas quaisquer cópias adicionais da mensagem que não a instância original da mesma. Do ponto de vista do TVG, nosso algoritmo encontra o caminho com a **jornada mais adiantada**, ou ***foremost journey***, entre um par de nós origem e destino deste TVG. Além disso, discutimos brevemente sobre o funcionamento do simulador ONE, aonde implementamos nosso algoritmo e fizemos testes com o mesmo. A nossa solução proposta com o TVFP requer cenários de mobilidade determinísticos e uma mensagem de tamanho pequeno para o seu funcionamento.

É fundamental a compreensão dos detalhes discutidos acerca do funcionamento do nosso algoritmo e do simulador ONE, uma vez ambos fazem parte de um processo amplo de mineração de dados com o intuito de gerar um TVG, para então ser utilizado e aplicado posteriormente no roteamento feito pelo algoritmo TVFP.

O algoritmo faz parte de um processo amplo de mineração de dados e posterior aplicação no roteamento, onde a nossa metodologia envolvida, desde a mineração dos dados, até posterior simulação de roteamento com estes dados, pode ser dividida em 4 fases.

Iniciamos o processo de mineração de dados com o tratamento de *traces* de mobilidade real de ônibus oriundos do repositório do CRAWDAD, de modo que sejam convertidos para formato compatível com o utilizado no simulador ONE, como será descrito na seção 4.2.

Após isso, realizamos uma simulação no ONE utilizando o *trace* convertido, gerando um relatório dos eventos decorridos na simulação, ou seja, o relatório *eventlog*. Em seguida, por

meio de outra ferramenta *parser*, geramos um TVG a partir deste *eventlog*.

Por fim, refazemos a mesma simulação anterior, com o mesmo *trace*, mas agora aplicando o TVFP, que utiliza o TVG gerado como entrada. A figura 4.1 resume as fase do nosso processo completo de mineração de dados, até a utilização dos mesmoms pelo TVFP.

Detalhes técnicos, mais aprofundados a cerca do que é discutido em cada uma das fases, podem ser encontrados nos Anexos deste trabalho.

Na sequência, explicamos o funcionamento de nosso algoritmo TVFP, e discutimos sobre cada fase.

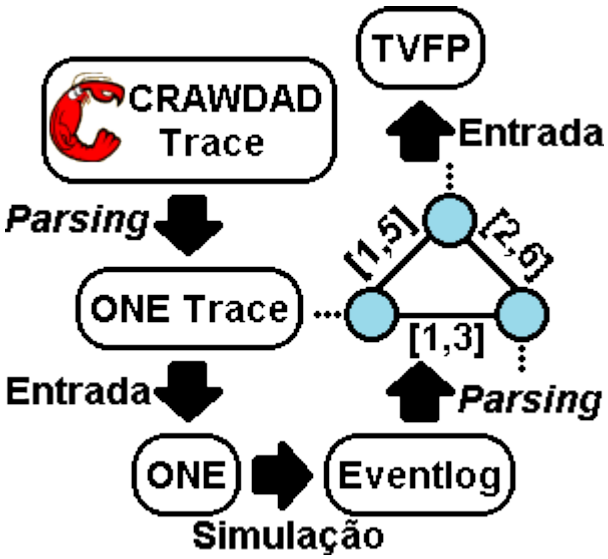


Figura 4.1: Metodologia do processo de mineração de dados a serem usados com o TVFP

4.1 Funcionamento do TVFP

Desenvolvemos um algoritmo distribuído de roteamento *single-copy* chamado ***Time-Varying Fastest Path*** (**TVFP**), que encontra o **melhor caminho** em um TVG para se enviar a mensagem de um nó origem a um nó destino, de modo que a mesma seja entregue no **menor tempo**, ou seja, o menor **tempo de chegada**, possível, não importando o tempo de viagem total ou número de *hops* percorridos pela mensagem. Este algoritmo é distribuído, pois cada nó na simulação executa uma instância do algoritmo consigo para decidir se devem ou não repassar a mensagem adiante, até que a mesma chegue ao destino. Do ponto de vista de um TVG, o nosso algoritmo TVFP encontra a **jornada mais adiantada**, ou ***foremost journey*** para se levar a mensagem de um nó origem do TVG a um nó destino do TVG, e utilizando apenas uma instância da mensagem. Além disso, implementamos este algoritmo para ser utilizado no simulador ONE, e demais algoritmos, como *Epidemic* e Spray-and-Wait, implementados no simulador tambem

são distribuídos. Por fim, com relação ao tempo de chegada mencionado é baseado nos tempos de conexão entre nós previstos pelo TVG, e o domínio temporal deste tempo é o domínio dos números reais, ou seja, um tempo contínuo. O melhor caminho encontrado pelo TVFP entre pares de nós é único, e da maneira que implementamos o algorito, só é retornado um melhor caminho entre pares de nós.

O contexto no qual aplicamos o TVFP trata-se de um cenário determinístico envolvendo uma frota de ônibus que, devido a seu itinerário, seguem um padrão de mobilidade fixo, previsível e determinístico, sendo que o TVG gerado de uma simulação de *traces* de ônibus no ONE contém os instantes de tempo exatos quando estes ônibus se conectaram, podendo prever quando os ônibus irão se conectar, viabilizando a sua utilização com o TVFP para calcular o melhor caminho. Assim, o cenário em que aplicaremos o TVFP é de uma rede DTN formada por ônibus, ou seja, os nós da rede consistem em ônibus, onde os nós possuem suas movimentações descritas por um *trace* externo de mobilidade de ônibus oriundo do repositório CRAWDAD.

Inicialmente, o TVFP calcula os melhores caminhos do nó origem até os demais nós alcançáveis por este e, posteriormente, dentre os caminhos calculados, extrai-se o melhor caminho, caso exista, entre este mesmo nó origem e um nó destino arbitrário. A obtenção do melhor caminho entre origem e destino é feita por meio de método auxiliar (a ser explicado depois), o qual recebe o nó destino como entrada.

Discutimos alguns detalhes que consideramos essenciais para a elaboração e implementação do TVFP para uso no simulador ONE:

- **TVFP como algoritmo *single-copy*:** Em nossa implementação o TVFP atua como um algoritmo *single-copy* para roteamento em redes DTN, ou seja, na rede em que ele estiver, só há uma única instância da mensagem em circulação, e a mesma possui um tamanho pequeno. Este detalhe influenciou as decisões sobre como estruturamos o melhor caminho e a escolha dos nós a serem tomados neste caminho, bem como facilitou a implementação. Além disso, por existir uma única mensagem, o atraso provocado pela mesma, em termos de permanência no buffer, é desprezível, e tambem não há espera entre os nós para transferir a mensagem, quando se conectam a transferência da mensagem é iniciada imediatamente. Um último detalhe, que é intrínseco às implementações de algoritmos de roteamento no ONE, é que a mensagem não pode ser entregue a um nó já antes previamente visitado, ou seja, que já recebeu a mensagem. Em outras palavras, um mesmo nó não recebe a mesma mensagem duas vezes. Como nosso algoritmo foi implementado no simulador ONE, ele tambem tem esta característica da mensagem não passar mais de uma vez pelo mesmo nó. Mesmo que um mesmo par de nós venha a se reconectar, a

mensagem ainda assim não passará por estes nós, pois já foram visitados antes.

- **Algoritmo distribuído:** Durante a simulação, cada nó da rede durante executa uma instância do algoritmo TVFP para então decidir se deve repassar a mensagem que carrega consigo quando se conecta com outro nó que utiliza o algoritmo TVFP.

- **Algoritmo baseado em Djikstra:** O TVFP assume comportamento similar ao algoritmo de Djikstra durante o cálculo do melhor caminho, no sentido que utiliza um sistema de pesos de nós e arestas.

- **Latência de transmissão igual:** Os nós possuem as mesmas interfaces de rede e largura de banda, resultando em igual tempo de transmissão da mensagem para todos. Além disso, todos os nós possuem *buffers* de igual capacidade, e a única mensagem enviada na rede é de tamanho pequeno. Desse modo, o atraso no armazenamento da mensagem é desprezível. Do exposto anteriormente, temos que a latência de transmissão da mensagem é sempre menor que os tempos de encontro, garantindo a transferência da mensagem antes que haja desconexão. Do ponto de vista do TVG, a função de latência não ultrapassa o limite superior dos intervalos de tempos de conectividade nas arestas. Em outras palavras, a mensagem é sempre transferida à tempo, de um nó a outro.

- **Melhor caminho calculado previamente:** O melhor caminho é calculado antes da transferência da mensagem ser iniciada no nó origem, sendo o caminho armazenado em lista global de nós chamada ***fastest_path***, contendo nós pertencentes ao caminho, e na ordem estipulada pelo caminho, e que o TVFP utiliza para saber o próximo nó para onde enviar a mensagem. Além disso o melhor é calculado de modo que um mesmo nó não possa ser visitado mais de uma vez, pois um nó não recebe a mesma mensagem mais de uma vez. Ou seja, não são considerados eventos de reconexão entre um mesmo par de nós, ou em outras palavras, arestas diferentes que ligam um mesmo par de nós, só se inclui no melhor caminho uma das arestas que conecte este par de nós. Da maneira que implementamos o TVFP só existe uma lista *fastest_path*, ou seja, só é calculado e retornado um melhor caminho entre um par de nós origem e destio do TVG.

- **Tempo como peso:** Devido à natureza temporal do TVG, o peso dos nós e das arestas é o **tempo**, de valor igual ao limitante inferior nos intervalos de tempo nas arestas, para o cálculo de melhor caminho, ou seja, de cada intervalo de tempo, só utilizamos um valor, sendo este o valor do limitante inferior de cada intervalo. Além disso, o tempo utilizado em nossa implementação de TVGs, seja ele o tempo de vida do sistema, como o tempo dos intervalos de conectividade, pertence ao domínio dos números reais, ou seja, é um

tempo contínuo, uma vez que o tempo utilizamos nas simulações com o ONE também pertence ao domínio dos números reais, sendo tambem um tempo contínuo.

Para compreender melhor, suponhamos, por exemplo, que no cálculo do melhor caminho pelo TVFP, no TVG há um nó $n79$ ligado a um nó $n22$, sendo que a origem o nó $n79$, e que o rótulo da aresta que liga estes nós é $[536.3000000000534, 696.0000000000897]$, ou seja, uma aresta que fica disponível dos 536.3000000000534 aos 696.0000000000897 segundos. Neste caso o peso da aresta será setado com o valor 536.3000000000534. Do ponto de vista de de uma simulação no ONE, onde a rede de nós é representada pelo TVG, estes mesmos nós $n79$ e $n22$ realmente se conectam a partir dos 536.3000000000534 segundos, desconectando-se a partir de algum tempo maior que 696.0000000000897 segundos. Quando os nós se conectam, dada as considerações feitas acerca da latência de transmissão entre nós e de haver uma única mensagem, e de tamanho pequeno, é esperado que quando os nós $n79$ e $n22$, a transferência da mensagem é imediatamente iniciada, partindo do nó $n79$, tenha entrega garantida ao nó $n22$, uma vez que o tempo gasto para se entregar a mensagem está contido no intervalo, por exemplo, a mensagem é enviada aos 536.3000000000534 segundos (Sendo que o valor do limitante inferior na aresta é 536.3000000000534 segundos, ou seja, transferência iniciada imediatamente e sem espera) e chega ao destino aos 536.6000000000535 segundos, ou seja, o tempo de transmissão é de 0.3000000000000682 segundos, além disso, o tempo de chegada é um tempo ainda dentro do intervalo de conectividade da aresta, o que garante que a transferência da mensagem é feita sem desconexões.

- **Intervalos de tempo separados em arestas individuais:** Separamos cada intervalo de tempo em uma aresta individual, similar à abordagem de (BISWAS S.S.; DOJA, 2013), uma vez que nos cálculos do TVFP, é utilizado um valor de peso, então foi coerente separar cada intervalo. Considerando o exposto sobre a mensagem que circula na rede, como um mesmo nó não recebe a mesma mensagem duas vezes, não é possível visitá-lo novamente, uma vez que já tiver recebido a mensagem, e consequentemente passado-a adiante. Assim temos que o a transferência da mensagem pelas arestas é sempre unidirecional, se a mensagem foi transferida do nó A para o nó B por uma aresta, nunca teremos a situação de que B enviará para A, pois A já foi visitado previamente. Na prática isso significa que nunca serão inclusas as múltiplas arestas de um mesmo par de nós no melhor caminho, apenas uma única aresa, e que for mais conveniente para o caminho, é incluída.

O TVFP separa os nós do TVG em dois conjuntos distintos, a princípio vazios, **Visitados** e **Não-Visitados**, onde o primeiro contém nós já visitados, e portanto avaliados, e o segundo nós

ainda não visitados, e portanto ainda não avaliados. Antes da execução do TVFP, nenhum nó foi visitado e avaliado, e quando existe um melhor caminho do nó origem até um nó arbitrário, este nó arbitrário é colocado no conjunto dos Visitados.

Primeiro atribui-se o valor 0 ao peso do nó origem e um valor infinito para os demais, indicando que nenhum foi alcançado ainda pelo nó origem e, em seguida, o nó origem é adicionado ao conjunto dos Não-Visitados, para ser então ser visitado, e consequentemente visitado, sendo colocado ao conjunto dos Visitados, e a partir daí o algoritmo permanece iterando até o conjunto dos Não-Visitados ficar vazio.

A cada iteração o algoritmo seleciona o nó de menor peso no conjunto dos Não-Visitados e o avalia, sendo este nó chamado de **nó de avaliação**, logo na 1ª iteração, o nó origem é o nó de avaliação. O TVFP procura no TVG por **nós vizinhos** do nó de avaliação que ainda não foram visitados e avaliados, e que possam ser alcançados. Ou seja, incluir nós que fazem sentido temporalmente. Para cada nó, é verificado se o tempo nos mesmos pode ser reduzido, e caso possa, o tempo é atualizado com um valor menor ao que tinha, e então adiciona-se este nó vizinho ao conjunto dos Não-Visitado para ser futuramente visitado e avaliado.

Este processo de redução dos tempos é conhecido como **processo de cálculo do caminho**, é por meio deles que são calculados melhores caminhos alcançáveis a partir de um dado nó origem. Dentre estes melhores caminhos, tambem se encontra o melhor caminho, caso possa ser calculado, procurado entre o nó origem e um nó destino arbitrário. O funcionamento deste processo é explicado na sequência: dado um nó de avaliação, e um nó vizinho alcançável por este e cujo tempo deste nó vizinho pode ser reduzido, se o tempo da aresta que conecta os dois nós, origem e vizinho, for menor que o tempo neste nó vizinho, atualiza-se o peso deste nó vizinho de modo a ter valor igual ao peso desta aresta.

Quando dizemos **nós vizinhos alcançáveis** para inclusão no caminho, referimo-nos aos vizinhos com peso maior que o do nó de avaliação, pois, como no TVG há uma ordem cronológica dos eventos de conexão e desconexão, não faz sentido reduzir o tempo em nós vizinhos com os quais o nó de avaliação nunca se conectaria, uma vez que no tempo que a mensagem chegar a este nó de avaliação, já não há mais conexão deste com seus vizinhos.

Seja o exemplo na figura 4.2, onde $n1$ e $n3$ são, respectivamente, origem e destino. Além disso, considerar que o tempo no TVG da figura pertence ao domínio dos números naturais, sendo, portanto, um tempo discreto. O tempo em $n1$ é 0, e este nó possui como vizinhos $n2$ e $n3$. $n1$ é o nó de avaliação e faz o relaxamento em seus vizinhos, primeiramente em $n2$, e depois em $n3$, cujos tempos passam a ser 4 e 15 respectivamente. O próximo nó de avaliação é $n2$, cujo tempo é 4, e só pode fazer relaxamento em $n3$, cujo tempo atual, 15 segundos, é

reduzido a 9 segundos, ignorando $n4$, pois, esse não é adicionado aos Não-Visitados, como nó para relaxamento, pois o tempo em $n2$ é 4 segundos, não havendo mais conexão de $n2$ com $n4$ no tempo 4. Assim o melhor caminho de $n1$ até $n3$ é $n1 \rightarrow n2 \rightarrow n3$.

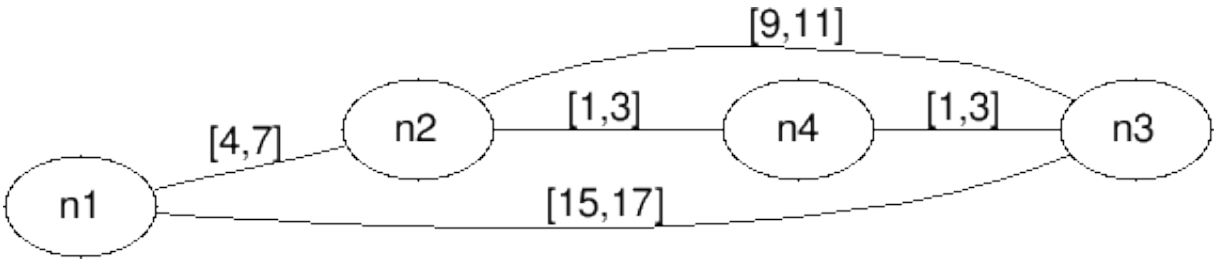


Figura 4.2: Processo de cálculo dos caminhos

Em nossas implementações, o algoritmo 4.1 representa o TVFP, o algoritmo 4.2 o processo de cálculo do melhor caminho, e o algoritmo 4.3 é um método auxiliar responsável por retornar o melhor caminho do nó origem até um nó destino arbitrário. Após o pseudocódigo de cada algoritmo, explicamos os termos presentes em cada algoritmo.

Algoritmo 4.1: *Time-Varying Fastest Path* (TVFP)

```
Entrada: TVG e um nó origem deste TVG.
Saída: Melhor caminho, caso exista, desde o nó origem até o nó destino.

for each node do:
   time[node] <- INFINITY;
end for

visitedNodes <- EMPTY;
unvisitedNodes <- EMPTY;

add sourceNode to unvisitedNodes;
time[sourceNode] <- 0;

while unvisitedNodes is NOT EMPTY do:
   evaluationNode <- getNodeWithLowestTime(unvisitedNodes);
    remove evaluationNode from unvisitedNodes;
    add evaluationNode to visitedNodes;
    pathsCalculationProcess(evaluationNode);
end while
```

Explicando os termos presentes algoritmo 4.1:

- ***time[node]*:** Peso, ou tempo, do nó, neste caso, *node*.
- ***unvisitedNodes*:** Conjunto dos nós ainda não visitados, e portanto não avaliados.
- ***visitedNodes*:** Conjunto dos nós já visitados, e avaliados.
- ***sourceNode*:** Nó origem que passado como entrada para o algoritmo, sendo este o nó a partir do qual serão calculados caminhos alcançáveis.
- ***evaluationNode*:** Nó de avaliação cujos nós vizinhos alcançáveis serão analisados.
- ***getNodeWithLowestTime(unvisitedNodes)*:** Método que procura, no conjunto dos nós não visitados, pelo nó não avaliado de menor peso, ou tempo, retornando este nó.
- ***pathsCalculationProcess(evaluationNode)*:** Chama o método do algoritmo pathsCalculationProcess, de modo que sejam reduzidos os tempos a partir do nó de avaliação (inicialmente sendo o nó origem), de modo que sejam inclusos nos possíveis melhores caminhos.

Algoritmo 4.2: ***pathsCalculationProcess***

```
Entrada: Nó de avaliação.
Saìda: Nós, com seus pesos reduzidos, alcançáveis pelo nó de avaliação, e que
    irão compor os possìveis melhores caminhos.

for each UNVISITED neighbourNode reachable by evaluationNode do:
   edgeTime <- getTime(edge(evaluationNode,neighbourNode));
    if time[neighbourNode] > edgeTime then:
      time[neighbourNode] <- edgeTime;
        add neighbourNode to unvisitedNodes;
        add (neighbourNode,evaluationNode) to predecessors;
end for
```

Explicando os termos presentes algoritmo 4.2:

- ***evaluationNode*:** Nó de avaliação cujos nós vizinhos alcançáveis serão analisados.
- ***neighbourNode*:** Nó vizinho alcançável do nó de avaliação.
- ***edgeTime*:** Peso, ou tempo, da aresta que interliga um par de nós.
- ***getTime(edge(evaluationNode,neighbourNode))*:** Método que retorna o peso, ou tempo, da aresta que interliga o nó de avaliação ao seu nó vizinho.

- ***predecessors*:** Uma estrutura de lista que armazena pares ordenados de nós, cuja estrutura contém sempre um par de nós, sendo um nó e seu nó predecessor. Dizemos predecessor como o nó que vem imediatamente antes desse nó, na formação do caminho. Então está lista armazena vários pares ordenados. Esta lista tem o intuito de armazenar os pares de nós e permitir descobrir a ordem dos nós que irão compor o melhor caminho, por meio deste sistema de parear um nó com seu nó predecessor. Um mesmo nó pode ter diferentes predecessores, o que significa que este mesmo nó faz parte de caminhos diferentes, sendo que em cada um ele pode ter diferentes predecessores.

- ***(neighbourNode,evaluationNode)*:** Par ordenados de nós, onde o primeiro nó, neste caso o nó vizinho, tem como predecessor o segundo nó, neste caso o nó de avaliação. Podemos interpretar como o par que nos diz que o nó de avaliação é predecessor ao nó vizinho.

Para se obter o melhor caminho do nó origem a um nó destino arbitrário, utilizamos um algoritmo auxiliar, chamado ***getPath*** (Algoritmo 4.3), que armazena todos os nós predecessores de um nó em uma estrutura chamada ***predecessors***.

Inicialmente, atribuímos o nó destino a um nó auxiliar ***step***, e a partir daí, o algoritmo itera procurando todos os predecessores de *step*, que na primeira iteração corresponde ao nó destino, e os adiciona ao caminho auxiliar *path*, até que não encontre nenhum nó predecessor adicional.

Da maneira que é montado ***path***, ele se encontra com os nós em ordem invertida, ou seja, do nó destino até o nó origem, sendo necessário corrigí-lo, ou seja, colocá-lo na ordem correta, do nó origem até o nó destino, para uso correto pelo TVFP. Esta correção é feita pelo método *reverse*, ao ser passado o caminho path como entrada para este método.

Antes de ser iniciada a simulação utilizando o TVFP, primeiro o ONE recebe o TVG como entrada, e em seguida o TVFP recebe um nó origem do TVG como entrada, calculando os melhores caminhos a partir deste.

Após isso, o método *getPath* recebe um nó destino qualquer como entrada e retorna, se houver, o caminho deste até a origem, armazenando-o em uma lista global chamada ***fastest_path***, que só pode ser acessada pelo nó contendo a mensagem, evitando *deadlocks*. Além disso a lista armazena os nós na ordem correta a ser tomada pelo caminho calculado pelo TVFP, desde a origem até o destino.

Como a mensagem parte do nó origem, este é removido da lista *fastest_path*, evitando que a mensagem não seja passada adiante, visto que o próximo nó é o próprio nó origem. Por fim, é iniciada a simulação e o nó origem consulta a lista *fastest_path* para saber o próximo nó para envio, removendo-o da mesma, e envia a mensagem para ele. O processo se repete até a

mensagem chegue ao destino.

Algoritmo 4.3: *getPath*

```
Entrada: Nó destino.
Saída: Melhor caminho, já ordenado, desde o nó origem até o nó destino.

path <- new LinkedList;
fastest_path <- new LinkedList;
step <- destinationNode

if predecessors.get(step) is NULL then:
   return NULL;
end if

add step to path;

while predecessors.get(step) is not NULL do:
   step <- predecessors.get(step);
    add step to path;
end while

fastest_path <- reverse(path);
```

Explicando os termos presentes algoritmo 4.3:

- ***path*:** Lista encadeada de nós que irão compor o melhor caminho entre nó origem e o nó destino arbitrário passado como entrada para este algoritmo. Nela serão colocados os nós, em ordem inversa, sendo o primeiro nó da lista o nó destino, e o último nó, o nó origem, portanto este caminho deve ser invertido posteriormente para que a ordem esteja correta.
- ***step*:** Variável auxiliar que armazena os nós a serem incluídos no caminho.
- ***predecessors.get(step)*:** Significa que está sendo consultado o método *get* de *predecessors* saber qual é o nó predecessor de *step*, e então este nó prededecessor é retornado.
- ***fastest_path*:** Lista encadeada de nós recebe os nós da lista *predecessors* ordenados na ordem correda, por meio do método *reverse*..

Por fim, do ponto de vista da implementação em Java utilizada no ONE, nosso algoritmo

TVFP encontra-se implementado em uma classe chamada ***TVFPRouter.java***, sendo utilizada apenas como ***TVFPRouter*** no arquivo de configuração do ONE.

Nos Anexos deste trabalho abordamos com maior detalhes a implementação em java utilizada no TVFP.

Na sequência descrevemos brevemente o funcionamento do simulador ONE.

4.2 O simulador ONE

O ***Opportunistic Network Environment*** (**ONE**) (KERäNEN; OTT; KäRKKäINEN, 2009) é um simulador de mobilidade e de redes implementado em Java, cuja versão mais recente é a 1.6.0, datada de outubro de 2015. Essa versão é a que utilizamos neste trabalho, para avaliação de roteamento e aplicações em redes DTN.

Uma vez que este simulador foi implementado em linguagem Java, está representado na forma de um projeto em Java facilmente importável para uso e programação em alguma IDE, por exemplo Eclipse [1] ou NetBeans [2].

Para avaliar o TVFP, implementamos código em Java relevante ao TVFP e compatível com o simulador ONE, e realizamos simulações no ONE com o TVFP, sendo que embarcamos no ONE, a funcionalidade de criação de TVGs, implementando estruturas de dados adequadas para armazenar o TVG. Além disso, o contexto sobre o qual aplicamos o TVFP é de uma frota de ônibus, representada por uma rede DTN com nós, que representam os ônibus, que seguem a movimentação descrita em um *trace* de mobilidade externo.

Na sequência, explicamos o funcionamento geral do simulador ONE.

4.2.1 Funcionamento geral do ONE

Uma simulação depende dos parâmetros lidos de um arquivo de configuração chamado ***default_settings.txt***, o qual é carregado em toda simulação. Desse modo, é possível configurar vários aspectos de uma simulação por meio de seus parâmetros, como por exemplo:

- **Tempo da simulação:** É o tempo limite, em segundos, enquanto a simulação permanecerá executando, e.g. um tempo de 20000 segundos, significa que a simulação executará

[1] http://www.eclipse.org/
[2] https://netbeans.org/

dos 0 a 20000 segundos. Além disso, estabelecemos para este trabalho que aas simulações utilizam um tempo contínuo, pertence ao conjunto dos números reais, sendo que este tempo é incrementando a cada 0.1 segundos.

Do ponto de vista de um TVG, considera-se este tempo de simulação como equivalente ao tempo de vida τ do sistema dinâmico modelado pelo TVG.

- **Grupos de nós:** Os nós em uma simulação são organizados em grupos, de modo que possa ser escolhido quantos nós pertencerão ao grupo, e também todos os nós de um mesmo grupo possuem características em comum, e.g. largura de banda, alcance de transmissão de suas interfaces de rede, rótulo identificador, *buffer* de armazenamento, **modelo de mobilidade** utilizado pelos nós do grupo, etc.

  A presença de grupos é interessante pois permite que em nossa simulação tenhamos nós dos mais variados tipos, por exemplo nós que simulam a mobilidade de pedestres, nós que simulam a mobilidade de veículos, nós que seguem uma rota fixa, etc.

  Para nós interessa este parâmetro, pois podemos organizar os nós dos *traces* em um só grupo, grupo de ônibus, de modo que o total de nós do grupo é o mesmo total de nós do *trace*. Para compreender melhor o que queremos dizer com relação ao total de nós no grupo e no *trace*, suponhamos, por exemplo que no *trace*, já convertido para formato compatível com o ONE, que será utilizado na simulação possui registrada a movimentação de um total de 1160 ônibus, sendo assim, o grupo de nós no ONE será configurado para que comporte 1160 nós, começando do 0 e indo até o 1159.

- **Algoritmo de roteamento do grupo de nós:** Permite configurar qual classe de algoritmo de roteamento será utilizada pelo grupo de nós, determinando como irão se comportar quando transmitem mensagems entre si.

  Este parâmetro é interessante para nosso trabalho, pois com eles podemos incorporar o TVFP a um grupo de nós específico, assim sendo, eles podem se comportar conforme as regras do algoritmo de roteamento TVFP. Para tanto, em nossa implementação, o TVFP se encontra em uma classe chamada ***TVFPRouter***.

- **Modelo de mobilidade:** Refere-se à maneira como os nós de um mesmo grupo irão se movimentar durante uma simulação, sendo que o simulador oferece suporte a diversos tipos de modelos de mobilidade, desde modelos aleatórios (*random waypoint*), onde os nós movem-se de maneira aleatória, até um modelo de mobilidade que reproduz a movimentação descrita em *traces* de mobilidade externos.

Este modelo de mobilidade em específico, que faz uso de *traces* externos, é conhecido como ***ExternalPathMovement***, e recebe como entrada dois arquivos, que representam um *trace* de mobilidade externo convertido para o formato compatível com o ONE: um arquivo de movimentos, ***traceFile***, e um arquivo de tempos de atividade, ***activeFile***.

Para este trabalho nos interessa utilizar o modelo de mobilidade *ExternalPathMovement*, visto que com o mesmo podemos reproduzir os *traces* com a mobilidade real de ônibus, oriundos do repositório CRAWDAD (JETCHEVA, 2003). Por meio de uma aplicação *parser* em Python, é convertido os arquivos *traceFile* e *activeFile*, que respectivamente, contêm todas posições ocupadas pelos ônibus do *trace* e os intervalos de tempo quando estes ônibus permaneceram em atividade.

- **Relatórios:** Permite gerar relatórios ao final da simulação. O ONE oferece suporte a vários relatórios por padrão, mas nos interessa o relatório dos eventos decorridos na simulação. Ou seja, o ***eventlog***, pois do mesmo, extraímos as informações referentes aos eventos de conexão e desconexão entre pares de nós, respectivamente eventos ***CONN UP*** e ***CONN DOWN***. Desse forma, com nossa outra aplicação *parser* em Python, processamos o *eventlog*, gerando como saída um TVG, que será utilizado junto ao TVFP em simulações.

Em resumo, são de interesse para este trabalho os parâmetros referentes ao modelo de mobilidade, ao algoritmo de roteamento, e por consequência os tempos de simulação e grupos de nós.

Nos Anexos deste trabalho discutimos sobre detalhes de implementação, do ponto de visto do desenvolvedor, relevantes à confecção deste trabalho, bem como um detalhemento aprofundado sobre os parâmetros da simulação.

Na sequência, explicamos os passos envolvidos na nossa metodologia que faz uso do simulador ONE para gerar dados a serem minerados, que serão posteriormente utilizados em roteamento pelo nosso algoritmo TVFP

4.3 Metodologia do processo de mineração de dados até posterior uso dos mesmos em roteamento

Após a explicação da contribuição principal, o algoritmo, e do funcionamento do simulador ONE, destacaremos um pequeno detalhamento de todas as fases para uma compreensão completa do nosso método.

Nosso processo de mineração de dados para geração de TVG, até a utilização dos mesmo em simulação é dividido em 4 fases, destacadas a seguir.

4.3.1 Fase 1: Tratamento de *traces*

Esta é a 1ª fase de nosso processo de mineração, onde desenvolvemos uma ferramenta *parser* em Python 3.4.3 com a qual processamos o *trace* de mobilidade real oriundo do CRAWDAD, convertendo-o para formato compatível com o simulador ONE.

Este tratamento é necessário pois antes dos *traces* do CRAWDAD possam ser utilizados em uma simulação no ONE, eles devem ser convertidos em dois outros arquivos, *traceFile* e *activeFile*, onde:

- ***traceFile*:** Arquivo que registra todas as posições ocupadas pelos nós, no caso os ônibus, e quando tais posições foram ocupadas, sendo portanto o arquivo responsável pelas posições que os nós ocuparão durante a simulação no ONE.
- ***activeFile*:** Arquivo que registra os intervalos de tempo em que cada ônibus permaneceu em atividade durante o *trace*. Quando dizemos que um ônibus permaneceu em atividade, nos referimos ao intervalo de tempo quando suas interfaces de rede permaneceram ativas recebendo transmissões de nós que estão ao seu alcance, ou seja, um nó que está efetivamente sendo útil durante uma simulação, no sentido que transmite informações. Do ponto de vista da simulação um nó está no alcance do outro quando estão dentro da área do alcance de suas interfaces de rede, podendo trocar mensagens (enviar e receber) com os nós que estão ao seu alcance, e o mesmo se aplica a estes pois o outro nó tambem estará ao alcance destes nós para os quais envia mensagem.

Em cada linha do *trace* original do CRAWDAD constam as seguintes informações:

- **Horário** em que foi registrada esta movimentação nesta entrada do *trace*.
- **Identificador** do ônibus cuja movimentação foi registrada nesta entrada do *trace*.
- **Rota** do ônibus durante sua movimentação, nesta entrada do *trace*.
- **Valor desconhecido** não utilizado.
- **Posições X e Y** ocupadas pelo ônibus registrada nesta entrada do *trace*.

Para a criação dos arquivos *traceFile* e *activeFile*, e consequentemente, uma simulação no ONE utilizando *traces* de mobilidade, dessas informações listadas, só interessam o horário, identificador do ônibus e suas posições X e Y, para a confecção do *traceFile* e *activeFile*.

Mais detalhes acerca dos *traces* do CRAWDAD, bem como dos arquivos *traceFile* e *activeFile* podem ser encontrados nos Anexos deste trabalho.

4.3.2 Fase 2: Gerando um *eventlog*

Esta é a 2ª fase do nosso processo de mineração, onde realizamos uma simulação no ONE que faz uso do *trace* convertido da 1ª fase, gerando um relatório dos eventos, especialmente os eventos de encontros, decorridos na simulação. Este relatório é conhecido como ***eventlog***.

No *eventlog* são registrados nas linhas do relatório todos os eventos que aconteceram do começo até o fim da simulação, onde em cada linha temos:

- **Tempo** quando o evento ocorreu.
- **Tipo do evento** ocorrido.
- **Nós envolvidos** no evento.
- **Parâmetros adicionais** que dependem do tipo de evento ocorrido.

4.3.3 Fase 3: Gerando um TVG

Esta é a 3ª fase do nosso processo de mineração, onde utilizando outra ferramenta *parser* em Python 3.4.3 que nós desenvolvemos, onde processamos o *eventlog* gerando anteriormente, de modo a gerar um TVG em formato compatível com o do software de vizualização e geração de grafos *GraphViz*.

Dentre os eventos ocorridos no *eventlog*, nos interessam aqueles que informam quando houve **conexão** e **desconexão** entre pares de nós, bem como suas possíveis reconexões, uma vez que estas informações sobre a conectividade entre nós é utilizada na geração de TVGs.

4.3.4 Fase 4: Simulação com TVFP utilizando o TVG

Esta é a 4ª e última de fase do nosso processo de mineração, onde refazemos a mesma simulação da 2ª fase com mesmo o *trace* de mobilidade utilizado, mas desta vez incorporando

o TVG gerado na 3ª fase ao processo de roteamento do simulador ONE, de modo que possamos realizar simulações com o TVFP, que faz uso deste TVG.

Capítulo 5

EXPERIMENTOS E RESULTADOS

Para avaliar o TVFP em nosso método proposto, conduzimos experimentos utilizamos o simulador ONE na versão 1.6.0 para reproduzir *traces* de mobilidade reais de ônibus obtidos do repositório CRAWDAD [1] (JETCHEVA, 2003). Sendo que estes incluem informações em pequenos intervalos de tempo da posição geográfica dos ônibus de King County, Seattle no estado de Washington (PORTUGAL-POMA, 2013, 2014).

Em específico, para os experimentos utilizamos *traces* das datas 31/10/2001 e 02/12/2001, onde o primeiro contém a movimentação dos ônibus de 00:00 até 19:00, e o segundo a movimentação de 00:00 até 17:17:20. Adicionalmente geramos um *trace* sintético derivado do *trace* de 02/12/2001, contendo os mesmos ônibus, porém ocupando posições levemente alteradas, para avaliarmos a precisão do TVFP utilizando um TVG dito "médio", originado de um TVG intermediário, ou médio, entre o TVG oriundo do *trace* original e o TVG oriundo do *trace* sintético.

Nossas simulações estão divididas em 2 grupos, ambos utilizando *traces* externos: (i) Simulação utilizando um algortimo auxiliar, ***ConnectionsOnly***, desenvolvido por nós, onde não há trocas de mensagens, de modo que o relatório de eventos resultante registra apenas os eventos de conexão e desconexão entre pares de nós, otimizando e acelerando o processo de *parsing* que gera TVGs a partir deste relatório, e (ii) Simulações com os algoritmos TVFP, *FirstContact*, *Epidemic* e *Spray-and-Wait* (SaW) Binário.

Utilizamos o SaW em seu modo binário pois o espalhamento das mensagens é controlado no mesmo, reduzindo o *overhead* de geração de mensagens. Os parâmetros avaliados, ao usar o SaW, foi o limite máximo de cópias da mensagem que podem estar presentes na rede durante uma simulacão. Assim sendo, para cada cenário configuramos um valor máximo específico

[1] https://crawdad.cs.dartmouth.edu/

para o limite de cópias de mensagens, que de acordo com nossos resultados, é o valor mínimo que deve-se setar o limite máximo de cópias para cada cenário para permitiu ao SaW entregar a mensagem no mesmo tempo que que o TVFP e *Epidemic*. Portanto, nós setamos o valor máximo de cópias para o SaW usando estes valores específicos para cada cenário. Adicionalmente, fizemos simulações com o algoritmo *FirstContact*, pois tal qual o TVFP, também trabalha em modo *single-copy*.

Fizemos dois tipos de experimento, no primeiro analisamos o **desempenho** do TVFP e no segundo a **precisão** do TVFP usando TVG médio, que descrevemos nas seções 5.1 e 5.2.

A tabela 5.1 lista os parâmetros de configuração utilizados em todas as simulações.

Tabela 5.1: Parâmetros utilizados em todas as simulações

Trace	31/10/2001	02/12/2001	Sintético
Mod. mobilidade	*Ext.PathMov.*	*Ext.PathMov.*	*Ext.PathMov.*
Algoritmo	**Grupo I:** *ConnectionsOnly* **Grupo II:** TVFP *FirstContact* *Epidemic* *Spray-and-Wait*	**Grupo I:** *ConnectionsOnly* **Grupo II:** TVFP	**Grupo II:** TVFP
Largura de banda	5MBps	5MBps	5MBps
Alcance	50m	50m	50m
Buffer	2GB	2GB	2GB
Tam. mensagem	128KB	128KB	128KB
Qtd. inicial de mensagens	1	1	1
Número de nós	1160	382	382

5.1 Análise de desempenho

Neste experimento utilizamos o *trace* de 31/10/2001, e o TVG derivado desse *trace*. Nós então analisamos o desempenho do TVFP comparado-o com os algoritmos *FirstContact*, *Epidemic* e *Spray-and-Wait*, em 5 cenários específicos. Sendo que em cada um realizamos simulações com os algoritmos, e escolhemos um nó origem e destino, analisando o tempo total gasto pela mensagem durante a viagem para ser levada da origem ao destino, sendo este tempo conhecido como **tempo total de viagem** da mensagem entre nó origem e destino, e o número **total de *hops*** feitos por ela durante esse trajeto e também o número de cópias.

Os valores registrados dos tempos de partida e de chegada são sempre relativos ao início da

simulação, ou seja, 0 segundos. Para exemplificar, quando dizemos que a mensagem chegou ao destino, utilizando o TVFP, aos 11152.8 segundos, queremos dizer que é a mesma chegou no tempo de 11152.8 segundos, decorridos desde o início da simulação, ou seja, o tempo 0 segundos. Já o tempo total de viagem não é regido por este início da simulação como referencialde tempo, ou seja, o tempo total de viagem não se refere a algum valor de tempo decorrido desde o início da simulação, pois o mesmo é calculado subtraindo o valor do tempo de partida do tempo de chegada, ou seja, $tempo\_total\_de\_viagem = tempo\_de\_chegada - tempo\_de\_partida$, de maneira análoga ao Δt da Física.

Fizemos simulações com o algoritmo SaW em modo binário pois provoca um menor *overhead* de geração de mensagens, e para cada cenário, especificamos um número máximo de cópias que podem estar presentes na rede durante a simulação, sendo este número de cópias os parâmetros avalaiados para o SaW. Isso foi feito de modo que as simulações realizadas, permitam ao SaW entregar a mensagem no mesmo tempo que o TVFP e o *Epidemic*. Assim sendo, para cada cenário foi setado um valor diferente para número máximo de cópias que o SaW pode ter na rede, sendo este valor específico para cada cenário o que permitiu que o SaW entregue a mensagem ao destino no mesmo tempo que o TVFP e *Epidemic*, bem como gastar o mesmo número de *hops* que os algoritmos TVFP e *Epidemic*. Este valor de limitante do número máximo de cópias se aplica apenas às cópias geradas pelo SaW durante a simulação, não contabilizando no seu contador de cópias a instância inicial da mensagem que é gerada na origem, por exemplo se em um cenário estiver estabelecido que o número máximo de cópias na rede do SaW, ou seja, seu valor de L, é 20, mas que o número máximos de mensagens geradas pelo SaW, foi 20, quer dizer que foram geradas 19 cópias da mensagem mais a mensagem original que foi gerada na origem, ou seja, não atingiu o limite máximo de cópias da mensagem presentes na rede, estipulado pelo SaW.

Além disso, também fizemos simulações com o algoritmo *FirstContact*, pois similarmente ao TVFP, também trabalha em modo *single-copy*.

Os nós, que fazem uso do TVFP durante a simulação, possuem conhecimento prévio do melhor caminho, uma vez que, anteriormente ao início da transmissão, já está calculado, quando existe, o melhor caminho, que fica armazenado na lista global *fastest_path*

A topologia do TVG utilizado neste experimento, compreende um tempo de simulação de 23250 segundos. Escolhemos esse tempo de simulação pois valores maiores tornam o processamento muito lento quando tentamos gerar o TVG em formato Graphviz. Além disso, na topologia do TVG gerado para os 23250 segundos de simulação, existe um total de 6 ***clusters* de nós conectados**, sendo cada um, individualmente, um grafo completamente conectado, mas

que não se conectaram, ao longo do tempo, com nenhum outro nó que não os do mesmo *cluster* de nós. Observamos a extensa figura do TVG completo, podemos analisar cada um dos 6 *clusters*, da esquerda para a direita, e fazemos as seguintes considerações sobre os mesmos:

- **1º *Cluster* de nós:** Um *cluster* pequeno que não foi utilizado nos experimentos.
- **2º *Cluster* de nós:** Outro *cluster* pequeno, utilizado nos cenários 1 e 2 por permitir caminhos alternativos até um mesmo nó, possibilitando comparar a heurística do TVFP comparando-a com a heurítica gulosa do *FirstContact*, visto que entregam a mensagem ao destino em tempos diferentes.
- **3º *Cluster* de nós:** Um *cluster* imenso,complexo e conectado, utilizado nos cenários 3, 4 e 5, com o qual demonstramos que o *FirstContact* muitas vezes não consegue entregar a mensagem ao destino.
- **4º *Cluster* de nós:** *cluster* pequeno que não foi utilizado nos experimentos.
- **5º *Cluster* de nós:** Um *cluster* pequeno que não foi utilizado nos experimentos.
- **6º *Cluster* de nós:** Um *cluster* pequeno que não foi utilizado nos experimentos.

A figura 5.1 representa um recorte do TVG completo gerado dos 23250 segundos de simulação, contados a partir do início da mesma, sendo que a figura representa o 2º *cluster* de nós comentado anteriormente, que é utilizado nos cenários 1 e 2 deste experimento. Geramos este recorte pois seria impossível representar o TVG completo em uma só figura e para que o leitor possa ter uma melhor compreensão dos tempos de conectividade e nós presentes no TVG gerado. Além disso, na figura, a alta precisão dos tempos de conectividade com várias casas decimais se dá pela maneira que a precisão de tempo é tratada no simulador, utilizando várias casas decimais. Além, disso, para facilitar a legibilidade dos múltiplos intervalos de conectividade, antes todos escritos na mesma linha, organizamo-os de modo que fiquem dispostos um acima do outro.

Por fim, configuramos um valor de tempo de vida TTL de 400 minutos, que equivale a 24000 segundos, sendo um valor alto o suficiente para que a mensagem gerada no início da simulação, ou seja 0 segundos, não sofra de eventos como *messge drop*, ou seja, a mensagem ser perdida, e possa percorrer quaisquer caminhos gerados durante o tempo de vida completo de 23250 segundos da simulação. A motivo de aumentar o valor de TTL da mensagem para um valor maior que o tempo de vida do sistema, ou tempo máximo da simulação, é devido ao fato que alguns nós só passarão a se mover, possivelmente se conectando com demais nós,

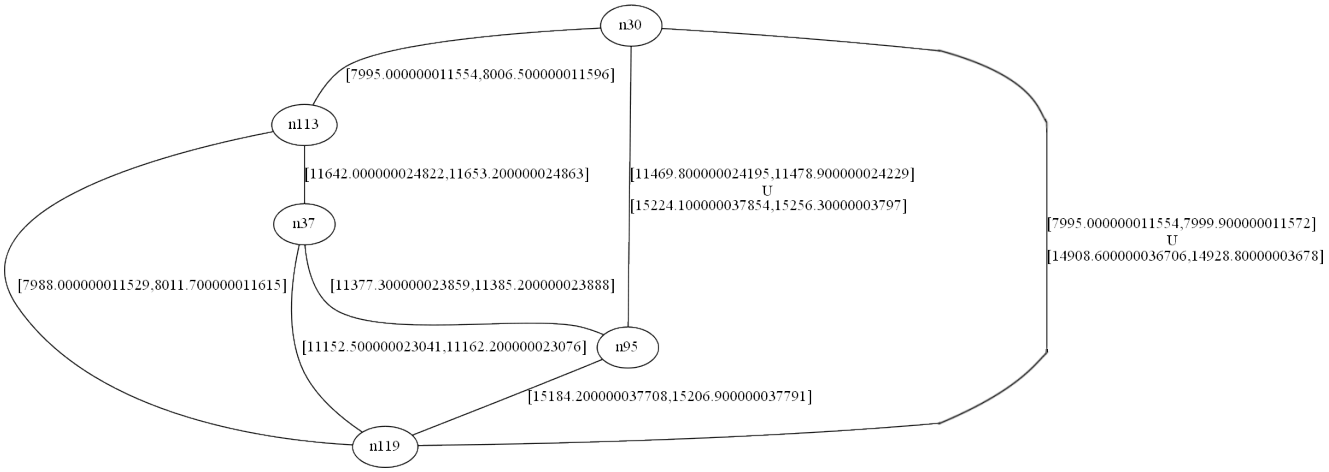


Figura 5.1: 2º *cluster* de nós da topologia utilizada no experimento

próximo ao final da simulação, aos 18000 segundos. Se um nó como este fosse o nó origem, onde a mensagem é gerada, e a mensagem tivesse um valor baixo de TTL, pode ocorrer da mensagem ser perdida antes mesmo da transferencia da mesma ser iniciada, ainda que tenha sido um calculado um caminho válido entre origem e destino, ou mesmo que a transferência da mensagem seja iniciada, a mensagem pode sofrer um evento *message drop* enquanto percorre pelo melhor caminho que foi calculado.

Em cada cenário listamos os resultados das simulações feitas nos experimentos em tabelas com os dados quantitativos, bem como geramos gráficos de barra que resumem de forma mais intuitiva os resultados desta tabela. Na parte superior dos gráficos de barra estão indicados o **total de mensagens** geradas por cada algoritmo durante os cenários, sendo que este total é composto da **mensagem gerada na origem** mais as **cópias redundantes** da mesma que foram geradas. Desta maneira, exemplificando, um valor 1 sobre as os gráficos barras indicaria que apenas a mensagem gerada foi gerada na simulação, enquanto que um valor 127 indica que tem-se a mensagem gerada na origem mais 126 cópias redundantes da mensagem. Por fim, descrevemos em detalhes para cada cenário, os resultados pertinentes. Sabendo desta informação, não devemos confudir, por exemplo, se no experimento é dito que o SaW trabalhou com no máximo 2 cópias, e foram geradas 3 mensagens pelo mesmo, quer dizer que ele esgotou o limite máximo de cópias da mensagem que pode ter na rede, ou seja 2, somada à mensagem padrão gerada na origem, portanto 3.

5.1.1 Cenário 1 - Origem: $n30$ / Destino: $n37$

Neste cenário de simulação, o melhor caminho calculado pelo TVFP neste TVG é $n30 \rightarrow n119 \rightarrow n37$, sendo este o caminho percorrido pela mensagem durante a simulação. O nó origem

é *n*30 e o nó destino é *n*37, e o *cluster* de nós utilizado pelos algoritmos é o **2º *cluster* de nós**.

Para o TVFP, o tempo de partida é 7995 segundos em relação ao início da simulação, o menor tempo de chegada possível da mensagem ao destino é 11152.8 segundos em relação ao início da simulação, tendo tempo total de viagem de 3157.8 segundos, e número total de *hops* necessários para se levar a mensagem ao destino sendo 2.

A tabela 5.2 e o gráfico da figura 5.2 mostram os resultados obtidos.

Tabela 5.2: Desempenho do TVFP ao enviar uma mensagem de *n*30 para *n*37

Origem: *n*30 / **Destino:** *n*37					
Algoritmo	**Tempo de viagem**	***Hops***	**Msg. criadas**	**Partida**	**Chegada**
TVFP	3157.8s	2	1	7995s	11152.8s
FirstContact	3647.3s	3	1	7995s	11642.3s
Epidemic	3157.8s	2	5	7995s	11152.8s
SaW (Máx.: 2 cópias)	3157.8s	2	3	7995s	11152.8s

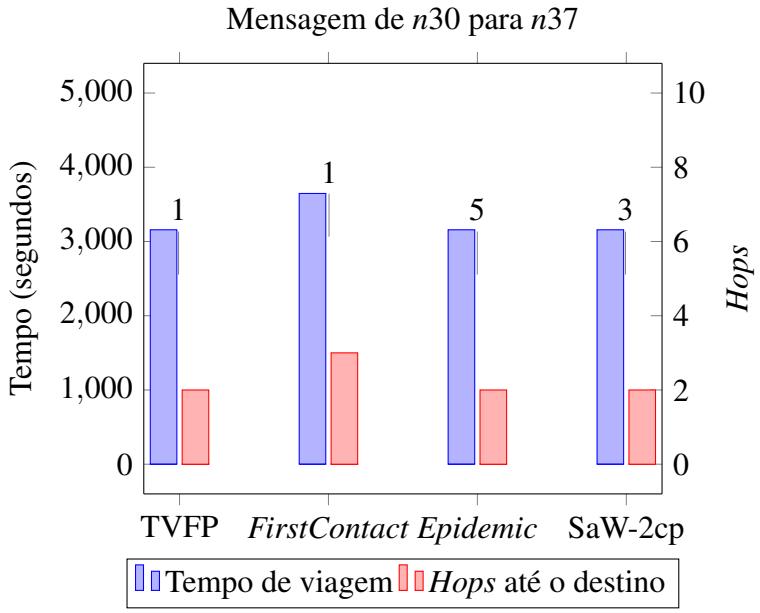


Figura 5.2: Comparando TVFP com demais algoritmos (*n*30 até *n*37)

- **Comparação TVFP X *FirstContact*:** Ambos possuem iguais tempos de partida, diferindo nos tempos de chegada (TVFP: 11152.8 segundos / *FirstContact*: 11642.3 segundos), e portanto, diferindo nos tempos totais de viagem (TVFP: 3157.8 segundos / *FirstContact*: 3647.3 segundos). Além disso, diferem nos números totais de *hops* (TVFP: 2 *hops* / *FirstContact*: 3 *hops*). O caminho percorrido pelo *FirstContact* neste cenário é $n30 \rightarrow n119 \rightarrow n113 \rightarrow n37$.

  Ainda que ambos trabalhem com uma única instância da mensagem, o controle de roteamento presente no TVFP o permitiu aguardar por melhores oportunidades de contato,

selecionando nós específicos para transmissão. Ainda que tenha igual tempo de partida ao TVFP, o *FirstContact*, percorreu um caminho mais longo, diferindo do TVFP em 1 *hop*, resultando em um tempo total de viagem maior, pois o *FirstContact* não possui ciência dos próximos contatos, sendo que usa uma heurística gulosa. Especificamente falando, nota-se que até o nó $n119$ o caminho percorrido por ambos TVFP e *FirstContact* estava idêntico, a diferença é que o *FirstContact*, por não aguardar para se conectar com o nó $n37$ posteriormente, assim chegando ao destino mais cedo, optou por se conectar mais cedo com o nó $n113$, que por sua vez se conectou ao nó $n37$ mais tarde, sendo que se a mensagem tivesse permanecido em $n119$, esperando pelo nó $n37$, o evento de conexão teria acontecido mais cedo, sendo esta decisão de espera a que o TVFP tomou por saber de antemão o melhor caminho a ser percorrido. É importante notar que o *cluster de nós* utilizado neste cenário é pequeno, tendendo, portanto, a aumentar as chances do *First-Contact* de entregar a mensagem ao destino no mesmo tempo do TVFP, que é o menor tempo possível previsto pelo TVG, uma vez que há poucos nós, portanto poucas possibilidades de caminhos alternativos, mas vemos pelos resultados que não foi isso que ocorreu. Além disso, sendo o *cluster* pequeno,são maiores as chances do *FirstContact* conseguir entregar a mensagem ao destino sem que a mesma se perca no caminho, não conseguindo atingir o destino, que é algo comum para o *FirstContact* quando trabalha com um *cluster* grande de nós, como é o caso dos cenários 3, 4 e 5 como veremos adiante.

Considerando o tempo total de viagem de 3647.3 segundos do *FirstContact* como 100%, então o tempo total de viagem de 3157.8 segundos do TVFP representa 86.58% disso, reduzindo em 13.42% o tempo total de viagem. Considerando os 3 *hops* do *FirstContact* como 100%, então os 2 *hops* do TVFP representam 66.67% disso, reduzindo em 33.33% o número total de *hops* gastos para se levar a mensagem da origem ao destino.

- **Comparação TVFP X *Epidemic*:** Ambos possuem iguais tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem o mesmo número de *hops* para entregarem a mensagem ao destino. Isso se deve à política de *flooding* do *Epidemic* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP.

  O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o *Epidemic* gerou um total de 5 instâncias da mensagem, ou seja, a mensagem original mais 4 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Este número de cópias poderia seria maior se houvessem mais oportunidades de contato com nós sem a cópia da mensagem, ou seja, propiciando a ocorrência de mais replicações da mensagem, conforme ocorre o processo de inundação inerente ao *Epidemic*.

O número pequeno de cópias adicionais, ou seja, 4, por parte do Epidemic neste cenário é compreensível, visto que o *cluster* dos nós que utilizam o Epidemic forma uma sub-rede esparsa e menos densa de nós, fazendo que o *overhead* gerado não seja significativo, mas como veremos nos cenários 3, 4 e 5, este número de cópias será maior, pois o *cluster* utilizado nestes cenários, o 3º *cluster*, é denso em nós e conexões, contrastando com o *cluster*.

Considerando as 5 mensagens do *Epidemic* como 100%, então a única mensagem do TVFP representa 20% disso, reduzindo em 80% o *overhead* de geração de mensagens.

- **Comparação TVFP X SaW (Número máximo de cópias: 2):** Ambos possuem iguais tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem os mesmos números totais de *hops* para levarem a mensagem da origem ao destino. Isso se deve à política de *flooding*, ainda que controlado por um limitante de número máximo de cópias presentes na rede, do *SaW* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP. O valor do limitante do número máximo de cópias do SaW neste cenário assume um valor pequeno, ou seja, 2, bem como o número pequeno de cópias adicionais geradas da mensagem, ou seja, 2, e dentre estas a cópia que chegou ao destino, por parte do SaW neste cenário são compreensíveis, uma vez que o *cluster* utilizado é pequeno, assim a tendência é que o SaW não precise espalhar uma grande quantidade de cópias até encontrar o o caminho idêntico ao percorrido pela mensagem no TVFP, o que justifica o baixo valor, no caso, 2, do limitante do número de cópias da mensagem que podem existir na rede.

  O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o SaW gerou um total de 3 instâncias da mensagem, ou seja, a mensagem original mais 2 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Neste cenário notamos que foi atingido o limite máximo de 2 cópias na rede estipulado pelo SaW, portanto replicações adicionais da mensagem não seriam possíveis caso surgissem mais oportunidades de contato posteriormente.

  Considerando as 3 mensagens do SaW como 100%, então a única mensagem do TVFP representa 33.33% disso, reduzido em 66.67% o *overhead* de geração de mensagens.

5.1.2 Cenário 2 - Origem: $n113$ / Destino: $n95$

Neste cenário de simulação, o melhor caminho calculado pelo TVFP neste TVG é $n113 \rightarrow n119 \rightarrow n37 \rightarrow n95$, sendo este o caminho percorrido pela mensagem durante a simulação. O

nó origem é $n113$ e o nó destino é $n95$, e o *cluster* de nós utilizado pelos algoritmos é o **2º *cluster* de nós**.

Para o TVFP, o tempo de partida é 7988 segundos em relação ao início da simulação, o menor tempo de chegada possível da mensagem ao destino é 11377.6 segundos em relação ao início da simulação, tendo tempo total de viagem de 3389.6 segundos, e número total de *hops* necessários para se levar a mensagem ao destino sendo 3.

A tabela 5.3 e o gráfico da figura 5.3 mostram os resultados obtidos.

Tabela 5.3: Desempenho do TVFP ao enviar uma mensagem de $n113$ para $n95$

Origem: $n113$ / **Destino:** $n95$					
Algoritmo	**Tempo de viagem**	***Hops***	**Msg. criadas**	**Partida**	**Chegada**
TVFP	3389.6s	3	1	7988s	11377.6s
FirstContact	3482.1s	3	1	7988s	11470.1s
Epidemic	3389.6s	3	5	7988s	11377.6s
SaW (Máx.: 7 cópias)	3389.6s	3	5	7988s	11377.6s

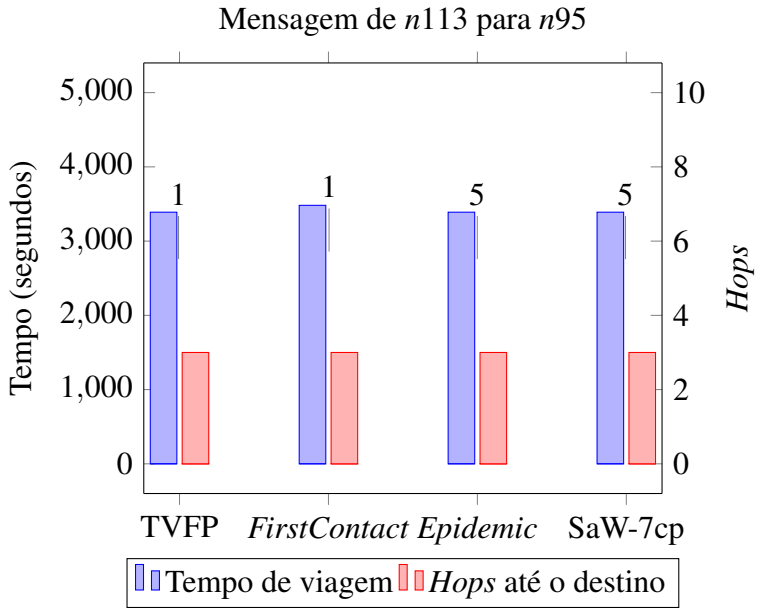


Figura 5.3: Comparando TVFP com demais algoritmos ($n113$ até $n95$)

- **Comparação TVFP X *FirstContact*:** Ambos possuem iguais tempos de partida, diferindo nos tempos de chegada (TVFP: 113777.6 segundos / *FirstContact*: 11470.1 segundos), e portanto, diferindo nos tempos totais de viagem (TVFP: 3389.6 segundos / *FirstContact*: 3482.1 segundos), além de gastarem o mesmo número de *hops* para entregarem a mensagem ao destino. O caminho percorrido pelo *FirstContact* neste cenário é $n30 \rightarrow n119 \rightarrow n30 \rightarrow n95$.

Ainda que ambos trabalhem com uma única instância da mensagem, o controle de roteamento presente no TVFP o permitiu aguardar por melhores oportunidades de contato, selecionando nós específicos para transmissão. Ainda que tenha igual tempo de partida ao TVFP, o *FirstContact*, percorreu fez uma escolha inadequada de nós no caminho percorrido pela mensagem, resultando resultando em um tempo total de viagem maior, pois o *FirstContact* não possui ciência dos próximos contatos, sendo que usa uma heurística gulosa. Especificamente falando, nota-se que até o nó $n119$ o caminho percorrido por ambos TVFP e *FirstContact* estava idêntico, a diferença é que o *FirstContact*, por não aguardar para conectar posteriormente com o nó $n30$, e depois ao $n95$, e assim chegando mais cedo ao destino, optou por se conectar mais cedo com o nó $n37$, que por sua vez se conectou ao nó $n95$ mais tarde, sendo que se a mensagem tivesse permanecido em $n119$, esperando pelo nó $n30$, que por sua vez se conectaria com $n95$, o evento de conexão teria acontecido mais cedo, sendo esta decisão de espera a que o TVFP tomou por saber de antemão o melhor caminho a ser percorrido. É importante notar que o *cluster de nós* utilizado neste cenário é pequeno, tendendo, portanto, a aumentar as chances do *FirstContact* de entregar a mensagem ao destino no mesmo tempo do TVFP, que é o menor tempo possível previsto pelo TVG, uma vez que há poucos nós, portanto poucas possibilidades de caminhos alternativos, mas vemos pelos resultados que não foi isso que ocorreu. Além disso, sendo o *cluster* pequeno,são maiores as chances do *FirstContact* conseguir entregar a mensagem ao destino sem que a mesma se perca no caminho, não conseguindo atingir o destino, que é algo comum para o *FirstContact* quando trabalha com um *cluster* grande de nós, como é o caso dos cenários 3, 4 e 5 como veremos adiante.

Considerando o tempo total de viagem de 3482.1 segundos do *FirstContact* como 100%, então o tempo total de viagem de 3389.6 segundos do TVFP representa 97.34% disso, reduzindo em 2.66% o tempo total de viagem.

- **Comparação TVFP X *Epidemic*:** Ambos possuem iguais tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem o mesmo número de *hops* para entregarem a mensagem ao destino. Isso se deve à política de *flooding* do *Epidemic* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP.

  O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o *Epidemic* gerou um total de 5 instâncias da mensagem, ou seja, a mensagem original mais 4 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Este número de cópias poderia seria maior se houvessem mais oportunidades de contato com nós sem a cópia da mensagem, ou seja, propiciando a ocorrência de mais re-

plicações da mensagem, conforme ocorre o processo de inundação inerente ao *Epidemic*. O número pequeno de cópias adicionais, ou seja, 4, por parte do Epidemic neste cenário é compreensível, visto que o *cluster* dos nós que utilizam o Epidemic forma uma sub-rede esparsa e menos densa de nós, fazendo que o *overhead* gerado não seja significativo, mas como veremos nos cenários 3, 4 e 5, este número de cópias será maior, pois o *cluster* utilizado nestes cenários, o 3º *cluster*, é denso em nós e conexões, contrastando com o *cluster*.

Considerando as 5 mensagens do *Epidemic* como 100%, então a única mensagem do TVFP representa 20% disso, reduzindo em 80% o *overhead* de geração de mensagens.

- **Comparação TVFP X SaW (Número máximo de cópias: 7):** Ambos possuem iguais tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem os mesmos números totais de *hops* para levarem a mensagem da origem ao destino. Isso se deve à política de *flooding*, ainda que controlado por um limitante de número máximo de cópias presentes na rede, do *SaW* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP. O valor do limitante do número máximo de cópias do SaW neste cenário assume um valor pequeno, ou seja, 7, bem como o número pequeno de cópias adicionais geradas da mensagem, ou seja, 4, e dentre estas a cópia que chegou ao destino, por parte do SaW neste cenário são compreensíveis, uma vez que o *cluster* utilizado é pequeno, assim a tendência é que o SaW não precise espalhar uma grande quantidade de cópias até encontrar o o caminho idêntico ao percorrido pela mensagem no TVFP, o que justifica o baixo valor, no caso, 7, do limitante do número de cópias da mensagem que podem existir na rede.

  O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o SaW gerou um total de 5 instâncias da mensagem, ou seja, a mensagem original mais 4 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Neste cenário notamos que não foi atingido o limite máximo de 7 cópias na rede estipulado pelo SaW, portanto replicações adicionais da mensagem seriam possíveis caso surgissem mais oportunidades de contato posteriormente.

  Considerando as 5 mensagens do SaW como 100%, então a única mensagem do TVFP representa 20% disso, reduzido em 80% o *overhead* de geração de mensagens.

5.1.3 Cenário 3 - Origem: $n153$ / Destino: $n12$

Neste cenário de simulação, o melhor caminho calculado pelo TVFP neste TVG é $n153 \rightarrow n333 \rightarrow n120 \rightarrow n261 \rightarrow n12$, sendo este o caminho percorrido pela mensagem durante a simulação. O nó origem é $n153$ e o nó destino é $n12$, e o *cluster* de nós utilizado pelos algoritmos é o **3º *cluster* de nós**.

Para o TVFP, o tempo de partida é 20314.3 segundos em relação ao início da simulação, o menor tempo de chegada possível da mensagem ao destino é 22645.7 segundos em relação ao início da simulação, tendo tempo total de viagem de 2331.4 segundos, e número total de *hops* necessários para se levar a mensagem ao destino sendo 4.

A tabela 5.4 e o gráfico da figura 5.4 mostram os resultados obtidos.

Tabela 5.4: Desempenho do TVFP ao enviar uma mensagem de $n153$ para $n12$

Origem: $n153$ / **Destino:** $n12$					
Algoritmo	**Tempo de viagem**	***Hops***	**Msg. criadas**	**Partida**	**Chegada**
TVFP	2331.4s	4	1	20314.3s	22645.7s
FirstContact	N/A	N/A	1	19135.8s	N/A
Epidemic	2331.4s	4	127	20314.3s	22645.7s
SaW (Máx.: 4048 cópias)	2331.4s	4	105	20314.3s	22645.7s

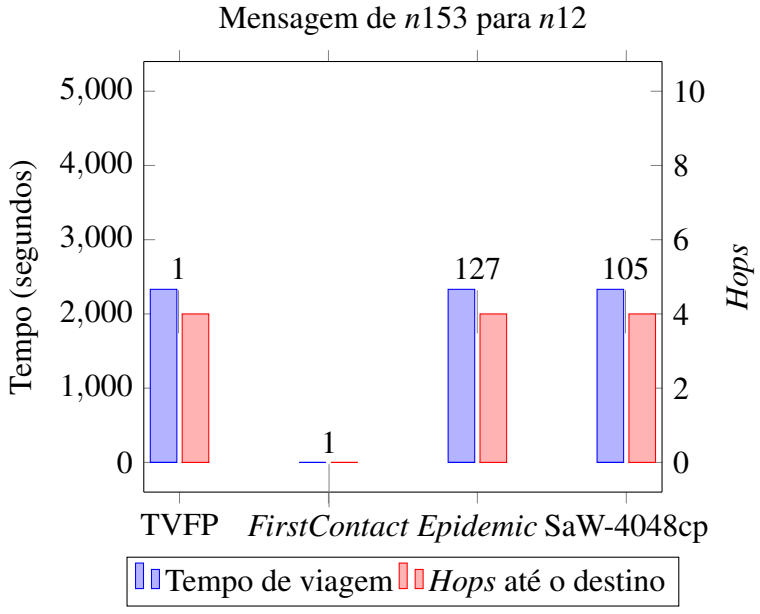


Figura 5.4: Comparando TVFP com demais algoritmos ($n153$ até $n12$). O algoritmo *FirstContact* não conseguiu entregar a mensagem ao destino $n12$.

- **Comparação TVFP X *FirstContact*:** Neste cenário o algoritmo *FirstContact* não conseguiu levar a mensagem do nó origem até o nó destino. Os tempos de partida do TVFP

e *FirstContact* diferem (TVFP: 20314.3 segundos / *FirstContact*: 19135.8 segundos), o que significa que, ainda que a mensagem, com ambos os algoritmos, parta do mesmo nó origem $n153$, o caminho tomado pelo *FirstContact* diverge do calculado pelo TVFP, pois o próximo nó para onde a mensagem é transferida é o nó $n196$, ao invés do nó 333. Assim, a mensagem, ao percorrer este caminho oriundo da escolha do *FirstContact*, perdeu-se no meio do *cluster* de nós, não conseguindo alcançar o nó destino $n12$, ainda que mais tarde.

Considerando que neste cenário o *cluster* de nós utilizado é muito complexo e denso em nós e conexões, é compreensível que o *FirstContact* possa não conseguir entregar a mensagem ao destino, uma vez que é um algoritmo *single-copy*, ou seja, trabalha com uma única instância da mensagem da rede, e por isso mesmo não pode utilizar de uma política de *flooding* como faz o algoritmo *Epidemic*, portanto deve esperar que as conexões que surgem no seu caminho sejam as conexões certas que possam levar a mensagem ao destino, porém com tantas conexões surgindo, possivelmente antes mesmo das conexões certas surgirem, ou seja, as conexões que levam aos nós do melhor caminho, são grandes as chances da mensagem divergir do caminho original, tomando um outro caminho, que pode resultar em a mensagem não conseguir chegar ao destino, ou resultar na mesma chegar ao destino num tempo posterior, ou às até mesmo chegar no mesmo tempo o TVFP, porém gastando um número maior de *hops*. Estas dificuldades em entregar a mensagem ao destino tendem a aumentar conforme aumenta a densidade de nós e conexões no *cluster*.

- **Comparação TVFP X *Epidemic*:** Ambos possuem iguais tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem o mesmo número de *hops* para entregarem a mensagem ao destino. Isso se deve à política de *flooding* do *Epidemic* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP.

  O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o *Epidemic* gerou um total de 127 instâncias da mensagem, ou seja, a mensagem original mais 1 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Este número de cópias poderia seria maior se houvessem mais oportunidades de contato com nós sem a cópia da mensagem, ou seja, propiciando a ocorrência de mais replicações da mensagem, conforme ocorre o processo de inundação inerente ao *Epidemic*. O número grande de cópias adicionais, ou seja, 126, por parte do Epidemic neste cenário é compreensível, visto que o *cluster* dos nós que utilizam o Epidemic forma uma sub-rede densa em nós e conexões, fazendo que o *overhead* gerado significativo, pois muitas outras conexões ocorreram, além das conexões envolvendo os nós do melhor

camiho.

Considerando as 127 mensagens do *Epidemic* como 100%, então a única mensagem do TVFP representa 0.79% disso, reduzindo em 99.21% o *overhead* de geração de mensagens.

- **Comparação TVFP X SaW (Número máximo de cópias: 4048):** Ambos possuem iguais tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem os mesmos números totais de *hops* para levarem a mensagem da origem ao destino. Isso se deve à política de *flooding*, ainda que controlado por um limitante de número máximo de cópias presentes na rede, do *SaW* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP. O valor do limitante do número máximo de cópias do SaW neste cenário assume um valor grande, ou seja, 4048, bem como o número grande de cópias adicionais geradas da mensagem, ou seja, 104, e dentre elas a cópia que chegou ao destino, por parte do SaW neste cenário são compreensíveis, uma vez que o *cluster* utilizado é grande, assim a tendência é que o SaW precise espalhar uma grande quantidade de cópias até encontrar o o caminho idêntico ao percorrido pela mensagem no TVFP, o que justifica o alto, no caso, 4048, do limitante do número de cópias da mensagem que podem existir na rede.

  O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o SaW gerou um total de 105 instâncias da mensagem, ou seja, a mensagem original mais 104 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Neste cenário notamos que não foi atingido o limite máximo de 4048 cópias na rede estipulado pelo SaW, portanto replicações adicionais da mensagem seriam possíveis caso surgissem mais oportunidades de contato posteriormente.

  Considerando as 105 mensagens do SaW como 100%, então a única mensagem do TVFP representa 0.95% disso, reduzido em 99.05% o *overhead* de geração de mensagens.

5.1.4 Cenário 4 - Origem: $n156$ / Destino: $n360$

Neste cenário de simulação, o melhor caminho calculado pelo TVFP neste TVG é $n156 \rightarrow n238 \rightarrow n333 \rightarrow n360$, sendo este o caminho percorrido pela mensagem durante a simulação. O nó origem é $n156$ e o nó destino é $n360$, e o *cluster* de nós utilizado pelos algoritmos é o **3º *cluster* de nós**.

Para o TVFP, o tempo de partida é 19778 segundos em relação ao início da simulação, o

menor tempo de chegada possível da mensagem ao destino é 21121. segundos em relação ao início da simulação, tendo tempo total de viagem de 1343 segundos, e número total de *hops* necessários para se levar a mensagem ao destino sendo 4.

A tabela 5.5 e o gráfico da figura 5.5 mostram os resultados obtidos.

Tabela 5.5: Desempenho do TVFP ao enviar uma mensagem de $n156$ para $n360$

Origem: $n156$ / **Destino:** $n360$					
Algoritmo	**Tempo de viagem**	***Hops***	**Msg. criadas**	**Partida**	**Chegada**
TVFP	1343s	4	1	19778s	22645.7s
FirstContact	N/A	N/A	1	19610s	N/A
Epidemic	1343s	4	150	19778s	22645.7s
SaW (Máx.: 20 cópias)	1343s	4	20	19778s	22645.7s

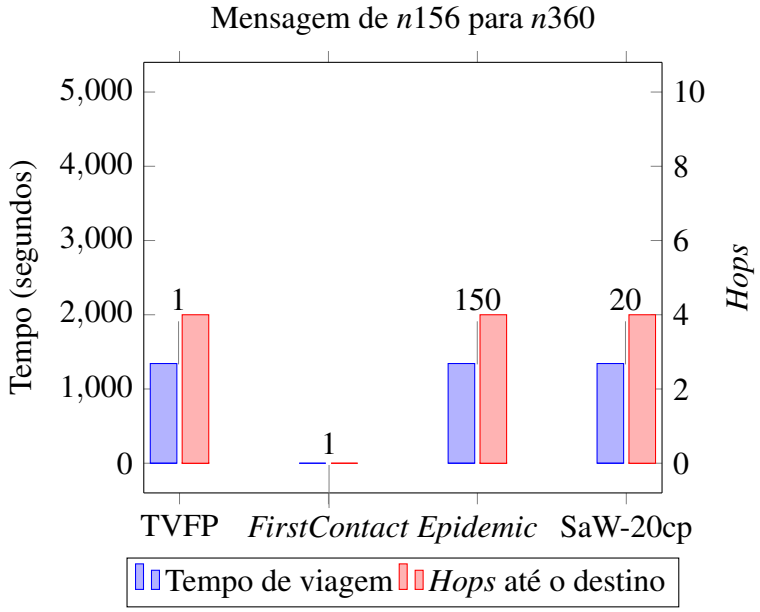


Figura 5.5: Comparando TVFP com demais algoritmos ($n156$ até $n360$). O algoritmo *FirstContact* não conseguiu entregar a mensagem ao destino $n360$.

- **Comparação TVFP X *FirstContact*:** Neste cenário o algoritmo *FirstContact* não conseguiu levar a mensagem do nó origem até o nó destino. Os tempos de partida do TVFP e *FirstContact* diferem (TVFP: 19778 segundos / *FirstContact*: 19610 segundos), o que significa que, ainda que a mensagem, com ambos os algoritmos, parta do mesmo nó origem $n156$, o caminho tomado pelo *FirstContact* diverge do calculado pelo TVFP, pois o próximo nó para onde a mensagem é transferida é o nó $n134$, ao invés do nó 238. Assim, a mensagem, ao percorrer este caminho oriundo da escolha do *FirstContact*, perdeu-se no meio do *cluster* de nós, não conseguindo alcançar o nó destino $n360$, ainda que mais tarde.

Considerando que neste cenário o *cluster* de nós utilizado é muito complexo e denso em nós e conexões, é compreensível que o *FirstContact* possa não conseguir entregar a mensagem ao destino, uma vez que é um algoritmo *single-copy*, ou seja, trabalha com uma única instância da mensagem da rede, e por isso mesmo não pode utilizar de uma política de *flooding* como faz o algoritmo *Epidemic*, portanto deve esperar que as conexões que surgem no seu caminho sejam as conexões certas que possam levar a mensagem ao destino, porém com tantas conexões surgindo, possivelmente antes mesmo das conexões certas surgirem, ou seja, as conexões que levam aos nós do melhor caminho, são grandes as chances da mensagem divergir do caminho original, tomando um outro caminho, que pode resultar em a mensagem não conseguir chegar ao destino, ou resultar na mesma chegar ao destino num tempo posterior, ou às até mesmo chegar no mesmo tempo o TVFP, porém gastando um número maior de *hops*. Estas dificuldades em entregar a mensagem ao destino tendem a aumentar conforme aumenta a densidade de nós e conexões no *cluster*.

- **Comparação TVFP X *Epidemic*:** Ambos possuem iguais tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem o mesmo número de *hops* para entregarem a mensagem ao destino. Isso se deve à política de *flooding* do *Epidemic* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP.

  O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o *Epidemic* gerou um total de 150 instâncias da mensagem, ou seja, a mensagem original mais 1 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Este número de cópias poderia seria maior se houvessem mais oportunidades de contato com nós sem a cópia da mensagem, ou seja, propiciando a ocorrência de mais replicações da mensagem, conforme ocorre o processo de inundação inerente ao *Epidemic*. O número grande de cópias adicionais, ou seja, 149, por parte do Epidemic neste cenário é compreensível, visto que o *cluster* dos nós que utilizam o Epidemic forma uma sub-rede densa em nós e conexões, fazendo que o *overhead* gerado significativo, pois muitas outras conexões ocorreram, além das conexões envolvendo os nós do melhor camiho.

  Considerando as 150 mensagens do *Epidemic* como 100%, então a única mensagem do TVFP representa 0.67% disso, reduzindo em 99.33% o *overhead* de geração de mensagens.

- **Comparação TVFP X SaW (Número máximo de cópias: 20):** Ambos possuem iguais

tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem os mesmos números totais de *hops* para levarem a mensagem da origem ao destino. Isso se deve à política de *flooding*, ainda que controlado por um limitante de número máximo de cópias presentes na rede, do *SaW* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP. O valor do limitante do número máximo de cópias do SaW neste cenário assume um valor bem menor que no cenário 3, ou seja, 20, bem como o número pequeno de cópias adicionais geradas da mensagem, ou seja, 19, e dentre elas a cópia que chegou ao destino, por parte do SaW neste cenário são compreensíveis, pois, ainda que o *cluster* utilizado neste cenário seja grande, ocorreu que os nós que compõe o melhor caminho estão relativamente próximos à origem, assim, mesmo o SaW não precisou espalhar muitas mensagens, ainda que esteja neste *cluster*, para localizar o caminho percorrido pelo TVFP.

O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o SaW gerou um total de 20 instâncias da mensagem, ou seja, a mensagem original mais 19 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Neste cenário notamos que não foi atingido o limite máximo de 20 cópias na rede estipulado pelo SaW, portanto 1 replicação adicional da mensagem seria possível caso surgissem mais oportunidades de contato posteriormente.

Considerando as 20 mensagens do SaW como 100%, então a única mensagem do TVFP representa 5% disso, reduzido em 95% o *overhead* de geração de mensagens.

5.1.5 Cenário 5 - Origem: $n128$ / Destino: $n607$

Neste cenário de simulação, o melhor caminho calculado pelo TVFP neste TVG é $n128 \rightarrow n176 \rightarrow n211 \rightarrow n607$, sendo este o caminho percorrido pela mensagem durante a simulação. O nó origem é $n128$ e o nó destino é $n607$, e o *cluster* de nós utilizado pelos algoritmos é o **3º *cluster* de nós**.

Para o TVFP, o tempo de partida é 18641.8 segundos em relação ao início da simulação, o menor tempo de chegada possível da mensagem ao destino é 22954.4. segundos em relação ao início da simulação, tendo tempo total de viagem de 4312.6 segundos, e número total de *hops* necessários para se levar a mensagem ao destino sendo 4.

A tabela 5.6 e o gráfico da figura 5.6 mostram os resultados obtidos.

- **Comparação TVFP X *FirstContact*:** Neste cenário o *FirstContact* não conseguiu levar a

Tabela 5.6: Desempenho do TVFP ao enviar uma mensagem de $n128$ para $n607$

Origem: $n128$ / **Destino:** $n607$					
Algoritmo	**Tempo de viagem**	***Hops***	**Msg. criadas**	**Partida**	**Chegada**
TVFP	4312.6s	4	1	18641.8s	22954.4s
FirstContact	N/A	N/A	1	19610s	N/A
Epidemic	4312.6s	4	76	18641.8s	22954.4s
SaW (Máx.: 11 cópias)	4312.6s	4	11	18641.8s	22954.4s

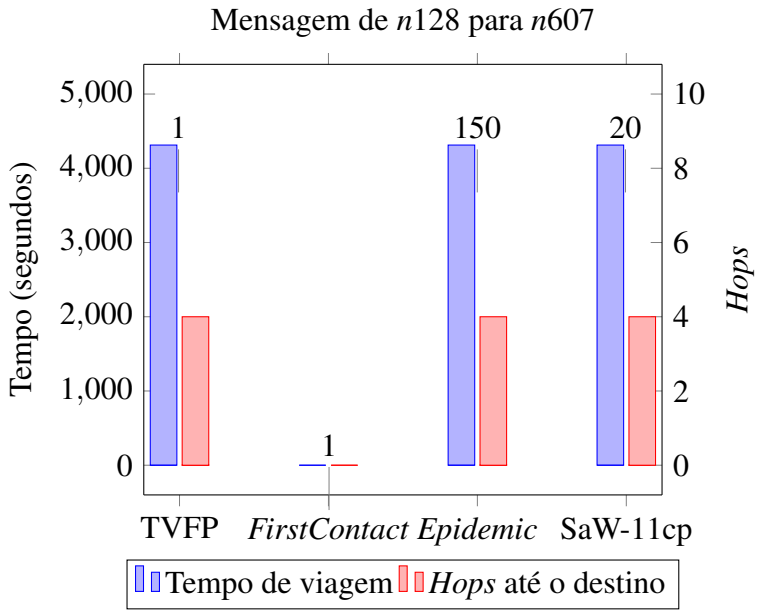


Figura 5.6: Comparando TVFP com demais algoritmos ($n128$ até $n607$). O algoritmo *FirstContact* não conseguiu entregar a mensagem ao destino $n607$.

mensagem do nó origem até o nó destino. Os tempos de partida do TVFP e *FirstContact* são iguais, mas ainda que a mensagem parta do mesmo nó origem $n128$ no caso dos dois algoritmos, o caminho tomado pelo *FirstContact* diverge do calculado pelo TVFP, pois o próximo nó para onde a mensagem é transferida é o nó $n329$, ao invés do nó 176. Assim, a mensagem, ao percorrer este caminho oriundo da escolha do *FirstContact*, perdeu-se no meio do *cluster* de nós, não conseguindo alcançar o nó destino $n607$, ainda que mais tarde.

Considerando que neste cenário o *cluster* de nós utilizado é muito complexo e denso em nós e conexões, é compreensível que o *FirstContact* possa não conseguir entregar a mensagem ao destino, uma vez que é um algoritmo *single-copy*, ou seja, trabalha com uma única instância da mensagem da rede, e por isso mesmo não pode utilizar de uma política de *flooding* como faz o algoritmo *Epidemic*, portanto deve esperar que as conexões que surgem no seu caminho sejam as conexões certas que possam levar a mensagem ao destino, porém com tantas conexões surgindo, possivelmente antes mesmo das conexões

certas surgirem, ou seja, as conexões que levam aos nós do melhor caminho, são grandes as chances da mensagem divergir do caminho original, tomando um outro caminho, que pode resultar em a mensagem não conseguir chegar ao destino, ou resultar na mesma chegar ao destino num tempo posterior, ou às até mesmo chegar no mesmo tempo o TVFP, porém gastando um número maior de *hops*. Estas dificuldades em entregar a mensagem ao destino tendem a aumentar conforme aumenta a densidade de nós e conexões no *cluster*.

- **Comparação TVFP X *Epidemic*:** Ambos possuem iguais tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem o mesmo número de *hops* para entregarem a mensagem ao destino. Isso se deve à política de *flooding* do *Epidemic* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP.

  O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o *Epidemic* gerou um total de 76 instâncias da mensagem, ou seja, a mensagem original mais 1 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Este número de cópias poderia seria maior se houvessem mais oportunidades de contato com nós sem a cópia da mensagem, ou seja, propiciando a ocorrência de mais replicações da mensagem, conforme ocorre o processo de inundação inerente ao *Epidemic*. O número grande de cópias adicionais, ou seja, 75, por parte do Epidemic neste cenário é compreensível, visto que o *cluster* dos nós que utilizam o Epidemic forma uma sub-rede densa em nós e conexões, fazendo que o *overhead* gerado significativo, pois muitas outras conexões ocorreram, além das conexões envolvendo os nós do melhor camiho. Um detalhe importante é que neste caso o número de cópias adicionais, 76, é menor que o total gerado nos cenários 3 e 4, que ocorrem no mesmo *cluster* de nós, e esta diferença no número de replicações se dá pelo fato que com a mensagem tendo o nó 128 surgiram menores oportunidades de contato. Assim, podemos concluir que mesmo que o *cluster* seja denso em nós e conexões, dependendo dos nós envolvidos nas transmissões, não necessariamente surgirão muitas oportunidades de contato.

  Considerando as 76 mensagens do *Epidemic* como 100%, então a única mensagem do TVFP representa 1.36% disso, reduzindo em 98.64% o *overhead* de geração de mensagens.

- **Comparação TVFP X SaW (Número máximo de cópias: 11):** Ambos possuem iguais tempos de partida e chegada, e portanto, iguais tempos totais de viagem, além de gastarem os mesmos números totais de *hops* para levarem a mensagem da origem ao destino. Isso

se deve à política de *flooding*, ainda que controlado por um limitante de número máximo de cópias presentes na rede, do *SaW* que permitiu a mensagem encontrar e percorrer, em meio as conexões que ocorrem entre nós conforme surgem oportunidades de contato, o mesmo caminho percorrido pelo TVFP. O valor do limitante do número máximo de cópias do SaW neste cenário assume um valor bem menor que no cenário 3, ou seja, 11, bem como o número pequeno de cópias adicionais geradas da mensagem, ou seja, 10, e dentre elas a cópia que chegou ao destino, por parte do SaW neste cenário são compreensíveis, pois, ainda que o *cluster* utilizado neste cenário seja grande, ocorreu que os nós que compõe o melhor caminho estão relativamente próximos à origem, assim, mesmo o SaW não precisou espalhar muitas mensagens, ainda que esteja neste *cluster*, para localizar o caminho percorrido pelo TVFP.

O diferencial do TVFP é que com uma única instância da mensagem pode-se entregá-la ao destino, já que o SaW gerou um total de 11 instâncias da mensagem, ou seja, a mensagem original mais 10 outras cópias da mesma, e dentre estas cópias a que chegou ao destino. Neste cenário notamos que não foi atingido o limite máximo de 11 cópias na rede estipulado pelo SaW, portanto 1 replicação adicional da mensagem seria possível caso surgissem mais oportunidades de contato posteriormente.

Considerando as 11 mensagens do SaW como 100%, então a única mensagem do TVFP representa 9.09% disso, reduzido em 90.91% o *overhead* de geração de mensagens.

5.1.6 Observações acerca da análise de desempenho

Analisando os 5 cenários temos que, em média, o tempo de chegada da mensagem ao destino, utilizando o TVFP, é de 19682.6 segundos (considerados desde o início da simulação, no tempo 0 segundos). Além disso, em média, o tempo total de viagem da mensagem da origem ao destino, utilizando o TVFP, é de 2906.88 segundos (desconsiderando os tempos de partidas nos diferentes cenários). E em média, o número de *hops* tomados para se levar a mensagem ao destino pelo TVFP é de 2.83. Vemos com estes valores que, em geral, o TVFP gasta um número relativamente pequeno de *hops* para levar a mensagem ao destino e que as viagens para levar a mensagem da origem ao destino, em geral, não demandam muito tempo.

Tanto o TVFP como o *FirstContact* causam baixo *overhead* de geração de mensagens, mas o destaque do TVFP se dá por seu mecanismo de escolha do próximo nó de repasse, o qual é relativamente aleatório no *FirstContact* dependendo da mobilidade, fazendo com que o *FirstContact* transmita a mensagem na primeira oportunidade de contato que tiver, fazendo com que siga por um caminho possivelmente maior, mais demorado até o destino, e até mesmo

inalcançável.

Os algoritmos *Epidemic* e TVFP garantem a entrega da mensagem ao destino nos menores tempos de chegada possíveis. Isso é esperado de algoritmos que usam de inundação. O destaque do TVFP se dá pelo baixo *overhead* de geração mensagens reduzido a uma única instância da mensagem, o que representa uma melhoria em relação ao *Epidemic* pois no mesmo há replicação de mensagens, e mesmo que o nó origem e destino fossem vizinhos separados por um *hop* de distância, necessariamente existiriam no mínimo 2 instâncias da mensagem na rede, uma da origem, e a cópia enviada ao destino.

De maneira análoga, o algoritmo TVFP tambem apresenta melhorias no *overhead* de geração de mensagens em relação ao SaW, pois este tambem replica mensagens, tendo como diferença um mecanismo que limita o número máximo de cópias na rede, o que representa uma melhoria em relação ao comportamento de inundação do *Epidemic*. A exceção é se o limite de cópias na rede produzidas pelo SaW for 1, fazendo a entrega ocorrer por *direct delivery*, no caso fazendo uma cópia da mensagem e entregando-a somente ao nó destino, se os nós estão diretamente conectados. Assim, no entanto, há o risco da mensagem nunca chegar ao destino, uma vez que mensagens entregue por *direct delivery* são entregues apenas ao nó destino. Consequentemente, o nó origem teria de estar conectado com o nó destino, e uma das possíveis situações ocorreria: ou se conecta rapidamente ao nó destino, se forem vizinhos, ou se conecta ao nó destino posteriormente, atrasa a entrega, ou nunca se conecta.

Foi possível observar como a complexidade da rede, em termos da densidade de nós e conexões, pode influenciar na quantidade de mensagens que precisam ser disseminadas ao longo da rede, para se atingir o nó destino, assim, em uma rede de alta densidade de nós e conexões, a tendência é que ocorram muitas conexões, e portanto, muitas replicações da mensagem, e que possivelmente caminhos até o destino sejam mais longos. Além disso vimos que isso não necessariamente a alta densidade de nós e conexões necessariamente implica em mais replicações, pois como vimos nos cenários 4 e 5, não ocorreram tantas replicações como no cenário 3, mostrando que numa mesma rede densa, um ou outro nó ainda podem se conectar mais rapidamente ao nó destino.

Do ponto de vista da heurística do TVFP para decisão do melhor caminho tendo conhecimento prévio da topologia, pode parecer injusta com os algoritmos *Epidemic* e *Spray-and-Wait*, visto que ambos não possuem o conhecimento prévio da topologia, portanto descobrindo os nós da mesma conforme disseminam a mensagem por *flooding*. Assim, dependendo da densidade de nós na topologia, o *overhead* de geração de mensagens pode ser maior ou menor, mas sempre tendo a certeza que não é necessário este cálculo do caminho em um TVG, e consequentemente,

uma mineração de dados na rede para extrair informação útil. O TVFP por sua vez, com a utilização do TVG, consegue extrair o melhor caminho, quando existe, de um nó origem ao nó destino do TVG que representa a topologia, ao custo de uma única instância da mensagem, sendo que este caminho é o mesmo melhor caminho que eventualmente os algoritmos *Epidemic* e SaW eventualmente descobrirão conforme disseminam as mensagens na rede por *flooding*.

Os algoritmos *Epidemic* e *Spray-and-Wait* se beneficiar da heurística do TVFP, em utilizar TVG para ter conhecimento prévio da topologia, pois com a mesma teriam de o conhecimento do caminho exato a ser percorrido pela mensagem entre origem e destino, e desta maneira evitaria-se que replicações desnecessárias da mensagem fossem feitas. Dizemos replicações desnecessários pois por padrão, os algoritmos *Epidemic* e *Spray-and-Wait* trabalham com replicações da mensagem. Assim, notamos que só seriam feitas cópias adicionais suficientes para que a mensagem possa ser entregue ao destino, visto que os nós não transmitiriam cópias da mensagem para qualquer nó com os quais se conectassem, apenas para os nós pertencentes ao melhor caminho, mas ainda assim o *overhead* de geração de mensagens seria maior que o do TVFP, pois o mesmo reduz a uma única mensagem este *overhead*, e se a rede for extremamente densa em nós, mesmo o *Epidemic* e *Spray-and-Wait* tendo conhecimento prévio da rede, o melhor caminho gerado ainda assim pode ser longo, e que pode resultar em um *overhead* de geração de mensagens considerável, ainda que não seja o pior caso possível deste *overehead*, que seria o fato das replicações ocorrerem entre vários nós do TVG, além daqueles pertencentes ao melhor caminho.

5.2 Análise de precisão

Os resultados anteriores utilizam um único *trace* que gera o TVG para obter as medidas de desempenho. Entretanto, na vida real, os ônibus, embora tenham um calendário bem definido podem, no entanto, sofrer pequenas variações de trajeto e tempo. Com base nisso, nós criamos os chamados *traces* sintéticos.

Neste experimento utilizamos o *trace* de 02/12/2001 e criamos um outro *trace* sintético gerado a partir deste. De cada um dos*traces* geramos um TVG, e destes dois geramos um terceiro TVG, o qual chamamos de **TVG intermediário** ou **TVG médio**.

O *trace* sintético é equivalente ao original, tendo exatamente os mesmos ônibus que o original. No entanto, no *trace* sintético, cada ônibus assume posições suavemente diferentes das assumidas pelo mesmo ônibus no original, mas os tempos das posições em ambos os *traces* permanecem os mesmos. Por exemplo, suponha que no *trace* original, o ônibus 1 ocupe a posiçao

(0,0) no tempo 0, no *trace* sintético, este mesmo ônibus 1 ocupa a posição (10,10) no tempo 0.

Geramos este *trace* sintético para simular o itinerário da mesma frota de ônibus do *trace* 02/12/2001, porém em um dia diferente, uma vez que em situações realísticas de trânsito espera-se que ônibus da mesma frota sigam um mesmo itinerário mesmo em dias diferentes. Note que certas variações podem ocorrer, por exemplo, um ônibus tomar uma **rota alternativa** em virtude da ocorrência de acidentes de trânsito em sua rota, bem como mover-se em velocidades diferentes, o que resultaria em intervalos de conectividade levemente diferentes no TVG, ou mesmo estar ausente no dia para manutenção, resultando em nós ausentes no TVG. Por fim, não alteramos as rotas que os ônibus utilizam, mantendo-se inalteradas quais as rotas originais tomadas pelos ônibus no *trace* original, ou seja, em ambos os *traces* os ônibus continuam seguindo as mesmas rotas, só mudam-se as posições ocupadas. Do ponto de vista do simulador ONE, as rotas tomadas pelos ônibus são irrelevantes, visto que o para o simulador, só interessam as informações contidas nos *traces* referentes às posições X e Y, bem como os tempos que foram ocupadas tais posições, de cada nó.

Nos *traces* do CRAWDAD há registros da movimentação dos ônibus desde o dia 30/10/2001 até o dia 02/12/2001. Analisando *traces* de duas datas diferentes, bem como seus TVGs resultantes, nota-se que não há correlação entre os nós de cada TVG, pois alguns nós de um TVG e suas respectivas conexões estão ausentes no outro TVG, sugerindo que os ônibus não pertencem à mesma frota, tampouco seguem os mesmos itinerários. Tais fatores nos impossibilitaram de trabalhar com *traces* de datas diferentes, justificando o uso de um *trace* sintético, para simular ônibus da mesma frota em dias diferentes.

Para a criação do TVG intermediário, algumas considerações tiveram de ser tomadas:

- Incluir apenas pares de nós existentes em ambos os TVGs para o cálculo da média aritmética dos tempos nos intervalos.
- A média aritmética entre intervalos calculados entre pares de nós equivalentes é feita obtendo-se a média aritmética entre os limitantes inferiores e a média aritmética dos limitantes superiores dos intervalos equivalentes dos pares de nós em cada TVG, original e sintético, resultando em um intervalo médio para este mesmo par de nós no TVG médio.
- Considerar o menor número de intervalos de conectividade entre pares de nós comuns aos dois TVGs. Por exemplo, se em um TVG, um par de nós possui 3 intervalos de conectividade, e no outro TVG, o mesmo par de nós possui apenas 2 pares, então, no TVG médio resultante de um cálculo feito com os dois TVGs (original e sintético) este mesmo par de nós só possuirá 2 intervalos de conectividade, resultantes da média aritmética entre

os 2 primeiros intervalos de conectividade do par de nós de ambos os TVGs.

Para compreender melhor a geração do TVG médio, ou intermediário, onsidere este exemplo numérico: seja o par de nós (A,B), presente nos dois TVGs, mas em um TVG possui 2 intervalos, $[1,2] \cup [7,10]$, e no outro apenas um, $[5,6]$, assim é feita aritmética entre $[1,2]$ e $[5,6]$, ou seja, $[(1+5) \div 2, (2+6) \div 2]$, que é igual a $[3,4]$, desconsiderado-se o intervalo $[7,10]$.

Tomamos 20 ônibus do *trace* de 02/12/2001 e do seu equivalente sintético, e analisamos o quanto diferem as posições de um ônibus no *trace* sintético quando comparadas às posições do mesmo ônibus no *trace* original, sendo que para cada posição de um ônibus no *trace* original e sua equivalente posição no *trace* sintético, obtemos a **distância**, a qual chamamos de **erro**, entre estas posições ocupadas pelos ônibus.

Calculando a diferença entre todas as posições ocupadas por um mesmo ônibus nos dois *traces*, obtemos o **erro médio**, que diz o quão distantes uma da outra estão, em geral, cada posição dos ônibus nos dois *traces*, e portanto, o quão eficaz seria o TVFP, utilizando o TVG intermediário e um dos *traces*, original ou sintético, em calcular os melhores caminhos. Uma vez que há um erro médio, entendemos que há uma porcentagem média de acertos por parte do TVFP, que como mostra nossos resultados é alta, no cálculo dos melhores caminhos.

Assim, verificaríamos se o caminho calculado no TVG intermediário seria condizente com a situação real de conectividade entre nós da rede, ou até mesmo, se caminhos antes existentes nos TVGs originais de um dos *traces*, continuam existindo, quando calculados através do TVG intermediário. Sendo assim, é esperado que o TVG intermediário não corresponda inteiramente aos dados do *trace* com o qual é utilizado em conjunto, e consequentemente, aos eventos de conexão durante a simulação. A tabela 5.7 mostra os erros entre as posições de 20 ônibus nos dois *traces*.

Em outras palavras, em um cenário onde são gerados TVGs para cada dia de movimentação de uma mesma frota de ônibus, com um TVG intermediário gerado a partir de todos estes TVGs, verificaria-se o quão preciso o TVFP seria em calcular o melhor caminho, utilizando este TVG intermediário em conjunto com o *trace* de qualquer um destes dias.

A tabela 5.7 mostra os valores das diferenças de distância analisadas de 20 ônibus do trace de 02/12/2001 e o equivalente sintético. Podemos perceber que a porcentagem de erros médios para cada ônibus é muito baixa, ou seja, em média, não há muita diferença de distância entre as posições, o que justifica a alta porcentagem de acertos para cada ônibus, se aplicássemos o TVFP em um dos dois *traces* utilizando-se do TVG intermediário.

Além disso, sumarizamos os dados da tabela 5.7 no gráfico da figura 5.7, a qual mostra os

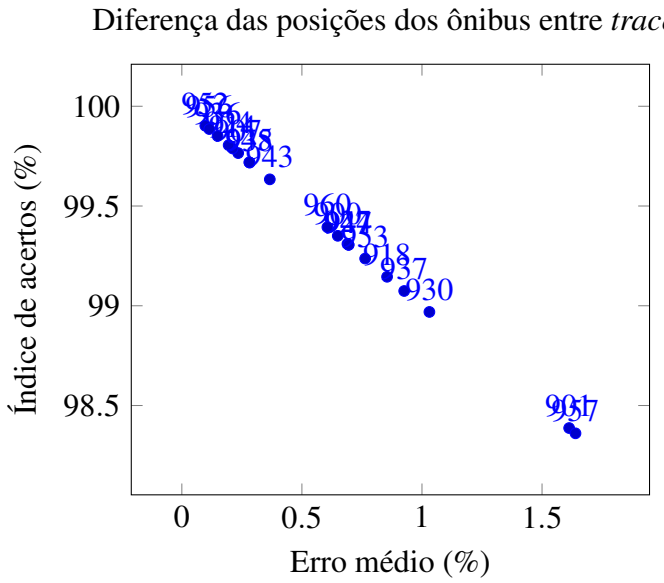


Figura 5.7: Diferença das posições dos ônibus entre *traces*

erros médios entre as posições dos 20 ônibus nos dois *traces*, onde o eixo X representa os erros médios, e o eixo Y os acertos.

Por fim, temos 5 gráficos, que utilizam os dados da tabela 5.7, cada um representando um par de ônibus, para observar visualmente essa diferença de distância entre os posições. Cada gráfico é organizado de modo que os eixos das abcissas e das ordenadas representam, respectivamente, as coordenadas X e coordenadas Y das posições dos ônibus, de modo que os pontos formados pelas intersecções dos valores destes eixos nos dão as posições ocupadas pelos ônibus. A disposição dos pontos nos gráficos não deve ser confundida com um gráfico temporal, onde pelo menos um dos eixo representaria o tempo decorrido, e, portanto, os pontos nos gráficos formados pelas intersecções das coordenadas X e Y, não seguem nenhuma ordem temporal específica. A ordem que respeitam é a ordem imposta pelos eixos das coordenadas, onde há uma ordem crescente de valor de coordenadas e cada um.

5.2.1 Observações acerca da análise de precisão

Considerando todos ônibus do *traces* original e sintético, nota-se que a diferença de distâncias dos 20 ônibus, em média é de 287.35 metros, representando 0.6% do total de médias, sendo diferenças pequenas de distância entre as posições de ônibus, portanto, tem-se uma média de acertos de 99.4%, implicando que o TVFP é preciso no cálculo de melhores caminhos mesmo quando utilizando o TVG médio com algum dos dois *traces*.

Tabela 5.7: Distância entre posições dos ônibus em dois traces

ID do ônibus	Erro (Distância) médio)	% média de erros	% acertos
900	287.31	0.65%	99.35%
901	306.77	1.61%	98.39%
907	287.7	0.23%	99.77%
914	280.4	0.21%	99.79%
918	284.89	0.85%	99.15%
924	279.58	0.19%	99.81%
927	287.46	0.69%	99.31%
930	282.97	1.03%	98.97%
936	282.17	0.15%	99.85%
937	279.58	0.93%	99.07%
938	283.55	0.28%	99.72%
943	288.20	0.37%	99.63%
944	284.4	0.69%	99.31%
945	291.74	0.28%	99.72%
947	297.19	0.69%	99.31%
952	287.20	0.1%	99.90%
953	297.36	0.76%	99.24%
956	283.85	0.11%	99.89%
957	279.97	1.64%	98.36%
960	294.66	0.60%	99.4%

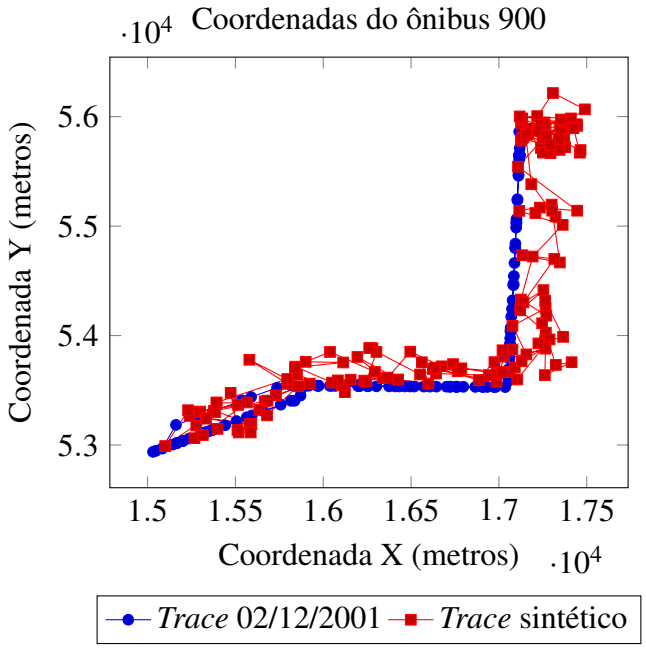


Figura 5.8: Comparando coordenadas ônibus de ID 900 de dois *traces*

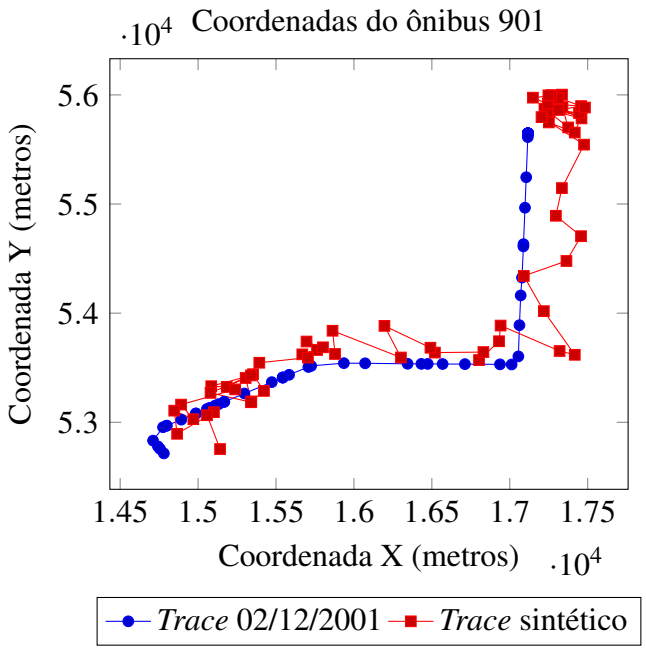


Figura 5.9: Comparando coordenadas ônibus de ID 901 de dois *traces*

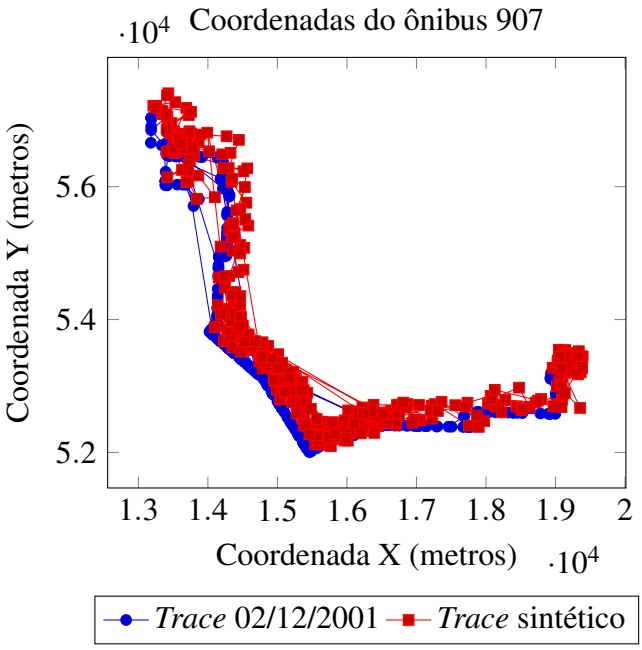


Figura 5.10: Comparando coordenadas ônibus de ID 907 de dois *traces*

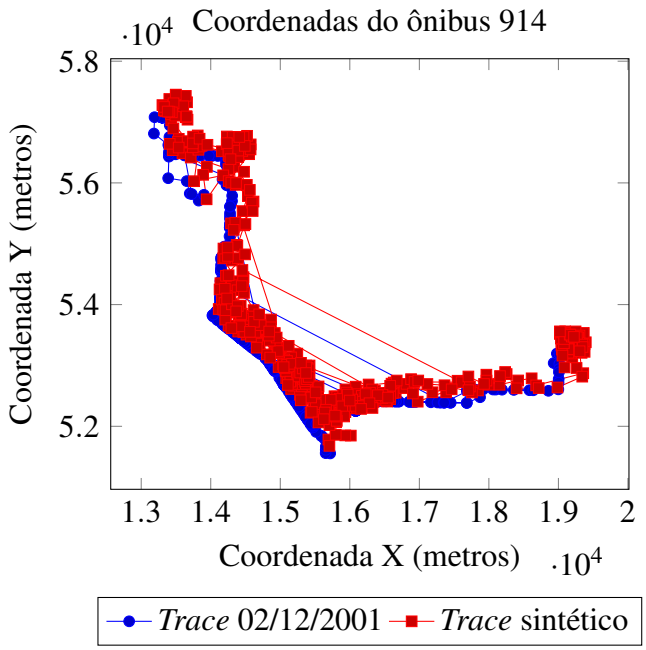


Figura 5.11: Comparando coordenadas ônibus de ID 914 de dois *traces*

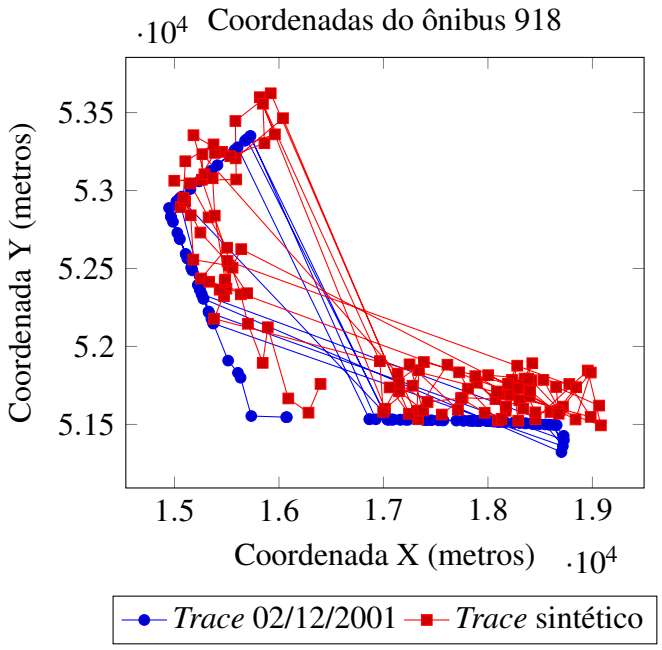


Figura 5.12: Comparando coordenadas ônibus de ID 918 de dois *traces*

Capítulo 6

CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho desenvolvemos um algoritmo de roteamento *single-copy* em redes DTN, *Time-Varying Fastest Path* (TVFP), que a partir de um TVG gerado de dados minerados de uma simulação no ONE em um cenário de natureza determinística, calcula o melhor caminho para entregar a mensagem ao nó destino no menor tempo possível, tendo a mesma partida de algum dado nó origem, sendo que os nós origem e destino mencionados são oriundos do TVG, e consequentemente, da rede modelada por este TVG.

Nossos resultados mostram que o TVFP se destaca em relação aos algoritmos *Epidemic* e *Spray-and-Wait* devido a seu baixo *overhead* de geração de mensagens, reduzido a uma única instância da mensagem na rede, e, graças a seu mecanismo de escolha de nós para onde enviar a mensagem, ou seja, o próximo nó de repasse, faz com que a mensagem sempre percorra o melhor caminho, quando existe, que garanta que seja entrege ao destino, e no menor tempo possível. A princípio parece injusto comparar o TVFP com os demais algoritmos *Epidemic* e *SaW*, visto que o TVFP possui conhecimento prévio dos eventos de conexão que ocorrerão na rede, enquanto que os algoritmos não, no entanto, o TVFP só funciona se conseguir obter os dados da simulação para assim gerar este conhecimento dos eventos de conexão sob a forma de um TVG, do contrário não funcionará, ou seja, o TVG precisá de uma fase inicial para coletar os dados e depende da rede seguir os padrões descritos no TVG, já os demais algoritmos são independem deste recurso externo, neste o caso o TVG, para funcionar, pois com a política de *flooding* que utilizam, conseguem descobrir o restante da topologia, ainda que possam provocar um certo *overhead* de geração de mensagens durante este processo de descoberta.

O TVFP tambem se destaca com relação ao algoritmo *FirstContact* pois, ainda que ambos causem o mesmo *overhead* de geração de mensagens, o TVFP garante que a mensagem será entregue e no menor tempo possível, passando pelo melhor caminho, quando este existe, calculado a partir do TVG que modela a rede da simulação, pois como vimos, quando a rede possui

alta densidade de nós e conexões, existe uma grande chance da única mensagem transmitida pelo *FirstContact* não conseguir chegar ao destino. Além disso, em redes de pequena densidade de nós, o *FirstContact* possui maiores chances de levar a mensagem ao destino, mas sem ter garantia de que percorrerá o melhor caminho até este destino. É notável como a heurística de escolha do caminhoa ser seguido, tendo os nós conhecimento prévio da rede e do caminho, pode tambem ser útil e benéfica a algoritmos para redes DTN conhecidos, como o *Epidemic* e *Spray-and-Wait*, visto que fazendo uso desta heurístic do TVFP, eles poderiam reduzir seu *overhead* de geração de mensagens, ainda que não possam reduzir totalmente tal *overhead* a uma única instância da mensagem, tal qual ocorre com o TVFP e o *FirstContact*. Além disso, vimos tambem, que o TVFP, é preciso no cálculo do melhor caminho utilizando um TVG intermediário aplicado a diversas variações da mesma rede.

Como trabalhos futuros pretendemos ampliar a implementação do TVFP de modo que calcule o melhor caminho dinamicamente durante uma simulação, pois isto melhor representaria situações reais de trânsito, e consequentemente da rede, que nunca se manteria igual aos dados e tempos previstos pelo TVG, uma vez que imprevistos podem acontecer, podendo inutilizar o melhor caminho previamente calculado, o qual necessita que a rede siga exatamente o comportamento descrito no TVG, o que não acontece em situações realísticas.

Adicionalmente, pretendemos adicionar suporte a roteamento de múltiplas mensagens e múltiplos melhores caminhos ao TVFP, uma vez que em situações realísticas da rede, os dados transmitidos em uma rede DTN possuem pacotes de tamanhos e quantidades diferentes, e consequentemente, os pacotes seriam transmitidos em tempos diferentes, portanto, percorrendo melhores caminhos possivelmente diferentes, além de um melhor caminho previamente calculado ter de ser recalculado.

Uma outra possibilidade para melhoria seria incluir suporte a cálculo de melhores caminhos em modalidades variadas do nosso algoritmo TVFP, no caso, uma versão ***shortest-path*** do algoritmo TVFP, onde seria retornado o caminho entre um par de nós origem e destino que gaste o maior número de *hops* possível, e com o menor tempo de chegada possível ao destino, tendo esta restrição do número de *hops* ser necessariamente o menor possível, e uma versão ***longest-path*** do algoritmo TVFP, onde seria retornado o caminho entre um par de nós origem e destino que gaste o maior número de *hops* possível, e com o menor tempo de chegada possível ao destino, tendo esta restrição do número de *hops* ser necessariamente o maior possível. Dizemos isso pois, da maneira que implementamos o TVFP, ele retorna o caminho que garanta a entrega da mensagem ao destino no menor tempo possível sem restrição de número *hops*, e com estas adições, poderíamos ter caminhos de *hops* diferentes. É notável que o tempo retornado por

estes caminhos diferentes não necessariamente será o menor possível, se comparado ao cálculo feito pelo TVFP original, mas ao menos, o tempo de chegada será o menor possível dentro de um escopo do onde é estabelecida uma restrição número de *hops*.

REFERÊNCIAS

BASAGNI, S. et al. *Mobile Ad Hoc Networking: Cutting Edge Directions*. 2nd ed. [S.l.]: Wiley-IEEE Press, 2013. ISBN 978-1-118-08728-2.

BISWAS S.S., B. A.; DOJA, M. A refinement of dijkstra's algorithm for extraction of shortest paths in generalized real time-multigraphs. *Journal of Computer Science*, Science Publications, p. 377–382, 2013. Disponível em: <http://thescipub.com/pdf/10.3844/jcssp.2013.377.382>.

BUI-XUAN, B.-M.; FERREIRA, A.; JARRY, A. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, v. 14, n. 2, p. 267–285, 2003.

CASTEIGTS, A. et al. Time-varying graphs and dynamic networks. In: *Proceedings of the 10th International Conference on Ad-hoc, Mobile, and Wireless Networks*. Berlin, Heidelberg: Springer-Verlag, 2011. (ADHOC-NOW'11), p. 346–359. ISBN 978-3-642-22449-2. Disponível em: <http://dl.acm.org/citation.cfm?id=2032462.2032496>.

DING, B.; YU, J. X.; QIN, L. Finding time-dependent shortest paths over large graphs. In: *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*. New York, NY, USA: ACM, 2008. (EDBT '08), p. 205–216. ISBN 978-1-59593-926-5. Disponível em: <http://doi.acm.org/10.1145/1353343.1353371>.

FERREIRA, A. On models and algorithms for dynamic communication networks: The case for evolving graphs. In: *4e rencontres francophones sur les Aspects Algorithmiques des Télécommunications ALGOTEL'2002*. [S.l.]: INRIA Press, 2002. p. 155–161.

FERREIRA, A. Building a reference combinatorial model for manets. *Netwrk. Mag. of Global Internetwkg.*, IEEE Press, Piscataway, NJ, USA, v. 18, n. 5, p. 24–29, Setembro 2004. ISSN 0890-8044. Disponível em: <http://dx.doi.org/10.1109/MNET.2004.1337732>.

FERREIRA, A.; GALTIER, J.; PENNA, P. Handbook of wireless networks and mobile computing. In: . New York, NY, USA: John Wiley & Sons, Inc., 2002. cap. Topological Design, Routing, and Handover in Satellite Networks, p. 473–493. ISBN 0-471-41902-8.

GOLDMAN, A.; FLORIANO, P.; FERREIRA, A. A tool for obtaining information on DTN traces. In: *4th Extreme Conference on Communication (ExtremeCom 2012)*. Zurich, Switzerland: [s.n.], 2012. p. 6. Disponível em: <https://hal.inria.fr/hal-00742993>.

HOSSMANN, T.; LEGENDRE, F.; SPYROPOULOS, T. From contacts to graphs: Pitfalls in using complex network analysis for dtn routing. In: *INFOCOM Workshops 2009, IEEE*. [S.l.: s.n.], 2009. p. 1–6.

JETCHEVA, J. et al. Design and evaluation of a metropolitan area multitier wireless ad hoc network architecture. In: *Mobile Computing Systems and Applications, 2003. Proceedings. Fifth IEEE Workshop on*. [S.l.: s.n.], 2003. p. 32–43.

KEMPE, D.; KLEINBERG, J.; KUMAR, A. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, Academic Press, Inc., Orlando, FL, USA, v. 64, n. 4, p. 820–842, Junho 2002. ISSN 0022-0000. Disponível em: <http://dx.doi.org/10.1006/jcss.2002.1829>.

KERäNEN, A.; OTT, J.; KäRKKäINEN, T. The ONE Simulator for DTN Protocol Evaluation. In: *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. New York, NY, USA: ICST, 2009. ISBN 978-963-9799-45-5.

PALLIS, G. et al. On the structure and evolution of vehicular networks. In: *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*. [S.l.: s.n.], 2009. p. 1–10. ISSN 1526-7539.

PORTUGAL-POMA, L. P. *Aprimorando o desempenho de algoritmos de roteamento em VANETs utilizando classificação*. São Carlos, SP, Brasil: [s.n.], Julho 2013.

PORTUGAL-POMA, L. P. et al. Applying machine learning to reduce overhead in dtn vehicular networks. In: *Computer Networks and Distributed Systems (SBRC), 2014 Brazilian Symposium on*. [S.l.: s.n.], 2014. p. 94–102.

SPYROPOULOS, T.; PSOUNIS, K.; RAGHAVENDRA, C. S. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. In: *Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-tolerant Networking*. New York, NY, USA: ACM, 2005. (WDTN '05), p. 252–259. ISBN 1-59593-026-4. Disponível em: <http://doi.acm.org/10.1145/1080139.1080143>.

VAHDAT, A.; BECKER, D. *Epidemic Routing for Partially-Connected Ad Hoc Networks*. [S.l.], 2000.

VENDRAMIN, A. C. K. et al. Grant: Inferring best forwarders from complex networks' dynamics through a greedy ant colony optimization. *Computer Networks*, v. 56, p. 997–1015, 2012.

GLOSSÁRIO

DTN – *Delay and Disruption Tolerant Network*

IDE – *Integrated Development Environment*

MANET – *Mobile Ad-hoc Network*

ONE – *One Network Environment*

SaW – *Spray and Wait*

TVFP – *Time-Varying Fastest Path*

TVG – *Time-Varying Graph*

VANET – *Vehicular Ad-hoc Network*

Anexo A

DETALHES TÉCNICOS SOBRE O TRATAMENTO DE TRACES

Neste anexo descrevemos todo o procedimento técnico envolvendo os *traces* oriundos do repositório CRAWDAD, e o formato utilizado pelos mesmos, bem como o formato que devem assumir, enquanto na forma de *traceFile* e *activeFile* para serem utilizados no simulador ONE.

A.1 Formato dos *traces* oriundos do repositório CRAWDAD

Cada linha nos *traces* no CRAWDAD segue um formato e padrão específico, devendo ser convertidos adequadamente, para uso posterior no ONE:

```
<month>-<day>:<time> <bus_id> <route_id> <unknown> <x_coord> <y_coord>
```

A tabela A.1 explica cada um dos parâmetros:

Tabela A.1: Formato do *trace* do CRAWDAD

Campo	O que faz
month	Mês quando esta entrada foi registrada no *trace*.
day	Dia do mês quando esta entrada foi registrada no *trace*.
time	Horário, no formato **horas:segundos:minutos**, de quando esta entrada foi registrada no *trace*. O tempo no ONE é em segundos, portanto este horário deve ser convertido integralmente para segundos.
bus_id	Identificador do ônibus cuja entrada foi registrada no *trace*.
route_id	Identificador da rota sendo percorrida pelo ônibus cuja entrada foi registrada no *trace*.
unknown	Valor de propóisto desconhecido.
x_coord	Coordenada X, em pés, da posição do ônibus quando cuja entrada foi registrada no *trace*. As coordenadas X no ONE estão em metros, portanto é necessário converter estas coordenadas de pés para metros.
y_coord	Coordenada Y, em pés, da posição do ônibus quando cuja entrada foi registrada no *trace*. As coordenadas Y no ONE estão em metros, portanto é necessário converter estas coordenadas de pés para metros.

A seguir um exemplo do conteúdo típico encontrado em um trace do Ad-Hoc City, oriundo do CRAWDAD:

```
...
02-12:13:46:31 9065 550 55001028 45566.875000 152401.937500
02-12:13:46:33 2443 255 25501379 44335.375000 154412.093750
02-12:13:46:33 2426 307 30701128 47950.875000 173393.875000
02-12:13:46:34 3109 016 01600095 45539.500000 176668.031250
02-12:13:46:34 2328 071 07100035 59960.250000 179441.437500
...
```

Neste exemplo, no dia 02 de dezembro, às 13 horas, 46 minutos e 31 segundos, o ônibus de id 9065, se encontrava na rota 550 ocupando a posição (em pés) (45566.875000, 152401.937500).

A.2 Formato dos *traces* utilizados no simulador ONE

No simulador ONE, para que possam ser utilizados traces em cada simulação, é necessário que o modelo de mobilidade escolhido para o grupo de nós seja o **ExternalPathMovement**,

e o trace do CRAWDAD precisa ser tratado e convertido em dois outros arquivos, ***traceFile*** e ***activeFile***, como discutido no capítulo 4.

Para converter o *trace* do CRAWDAD para o formato compatível com o simulador ONE, apenas alguns campos do trace original são utilizados, sendo eles *time*, *bus_id*, *x_coord*, *y_coord*. Para cada um deles, é necessário fazer um tratamento específico:

- ***time*:** Deve ser convertido para segundos, que é a unidade de tempo utilizada no ONE.
- ***bus_id:*** Os identificadores dos ônibus no *trace* original possuem valores não necessariamente sequenciais, por exemplo 900, 903, 906, ... e no ONE, em um grupo de nós, a ordem dos identificadores é sequencial, começando do número 0, assim teríamos 0, 1, 2, 3,..., portanto deve ser feito o mapeamento adequado de cada ônibus com os ids dos nós no ONE,.
- ***x_coord*** e ***y_coord:*** Devem ser convertidas para metros, que é a unidade de medida utilizada no ONE.

Além disso, o total de nós presentes no grupo de nós deve ser igual ao total de nós registrados no *traceFile*. Por exemplo se o traceFile registrou um total de 1160 nós, no arquivo de configuração *default_settings.txt* o grupo de nós tambem deve conter 1160 nós.

Nós desenvolvemos uma aplicação parser em Python (Versão 3.4.3) que recebe como entrada um *trace* do CRAWDAD, e como saída gera um arquivo traceFile e um arquivo activeFile, já fazendo todas as conversões descritas anteriormente. Na sequência uma explanação dos arquivos *traceFile* e *activeFile*.

A.2.1 traceFile

Este arquivo descreve todas as posições, em termos de coordenadas X e Y, que cada nó ocupa, e em que tempo ocupou a posição, durante a simulação, ou seja, é o arquivo contendo as posições que os ônibus ocupam, ou ocuparão, durante uma simulação.

A primeira linha do traceFile sempre segue o padrão:

```
<total_nós> <tempo_mín> <tempo_máx> <x_mín> <x_máx> <y_mín> <y_máx>
```

Esta linha faz um resumo contendo o total de nós presentes no trace, as mínimas posições X e Y que foram ocupadas no trace, bem como as máximas posições, assim como também contém

o menor e maior tempo quando um nó ocupou uma posição X e Y. A tabela explica cada campo. Por posições Xe Y mínimas e máximas, referimo-nos, respecitvamente, às menores e maiores posições ocupadas pelos nós nos *traces*, ou seja, as posições de menor e maior valor.

Tabela A.2: Formato do primeira linha do *traceFile*

Campo	**O que faz**
total_nós:	Total de nós presentes no *trace*, ordenados sequencialmente começando sempre do valor 0, e no grupo nós com o modelo de mobilidade ExternalPathMovement.
tempo_mín	Tempo mínimo, em segudos, registrado no *trace*. Não há tempo menor que este em todo o *trace*, podendo ser considerado o primeiro e menor tempo em que algum ônibus ocupou uma posição.
tempo_máx	Tempo máximo, em segudos, registrado no *trace*. Não há tempo maior que este em todo o *trace*, podendo ser considerado o último e menor tempo em que algum ônibus ocupou uma posição.
x_mín	Coordenada X mínima, em metros, registrada no *trace* Não há coordenada X menor que esta em todo o *trace*, podendo ser considerada a menor posição X ocupada por algum ônibus.
x_máx	Coordenada X máxima, em metros, registrada no *trace* Não há coordenada X maior que esta em todo o *trace*, podendo ser considerada a maior posição X ocupada por algum ônibus.
y_mín	Coordenada Y mínima, em metros, registrada no *trace* Não há coordenada Y menor que esta em todo o *trace*, podendo ser considerada a menor posição Y ocupada por algum ônibus.
y_máx	Coordenada Y máxima, em metros, registrada no *trace* Não há coordenada Y maior que esta em todo o *trace*, podendo ser considerada a maior posição Y ocupada por algum ônibus.

Da segunda linha em diante do traceFile, segue-se o padrão:

```
<id_nó> <tempo1,x1,y1> <tempo2,x2,y2> ...
```

Desta linha em diante descreve-se para cada nó do trace todas as posições ocupadas pelo mesmo, e em que instante de tempo ocupou tal posição. Cada posição é representada por uma tupla (tempo,x,y). A tabela A.3 explica cada campo:

Tabela A.3: Formato das demais linhas do *traceFile*

Campo	O que faz
(tempo,x,y)	Tupla descrevendo uma posição geográfica do ônibus e o tempo em que o mesmo ocupou tal posição.

Um exemplo do conteúdo típico encontrado num traceFile:

```
4 0 43200 0 3000 0 3000
0 0,0,0 1000,0,750 2000,0,1500 3000,0,2250 ...
1 0,750,0 1000,750,750 2000,750,1500 3000,750,2250 ...
2 0,1500,0 1000,1500,750 2000,1500,1500 3000,1500,2250 ...
3 0,2250,0 1000,2250,750 2000,2250,1500 3000,2250,2250 ...
```

Neste exemplo o trace possui 4 nós, onde o tempo mínimo é 0 segundos, e o tempo máximo é 43200 segundos, enquanto que as posições X mínima e máxima são 0 e 3000, e as posições Y mínima e máxima são 0 e 3000. Da segunda linha em diante descrevendo todas posições de cada nó, especificamente, a segunda linha descreve as posicões do nó 0, notamos que sua primeira posição é (0,0), ocupada no início da simulação (Tempo 0 segundos). Após isso no tempo 1000 segundos, ele ocupa a posição (0,750), e assim sucessivamente.

O total de nós presentes registrados no traceFile será exatamente o mesmo total que deverá ser configurado para o grupo de nós no arquivo **default_settings.txt**. Por exemplo se o traceFile registrou um total de 1160 nós, no arquivo de configuração *default_settings.txt*, o grupo de nós tambem deve conter 1160 nós.

A.2.2 activeFile

Este arquivo descreve os intervalos de tempo, sendo um tempo inicial e um tempo final, em que cada nó permaneceu em atividade, ou seja, quando suas interfaces de rede permaneceram ativadas, recebendo transmissões conforme um nó entra em seu alcance. Estes tempos inicial e final são derivados do próprio *traceFile*, onde captura-se os primeiros tempos de atividade dos nós, que encontra-se as primeira posições que ocuparam no trace, assim como captura-se os últimos tempos de atividade do nós, que encontram-se junto às últimas posições ocupadas pelos nós no *trace*.

Cada linha do *activeFile* segue o seguinte padrão:

```
<id_nó> <tempo_inicial> <tempo_final>
```

A tabela A.4 explica cada campo:

Tabela A.4: Formato das linhas do *activeFile*

Campo	**O que faz**
id_nó	Identificador do nó cujos tempos inicial e final de atividade foram registrados no *trace*.
tempo_inicial	Tempo inicial de atividade do nó. É o tempo da primeira posição ocupada pelo nó no *trace*.
tempo_final	Tempo final de atividade do nó. É o tempo da última posição ocupada pelo nó no *trace*.

Segue um exemplo do conteúdo típico encontrado em um *activeFile*. Considere como sendo um activeFile relacionado ao traceFile da subseção anterior, pois como sabemos, o *activeFile* é gerado com base nos dados do *traceFile*:

```
0 0 43200
1 0 43200
2 0 43200
3 0 43200
```

Neste exemplo, dizemos que os nós 0, 1, 2 e 3, estiveram ativos desde o tempo 0 a 43200 segundos.

Anexo B

DETALHES TÉCNICOS SOBRE A VISUALIZAÇÃO DOS TVGS

Para a geração de TVGs neste trabalho, processamos o relatório de eventos de uma simulação no ONE, o ***eventlog***, por meio de uma ferramenta parser desenvolvida por nós em Python (Versão 3.4.3), que os converte para linguagem do software GraphViz.

Tomando como exemplo o TVG da imagem 2.7 do capítulo 2, seu código em GraphViz é:

```
graph {
rankdir=LR;
n1--n2 [label="[2,6] U [10,20]"];
n2--n3 [label="[1,3]"];
n2--n5 [label="[3,4]"];
n3--n4 [label="[1,7]"];
n3--n5 [label="[1,5]"];
n4--n5 [label="[3,8]"];
}
```

Define-se um grafo com as palavra-chave ***graph***, inserindo o conteúdo do grafo entre o par de chaves, e cada linha é terminada em ponto-e-vírgula, tal como nas linguagens C++, C ou Java.

A primera linha do exemplo, **rankdir=LR**, configura o grafo para ser representado de modo que seus nós estejam posicionados de maneira mais horizontal, ou seja, um ao lado do outro. As demais linhas representam a definição das arestas entre nós, e podemos definir o rótulo das arestas, ***label***, e definido os intervalos de conectividade como descritos pelo rótulo, geramos uma representação visual coerente com a do grafo subjacente do TVG.

Anexo C

DETALHES TÉCNICOS SOBRE O SIMULADOR ONE

Comentamos aqui sobre detalhes mais técnicos no ONE, incluindo seus detalhes de implementação acerca dos métodos interessantes ao nosso trabalho, bem como os parâmetros do arquivo de configuração *default_settings*.txt.

C.1 O arquivo de configuração *default_settings.txt*

No capítulo 4 comentamos superficialmante sobre o arquivo de configuração *default_settings.txt*, o qual é essencial para o funcionamento de uma simulação, pois é dele que são lidos os parâmetros utilizados em uma simulação. Neste Anexo nós o detalhamos.

Na sequência, temos um exemplo do conteúdo típico encontrado no arquivo *default_settings.txt*, sendo similar às configurações utilizadas neste trabalho.

```
...
Scenario.nrofHostGroups = 1
Scenario.endTime = 20000.0

Group.groupID = n
Group.movementModel = ExternalPathMovement
Group.nrofHosts = 1160
Group.traceFile = traces/Baseline-Path_File.txt
Group.activeFile = traces/Baseline-Active_File.txt
Group.router = TVSPRouter
Group.bufferSize = 2G
```

```
Group.nrofInterfaces = 1
Group.interface1 = highspeedInterface

highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 0.625M
highspeedInterface.transmitRange = 50

Events.nrof = 1
Events1.class = ExternalEventGenerator
Events1.filePath = tvg/ExternalMessageEvent.txt

Report.nrofReports = 1
Report.report1 = EventLogReport
...
```

Na tabela C.1 explicamos os parâmetros mais importantes para a execução do nosso algoritmo TVFP em uma simulação.

Tabela C.1: Parâmetros encontrados no *default_settings.txt*

Parâmetro	**O que faz**
Scenario.nrofHostGroups	Quantidade de grupos de nós presentes na simulação. Neste trabalho agrupamos os nós em um único grupo.
Scenario.endTime	Tempo máximo, em segundos, da simulação.
Group.groupID	Identificador do rótulo do grupo de nós. Utilizamos o rótulo "*n*"para o grupo de nós.
Group.movementModel	Modelo de mobilidade utilizado pelos nós do grupo. O modelo *ExternalPathMovement* reproduz *traces* externos.
Group.nrofHosts	Total de nós presentes em um grupo de nós.
Group.traceFile	Caminho do diretório onde se encontra o *traceFile*, utilizado pelo *ExternalPathMovement*.
Group.activeFile	Caminho do local onde se encontra o *activeFile*, utilizado pelo *ExternalPathMovement*.
Group.router	Algoritmo de roteamento utilizado pelo grupo de nós. O TVFP é representado pela classe ***TVFPRouer***.
Group.bufferSize	Capacidade máxima do *buffer* de armazenamento dos nós do grupo.
Group.nrofInterfaces	Quantidade total de interface de redes que cada nó do grupo possui.
Group.interface1	Identificador da interface de rede dos nós do grupo.
highspeedInterface.type	Tipo da interface de rede utilizada.
highspeedInterface.transmitSpeed	Largura de banda da interface de rede.
highspeedInterface.transmitRange	Alcance, em metros, da interface de rede.
Events.nrof	Total de geradores de mensagens utilizados na simulação.
Events1.class	Gerador de mensagens utilizado. Utilizamos o ***ExternalEventGenerator***, que gera mensagens lidas de um arquivo externo, permitindo gerar uma única mensagem na rede.
Events1.filePath	Caminho do diretório onde se encontra o arquivo de geração de mensagens.
Report.nrofReports	Quantidade de relatórios que serão gerados.
Report.report1	Tipo de relatório a ser gerado na simulação. Neste trabalho utilizamos o ***EventLogReport***.

C.2 Detalhes do relatório *EventLog*

Como discutido anteriormente, para este trabalho nos interessa, dos muitos relatórios existentes no ONE, o relatório ***EventLog***, o qual registra todos os eventos ocorridos durante a simu-

lação.

Além disso, tambem comentamos que dos eventos da simulação registrados no relatório, nos interessam apenas os eventos referentes a conexão e desconexão entre pares de nós, ou seja, ***CONN UP*** e ***CONN DOWN***. Esta informação referente ao estado de conectividade entre nós é utilizada para gerarmos TVGs.

Cada linha do *eventlog* segue o seguinte formato e padrão:

```
<tempo> <tipo> <nó_1> [<nó_2>] [<parâmetros_específicos]
```

A tabela C.2 explica os campos:

Tabela C.2: Campos do relatório *eventlog*

Campo	O que faz
tempo	Tempo, sem segundos, quando ocorreu o evento.
tipo	Tipo do evento ocorrido. Pode assumir vários tipo, sendo que para nós nos interessa os eventos *CONN* referentes a conectividade entre pares de nós.
nó_1	Um dos nós envolvidos no evento.
nó_2	Outro dos nós envolvidos no evento. Este é que não aparecem todos os eventos do *eventlog*, mas para os eventos *CONN* é obrigatório, uma vez que eventos de conectividade envolvem pares de nós.
parâmetros_específicos	Parâmetros dependentes do tipo do evento ocorrido. No caso de eventos *CONN* têm-se os valores ***UP***, para eventos de conexão, e ***DOWN***, para eventos de desconexão.

Na sequência, um exemplo do conteúdo típico encontrado em um *eventlog*:

```
1.0 C n79 M1
81.59999999999964 CONN n79 n43 up
...
117.4999999999976 CONN n79 n43 down
214.3999999999921 CONN n83 n89 up
273.29999999999364 CONN n83 n89 down
536.3000000000534 CONN n79 n22 up
...
536.4000000000534 CONN n43 n22 up
...
```

Na primeira linha, no segundo 1 da simulação, ocorre um evento do tipo *C*, ou seja, de criação de mensagens, onde é criada a mensagem $M1$ no nó $n79$. Nas demais linhas vemos uma sucessão de eventos de conexão e desconexão, por exemplo, aos 81.6 segundos, entre os nós $n79$ e $n43$, no caso *CONN UP*, que assim permaneceram até se desconectarem, ou seja, *CONN DOWN*, aos 117.5 segundos.

C.3 Detalhes de implementação do ONE

Como sabemos, o ONE é um simulador implementando na linguagem Java, sendo , portanto, seu código fonte composto de várias classes em Java.

A classe principal do simulador, sendo a primeira a ser carregada duranta a execução do simulador junto com uma simulação, é a classe ***DTNSim.java***, a qual se encontra no diretório ***core***, que junto com esta classe, armazena outras classes com funcionalidades principais do simulador. Na classe *DTNSim.java* temos a função ***main*** conhecida do Java, e é nela que são carregados todos os elementos da simulação, bem como é nela que é feita a leitura dos parâmetros do *default_settings.txt*. Nos interessa esta classe pois a partir dela podemos carregar elementos globais que permanecerão executando durante a simulação.

Para os propósitos deste trabalho são interessantes as classes relacionadas aos algoritmos de roteamento, bem como seus métodos, localizadas no diretório ***routing***, portanto a classe JAVA com a implementação TVFP, a qual chamamos de ***TVFPRouter***, também deve ser mantida neste diretório.

Esta nomenclatura *TVFPRouter* é equivalente ao nome da classe *TVFPRouter.java*, do mesmo modo que temos *EpidemicRouter* referindo-se à classe *EpidemicRouter.java*. Com ela configuramos qual será o algoritmo de roteamento utilizado pelo grupo de nós durante a simulação, portanto basta escrever no parâmetro referente ao algoritmo de roteamento o nome da classe sem a extensão.

Todas as classes dos algoritmos de roteamento, que englobam desde simples implementaçãoes de algoritmos como Epidemic, até algoritmos mais complexos como, PROPhet, são subclasses de duas classes mãe, ***MessageRouter*** e ***ActiveRouter***, sendo que a ActiveRouter deriva da MessageRouter, logo a MessageRouter é a superclasse "avó", e a ActiveRouter a superclasse "mãe"dos algoritmos de roteamento.

Nestas duas classes existem diversos métodos, que são chamados em algum momento do processo de transferência da mensagem de um nó a outro, e.g. métodos que são chamados

quando a transferência da mensagem está para ser iniciada ou quando a mensagem chega a um nó, bem como há métodos referentes ao que acontece quando uma conexão é estabelecida, ou desfeita, entre pares de nós.

Para nosso trabalho é interessante um método específico da classe *ActiveRouter*, chamado ***tryMessagesToConnections***, que é chamado ANTES de um nó tentar enviar uma ou mais mensagems para os nós vizinhos com quem se encontra conectado no momento. Observe o exemplo de código abaixo:

Algoritmo C.1: ***tryMessagesToConnections***

```
protected Connection tryMessagesToConnections(List<Message> messages,
     List<Connection> connections) {
   for (int i=0, n=connections.size(); i<n; i++) {
      Connection con = connections.get(i);
      Message started = tryAllMessages(con, messages);
      if (started != null) {
         return con;
      }
   }
   return null;
}
```

Ele recebe como argumentos a **lista de mensagens** das mensagens que o nó carrega em seu *buffer*, e uma **lista de conexões** contendo os nós com os quais este nó está conectado no momento, respectivamente, os argumentos ***List<Messages> messages*** e ***List<Connection> connections***. Uma vez que o nó com a lista de mensagens chamar este método, ele itera por todas as conexões, enviando mensagens, enquanto tiver, para cada um dos nós vizinhos com os quais está conectado.

Um detalhe importante é que apenas o nó contendo a lista de mensagens chama este método, uma vez que não faz sentido um nó sem mensagem alguma tentar enviar mensagems.

Neste código, se inserirmos condicionais, podemos controlar o fluxo de envio das mensagens, alterando o comportamento padrão dos nós de enviarem a mensagem na primeira oportunidade de contato, desta forma restringindo para qual nó enviaremos a mensagem.

Na sequência explicamos as alterações que fizemos na classe *DTNSim.java* e no método *tryMessagesToConnections*, de modo a satisfazer nossos propósitos com o TVFP e TVGs.

Anexo D

DETALHES TÉCNICOS SOBRE O TVFP E OS TVGS

Aprendemos anteriormente sobre alguns detalhes de implementação no código JAVA do ONE, em especial o que ocorre na classe principal *DTNSim.java*, bem como o propósito do método *tryMessagesToConnections*, e neste anexo explicamos sobre o código implementado para este trabalho.

D.1 Detalhes de implementação dos TVGs

Aprendemos a gerar um TVG de uma simulação de *traces* de ônibus no ONE, de modo que seu formato final seja compatível com o software GraphViz. No entanto para se utilizar este TVG com o ONE, é necessário processar o grafo TVG, por meio de uma ferramenta *parser* desenvolvida por nós em linguagem Python 3.4.3, para convertê-lo para um formato compatível com *GraphViz*, de modo que possamos armazená-lo em estruturas de dados em Java, assim, tornando-o compatível e utilizável com o simulador ONE, que é feito em linguagem Java. Este TVG gerado faz parte de um processo maior de mineração de dados de uma simulação no ONE, para que tais dados possam ser utilizados, na forma de um TVG, com nosso algoritmo TVGP.

Na classe *DTNSim.java* declaramos um objeto *singleton* para representar nosso TVG, e o representamos segundo este padrão de projeto de modo que só haja uma única instância do TVG executando durante a simualação. Ainda na *DTNSim.java*, chamamos o método ***populateGraph*** do TVG de modo que seja feito um processamento dentro do simulador ONE, de modo a converter o TVG, atualmente em formato GraphViz, para as estruturas de dados em Java adequadas que o armazenem adequadamente.

Com o TVG populado, agora fazemos a leitura do arquivo externo de geração de mensagens, que além de conter a única mensagem que é utilizada na rede, que também contém quem são os nós origem e destino, que serão passados como parâmetros para o TVFP. Após isso, o TVFP calcula o caminho, caso exista, e o armazena na lista global *fastest_path*.

D.2 Detalhes de implementação do TVFP

O algoritmo TFVP foi implementado como um algoritmo de roteamento na sua respectiva classe ***TVFPRouter.java***, e este como qualquer algoritmo de roteamento no ONE, encontra-se armazenado no diretório *routing*, além disso, é uma classe derivada das super classes *MessageRouter.java* e *ActiveRouter.java*.

Para que o TVFP possa seguir fielmente o melhor caminho calculado pelo TVFP, alteramos o método *tryMessagesToConections*, de modo como ele envia as mensagens, restringindo o envio apenas a nós específicos. Observe o código D.1 que mostra as alterações que fizemos neste método.

Algoritmo D.1: ***tryMessagesToConnections (Versão TVFP)***

```
  protected Connection tryMessagesToConnections(List<Message> messages,
      List<Connection> connections) {
    for (int i=0, n=connections.size(); i<n; i++) {
      Connection con = connections.get(i);
      String a = con.getOtherNode(this.getHost()).toString();
      String b =
          TVG.getInstance().getCurrentFastestPath().getFirst().toString();
      if(a.equals(b)){
        TVG.getInstance().getCurrentFastestPath().removeFirst();
        Message started = tryAllMessages(con, messages);
        if (started != null) {
          return con;
        }
      }
    }
    return null;
  }
}
```

Suponhamos que a mensagem esteja no nó origem e, conforme este nó percorre o mapa na simulação, eventualmente ele se conectará com outros nós, os quais serão adicionados à sua lista de conexões ativas.

Para cada um dos nós conectados à origem, a mesma verifica se o rótulo destes nós são idênticos ao rótulo do primeiro nó na lista *fastest_path*. Se forem iguais, ele remove este nó da lista, e envia a mensagem para ele. Caso os rótulos não coincidam, ele simplesmente ignora este nó, e itera para a próxima conexão, ou seja passa para o próximo nó com o qual está conectado para verificar se deve transmitir a mensagem para o mesmo.