UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

LUÍS CÉSAR MARQUES DE VASCONCELOS

TRGET: GERAÇÃO DE TRACES PARA SIMULAÇÃO DE VANETS POR MEIO DE REDES DE PETRI COLORIDAS

FORTALEZA
2017

LUÍS CÉSAR MARQUES DE VASCONCELOS

TRGET: GERAÇÃO DE TRACES PARA SIMULAÇÃO DE VANETS POR MEIO DE REDES DE PETRI COLORIDAS

Dissertação apresentada ao Programa de Pós-graduação em Engenharia de Teleinformática do Departamento de Engenharia de Teleinformática da Universidade Federal do Ceará, como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia de Teleinformática. Área de concentração: Sinais e Sistemas.

Orientador: Prof. Dr. José Marques Soares
Coorientador: Prof. Dr. Giovanni Cordeiro Barroso.

FORTALEZA
2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

V45t Vasconcelos, Luís César Marques de.
Trget: Geração de traces para simulação de VANETs por meio de redes de petri coloridas / Luís César Marques de Vasconcelos. – 2017.
86 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2017.
Orientação: Prof. Dr. José Marques Soares.
Coorientação: Prof. Dr. Giovanni Cordeiro Barroso.

1. VANET. 2. Modelos de Mobilidade. 3. Simulação. 4. Rede de Petri colorida. I. Título.
CDD 621.38

LUÍS CÉSAR MARQUES DE VASCONCELOS

TRGET: GERAÇÃO DE TRACES PARA SIMULAÇÃO DE VANETS POR MEIO DE REDES DE PETRI COLORIDAS

Dissertação apresentada ao Programa de Pós-graduação em Engenharia de Teleinformática do Departamento de Engenharia de Teleinformática da Universidade Federal do Ceará, como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia de Teleinformática. Área de concentração: Sinais e Sistemas.

Aprovada em: 15/12/2017.

BANCA EXAMINADORA

Prof. Dr. José Marques Soares (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Antonio Alfredo Ferreira Loureiro
Universidade Federal de Minas Gerais (UFMG)

Prof. Dr. Danielo Gonçalves Gomes
Universidade Federal do Ceará (UFC)

Prof. Dr. Giovanni Cordeiro Barroso (Coorientador)
Universidade Federal do Ceará (UFC)

Dedico este trabalho ao meu pai, Aurélio, minha mãe, Liduina, minha esposa, Lia, e meus filhos Guilherme e Isabela, por todo o amor, incentivo e apoio incondicional durante toda esta jornada.

AGRADECIMENTOS

À Deus por iluminar meu caminho e me dar forças para seguir sempre em frente.

Aos meus pais Aurélio e Liduina, minha esposa Lia e meus filhos Guilherme e Isabela, pelo apoio, paciência, suporte e amor incondicional.

Aos meus sogros Juscelino e Antonieta pelo incentivo, suporte e apoio em todas as horas.

Ao meus orientadores, Prof. Dr. José Marques Soares e Prof. Dr. Giovanni Cordeiro Barroso, pela excelente orientação.

Ao meu chefe, prof. Dr. Joaquim Bento Cavalcante Neto, pelos sábios conselhos e tutoria, além da grande paciência e altruísmo.

Ao meu colega de Divisão de Sistemas da Informação, da Secretaria de Tecnologia da Informação da UFC, Sílvio Amarante, pelo apoio durante todo este trabalho.

Ao meu colega de trabalho David Bôto, pelas longas conversas acerca da temas científicos e a suas sábias sugestões sobre o andamento desta pesquisa.

A Universidade Federal do Ceará e a Secretaria de Tecnologia da Informação, meu local de trabalho, pelo apoio e exemplo de órgão público eficiente.

Ao Programa de Pós-Graduação em Engenharia de Teleinformática, pela formação sólida.

Aos organizadores e revisores do I Workshop de computação urbana (CoUrb) pela oportunidade de mostrar os resultados deste trabalho e por suas sábias considerações.

"Nossas virtudes e nossas faltas são inseparáveis, como a força e a matéria. Quando elas se separam, o homem não é mais nada."
(Nikola Tesla)

RESUMO

As redes veiculares *ad hoc* (VANET - *Veicular ad hoc networks*) apresentam o potencial de aumentar a segurança nas estradas e prover serviços de rede para os passageiros. As VANETs possibilitam a criação de aplicações como: detecção de congestionamentos, detecção de condições da estrada, detecção de acidentes, fiscalização e cobranças automáticas, assistência e entretenimento. A realização de testes em cenários reais é uma tarefa árdua devido a questões como alto custo econômico, fazendo com que a simulação seja considerada a melhor forma de se realizar a análise de desempenho. Neste trabalho, apresentamos o modelo *TrGeT* desenvolvido em rede de Petri colorida hierárquica temporizada que gera *traces* de mobilidade utilizando como base modelos de tráfego microscópicos para simulação de VANETs em simuladores de rede. A configuração dos cenários para geração dos *traces* é realizada de forma simplificada e sem necessidade de conhecimento de simuladores de tráfego e mobilidade urbana. A análise de desempenho do modelo mostra a eficácia dos *traces* de mobilidades gerados.

Palavras-chave: VANETs. Modelos de Mobilidade. Rede de Petri colorida. Simulação.

ABSTRACT

Vehicular ad hoc networks (VANET) have the potential to increase road safety and provide network services for passengers. VANETs enable the creation of applications such as: congestion detection, detection of road conditions, accident detection, monitoring and automatic fines collection, assistance and entertainment. Performing tests in real scenarios is an arduous task due to issues such as high economic cost, making simulation the best way to perform performance analysis. In this work, we introduce the *TrGeT* model developed in timed hierarchical coloured Petri nets that generates mobility traces using as base models microscopic traffic flow models to simulate VANETs in network simulators. The scenarios configuration for generation of traces is carried out in a simplified way and without the need of knowledge of traffic and urban mobility simulators. The performance evaluation of the model shows the effectiveness of the generated mobility traces.

Keywords: VANETs. Mobility Models. Colored Petri nets. Simulation.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MANET	*Mobile ad hoc network*
VANET	*Veicular ad hoc networks*
DSRC	*Dedicated short range communications*
WAVE	*wireless access in vehicular environments*
ITS	*Intelligent transportation systems*
MH	*Mobile Host*
IDM	*Intelligent driver model*
CFM	*Car following models*
RPC	Rede de Petri colorida
WMN	*Wireless mesh network*
RW	*Random waypoint*
RSU	*Roadside unit*
V2V	*vehicle to vehicle*
V2I	*vehicles to the infrastructure*
I2V	*infrastructure to vehicle*
QoS	*Quality of service*
OBU	*On board unit*
AU	*Application unit*
PDA	*Personal digital assistant*
OFDM	*Orthogonal Frequency Division Multiplexing*
AP	*Access Point*
LTE	*Long-Term Evolution*
GPS	*Global Position System*
LIDAR	*Light Detection And Ranging*
CSV	*Comma-separated values*
MPH	*Miles per hour*
DS	*Data Set*

LISTA DE SÍMBOLOS

α	Aceleração de um veículo
δ	Expoente de aceleração livre
Δv	Variação entre as velocidades de dois veículos

SUMÁRIO

1 INTRODUÇÃO

As redes veiculares *ad hoc* (VANET - *Veicular ad hoc networks*) representam um sistema de comunicação veicular que tem potencial para melhorar a segurança no trânsito por meio do compartilhamento de informações para a prevenção de acidentes, além de poderem auxiliar também na investigação pós-acidente. As VANETs são uma sub-classe das redes *ad hoc* móveis (MANET - *Mobile ad hoc network*).

Em redes de computadores tradicionais, em que os enlaces tipicamente existem permanentemente, e constantemente, a topologia da rede varia pouco. Nesse tipo de rede o problema mais importante para resolver é o de comunicação de dados. Sem a comunicação a rede se torna inútil, não permitindo a existência de serviços e aplicações. Por outro lado, em uma rede onde há a mobilidade das entidades, como em sistemas móveis e redes veiculares, o problema fundamental não é o problema da comunicação de dados, e sim, o de entender tal mobilidade. Se não for possível entender a mobilidade, não será capaz de propor soluções adequadas de comunicações de dados. Existe uma deficiência muito grande em relação a *traces* que registram a mobilidade para que possamos entendê-la, e assim, projetar soluções que sejam adequadas.

A principal maneira para se realizar a análise de desempenho de um aplicativo ou um protocolo de uma VANET se dá por meio de testes. Porém, a realização de testes em cenários reais é uma tarefa árdua devido a questões como alto custo econômico e constantes mudanças na topologia da rede, fazendo com que a simulação seja considerada a melhor forma de se realizar a análise de desempenho em uma VANET (Al-Sultan *et al.*, 2013).

1.1 Contextualização e Motivação

Para avaliar o desempenho de um protocolo utilizado em uma VANET, é crítica a necessidade de um modelo de mobilidade, em que principal papel é imitar o comportamento do movimento do móvel. Uma das principais diferenças entre as VANETs e as MANETs é a restrição de mobilidade que os nós móveis (automóveis) possuem. Em uma VANET os nós se movem em ruas, avenidas, estradas ou rodovias que apresentam características geométricas próprias que devem ser seguidas. Essa restrição de mobilidade traz às VANETs uma outra característica que é uma maior previsibilidade pelo fato de tratar de veículos que tem que seguir determinadas regras e andar sobre uma malha viária já conhecida. Já as MANETs não apresentam tal restrição, possibilitando a utilização de modelos de mobilidade mais simples como o modelo de mobilidade aleatória (*Random Walk*), em que cada nó da rede gera um percurso aleatório, com variações, também aleatórias, nos módulos das velocidades. Além disso, o modelo apresenta uma independência temporal em que, tanto o módulo quanto a direção da velocidade de um nó, num determinado mo-

mento, independe dos valores anteriores, tornando o modelo pouco realístico e ineficiente para VANETs.

Ao avaliar as soluções de redes veiculares por meio de simulação, o modelo de mobilidade utilizado apresenta um papel importante na confiabilidade dos resultados obtidos (D. Cabarkapa, 2016). De acordo com Camp, Boleng, and Davies (2002), dependendo do modelo de mobilidade os resultados da avaliação de desempenho são diferentes para o mesmo protocolo de roteamento.

O principal papel de um modelo de mobilidade é imitar o comportamento do movimento do dispositivo móvel. O modelo de mobilidade deve refletir as restrições de movimentação dos veículos (*mobile hosts* - MH), incluindo os dados de modelos de tráfego. De acordo com a taxonomia utilizada em Harri, Filali, and Bonnet (2009), os modelos de mobilidade podem ser classificados como: *traces* sintéticos baseados em pesquisa, *traces* de aplicações reais e *traces* sintéticos gerados por simuladores de tráfego e mobilidade urbana. Como as VANETs ainda não são utilizadas em larga escala, existe dificuldade em se obter *traces* reais. Portanto, os *traces* sintéticos representam grande importância para a simulação das redes veiculares.

Os modelos de mobilidade veiculares podem ser classificados como macroscópicos e microscópicos. Os macroscópicos consideram valores como densidade e média das velocidades dos veículos, tempo total de viagem, chegadas e partidas de veículos em larga escala, tratando o tráfego veicular de acordo com a dinâmica dos fluidos. A abordagem microscópica considera cada veículo como uma entidade distinta, modelando o seu comportamento (aceleração, desaceleração, mudança de faixa) de forma mais precisa Harri, Filali, and Bonnet (2009). Os modelos de mobilidade baseados em simuladores de tráfego realizam uma modelagem microscópica. Neles são determinados os movimentos de cada veículo, podendo simular realisticamente infraestrutura de ruas, avenidas e estradas, bem como interações entre veículos. Os modelos de mobilidade mais representativos que vem sendo utilizados em simulação são: *Random Waypoint* (RW), *Manhattan model*, simuladores de tráfego microscópicos e *traces* reais retirados de conjuntos de dados (*datasets*) disponíveis publicamente.

RW foi o primeiro modelo utilizado nos estudos de VANETs. Neste modelo cada nó escolhe aleatoriamente um ponto de origem e destino e uma velocidade entre zero e um dado limite máximo. A velocidade se mantém fixa ao longo da viagem. Ao chegar no ponto de destino o nó seleciona um novo destino e uma nova velocidade e reinicia a viagem. Em Rivoirard *et al.* (2016) os autores elencam os pontos negativos deste modelo: o caminho escolhido sempre é uma linha reta não levando em conta as restrições de mobilidade dos veículos, como as ruas; a dispersão dos valores da velocidade ao longo do tempo e o espaço não é levado em consideração, não sendo possível a representação das variações do tráfego em situações de transito fluido e congestionado.

O modelo *Manhattan* implementa as restrições de mobilidade que não são

encontradas no RW. Nele os nós se movem em uma estrada representada por um *grid* que limita o movimento do veículo. As origens e os destinos dos veículos são obtidos aleatoriamente e as trajetórias são calculadas por algorítimos como *Dijkstra short path*, por exemplo, ou utilizando probabilidade quando o veículo chega a um cruzamento. Como pontos negativos os autores em Rivoirard *et al.* (2016) citam: os veículos podem se mover através dos outros, não simulando as interações entre os mesmos.

Os simuladores de tráfego microscópico procuram modelar o comportamento do condutor. Para tal devem implementar um dos seguintes modelos: modelos estocásticos, modelos de fluxo de tráfego, modelos de filas, modelos de *car following*, ou modelos de comportamento. Nesses simuladores são simulados cada carro em detalhes como, frenagem, aceleração e mudança de faixas, assim como uma representação das estradas e suas restrições como limite de velocidade, sentidos permitidos e quantidade de faixas. Em Rivoirard *et al.* (2016) os autores ressaltam que os pontos de origem e destino são escolhidos aleatoriamente sem levar em consideração as especificidades das estradas e que os perfis de aceleração e desaceleração são diferentes da realidade por ignorarem os comportamentos extremos de alguns condutores.

Existe uma gama de ferramentes disponíveis gratuitamente e pagas que geram *traces* de mobilidade que atendem aos requisitos necessários para simulação de VANETS. A maioria delas são ferramentas de simulação específica de tráfego e mobilidade urbana que possuem *parsers* para a geração de *traces* em determinados formatos. Porém, a utilização de tais ferramentas demandam o entendimento básico de como funciona um cenário de tráfego com todos os seus componentes como, por exemplo, junções entre ruas e temporizações de sinais de trânsito. Esta restrição afasta o pesquisador que busca em tais ferramentas apenas uma maneira de obter *traces* para simular uma aplicação em um rede veicular por meio de um simulador de rede.

Para mitigar essa restrição, foram desenvolvidas ferramentas que permitem a geração de *traces* de forma rápida onde é necessário uma configuração mínima de um mapa (algumas permitem a geração aleatória do mesmo) e de grupos de nós (veículos) e suas características mais básicas como tamanho do veículo, velocidade mínima e máxima.

A documentação de tais ferramentas não deixa explícita a existência de opções de interação com a simulação por meio da manipulação das variáveis envolvidas, como velocidade ou localização dos veículos, ou modificar o estado inicialmente configurado, como adicionar novos veículos. Trazer a possibilidade de interação com a simulação, junto à possibilidade de gerar cenários de simulação de mobilidade urbana de forma rápida motivam a elaboração deste trabalho.

1.2 Objetivos e Contribuições

O objetivo deste trabalho é apresentar uma posposta para a geração de *traces* para simulação de VANETs implementada por meio do modelo *TrGeT: Traces Generator Tool*, desenvolvido em rede de Petri colorida (RPC) hierárquica temporizada por meio da ferramenta *CPN Tools* (CPN Tools, 2015).

Assim como as ferramentas já existentes como *VanetMobiSim* e MOVE, *TrGeT* gera *traces* de mobilidade para simulação de VANETs de forma rápida. Porém, como diferencial, no *TrGeT* é possível alterar o comportamento do modelo ao longo da simulação, permitindo o acompanhamento passo a passo (forma *stepwise*) da geração de *traces* para estudos e análises específicas. Uma outra contribuição do modelo *TrGeT* é a possibilidade de estender e modificar o modelo de mobilidade utilizado na simulação por meio de mudanças no modelo RPC.

1.3 Publicação

Parte dos resultados deste trabalho, especialmente o cenário 2 do Capítulo 4, foram publicados no artigo *TrGeT*: geração de traces para simulação de Redes Veiculares *Ad Hoc* por meio de Redes de Petri Coloridas, e estão disponíveis nos anais do I *Workshop* de computação urbana (CoUrb) (Vasconcelos *et al.*, 2017). O artigo foi apresentado em maio de 2017 no evento. O CoUrb é um *workshop* integrante da grade do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - XXXV SBRC 2017.

1.4 Organização da Dissertação

Este trabalho é organizado da seguinte forma: No Capítulo 2 são apresentados os conceitos necessários ao entendimento do trabalho, assim como os trabalhos relacionados. No Capítulo 3 o modelo *TrGeT* é detalhado, assim como é descrito o processo completo de simulação e descrito os arquivos gerados ao fim da mesma. No Capítulo 4 são apresentados os cenários de simulação e a discussão dos resultados. Finalmente, no Capítulo 5 são apresentados os resultados obtidos e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

As redes veiculares *ad hoc* (VANET - *Veicular ad hoc networks*) são uma subclasse das redes *ad hoc* móveis (MANET - *Mobile ad hoc network*). Elas representam um sistema de comunicação veicular que tem potencial para melhorar a segurança no trânsito por meio do compartilhamento de informações para a prevenção de acidentes, engarrafamentos, além de poderem auxiliar também na investigação pós-acidente. Outro tipo de informação que pode ser compartilhada são as informações relacionadas aos viajantes que são consideradas como não relacionadas à segurança (Al-Sultan *et al.*, 2013). Um problema em relação ao aumento do uso do transporte privado é o número crescente de mortes que ocorrem devido a acidentes nas estradas. A intenção por trás da distribuição e compartilhamento desta informação é fornecer uma mensagem de segurança para alertar os motoristas sobre os perigos esperados, a fim de diminuir o número de acidentes e salvar a vida das pessoas, ou para proporcionar aos passageiros viagens mais agradáveis.

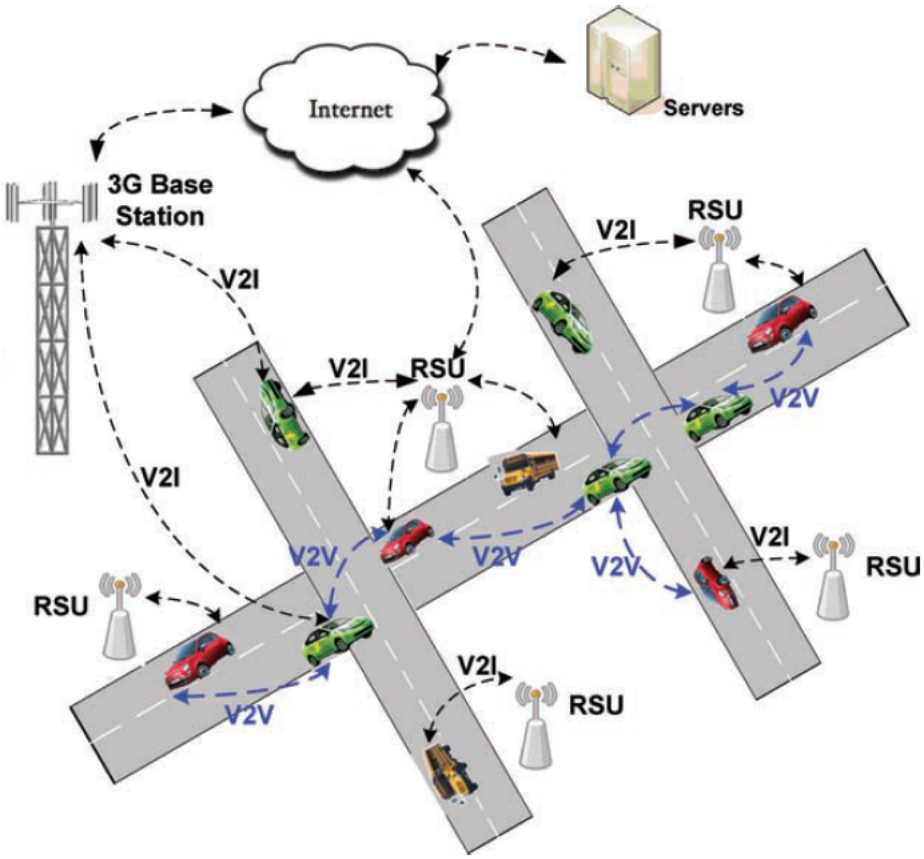
Segundo Basagni *et al.* (2013), VANETs são uma parte fundamental da estrutura de sistemas de transporte inteligente (ITS). No trabalho os autores definem VANETs como redes distribuídas, auto-organizadas e potencialmente de alta mobilidade, compostas de veículos que se comunicam por um meio sem fio. O movimento de cada nó (representado por um veículo) é restringido pela direção da estrada, abrangendo o tráfego e os regulamentos de trânsito. Essas restrições introduzem características específicas às VANETs como a alta dinâmica na topologia da rede e a conectividade intermitente. A direção do movimento dos veículos é previsível até certo ponto, e o consumo de energia pode ser compensado pelas suas baterias.

De acordo com Al-Sultan *et al.* (2013), as VANETS possuem suas próprias características quando comparadas às MANETs. Podem-se citar: mobilidade previsível devido às restrições de tráfego (sinais e regras de trânsito) e a topologia das ruas e estradas; provimento de segurança e eficiência no tráfego; sem restrições de energia já que o próprio veículo fornece; rede de densidade variável devido a variações na concentração de veículos causadas por engarrafamentos, acidentes e concentrações ou de áreas mais densas das cidades como centro; mudanças na topologia da rede causadas pelas altas velocidades dos veículos; redes de alta escala; alto poder computacional devido aos nós da rede serem veículos que podem estar equipados com diversos sensores, alta capacidade de memória e processamento, antenas eficientes e sistema GPS (*Global Position System*).

O ITS é uma arquitetura integrada flexível e escalável. Neste sistema, protocolos baseados em infraestrutura, como protocolos de comunicação celular e protocolos de VANETs, estão sendo integrados para facilitar o processo de coleta de informações. Em uma VANET existem duas formas de comunicação: V2V ou V2I. Na forma V2V (*vehicle to vehicle*), os veículos se comunicam diretamente formando uma comunicação entre veículos. Na forma V2I (*vehicle to infrastructure*) os veículos se comunicam com um equi-

pamento que fica próximo a estrada, chamado de *roadside unit* [1] (RSU) formando uma comunicação entre o veículo e a infraestrutura. Na Figura 1 são mostradas as diversas formas de comunicação em um sistema ITS. Nela podemos verificar os veículos se comunicando diretamente com outros veículos (V2V), veículos se comunicando por meio de uma RSU (V2I), veículos acessando serviços na internet diretamente por meio de estações celulares (V2I) e por intermédio de uma RSU (V2I). Essa integração de serviços é a base de sistemas ITS.

Figura 1 – Modelo de sistema de aplicação ITS



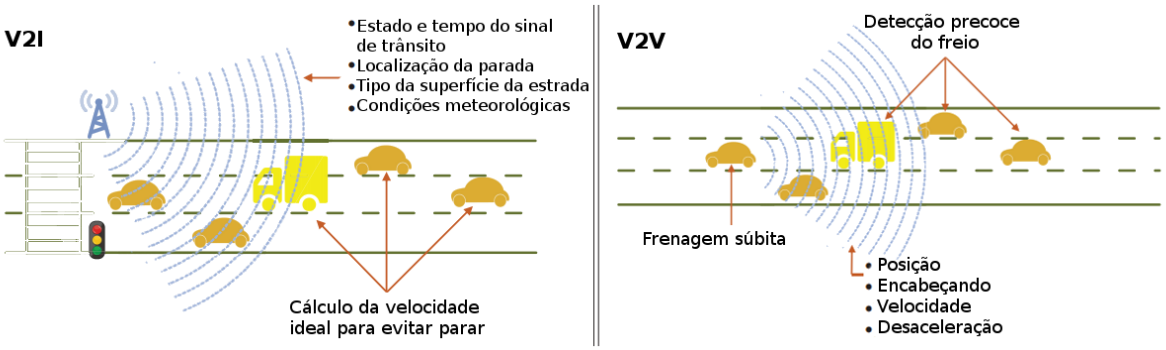
Fonte:Basagni *et al.* (2013).

As formas de comunicação V2V e V2I possuem diferentes funcionalidades e vantagens. As comunicações V2V oferecem vantagens em dar suporte a aplicações de segurança de tempo crítico, como evitar colisões, transmissão de eventos abruptos, informações sobre atividades comerciais nas proximidades (anúncios por meio de RSU), provendo serviços de localização, e evitando custos de infraestrutura e taxas para os usuários. A vantagem do V2I em relação ao V2V é o acesso a aplicativos na Internet, coordenação global do tráfego e previsão (por fusão de dados de informações coletadas em centros de controle), confiabilidade, simplicidade técnica, garantias de QoS, cobertura mais ampla e manutenção profissional (Basagni *et al.*, 2013). Na Figura 2 são ilustradas

[1] Tradução livre: unidades de acostamento

as formas de comunicação V2I e V2V.

Figura 2 – Comunicações V2I e V2V



Fonte: Adaptado de Omar, Lu, and Zhuang (2016).

A implantação de VANETs em escala real traz vários desafios e áreas de pesquisa. Dentre eles podemos citar: desenvolvimento de aplicações, protocolos e ferramentas de simulação.

2.1 Arquitetura

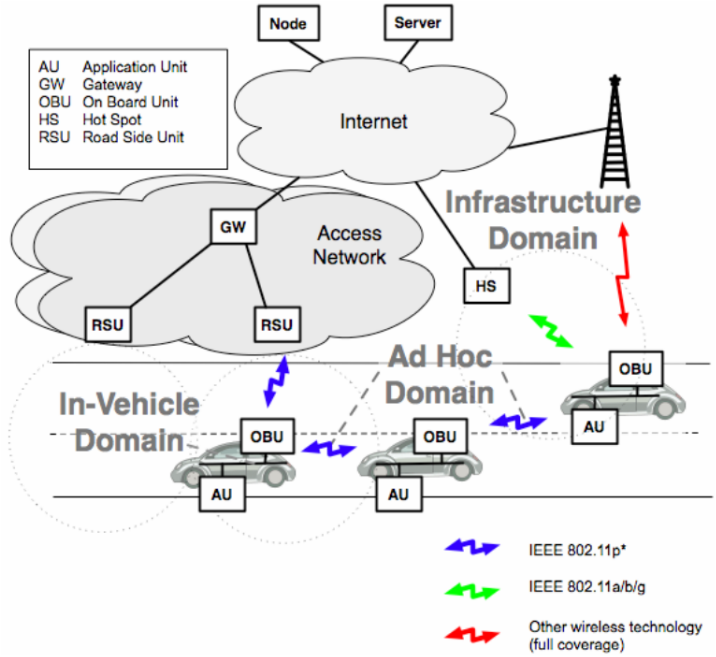
O meio de comunicação sem fio que possibilita que os veículos e as RSUs se comuniquem é chamado de WAVE (*wireless access in vehicular environments*). Os componentes que permitem essa comunicação são: *application unit* (AU), *on board unit* (OBU) e *roadside unit* (RSU). Cada componente possui suas próprias propriedades e relações com os outros componentes.

Uma OBU é responsável pela comunicação V2V, em que um veículo utiliza sua OBU para se comunicar com outro veículo, e pela comunicação V2I, em que um veículo utiliza sua OBU para se comunicar com uma RSU. A OBU também é responsável por fornecer serviços de comunicação para AUs e encaminhar dados em nome de outras OBUs. Uma OBU está equipada com, pelo menos, um único dispositivo de rede para comunicação sem fio de curto alcance baseado na tecnologia de rádio IEEE 802.11p *.

AU é um dispositivo interno ao veículo e que executa aplicações que podem utilizar as capacidades de comunicação da OBU. Uma AU pode tanto ser construída como parte de um veículo (embarcado) e estar permanentemente conectada à OBU, como pode ser dinamicamente conectada na rede interna do veículo, como um *smartphone*.

RSU é um dispositivo físico de localização fixa posicionado ao lado de estradas e rodovias, ou em localizações específicas, como em postos de gasolina, estacionamentos e restaurantes. Uma RSU é equipada com pelo menos um dispositivo de rede para comunicação sem fio de curto alcance baseado na tecnologia de rádio IEEE 802.11p * (Baldessari *et al.*, 2007).

Figura 3 – Arquitetura de referência de uma VANET



Fonte: Baldessari *et al.* (2007).

Na Figura 3 é apresentada a arquitetura de uma VANET com seus componentes e domínios de comunicação. O projeto da arquitetura de uma VANET compreende três tipos de domínios: *in-vehicle* (interno), *ad hoc*, e infraestrutura. O domínio *in-vehicle* consiste de uma OBU e uma ou mais AUs. O domínio *ad hoc* é composto pelos veículos equipados com suas OBUs e as RSUs ao longo das estradas. O domínio de infraestrutura trata da conexão das RSUs com as redes que compõem a infraestrutura de comunicação, como a internet, permitindo que os veículos tenham acesso as mesmas. (Al-Sultan *et al.*, 2013).

2.2 Protocolos

Para possibilitar a implantação de uma rede veicular, uma padrão de acesso sem fio deve ser estabelecido para uma conectividade suave entre os veículos e entre os veículos e as RSUs. DSRC é a sigla para *Dedicated Short Range Communications*, comunicações dedicadas em curto alcance, é usada como padrão para comunicação sem fio entre veículos. O DSRC é um serviço de comunicações de curto a médio alcance que foi desenvolvido para suportar comunicações de veículo para veículo (V2V) e veículo para acostamento (V2I, por meio das RSUs), comunicação de mensagens de segurança V2V, informações de trânsito, cobrança de pedágios (Dalal and Dahiya, 2017).

O outro padrão desenvolvido é o WAVE (*Wireless Access in a Vehicular Environment*) IEEE 802.11p, que é a principal solução proposta para acesso sem fio em VANETs. O DSRC é padronizado por meio do protocolo IEEE 802.11p, que é essencial-

mente o 802.11 alterado para permitir uma baixa sobrecarga. O IEEE padroniza toda a pilha de comunicação pela família 1609 referente ao acesso sem fio em ambientes veicular (WAVE).

As tecnologias de redes celulares, como o padrão LTE (*Long-Term Evolution*), estão atualmente sendo utilizadas pelos fabricantes de automóveis para fornecer seus veículos com algumas aplicações e serviços, como BMW *ConnectedDrive*, Audi *connect* e OnStar (uma subsidiária da General Motors). Tais serviços de rede celular são direcionados principalmente para aplicações que fornecem assistência de direção (por exemplo, aplicativos de navegação), entretenimento de passageiros (por exemplo, conectividade com a Internet) e diagnóstico remoto de veículos. No entanto, não existe uma solução atual para suportar aplicativos de segurança baseados em VANET por meio de redes celulares, mesmo com a implantação do LTE-*Advanced*, que é o último padrão de comunicações móveis da quarta geração (4G) (Omar, Lu, and Zhuang, 2016).

2.3 Aplicações

De acordo com Al-Sultan *et al.* (2013), as comunicações V2V e V2I possibilitam o desenvolvimento de um grande número de aplicações e pode prover uma variedade de informações aos motoristas e viajantes. A integração de dispositivos de bordo com a interface de rede, diferentes tipos de sensores e receptores GPS, concede aos veículos a habilidade de coletar, processar e disseminar informações sobre ele mesmo e seu ambiente a outros veículos em sua proximidade. Isso aumenta a segurança nas estradas e provê conforto aos passageiros. As aplicações da VANET são classificadas de acordo com seu objetivo principal em duas categorias principais: conforto/entretenimento e aplicações de segurança.

As aplicações de conforto/entretenimento são também referenciadas como não relacionadas à segurança. Elas tem o objetivo de melhorar o conforto dos motoristas e passageiros por meio de informações de clima e tráfego, navegação, serviços de localização de restaurantes, postos de gasolina, hotéis, incluindo serviços de anúncio. Os passageiros podem jogar jogos *online*, acessar a internet e seus serviços enquanto o veículo está conectado a uma rede de infraestrutura (Al-Sultan *et al.*, 2013).

Já as aplicações de segurança utilizam a comunicação sem fio entre os veículos ou entre os veículos e a infraestrutura para aumentar a segurança nas estradas e evitar acidentes. Segundo Al-Sultan *et al.* (2013), a intenção é a de salvar vidas e proporcionar um ambiente limpo. Nas aplicações de segurança, os veículos devem possuir a habilidade de coletar informações suas e de outros veículos por meio de sensores, processar e disseminar tais informações na forma de mensagens de segurança para outros veículos e/ou infraestrutura, dependendo da aplicação e suas funções.

Na Figura 4 são mostrados exemplos dessas aplicações e sua classificação.

Figura 4 – Aplicações VANET

2.4 Simulação

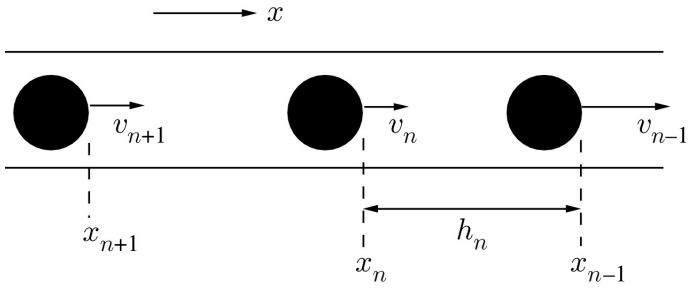
Devido a várias questões, como reprodutibilidade, custos econômicos, falta de escalabilidade, ainda não existe uma rede veicular real em larga escala disponível como cenário de testes. Assim, simulação é um dos métodos mais utilizados para se realizar a análise de desempenho em uma VANET.

2.4.1 Car following models

Em Harri, Filali, and Bonnet (2009), os autores sugerem uma visão dos modelos de mobilidade por meio da identificação dos seguintes blocos funcionais: *restrições de mobilidade*, *gerador de tráfego*, *tempo* e *influências externas*. O bloco funcional *tempo* leva em consideração horários específicos de um dia ou um dia da semana na configuração da mobilidade. O bloco funcional *influências externas* modela o impacto de um protocolo de comunicações ou qualquer outra fonte de informação nos padrões de movimento. O bloco funcional *restrições de mobilidade* descreve o relativo grau de mobilidade de cada veículo. Macroscopicamente são modelados por ruas ou construções, mas microscopicamente pelos carros vizinhos, pedestres ou por diversidades da modelização devido ao tipo de carro ou aos hábitos do condutor. O bloco funcional *gerador de tráfego* define diferentes tipos de carros e lida com suas interações de acordo com o ambiente em estudo. Macroscopicamente, ele modela densidades de tráfego, velocidades e fluxos, enquanto microscopicamente trata de propriedades como a distância entre os carros, aceleração, frenagem, ultrapassagem. Os modelos *Car following models* (CFM) são internos ao bloco funcional *gerador de tráfego*. Na Figura 5 é apresentado o esquema geral do modelo.
em que, $v_n(t)$ e $x_n(t)$ representam a velocidade e o deslocamento do veiculo n, respectivamente, e h_n o espaço entre os para-choques dos veículos (*headway*).

A classe de CFM é a maior classe de modelos microscópicos que implementam padrões de mobilidade humana. Os CFM adaptam a forma dos veículos se moverem de acordo com um conjunto de regras que procuram evitar o contato com o veículo que está a frente. Segue abaixo uma lista com os CFM mais representativos segundo Harri, Filali,

Figura 5 – Esquema geral e notação para CFM.



Fonte: Wilson (2008).

and Bonnet (2009):

- Krauss Model (KM);
- Nagel and Schreckenberg Model (N-SCHR);
- Wiedeman Psycho-Physical Model (Psycho);
- General Motors Model (GM);
- Gipps Model (GP);
- Intelligent Driver Model (IDM).

Neste trabalho utilizaremos o CFM *Intelligent driver model* (IDM), descrito em seguida.

2.4.1.1 Intelligent driver model (IDM)

O modelo de mobilidade *Intelligent driver model* (IDM) (Kesting, Treiber, and Helbing, 2010) mostra uma dinâmica coletiva livre de colisões, exibe propriedades de estabilidade controláveis e implementa uma estratégia de frenagem inteligente com transições suaves entre comportamento de aceleração e desaceleração. No algoritmo IDM, se não existir outro veículo à frente, um veículo segue livre até que alcance a velocidade máxima desejada, caso contrário, regras precisam ser obedecidas para evitar a colisão. O modelo foi desenvolvido para delinear o comportamento da condução humana. Portanto, possui os seguintes parâmetros comportamentais: velocidade desejada, aceleração, desaceleração confortável e tempo de segurança para avanço.

A aceleração no modelo IDM é uma função contínua que incorpora diferentes modos de condução para todas as velocidades no tráfego de rodovia, assim como no tráfego urbano. Além da distância s para o veículo da frente (principal) e a velocidade atual v, o modelo IDM também leva em conta a diferença de velocidade do veículo principal (taxa de aproximação) $\Delta v = v - v_1$, sendo v_1 a velocidade do veículo da frente e v a velocidade do veículo que o está seguindo. A aceleração do modelo IDM é calculada de acordo com a Equação (1).

$$^{\alpha}IDM\left(s, v, \Delta v\right) = \frac{dv}{dt} = \alpha \left[1 - \left(\frac{v}{v_0}\right)^{\delta} \left(\frac{s^*\left(v, \Delta v\right)}{s}\right)^2\right] \tag{1}$$

em que

$$s^*(v, \Delta v) = S_0 + vT + \frac{v\Delta v}{2\sqrt{ab}}. \quad (2)$$

Na Equação (1), v_0 representa a velocidade desejada, δ o expoente de aceleração livre. O expoente δ fica tipicamente entre 1 e 5. Ele permite descrever que o comportamento de aceleração realista dos condutores está entre uma aceleração constante α até a velocidade desejada v_0 ($\delta \to \infty$) e um comportamento de aceleração exponencial ($\delta = 1$). Na Equação (2), T representa o intervalo de tempo desejado entre os veículos, S_0 a distância de congestionamento, a a aceleração máxima, b a desaceleração confortável (ou desejada). Os parâmetros IDM v_0, T, S_0, a e b são empiricamente mensuráveis e têm valores realistas (Treiber, Hennecke, and Helbing, 2000).

2.5 Trabalhos Relacionados

A utilização de ferramentas específicas de simulação de tráfego e mobilidade urbana demandam estudo de parâmetros do cenário de simulação como cruzamentos, junções, sinais de trânsito, diferentes tipos de vias e fluxos de trânsito. Para mitigar essa complexidade na configuração dos cenários de simulação, foram desenvolvidas ferramentas que permitem a geração de *traces* de forma rápida, como o *VanetMobiSim* e MOVE. Nessas ferramentas a configuração é simplificada por não levar em consideração parâmetros que não tenham relevância para geração de *traces* de mobilidade (Karnadi, Mo, and Lan, 2007).

SUMO (*Simulation of Urban MObility*) (Behrisch *et al.*, 2011) é um simulador de tráfego rodoviário microscópico de código aberto projetado para lidar com grandes redes rodoviárias. Possui funcionalidades como: movimento veicular sem colisões, roteamento individual dos veículos, ruas com várias pistas com mudança de faixa. O CFM incluído no SUMO como padrão é uma modificação do modelo Krauß (Krauß, 1998) em que cada veículo acelera até uma "velocidade desejada", enquanto mantém uma distancia de segurança em relação ao veículo a sua frente. Porém, como afirmam Karnadi, Mo, and Lan (2007), o esforço de escrever *scripts* de simulação, bem como aprender sobre os detalhes internos do simulador (parametrização do cenário de simulação com cruzamentos, conexões entre as ruas, sinais de trânsito, diferentes tipos de vias e veículos, fluxos de trânsito) torna demorado o processo de coletar *traces*. A utilização de ferramentas de simulação de tráfego e mobilidade urbana são indicados para *testbeds* de cenários que demandam a utilização de grandes redes rodoviárias e onde se deve considerar os múltiplos fatores envolvidos na mobilidade urbana com o maior nível de realismo possível. Para se obter uma avaliação rápida de protocolos de rede ou de ferramentas que exijam um *time to market* curto, as ferramentas de geração de *traces* de forma rápida são indicadas (D. Cabarkapa, 2016).

MOVE (*MObility model generator for VEhicular networks*) (Karnadi, Mo, and

Lan, 2007) gera modelos de mobilidade para simulações VANET. A ferramenta é integrada ao SUMO, funcionando o MOVE como *parser* para o SUMO. O MOVE gera *traces* de mobilidade que podem ser usados por ferramentas de simulação de rede, como o NS-2, e possui uma interface gráfica que permite gerar rapidamente cenários de simulação. Entretanto, uma limitação da ferramenta é a exigência de instalação da versão 0.12.3 do SUMO, sendo que a versão corrente (no momento em que esse trabalho está sendo escrito) é a 0.30. Apenas a versão 0.12.3 do SUMO é compatível com o MOVE, comprometendo possíveis atualizações da ferramenta, além de possuir uma instalação não trivial.

VanetMobiSim (*Vehicular Ad Hoc Networks Mobility Simulator*) (Härri *et al.*, 2006) é um simulador de modelos de mobilidade desenvolvido em *Java*. O *VanetMobiSim* gera *traces* de movimento em diferentes formatos, suportando diferentes ferramentas de simulação para redes móveis, incluindo NS-2, GloMoSim e QualNet. *VanetMobiSim* apresenta modelos de movimentos automotivos, tanto em níveis macroscópicos como microscópicos. Permite importar mapas do banco de dados digital TIGER United States Census Bureau (2007) e adiciona suporte para estradas com várias pistas, fluxos direcionais separados, restrições de velocidade diferenciadas e sinais de trânsito nas interseções. Para que a ferramenta *VanetMobiSim* seja estendida ou modificada é necessário o conhecimento de sua estrutura interna, possuindo a ferramenta mais de 120 classes Java sem documentação para desenvolvedores (conta-se apenas com o manual do usuário).

Em (Khan, Ahmad, and Baig, 2014) é apresentado um modelo de redes de Petri coloridas em que nós móveis se deslocam em um terreno bidimensional de forma aleatória, e de acordo com o modelo de mobilidade *Random Walk* (RW) Camp, Boleng, and Davies (2002), sem possuir o efeito de borda e decaimento de velocidade. Os autores afirmam que, até a época em que o trabalho foi desenvolvido, era a primeira implementação do RW usando RPC. Em Khan, Ahmad, and Khan (2015) os autores criam em RPC o modelo SHER (*Show Home and Exclusive Regions*), no qual o modelo detalhado em Khan, Ahmad, and Baig (2014) é estendido. De acordo com os autores, o modelo elimina as seis questões-chave dos modelos de mobilidade aleatória, isto é, paradas repentinas, movimentos sem memória, efeito de borda, dependência temporal da velocidade, dependência do tempo de pausa e decaimento da velocidade. Ambas as implementações geram modelos de mobilidade para serem utilizados em redes de malhas (*Wireless Mesh Network - WMN)* Akyildiz, Wang, and Wang (2005), onde não é necessário modelar as restrições de mobilidade como ruas, avenidas e veículos, não sendo adequados para simulações de VANETs. Por meio da Tabela 1 é apresentada uma síntese da comparação entre os trabalhos relacionados.

Um estudo mais aprofundado sobre simulação de VANETs pode ser encontrado em Harri, Filali, and Bonnet (2009) e Al-Sultan *et al.* (2013).

Tabela 1 – Resumo comparativo entre os trabalhos relacionados

Ferramenta	**Gera *traces* VANET**	**Forma rápida**	**Necessita outra ferramenta**	**Interação durante simulação**
SUMO	Sim	Não	Não	Não
VanetMobiSim	Sim	Sim	Não	Não
MOVE	Sim	Sim	Sim (SUMO)	Não
RW RPC	Não	Sim	Não	Sim
SHER	Não	Sim	Não	Sim
TrGeT	Sim	Sim	Não	Sim

2.6 Considerações Finais

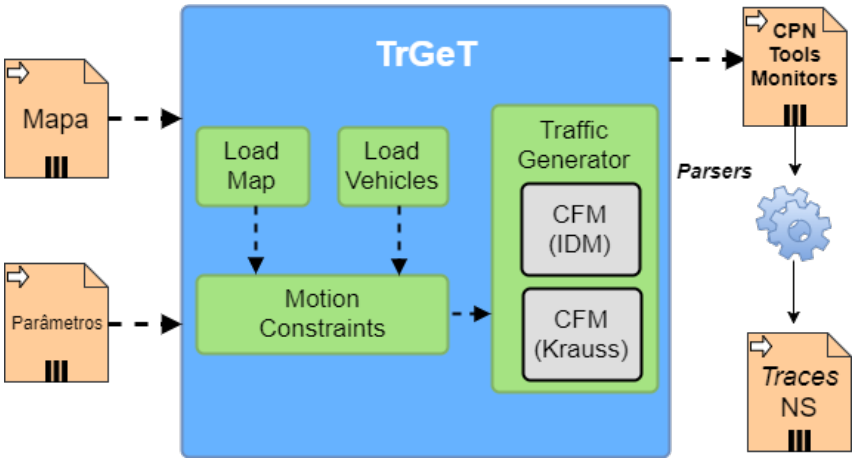
Neste capítulo foram apresentadas as principais bases teóricas que norteiam este trabalho. Foi apresentado uma introdução às VANETs, seus protocolos, sua arquitetura, as principais aplicações e os modelos de mobilidade mais utilizados para realização de simulações. Por fim, foram apresentadas as ferramentas relacionadas à este trabalho. No próximo capítulo a proposta e apresentada em detalhes: o desenvolvimento do modelo *TrGeT: Traces Generator Tool*.

3 TRGET: TRACES GENERATOR TOOL

Propõe-se neste trabalho um modelo para geração de *traces* chamado *TrGeT: Traces Generator Tool*, desenvolvido em rede de Petri colorida (RPC) hierárquica temporizada por meio da ferramenta *CPN Tools* (CPN Tools, 2015). Assim como as ferramentas *VanetMobiSim* e MOVE, *TrGeT* gera *traces* de mobilidade para simulação de VANETs de forma rápida. Como diferencial, no *TrGeT* é possível alterar o comportamento do modelo ao longo da simulação, permitindo o acompanhamento passo a passo (forma *stepwise*) da geração de *traces* para estudos e análises específicas. O *TrGeT* pode ser estendido e modificado por meio de mudanças no modelo RPC. Embora tenha sido pensado para gerar *traces* para simulação de VANETs, os *traces* gerados podem ser utilizados para análise de tráfego, assim como SUMO, como é demostrado no Capítulo 4.

No *TrGeT* o usuário pode visualizar o comportamento do modelo pela interação com a ferramenta e pode também alterar parâmetros do sistema durante a simulação. O modelo de mobilidade adotado foi o IDM, dado que o mesmo é um dos modelos mais utilizados em simulações de tráfego (Harri, Filali, and Bonnet, 2009), mas é possível realizar mudanças no modelo de mobilidade utilizado e estender a ferramenta com a adição de novos modelos, sem a necessidade de alterar os demais módulos do modelo. Uma visão geral do modelo é apresentada na Figura 6.

Figura 6 – Visão geral do modelo



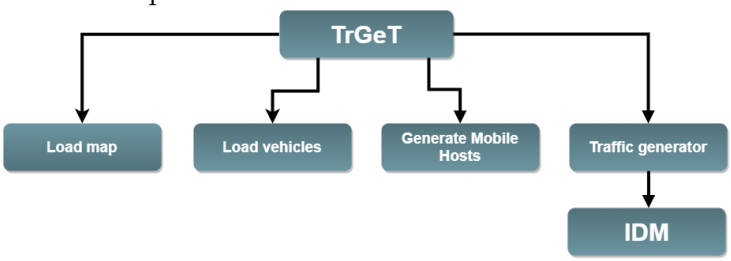
O modelo RPC implementa a lógica da simulação e, após a finalização da mesma, gera arquivos de texto no formato CSV (*Comma-separated values*) que servem de entrada para um *parser* que transforma o arquivo gerado pelo *CPN Tools* em *traces* que podem ser lidos por simuladores de rede.

3.1 Modelo RPC

O modelo *TrGeT* foi desenvolvido em RPC hierárquicas temporizadas por meio da ferramenta *CPN Tools* (CPN Tools, 2015). Redes hierárquicas permitem a construção de modelos com múltiplos módulos organizados em diferentes níveis de abstração, como em linguagens de programação. A hierarquização é realizada em RPC, substituindo-se uma transição de substituição por uma sub-rede. Uma RPC temporizada possui, além das cores das fichas, um valor chamado de "selo de tempo" e um "relógio global". Isso permite que sistemas possam ser modelados e avaliados levando em consideração restrições de tempo. Mais informações sobre as RPC hierárquicas pode ser encontrada no APÊNDICE A – REDES DE PETRI COLORIDAS, assim como a definição formal das RPC hierárquicas temporizadas pode ser encontrada em Jensen and Kristensen (2009).

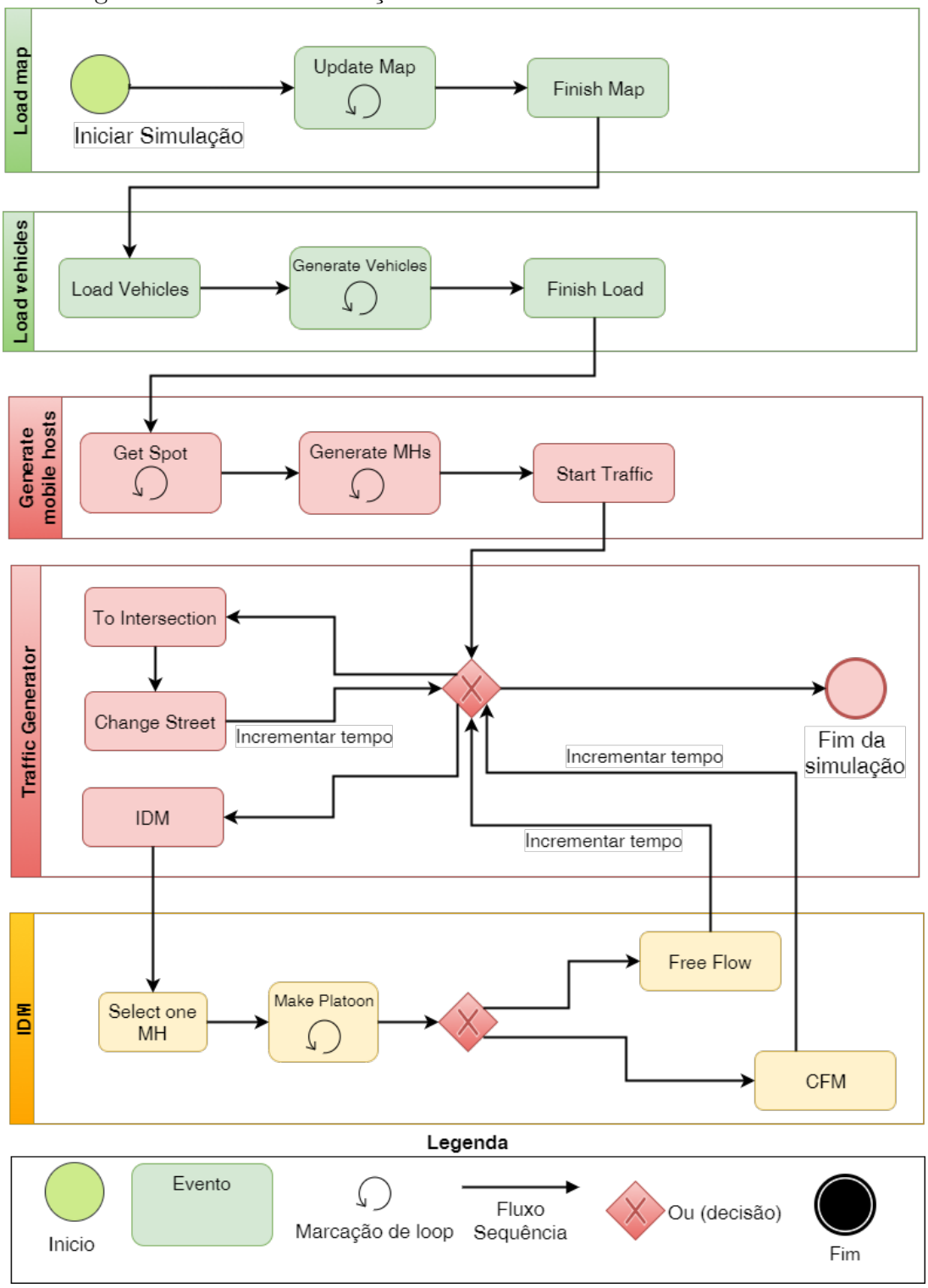
O modelo RPC possui cinco módulos assim denominados: *Load map*, *Load vehicles*, *Motion constraints*, *Traffic generator* e *IDM*, sendo este último correspondente à parte do modelo que pode ser estendida por meio da adição (ou substituição) de novos CFMs no mesmo nível hierárquico do submódulo *IDM*. A hierarquia é apresentada na Figura 7.

Figura 7 – Hierarquia do modelo



O fluxo de execução do modelo apresentado na Figura 8 ilustra, em um nível de abstração mais elevado, a sequência de atividades realizadas durante a execução de uma simulação. O processo é iniciado dentro da página *Load map*, em que a transição *Update map* executa várias vezes até que todo o mapa seja processado e esteja pronto para o uso. A página *Load Vehicles* trata da geração dos veículos tendo como entrada as configurações dos grupos de veículos. Cada grupo de veículos é transformado por meio da transição *Generate Vehicles* em uma frota de carros seguindo as características de cada grupo. Segue-se para a página *Generate mobile hosts*, em que cada veículo será alocado dentro do mapa da simulação por meio de dois passos: *Get spot* executa várias vezes até que seja gerado um conjunto de possíveis localizações; em *Generate MHs* cada veículo é alocado em uma localização e é preparado para iniciar a simulação. Na página *Traffic Generator* a simulação chega a um ponto de decisão onde pode-se realizar três uma de três ações, dependendo do que ocorrer em um passo de simulação, que representa 1 segundo me tempo real:

Figura 8 – Fluxo de execução do modelo



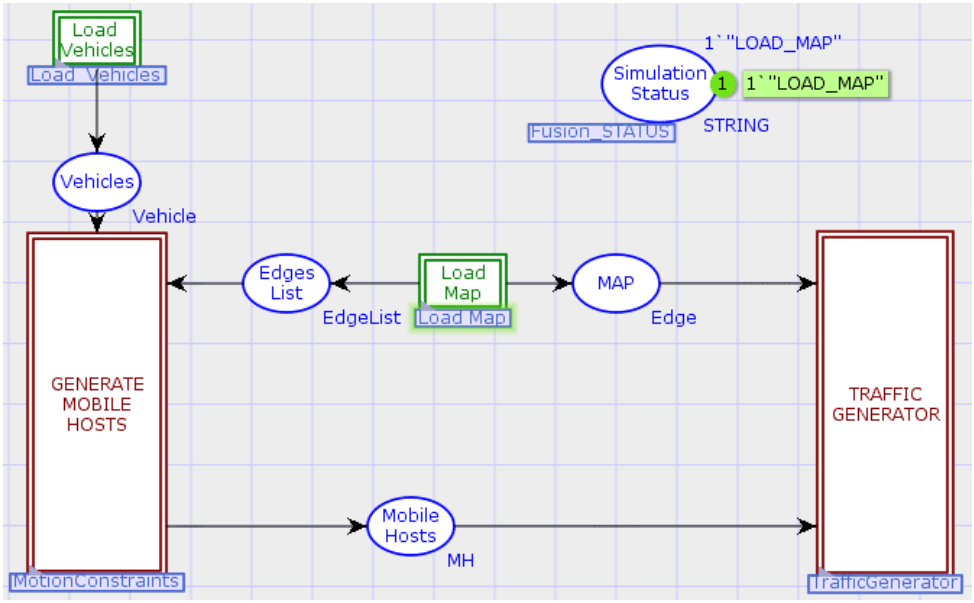
- O veículo alcança um cruzamento e deve mudar de rua. O tempo é incrementado e volta-se ao ponto de decisão;
- Longe de um cruzamento, o veículo entra na página IDM e deve obedecer ao algoritmo *car fallowing model* IDM. Para tal, é montada uma lista com todos os veículos que estão na mesma rua e faixa do veículo em questão e, se for o veículo principal

(a frente de todos), o mesmo acelera para tentar chegar a velocidade desejada (*Free flow*) Se o veículo está seguindo um outro, o algorítimo CFM IDM calcula a velocidade que o veículo deve apresentar no próximo passo de simulação. Após *Free flow* ou *CFM* o tempo é incrementado e a simulação retorna ao ponto de decisão da página *Traffic Generator*;

- Se o tempo alcançar o tempo desejado de simulação, a mesma é finalizada.

Em RPC, uma sub-rede também é denominada página, acomodando-se, portanto, cada módulo do simulador em uma página. A página *TrGeT* apresentada na Figura 9 é a de maior nível hierárquico, correspondendo à página principal do modelo. Nela podem-se observar quatro transições de substituição (TS) representadas por retângulos de borda dupla.

Figura 9 – Página principal



A TS *Load map* recebe um arquivo de texto contendo o mapa da área que é utilizada na simulação, e retorna, por meio do arco de saída, após executada a TS, uma lista contendo todas as ruas que pertencem ao mapa. A TS *Load vehicles* recebe outro arquivo de texto contendo os grupos de veículos que serão utilizados na simulação, e retorna, por meio do arco de saída, após executada a TS, um conjunto de veículos gerados de acordo com as especificações de cada grupo. Em um grupo é possível especificar as seguintes características: quantidade de veículos, tamanho, aceleração, desaceleração e velocidade máxima de cada veículo. A TS GENERATE MOBILE HOSTS recebe, por meio do arco de entrada, a lista das ruas que compõem o mapa e o conjunto de veículos gerados anteriormente, e retorna, por meio do arco de saída, após executada a TS, um conjunto de *Mobile Hosts* (MHs), cada um com sua posição inicial, velocidade inicial zero e velocidade desejada escolhida aleatoriamente dentro das faixas definidas por meio de parâmetros de entrada. A TS TRAFFIC GENERATOR recebe pelos arcos de entrada o

mapa e os MHs gerados anteriormente.

A estrutura e o comportamento das quatro transições de substituição da página principal são descritas nas sub-redes *Load map*, *Load vehicles*, *Motion Constraints* e *Traffic Generator*. A página *Load map* é responsável pela geração inicial do mapa da área a ser simulada. A página *Load vehicles* é responsável pela geração dos veículos. A página *Motion Constraints* é responsável pela geração dos MHs e por direcioná-los ao gerador de tráfego. A página *Traffic Generator* recebe os MHs e verifica se os mesmos estão próximos o suficiente de um cruzamento para determinar se mudam de via ou se continuam a se locomover na mesma via, obedecendo a um algoritmo CFM. No modelo *TrGeT*, o algoritmo escolhido foi o IDM, modelado na página de mesmo nome.

As declarações dos conjuntos de cores utilizados no modelo *TrGeT* estão listadas na Tabela 2.

Tabela 2 – Conjuntos de cores declarados

Conjunto de cores	**Declaração**
ID	INTINF
Direction	int with 1..2
Point	record x:X*y:Y
Spot	record edge:Edge*p:Point*dir:Direction*v:Vehicle
Spots	list Spot
Vertex	record id:ID*x:X*y:Y*tp:vType*return:BOOL
Edge	record id:ID*v1:Vertex*v2:Vertex*way:Direction*n:REAL*vmax:REAL
EdgeList	list Edge
InUse	UNIT
VehEntry	record n:INT*l:REAL*a:REAL*b:REAL*vmax:REAL
Vehicle	record id:ID*l:REAL*a:REAL*b:REAL*vmax:REAL
VehList	list Vehicle
vType	with join |inductor |traffic |intersection
MH	record id:ID*pos:Point*ed:ID*dir:Direction*v:REAL*toVertex : REAL*v_desired:REAL*l:REAL*a:REAL*b:REAL*vmax:REAL timed
Platoon	list MH
Step	record id:ID*p:Point*v:REAL timed

O conjunto de cores *ID* é utilizado para identificar de forma única alguns elementos presentes na simulação, porém não especifica diretamente o conjunto de cores de nenhum dos lugares da rede, mas apenas para compor outros conjuntos de cores como *Vertex*, *Edge*, *Vehicle* e *MH*. O tipo INTINF é similar ao INTEGER, porém sem limite de tamanho. O conjunto de cores *Direction* representa o sentido que uma faixa de transito pode ter, sendo limitado a 1 (fluxo crescente no plano cartesiano) e 2 (fluxo decrescente no plano cartesiano). O conjunto de cores *Direction* não é associado a nenhum lugar da rede, mas é utilizado para compor outros tipos de conjuntos de cores como *Spot*, *Edge* e *MH*. O conjunto de cores *Point* representa um ponto no plano cartesiano, uma coordenada

(x, y). O conjunto de cores *Point* é utilizado para compor outros conjuntos de cores como *Spot*, *MH* e *Step*, não sendo associado a nenhum lugar da rede. O conjunto de cores *Spot* representa a localização em que um MH pode estar, incluindo a faixa. O conjunto *Spot* é utilizado apenas na geração de uma localização aleatória dentro do mapa, indicando onde os MHs serão posicionados no inicio da simulação.

O mapa do cenário modelado é representado por meio do lugar *MAP*, associado ao conjunto de cores *Edge* na página principal. O conjunto *Edge* é a representação de uma rua. Para modelar o mapa utilizamos dois conjuntos de cores: *Vertex* e *Edge*. O *Vertex* pode representar um dos três elementos do conjunto de cores *vType*: *inductor*, representando um *loop* indutor que captura dados referentes aos veículos que passam por ele, *intersection*, representando um cruzamento entre ruas e, *traffic*, representando uma ligação entre duas ruas sem um cruzamento. O *traffic* é utilizado quando há necessidade de representarmos uma curva em uma rua. Para tal, temos que unir vários seguimentos de retas com ângulos diferentes. O *Vertex* é representado por uma coordenada (x, y), o tipo de elemento o qual representa, um determinado *ID* e uma indicação (*return*) que indica se é permitido que o veículo retorne pela mesma rua ao alcançar o *Vertex*, simulando uma rotatória ao final da rua.

Um *Edge* é o equivalente a uma reta no plano cartesiano, ligada por dois pontos, no caso dois *Vertex*. Um *Edge* possui uma *Direction* que indica a direção da faixa de trânsito da rua. Atualmente o modelo só permite ruas com uma faixa. O lugar *Mobile Hosts*, de cor *MH*, da página principal representa os veículos (*mobile hosts* - MH) presentes na simulação. Durante a simulação, um MH está em uma determinada rua com uma dada velocidade. O lugar MHs presente no submódulo *IDM* recebe fichas da cor *Platoon*, que é modelada como uma lista de MHs. O lugar é utilizado para tratar a posição em que um MH está em relação aos outros MHs na mesma rua e faixa que ele se encontra, de acordo com o algoritmo IDM (Treiber, Hennecke, and Helbing, 2000).

As funções utilizadas no modelo estão disponíveis no APÊNDICE C - FUNÇÕES SML. Um exemplo é a função ilustrada no Código 1.

Código 1: Função addMoviment

```
1  (* adds s sized movement to a MH at P(px,py) obeying the straight line beetween points v1(x1,y1) and
       v2(x2,y2) in a cartesian coordinate system *)
2  fun addMovement (px,py,x1,x2,y1,y2,s,n,1) =
3  let
4  val dasc = (Math.sqrt(Math.pow((px-x1),2.0)+ Math.pow((py-y1),2.0))) + s;
5  val desc = (Math.sqrt(Math.pow((px-x1),2.0)+ Math.pow((y1-py),2.0))) + s;
6  in
7  if Real.== (x1,x2) then {x=px, y=py+s}
8  else if Real.==(y2,y1) then {x=px+s,y=py}
9  else if y1<y2 then {x=((dasc/(n))*(x2-x1))+x1 , y=((dasc/n)*(y2-y1))+y1}(* Asc *)
10 else {x=((desc/(n))*(x2-x1)) + x1, y= ~((desc/n)*(y1-y2)) + y1} (* Down *)
11 end
12 | addMovement (px,py,x1,x2,y1,y2,s,n,2) =
13 let
```

```
14 val dasc = (Math.sqrt(Math.pow((x2-px),2.0)+ Math.pow((y2-py),2.0))) + s;
15 val desc = (Math.sqrt(Math.pow((x2-px),2.0)+ Math.pow((py-y2),2.0))) + s;
16 in
17 if Real.== (x1,x2) then {x=x1, y=py-s}
18 else if Real.==(y2,y1) then {x=px-s,y=y1}
19 else if y1<y2 then {x= ~((dasc/(n))*(x2-x1))+x2 ,y= ~((dasc/n)*(y2-y1))+y2}(* Asc *)
20 else {x= ~((desc/(n))*(x2-x1)) +x2, y= ((desc/n)*(y1-y2)) + y2} (* Down *)
21 end;
```

A função recebe um ponto $P(px, py)$ que representa a localização do MH, a rua, representada por meio da reta que passa pelos pontos $v1(x1, y1)$ e $v2(x2, y2)$, o deslocamento s que será realizado, a distância euclidiana n entre os pontos $v1$ e $v2$ (tamanho da rua) e a faixa em que o MH se encontra. A função adiciona a distância n ao ponto P em que o MH se encontra, e de acordo com a faixa da rua em que se encontra. Foi utilizado o sentido de circulação pela direita (mão francesa) em que os veículos trafegam pela faixa da direita, que no caso do modelo *TrGeT* é a *Direction* 1. Os MH que trafegam pela *Direction* 1 se movem no sentido crescente do eixo x (abscissa) e y (ordenada), enquanto os que trafegam na *Direction 2* se movem no sentido decrescente dos eixos x e y. As coordenadas dos pontos são limitadas ao 1º quadrante do plano cartesiano.

3.2 Parsers

Os *parsers* foram implementados em *Python* e transformam a saída da simulação RPC (monitores gerados no *CPN Tools*) em *traces* de mobilidade que podem ser lidos pelos simuladores de rede NS2 e NS3.

Um monitor é um mecanismo da ferramenta *CPN Tools* que é usado para observar, inspecionar, controlar ou modificar uma simulação de uma RPC. Não existe limite para a adição de monitores em uma RPC. Os monitores podem inspecionar as marcações de lugares e os elementos de ligação (*bindings*) que ocorrem durante uma simulação, podendo tomar ações apropriadas com base nas observações. Dentre as ações possíveis de se realizar por meio de monitores podem-se citar:

- Parar uma simulação quando um lugar específico estiver vazio;
- Contar o número de vezes que uma transição dispara;
- Atualizar um arquivo quando uma transição ocorre com uma variável ligada a um valor específico;
- Calcular o número médio de fichas em um lugar.

No modelo *TrGeT* foram criados monitores do tipo *Write-in-file* que coletam dados da simulação e gravam em um arquivo no formato CSV de acordo com a Tabela 3.

Além de gerar os arquivos de *traces* por meio dos *parsers*, os monitores possibilitam o levantamento dos dados necessários à análise de desempenho do modelo, como descrito no Capítulo 4. O código fonte dos *parsers* consta no APÊNDICE D - *PARSERS*.

Tabela 3 – Monitores *CPN Tools* - *TrGeT*

Monitor	**Descrição**	**Formato**
changeStreet	Guarda informações dos MHs que passam por um *inductor*	MH, t, ID_Vertex, x, y, v
*initial*_pos	Guarda posição inicial dos MHs	X, Y
FreeFlow	Guarda informações relativas aos passos de simulação dos MHs que estão em *Free Flow*	MH, x, y, v, t
CFM	Guarda informações relativas aos passos de simulação dos MHs que estão *Car Fallowing*	MH, x, y, v, t

3.3 Simulação

Para realizar uma simulação no modelo *TrGeT* é necessário configurar *a priori* a lista de parâmetros indicados na Tabela 4. Os valores configurados para os cenários de simulação do modelo são descritos no Capítulo 4 . O formato e unidade dos parâmetros estão listados na terceira coluna desta tabela.

Tabela 4 – Parâmetros da simulação

Parâmetro	**Descrição**	**Unidade**
$t : int$	Duração da simulação	Segundos (s)
$nm : Edge$	Mapa da simulação	Carregado por meio de arquivo txt com formato da cor *Edge*
$vehs : VehEntry$	Grupo de veículos	Carregado por meio de arquivo txt com formato da cor *VehEntry*
$T : REAL$	Tempo de segurança entre os MHs	Segundos (s)
$S_0 : REAL$	Distância mínima entre os MHs	Metros (m)
$v\_min : REAL$	Velocidade mínima	Metros por segundo (m/s)
$delta : REAL$	Expoente de aceleração δ	Metros por segundo ao quadrado (m/s^2)
$kappa : REAL$	Fator de desaceleração κ	Metros por segundo ao quadrado (m/s^2)

Os parâmetros *t*, *nm* e *vehs* são inerentes à simulação. Os demais parâmetros são inerentes ao modelo IDM. Mais detalhes sobre a escolha dos valores para os parâmetros IDM e como estes interagem entre si podem ser encontrados em Kesting, Treiber, and Helbing (2010).

O parâmetro *nm* configura o mapa do local que será simulado. As coordenadas das ruas e dos cruzamentos dos mapas podem ser obtidos por meio de serviços de mapa como o **OpenStreetMap** (Open Street Map, 2017). Porém, é necessário um tratamento posterior do arquivo gerado nesses serviços para transformá-lo em marcações de fichas que podem ser lidas pelo *CPN Tools*. O parâmetro *vehs* configura os grupos de veículos que serão transformados em MHs na simulação. Para configurar os parâmetros dos grupos de

veículos foi utilizada como referência a tabela de parâmetros padrões do veículo, retirado do sítio do SUMO (SUMO, 2016).

3.3.1Página principal: coordenação da simulação no TrGeT

A coordenação de uma simulação é realizada pela execução sequencial de quatro blocos. O primeiro é representado pela *TS Load Map*. Na Figura 9 pode-se ver a página principal no estado inicial da RPC, na qual apenas a TS *Load Map* está habilitada, o que é garantido pela disponibilidade de 1 ficha do tipo STRING no lugar *Simulation Status* que pertence a um conjunto de lugares de fusão com nome *Fusion_STATUS*. O lugar de fusão *Fusion_STATUS* garante que a simulação siga a seguinte lógica de execução: *LOAD_MAP* (carrega o mapa), *LOAD_VEH* (gera os veículos), *LOAD_MOTION* (cria os MHs).

Após a execução da TS *Load Map*, o lugar *Edges List* será alimentado por fichas da cor *EdgeList*, o lugar MAP receberá fichas da cor *Edge* e é alterado o valor dos lugares pertencentes a *Fusion_STATUS* para *"LOAD_MAP"* habilitando a TS *Load Vehicles*. Após a execução da TS *Load Vehicles*, o lugar *Vehicles* será alimentado por fichas do tipo *Vehicle* e é alterado o valor das fichas nos lugares pertencentes a *Fusion_STATUS* para *"LOAD_VEH"*. As fichas criadas no lugar *Vehicle* e *EdgesList*, junto ao *Fusion_STATUS* em *"LOAD_VEH"*, habilitam a TS *GENERATE MOBILE HOSTS*.

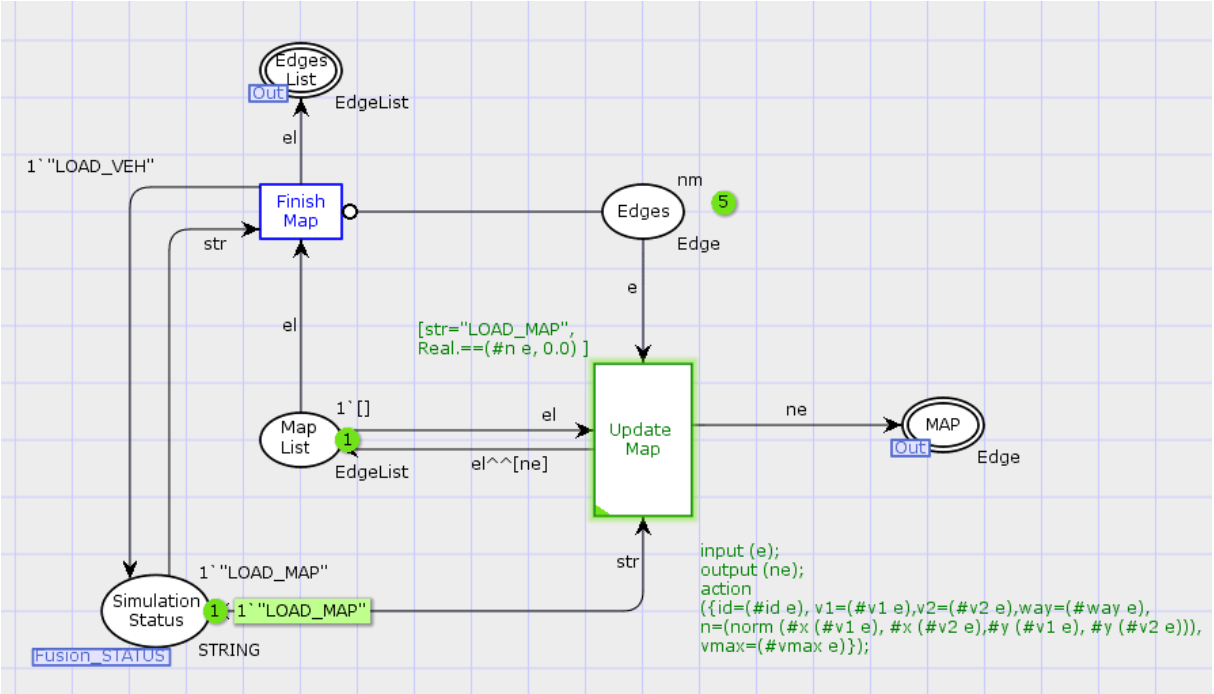
Após a execução da *TS GENERATE MOBILE HOSTS*, o lugar *Mobile Hosts* será alimentado por fichas do tipo MH, habilitando a *TS TRAFFIC GENERATOR*. Portanto, este bloco recebe como valores de entrada tanto o mapa (fichas do lugar *MAP*) como os MHs gerados no bloco *GENERATE MOBILE HOSTS* (fichas do lugar *Mobile Hosts*). Os lugares *Vehicles*, *Edges List*, *MAP* e *Moobile Hosts* da página principal do *TrGeT* correspondem a *sockets de RPC* que podem ser acessados nas sub-páginas pelas suas respectivas *ports*.

3.3.2Página Load Map: Geração do mapa da simulação

A página *Load Map* recebe em um arquivo de texto no lugar *Edges* o mapa da simulação. Apenas a transição *Update Map* estará habilitada na marcação inicial. Ela irá disparar até que todas as fichas contidas no lugar *Edges* sejam retiradas e, consequentemente, as distâncias euclidianas n das fichas contidas no lugar *Edges* sejam calculadas. A transição *Update Map* retira do lugar *Edges* por meio de um arco de entrada uma ficha da cor *Edge* representando uma rua no mapa e, do lugar *Map List* uma ficha da cor *EdgeList* que contém uma lista, no início vazia, com as ruas. O uso de cores do tipo lista em alguns lugares é devido a impossibilidade de retirar mais de uma ficha em um disparo de transição, não permitindo que se trabalhe com um conjunto de valores presente em uma marcação de um lugar. Um exemplo da utilização de listas é o mapa que em algumas funções deve ser passado por completo para efetuar um cálculo. A ficha do tipo *Edge* é

passada na chamada da função *norm* que calcula a distância euclidiana, representando o comprimento da rua. Após o disparo da transição *Update Map* uma ficha do tipo *Edge* é depositada no lugar *MAP* (que representa uma porta de saída, *out port*) e uma ficha contendo a lista com o mapa (*EdgeList*) é atualizada e devolvida para o lugar *Map List*. Após todas as fichas do lugar *Edges* serem processadas, a transição *Update Map* fica desabilitada. Então, a transição *Finish Map* se torna habilitada pela presença de um arco inibidor que liga a transição ao lugar *Edges*, agora vazio. No disparo da transição *Finish Map* a ficha contida no lugar *Map List* é enviada ao lugar *EdgesList*, que representa uma porta de saída (*out port*), e o lugar *Simulation Status* que é um lugar de fusão (*Fusion_STATUS*) recebe uma ficha com o conteúdo *"LOAD_VEH"*. A página *Load Map* é apresentada na Figura 10.

Figura 10 – Página *Load Map*.

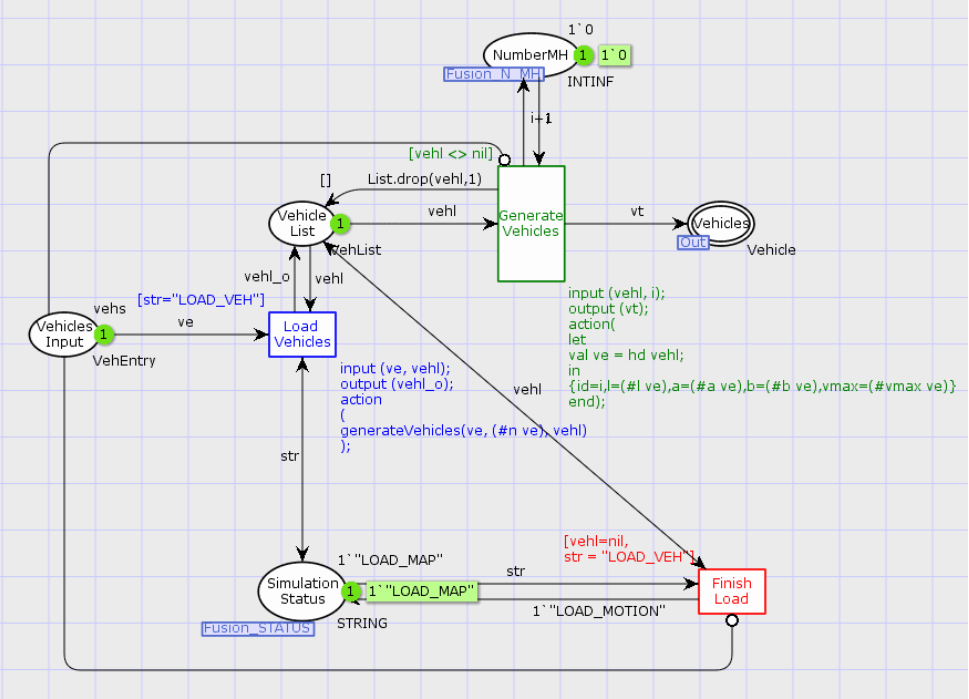


3.3.3Página Load Vehicles: geração dos veículos

A página *Load Vehicles* recebe em um arquivo de texto no lugar *Vehicles Input* as fichas da cor *VehEntry* que representam os grupos (ou classes) de veículos que serão transformados em veículos. Apenas a transição *Load Vehicles* está habilitada pela presença de uma ficha no lugar *Vehicles Input* e uma ficha com uma lista vazia no lugar *Vehicle List*. A página é apresentada na Figura 11.

No disparo da transição é chamada a função *generateVehicles*, passando como parâmetro a ficha recebida pelo arco de entrada do lugar *Vehicles Input* e a ficha recebida

Figura 11 – Página *Load Vehicles*



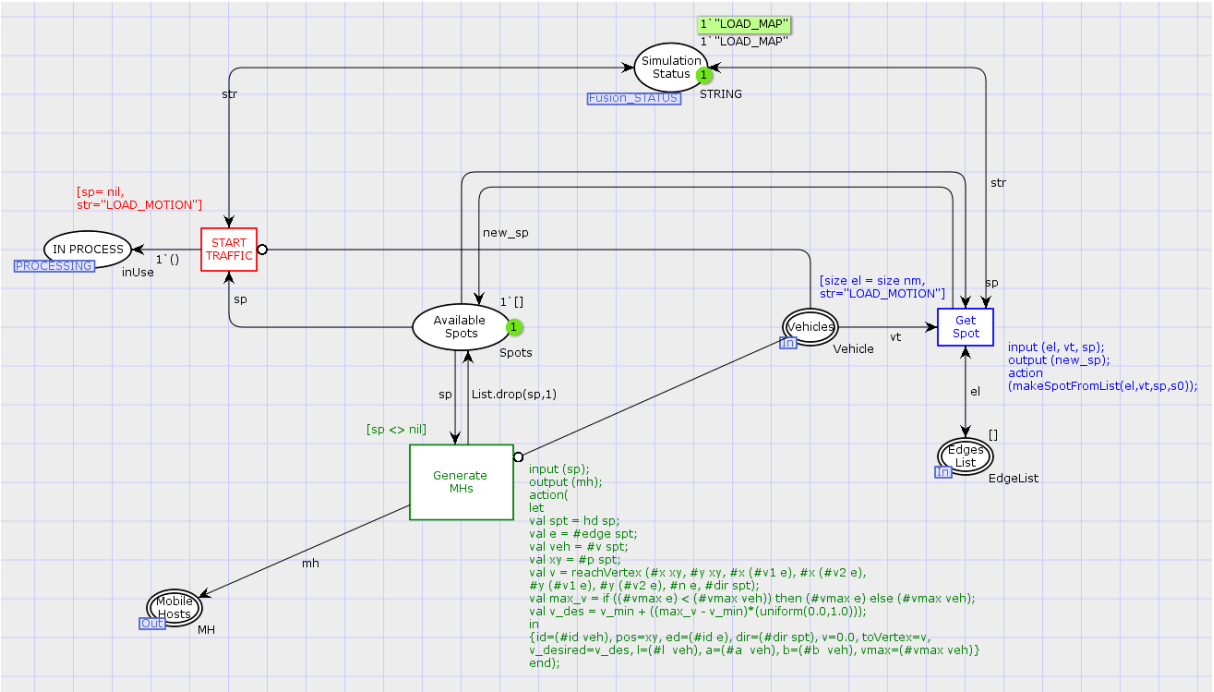
do lugar *Vehicle List* contendo a lista de veículos. Após o disparo da transição *Load Vehicles*, uma ficha da cor *VehList* é depositada no lugar *Vehicle List* contendo os veículos gerados. A transição, então, torna-se desabilitada por não haver mais fichas no arco de entrada do lugar *Vehicles Input*. A transição *Generate Vehicles* torna-se habilitada quando a lista de veículos que é recebida por meio do arco de entrada do lugar *Vehicle List* é diferente de vazia (*vehl <> nill*). A transição recebe por meio do arco de entrada do lugar *Vehicle List* a lista de veículos e do lugar de fusão *NumberMH* (*Fusion_N_MH*), uma ficha da cor INTINF que representa um contador do número de veículos na simulação. O contador será utilizado na Página IDM para a realização de cálculos. No disparo da transição *Generate Vehicles* é retirado o primeiro veículo da lista, a lista atualizada é devolvida, uma ficha da cor *Vehicle* é deposita no lugar *Vehicles* (*out port*) e o contador do lugar *NumberMH* é incrementado de um. A transição dispara até que a lista de veículos fique vazia, o que torna a transição *Finish Load* habilitada. O disparo da transição *Finish Load* altera o valor da ficha no lugar *Simulation Status* (*Fusion_STATUS*) para *"LOAD_MOTION"*. Quanto isto acontece, não há mais transições habilitadas na página *Load Vehicles*.

3.3.4Página Motion Constraints: geração e posicionamento inicial de MHs no mapa

A página *Motion Constraints* apresentada na Figura 12 recebe, pela porta de entrada (*port in*) do lugar *Edges List*, a lista contendo o mapa das ruas, e pela porta de entrada do lugar *Vehicles* os veículos que participarão da simulação. Apenas a transição *Get Spot*

estará habilitada com as fichas geradas pelos blocos *Load Map* e *Load Vehicles*.

Figura 12 – Página *Motion Constraints*



O lugar *Available Spots* possui uma ficha da cor *Spots*, que representa uma lista de *Spot*. O conjunto de cor *Spot* representa a localização aleatória dentro do mapa em que um MH pode estar no inicio da simulação. A cada disparo da transição *Get Spot* é retirada uma ficha do lugar *Vehicles* e uma ficha do lugar *Edges List* com o mapa e é executada a função *makeSpotsFromList* que posiciona um veículo em um lugar único do mapa, adicionando essa posição na lista por meio da alteração da marcação do lugar *Available Spots*. Ao final desse processo, a marcação do lugar terá uma lista de todos os lugares disponíveis (*Spot*) para a alocação dos MHs.

Após a retirada de todas as fichas do lugar *Vehicles*, a transição *Generate MHS* é habilitada. Ela é responsável por inserir um veículo na posição já definida na marcação da ficha retirada do lugar *Available Spots* na simulação com uma velocidade desejada para a primeira viagem. A velocidade desejada é calculada aleatoriamente tendo como limite inferior o parâmetro da simulação velocidade mínima (*v_min*) e limite superior à velocidade máxima da rua. Se a velocidade máxima do veículo é inferior a velocidade máxima da via, a velocidade máxima do veículo é o limite superior. O disparo da transição gera um veículo (MH) no lugar *Mobile Hosts* que funciona como uma porta de saída (*out port*). A transição permanece habilitada até que todos os veículos sejam posicionados com sua velocidade desejada escolhida.

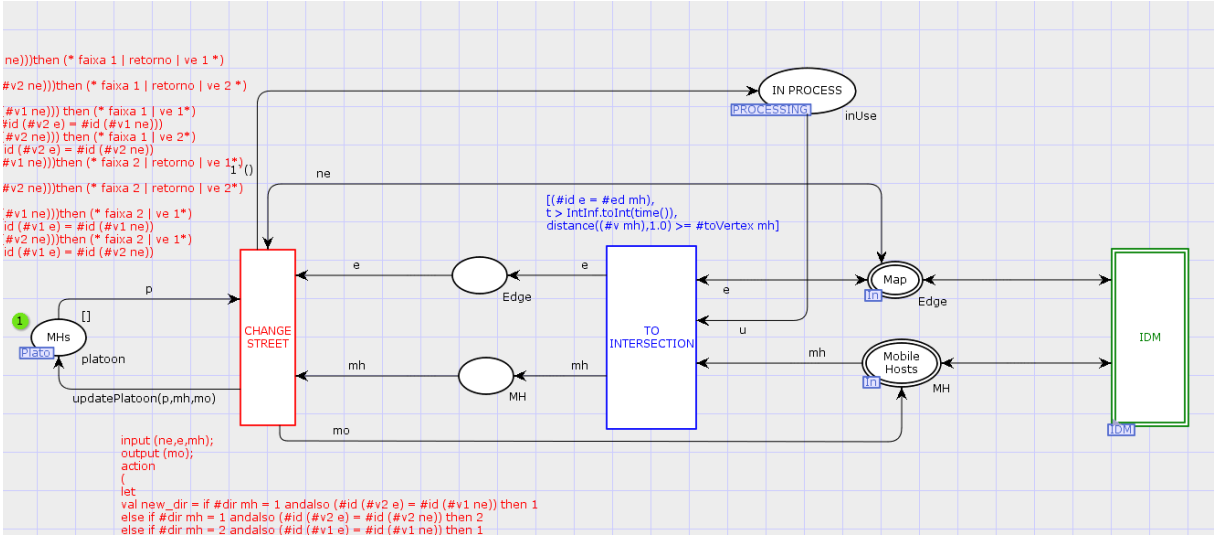
A última transição da página *Motion Constraints* é a *START TRAFFIC* que adiciona uma ficha no lugar de fusão *IN PROCESS* (*PROCESSING*), o que sinaliza que a simulação está completamente configurada e os veículos, agora MHs, podem iniciar a

movimentação.

3.3.5 Traffic Generator: gerador de movimentação dos veículos

As fichas chegam na página *Traffic Generator* por meio das portas de entrada (*in ports*) dos lugares *Map* e *Mobile Hosts* de acordo com a Figura 13. Esses lugares são interfaces com a página principal. A página *Traffic Generator* é responsável pela movimentação dos veículos pelas ruas, identificando se o mesmo deve permanecer na mesma ou chegou a um cruzamento.

Figura 13 – Página *Traffic Generator*



A página possui três transições: *TO INTERSECTION*, *CHANGE STREET* e *IDM*. De acordo com a velocidade que o MH se move é calculada a distância percorrida em $1s$, tempo que dura um passo de simulação (*step*). Se a distância a ser percorrida no passo for maior do que a distância para alcançar o fim da rua o veículo deve se preparar para chegar a um *vertex* que, como anteriormente exposto, pode representar um indutor (*inductor*), um cruzamento (*intersection*), ou uma junção de trechos da mesma rua (*traffic*), habilitando a transição *TO INTERSECTION*. Caso contrário, o veículo permanecerá na rua e habilitará a transição *IDM*, responsável pelo *Car Following Model IDM*.

A transição *TO INTERSECTION* seleciona um MH que está próximo ao final do seguimento de rua (*Edge*). O MH selecionado é direcionado por meio do arco de entrada para a transição *CHANGE STREET*. A transição recebe o MH e verifica a rua (*Edge*) e faixa (*Direction*) que o mesmo se encontra. Se o *Vertex* que será alcançado for do tipo *traffic*, o veículo irá para o próximo segmento de rua com a mesma velocidade, já que alcançou uma curva na rua. Se o tipo for *inductor*, o comportamento será similar ao de *traffic*. O MH irá para o próximo segmento de rua com a mesma velocidade, porém

o monitor *changeStreet* irá guardar informações de velocidade e tempo de simulação do MH.

Se o tipo for *intersection*, o MH chegou a um cruzamento e está apto a mudar de rua. Na função *changeStreet* é verificado quais ruas se ligam ao cruzamento que o MH se encontra e aleatoriamente é escolhida uma delas. Então, a marcação do MH é atualizada com sua nova localização, velocidade e tempo de simulação. Para modelar o comportamento real do condutor ao se deparar com um cruzamento, foi adicionada uma probabilidade de 50% para que o veículo pare (velocidade zero) ou continue sua viagem com a mesma velocidade. Se o MH parar, isto significa que, neste momento, encerra-se uma viagem, tendo tido como início o momento diretamente anterior que o veículo zerou sua velocidade e destino este ponto da viagem, onde o mesmo zera a velocidade e inicia uma nova.

3.3.6IDM: car following model

A página IDM controla a movimentação dos veículos que permanecem na mesma faixa e rua. É importante ressaltar que o *car following model* IDM pode ser substituído por outro modelo que implemente um algoritmo CFM. Para tal, basta que o código da função do segmento de código da transição *CFM* seja alterado de acordo com a nova dinâmica do modelo.

Na sub-página IDM temos as transições *SELECT ONE MH*, *MAKE PLATOON*, *FREE FLOW* e *CFM*. A transição *SELECT ONE MH* seleciona aleatoriamente um dos MH que estão aptos a se movimentar, de acordo com as restrições temporais inerentes às RPC e aos critérios já citados anteriormente que levaram o MH a necessidade de se mover na mesma rua e faixa. A transição *MAKE PLATOON* monta um pelotão (*PLATOON*) com os MHs que estão na mesma rua e faixa do MH selecionado. O pelotão só é montado completamente uma única vez durante a simulação. Após sua montagem inicial, a cada passo dado por um MH, a posição do mesmo é atualizada dentro do pelotão.

Se o MH estiver à frente do pelotão, ele estará livre para se mover e alcançar a velocidade desejada *v_desired*. Isso ocorre no disparo da transição *FREE FLOW* em que, de acordo com a velocidade atual do MH e a aceleração a (configurada de acordo com a classe do veículo), será calculada a distância percorrida, a nova posição e velocidade do MH.

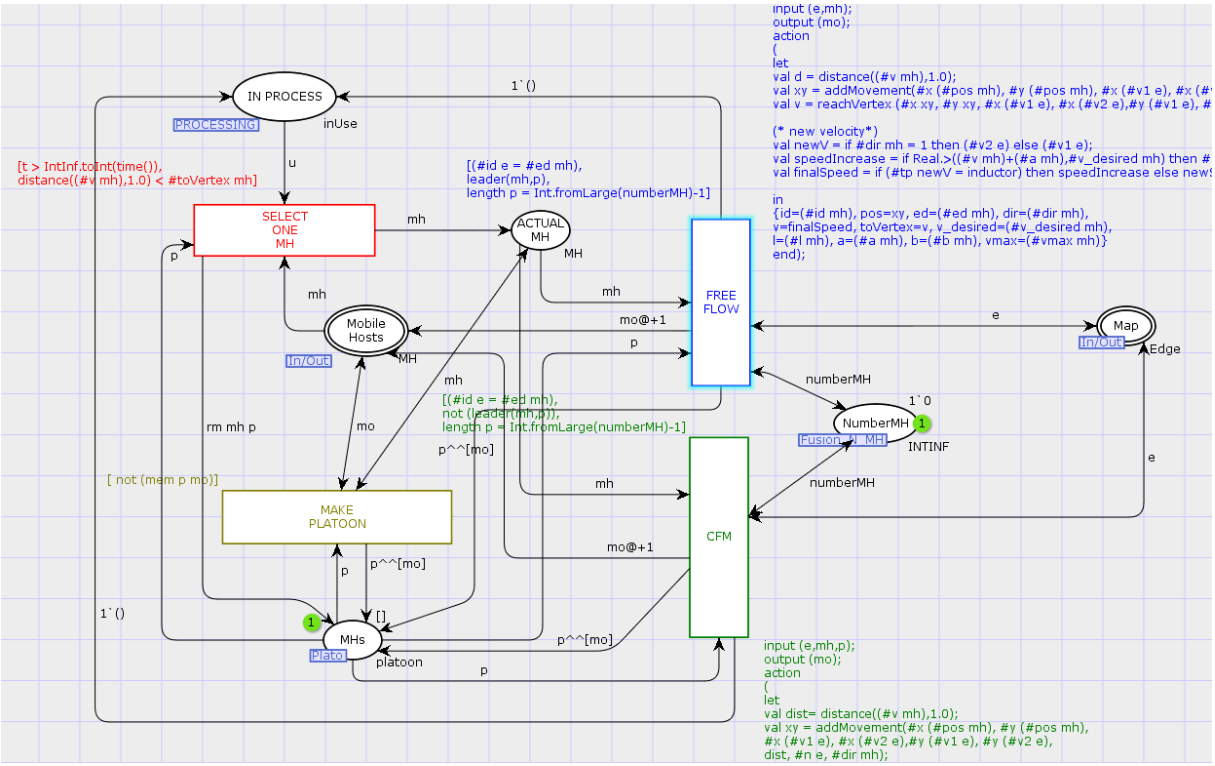
Caso o MH possua outro MH a sua frente, sua nova velocidade deve ser calculada de acordo com o algorítimo IDM. Isso ocorre no disparo da transição *CFM*. Antes do cálculo da nova velocidade é calculada a distância percorrida com a velocidade atual e sua nova localização. Após isso, será calculada a nova velocidade e atualizada a marcação do MH junto com a nova posição.

Após o disparo da transição *CFM* ou *FREE FLOW*, o lugar *Mobile Hosts*, pertencente às sub-páginas *Traffic Generator* e *TrGeT*, será atualizado e a simulação

seguirá até que todos os MHs se movam e alcancem o tempo de parada configurado por meio da constante t, que representa o tempo de simulação desejado em segundos.

A página do módulo IDM é apresentada na Figura 14.

Figura 14 – Página com o módulo IDM



3.4 Considerações Finais

Neste capítulo foi apresentado em detalhes como o modelo *TrGeT* foi desenvolvido na ferramenta *CPN Tools*. Para realizar uma simulação, além da parametrização inicial, é possível escolher a forma na qual a simulação será executada, que pode ser de duas formas: (i) *fast forward*, em que, como nas ferramentas convencionais, o usuário inicia a simulação e aguarda o resultado final; (ii) *stepwise*, na qual a simulação evolui passo a passo de acordo com a interação do usuário. Esta última forma permite a manipulação das variáveis envolvidas na simulação por meio da alteração do estado atual do modelo, possibilitando levar a simulação a situações específicas. É possível, por exemplo, elevar a velocidade de um veículo para além da permitida para verificar o comportamento do modelo ou alterar sua localização. Na forma *stepwise* é também possível verificar determinadas regras de um modelo sem a necessidade de iniciar uma nova simulação e aguardar até que a mesma alcance o estado desejado. Com isso, a modificação e, consequentemente, a evolução do modelo pode ser feita de forma mais rápida.

O modelo de mobilidade adotado foi o IDM, dado que o mesmo é um dos modelos mais utilizados em simulações de tráfego segundo Harri, Filali, and Bonnet (2009).

Porém, como mencionado anteriormente, é possível realizar mudanças no modelo de mobilidade utilizado e estender a ferramenta com a adição de novos modelos.

No próximo capítulo é feita a simulação do modelo, iniciando com a apresentação dos cenários de simulação e, seguindo com a discussão dos resultados.

4 SIMULAÇÕES E RESULTADOS

Neste capítulo, descrevemos os dois cenários e resultados das simulações realizadas para a validação do modelo *TrGeT*.

O primeiro cenário foi concebido para a validação dos *traces* gerados comparando-os aos resultados esperados de acordo com a teoria de tráfego (Gartner *et al.*, 2001), tendo como objetivo demostrar que o modelo apresenta resultados similares aos encontrados na literatura e aos obtidos de *traces* reais disponíveis publicamente. Neste cenário, tenta-se analisar não apenas as semelhanças, mas também as diferenças entre os dados encontrados em cenários reais e os obtidos por meio de simulação e, ao mesmo tempo, procura-se na literatura bases que sustentem estas diferenças.

O segundo cenário realiza simulações de uma VANET no simulador de rede NS-2 utilizando os *traces* de mobilidade gerados pelo modelo *TrGeT* e os traces gerados pela ferramenta *VanetMobiSim*. Em seguida é feita uma comparação dos resultados gerados pelos dois simuladores de traces. Os arquivos gerados para a análise de desempenho se encontram no repositório GitHub [2].

Um breve resumo da teoria de tráfego utilizada na análise de desempenho é encontrado no APÊNDICE B - TEORIA DO FLUXO DE TRÁFEGO. Todos os gráficos apresentados neste capítulo são de autoria própria e os *scripts* utilizados para a geração dos mesmos foram desenvolvidos na linguagem *Python* com a utilização da biblioteca *Matplotlib*.

4.1 Cenário 1

Para o Cenário 1, que mostra dados retirados do conjunto de dados (*data set* - DS) *The Portland State University Multimodal Test Data Set* (Portland State University, 2012), publicado na Internet, foram realizadas as seguintes ações: (i) elaboração de um gráfico contendo a variação do fluxo, densidade e velocidade média durante um dia em uma autoestrada dos Estados Unidos da América (EUA); (ii) realização de uma simulação utilizando o mapa do DS e gera um gráfico com as mesmas métricas de (i); (iii) comparação dos resultados obtidos pelo DS e pela simulação.

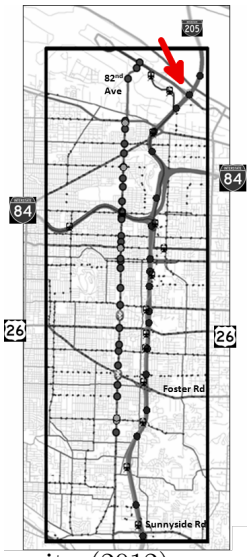
O DS inclui dados de autoestradas, vias arteriais e trânsito para o corredor I-205 em Portland, Oregon (EUA). O corredor possui aproximadamente 16 km de comprimento. Entre outras informações, o DS contém os dados de detectores de *loop* indutores utilizados nesta análise de desempenho. Os detectores são posicionados de acordo com os círculos escuros do mapa e descrito nos arquivos contidos no DS. Os dados foram coletadas ao longo de dois meses (de 15 de setembro a 15 de novembro de 2011) para uso em testes de aplicações envolvendo veículos conectados. O arquivo utilizado foi o

[2] https://github.com/lcvasconcelos/trget

freeway_loopdata5min.csv que contém dados de medidas agregadas a cada 5 minutos. O arquivo foi selecionado levando-se em consideração que o mesmo contém quantidade de pontos de medidas suficiente para representar o dia. O detector escolhido foi o de ID 1385O, localizado na I-205 próximo ao *Airport Way* no dia 17/10/2011. O critério para a escolha da data e detector foi o de maior medida de fluxo obtida dentro do DS. Os *scripts* para manipulação do DS e geração dos gráficos se encontram no APÊNDICE E - PARSER I-205.

O mapa e as fontes de dados incluídas na apresentação do DS são ilustrados na Figura 15. O ponto indicado pela seta sinaliza o local exato do detector utilizado.

Figura 15 – I205, Portland, Oregon. EUA



Fonte: Portland State University (2012)

Na Tabela 5 é apresenta a parte utilizada do dicionário de dados do DS.

Tabela 5: Dicionário de dados do DS

Atributo	**Tipo**
detectorid	ID (chave) do detector
starttime	Hora de início do intervalo de hora coletado
volume	Volume total de veículos que viajaram pelo segmento durante o período de 5 minutos.
speed	Velocidade média ponderada (milhas por hora) dos 5 minutos

Neste cenário, cada simulação é realizada para uma quantidade fixa de veículos. Todos os veículos estão presentes desde o início (com uma posição e velocidade inicial) e permanecem na simulação até o final. Foram realizadas 10 simulações para cada quantidade de veículos, registrando-se a velocidade média e a quantidade de veículos que passam pelo sensor. A quantidade de veículos varia de 50 a 4250, sendo o incremento entre as simulações de 50 veículos. Inicia-se uma simulação com uma quantidade de veículos e os resultados são colhidos. Então, inicia-se uma nova simulação com a quantidade de veículos incrementada em 50, e assim sucessivamente. O aumento gradual na quantidade de veículos em cada simulação foi usado para representar a elevação da densidade de

veículos até que a via sature, não suportando mais a entrada de novos veículos. Esse aumento gradual gera um tráfego sincronizado em que a velocidade do líder do pelotão (veículos mais a frente em uma via) limita a velocidade dos demais. Para montar um cenário que represente o tráfego de uma via ao longo do dia é necessário um modelo de distribuição das chegadas dos veículos ao longo do tempo (ondas de chegadas). Em um cenário real, como o mostrado no DS, a chegada dos veículos ocorre de forma variada ao longo do tempo, apresentando ondas de pico em períodos de *rush* que acarretam um aumento da quantidade de entradas de veículos e, fora dos mesmos, uma diminuição. Em outros momentos, como as madrugadas, observa-se uma quantidade de entradas de veículos espaçadas. Para simular esse comportamento de ondas seria necessário um modelo que o represente, ficando esta tarefa relacionada como um trabalho futuro.

Os parâmetros utilizados na simulação do Cenário 1 estão descritos na Tabela 6.

Tabela 6: Parâmetros da simulação - Cenário 1

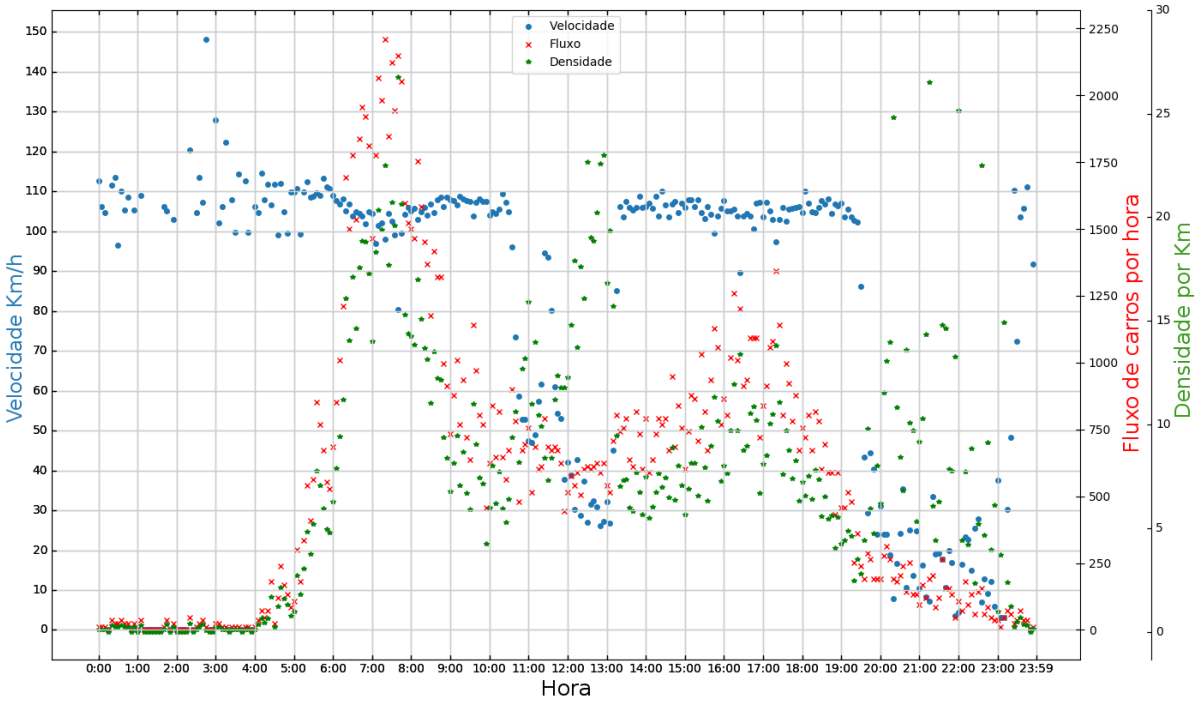
Parâmetro	**Descrição**	**Valor**
t	Duração da simulação	300 s
$VehEntry - n$	Número de MH	50-4250 (+50)
$VehEntry - l$	Tamanho do veículo	5,0 m
$VehEntry - a$	Aceleração	1,0 m/s^2
$VehEntry - b$	Desaceleração confortável	2,5 m/s^2
$VehEntry - v\_max$	Velocidade máxima	50,00 m/s
T	Tempo de segurança entre os MHs	0,5 s
S_0	Distância mínima entre os MHs	2,0 m
v_min	Velocidade mínima	15,00 m/s
$delta$	Expoente de aceleração δ	4,0 m/s^2
$kappa$	Fator de desaceleração κ	5,0 m/s^2

4.1.1Resultados: Cenário 1

O Gráfico 1 foi elaborado tendo como fonte os dados do DS coletados pelo detector de ID 1385O da autoestrada I-205 durante todo o dia 17/10/2011. Por meio do gráfico é mostrada a variação do fluxo, densidade e velocidade média da faixa da esquerda (alta velocidade).

Cada ponto visível no mapa representa o volume, velocidade e densidade, respectivamente, calculados por meio da medida agregada de 5 minutos. Observa-se que o gráfico gerado apresenta uma grande variação nos valores dos pontos coletados ao longo do dia. Entre 0 e 6 horas o fluxo e a densidade apresentam valores mínimos, enquanto a velocidade apresenta as maiores médias. Nesse horário há poucos veículos e eles trafegam em velocidades altas, sem a presença de congestionamentos. Este padrão representa a Fase 1 no diagrama de fases de tráfego, de acordo com Das and Levinson (2004) e explicado no APÊNDICE B - TEORIA DO FLUXO DE TRÁFEGO. Entre 6 e 9 horas observa-se um aumento do fluxo e da densidade acompanhados por uma pequena diminuição da

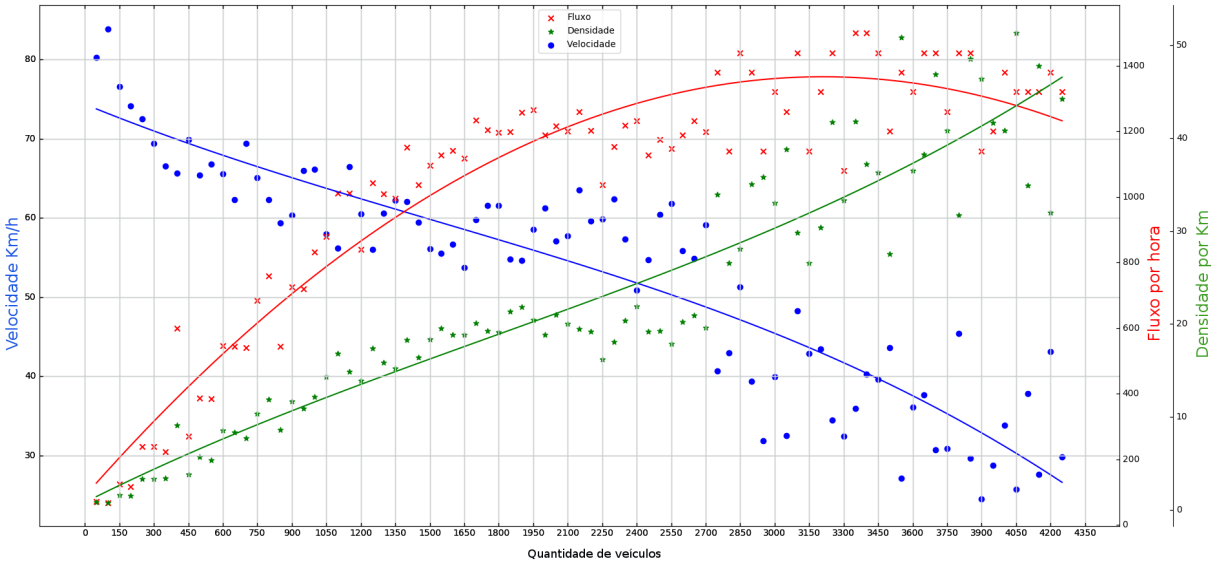
Gráfico 1 – I-205: Gráfico de Fluxo, densidade, velocidade média - I 205 - DS



velocidade, representando a Fase 2 onde a densidade aumenta, porém a introdução de novos veículos acarreta a diminuição da velocidade, mas mantendo o fluxo. Entre 10 e 13 horas, o tráfego entra na Fase 3, onde a densidade aumenta e as velocidades e o fluxo caem, gerando um congestionamento. Entre 13 e 18 horas o tráfego representa a Fase 4, uma fase de recuperação onde, a densidade começa a diminuir e a velocidade aumentar. Entre 18 e 23 horas o tráfego retorna à Fase 3, apresentando um novo congestionamento. Entre 23 horas e 23:59 a quantidade de veículos diminui, reduzindo o fluxo e a densidade e, aumentando as velocidades médias já que o condutor apresenta mais liberdade de acelerar.

O Gráfico 2 foi elaborado utilizando pontos coletados por meio da simulação do tráfego na autoestrada I-205. Diferentemente do Gráfico 1, o eixo x representa a quantidade de veículos presente em cada rodada de simulação e não as horas ao longo do dia. Isso ocorre pela ausência das ondas de chegadas no modelo *TrGet*, como dito anteriormente. Cada ponto visível no gráfico representa o volume, a velocidade média e a densidade calculada por meio da mediana das medidas médias agregadas de 60 segundos de simulação. Pela observação do gráfico podem-se verificar os padrões de mudanças de fases de tráfego descritos na literatura, porém, propositalmente de forma sequencial, partindo da Fase 1 para a 2 em seguida, desta para a Fase 3. Para facilitar a leitura do Gráfico, foi traçada uma linha de tendência para cada métrica medida utilizando o método dos mínimos quadrados. Não há um método exato para especificar os pontos

Gráfico 2 – I-205: Gráfico de Fluxo, densidade e velocidade média - I 205 - Simulação



de transição entre as fases, sendo os mesmos estabelecidos de forma empírica (Das and Levinson, 2004).

Iniciamos a Fase 1 com 50 veículos indo até aproximadamente 1.000 veículos. Nessa fase não há um congestionamento e os veículos se movimentam em fluxo livre com a média da velocidade variando de 110 km/h a 60 km/h. Tal variação de velocidade se dá pela natureza estocástica da simulação em que um veículo tem uma velocidade desejada entre a velocidade mínima (54 km/h) e a velocidade máxima (120 km/h). Nessa fase existem mais veículos com a velocidade próxima à máxima. A intensidade do fluxo cresce em forma de parábola.

A Fase 2 tem início em torno de 1.200 veículos e permanece até a proximidade da saturação da via, em torno de 3.150 veículos, não sendo mais possível manter a velocidade e o aumento do fluxo. Durante toda essa fase acompanhamos o estado de fluxo sincronizado em que os veículos se movimentam perto uns dos outros. No fluxo sincronizado, a velocidade média é consideravelmente menor do que a velocidade no fluxo livre. Nessa fase, as médias das velocidades variam de 70 km/h a 40 km/h, apresentando uma queda nos valores medidos. De acordo com Treiber, Hennecke, and Helbing (2000), no tráfego sincronizado, o fluxo pode ser mais ou menos homogêneo, mostrando diferentes oscilações em séries temporais nos dados coletados, exatamente como podemos ver no Gráfico 2 durante essa fase.

A Fase 3 tem início no ponto de fluxo máximo da simulação (q_{max}) em torno de 3.300 veículos e permanece até o fim da mesma por razão da impossibilidade de adicionar novos veículos sem que os mesmos desrespeitem o limite mínimo de espaço entre os veículos S_0 que é de 2 m. Nessa fase é evidente a diminuição do fluxo e das velocidades gerando um congestionamento. As velocidades médias iniciam uma descida vertiginosa, iniciando

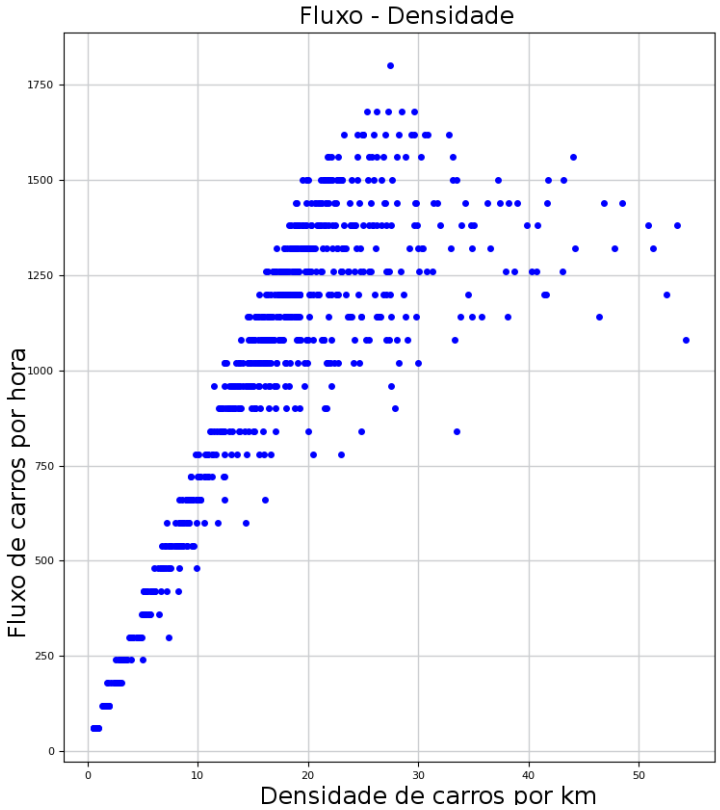
em 40 km/h finalizando a simulação em torno 20 km/h, enquanto que a maior densidade medida está em torno de 50 veículos por quilômetro.

Dentro dos modelos *car fallowing models*, como o caso do IDM utilizado neste trabalho, os veículos tem como comportamento principal evitar a colisão com o veículo à frente, tendendo a um tráfego sincronizado mesmo com o aumento da quantidade de veículos. A velocidade do veículo líder do pelotão sempre funcionará como limite dos outros veículos da via. Ainda, segundo Banks (1989), se os motoristas fossem capazes de conduzir atrás do veículo à frente utilizando apenas o espaçamento mínimo entre os veículos, e os outros motoristas, assim por diante, conduzindo em altas velocidades, muitos outros veículos poderiam utilizar a estrada em uma mesma hora, mantendo o fluxo. Este é o comportamento encontrado em simulações e demostrado nesta análise, já que os condutores "virtuais" não apresentam o comportamento humano de cautela e segurança. Porém, em situações reais, como as encontradas no DS, quando uma certa compressão do espaço entre veículos ocorre em um tráfego intenso, a situação se torna instável e o motorista tende a frear ou mesmo deixar de acelerar o veículo por várias razões, levando à diminuição da velocidade, o que, por sua vez, diminuirá o fluxo. Desta forma, o fluxo máximo possível (q_{max}) é também obtido em função dos condutores, como apresentado neste cenário de simulação quando tenta-se analisar as semelhanças e diferenças entre os dados encontrados em cenários reais e os obtidos por meio de simulação.

De acordo com Kesting, Treiber, and Helbing (2010), como a simulação de tráfego é um processo complexo que envolve modelos não lineares e influências aleatórias (como o tipo de veículo inserido, espaço entre carros, aceleração mínima e máxima, etc.), espera-se que o fluxo livre máximo q_{max}, que representa o limite em que o tráfego passa do estado livre para o sincronizado ou congestionado, seja uma quantidade variável estocasticamente, levando a diferentes medidas, mesmo para limite idêntico e condições iniciais (assumindo uma semente aleatória do gerador de números pseudo-aleatórios do computador). Consequentemente, considera-se o fluxo livre máximo como uma variável aleatória que reflete a natureza probabilística do colapso do fluxo de tráfego.

O diagrama fundamental de fluxo de tráfego (Fluxo-Densidade) (Das and Levinson, 2004) apresentado no APÊNDICE B - TEORIA DO FLUXO DE TRÁFEGO é reproduzido no Gráfico 3. Por meio do diagrama é mostrado como o fluxo e a densidade se relacionam. No Gráfico 3, o fluxo tem a forma similar ao de uma parábola, o que é esperado quando a densidade é aumentada, como no cenário simulado, por meio do incremente de veículos a cada nova rodada de simulação, corroborando com os resultados que validam o modelo. Segundo Kerner and Rehborn (1996), geralmente um tráfico de fluxo congestionado corresponde a pontos amplamente dispersos no plano fluxo-densidade, conforme podemos observar no Gráfico 3 quando a densidade ultrapassa 30 veículos por quilômetro após o fluxo chegar próximo ao ponto de fluxo máximo (q_{max}) de 1750 veículos por hora e iniciar uma queda nos valores medidos.

Gráfico 3 – I-205: Diagrama fundamental de fluxo de tráfego (Fluxo-Densidade) - I 205 - Simulação



Como foi mostrado ao longo desta Seção, o modelo *TrGet* apresenta os resultados analíticos esperados (transições entre as fases e relação entre fluxo, velocidade e densidade) quando se utiliza um modelo CFM, como o IDM utilizado neste trabalho, tendo como referencial a teoria do tráfego, validando o modelo.

4.2 Cenário 2

No Cenário 2 é realizada a simulação de uma VANET utilizando os *traces* coletados por meio de simulação e realiza as seguintes ações: (i) simulação de um modelo de mobilidade por meio do modelo *TrGeT* utilizando o mapa da cidade de Málaga, Espanha; (ii) simulação de um modelo de mobilidade por meio da ferramenta *VanetMobiSim* utilizando o mesmo mapa de (i); (iii) simulação da VANET utilizando os *traces* da simulação (i); (iv) simulação da VANET utilizando os *traces* da simulação (ii); (v) simulação da VANET utilizando os *traces* da simulação (ii); (vi) comparação dos resultados das simulações das VANETs. Na Figura 16 é mostrado o mapa da área simulada.

O cenário de simulação no *VanetMobiSim* é configurado por meio de um arquivo *xml*. O arquivo *scenario_malaga.xml* foi configurado com as mesmas propriedades utilizadas pelo modelo *TrGeT* e modelo de mobilidade IDM. Foi utilizada apenas uma classe de veículo (*VehEntry*). No final da execução é gerado o arquivo de *traces dump-*

Figura 16 – Mapa - Málaga, Espanha

Traces.tcl por meio do *parser* que será utilizado na simulação da VANET no NS-2.

Na Tabela 7 são apresentados os valores dos parâmetros utilizados nas simulações.

Tabela 7 – Parâmetros da simulação - *TrGeT*

Parâmetro	**Descrição**	**Valor**
t	Duração da simulação	180 s
$VehEntry - n$	Número de MH	10
$VehEntry - l$	Tamanho do veículo	5,0 m
$VehEntry - a$	Aceleração	0,6 m/s^2
$VehEntry - b$	Desaceleração confortável	0,9 m/s^2
$VehEntry - v\_max$	Velocidade máxima	50,00 m/s
T	Tempo de segurança entre os MHs	0,5 s
S_0	Distância mínima entre os MHs	2,0 m
v_min	Velocidade mínima	2,78 m/s
$delta$	Expoente de aceleração δ	4,0 m/s^2
$kappa$	Fator de desaceleração κ	5,0 m/s^2

Para a simulação do cenário da VANET no NS-2, foi definido um fluxo de dados específico representando as diferentes possibilidades de comunicação existentes entre os veículos, tomando como base os experimentos realizados em Toutouh, García-Nieto, and Alba (2012). O modelo de fluxo de dados executa 10 sessões de um gerador de dados com taxa de bits constante (*constant bitrate* - CBR). Este fluxo opera sobre agentes UDP (*user datagram protocol*) definidos nos nós (veículos). Assim, os veículos interconectados trocam os dados gerados pelos agentes CBR. O tamanho do pacote de dados CBR é 512 *bytes* a taxa de 4 pacotes/s. O restante dos parâmetros de simulação estão resumidos na Tabela 8.

4.2.1Resultados: Cenário 2

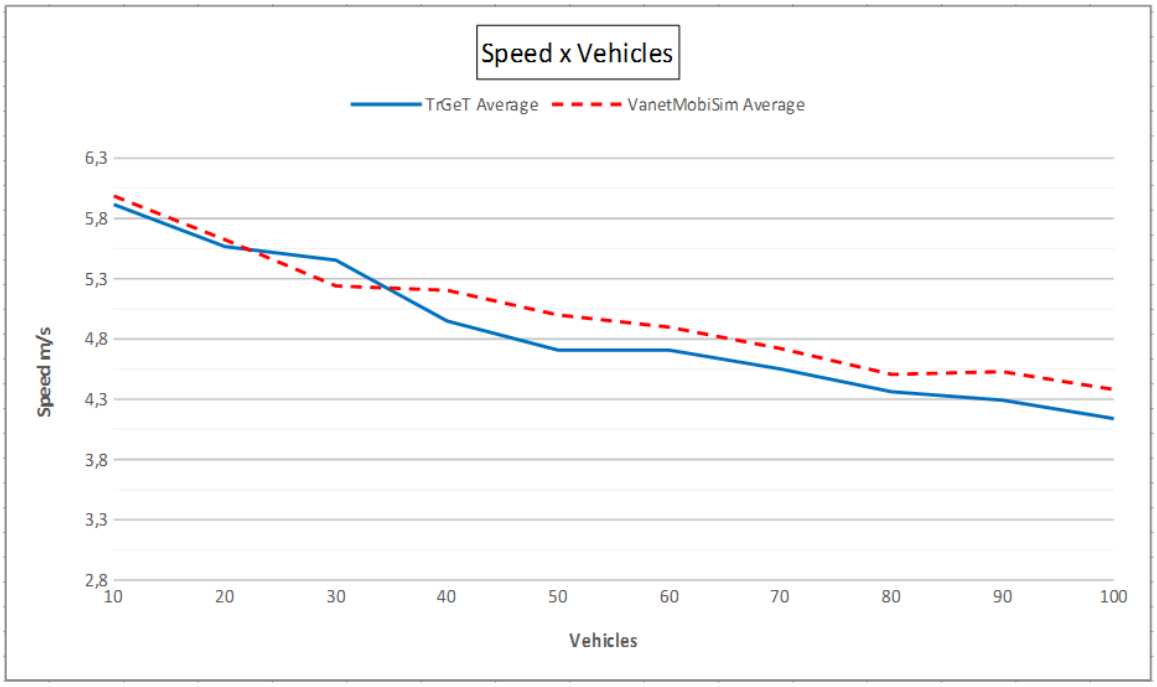
Foram realizadas 10 rodadas de simulação para cada uma das seguintes quantidades de veículos: 10, 20, 30, 40, 50, 60, 70, 80, 90 e 100. O Gráfico 4 mostra como a média das

Tabela 8 – Parâmetros da simulação NS-2

Parâmetro	**Valor**	**Parâmetro**	**Valor**
Modelo de propagação	*Nakagami*	Frequência da portadora	5.89 GHz
Largura de banda	6 $Mbps$	Camada PHY/MAC	802.11p
Protocolo de roteamento	$OLSR$	Camada de transporte	UDP
Taxa de transmissão CBR	100 $kbps$	Intervalo CBR	60 s

velocidades se comporta à medida que a quantidade de veículos aumenta, simulando um congestionamento. No Gráfico 4 é mostrada a variação das médias das velocidades por quantidade de veículos.

Gráfico 4 – Média das velocidades por quantidade de MHs



Pode-se observar a tendência de diminuição na média das velocidades à medida que a quantidade de veículos é incrementada em ambas as ferramentas. Pelo fato da aleatoriedade permear toda a simulação (escolha da localização inicial de cada veículo, velocidades desejadas e locais de origem e destino de cada jornada) é esperado que as linhas do gráfico apresentem pontos diferentes para cada ferramenta. Porém, a tendência da diminuição das médias das velocidades também é esperada e é confirmada por meio do gráfico, mostrando o mesmo padrão de comportamento das velocidades nos *traces* gerados.

Para avaliar a qualidade do modelo *TrGeT*, utilizamos duas métricas de *QoS*, como proposto em Toutouh, García-Nieto, and Alba (2012): (i) taxa de entrega de pacotes (*packet delivery ratio* - PDR), correspondente à fração de pacotes de dados originado por um aplicativo completamente e corretamente entregue; (ii) o atraso de fim a fim (*end-to-end delay* - E2ED), que é a diferença entre o tempo em que um pacote de dados é originado por um aplicativo e o tempo que o mesmo é recebido em seu destino.

Na simulação da VANET no NS-2 os seguintes resultados foram obtidos: a simulação realizada por meio dos *traces* de mobilidade gerados pelo modelo *TrGeT* apresentou a mediana da PDR de 99,52% e a mediana do E2ED de 3,76 ms; a simulação realizada com os *traces* de mobilidade gerados por meio da ferramenta *VanetMobiSim* apresentou a mediana da PDR de 99,99% e a mediana do E2ED de 1,97 ms. Os valores das métricas apresentam variações aceitáveis e de acordo com os encontrados em Toutouh, García-Nieto, and Alba (2012), validando o modelo *TrGeT*. A variação das medianas de PDR e E2ED entre as ferramentas é de 0,47% e 1,79 ms, respectivamente.

4.3 Considerações finais

Por meio das experimentações foi mostrado que o modelo TrGeT gera *traces* de forma rápida que podem ser utilizados para a simulação de VANETs em simuladores de redes. Atualmente, possui suporte para os simuladores NS2 e NS3, podendo ser expandido em trabalhos futuros para outros por meio da adição de métodos no *parser* implementado em *Python*. Por meio dos gráficos apresentados foi possível validar o modelo, comparando-os aos gráficos disponíveis na literatura e disponibilizados no APÊNDICE B - TEORIA DO FLUXO DE TRÁFEGO deste trabalho.

No cenário 1, a diferença no ponto máximo de fluxo e densidade entre os dados do DS (Gráfico 1) e os dados simulados (Gráfico 2) explica-se por alguns fatores. Um deles é a escolha dos parâmetros. Um parâmetro que apresenta impacto direto em engarrafamentos é a distância mínima entre os MHs (S_0), pois representa a distância mínima segura entre os carros (para-choque a para-choque). Em simulações, a literatura comumente utiliza 2 m, mas em casos reais esse valor é variável. Outro parâmetro é a classes de veículos que varia em comprimento, aceleração, desaceleração e velocidade máxima. Na simulação foi utilizada a classe de veículo de passeio com 5 m de comprimento, mas em casos reais esse valor varia, indo, por exemplo, de aproximadamente 3,5 m (carro compacto) a 6,5 m (furgão). Outro fator é a limitação da velocidade máxima permitida na autoestrada, que está configurada na simulação como 31,29 m/s (70,00 MPH ou 112,644 km/h), o atual limite da autoestrada I-205. Os dados do DS apresentam várias vezes velocidades acima da máxima da via. Na simulação a velocidade desejada fica entre a mínima e a máxima da rodovia, no caso 15,00 m/s e 31,29 m/s, sendo uma função probabilística (e estocástica) e não sendo possível ultrapassar o limite máximo de velocidade.

5 CONCLUSÃO

Nesta dissertação foi apresentado o modelo *TrGeT*, que tem por objetivo gerar, de forma rápida e com configuração de cenário simplificada, *traces* de mobilidade usados para simulação de VANETs. Os *traces* gerados, por sua vez, servem como dados de entrada para simuladores de rede.

Constituindo-se em um simulador modelado em RPC, o que sugere grande potencial de expansão e adaptação quando comparado a implementações feitas exclusivamente em linguagem de programação, o modelo *TrGeT*, durante a análise de desempenho, mostrou que gera *traces* de movimentação de nós móveis com comportamento dentro do esperado na teoria do tráfego. Em comparação direta aos resultados obtidos por meio de DS disponível na internet, que apresenta movimentos reais de veículos em uma auto-estrada, foram encontradas semelhanças e diferenças, que podem ser justificadas da seguinte maneira:

- O comportamento do modelo de mobilidade IDM tenta sempre evitar a colisão com o veículo à frente, porém, permite que os veículos utilizem o espaçamento mínimo entre os mesmos, ainda que conduzindo em altas velocidades. Este comportamento não representa o condutor humano de cautela e segurança;
- A simulação de tráfego envolve modelos não lineares e influências aleatórias;
- A simulação envolve funções probabilísticas (e estocásticas);
- Os resultados variam de acordo com os parâmetros utilizados, sendo necessário uma calibragem dos mesmos;
- Não foi utilizado no simulador um modelo de distribuição das chegadas dos veículos ao longo do tempo (ondas de chegadas).

A análise dessas limitações ajudaram a prospectar a evolução deste trabalho, oferecendo vários direcionamentos acerca da montagem de cenários de simulação mais realistas.

O modelo *TrGeT* mostrou também que apresenta resultados comparáveis à ferramenta de geração de *traces* *VanetMobiSim* (já previamente validada, segundo a literatura) em um cenário urbano. A verificação foi efetuada por meio do gráfico da variação das médias das velocidades dos veículos por quantidade de veículos, bem como pelo uso do modelo de mobilidade IDM gerado como entrada no simulador de rede NS-2 e obtenção das métricas de rede taxa de pacotes entregues (PDR) e atraso fim a fim (E2ED).

As principais contribuições deste trabalho são:

- Apresentar um modelo que possibilite a geração de *traces* que podem ser utilizados como entrada de simulações de VANETs;
- O modelo pode ser estendido por meio da alteração/adição de partes no modelo RPC;
- Como diferencial, relativamente a outras ferramentas, pelo fato de ser modelado

em RPC, o modelo permite a depuração da geração de *traces* e a interferência no estado do sistema ao longo da simulação, o que pode ser útil para estudos e análises específicas do modelo de mobilidade utilizado.

- Apresentou-se uma análise acerca das semelhanças e diferenças entre dados encontrados em cenários reais e os obtidos por meio de simulação, procurando entender e justificar as descobertas com base na literatura existente para que sirva como base para montagens de novos cenários de simulações mais próximos da realidade;

Foram observadas limitações na simulação de cenários urbanos (cenário 2). De acordo com a literatura pesquisada, a ausência de sinais de trânsito e ruas com mais de uma faixa na modelagem podem levar a resultados diferentes dos obtidos. Tais características aumentaria o realismo da simulação em tais cenários. Outra limitação apresentada é o tempo de simulação (tempo real) em cenários grandes (maiores de 10 km de extensão) e/ou com grande quantidade de veículos (mais de 2.000). Simulações com tais características podem demorar em torno de 2 horas. Em trabalhos futuros pretende-se trabalhar o modelo de forma a diminuir a quantidade de passos de simulação e, consequentemente, reduzir o tempo de simulação, assim como estender a modelagem com a adição de sinais de trânsito e ruas com mais de uma faixa.

Durante a elaboração deste trabalho surgiram ideias que podem agregar ou mesmo levar a pesquisa a outras áreas. Dentre elas, podemos citar:

- Atualmente o modelo *TrGeT* implementa dois algorítimos CFM: IDM e Krauß (conforme podemos verificar no APÊNDICE C: FUNÇÕES SML), porém somente o IDM foi utilizado nas análises de desempenho. Pretende-se realizar uma nova análise de desempenho utilizando como CFM o modelo Krauß;
- Realizar um estudo minucioso sobre os algoritmos de mobilidade microscópicos CFM para posterior adição ao modelo *TrGeT*;
- Adicionar mais restrições de mobilidade como: sinais de trânsito e possibilidade de múltiplas faixas nas ruas;
- Adicionar uma página ao modelo que possa modelar a distribuição de veículos em ondas de chegada;
- Estudar formas de acoplamento bidirecional entre a elaboração dos *traces* e simuladores de rede, como ocorre por meio do *framework Veins* (*Vehicles in Network Simulations*) (Sommer, German, and Dressler, 2011);

REFERÊNCIAS

Akyildiz, Ian F; Wang, Xudong; Wang, Weilin. Wireless mesh networks: a survey. *Computer networks*, v. 47, n. 4, p. 445–487, 2005.

Al-Sultan, Saif; Al-Doori, Moath M; Al-Bayatti, Ali H; Zedan, Hussien. A comprehensive survey on vehicular Ad Hoc network. *Journal of network and computer applications*, v. 37, p. 380–392, 2013.

Baldessari, Roberto; Bödekker, Bert; Deegener, Matthias; Festag, Andreas; Franz, Walter; Kellum, C. Christopher; Kosch, Timo; Kovacs, Andras; Lenardi, Massimiliano; Menig, Cornelius; Peichl, Timo; Röckl, Matthias; Seeberger, Dieter; Straßberger, Markus; Stratil, Hannes; Vögel, Hans-Jörg; Weyl, Benjamin; Zhang, Wenhui. *Car-2-Car Communication Consortium - Manifesto*. Tech. rep., 2007. URL `http://elib.dlr.de/48380/`.

Banks, James H. Freeway speed-flow-concentration relationships: more evidence and interpretations (with discussion and closure). *Transportation Research Record*, , n. 1225, 1989.

Basagni, Stefano; Conti, Marco; Giordano, Silvia; Stojmenovic, Ivan. *Mobile Ad Hoc networking: the cutting edge directions*, v. 35. John Wiley & Sons, 2013.

Behrisch, Michael; Bieker, Laura; Erdmann, Jakob; Krajzewicz, Daniel. SUMO–simulation of urban mobility: an overview. *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

Camp, Tracy; Boleng, Jeff; Davies, Vanessa. A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, v. 2, n. 5, p. 483–502, 2002.

CPN Tools. CPN Tools. `http://cpntools.org/`, 2015. [Online; acessado em 22/02/2017].

Crozet, Yves; Marlot, Gregoire. Congestion and road pricing: where is the'bug'. *The 9th World Conference on Transport Research*. 2001.

D. Cabarkapa, M. Milicevic. Importance of Realistic Mobility Software Models for VANETs Simulations. *The 7th International Conference - Information Technologies and Education Development - ITRO 2016*. 2016.

Dalal, Kusum; Dahiya, Pawan. State-of-the-Art in VANETs: The Core of Intelligent Transportation System. *IUP Journal of Electrical and Electronics Engineering*, v. 10, n. 1, p. 27, 2017.

Das, Shantanu; Levinson, David. Queuing and statistical analysis of freeway bottleneck formation. *Journal of transportation engineering*, v. 130, n. 6, p. 787–795, 2004.

Gartner, N.H.; Messer, C.J.; Rathi, A.K.; on Traffic Flow Theory, Committee; (AHB45), Characteristics. *Traffic Flow Theory: A State-of-the-art Report*. Committe on Traffic Flow Theory and Characteristics (AHB45), 2001. URL `https://books.google.com.br/books?id=hdBSMwEACAAJ`.

Harri, Jerome; Filali, Fethi; Bonnet, Christian. Mobility models for vehicular ad hoc networks: a survey and taxonomy. *IEEE Communications Surveys & Tutorials*, v. 11, n. 4, 2009.

Härri, Jérôme; Filali, Fethi; Bonnet, Christian; Fiore, Marco. VanetMobiSim: generating realistic mobility patterns for VANETs. *Proceedings of the 3rd international workshop on Vehicular ad hoc networks*. ACM, 2006, p. 96–97.

Jensen, Kurt; Kristensen, Lars M. *Coloured Petri nets: modelling and validation of concurrent systems*. Springer Science & Business Media, 2009.

Karnadi, Feliz Kristianto; Mo, Zhi Hai; Lan, Kun-chan. Rapid generation of realistic mobility models for VANET. *Wireless Communications and Networking Conference, 2007. WCNC 2007. IEEE*. IEEE, 2007, p. 2506–2511.

Kerner, Boris S; Rehborn, Hubert. Experimental properties of complexity in traffic flow. *Physical Review E*, v. 53, n. 5, p. R4275, 1996.

Kesting, Arne; Treiber, Martin; Helbing, Dirk. Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, v. 368, n. 1928, p. 4585–4605, 2010. URL `http://rsta.royalsocietypublishing.org/content/368/1928/4585`.

Khan, NA; Ahmad, F; Baig, WA. Formal model of random walk mobility model for WMN using coloured petri nets. *Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on*. IEEE, 2014, p. 436–444.

Khan, Naeem Akhtar; Ahmad, Farooq; Khan, Sher Afzal. SHER: a colored Petri net based random mobility model for wireless communications. *PloS one*, v. 10, n. 8, p. e0133634, 2015.

Krauß, Stefan. *Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics*. 1998.

Murata, Tadao. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, v. 77, n. 4, p. 541–580, 1989.

Omar, Hassan Aboubakr; Lu, Ning; Zhuang, Weihua. Wireless access technologies for vehicular network safety applications. *IEEE Network*, v. 30, n. 4, p. 22–26, 2016.

Open Street Map. OpenStreetMap. `https://www.openstreetmap.org`, 2017. URL `https://www.openstreetmap.org`. [Online; acessado em 22/02/2017].

Peterson, James L. Petri net theory and the modeling of systems. 1981.

Portland State University. Multimodal Data Set for Portland Oregon RegionTest Data Set for the FHWA Connected Vehicle InitiativeReal-Time Data Capture and Management Program. `https://portal.its.pdx.edu/fhwa`, 2012. [Online; acessado em 01/07/2017].

Rivoirard, Lucas; Wahl, Martine; Sondi, Patrick; Berbineau, Marion; Gruyer, Dominique. Using Real-World Car Traffic Dataset in Vehicular Ad Hoc Network Performance Evaluation. *International Journal of Advanced Computer Science and Applications (IJACSA)*, v. 7, n. 12, p. p390–398, 2016.

Sommer, Christoph; German, Reinhard; Dressler, Falko. Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE Transactions on Mobile Computing*, v. 10, n. 1, p. 3–15, 2011.

SUMO. Vehicle Type Parameter Defaults. `http://sumo.dlr.de/wiki/Vehicle_Type_Parameter_Defaults`, 2016. [Online; acessado em 20/07/2017].

Toutouh, Jamal; García-Nieto, José; Alba, Enrique. Intelligent OLSR routing protocol optimization for VANETs. *IEEE transactions on vehicular technology*, v. 61, n. 4, p. 1884–1894, 2012.

Treiber, Martin; Hennecke, Ansgar; Helbing, Dirk. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, v. 62, n. 2, p. 1805, 2000.

Ullman, Jeffrey D. *Elements of ML programming*. Prentice-Hall, Inc., 1994.

United States Census Bureau. TIGER - Topologically Integrated Geographic Encoding and Referencing. `https://tigerweb.geo.census.gov/tigerwebmain/TIGERweb_main.html`, 2007. [Online; acessado em 22/02/2017].

Vasconcelos, L.C.M. GitHub Repository - TrGeT. 2017. URL `https://github.com/lcvasconcelos/trget`. [Online; acessado em 22/02/2017].

Vasconcelos, L.C.M; Furtado Júnior, C. G.; Soares, J. M.; Barroso, G. C. *TrGeT*: geração de traces para simulação de Redes Veiculares Ad Hoc por meio de Redes de Petri Coloridas. `https://sbrc2017.ufpa.br/workshop/courb/`, 2017.

Wilson, R Eddie. Mechanisms for spatio-temporal pattern formation in highway traffic models. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, v. 366, n. 1872, p. 2017–2032, 2008.

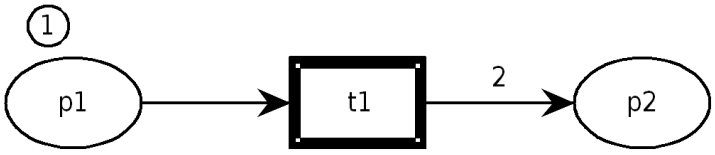
APÊNDICE A – REDES DE PETRI COLORIDAS

As redes de Petri (RP) foram originalmente descritas por Carl Adam Petri em sua tese de doutorado intitulada Comunicação entre Autômatos, defendida em 1962 (PETRI, 1962). A partir de então, consideráveis trabalhos teóricos e aplicações práticas com RP têm sido realizados, principalmente nas áreas de modelagem ("*software*" e "*hardware*"), redes de computadores, comunicações, sistemas distribuídos, sistemas elétricos de potênica, protocolos de comunicação, sistemas operacionais, sistemas de controle de produção, automação industrial, modelagem de controladores lógicos, análise de fluxo de tarefas, "chips" VLSI, modelagem de sistemas a eventos discretos (Jensen and Kristensen, 2009).

As Redes de Petri são uma ferramenta matemática e gráfica, para modelagem, análise, controle, validação e implementação de muitos sistemas, especialmente sistemas que possam ser interpretados como sistemas a eventos discretos (Murata (1989); Peterson (1981)).

Uma RP é um tipo de grafo bipartido e direcionado, em que os arcos nunca ligam dois nós do mesmo tipo. A Figura 17 apresenta os elementos gráficos que compõem uma rede de Petri.

Figura 17 – Ilustração de uma RP



Na Figura 17, p_1 e p_2 são os lugares, t_1 é a transição. Neste caso, p_1 é um lugar de entrada de t_1 e p_2 é um lugar de saída de t_1. O arco que liga p_1 a t_1 possui peso 1. O arco que liga t_1 a p_2 possui peso 2. O lugar p_1 possui uma ficha, representada na figura pelo valor "1" próximo ao lugar.

Nas RP, a ocorrência de um evento está associada ao disparo de uma transição e os lugares de entrada e saída da transição representam, respectivamente, as pré-condições e pós-condições associadas à ocorrência do evento. Os arcos de entrada de uma transição t_j têm origem em um ou mais lugares de entrada p_i de t_j e terminam na transição t_j; os arcos de saída têm origem na transição t_j e terminam em um ou mais lugares de saída p_i

de t_j. Conforme a Figura 17, o lugar p_1 é entrada de t_1, visto que um arco se origina em p_1 e termina em t_1. O lugar p_2 é saída de t_1, visto que um arco de peso 2 se origina em t_1 e termina em p_2.

As fichas são usadas nas RPs para simular a dinâmica e as atividades concorrentes do sistema. O estado de uma RP é representado por um número k_i de fichas contidas em cada lugar p_i, chamada marcação, conforme apresentado na Figura 17. O estado do sistema é dado pela distribuição de fichas nos lugares da RP e cada lugar representa um estado parcial do sistema. A mudança de estado é representada pelo movimento de fichas na RP, que acontece quando ocorre o disparo de transições. Cada evento que ocorre no sistema é associado ao disparo de uma transição no modelo RP. O disparo de uma transição significa que o seu evento correspondente ocorreu.

Uma transição é dita habilitada se cada lugar de entrada da transição contém um número de fichas maior ou igual ao peso do arco que o conecta à transição. Uma transição habilitada pode ou não disparar. Quando ocorre o disparo de uma transição, fichas são removidas dos lugares de entrada da transição e fichas são adicionadas aos lugares de saída. A quantidade de fichas removidas e acrescentadas depende do peso do arco. A nova marcação resultante do disparo da transição representa o novo estado do sistema. A marcação inicial M_0 representa o estado inicial da RP.

A definição formal de uma RP é apresentada a seguir (Murata, 1989).

Uma rede de Petri é uma 5-upla, $PN = (P, T, F, W, M_0)$ em que:

- $P = \{p_1, p_2, .., p_m\}$ é um conjunto finito de lugares;
- $T = \{t_1, t_2, ..., t_n\}$ é um conjunto finito de transições;
- $F \subseteq (PxT) \cup (TxP)$ é um conjunto de arcos (fluxo de relações);
- $W : F \rightarrow \{1, 2, 3...\}$ é uma função peso;
  - $w(p, t)$ peso do arco que liga o lugar à transição;

- $w(t, p)$ peso do arco que liga a transição ao lugar;

- $M_0 : P \rightarrow N^*$ é a marcação inicial, em que N denota os números naturais e M_0 a marcação inicial;

- $P \cap T = \varnothing$ e $P \cup T \neq \varnothing$

O comportamento dinâmico das RPs obedece à regra de disparo de transições, a saber:

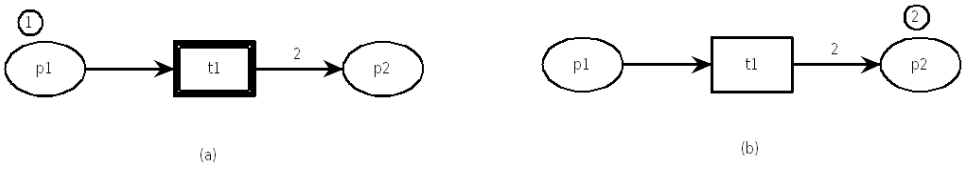
- uma transição é dita habilitada se cada lugar de entrada p de t é marcado com pelo menos $w(p, t)$ fichas, em que $w(p, t)$ é o peso do arco de p para t;

- uma transição habilitada pode ou não disparar;

- o disparo de uma transição t remove $w(p, t)$ fichas de cada lugar de entrada p de t, e adiciona $w(t, p)$ fichas a cada lugar de saída p de t, em que $w(t, p)$ é o peso do arco direcionado de t para p.

A RP apresentada na Figura 18 ilustra a regra de disparo através da modelagem do comportamento dinâmico de um sistema e sua evolução. A Figura 18(a) apresenta o estado inicial do sistema. A mudança de estado, que acontece através do disparo da transição $t1$, é apresentada na Figura 18(b). Na Figura 18(a), existe uma ficha no lugar de entrada $p1$ e nenhuma ficha no lugar de saída $p2$. A marcação da rede é $M_0 = (1, 0)$. Nesta marcação a transição $t1$ está habilitada e pode disparar. Conforme apresentado na Figura 18(b), no disparo da transição $t1$, uma ficha é removida do lugar de entrada $p1$ e duas fichas são adicionadas ao lugar de saída $p2$, originando uma nova marcação ou estado do sistema $M_1 = (0, 2)$. Como pode ser observado neste exemplo, a quantidade de fichas removidas do lugar de entrada e adicionada ao lugar de saída depende do peso dos arcos.

Figura 18 – Representação do disparo da transição $t1$. (a) estado inicial da RP antes do disparo, (b) estado da RP após o disparo.



A aplicação de RP na modelagem de sistemas tem a vantagem de permitir verificar as propriedades dos modelos construídos através dos métodos de análise formais, a saber: Árvore (Grafo) de Alcançabilidade ou Cobertura, Matriz de Incidência e Equação de Estado e Técnicas de Redução e Decomposição (Murata (1989); Jensen and Kristensen (2009)). A análise das propriedades das RP com estes métodos pode revelar informações importantes sobre a estrutura e comportamento do sistema modelado, permitindo ao projetista realizar modificações e as correções antes da implementação. Algumas destas propriedades são:

- (*Liveness* - vivacidade) permite saber se um sistema não possui bloqueio (*deadlock*) e se todos os elementos do sistema estão ativos;

- (*Reversibility* - reversibilidade ou reinicialização) permite saber se o sistema é capaz de sempre retornar ao estado inicial após uma tarefa realizada;

- (*Boundedness* - limitação) permite verificar a consistência de um sistema quanto aos limites de sua capacidade de armazenamento e de realização de tarefas.

Em geral, os sistemas do mundo real são complexos e possuem vários processos com características similares, mas não idênticos. As RP possuem apenas um tipo de ficha, que pode ser inteiro ou booleano. O fato das RP não manipularem tipos de dados diferentes, dificulta a modelagem de sistemas reais e complexos (Jensen and Kristensen, 2009).

Para modelar sistemas reais, muitas vezes é necessário construir várias subredes independentes com estruturas basicamente idênticas para processos similares. Isto pode tornar o modelo RP extremamente grande, dificultando o desenvolvimento do projeto e a visualização dos modelos na sua totalidade. Além disso, pode ser difícil observar similaridades e diferenças entre as redes individuais que representam as partes similares. Outro fato é que as RP não tratam de restrições de tempo, características inerentes aos sistemas reais.

Para contornar estes problemas foram desenvolvidas extensões de RP, capazes de descrever sistemas mais complexos de forma mais compacta, entre as quais estão as Redes de Petri Coloridas (RPC) que são redes de Petri de alto nível, e as redes de Petri

com restrições de tempo (Jensen and Kristensen, 2009).

Neste trabalho estamos usando as redes de Petri coloridas, portanto, a seguir será feita uma introdução desta extensão das redes de Petri.

Uma Breve descrição das Redes de Petri Coloridas

As redes de Petri Coloridas (RPC) são uma linguagem gráfica para a construção de modelos de sistemas a eventos discretos e análise de suas propriedades. RPC são uma linguagem de modelagem que combina as capacidades das redes de Petri com os recursos de uma linguagem de programação de alto nível (Jensen and Kristensen, 2009). A linguagem de programação das RPC é a CPN ML, que se baseia na linguagem de programação funcional Standard ML (Ullman, 1994). RPC são destinadas ao uso prático, em especial, porque elas permitem a construção de modelos compactos e paramétricos. A vantagem das RPC sobre as outras redes de Petri é a capacidade de modelar sistemas complexos e fornecer modelos com um alto nível de abstração e da representação gráfica (Jensen and Kristensen, 2009).

O tempo desempenha um papel significativo numa vasta gama de sistemas concorrentes. O funcionamento correto de alguns sistemas depende crucialmente do tempo tomado por certas atividades, e diferentes decisões de projeto podem ter um impacto significativo sobre o desempenho de um sistema. RPC incluem um conceito de tempo que faz com que seja possível capturar o tempo tomado por eventos no sistema. Isto significa que as RPC podem ser aplicadas para a análise de desempenho baseada em simulação para investigar métricas, tais como atrasos, taxa de transferência e tamanho de fila, e para a modelagem e validação de sistemas de tempo real.

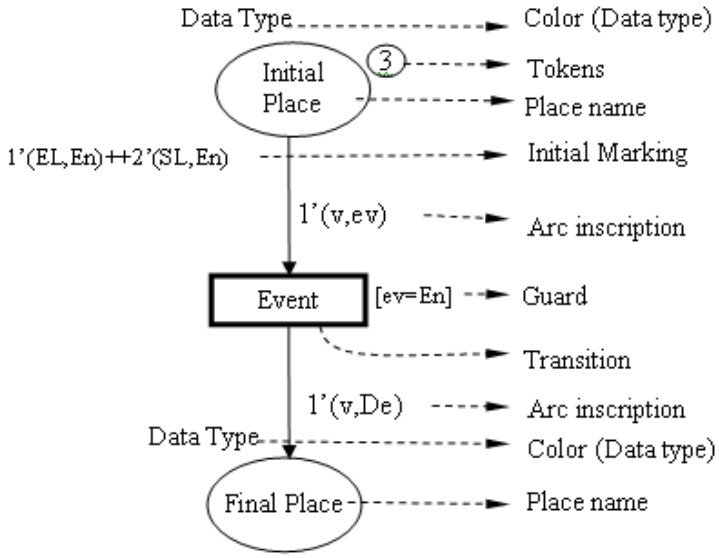
Modelos RPC podem conter diferentes níveis de abstração. Podem ser construídos em um conjunto de módulos, que são importantes quando lidamos com modelos de sistemas de grande porte. Os módulos interagem uns com os outros através de um conjunto de interfaces bem definidas, de uma forma semelhante às linguagens de programação. O conceito de módulos em RPC baseia-se num mecanismo de estruturação hierárquica, o que permite que um módulo tenha submódulos que podem ser reutilizados em diferentes partes do modelo.

RPC são ferramentas adequadas para a modelagem de modelos de mobilidade, pois fornecem recursos importantes, como a modularidade, facilidade de manutenção e capacidade de expansão. Estas características permitem a adição de novas funcionalidades ao modelo original, tornando possível adicionar novas características ou novos modelos CFM ao modelo.

O CPN Tools (CPN Tools, 2015) é uma ferramenta para edição, simulação, análise de espaço de estado, e análise de desempenho de modelos RPC. O usuário do CPN Tools trabalha diretamente com a representação gráfica do modelo RPC (Jensen and Kristensen, 2009).

A Figura 19 mostra a estrutura principal de um modelo RPC, constituído por lugares, transições e arcos. Com RPC, é possível usar os tipos de dados e manipulação de dados complexos.

Figura 19 – Representação de uma rede de Petri colorida.



Os lugares representam o estado do sistema modelado. Os nomes dos lugares são escritos dentro das elipses, e não têm nenhum significado formal - mas eles têm enorme importância prática para a legibilidade de um modelo RPC. Cada lugar pode ser marcado com uma ou mais fichas, e cada ficha tem um valor de dados ligado a ele. Este valor de dados é chamado cor da ficha. É o número de fichas e as cores das fichas nos lugares individuais que, em conjunto, representam o estado do sistema. Isto é chamado uma marcação do modelo RPC. As fichas em um lugar específico constituem a marcação daquele lugar.

As transições representam os eventos que podem ocorrer no sistema. Tal como acontece

com os lugares, os nomes das transições são escritos dentro dos retângulos. Quando a transição ocorre, ela remove fichas de seus lugares de entrada (aqueles lugares que têm um arco que conduz à transição) e adiciona fichas aos seus lugares de saída (os lugares que têm um arco proveniente da transição). As fichas que são removidas dos lugares de entrada e adicionadas aos lugares de saída, quando ocorre uma transição, são determinadas por meio das expressões de arco, que são as inscrições textuais posicionadas ao lado dos arcos individuais. As expressões de arco são escritas na linguagem de programação CPN ML e são construídas a partir de variáveis, constantes, operadores e funções. Quando todas as variáveis em uma expressão são ligadas a valores do tipo correto, a expressão pode ser avaliada.

As transições possuem uma guarda, que é uma expressão booleana. Quando uma guarda está presente, ela deve ser avaliada como verdadeira para que a transição possa ser habilitada, caso contrário, a transição estará desabilitada e não pode ocorrer. Assim, uma guarda coloca uma restrição adicional sobre a habilitação da transição. Com um modelo RPC temporizado, medidas de desempenho, tais como comprimento máximo da fila e média de tempo de espera, podem ser calculadas. Podemos verificar, também, se a operação de um sistema em tempo real obedece aos prazos exigidos.

Em um modelo de RPC temporizado, as fichas podem transportar um selo de tempo, além de sua cor. Isto significa que a marcação de um lugar em que as fichas possuem selo de tempo é agora um multiconjunto temporizado. Além disso, o modelo RPC possui um relógio global que representa o tempo de modelo. A distribuição das fichas nos lugares, em conjunto com as suas temporizações e o valor do relógio global, é chamada de marcação temporizada. Mesmo num modelo RPC hierárquico temporizado há um único relógio global. O selo de tempo especifica o tempo em que a ficha está pronta para ser utilizada, ou removida por uma transição que ocorre.

Retardos de tempo podem ser implementados nos arcos e nas transições. Se houver um tempo de retardo em uma transição, a sua ocorrência adiciona o valor do retardo a todas as fichas de saída que carregam um selo de tempo. Se há um tempo de retardo associado a um arco de saída de uma transição, o retardo é adicionado apenas para as fichas temporizadas que são colocadas no lugar de saída associado a esse arco.

APÊNDICE B – TEORIA DO FLUXO DE TRÁFEGO

Este apêndice traz um breve resumo das medidas utilizadas na análise de desempenho com base na teoria de tráfego. A teoria de tráfego traz três medidas que permeiam a maioria dos experimentos de tráfego: fluxo (q), que representa a taxa de veículos que passam por um ponto fixo, medido em veículos por hora; densidade (κ), que representa o número de veículos sobre um trecho de uma rodovia medido em unidade de veículos por quilômetro; velocidade, que representa a distância coberta por unidade de tempo, medida em quilômetros por hora. De acordo com o diagrama fundamental do fluxo de tráfego (Gartner *et al.*, 2001), as variáveis fluxo, velocidade e densidade são relacionadas de acordo com a Equação 3.

$$q = \kappa \, \bar{v_s} \tag{3}$$

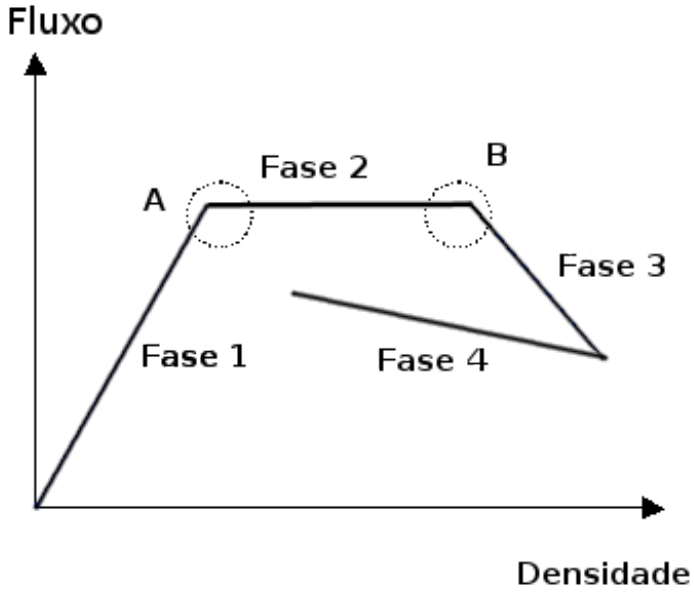
Na Equação (3), q representa o fluxo, κ a densidade e $\bar{v_s}$ velocidade média.

De acordo com Das and Levinson (2004), se o tráfego de uma autoestrada é permitido sem restrições, ele exibe quatro fases na curva de fluxo por densidade, conforme mostrado na Figura 20.

A Fase 1 representa a fase não congestionada, onde não há influência do aumento da densidade nas velocidades dos veículos. As velocidades não caem com a introdução de novos veículos na rodovia. Na Fase 2, a densidade aumenta, porém a introdução de novos veículos acarreta diminuição da velocidade, mantendo o fluxo. Na Fase 3, as velocidades e o fluxo caem, podendo gerar congestionamentos. A Fase 4 é a fase de recuperação onde, a densidade do tráfego começa a diminuir e a velocidade começa a aumentar.

Um gargalo é definido como uma constrição do fluxo de tráfego, onde a demanda excede uma capacidade disponível. Falando em infra-estrutura, um gargalo pode ser representado por uma faixa de uma avenida que está indisponível, ocasionando a diminuição da largura da pista, ou uma obstrução vista que reduz a velocidade de fluxo livre. Porém, os gargalos também podem se formar devido ao tráfego. O tráfego caótico, principalmente onde a mudança de faixas estão ocorrendo, é uma das principais fontes

Figura 20 – Diagrama de fases de tráfego



Fonte: Adaptado de Das and Levinson (2004)

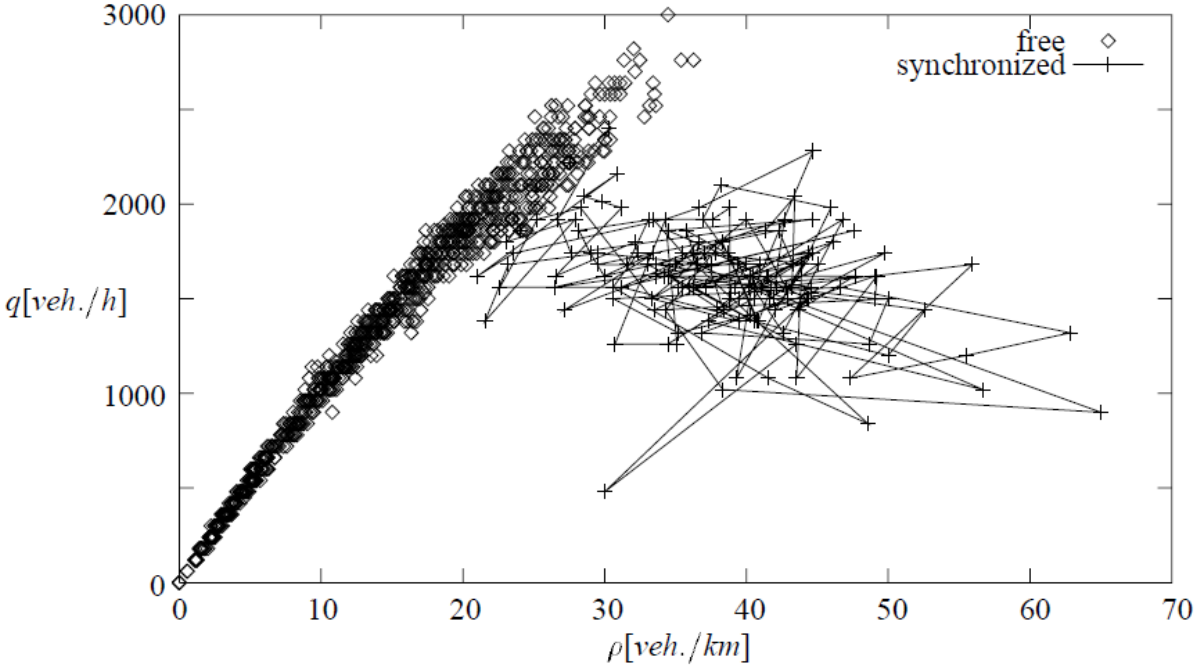
de um gargalo. Áreas de desvio são exemplos comuns.

Segundo Crozet and Marlot (2001), um mesmo fluxo pode ser alcançado em duas velocidades diferentes. Quando a demanda está abaixo da capacidade do gargalo, um fluxo pode ser alcançado em alta velocidade. Quando a demanda está acima da capacidade de gargalo, o mesmo fluxo só pode ser alcançado a uma baixa velocidade devido a fila que se forma.

Segundo a literatura Krauß (1998), existem três estados de tráfico: fluxo engarrafado, fluxo livre e fluxo sincronizado. No fluxo livre os veículos movem-se livremente sem que haja restrições a tal movimento. No tráfego engarrafado, a densidade de veículos é consideravelmente maior do que no fluxo livre. O fluxo sincronizado corresponde aos veículos movendo-se perto um dos outros quase sincronizadamente em diferentes faixas da estrada, No fluxo sincronizado, o tráfego se encontra em condições congestionadas, onde as interações entre veículos são significativas, as mudanças na faixa são raras e as velocidades nas duas faixas são aproximadamente iguais. A velocidade média no fluxo sincronizado é consideravelmente menor do que a velocidade no fluxo livre.

A Figura 21 mostra os tráfegos livre e congestionado no plano (p,q), onde p e q representam o fluxo e a densidade, respectivamente. No tráfego congestionado, os

Figura 21 – Diagrama fundamental de fluxo de tráfego (Fluxo-Densidade). Tráfego livre e congestionado na Autobahn A4 3



Fonte: (Krauß, 1998)

pontos são conectados por linhas. Cada ponto representa uma média medida em um minuto de tráfego.

APÊNDICE C – FUNÇÕES SML

Código 2: Arquivo *functions.sml*

```
fun norm (x1,x2,y1,y2) =
(Math.sqrt(Math.pow((x2-x1),2.0)+ Math.pow((y2-y1),2.0)));

fun addCoordinate(x1,x2,y1,y2,d, n)=
    if Real.== (x1,x2) then {x=x1, y=y1+d}
    else if Real.==(y2,y1) then {x=d+x1,y=y1}
    else if y1<y2 then {x=((d/(n))*(x2-x1))+x1 ,y=((d/n)*(y2-y1))+y1}
    else {x=((d/(n))*(x2-x1))+x1,y= ~((d/n)*(y1-y2)) + y1}

fun addCoordinateEdge(e : Edge, d)=
  addCoordinate(#x (#v1 e), #x (#v2 e), #y (#v1 e), #y (#v2 e), d, #n e);


fun addMovement (px,py,x1,x2,y1,y2,s,n,1) =
  let
      val dasc = (Math.sqrt(Math.pow((px-x1),2.0)+ Math.pow((py-y1),2.0))) + s;
      val desc = (Math.sqrt(Math.pow((px-x1),2.0)+ Math.pow((y1-py),2.0))) + s;
  in
     if Real.== (x1,x2) then {x=px, y=py+s}
     else if Real.==(y2,y1) then {x=px+s,y=py}
   else if y1<y2 then {x=((dasc/(n))*(x2-x1))+x1 , y=((dasc/n)*(y2-y1))+y1} Asc
 else {x=((desc/(n))*(x2-x1)) + x1, y= ~((desc/n)*(y1-y2)) + y1}  Desc
  end
| addMovement (px,py,x1,x2,y1,y2,s,n,2) =
  let
      val dasc = (Math.sqrt(Math.pow((x2-px),2.0)+ Math.pow((y2-py),2.0))) + s;
      val desc = (Math.sqrt(Math.pow((x2-px),2.0)+ Math.pow((py-y2),2.0))) + s;
  in
     if Real.== (x1,x2) then {x=x1, y=py-s}
     else if Real.==(y2,y1) then {x=px-s,y=y1}
     else if y1<y2 then {x= ~((dasc/(n))*(x2-x1))+x2 ,y= ~((dasc/n)*(y2-y1))+y2} Asc
     else {x= ~((desc/(n))*(x2-x1)) +x2, y= ((desc/n)*(y1-y2)) + y2}  Desc
  end;

fun reachVertex (px,py,x1,x2,y1,y2,n,1) =
     if Real.== (y1,y2) then x2-px
     else if Real.==(x1,x2) then y2-py
     else if y1<y2 then n*((y2-py)/(y2-y1))  Asc
     else n*((py-y2)/(y1-y2))  Desc

  | reachVertex (px,py,x1,x2,y1,y2,n,2) =
     if Real.== (y1,y2) then px-x1
     else if Real.==(x1,x2) then py-y1
     else if y1<y2 then n * ((py-y1)/(y2-y1))  Asc
     else n * ((y1-py)/(y1-y2))  Desc

fun changeStreet (x1,x2,y1,y2,s,n,dir) =
  if dir=1 then
       v1
      addMovement (x1,y1,x1,x2,y1,y2,s,n,1)
  else  v2
      addMovement (x2,y2,x1,x2,y1,y2,s,n,2)

```

```
fun finalVelocity(v_init, v_max, a, t : real) =
  let val v = v_init + (a*t)
  in
  if v > v_max then v_max else v
  end;

fun distance(v, t : real) =
  v*t

fun freeToRide(id1,eid1,v1,dir1,id2,eid2,v2,dir2) =
if eid1=eid2 andalso dir1=dir2 then  same edge and direction
if  Real.>(v2, v1) then true  in front
else false  following
else
true  edge or direction different

fun leader(m : MH, []) = true
|leader(m : MH, x::p : platoon) =
if (not (freeToRide(#id m, #ed m, #toVertex m,#dir m, #id x, #ed x, #toVertex x, #dir x)))
then false else leader(m,p)

fun sm_platoon (m : MH, [], np :platoon) = np
  | sm_platoon (m : MH, x::p : platoon, np :platoon) =
if (#ed m = #ed x) andalso (#dir m = #dir x) then sm_platoon(m,p,np^^[x])
    else (sm_platoon(m,p,np))

fun close(m1 :MH_NT, m2 :MH_NT) =
if Real.>(#toVertex m2, #toVertex m1) then true
else false

fun following (m : MH, []) = m
  | following (m : MH, x::p : platoon) =
if (#id m = #id x) orelse (Real.>(#toVertex x, #toVertex m)) then following(m,p)
    else x

fun updatePlatoon (p, mh, mo) =
  let
  val new_p = rm mh p;
  in
  new_p^^[mo]
end;

fun inFront(s1: Spot, s2: Spot) =
let
   val x1 = #x (#p s1);
    val y1 = #y (#p s1);
    val x2 = #x (#p s2);
    val y2 = #y (#p s2);
   val dir = #dir s1;
in
       if (dir=1) andalso (x1 <> x2) andalso (x1 > x2) then s1
   else if (dir=1) andalso (x1 <> x2) andalso (x2 > x1) then s2
   else if (dir=1) andalso (x1 = x2) andalso (y1 > y2) then s1
   else if (dir=1) andalso (x1 = x2) andalso (y2 > y1) then s2
   else if (dir=2) andalso (x1 <> x2) andalso (x1 < x2) then s1
   else if (dir=2) andalso (x1 <> x2) andalso (x2 < x1) then s2
   else if (dir=2) andalso (x1 = x2) andalso (y1 < y2) then s1
   else s2  if (dir=2) andalso (x1 = x2) andalso (y2 < y1) then
end;
```

```
113
114 fun getBackSpot(s1: Spot, s2: Spot) =
115   let
116    val spotAhead : Spot = inFront(s1,s2);
117   in
118    if (s1 = spotAhead) then s2 else s1
119 end;
120
121 fun isSameSpot(s1: Spot, s2: Spot, s0) =
122   let
123     val spotAhead : Spot = inFront(s1,s2);
124    val point : Spot = if (s1 = spotAhead) then s2 else s1;
125
126    val xc = #x (#p spotAhead);
127     val yc = #y (#p spotAhead);
128     val lc = #l (#v spotAhead);
129
130    val xp = #x (#p point);
131     val yp = #y (#p point);
132
133    val eid1 = #id (#edge s1);
134    val eid2 = #id (#edge s2);
135    val dir1 = #dir s1;
136    val dir2 = #dir s2;
137    val sameRoad = if eid1=eid2 andalso dir1=dir2 then true else false;  same edge and direction
138
139    val d = (Math.sqrt(Math.pow((xp-xc),2.0)+ Math.pow((yp-yc),2.0)));
140    val radius = lc+s0;
141
142   in
143    if (sameRoad = true) andalso (Real.<(d,radius)) then true
144    else false
145 end;
146
147 fun isOccupied (spot : Spot, [], s0) = false
148   | isOccupied (spot : Spot, x::s : Spots, s0) =
149       if (not (isSameSpot(spot, x, s0))) then isOccupied(spot, s, s0)
150     else true
151
152 fun makeSpotFromList(el: EdgeList, vh: Vehicle, lst: Spots, s0) =
153  let
154    val e : Edge = List.nth(el,discrete(0, (length el)-1));
155     val direction = if (#way e = 1) then 1 else if (#way e = 2) then 2 else discrete(1,2);
156     val p = if (direction = 1) then uniform (#l vh, ((#n e) - 0.10)) else uniform (0.10, ((#n e) - (#l
          vh)));
157      val coord : Point = addCoordinateEdge(e, p);
158     val spt : Spot = {edge= e, p=coord, dir=direction, v=vh};
159   in
160     if (isOccupied(spt, lst, s0)) then makeSpotFromList(el, vh, lst, s0)
161     else spt::lst
162 end;
163
164 fun generateVehicles(ve :VehEntry, 0, vehs :VehList) = vehs
165    |generateVehicles(ve :VehEntry, h, vehs :VehList) =
166       generateVehicles(ve, h-1, vehs^^[{id=0,l=(#l ve),a=(#a ve),b=(#b ve),vmax=(#vmax ve)}])
167
168 fun getIntersection(mh : MH, ne : Edge, e :Edge) =
169 let
170    val new_dir = if #dir mh = 1 andalso (#id (#v2 e) = #id (#v1 ne)) then 1
```

```
171       else if #dir mh = 1 andalso (#id (#v2 e) = #id (#v2 ne)) then 2
172       else if #dir mh = 2 andalso (#id (#v1 e) = #id (#v1 ne)) then 1
173       else 2;
174    val x1 = #x (#v1 ne);
175    val x2 = #x (#v2 ne);
176    val y1 = #y (#v1 ne);
177    val y2 = #y (#v2 ne);
178  in
179    if new_dir=1 then
180        v1
181     IntInf.toString (#id (#v1 ne)) ^ "," ^ Real.toString x1 ^ "," ^ Real.toString y1
182    else  v2
183     IntInf.toString (#id (#v2 ne)) ^ "," ^ Real.toString x2 ^ "," ^ Real.toString y2
184  end;
185
186  fun getNewV(mh : MH, ne : Edge, e :Edge) =
187  let
188    val new_dir = if #dir mh = 1 andalso (#id (#v2 e) = #id (#v1 ne)) then 1
189       else if #dir mh = 1 andalso (#id (#v2 e) = #id (#v2 ne)) then 2
190       else if #dir mh = 2 andalso (#id (#v1 e) = #id (#v1 ne)) then 1
191       else 2;
192  in
193    if new_dir=1 then #v1 ne else #v2 ne
194  end;
195
196  fun getWCT () = Date.fmt "%H:%M:%S" (Date.fromTimeLocal(Time.now()))
197
198  fun speedFree(mh : MH) =
199  let
200    val gap = #toVertex mh;
201    val deltaV = (#v mh);
202    val new_aceleration = (#a mh)* (1.0 - Math.pow(((#v mh)/(#v_desired mh)),delta));  velocity
203    val sp = if Real.isNan((#v mh)+new_aceleration) orelse Real.<(((#v mh)+new_aceleration),0.0) then 0.0
         else (#v mh)+new_aceleration;
204  in
205    if Real.>(sp,#v_desired mh) then #v_desired mh else sp
206  end;
207
208  fun newSpeed(mh : MH, p :platoon) =
209  let
210    val close_mh = following(mh, (rev (sort close (sm_platoon(mh,p,nil)))));
211    val gap = (#x (#pos close_mh)) - (#x (#pos mh)) - (#l mh);
212    val deltaV = (#v mh) - (#v close_mh) ;
213    val ss = s0 + Real.max((((#v mh) * T) + ((#v mh) *(deltaV))/2.0*Math.sqrt((#a mh)*(#b mh))),0.0);
         effective desired distance
214    val new_aceleration = (#a mh)* (1.0 - Math.pow(((#v mh)/(#v_desired mh)),delta) -
         Math.pow((ss/gap),2.0));
         velocity
215
216    val sp = if Real.isNan((#v mh)+new_aceleration) orelse Real.<(((#v mh)+new_aceleration),0.0) then 0.0
         else (#v mh)+new_aceleration;
217
218  in
219    if Real.>(sp,#v_desired mh) then #v_desired mh else sp
220  end;
```

Código 3: Arquivo *krauss.sml*

```
(* Krauss original thesis new velocity. Self contained method *)
fun newV(gap, vf, vl, vmax, a, b, l, epsilon) =
let
    val n = uniform (0.0, epsilon*a);
    val tau_b = (vl + vf)/(2.0*b);
    val vn = if (vf < vmax - a) then vf + a else vmax;

    val vsafe = vl + ((gap - vl)/(tau_b +1.0));
    val vnext_s = if (vn > vsafe) then vsafe else vn;
    val vnext = vnext_s - n;
in
    if (vnext < 0.0) then 0.0 else vnext

end;

(* Krauss safe velocity - SUMO Implementation - Tau = 1s *)
fun vsafe(gap, vl, b) =
  if (Real.<(vl, 0.01) andalso Real.<(gap, 0.01)) then 0.0
  else if (Real.<(vl, 0.01) andalso Real.<=(gap, b)) then Real.min(b, gap)
  else ((~1.0 * b) + (Math.sqrt((b*b) + (vl*vl) + (2.0 * b * gap))));

(* Krauss next max speed - SUMO Implementation - Tau = 1s *)
fun maxNextSpeed(speed, a, vehicleTypeMaxSpeed)= Real.min((speed + a) , vehicleTypeMaxSpeed);

(* Krauss safe speed for approaching a non-movine obstacle - SUMO Implementation - Tau = 1s *)
fun stopSpeed(speed, gap, a, b, vehicleTypeMaxSpeed) = Real.min(vsafe(gap, 0.0, b) ,maxNextSpeed(speed, a,
     vehicleTypeMaxSpeed));

(* Krauss follow speed no dawnling (epsilon)- SUMO Implementation - Tau = 1s *)
fun followSpeed(speed, gap, vl, a, b, vehicleTypeMaxSpeed) = Real.min(vsafe(gap, vl, b)
     ,maxNextSpeed(speed, a, vehicleTypeMaxSpeed));
```

APÊNDICE D – PARSERS

Código 4: Arquivo *Parse.py*

```
import re
from operator import itemgetter
import csv
import os
import errno
import matplotlib.pyplot as plt
import sys
import zipfile
import shutil
from decimal import Decimal


def joint(filenames):
    # filenames = ['output\GenerateMH.txt','output\CFM.txt', 'output\FreeFlow.txt']
    with open('output\Steps.txt', 'w') as outfile:
        for fname in filenames:
            with open(fname) as infile:
                for line in infile:
                    outfile.write(line)
    print('Steps joined...')


def read_map_line(line):
    # 1‘{id=100,v1={id=1, x=0.00, y=0.00, tp=join, return=false},v2={id=11, x=19000.00,
    # y=0.00,tp=inductor, return=false},way=1,n=0.0,vmax=31.29}++

    ln = line.split(",")
    x1 = re.findall("\d+\.\d+", ln[2])[0]
    x2 = re.findall("\d+\.\d+", ln[7])[0]
    y1 = re.findall("\d+\.\d+", ln[3])[0]
    y2 = re.findall("\d+\.\d+", ln[8])[0]

    return x1 + "," + x2 + "," + y1 + "," + y2 + "\n"


def create_map(filename):
    with open('temp\map.dat', 'w') as outfile:
        with open(filename) as infile:
            for line in infile:
                outfile.write(read_map_line(line))
    print('Map created...')


def read_step_line(line):
    # 4,6847.53627779,1000.0,0.0,2
    # id, x, y, v,t

    id = line[0]
    x = line[1]
    y = line[2]
    v = line[3]
    t = line[4]

    return [t, id, x, y, v]

```

```
56
57  def create_tcl(filename):
58      steps = []
59
60      with open(filename, 'r') as f:
61          reader = csv.reader(f, quoting=csv.QUOTE_NONNUMERIC)
62          for i in reader:
63              steps.append(read_step_line(i))
64
65      # [t,id,x,y,v]
66      sst = sorted(steps, key=itemgetter(0))
67
68      with open('temp\mm.tcl', 'w') as outfile:
69          for st in sst:
70              new_plot = "$ns_ at " + str(st[0] - 1.0) + " " + '"$node_(' + str(st[1]) + ") setdest " + str(
71                  st[2]) + " " + str(st[3]) + " " + str(st[4]) + '"\n'
72              outfile.write(new_plot)
73      print('TCL trace file created...')
74
75      with zipfile.ZipFile('temp\mm.tcl.zip', 'w', zipfile.ZIP_DEFLATED) as file:
76          file.write('temp\mm.tcl', 'mm.tcl')
77      print('Zip TCL trace file created...')
78      os.remove('temp\mm.tcl')
79
80
81  def read_intersections(filename):
82      intersections = []
83      with open(filename, 'r') as f:
84          inter = csv.reader(f, quoting=csv.QUOTE_NONNUMERIC)
85          for i in inter:
86              intersections.append(i)
87
88      return intersections
89
90
91  def get_index(marks_counter, number):
92      for mk in marks_counter:
93          if mk[0] <= number <= mk[1]:
94              return marks_counter.index(mk)
95
96      return 0
97
98
99  def get_mark(marks_value, index, id_vertex):
100     # time,id_vertex,count
101     for value in marks_value:
102         if value[1] == id_vertex and value[0] == index:
103             return value
104
105     return None
106
107
108 def flux(intersections, time, marks):
109     marks_counter = []
110     last_index = 0
111
112     for i in range(1, marks + 1):
113         marks_counter.append([last_index, last_index + time])
114         last_index = last_index + time + 1
```

```
115
116     mark_values = []
117     for it in intersections:
118         # MH,t,ID_Vertex,V_X,V_Y,Velo
119         index = get_index(marks_counter, int(it[1]))
120
121         mf = get_mark(mark_values, index, int(it[2]))
122         # time,id_vertex,count
123         if mf is None:
124             mark_values.append([index, int(it[2]), 1])
125         else:
126             mf[2] = mf[2] + 1
127
128     return mark_values
129
130
131 def speed(intersections, time, marks):
132     marks_counter = []
133     last_index = 0
134
135     for i in range(1, marks + 1):
136         marks_counter.append([last_index, last_index + time])
137         last_index = last_index + time + 1
138
139     mark_values = []
140     for it in intersections:
141         # MH,t,ID_Vertex,V_X,V_Y,Velo
142         index = get_index(marks_counter, int(it[1]))
143
144         mf = get_mark(mark_values, index, int(it[2]))
145         # time,id_vertex,velo
146         if mf is None:
147             mark_values.append([index, int(it[2]), [it[5]]])
148         else:
149             mf[2].append(it[5])
150
151     return mark_values
152
153
154 def compute_flow(hits):
155     return 3600 * hits / 60.0
156
157
158 def compute_density(q, vs):
159     return q / vs
160
161
162 def ms_to_kmh(v):
163     return v * 3.6
164
165
166 def kmh_to_ms(v):
167     return v / 3.6
168
169
170 def time_mean_speed(speeds):
171     first = 1.0 / speeds.size()
172     second = 0.0
173
```

```
174     for sp in speeds:
175         second = second + ms_to_kmh(sp)
176 
177     return first * second
178 
179 
180 def space_mean_speed(speeds):
181     n = len(speeds)
182     mean = 0.0
183 
184     for sp in speeds:
185         mean = mean + (1.0 / ms_to_kmh(sp))
186 
187     return n / mean
188 
189 
190 def report_flux_density(fms, sms, intersection, marks):
191     sj = []
192 
193     for i in range(0, marks):
194         mf = get_mark(fms, i, intersection)
195 
196         if mf is not None:
197             q = compute_flow(get_mark(fms, i, intersection)[2])
198             vs = space_mean_speed(get_mark(sms, i, intersection)[2])
199             k = compute_density(q, vs)
200 
201             sj.append('{0:.3f}'.format(k) + "," + '{0:.3f}'.format(q))
202         else:
203             sj.append("0.00" + "," + "0.00")
204 
205     with open('temp/report_qk.dat', 'w') as outfile:
206         for s in sj:
207             outfile.write(s + '\n')
208 
209     print("Done Report q x v file...")
210 
211 
212 def report_flux_speed(fms, sms, intersection, marks):
213     sj = []
214 
215     for i in range(0, marks):
216         mf = get_mark(fms, i, intersection)
217 
218         if mf is not None:
219             q = compute_flow(get_mark(fms, i, intersection)[2])
220             vs = space_mean_speed(get_mark(sms, i, intersection)[2])
221 
222             sj.append('{0:.3f}'.format(vs) + "," + '{0:.3f}'.format(q))
223         else:
224             sj.append("0.00" + "," + "0.00")
225 
226     with open('temp/report_qv.dat', 'w') as outfile:
227         for s in sj:
228             outfile.write(s + '\n')
229 
230     print("Done Report q x k file...")
231 
232 
```

```
233 def graphics():
234     # Plot properties
235     bg = 'w'
236     ax_bg = '#ccced1'
237     mk = 4
238     dot_color = 'b'
239     style = '-.'
240     ln_width = 1.1
241 
242     k = [] # density
243     q = [] # flux
244 
245     with open('temp/report_qk.dat', 'r') as f:
246         for line in f:
247             k.append(Decimal(line.split(",")[0]))
248             q.append(Decimal(line.split(",")[1].split('\n')[0]))
249 
250     plt.figure(1)
251 
252     plt.subplot(121, facecolor=bg)
253     plt.plot(k, q, 'ko', markersize=mk, color=dot_color)
254     # plt.plot(k, q)
255 
256     # Set chart title and label axes.
257     plt.title("Flow-Density", fontsize=10)
258     plt.xlabel("Density in cars per km", fontsize=8)
259     plt.ylabel("Flow in cars per hour", fontsize=8)
260     plt.grid(True, color=ax_bg, linestyle=style, linewidth=ln_width)
261     plt.tick_params(axis='both', labelsize=8)
262 
263     v = [] # speed
264     q = [] # flux
265 
266     with open('temp/report_qv.dat', 'r') as f:
267         for line in f:
268             v.append(Decimal(line.split(",")[0]))
269             q.append(Decimal(line.split(",")[1].split('\n')[0]))
270 
271     plt.subplot(122, facecolor=bg)
272     plt.plot(v, q, 'ko', markersize=mk, color=dot_color)
273 
274     # Set chart title and label axes.
275     plt.title("Flow-Velocity", fontsize=10)
276     plt.xlabel("Car velocity km/h", fontsize=8)
277     plt.ylabel("Flow in cars per hour", fontsize=8)
278     plt.grid(True, color=ax_bg, linestyle=style, linewidth=ln_width)
279     plt.tick_params(axis='both', labelsize=10)
280 
281     plt.show()
282 
283 
284 def copytree(src, dst, symlinks=False, ignore=None):
285     try:
286         os.makedirs(dst)
287     except OSError as e:
288         if e.errno != errno.EEXIST:
289             raise
290 
291     for item in os.listdir(src):
```

```
292         s = os.path.join(src, item)
293         d = os.path.join(dst, item)
294         if os.path.isdir(s):
295             shutil.copytree(s, d, symlinks, ignore)
296         else:
297             shutil.copy2(s, d)
298 
299 
300 # Make temp directory and call functions
301 if __name__ == '__main__':
302 
303     copytree('..\output', 'output')
304 
305     time = 60
306     intersections = read_intersections('output\changeStreet.txt')
307     final_mark = int(intersections[-1][1])
308     marks = int(final_mark // time)
309     remainder = int(final_mark % time)
310 
311     if remainder != 0:
312         marks = marks + 1
313 
314     mf = flux(intersections, time, marks)
315     ms = speed(intersections, time, marks)
316 
317     try:
318         os.makedirs('temp')
319     except OSError as e:
320         if e.errno != errno.EEXIST:
321             raise
322 
323     # create_map("declarations\map.sml")
324     joint(['output\GenerateMH.txt', 'output\CFM.txt', 'output\FreeFlow.txt'])
325     create_tcl("output\Steps.txt")
326 
327     if len(mf) > 0:
328         id_vertex = mf[0][1]
329         report_flux_density(mf, ms, id_vertex, marks)
330         report_flux_speed(mf, ms, id_vertex, marks)
331         graphics()
332         shutil.rmtree('output')
333     else:
334         print('No marks in intersection')
```

APÊNDICE E – PARSER I-205

Código 5: Arquivo *read.py*

```
from decimal import Decimal
import matplotlib.pyplot as plt
from pylab import figure, show, legend, ylabel
import numpy as np
from mpl_toolkits.axes_grid1 import host_subplot
import mpl_toolkits.axisartist as AA


class FreewayPlot:
    """ Class to read the freeway plot """

    # detectorid,starttime,volume,speed,occupancy,status,dqflags

    def __init__(self, line, measure_time):
        self.plot = line
        self.detectorid = line.split(",")[0]
        self.starttime = line.split(",")[1]
        self.volume = Decimal(line.split(",")[2]) if line.split(",")[2] != '' else 0
        self.speed = Decimal(line.split(",")[3]) if line.split(",")[3] != '' else 0
        self.occupancy = line.split(",")[4]
        self.measure_time = measure_time

        self.q = 3600 * self.volume / self.measure_time
        self.k = self.q / self.vel_kmh() if self.vel_kmh() != 0 else 0

    def vel_kmh(self):
        """ Converts mph to kmh"""
        return Decimal(self.speed) * Decimal(1.6093440) if self.speed != 0 else 0

    def vel_mps(self):
        """ Converts mph to m/s """
        return Decimal(self.speed) * Decimal(0.44704) if self.speed != 0 else 0

    def to_string(self):
        print('Detector id:', self.detectorid)
        print('Start Time:', self.starttime)
        print('Volume', self.volume)
        print('Speed: mph:', self.speed)
        print('Speed: kmph:', self.vel_kmh())
        print('Occupancy:', self.occupancy)
        print('Flux q:', 3600 * self.volume / 20)


def convert_to_min(o_clock):
    h, m, s = o_clock.split(':')
    return int(h) * 60 + int(m)

def time_mean_speed(speeds):
    """ Calculates the time mean speed """

    first = Decimal(1.0 / len(speeds))
    second = Decimal(0.0)

    for sp in speeds:
        second = second + sp
```

```
56
57      return first * second
58
59
60  def space_mean_speed(speeds):
61      """ Calculates the space mean speed """
62
63      n = Decimal(len(speeds))
64      mean = Decimal(0.0)
65
66      for sp in speeds:
67          if sp != 0:
68              mean = mean + (Decimal(1.0) / sp)
69
70      return n / mean
71
72
73  def marks_by_detector_speed(detector, date, qtd, sp):
74      """ Prints the speed by detector and date """
75
76      vehs = []
77      with open('data/freeway_loopdata%s.csv' % qtd) as fp:
78          for line in fp:
79              plot = FreewayPlot(line, 3600)
80              if plot.detectorid == detector and plot.starttime.split(" ")[0] == date and plot.speed >= sp:
81                  vehs.append(plot.vel_kmh())
82                  # print(plot.vel_mps())
83
84      print("Length", len(vehs))
85      print("Max Speed:", max(vehs), 'km/h')
86      print("Min Speed:", min(vehs), 'km/h')
87      print("Time Mean Speed Speed:", time_mean_speed(vehs), 'km/h')
88      print("Space Mean Speed:", space_mean_speed(vehs), 'km/h')
89
90
91  def marks_by_detector_flux(detector, date):
92      """ Prints the flux by detector and date """
93
94      maxi = 0
95      with open('freeway_loopdata.csv') as fp:
96          for line in fp:
97              plot = FreewayPlot(line)
98              if plot.detectorid == detector and plot.starttime.split(" ")[0] == date:
99                  print(plot.q)
100                 if plot.q > maxi:
101                     maxi = plot.q
102     print("Max Flux:", maxi)
103
104
105 def marks_by_detector_density(detector, date):
106     """ Prints the density by detector and date """
107
108     maxi = 0
109     with open('data/freeway_loopdata.csv') as fp:
110         for line in fp:
111             plot = FreewayPlot(line)
112             if plot.detectorid == detector and plot.starttime.split(" ")[0] == date:
113                 print(plot.k)
114                 if plot.k > maxi:
```

```
115                     maxi = plot.k
116     print("Max K:", maxi)
117
118
119 def create_log(detector, date, qtd, measure_time):
120     """ Create report_qv.dat e report_qk.dat files to plot graphics """
121
122     qv = open('report_qv.dat', 'w')
123     qk = open('report_qk.dat', 'w')
124
125     with open('data/freeway_loopdata%s.csv' % qtd) as fp:
126         for line in fp:
127             if not line[0].isdigit():
128                 continue
129
130             plot = FreewayPlot(line, measure_time)
131             if plot.detectorid == detector and plot.starttime.split(" ")[0] == date:
132                 qv.write("%f,%f,%s\n" % (plot.vel_kmh(), plot.q, plot.starttime.split()[1].split("-")[0]))
133                 qk.write("%f,%f,%s\n" % (plot.k, plot.q, plot.starttime.split()[1].split("-")[0]))
134
135     qv.close()
136     qk.close()
137     print('Finish write files.')
138
139
140 def graphics():
141     # Plot properties
142     bg = 'w'
143     ax_bg = '#ccced1'
144     mk = 4
145     dot_color = 'b'
146     style = '-'
147     ln_width = 1
148
149     k = [] # density
150     q = [] # flux
151     v = [] # speed
152     t = [] # time
153
154     with open('report_qk.dat', 'r') as f:
155         for line in f:
156             k.append(Decimal(line.split(",")[0]))
157             q.append(Decimal(line.split(",")[1].split('\n')[0]))
158             t.append(line.split(",")[2])
159
160     with open('report_qv.dat', 'r') as f:
161         for line in f:
162             v.append(Decimal(line.split(",")[0]))
163
164     fig1 = plt.figure(1)
165
166     plt.subplot(121, facecolor=bg)
167     plt.plot(k, q, 'ko', markersize=mk, color=dot_color)
168
169     # Set chart title and label axes.
170     plt.title("Fluxo-Densidade", fontsize=10)
171     plt.xlabel("Densidade de carros por km", fontsize=8)
172     plt.ylabel("Fluxo de carros por hora", fontsize=8)
173     plt.grid(True, color=ax_bg, linestyle=style, linewidth=ln_width)
```

```
174     plt.xticks(range(0, 30 + 1, 5))
175     plt.yticks(range(0, 2250 + 1, 250))
176     plt.tick_params(axis='both', labelsize=8)
177
178     plt.subplot(122, facecolor=bg)
179     plt.plot(v, q, 'ko', markersize=mk, color=dot_color)
180
181     # Set chart title and label axes.
182     plt.title("Fluxo-Velocidade", fontsize=10)
183     plt.xlabel("Velocidade km/h", fontsize=8)
184     plt.ylabel("Fluxo de carros por hora", fontsize=8)
185     plt.grid(True, color=ax_bg, linestyle=style, linewidth=ln_width)
186     plt.xticks(range(0, 150 + 1, 10))
187     plt.yticks(range(0, 2250 + 1, 250))
188     plt.tick_params(axis='both', labelsize=10)
189
190     fig2 = figure(2)
191
192     # using this function create a list times in minutes for each time
193     min_col = [convert_to_min(t) for t in t]
194     time = ['0:00', '1:00', '2:00', '3:00', '4:00', '5:00', '6:00', '7:00', '8:00', '9:00', '10:00',
            '11:00', '12:00',
195           '13:00', '14:00', '15:00', '16:00', '17:00', '18:00', '19:00', '20:00', '21:00', '22:00',
                '23:00','23:59']
196
197     # this will place the strings above at every 60 min
198     tick_location = [t * 60 for t in range(25)]
199
200     host = host_subplot(111, axes_class=AA.Axes)
201     plt.subplots_adjust(right=0.75)
202
203     par1 = host.twinx()
204     par2 = host.twinx()
205
206     offset = 60
207     new_fixed_axis = par2.get_grid_helper().new_fixed_axis
208     par2.axis["right"] = new_fixed_axis(loc="right",
209                                         axes=par2,
210                                         offset=(offset, 0))
211
212     par2.axis["right"].toggle(all=True)
213
214     # host.set_xlim(0, len(t))
215     # host.set_ylim(0, 200)
216     plt.yticks(range(0, 150 + 1, 10))
217
218     host.set_xlabel("Hora")
219     host.set_ylabel("Velocidade km/h")
220     par1.set_ylabel("Fluxo por hr ")
221     par2.set_ylabel("Densidade por Km")
222
223     p1, = host.plot(min_col, v, 'o', markersize=mk, label="Velocidade")
224     p2, = par1.plot(min_col, q, 'xr', markersize=mk, label="Fluxo")
225     p3, = par2.plot(min_col, k, '*g', markersize=mk, label="Densidade")
226
227     # par1.set_ylim(0, 2250)
228     # par2.set_ylim(0, 30)
229
230     # host.legend()
```

```
231     host.legend(loc='upper center', bbox_to_anchor=(0.5, 1.00), ncol=1)
232
233     host.set_xticks(tick_location)
234     host.set_xticklabels(time)
235
236     host.axis["left"].label.set_color(p1.get_color())
237     par1.axis["right"].label.set_color(p2.get_color())
238     par2.axis["right"].label.set_color(p3.get_color())
239
240     par1.set_yticks(range(0, 2250 + 1, 250))
241     par2.set_yticks(range(0, 30 + 1, 5))
242
243     plt.grid(True, color=ax_bg, linestyle=style, linewidth=ln_width)
244     plt.draw()
245     plt.show()
246
247     show()
248
249 qtd = '5min' # 5min,15min,1hr
250
251 if qtd == '5min':
252     measure_time = 300
253 elif qtd == '15min':
254     measure_time = 900
255 elif qtd == '1hr':
256     measure_time = 3600
257 else:
258     measure_time = 20
259
260 # marks_by_detector_speed('1385', '2011-10-17', qtd, 0.0)
261 # marks_by_detector_density('1385', '2011-10-17')
262 create_log('1385', '2011-10-17', qtd, measure_time)
263 graphics()
```