Encontrando estacionamentos de carros: uma aplicação de névoa computacional

Douglas Afonso de Sousa Costa

Trabalho de Conclusão do Curso

de

Engenharia de Computação

no

Instituto de Computação Universidade Estadual de Campinas

Encontrando estacionamentos de carros: uma aplicação de névoa computacional

Douglas Afonso de Sousa Costa

ra104825@students.ic.unicamp.br

Supervisor: Edmundo Madeira

Campinas
2015

Sumário

Dedico este trabalho a todos que se dispuseram a me ajudar nessa caminhada até aqui, a todos que compartilharam comigo seus sucessos e falhas, tempo e sabedoria. Que essas pessoas continuem na minha vida.

”It’s fine to celebrate success but it is more important to heed the lessons of failure.” Bill Gates

Abstract. *Basing in a model of sensors and servers in the parking lots of establishments (shopping malls, restaurant, banks, etc), was designed a Cloud and Fog infrastructure that, along with the vehicular networks, are a fast and low latency communication network that is, the common person, a fast and reliable way to find a parking spot, saving time, fuel and generating less traffic.*
For this problem, I designed a simplified simulation of a vehicle that changes its routes and an application that show available parking spots near its destination. The simulation was made using SUMO (Simulation of Urban Mobility) by its Java interface TraCI4J.

Resumo. *Baseando-se num modelo de sensores e servidores presentes nos estacionamentos de estabelecimentos (shoppings, restaurantes, bancos, etc), foi desenvolvido uma estrutura de Cloud(nuvem) e Fog(névoa) que, junto com as redes veiculares, formam uma rede de conexão de baixa latência que dá o usuário uma maneira rápida e confiável de achar uma vaga de estacionamento para seu veículo, possibilitando uma maior economia de tempo, combustível e diminuição do tráfego.*
Para este problema, montei uma simulação simplificada de um veículo que muda suas rotas e uma aplicação que mostra vagas estacionamentos disponíveis próximo de seu destino. A simulação foi feita usando o SUMO (Simulation of Urban Mobility) através de sua interface para Java, TraCI4J.

1. Introdução

Se te perguntarem neste momento aonde está seu carro, provavelmente a resposta será: ”está estacionado”. Segundo o *International Parking Institute* (IPI), veículos privados passam 95% do tempo parados numa vaga enquantos seus donos trabalham, estudam, comem, dormem, fazem compras ou alguma outra atividade rotineira. Estamos sempre nos movendo, somos afetados diretamente pela localização onde deixamos nossos veículos perto de nossos destinos.

A cidades estão crescendo, estima-se que em 2030 60% da população mundial viverá em cidades[1]. No terceiro trimestre de 2013 foi estimado que 247,9 milhões de carros caminhões circularam nas ruas, e é esperado que esse numero atinja 284 milhões próximo do ano de 2025.
Em um estudo realizado numa área de 15 quarteirões na área de Los Angeles, em um ano foram medidas 950 mil milhas rodadas de veículos a procura de vagas. Para se ter noção da quantidade, isso é equivalente a 38 voltas no Planeta Terra.

Não podemos deixar de citar que além dos problemas do trânsito, isso tem um impacto imenso na quantidade de emissão de gases de efeito estufa. Várias cidades e grandes centros já estão tomando iniciativas e considerando vagas para estacionar uma grande preocupação em seus planejamentos.

Com isso em mente, foi pensado um modelo de aplicação que roda na névoa e se comunica com os estabelecimentos com estacionamentos para encontrar vagas para o motorista [2] e feito uma simulação simplificada da aplicação funcionando usando a ferramenta SUMO [3].

2. Conceitos

Aqui usamos alguns conceitos de IoT e VANETs, Nuvem e Névoa (Fog).

2.1. Internet of Things

Internet of Things descreve uma visão da rede mundial de computadores como uma rede central acessada por todo tipo de *smart objects*, ou seja, qualquer dispositivo esperto o suficiente para se conectar a internet e trocar informações com outros dispositivos.
Recentemente a Cisco expandiu sua definição para Internet of Everything (IoE), onde além das coisas inclue pessoas, processos e dados [4].
Em nosso modelo temos variados dispositivos, desde veículos e servidores até sensores de vagas, que conectados por Bluetooth ou WiFi informarão se temos ou não vagas disponíveis.

2.2. VANET

Restringindo para o caso dos veículos, podemos falar de VANET (Vehicular ad hoc network) que é uma rede distribuída de intercomunicação entre os veículos[5]. Além de entre si, comunicam-se também com as RSUs (Roadside Units), unidades a beira da estrada que concentram informações locais, têm acesso a internet e se comunicam com a Roadside Cloud, a unidade central das RSUs responsável pelo maior processamento de informações. Podem entrar nas VANETs outros dispositivos, como semáforos, praças de pedágios e centrais de trânsitos.

As VANETs foram pensadas para diminuir a latência da obtenção de informações: ao invés de todos se concentrarem numa nuvem central, as informações são obtidas localmente com requisições de baixas latências resultando em informações mais precisas e tomadas de decisões mais rápidas.
As comunicações entre veículos são usadas para traçar rotas de auto-guiados e para se evitar colisões. No caso de um veículo colidir no meio da via, este consegue imediatamente avisar os mais próximos da via a tempo de evitar acidentes, o que talvez não fosse possível se tivesse de aguardar a nuvem avisar os outros veículos.

2.3. RSU

As RSUs são para informações locais, no exemplo acima do acidente, o veículo parado também informaria a RSU mais próxima que passaria a

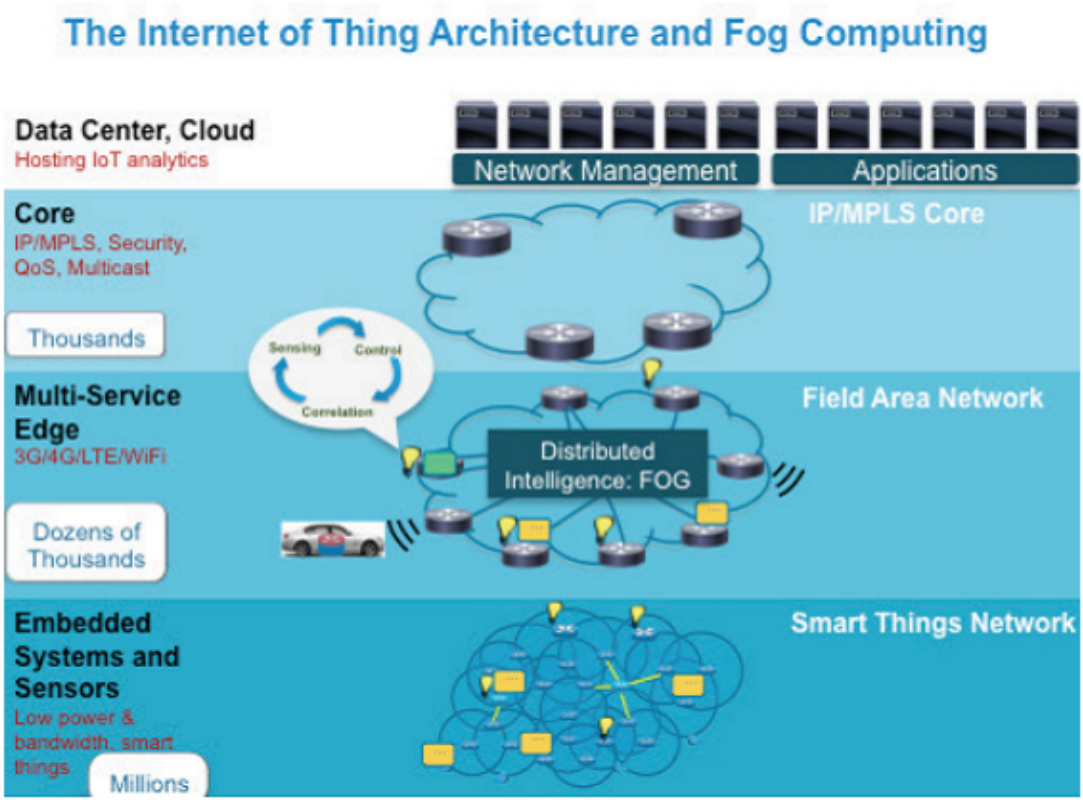

Figura 1. Níveis de abstração da rede.

informação adiante, fazendo com que fosse possível avisar para outros carros da via antecipadamente sobre um possível engarrafamento e sugerir rotas alternativas.
Estes dados também seriam sincronizados com a Roadside Cloud para que esta pudesse acionar as autoridades competentes (ambulância, bombeiros, polícia) e mais tarde sejam analisados mais de uma maneira mais específica pelo departamento de trânsito ou pela concessionária da via, como por exemplo detectar áreas de maior perigo e decidir onde colocar semáforos, rotatórias ou diminuir a velocidade máxima permitida no trecho.

2.4. Névoa computacional

Tudo isso nos leva ao conceito de Névoa (Fog) que deriva seu comportamento da Nuvem (Cloud) e o realiza de forma mais distribuída nas bordas da rede. O termo névoa é por ser uma nuvem mais próxima do chão, que é análogo ao seu conceito de uma nuvem menos densa e mais distribuída [6].

Se caracteriza por ser uma plataforma altamente virtualizada que provê processamento, armazenamento e comunicação entre os dispositivos

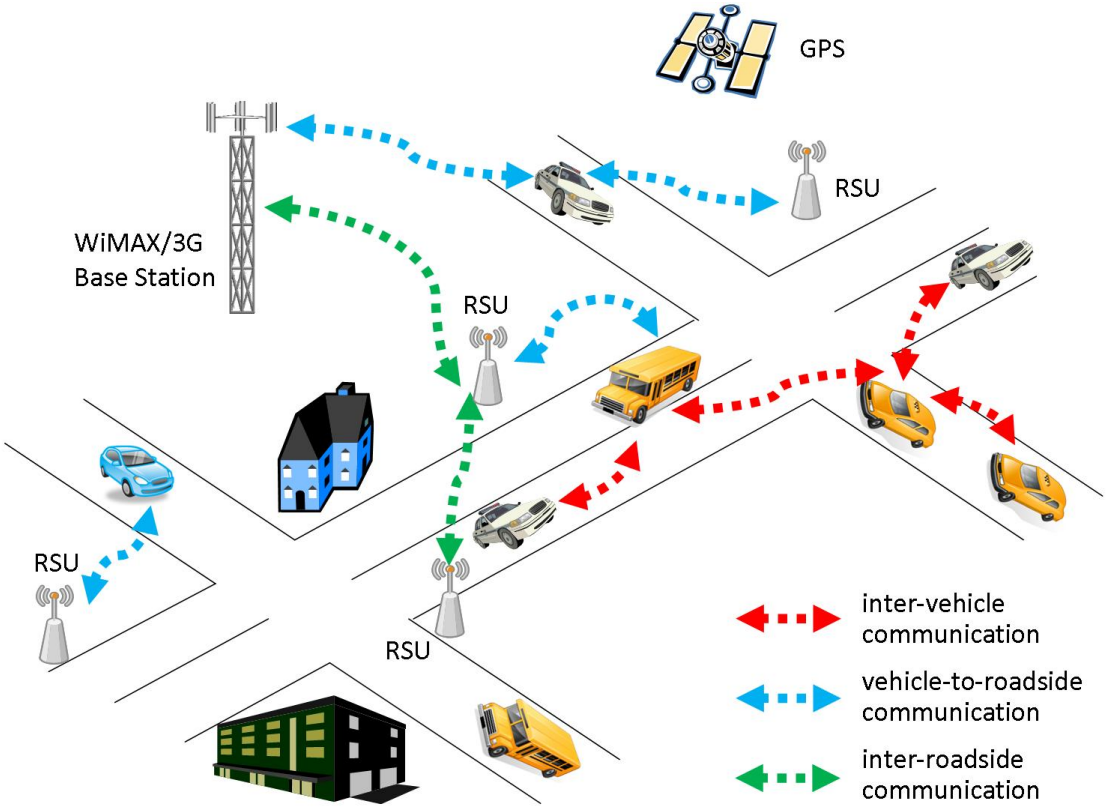

Figura 2. Comunicações numa VANET.

finais e a nuvem. Se caracteriza pelas seguintes características:

- Localização na borda da rede;
- Distribuída geograficamente: em maior quantidade e em mais locais;
- Consciência de seu local: sabe de sua localização e ela importa no processamento e tomadas de decisão;
- Baixa latência: consequência da menor distância geográfica dos dispositivos;
- Menor tempo de resposta: como tem em maior quantidade, cada uma processa bem menos quantidade de informações.

Com a névoa podemos desafogar a nuvem, termos repostas mais rápidas e ainda a maior confiabilidade nos dados pelo fato de serem avaliadas informações mais específicas sobre o local.

3. Modelo

O modelo proposto em [2] foi pensado para vagas de estacionamento compartilhadas entre vários tipos de estabelecimentos. Sendo estes próximos mas de seguimentos variados poderiam compartilhar vagas de estacionamento, visto que operariam com maior público em horários diferentes. Por exemplo uma casa noturna pode ter vagas compartilhadas com um banco ou salão de beleza. O salão e o banco têm públicos diurnos, enquanto a outra possui maior movimento a noite. Na prática isto já existe, porém são apenas acordos informais e convênios fechados entre poucos estabelecimentos.

O modelo propõe uma aplicação rodando que recebe as solicitações de vagas dos usuários e se comunica com os estabelecimentos para descobrir vagas. Essa aplicação estaria rodando no nível de névoa, pois seria responsável por apenas uma área definida, com outros servidores rodando para outras localizações.

Essa aplicação terá acesso a um banco de dados, com os estacionamentos em sua área de atuação previamente cadastrados e o endereço de comunicação com seus servidores (ip e porta).
Ao ser solicitado uma vaga pelo usuário que está em sua área, a aplicação analisa o destino de sua rota e se também for dentro dentro de seu alcance, ela se comunica com os estacionamentos próximos e informa o usuário.
Se o destino do usuário for para outra localização, o servidor deste local informa o servidor responsável.

Para cada estabelecimento teríamos um servidor de baixo custo responsável pela gerência das vagas dos mesmos. A ideia é que sensores sejam usados nas vagas e se comuniquem com este servidor local por tecnologias sem fio.
Estes servidores estariam conectados a internet e previamente cadastrados num banco de dados e receberiam requisições da aplicação por vagas disponíveis em seus estabelecimentos. Estabelecimentos no plural, pois os próximos poderiam compartilhar a mesma máquina física de baixo custo como servidor, podendo conter várias VMs cada uma com instância escutando requisições em portas diferentes.

Os sensores teriam a capacidade de saber se a vaga está ocupada ou não. Podem ser utilizados sensores simples de proximidade com comunicação Bluetooth, ou qualquer outra tecnologia sem fio, até sistemas de análise de

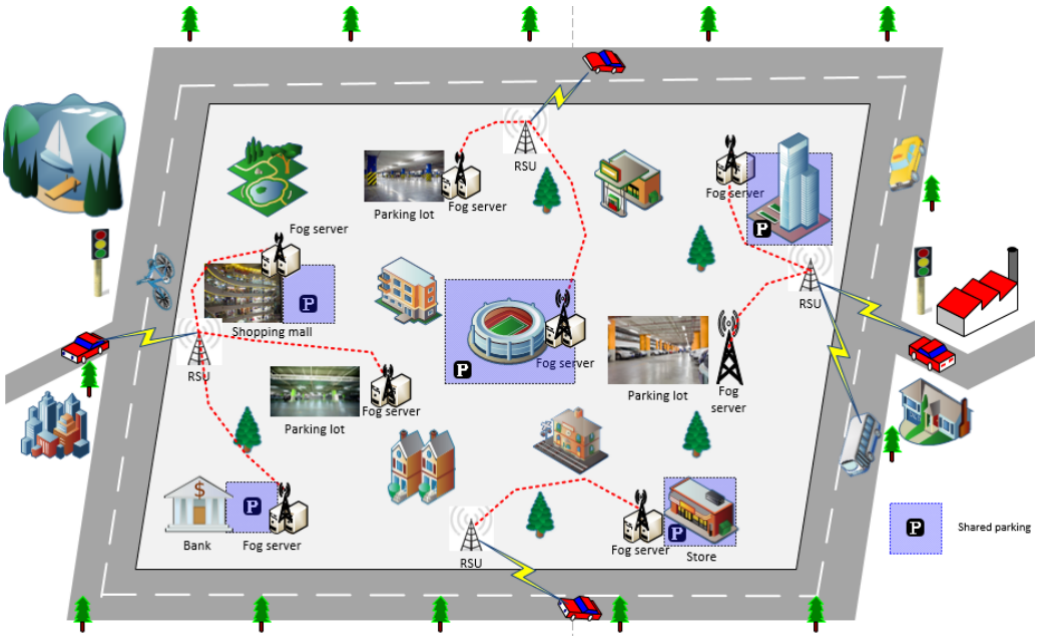

Figura 3. Esquema representando as comunicações entre servidores Fog, RSUs e a Cloud[2].

imagem usando as próprias câmera de vigilância.

4. Ferramentas

4.1. SUMO

A simulação facilita a avaliação de modelos de tráfego propostos antes de sua implementação, por exemplo a adição de um semáforo ou cruzamento na via pode ser testado e otimizado antes de ser aplicado no mundo real.
O SUMO (Simulation of Urban Mobility) [3] é uma ferramenta Open Source criada em 2001 que possibilita a modelagem e testes do modelo como um todo, incluindo veículos, pedestres e transporte público.
Para a simulação, fui utilizado a versão 0.24 do SUMO e a 2.4 do TraCI4J.

Sua configuração é feita em vários arquivos no formato xml, sendo possível informar a rota de cada veículo, sua velocidade média ou instantânea, o mapa das vias, cores dos veículos, os POIs (Pontos de Interesse) que representam os estacionamentos, velocidade da simulação e muitas outras.

A simulação é feita em passos, a cada passo os veículos avançam conforme sua rota, é possível analisar o estado da simulação, a posição de cada veículo, sua velocidade, se houve colisão ou mesmo forçar uma e verificar o tráfego gerado por ela.
É uma ferramenta muito poderosa, sendo possível analisar até o consumo de CO2 de cada veículo, mas para este trabalho não será considerado.

Possui versões para Windows e Linux, para a simulação foi utilizado no Ubuntu 14.04.

4.2. TraCI4J

Para controlar as simulações foram desenvolvidas várias APIs que se integram com as funções do SUMO. Chamadas de TraCI, existem APIs para Python [7], Java [8], C++ e ainda uma interface Webservice para ser usada por qualquer tipo de linguagem que suporte protocolo SOAP [9].

Neste trabalho utilizei a API para Java, chamada de TraCI4J. Ela ainda está em versão alfa e possui algumas limitações, mas para este caso foi suficiente.
Através dela temos acesso aos objetos na simulação: uma lista de veículos e lista de POIs (estacionamentos). Para cada objeto de veículo temos acesso a sua rota, a sua posição espacial naquele passo, a cor do carro, emissão de CO2, sua velocidade e podemos também alterar alguns desses parâmetros.

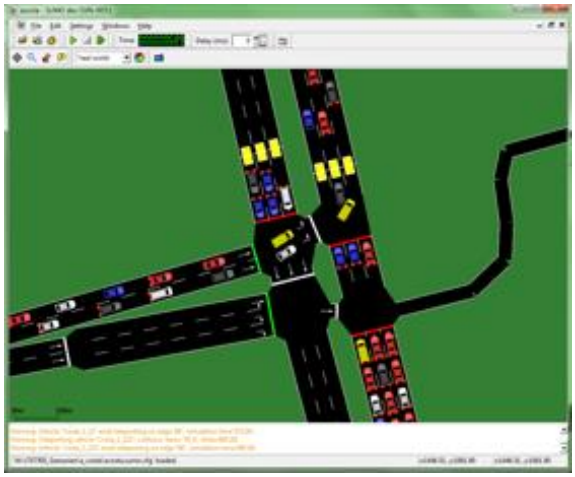

Figura 4. Interface gráfica SUMO.

Para este grau de complexidade da simulação, o TraCI4J supre muito bem como API. Foi utilizando junto com JDK 7.

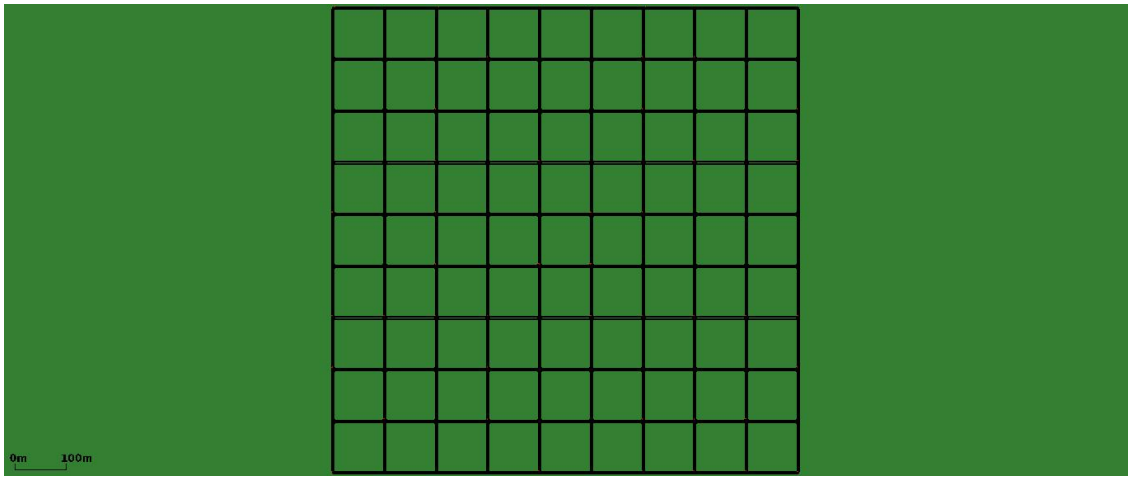

Figura 5. Área utilizada para simulações.

5. Desenvolvimento e experimentos

Para as simulações utilizei um mapa quadrado de 9x9 quarteirões, com 100m^2 cada. No pacote temos simulações de 200 até 600 carros, mas por motivos visuais, foi utilizado menos carros.
Além dos carros, na simulação foram inseridos POIs (Point of Interest) que representam os estacionamentos cadastrados.

Na simulação podemos ter uma visão macro e simplificada do ecossistema. Como foi dito, a aplicação rodará na névoa local, que se comunica com algum usuário aquela área. Esta simulação foca, no comportamento do veículo e sua comunicação com a névoa.
Simplificando, consideramos um servidor por estacionamento. A comunicação com os servidores foi simulada como uma escrita em arquivo, quando a aplicação solicita o número de vagas ela dispara um script que escreve num arquivo de texto um número aleatório que representa a quantidade de vagas disponíveis naquele estacionamento. A aplicação lê esse valor e o considera nas decisões.

No início da simulação é escolhido um carro aleatoriamente que vai representar o usuário que precisa de uma vaga de estacionamento. Na interface gráfica este veículo será da cor amarela, enquanto que os outros serão rosa claro.
No caso dos POIs, os que não estão disponíveis no momento para aquele veículo estarão com a cor vermelha, se estiverem disponíveis serão da cor verde claro.

,

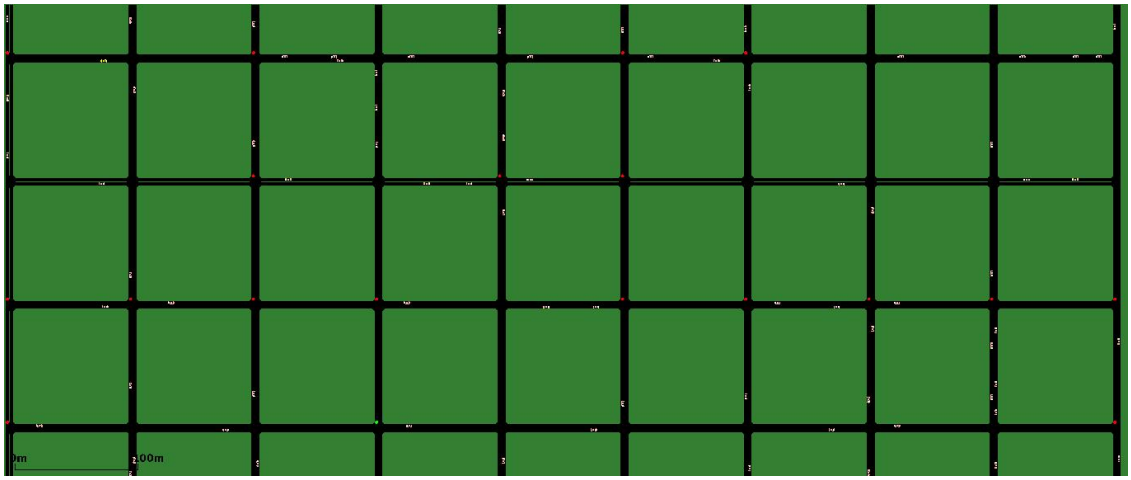

Figura 6. Área com 200 carros inseridos na simulação.

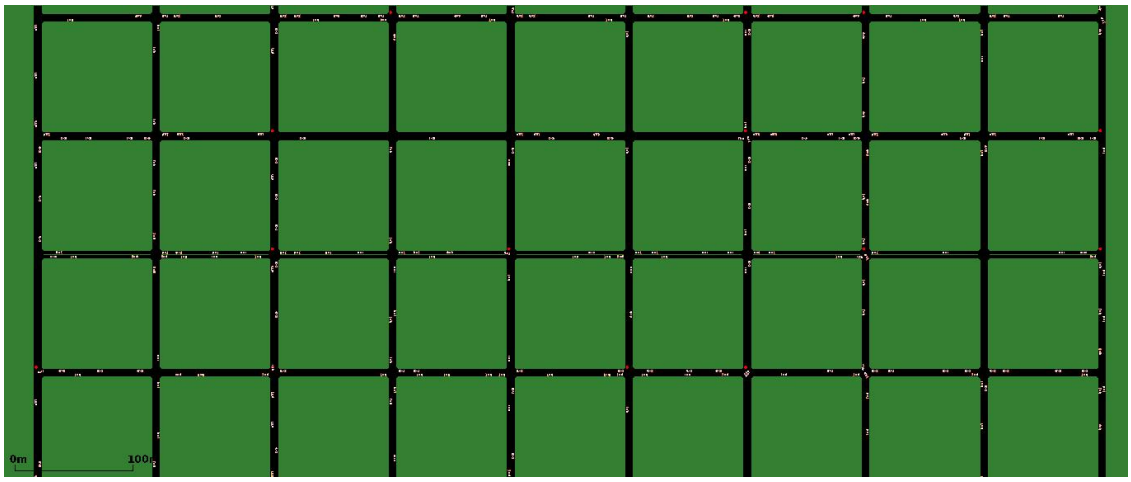

Figura 7. Área com 600 carros inseridos na simulação.

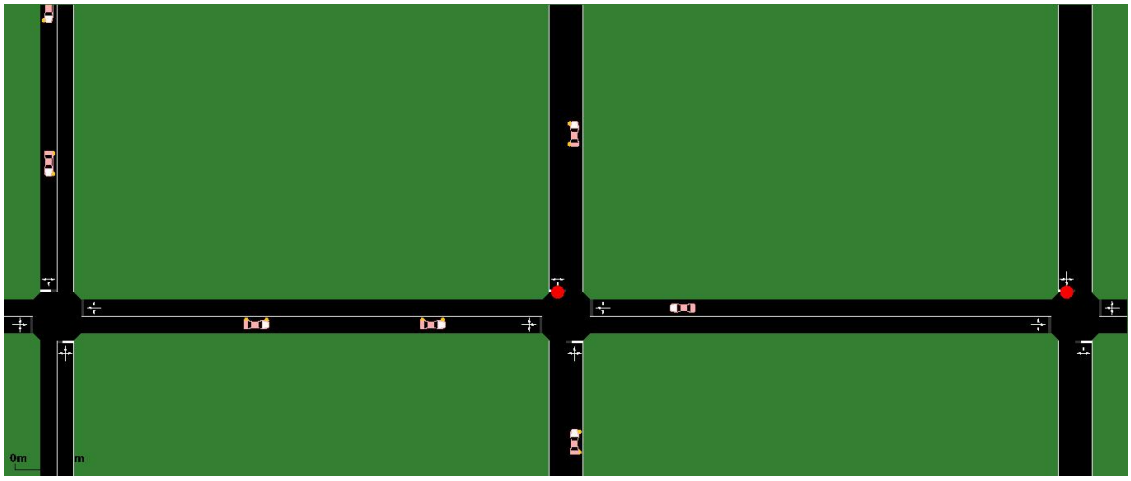

Figura 8. Carros e estacionamentos na simulação.

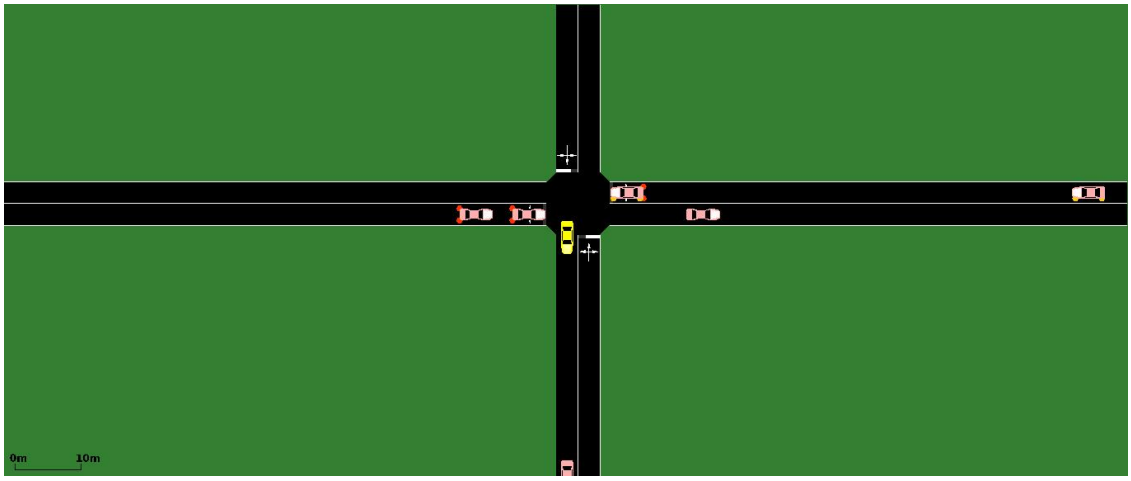

Figura 9. Veículo escolhido na simulação (cor amarela).

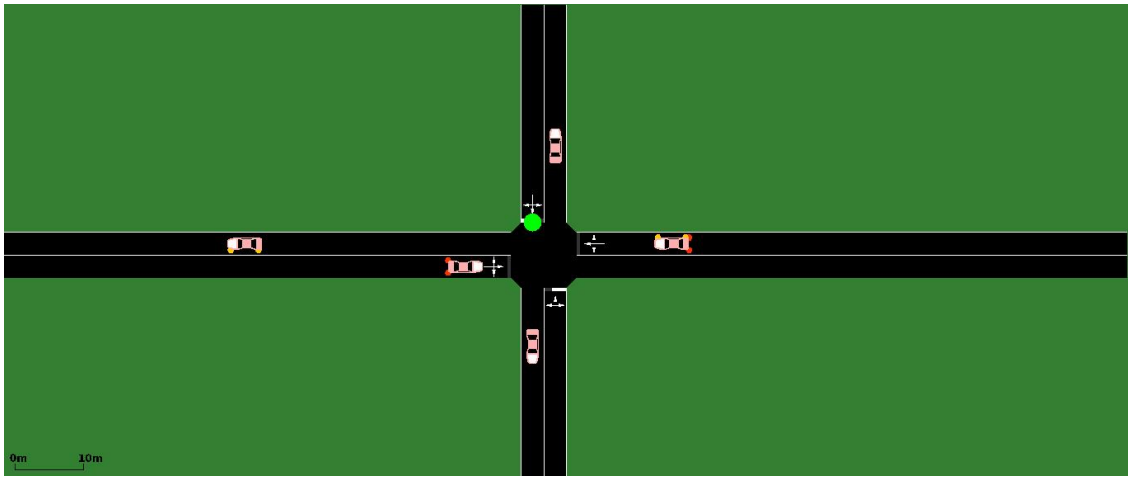

Figura 10. Estacionamento disponível próximo do destino do veículo escolhido (cor verde claro).

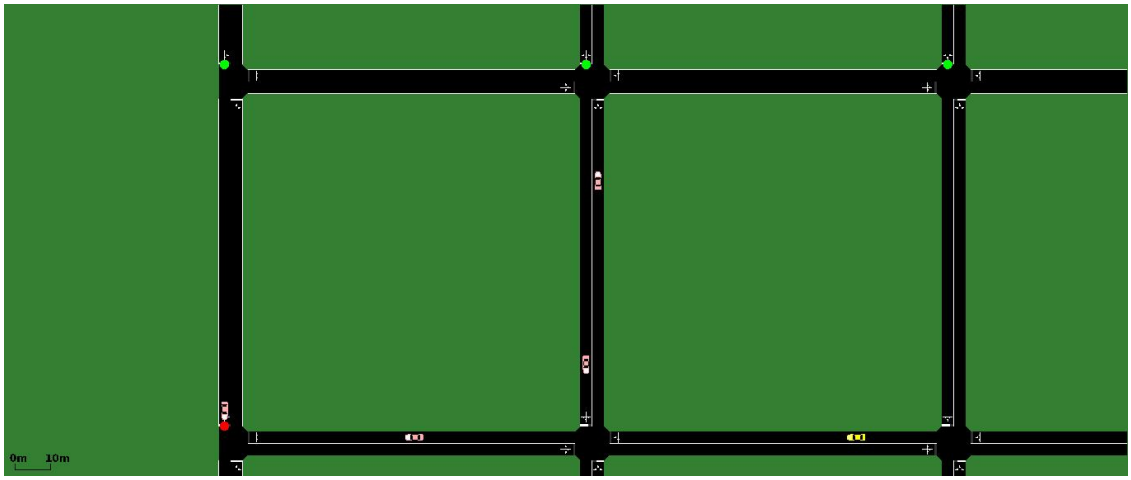

Figura 11. Carro escolhido próximo de seu destino e com estacionamentos disponíveis.

Logo que o carro é inserido na simulação, é levantado uma flag que representa a solicitação de vaga por parte do usuário para a aplicação. Quando a aplicação detecta a flag, ela analisa a rota do veículo e baseado no seu destino informa os estacionamentos com vagas disponíveis naquele momento.

A aplicação verifica se há estacionamentos num raio de 110m (pouco mais que um quarteirão) de distância do destino do veículo. Se houver ele simula uma comunicação com o servidor do estacionamento perguntando por vagas e este responde quantas ele tem disponível. Se possuir pelo menos uma vaga disponível, o POI que o representa é pintado de verde claro, caso contrário mantem-se na cor vermelha.

Para efeitos de simulação, foi configurado para o veículo escolhido mudar sua rota a cada 40 passos. Quando isto acontece, é levantada a flag novamente, que representa a solicitação de vaga para a aplicação na névoa e este refaz o processo de verificação de vaga.

A simulação termina quando o carro principal chega em seu destino (e provavelmente estaciona).

Para o deploy da névoa, foi feito pensado varias VMs Ubuntu (ou Debian), cada uma rodando uma instância do servidor com a aplicação em questão responsável por uma área de de $900m^2$.
Caso o destino do usuário esteja em outra área, a comunicação é simulada por escrita e leitura em arquivos.

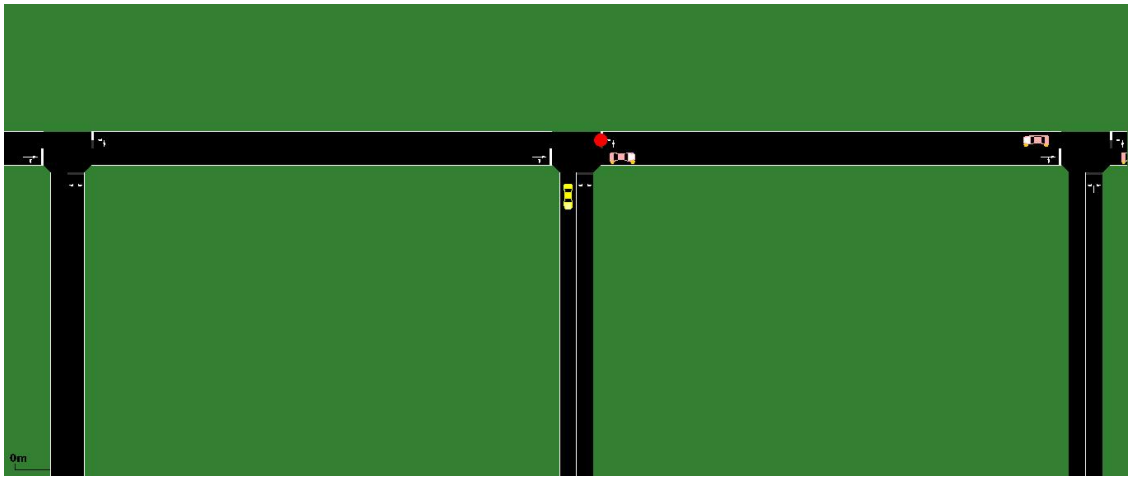

Figura 12. Passo 1: carro é inserido na simulação e começa sua rota.

```
Following vehicle 123
Waiting for vehicle to appear...
Vehicle found!
Step 1
Destination: 3/0
Parking lots available near 3/0 : 31(5 vagas) 20(9 vagas)
```

Figura 13. Passo 1: aplicação encontra vagas próximo de seu destino na esquina 3/0 (três quarteiros na horizontal e 0 na vertical).

6. Resultados

Podemos ver o exemplo da simulação da figura 12 em diante. As três primeiras são do primeiro passo, onde o carro é adicionado na simulação e a aplicação descobre estacionamentos disponíveis pela primeira vez.
Nas outras três temos o passo 40 que é onde o veículo resolve mudar a rota e avisa a aplicação através da flag e esta recalcula estacionamentos disponíveis.
Na última imagem podemos ver o carro já próximo de seu destino com 2 estacionamentos próximos com vagas para poder escolher.

Uma pendência importante no trabalho é que, até esta data que lhes escrevo, não foi possível configurar o ambiente para rodar uma simulação com mais de uma VM de névoa e fazê-las comunicarem entre si. Portanto as simulações rodaram com destinos apenas dentro de sua área de atuação, como algo modular. Mas acredito que com a unidade funcionando, não deve ser difícil escalá-la.

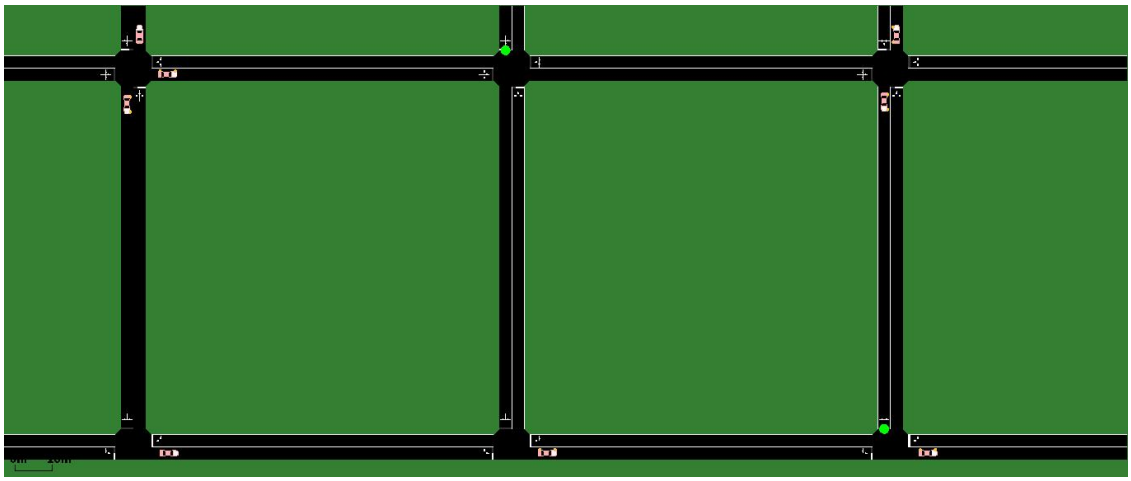

Figura 14. Passo 1: estacionamentos disponíveis na esquina 3/0.

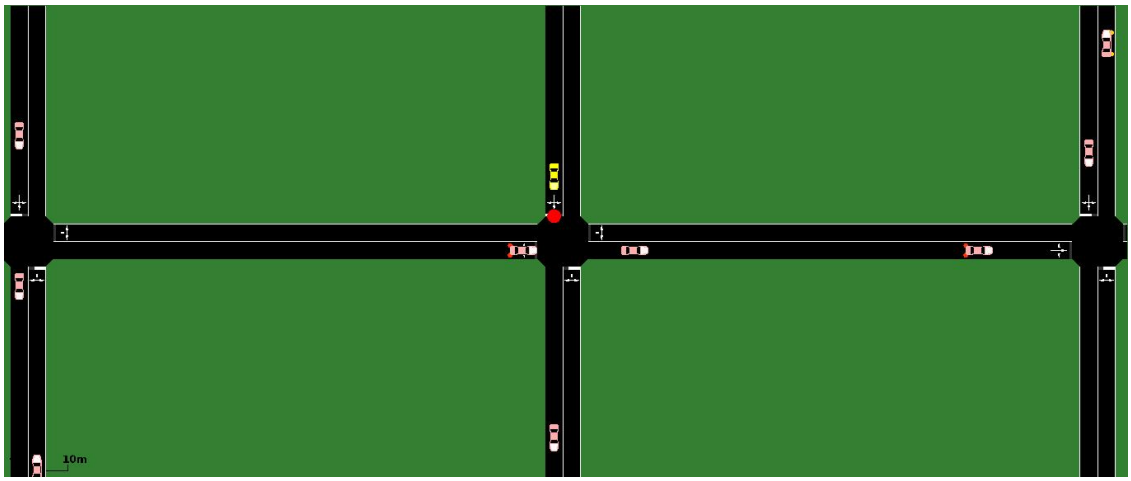

Figura 15. Passo 40: carro ainda não chegou ao seu destino, mas muda sua rota.

```
Step 38
Step 39
Step 40
Old target 2/0to3/0
New target 8/2to9/2
Destination: 9/2
Parking lots available near 9/2 : 30(8 vagas) 10(1 vagas)
```

Figura 16. Passo 40: Log da aplicação ao encontrar os estacionamentos perto do novo destino 9/2.

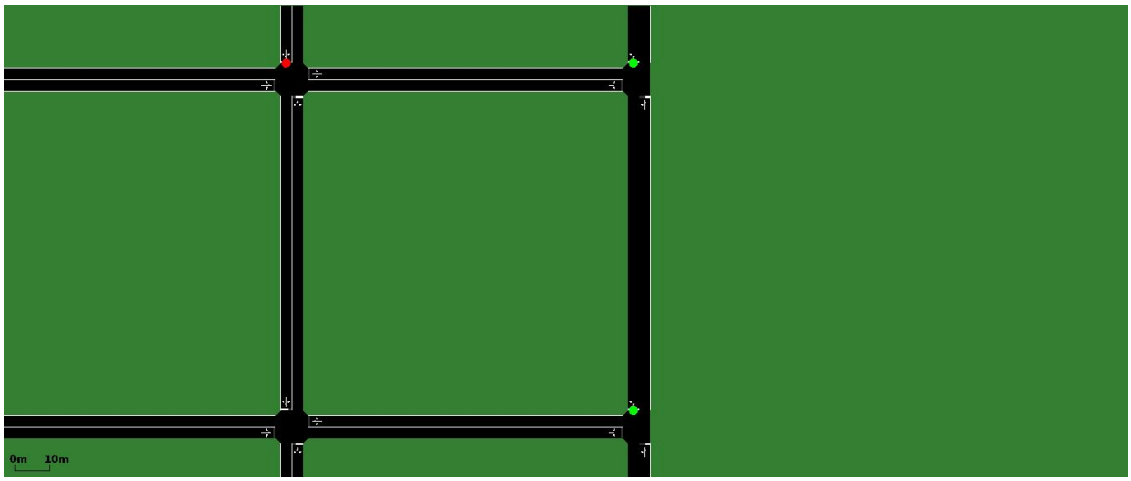

Figura 17. Passo 40: estacionamentos disponíveis no destino 9/2.

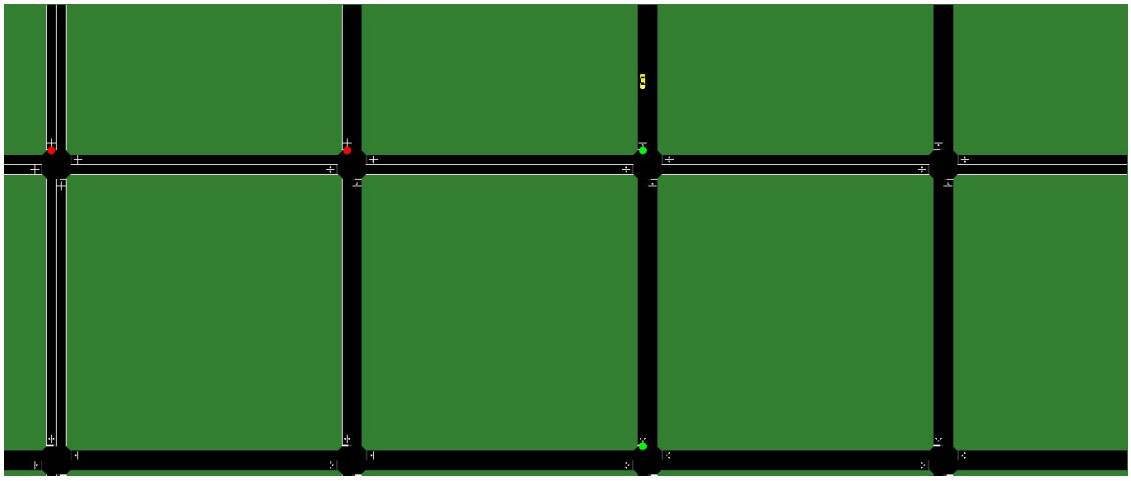

Figura 18. Passo final: veículo em seu destino final próximo dos estacionamentos.

7. Trabalhos futuros

8. Conclusão

9. Código-fonte

Todo código da aplicação e simulação está disponível em

https://github.com/douglasascosta/parkinglot-cloud

Referências

[1] International Parking Institute. Why parking matters. *The Role of Parking in Our Future Cities*, 2014.

[2] VanDung Nguyen Nguyen H. Tran Choong Seon Hong Oanh Tran Thi Kim, Nguyen Dang Tri. A shared parking model in vehicular network using fog and cloud environment. *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, pages 321–326, 2015.

[3] Institute of Transportation Systems. Simulation of urban mobility. `http://sumo.dlr.de/wiki/Main_Page`, 2015. [Online; accessed 09-November-2015].

[4] Cisco. Value of the internet of everything for cities, states & countries. `http://internetofeverything.cisco.com/vas-public-sector-infographic/`, 2015. [Online; accessed 09-November-2015].

[5] George Corser. Vanet introduction. `https://www.youtube.com/watch?v=DrH-1505-Mg`, 2013. [Online; accessed 09-November-2015].

[6] Jiang Zhu Sateesh Addepalli Flavio Bonomi, Rodolfo Milito. Fog computing and its role in the internet of things. 2014.

[7] Institute of Transportation Systems. Traci for python. `http://sumo.dlr.de/wiki/TraCI/Interfacing_TraCI_from_Python`, 2015. [Online; accessed 09-November-2015].

[8] Enrico Gueli. Traci4j. `https://github.com/egueli/TraCI4J`, 2015. [Online; accessed 09-November-2015].

[9] Institute of Transportation Systems. Traas - traci as a service. `http://traas.sourceforge.net/cms/`, 2015. [Online; accessed 09-November-2015].