LUIZ AUGUSTO GUIMARÃES COSTA

ANÁLISE DA UTILIZAÇÃO DE AGENTES DE SOFTWARE EM REDES VEICULARES HÍBRIDAS UTILIZANDO ZIGBEE

LAVRAS - MG
2015

LUIZ AUGUSTO GUIMARÃES COSTA

ANÁLISE DA UTILIZAÇÃO DE AGENTES DE SOFTWARE EM REDES VEICULARES HÍBRIDAS UTILIZANDO ZIGBEE

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Ciência da Computação, para obtenção do título de Mestre.

Orientador
Dr. Wilian Soares Lacerda

LAVRAS - MG
2015

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Costa, Luiz Augusto Guimarães.
Análise da utilização de agentes de software em redes veiculares híbridas utilizando ZigBee / Luiz Augusto Guimarães Costa. – Lavras : UFLA, 2015.
68 p. : il.

Dissertação (mestrado acadêmico)–Universidade Federal de Lavras, 2015.
Orientador: Wilian Soares Lacerda.
Bibliografia.

1. Transporte de Agentes. 2. Comunicação Oportunistas. 3. Agentes de Software. 4. Redes Ad Hoc. 5. VANET. 6. MANET. I. Universidade Federal de Lavras. II. Título.

LUIZ AUGUSTO GUIMARÃES COSTA

ANÁLISE DA UTILIZAÇÃO DE AGENTES DE SOFTWARE EM REDES VEICULARES HÍBRIDAS UTILIZANDO ZIGBEE

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Ciência da Computação, para obtenção do título de Mestre.

APROVADA em 27 de fevereiro de 2015.

Dr. Luiz Henrique Andrade Correia	UFLA
Dr. Daniel Fernandes Macedo	UFMG

Dr. Wilian Soares Lacerda
Orientador

LAVRAS - MG
2015

Dedico esse trabalho ao meu pai José Augusto, minha mãe Piedade, a minha irmã Fernanda e a minha noiva Ludhiana.

AGRADECIMENTOS

Agradeço a minha familia por me ajudar com motivação para finalizar esse trabalho.

Agradeço ao meu orientador Wilian e ao professor Luiz Henrique.

Agradeço ao pessoal do LEMAF por ter me apoiado quando eu precisei de tempo para estudar.

Agradeço ao pessoal da GT4W por ter permitido que eu realizasse o mestrado sem me preocupar.

RESUMO

As redes veiculares *ad hoc* (*Vehicular Ad Hoc Networks* - VANETs) são redes com alta mobilidade onde veículos podem se comunicar um com os outros ou com infra-estrutura nas vias, como também podem transportar dados fisicamente de um ponto ao outro. Como existe uma grande mobilidade dos veículos em uma cidade, a rede veicular precisa de uma capacidade de adaptação rápida ao contexto. Os agentes de software encaixam nesse tipo de problema. Os agentes de software possuem a capacidade de controlar o próprio comportamento e interagir com o ambiente. Podendo se adaptar ao contexto trocando mensagens com os veículos vizinhos e recuperando informações do ambiente. O objetivo desse trabalho é analisar o desempenho de agentes de software em redes veiculares construídas utilizando a abordagem ad hoc e híbrida. Para analisar o desempenho das duas abordagens foram realizadas simulações utilizando o software GRUBiX. Nas simulações os resultados demonstraram que os agentes apresentam desempenho melhores em redes híbridas que em redes ad hoc. Então foi desenvolvido um protótipo baseado em kit com Arduíno e Zigbee para avaliar a viabilidade de uma rede veicular utilizando agentes de software para transportar dados. Os resultados demonstraram a viabilidade na utilização de agentes de software para o desenvolvimento de aplicações em redes veiculares construídas com o padrão Zigbee.

Palavras-chaves: Transporte de agentes. Comunicação Oportunista. Agentes de Software. Redes Ad Hoc. VANET. MANET.

ABSTRACT

Vehicular *Ad Hoc* Networks (VANETs) are high mobility networks in which vehicles can communicate with each other or with infrastructures on roads, in addition to allowing them to physically carry data from one point to another. As there is high mobility of vehicles within a city, the vehicular network needs an ability to rapidly adapt to the context. Software agents fit into this kind of issue. They have the ability to control their own behavior and to interact with the environment. Being able to adapt to the context by exchanging messages with neighboring vehicles and retrieving information from the environment. The objective of this study was to analyze the performance of software agents in vehicular networks built using the ad hoc and hybrid approach. To analyze the performance of both approaches, we performed simulations using the GRUBiX software. In the simulations, the results showed that the agents perform better in hybrid networks than in ad hoc networks. Therefore, we developed a prototype based on a kit with Arduino and Zigbee in order to evaluate the feasibility of a vehicular network using software agents to carry data. The results showed the feasibility of using software agents for developing applications in vehicular networks built with the Zigbee pattern.

Keywords: Agent Ferrying. Opportunistic Communication. Software Agents. Ad Hoc Networks. VANET. MANET.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

UML	Unified Modeling Language
VR	Velocidade Relativa
VANET	Veicular Ad Hoc Networks
GPS	Global Positioning System
DTN	Delay-tolerant networking
MANET	Mobile Ad Hoc Networks

SUMÁRIO

1 INTRODUÇÃO

Comunicação sem fio permite novas aplicações que exploram a transferência de dados entre nós móveis. Redes oportunistas são redes onde a comunicação não é continua, isto é, nunca deve existir um caminho pré-definido entre o nó de origem e o nó destino. Isto proporciona suporte para os cenários em que os dados são transportados a partir de uma fonte para um destino através dos nós móveis que levam fisicamente os dados de um lugar para outro, com ou sem auxilio de uma infraestrutura fixa. Essas características podem ocorrer em uma série de aplicações tolerantes ao atraso (FALL, 2003). Entretanto, esse paradigma exige um maior número de nós para auxiliar o transporte dos dados, assim melhorando a probabilidade de eles serem eventualmente entregues. Neste ponto, a utilização dos nós mais promissores para transportar os dados proporcionam uma maior probabilidade de esses dados alcançarem o seu destino (FREITAS et al., 2013).

O uso de dispositivos móveis na formação de uma rede *ad hoc* abriu um novo horizonte para exploração de técnicas de comunicação oportunísticas. Um exemplo de comunicação oportunística acontece quando dados são transportados por nós móveis de uma origem para um destino utilizando-se a movimentação física do nó. As conexões temporárias entre nós que estão próximos geograficamente durante um pequeno intervalo de tempo são explorados sem a existência de uma rede conectada constantemente. Esse tipo de comunicação exige da aplicação tolerância a atrasos consideráveis e desconexões frequentes. Muitos cenários reais podem fazer uso dessa abordagem.

O uso de agentes móveis representa uma tecnologia promissora (URRA et al., 2010) para tratar os problemas aqui apresentados. O seu uso permite o encapsulamento de inteligência na rede, melhorando as decisões que serão responsáveis pelo tráfego de dados (FREITAS et al., 2010). Essa abordagem oferece flexibilidade para elaborar estratégias de transporte dos dados, assim permitindo enfrentar possíveis problemas encontrados de forma mais ágil.

O transporte de agente utiliza o mesmo principio de transporte de dados, mas em vez de apenas dados, um agente também transfere o seu comportamento.

A vantagem de usar agentes para transportar dados é que eles não são limitados a um protocolo de roteamento instalados nos nós, mas como parte de sua própria inteligência, eles podem fornecer a sua própria estratégia para mover através dos nós na rede. Isto é, a responsabilidade da decisão de o dado migrar ou não de um nó para outro é delegada ao agente, assim aumentando a capacidade de contornar as adversidades proporcionadas por ambientes instáveis.

Com essa abordagens as redes veiculares deixam de ser construídas para uma determinada finalidade, deixando para o software do agente definir a finalidade. Assim a mesma rede onde existe um ciclista com equipamentos que permitem o envio de agentes que avisam os veículos de sua presença pode ser utilizada por um hotel que oferece quartos para motoristas cansados.

A versatilidade do agente aumenta se os veículos possuírem sensores que possam captar a situação do condutor do veículo, por exemplo se ele está embriagado avisar a polícia. Outra aplicação é em caso de acidente o próprio veículo cria agentes que avisam os veículos vizinhos, cria outro agente para buscar um veículo da policia nas proximidades e outro para buscar uma ambulância mais próxima.

1.1 Objetivo

O objetivo principal desse projeto é a exploração de agentes de software para encaminhar dados utilizando informação geográfica sobre uma VANET. Pretende-se estudar o impacto da utilização da infraestrutura fixa sobre o agente de software.

Como um segundo objetivo é avaliar o padrão ZigBee quanto aos requisitos necessários para a criação de uma VANET capaz de transportar agentes.

Serão propostas duas redes veiculares *ad hoc* (VANET), uma rede híbrida e outra somente com veículos. Os agentes serão desenvolvidos no ambiente de simulação GRUBIX, desenvolvido pelo Grupo de Redes Ubíquas do Departamento de Ciência da Computação da Universidade Federal de Lavras, baseado no simulador Shox (LESSMANN; HEIMFARTH; JANACIK, 2008).

Para realizar a avaliação dos agentes serão desenvolvidos dois cenários. O primeiro, uma rede somente com veículos e o segundo, uma rede híbrida. Ambos

os cenários serão examinados em uma cidade com quadras do mesmo tamanho, essa cidade é baseada no modelo de movimento *Manhattan* (BAI; SADAGOPAN; HELMY, 2003). O desenvolvimento dos dois cenários servirá para análise do desempenho dos agentes em ambientes diferentes e com distribuição dos veículos. Cada veículo presente nos cenários dispõe de um dispositivo de comunicação sem fio e um GPS (*Global Positioning System*) que serviram de infraestrutura para o agente.

Após a realização das simulações será desenvolvido um equipamento capaz de hospedar o agente. Também esse equipamento será avaliado nos dois cenários simulados.

1.2 Motivação

Segundo a (OMS, 2013), 1,24 milhões de pessoas morreram no trânsito em todo mundo em 2010. Um estudo (CINTRA, 2012) aponta que o trânsito caótico da cidade de São Paulo impõe um prejuízo de R$ 50 bilhões por ano ao Brasil. As redes sem fio *ad hoc* proporcionam um novo horizonte para a melhoria das condições do trânsito em grandes metrópoles. Essas redes podem recolher dados em tempo real sobre o trânsito, assim proporcionando aos condutores e autoridades informações importantes para favorecer a segurança no trânsito.

Entre várias aplicações possíveis em VANET, as que proveem maior segurança do trânsito são as que recebem maior atenção. Alguns situações de riscos poderiam ser evitadas caso os motoristas tivessem informações de eventos ou situações do trânsito em tempo real.

Na literatura existem poucos trabalhos práticos em VANET se comparado com os trabalhos envolvendo simulações. Isto acontece por que existem poucos equipamentos e os custos deles são elevados. Atualmente existe um padrão de redes veiculares IEEE 802.11p (JIANG; DELGROSSI, 2008). O padrão 802.11p pode ser atualizado assim que as pesquisas avançarem. Então o ZigBee pode ser uma nova direção para o desenvolvimento dessas redes (BHARGAV; SINGHAL, 2013).

As redes veiculares são um tópico em constante pesquisa pelo setor público e privado. Sua grande flexibilidade e suporte a várias aplicações para

melhorar a qualidade de vida nos grandes centros urbanos foram atributos que motivaram o desenvolvimento deste trabalho.

1.3 Estrutura do trabalho

Este trabalho está organizado da seguinte forma: os conceitos sobre redes veiculares, agentes de software e modelos de movimentos são apresentados na Seção 2. Os procedimentos para a realização das simulações e a especificação para a construção do protótipo são apresentados nas Seções 3 e 4. Na Seção 5 os resultados são apresentados e discutidos. Enfim, a conclusão do trabalho e proposta de continuidade são apresentados na Seção 6.

2 REFERENCIAL TEÓRICO

O presente capítulo apresenta os conhecimentos necessários para a compreensão dos assuntos relacionados ao trabalho.

2.1 Redes *Ad Hoc*

Uma rede *ad hoc* é definida como uma coleção de entidades móveis interligadas por uma tecnologia sem fio, formando uma rede temporária sem a ajuda de qualquer administração ou qualquer tipo de apoio fixo (ZAFOUNE; KANAWATI; MOKHTARI, 2007). É um tipo descentralizado de rede, visto que não depende de uma infraestrutura pré-existente, como roteadores em redes com fio ou pontos de acesso em redes sem fio (NIAZI; HUSSAIN, 2009). Outra característica é a capacidade de as redes *ad hoc* adequarem a condição da conectividade da rede, distribuindo a responsabilidade de organização e controle da rede para os nós, podendo determinar se um nó vai transmitir ou não dinamicamente (REZENDE; ROCHA; LOUREIRO, 2008).

As redes *ad hoc* possuem características próprias que as tornam adequadas a situações onde não existe uma infraestrutura de comunicação presente (NICULESCU; NATH, 2003). Elas também podem ser utilizadas em situações onde não seja possível utilizar nós críticos para organização ou controle da rede, pois o desempenho da rede *ad roc* não é afetado se um nó falhar ou mesmo sair da rede (DRESSLER, 2008).

2.2 Redes Tolerantes à interrupção

Segundo Nichols e Hammons (2007), *Delay-tolerant networking* (DTN) são redes projetadas para trabalhar em ambientes com comunicação instável ou temporária produzidos por limitações ou anomalias, causando o menor impacto negativo possível na aplicação.

O autor Burgess et al. (2006) sugere que rotas instáveis podem ser causadas por problemas como: a alta mobilidade dos nós, a baixa densidade de nós, uma curta faixa para transmissão de dados ou interferências ambientais.

A DTN é uma arquitetura de rede muito utilizada atualmente na exploração do espaço ou de planetas distantes (SPYROPOULOS et al., 2010), por que o espaço é um ambiente hostil que proporciona um longo período de latência no sinal, como também existe uma grande interferência eletromagnética.

2.3 Redes móveis *Ad Hoc*

A *Mobile Ad Hoc Networks* (MANET) é uma categoria de redes *ad hoc* (REZENDE; ROCHA; LOUREIRO, 2008) onde os nós são livres para deslocarem enquanto se comunicam com outros nós. Por causa da natureza dinâmica da MANET os dados que trafegam por ela podem ser corrompidos ou perdidos com facilidade (BAI; SADAGOPAN; HELMY, 2003).

Essa configuração de rede proporciona uma frequente perda de conexão entre os nós, portanto as MANETs precisam de protocolos que possam estabelecer uma boa qualidade de serviço (BAI; SADAGOPAN; HELMY, 2003). Essas características móveis permitem que grupos de celulares possam se conectar sem precisar de uma infraestrutura fixa somente criando uma rede *ad hoc* (PIORKOWSKI; SARAFIJANOVIC-DJUKIC; GROSSGLAUSER, 2009).

2.4 Redes veiculares Ad Hoc

As *Veicular Ad Hoc Networks* (VANET) são um conjunto de veículos que possuem capacidade computacional e estão ligados em uma rede. A VANET é um tipo de MANET que não possui recursos limitados, então não existe a necessidade de preocupação com uso de recursos como por exemplo energia, processamento e memória (LOULLOUDES; PALLIS; DIKAIAKOS, 2010).

A velocidade dos nós é outro fator importante no desenvolvimento de uma VANET (KUMAR; DAVE, 2011), partindo do princípio que veículos possuem uma velocidade muito maior de deslocamento que dispositivos móveis.

O autor Kumar e Dave (2011) afirma que as redes veiculares podem ser construídas de três maneiras. A primeira é utilizando um modelo celular onde pontos fixos espalhados funcionam como ponte para conexão dos veículos. O segundo é uma rede onde as intersecções dos veículos são pontes para transmissão

dos dados. A terceira maneira para construir uma rede veicular é uma técnica híbrida com a rede *ad Hoc* e celular onde existem os pontos fixos e a intersecções para trafegar os dados entre os veículos. Esse modelos são exemplificados na Figura 1.

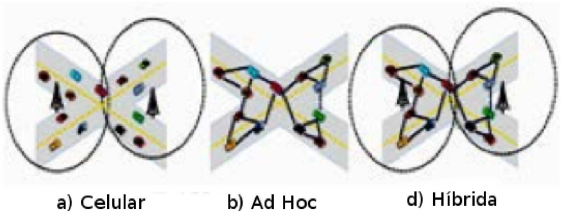

Figura 1 Tipos de VANET
Fonte: (KUMAR; DAVE, 2011).

De acordo com Piran, Murthy e Babu (2011), com a utilização de sensores nos veículos podem ser desenvolvidas aplicações de dois tipos:

Aplicação para Segurança: são utilizadas para reduzir os acidentes envolvendo veículos, por exemplo monitorar velocidade, notificação de acidente, notificação de perigo na estrada e alerta de colisão.

Aplicação para Apoio: são utilizados para o conforto dos condutores dos veículos por exemplo para notificações de roubo, de congestionamento, de multas de trânsito e informações sobre o tráfego de veículos em um determinado trajeto.

O desempenho de uma VANET pode ser afetado pela topologia da rede que é alterada frequentemente (LIDSTRÖM; LARSSON, 2010), prejudicando o desenvolvimento de aplicações em tempo real por possuir baixo nível de confiança.

2.5 Sistemas multi-agentes

Segundo Woolridge (2001), sistemas multi-agente são sistemas compostos por agentes inteligentes com capacidade de controlar o próprio comportamento e interagir com o ambiente. O autor Woolridge (2001), sugere que os agentes podem ser agente de software, robôs ou qualquer outro tipo de recurso que realize tarefas.

De acordo com Woolridge (2001), os agentes em um sistema multiagente

têm algumas características importantes:

Autonomia: os agentes são pelo menos parcialmente autônomos.

Visão local: nenhum agente tem uma visão completa do sistema.

Descentralização: não há nenhum agente designado para controlar.

O agente possui um conjunto de tarefas para serem executadas nos nós de uma rede. Os nós possuem uma certa capacidade para realizar essas tarefas e também possuem uma quantidade limitada de recursos que podem ser utilizados pelos agentes. Para executar essas tarefas os agentes precisam percorrer a rede para poder acessar os recursos dos nós (SHEHORY et al., 1998).

Para Freitas et al. (2011), existem duas estratégias para que o agente se locomova sobre a rede. A primeira é através da clonagem dos agentes, enquanto o segundo é chamado de migração dos agentes. Ambos os mecanismos utilizam o mesmo agente que se desloca pelos nós em uma rede. No entanto, eles se distinguem pelo fato de o agente do primeiro criar uma cópia de si mesmo e este clone é enviado para outro nó, enquanto no segundo, o próprio agente é transmitido para outro nó.

A utilização de um sistema multi-agente sobre VANETs é interessante por que a topologia da rede está em constante mudança, sendo assim é necessário que as escolhas sejam realizadas de acordo com a organização da rede (FREITAS et al., 2011).

Essas características de sistemas multi-agentes citadas anteriormente tornam a utilização das VANETs mais flexível.

2.6 Modelo de movimentos

Para realizar as simulações nesse trabalho foram utilizados modelos de movimentos, que simulam o comportamento dos nós.

Os modelos de movimentos representam o movimento de componentes móveis, a mudança de sua localização, velocidade e aceleração ao longo do tempo. Tais modelos são frequentemente utilizados para fins de simulação, quando

novas técnicas de navegação ou de comunicação são investigadas (NICHOLS; HAMMONS, 2007).

A utilização de modelos de movimentos para realizar simulações em VANET serve para simular os padrões de comportamento dos componentes da rede, por que utilizar componentes reais é inviável (FREITAS et al., 2011).

O autor Sun e Sauvola (2002) sugere duas abordagens para construir um modelo de movimento:

a) **Analítico**: esse modelo de movimento é construído por uma fórmula matemática que define os padrões de comportamento dos elementos representados no modelo.

b) **Simulado**: é construído através da observação do comportamento dos nós no ambiente em que se deseja criar a simulação.

2.7 Simulador GRUBiX

Para realização das simulações deste trabalho foi utilizado o simulador GRUBiX, que é um simulador *open source* desenvolvimento na Universidade Federal de Lavras. Ele pode ser encontrado no endereço *. Este simulador é uma variação do projeto SHOX (LESSMANN; HEIMFARTH; JANACIK, 2008). Assim como o SHOX, o simulador GRUBiX caracteriza-se como um ambiente de simulação orientado a eventos. De forma simplificada, em um ambiente de simulação orientado a eventos, a linha do tempo de uma simulação é representada por uma lista de eventos, em que cada novo evento é inserido nesta lista. A posição em que cada evento é inserido nesta lista varia de acordo com o momento em que cada evento deve ocorrer.

Adicionalmente, estes simuladores oferecem um conjunto de ferramentas para que se possa desenvolver toda a pilha de protocolos presente em um equipamento com rede sem fios. Essa disponibilidade permite que cada camada da pilha de protocolos seja personalizada de acordo com as necessidades de cada aplicação.

*http://sourceforge.net/projects/grubix/

Assim como o SHOX, o simulador GRUBiX utiliza a linguagem de programação Java. O uso de uma linguagem orientada a objetos, nesse caso Java, indica que as camadas da pilha de protocolos sejam modeladas como objetos. O uso da orientação a objetos permite que cada camada seja personalizada através de mecanismos de herança. Portanto, basta que a nova camada personalizada herde as funcionalidade de uma camada base para que se tenha a possibilidade de alterar o seu comportamento padrão.

2.8 Padrão ZigBee

O ZigBee é um padrão de radio frequência que tem como objetivo a padronização e a interoperabilidade de produtos. Ele foi criado e é mantido por um grupo de empresas denominado ZigBee Alliance (ZIGBEEALLIANCE, 2015). O nome ZigBee é uma analogia da forma que as abelhas transitam entre as flores (SAFARIC; MALARIC, 2006). A especificação é a IEEE 802.15.4 na camada física e na *Medium Access Control* (MAC) e para camada de rede e aplicação utiliza o ZigBee. Essa pilha pode ser observada na Figura 2.

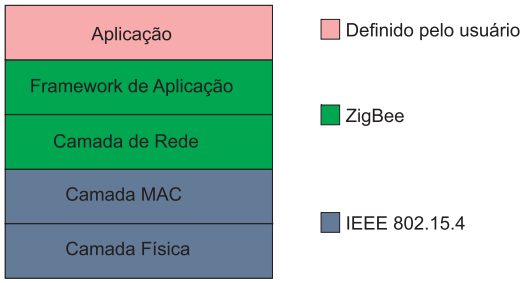

Figura 2 Pilha de protocolo ZigBee

Segundo Ramya, Shanmugaraj e Prabakaran (2011), as principais características do padrão ZigBee são:

a) Auto reparação.

b) Suporte a muitos nós.

c) Facilidade de implementação.

d) Baixo consumo de energia.

e) Padrão aberto.

2.9 Arduíno

O Arduíno é uma plataforma aberta composta por hardware e software. Geralmente são construídos com controladores Atmel AVR de 8 bits. O Arduíno pode ser interligado a outros módulos através de hardwares denominados *shields* (ARDUINO, 2015).

Nesse trabalho é utilizado o modelo Uno, Figura 3, que possui um microcontrolador Atmel ATmega 328p com CPU operando a 16MHz. Oferece quatorze portas de entrada e saída, 32kb de memória de programa, 2Kb de SRAM e duas interrupções externas que são mapeadas pelos pinos 2 e 3 da placa. A comunicação com periféricos externos é realizada de forma serial através de uma porta USB (ARDUINO, 2015).

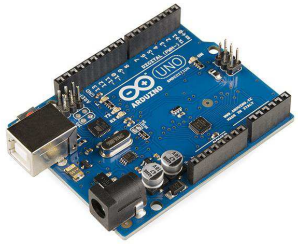

Figura 3 Arduino UNO

Para programar o Arduíno utiliza-se C ou C++. Ele também possui uma IDE com editor de texto e também responsável por carregar o programa na placa do Arduíno.

Para realizar a comunicação esse trabalho utiliza um *shield* para integrar com o módulo Xbee denominado *Xbee shield*. Na Figura 4 é possivel observar o

Xbee shield com o módulo Xbee integrado.

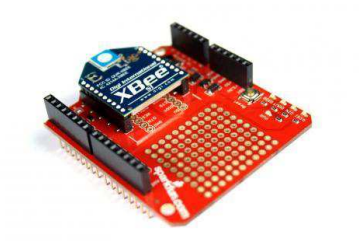

Figura 4 *Xbee shield* com o modulo de Xbee

2.10 Trabalhos relacionados

No trabalho de Santana (2014), utiliza-se o protocolo ZigBee para desenvolver um mecanismo para evitar colisão. O Zigbee é utilizado para enviar aos vizinhos a posição geográfica do veículo obtida através de um dispositivo Android. Quando o dispositivo identifica uma possível colisão ele emite um alerta. Para identificar a colisão o dispositivo utiliza a longitude e latitude do veículo, a precisão e velocidade. Essas informações são transmitidas pelos vizinhos utilizando o ZigBee.

Os testes realizados por Santana (2014) obtiveram uma latência máxima de 65 ms. Esse resultado mostra que o padrão ZigBee atende aos requisitos de latência para rede veiculares.

O trabalho de Freitas et al. (2011) analisa a utilização de agentes sensitivos a posições geográficas com diferentes níveis de inteligência. Nele são simulados três níveis de inteligência dos agentes. Os resultados das simulações fornecem informações sobre a importância do uso do contexto para que os agentes possam cumprir a sua missão.

3 METODOLOGIA DA SIMULAÇÃO

O desenvolvimento da simulação foi dividido em três etapas. A primeira etapa é a construção do cenário com o modelo de movimento, veículos e infraestrutura. A segunda etapa é o desenvolvimento dos agentes de software, e por último a fase de simulação.

A simulação foi desenvolvida utilizando o simulador GRUBiX e executada em um notebook com configuração detalhada na Tabela 1.

Tabela 1 Configuração do notebook utilizado nas simulações

Sistema Operacional	Ubuntu Linux 10.10
Processador	Intel Core i7-2630QM 2.00GHz (6M Cache)
Memória RAM	8GB SSD
Disco Rígido	120 GB SSD

3.1 Desenvolvendo o cenário

O cenário é uma cidade dividida em quadras seguindo o modelo de movimento *Manhattan*. Nessa cidade possuem veículos que percorrem as quadras. Alguns cruzamentos possuem uma infraestrutura e elas estão conectadas através de cabos.

A Figura 5 apresenta um esboço do cenário onde se pode observar as ruas, os veículos e a infraestrutura. Esse cenário é utilizado em todas as simulações, entretanto a infraestrutura é desativada quando não é utilizada.

O comprimento e a largura da região de simulação são levados em consideração na construção do cenário. As linhas horizontais e verticais são as ruas e os cruzamentos delas são as quadras. Para obter a quantidade de ruas horizontais e verticais são utilizadas as Equações 3.1 e 3.2.

$$Q_{rh} = \frac{A_{at}}{A_{aq}} \tag{3.1}$$

Onde:

Q_{rh} = quantidade de ruas na horizontal

A_{at} = altura total do cenário

A_{aq} = altura das quadras

$$Q_{rv} = \frac{L_{lt}}{L_{lq}} \tag{3.2}$$

Onde:

Q_{rh} = quantidade de ruas na vertical

L_{lt} = largura total do cenário

L_{lq} = largura das quadras

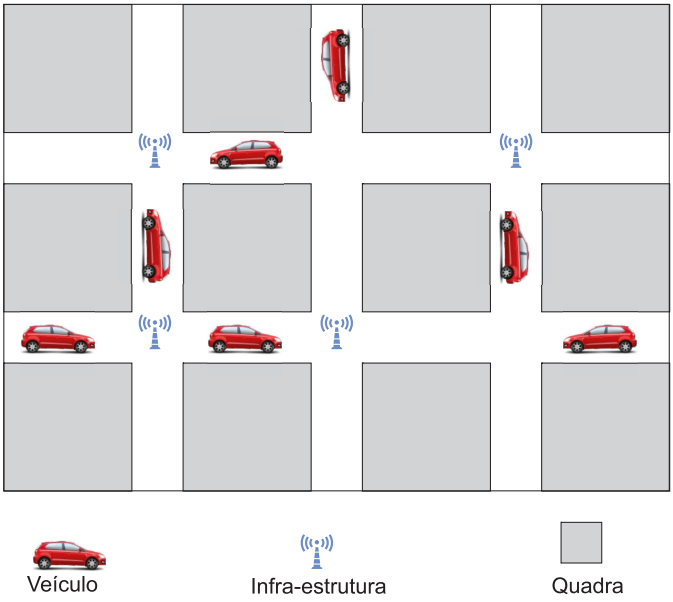

Figura 5 Modelo de cidade utilizado

3.1.1 Veículos e infraestrutura

O agente necessita que os nós (veículos e a infraestrutura) tenham dois recursos fundamentais para o seu funcionamento: a capacidade de comunicação; a execução de suas instruções pelo agente. Na simulação todos os nós possuem essa competência.

Quando um agente chega em um nó, ele logo executa as suas instruções. No simulador o nó executa um evento para verificar se existe algum agente hospedado nele. Se existir o evento retorna um objeto que deve implementar a interface agente que se encontra no Anexo A. Com o objeto o nó executa o método execute(). As classes do programa em Java que simulam o comportamento do nó se encontram no Anexo B. O Anexo C descreve a classe java que implementa o modelo de movimento.

Todos os veículos e infraestruturas possuem o mesmo comportamento em relação ao agente. Por outro lado, eles possuem comportamentos diferentes em relação ao ambiente. Primeiramente os veículos possuem a capacidade de locomoção. Embora a infraestrutura não possua essa capacidade, todas elas estão conectadas. Em outras palavras, quando um agente está em uma infraestrutura ele não consegue se locomover, porém consegue se comunicar com todas as outras infraestruturas espalhadas pelo cenário.

Ao iniciar a simulação os veículos e as infraestruturas são distribuídos entre os cruzamentos de maneira aleatória.

O Algoritmo 1 de locomoção leva em consideração a equação da reta que representa a rua. Isto significa que quando o veículo está em uma rua do tipo horizontal ele se movimenta em X e quando estiver em uma rua do tipo vertical o veículo se movimenta em Y. Para simular a velocidade quando um veículo entra em uma rua, o tipo dela é verificado e então soma ou subtrai um valor da posição atual dele.

Quando a velocidade do veículo é maior que a distância entre a posição atual dele e de um cruzamento, a sua velocidade diminui até chegar em zero. Isto significa que o veículo chegou em um cruzamento, então o veículo deve decidir entre continuar, virar para esquerda, virar para direita ou voltar. Cada direção

possui uma probabilidade como demonstra a Tabela 2.

Algoritmo 1: Algoritmo movimento dos nós.

```
1  início
2      nos ← pegarListaNos();
3      i ← 0;
4      repita
5          if nos[i].eVeiculo() then
6              pontoProximoCruzamento = nos[i].pegarCruzamentoMaisProximo();
7              distancia = pontoProximoCruzamento.calcularDistancia(nos[i].posicao);
8              if distancia < nos[i].velocidade then
9                  nos[i].velocidade = distancia;
10             else
11                 if distancia == 0 then
12                     nos[i].mudarDirecao();
13                 else
14                     nos[i].andar();
15                 end
16             end
17         if nos[i].possuiAgente() then
18             nos[i].pegarAgente().execute();
19         i++;
20     até i == nos.length;
21 fim
```

No caso da infraestrutura a velocidade é sempre zero. Assim elas utilizam o mesmo algoritmo dos veículos porém não mudam de posição.

Tabela 2 Probabilidade do veículo escolher uma direção

Direção	Probabilidade
Frente	50%
Direita	20%
Esquerda	20%
Voltar	10%

O Algoritmo 2 demonstra que em cada movimento do veículo são calculadas as distâncias entre a sua posição atual e de todos os cruzamentos contrários ao tipo de rua que o veículo está locomovendo. Por exemplo, quando um veículo se locomove em uma rua horizontal, a cada movimento o algoritmo

calcula as distâncias entre a posição do veículo e as ruas verticais.

Para definir as ruas horizontais é utilizada a Equação 3.3 que é a equação da reta quando a reta cruza o eixo *Y*.

Algoritmo 2: Algoritmo que identifica a aproximação com um cruzamento.

```
1  início
       Entrada: O nó a ser testado
2      i ← 0;
3      if no.tipoRua == horizontal then
4          y ← no.posicao.getY();
5          repita
6              y_rua ← A_aq(1 + i) ;
7              if y_rua − y < no.velocidade then
8                  retorna y_rua − y ;
9              i++;
10         até i < Q_rh;
11     else
12         if no.tipoRua == vertical then
13             x ← no.posicao.getX();
14             repita
15                 x_rua ← L_lq(1 + i) ;
16                 if x_rua − x < no.velocidade then
17                     retorna x_ruax − x ;
18                 i++;
19             até i < Q_rv;
20
21     end
22 fim
```

A Equação 3.4 é utilizada para definir as ruas verticais. Todas as ruas horizontais se cruzam com as ruas verticais, então para descobrir um cruzamento é necessário indicar o número da rua nas Equações 3.3 e 3.4 respectivamente. Com isso é obtido um ponto que corresponde ao cruzamento.

$$Y = A_{aq}(1 + N_{horizontral}) \tag{3.3}$$

Onde:

Y = posição da rua

A_{aq} = altura das quadras

$N_{horizontal}$ =número de quadras onde $N_{horizontal} \in I, 0 \leq N_{horizontal} < Q_{rv}$

$$X = L_{lq}(1 + N_{vertical}) \tag{3.4}$$

Onde:
X = posição da rua
L_{lq} = largura das quadras
$N_{vertical}$ =número de quadras onde $N_{vertical} \in I, 0 \leq N_{vertical} < Q_{rv}$

3.1.2 Agente de software

Foi utilizado Freitas et al. (2011) como referência para a construção do agente de software. A missão do agente é permanecer dentro de uma região (região alvo) no cenário. A região é definida por dois pontos, sendo um limite superior e outro o limite inferior, como demonstra a Figura 6.

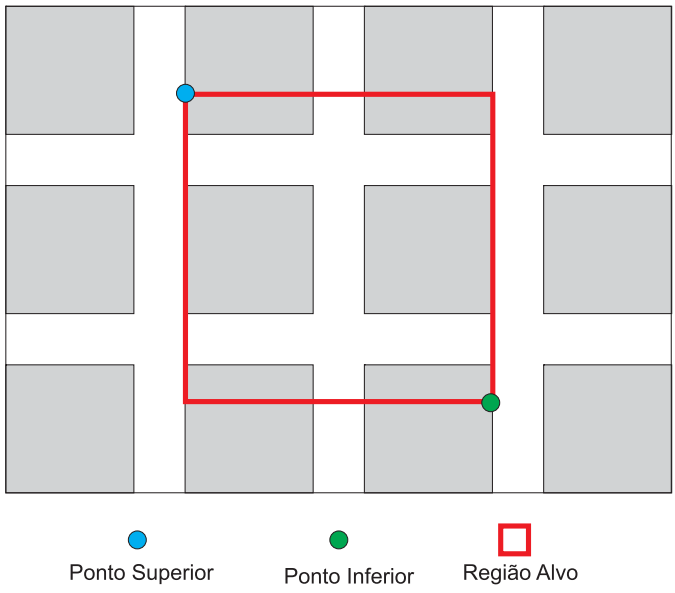

Figura 6 Região Alvo

Para a simulação foram criados dois tipos de agentes. O agente que deve

se manter dentro da região alvo é o agente principal. O agente principal pode criar um tipo de agente mais simples, o *mini-agente*, que é responsável por buscar nós que possam propiciar ao agente principal maior possibilidade de finalizar a sua missão.

Quando a simulação inicia é escolhido um nó aleatoriamente para receber um agente principal. Então o nó executa as instruções do agente. Para o funcionamento dessa arquitetura o agente precisa possuir uma interface onde o veículo consiga acessar as instruções do agente. Os nós também devem ter uma interface onde o agente consiga acessar recursos necessários para o seu funcionamento. Nesse trabalho, os nós possuem três recursos disponíveis:

a) Posição atual

b) Posição com o destino do nó

c) Comunicação com os nós vizinhos

Na Figura 7 pode-se visualizar a arquitetura das simulações. Essa arquitetura permite adicionar mais atores na simulação, por exemplo, agentes como missões diferentes ou até mesmo uma pessoa com algum equipamento que permita o transporte dos agentes. Assim, possibilita ao agente acessar regiões antes impossíveis para acesso de veículos.

A missão do agente é alcançar a região alvo e permanecer nela. Para isto, o agente deve conhecê-la. Então a primeira ação do agente é solicitar ao nó onde ele está hospedado a sua posição. Se o veículo estiver na região alvo o algoritmo termina a sua execução, e nesse momento poderia realizar outras atividades como recolher dados ou disseminar propaganda.

Caso a posição do nó não esteja no interior da região alvo, o agente precisa procurar um hóspede mais apropriado. Para isso é criado um segundo agente, mais simples que o primeiro, com a missão de procurar entre os nós vizinhos um mais apropriado. O Algoritmo 3 descreve o funcionamento do agente principal. Cada nó vizinho recebe uma cópia do *mini-agente* e o executa. A estrutura do *mini-agente* contém a região alvo e a distância do destino com a região alvo. Esses dados são comparados pelo *mini-agente* com os dados dos nós que o receberam. Caso algum

nó tenha uma condição melhor que o nó onde o agente principal está hospedado, o *mini-agente* envia um sinal avisando o agente principal para ele migrar. Após finalizar a missão o *mini-agente* é descartado. O funcionamento do *mini-agente* é reproduzido pelo algoritmo 4.

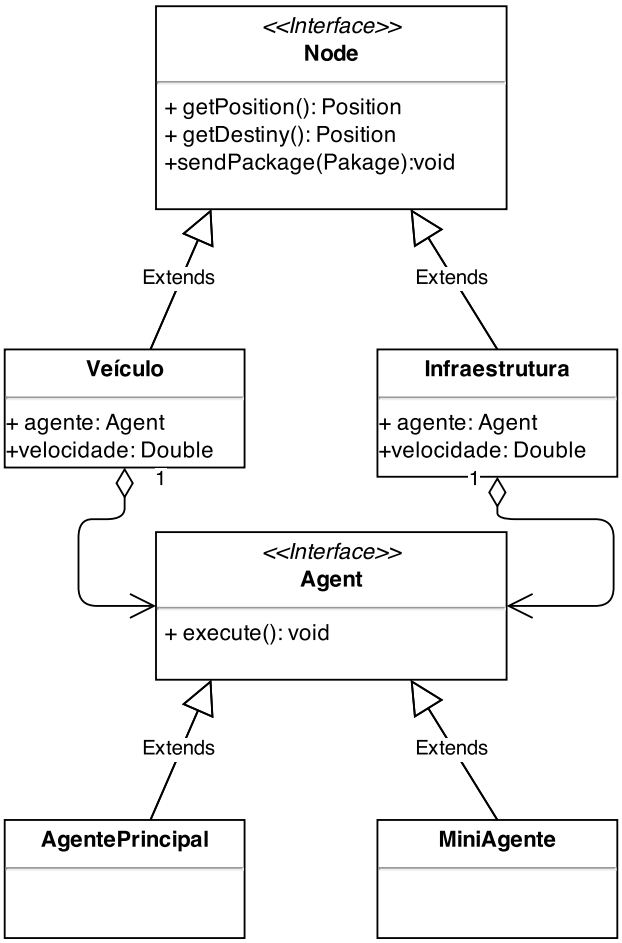

Figura 7 Diagrama UML com a relação entre as entidades

Algoritmo 3: Algoritmo do Agente Principal.

```
1  início
2      repita
3          regiaoAlvo ← pegarRegiaoAlvo();
4          posicaoNo ← no.pegarPosicao();
5          destinoNo ← no.pegarDestino();
6          if posicaoNo ∈ regiaoAlvo then
7              executarTarefasDaRegiaoAlvo();
8          else
9              miniAgente ← criarMiniAgente(posicaoNo, destinoNo, regiaoAlvo);
10             mensagem ← no.enviarMensagem();
11             reposta = escutarReposta(mensagem);
12             if reposta = vazio then
13                 executarTarefasForaRegiaoAlvo();
14             else
15                 no.enviarMensagem(agentePrincipal, reposta.idImovelSelecionado());
16             end
17         end
18         hibernarPeriodo();
19     até no.agentePrincipal ≠ vazio;
20 fim
```

Algoritmo 4: Algoritmo do Mini-agente.

```
1  início
       Entrada: posicaoNoHospedeiro, destinoNoHospedeiro, regiaoAlvo
2      distanciaRegiaoAlvoDestinoNoHospedeiro ←
       regiaoAlvo.calcularDistancia(destinoHospedeiro);
3      distanciaRegiaoAlvoDestinoNo ← regiaoAlvo.calcularDistancia(no.destino);
4      if no.posicao ∈ regiaoAlvo then
5          no.enviarSinal(no.id);
6      else
7          if no.destino ∈ regiaoAlvo then
8              no.enviarSinal(no.id);
9          else
10             if distanciaRegiaoAlvoDestinoNoHospedeiro < distanciaRegiaoAlvoDestinoNo then
11                 no.enviarSinal(no.id);
12
13         end
14     end
15     no.removerMiniAgente();
16     s
17 fim
```

Durante a seleção três perguntas devem ser respondidas e, se uma das respostas for afirmativa, o *mini-agente* envia o sinal para o agente principal migrar:

1. Nó está dentro da região alvo? (Figura 8)
2. Destino do nó é dentro da região alvo? (Figura 9)
3. Destino do nó é mais próximo da região alvo? (Figura 10)

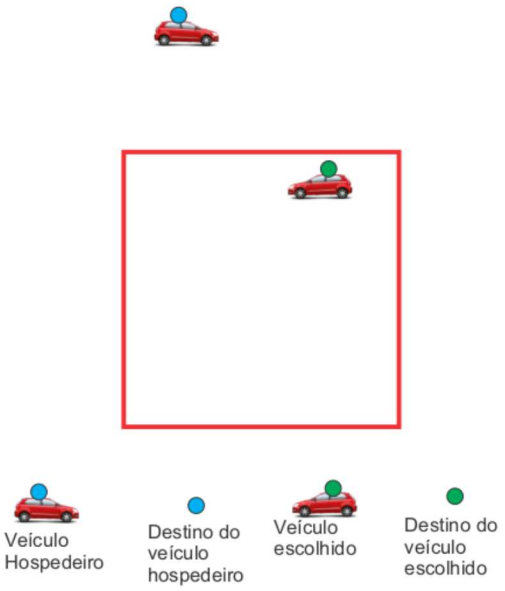

Figura 8 Veículo dentro da Região Alvo

O primeiro sinal que alcançar o agente principal realiza a migração para o nó que enviou o sinal. Essa abordagem é utilizada por dois motivos:

a) **Primeiro:** O sinal contendo somente o identificador do nó possui poucos bytes.

b) **Segundo:** Elimina a necessidade do desenvolvimento de um mecanismo para esperar que todos os *mini-agente* executem a sua missão e só depois tomar a decisão de escolher o melhor nó.

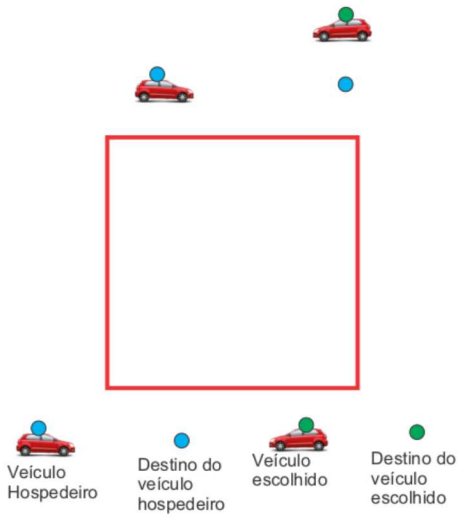

Figura 9 Destino veículo dentro da Região Alvo

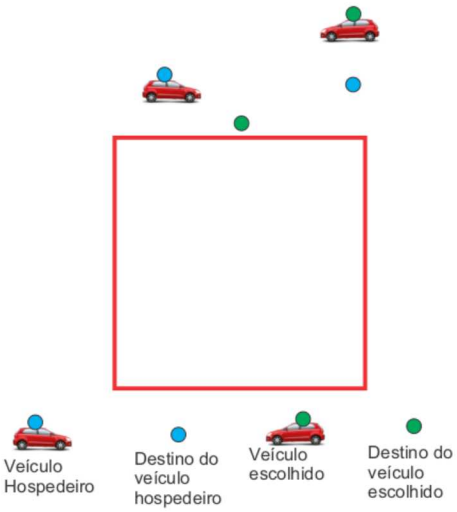

Figura 10 Destino do veículo selecionado dentro da Região Alvo

Na simulação que a infraestrutura está ativada, ela pode replicar o *mini-agente* para todos os outros nós próximos às infraestruturas espalhadas pelo cenário, aumentando a abrangência da busca. Se um nó distante for escolhido, o agente principal pode utilizar a infraestrutura para alcançá-lo. A Figura 11 demonstra uma situação onde o *mini-agente* encontra uma infraestrutura.

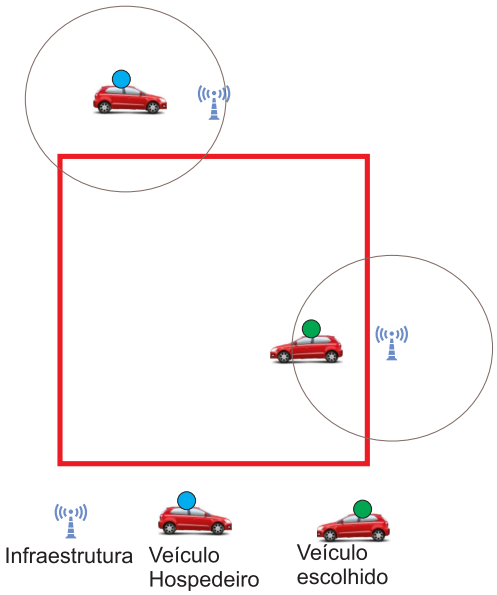

Figura 11 Exemplo com infraestrutura

Para não congestionar a rede, o agente principal deve evitar procurar novos nós por um período de tempo, e durante esse período ele pode executar outras tarefas. Esse comportamento pode ser repetido infinitamente ou o agente pode ter um tempo determinado para executar a sua missão. Ao fim desse tempo, o agente deixa de existir.

O objetivo da simulação é averiguar se a adição de uma infraestrutura melhora a locomoção dos agentes de software, favorecendo que ele desempenhe a sua missão. Após o desenvolvimento e testes do cenário e de todos os atores da simulação é necessário configurar o simulador. As varáveis que o simulador

GRUbiX utiliza são: quantidade dos nós, alcance do sinal, largura e altura do cenário.

As variáveis largura e altura do cenário são constantes em 120 metros. Por outro lado, a quantidade de nós e alcance de cada nó são variáveis. Essa variação permite simular o comportamento do agente em ambientes com alta e baixa densidade de veículos. Com a altura e largura do cenário é possível obter as quadras. Elas possuem 10 metros de altura e 10 de largura, totalizando 144 quadras.

Após configurar as variáveis de ambiente do GRUBiX é necessário configurar os atores da simulação. Os atores da simulação são: veículos, infraestruturas, agente principal e o agente secundário. As variáveis dos nós que representam os veículos são o alcance e a velocidade. Os nós que representam as infraestruturas possuem o mesmo alcance dos veículos, porém a velocidade é nula e também podem ser ativados e desativados. O agente principal só precisa conhecer a região alvo, onde o ponto superior é S(80,100) e o inferior é I(100,80) sendo a escala em metros.

Então foram modeladas vinte e quatro situações para serem simuladas. Elas consistem em doze com a infraestrutura ligada e outras doze com a infraestrutura desligada. A Tabela 3 apresenta um resumo das configurações simuladas.

Tabela 3 Resumo das configurações da simulação

Infraestrutura	Ativada				Desativada			
Quantidade de nós	25	50	75	100	25	50	75	100
Alcance dos nós (metros)	5	5	5	5	5	5	5	5
Alcance dos nós (metros)	10	10	10	10	10	10	10	10
Alcance dos nós (metros)	15	15	15	15	15	15	15	15

O simulador permite uma visualização da simulação como apresenta a Figura 12. Nela as linhas representam as ruas, os pontos amarelos são os veículos e os vermelhos são a infraestrutura. Nas simulações, quando a infraestrutura está ativada, cinco nós que seriam veículos se tornam infraestrutura e a sua velocidade se mantém nula. Quando a infraestrutura está desligada os nós continuam sendo

veículos. Exemplificando: com a infraestrutura ligada, quando existir vinte e cinco nós, cinco nós são infraestrutura e o restante veículos. Por outro lado, quando a infraestrutura estiver desligada, todos os nós são veículos.

Foram realizadas mil simulações para cada uma das vinte e quatro situações totalizando vinte e quatro mil simulações. Os dados recolhidos foram o tempo de simulação e o tempo que o agente permaneceu dentro da região alvo.

Com esses dados é possível obter a porcentagem de tempo que o agente principal permaneceu na região alvo e o impacto que cada variável tem no desempenho do agente em cumprir a missão. Com os resultados finais foram calculados desvio padrão, variância e assim analisados.

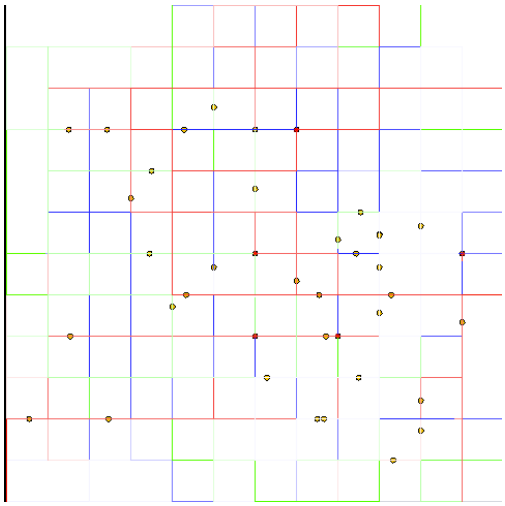

Figura 12 Visualização no simulador

4 METODOLOGIA DO PROTÓTIPO

Um protótipo foi desenvolvido para avaliar o uso dos agentes fora do ambiente virtual. O objetivo desta fase é recolher dados sobre a latência e taxa de perda do agente. A latência é importante para alguns tipos de aplicações como por exemplo para detectar colisão (SANTANA, 2014). Assim, essas informações são importantes para identificar em quais tipos de aplicações os agentes são melhores empregados. Uma rede para ser bem sucedida com a utilização dos agentes de software, necessita mantê-los ativos até o fim de sua missão. Na Figura 13 pode-se visualizar um nó instalado em um veículo.

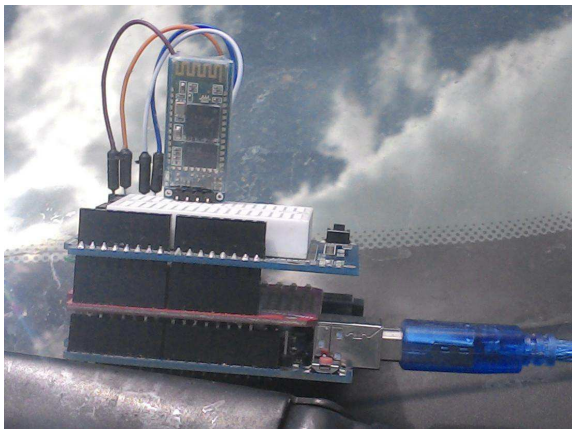

Figura 13 Nó instalado em um veículo

O experimento utiliza três nós, onde dois nós são para os veículos e um desempenha o papel de infraestrutura. A estrutura dos nós foi reaproveitada do trabalho de (SANTANA, 2014). Os equipamentos necessários para a construção dos nós podem ser visualizados na Tabela 4.

Ao iniciar o experimento é introduzido um agente em um dos nós. Então o agente deve migrar sempre que encontrar um outro nó.

Tabela 4 Equipamentos utilizados no experimento

Equipamento	Quantidade
Notebook	1
Módulo bluetooth modelos JY-MCU	2
Arduíno UNO	2
Xbee shield	2
Mini protoboard 170 pontos	2
Xbee S1	3
XBee Explorer USB Adapter	1
Motorola RAZR I	1
Motorola Moto G	1

4.1 Arquitetura do nó

A Figura 14 demonstra a arquitetura da infraestrutura que utiliza um Xbee e um XBee Explorer USB Adapter conectado em um notebook.

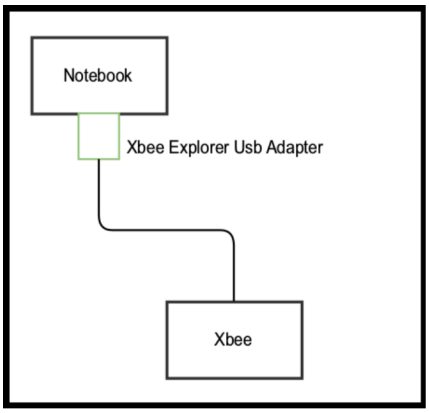

Figura 14 Arquitetura da infraestrutura

O agente fica hospedado no smartphone, o arduíno serve somente como interface entre o smartphone e o módulo Xbee. A comunicação entre o arduíno e o smartphone acontece através do bluetooth.

O smartphone acolhe o agente por que possui um ambiente mais rico em

recursos que o arduíno, isto significa que o smartphone possui GPS, mais memória e outros recursos para que o agente possa desempenhar a sua missão. A Figura 15 ilustra a estrutura do nó que atua nos veículos.

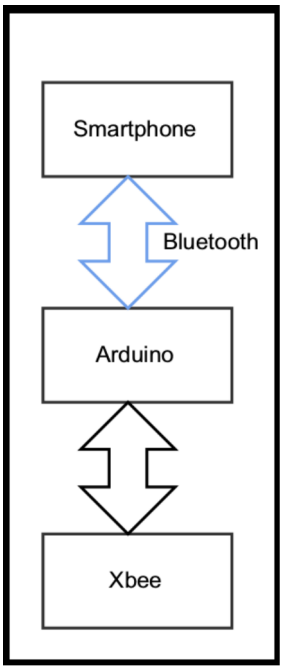

Figura 15 Estrutura do nó no veículo

Para sincronizar o relógio dos smartphones utilizados no experimento foi utilizado o aplicativo NTPSync (NTPSYNC, 2015), que é um aplicativo simples para sincronizar o tempo. Manter o relógio sincronizado é importante para medir a latência com precisão.

Para hospedar o agente foi desenvolvido um aplicativo Android, esse aplicativo fornece ao agente os mecanismos necessários para realizar a sua missão. Através dele o agente consegue acessar o GPS, o bluetooth e a tela do smartphone. No display do aplicativo tem um alerta que informa se existe algum agente hospedado no smartphone e um botão para criar novos agentes. A Figura 16 demonstra a tela do aplicativo responsável pelo agente.

Figura 16 Tela do aplicativo

Esse aplicativo também realiza a coleta dos dados em arquivo. Os dados coletados são a hora que o agente migrou, a hora que recebeu um agente em milisegundo. Então para calcular a latência faz-se a subtração da hora de quando o novo hóspede recebeu o agente e a hora que o agente iniciou a migração no antigo hóspede.

4.2 Agente

A missão do agente neste experimento é migrar sempre que encontrar um novo nó. O agente deve transportar as instruções, o identificador da migração e a posição do antigo nó hóspede para o novo nó. As instruções utilizadas pelo agente são apresentadas na Tabela 5, elas são formadas por uma letra maiúscula, uma sequência de três letras minúsculas e um número, esse padrão é utilizado pelo mecanismo de validação e recuperação. Para delimitar o fim da instrução é utilizado o caractere de ponto e vírgula (;). O identificador da migração é um número que serve para auxiliar no cálculo de latência. Assim com o identificador é possível encontrar a hora que o agente migrou e a hora que o agente foi recebido

por completo no hospedeiro em dois arquivos diferentes. Sempre antes de o agente migrar, o identificador de migração é incrementado.

Tabela 5 Lista de instruções

Instrução	Ação
Spap1	solicitar e armazenar no agente a posição atual para o nó hospedeiro
Remi2	realizar migração.
Bnov3	buscar nós vizinhos.

O agente é formado por dois elementos, como demonstra a Figura 17. O primeiro elemento denominado *cabeçalho* é a região onde estão as instruções. O *corpo* é o elemento onde os dados do agente são armazenados. O corpo é envolvido pelos caracteres de colchetes (*[ ]*), essa estrutura é utilizada para separar o cabeçalho do corpo.

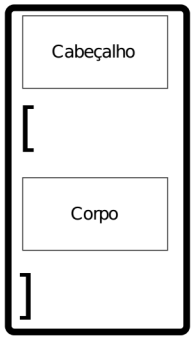

Figura 17 Nó instalado em um veículo

4.3 Mecanismo validação e recuperação

Após um nó receber o último colchete (*]*) ou 500 ms com um agente incompleto o mecanismo de validação e recuperação é ativado. A principal função deste mecanismo é tentar recuperar o agente danificado e se não conseguir descartá-lo.

A validação verifica se todas as instruções têm cinco caracteres, se a

primeira letra é maiúscula e o último caractere é um número, além de verificar as instruções ele valida a estrutura do corpo. Quando alguma instrução não atende uma das três características mencionadas anteriormente ou o corpo não atende o padrão, o mecanismo de recuperação entra em ação.

Se o erro for no *cabeçalho*, o algoritmo verifica o tamanho da instrução se for maior que cinco ele descarta o agente. Se for menor ele confere a primeira letra, se ela atender algumas das instruções suportadas ele recupera colocando a instrução correta. Se não conseguir através do primeiro caractere a próxima tentativa é o último caractere que deve ser um número. Se nenhuma das tentativas forem alcançadas o agente é descartado. A Figura 18 demonstra a estrutura da instrução utilizada neste trabalho.

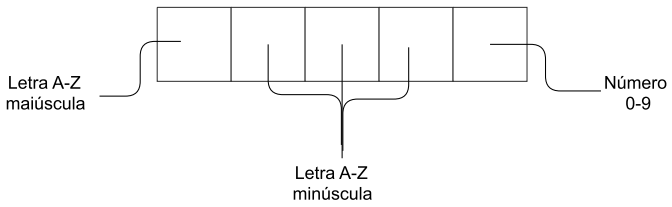

Figura 18 Estrutura da Instrução

Para tentar recuperar o corpo, primeiro verifica se existe um caractere de colchete ([) iniciando o corpo, se não existir o agente é descartado. Se existir é verificado se o último caractere é um colchete finalizando a estrutura, se não existir o mecanismo coloca o caractere que representa fim. Esse roteiro é reproduzido pelo Algoritmo 5 e a Figura 19 apresenta o agente utilizado neste trabalho.

```
Spap1;
if Remi2 {
  Bnov3;
};
[
  25;
  P(x-21.2252569,y-44.9743734,20);
]
```

Figura 19 Agente Software

Algoritmo 5: Mecanismo validação e recuperação.

```
1  início
       Entrada: Agente
2      instrucoes = agente.pegarInstrucoes();
3      i ← 0;
4      if agente.contem([) then
5          repita
6              instrucao ← instrucao[i] ;
7              if instrucao.tamanho < 5 then
8                  if verificar(instrucao.primeiroCaracter) then
9                      instrucao ← recuperar(instrucao.primeiroCaracter);
10                 else
11                     if verficar(instrucao.ultimoCaracter) then
12                         instrucao ← recuperar(instrucao.ultimoCaracter);
13                     else
14                         agente.remover();
15                     end
16                 end
17             else
18                 agente.remover();
19             end
20             i++;
21         até i < instrucoes.quantidade;
22         if agente.ultimoCaracter ≠] then
23             agente.ultimoCaracter ←]
24
25     else
26         agente.remover();
27     end
28 fim
```

4.4 Trajeto e cenário

Como trajeto é utilizada a avenida norte que está localizada no campus da Universidade Federal de Lavras. O trajeto pode ser visualizado na Figura 20. Nesse trajeto o veículo percorre 344,45 metros.

Os veículos trafegam sobre o roteiro em velocidade média de 20 km/h, na Figura 21 pode-se visualizar os veículos utilizados no experimento. Existem dois cenários para realizar o experimento, sendo o primeiro caso com infraestrutura e outro somente com veículos. A infraestrutura está localizada ao centro do trajeto

como demonstra a Figura 22.

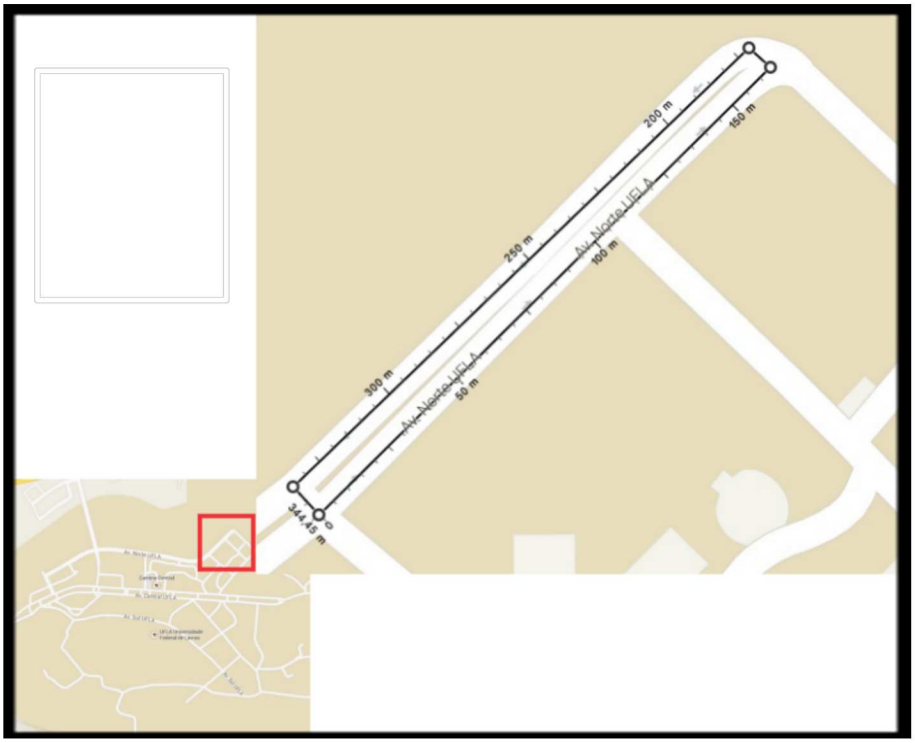

Figura 20 Modelo de cidade utilizado

Figura 21 Veículos utilizados no experimento

A infraestrutura é colocada aproximadamente ao centro do cenário. Quando a infraestrutura não é utilizada ela é desligada. A infraestrutura tem a função de receber e armazenar o agente até um novo nó se aproximar. Para saber se o agente passou pela infraestrutura é adicionado um caractere *I* ao corpo do agente. Esse caractere vai auxiliar no cálculo da quantidade de vezes que o agente passou pela infraestrutura.

Figura 22 Infraestrutura utilizada no experimento

No experimento realizado, cada veículo percorreu cinquenta vezes o trajeto, vinte e cinco com a infraestrutura ativada e vinte e cinco com a infraestrutura desativada. Foram recolhidas informações sobre a latência e taxa de perda do agente. Ao fim dessa fase foram calculados o desvio padrão, variância e os resultados foram analisados.

5 RESULTADOS E DISCUSSÕES

Os resultados apresentados nesta seção estão divididos em duas partes. A primeira parte abrange as simulações realizadas e os resultados obtidos como é descrito nas Seções 3 e 4. A segunda parte abrange o experimento prático realizado na UFLA com estudo da taxa de perda do agente e latência como descrito na Seção 4.

5.1 Resultados da simulação

Diversas simulações foram realizadas no simulador GRUBiX, como descrito na Seção 3. As simulações foram realizadas para avaliar o comportamento do agente em redes com diferentes níveis de densidade de nós e distância de comunicação dos nós. Também foram simuladas redes somente com veículos sem uma infraestrutura de apoio e outra híbrida.

Após realizar as simulações, os dados foram organizados e analisados em três fases:

1. Resultados obtidos na rede sem infraestrutura.
2. Resultados obtidos na rede híbrida.
3. Comparação dos resultados obtidos nas duas situações.

5.1.1 Rede sem infraestrutura

Os resultados obtidos da rede sem infraestrutura mostram que o raio de alcance e a quantidade dos nós afetam o desempenho do agente. A Figura 23 demonstra a evolução do tempo que o agente permaneceu na região alvo (desempenho do agente) com o aumento do raio e a quantidade de nós.

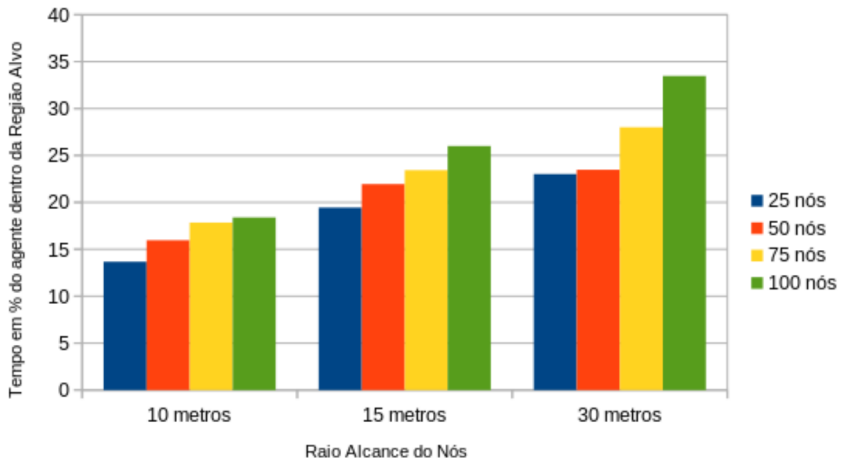

Figura 23 Resultados obtidos nos cenários sem infraestrutura

Na Tabela 6 é possível visualizar a média aritmética, o desvio padrão e a variância dos resultados obtidos.

Tabela 6 Estátistica dos resultados obtidos relativos a situação sem infraestrutura

25 nós			50 nós		
Alcance	Média Aritmética	Desvio Padrão	Alcance	Média Aritmética	Desvio Padrão
10 metros	13,67	2,25	10 metros	15,96	1,41
15 metros	19,45	1,71	15 metros	21,95	1,43
30 metros	23,03	0,81	30 metros	23,48	1,12

75 nós			100 nós		
Alcance	Média Aritmética	Desvio Padrão	Alcance	Média Aritmética	Desvio Padrão
10 metros	17,84	2,63	10 metros	18,40	2,86
15 metros	23,44	2,33	15 metros	26,01	1,40
30 metros	28,01	1,41	30 metros	33,48	1,10

Conforme o gráfico, a cada vinte e cinco nós que são adicionados à rede, a eficiência do agente aumenta. Quando o aumento ocorre no raio do alcance do nó, a melhora do desempenho também é perceptível.

O aumento da eficiência do nó quando comparando o aumento do alcance de 10 metros para 15 metros foi maior que o aumento do alcance de 15 metros para 30 metros. Isso ocorre porque com o aumento do raio somente uma pequena parte do aumento da área coberta fica sobre a rua onde estão os veículos. A Figura 24 demonstra o problema. A região cinza representa as quadras e a região vermelha é a representação do sinal desperdiçado. Quanto maior o raio de alcance do nó, maior o desperdício. Essa é uma desvantagem em não usar rede híbrida porque dentro das quadras poderiam haver outros dispositivos que auxiliariam o agente. Assim o agente poderia encontrar rotas através das quadras, aumentando a quantidade de rotas disponíveis para ele usar.

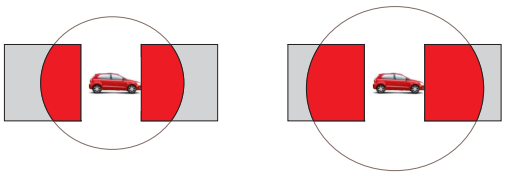

Figura 24 Exemplificação do disperdicio

A princípio, não é necessário se preocupar com a economia de energia do transmissor/receptor instalado em cada veículo. Assim, o alcance da comunicação dos nós pode ser alta, melhorando o desempenho do agente.

Quanto maior é a quantidade de veículos (nós) que formam a rede, maior é a probabilidade de o agente alcançar novos nós hospedeiros, e assim maiores são as chances de ele conseguir se manter na região alvo. A melhora de desempenho do agente comparando entre a situação onde o agente tem à disposição o menor alcance e a menor densidade de veículos (25 nós e 10 metros) para a situação mais farta de recursos (100 nós e 30 metros) é de 145% aproximadamente.

5.1.2 Rede com infraestrutura

Os resultados obtidos em uma rede veicular com infraestrutura demonstrou que a densidade e raio de alcance têm os mesmos impactos discutidos na Seção 5.1.1.

A Figura 25 demonstra a melhoria de desempenho do agente ao aumentar a densidade dos nós e o raio de alcance. A melhoria de desempenho do agente na configuração com a menor quantidade de recursos para a rede com a configuração com maior quantidade de recursos foi de 114,43%.

Para cada vinte e cinco nós que entram na rede, o agente melhora o seu desempenho (tempo médio dentro da região alvo) significativamente. O aumento do raio de alcance de cada nó também aumenta o desempenho do agente. O aumento combinado do raio e a densidade causa um impacto diretamente na quantidade de veículos que o agente tem acesso, aumentando o seu desempenho.

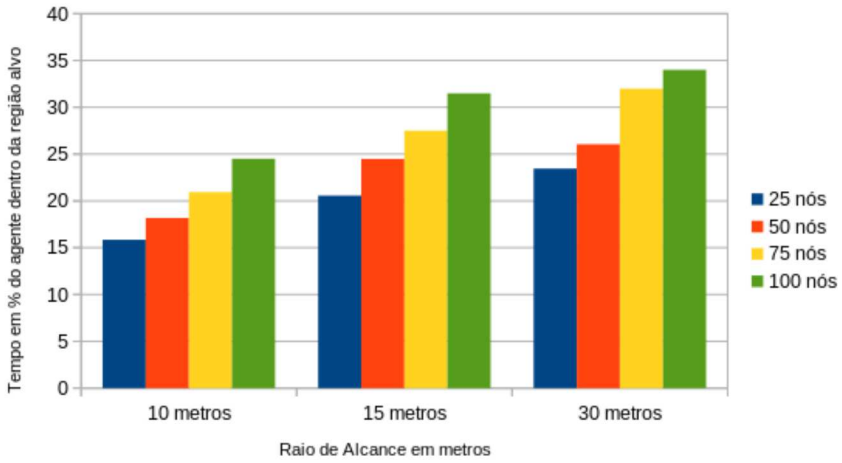

Figura 25 Resultados obtidos no cenários com infraestrutura

Na Tabela 7 é possível visualizar a média aritmética, o desvio padrão e a variância dos resultados obtidos.

Tabela 7 Estatística dos resultados obtidos

25 nós			50 nós		
Alcance	Média Aritmética	Desvio Padrão	Alcance	Média Aritmética	Desvio Padrão
10 metros	15,85	1,40	10 metros	18,18	1,43
15 metros	20,57	1,68	15 metros	24,48	1,12
30 metros	23,45	1,11	30 metros	26,05	1,41

75 nós			100 nós		
Alcance	Média Aritmética	Desvio Padrão	Alcance	Média Aritmética	Desvio Padrão
10 metros	20,94	2,05	10 metros	24,51	1,10
15 metros	27,50	1,69	15 metros	31,49	1,13
30 metros	31,99	1,38	30 metros	34,00	1,42

5.2 Resultados com protótipo

Como descrito na Seção 4 foram realizados dois testes com o protótipo, o primeiro utilizando uma infraestrutura fixa e outro apenas com os veículos.

A Figura 26 apresenta a comparação entre as médias das latências com infraestrutura e sem infraestrutura. Pode-se observar que a infraestrutura causou pouco impacto na latência. Isso ocorre pelo fato de que ambos os experimentos utilizaram os mesmos algoritmos de locomoção do agente e equipamentos. Levando em consideração que o ambiente era controlado e não existiam outras fontes de ruído, o ZigBee apresentou uma baixa variação da latência. A Tabela 8 apresenta informações como média e variância.

Tabela 8 Valores medidos da latência

	Média Aritmética	Desvio Padrão	Variância
Sem infraestrutura	126 (ms)	2,3123	5,6288
Com infraestrutura	124 (ms)	1,7654	2,8354

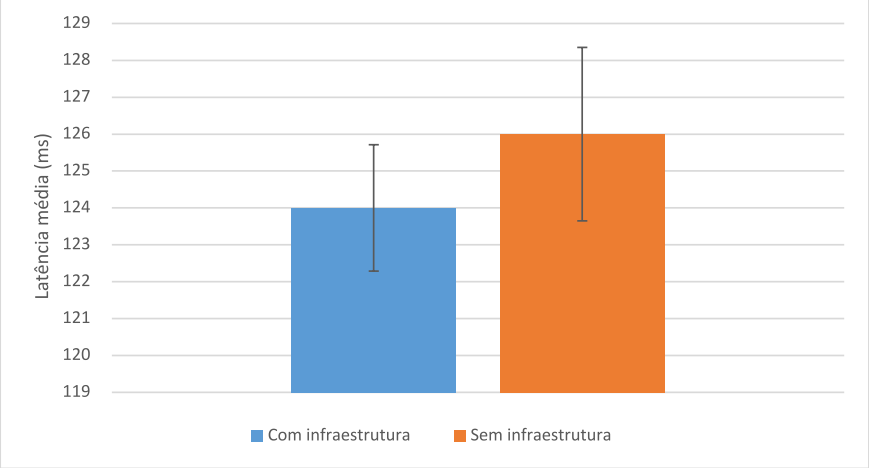

Figura 26 Latência obtida no experimento com o protótipo

Com infraestrutura o agente realizou duzentos e setenta migrações durante o teste, e em vinte e nove o agente foi perdido. Quando são observadas as migrações na situação sem infraestrutura, o agente realizou duzentos e onze migrações e se perdeu em vinte e três migrações. A Tabela 9 apresenta o resultado dos testes realizados em relação à migração e perda do agente.

Tabela 9 Relação quantidade migração e perda

	Migrações	Perda do agente	%
Sem infraestrutura	211	45	21,32%
Com infraestrutura	270	23	10,74%

O agente realiza mais migrações na situação com infraestrutura por que ela possui um nó a mais. Outro ponto importante é a porcentagem da taxa de perda do agente em relação à quantidade de migração. Como a infraestrutura permanece estática, a velocidade relativa entre o nó no veículo e o nó da infraestrutura é a velocidade do veículo. Então o tempo que o agente tem para migrar é maior por

que aumenta o tempo de comunicação entre os nós. No caso do experimento só com veículos, a velocidade relativa é a soma das velocidades dos dois veículos, isso diminui o tempo que o agente tem para migrar, causando o aumento da perda do agente. As Figuras 27, 28 e 29 exemplificam a velocidade relativa.

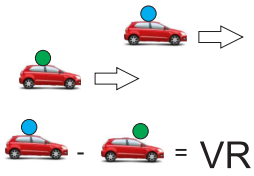

Figura 27 VR negativa

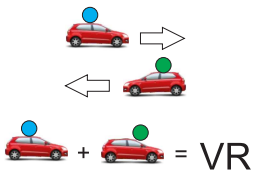

Figura 28 VR positiva

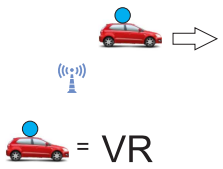

Figura 29 VR neutra

6 CONCLUSÕES

A área de redes veiculares tem um potencial muito amplo para realização de pesquisas. As redes veiculares em sua configuração híbrida, até o momento, não foram totalmente exploradas e podem ser aprimoradas.

Os agentes móveis adicionam às redes veiculares maior flexibilidade para trafegar dados. Através do agente os dados podem ser transportados levando em consideração o contexto, assim o agente decide se deve ou não migrar para outro nó. Outro ponto importante é que o agente pode decidir, dependendo do contexto, em usar outras formas de comunicação.

Nas simulações realizadas neste trabalho é possivel observar que a rede com infraestrutura melhora o desempenho do agente em comparação a uma rede sem infraestrutura. Porém, o impacto da infraestrutura em redes densas é menor. Essa informação é importante para auxiliar na escolha das regiões onde serão colocadas as infraestruturas. Regiões mais densas de veículos podem receber menos infraestruturas e regiões menos densas recebem mais infraestrutura.

No experimento com o protótipo desenvolvido, o ZigBee se mostrou viável para transportar agente de software simples. Porém, os agentes possuem uma complexidade maior que as informações transportadas no trabalho de Santana (2014) que obteve latência média de 64ms. No presente trabalho, a latência média foi de 124ms com infraestrutura e 126ms sem infraestrutura. Assim a latência se mostrou alta para aplicações críticas como, por exemplo, alerta de colisão entre veículos.

A taxa de perda do agente com infraestrutura foi de aproximadamente 10% nos experimentos com o protótipo, enquanto que sem infraestrutura foram obtidos aproximadamente 21% de perda. Essa informação mostra que o uso de infraestrutura como suporte na migração ajuda significativamente a diminuir a perda do agente.

6.1 Proposta de continuidade

Como proposta de continuidade para este trabalho, puderam ser adicionados mais dispositivos de hardware ao protótipo para que o agente móvel

possa extrair mais informações do ambiente para realizar a sua missão.

Outro trabalho a realizar é comparar o padrão ZigBee com o padrão 802.11p para transporte de agentes. Nesse trabalho é importante observar a latência e a perda do agente para comparar qual forma de comunicação é mais eficaz.

Desenvolver mecanismos mais eficientes para recuperar os agentes danificados durante a migração seria outra proposta de continuidade dos estudos para diminuir a taxa de perda do agente.

REFERÊNCIAS

ARDUINO. Disponível em: <http://www.arduino.cc/>. Acesso em:11 jan.2015.

BAI, F.; SADAGOPAN, N.; HELMY, A. Important: a framework to systematically analyze the impact of mobility on performance of routing protocols for adhoc networks. In: ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS, 22., 2003, San Francisco. **Proceedings...** San Francisco: IEEE, 2003. p. 825-835.

BHARGAV, K.; SINGHAL, R. Zigbee based vanets for accident rescue missions in 3g wcdma networks. In: GLOBAL HUMANITARIAN TECHNOLOGY CONFERENCE, 2013, Trivandrum. **Proceedings...** Trivandrum: IEEE, 2013. p. 310-313.

BURGESS, J. et al. Maxprop: routing for vehicle-based disruption-tolerant networks. In: INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS, 25., 2006, Barcelona. **Proceedings...** Barcelona: IEEE, 2006. p. 01-11.

CINTRA, M. **Os custos do congestionamento na cidade de São Paulo**. São Paulo: Editora da FGV, 2012.

DRESSLER, F. A study of self-organization mechanisms in ad hoc and sensor networks. **Computer Communications**, Guildfod, v. 31, n. 13, p. 3018-3029, Aug. 2008.

FALL, K. A delay-tolerant network architecture for challenged internets. In: CONFERENCE ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES, AND PROTOCOLS FOR COMPUTER COMMUNICATIONS, 2003, New York. **Proceedings...** New York: ACM, 2003. p. 27-34.

FREITAS, E. et al. Geo-aware handover of mission agents using opportunistic communication in vanet. In: BALANDIN, S.; DUNAYTSEV, R.; KOUCHERYAVY, Y. (Ed.). **Smart spaces and next generation wired/wireless networking:** volume 6294. Saint Petersburg: Springer Berlin Heidelberg, 2010. p. 365-376.

FREITAS, E. P. de et al. Analyzing different levels of geographic context awareness in agent ferrying over vanets. In: SYMPOSIUM ON APPLIED COMPUTING, 2011, New York. **Proceedings...** New York: ACM, 2011. p. 413-418.

FREITAS, E. P. de et al. Exploring geographic context awareness for data dissemination on mobile ad hoc networks. **Ad Hoc Networks**, Amsterdam, v. 11, n. 6, p. 1746-1764, Aug. 2013.

JIANG, D.; DELGROSSI, L. IEEE 802.11p: towards an international standard for wireless access in vehicular environments. In: VEHICULAR TECHNOLOGY CONFERENCE, 2008. **Proceedings...** Singapore: VTC Spring, 2008. p. 2036-2040.

KUMAR, R.; DAVE, M. A comparative study of various routing protocols in vanet. **International Journal of Computer Science Issues**, London, v. 8, n. 4, p. 643-648, July 2011.

LESSMANN, J.; HEIMFARTH, T.; JANACIK, P. Shox: an easy to use simulation platform for wireless networks. In: COMPUTER MODELING AND SIMULATION, 2008. Proceedings... Cambridge: IEEE, 2008. p. 410-415.

LIDSTRÖM, K.; LARSSON, T. A spatial QoS requirements specification for v2v applications. In: INTELLIGENT VEHICLES SYMPOSIUM, 4., 2010, San Diego. **Proceedings...** San Diego: IEEE, 2010. p. 548-553.

LOULLOUDES, N.; PALLIS, G.; DIKAIAKOS, M. D. **The dynamics of vehicular networks in urban environments**. Cyprus: Department of Computer Science, 2010.

NIAZI, M.; HUSSAIN, A. Agent-based tools for modeling and simulation of selforganizationin peer-to-peer, ad hoc, and other complex networks. **IEEE Communications Magazine**, New York, v. 47, n. 3, p. 166 -173, Mar. 2009.

NICHOLS, R. A.; HAMMONS, A. R. Performance of DTN-based free-space optical networks with mobility. In: MILITARY COMMUNICATIONS CONFERENCE, 2007. **Proceedings...** Orlando: IEEE, 2007. p. 01-06.

NICULESCU, D.; NATH, B. Localized positioning in ad hoc networks. **Ad Hoc Networks**, Amsterdam, v. 1, n. 2-3, p. 247-259, 2003.

NTPSYNC. Disponível em: <https://play.google.com/store/apps/details?id =org.ntpsync>. Acesso em: 15 jan. 2015.

PIORKOWSKI, M.; SARAFIJANOVIC-DJUKIC, N.; GROSSGLAUSER, M. A parsimonious model of mobile partitioned networks with clustering. In: COMMUNICATION SYSTEMS AND NETWORKS AND WORKSHOPS, Lausanne, 2009. **Proceedings...** Lausanne: [s.n.], 2009. p. 01-10.

PIRAN, M. J.; MURTHY, G. R.; BABU, G. P. Vehicular ad hoc and sensor networks; principles and challenges. **Ad Hoc Networks**, Amsterdam, v. 2, n. 2, p. 38-49, Aug. 2011.

RAMYA, C.; SHANMUGARAJ, M.; PRABAKARAN, R. Study on ZigBee technology. In: ELECTRONICS COMPUTER TECHNOLOGY, 3., 2011, Kanyakumari. **Proceedings...** Kanyakumari: IEEE, 2011. p. 297-301.

REZENDE, C. G.; ROCHA, B. P. S.; LOUREIRO, A. A. F. Publish/subscribe architecture for mobile ad hoc networks. In: SYMPOSIUM ON APPLIED COMPUTING, 2008, New York. **Proceedings...** New York: ACM, 2008. p. 1913-1917.

SAFARIC, S.; MALARIC, K. Zigbee wireless standard. In: MULTIMEDIA SIGNAL PROCESSING AND COMMUNICATIONS, 48., 2006, Zadar. **Proceedings...** Zadar: IEE, 2006. p. 259-262.

SANTANA, A. R. **Sistema de detecção de colisão entre veículos utilizando GPS eZigbee**. 2014. 77 p. Dissertação (Mestrado em Ciência da Computação) -Universidade Federal de Lavras, Lavras, 2014.

SHEHORY, O. et al. Agent cloning: an approach to agent mobility and resource allocation. **IEEE Communications Magazine**, New York, v. 36, n. 7, p. 63-67, July 1998.

SPYROPOULOS, T. et al. Routing for disruption tolerant networks: taxonomy and design. **Wireless Networks**, Amsterdam, v. 16, n. 8, p. 2349-2370, Nov. 2010.

SUN, J.-Z.; SAUVOLA, J. Mobility and mobility management: a conceptual framework. In: INTERNATIONAL CONFERENCE ON, 10., 2002, New York. **Proceedings..**: New York: IEEE, 2002. p. 205-210.

URRA, O. et al. Mobile agents in vehicular networks: taking a first ride. In: DEMAZEAU, Y. et al. (Ed.). **Advances in practical applications of agents and multiagent systems**. Saint Petersburg: Springer Berlin Heidelberg, 2010. p. 119-124.

WOOLRIDGE, M. **Introduction to multiagent systems**. New York: John Wiley & Sons, 2001.

WORLD HEALTH ORGANIZATION. **Global status report on road safety**. Switzerland: World Health Organization, 2013.

ZAFOUNE, Y.; KANAWATI, R.; MOKHTARI, A. Mobile agents localization in ad hoc networks: a comparative study of centralized and distributed approaches. In: INTERNATIONAL CONFERENCE ON, 5., 2007, Cairo. **Proceedings...** Cairo: IEEE, 2007. p. 269-275.

ZIGBEEALLIANCE. Disponível em: <http://zigbee.org/>. Acesso em: 18 jan. 2015.

ANEXO A - Classe Java do Agente

```
1
2
3  public class SoftwareAgent implements Agent{
4
5      private TargetRegion target; //região alvo
6
7      public Agent(TargetRegion t) {
8          target = t;//seta a região alvo quando o agente é criado
9      }
10
11     public TargetRegion getTarget() {
12         return target;//retorna quando o nó pede a região alvo
13     }
14
15
16     @Override
17     public void execute(VehicleLayer vehicle) {//classe que executa as
            instrucoes no vehicle
18         GetDestinationCommand command = new GetDestinationCommand(
               vehicle.getNode().getId().asInt());
19         command.send();//envia o comando para o vehicle enviar os
               agentes de suporte
20         if(vehicle.getAgent() == null)//se o vehicle nao tiver agente
               executa as instruções
21             vehicle.sendEventSelf(new CollisionAvoidanceWUC(vehicle.
```

```
                getSender(), Math.random() * 5,
22              new ReturnDestinyPacket(vehicle.getSender(), this.
                    getSender().getId(), command.getGPS(), vehicle.
                    getNode().getPosition())));
23          }
24
25      }
```

ANEXO B - Classe Java do modelo de comportamento do nó

```
1
2  public class VehicleLayer extends ApplicationLayer {
3
4      private Agent agent;//agente hospedado no veiculo
5
6      public Agent getAgent(){
7          return this.agent;
8      }
9
10     public void setAgent(Agent agent){
11         this.agent = agent;
12     }
13
14
15
16
17
18     @Override
19     public void lowerSAP(Packet packet) throws LayerException {
20
21         if (packet.getSender() != sender) {//enviar o agente
22             ((PacketTask) packet).execute(this);
23         }
24
25     }
```

```
26
27      protected void processEvent(StartSimulation start) {
28          if (this.node.getId().asInt() == CityStartPositions.
                getStartNode()){
29              this.agent = new Agent(new TargetRegion());
30          }
31
32          TargetRegionCheckerWUC wuc = new TargetRegionCheckerWUC(sender
                , 1000);//Verifica se o vehicle esta dentro da região alvo
33          this.sendEventSelf(wuc);
34      }
35
36      protected void processEvent(Finalize finalize) {
37          if (this.node.getId().asInt() == 1)
38              GetStatistics.getInstance().generateStatistics();
39      }
40
41      public void processWakeUpCall(WakeUpCall wuc) {//ativa o vehicle
42          try{
43              if(!(wuc instanceof DecisionEvent) || this.getAgent() !=
                    null)//verifica se tem agente
44                  ((WakeUpCallTask) wuc).execute(this);// executa as
                        instruções do agente
45          }catch (Exception e) {//tratamento de erro quando o agente é
                perdido
46              System.out.println("Pacotes com problemas = " + wuc.
                    getReceiver() + " qual o problema " + wuc.getClass().
                    getCanonicalName());
```

```
47                e.printStackTrace();
48            }
49        }
50    }
```

ANEXO C - Classe Java do modelo de movimento

```
1
2  public class CityMovement extends MovementManager {
3
4      //Inicia a configuração da simulacao
5      public void initConfiguration(Configuration config)
6              throws ConfigurationException {
7          super.initConfiguration(config);
8      }
9
10     //verifica se o veiculo chegou dentro da regiao alvo
11     private boolean isNear (Position p1, Position p2) {
12         boolean result = false;
13         if (p1.equals(p2))
14             result = true;
15
16         return result;
17     }
18
19     //faz o veiculo se locomover
20     private Position interpolatedPosition(Position pinitial, Position
           pfinal){
21         double distance = pinitial.getDistance(pfinal);
22         double step = 5.0;
23         double p = step/distance;
24         if (p>1) p=1;
```

```
25          double newx = pinitial.getXCoord() * (1-p) + pfinal.getXCoord
                () * p;
26          double newy = pinitial.getYCoord() * (1-p) + pfinal.getYCoord
                () * p;
27          return new Position(newx, newy);
28      }
29
30      //armazena os movimentos
31      public final Collection<Movement> createMoves(Collection<Node>
            allNodes) {
32          Iterator<Node> iter = null;
33          iter = allNodes.iterator();
34          List<Movement> nextMoves = new LinkedList<Movement>();
35          while (iter.hasNext()) {
36              Node node = iter.next();
37              Position next = Routes.nextMoviment(node.getId().asInt());
38
39              if(this.isNear(next,node.getPosition())){
40                  Routes.changeMoviment(node.getId().asInt());
41              }
42
43              Position newPos = this.interpolatedPosition(node.
                    getPosition(), next);
44
45              nextMoves.add(new Movement(node, newPos, 0));
46          }
47
48
```

```
49          return nextMoves;
50      }
51
52      //envia o agente com as informações do destino
53      public void sendCommand(Command c) {
54          if (c instanceof GetDestinationCommand) {
55              GetDestinationCommand command = (GetDestinationCommand) c;
56              command.setDestiny(Routes.getDestiny(command.getNode()));
57              command.setGPS(Routes.GPS(command.getNode()));
58          }
59      }
60
61  }
```