UNIVERSIDADE ESTADUAL PAULISTA

“Júlio de Mesquita Filho”

Pós-Graduação em Ciência da Computação

MÁRIO CLEBER BIDÓIA

SIMULAÇÃO DE UM SISTEMA DE REPUTAÇÃO CENTRALIZADO PARA VANETs

Sao José do Rio Preto

2015

MÁRIO CLEBER BIDÓIA

SIMULAÇÃO DE UM SISTEMA DE REPUTAÇÃO CENTRALIZADO PARA VANETs

Orientadora: Profa. Dra. Roberta Spolon

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação – Área de Concentração em Computação Aplicada, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

São José do Rio Preto
2015

Bidóia, Mário Cleber.
Simulação de um sistema de reputação centralizado para VANETs / Mário Cleber Bidóia. -- São José do Rio Preto, 2015
139 f. : il., gráfs., tabs.

Orientador: Roberta Spolon
Dissertação (mestrado) – Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas

1. Computação. 2. Tecnologia da informação - Sistemas de segurança. 3. Redes ad hoc veiculares (Redes de computadores) 4. Segurança de trânsito. I. Spolon, Roberta. II. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU – 681.3.025

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Mário Cleber Bidoia

Simulação de um sistema de reputação centralizado para VANETs

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Comissão Examinadora

Profª. Drª. Roberta Spolon
UNESP – Bauru
Orientador

Prof. Dr. Aparecido Nilceu Marana
UNESP – Bauru

Prof. Dra. Sarita Mazzini Bruschi
USP – ICMC São Carlos

São José do Rio Preto

13 de outubro de 2015

AGRADECIMENTOS

Agradeço à minha amada esposa, Aline Luiza, pela sua paciência nos momentos em que estive ausente. Por sempre me apoiar e me incentivar a continuar mesmo com todos obstáculos que apareceram. Sem ela ao meu lado não teria conseguido chegar até o final.

Agradeço aos meus pais por me darem educação e suporte necessários para que eu pudesse realizar este trabalho.

Agradeço aos familiares e amigos que, indiretamente, colaboraram com a realização deste trabalho.

Agradeço ao Professor Emerson Marconato, meu professor da faculdade e hoje colega de trabalho, com quem sempre pude contar com as ajudas sobre Linux.

Agradeço ao Prof. Dr. Marcos Cavenaghi, que me deu a oportunidade de iniciar o mestrado e que, mesmo distante, continuou a me dar orientações.

Agradeço ao Programa de Pós-Graduação da UNESP que forneceu recursos para este trabalho ser realizado e eu conseguisse o título de Mestre.

Agradeço à Profa. Dra. Roberta Spolon, pela continuidade no desenvolvimento do trabalho. Deu total apoio e suas orientações foram fundamentais para eu finalizar esta dissertação.

RESUMO

Atualmente um paradigma de redes Ad Hoc tem sido amplamente estudado e desenvolvido, estas são as redes Ad hoc veiculares (VANETs – *Vehicular Ad hoc NETworks).* Partindo do pressuposto que os carros estão cada vez mais "inteligentes", as VANETs surgiram com a proposta de proporcionar a comunicação entre os veículos das rodovias. Esta comunicação visa garantir maior segurança para os membros envolvidos no trânsito. Para garantir esta segurança é necessário que os dados trocados entre os veículos sejam confiáveis. Uma das técnicas abordadas nesta dissertação é o uso de sistemas de reputação, no qual a confiança nos veículos é baseada em votos de seus vizinhos. Ou seja, veículos com mais votos positivos, mais confiáveis são e vice-versa. Neste trabalho, foi modelado e simulado um sistema de reputação centralizado, em que uma unidade central é responsável por gerenciar as reputações. Com este sistema centralizado as reputações dos veículos que já foram avaliados ficam armazenadas por tempo indeterminado e podem ser consultadas a qualquer momento e em qualquer local.

Palavras-chaves: VANET, segurança, sistema de reputação

ABSTRACT

Currently a network paradigm Ad HoC has been widely studied and developed, these are the vehicular ad hoc networks (VANETs - Vehicular Ad hoc Networks). Assuming that cars are increasingly "smart", the VANETs emerged with the proposal to provide communication between vehicles of highways. This Communication aims to ensure greater security for members involved in transit. To seek to ensure that security is necessary that the data exchanged between vehicles are reliable. One of the techniques discussed in this proposal is the use of reputation systems. In this technique the confidence in the vehicles is based on votes from its neighbors. That is, vehicles with more positive votes, more reliable and vice versa. In this work, the proposal was to model and simulate a centralized reputation system, where a central unit is responsible for managing reputations. With this centralized system the reputations of vehicles that have been evaluated are stored indefinitely and can be accessed anytime and anywhere.

Keywords: VANET, security, reputation system

LISTA DE FIGURAS

Figura 1 - Arquitetura básica de uma VANET.....16
Figura 2 - Um veículo equipado com os componentes. 19
Figura 3 - Disseminação de mensagem em cenário de perigo. 21
Figura 4 - Exemplo de um ataque de alteração. 25
Figura 5 - Divisão de Espectro utilizado pela FCC.....30
Figura 6 - Arquitetura do NCTUns. 50
Figura 7 - Arquitetura distribuída do NCTUns. 52
Figura 8 - Arquitetura da VANET com Reputação Centralizada. 63
Figura 9 - Modelo Cliente-Servidor. 64
Figura 10 - Arquitetura para atualizar reputação.....67
Figura 11 - Aba Application do editor de nó.....69
Figura 12 - Régua do NCTUns.....73
Figura 13 - Ambiente urbano simulado.....74
Figura 14 - Comprimentos das ruas e cruzamentos do ambiente simulado 74
Figura 15 - Áreas cobertas pela RSUs.....75
Figura 16 - Configuração do arquivo de profile do veículo. 76
Figura 17 - Distribuição dos profiles entre os veículos da rodovia. 76
Figura 18 - Configuração do tempo de simulação. 77
Figura 19 - Configuração do envio de alerta. 77
Figura 20 - Ambiente para teste do cenário 1. 78
Figura 21 - Cenário 01 simulado no NCTUns 79
Figura 22 - Veículo atacante na Região 3. 80
Figura 23 - Cenário 02 simulado no NCTUns. 81
Figura 24 - Ambiente para o cenário 03. 82
Figura 25 - Cenário 03 simulado no NCTUns. 83
Figura 26 - Ambiente para o cenário 04. 84
Figura 27 - Cenário 04 simulado no NCTUns. 85
Figura 28 - Gráfico das distâncias percorridas pelos veículos no cenário 01 87
Figura 29 - Gráfico das velocidades antes e depois de receberem alerta no cenário 01. 87
Figura 30 - Gráfico da média de velocidade no cenário 01.....88
Figura 31 - Gráfico dos tempos percorrido pelos veículos no cenário 01.....89

Figura 32 - Gráfico das médias de distância e tempo percorrido pelos veículos no cenário 01. 89
Figura 33 - Posição dos veículos após o final da simulação. 90
Figura 34 - Gráfico das distâncias percorridas pelos veículos no cenário 02. 91
Figura 35 - Gráfico das velocidades antes e depois de receberem alerta no cenário 02. 92
Figura 36 - Gráfico da média de velocidade no cenário 02..... 92
Figura 37 - Gráfico dos tempos percorridos pelos veículos no cenário 02. 93
Figura 38 - Gráfico das médias de distância e tempo percorrido pelos veículos no cenário 02. 94
Figura 39 - Posição dos veículos após o final da simulação do cenário 02. 95
Figura 40 - Gráfico das distâncias percorridas pelos veículos no cenário 03. 96
Figura 41 - Gráfico das velocidades antes e depois de receberem alerta no cenário 03. 97
Figura 42 - Gráfico da média de velocidade no cenário 03..... 97
Figura 43 - Gráfico dos tempos percorrido pelos veículos no cenário 03..... 98
Figura 44 - Gráfico das médias de distância e tempo percorrido pelos veículos no cenário 03. 99
Figura 45 - Posição dos veículos após o final da simulação do cenário 03 100
Figura 46 - Gráfico das distâncias percorridas pelos veículos no cenário 04. 101
Figura 47 - Gráfico das velocidades dos veículos antes e depois de receberem alerta no cenário 04. 102
Figura 48 - Gráfico da média de velocidade no cenário 04..... 102
Figura 49 - Gráfico dos tempos percorrido pelos veículos no cenário 04..... 103
Figura 50 - Gráfico das médias de distância e tempo percorrido pelos veículos no cenário 04. 104
Figura 51 - Posição dos veículos após o final da simulação do cenário 04. 105
Figura 52 - Tela inicial do NCTUns..... 150
Figura 53 - Modo Desenhar Topologia 151
Figura 54 - Itens ITS road segment 151
Figura 55 - Exemplo de um cenário de rodovia 152
Figura 56 - Componentes de RSU e veículos..... 152
Figura 57 - Selecionar modo Edit Property 153
Figura 58 - Caixa de diálogo para configurar propriedades do nó 153

Figura 59 - Modo *Run Simulation* 154

LISTA DE SIGLAS

API	Application Programming Interface
CMAC	Cipher-based MAC
DDoS	Distributed DoS
DoS	Denial of Service
GUI	Graphical User Interface
HMAC	keyed-Hash Message Authentication Code
IP	Internet Protocol
ITS	Intelligent Transport Systems
IDM	Intelligent Driver Model
IVG	Inter-Vehicle Geocast
LAN	Local Area Network
LDW	Local Danger Warnings
MAC	Message Authentication Code
MANETs	Mobile Ad hoc NETwork
NED	Network Description Language
OBU	OnBoard Unit
P2P	Peer-to-Peer
PTV	Planung Transport Verkehr
RME	Route Management and Execution
RSU	Road-Side Unit
RTM	Random Trip Mobility
SGBD	Sistema de Gerenciador de Banco de Dados
TCL	Tool Command Language
TCP	Transmission Control Protocol
TraCI	Traffic Control Interface
V2V	Vehicle to vehicle
V2I	Vehicle to infrastructure
VANETs	Vehicular Ad hoc NETworks
VARS	Vehicle Ad hoc Network Reputation System
UC	Unidade Central

SUMÁRIO

CAPÍTULO 1 - INTRODUÇÃO 12
CAPÍTULO 2 – VANETS – *Vehicular Ad hoc NETworks* 14
2.1 – Introdução 14
2.2 – Redes móveis *ad hoc* 14
2.3 – Arquitetura das VANETs 16
2.3.1 – Estações base 18
2.3.2 – Veículos 18
2.4 – Aplicações em VANETs 19
2.5 – Possíveis Ataques às Redes VANETs 21
2.5.1 – Ameaças à Disponibilidade 21
2.5.2 – Ameaças à autenticidade 23
2.5.3 – Ameaças à Confidencialidade 26
2.6 – Segurança em VANETs 27
2.6.1 - Mensagens de Segurança 27
2.6.2 - Protocolo para Mensagens de Segurança 29
2.6.3 – Ataques contra VANETs 30
2.6.4 – Atacante 30
2.7 – Ferramentas para segurança 31
2.7.1 – Criptografia 31
2.7.2 – Assinaturas Digitais 32
2.7.3 – Códigos de Autenticação de Mensagens 34
2.8 – Reputação 34
2.9 – Considerações Finais 36
CAPÍTULO 3 – SIMULADORES 37
3.1 – Simuladores de rede 37
3.1.1 – NS-2 38
3.1.2 – GloMoSim e QualNet 38
3.1.3 - OPNET 39
3.1.4 - OMNeT++ 40
3.1.5 - J-Sim 40

3.1.6 - SWANS....41
3.2 - Simuladores de mobilidade....41
3.2.1 - VisSim....41
3.2.2 - PARAMICS....42
3.2.3 - SUMO e MOVE....42
3.3 – Simuladores fortemente integrados....43
3.3.1 - SWANS++....44
3.3.2 - GrooveNet....45
3.3.3 - TraNS....45
3.3.4 - Veins....46
3.3.5 - ASH....46
3.4 - SIMULADOR NCTUNS....47
3.4.1 - Requisitos do Sistema....49
3.4.2 - Arquitetura do Simulados NCTUns....50
3.4.3 - Integração de Emulação e Simulação....53
3.5 – Considerações sobre os simuladores VANET....53
3.6 Considerações Finais....54
CAPÍTULO 4 – SISTEMAS DE REPUTAÇÃO PARA REDES AD HOC....55
4.1 - Introdução....55
4.2 – Modelo de Sistema de Reputação....55
4.3 – Sistemas de Reputação em MANET....56
4.4 – Sistemas de reputação em VANET....57
4.5 – Considerações Finais....61
CAPÍTULO 5 – SISTEMA DE REPUTAÇÃO CENTRALIZADO – SRC....62
5.1 - Introdução....62
5.2 – Arquitetura do Sistema de Reputação Centralizado....62
5.3 – Funcionamento do Sistema de Reputação Centralizado....65
5.4 - Alteração efetuadas no simulador NCTUns....68
5.4.1 – Programas Agentes....68
5.4.2 – CarAgentModificado....70
5.4.3 – RSUAgentModificado....70
5.4.4 – UCAgent....71
5.5 – Considerações Finais....71
CAPÍTULO 6 - SIMULAÇÃO DO SRC E RESULTADOS....72

6.1 – Introdução 72
6.2 – Comportamentos avaliados 72
6.3 – Descrição do cenário da simulação do SRC 73
6.3.1 – Cenário 1 77
6.3.2 - Cenário 2 79
6.3.3 – Cenário 3 81
6.3.4 – Cenário 4 83
6.4 – Resultados das Simulações 85
6.4.1 – Cenário 01 86
6.4.2 – Cenário 02 90
6.4.3 – Cenário 03 95
6.4.4 – Cenário 04 100
6.5– Considerações Finais 105
CAPÍTULO 7 – CONCLUSÕES 109
7.1. Contribuições desse trabalho 109
7.2. Dificuldades encontradas 110
7.3. Trabalhos futuros 111
REFERÊNCIAS BIBLIOGRÁFICAS 112
APÊNDICE A – conexao_BD.h 118
APÊNDICE B – broascast.h 122
APÊNDICE C – MeuCarAgent.cc 126
APÊNDICE D – MeuRSUagent.cc 142
Anexo A - Instalação do simulador NCTUns 148
Anexo B - Interface do NCTUns 150

CAPÍTULO 1 - INTRODUÇÃO

Segundo (KONOMI e ROUSSOS, 2007), sob a óptica da computação pervasiva ou ubíqua (ou onipresente), todos os lugares conterão dispositivos com capacidade de processamento de dados totalmente integrados, comunicando-se por meio de uma rede sem fio de forma que a tecnologia responsável pela implementação dessa comunicação fique oculta aos olhos de seus usuários. Diversos esforços necessitam ser feitos em termos de pesquisas até se chegar a esse cenário. Essa visão fomentou diversas pesquisas na área da computação ubíqua.

Existem hoje diversos mecanismos, técnicas, métodos e sistemas que surgiram de trabalhos prévios e que estão sendo usados para construir sistemas reais que implementam ambientes onde a computação seja onipresente, como por exemplo, os sistemas e serviços apresentados em (PARK, 2009). Tais sistemas, apesar de ainda bastante limitados, já provaram ser capazes de suportar aplicações reais.

Mesmo estes sistemas não estando suficientemente completos para implementar de forma plena a computação ubíqua, eles corroboram para o entendimento dos problemas ainda não resolvidos, apontando rumos e levantando novos problemas que ainda deverão ser tratados.

Um dos problemas apontados pela comunidade científica e que ainda está longe de ser resolvido completamente é a privacidade ou segurança das informações dos usuários desses sistemas onipresentes (SPIEKERMANN e LANGHEINRICH, 2009). Durante as interações entre usuário e sistema, diversas informações pessoais podem ser coletadas.

Tais informações podem comprometer a privacidade do usuário, expondo-o a um adversário. Por se tratar de um sistema onipresente, o adversário poderá estar em qualquer lugar. Isso faz com que o desafio de implementar sistemas capazes de proteger a privacidade de seus usuários por meio da disponibilidade de serviços de segurança seja ainda maior.

Com base nestas teorias surgiu um paradigma na computação, são as chamadas Redes *Ad hoc* Veiculares (*Vehicular Ad hoc NETworks* – VANETs). Atualmente essas redes têm sido alvo de diversas pesquisas, principalmente voltadas para a área de segurança no trânsito, como informações do tráfego ou alertas de colisões. Contudo, como diz respeito à troca de informações entre os veículos, é importante que estas sejam confiáveis. Como este tipo de rede tem uma característica efêmera, devido à alta mobilidade e velocidade dos nós, é necessário utilizar técnicas mais robustas para garantir a segurança das informações trocadas pelos veículos.

Uma das propostas que tem sido estudada é baseada em sistemas de reputação. Nestes sistemas a confiabilidade dos veículos é baseada em uma coleção de opiniões a respeito de um determinado veículo. Esta técnica utiliza o uso de votos dos veículos na proximidade de certo indivíduo para definir se suas informações devem ser aceitas ou revogadas.

Para testar a eficiência e buscar novas técnicas para estes sistemas de reputação é preciso realizar determinadas simulações e propor, melhorar e modelar novas arquiteturas. Este trabalho tem por objetivo a implementação e simulação de um sistema de reputação centralizado. Para isso foi utilizado o simulador NCTUns (WANG et.al, 2007) para implementar e simular um sistema de reputação centralizado.

Os resultados obtidos com o sistema implementado e simulado demonstraram a eficiência do sistema centralizado para gerenciar a reputação dos veículos, para que os veículos maliciosos, uma vez que foram avaliados negativamente, tenham as reputações disponíveis a qualquer momento e em qualquer local. Desta maneira se um veículo teve sucesso em um ataque em um determinado momento, este não o terá nos próximos ataques, pois sua reputação já estará disponível para todos os outros veículos. Com a proposta da arquitetura abordada neste trabalho também se verificou a possibilidade de se ter uma quantidade indeterminada de reputações armazenas, já que os responsáveis por armazená-las são as Estações Bases, que possuem um espaço físico de armazenamento maior do que a dos veículos.

No capítulo 2 é apresentado um estudo sobre VANET e suas principais características, assim como as aplicações e tipos de mensagens que poderão executar na rede. No capítulo 3 são apresentados alguns simuladores de rede. No capítulo 4 são apresentados os conceitos e estudos mais recentes de sistemas de reputação para redes Ad Hoc. No capítulo 5 são apresentadas as características do sistema de reputação centralizado proposto neste trabalho. No capítulo 6 são apresentadas as simulações e seus respectivos resultados. No capítulo 7 são apresentadas as conclusões e sugestões de trabalhos futuros.

CAPÍTULO 2 – VANETS – *VEHICULAR AD HOC NETWORKS*

2.1 – Introdução

Neste capítulo é definida a arquitetura das VANETs (*Vehicular ad hoc Network*) e o que são redes móveis *ad hoc*. Como as VANETs buscam, além de outras funções, garantir a segurança dos motoristas e pedestres, é importante que ela satisfaça também a confiabilidade e a autenticidade deste tipo de rede. Assim, neste capítulo também são abordadas as principais ameaças, técnicas e ferramentas de segurança aplicadas a VANETs, com um maior destaque para os sistemas de reputação, tema principal desta dissertação.

2.2 – Redes móveis *ad hoc*

Uma evolução na área automobilística é a ideia de "consciência", significando que um veículo é consciente de sua vizinhança, inclusive da presença e localização de outros veículos. Veículos conscientes possuem uma rede de sensores conectados a uma central de processamento de dados que provê interfaces de comunicação baseadas nos padrões Ethernet, USB, Bluetooth, 802.11, dentre outros.

Neste cenário, um conjunto de veículos que se comunicam entre si é um exemplo de rede móvel sem fio inserida em um contexto de computação ubíqua. Neste caso, além dos sistemas implementarem a comunicação entre esses veículos, também precisarão proteger as informações pessoais de seus usuários.

Nos últimos anos, um grande número de pesquisas tem abordado o tema a cerca de MANETs (*Mobile Ad hoc NETwork*) (LI, JOSHI e FININ, 2013), (MENON et al, 2013), (NICE, 2013), (SINGH, JOSHI, e SINGHA, 2013). A princípio, estas pesquisas eram voltadas para a área militar e de emergência. Devido à popularização dos dispositivos móveis, como *palms* e *smartphones*, tem-se dado uma maior atenção para o desenvolvimento de aplicativos para estes dispositivos. Sendo assim, MANET pode ser definida da seguinte maneira (MANET, 2012):

> Uma MANET é um sistema autônomo de roteadores móveis (e *hosts* associados) conectados por enlaces sem fio - a união desses dispositivos forma um grafo arbitrário. Os roteadores são livres para se mover aleatoriamente e se organizar arbitrariamente, assim, a topologia da rede pode se alterar de forma rápida e imprevisível. Tal rede pode ainda operar isoladamente ou conectada à Internet.

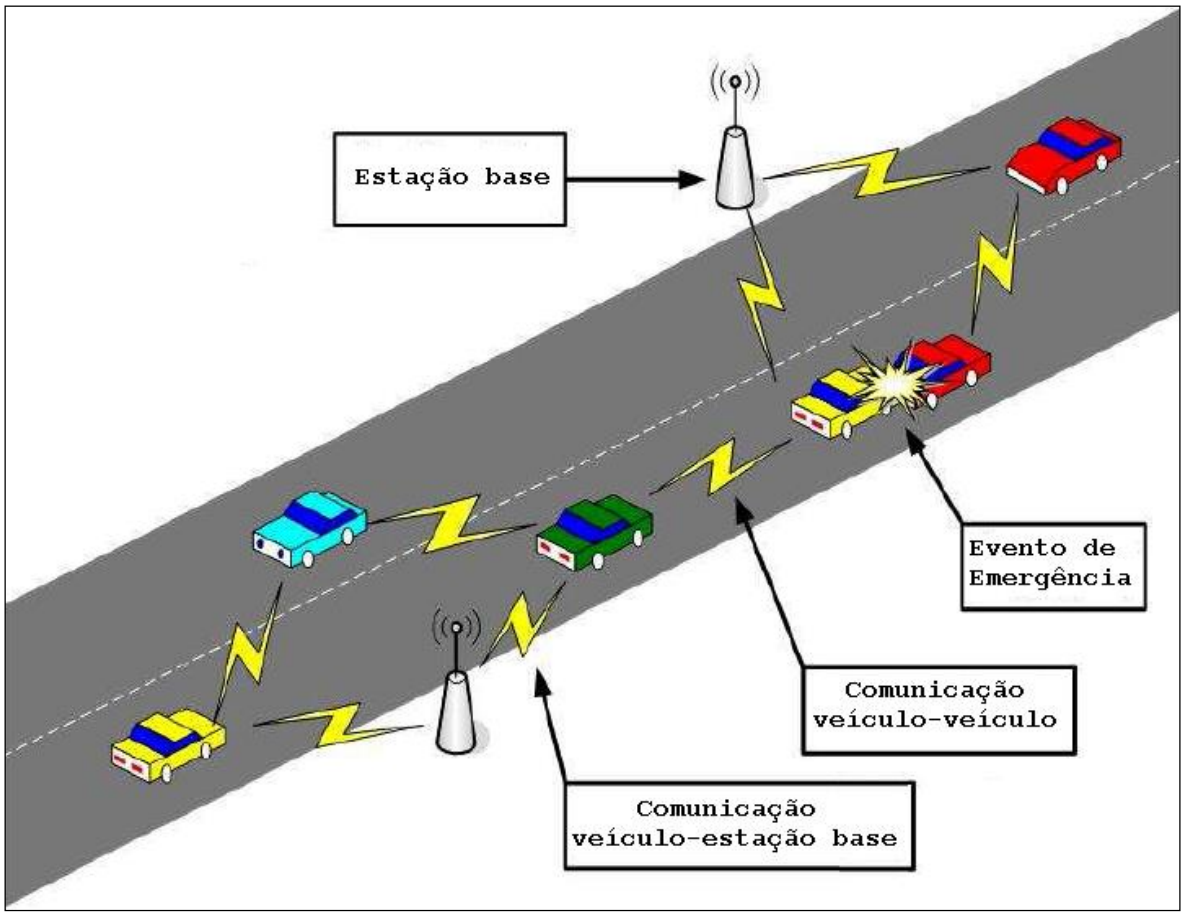
Um dos tipos de MANETs que tem sido amplamente estudado são as VANETs (*Vehicular Ad hoc NETworks*). Neste tipo de rede os veículos atuam como nós. Um dos principais objetivos desta rede é prover condições seguras para o tráfego dos veículos. Isto pode ser feito por meio da troca de informações entre os veículos, avisando a respeito de acidentes, congestionamentos, entre outros. Segundo (RAYA e HUBAUX, 2005), destacam-se como características das VANETs:

- **Banda disponível:** as tecnologias disponíveis atualmente possuem capacidades de transmissão significativamente menores que aquelas disponíveis em redes cabeadas;
- **Conectividade variável no tempo:** a conectividade da rede é dependente de fatores como sua densidade em determinado ponto, a velocidade de deslocamento dos veículos, o sentido desse deslocamento e o raio de alcance dos dispositivos móveis instalados nesses veículos. Assim, da mesma forma que podem existir áreas nas quais um veículo mantém um grande número de conexões, também é possível a formação de ilhas de conexão em certos pontos da rede;
- **Cooperação:** a funcionalidade da rede recai totalmente sobre a cooperação dos veículos que a compõem. Sem a participação destes, as informações geradas não se tornam de conhecimento geral;
- **Escala:** com milhares de veículos distribuídos por todos os lugares, as VANETs poderão se tornar a maior rede *ad hoc* móvel existente no futuro;
- **Mobilidade organizada:** diferentemente das MANETs tradicionais, os membros de uma VANET não se movimentam de maneira aleatória, mas sim dentro de vias de tráfego existentes e sob a regência de leis de circulação que definem, por exemplo, o sentido de circulação e a velocidade máxima permitida;
- **Topologia dinâmica da rede:** graças à alta mobilidade aliada a grandes velocidades, a topologia das VANETs pode mudar rápida e frequentemente;
- **Recursos energéticos e computacionais:** considera-se que VANETs possuem recursos energéticos e computacionais suficientes para as aplicações desenvolvidas, ao contrário das MANETs tradicionais;
- **Segurança:** com o acesso compartilhado ao meio, essas redes são muito mais suscetíveis a ataques que as redes cabeadas convencionais.

2.3 – Arquitetura das VANETs

A arquitetura de uma VANETs é formada basicamente pela comunicação entre os veículos, por meio de conexões *wireless*, e também entre os veículos e estações base fixas que compõe a infraestrutura da rodovia ou rua (Figura 1). Embora existam diversas aplicações vislumbradas para as VANETs (Seção 2.4), o objetivo principal da comunicação é a troca de informações de forma a aumentar a segurança e a eficiência do tráfego na via. Veículos podem ser particulares (pertencentes a uma pessoa/empresa) ou públicos (pertencentes a organizações públicas como, por exemplo, os ônibus de transporte urbano). As estações base podem pertencer a órgãos públicos ou concessionárias que administram tal infraestrutura. Nesta arquitetura, os componentes da rede podem trocar informações a respeito da condição do tráfego na rodovia, por exemplo, envio de mensagem de alerta de acidente, de modo que desta maneira os veículos poderiam diminuir sua velocidade.

Figura 1 - Arquitetura básica de uma VANET



Fonte: Adaptada de Raya e Hubaux, 2005.

Devido à maioria dos nós da rede ser composta por veículos, tem-se uma mobilidade permanente na dinâmica da rede. Como consequência, o tempo de conexão entre vizinhos é curto, pois em rodovias a velocidade média dos veículos é acima de 100Km/h, com velocidades relativas acima de 200Km/h. Outro aspecto da dinâmica da rede é que as trajetórias dos veículos são definidas pelo traçado das rodovias/ruas.

Outra característica importante é o tamanho da VANET. Ainda que possua milhões de nós distribuídos por todo o mundo, a comunicação nessa rede será local, permitindo o particionamento da rede em sub-redes menores e totalmente escaláveis.

Uma vantagem de uma VANET sobre uma rede *ad hoc* convencional é que os veículos não estão limitados em termos de poder computacional e consumo de energia. Um veículo poderá ter dezenas ou centenas de microprocessadores interligados, cada um responsável pelo processamento de dados específicos.

Segundo (RAYA e HUBAUX, 2005), as aplicações que serão desenvolvidas para uma VANET podem ser divididas em duas categorias:

- **Aplicações Relacionadas à Segurança:** são aplicações para evitar colisões, fornecer direção cooperativa e otimização de tráfego. A característica comum às aplicações dessa categoria é a preocupação com as situações onde uma falha no sistema pode causar danos à vida de seres humanos. Assim, o perfeito funcionamento das aplicações nessa categoria é fundamental, independentemente das possíveis ameaças introduzidas por um adversário.
- **Outras Aplicações:** estão incluídas nessa categoria as aplicações relacionadas ao pagamento de serviços (ligações telefônicas, por exemplo), serviços de localização (de um posto de combustível, por exemplo), entretenimento (acesso à internet, por exemplo), dentre outros. Logicamente, alguns desses serviços demandam segurança, mas por não envolver diretamente o risco à vida humana, são classificados nessa categoria. Além do mais, o problema de segurança dessas aplicações não está diretamente relacionado a uma rede VANET, mas sim a qualquer tipo de rede computacional.

2.3.1 – Estações base

Estações base RSU (Road Side Unit) são dispositivos que, na maioria das vezes, encontram-se posicionados na lateral da rodovia, com intuito de promover a comunicação entre a rede veicular e outros tipos de redes, como por exemplo, um serviço comercial que envia mensagens de propaganda sobre produtos.

Em alguns projetos estas estações são responsáveis pela criação e gerenciamento de chaves criptográficas. Estas podem ser administradas tanto por instituições governamentais como privadas.

2.3.2 – Veículos

Os integrantes principais das VANETs são os veículos. Estes são os principais responsáveis pelas trocas de informações, pois podem transmitir mensagens sobre o estado da rodovia, informando as suas condições em locais onde não há estações base.

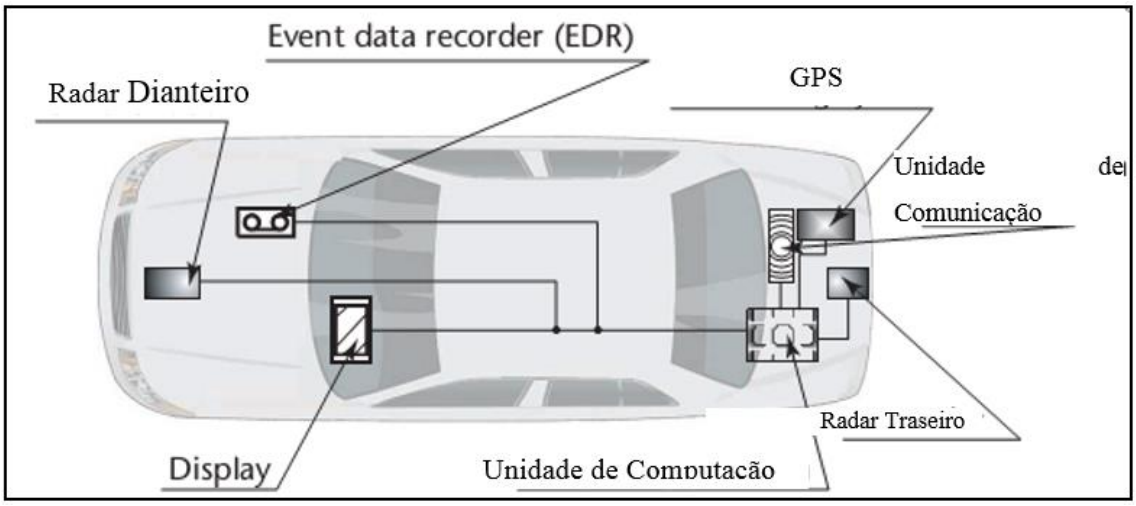
Estes veículos podem ser públicos, privados ou até mesmo oficiais. Todavia, estes integrantes da rede, chamados de "veículos inteligentes", necessitam de componentes extras para que possam efetuar a comunicação e executar os aplicativos. De acordo com (HUBAUX, CAPKUN e LUO, 2004) estes componentes são (Figura 2):

- **Sensores:** são os responsáveis pelo monitoramento dos eventos ocorridos com o veículo e ao seu redor. Variam desde um simples sensor de temperatura até um radar capaz de determinar a velocidade dos outros veículos trafegando na via;
- **Unidade de computação:** responsável por efetuar os cálculos e processar as informações recebidas;
- **GPS – *Global Position System:*** utilizado para informar a posição correta das ocorrências. Este recurso também pode ser utilizado pela unidade de computação para efetuar o cálculo de distância de uma ocorrência;
- ***Event Data Recorder*:** este dispositivo é responsável por gravar informações do veículo, para possível análise posterior. Foi baseado na caixa-preta utilizada

em aviões. Este pode ser utilizado, por exemplo, para a reconstituição de um acidente;

- **Unidade de comunicação:** habilita a comunicação sem fio com outros veículos e com as estações base;
- ***Display*:** apresenta de forma visual e amigável, as informações recebidas e processadas.

Figura 2 - Um veículo equipado com os componentes.



Fonte: Adaptado de HUBAUX, CAPKUN e LUO, 2004.

2.4 – Aplicações em VANETs

Há muitos tipos de aplicações propostas para as VANETs (Car2Car, 2009). Pode-se dividir estas aplicações em dois grandes grupos, segurança e controle de tráfego e entretenimento e outras.

1. Segurança e controle de tráfego
    a. Mensagem de Alerta: o veículo ao identificar um perigo na rodovia, como um acidente, ou um carro quebrado em uma curva, envia mensagens de alerta informando aos outros veículos a situação;
    b. Otimização do tráfego: visa melhorar o fluxo dos veículos por meio de mensagens informando a condição do transito em determinadas rodovias, possibilitando que outros veículos tomem rotas alternativas.

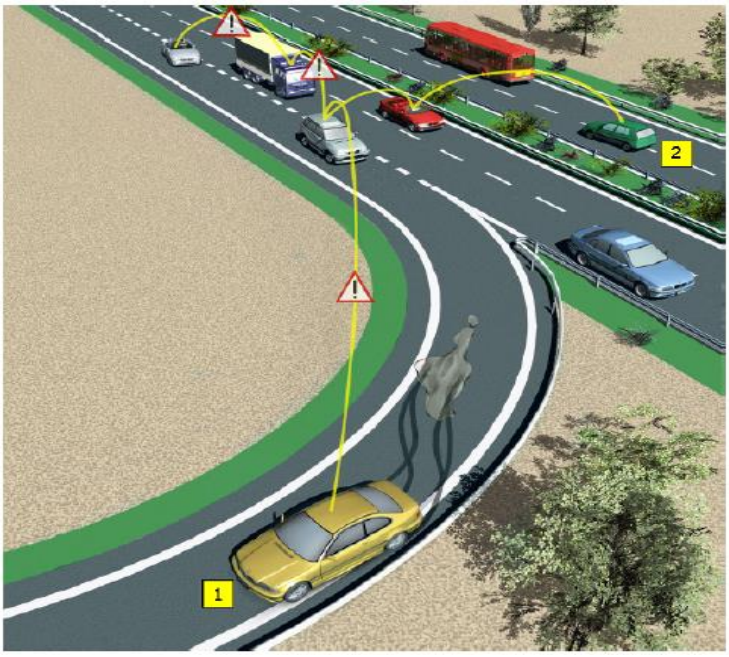
2. Entretenimentos e outras
    a. Acesso à Internet: veículos acessam a Internet por meio da infraestrutura da VANET;
    b. Propagandas: veículos recebem informações de propagandas comerciais, como preço de gasolina, desconto em determinada loja, entre outros.

Para este projeto são consideradas e estudadas as aplicações relacionadas à segurança e o controle de trafego.

Dentre as aplicações relacionadas à segurança, a troca de Mensagens de Perigo Local (*Local Danger Warnings* - LDWs) vem manifestando-se como a melhor técnica, devido ao benefício coletivo trazido pela disseminação de mensagens informando situações de risco na rodovia. Nesta técnica as informações são baseadas nas leituras obtidas pelos sensores locais dos veículos. Definida inicialmente em (KOSCH, 2004), essa aplicação também foi alvo de estudos em (OSTERMAIER, 2005), (ADLER e STRASSBERGER, 2006) e (CAO et al, 2010).

Nos últimos anos, o número de veículos nas cidades tem aumentado consideravelmente. Este aumento tem trazido alguns malefícios, como o aumento de congestionamento e de acidentes nas rodovias. Situações como essas além de representarem perda de tempo (diminui o fluxo de movimentos dos veículos), também representam perigos aos motoristas. Desta maneira, seria interessante se os motoristas fossem avisados previamente sobre essas situações, para que assim medidas preventivas pudessem ser efetuadas. Um cenário de perigo é ilustrado na Figura 3. Nesta imagem o veículo 1 passa por um local que contém alguma substancia escorregadia (por exemplo, óleo) na pista e derrapa, este veículo então alerta os demais veículos sobre este perigo naquele local.

Figura 3 - Disseminação de mensagem em cenário de perigo.



Fonte: (Kosch, 2004)

2.5 – Possíveis Ataques às Redes VANETs

Um ataque é uma tentativa do inimigo de efetuar alguma ação não prevista na rede, com o objetivo de prejudicar o funcionamento das aplicações ou obter benefícios. Embora seja impossível prever todos os possíveis ataques em VANETs, uma classificação geral pode ser adotada, com base nos tipos de ataques identificados em Raya e Hubaux (2005). Estes ataques podem ser classificados em três grupos principais: os que representam uma ameaça à disponibilidade, aqueles que representam uma ameaça à autenticidade, aqueles que representam uma ameaça à confidencialidade do motorista e diversos.

2.5.1 – Ameaças à Disponibilidade

Um dos requisitos de segurança é que os serviços estejam disponíveis a todos a qualquer momento. Porém alguns tipos de ataques buscam exatamente comprometer esse requisito, tornando o serviço indisponível aos seus usuários Raya e Hubaux (2005). Esta seção descreve esses possíveis ataques.

- **Ataque de Negação de Serviço (DoS – *Denial of Service*)**

No DoS o objetivo principal é evitar que usuários legítimos acessem os serviços e recursos da rede. O ataque DoS pode ocorrer por congestionamento do canal de comunicação do sistema fazendo com que veículos autênticos não possam acessá-lo. Em VANETs pode se considerar um problema mais sério, quando os usuários não podem se comunicar com a rede e passar informações a outros veículos, o que poderia resultar em um desastre em aplicações crítica de vida. Este pode ser feito de três maneiras diferentes:

- No nível básico o atacante sobrecarrega o recurso do nó de modo que este não pode realizar outras tarefas, o que resulta em tornar o nó continuamente ocupado e incapaz de realizar qualquer outra tarefa;
- Em um nível estendido o atacante congestiona o canal gerando uma alta frequência de dados no canal, fazendo com que os veículos não possam se comunicar com outros veículos na rede;
- Derrubando os pacotes da rede.

- **Ataque de Negação de Serviço Distribuído (DDoS – *Distributed DoS*)**

Um ataque DDoS é mais severo que um ataque DoS devido a sua característica distribuída. Neste ataque os usuários lançam os ataques de localizações diferentes. Podem ser utilizados intervalos de tempo diferentes para enviar as mensagens. O intervalo de tempo e a natureza das mensagens podem ser variados de veículos para veículos dos atacantes. O principal objetivo é derrubar a rede tornando-a indisponível para os usuários. As duas possibilidades de ataque DDoS são Veículo para veículo e Veículo para infraestrutura.

- **Ataque de *Spam***

Para consumir a largura da banda da rede e para aumentar a latência da transmissão, atacantes enviam mensagens de *spam* pela rede. Este tipo de ataque é difícil de controlar devido à falta de infraestrutura e administração centralizada. Neste tipo de ataque os atacantes disseminam mensagens de *spam* para um grupo de usuários. Estas mensagens não interessam para os usuários como, por exemplo, mensagens publicitárias.

- ***Malwares – Malicious Softwares***

Estes tipos de ataques são como vírus na VANET o que dificulta o funcionamento normal da rede. VANET são infectadas por estes ataques normalmente em softwares

desatualizados em veículos ou em Estações Base. Neste tipo de ataque os atacantes são normalmente membros internos da rede.

2.5.2 – Ameaças à autenticidade

A autenticidade trata da legitimidade da informação seja no formato físico ou eletrônico. Desta maneira são utilizados mecanismos de controle de acesso, em comunicações e transações eletrônicas, assim como na autenticação de arquivos Raya e Hubaux (2005). Esta seção apresenta alguns ataques que buscam comprometer essa autenticidade.

- **Ataque Sybil**

Este é um ataque crítico. Neste tipo de ataque um atacante transmite múltiplas mensagens com diferentes IDs para outros veículos. Desta maneira, os outros veículos consideram que as mensagens são provenientes de veículos diferentes, e que há um congestionamento adiante e estes são forçados a buscar uma rota alternativa. Em outras palavras pode-se dizer que a principal tarefa do atacante é fornecer a ilusão de múltiplos veículos para os demais veículos e força-los a escolher uma rota alternativa e deixar a estrada livre para utilização do atacante.

- **Ataque de representação de nó**

Nas VANET cada veículo tem um ID único e com a ajuda destes IDs cada veículo é identificado na VANET. Isto se torna mais importante quando ocorre um acidente. Num ataque de representação de nó um atacante pode trocar o seu ID e atuar como o originador real da mensagem. Um atacante recebe a mensagem e troca o conteúdo da mensagem em seu benefício. Posteriormente, o atacante envia esta mensagem para os outros veículos.

- **Ataque de Tempo**

Tempo é um aspecto crucial em qualquer aplicação em que o usuário necessita de uma informação precisa no momento certo, sem qualquer atraso. Tempo também é uma importante questão em aplicações de segurança em ITS (*Intelligent Transport Systems - Sistemas de Transportes Inteligentes*). Neste ataque, o atacante sem manipular o conteúdo atual adiciona um intervalo de tempo para criar um atraso na mensagem, fazendo com que o usuário receba a mensagem após o tempo requerido. Aplicações de segurança em ITS são aplicações de tempo

crítico, o que exige a transmissão dos dados no momento exato, caso contrário podem ocorrer acidentes.

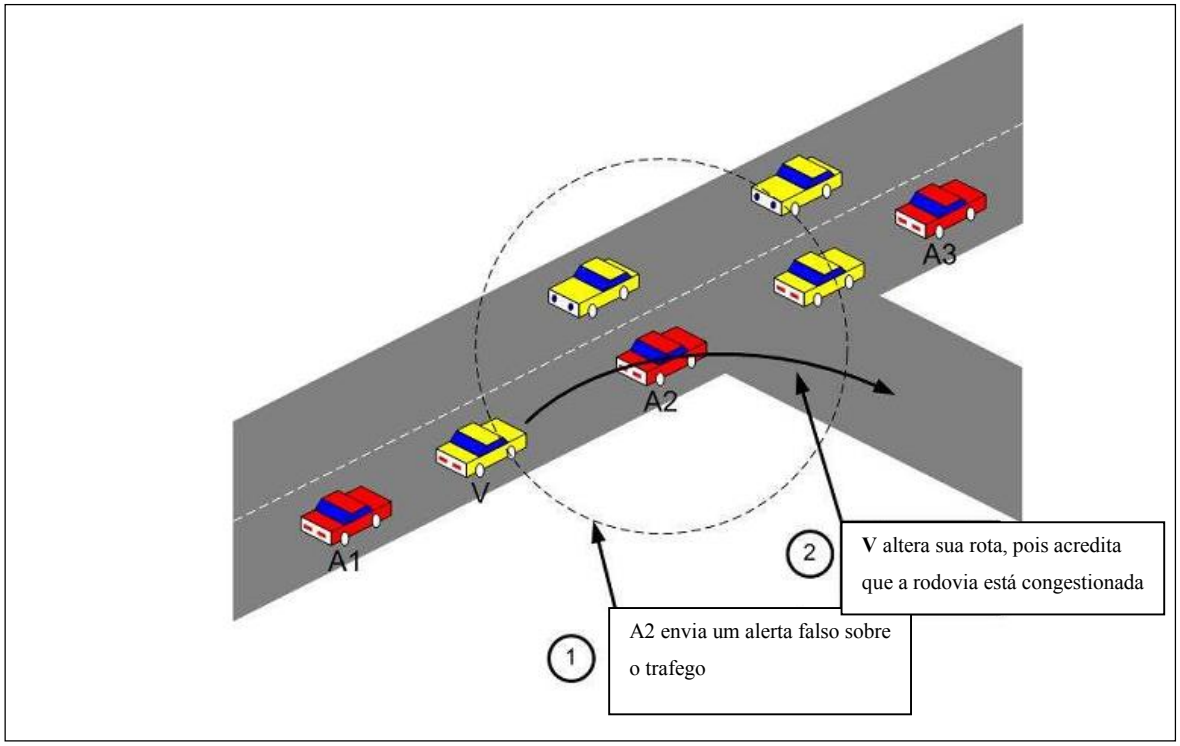
- **Supressão de Mensagem**

Neste tipo de ataque o atacante pode seletivamente descartar pacotes da rede que contém informação crítica para o recebedor. Por exemplo, o atacante pode remover os alertas de congestionamento recebido, para evitar que os nós escolham uma rota alternativa para o destino e forçá-los a esperar no trânsito. O atacante também pode utilizar estes pacotes posteriormente em seu benefício. O principal objetivo do atacante é evitar que autoridades e Estações Base saibam, por exemplo, sobre uma colisão ou acidente.

- **Alteração**

Como o nome sugere, este tipo de ataque significa alterar ou modificar um dado existente. Este ataque pode ocorrer atrasando a transmissão de mensagens de maneira deliberada, repetindo mensagens transmitidas anteriormente ou alterando partes específicas da mensagem. Por exemplo, como ilustra a Figura 4, o atacante, veículo **A2**, recebe mensagem que o tráfego na rodovia está normal, porém manipula a mensagem e enganosamente indica um tráfego intenso na rodovia (1), por consequência, o veículo V recebe esta mensagem falsa e altera sua rota (2).

Figura 4 - Exemplo de um ataque de alteração.



Fonte: Adaptada de Raya e Hubaux, 2005

- ***Bogus Information***

Neste ataque o atacante pode ser um usuário interno ou externo. O atacante envia informações falsas na rede veicular para afetar a decisão dos outros veículos. Por exemplo, um veículo pode imitar um tráfego intenso na rodovia para evitar que os veículos escolham esta rota. Este ataque é um exemplo de ataque de aplicação.

- **Falsidade de GPS**

Para manter a identidade e a localização geográfica de todos os veículos na rede, uma tabela de localização é mantida em um satélite GPS. O atacante utiliza um simulador de satélite GPS para gerar sinais que são mais efetivos que o sinal original. O atacante GPS produz falsas leituras por meio do simulador para enganar veículos a pensar que eles estão em local diferente.

2.5.3 – Ameaças à Confidencialidade

Confidencialidade é o conceito que assegura a proteção da informação contra o acesso e divulgação não autorizada, esteja essa armazenada, em processamento ou em transferência Raya e Hubaux (2005). Desta maneira faz se necessário conhecer os possíveis ataques contra a confidencialidade para assim buscar meios para impedir esses ataques. Nesta seção são apresentados alguns desses possíveis ataques.

- **Ataque de Homem ao Meio**

Como o nome sugere o atacante posiciona-se no meio da comunicação de dois veículos e lança o ataque. O atacante controla toda a comunicação entre o remetente e o recebedor, porém os veículos envolvidos acham que estão se comunicando diretamente. Neste tipo de ataque o atacante ouve a comunicação entre os veículos e modifica ou injeta falsa mensagem entre os veículos.

- **Análise de Tráfego**

Este ataque pode ser considerado um nível sério de ameaça à privacidade do usuário em VANET. Neste ataque o atacante analisa o tráfego de pacotes entre o V2V (*Vehicle to vehicle*) ou V2I (*Vehicle to infrastruture*). O atacante pode usar os pacotes que contém a localização do veículo e rotas de percursos que podem ser úteis para obter a informação requerida para as suas propostas.

- **Força Bruta**

Informações relacionadas à segurança são críticas em VANET. Para buscar garantir esta segurança na VANET, aplicações apropriadas de algoritmos criptográficos são amplamente utilizadas para protegê-la contra estes ataques. O atacante pode usar o ataque de força bruta para tentar quebrar a chave de criptografia.

2.6 – Segurança em VANETs

Um dos objetivos das VANETs é proporcionar segurança e comodidade aos condutores, contudo para que isso ocorra é preciso que os dados transmitidos sejam confiáveis.

De acordo com (RAYA e HUBAUX, 2005) um sistema seguro deve oferecer os seguintes requisitos de segurança:

- **Autenticação:** garantir que a informação recebida foi enviada por um membro confiável. Esta garantia pode ser feita por meio de uso de assinaturas digitais com chaves criptográficas;
- **Disponibilidade:** os integrantes da rede devem estar sempre disponíveis. Esta premissa é difícil de ser garantida devida a efemeridade das VANETs;
- **Não repúdio**:
    - Os veículos não podem alegar serem outros veículos (ataque de máscaras), pois todas as mensagens que transmitem são assinadas por suas chaves públicas;
    - Um veículo não pode negar ter enviado uma mensagem, porque é assinado por uma chave que pertence exclusivamente ao remetente, do mesmo modo, o veículo não pode alegar que a mensagem foi repetida porque o horário está incluído em cada mensagem;
    - Veículos não podem enganar sobre a sua posição e parâmetros relacionados se uma solução de posicionamento seguro é utilizada.
- **Privacidade:** os condutores não querem informações pessoais sendo transmitida pela rede.

Estes são os principais requisitos que um sistema pode executar com segurança. Estes requisitos são importantes para tentar evitar os diversos tipos de ataque que uma VANET pode sofrer.

2.6.1 - Mensagens de Segurança

Como apontado na Seção 2.4 neste trabalho, apenas as aplicações relacionadas à segurança são consideradas para estudos. Nesse contexto, podem-se classificar as mensagens

em três classes baseadas nas suas propriedades relacionadas à privacidade e necessidades de tempo real:

- **Mensagens sobre Informações de Tráfego:** são usadas para disseminar as condições de tráfego em uma determinada região, portanto, afetam a segurança dos usuários apenas de forma indireta (prevenindo acidentes em potencial devido ao congestionamento de veículos). Assim, elas não necessitam de um mecanismo de tempo elaborado;
- **Mensagens Gerais de Segurança:** são usadas por aplicações públicas de segurança, como por exemplo, direção cooperativa para evitar colisões. Tais aplicações são sensíveis ao tempo, necessitando ser enviadas dentro de intervalos de tempo curtos para que possam ser efetivas;
- **Mensagens de Responsabilidade:** distinguem-se das outras por envolver troca em situações de responsabilidade do emissor tais como em acidentes, catástrofes naturais, entre outros. Portanto, a responsabilidade do emissor da mensagem deve ser determinada revelando-se sua identidade às autoridades competentes.

Uma propriedade comum de todas as classes de mensagens é que cada mensagem chega ao seu destino por meio de *broadcast* e geralmente não possuem relação com outras mensagens anteriores. O conteúdo de uma Mensagem Geral de Segurança típica inclui a posição, velocidade, direção e aceleração do veículo, acrescentando mensagens específicas sobre eventos de tráfego (notificação de congestionamento ou acidente, por exemplo).

Caso o emissor encontre uma situação anormal (um acidente, por exemplo), esses dados ajudam os receptores a calcularem suas posições em relação ao emissor e a determinarem se eles estão em perigo. Nesse caso, as mensagens não necessariamente vêm acompanhadas do identificador do emissor.

Uma característica importante em redes *ad hoc* é o *multihopping*, em que alguns nós necessitam entregar os dados para a estação base através de nós intermediários. Mas, devido às características de *broadcast* do protocolo 802.11p (IEEE P802.11), as mensagens gerais de segurança são transmitidas sobre *singlehop*, no qual todos os nós sensores conseguem se comunicar diretamente com a estação base, com potência do sinal suficientemente grande para chegar a veículos dentro de um raio considerável, eliminando a necessidade de *multihop*.

Porém, alguma forma de *multihop* ainda pode ser implementada: caso um veículo receba uma mensagem e julgue que o conteúdo daquela mensagem poderia afetar a segurança de outros

veículos trafegando mais atrás, esse veículo poderia reenviar a mensagem, avisando aos outros veículos dos problemas à frente.

2.6.2 - Protocolo para Mensagens de Segurança

Comitês estão dedicando esforços para finalizar os padrões para VANET:

- IEEE 1609.x;
- IEEE 802.11p;
- Wireless Access in Vehicular Environment (WAVE).

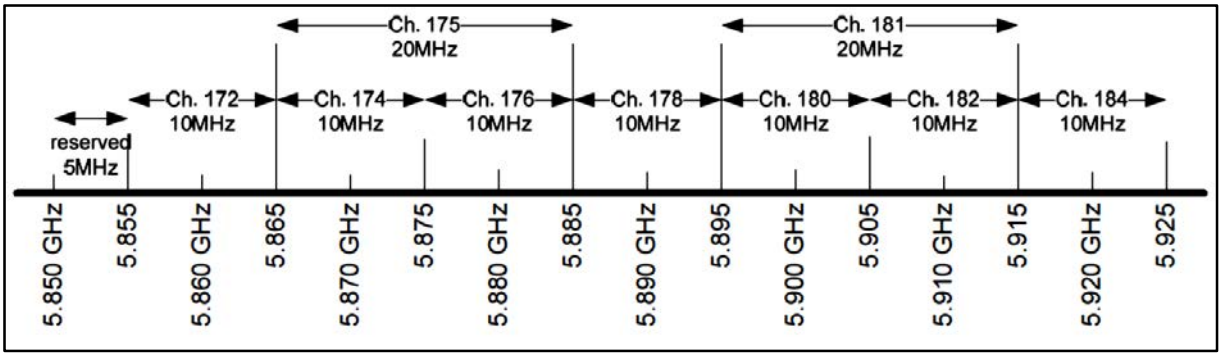
Em linhas gerais, um protocolo para mensagens de segurança deverá possuir algumas características (IEEE WAVE, 2013):

- De acordo com as especificações do IEEE 802.11p, assume-se que cada veículo V envia mensagens periódicas do tipo *singlehop* a cada 300ms com um raio de alcance de 110m a 300m;
- O intervalo entre cada mensagem cai para 100ms e o raio para 15m se o veículo estiver parado ou a uma velocidade baixa (menor que 16Km/h, por exemplo);
- Cada veículo toma decisões com base nas mensagens recebidas, podendo transmitir mensagens novas relacionadas a essas mensagens. Por exemplo, se V recebe um aviso de emergência de outro veículo W e com base em suas posições relativas determina que V também está em perigo, V envia suas próprias mensagens de aviso.

A Figura 5 mostra a divisão dos canais que serão utilzados pela *Federal Communications Commission* (FCC) nos EUA. Outras regiões podem ter outros planos para esses canais. O espectro será de 5.850 GHz à 5.925GHz. Dividido em 7 canais de 10 MHz, numerados 172, 174, 176, 178, 180, 182 e 184.

Canais 172 e 184 são designados para aplicações de segurança pública que envolvem a segurança de vidas e bens. Especificamente, por FCC, o canal 172 é para comunicações de segurança do veículo-a-veículo para evitar e mitigação de acidentes e aplicações de segurança de vidas e propriedades; Canal 184 será de alta potência, comunicações de longa distância a ser utilizado para aplicações de segurança pública que envolvem a segurança de vidas e propriedades, incluindo colisão interseção da estrada.

Figura 5 - Divisão de Espectro utilizado pela FCC



Fonte: (IEEE Draft Guide for WAVE - Architecture, 2013)

Na seção 2.5.3 serão abordados os possíveis ataques lançados contra as redes veiculares por adversários tanto internos quanto externos.

2.6.3 – Ataques contra VANETs

Como não é possível prever todos os tipos de ataques que uma rede VANET estará sujeita no futuro, essa seção apresenta algumas considerações sobre um possível adversário.

Tais considerações ajudarão a nortear possíveis medidas a serem tomadas para, senão eliminar o risco por completo, conter alguns dos ataques mais prováveis de serem lançados por um adversário sobre uma rede VANET.

2.6.4 – Atacante

Os atacantes podem ter perfis variados. Estes perfis podem ser divididos em três dimensões (RAYA e HUBAUX, 2005):

- **Inimigo interno vs. externo:** um inimigo interno é um veículo autenticado na rede que pode se comunicar com os outros veículos. O inimigo externo (*outsider)* é considerado um intruso na rede, de forma que os possíveis ataques por ele executados são limitados;
- **Inimigo malicioso vs. racional:** um inimigo malicioso é aquele que não visa obter benefícios próprios a partir de seus ataques. Seu único objetivo é

prejudicar os veículos ou a funcionalidade da rede. Por outro lado, o inimigo racional visa algum ganho com suas atitudes de forma que seus ataques são mais previsíveis tanto em termos de ações quanto em termos de objetivos;

- **Inimigo ativo vs. passivo:** um inimigo ativo é aquele que injeta dados na rede, enquanto que o passivo atua apenas obtendo informações do canal sem fio, para análises posteriores.

2.7 – Ferramentas para segurança

Há atualmente um número de métodos e técnicas conhecidas para evitar os ataques mencionados na seção anterior. Nessa seção são apresentadas abordagens para implementação de segurança em redes VANET.

2.7.1 – Criptografia

Criptografia é uma ciência que usa a matemática para cifrar (encriptar) e decifrar (decriptar) informação (SCHNEIER, 1996) (STALLINGS, 2008). A encriptação permite guardar informação importante de uma forma segura e transmiti-la através de redes de comunicação, como a Internet, sem que possa ser lida por outrem a não ser pelo destinatário pretendido.

Criptoanálise é a ciência que analisa a informação segura. A criptoanálise clássica envolve combinações analíticas, ferramentas matemáticas, paciência, determinação e sorte. O poder da criptografia é medido pelo tempo e pelos recursos que devem ser gastos para repor o texto original, podendo ser classificada grosseiramente em criptografia fraca e criptografia forte. O resultado da criptografia fraca é texto cifrado, porém este texto pode ser facilmente reescrito na sua forma original.

O resultado da criptografia forte é o texto cifrado, difícil de quebrar sem o uso da ferramenta adequada. Na maioria das vezes, mesmo com a ferramenta adequada, o tempo para recompor o texto original fica na ordem de milhões de anos. Um algoritmo criptográfico é uma função matemática usada no processo de encriptação e decriptação. Um algoritmo criptográfico trabalha com uma chave (um número, uma palavra ou uma frase) para criptografar texto puro.

Para desenvolver algoritmos de criptografia são aplicados métodos matemáticos, geralmente complexos, que consequentemente efetuam determinadas transformações nos dados da mensagem.

> Um algoritmo é definido na criptografia como uma transformação matemática. Logo, o algoritmo converte uma mensagem clara em uma mensagem cifrada e vice-versa. Quando cifra uma mensagem, a origem utiliza um algoritmo de ciframento para transformar o conteúdo claro da mensagem em texto cifrado. Quando decifra uma mensagem, o destinatário utiliza o algoritmo de deciframento correspondente para converter o texto cifrado de novo em uma mensagem clara. (Gomes, 2004. p. 210).

De acordo com (GOMES e RIBEIRO, 2004) o mais inteligente não é tentar desenvolver um algoritmo novo com o intuito de aumentar a segurança, mas sim utilizar os algoritmos públicos mais conhecidos, atribuindo maior importância na escolha da chave que será utilizada, sabendo-se que o segredo da criptografia está neste valor e não no algoritmo.

O mesmo texto puro pode ser criptografado com resultados diferentes usando chaves diferentes. A segurança do texto criptografado depende da força do algoritmo criptográfico e do secretismo da chave. Um algoritmo criptográfico com todas as possíveis chaves e todos os protocolos que o fazem funcionar compõe um criptossistema.

2.7.2 – Assinaturas Digitais

Em Criptografia, a assinatura digital é um método de autenticação de informação digital tipicamente tratada como análoga à assinatura física em papel (STALLINGS, 2008). Embora existam analogias, existem diferenças importantes. A utilização da assinatura digital providencia a prova inegável de que uma mensagem veio do emissor. Para verificar este requisito, uma assinatura digital deve ter as seguintes propriedades:

- **Autenticidade:** o receptor deve poder confirmar que a assinatura foi feita pelo emissor;
- **Integridade:** qualquer alteração da mensagem faz com que a assinatura não corresponda mais ao documento;
- **Não Repúdio:** ou irretratabilidade, significando que o emissor não pode negar a autenticidade da mensagem.

Existem diversos métodos para assinar digitalmente documentos, e esses métodos estão em constante evolução. Porém, de maneira resumida, uma assinatura típica envolve dois processos criptográficos: o *hash* (resumo) e a encriptação deste *hash*.

Em um primeiro momento é gerado um resumo criptográfico da mensagem por meio de algoritmos complexos (MD5, SHA-1, SHA-256 e SHA-512 são exemplos) que reduzem qualquer mensagem sempre a um resumo de mesmo tamanho (128bits, 160bits, 256bits e 512bits respectivamente). A este resumo criptográfico dá-se o nome de *hash*.

Uma função de *hash* deve apresentar necessariamente as seguintes características: deve ser impossível encontrar a mensagem original a partir do *hash* da mensagem; O *hash* deve parecer aleatório, mesmo que o algoritmo seja conhecido. Uma função de *hash* é dita forte se a mudança de um *bit* na mensagem original resulta em um novo *hash* totalmente diferente. Deve ser impossível encontrar duas mensagens diferentes que levam a um mesmo *hash*.

Porém, como as mensagens possíveis são infinitas, mas o tamanho do *hash* é fixo, é impossível impedir que mensagens diferentes levem a um mesmo *hash*. Quando se encontram mensagens diferentes com *hashes* iguais é dito que foi encontrada uma colisão de *hashes*. Um algoritmo onde isso for obtido deve ser abandonado.

As funções de *hash* estão em constante evolução para evitar que colisões sejam obtidas. Cabe destacar que a colisão mais simples de encontrar é uma aleatória, ou seja, obter colisões com duas mensagens geradas aleatoriamente, sem significado real. Quando isto ocorre os estudiosos de criptografia já ficam atentos.

Porém para comprometer de maneira imediata a assinatura digital seria necessário obter uma mensagem adulterada que tenha o mesmo *hash* de uma mensagem original fixa, o que é computacionalmente impossível de ocorrer com os algoritmos (e o poder computacional) existentes hoje. Desta forma, garante-se a integridade da assinatura digital.

Após gerar o *hash*, ele deve ser criptografado por meio de um sistema de chave pública, para garantir a autenticação e a irretratabilidade. O autor da mensagem deve usar sua chave privada para assinar a mensagem e armazenar o *hash* criptografado junto à mensagem original.

Para verificar a autenticidade do documento, deve ser gerado um novo resumo a partir da mensagem que está armazenada, e este novo resumo deve ser comparado com a assinatura digital. Para isso, é necessário decifrar a assinatura obtendo o *hash* original. Se ele for igual ao *hash* recém gerado, a mensagem está íntegra. Além da assinatura, existe o selo cronológico que atesta a referência de tempo à assinatura.

2.7.3 – Códigos de Autenticação de Mensagens

Segundo (STALLINGS, 2008), uma técnica de autenticação alternativa envolve o uso de uma chave secreta para gerar um bloco de dados de tamanho fixo conhecido como Código de Autenticação de Mensagem (MAC - *Message Authentication Code*). Esta técnica assume que ambos os lados envolvidos na comunicação compartilham uma mesma chave. Uma função que gera um código desse tipo é similar a uma função criptográfica, exceto que o algoritmo não precisa ser reversível.

O MAC é incluído na mensagem na origem para que o destinatário possa verificar a autenticidade da mensagem recebida. O destinatário calcula o MAC da mensagem recebida. Caso esse MAC seja igual ao MAC que estava incluído na mensagem, então:

- O receptor tem garantias de que a mensagem não foi alterada. Se um atacante alterar a mensagem, mas não alterar o MAC, então o cálculo do MAC pelo receptor será diferente do MAC recebido. Como se considera que o atacante não conhece a chave secreta, ele não poderá alterar o MAC para corresponder às alterações na mensagem.
- O receptor tem garantias de que a mensagem é do emissor declarado. Como ninguém mais conhece a chave secreta, ninguém mais poderia preparar uma mensagem com um MAC apropriado.
- Se a mensagem inclui um número de sequência, então o receptor pode ter certeza da sequência apropriada, pois um adversário não poderá alterar o número de sequência.

Existem duas formas de se implementar um MAC: utilizando-se uma cifra de blocos (CMAC - *Cipher-based* MAC) ou utilizando-se uma função de *hash* (*keyed-**H**ash **M**essage **A**uthentication **C**ode (HMAC)*). A descrição detalhada de cada uma delas pode ser encontrada em (STALLINGS, 2008).

2.8 – Reputação

Sistemas distribuídos nos quais não existem uma coordenação centralizada, como redes *Peer-to-Peer* (P2P), MANETs e VANETs, estão sujeitos a diversos tipos de inimigos e

ataques. Algumas técnicas simples, como criptografia, podem neutralizar certas ameaças externas, pois os inimigos não são capazes de acessar os dados trafegados na rede. Porém, tem-se o risco de um membro autenticado aproveitar-se desta situação para atacar outros integrantes da rede. Desta maneira faz-se necessário o uso de mecanismos de defesas mais efetivos, capazes de abater estas deficiências. Uma proposta que tem sido abordada para tentar garantir o cooperativismo entre os nós da rede é a utilização de sistemas de reputação.

O objetivo do sistema de reputação é construir valor de confiança para cada nó dessa rede, e com base nesses valores, os outros nós decidem em quem confiar, assim, incentivar comportamento confiável. Resnick e Zeckhauser (2002) listam três metas para sistemas de reputação:

- Fornecer informações para distinguir entre um par confiável e um par não confiável;
- Incentivar pares a agir de uma maneira confiável;
- Para desencorajar pares suspeitos de participar do serviço, o mecanismo de reputação é apresentado.

O conceito de reputação pode ser definido como uma medida coletiva de confiabilidade em uma pessoa ou coisa baseada em indicações ou avaliações de membros de uma comunidade. Assim, o nível individual de confiança em tal pessoa ou coisa pode ser obtido a partir de uma combinação das indicações recebidas e das experiências pessoais (SWAMYNATHAN et al, 2008). Em VANETs, pode considerar-se a reputação de um veículo como um conjunto de opiniões mantidas por outros veículos sobre este. Essa reputação é então utilizada como critério importante na tomada de decisões, tais como encaminhar ou descartar pacotes enviados por esse veículo, considerá-lo ou desconsiderá-lo como opção no roteamento de dados, considerar ou desconsiderar informações por ele repassadas, entre outros.

Nos últimos anos, o sistema baseado em reputação tem sido abundantemente estudado (LO e TSAI, 2009), (DHURANDHER et al.,2010), (HUANG, 2011), (Li et al.,2012), (CHEN et al., 2013). Ele fornece informações para a confiança computacional como previsões sobre o comportamento futuro com base em ações passadas do nó. Normalmente, um intervalo é introduzido para implicar o grau de confiabilidade, em que o limite superior representa que o nó é de confiança, enquanto o limite inferior significa que os pares não são confiáveis.

2.9 – Considerações Finais

A VANET tem apresentado ser uma possível conexão dos veículos inteligentes, buscando garantir a segurança dos condutores e consequentemente dos pedestres, assim como também busca um melhor fluxo do tráfego dos veículos por meio de informações do trânsito em tempo real.

Da mesma maneira que qualquer rede de computador, a VANET também é alvo de ameaças. Portanto, diversas pesquisas estão direcionadas para lidar com esta questão apresentando diversas técnicas, entre estas encontram-se os sistemas de reputação.

Diversos trabalhos buscam solução para o sistema de reputação para VANET, uma vez que as soluções para MANET não são suficientes para tratar a mobilidade das VANETs.

Mesmos as soluções direcionadas para VANET não são totalmente funcionais e ainda não há uma solução perfeita. Todas as soluções propostas são distribuídas, ou seja, a avaliação de um veículo é feita em um determinado momento e cada veículo possui distribuidamente essas informações. Nesta abordagem, um veículo que busca informação da reputação de outro veículo, a avaliação é feita naquele momento e baseada no comportamento do veículo naquele momento. Os problemas nesta abordagem são o tempo de armazenamento sobre um veículo e a troca de localização do veículo. O problema relacionado com tempo de armazenamento está em saber quando um veículo dever se desfazer da informação sobre um veículo e o em relação à troca de localização é que caso o veículo vá para outra região ele não leva sua reputação com ele e a mesma deveria ser reavaliada. Sendo assim novas abordagens são necessárias e é o que este trabalho propõe, um sistema de reputação centralizado, descrito no Capítulo 4.

CAPÍTULO 3 – SIMULADORES

Para realizar avaliações de sistemas de computação geralmente são utilizadas uma das seguintes técnicas: medição, análise ou simulação (IZQUIERDO e REEVES, 1999). A medição é geralmente utilizada quando existe a possibilidade de obtenção de dados reais do sistema estudado. Por outro lado, a técnica de análise se baseia na modelagem do sistema e seus possíveis comportamentos a partir de modelos matemáticos imensamente simplificados. Por último, a simulação é baseada na construção e execução de processos, com o objetivo de reproduzir o comportamento do sistema.

Neste trabalho, foi escolhida a simulação como técnica de avaliação do Sistema de Reputação. Deve-se a esta escolha o fato de que, comparada com a técnica de análise, a simulação permite um maior detalhamento da rede que o uso de modelos matemáticos. O desenvolvimento de um protótipo, o que permitiria o uso de medições, apresenta um custo elevado, gerado principalmente pela compra, instalação e manutenção dos dispositivos em veículos reais, o que inviabilizaria o projeto.

Além disso, a simulação permite a repetição das análises e dos resultados, o que nem sempre é possível em um protótipo. Todavia, uma vez que a simulação não considera todos os aspectos da rede, mas apenas os mais importantes, seus resultados devem ser cuidadosamente interpretados.

3.1 – Simuladores de rede

Partindo do pressuposto que redes veiculares envolvem somente comunicação sem fio, todos os simuladores apresentados neste capítulo suportam simulação com nós móveis.

Para simular uma rede veicular, duas aplicações diferentes são necessárias, um simulador de rede e um de mobilidade. Um simulador de mobilidade é geralmente utilizado para produzir as rotas de movimentação dos nós que são enviadas para o simulador de rede. Do outro lado, o simulador de rede controla a comunicação entre os nós móveis.

3.1.1 – NS-2

Em 1989, o NS-2 surgiu como um simulador de rede que oferecia simulação de transporte, roteamento e *multcast* sobre redes cabeadas e sem fio. O NS-2 foi desenvolvido pelo Instituto de Ciência da Informação da Universidade da Califórnia do Sul (NS-2,2011).

O núcleo do NS-2 é escrito em C++, porém os usuários interagem com o NS-2 por meio da escrita de *script* em *TCL (Tool Command Language).* Estes *scripts* devem conter todos os comandos necessários para especificar a simulação (por exemplo, configuração da topologia, parâmetros de especificações sem fio, entre outros).

Há implementações de alguns modelos de mobilidade disponíveis para NS-2, incluído *Random Trip Mobility* (The Random Trip Mobility ,2011).

Nas simulações de redes sem fio utilizando o NS-2 os nós precisam ser programados manualmente para detectar e transmitir dados entre si. Não há recurso implementado disponível para detectar os nós ao redor

Contudo, algumas características fazem do NS-2 o simulador de rede amplamente utilizado no mundo: é gratuito, código aberto, muitos componentes e protocolos de rede padrões são disponíveis e há uma boa documentação.

3.1.2 – GloMoSim e QualNet

O GloMoSim (*Global Mobile Information System Simulator*) (ZENG, BAGRODIA e GERLA, 1998) é um simulador de rede que foi desenvolvido no Laboratório de Computação Paralela da Universidade da Califórnia em Los Angeles (UCLA), em 1998. Este foi principalmente projetado para simular redes sem fio e suportar os seguintes modelos móveis: *Random Waypoint, Random Drunken,* e *Trace-based*. No princípio o GloMoSim não tinha uma Interface Gráfica (GUI), mas agora contém uma interface baseada em Java.

O simulador foi escrito em PARSEC (BAGRODIA et al,1998) e todos os novos protocolos devem ser definidos em PARSEC também. PARSEC é uma linguagem de simulação baseada em C desenvolvida pela UCLA, para execução sequencial e paralela de modelos de simulações de evento discretos. Esta também pode ser usada como uma linguagem de programação paralela.

Esse simulador é distribuído com bibliotecas de diversos modelos móveis incluindo *Random Drunken* (para a construção de novos modelos de mobilidade em que o nó escolhe sua direção a partir de quatro opções para escolher o caminho aleatoriamente) e baseados em Rotas.

GloMoSim foi projetado para suportar milhões de nós em uma única simulação usando técnicas de paralelismo. Agregação de nó tem sido sempre o gargalo na maioria destas simulações, mas este simulador foi o primeiro a implementá-las com sucesso. Também tem a capacidade para suportar a agregação de camada.

O GloMoSim versão 2.0 foi lançado em 2000, após isso, PARSEC parou de trabalhar com software livre e lançou uma versão comercial do GloMoSim chamada QualNet.

O QualNet inclui uma interface gráfica para configuração e execução das simulações, e oferece um amplo conjunto de modelos de camada física e MAC sem fio. QualNet está disponível para plataformas Windows e Linux.

3.1.3 - OPNET

Este é um simulador de rede comercial utilizado para comunicações de redes cabeadas e sem fio (OPNET, 2013). Suporta uma ampla variedade de tecnologias sem fio como MANETs, IEEE 802.11 WLANs, WiMAX, *Bluetooth* e redes de satélite.

Este simulador fornece um editor de interface gráfica para construir modelos para diversas entidades da rede desde modulador de camada física até processos de aplicação e inclui bibliotecas e pacotes gráficos para a apresentação dos cenários e resultados das simulações.

Existem três fases básicas do processo de implantação OPNET. Primeiro, escolher e configurar modelos de nó para usar em simulações, como um nó sem fio, trajetórias, e assim por diante. Em segundo lugar, construir e organizar a rede por meio da criação de conexões para diferentes entidades. Em terceiro lugar, selecionar as estatísticas desejadas (local ou global) para coletar durante a simulação.

O arquivo de cenário descreve os parâmetros de rede, parâmetros de mobilidade e parâmetros globais (por exemplo, o tempo de simulação). Um conversor de arquivo de configuração recupera elementos específicos da rede e os arquivos de configuração do nó e reordena esses elementos em um padrão (*default*) formato de configuração. O arquivo de configuração de rede especifica todos os parâmetros relacionados à rede, por exemplo, nós, roteadores, aplicações, protocolos, rádios e assim por diante. O arquivo de configuração do nó inclui as identidades dos nós (tais como endereços IP - *Internet Protocol*). Os parâmetros globais definem o tempo de simulação, sistema de coordenadas, a semente de números

aleatórios, a colocação do nó baseado em arquivos, pilha de protocolos, filtro de estatística, e assim por diante. Além da saída GUI – *Graphical User Interface*, a simulação produz um ficheiro de estatísticas, que contém todas as estatísticas geradas a partir da execução da simulação.

3.1.4 - OMNeT++

OMNeT++ é um ambiente de simulação de código aberto. A principal simulação é voltada para aplicações de simulações de Internet, mobilidade e simulações *ad hoc*. Este simulador tem um *design* baseado em componentes, o que significa que novos recursos e protocolos podem ser suportados por meio de módulos. OMNeT++ suporta modelos de rede e mobilidade por meio de desenvolvimento independente dos módulos *Mobility Framework* e *Framework INET* (PONGOR, 1993) (OMNet++,2013).

Simulações no OMNeT++ consistem em módulos simples que implementam o comportamento atômico de um modelo, por exemplo, um protocolo particular. Vários módulos simples podem ser ligados em conjunto e formam um módulo composto. Esta ligação e a configuração da rede da simulação ocorrem no NED (*Network Description Language*), linguagem de descrição de rede do Omnet++.

Além disso, o NED suporta a especificação de parâmetros variáveis na descrição da rede: por exemplo, o número de nós em uma rede pode ser marcado para ser dinâmico e, mais tarde, ser configurado em tempo de execução. O Projeto de simulação em OMNeT++ é baseado em GUI, e os dados de saída podem ser apresentados também por meio de GUI. Há uma versão comercial do OMNeT++ chamada OMNEST.

3.1.5 - J-Sim

J-Sim é um ambiente de simulação de código aberto, desenvolvido totalmente em Java. Este simulador oferece dois modelos de mobilidade: baseado em trajetória e percurso randômico. J-Sim é apresentado como uma alternativa para NS-2, uma vez que foi concebido para ser mais fácil de utilizar (J-SIM, 2013).

Em J-Sim, as aplicações são construídas como um conjunto de componentes que podem ser concebidos e testados separadamente. J-Sim pode ter como entrada um arquivo *TCL* como entrada, semelhante ao NS-2, mas com um formato diferente. Como NS-2, este simulador produz um arquivo de rastreamento de eventos e um arquivo de animação.

J-Sim também fornece uma interface de *script* para permitir a integração com diferentes linguagens de script como Perl, Tcl ou Python.

3.1.6 - SWANS

SWANS (*Scalable Wireless Ad hoc Network Simulator*) (SWANS, 2013) foi desenvolvido para ser uma alternativa escalável para NS-2 para simulação de redes sem fio. SWANS é organizado como componentes de software independentes que podem ser compostas para formar configurações completas de rede wireless ou rede de sensores.

Foi criado principalmente porque as ferramentas de simulação de rede existentes não eram suficientes para as necessidades de pesquisa, e seu desempenho serve como uma validação da abordagem baseada em máquina virtual para a construção do simulador.

A entrada para SWANS é um arquivo Java que cria os nós e especifica como esses nós devem se mover (o cenário de movimento do nó) e como eles devem se comunicar (o cenário da comunicação). O usuário pode selecionar qualquer uma das aplicações prontas em SWANS e associá-lo com qualquer nó (s) para executá-lo na camada de aplicação nó. Além disso, SWANS dá ao usuário a flexibilidade para construir um aplicativo personalizado e executá-lo na camada de aplicação de qualquer nó.

3.2 - Simuladores de mobilidade

3.2.1 - VisSim

VisSim (VisSim, 2013) é um simulador comercial desenvolvido pela PTV - Planung Transport Verkehr AG em Karlsruhe, na Alemanha, que permite aos usuários simular grandes e complexos cenários de tráfego, incluindo muitos objetos diferentes, como carros, caminhões, pedestres, ciclistas, entre outros.

O VisSim é um *software* de simulação do fluxo de tráfego microscópico multimodal, o que significa que cada entidade da realidade que deve ser simulada o é individualmente, ou seja, é representado por uma entidade correspondente na simulação, considerando-se, assim, todas as propriedades relevantes. O mesmo vale para as interações entre as entidades. O oposto poderia ser uma simulação macroscópica, em que a descrição da realidade é transferida dos indivíduos para variáveis médias como o fluxo e densidade.

Também é possível realizar e gerar animações 3D utilizando uma interface gráfica (GUI). VisSim só está disponível para as plataformas Microsoft Windows.

3.2.2 - PARAMICS

PARAMICS é um conjunto de *softwares* desenvolvido pela Quadstone Ltda, que adota o paradigma microssimulação e funciona apenas em plataformas baseadas em Microsoft Windows (CAMERON e DUNCAN, 1996) (PARAMICS, 2013).

PARAMICS é possível simular cruzamentos, estradas, zonas urbanas e, como VisSim, ele inclui uma ferramenta de visualização 3D.

PARAMICS é composto de vários módulos: modelador, analisador, processador, estimador, desenhista, conversor, programador e monitor. É uma ferramenta projetada para engenheiros de tráfego e planejadores urbanos. Pode efetuar simulações de transportes urbanos e as interações com os pedestres.

3.2.3 - SUMO e MOVE

"*Simulation of Urban MObility*" é um *software* de código aberto, um pacote de simulação de tráfego de estrada microscópico, altamente portável, projetado para lidar com redes rodoviárias de grande porte (BEHRISCH et. al, 2011) (SUMO, 2013). Foi desenvolvido principalmente por funcionários do Instituto de Sistemas de Transporte do Centro Aeroespacial Alemão. SUMO está licenciado sob a GPL.

Esse permite simular como uma determinada demanda de tráfego, que consiste de veículos individuais, se move por meio de uma determinada rede rodoviária. A simulação permite abordar um grande conjunto de temas de gestão de tráfego. É puramente microscópico: cada veículo é modelado explicitamente, tem o seu próprio trajeto, e move-se individualmente, através da rede.

Com o SUMO cerca de 100.000 a 200.000 veículos podem ser simulados em tempo real em um PC *desktop*, incluindo:

- Movimentação dos veículos sem colisões;
- Diferentes tipos de veículos;
- Estradas multi-faixas e troca de faixas e

- Diferentes regras de direito de passagem, semáforos.

MOVE é uma extensão para SUMO que adiciona uma interface gráfica para descrever mapas e definir a movimentação do veículo e permite ao usuário importar mapas Google Earth. MOVE também inclui uma ferramenta de visualização que permite aos usuários ver a rota de mobilidade gerada (KARNADI, MO e LAN, 2007).

Basicamente, o MOVE é composto por dois componentes: o editor de mapa de estrada e o editor de movimento do veículo. O primeiro serve para manualmente e/ou aleatoriamente gerar um mapa de estrada, seja de arquivos TIGER/line ou Google Earth, enquanto o segundo permite especificar as propriedades de cada veículo, como a velocidade máxima, a aceleração, a probabilidade de virar em um cruzamento, o caminho a tomar, etc.

As informações coletadas pelos dois editores são enviadas para o compilador SUMO, em seguida, um arquivo de rota no formato ns-2 ou Qualnet é gerado. Os resultados mostram que o MOVE provoca baixa taxa de recepção.

3.3 – Simuladores fortemente integrados

A simulação de aplicações VANET exige não apenas a simulação da comunicação sem fio entre os veículos, mas também exige a simulação da mobilidade dos veículos. Infelizmente, esses dois aspectos da simulação VANET têm sido muitas vezes dissociados.

O problema é como mesclar os dois tipos de simuladores (simulador de rede e simulador de mobilidade). Um método simples para alcançar esta fusão e criar uma simulação totalmente integrada é a implementação de modelos de mobilidade em um simulador de rede, mas sem permitir que as mensagens de rede retornem para o modelo de mobilidade. Esse tipo de simulação é chamado de comunicação *one-way* (direção única) (a partir de modelo de mobilidade para a rede). Estes tipos de simuladores são adequados para simular aplicações VANET relacionadas à informação e entretenimento, incluindo conectividade com a Internet, aplicações multimídia e aplicações *peer-to-peer*, onde a comunicação não afeta o movimento dos veículos.

Em contraste, os simuladores totalmente integrados que oferecem comunicação bidirecional geralmente consistem de dois sub-simuladores (rede e mobilidade) que podem se comunicar um com o outro. Estes simuladores são mais adequados para aplicações de segurança e aplicações de informação de tráfego que assumem que o *feedback* da rede vai afetar os movimentos dos veículos.

Nestes tipos de aplicações, o simulador de tráfego alimenta o simulador de rede com a informação da posição, velocidade, aceleração, a direção, e assim por diante. A aplicação VANET que é executada no nível superior do simulador de rede incorpora essa informação com informações sobre os veículos, a fim de notificar o motorista do próximo congestionamento ou uma possível colisão. Com base nessa notificação, decisões na direção (ou seja, a mobilidade do veículo) podem ser afetadas. Por exemplo, em um sistema de notificação de congestionamento, o motorista pode optar por mudar de faixa ou tomar um caminho diferente. Estas decisões têm de se propagar de volta para o simulador de mobilidade a ser refletido nas informações mobilidade veículo.

Esta seção descreve as características de alguns simuladores fortemente integrados. E a seguir um comparativo entre eles e o simulador escolhido para realizar os testes neste trabalho.

3.3.1 - SWANS++

SWANS++ (SWANS++, 2013) estende o simulador de rede SWANS, adicionando uma GUI para visualizar o cenário e um modelo de mobilidade, STRAW (*STreet RAndom Waypoint*) (CHOFFNES e BUSTAMANTE, 2005), para o movimento de veículos em cenários de rua. STRAW usa o modelo de mobilidade ponto de passagem aleatória simples, mas restringe o movimento de veículos aos limites de rua reais, carregadas de arquivos de dados TIGER/Line. Arquivos de dados TIGER/Line contêm características, tais como estradas, ferrovias, rios, bem como as áreas geográficas legais e estatísticas.

STRAW consiste principalmente de três componentes: a mobilidade intra-segmento, mobilidade intersegmento e gerenciamento e execução de rota. Em mobilidade intersegmento, os veículos movem-se de acordo com um modelo de carro seguinte e mudam sua velocidade apenas em determinadas situações. Para a mobilidade intersegmentos, de acordo com o projeto do sistema, há um sinal de controle de tráfego ou um sinal de parada em cada interseção que obriga o veículo a alterar a sua velocidade. O modelo de mobilidade implementado em STRAW (e, portanto, SWANS++) não suporta a mudança de faixa. O módulo de gestão e execução da rota (RME - ***R**oute **M**anagement and **E**xecution*) é responsável por determinar as rotas de veículos durante a simulação.

SWANS + + é um simulador totalmente integrado, mas não fornece *feedback* entre a mobilidade e os módulos de rede.

3.3.2 - GrooveNet

GrooveNet (originalmente conhecido como GrooveSim) é um simulador de rede e mobilidade integrada que permite a comunicação entre os veículos reais e simulados. Originalmente, GrooveNet estendeu o simulador de código aberto Roadnav pela adição de um modelo de rede e uma interface gráfica, que pode carregar mapas de ruas reais do banco de dados TIGER/Line, a fim de simular a mobilidade de veículos em estradas reais. GrooveNet inclui mobilidade fixa, velocidade de rua, velocidade uniforme, e modelos de mobilidade do carrros-seguintes. Este simulador suporta vários modos de operação, incluindo o modo de carro, modo simulação, o modo de reprodução, o modo de simulação híbrida e modo de geração de teste (MANGHARAM et al., 2005)) (MANGHARAM et al., 2006).

A capacidade única do GrooveNet de integrar veículos simulados com veículos reais lhe permite funcionar como um software fase de teste, bem como um simulador.

3.3.3 - TraNS

TraNS[1] (*Traffic and Network Simulation Environment*), que foi apresentado na MobiCom de 2007, pode ser considerado como o primeiro simulador VANET puro. Foi o primeiro trabalho a combinar um simulador de rede NS-2, com um simulador de mobilidade veicular, SUMO, e fornecer *feedback* do simulador de rede para o simulador de mobilidade (PIORKOWSKI, 2007).

TraNS pode operar em dois modos: modo centrado na rede e modo centrado na aplicação. No modo centrado na rede, não há nenhuma resposta fornecida a partir de NS-2 para o SUMO, de modo que o arquivo da rota de mobilidade dos veículos pode ser pré-gerado e enviado para o simulador de rede posteriormente. A conexão entre os dois simuladores, neste caso, é feita por meio de um *parser* que analisa o arquivo de rastreamento de mobilidade gerada pelo SUMO e converte para um formato adequado para NS-2. No modo de centrado na aplicação, o *feedback* entre o NS-2 e SUMO é fornecida por meio de uma interface chamada TraCI (*Traffic Control Interface*). Neste modo, os dois simuladores (SUMO e NS-2) deve ser executado simultaneamente. A TraCI realiza a ligação entre o NS-2 e SUMO por conversão dos comandos de mobilidade provenientes de NS-2 com uma sequência de comandos de

[1] http://trans.ep.ch/

primitivas de mobilidade, tais como a paragem, mudança de faixa, a velocidade de mudança, e assim por diante, que podem ser enviados para o SUMO. Como ambos os simuladores estão funcionando separadamente, ao mesmo tempo, a comunicação em dois sentidos no modo de aplicação centrada utiliza duas filas de eventos separados.

3.3.4 - Veins

Veins (*Vehicles in Network Simulation*) é um outro simulador que une um simulador de mobilidade com um simulador de rede (VEINS, 2014). Em Veins, SUMO está emparelhado com OMNeT++ por meio de uma extensão SUMO para permitir que ele se comunique com OMNeT++ por meio de uma conexão TCP.

A simulação do tráfego rodoviário é realizada por SUMO, que é bem estabelecida no domínio da engenharia de tráfego. A simulação de rede é realizada por OMNeT++ junto com o kit de ferramentas de modelagem de camada física Mixim, o que torna possível empregar modelos precisos para a interferência de rádio, bem como sombreamento por obstáculos estáticos e em movimento.

Ambos os simuladores são bi-direcionalmente acoplados e as simulações são realizadas *online*. Deste modo, a influência do tráfego em redes veiculares estrada pode ser modelada e as interações complexas entre os dois domínios examinados.

Para realizar avaliações da Comunicação Inter-Veicular (CIV), os dois simuladores são executados em paralelo, ligados por meio de um *socket* TCP. O protocolo para esta comunicação foi padronizado como a *Traffic Control Interface* (TraCI). Isso permite a simulação bidirecionalmente acoplada do tráfego rodoviário e do tráfego de rede. Movimento de veículos no tráfego rodoviário no simulador SUMO é refletido no movimento de nós na simulação do OMNeT++. Os nós podem então interagir com a execução de simulação de trânsito, por exemplo, para simular a influência da CIV no tráfego rodoviário.

3.3.5 - ASH

ASH (*Application-aware SWANS with Highway mobility*) (IBRAHIM e WEIGLE, 2008) é uma extensão do simulador de rede sem fio SWANS. ASH suporta *feedback* entre a

sub-simulador de mobilidade veicular e de um sub-simulador de rede, tornando-se um dos simuladores de comunicação nos dois sentidos com uma única fila de eventos. ASH permite aos usuários projetar um segmento de rodovia simples e personalizá-lo, especificando as direções (unidirecional ou bidirecional), o número de vias, o número de entradas e saídas e suas localizações correspondentes ao longo do segmento.

Além de adicionar modelos de mobilidade de estrada para SWANS, ASH amplia os tipos de nós disponíveis:

- Métodos para fornecer *feedback* entre a camada de aplicação e o modelo de mobilidade, permitindo a comunicação bidirecional necessária;
- Uma topologia de rodovia personalizável, suportando múltiplas pistas, entradas e saídas;
- Uma implementação do modelo de carro-seguinte da *Intelligent Driver Model* (IDM) (TREIBER, 2000);
- Uma implementação do modelo de mudança de faixa MOBIL (KESTING, 2007);
- Uma implementação técnica de *broadcasting Inter-Vehicle Geocast* (IVG) (BACHIR e BENSLIMANE, 2003), bem como uma implementação de um aprimoramento IVG, probabilística do IVG (p-IVG) (IBRAHIM e WEIGLE, 2008);
- Um aprimoramento do modelo de nó permitindo a adição de veículos sem-comunicações, das unidades do lado da estrada (RSU) e dos obstáculos a uma simulação e
- Estatísticas e registro de relatórios das métricas da simulação.

ASH aceita um arquivo de configuração para o cenário da rodovia. A criação de nós e o cenário da comunicação devem ser especificados em um arquivo Java como realizado no SWANS.

3.4 - SIMULADOR NCTUNS

NCTUns (*National Chiao Tung University Network Simulator*) implementa comunicação em duas vias com uma única fila de eventos. NCTUns 1.0 foi desenvolvido apenas como um simulador de rede, mas a versão mais recente, NCTUns 6.0 (Lançado em janeiro de

2010), integra algumas capacidades de simulação de tráfego, como a concepção de mapas e controle de veículos de mobilidade. Além disso, NCTUns inclui uma interface gráfica para ajudar no processo de criação de mapas (WANG et.al, 2007). Nesta seção, será feita uma breve descrição destas características e capacidades.

Por causa de recursos reduzidos, os simuladores tradicionais costumam trabalhar com certas limitações. Isso acontece porque um simulador apresenta sempre uma versão simplificada dos resultados, ao contrário do que se pode obter com um dispositivo real, e simulações executam implementações de protocolo com poucos detalhes para reduzir a complexidade e o custo do desenvolvimento.

Outra desvantagem na maioria dos simuladores é que as aplicações devem ser escritas para usar a API interna (*Application Programming Interface*), do seu simulador, caso contrário, tem de ser recompilado para o simulador, criando um programa grande e complexo.

Para superar essas duas limitações, os autores do NCTUns (*National Chiao Tung University Network Simulator*) (WANG e KUNG, 99), propuseram um novo método de simulação de *kernel* re-entry, o que significa que o *kernel* do Linux foi modificado. Usando este método, é criada uma implementação real da pilha de protocolo que irá fornecer resultados mais realistas. Dessa maneira, este método permitirá aplicações reais a serem executados na rede simulada, e isso significa que NCTUns vai funcionar como um emulador também, função não fornecida por outros simuladores.

O simulador de rede NCTUns tem muitos recursos úteis: utiliza diretamente a pilha TCP/IP do Linux para gerar os resultados da simulação de alta fidelidade, fornece um ambiente GUI altamente integrada e profissional e seu mecanismo de simulação adota uma arquitetura de sistema aberto e é de código aberto.

Além disso, ele simula diversas redes importantes: Internet fixa baseada em Ethernet, IEEE 802.11 (b) como LANs sem fio, redes (sensor) móveis *ad hoc* (MANETs), redes celulares GPRS, redes ópticas (incluindo a comutação de circuitos e redes de comutação de explosão) IEEE 802.11 (b) redes dual rádio em malha sem fio, IEEE 802.11 (e), IEEE 802.16 (d) como redes sem fio WiMAX (incluindo o PMP e modos de malha), DVB-RCS como redes de satélites, as redes veiculares sem fio para Sistemas Inteligentes de Transporte (incluindo V2V e V2I), nós de móveis multi-interface para redes sem fio heterogêneas, IEEE 802.16 (e) redes móveis WiMAX, IEEE 802.11 (p) / 1609 WAVE redes veiculares sem fio, vários modelos realista de canais sem fio, IEEE 802.16 (j) O modo transparente e não-transparentes de redes WiMAX , etc.

A partir da versão 5.0, o simulador passou a fornecer uma implementação completa para o protocolo 802.11(p), padrão para redes *ad hoc* veiculares (VANETs). Há dois modos de operações, pré-especificado e piloto automático. No modo pré-especificado pode-se definir a rota que os veículos irão percorrer no mapa desenhado. No modo piloto automático o processo agente, discutido nas seções seguintes, é responsável por determinar a rota a percorrer. Em ambos os modos, parâmetros como velocidade inicial, aceleração inicial, aceleração máxima, velocidade máxima, entre outros podem ser especificados no momento que se desenha o ambiente da simulação.

No dia 1 de dezembro de 2010, a versão comercial do NCTUns foi anunciada a todos os usuários do simulador. Esta versão chamada de EstiNet. De acordo com seus desenvolvedores, EstiNet é lançada para proporcionar uma melhor qualidade, funcionalidade, desempenho e suporte.

Neste projeto foi utilizado a versão NCTUns 6.0. Nas seções seguintes será descrito a arquitetura e os principais componentes do NCTUns, a especificação da máquina utilizada para a simulação e os passos para instalação e execução do NCTUns 6.0.

Nas seções seguintes deste capítulo serão apresentados os requisitos do sistema, configuração, e arquitetura do simulador NCTUns 6.0.

3.4.1 - Requisitos do Sistema

Para o funcionamento correto do simulador NCTUns, a máquina utilizada neste projeto atende aos requisitos mínimos de software e hardware. Na Tabela 1 são ilustrados os requisitos recomendados para a versão 6.0.

Tabela 1 - Requisitos mínimos do sistema.

Sistema Operacional	Hardware	Software
Fedora 11	Processador 1.6 GHz 256 MB de RAM 300 MB livres no disco rígido	Compilador gcc 4.3. + Permissão de administrador

A partir da versão 2.0 o NCTUns passou a suportar apenas o sistema operacional Linux e não mais o FreBSD. Atualmente a distribuição Linux suportada é a Fedora 11, que utiliza a versão do kernel 2.6.28.9. Outras distribuições Linux, como Debian, também podem ser suportadas, porém são necessárias algumas mudanças de configurações no sistema.

Neste projeto, a máquina utilizada tem a seguinte configuração:

Processador: Intel Celeron @ 1.70 GHz

Memória RAM: 1GB

Espaço livre em disco: 15GB

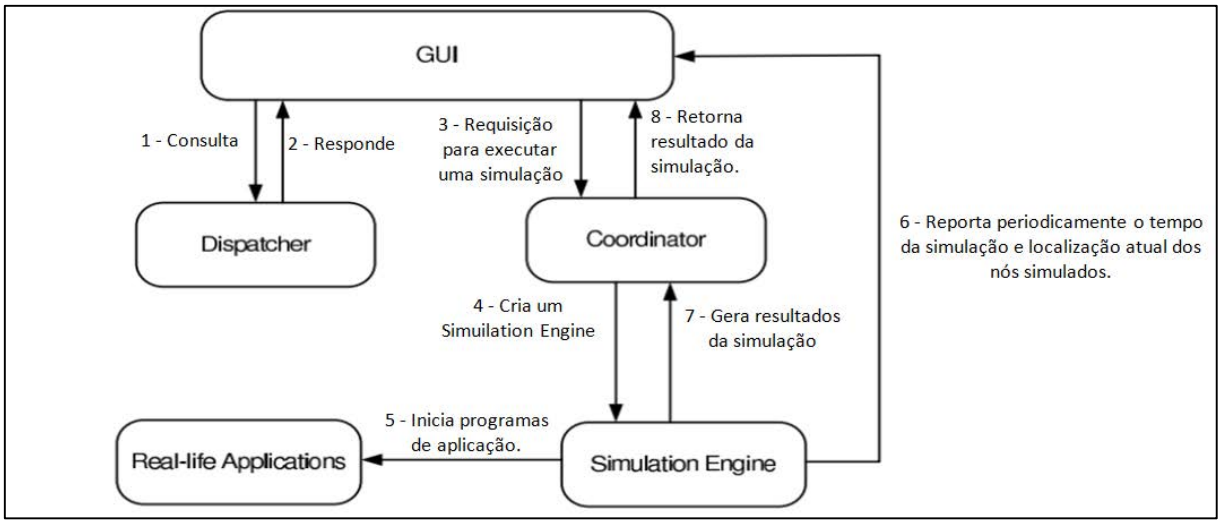
3.4.2 - Arquitetura do Simulados NCTUns

O NCTUns adota uma arquitetura distribuída, que pode ser vista como um bloco de oito componentes: GUI, conjunto de módulos de protocolo, *simulation engine, coordinator, dispatcher,* programas aplicações, *patches* para o *kernel* e *daemons* de nível de usuários. Nesta seção é apresentada a arquitetura e os seis principais componentes do NCTUns, que são:

1. Interface Gráfica do Usuário (GUI)
2. *Dispatcher*
3. *Coordinator*
4. *Simulation Engine*
5. Programas de Aplicações
6. *Patches* para o kernel da pilha de protocolo TCP, UDP e IP.

As interações destes componentes são ilustradas na Figura 6.

Figura 6 - Arquitetura do NCTUns.



Fonte: Adaptado de (WANG, CHOU e LIN, 2010).

As principais funções destes componentes são:

Interface Gráfica do Usuário (GUI): NCTUns fornece um programa GUI, chamado neste pacote de "nctunscliente", que fornece facilidades úteis para os usuários criarem eficientemente casos de emulação e simulação. Necessidades comuns dos usuários para gerarem casos de simulação/emulação são separadas em quatro modos de operação: modo "*Draw Topology"*, modo *"Edit Property"*, modo *"Run Simulation"* e modo *"Play Back"*. Estes modos serão descritos no Anexo B.

Dispatcher: O NCTUns fornece uma arquitetura de simulação flexível, da maneira que o programa GUI e o processo *simulation engine* possam ser executados em máquinas diferentes. No NCTUns, o programa GUI não se responsabiliza por encontrar um servidor de simulação para executar o processo *simulation engine* para a simulação. Instanciado, este envia ao processo *Dispactcher* uma mensagem para saber qual servidor de simulação está disponível atualmente. O processo *Dispatcher* é responsável por monitorar os *status* dos servidores de simulação por este gerenciado e selecionar um servidor de simulação disponível (se este existir) para servir a simulação requisitada pelo programa GUI.

***Coordinator*:** o processo *Coordinator* tem quatro tarefas: processar os comandos vindos do *Dispatcher,* criar um processo *simulation engine* para executar uma simulação, reportar o *status* do processo *simulation engine* criado para o processo *Dispatcher* e coletar os resultados produzidos pelo processo *simulation engine* e enviá-los para o programa GUI. Antes de iniciar qualquer simulação em um servidor de simulação, um processo *Coordinator* deve estar em execução.

***Simulation Engine*:** O processo *simulation engine* é composto de um conjunto de vários módulos de protocolo e um agendador de evento. O primeiro é responsável pelos comportamentos dos protocolos da simulação, enquanto o segundo é responsável por agendar os eventos em uma ordem crescente baseada nos seus *timestamps*. Além disso, durante a simulação o processo *simulation engine* reporta periodicamente o tempo atual da simulação para o programa GUI.

Programa de Aplicação: programas de aplicação são responsáveis por gerarem tráfego de rede na rede simulada. A maioria dos programas de aplicações podem ser executados em um nó simulado pelo NCTUns para gerar tráfegos de redes realistas.

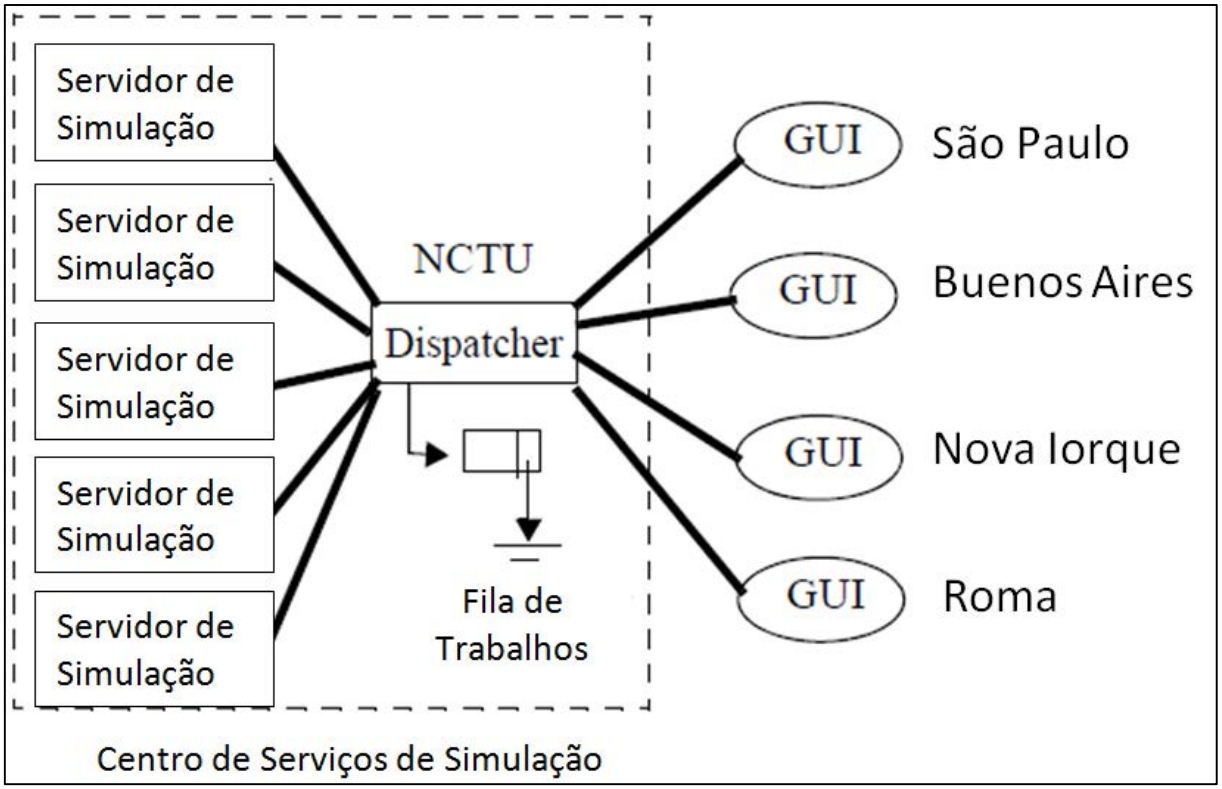
***Kernel Patches*:** O NCTUns utiliza a pilha real de protocolo de rede do Linux para simular os protocolos das camadas de rede e transporte, como TCP, UDP e IP. Pequenas modificações nos temporizadores do *Kernel* do Linux são requeridas para que assim os

temporizadores utilizados pelas pilhas de protocolos no *kernel* de cada nó simulado possam avançar seus tempos baseado no tempo do relógio simulado (controlado pelo NCTUns)

Devido a esta arquitetura distribuída, um usuário remoto pode enviar seu trabalho de simulação para um processo *dispatcher*, e o *dispatcher*, então, encaminhará o trabalho para um servidor de simulação disponível para execução. O servidor irá processar (simular) o trabalho e, posteriormente, retornar os resultados para o programa GUI remoto para análises posteriores. Este esquema pode facilmente suportar o modelo de fazenda de servidores em que vários postos de trabalho de simulação são realizados simultaneamente em diferentes servidores de simulação. Na Figura 7 é ilustrada a arquitetura distribuída dos NCTUns.

Além do modo "multi-máquina", citado anteriormente, outro modo chamado modo de "máquina única" também é suportado. Nesse modo, todos esses componentes são instalados e executados em uma única máquina. Uma vez que a maioria dos usuários tem apenas uma máquina para executar as simulações, este modo pode ser mais apropriado. Sendo assim este é o modo padrão após o pacote NCTUns ser instalado.

Figura 7 - Arquitetura distribuída do NCTUns.



Fonte: Adaptado de (WANG, CHOU e LIN, 2010).

3.4.3 - Integração de Emulação e Simulação

NCTUns pode ser transformado em um emulador . Em uma emulação, nós em uma rede simulada podem trocar pacotes reais com máquinas do mundo real por meio da rede simulada. Isto é, a rede simulada é integrada com a rede física real, de modo que os nós simulados da vida real podem trocar os seus pacotes entre as redes simuladas e reais.

Este recurso é útil para testar as funções e desempenho de um dispositivo da vida real (por exemplo, um telefone VoIP) sob várias condições da rede. Num processo de emulação NCTUns, um dispositivo de vida real externo pode ser um hospedeiro fixo, um hospedeiro celular ou um roteador.

3.5 – Considerações sobre os simuladores VANET

De todos os simuladores citados neste capítulo, o NCTUns foi o escolhido para realizar os testes do projeto. As principais razões para utilizar o NCTUns são: é uma ferramenta gratuita e de código aberto e de capacidades importantes para a pesquisa de rede veicular sem fio. Além disso, o usuário não precisa lidar com a complexidade de codificação, graças ao apoio da GUI fornecida por este simulador. Usando a ferramenta GUI, pode-se desenhar um mapa de estradas facilmente e os veículos podem ser implantados automaticamente.

Os desenvolvedores do NCTUns mostraram interesse em Sistemas Inteligentes de Transporte (ITS) desde a versão 4.0, quando simuladores de tráfego e de rede foram acoplados dentro de um único módulo para fornecer um único ambiente veicular. Com a versão 4.0, foram acrescentadas algumas características importantes para a sua simulação:

- Modelo de comportamento do motorista;
- Construção das rodovias;
- Componentes RSU e OBU para simulação e
- Módulo Agente: Com este comportamento de condução inteligente, o agente pode modelar um carro para obedecer alguns parâmetros, como semáforo, a vizinhança do veículo, troca de faixa, fazer conversões e modelo de carro seguinte.

3.6 Considerações Finais

Este capítulo apresentou os diversos simuladores de redes e de mobilidade encontrados, tanto *open source* como comerciais. Foram apresentadas as características, vantagens e desvantagens de cada um, de modo que foi possível avaliar qual o melhor a ser utilizado para o desenvolvimento do trabalho. O simulador NCTUns foi escolhido por apresentar diversas características que vinham ao encontro das necessidades do projeto, como por exemplo, ser *open source*, permitir não só simulação, mas também emulação de redes, ser desenvolvido com foco já voltado para VANETs, entre outros. A arquitetura, bem como o detalhamento do NCTUns foi apresentado. Detalhes das modificações necessárias para que todos as simulações relacionadas a reputação foram apresentadas.

O próximo capítulo apresenta o sistema de reputação centralizado desenvolvido, bem como os resultados obtidos com o mesmo.

CAPÍTULO 4 – SISTEMAS DE REPUTAÇÃO PARA REDES AD HOC

4.1 - Introdução

Neste capítulo são descritos os modelos de reputação existentes para as redes ad hoc, as diferenças e características entre eles, suas vantagens e desvantagens. São apresentadas também algumas propostas de sistema de reputação para VANETs.

4.2 – Modelo de Sistema de Reputação

Sistema de reputação é muito semelhante com o que se conhece como "crédito" na sociedade. De acordo com o dicionário Houaiss, reputação é "o conceito de que alguém ou algo goza num grupo". Reputação e confiança estão de certo associados, mas um não implica em outro: um indivíduo pode confiar em outro devido à sua boa reputação, ou apesar de sua má reputação.

O processo de modelagem de reputação em redes móveis *ad hoc* é dividido basicamente em três etapas: 1) a geração da reputação, que reúne informações de confiança de nós com base em seus comportamentos passados; 2) cálculo da confiança, 3) a tomada de decisão (MARMOL e PéREZ, 2010).

Na fase de geração da reputação, o nó reúne informações suficientes para um nó de destino específico por meio de interações passadas entre eles. Se os comportamentos passados deste nó específico são maliciosos, então sua reputação é baixa. No entanto, há uma possibilidade de que interações entre o nó e o nó de destino nunca tenham ocorrido, nesta situação, a informação reputação do nó de destino será recolhida a partir dos nós que tenham se comunicado com ele em toda a rede.

Durante a fase do cálculo da confiança, cada nó usa o mecanismo existente para calcular o valor de confiança pela entrada de informações de reputação obtida na primeira fase. Esta função de cálculo depende da aplicação, e um estudo está fora do escopo deste trabalho.

Na fase 3, com base no valor de confiança, os nós devem determinar se a acreditam no nó de destino ou não. Este passo requer um limiar para tomar a decisão. Valores acima do limiar implicam que este nó é confiável, enquanto um valor abaixo do limiar implica em um nó pouco confiável.

4.3 – Sistemas de Reputação em MANET

Mecanismos de reputação propostos para redes P2P e MANETs geralmente consideram uma taxa de conectividade suficiente para garantir a existência de caminhos fim-a-fim a todo o momento entre quaisquer dois membros da rede. Assim, tornam-se possíveis soluções nas quais dados utilizados pelo mecanismo de reputação são armazenados de forma distribuída na rede ou é permitido ao mecanismo esperar a completa execução do serviço solicitado, como, por exemplo, aguardar o recebimento de uma mensagem de ACK confirmando o roteamento correto de um pacote de dados enviado pela rede (DEWAN, DASGUPTA e BHATTACHARYA, 2004).

Existem ainda propostas nas quais se considera que a movimentação dos membros da rede, de maneira geral, causará apenas pequenas alterações em sua topologia, de forma que o compartilhamento de observações diretas entre pequenos grupos, aliadas às experiências locais, seja suficiente para a construção de um mecanismo robusto de reputação (BUCHEGGER e LE BOUDEC, 2004).

Reputação foi usada inicialmente na Internet e depois se espalhou para rede *ad hoc* móvel. Devido às características de MANET, Adams e Davis, (2005), Leschenauer, Gligor e Baras, (2002) e Sun, Han e Liu (2006) dão algumas características principais de confiança:

- Uma autoridade de certificação centralizada confiável não pode ser considerada. Portanto, os métodos de decisão devem ser distribuídos. Em outras palavras, um mecanismo descentralizado é necessário;
- Os nós na MANET nem sempre são cooperativos. Em ambiente restrito, nós podem se recusar a cooperar, a fim de economizar a energia ou por algumas razões egoístas;
- A confiança não é necessariamente transitiva. **A** confia em **B** e **B** confia em **C**, isto não significa que **A** confia em **C**. Cada nó toma sua decisão por si só e
- Reunir informações sobre a reputação de relacionamentos passados é de alto processamento computacional.

Diversos métodos têm sido propostos em relação a sistemas de reputação para MANET. Para conhecer estes métodos, a seguir é apresentada uma revisão dos sistemas de reputação existentes.

O trabalho de Wang et al. (2005) propôs um modelo de confiança para ser usado no roteamento para MANET. Em seu sistema, para cada nó é atribuído um valor de confiança inicial e atualizado o nível de confiança por relatórios de ferramentas de detecção. Esses relatórios são gerados a partir dos vizinhos do nó de destino controlando o comportamento alvo. O nó de origem usa esses níveis de confiança para selecionar o melhor caminho confiável para o destino.

Em Zouridaki et al. (2005) foi introduzido um sistema de confiança, chamado Hermes, com o objetivo de entrega e roteamento confiável. Os autores aplicam abordagem Bayesiana para calcular a confiabilidade entre dois nós vizinhos com base no conjunto de observações em primeira mão de encaminhamento de pacotes. O nó de origem escolhe a melhor rota calculando o peso do caminho por meio de um conjunto de redes *peer-to-peer* de confiança entre os colegas ao longo do caminho.

O trabalho de Marti et al. (2000) propôs um sistema de reputação para redes *ad hoc*, onde um nó monitora a transmissão de um vizinho para se certificar de que o vizinho encaminha o tráfego outro. Em essência, pode-se considerar esse sistema uma reputação como um jogo repetido, cujo objetivo é estimular a cooperação.

O projeto de Michiardi et al. (2002) propôs um mecanismo, chamado CORE, para reforçar a cooperação nó com base em uma técnica de monitoramento colaborativo. Cada entidade de rede no núcleo se mantém informado da colaboração de outra entidade usando um esquema de reputação.

4.4 – Sistemas de reputação em VANET

A aplicabilidade de mecanismos de reputação propostos para redes P2P e MANETs no contexto das VANETs é bem restrita, uma vez que algumas considerações feitas nessas soluções, como aquelas citadas anteriormente, não podem ser garantidas em redes veiculares, dadas as grandes velocidades de deslocamento dos veículos que compõem a rede. Assim, surge a necessidade do desenvolvimento de soluções próprias para as redes veiculares, nas quais suas características particulares são levadas em conta. A seguir são discutidas algumas propostas disponíveis na literatura.

No sistema de reputação para redes *ad hoc* veiculares (*A vehicle ad hoc network reputation system* - VARS) (DÖTZER, FISCHER e MAGIERA, 2005), cada veículo adiciona sua opinião sobre a veracidade das mensagens por ele recebidas e encaminhadas. Essa opinião é baseada na combinação dos seguintes fatores: experiência do veículo (caso o evento relatado

já tenha sido detectado por ele), reputação da fonte da mensagem e reputação dos responsáveis pelas opiniões já adicionadas à mensagem. As mensagens recebidas são armazenadas para serem utilizadas posteriormente no processo de decisão sobre o evento (o mecanismo faz uso dos conceitos de regiões geográficas ao redor dos eventos). Quando a decisão é tomada, apenas a reputação das fontes dessas mensagens é atualizada.

Os primeiros veículos a receber mensagens geradas normalmente são aqueles mais próximos da área de reconhecimento e, por conseguinte, da área de decisão do evento. Em situações nas quais a fonte dos dados é desconhecida do receptor, não existe nenhuma base de confiança para o julgamento dos dados, uma vez que poucas ou nenhuma opinião foi adicionada à mensagem antes de seu recebimento. Logo, nesses casos, mensagens geradas por veículos têm o mesmo peso que aquelas geradas por intrusos.

O trabalho de (WANG e CHIGAN, 2007) propõe um mecanismo que, ao invés de considerar registros passados, conta apenas com o comportamento dos veículos em tempo de execução para a definição de reputações instantâneas. Segundo a proposta, um emissor envia o pacote de dados para seus vizinhos. Cada vizinho, caso não seja o destino final, reencaminha o pacote recebido. O emissor então monitora essas retransmissões. Se o pacote retransmitido não sofreu nenhuma alteração, o emissor envia um *token* de confiança, assinado digitalmente, para o respectivo vizinho. Os vizinhos devem reencaminhar este *token*, que os certifica como confiáveis, para os veículos responsáveis pelo próximo salto. O processo de escuta e emissão do *token*, feito pelo emissor da mensagem, agora é feito pelos vizinhos certificados. Esse ciclo se repete em todos os saltos até que o pacote atinja seu destino. O trabalho se preocupa apenas com possíveis alterações dos dados da mensagem durante o roteamento, sem avaliar o conteúdo da mensagem em si. Assim, mensagens contendo informações erradas podem facilmente ser distribuídas pela rede.

Embora os resultados apresentados mostrem que cada passo da sequência envia mensagem – escuta reenvio - envia *token*, adiciona, separadamente, uma sobrecarga aceitável na rede, não é feita uma avaliação dessa sequência no que diz respeito ao tempo total necessário para sua execução, uma vez que a topologia dinâmica das VANETs cria desafios consideráveis para soluções que necessitam de mais de uma conexão em uma mesma transação.

Em (OSTERMAIER, DOTZER e STRASSBERGER, 2007) os autores apresentam um esquema baseado em votos para aumentar a segurança das decisões tomadas pelos veículos sobre eventos reportados em aplicações LDW (*Local Danger Warnings*). A partir das mensagens recebidas dentro da área de disseminação, o trabalho avalia o desempenho de quatro

diferentes métodos de decisão, executados quando o veículo adentra a área de decisão do evento: Última mensagem, Maioria das mensagens, Maioria das últimas x mensagens e Maioria das últimas x mensagens considerando um limite inferior. A eficiência desses quatro métodos é medida em uma rede submetida a ataques do tipo alarmes falsos e alarmes trocados.

Os resultados apresentados mostram que o método Maioria das últimas x mensagens considerando um limite inferior, onde o valor de x define as últimas x mensagens mais recentes, conseguem resultados significativos mesmo em um cenário com 40% de veículos maliciosos. Entretanto, como nenhum mecanismo de reputação é utilizado, esses intrusos podem executar seus ataques livremente, na certeza de que suas mensagens serão sempre consideradas no processo de decisão dos outros veículos, uma vez que não existe nenhum tipo de punição a seu mau comportamento.

Em Lo e Tsai (2009) é apresentado um sistema de reputação baseado em eventos. O valor da reputação de um evento define a intensidade deste evento na rede em que o seu valor inicial é zero.

O mecanismo de (PATWARDHAN, A. et al, 2006) acrescenta o uso de reputações a um esquema simples de votação, parecido com o Maioria das mensagens (OSTERMAIER, DOTZER e STRASSBERGER, 2007). Assim, um veículo colhe informações sobre eventos até que sejam recebidos dados transmitidos por uma fonte confiável, quando a decisão é tomada apenas com base nas informações contidas nessa mensagem, ou uma quantidade mínima de mensagens é recebida, quando o esquema de votação é executado. Uma vez que a decisão é tomada, o mecanismo de reputação proposto promove os veículos cujas mensagens enviadas corroboram com tal decisão enquanto rebaixa aqueles responsáveis por mensagens com opiniões contrárias. A partir do momento em que um veículo da rede é considerado malicioso, suas mensagens serão sempre descartadas.

A solução proposta não permite o compartilhamento de dados de reputação entre os veículos da rede, fato que pode representar uma demora considerável no processo de estabelecimento de relações de confiança e detecção de intrusos. Essa situação, aliada ao uso de uma metodologia na qual a promoção ou rebaixamento das fontes é feito com base apenas na opinião da maioria local, sem nenhuma validação posterior, possui uma implicação considerável. Em áreas onde é grande a concentração de veículos maliciosos ainda não descobertos, é provável que as impressões disseminadas por esses intrusos prevaleçam, levando à sua promoção com consequente rebaixamento de veículos cooperativos. Além disso, como

não é definida nenhuma política de redenção de intrusos, a quantidade de falsos positivos pode se tornar um grande problema em longo prazo.

Em Dhurandher et al. (2010), os autores introduziram VSRP (Vehicular Security through Reputation and Plausibility), um algoritmo com base na reputação que utiliza valores de confiança atribuídos a nós para detectar e eliminar os nós maliciosos da rede. Na verdade, este trabalho está focado principalmente na detecção de nós maliciosos. Os resultados das simulações são muito eficazes e adequados para a rede *ad hoc* móvel.

O projeto de Huang (2011) apresenta um sistema de reputação baseado em votos cascatas, em que o peso das informações está relacionado com a posição do veículo, ou seja, os veículos mais próximos a um evento têm peso maior para avaliar uma informação. Este sistema busca avaliar a informação, ou seja, ele tenta avaliar quais informações são falsas e quais não são ao invés de tentar encontrar um veículo malicioso.

A reputação representa o quanto um nó é seguro em qualquer interação com outro nó. Durante uma comunicação iniciada por um determinado nó, outro nó só poderá participar da comunicação quando possuir certo grau de confiabilidade para o primeiro. Assim, um mesmo nó pode ser visto por outros nós como tento diferentes graus de confiabilidade.

Em Li et al. (2012) é proposto um sistema de reputação para redes veiculares que permite avaliar a confiabilidade da mensagem recebida. A confiabilidade da mensagem é avaliada de acordo com a reputação do veículo que gerou esta mensagem, sendo esta representada por uma pontuação numérica. O sistema é composto por três entidades: servidor de reputação, *access-point* e veículos. É proposto um esquema de anúncio para VANETs com base em um sistema de reputação que permite a avaliação da confiabilidade da mensagem. O esquema apresentado é seguro e eficiente e que é robusto e tolerante a falhas contra a indisponibilidade temporária do servidor central.

O projeto de Chen et al. (2013) apresenta um sistema que facilita veículos transmitirem informações relacionadas à estrada em redes ad hoc veiculares (VANETs), a fim de melhorar a segurança rodoviária e eficiência. Neste trabalho, foi proposto um esquema de reputação baseado em anúncio com reconhecimento de privacidade que fornece avaliação da confiabilidade da mensagem, auditabilidade e robustez.

4.5 – Considerações Finais

Este capítulo apresentou alguns sistemas de reputação propostos no meio acadêmico. Por meio da revisão bibliográfica mostrou-se que o maior uso de reputação em Manets e VANETs se dá por meio de um sistema distribuído.

Dessa maneira este trabalho tem o objetivo de apresentar uma proposta centralizada para o sistema reputação. A arquitetura e funcionamento deste sistema de reputação centralizado são apresentados no próximo capítulo.

CAPÍTULO 5 – SISTEMA DE REPUTAÇÃO CENTRALIZADO – SRC

5.1 - Introdução

Neste capítulo é abordada a organização e arquitetura do sistema de reputação centralizado, sendo também apresentado como deve ser o funcionamento do SRC (Sistema de Reputação Centralizado), com suas entidades e funções.

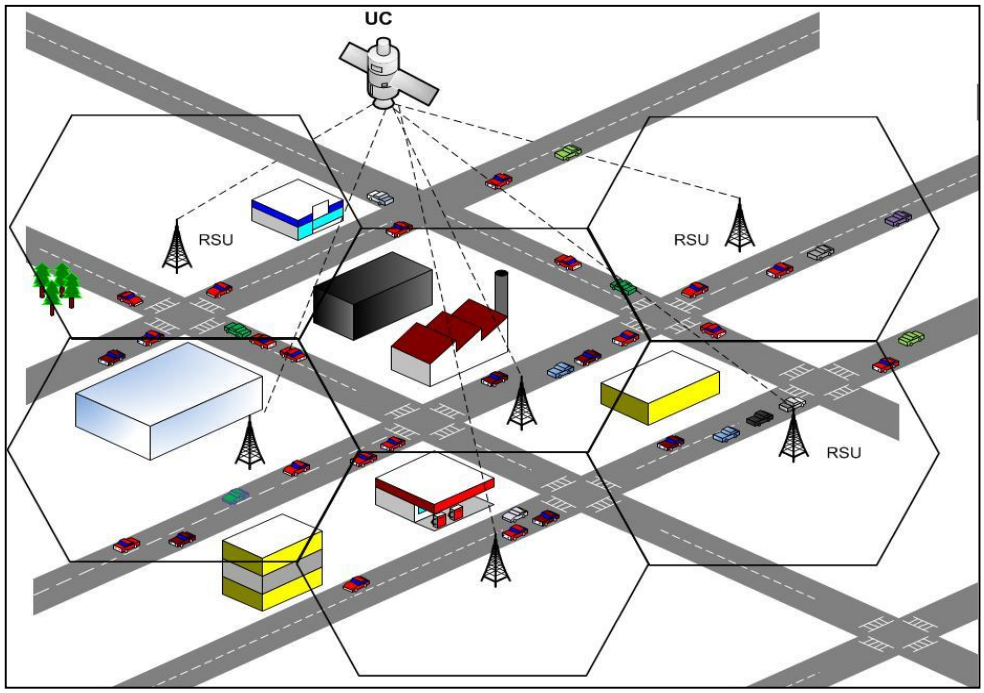
5.2 – Arquitetura do Sistema de Reputação Centralizado

Grande parte das pesquisas e trabalhos direcionados para os sistemas de reputação vêm abordando sistemas baseados em arquiteturas distribuídas, em que os próprios carros armazenam informações relacionadas a reputações dos veículos vizinhos. Esses sistemas parecem o mais sensato para este tipo de abordagem, devido ao pressuposto de que deve ser o veículo responsável por guardar informações a respeito dos veículos vizinhos. Porém devido a alta mobilidade das conexões entre os veículos tem-se alguns problemas relacionados ao tráfego de dados das conexões: como quantidade de informações trocadas; por quanto tempo um veículo deve armazenar a informação a respeito do veículo; entre outros.

Para minimizar estes problemas e buscando uma nova abordagem, este trabalho tem como objetivo desenvolver e simular um sistema de reputação centralizado para redes veiculares *ad hoc*. Neste sistema, os responsáveis por armazenar as informações a respeito das reputações dos veículos são as Estações Base (RSU), posicionadas nas laterais das rodovias. Além disso, haverá uma unidade central (UC) que conectará todas as RSU, e que será responsável por armazenar as reputações de todos os veículos.

Esta UC pode ser a mesma responsável por gerar os certificados de autenticação dos veículos na rede. A UC conterá as reputações de todos os veículos de uma região. Esta região pode ser uma cidade, estado ou até país, dependendo da estrutura da arquitetada. Desta maneira sempre que uma RSU for solicitada a informar a reputação de um veículo, se esta não tiver a informação, a mesma consulta a informação na UC. Esta arquitetura pode ser vista na Figura 8 e será detalhada na seção 5.3.

Figura 8 - Arquitetura da VANET com Reputação Centralizada.



Fonte: elaborado pelo autor

Estas Estações Base (RSU) discutidas na seção 2.3.2, conterão um sistema gerenciador de banco de dados (SGBD) que armazenará as informações sobre as reputações dos veículos. Os campos da tabela serão os seguintes:

- **ID_Veículo:** campo de identificação do veículo. Este campo será único para cada veículo. Alguns trabalhos (Hubaux, 2004) propõe a utilização de pseudônimos ao invés de usar um ID para o veículo, para que assim garanta a privacidade dos usuários. Porém a unidade certificadora que gerencia estes pseudônimos conhece o ID do veículo, portanto este ID será considerado na arquitetura aqui proposta;
- **Reputação:** grau de reputação do veículo. O grau da reputação está contido no intervalo [0;10], em que 0 é o valor mínimo e 10 o valor máximo.

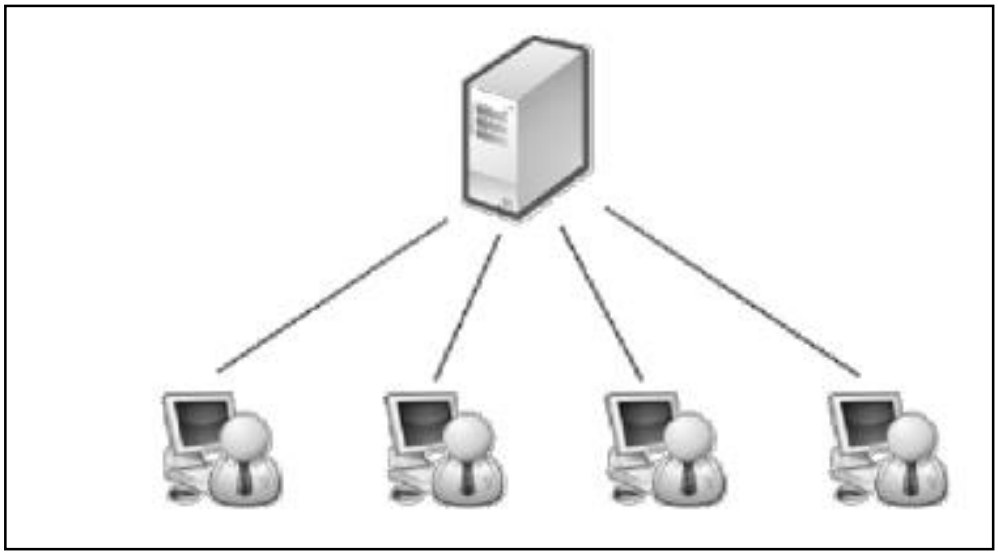
Os veículos irão tomar as decisões a respeito de uma informação baseado no grau de reputação do veículo emissor da informação. Quanto maior o grau de reputação do veículo mais confiável ele será, e vice-versa.

Todos os veículos iniciam com o grau de reputação igual a zero e no momento em que as informações são enviadas esta reputação é alterada com base no cálculo de um algoritmo.

Para implementar o sistema de reputação proposto foi utilizada uma arquitetura baseada em Cliente/Servidor (PATWARDHAN, A. et al, 2006) (COMER, 2007) (Figura 9), em que os Clientes são os veículos e os Servidores são as Estações Base.

Nesta arquitetura cada instância de um cliente pode enviar requisições para algum dos servidores conectados e esperar pela resposta. Por sua vez, alguns dos servidores disponíveis podem aceitar tais requisições, processá-las e retornar o resultado para o cliente.

Figura 9 - Modelo Cliente-Servidor.



Fonte: Elaborado pelo autor

A comunicação dos veículos com as RSU se dá por meio de transmissão sem fio utilizando o padrão 802.11p. Porém, as comunicações entre RSU com RSU ou RSU com UC pode ser por rede cabeada, de preferência com fibra ótica, pode ser também através de links de micro-ondas ou de tunelamento entre 802.11p e 4G/5G. Como a capacidade de armazenamento e processamento de uma RSU ou UC é muito maior que a dos veículos, a RSU se tornam os membros ideais para gerenciar as reputações, baseando-se na arquitetura deste projeto.

Considerando esta arquitetura, a VANET estará totalmente conectada e os veículos terão acesso às reputações de qualquer local. Sendo assim, algumas vantagens que se acredita que uma arquitetura centralizada poderá oferecer são apresentadas:

- Maior capacidade de armazenamento e processamento: devido as informações das reputações ficarem armazenadas nas RSU e UC que possuem maior capacidade de processamento e armazenamento, o algoritmo para classificar um veículo ou a capacidade de receber diversas solicitações/atualizações de

reputação é alto e o tempo de resposta é muito curto. Desta maneira o veículo iria apenas consultar a reputação e tomar a decisão com base na resposta obtida, ao contrário das arquiteturas distribuídas em que o veículo precisaria coletar informações de outros veículos, processar esta informação e então tomar uma decisão;

- Tempo de armazenamento da informação: como os dispositivos de armazenamentos estão cada dia mais avançados e com alta capacidade de armazenamento, a RSU e a UC poderiam armazenar a informação sobre um veículo por tempo indeterminado.

Deve-se considerar que as UCs poderão e possivelmente deverão se reportar a uma unidade de nível superior, fazendo uma conexão total entre as regiões, isso para que, mesmo se um veículo mudar de estado ou até país seja possível ter informação sobre sua reputação. A arquitetura pode utilizar também recursos do armazenamento em estruturas de *cloud*, buscando diminuir os custos e aumentando a disponibilidade e escalabilidade das reputações armazenadas.

5.3 – Funcionamento do Sistema de Reputação Centralizado

Nesta seção é apresentado o funcionamento do SRC (Sistema de Reputação Centralizado), como as entidades interagem entre si, as funções a serem executadas e os algoritmos desenvolvidos.

O Algoritmo 1 deverá ser executado quando um veículo chamando a função AvaliaReputação avalia um veículo com base em informações de perímetro. No Algoritmo 1 a função BuscaVeiculo deve retornar o campo com a chave ID informada, após o sistema recuperar a informação a respeito do registro, é aplicado um algoritmo para classificar a reputação do veículo. Este algoritmo pode ser qualquer algoritmo de votos conhecido, pois neste caso o algoritmo não é o mais importante e está fora do escopo deste projeto. Caso a função BuscaVeiculo não retorne registro, a RSU consulta a UC para verificar se esta possui informação sobre o veículo consultado, pois a UC pode ter recebido informações deste veículo de outras regiões. Caso a UC também não tenha informações sobre o veículo consultado um

novo campo é adicionado à tabela do SGBD da RSU com os valores de ID e VR (Valor de Reputação) informados. Após ter executado todas as operações de avaliação da reputação, a informação sobre o veículo atualizada é enviada para a UC.

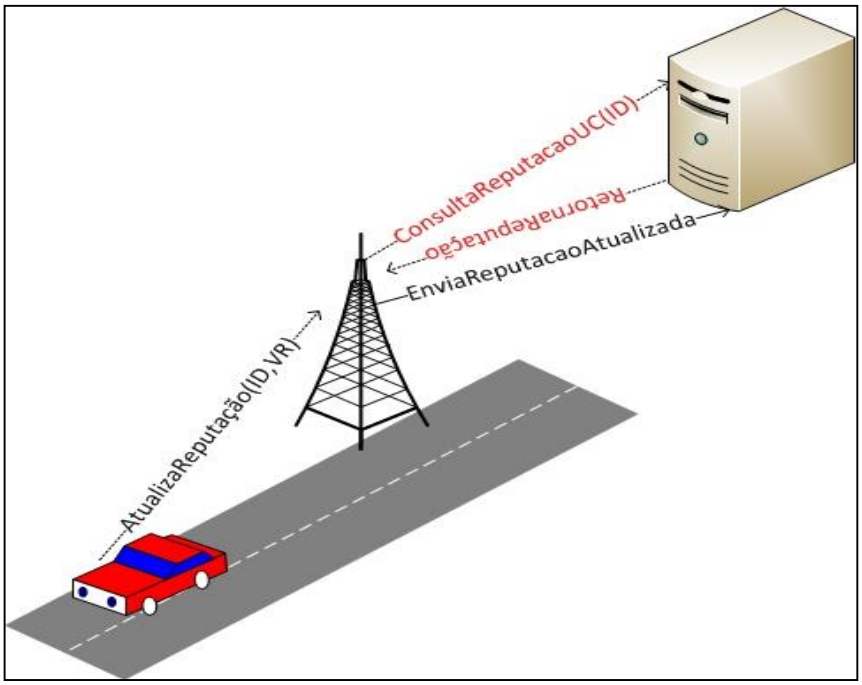
Algoritmo 1 AtualizaReputacaoRSU

Entrada: ID do veículo ID, valor da reputação VR

```
1: BuscaVeiculo(ID)
2: se a BuscaVeiculo retornar  algum valor então
3: Aplica o algoritmo para avaliar a reputação, de acordo com o valor de VR
4: senão ConsultaReputaçãoUC(ID)
5:         se retornar algum registro então
6:               Aplica o algoritmo para avaliar a reputação, de acordo com o valor
           de VR
7:         senão
8:              Adiciona um novo campo à tabela com o ID e VR informados.
9: Depois de finalizada a avaliação, EnviaReputacaoAtualizada(ID,VR)
```

Na Figura 10 é possível ver a arquitetura da atualização de reputação no modelo centralizado.

Figura 10 - Arquitetura para atualizar reputação.



Fonte: Elaborada pelo autor.

O Algoritmo 2 deverá sempre ser executado quando um veículo recebe um alerta de outro veículo e deseja verificar a reputação do mesmo. Esta consulta é sempre feita por meio do método ConsultarReputacao(ID), que faz uma solicitação à RSU, perguntando se esta tem informações sobre o veículo em questão. Se a RSU tiver a informação, esta deve retornar a reputação para o veículo que fez a solicitação. Caso a RSU não tenha a informação ela deve consultar a UC para verificar se esta possui a informação requerida. Se a UC tiver a informação, ela deve enviá-la para a RSU que a envia para o veículo que iniciou a consulta. Se a informação a respeito da reputação não for encontrada nem na RSU nem na UC deve ser retornado o valor -1. Neste caso o sistema não tem informação de reputação para tomar alguma decisão.

Algoritmo 2 ConsultarReputacao

Entrada: ID do veículo ID

Saída: valor da reputação do veículo ID

1: Solicita a reputação do veículo ID para a RSU
2: BuscaReputacao (ID)

```
3:  se encontrou  algum valor então
4:  Retorna o valor da reputação encontrado
5:  senão ConsultaReputaçãoUC(ID)
6:          se retornar algum registro então
7:          Retorna o valor da reputação encontrado
8:          senão
9:              Retorna -1
```

Estes algoritmos interagindo com as ações dos agentes do simulador NCTUns farão o sistema de reputação centralizada ser executado. As implementações destes algoritmos, as modificações no simulador NCTUns, a implantação do SGBD MySql e integração com o simulador são apresentados nos apêndices A, B e C.

5.4 - Alteração efetuadas no simulador NCTUns

O simulador NCTUns é um software de código aberto o que permite desenvolvedores modificarem seus códigos para adicionar novas funções e capacidades. Além disso, pode-se adicionar novas funções e protocolos ao *engine* do simulador para serem utilizadas nas simulações.

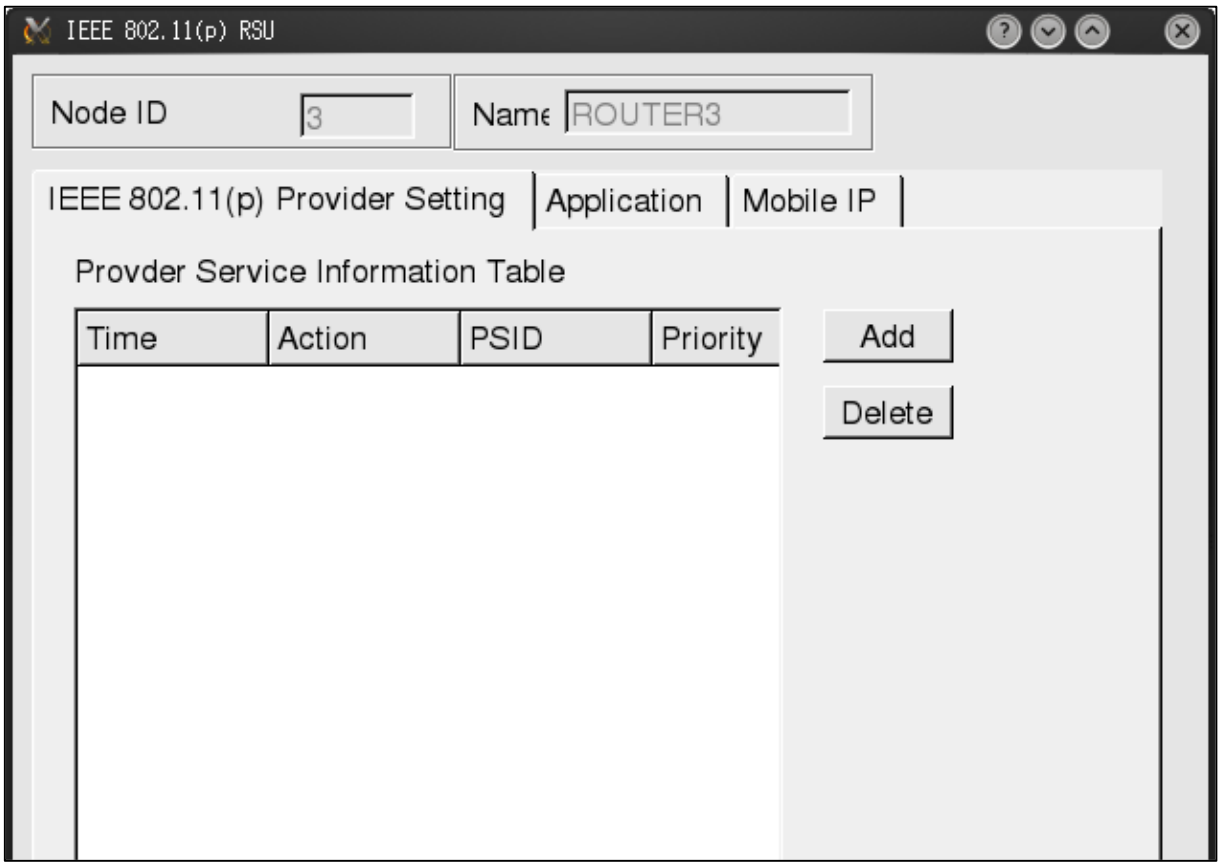
As alterações foram efetuadas nos programas agentes: CarAgent e RSUAgent. Além da criação de um novo agente: UCAgent. Também foi adicionada uma funcionalidade para o simulador NCTUns poder se conectar a o SGBD MYSQL. Como estes componentes foram modificados e os arquivos criados para esta proposta são descritos nas seções 5.4.1, 5.4.2 e 5.4.3.

5.4.1 – Programas Agentes

Quando um cruzamento é implementado, um “agente sinal” será executado automaticamente quando uma simulação se inicia. O “agente sinal” é responsável por mudar os estados do semáforo. Da mesma forma, quando um carro ITS é implantado, um “agente carro” será automaticamente executado quando uma simulação se inicia. O “agente carro” é responsável por dirigir o carro.

Todos esses programas agentes são programas de nível de usuário escritos em C/C++ e as *strings* de comando para o lançamento desses programas durante a simulação deve ser digitado na aba "*Application*" de cada um desses nós (Figura 11).

Figura 11 - Aba Application do editor de nó.



Fonte: The GUI User Manual for the NCTUns 5.0 Network Simulator and Emulator

Os programas agentes são responsáveis por controlar os nós envolvidos na simulação, estes nós podem ser: carros, RSUs, semáforos, entre outros. Para este projeto foram modificados os programas agentes: CarAgent e RSUAgent. Entretanto, para que estes agentes originais possam ser utilizados em alguma simulação futura, novos arquivos com as alterações feitas foram criados e chamados respectivamente: CarAgentModificado e RSUAgentModificado.

Um novo programa agente, UCAgent, foi criado baseado no RSUAgent. Este novo agente representa a UC (Unidade Central) responsável por gerenciar as RSUs (*Road-Side Units*).

As modificações e funcionamento destes programas agentes são descritos nas seções 5.4.2 e 5.4.3. Os códigos completos dos programas agentes são apresentados nos Apêndices.

5.4.2 – CarAgentModificado

O programa agente CarAgent é responsável pela movimentação dos veículos. Este controla os movimentos dos carros, escolhendo a direção nos cruzamentos, aceleração, entre outros. Neste projeto o CarAgent foi modificado para executar as seguintes funções:

- Sempre que um veículo receber uma mensagem de alerta (acidente, congestionamento, entre outras) este deve consultar a RSU para obter informação sobre a reputação do veículo que enviou a mensagem de alerta e então baseado neste valor tomar uma decisão.
- Em determinado momento o veículo pode enviar para a RSU uma avaliação, negativa ou positiva, sobre um determinado veículo.

O módulo CarAgent original não foi modificado, e sim um novo módulo que controla o veículo foi criado, o módulo CarAgentModificado. Este novo módulo tem as mesmas funções que o CarAgent, acrescentado das novas funcionalidades de consulta de reputação e avaliação de veículo.

5.4.3 – RSUAgentModificado

Este agente é responsável por controlar as ações das Estações Bases (RSU – *Road-Side Unit*), ou seja, repassar mensagens recebidas para os veículos em sua região. Lembrando que em uma aplicação real, as RSUs podem fornecer funções adicionais como: acesso à Internet, mensagens de propagandas, etc. No simulador NCTUns a função das RSUs é apenas repassar mensagens, função necessária para este projeto.

Neste projeto o módulo RSUAgent foi modificado para executar as seguintes funções:

- Conectar-se a o SGBD MYSQL;
- Aceitar requisições de informações sobre reputação dos veículos;
- Responder as requisições aos veículos;
- Atualizar à UC sobre as reputações dos veículos;
- Consultar à UC para verificar reputação de um veículo.

Assim como explicado na seção 3.5.2 o módulo RSUAgent original não foi modificado, e sim um novo módulo que controla a RSU foi criado, o módulo

RSUAgentModificado. Este novo módulo tem as mesmas funções que o RSUAgent, acrescentando as novas funcionalidades.

5.4.4 – UCAgent

Este é um novo agente criado e foi implementado baseado no agente RSUAgent, porém este não repassa mensagens. Este agente é responsável por gerenciar as reputações entre todas as RSUs de uma região, que pode ser uma cidade, estado, país, entre outros.

Com este agente é possível manter todas as RSUs sempre atualizadas, pois esta recebe informação sobre reputação de todas as RSUs. Para que isso ocorra o UCAgent terá as seguintes funcionalidades:

- Conexão com SGBD MYSQL;
- Aceitar requisições das RSUs e
- Responder as requisições das RSUs.

5.5 – Considerações Finais

Este capítulo apresentou o sistema de reputação centralizado desenvolvido neste trabalho e as modificações feitas no NCTUns para implementá-lo. Foram descritas as funções principais que compõe essa arquitetura. Entretanto, faz-se necessário, frente as vantagens aparentes de uma arquitetura centralizada, averiguar quais são os resultados para o uso de um sistema de reputação que seja centralizado. Também foram apresentadas as alterações efetuadas no simulador NCTUns para serem feitas as simulações da arquitetura.

Com o objetivo de demonstrar a viabilidade de um sistema de reputação centralizado, o próximo capítulo apresenta a simulação desse sistema, bem como os resultados das simulações realizadas.

CAPÍTULO 6 - SIMULAÇÃO DO SRC E RESULTADOS

6.1 – Introdução

Neste capítulo são apresentados os cenários utilizados para a simulação do Sistema de Reputação Centralizado e seus respectivos resultados.

Como apresentado no Capítulo 4 um veículo pode tomar sua decisão baseada na reputação de outro veículo e esta é obtida por meio de uma consulta à RSU. Além disso um veículo poder reportar sua avaliação sobre outro veículo para a RSU. Desta maneira os cenários simulados têm estas características e assim será possível avaliar o comportamento dos veículos utilizando a arquitetura de um sistema de reputação centralizado.

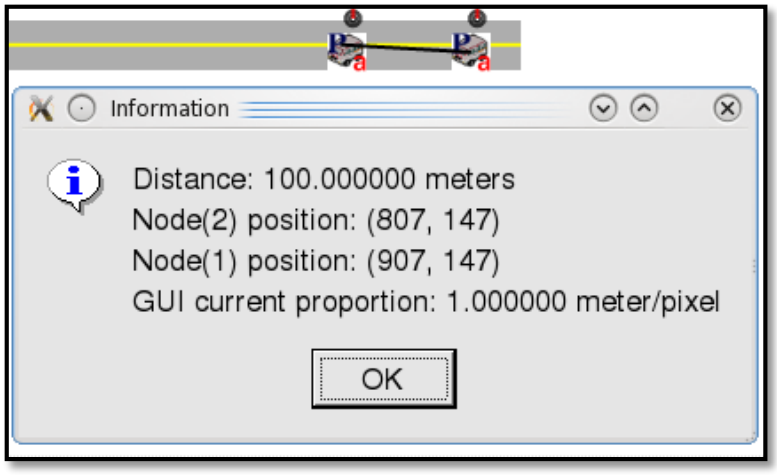
6.2 – Comportamentos avaliados

Para a avaliação dos resultados obtidos certos comportamentos dos veículos foram verificados.

- **Velocidade média**: velocidade média dos veículos antes e depois de receber uma mensagem de alerta;
- **Distância percorrida durante espera por uma resposta da RSU**: conforme ilustrado na Figura 12, cada ponto na coordenada do simulador NCTUns representa 1 metro, portanto utilizando a distância euclidiana (Equação 01) entre dois pontos é possível determinar a distância percorrida.

$$d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2} \quad \textbf{(Equação 01)}$$

Figura 12 - Régua do NCTUns.



Fonte: The GUI User Manual for the NCTUns 5.0 Network Simulator and Emulator

- **Tempo de resposta:** Analisando os dados de distância e velocidade é possível determinar o tempo de espera pela resposta da RSU (Equação 2).

$$\boldsymbol{Tempo = \frac{Distância\ Percorrida\ (m)}{Velocidade\ (m/s)}} \quad \textbf{(Equação 2)}$$

Além destes comportamentos também são apresentadas as posições finais dos veículos no mapa simulado para que se possa visualizar a distribuição dos veículos e as tabelas do banco de dados com as informações da reputação dos veículos.

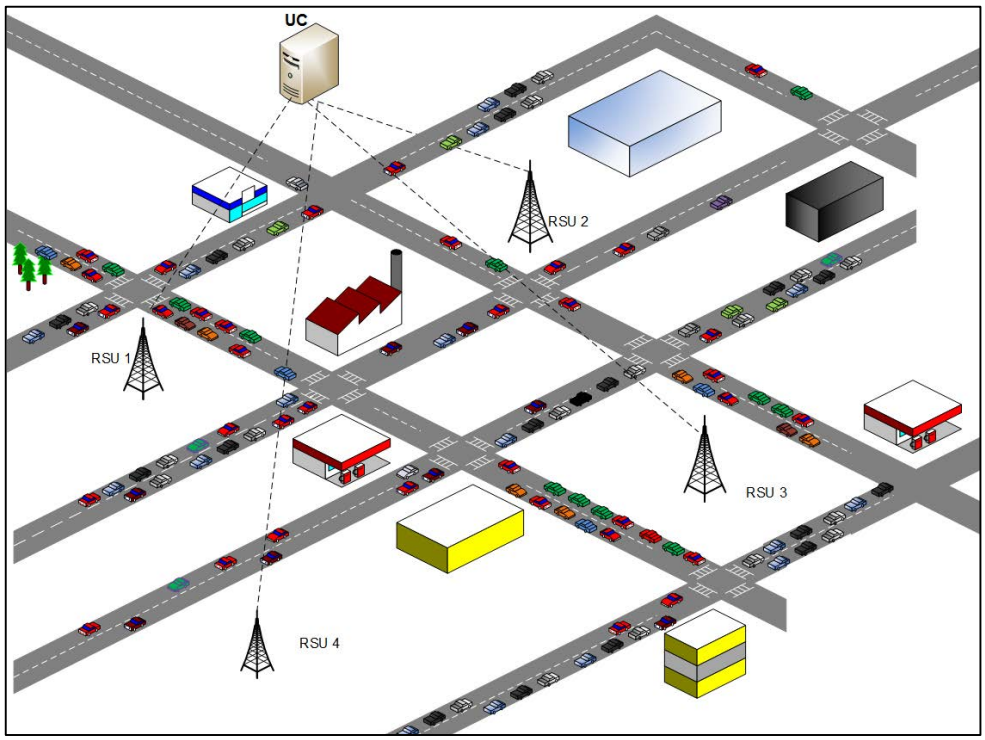
6.3 – Descrição do cenário da simulação do SRC

Para demonstrar o funcionamento deste SRC nesta seção são apresentados o ambiente simulado e os resultados obtidos.

O cenário modelado é um ambiente urbano dividido em quatro regiões, denominadas regiões 1, 2, 3 e 4 (Figura 13). Com estas quatro regiões pode se ter um entendimento de como deve ser a arquitetura da VANET para um correto funcionamento do SRC. As RSUs são responsáveis por determinadas regiões e se reportam a uma Unidade Central. Para cidades, estado ou até países inteiros esta arquitetura deve se manter, apenas aumentando a quantidade

de RSU e UC. Desta maneira espera-se que o desempenho não seja prejudicado, pois hoje se tem uma capacidade de processamento muito alta, o que garante um tempo de resposta adequado para os veículos solicitantes.

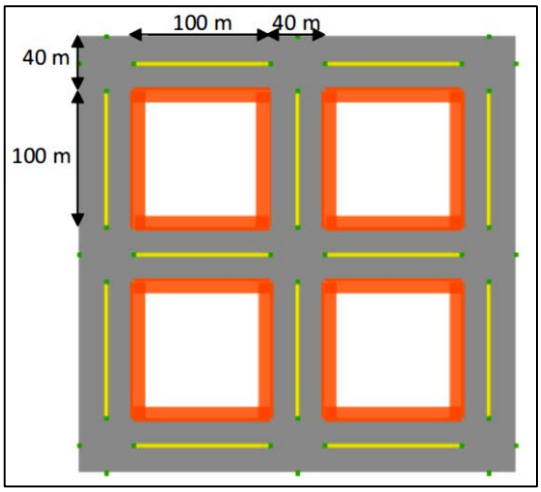
Figura 13 - Ambiente urbano simulado



Fonte: Elaborada pelo autor

Na Figura 14 são apresentados os comprimentos das ruas e cruzamentos.

Figura 14 - Comprimentos das ruas e cruzamentos do ambiente simulado

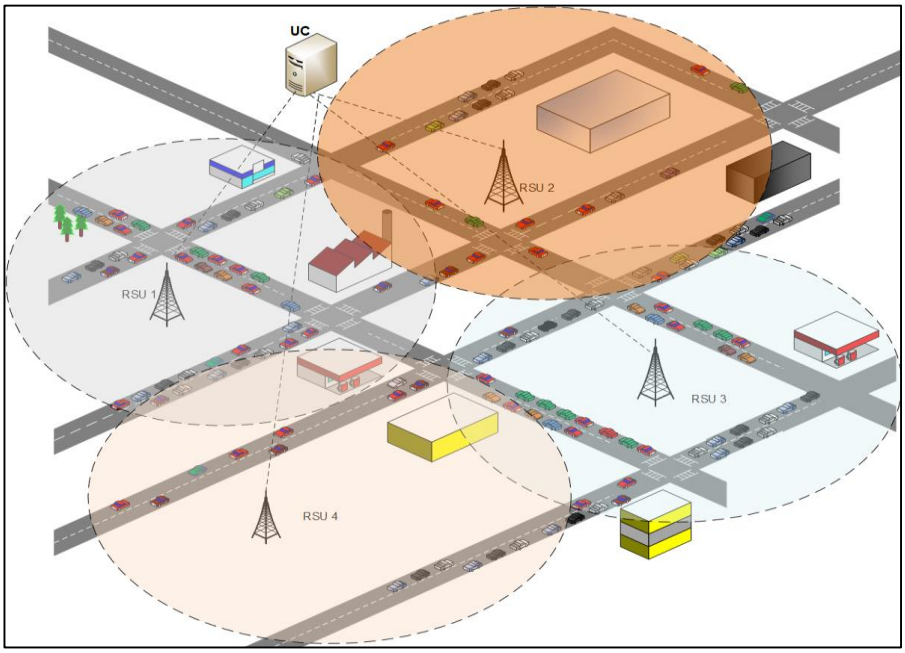


Fonte: Elaborada pelo autor

Cada região tem uma RSU que é responsável pelo gerenciamento da reputação dos veículos desta região (Figura 15). Em cada região existem alguns veículos que possuem velocidades e acelerações diferentes. Considerando um ambiente urbano os perfis dos veículos foram configurados para velocidade máxima de 60 Km/h (~16m/s). O simulador NCTUns oferece uma ferramenta para estas configurações em que pode-se determinar a velocidade máxima, aceleração máxima e aceleração mínima (Figura 16) e em seguida distribuí-las entre os veículos (

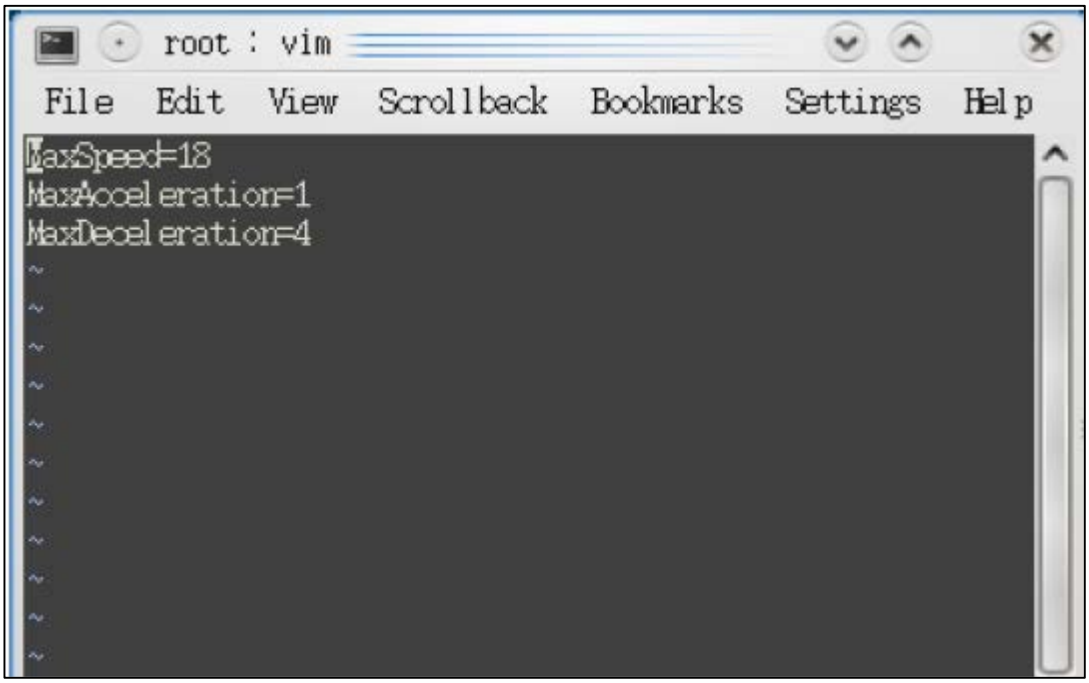
Figura 17).

Figura 15 - Áreas cobertas pela RSUs.



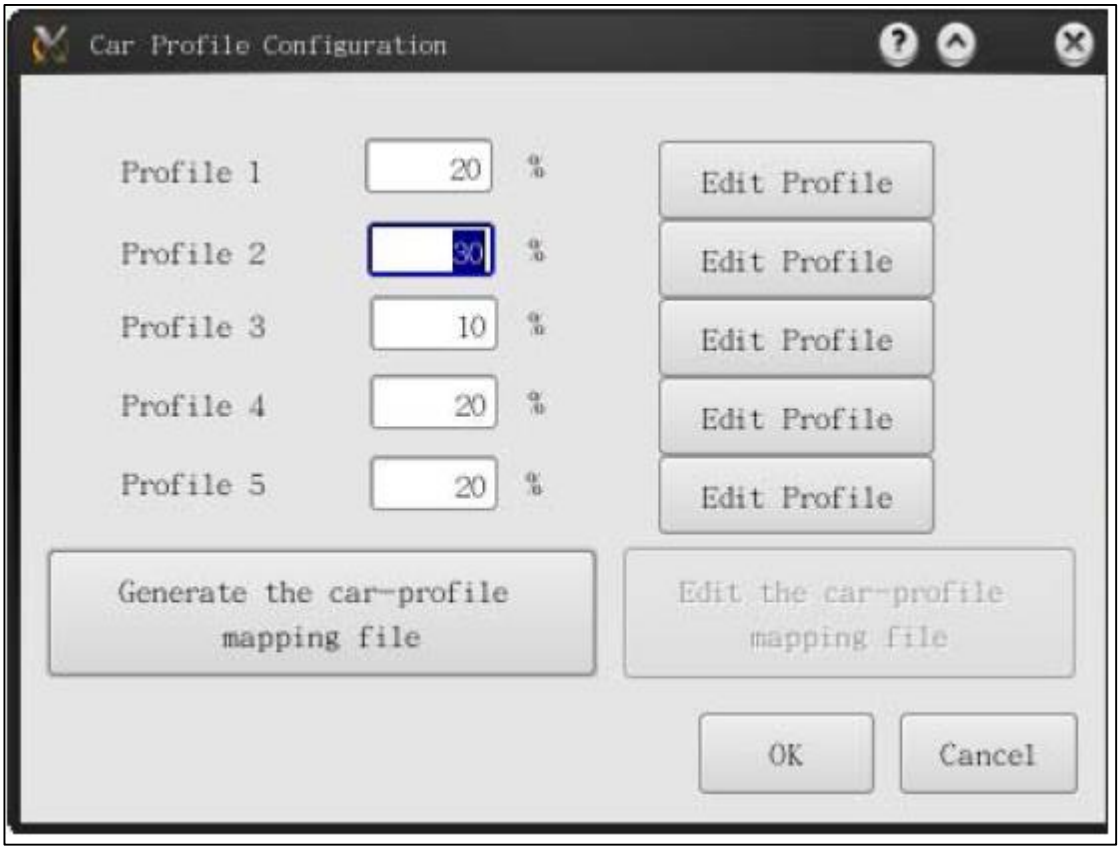
Fonte: Elaborada pelo autor

Figura 16 - Configuração do arquivo de profile do veículo.



Fonte: Software NCTUns 5.0

Figura 17 - Distribuição dos profiles entre os veículos da rodovia.



Fonte: Software NCTUns 5.0

O tempo da simulação foi de 60 segundos (Figura 18). O veículo emissor inicia o envio de mensagens a partir dos 10 segundos decorridos da simulação (Figura 19).

Figura 18 - Configuração do tempo de simulação.

Fonte: Software NCTUns 5.0

Figura 19 - Configuração do envio de alerta.

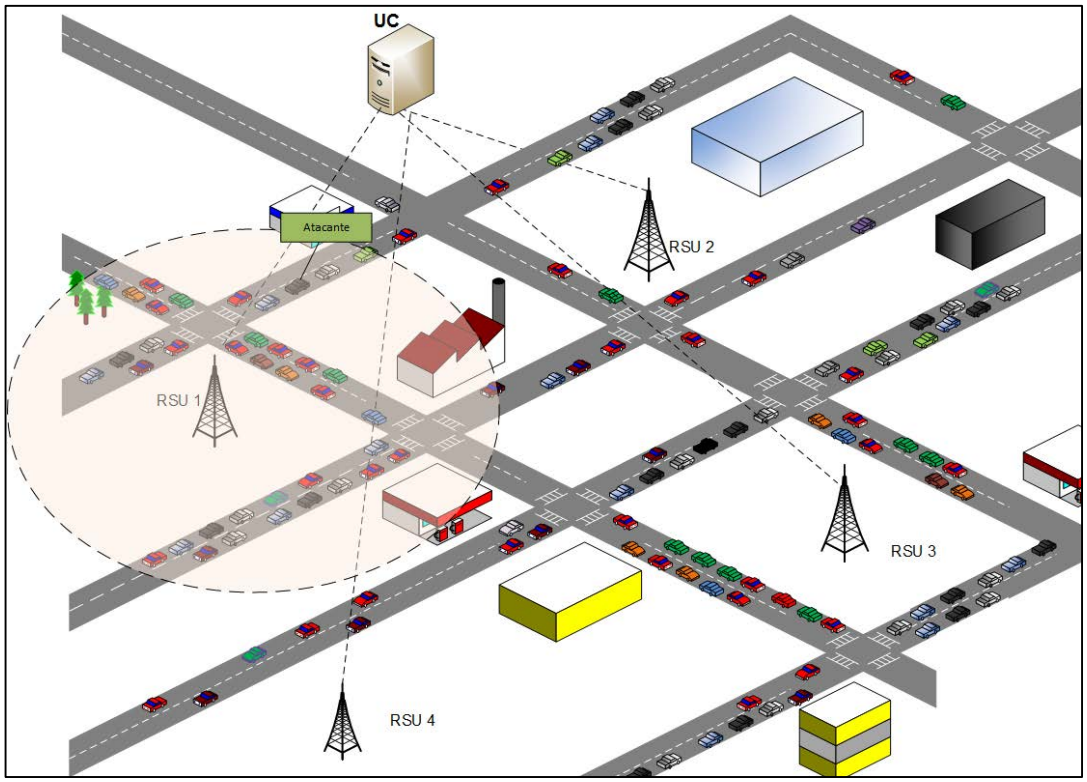
Fonte: Software NCTUns 5.0

6.3.1 – Cenário 1

O primeiro cenário simulado representa a região do ambiente urbano denominado Região 1. Neste primeiro cenário um veículo malicioso lança um ataque, enviando um alerta falso (alerta de congestionamento, por exemplo), na região 1 (Figura 20).

No primeiro momento não se tem informação a respeito da reputação do veículo que enviou a mensagem, desta maneira os demais veículos tendem a confiar na mensagem e tomam alguma decisão.

Figura 20 - Ambiente para teste do cenário 1.



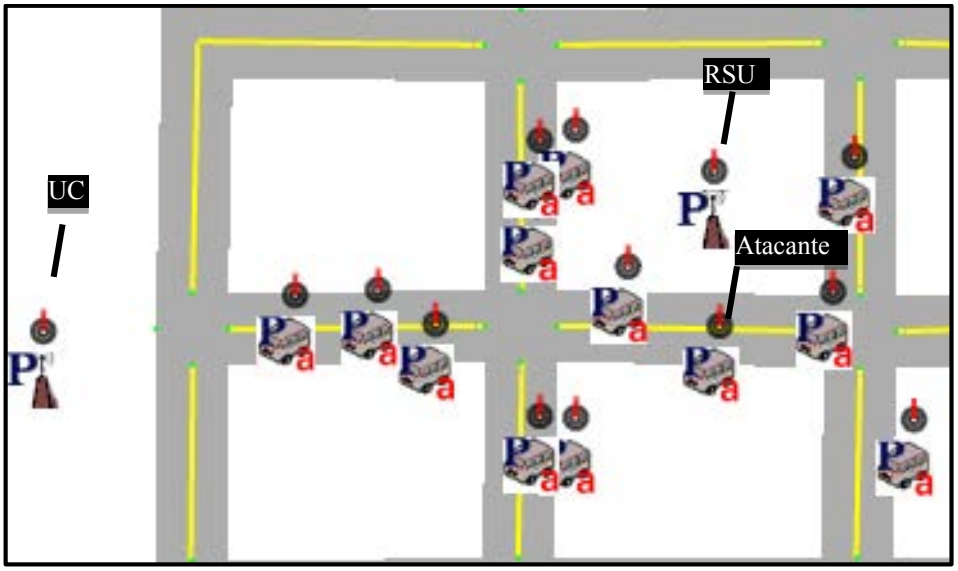
Fonte: Elaborada pelo autor

Neste ambiente um veículo malicioso tenta obter algum benefício lançando um ataque. Os veículos conectados nesta região devem verificar se o veículo que está enviando a mensagem é confiável. Para isso eles irão utilizar os métodos implantados na arquitetura centralizada.

Primeiro os veículos irão consultar a RSU 1, executando o método ConsultarReputação(id), passando o ID do veículo, neste caso o ID 28. Neste momento a RSU 1 não possui informação a respeito da reputação deste veículo, sendo assim a RSU 1, efetua a consulta à sua Unidade Central, para verificar se esta contém a informação. Para este caso a UC também não terá esta informação, portanto retornará -1, de acordo com o Algoritmo 2, da seção 5.3. Consequentemente a RSU 1 também retornará -1 e, portanto, os veículos ainda não têm informações suficiente para tomar a decisão correta e até o momento devem supor que a mensagem é verdadeira.

Na Figura 21 é apresentado este cenário modelado no simulador NCTUns.

Figura 21 - Cenário 01 simulado no NCTUns



Fonte: Elaborada pelo autor

Na Tabela 2 é apresentado um resumo de todas as informações da simulação.

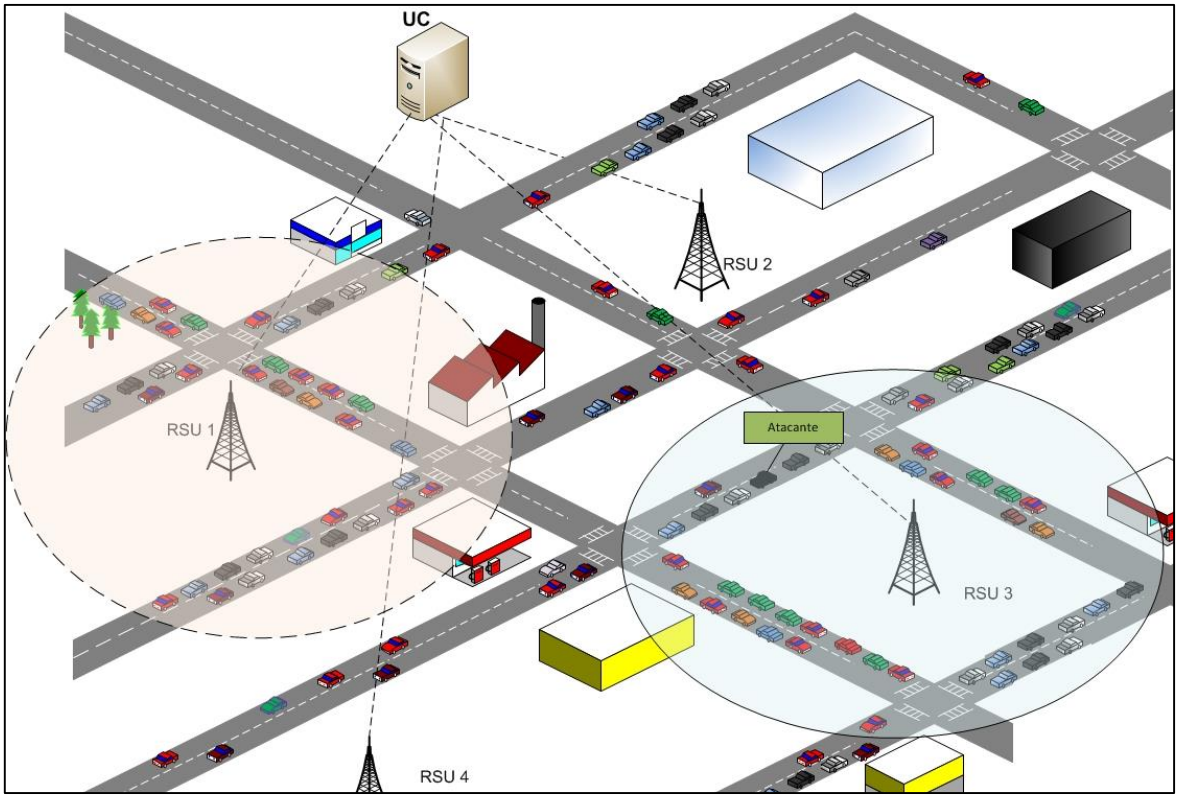
Tabela 2 - Dados da simulação do cenário 01.

Tempo da simulação	60 segundos
Início do ataque	a partir de 10 seg.
Quantidade de veículos	13
ID do veículo atacante	28
Quantidade de RSU	1

6.3.2 - Cenário 2

Considerar-se-á agora, que o mesmo veículo atacante se movimentou para outra região, a Região 3 (Figura 22). Assim que chega na Região 3, o mesmo veículo atacante, de ID 28, inicia um novo ataque, enviando um alerta falso novamente. Porém, para este cenário considera-se que após o primeiro ataque este veículo foi avaliado negativamente e, portanto, está com uma reputação baixa na UC, considerando que a RSU1 repassou a informação de reputação.

Figura 22 - Veículo atacante na Região 3.



Fonte: Elaborada pelo autor

Neste estudo de caso os parâmetros utilizados serão os mesmos do anterior. Quando os veículos da Região 3 recebem a mensagem, estes irão consultar a RSU 3 para obter informação da reputação do veículo de ID 28. No primeiro momento a RSU 3 não contém esta informação e, portanto, consulta a sua UC. Como a UC foi atualizada anteriormente ela contém a informação em questão e a retorna para a RSU 3 que consequentemente repassa para os veículos. A Tabela 3 apresenta os parâmetros da simulação.

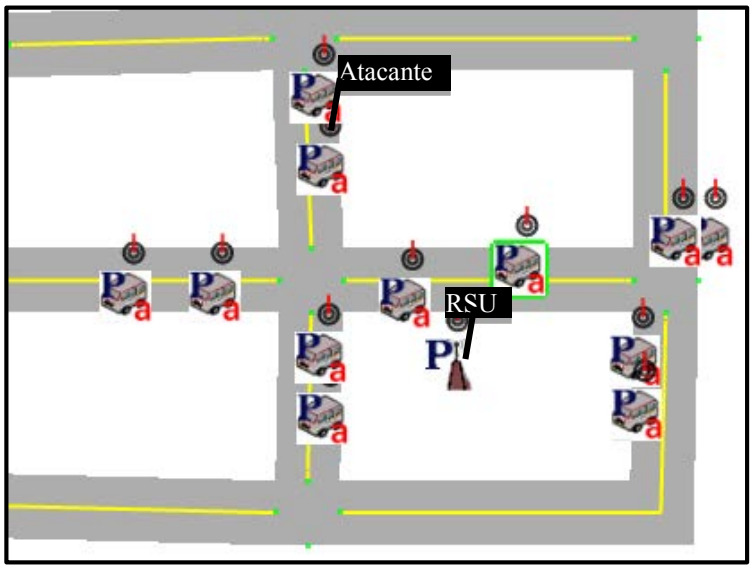
Tabela 3 - Dados da simulação do cenário 02

Tempo da simulação	60 segundos
Início do ataque	a partir de 10 seg.
Quantidade de veículos	12
ID do veículo atacante	28
Quantidade de RSU	1

Como o veículo atacante havia sido avaliado negativamente e, portanto, tem uma reputação baixa, espera-se que os veículos ao receberem a informação da reputação ignoram as

mensagens recebidas deste veículo e não alteram suas rotas e velocidades, tornando o ataque ineficiente. Na Figura 23 é apresentado este cenário simulado no NCTUns.

Figura 23 - Cenário 02 simulado no NCTUns.

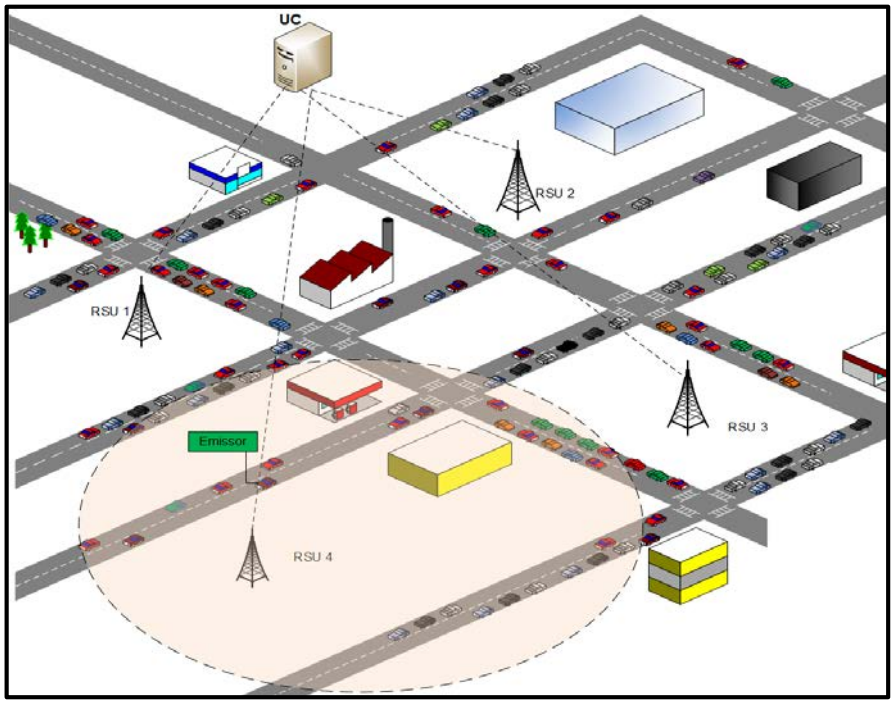


Fonte: Elaborada pelo autor

6.3.3 – Cenário 3

Neste estudo de caso é considerado um veículo com reputação alta, ou seja, é um veículo confiável. Este veículo emissor enviará uma mensagem de alerta verdadeira. Assim que os veículos recebem esta mensagem eles, seguindo a arquitetura de reputação centralizada, consultam a RSU para verificar a reputação do veículo emissor. O ambiente para este estudo de caso é apresentado na Figura 24.

Figura 24 - Ambiente para o cenário 03.



Fonte: Elaborada pelo autor

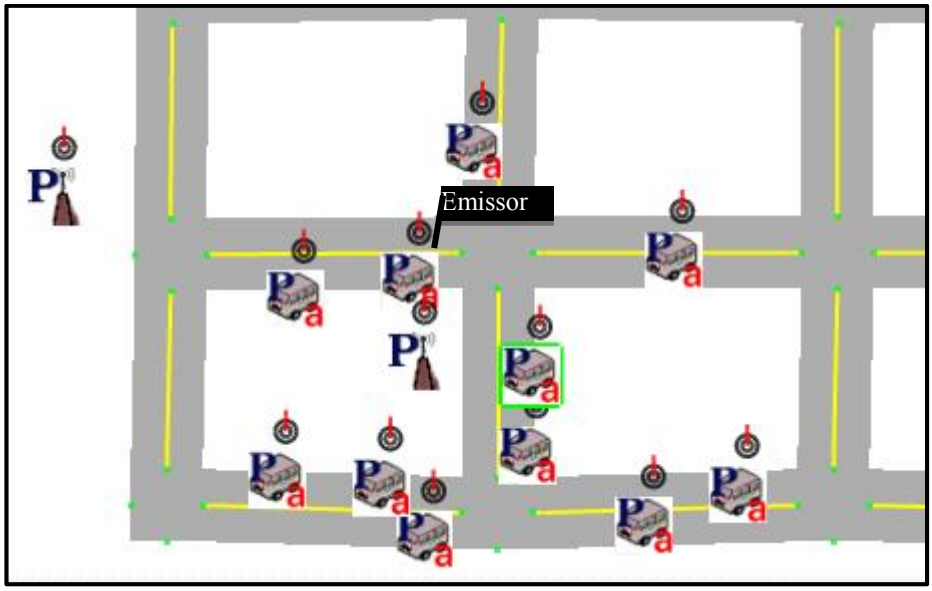
Neste caso o esperado é que após os veículos receberem o alerta e consultarem a reputação do veículo emissor que estes constatem que a reputação é alta e, portanto, confiem no alerta enviado, tomando as possíveis medidas.

Novamente os parâmetros da simulação são similares aos anteriormente apresentados, e são ilustrados na Tabela 4. O cenário simulado no NCTUns é ilustrado na Figura 25.

Tabela 4 - Dados da simulação do cenário 03.

Tempo da simulação	60 segundos
Início do alerta	a partir de 10 seg.
Quantidade de veículos	12
ID do veículo emissor	25
Quantidade de RSU	1

Figura 25 - Cenário 03 simulado no NCTUns.



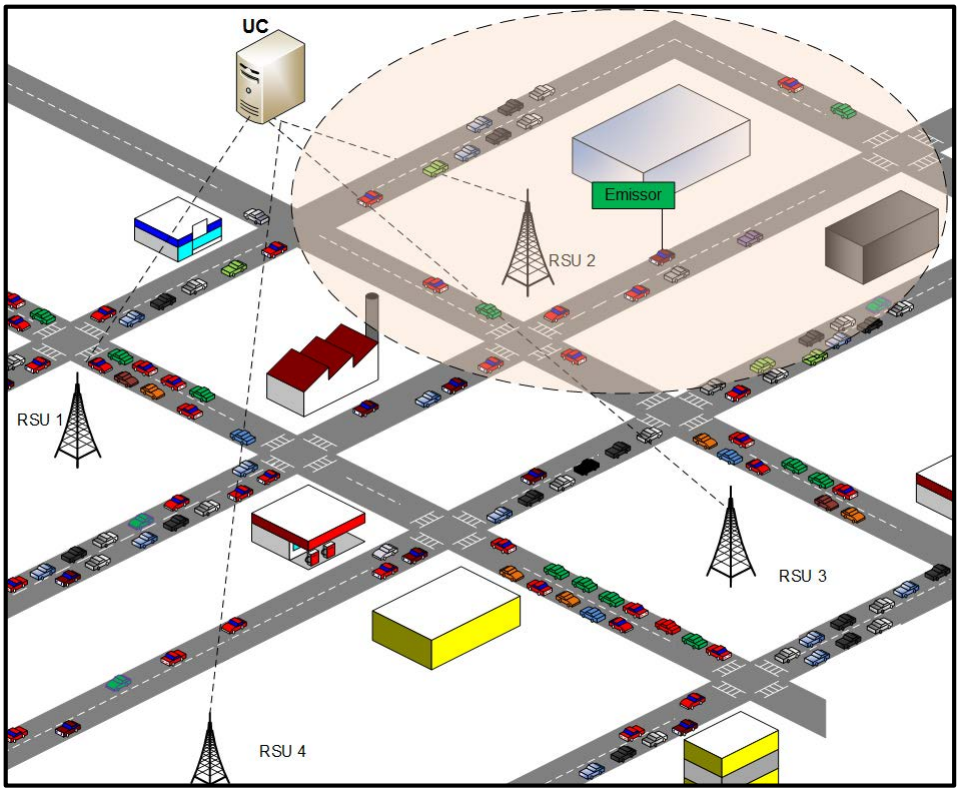
Fonte: Elaborada pelo autor

6.3.4 – Cenário 4

Para este estudo de caso considera-se que o mesmo veículo, do estudo anterior, de ID 25 e reputação alta, siga agora para a região 2, porém desta vez envia um alerta falso (Figura 26).

Neste caso os veículos ao receberem o alerta, consequentemente consultam a RSU2, que por consequência consulta a UC que retorna uma reputação alta, a RSU2, portanto, retorna a reputação aos veículos. Espera-se que os veículos ao receberem um valor de reputação alto consequentemente confiem no alerta enviado pelo veículo de ID 25 e alterem suas rotas e velocidades.

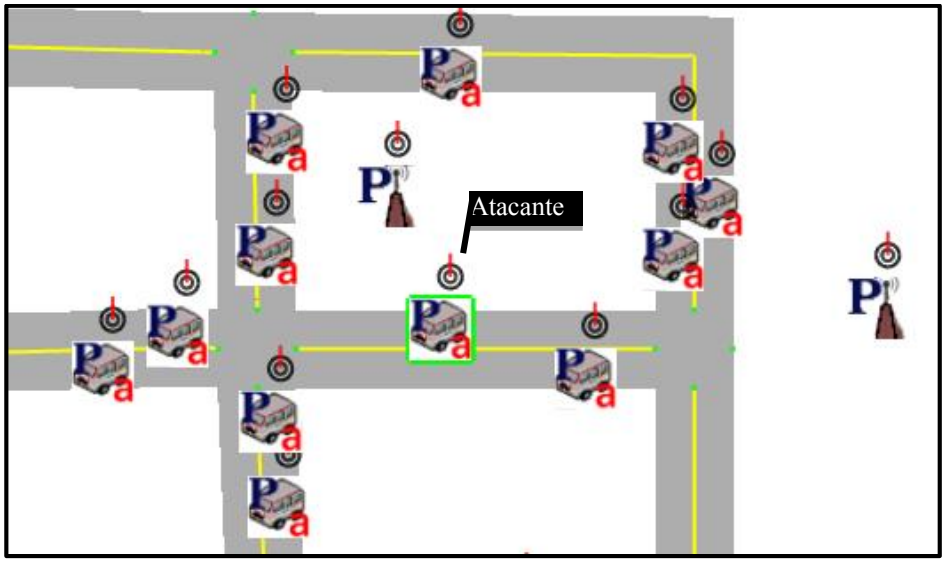
Figura 26 - Ambiente para o cenário 04.



Fonte: Elaborada pelo autor

Na Figura 27 é apresentado o cenário 04 simulado no NCTUns. Como nos testes anteriores os parâmetros da simulação são similares e são ilustrados na Tabela 5.

Figura 27 - Cenário 04 simulado no NCTUns.



Fonte: Elaborada pelo autor

Tabela 5 - Dados da simulação do cenário 04.

Tempo da simulação	60 segundos
Início do alerta	a partir de 10 seg.
Quantidade de veículos	12
ID do veículo atacante	25
Quantidade de RSU	1

6.4 – Resultados das Simulações

Nesta seção são apresentados os resultados das simulações dos cenários descritos na seção anterior. Como foi descrito anteriormente foi realizado quatro simulações para quatro cenários diferentes e sendo assim os resultados estão divididos em quatro seções.

Com os resultados obtidos nas simulações é possível verificar as vantagens e desvantagens de utilizar um sistema de reputação centralizado.

6.4.1 – Cenário 01

Neste cenário foi simulado um ambiente com um veículo de reputação desconhecida enviando um alerta falso. Os dados coletados foram tabelados e apresentados em figuras.

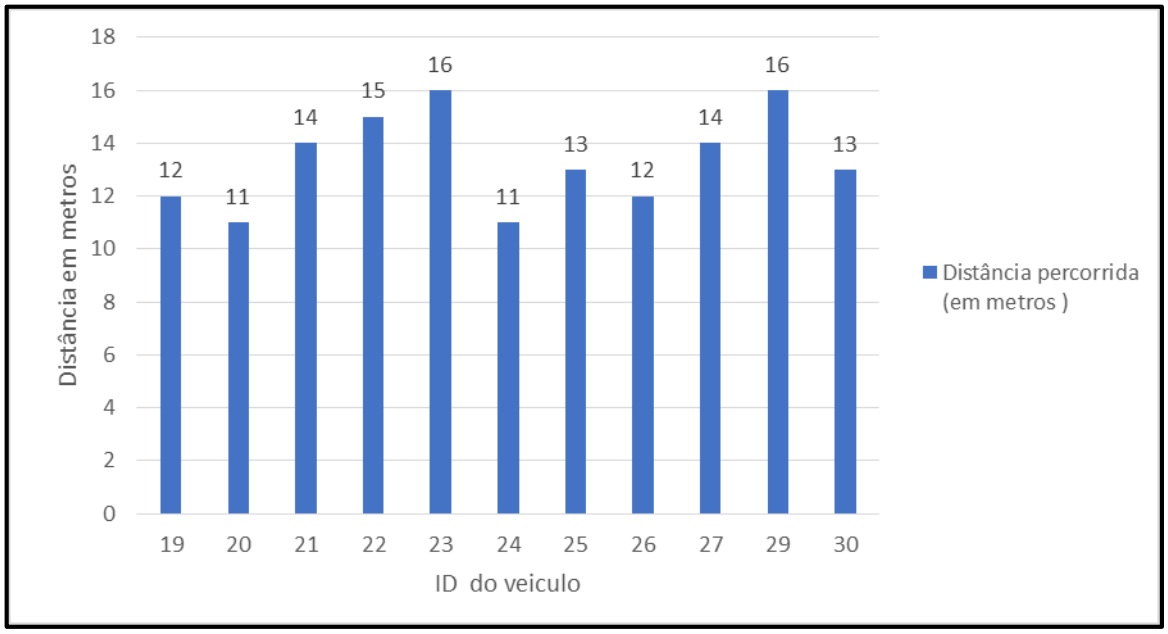
Os primeiros dados apresentados são as distâncias percorridas entre os veículos do momento em que eles recebem o alerta até o momento em que eles recebem a resposta da RSU sobre a reputação.

Na Tabela 6 são apresentadas as posições dos veículos no momento que recebem a mensagem e consultam a RSU e o momento que recebem a resposta. Na Figura 28 é ilustrado o gráfico com estes valores, para uma melhor comparação entre as distâncias percorridas.

Tabela 6 - Distância percorrida pelos veículo no cenário 01.

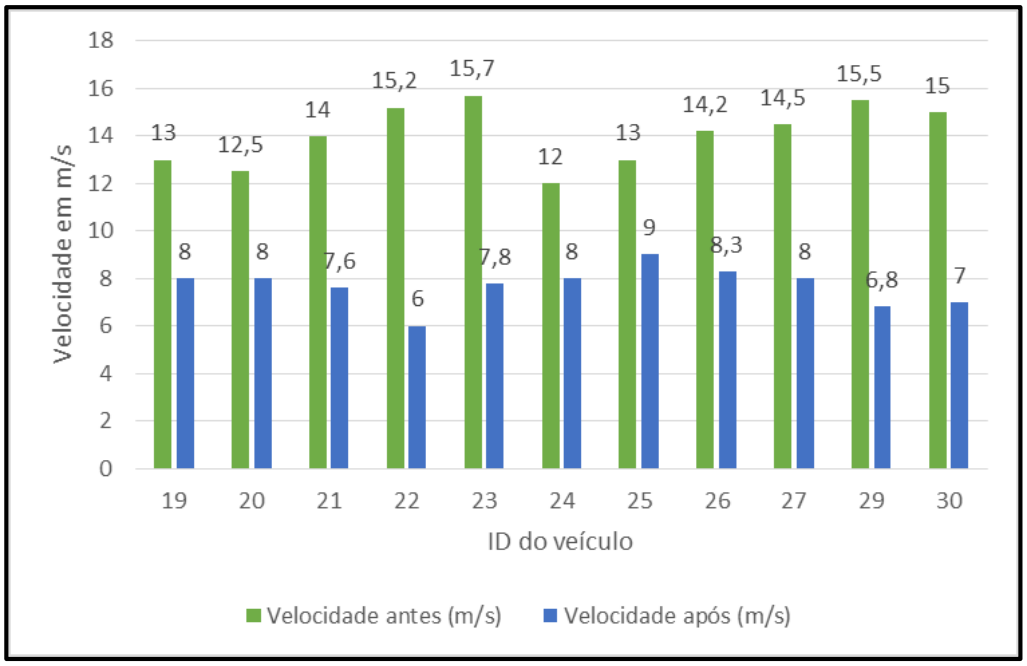
ID do Veículo	Posição antes	Posição após	Distância percorrida (em metros
19	(506 , 137)	(506 , 149)	12
20	(371, 187)	(371, 198)	11
21	(527, 95)	(527, 109)	14
22	(507, 101)	(507, 116)	15
23	(527, 255)	(527, 271)	16
24	(507, 254)	(507, 265)	11
25	(417, 184)	(430, 184)	13
26	(449, 203)	(461, 203)	12
27	(670, 185)	(684, 185)	14
29	(682, 110)	(682, 126)	16
30	(556, 163)	(569, 163)	13

Figura 28 - Gráfico das distâncias percorridas pelos veículos no cenário 01



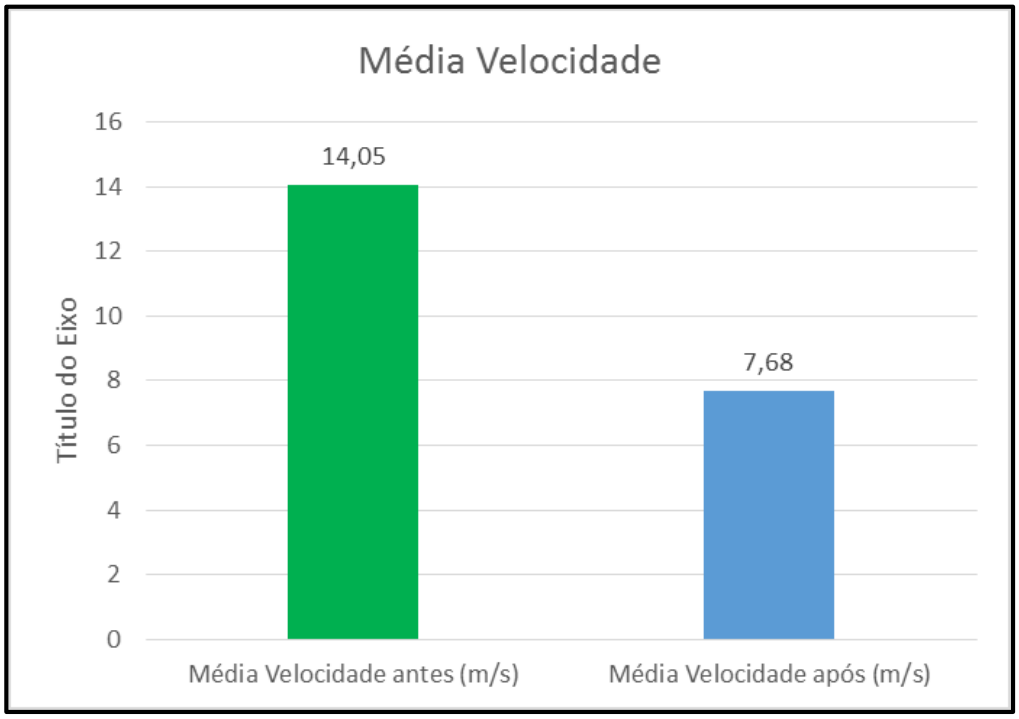
Os dados apresentados na Figura 30 ilustram as velocidades dos veículos antes de receberem a mensagem de alerta e consultarem a RSU e após receberem a resposta da mesma.

Figura 29 - Gráfico das velocidades antes e depois de receberem alerta no cenário 01.



Na Figura 30 é apresentado o gráfico com as médias das velocidades antes e depois dos veículos receberem o alerta.

Figura 30 - Gráfico da média de velocidade no cenário 01.

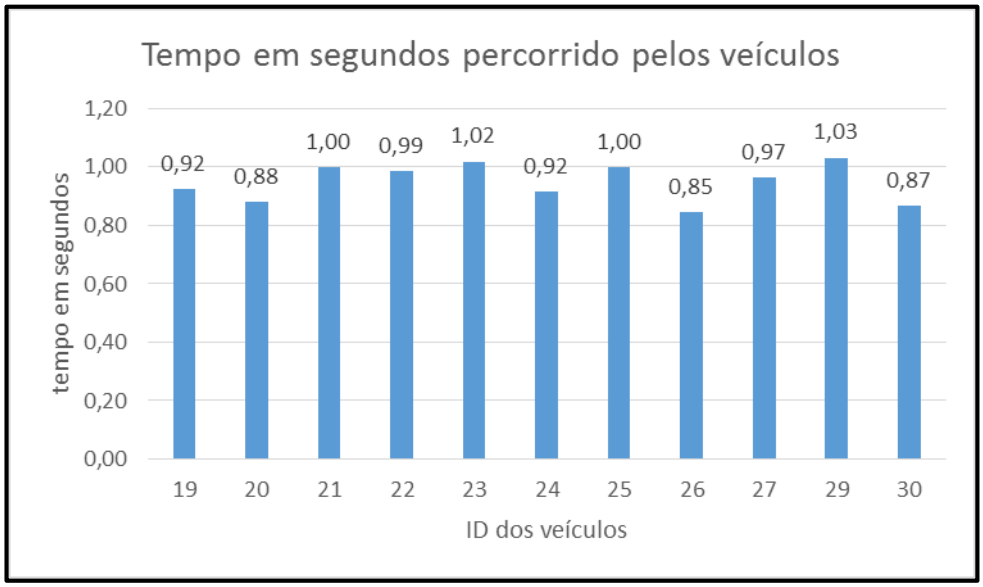


Na Tabela 7 são ilustrados os resultados da equação citada na seção 5.2 aplicada nas distâncias e velocidades dos veículos durante a simulação. Na Figura 31 é ilustrado o gráfico desses valores.

Tabela 7 - Tempo em segundos percorrido pelos veículos no cenário 01.

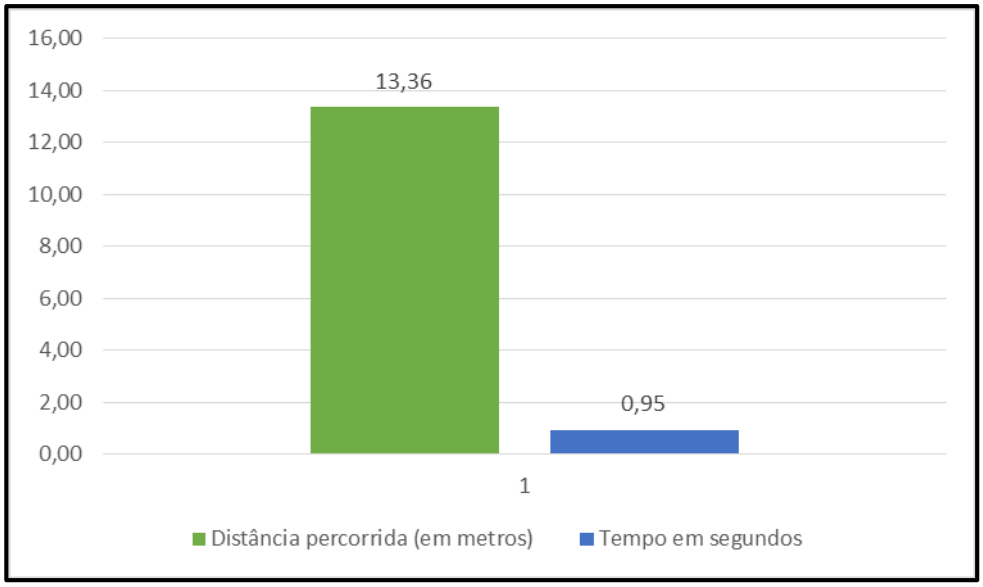
ID do veículo	Tempo em segundos
19	0,92
20	0,88
21	1,00
22	0,99
23	1,02
24	0,92
25	1,00
26	0,85
27	0,97
29	1,03
30	0,87

Figura 31 - Gráfico dos tempos percorrido pelos veículos no cenário 01.



Na Figura 32 são apresentados a média das distâncias e o tempo percorrido pelos veículos durante a simulação do cenário 01.

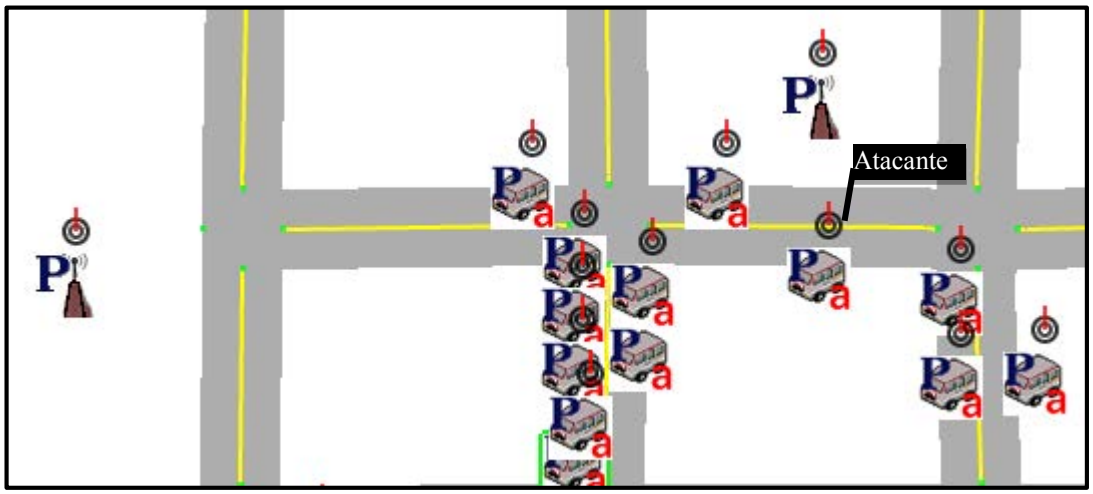
Figura 32 - Gráfico das médias de distância e tempo percorrido pelos veículos no cenário 01.



Na Figura 33 é apresentado o posicionamento dos veículos ao final da simulação. Partindo do pressuposto que o objetivo do ataque era enviar um alerta falso e liberar a pista para o veículo malicioso, pode-se considerar que o ataque foi realizado com sucesso. Na Figura 36 é possível verificar um aglomerado de veículos em uma determinada rua do mapa, enquanto a rua em que o veículo malicioso se encontra está mais liberada.

Este aglomerado ocorre devido a troca de direção dos veículos para buscarem uma nova rota e a redução de velocidade verificada nos dados apresentados.

Figura 33 - Posição dos veículos após o final da simulação.



Deve se considerar que neste cenário os veículos não obtiveram uma resposta sobre reputação do veículo emissor do alerta (ou melhor, a resposta era de que não se sabia nada sobre o mesmo), desta maneira optaram por buscar uma nova rota e diminuir a velocidade para evitar outro acidente.

6.4.2 – Cenário 02

Neste cenário foi simulado um ambiente em que o mesmo veículo que enviou o alerta falso, se movimentoU para outra região. Portanto, neste cenário as RSUs têm informações sobre a reputação deste veículo, pois foi levado em consideração que os demais veículos o avaliaram negativamente. Os dados coletados foram tabelados e apresentados em figuras.

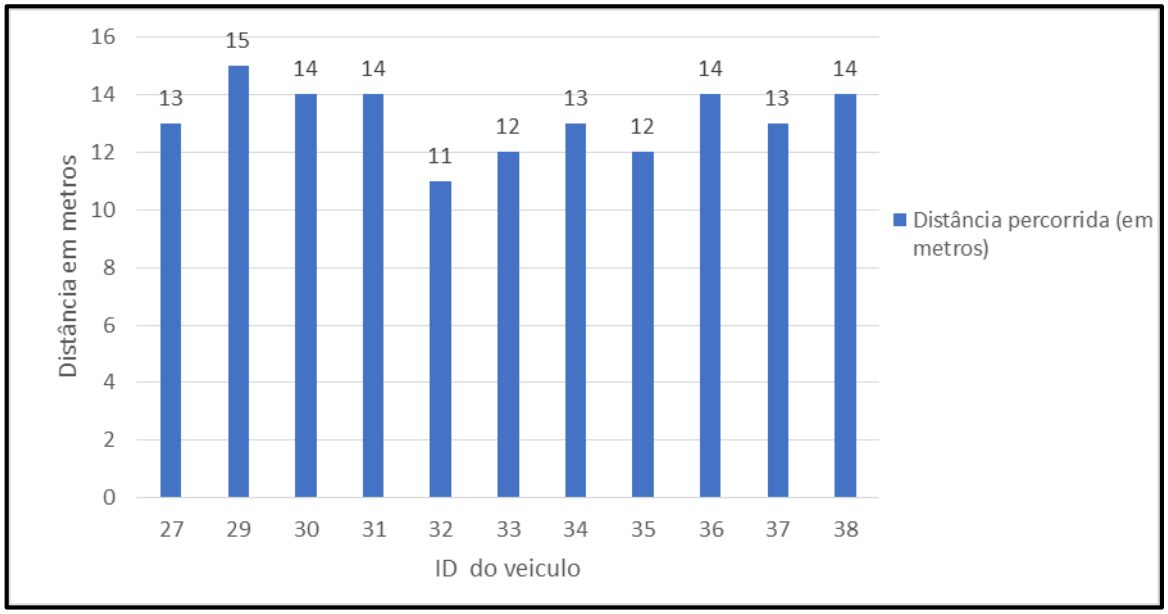
Os primeiros dados apresentados são as distâncias percorridas entre os veículos a partir do momento em que eles recebem o alerta até o momento em que eles recebem a resposta da RSU sobre a reputação do veículo que enviou o alerta.

Na Tabela 8 são mostradas as posições dos veículos no momento que recebem a mensagem e consultam a RSU e o momento que recebem a resposta. Na Figura 34 é ilustrado o gráfico com estes valores, para uma melhor comparação entre as distâncias percorridas.

Tabela 8 - Distância percorrida pelos veículo no cenário 02.

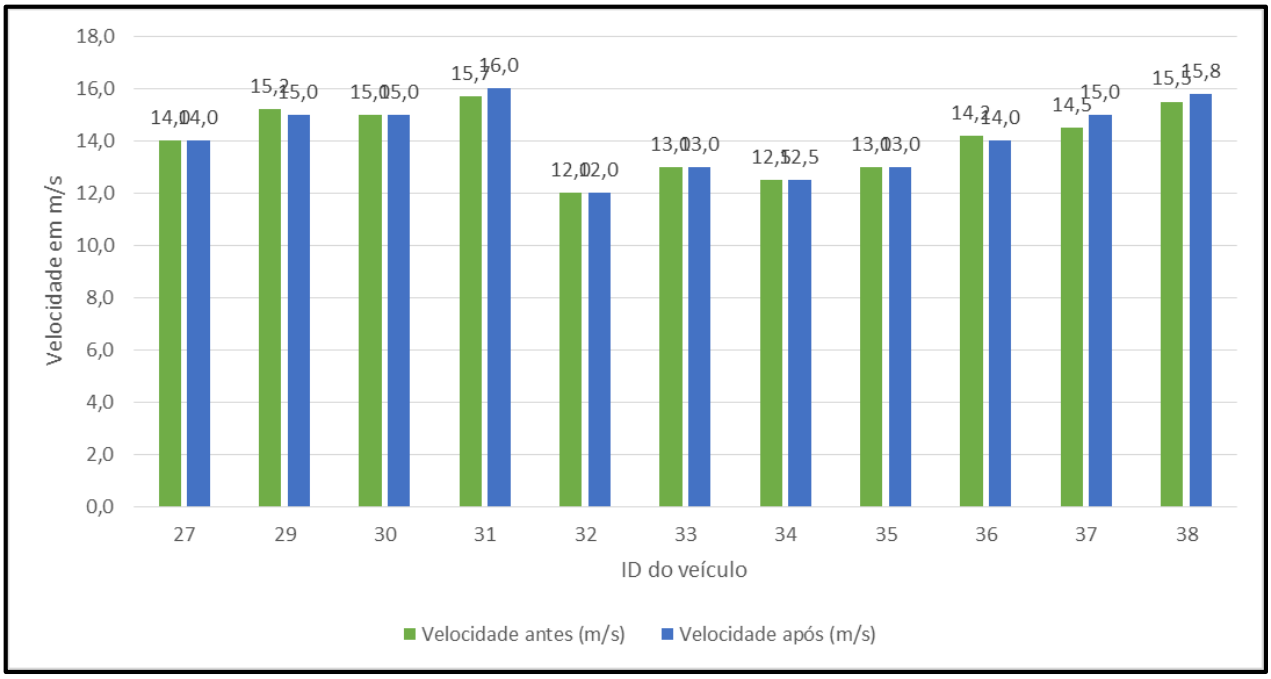
ID do Veículo	Posição antes	Posição após	Distância percorrida (em metros
27	(846, 336)	(859, 336)	13
29	(1126, 293)	(1141, 293)	15
30	(908, 384)	(908, 498)	14
31	(813, 340)	(813, 354)	14
32	(906 , 256)	(917, 256)	11
33	(912, 385)	(924, 385)	12
34	(1105, 348)	(1105, 361)	13
35	(1101, 348)	(1101, 360)	12
36	(1130, 265)	(1144, 265)	14
37	(906, 227)	(919, 227)	13
38	(964, 308)	(978, 308)	14

Figura 34 - Gráfico das distâncias percorridas pelos veículos no cenário 02.



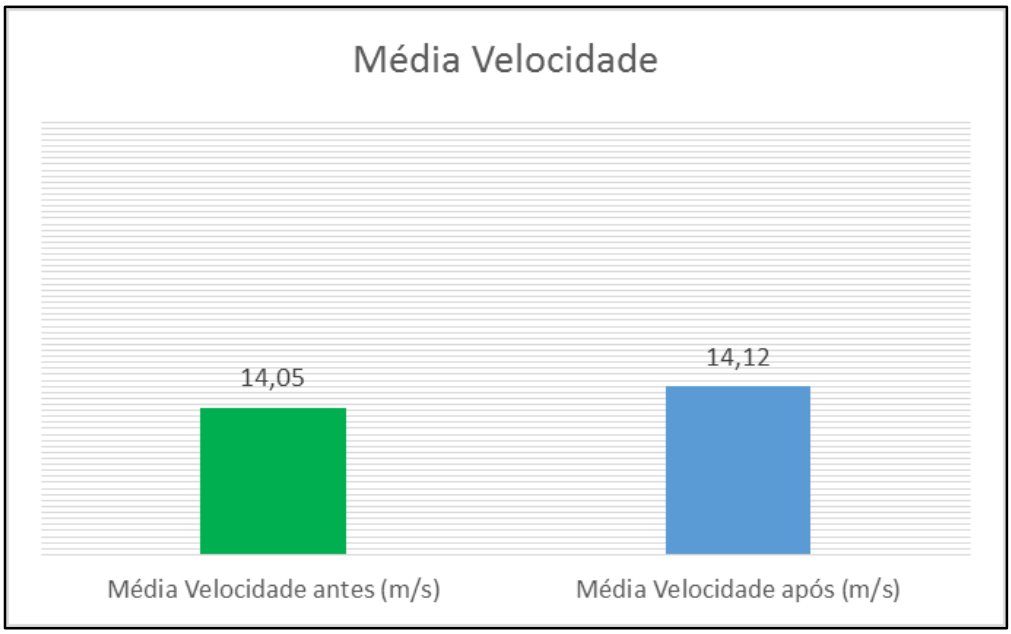
Os dados apresentados na Figura 35 representam as velocidades dos veículos antes de receberem a mensagem de alerta e consultarem a RSU e após receberem a resposta da mesma.

Figura 35 - Gráfico das velocidades antes e depois de receberem alerta no cenário 02.



Na Figura 36 é mostrado o gráfico com as médias das velocidades antes e depois dos veículos receberem o alerta.

Figura 36 - Gráfico da média de velocidade no cenário 02.

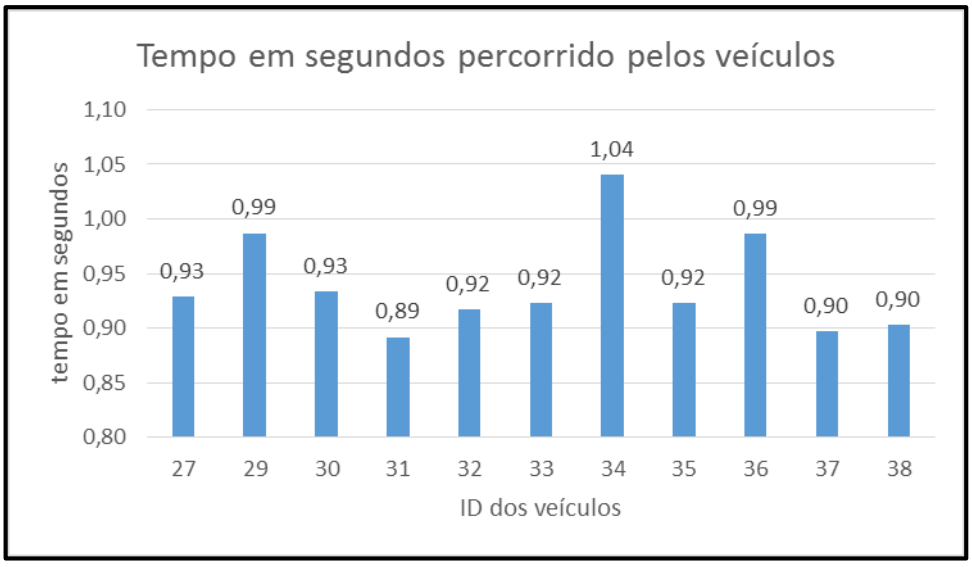


Na Tabela 9 são apresentados os tempos que os veículos percorrem durante a simulação. Na Figura 37 é ilustrado o gráfico desses valores.

Tabela 9 - Tempos em segundos percorridos pelos veículos no cenário 02.

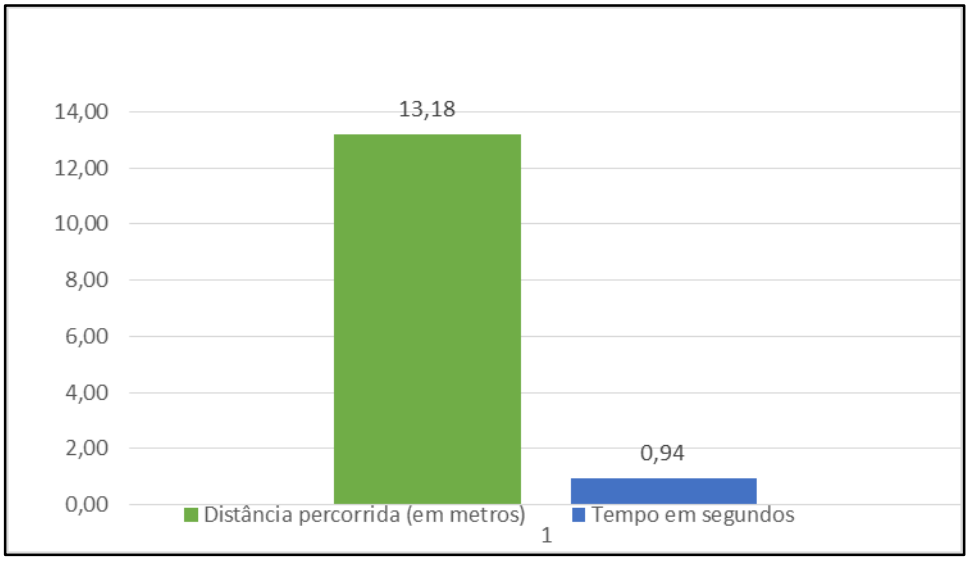
ID do veículo	Tempo em segundos
27	0,93
29	0,99
30	0,93
31	0,89
32	0,92
33	0,92
34	1,04
35	0,92
36	0,99
37	0,90
38	0,90

Figura 37 - Gráfico dos tempos percorridos pelos veículos no cenário 02.



Na Figura 38 é ilustrada a média das distâncias e tempo percorrido pelos veículos durante a simulação do cenário 02.

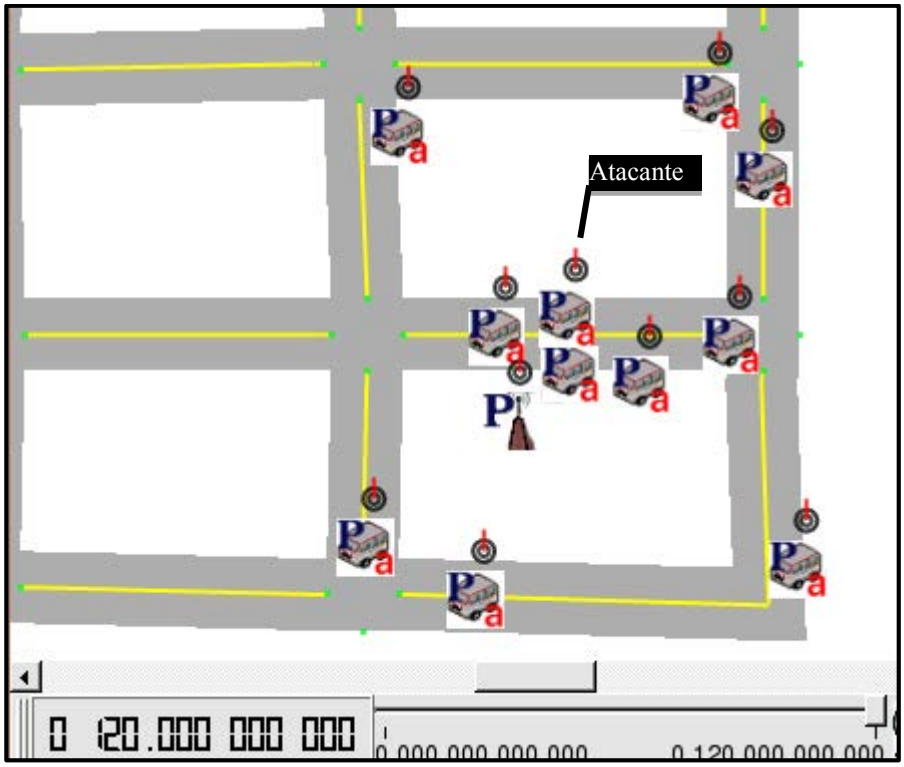
Figura 38 - Gráfico das médias de distância e tempo percorrido pelos veículos no cenário 02.



Na Figura 39 é ilustrado o posicionamento dos veículos ao final da simulação. Partindo do pressuposto que o objetivo do ataque era enviar um alerta falso e liberar a pista para o veículo malicioso, pode-se considerar um ataque com insucesso. Na Figura 42 é possível verificar que nesta simulação os veículos ficaram mais dispersos, pois desta vez os demais veículos não tentaram evitar a rua de onde partiu o alerta.

Para este caso os veículos ignoraram a mensagem de alerta e seguiram seus percursos normalmente sem alterarem rotas ou velocidades.

Figura 39 - Posição dos veículos após o final da simulação do cenário 02.



Deve-se considerar que neste cenário os veículos obtiveram uma resposta positiva com relação à reputação do veículo emissor do alerta, como a reputação era baixa os demais veículos ignoraram a mensagem.

6.4.3 – Cenário 03

Neste cenário foi simulado um ambiente em que o veículo que envia um alerta verdadeiro tem sua reputação alta, ou seja, é confiável.

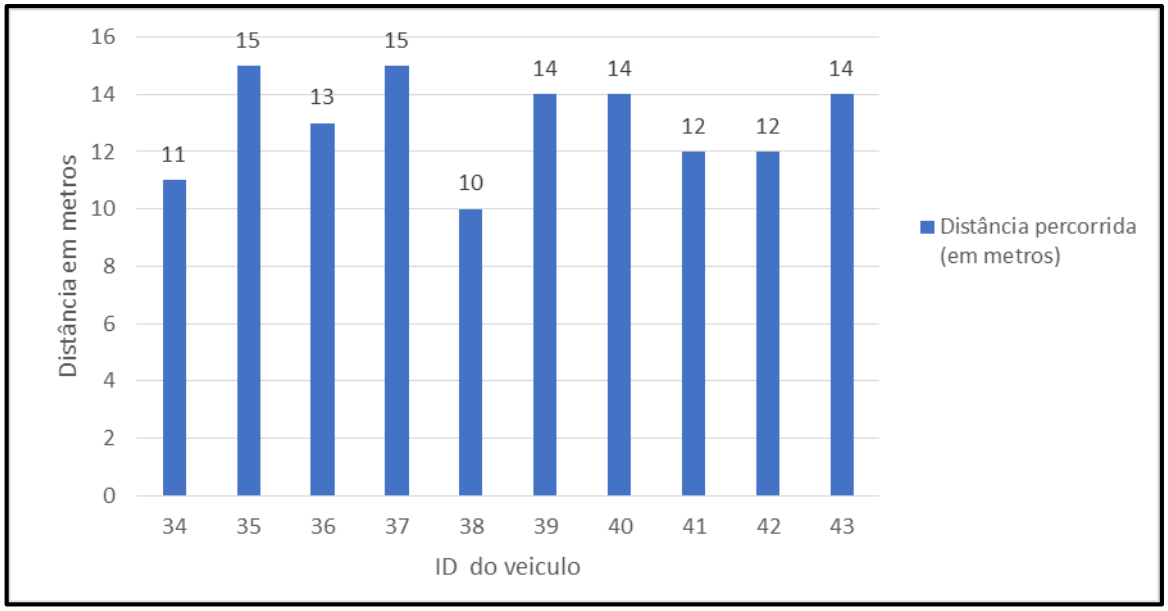
Os primeiros dados apresentados são as distâncias percorridas entre os veículos do momento em que eles recebem o alerta até o momento em que eles recebem a resposta da RSU sobre a reputação.

Na Tabela 10 são mostradas as posições dos veículos a partir do momento que recebem a mensagem e consultam a RSU e o momento que recebem a resposta. Na Figura 40 é ilustrado o gráfico com estes valores, para uma melhor comparação entre as distâncias percorridas.

Tabela 10 - Distância percossida pelos veículo no cenário 03.

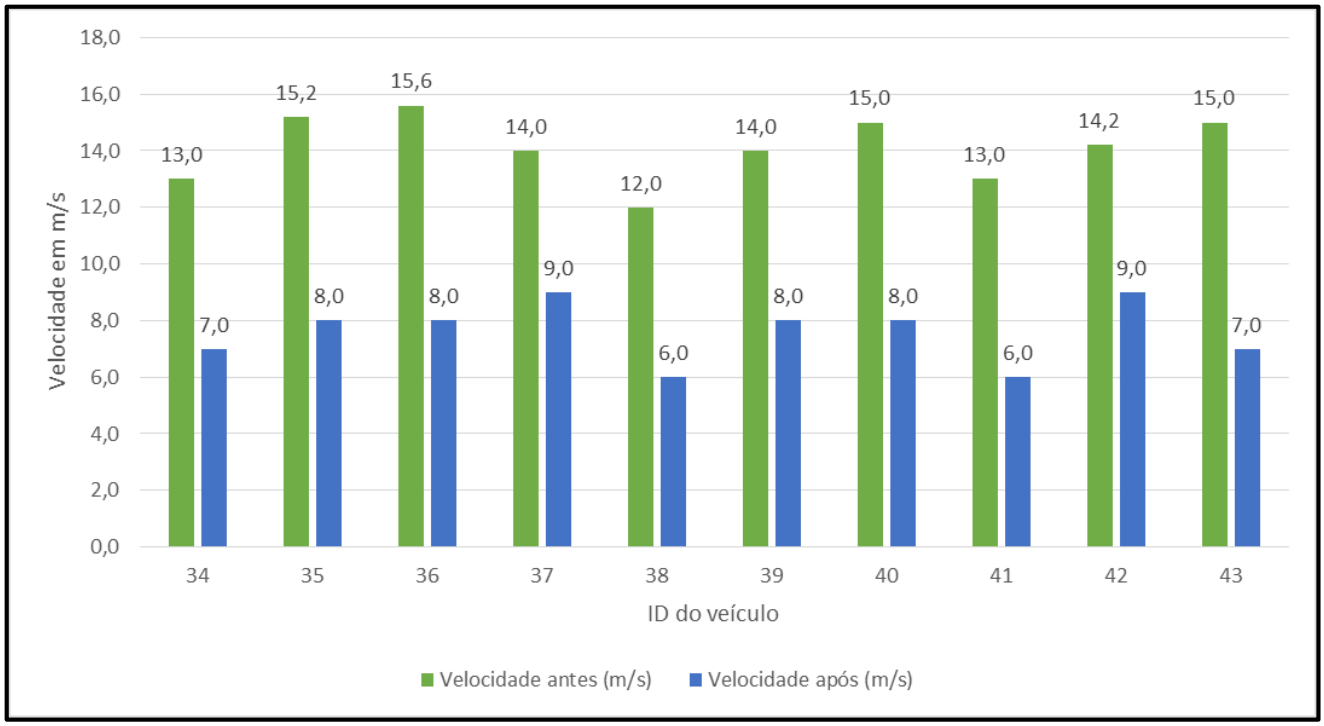
ID do Veículo	Posição antes	Posição após	Distância percorrida (em metros
34	(625, 448)	(636, 448)	11
35	(453, 482)	(453, 597)	15
36	(484, 250)	(484, 263)	13
37	(514, 376)	(529, 376)	15
38	(387, 342)	(397, 342)	10
39	(443, 342)	(443, 356)	14
40	(513, 414)	(527, 414)	14
41	(433, 458)	(445, 458)	12
42	(560, 477)	(572, 477)	12
43	(368, 451)	(368, 465)	14

Figura 40 - Gráfico das distâncias percorridas pelos veículos no cenário 03.



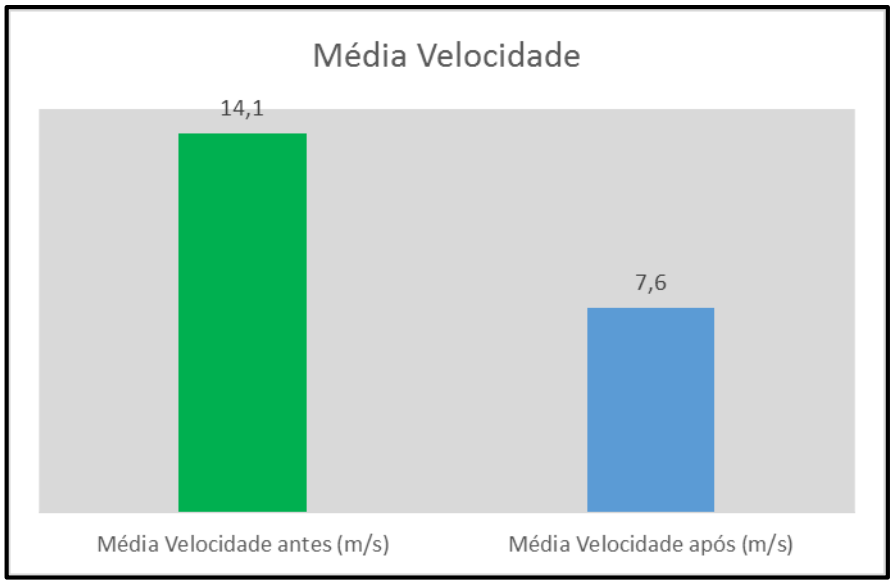
Os dados apresentados na Figura 41 representam as velocidades dos veículos antes de receberem a mensagem de alerta e consultarem a RSU e após receberem a resposta da mesma.

Figura 41 - Gráfico das velocidades antes e depois de receberem alerta no cenário 03.



Na Figura 42 é ilustrado o gráfico com as médias das velocidades antes e depois dos veículos receberem o alerta.

Figura 42 - Gráfico da média de velocidade no cenário 03.

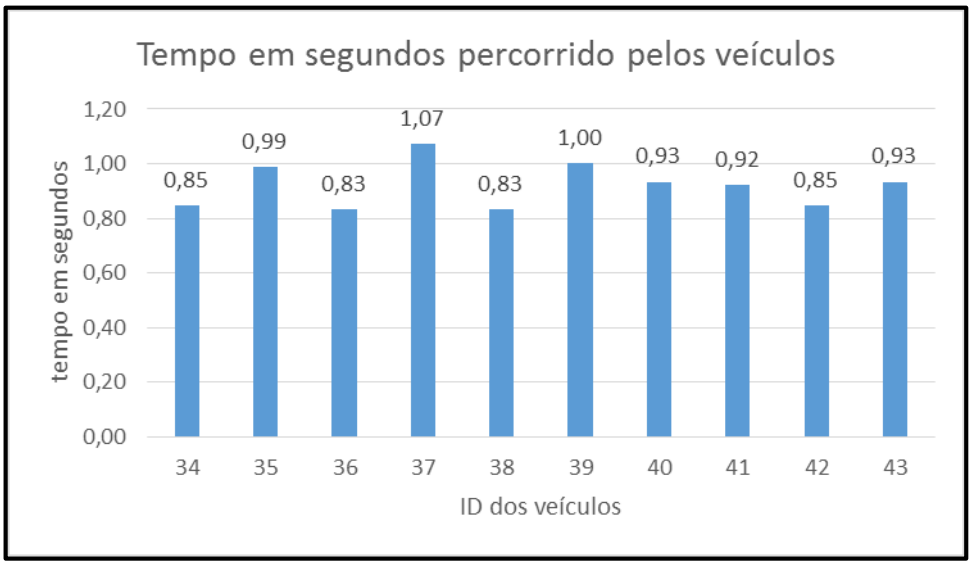


Na Tabela 11 são mostrados os resultados da equação 1 citada na seção 5.2 aplicada nas distâncias e velocidades dos veículos durante a simulação. Na Figura 43 é ilustrado o gráfico desses valores.

Tabela 11 - Tempo em segundos percorrido pelos veículos no cenário 03.

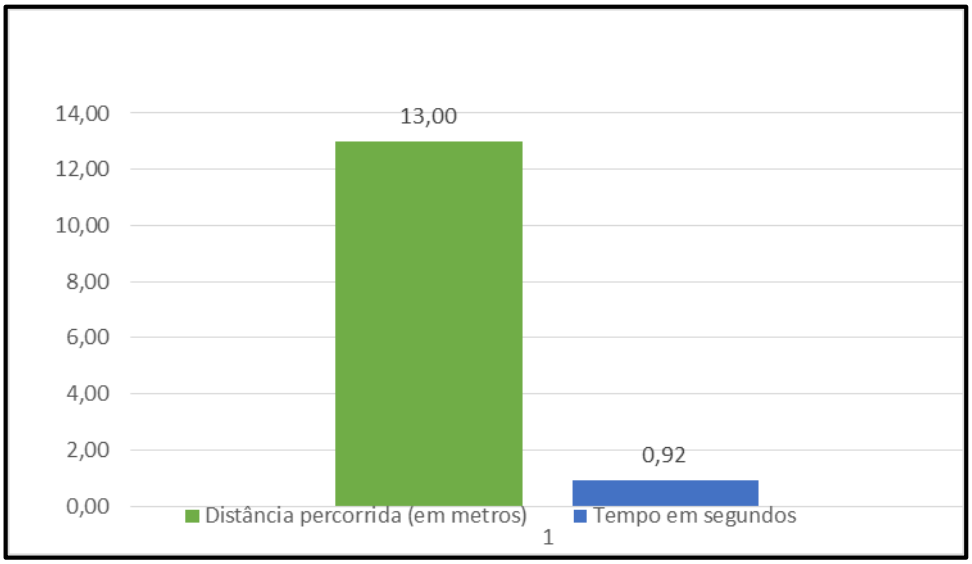
ID do veículo	Tempo em segundos
34	0,85
35	0,99
36	0,83
37	1,07
38	0,83
39	1,00
40	0,93
41	0,92
42	0,85
43	0,93

Figura 43 - Gráfico dos tempos percorrido pelos veículos no cenário 03.



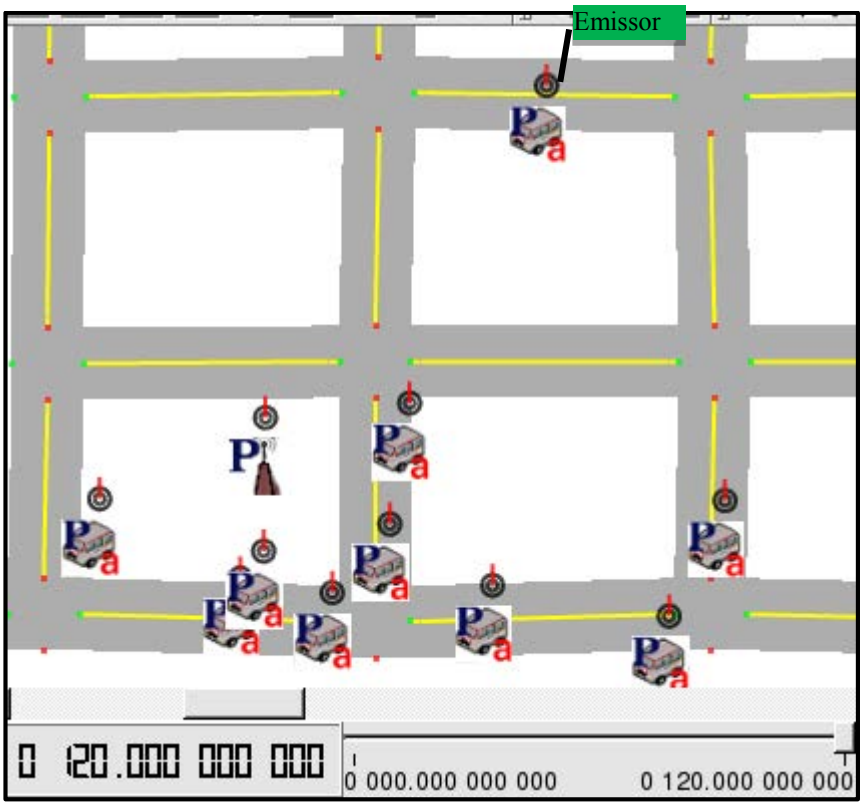
Na Figura 44 são mostradas a média das distâncias e tempo percorrido pelos veículos durante a simulação do cenário 03.

Figura 44 - Gráfico das médias de distância e tempo percorrido pelos veículos no cenário 03.



A Figura 45 ilustra os posicionamentos dos veículos ao final da simulação. Partindo do pressuposto que o objetivo do veículo era informar aos demais veículos sobre algum acontecimento na rua, para que esses pudessem tomar determinadas medidas. Na Figura 44 é possível verificar que nesta simulação os veículos evitaram a rua em que se originou o alerta.

Figura 45 - Posição dos veículos após o final da simulação do cenário 03



O resultado desta simulação é similar ao da simulação do cenário 01, em que a reputação era desconhecida. Deve-se considerar que neste cenário os veículos obtiveram uma resposta com a reputação do veículo emissor do alerta, como a reputação era alta os demais veículos confiaram na mensagem e alteraram suas rotas e velocidades, conforme o esperado.

6.4.4 – Cenário 04

Neste cenário foi simulado um ambiente em que o mesmo veículo que enviou o alerta verdadeiro anterior, agora se movimento para outra região. Porém neste caso este veículo enviou um alerta falso.

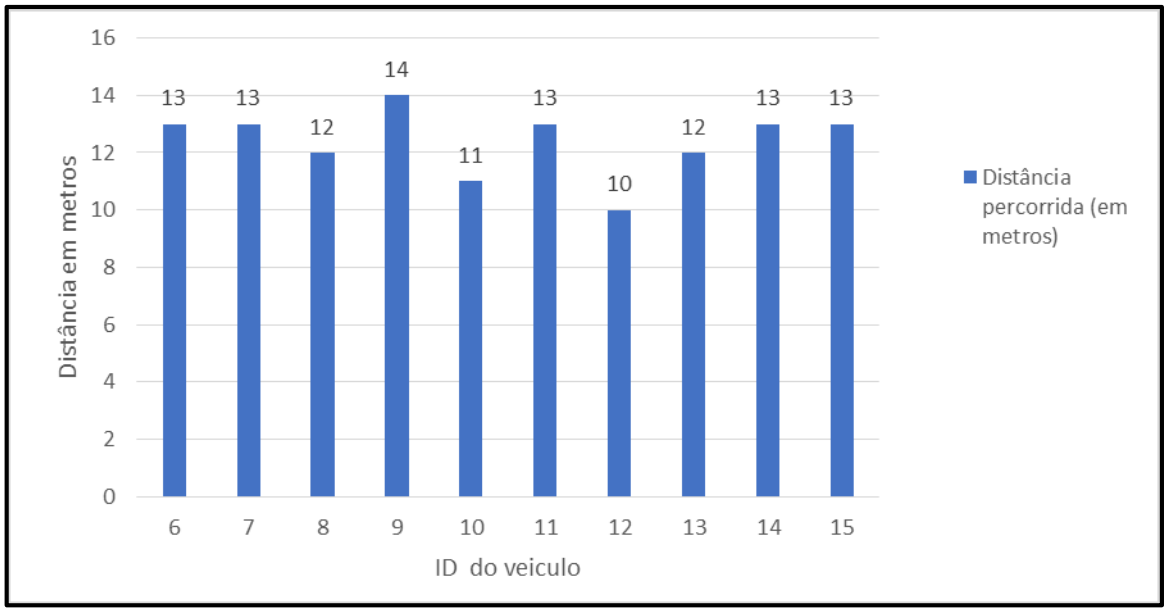
Os primeiros dados apresentados são as distâncias percorridas pelos veículos a partir do momento em que eles recebem o alerta até o momento em que eles recebem a resposta da RSU sobre a reputação.

Na Tabela 12 são mostradas as posições dos veículos no momento que recebem a mensagem e consultam a RSU e o momento que recebem a resposta. Na Figura 46 é ilustrado o gráfico com estes valores, para uma melhor comparação entre as distâncias percorridas.

Tabela 12 - Distância percossida pelos veículo no cenário 04.

ID do Veículo	Posição antes	Posição após	Distância percorrida (em metros
6	(840, 311)	(853, 311)	13
7	(783, 331)	(896, 331)	13
8	(914, 416)	(914, 428)	12
9	(914, 372)	(914, 386)	14
10	(904, 228)	(904, 239)	11
11	(904, 272)	(904, 285)	13
12	(1131, 258)	(1141, 258)	10
13	(1033, 336)	(1045, 336)	12
14	(1100, 223)	(1100, 236)	13
15	(1099, 263)	(1099, 276)	13

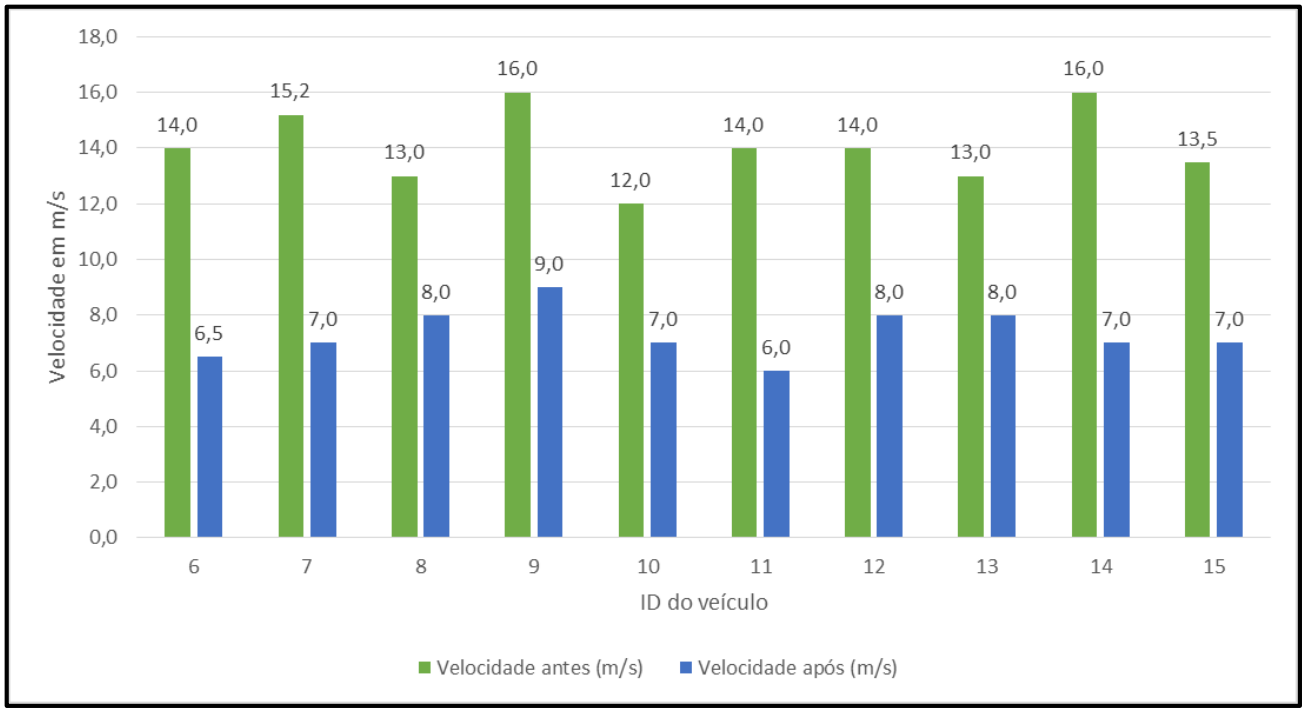
Figura 46 - Gráfico das distâncias percorridas pelos veículos no cenário 04.



Os dados apresentados representam as velocidades dos veículos antes de receberem a mensagem de alerta e consultarem a RSU e após receberem a resposta da mesma.

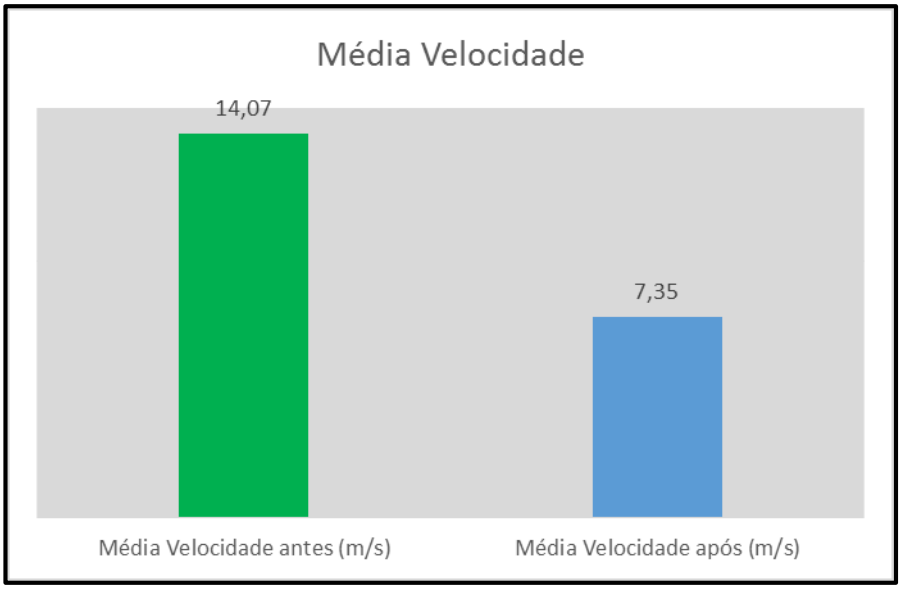
Na Figura 47 é ilustrado o gráfico com as comparações das velocidades de cada veículo da simulação.

Figura 47 - Gráfico das velocidades dos veículos antes e depois de receberem alerta no cenário 04.



Na Figura 48 é ilustrado o gráfico com as médias das velocidades antes e depois dos veículos receberem o alerta.

Figura 48 - Gráfico da média de velocidade no cenário 04.

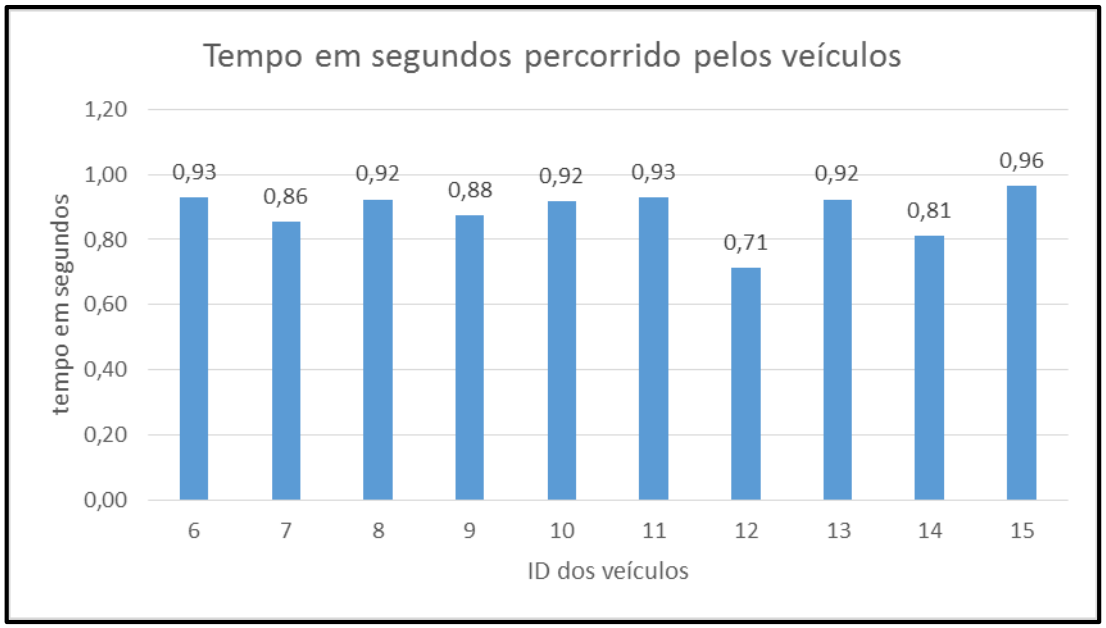


Na Tabela 13 são mostrados o resultado da equação 1, citada na seção 5.2 aplicada nas distâncias e velocidades dos veículos durante a simulação. Na Figura 49 é apresentado o gráfico desses valores.

Tabela 13 - Tempo em segundos percorrido pelos veículos no cenário 04

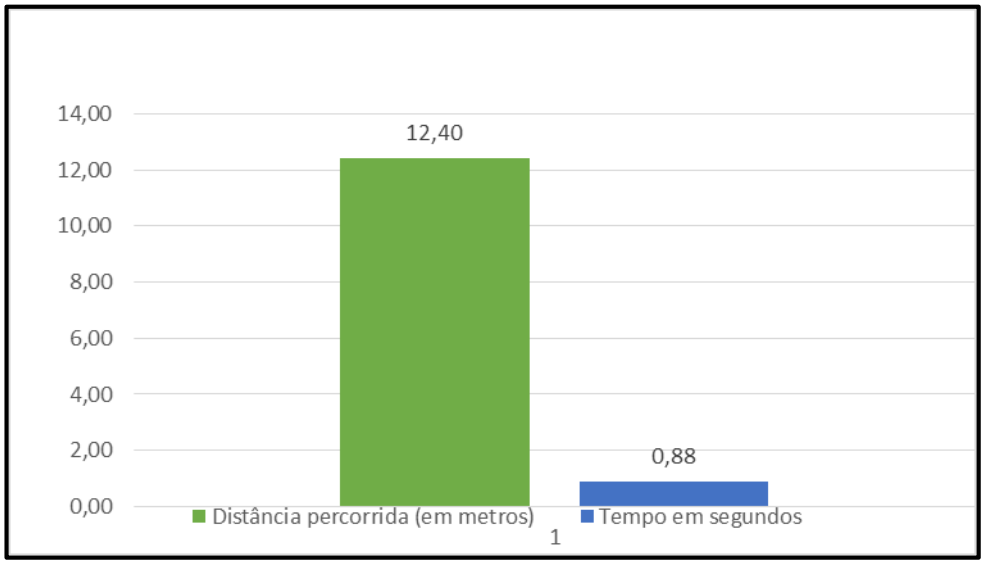
ID do veículo	Tempo em segundos
6	0,93
7	0,86
8	0,92
9	0,88
10	0,92
11	0,93
12	0,71
13	0,92
14	0,81
15	0,96

Figura 49 - Gráfico dos tempos percorrido pelos veículos no cenário 04.



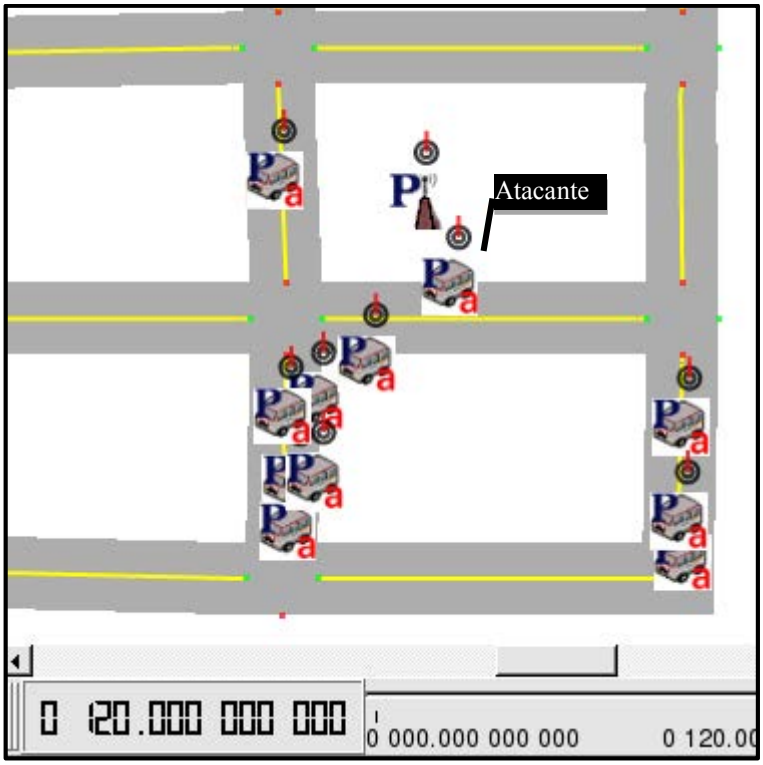
Na Figura 50 são mostrados as médias das distâncias e dos tempos percorridos pelos veículos durante a simulação do cenário 04.

Figura 50 - Gráfico das médias de distância e tempo percorrido pelos veículos no cenário 04.



Na Figura 51 são mostrados os posicionamentos dos veículos ao final da simulação. Partindo do pressuposto que o objetivo do veículo era informar aos demais veículos sobre um falso acontecimento na rua para que esses pudessem tomar determinadas medidas. Nesta figura também é possível verificar que nesta simulação os veículos evitaram a rua em que se originou o falso alerta.

Figura 51 - Posição dos veículos após o final da simulação do cenário 04.



O resultado desta simulação é similar ao da simulação do cenário 03. Deve se considerar que neste cenário os veículos obtiveram uma resposta com a reputação do veículo emissor do alerta, como a reputação era alta os demais veículos confiaram na mensagem, apesar de ser falsa, e alteraram suas rotas e velocidades. Esta é uma situação crítica, que é discutida na próxima sessão.

6.5– Considerações Finais

Nesta seção são discutidos os resultados das simulações efetuadas.

No cenário 01 o ataque pode ser considerado um ataque bem-sucedido, pois o atacante conseguiu o que queria, uma rodovia livre. Isto se deu pelo fato de sua reputação ser desconhecida, e para este caso neste trabalho optou-se por fazer os veículos confiarem nas mensagens, o que levou ao resultado apresentado. Entretanto este veículo não teria sucesso nos ataques posteriores, pois foi avaliado negativamente e como foi apresentado neste trabalho, esta reputação é compartilhada pelas RSUs e UC.

O resultado desta avaliação negativa do veículo, ou seja, da alteração da sua reputação, foi apresentado no cenário 02 em que o mesmo veículo, que efetuou o ataque anterior, tenta executar outro ataque. Neste cenário é verificada uma das principais vantagens de um sistema de reputação centralizado, uma vez que se tem a informação sobre a reputação de um veículo não importa em qual localidade ele se encontra é possível obter resposta sobre esta reputação, o que não ocorreria em um sistema de reputação distribuído.

No cenário 02, pode-se considerar que o ataque não obteve sucesso, pois os veículos ao receberem o alerta consultam a RSU, que por sua vez não possuía informação sobre a reputação do veículo e consulta a UC para obter uma resposta. A UC por ter recebido a atualização da reputação deste veículo, retorna a resposta a RSU, que consequentemente, a repassa aos veículos. Os veículos ao receberem a resposta, verificam que a reputação do veículo emissor é baixa e, assim, não confiam nas mensagens enviadas por este veículo. Com os resultados apresentados na simulação do cenário 02, é possível verificar que o ataque não afetou o comportamento dos veículos, que mantiveram suas rotas e velocidades. Isso permitir averiguar a efetividade do uso de um sistema de reputação, mesmo centralizado.

O ambiente simulado no cenário 03 apresentou um veículo com reputação alta enviando uma mensagem de alerta. Com os resultados da simulação é possível verificar que o sistema de reputação centralizado funcionou da maneira esperada, ou seja, os veículos, ao receberem uma mensagem de alerta de um veículo com reputação alta, confiaram nesta mensagem e consequentemente tomaram as medidas de precaução, ou seja, diminuíram a velocidade e buscaram uma nova rota.

O cenário 04 apresentou uma das principais desvantagens do SRC, em que um veículo com reputação conhecida e alta, resolve por algum motivo enviar um alerta falso. Este é um caso crítico, pois por ter uma reputação alta, os demais veículos tendem a confiar na mensagem recebida, porém desta vez a mensagem é falsa. Com os resultados coletados da simulação é possível verificar que o ataque foi bem sucedido, pois os demais veículos alteraram suas rotas e diminuíram a velocidade. Entretanto, cabe ressaltar que esse tipo de resultado seria obtido em um sistema de reputação distribuído também, pois o problema não é da topologia do sistema, mas sim do fato de ser um sistema baseado em reputação, e um usuário mal intencionado pode se fazer passar por um usuário bem intencionado para ganhar a confiabilidade do sistema e, posteriormente, fazer uso dessa vantagem para atacar o sistema.

Outro dado analisado foi o tempo de resposta da RSU ao veículo que efetua uma consulta. Em todos os cenários a média do tempo de resposta foi menor que 1 segundo, o que

faz com que os veículos percorrem uma distância pequena e torna possível uma tomada de decisão em um tempo que possibilita a reação do motorista.

Com o estudo realizado foi possível verificar algumas vantagens e desvantagens da arquitetura centralizada proposta neste trabalho em relação às demais:

- **Vantagens**
  - Tempo indeterminado de armazenamento da reputação;
  - Quantidade infinita de reputações armazenadas;
  - A reputação pode ser consultada em qualquer local e momento.
- **Desvantagens**
  - Alto custo para implementação da arquitetura, para disponibilizar RSU em todas as regiões;
  - Alto custo com processadores de alto desempenho, para poder processar todas as requisições.
  - Overhead de comunicação, no caso em um horário de *rush* todos os veículos decidirem enviar ou consultar reputações ou mensagens.
  - O problema de um veículo com reputação alta, enviar um alerta falso;

Uma das principais vantagens é relacionada à posição do veículo atacante, pois nesta arquitetura se um veículo foi avaliado negativamente em determinada região, esta sua reputação poderá ser consultada a qualquer momento e em qualquer local, ou seja, mesmo que o veículo mude de bairro, cidade, estado ou até país (dependendo da infraestrutura) se este iniciar um novo ataque, os veículos conectados a ele poderão consultar a sua reputação, pois a Unidade Central terá recebido em algum momento anterior esta informação. Nas outras arquiteturas isto não poderia ser verificado, pois a reputação do veículo seria avaliada apenas naquele momento.

Outra vantagem que se pode notar é relacionada com o tempo de armazenamento da reputação do veículo. Nas arquiteturas distribuídas há uma grande discussão em por quanto tempo o veículo deveria armazenar a informação sobre a reputação de um outro veículo, pois devido a mobilidade da rede, pode ser que o veículo nunca mais se conecte novamente com o veículo que ele mantém a informação, mas pode ocorrer que eles venham a se conectar novamente no dia seguinte, ou no mês seguinte, e assim por diante.

Como pôde ser verificado o sistema de reputação centralizado apresenta ser mais vantajoso do que uma arquitetura distribuída, pois delega à outra entidade a responsabilidade. Todavia, esta arquitetura centralizada tem suas desvantagens. A principal delas é relacionada

com a infraestrutura. Para aplicar esta arquitetura fazem-se necessárias a instalação de Estações bases por todas as ruas, estradas e rodovias para ser perfeitamente funcional e além de possuir processadores de alto desempenho para atender todas as requisições, e isto não é fácil de aplicar, principalmente devido ao custo financeiro.

O próximo capítulo apresenta as conclusões deste trabalho, as contribuições obtidas, os problemas enfrentados e os trabalhos futuros.

CAPÍTULO 7 – CONCLUSÕES

VANETs têm se mostrado ser a próxima geração de redes quando se fala em veículos inteligentes. Desta maneira, garantir a segurança e confiabilidade da rede é o alvo das principais pesquisas desenvolvidas. Entre estas pesquisas estão os sistemas de reputação, que conforme foi estudado e apresentado neste trabalho tem o objetivo de verificar quais nós são confiáveis dentro da rede.

Atualmente há diversos sistemas de reputação propostos, contudo como foram apresentados, todos trabalham em uma arquitetura distribuída. No trabalho desenvolvido nesta dissertação optou-se por apresentar e simular uma nova arquitetura para o sistema de reputação. Esta arquitetura constitui um Sistema de Reputação Centralizado, em que uma Unidade Central é responsável pela reputação dos veículos.

Após os estudos das arquiteturas existentes, a pesquisa apresentou uma nova abordagem para os sistemas de reputação. Foram desenvolvidos diferentes cenários em que se avaliou a efetividade ou não de ataques quando proferidos em um ambiente de reputação centralizado.

Como pode ser observado, a abordagem centralizada apresenta algumas vantagens quando comparada a abordagem descentralizada,. O grande problema dessa abordagem centralizada é o custo envolvido na geração da infraestrutura necessária para que ela possa ser utilizada. Entretanto, o que se observa é uma redução constante no custo das tecnologias atuais, de modo que, acredita-se que em um curto período de tempo esse não mais será um problema real.

7.1. Contribuições desse trabalho

Este trabalho apresentou contribuições para a área de redes de computadores, mais especificamente com relação a computação ubíqua, levando em consideração principalmente aqueles que se encaixam na categoria de ordem crítica. Veículos terrestres, mais especificamente os carros de passeio, são hoje muito utilizados e se apresentam a cada dia como um dispositivo que pode se comunicar com outros sistemas, pela presença de computadores de bordo. Sendo assim, encontrar soluções de segurança para o contexto deste tipo de sistemas é fundamental para auxiliar em aplicações futuras.

A principal contribuição deste trabalho é a realização de experimentos por meio de simulação que demonstrar a viabilidade em se utilizar um sistema de reputação centralizado. Foi utilizado o simulador NCTUns, que não só permite fazer a simulação, mas também emular o ambiente. Esse simulador permite que VANETs sejam simuladas, de modo que permite a obtenção de informações mais efetivas quanto ao ambiente simulado.

Apesar do simulador ser específico para VANETs, não possui todas as características necessárias para o desenvolvimento integral da proposta. Devido a isto, novos algoritmos foram propostos e implementados para que o sistema de reputação centralizado pudesse ser executado.

Os resultados apresentados em forma de estudos de caso foram dispostos em gráficos que auxiliam na percepção das propostas que apresentam melhor desempenho.

Este trabalho proveu também resultados concretos que podem auxiliar na escolha do tipo de arquitetura a ser utilizada quando da reputação em VANETs.

Com a pesquisa e estudos realizados pelos integrantes deste projeto foi desenvolvido um artigo com o título: SIMULATION OF A CENTRALIZED REPUTATION SYSTEM FOR VANETS e publicado no congresso: **WORLDCOMP'14** - The 2014 World Congress in Computer Science, Computer Engineering, and Applied Computing **PDPTA'14** - The 2014 International Conference on Parallel and Distributed Processing Techniques and Applications.

E por fim, este trabalho teve como contribuições a geração de um documento que permite uma reunião de informações sobre VANETs, desde as suas características, modo de funcionamento, até mesmo os diferentes simuladores que podem ser utilizados para esse tipo específico de rede.

7.2. Dificuldades encontradas

Foram encontradas as seguintes dificuldades durante o desenvolvimento deste projeto:

- **Versão do Linux**: o simulador NCTUns pode ser instalado em outras versões e distribuições Linux, porém este é totalmente funcional quando instalado em distribuição Fedora 12. Portanto foi necessário buscar e instalar uma versão mais antiga do Linux Fedora.
- **Bibliotecas necessárias:** para a correta instalação do simulador NCTUns é necessária a instalação de algumas bibliotecas, antes de iniciar a instalação do simulador e o script de instalação do simulador não as instalam. Portanto nas primeiras tentativas de instalação ocorreu certos erros decorrente da falta destas bibliotecas. Após verificar a

falta destas bibliotecas e efetuar as instalações necessárias a instalação do simulador executou corretamente. O passo a passo para a correta instalação é apresentado no Anexo A.

7.3. Trabalhos futuros

VANETs são consideradas hoje as redes atuais quando se fala em veículos. Por ser relativamente uma área nova, constitui uma das áreas de intensa pesquisa atualmente, e por isso várias são as possibilidades de extensão deste trabalho. Algumas sugestões são:

- Avaliação e comparação dos diferentes algoritmos de votação que podem ser utilizados para realização do processo de reputação;
- Análise dos tempos de comunicação entre diferentes RSUs localizadas em regiões distantes, de modo a averiguar a qualidade da comunicação entre as diferentes RSUs;
- Estudo de como definir a melhor atitude a ser tomada quando a reputação é desconhecida. Possivelmente fazendo uso de algoritmos inteligentes, *machine learning*, entre outras técnicas;
- Estudo e comparação entre os algoritmos que permitem avaliar as reputações dos veículos;
- Modelar um ambiente com servidores replicados, para incluir tolerância a falhas de dos servidores que atuam como RSUs;
- Averiguar qual a relação entre as avaliações de reputação dos veículos em relação às distâncias relativas entre os veículos e;
- Estudar e implementar um algoritmo que trate uma reputação que acompanhe o motorista. Quando se tira um carro zero, o sistema de reputação não possui um registro do veículo em questão e, apesar do motorista poder ser uma pessoa muito responsável essa informação não está disponível na base de dados do sistema de reputação. Um sistema de reputação que pudesse relacionar motorista mais carro poderia ser mais adequado. A arquitetura apresentada neste projeto esta abordagem não é estudada. Optou-se por estudar apenas as reputações nos veículos e verificar e analisar os resultados obtidos.

REFERÊNCIAS BIBLIOGRÁFICAS

ADAMS, J. W.; DAVIS, N. J.. ***Toward a decentralized trust-based access control system for dynamic collaboration.*** In IEEE Workshop on Information Assurance, pages 317-324, 2005.

ADLER, C. e STRASSBERGER, M. ***Putting together the pieces - a comprehensive view on cooperative local danger warning.*** In ITS '06: Proceedings of 13th ITSWorld Congress and Exhibition on Intelligent Transport Systems and Services - 2006.

BACHIR, A. e BENSLIMANE, A. ***"A multicast protocol in ad hoc networks inter-vehicle geocast,"*** em Proceedings of IEEE VTC-Spring, 22-25 April 2003, pp. 2456–2460.

BAGRODIA, R.; MEYER, R.; TAKAI, M.; Yu-An Chen; Xiang Zeng; MARTIN, J.; Ha Yoon Song;. **Parsec: A Parallel Simulation Environment for Complex Systems.** Computer, Vol. 31(10), October 1998, pp. 77-85.

BEHRISCH, M., BIEKER, L., ERDMANN, J e KRAJZEWICZ, D. **SUMO -** ***Simulation of Urban MObility: An Overview*** In: SIMUL 2011, The Third International Conference on Advances in System Simulation, 2011

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML.** 1ed. Campos, 2006. 300 p.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML: guia do usuário**. Trad. Fábio Freitas da Silva e Cristina de Amorin Machado. - Rio de Janeiro: Elsevier, 2005 – 4ª Reimpressão.

BUCHEGGER, S. e LE BOUDEC, J. Y.***A robust reputation system for p2p and mobile ad hoc networks.***In Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems, 2004.

Car2Car. Car2car communication consortium. http://www.car-2-car.org- 2009.

CHEN, L.; LI, Q.; MARTIN, K. M. e NG, Siaw-Lynn. ***A Privacy-aware Reputation-based Announcement Scheme for VANETs.*** Wireless Vehicular Communications (WiVeC), 2013 IEEE 5th International Symposium on. Page 1-5. junho de 2013.

CHOFFNES, D. R. e BUSTAMANTE, F. E.. ***An integrated mobility and tra_c model for vehicular wireless networks***. In VANET '05: Proceedings of the 2nd ACM international workshop on Vehicular ad hoc networks, pag. 69-78, New York, NY, USA, 2005. ACM.

CAO, Y.; MAHR, A.; CASTRONOVO, S.; THEUNE, M.; STAHL, C. and MÜLLER, C. ***Local danger warnings for drivers: the effect of modality and level of assistance on driver reaction***. In: Proceedings of the 14th International Conference on Intelligent User Interfaces, Hong Kong (pp. pp. 239-248), 2010.

COMER, Douglas E. **Redes de computadores e internet.** 4 ed. São Paulo-SP. Bookman, 2007. 632p

COMER, Douglas E.**Sistemas Distribuídos - Conceitos e Projeto**.4ed. Bookman.São Paulo-SP, 2006. 784p

DEEPADASARATHAN (Dept. of Comput. Sci. & Eng., St. Peter's Coll. of Eng. & Technol., Chennai, India) e KUMAR, P.N. ***A novel method to avoid stale route cache problem of dynamic source routing protocol for mobile ad hoc network.*** Internacional Conference on Current Trends in Engineering and Technology, ICCTET'13. ISBN: 978-1-4799-2583-4. P. 167 – 170. July 3, 2013, Coimbatore, India

DEWAN, P. e DASGUPTA, P.***Securing reputation data in peer-to-peer networks.*** In PDCS: Proceedings International Conference on Parallel and Distributed Computing and Systems, MIT Cambridge, USA, 2004.

DEWAN, P.; DASGUPTA, P. e BHATTACHARYA, A.***On using reputations in ad hoc networks to counter malicious nodes.*** In ICPADS '04: Proceedings of the Parallel and Distributed Systems, Tenth International Conference, p. 665, Washington, DC, USA. IEEE Computer Society, 2004.

DOTZER, F.; FISCHER, L. e MAGIERA, P. ***Vars: A vehicle ad hoc network reputation system.*** In WOWMOM '05: Proceedings of the Sixth IEEE International Symposium on World of Wireless Mobile and Multimedia Networks, pp. 454-456, Washington, DC, USA. IEEE Computer Society - 2005.

DHURANDHER, S. K.; OBAIDAT, M. S.; JAISWAL, A.; TIWARI, A.; and TYAGI, A.. ***Securing vehicular networks a reputation and plausibility checks-based approach.*** IEEE GLOBECOM Workshops, pages 1550-1554, 2010.

GOMES, C. F.; RIBEIRO, P. C.**Gestão da Cadeia de Suprimentos Integrada à Tecnologia da Informação.**1ed.Cengage Learning, 2004. 360p

CAMERON, G. D.B.; DUNCAN, G. I.D. ***PARAMICS—Parallel microscopic simulation of road traffic.*** The Journal of Supercomputing. ISSN 0920-8542. Volume 10, Issue 1 , pp 25-53. 01/03/1996

HOUAISS **Dicionário Eletrônico**. Versão monousuário 3.0. junho de 2009.

HUBAUX, J. P.; CAPKUN, S. e LUO, J..***The security and privacy of smart vehicles***.*IEEE Security andPrivacy*, 2(3):49-55- 2004.

IBRAHIM, K. e WEIGLE, M. C. "**CASCADE:** ***Cluster-based accurate syntactic compression of aggregated data in VANETs***," em Submission, 2008.

IBRAHIM, K.; WEIGLE, M.C., "***ASH: Application-aware SWANS with highway mobility,***" *INFOCOM Workshops 2008, IEEE* , vol., no., pp.1,6, 13-18 April 2008

IEEE P802.11 . 802.11p. Dispinível em: http://www.ieee802.org/11/Reports/tgp_update.htm Acessado em novembro de 2012.

IEEE Draft Guide for Wireless Access in Vehicular Environments (WAVE) - Architecture," in IEEE P1609.0/D7.0, August 2013, vol., no., pp.1-77, Aug. 8 2013

IZQUIERDO, M. R. e REEVES, D. S. **A survey of statistical source models for variable-bit-rate compressed video.** Multimedia Syst., 1999. 7(3):199_213.

J-SIM. Disponível em http://j-sim.cs.uiuc.edu/. Acessaod em janeiro de 2013.

KARNADI, F.K., MO, Z. H., e LAN, K.. ***Rapid Generation of Realistic Mobility Models for VANET.*** Wireless Communications and Networking Conference, 2007.WCNC 2007. IEEE, pages 2506{2511, March 2007.

KESTING, A.; TREIBER, M. e HELBING, D. "MOBIL: ***General lane changing model for car-following models*** em Proceedings of the Transportation Research Board Annual Meeting, Jan. 2007.

KONOMI, S.; ROUSSOS, G. Editorial: ubiquitous computing in the real world. **Personal and Ubiquitous Computing**. Londres, 01 de outubro de 2007. V.11:505–506.

KOSCH, T. ***Local danger warning based on vehicle ad hoc networks: Prototypeand simulation.***InWIT '04: Proceedings of 1st International Workshop on IntelligentTransportation - 2004.

LESCHENAUER, L.; GLIGOR, V. D.; BARAS, J.. ***On trust establishment in mobile ad hoc networks.*** In Proceedings of the Security Protocols Workshop, pages 47-66. Springer-Verlag, 2002.

LI Qin, AMIZAH Malip, KEITH M. Martin, SIAW-LYNN Ng, JIE Zhang, ***A Reputation-based Announcement Scheme for VANETs***. In Vehicular Technology, IEEE Transactions on.Volume 61 p 4095-4108, 2012.

LI, W.; JOSHI, A. e FININ, T. ***CAST: Context-Aware Security and Trust framework for Mobile Ad hoc Networks using policies.*** Distributed and Parallel Databases. Print ISSN 0926-8782; Online ISSN 1573-7578. Volume 31, Issue 2, pp 353-376. June 2013.

MANET.**Internet engineering task force manet working group**. Disponível em: http://www.ietf.org/html.charters/manet-charter.html. Acessado em agosto 2012.

MANGHARAM, R. , WELLER, D., RAJKUMAR, R., MUDALIGE, P., e BAI, F.. ***GrooveNet: A Hybrid Simulator for Vehicle-to-Vehicle Networks.*** Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on , vol., no., pp.1,8, 17-21 July 2006.

MANGHARAM, R., WELLER, D. S., STANCIL, D. D., RAJKUMAR, R. e PARIKH, J. S. ***GrooveSim: A Topography-Accurate Simulator for Geographic Routing in Vehicular Networks***. In Proceedings of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks, pages 59-68, 2005.

MARTI, S.; GIULI, T. J.; LAI, K.; BAKER, M.. ***Mitigating routing misbehavior in mobile ad hoc networks.*** In International Conference on Mobile Computing and Networking, pages 255-265, 2000.

MARMOL, F. G.; PéREZ, G. M. ***Towards pre-standardization of trust and reputation models for distributed and heterogeneous systems.*** Computer Standards & Interfaces, Elsevier B.V., v. 32, n. 4, p. 185–196, jun. 2010. ISSN 09205489

MENON, V. G. et al***. Performance Analysis of Traditional Topology based Routing Protocols in Mobile Ad hoc Networks.*** The International Journal of Computer Science & Applications (TIJCSA) ISSN – 2278-1080, Vol. 2 No. 01 March 2013

MICHIARDI, P.; MOLVA, R***.. Core: A collaborative reputation mechanism to enforce node cooperation. In Mobile Ad Hoc Networks***. Communication and Multimedia Security, pages 107{121, 2002.

NICE, T. P. A. ***A Survey on Intrusion Detection Systems in Mobile Ad Hoc Networks.*** International Journal of Advanced Research in Computer and Communication Engineering. ISSN (Print) : 2319-5940; ISSN (Online) : 2278-1021 Vol. 2, Issue 5, May 2013

OMNet++. Disponível em http://www.omnetpp.org/. Acessado em janeiro de 2013.

OPNET - Network Simulator. Disponivel em: http://www.riverbed.com/products-solutions/products/opnet.html?redirect=opnet. Acessado em janeiro de 2013.

OSTERMAIER, B.***Analysis and improvement of inter-vehicle communication security by simulation of attacks.***Master's thesis, TechnischeUniversitätMünchen, 2005.

OSTERMAIER, B.; DOTZER, F. e STRASSBERGER, M. ***Enhancing the security of local danger warnings in vanets - a simulative analysis of voting schemes.*** In ARES '07:

PARAMICS – 2013. Disponível em http://www.sias.com/2013/sp/sparamicshome.htm. Acessado em janeiro de 2013.

Proceedings of the The Second International Conference on Availability, Reliability and Security, pp. 422-431, Washington, DC, USA. IEEE Computer Society - 2007.

PARK, J. et al. Specialissueon “Intelligent systems and services for ubiquitous computing”. **Personal and Ubiquitous Computing,** Londres, 01 de outubro de 2009. V.13:445–447 –

PATWARDHAN, A. et al. ***A data intensive reputation management scheme for vehicular ad hoc networks.***In Proceedings of the Second International Workshop on Vehicle-to-Vehicle Communications.IEEE - 2006.

PIORKOWSKI, M., RAYA, M., LUGO, A. L., PAPADIMITRATOS, P., GROSSGLAUSER, M. e HUBAUX, J.. **TraNS*: Realistic Joint Tra_c and Network Simulator for VANETs***. ACM SIGMOBILE Mobile Computing and Communications Review, 2007.

PONGOR, György, "OMNeT: Objective Modular Network Testbed", MASCOTS '93: Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems: 323—326, 1993.

RAYA, M.;HUBAUX, J.P.; ***The Security of Vehicular Ad Hoc Networks***. SASN, Alexandria, VA, USA – 2005.

RESNICK, P.; ZECKHAUSER, R.. ***Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system.*** In Michael R. Baye, editor, The Economics of the Internet and E-Commerce, volume 11 of Advances in Applied Microeconomics, pages 127-157. Elsevier Science, 2002.

SINGH, R. K.; JOSHI, R. e SINGHA, M. ***Analysis of Security Threats and Vulnerabilities in Mobile Ad Hoc Network (MANET).*** International Journal of Computer Applications (0975 – 8887) Volume 68– No.4, April 2013

SCHNEIER, B. ;***Applied Cryptography: Protocols, Algorithms, and Source Code in C*** – 2a. Edição – John Wiley & Sons, 1996. 758p.

SPIEKERMANN, S.; LANGHEINRICH, M.An update on privacy in ubiquitous computing.**Personal and Ubiquitous Computing.** Londres, 01 de agosto de 2009. V.13:389–390 –

STALLINGS, W.;**Criptografia e Segurança de Redes: Princípios e Práticas**. 4ª. Edição. Pearson, 2008.

SUMO – 2013. Disponível em http://sumo-sim.org/. Acessado em janeiro de 2013.

SUN, Y. L.; HAN, Z.; and LIU, K. J. R.. ***Information theoretic framework of trust modeling and evaluation for ad hoc networks.*** IEEE Journal on Selected Area in Communications, 24(2):305{317, 2006

SWAMYNATHAN, G. et al.**Globally decoupled reputations for large distributed networks**. Adv. MultiMedia, 2007(1):12-12.

SWANS– Scalable Wireless Ad hoc Network Simulator. Disponível em http://jist.ece.cornell.edu/. Acessado em janeiro de 2013.

SWANS++ - Extensions to the Scalable Wireless ad hoc Network Aimulator. Disponível em: http://www.aqualab.cs.northwestern.edu/projects/143-swans-extensions-to-the-scalable-wireless-ad hoc-network-simulator. Acessado em janeiro de 2013.

The Random Trip Mobility (2011). Disponível em: http://ica1www.epfl.ch/RandomTrip. Acessado em novembro de 2012.

TREIBER, M.; HENNECKE, A. e HELBING, D. "***Congested traffic states in empirical observations and microscopic simulations***," Physical Review E, vol. 62, p. 1805, 2000.

VisSIM. 2013 Disponível em http://vision-traffic.ptvgroup.com/en-uk/home/. Acessado em janeiro de 2013

VEINS, 2014. ***Vehicles in Networks Simulation***. Disponível em http://veins.ca2x.org. Acessado em janeiro de 2014.

WASHINGTON, L.C.; **Elliptic *Curves: Number Theory and Cryptography*** – 2Edição - Chapman & Hall/CRC, 2008. 440p

WANG, Z. e CHIGAN, C. ***Countermeasure uncooperative behaviors with dynamic trust-token in vanets.*** In ICC '07: Proceedings of IEEE International Conference on Communications, pp. 3959-3964, Glasgow, Scotland. IEEE - 2007.

WANG, B.; SOLTANI, S.; SHAPIRO, J. K.; TAN, P. ning. ***Local detection of selfish routing behavior in ad hoc networks.*** In International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN), Las Vegas, pages 392-399, 2005.

WANG, S.Y, CHOU, C. L., CHIU, Y. H., TZENG, Y. S., HSU, M. S., CHENG, Y. W., LIU, W. L. e HO, T. W. **NCTUns 4.0:** ***An Integrated Simulation Platform for Vehicular Traffic. In Communication, and Network Researches,*** WiVec'07, 2007.

WANG, S.Y, CHOU, C. L., LIN, Chih-Che. ***The GUI User Manual for the NCTUns 6.0 Network Simulator and Emulator.*** 2010

ZENG, X.; BAGRODIA, R. e GERLA, M. ***GloMoSim: a library for parallel simulation of large-scale wireless networks.*** Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Print ISBN:0-8186-8457-7; pag. 154 – 161. -29 May 1998

ZOURIDAKI, C.; MARK, B. L.; HEJMO, M.. ***A quantitative trust establishment framework for reliable data packet delivery in manets.*** In Proceedings of the Third ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN), pages 1-10. ACM, 2005.

APÊNDICE A – conexao_BD.h

Este apêndice descreve o arquivo que foi inserido no simulador NCTUns para fornecer a conexão com o banco de dados MYSQL.

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "brodcast.h"
#include "mysql/mysql.h"


//Parâmetros para conexão com BD.
#define HOST "localhost"
#define USER "root"
#define PASSWD "mestrado"

#define DESCONHECIDA -2 //-2 é o valor de uma reputação desconhecida.

#define LIMITE_INFERIOR -1 //Limite inferior dos valores das reputaçãoes
#define LIMITE_SUPERIOR 10 //Limite superior dos valores das reputaçãoes

//Posição dos campos na tabela reputacao
#define CAMPO_ID 0
#define CAMPO_VALOR 1


MYSQL *Conexao;
MYSQL_RES *Result;
MYSQL_ROW Row;
MYSQL_FIELD *field;

int res;
int RowsReturned;
int valorReputacao = 0;
char RSU[10];
char veiculo[10];
char valor[10];
char nameBD[15] = "RSU";


//Abre uma conexão com o BD.
void ConectarBD(char BD[80])
 {
      Conexao = mysql_init( NULL );


      if ( mysql_real_connect(Conexao, HOST, USER, PASSWD, BD, 0, NULL, 0) )
      {
            printf("conectado com sucesso no BD = %s!\n",BD);
      }
      else
      {
            printf("Falha de conexao\n");
            printf("Erro %d : %s\n", mysql_errno(Conexao), mysql_error(Conexao));
      }
 }
```

```
 //Fecha a conexão com o BD.
 void CloseBD()
{
    mysql_free_result( Result );
    mysql_close( Conexao );
}

 /*
 Verifica se há informação sobre o ID consultado.
 */
 int VerificaVeiculo(char veiculo[10])
 {
   //int res;

   //Monta a string de comando SQL.
   char sql[90] = "SELECT * FROM reputacao WHERE ID = ";
   strcat(sql,veiculo);
   strcat(sql,";");

   printf("\nSQL: %s\n",sql);

   //Executa a query no BD.
   res = mysql_query(Conexao,sql);
   Result = mysql_store_result(Conexao);
   RowsReturned = mysql_num_rows( Result );

   return RowsReturned;
 }


 void InserirReputacao(char veiculo[10], char valor[10])
 {
     //Monta a string de comando SQL.
     char sql[90] = "INSERT INTO reputacao(ID,valor) VALUES (";//21,99);"
     strcat(sql,veiculo);
     strcat(sql,",");
     strcat(sql,valor);
     strcat(sql,");");
     printf("\nSQL: %s\n",sql);

     //Executa a query no BD.
     res = mysql_query(Conexao,sql);
     printf("RES = %d\n",res);
     if (!res)
       printf("Registros INSERIDOS %d\n", mysql_affected_rows(Conexao));
     else
       printf("Erro na inserção %d : %s\n", mysql_errno(Conexao),
mysql_error(Conexao));
 }


 void AtualizarReputacao(char veiculo[10], char valor[10])
 {
      //Monta a string de comando SQL.
      char sql[90] = "UPDATE reputacao SET valor = ";
      strcat(sql,valor);
      strcat(sql," WHERE ID = ");
      strcat(sql,veiculo);
      strcat(sql,";");
```

```
      printf("\nSQL: %s\n",sql);

      //Executa a query no BD.
      res = mysql_query(Conexao,sql);

      if (!res)
      {
       printf("Registros ALTERADOS %d\n", mysql_affected_rows(Conexao));
      }
      else
       printf("Erro na inserção %d : %s\n", mysql_errno(Conexao),
mysql_error(Conexao));
 }
 /*
 Método que atualiza ou insere um novo valor de reputação de um veículo
 <parâmetros>
 idBD = ID da RSU que está efetuando a atualização no BD
 idVeiculo = número ID do veículo a ser avaliado
 voto = voto da reputação fornecida (-1 avaliação negativa; +1 avaliação positiva)
 </parâmetros>
 */
 void UpdateReputacao(int idBD, int idVeiculo,int voto)
 {
   char nameBD[15] = "RSU";
   sprintf(RSU, "%d", idBD); //Converte em string o parâmetro idBD e armazena na
variável RSU
   strcat(nameBD,RSU);
   sprintf(veiculo, "%d", idVeiculo); //Converte em string o parâmetro idVeiculo e
armazena na variável veiculo

   /*
      Verifica se o veículo já está no Banco de Dados.
      Se não estiver insere um novo registro de reputação com o ID do veículo.
      Se estiver atualiza a reputação do veículo.
   */
   ConectarBD(nameBD);

   if(!VerificaVeiculo(veiculo))
   {
     sprintf(valor, "%d", voto); //Converte em string o parâmetro voto e armazena na
variável valor
     InserirReputacao(veiculo, valor);
   }
   else
   {
     Row = mysql_fetch_row(Result);
     valorReputacao = atoi(Row[CAMPO_VALOR]);

      //Intervalo dos valores das reputações entre -1 e 10 [1,10];
      //Se o valor atual da reputação estiver em um dos limites do intervalo não
atualiza mais o valor da reputação.
       if(valorReputacao > LIMITE_INFERIOR && valorReputacao < LIMITE_SUPERIOR)
        {
         valorReputacao = valorReputacao + voto;
         sprintf(valor, "%d", valorReputacao); //Converte em string o parâmetro
valorReputacao e armazena na variável valor
          AtualizarReputacao(veiculo, valor);
        }
   }

   //Atualiza reputação do veículo para a Unidade Central
```

```
   AtualizaUC(veiculo, valor);

   //Fecha o BD e limpa as variáveis.
   CloseBD();
   strcpy(veiculo,"");
   strcpy(nameBD,"");
   strcpy(valor,"");
   strcpy(nameBD,"");
 }


//Retorna o valor da reputação do veículo informado no parâmetro idVeiculo
 int RetornaValorDaReputacao(int idBD, int idVeiculo)
 {
    char nameBD[15] = "RSU";
   sprintf(RSU, "%d", idBD); //Converte em string o parâmetro idBD e armazena na
variável RSU
   strcat(nameBD,RSU);
   sprintf(veiculo, "%d", idVeiculo); //Converte em string o parâmetro idVeiculo e
armazena na variável veiculo

   ConectarBD(nameBD);

   if(!VerificaVeiculo(veiculo))
   {
     printf("DESCONHECIDA\n");
     return DESCONHECIDA;
   }
   else
   {
     printf("CONHECIDA\n");
     Row = mysql_fetch_row(Result);
     valorReputacao = atoi(Row[CAMPO_VALOR]);

     return valorReputacao;
   }
   CloseBD();
         }
```

APÊNDICE B – broascast.h

Este aquivo é um processo auxiliar para as trocas de mensagens e reputações entre veículos e RSUs.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "tactic_api.h"

void checarTipo(struct typeChecker *tipo)
{
  FILE *fp;
  char str[30];
  int pp = 0;
  //int i = 1;
  if(!(fp = fopen("/usr/local/arquivos/teste", "r")))
       {
              printf("ERRO para abrir arquivo");
       }
       fgets(str,30,fp);
       //printf("LINHA checar tipo= %s", str);

       pp = atoi(str);
  printf("PP = %d \n\n", pp);
       tipo->type = atoi(str);
       printf("TIPO TYPE = %d\n",tipo->type);
}

void lerBufferCarAgent(struct agentClientReportStatus *car, char arquivo[30])
{
  FILE *fp;
  char path_buffer[100] = "/usr/local/arquivos/";
  strcat( path_buffer,arquivo);
  char str[30];
  int i = 1;
  if(!(fp = fopen(path_buffer, "r")))
  {
       printf("ERRO para abrir arquivo");
  }
  while (i <= 11) {
    fgets(str,30,fp);
    printf("Linha = %s\n", str);
    switch(i) {
      case 1: car->type = atoi(str);
              break;
      case 2: car->moreMsgFollowing = atoi(str);
              break;
      case 3: car->nid = atoi(str);
              break;
      case 4: car->x = atof(str);
              break;
      case 5: car->y = atof(str);
              break;
      case 6: car->acceleration = atof(str);
              break;
      case 7: car->speed = atof(str);
```

```
                break;
        case 8: car->direction = atof(str);
                break;
        case 9: car->timeStamp = atof(str);
                break;
        case 10: car->TTL = atof(str);
                break;
        case 11: car->seqNum = atof(str);
                break;
        default: printf("DEFAULT/n");
         break;
      }
      i++;
    }
}

void escreveBufferCarAgent(struct agentClientReportStatus car, char arquivo[30])
{
  FILE *fp;

  char path_buffer[100] = "/usr/local/arquivos/";
  strcat( path_buffer,arquivo);

  if(!(fp = fopen(path_buffer, "w")))
       {
              printf("ERRO para abrir arquivo");
       }
  char str[50];
  sprintf(str, "%d",car.type);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%d",car.moreMsgFollowing);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%d",car.nid);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%f", car.x);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%f", car.y);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%f", car.acceleration);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%f", car.speed);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%f", car.direction);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%f", car.timeStamp);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%d", car.TTL);
  fputs(str,fp);
  fputs("\n",fp);
  sprintf(str, "%d", car.seqNum);
  fputs(str,fp);
  fputs("\n",fp);
```

```
  fclose(fp);
}

void AtualizaUC(char veiculo[10], char valor[10])
{
  FILE *fp;
  char linha[30] = "";
  if(!(fp = fopen("/usr/local/arquivos/buffer_UC", "a")))
  {
    printf("ERRO para abrir arquivo");
  }
  strcat(linha,veiculo);
  strcat(linha,";");
  strcat(linha,valor);
  fputs(linha,fp);
  fputs("\n",fp);
  printf("UC Atualizada!!!\n");
  fclose(fp);
}

void LimpaBuffer(char arquivo[30])
{
  FILE *fp;
  char path_buffer[100] = "/usr/local/arquivos/";
  strcat( path_buffer,arquivo);
  char str[30];
  int i = 1;
  if(!(fp = fopen(path_buffer, "w")))
  {
       printf("ERRO para abrir arquivo");
  }
}

FILE *lerBufferAtualizacaoRSUAgent(char arquivo[30])
{
  FILE *fp;
  char path_buffer[100] = "/usr/local/arquivos/atualizar";
  strcat(path_buffer,arquivo);

  if(!(fp = fopen(path_buffer, "r")))
  {
       printf("ERRO para abrir arquivo");
       return 0;
  }
  return fp;
}

FILE *lerBufferConsultaReputacaoRSUAgent(char arquivo[30])
{
  FILE *fp;
  char path_buffer[100] = "/usr/local/arquivos/consulta";
  strcat(path_buffer,arquivo);

  if(!(fp = fopen(path_buffer, "r")))
  {
       printf("ERRO para abrir arquivo");
       return 0;
  }
  return fp;
}
```

```
void RespondeReputacaoBuffer(char veiculo[10], char valor[10])
{
  FILE *fp;
  char linha[30] = "";
  char path_buffer[35] = "/usr/local/arquivos/consulta";
  strcat(path_buffer,veiculo);
  printf("CAMINHO ARQUIVO %s\n",path_buffer);
  if(!(fp = fopen(path_buffer, "w")))
  {
    printf("ERRO para abrir arquivo");
  }

  strcat(linha,valor);
  fputs(linha,fp);
  fputs("\n",fp);
  printf("Reputação Respondida!!!\n");
  fclose(fp);
}
```

APÊNDICE C – MeuCarAgent.cc

Este apêndice é o arquivo CarAgent modificado, foram inseridos alguns métodos, para consulta e atualização das reputações dos veículos.

```
/*
 * Copyright (c) from 2000 to 2010
 *
 * Network and System Laboratory
 * Department of Computer Science
 * College of Computer Science
 * National Chiao Tung University, Taiwan
 * All Rights Reserved.
 *
 * This source code file is part of the NCTUns 6.0 network simulator.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation is hereby granted (excluding for commercial or
 * for-profit use), provided that both the copyright notice and this
 * permission notice appear in all copies of the software, derivative
 * works, or modified versions, and any portions thereof, and that
 * both notices appear in supporting documentation, and that credit
 * is given to National Chiao Tung University, Taiwan in all publications
 * reporting on direct or indirect use of this code or its derivatives.
 *
 * National Chiao Tung University, Taiwan makes no representations
 * about the suitability of this software for any purpose. It is provided
 * "AS IS" without express or implied warranty.
 *
 * A Web site containing the latest NCTUns 6.0 network simulator software
 * and its documentations is set up at http://nsl.cs.nctu.edu.tw/nctuns.html.
 *
 * Project Chief-Technology-Officer
 *
 * Prof. Shie-Yuan Wang <shieyuan@csie.nctu.edu.tw>
 * National Chiao Tung University, Taiwan
 *
 * 01/15/2010
 */

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <fcntl.h>
#include <math.h>
#include <time.h>
#include "math_fun.h"
#include "tactic_api.h"
#include "sock_skel.h"
#include "road.h"
#include "conexao_BD.h"
//#include "brodcast.h"

#include <deque>

#include "mysql/mysql.h"
```

```
#define ITS_NET 1
#define ITS_COLLISION_AVOIDANCE 1
#define ITS_SHOW_MSG 1

#define MOTION_LINEAR 0x30
#define MOTION_TAKING_TURNS_STEP_1 0x40
#define MOTION_TAKING_TURNS_STEP_2 0x41
#define MEGA 1000000 //micro secs
#define SLEEPING_PERIOD 100000 //micro secs
#define VISIBILITY_SCALE_IN_DEGREE 40.0 //degree
#define VISIBILITY_SCALE_IN_DISTANCE 100.0 //meters

FILE *fp;

enum EventType
{
	APPROACHING_INTERSECT,
	IN_INTERSECT,
	LEAVING_INTERSECT
};

class msgSequence
{
	public:
		u_int32_t nid;
		int seqNum; // sequence number
};

//If car agent reach to the position (x, y), it change its direction into "the
direction of class Event"
class Event
{
	public:
		double x, y;
		double direction; // the directeion of class Event
		EventType type;
};

timeval now;


//-----Global variables for IPCs and APIs
int mynid, myTCPsockfd, myUDPsockfd, socketfd2;
int agentUDPportNum = 4000;
int oldCollisionCarID = 0;

//-----Global variables for car's performances.
double MaxAcceleration, MaxDeceleration, MaxVelocity;

//Global variables for car's condition
//1 m/s = 3.6 km/hr
double CurrentVelocity = 0, CurrentAcceleration, CurrentDirection;
double CurrentPOS_x, CurrentPOS_y, CurrentPOS_z = 0;
double ExpectedDirection;
double brokenCarDirection; // It will be a problem if there are many broken cars.
double roadWidth;
Event CurrentEvent;
int WarningDecelerate = 0;
int WakedTimes = -1;
int agentReceivedBrokenCarMsg = 0;
```

```
int brokenCarID = 0;

//About the traffic light
int SeenTheTrafficLightOrNot = 0, SigLight, SigGID = -1, SignalIndex = -1;
double SigPOS_x, SigPOS_y;
vector<msgSequence> msgSeq;

int CacheOrNot = 1;
deque<Event> EventQueue;

CacheRecord myCache;
CacheRecord otherCache;

void FillTheQueue(int NumOfTurns, double *turningPOS_x, double *turningPOS_y, double
*directions, double FirstPointAfterTheTurn_x, double FirstPointAfterTheTurn_y, double
ExpectDirection)
{
	//This function fulls the queue when agent just has reached a new road block.
	//After agent calls the takeATurn function, it uses "pass by reference
parameters" to full the queue by this function.
	Event temp;
	for (int i = 0; i<NumOfTurns; i++){
		//put every turning points into the queue.
		temp.x = turningPOS_x[i];
		temp.y = turningPOS_y[i];
		temp.direction = directions[i];
		EventQueue.push_back(temp);

	}
	//Finally, put the end point of the turning into the queue.
	temp.x = FirstPointAfterTheTurn_x;
	temp.y = FirstPointAfterTheTurn_y;
	temp.direction = ExpectDirection;
	EventQueue.push_back(temp);

}

void reportMyStatusToAGroupOfNode()
{
	static int seqNum=0;
	double lastTime = (double) now.tv_sec * 1000000 + (double) now.tv_usec;
	int value;
	socklen_t len;

	agentClientReportStatus *msg;
	msg = new agentClientReportStatus;

	msg->x = CurrentPOS_x;
	msg->y = CurrentPOS_y;
	msg->type = AGENT_CLIENT_REPORT_STATUS;
	msg->nid = mynid;
	msg->moreMsgFollowing = 1;
	msg->acceleration = CurrentAcceleration;
	msg->speed = CurrentVelocity;
	msg->direction = CurrentDirection;
	msg->seqNum = seqNum;
	msg->timeStamp=lastTime;
	msg->TTL=3;
	seqNum++;
	sockaddr_in cli_addr;
```

```
	//Broadcast start

	value = 1;
	setsockopt(myUDPsockfd , SOL_SOCKET , SO_BROADCAST , &value , sizeof(value));
	len =  sizeof(cli_addr);
	memset(&cli_addr, 0, sizeof(cli_addr));
	cli_addr.sin_family = AF_INET;
	cli_addr.sin_port   = htons(agentUDPportNum);
	cli_addr.sin_addr.s_addr = inet_addr("1.0.1.255");
	int n = sendto(myUDPsockfd, msg, sizeof(struct agentClientReportStatus), 0,
(struct sockaddr *) &cli_addr, len);
	//if(!(fp = fopen("/usr/local/arquivos/teste", "w")))
	//{
	//     printf("ERRO para abrir arquivo");
	//}
	//putc('5',fp);
	//fclose(fp);
	//escreveAquivo(*msg);
	if (n < 0) {
		printf("Agent (%d) sendto failed\n", mynid);
	}
	else{
		printf("Car Agent (%d) broadcast pkt, Size %d, seqNum %d, packet
type==%d\n", mynid, sizeof(struct agentClientReportStatus), msg->seqNum, msg->type);
	}
	//Broadcast end
	free(msg);
}

void receiveMsg()
{
	//char ch;
	socklen_t len;
	int n = 1;
	//char str[20];
	sockaddr_in cli_addr;
	while(n > 0){
		typeChecker p;
		len = sizeof(cli_addr);
		n = recvfrom(myUDPsockfd, (char *) &p, sizeof(struct typeChecker),
MSG_PEEK, (struct sockaddr *) &cli_addr, &len);
		//checarTipo(&p);
		//printf("VALOR de p.type = %d \n", p.type);
		if(n == 0){
			//printf("Car Agent: %d UDP socket error\n", mynid);
			return;
		}
		//if(p.type == RSUAGENT_REPORT_WARNING){
		if(p.type == RSUAGENT_REPORT_WARNING){
			RSUAgentReportWarning msg;
			n = recvfrom(myUDPsockfd, (char *)&msg, sizeof(struct
RSUAgentReportWarning), 0, (struct sockaddr *) &cli_addr, &len);
			printf("Agent %d : Receive Warning from RSU %d\n", mynid,
msg.RSUnid);
			if(msg.AccelerationOrDeceleration == -1)
				WarningDecelerate = -10;
			continue;
		}
		else if(p.type == AGENT_CLIENT_REPORT_STATUS || p.type ==
AGENT_CLIENT_IS_A_BROKEN_CAR){
			agentClientReportStatus msg;
```

```
                    lerArquivo(&msg);
                    lerBufferCarAgent(struct agentClientReportStatus *car, char
arquivo[30])
                    lerBufferCarAgent(struct agentClientReportStatus *car, char
arquivo[30])
                    n = msg.seqNum;
                    if(n > 0){
                           printf("msg.seqNum = %d\n", msg.seqNum);
                           if(msg.seqNum > 0){
                                  int nidExists = 0;
                                  for (int i=0 ; i<(int)(msgSeq.size()); i++)
                                  {
                                         if(msg.nid == msgSeq[i].nid)
                                         {
                                                nidExists=1;
                                                if(msg.seqNum > msgSeq[i].seqNum)
                                                {
                                                       double twoDistance =
Distance_BetweenTwoNode(CurrentPOS_x, CurrentPOS_y, msg.x, msg.y);
                                                       printf("Agent %d receive msg
from agent %d, distance %lf\n", mynid, msg.nid, twoDistance);
                                                       msgSeq[i].seqNum = msg.seqNum;
                                                       if(msg.type ==
AGENT_CLIENT_IS_A_BROKEN_CAR){
                                                              agentReceivedBrokenCarMsg
= 1;
                                                              if(ITS_SHOW_MSG)
                                                              {

       printf("CarAgent(%d) recebeu um alerta de Carro quebrado do Agent(%d): %d\n",
mynid, msg.nid, msg.seqNum);

                                                              }


       if(ITS_COLLISION_AVOIDANCE){
                                                                     brokenCarDirection
= msg.direction;
                                                                     brokenCarID =
msg.nid;
                                                                     printf("Evitar
Colisão se necessario.\n");
                                                              }
                                                              break;
                                                       }
                                                       if(ITS_SHOW_MSG)
                                                              printf("CarAgent(%d)
received a status report from Agent (%d): %d\n", mynid, msg.nid, msg.seqNum);

                                                }
                                                else{
                                                       printf("CarAgent(%d): Message is
already received from agent (%d)_%d\n", mynid, msg.nid, msg.seqNum);
                                                }
                                                break;

                                         }

                                  }
                                  if(nidExists == 0)
                                  {
```

```
                                        msgSequence temp;
                                        temp.nid = msg.nid;
                                        temp.seqNum = msg.seqNum;
                                        msgSeq.push_back(temp);
                                        double twoDistance =
Distance_BetweenTwoNode(CurrentPOS_x, CurrentPOS_y, msg.x, msg.y);
                                        printf("Agent %d receive msg from agent %d,
distance %lf\n", mynid, msg.nid, twoDistance);
                                        if(msg.type == AGENT_CLIENT_IS_A_BROKEN_CAR){
                                                agentReceivedBrokenCarMsg = 1;
                                                if(ITS_SHOW_MSG)
                                                        printf("CarAgent(%d) recebeu
um alerta de Carro quebrado do Agent(%d): %d\n", mynid, msg.nid, msg.seqNum);
                                                //printf("CarAgent(%d) received
a broken car report from Agent(%d): %d\n", mynid, msg.nid, msg.seqNum);

                                                if(ITS_COLLISION_AVOIDANCE)
                                                {
                                                        brokenCarDirection =
msg.direction;
                                                        printf("Evitar Colisão se
necessario.\n");
                                                }
                                        }
                                        else {
                                                if(ITS_SHOW_MSG)
                                                        printf("CarAgent(%d) received a
status report from Agent(%d): %d\n", mynid, msg.nid, msg.seqNum);
                                        }
                                }
                        }
                        n = 0;
                }
        }
    }
}

double RandomDirection(int NumOfDirections, double *directions)
{
    int RandInt = rand() % NumOfDirections;
    return directions[RandInt];
}

void init()
{
    int n;
    char portNumStr[32];
    double CorrectedPOS_x, CorrectedPOS_y;

    // get car node id
    mynid = getMyNodeID();
    srand(mynid);

    // create IPC connection with SE and set agent in group 1
    myTCPsockfd = createTCPSocketForCommunicationWithSimulationEngine(mynid, 1, -1,
-1, -1, -1, -1, -1, -1, 0, socketfd2, PROCESS_TYPE_AGENT, 1);

    // create UDP socket for listening broadcasted message
    sprintf(portNumStr, "%d", agentUDPportNum);
    myUDPsockfd = passiveUDP(portNumStr);
    printf("Agent(%d) created myUDPsockfd %d\n", mynid, myUDPsockfd);
```

```
	if (myUDPsockfd < 0) {
		printf("Agent(%d): Creating myUDPsockfd failed\n", mynid);
		exit(0);
	}
	n = fcntl(myUDPsockfd, F_SETFL, O_NONBLOCK);

	// Get driving behavior from car profile
	n = getProFileData(mynid, MaxVelocity, MaxAcceleration, MaxDeceleration, 1);
	if(n == 0){
		printf("Car %d open car profile fail. Use default value\n", mynid);
		MaxVelocity = 18;
		MaxAcceleration = 1;
		MaxDeceleration = 4;
	}

	// Construct the whole road map by using this function
	constructRoadMapofTheWholeField(1);

	n = getInitialNodePosition(mynid, CurrentPOS_x, CurrentPOS_y, CurrentPOS_z);
	CurrentPOS_z = 0;

	/* correct my position into the middle of the road */
	n = selfCorrectness(CurrentPOS_x, CurrentPOS_y, CorrectedPOS_x, CorrectedPOS_y,
CacheOrNot, &myCache);
	if(n == 1)
	{
		CurrentPOS_x = CorrectedPOS_x;
		CurrentPOS_y = CorrectedPOS_y;
	}
	n = setCurrentWaypoint(myTCPsockfd, mynid, CurrentPOS_x, CurrentPOS_y,
CurrentPOS_z, 1);
	/* default parameter */
	//CurrentVelocity = 0;
	CurrentAcceleration = 0; /* acceleration = 1 m/sec^2 */
	//setCurrentMovingSpeed(myTCPsockfd, mynid, CurrentVelocity, 1);

	getRoadWidth(roadWidth);
	//char bd[80] = "RSU3";
	//ConectarBD(bd);
}

inline int ReachTheNextTriggerPointOrNot()
{
	if(((int) EventQueue.size()) == 0)
		return -1;// Error!!

	double tempAngle = fmod(atan2(CurrentEvent.y- CurrentPOS_y, CurrentEvent.x -
CurrentPOS_x)/ PI* 180+ 360, 360);
	tempAngle = fmod(360- tempAngle, 360);

	double angleDiff = fabs(tempAngle- CurrentDirection);
	if(angleDiff > 180)
		angleDiff = 360- angleDiff;

	if(angleDiff >= 90){
		// move over the event point, which means reach the event point
		if(EventQueue.empty() != true)
			EventQueue.pop_front();
		return 1; // reach Event point
	}
	else
```

```
		return 0; // unreach
}

inline double SelectMinimum(double a, double b)
{
	if(a <= b)
		return a;
	else
		return b;
}

inline double SelectMaximum(double a, double b)
{
	if(a >= b)
		return a;
	else
		return b;
}

/*
 * S: distance in meter
 * v0: current velocity in meter/s (> 0)
 * vt: target velocity in meter/s (> 0, because movng in the same direction with v0)
 * a: acceleration in meter/(s^2)
 * t: time in second
 *
 * E1: vt = v0 + a*t
 * E2: S = v0*t + (1/2)*a*(t^2)
 *
 * From E1, we get a = (vt - v0)/t
 * Put a into E2, we get S = 0.5*(v0 + vt)*t
 * Therefore, we get
 * E3: t = 2 * S / (v0 + vt)
 */
void DetermineVc(double &BufferTime, double &vt)
{
	double NearestPrecedingCarPOS_x, NearestPrecedingCarPOS_y,
NearestPrecedingCarPOS_z, NearestPrecedingCarDirection;
	double NearestPrecedingCarSpeed = 999999;
	double DesiredMaxSpeed, DistanceToNextEventPoint; // in meter/secs.
	double SAFETY_INTERVEHICLE_DISTANCE = 10;
	double BufferTime1, BufferTime2, BufferTime3 = 999999;
	int NearestPrecedingCarID, n = 1;

	//Desired max speed should be lower at the corner.
	DistanceToNextEventPoint = Distance_BetweenTwoNode(CurrentEvent.x,
CurrentEvent.y, CurrentPOS_x, CurrentPOS_y);

	/* Cosider what to do when close to the end of a road block */
	if((DistanceToNextEventPoint != 0) && (CurrentVelocity != 0))
	{
		BufferTime3 = roundf(DistanceToNextEventPoint / CurrentVelocity);
	}

	if(DistanceToNextEventPoint >= 30)
	{
		DesiredMaxSpeed = MaxVelocity; // meter/second
	}
	else
	{
		// If close to the next event point, don't drive too fast.
```

```
                DesiredMaxSpeed = rand() % 6 + 5; // 10 m/s => 36 km/hr at maxmimum
        }

        n =
getNearestNodePositionAlongTheSpecifiedDirectionAndRangeViewedOnGUIScreen(myTCPsockfd,
mynid, CurrentPOS_x, CurrentPOS_y, 0, CurrentDirection, VISIBILITY_SCALE_IN_DEGREE,
VISIBILITY_SCALE_IN_DISTANCE, 1, NearestPrecedingCarPOS_x, NearestPrecedingCarPOS_y,
NearestPrecedingCarPOS_z, NearestPrecedingCarDirection, NearestPrecedingCarID, 1);

        double vt1, vt2;

        vt1 = DesiredMaxSpeed;
        vt2 = DesiredMaxSpeed;

        // set default time, make car run as fast as possible
        /* Put a = MaxAcceleration into E1 and get BufferTime1 */
        BufferTime1 = fabs(vt1 - CurrentVelocity) / MaxAcceleration;
        BufferTime2 = BufferTime1;

        if(n == 0)
        {
                /* There's a car in front of me. */

                n = checkIfOnTheSameLane(CurrentPOS_x, CurrentPOS_y,
NearestPrecedingCarPOS_x, NearestPrecedingCarPOS_y, CacheOrNot, &myCache,
&otherCache);
                if(n != 0)
                {
                        /* The preceding car is on the same lane with me. */

                        double distance = 999999;

                        // Get the distance between me and the preceding car.
                        distance = Distance_BetweenTwoNode(CurrentPOS_x, CurrentPOS_y,
NearestPrecedingCarPOS_x, NearestPrecedingCarPOS_y);
                        if((distance <= 0.3) && (oldCollisionCarID !=
NearestPrecedingCarID)){
                                printf("Collision of Car %d (%lf, %lf) and Car %d (%lf,
%lf).\n",
                                                mynid, CurrentPOS_x, CurrentPOS_y,
                                                NearestPrecedingCarID,
NearestPrecedingCarPOS_x,
                                                NearestPrecedingCarPOS_y);
                                oldCollisionCarID = NearestPrecedingCarID;
                        }

                        // Get preceding car's velocity.
                        getCurrentMovingSpeed(myTCPsockfd, NearestPrecedingCarID,
NearestPrecedingCarSpeed, 1);

                        if(agentReceivedBrokenCarMsg == 1){
                          printf("VERIFICANDO nova rota!!!\n");
                                if((brokenCarDirection != ExpectedDirection) &&
(brokenCarID == NearestPrecedingCarID)){
                                        /* Broken car is on the same lane but I will switch
to other lane.
                                         * So, the preceding car shouldn't affect me.
                                         */
                                        NearestPrecedingCarSpeed = 999999;
                                        distance = 9999999;
                                        BufferTime1 = 999999;
```

```
                        }
                }

                /* Too close to the preceding car. */
                if(distance <= 3)
                {
                        BufferTime = 0;
                        vt = 0;
                        return;
                }

                /* Consider the effects of the preceding car's velocity and
position. */
                if(NearestPrecedingCarSpeed == 0)
                {
                        vt1 = NearestPrecedingCarSpeed;

                        if(CurrentVelocity == 0)
                        {
                                vt = 0;
                                BufferTime = 0;
                                return;
                        }
                        /* Set vt to 0 and put vt into E3.
                         * We get t = 2 * S / v0
                         */
                        BufferTime1 = 2 * (distance - 3) / CurrentVelocity;
                }
                else
                {
                        if(distance < 2 * SAFETY_INTERVEHICLE_DISTANCE)
                        {
                                /* Set the target velocity to
NearestPrecedingCarSpeed. */
                                double tmpV;

                                tmpV = CurrentVelocity + NearestPrecedingCarSpeed;
                                BufferTime1 = 2*(distance -
SAFETY_INTERVEHICLE_DISTANCE)/tmpV;
                                vt1 = NearestPrecedingCarSpeed;
                        }
                        else
                        {
                                /* Preceding car is far from me.
                                 * Using default vt1 and BufferTime1.
                                 */
                        }
                }
        }
    }

    /* Cosider what to do when close to the traffic light */
    if((SeenTheTrafficLightOrNot == 1) && (SigLight == RED || SigLight == YELLOW))
    {
            double DistanceToTrafficLight = 9999999;

            DistanceToTrafficLight = Distance_BetweenTwoNode(SigPOS_x, SigPOS_y,
CurrentPOS_x, CurrentPOS_y);
            if(DistanceToTrafficLight <= 2*SAFETY_INTERVEHICLE_DISTANCE){
                    // Getting close to the traffic light, need to slow down a little
```

```
                    vt2 = SelectMinimum(0 - (DistanceToTrafficLight -
SAFETY_INTERVEHICLE_DISTANCE) / 10 , DesiredMaxSpeed);
             }
             if((DistanceToTrafficLight != 0) || (CurrentVelocity != 0))
             {
                    if((BufferTime2 = DistanceToTrafficLight / CurrentVelocity) < 0)
                    {
                           BufferTime = 0;
                           return;
                    }

                    /*
                     * From E1, put vt = 0 and we get 0 = v0 + at
                     * Ignore (1/2)*a*(t^2) part in E2 because t is very small.
                     * We get
                     * E1': 0 = v0 + a*t
                     * E2': S = v0*t
                     *
                     * From E1' and E2' we get v0 = sqrt(a * S)
                     */
                    double scv1 = 0; // safe current velocity

                    scv1 = sqrt(MaxDeceleration * DistanceToTrafficLight);

                    /* safe check */
                    if(CurrentVelocity >= scv1){
                           /* Dangerous Velocity!! Need to slow down fast. */
                           BufferTime = 0;
                           return;
                    }
             }
       }

       BufferTime = SelectMinimum(BufferTime1, BufferTime2);
       BufferTime = SelectMinimum(BufferTime, BufferTime3);

       vt = SelectMinimum(vt1, vt2);
}

double DetermineAcceleration()
{
       /*
        * The following implementation is based on
        * VATSIM: A Simulator for Vehicles and Traffic ,
        * Jia Lei Keith Redmill Umit Ozguncr ,
        * Department of Electrical Engineering, The Ohio State University,
        * 2001 IEEE Intelligent Transportation Systems Conference Proceedings
        *
        *
        * vt:    the target velocity
        * bfTime: the buffer time to achieve vt.
        */
       double vt, bfTime;

       DetermineVc(bfTime, vt);

       double acc;

       if(bfTime > 0)
       {
              /* Equation: vt = CurrentVelocity + acc * bfTime */
```

```
                acc = roundf((vt- CurrentVelocity) / bfTime * 10);
                acc /= 10;
        } else {
                /* Emergent Stop */
                acc = -MaxDeceleration;
        }

        //Our behavior was changed due to warning from RSU
        if(WarningDecelerate < 0)
                return -MaxDeceleration;
        if(acc > MaxAcceleration)
                return MaxAcceleration;
        if(acc < -MaxDeceleration)
                return -MaxDeceleration;
        else
                return  acc;
}

int main()
{
        int sleepingPeriod = 100000;
        init();
        while(1)
        {
          printf("\n========NOVO_CAR_AGENT=========\n");
                if(ITS_NET)
                reportMyStatusToAGroupOfNode();
                fflush(stdout);

                /* FIXME: If necessary, make every node wake up in different time.
                 * (Avoid collision of broadcast pkt on mac)
                 */
                //usleepAndReleaseCPU(myTCPsockfd, mynid, SLEEPING_PERIOD, 1);
                usleepAndReleaseCPU(myTCPsockfd, mynid, sleepingPeriod, 1);
                WarningDecelerate++;
                WakedTimes++;
                gettimeofday(&now, 0);

                if(ITS_NET){
                        // receive msg from other nodes
                        receiveMsg();
                }

                int n = 0;
                double *CandidateDirection = NULL;

                n = getCurrentPosition(myTCPsockfd, mynid, CurrentPOS_x, CurrentPOS_y,
CurrentPOS_z, 1);

                // See the Traffic light signal.
                n = getTheNearestTrafficLightInfrontOfMe(myTCPsockfd, mynid,
CurrentPOS_x, CurrentPOS_y, CurrentDirection, 100.0, SigGID, SigLight, SigPOS_x,
SigPOS_y, SignalIndex, 1);
                if(n == 1)
                {
                        SeenTheTrafficLightOrNot = 1;
                }
                else
                        SeenTheTrafficLightOrNot = 0;

                // get next moving event (point) from event queue
```

```
		CurrentEvent = EventQueue.front();

		getCurrentMovingSpeed(myTCPsockfd, mynid, CurrentVelocity, 1);
		CurrentAcceleration = DetermineAcceleration();

		setCurrentSpeedAcceleration(myTCPsockfd, mynid, CurrentAcceleration, 1);

		if(EventQueue.empty() == false)
		{
			n = ReachTheNextTriggerPointOrNot();
			// n = 1; Reach the event point.
			// n = 0; Unreach the event point.
		}
		else
		{
			n = 1;
		}

		if(n == 0)
		{
			//Not reach the event point, do nothing, keep going.
			sleepingPeriod = 100000;
			continue;
		}
		else if(n == 1)
		{
			sleepingPeriod = 100000;
			/* create new event queue */
			if(((int)EventQueue.size()) == 0)
			{
				/* End of taking turns, so the queue became empty */
				int numOfDirections;
				double CorrectedPOS_x, CorrectedPOS_y;
				double a, b, c;

				// correct my position into the middle of the road
				n = selfCorrectness(CurrentPOS_x, CurrentPOS_y,
CorrectedPOS_x, CorrectedPOS_y, CacheOrNot, &myCache);
				if(n == 1)
				{
					CurrentPOS_x = CorrectedPOS_x;
					CurrentPOS_y = CorrectedPOS_y;
				}
				n = setCurrentWaypoint(myTCPsockfd, mynid, CurrentPOS_x,
CurrentPOS_y, CurrentPOS_z, 1);

				/* get front node block ID which connected by current edge.
				 * Note:
				 *    node block ID is the same as signal group ID
				 */
				getFrontNID(CurrentPOS_x, CurrentPOS_y, SigGID, CacheOrNot,
&myCache);

				int RoadType;
				double endPOS_x, endPOS_y;

				n = getCurrentRoadInformation(CurrentPOS_x, CurrentPOS_y,
numOfDirections, CandidateDirection, a, b, c, endPOS_x, endPOS_y, RoadType,
CacheOrNot, &myCache);
				if(n <= 0){
```

```
                                printf("Warning_1_2: Node is not on the lane!! Node
%d position(%lf, %lf), current direction %lf, eventPos (%lf, %lf), eventDirection
%lf\n",
                                                mynid, CurrentPOS_x, CurrentPOS_y,
CurrentDirection, CurrentEvent.x, CurrentEvent.y, CurrentEvent.direction);
                                fflush(stdout);
                                stopSimulation(myTCPsockfd, mynid);
                        }

                        if(RoadType == ROAD_TYPE_NODE)
                        {
                                double distanceToNextEventPoint =
Distance_BetweenTwoNode(CurrentPOS_x, CurrentPOS_y, CurrentEvent.x, CurrentEvent.y);
                                double angleDiff = fabs(CurrentDirection -
CurrentEvent.direction);
                                if(angleDiff > 180)
                                        angleDiff = 360 - angleDiff;

                                if(((CurrentEvent.x <= 0.000001) && (CurrentEvent.y
<= 0.000001)) ||
                                        ((distanceToNextEventPoint >= roadWidth) &&
(angleDiff < 90)))
                                {
                                        /*
                                         * Node is in the node block at beginning ||
                                         * Node skips one lane and directly enters
into the node block
                                         * (It happens when the road length is very
small)
                                         */
                                        double exitPOS_x, exitPOS_y, exitDirection;
                                        double nextDirection;

                                        n = getCurrentNodeExit(CurrentPOS_x,
CurrentPOS_y, exitPOS_x, exitPOS_y, exitDirection, nextDirection, CacheOrNot,
&myCache);
                                        if(n < 0)
                                        {
                                                printf("Node[%d] getCurrentNodeExit
error\n", mynid);
                                        }

                                        CurrentDirection = exitDirection;
                                        // set current moving direction viewed on GUI
screen
                                        n =
setCurrentMovingDirectionViewedOnGUIScreen(myTCPsockfd, mynid, CurrentDirection, 1);

                                        FillTheQueue(0, NULL, NULL, NULL, exitPOS_x,
exitPOS_y, nextDirection);
                                }
                                else
                                {
                                        /* After the turn, if car is still on the
previous road,
                                         * make this car go to the final point of the
previous road
                                         * one more time.
                                         */
                                        FillTheQueue(0, NULL, NULL, NULL,
CurrentEvent.x, CurrentEvent.y, CurrentEvent.direction);
```

```
                                }
                                free(CandidateDirection);
                                CandidateDirection = NULL;
                                continue;
                        }

                        CurrentDirection = RandomDirection(numOfDirections,
CandidateDirection);
                        free(CandidateDirection);
                        CandidateDirection = NULL;

                        n = setCurrentMovingDirectionViewedOnGUIScreen(myTCPsockfd,
mynid, CurrentDirection, 1);

                        n = getNextRoadInformation(CurrentPOS_x, CurrentPOS_y,
CurrentDirection, numOfDirections, CandidateDirection, a, b, c, CacheOrNot, &myCache);
                        if(n <= 0){
                                printf("Warning_1_3: Cannot find the next road!!
Node %d position(%lf, %lf)\n", mynid, CurrentPOS_x, CurrentPOS_y);
                                fflush(stdout);
                                stopSimulation(myTCPsockfd, mynid);
                        }

                        if(numOfDirections == 0){
                                printf("Error: numOfDirections can't be zero\n");
                                fflush(stdout);
                                stopSimulation(myTCPsockfd, mynid);
                        }

                        ExpectedDirection = RandomDirection(numOfDirections,
CandidateDirection);
                        if(agentReceivedBrokenCarMsg == 1){
                                printf("Verificando nova rota 2!!!");
                                if(numOfDirections >= 2){
                                        while(ExpectedDirection ==
brokenCarDirection){
                                                ExpectedDirection =
RandomDirection(numOfDirections, CandidateDirection);
                                        }
                                }
                        }
                        free(CandidateDirection);
                        CandidateDirection = NULL;

                        int numOfTurns;
                        double *turningPOS_x = NULL, *turningPOS_y = NULL,
*DirectionQueue = NULL;
                        double FirstPointAfterTheTurn_x, FirstPointAfterTheTurn_y;

                        n = takeATurn(CurrentPOS_x, CurrentPOS_y, CurrentDirection,
ExpectedDirection, numOfTurns, turningPOS_x, turningPOS_y, FirstPointAfterTheTurn_x,
FirstPointAfterTheTurn_y, DirectionQueue, CacheOrNot, &myCache);
                        if(n <= 0)
                        {
                                printf("Error: cannot get the taking turns info,
error no. %d, curPos (%lf, %lf), curDirection %lf, expectedDirection %lf\n", n,
CurrentPOS_x, CurrentPOS_y, CurrentDirection, ExpectedDirection);
                                fflush(stdout);
                                stopSimulation(myTCPsockfd, mynid);
                        }
```

```
                        FillTheQueue(numOfTurns, turningPOS_x, turningPOS_y,
DirectionQueue, FirstPointAfterTheTurn_x, FirstPointAfterTheTurn_y,
ExpectedDirection);

                        delete turningPOS_x;
                        delete turningPOS_y;
                        delete DirectionQueue;

                }
                else if(((int)EventQueue.size()) > 0)
                {
                        //Pop an item from the queue
                        CurrentDirection = CurrentEvent.direction;
                        CurrentPOS_x = CurrentEvent.x;
                        CurrentPOS_y = CurrentEvent.y;
                        n = setCurrentWaypoint(myTCPsockfd, mynid, CurrentPOS_x,
CurrentPOS_y, CurrentPOS_z, 1);
                        n = setCurrentMovingDirectionViewedOnGUIScreen(myTCPsockfd,
mynid, CurrentDirection, 1);
                }
        }
        else
        {
                printf("Node[%d]: Error for reached or not, n %d\n", mynid, n);
                fflush(stdout);
                stopSimulation(myTCPsockfd, mynid);
        }
    }
      }
```

APÊNDICE D – MeuRSUagent.cc

Este arquivo foi criado para ser o processo executado pelas RSUs do simulador NCTUns.

```
/*
 * Copyright (c) from 2000 to 2010
 *
 * Network and System Laboratory
 * Department of Computer Science
 * College of Computer Science
 * National Chiao Tung University, Taiwan
 * All Rights Reserved.
 *
 * This source code file is part of the NCTUns 6.0 network simulator.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation is hereby granted (excluding for commercial or
 * for-profit use), provided that both the copyright notice and this
 * permission notice appear in all copies of the software, derivative
 * works, or modified versions, and any portions thereof, and that
 * both notices appear in supporting documentation, and that credit
 * is given to National Chiao Tung University, Taiwan in all publications
 * reporting on direct or indirect use of this code or its derivatives.
 *
 * National Chiao Tung University, Taiwan makes no representations
 * about the suitability of this software for any purpose. It is provided
 * "AS IS" without express or implied warranty.
 *
 * A Web site containing the latest NCTUns 6.0 network simulator software
 * and its documentations is set up at http://nsl.cs.nctu.edu.tw/nctuns.html.
 *
 * Project Chief-Technology-Officer
 *
 * Prof. Shie-Yuan Wang <shieyuan@csie.nctu.edu.tw>
 * National Chiao Tung University, Taiwan
 *
 * 01/15/2010
 */

/*
Scenario Description:
       This road side unit ( RSU ) will receive message from Agent and forward message
to other Agent.
       Agent can be RSUAgent or CarAgent.

       By automatic vehicle group.
       5/31/2007
 */

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <fcntl.h>
#include <time.h>
#include <math.h>
```

```
#include "math_fun.h"
#include "tactic_api.h"
#include "sock_skel.h"
#include "road.h"
#include "conexao_BD.h"
//#include "brodcast.h"

#define ITS_SHOW_MSG 1

int mynid, n;
int myUDPsockfd;
int agentUDPportNum = 4000;
struct timeval now;
double lastTime = 0.0;
char portNumStr[32];

class msgSequence{
       public:
       u_int32_t nid;
       int seqNum; // sequence number
};
vector<msgSequence> msgSeq;

void RecebeAtualizacaoReputacao()
{
  FILE *fp;
  char arquivo[30] = "RSU3";
  fp = lerBufferAtualizacaoRSUAgent(arquivo);
  char str[30];
  char veiculo[5] = "";
  char valor[5] = "";
  int i = 1;
  int idVeiculo = 0;
  int valorReputacao = 0;
  int res = 0;

  while(fgets(str,30,fp))
  {
    printf("STR = %s\n",str);
       i = 0;
       while(str[i] != ';')
       {
         veiculo[i] = str[i];
         i++;
       }
       veiculo[i] = '\n';
       i++;

       int j = 0;
       while(str[i] != '\n')
       {
         valor[j] = str[i];
         i++;
         j++;
       }
       valor[j] = '\n';

       idVeiculo = atoi(veiculo);
       valorReputacao = atoi(valor);
       printf("\nID veículo = %d\n",idVeiculo);
       printf("Reputação %d\n",valorReputacao);
```

```
        UpdateReputacao(mynid,idVeiculo,valorReputacao);

        //Limpa variáveis
        for(i = 0;i<5;i++)
        {
          valor[i] = '\n';
          veiculo[i] = '\n';
        }
  }
}

void RespondeConsultaReputacao()
{
  FILE *fp;
  char arquivo[30] = "RSU3";
  fp = lerBufferConsultaReputacaoRSUAgent(arquivo);
  char str[30];
  char veiculoSolicitante[5] = "";
  char veiculoConsultado[5] = "";
  int i = 1;
  int idVeiculoSolicitante = 0;
  int idVeiculoConsultado = 0;
  int valorReputacao = 0;
  char reputacaoRespondida[10];


  while(fgets(str,30,fp))
  {
    printf("STR = %s\n",str);
       i = 0;
       while(str[i] != ';')
       {
         veiculoSolicitante[i] = str[i];
         i++;
       }
       veiculoSolicitante[i] = '\0';
       i++;

       int j = 0;
       while(str[i] != '\n')
       {
         veiculoConsultado[j] = str[i];
         i++;
         j++;
       }
       veiculoConsultado[j] = '\n';

       idVeiculoSolicitante = atoi(veiculoSolicitante);
       idVeiculoConsultado = atoi(veiculoConsultado);
       printf("\nID Veículo Solicitante = %d\n",idVeiculoSolicitante);
       printf("ID Veículo Consultado %d\n",idVeiculoConsultado);
       valorReputacao = RetornaValorDaReputacao(mynid, idVeiculoConsultado);

       printf("VALOR REPUTAÇÃO NOVO %d\n",valorReputacao);
       //void RespondeReputacaoBuffer(char veiculo[10], char valor[10])

       sprintf(reputacaoRespondida, "%d" , valorReputacao);
       printf("VALOR %s\n",reputacaoRespondida);
       char f[10] = "1";
       RespondeReputacaoBuffer(veiculoSolicitante, reputacaoRespondida);
```

```
        //Limpa variáveis
        for(i = 0;i<5;i++)
        {
          veiculoConsultado[i] = '\0';
          veiculoSolicitante[i] = '\0';
        }
    }
}

void reportMyStatusToAGroupOfNode(struct agentClientReportStatus msg)
{
        //double lastTime = (double) now.tv_sec * 1000000 + (double) now.tv_usec;
        char hostname[128];
        int value;
        int len;

        sockaddr_in cli_addr;

//Braod cast start
        value = 1;
        setsockopt( myUDPsockfd , SOL_SOCKET , SO_BROADCAST , &value , sizeof(value) );
        len =  sizeof(struct sockaddr);
        strcpy(hostname, "1.0.1.255");
        memset( &cli_addr, 0, sizeof(cli_addr) );
        cli_addr.sin_family = AF_INET;
        cli_addr.sin_port   = htons(agentUDPportNum);
        cli_addr.sin_addr.s_addr = inet_addr( hostname );
        int n = sendto(myUDPsockfd, &msg, sizeof(struct agentClientReportStatus), 0,
(struct sockaddr *) &cli_addr, len);
         printf("valor de n broadcast sent n= %d", n);

        if (n < 0) {
                printf("Agent (%d) sendto failed\n", mynid);
        }
        else{
                if(ITS_SHOW_MSG){
                        printf("RSU_AGENT: Car Agent (%d) broadcast pkt, Size %d, seqNum
%d, packet type==%d\n", mynid, sizeof(struct agentClientReportStatus), msg.seqNum,
msg.type);
                }
        }
}

void receiveMsg(){
        int len = sizeof(struct sockaddr);
        int n = 1;
        sockaddr_in cli_addr;
        while( n > 0 ){
                agentClientReportStatus msg;
                n = recvfrom(myUDPsockfd, &msg, sizeof(struct agentClientReportStatus),
0, (struct sockaddr *) &cli_addr, (socklen_t*)&len);
        //printf("Valor de n recebendo n = %d",n);
        //system("pause");
        if (n > 0) { // Someone has sent me a message, I can do something by this
message.
                        if ((msg.type != AGENT_CLIENT_REPORT_STATUS) && (msg.type !=
AGENT_CLIENT_IS_A_BROKEN_CAR)) {
                                printf("Agent(%d): no such message type.\n", mynid);
                        }
                        if(msg.seqNum > 0){
                                int nidExists = 0;
```

```
                        for (int i=0 ; i<(int)(msgSeq.size()); i++)
                        {
                                if(msg.nid == msgSeq[i].nid)
                                {
                                        nidExists=1;
                                        if(msg.seqNum > msgSeq[i].seqNum)
                                        {
                                                printf("RSUAgent(%d) receive message of
agent %d.\n", mynid, msg.nid);
                                                msgSeq[i].seqNum = msg.seqNum;
                                                reportMyStatusToAGroupOfNode(msg);
                                        }
                                        else{
                                                printf("RSUAgent(%d): Message is
already received from agent (%d)_%d\n", mynid, msg.nid, msg.seqNum);
                                        }
                                        break;
                                }
                        }
                        if(nidExists == 0)
                        {
                                if(ITS_SHOW_MSG){
                                        printf("RSUAgent(%d) receives message of agent
%d.\n", mynid, msg.nid);
                                }
                                reportMyStatusToAGroupOfNode(msg);
                                msgSequence temp;
                                temp.nid = msg.nid;
                                temp.seqNum = msg.seqNum;
                                msgSeq.push_back(temp);
                        }
                }
        }
    }
}

int main(int argc, char *argv[]) {
     mynid = getMyNodeID();

     sprintf(portNumStr, "%d", agentUDPportNum);
     myUDPsockfd = passiveUDP(portNumStr);
     printf("Agent(%d) created myUDPsockfd %d\n", mynid, myUDPsockfd);
     if (myUDPsockfd < 0) {
       printf("Agent(%d): Creating myUDPsockfd failed\n", mynid);
       exit(0);
     }
     n = fcntl(myUDPsockfd, F_SETFL, O_NONBLOCK);
     while (1) {
       //gettimeofday(&now, 0);
       receiveMsg();
       //printf("MYNID = %d\n",mynid);
       printf("\n=============RSUAgent================ \n");
       //InserirRegistro(mynid, 3);
       //UpdateReputacao(mynid,5,33);
       //RecebeAtualizacaoReputacao();
       //RespondeConsultaReputacao();
//     AtualizaUC("2", "10");
       usleep(100000);
       /* The above control is exerted every 0.1 second. */
    }
        }
```

ANEXO A - INSTALAÇÃO DO SIMULADOR NCTUNS

Cada versão de NCTUns é desenvolvido para uma versão específica do Fedora Core. A razão desta exigência é as modificações que o instalador do simulador faz em certas partes do kernel do Linux. Obviamente, a fim de evitar o mau funcionamento do simulador, o kernel do sistema não deve ser atualizado a partir do que vem por padrão com o lançamento do Fedora Core.

A distribuição Linux que funciona com a versão do NCTUns 6.0 e foi totalmente testado é da Red Hat Fedora 12. Outras distribuições Linux, como o Ubuntu podem trabalhar com NCTUns pois usam o mesmo kernel do Linux como o Fedora. No entanto, configurações e definições sobre essas distribuições podem ser diferentes daqueles encontradas no Fedora. É de responsabilidade do usuário ajustar as configurações, se ele (ela) gostaria de instalar o NCTUns 6.0 em uma distribuição Linux diferente do Fedora 12. Como a NSL (*Network and System Laboratory*) não testou o NCTUns 6.0 em outras distribuições Linux, não é garantido que este venha a funcionar corretamente em diferentes distribuições Linux.

Passos da instalação

Inicialmente é necessário ter permissão de administrador (*root*) para efetuar as instalações. Para garantir a instalação do simulador sem erros e recomendável a instalação de certas dependências.:

```
# yum install gcc-c++ readline-devel rsh-server xinetd tclreadline-devel
```

Em seguida, descompacte a distribuição NCTUns e execute o script *install.sh*. Todo o software é instalado na pasta **/usr/local/nctuns**.

Durante o processo, o instalador pergunta se um usuário nctuns deve ser criado, se o núcleo tem de ser corrigido, e se o SELinux devem ser desativados. O recomendável é responder todas essas perguntas com "*yes*".

Quando a instalação estiver concluída, o sistema deve ser reinicializado e iniciado com o novo kernel selecionando a opção "NCTUNS" no carregador de boot GRUB. O último passo é adicionar as seguintes variáveis de ambiente para o arquivo *bashrc*.:

```
export NCTUNSHOME=/usr/local/nctuns
export NCTUNS_TOOLS=$NCTUNSHOME/tools
```

```
export NCTUNS_BIN=$NCTUNSHOME/bin
export PATH=${PATH}:${NCTUNS_BIN}
```

Iniciando o simulador NCTUns

Após ter concluído a instalação e reiniciado o sistema é possível iniciar o simulador NCTUns. Como foi descrito anteriormente o NCTUns é dividido em três componentes: *coordinator, dispatcher e nctunsclient*. Estes componentes devem ser executados como *root* e não necessariamente precisam estar na mesma máquina.

Uma maneira recomendável para executar estes componentes é executá-los em terminais diferentes, desta maneira é possível verificar em tempo de execução algumas informações sobre a simulação. Para executar os processos é só digitar na sequência os comandos nos seus respectivos terminais:

```
dispatcher
coordinator
nctunsclient
```

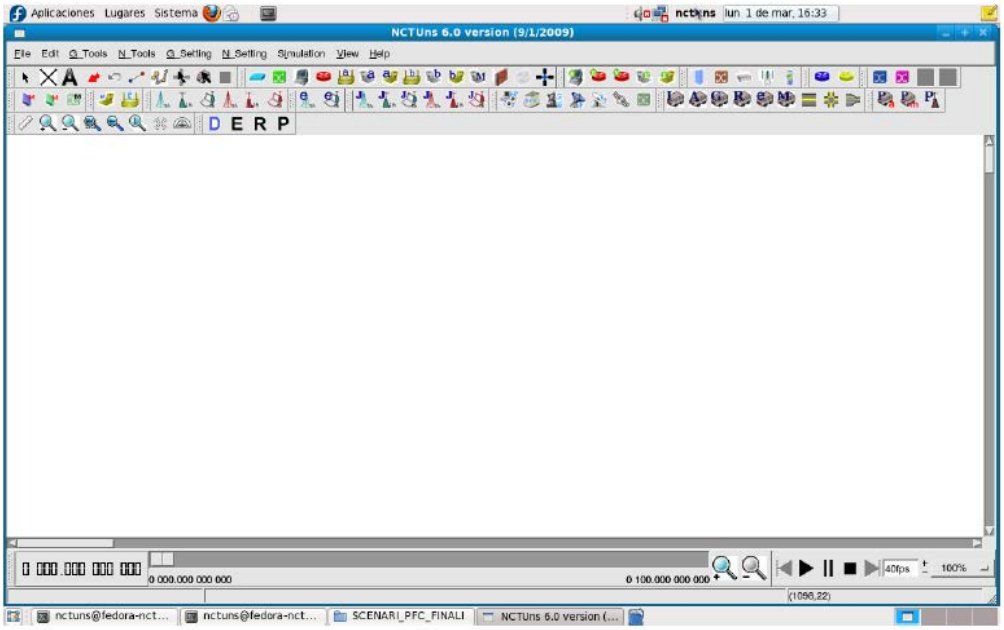
ANEXO B - INTERFACE DO NCTUNS

Para apresentar a produtividade e características do simulador NCTUns, esta seção apresenta algumas janelas da interface do ambiente GUI.

Tela Inicial

Sempre que se executa o programa GUI a tela inicial será exibida (Figura 52). Nesta janela tem-se acesso aos menus e diversos ícones de atalhos para modelagem dos cenários e execução da simulação.

Figura 52 - Tela inicial do NCTUns



Desenhando um cenário

Para desenhar um cenário é preciso primeiramente alterar para o modo topologia (*Draw Topology)* (**Erro! Fonte de referência não encontrada.**). O editor de topologia fornece ma maneira conveniente e intuitiva para construir graficamente uma topologia de rede. A rede construída pode ser uma rede com fio ou uma rede móvel sem fios. Para as aplicações, uma rodovia de estradas também pode ser construída. Devido ao design de fácil utilização, todas as

operações de GUI podem ser realizadas de forma fácil e intuitiva. Para desenhar a rodovia, deve-se selecionar um dos segmentos ***"ITS road segment"*** (Figura 54).

Figura 53 - Modo Desenhar Topologia

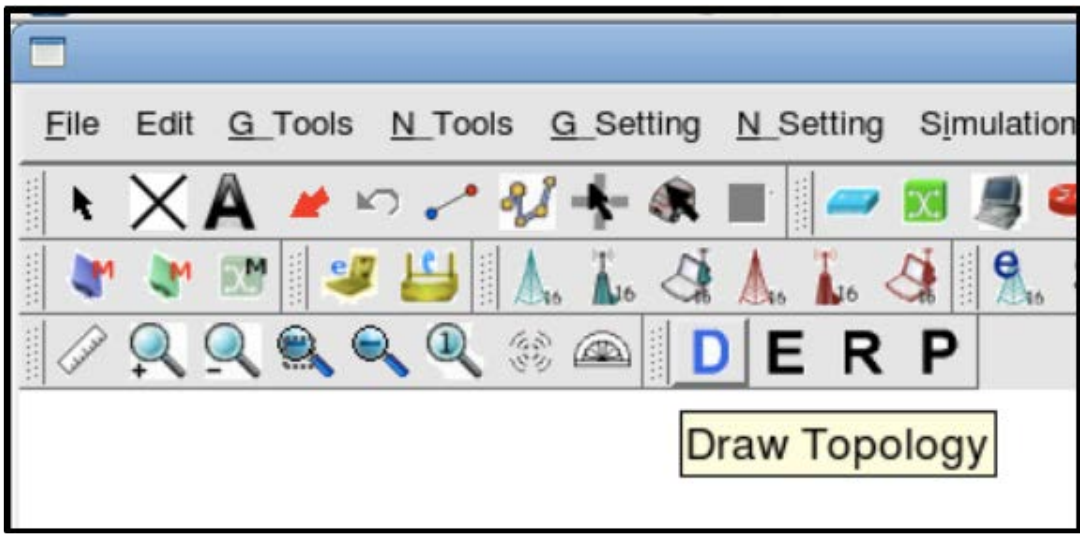
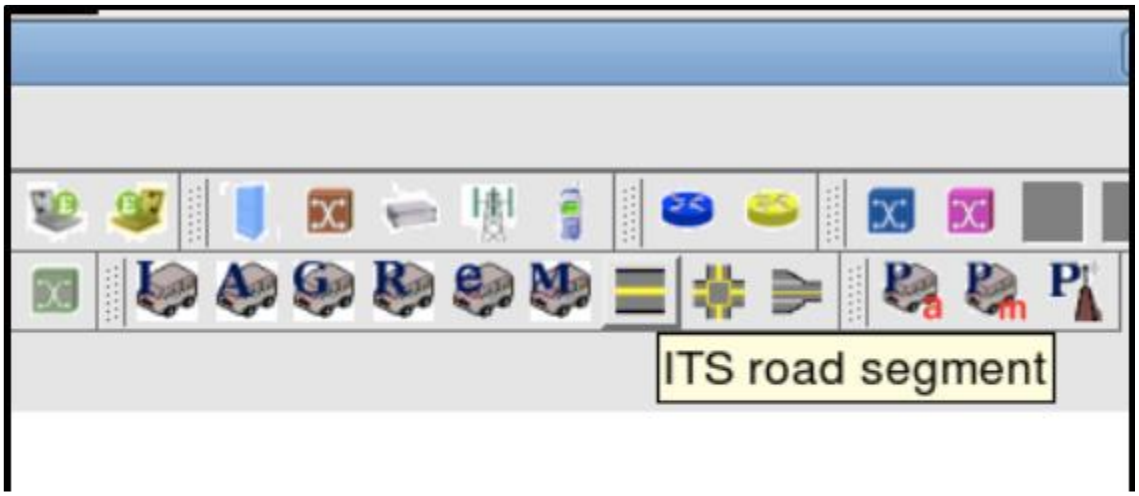
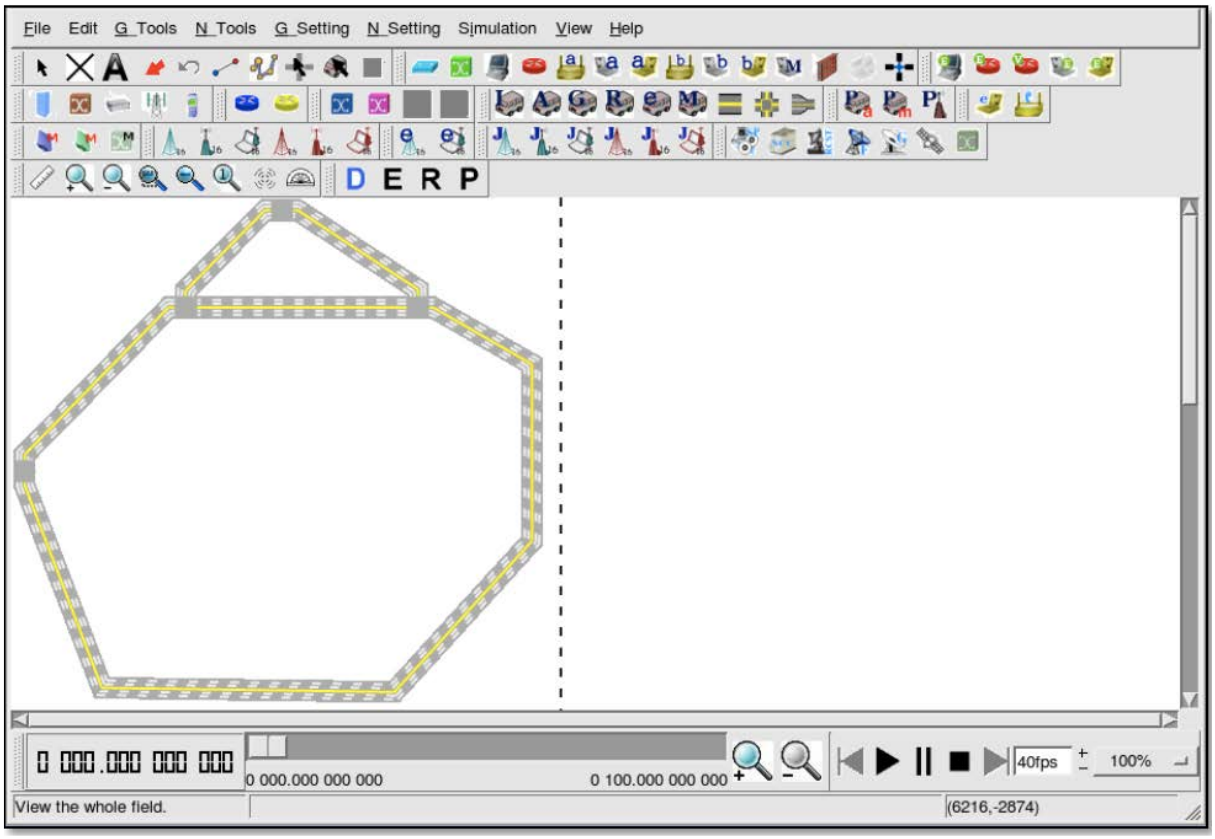


Figura 54 - Itens ITS road segment



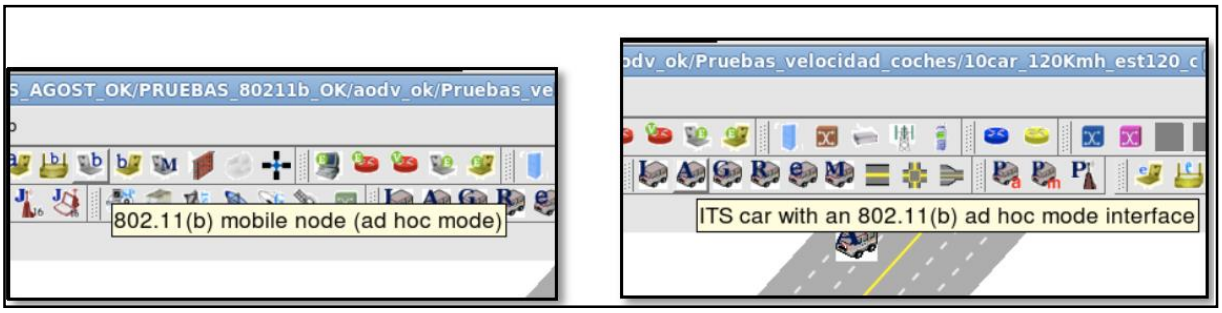
Quando se desenha uma rodovia é importante lembrar que os caminhos devem ser fechados e deve ter ao menos uma intersecção, caso contrário os veículos não se movimentarão. A Figura 55apresenta um exemplo.

Figura 55 - Exemplo de um cenário de rodovia



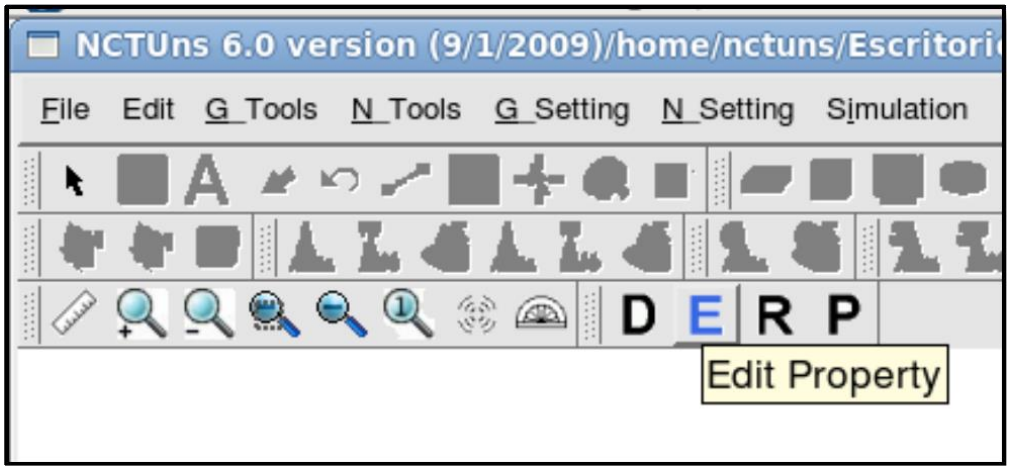
Após desenhar a rodovia, os veículos e RSU podem ser posicionados na estrada. Isto é feito arrastando os ícones correspondentes aos componentes ao cenário. É possível adicionar componentes de diversos tipos de protocolo: 802.11b, 802.11p, etc. (Figura 56).

Figura 56 - Componentes de RSU e veículos.



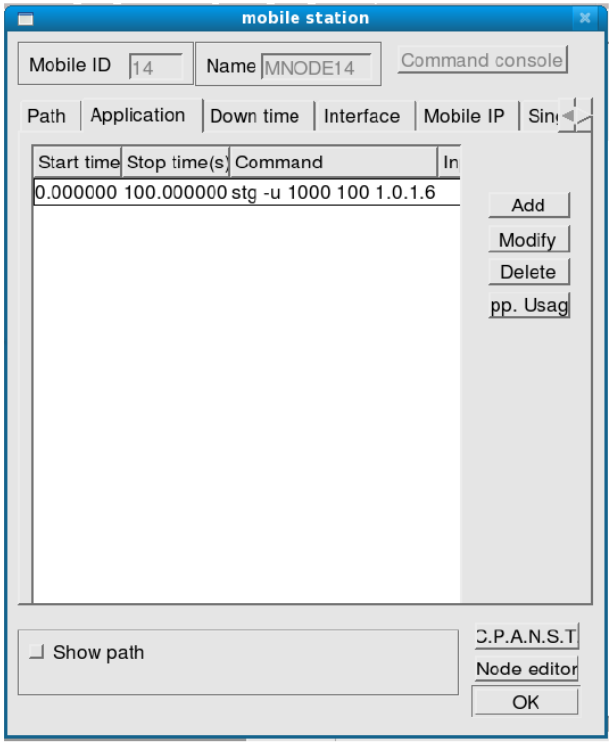
Finalizada a distribuição e posicionamento dos nós da rede, pode-se configurar certas propriedades destes nós. Para isso deve-se mudar para o modo *"Edit Property"* (Figura 57).

Figura 57 - Selecionar modo Edit Property



Um dispositivo de rede (nó) pode ter muitos atributos. Definir e modificar os atributos de um nó da rede pode ser feito facilmente. Basta fazer duplo clique sobre o ícone de um nó da rede. Uma caixa de diálogo de atributo que pertence a este nó irá aparecer (Figura 58). Um usuário pode, então, definir os atributos do dispositivo na caixa de diálogo.

Figura 58 - Caixa de diálogo para configurar propriedades do nó



Depois de finalizadas todas as configurações da rede, pode-se então executar a simulação. Para isso, deve-se selecionar o modo *"Run Simulation"* (Figura 59), neste momento a topologia é salva e alguns arquivos são gerados. Para iniciar a simulação basta selecionar a opção "Simulation" > "Run".

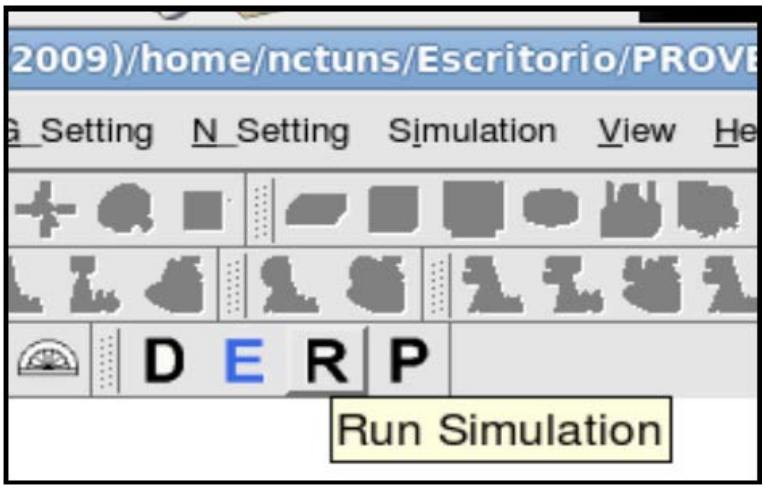



Figura 59 - Modo *Run Simulation*