Sistema de Semáforos Inteligentes Suportado por Comunicações V2X

Ricardo Silva PG54188, Ivo Ribeiro PG53886, and Paulo Oliveira PG54133

Universidade do Minho

Abstract. Este relatório apresenta o desenvolvimento e implementação de um sistema de semáforos inteligentes suportado por comunicações V2X. O objetivo principal é melhorar a fluidez do tráfego e a segurança rodoviária utilizando comunicações veículo-para-tudo (V2XMessage) para adaptar dinamicamente os sinais de tráfego com base nas condições de trânsito em tempo real. O sistema foi desenvolvido em etapas, começando com um protótipo simplificado, envolvendo um semáforo físico, até um sistema de semáforos totalmente virtual e distribuído. As várias soluções de gestão do tráfego serão realizadas utilizando a plataforma **Eclipse MOSAIC** em conjunto com um cenário desenvolvido no *SUMO*.

Keywords: eclipse Mosaic · Gestão de tráfego · semáforos inteligentes · Comunicação V2X · Encaminhamento Multihop.

– ***Etapa 1***

1 Introdução

Nesta fase do projeto, vamos testar uma versão simplificada do sistema. Na verdade, Os veículos apenas devem adquirir dados e enviá-los via *multicast* para um único salto de distância. Devemos definir claramente quais informações serão enviadas, o formato das mesmas e a periodicidade adequada para o envio. Os dados recebidos pela *RSU* dentro do seu raio de ação devem ser reunidos no servidor. Com base nesses dados, o algoritmo no servidor deve determinar quais vias terão sinal verde e quais terão sinal vermelho. O objetivo do algoritmo é o de melhorar a fluidez do tráfego, evitando, ao mesmo tempo, que os veículos fiquem parados por muito tempo nos semáforos.

2 Especificação do sistema e dos protocolos de suporte

Esta secção fornece uma visão geral dos componentes e mecanismos essenciais para o funcionamento do sistema de semáforos inteligentes suportado por comunicações V2X. Será abordada a topologia da rede veicular, a infraestrutura que inclui as aplicações instaladas nos veículos e nas unidades rodoviárias, as primitivas de comunicação que definem as operações básicas de troca de mensagens, o formato das mensagens protocolares (PDU) utilizadas para a comunicação entre os componentes, e as interações que detalham os fluxos de comunicação e processos de troca de informações entre os veículos, RSUs e o servidor.

2.1 Topologia

A topologia da rede veicular consiste em num cruzamento de vias onde veículos equipados com Unidades de Comunicação Embarcadas (OBU - On-Board Units) circulam e um *RSU* instalada no centro do cruzamento. Neste caso em especifico, a nossa simulação ocorre num cenário simples composto por um cruzamento de quatro vias, bastante comum nas estruturas rodoviárias, daí o nosso grande interesse em poder incorporar uma solução por nós desenvolvida nos cenários reais. Neste caso em específico podemos ver que nas estradas na vertical temos duas rotas possíveis, o que nos levou a aumentar o fluxo nestas duas em futuros testes.

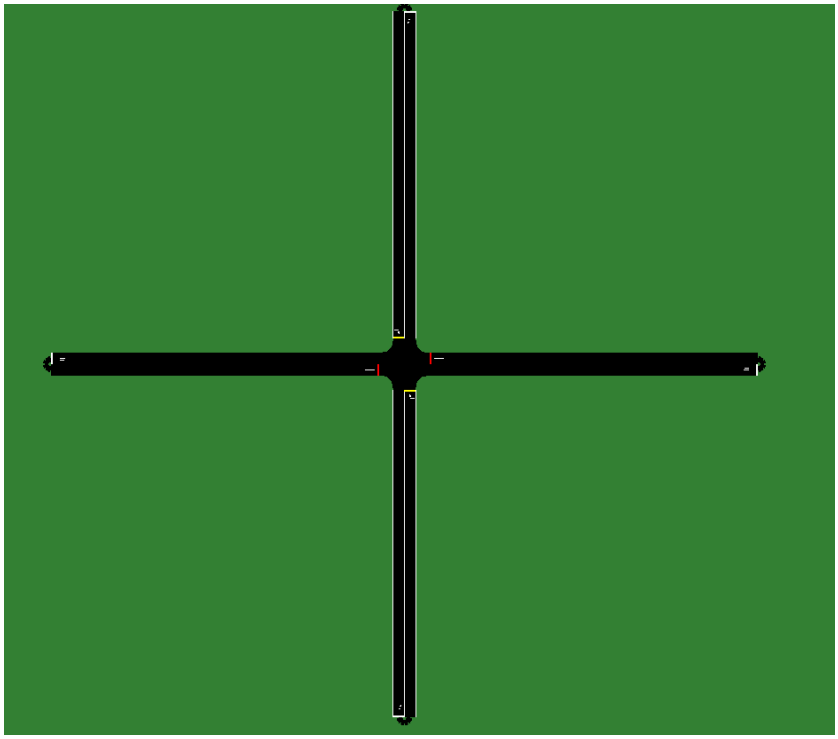

Fig. 1. Topologia

2.2 Infraestrutura

- **Aplicação dos Veículos**: Adquire e envia dados de mobilidade sobre o próprio veículo, *CAMs*, e respeita a sinalização do semáforo no que toca a seguir a sua rota.
- **Aplicação *RSU***: Recebe os dados dos veículos, processa-os de modo a fazer o mapeamento da rede em tempo real, podendo assim enviar mensagens ao semáforo para que este mude a sua sinalização consoante o estado da rede atual.
- **Aplicação Semáforo**: Processa os dados recebidos pelo *RSU* de modo a controlar a sinalização do semáforo.

2.3 Primitivas de comunicação

As mensagens trocadas no nosso sistema estendem todas para o *V2XMessage* e, portanto, são reconhecidas pelo *SUMO* como tal. São propagadas em modo *ad-hoc* segundo um raio de alcance e um poder de propagação específicos abaixo mencionados. Para uma melhor gestão de congestão de rede, decidimos separar as mensagens *CAM* das mensagens de controlo do semáforo em canais distintos,

nomeadamente *SCH1* para as *CAMs* e *SCH6* para as mensagens de controlo dos semáforos.

```
getOs().getAdHocModule().enable(new AdHocModuleConfiguration()
    .addRadio()
    .channel(AdHocChannel.CCH)
    .distance(40)
    .power(0.002)
    .create());
```

Fig. 2. Inicialização do modulo AdHoc

Nesta fase, são estabelecidas comunicações entre os *OBU's* e o*RSU* de modo a que seja possível a este *RSU* fazer um mapeamento da rede ao seu redor obtendo informações como posição. velocidade e rota dos veículos através das *CAM's* por estes enviadas. O *RSU*, perante o ponto de situação da rede atual, ordena a mudança de sinal do semáforo através de uma mensagem de controlo também enviada no mesmo formato das mensagens *CAM*.

2.4 Formato das mensagens protocolares (PDU)

Nesta secção vamos abordar a estrutura e conteúdo das mensagens trocadas entre veículos, RSUs e o servidor, garantindo a correta interpretação dos dados e ilustrar com exemplos para uma melhor compreensão.

PacketMessage A **PacketMessage** é uma mensagem de protocolar utilizada para encapsular e transmitir dados entre os veículos e as *RSUs*. Esta mensagem inclui informações essenciais sobre o estado e a mobilidade dos veículos no exato momento, como a posição, velocidade e rota. O formato padrão da *PacketMessage* garante que todos os componentes do sistema possam interpretar corretamente os dados recebidos, facilitando a comunicação eficiente e precisa entre os veículos e a infraestrutura rodoviária. São numeradas de modo a facilitar a noção temporal do sistema, permitindo a resolução de problemas que podem acontecer em ambiente real como ruído e dificuldade de propagação de mensagens.

```
public final class PacketMsg extends V2xMessage {
    private final String name;
    private final String route;
    private final Double lon;
    private final Double lat;
    private final Double velo;
    private final int MsgNumber;
    private final EncodedPayload payload;
    private final static long minLen = 128L;
```

Fig. 3. Estrutura do Mensagem CAM

LightMessage A **LightMessage** é uma mensagem de controlo utilizada para gerir o estado dos semáforos no sistema de tráfego inteligente. Esta mensagem contém comandos específicos para alterar o programa executado pelo semáforo, assim como um segredo que garante a identidade do *RSU*. As **LightMessages** permitem uma gestão dinâmica e adaptativa do fluxo de tráfego com base nas condições de trânsito em tempo real.

```
public final class LightMsg extends V2xMessage {
    private final String program;
    private final String secret;
    private final EncodedPayload payload;
    private final static long minLen = 128L;
```

Fig. 4. Estrutura do Mensagem de controlo do Semáforo

3 Implementação

Nesta secção, detalharemos a implementação dos componentes principais do sistema de semáforos inteligentes suportado por comunicações V2X, na etapa 1.

Vehicle Application A aplicação *OnVehicleUpdatedApp* é responsável por enviar para a rede os dados relativos ao veículo onde esta esta a correr. Ao inicializar a aplicação a função *"onStartup"* é executada habilitando o módulo de comunicação *ADhoc* do sistema operacional do veículo. De seguida são definidos eventos como o envio da primeira mensagem CAM ou a função de obter o nome e rota do veículo. Alem desse método temos ainda o *"onVehicleUpdated"* que é acionado sempre que existe um movimento do mesmo na simulação e é responsável por obter informação sobre o veículo em tempo real. Essa informação é processada nesse mesmo método e caso a velocidade esteja acima de um limite

definido é chamado o método de enviar uma mensagem CAM. Esta funcionalidade mostra-se muito útil em ambiente real e daí a implementação no nosso sistema, pois quanto maior a velocidade de um veículo maior deve ser a frequência de envio de mensagens de modo a que a rede esteja o mais atualizada possível mesmo em para casos de elevada mobilidade dos nós. O método abordado ate agora responsável por enviar mensagens CAM é o *"sendAdHocBroadcast"*, que é caracterizado por enviar um pacote de dados para a rede contendo a informação atualizada do veículo a cada chamada. Este método é chamado periodicamente, no caso da nossa implementação a cada dois segundos de simulação, sendo que esta frequência não afeta as chamadas deste método quando a velocidade do veículo é elevada diminuindo assim esta periodicidade.

RSU Application A aplicação *RoadSideUnitApp* gerência e controla o mapeamento da rede ao seu redor. Ela recebe os pacotes enviados pelas aplicações presentes nos veículos quando estas são enviadas dentro do raio de alcance do modulo *ADhoc* nele presente. Estas mensagens são posteriormente processadas de modo a mapear a rede ao seu redor e garantir que a sinalização do semáforo é a ideal para a situação de rede atual.

Ao inicializar a aplicação a função *"onStartup"* é executada habilitando o módulo de comunicação *ADhoc* do sistema operacional do *RSU*. De seguida e a quando de uma receção de um pacote de dados proveniente de um veículo é processada a informação nele presente de modo a construir um mapeamento de rede, sendo que essa informação é guardada no *RSU* numa lista de estruturas de dados que contem a informação sobre nome, rota, posição, numeração da mensagem, distancia ao *RSU* e o instante da receção. Assim a cada receção de uma nova mensagem, isto é um pacote do mesmo veículo mas com numeração superior esta informação é atualizada. Caso a distancia ao *RSU* diminua é adicionada a informação de que este veículo esta a aproximar-se do semáforo e é adicionado à lista de carros à espera nas vias a vertical ou vias a horizontal dependendo da sua rota. Caso a distância aumente a informação que o veículo já passou pelo cruzamento é processada pelo *RSU* retirando este veículo da lista de veículos à espera onde este estava. Para além disso a cada atualização do mapeamento de rede como estas acima descritas é chamado o método de *"clearNeighList"* que é caracterizado por limpar a lista de vizinhos, removendo a informação dos veículos sobre os quais o *RSU* não recebe informação num período de quinze segundos (15 seg), impedindo overflows de memoria neste sistema que deve estar disponível ao longo de toda a simulação. Por fim e tambem no final de cada atualização do mapeamento de rede é chamada a função *"updateTrafficLight"* que é caracterizada por analisar o numero de veículos nas vias a horizontal e a vertical, e consoante o volume de veículos nestas vias define qual a sinalização do semáforo, sendo priorizada a via com maior volume de carros. A comunicação com o semáforo é também enviado no formato de um pacote de dados em modo *"ADhoc"* com a indicação do programa que deve ser usado acompanhado pelo segredo garantindo que só a informação enviado pelo legitimo *RSU* é processada.

Traffic Light app A aplicação *TrafficLightApp* foi desenvolvida para controlar o estado do semáforo com base nas informações de tráfego recebidas. Esta aplicação executa a lógica do semáforo, processando dados recebidos pelo *RSU*. São definidos dois programas possíveis um indica que as vias na horizontal ficam a verde a as restantes a vermelho e o outro programa é o inverso.

```
<tlLogic id="J9" type="static" programID="0" offset="0">
    <phase duration="3"   state="rryrry"/>
    <phase duration="900" state="ggrggr"/>
</tlLogic>

<tlLogic id="J9" type="static" programID="1" offset="0">
    <phase duration="3"   state="yyryyr"/>
    <phase duration="900" state="rrgrrg"/>
</tlLogic>
```

Fig. 5. Programas do TrafficLightApp

Esta aplicação é caracterizada por na função *"onStartup"* habilitar o módulo de comunicação *ADhoc*, proporcionando a comunicação com o *RSU*. Após a inicialização apenas fica à espera de receber mensagens de um *RSU* legitimo de modo a mudar a sinalização do semáforo segundo o programa recebido no pedido.

4 Testes e Resultados

Nesta primeira fase já conseguimos observar uma boa resposta do *RSU*, resultando numa boa organização de prioridades no cruzamento, fazendo com que os carros não estejam parados durante longos períodos de tempo. Para além disso, garante principalmente que o semáforo não tem um ciclo constante e sim dinâmico que responde à situação atual da rede. Contudo, como o nosso algoritmo se baseia no controlo de posicionamento do carro, isto é, controla se um carro está a aproximar-se ou afastar-se do *RSU* consoante a posição do mesmo presente nas mensagens *CAM* que este recebe, ocorre por vezes que um carro já tenha passado o semáforo e, como ainda não enviou uma nova *CAM*, o servidor não consegue atualizar a tabela de vizinhos de forma imediata. Ainda assim, este delay não se torna significativo nem penoso para a simulação, resultando num bom equilíbrio entre tempos de espera e números de carros à espera.

– ***Etapa 2***

5 Introdução

Na segunda etapa, o objetivo é melhorar a perceção que o semáforo físico tem do ambiente à volta. Para isso, implementamos um protocolo de encaminhamento **multihop** que é executado pelos veículos. Um veículo, ao receber uma mensagem de outro veículo, deve, em determinadas condições, reencaminhar essa mensagem. Dessa forma, o semáforo pode detetar veículos antes de eles entrarem no seu raio de alcance, melhorando a gestão do trânsito.

6 Especificação do sistema e dos protocolos de suporte

Nesta segunda fase, mantivemos a topologia, uma vez que só se trata da implementação de um protocolo *multihop* executado nos veículos.

6.1 Infraestrutura

- **Aplicação dos Veículos**: Adquire e envia dados de mobilidade sobre o próprio veículo, *CAMs*; Reenvia mensagens de outros veículos se se encontrar em melhor posição que os respetivos de modo a propagar os dados na rede; Envia e retransmite informações importantes como o estado da via de transito, no formato de *DENMs*; Respeita a sinalização do semáforo no que toca a seguir a sua rota.
- **Aplicação *RSU***: Recebe os dados dos veículos, processa-os de modo a fazer o mapeamento da rede em tempo real, podendo assim enviar mensagens ao semáforo para que este mude a sua sinalização consoante o estado da rede atual.
- **Aplicação Semáforo**: Processa os dados recebidos pelo *RSU* de modo a controlar a sinalização do semáforo.

6.2 Primitivas de comunicação

Como pretendido pela implementação de um protocolo *multihop*, para além do envio de CAM's pelos veículos, existe agora uma propagação destas mesmas mensagens pelos veículos em seus raios de alcance. Esta estratégia define que uma mensagem pode ser retransmitida por veículos no raio de alcance do veículo emissor até dois saltos. Assim, é possível a receção de mensagens no *RSU* contendo informações de veículos que ainda não se encontram no raio de alcance do mesmo proporcionando um conhecimento da rede mais alargado de até dois saltos. Este conhecimento aliado ao nosso algoritmo de controlo do semáforo permite que o sinal seja definido perante a situação de rede atual aliada a informação futura, o que proporciona uma gestão antecipada do tráfego. Esta estratégia permite ainda uma maior certeza de que as mensagens enviadas pelos veículos chegam

efetivamente ao *RSU*, uma vez que são retransmitidas e propagadas por outros nós da rede.

Para além disso, foi implementado um protocolo de mensagens *DENM* que é caracterizado por sinalizar situações de emergência. No nosso caso em particular, apenas é enviada uma mensagem por cada veículo que indica o estado do piso ao segundo 9 da simulação. Assim como as *CAM*, esta mensagem é retransmitida pelos veículos também até dois saltos, dando assim a conhecer o estado do piso aos veículos próximos. Estas mensagens têm uma particularidade: se informarem o veículo de piso molhado ou nevoeiro, este, ao processar essa informação, irá reduzir a sua velocidade num período de 5 segundos, garantindo, assim, que tomou as devidas precauções perante tal aviso.

6.3 Formato das mensagens protocolares (PDU)

PacketMessage A *PacketMessage*, assim como na fase anterior, é utilizada para encapsular e transmitir dados de mobilidade dos veículos, incluindo informações como nome do veículo, rota, longitude, latitude, velocidade e número da mensagem. No entanto, esta versão foi adaptada em relação à primeira etapa para incluir um campo adicional, *numero de hops*, que permite o encaminhamento *multihop* das mensagens entre veículos. Garantindo que não existem *flooding* de mensagens exagerado e aumentando a perceção do estado do tráfego além do alcance direto da *RSU*, melhorando a eficiência na gestão do tráfego.

```
public final class PacketMsg extends V2xMessage {
    private final String name;
    private final String route;
    private final Double lon;
    private final Double lat;
    private final Double velo;
    private final int MsgNumber;
    private int hops;
    private final EncodedPayload payload;
    private final static long minLen = 128L;
```

Fig. 6. Estrutura do Mensagem CAM

LightMessage A *LightMessage* assim como na fase anterior serve de controle e é utilizada para gerir o estado dos semáforos no sistema de tráfego inteligente.

```
public final class LightMsg extends V2xMessage {
    private final String program;
    private final String secret;
    private final EncodedPayload payload;
    private final static long minLen = 128L;
```

Fig. 7. Estrutura do Mensagem de controlo do Semáforo

6.4 DENMMessage

A *DENMMessage (Decentralized Environmental Notification Message)* é utilizada para notificar eventos específicos que podem afetar a segurança e a fluidez do tráfego, como condições de estrada adversas. Esta mensagem inclui informações detalhadas sobre o evento, como nome do veículo, rota, longitude, latitude, estado da estrada e número da mensagem. A *DENMMessage* é essencial para fornecer informações contextuais em tempo real aos veículos, permitindo que tomem decisões mais informadas ao se aproximarem de um evento reportado.

```
public final class DENMMsg extends V2xMessage {
    private final String name;
    private final String route;
    private final Double lon;
    private final Double lat;
    private final String roadState;
    private final int MsgNumber;
    private int hops;
    private final EncodedPayload payload;
    private final static long minLen = 128L;
```

Fig. 8. Estrutura do Mensagem DENM

7 Implementação

Vehicle Application A aplicação *OnVehicleUpdatedApp* é a que mais sofreu alterações em relação à fase anterior. Assim adicionamos a funcionalidade de enviar e processar mensagens *DENM* que é utilizada uma vez por veículo num período pré-definido chamando a função *"getRoadState"*. Esta função é caracterizada por enviar uma mensagem *DENM* em *broadcast* contendo a informação do estado do piso da via onde se encontra o veículo. Assim como esta funcionalidade, foi adicionada a de receber e processar estas mesma mensagens. Para o caso de a mensagem recebida conter informação sobre o piso da via ao seu redor informando que está molhado ou nevoeiro, a aplicação veículo responde a esta

mensagem com uma desaceleração do veículo por 5 segundos de modo a prevenir e controlar estas situações adversas. De modo a proporcionar o *fowarding* dos pacotes enviados pelos carros, tanto os pacotes das *CAMs* como os das *DENMs* tem um atributo correspondente ao número de saltos possível. Nesta nossa implementação cada uma das mensagens pode ser reenviada até dois saltos de distancia. Este *fowarding* permite que a informação das *DENMs* seja partilhada e propagada advertindo corretamente os nós da rede. Para além disso o *RSU* pode mapear a rede com até dois saltos de antecedência, antecipando o tráfego no semáforo e proporcionando uma atualização antecipada.

RSU Application A aplicação *RoadSideUnitApp* gerencia e controla o mapeamento da rede ao seu redor e não sofreu muitas alterações em relação à da fase anterior. No entanto, como esta recebe agora um maior volume de dados do que a anterior, foi necessário desenvolver um mecanismo de controlo onde caso a mensagem já tenha sido recebida não se deixa afetar por uma replica da mesma. Seguimos com esta resolução durante praticamente toda a fase de implementação e testes desta etapa, porém para um caso especifico, o de haver um volume de veículos muito superior num dos lados, a nossa aplicação tornava-se ineficiente. Perante este impasse, decidimos fazer uma alteração ao algoritmo de escolha da sinalização no momento. Substituímos o anterior algoritmo que apenas tem em conta o número de veículos em cada uma das vias, por um que não só tem em conta a informação do número de carros, mas também o tempo de espera dos mesmos. Simplificando, o algoritmo soma os tempos de espera dos nós que já estão sob o mapeamento do *RSU* nas vias a vertical e horizontal e favorece, isto é, torna verde a sinalização para as vias onde o valor é maior. Assim, mesmo no caso do fluxo de veículos ser muito maior numa das vias, os veículos nunca ficam demasiado tempo à espera.

Traffic Light app A aplicação *TrafficLightApp* que foi desenvolvida para controlar o estado do semáforo com base nas informações de tráfego recebidas pelo *RSU* é a mesma da fase anterior.

8 Testes e Resultados

Quanto aos resultados desta fase, destacamos que foram bastante idênticos aos da fase anterior. Contudo, como o *RSU* conhece a topologia de forma mais alargada, nomeadamente a dois saltos de distância, o mapeamento de rede por ele construido tem consequentemente um maior volume de informação. Perante este facto, podemos afirmar que esta etapa é favorável a casos onde o volume de dados nas vias é dinâmico e com um leque de fluxos mais distintos, isto é, para os casos onde determinada via tem um fluxo baixo e outra um fluxo muito alto. Nestes casos, este algoritmo garante que as vias com maior fluxo não necessitam de ser mais demoradas a percorrer. Uma vez que na etapa anterior só eram tidos em consideração os veículos perto do semáforo na escolha da sinalização, as vias com maior volume eram representadas da mesma forma que as de baixo volume. Contudo, as vias com um menor volume de veículos não são favorecidas nesta etapa, obtendo um maior intervalo de tempo necessário para as percorrer, no entanto, não é preocupante uma vez que o nosso algoritmo tem em conta o tempo de espera dos veículos, e se eles ficarem muito tempo à espera, o algoritmo favorece-os e é enviado um pedido para mudar a sinalização, de modo a que seja favorável aos mesmos. Em suma, descrevemos com esta etapa uma solução ideal para quando é útil garantir que as vias com maior fluxo e/ou maior prioridade de tráfego sejam favorecidas em relação a outras.

– ***Etapa 3***

9 Introdução

Na terceira etapa, o objetivo é substituir o semáforo físico por um semáforo virtual. A comunicação entre veículos e *RSU* continua essencial, mas agora a *RSU* envia diretamente mensagens de controlo aos veículos, indicando se devem parar ou avançar.

10 Especificação do sistema e dos protocolos de suporte

Nesta terceira fase, mantivemos a topologia, porém substituímos os semáforos que até agora eram responsáveis por indicar aos veículos a sinalização no entroncamento para a sua via, por passarem a estar sempre com indicação verde, o que significa que os veículos por ele podem passar. Isto garante que o *RSU* tem o completo controlo dos carros.

10.1 Infraestrutura

- **Aplicação dos Veículos**: Adquire e envia dados de mobilidade sobre o próprio veículo, *CAMs*; Reenvia mensagens de outros veículos se se encontrar em melhor posição que os respetivos de modo a propagar os dados na rede; Envia e retransmite informações importantes como o estado da via de trânsito, no formato de *DENMs*; Respeita as mensagens de controlo enviadas pelo *RSU*.
- **Aplicação *RSU***: Recebe os dados dos veículos, processa-os de modo a fazer o mapeamento da rede em tempo real; Envia mensagens de controlo para os veículos como mensagens para que estes reduzam a velocidade, parem ou avancem no entroncamento.
- **Aplicação Semáforo**: Sinaliza todas as vias com "verde" indicando que por ele podem prosseguir a rota.

10.2 Primitivas de comunicação

Nesta fase não há mensagens trocadas com o semáforo e, portanto, todas as mensagens enviadas no sistema são agora *PacketMSG*, classe definida para representação das *CAMs*, *DENMMSG*, classe definida para representação de *DEMN* mensagens de alerta para perigos que possam existir e ainda *RSUMsg*, classe definida para o envio de mensagens de controlo do *RSU* para os veículos, indicando que comportamento estes devem adotar. Este modelo de mensagens é usado de maneira muito peculiar e iremos retratá-lo na secção de implementação.

10.3 Formato das mensagens protocolares (PDU)

PacketMessage A *PacketMessage*, assim como na fase anterior, é utilizada para encapsular e transmitir dados de mobilidade dos veículos, incluindo informações como nome do veículo, rota, longitude, latitude, velocidade, número da mensagem e número de saltos possíveis.

10.4 DENMMessage

A *DENMMessage* é utilizada para notificar eventos específicos que podem afetar a segurança e a fluidez do tráfego, como condições de estrada adversas assim como na fase anterior.

10.5 RSUMessage

A *RSUMessage* é utilizada para controlar o comportamento dos veículos até que estes cheguem ao cruzamento, como indicar que está a chegar um cruzamento, solicitando uma redução da velocidade os mesmos, assim como a ordem de passagem pelo entroncamento indicando quais os veículos que devem parar e quais veículos devem avançar a cada instante.

```
public final class RSUPacketMsg extends V2xMessage {
    private final String secret;
    private final String name;
    private final String type;
    private final EncodedPayload payload;
    private final static long minLen = 128L;
```

Fig. 9. Estrutura do Mensagem de controlo do RSU

11 Estratégias de implementação e discussão de resultados

Para esta terceira fase a ideia inicial era implementar um serviço de resposta pelo *RSU* a mensagens CAM recebidas pelos veículos indicando com alguma antecedência o ponto de paragem para cada veículo consoante a sua rota. Esta versão permitia garantir que o veículo recebia antecipadamente a informação de parar antes do entroncamento e poderia fazê-lo em segurança. Esta paragem seria então proporcionada pelo método **stop** da aplicação do veículo uma vez que este necessita do ponto de paragem, e seria garantido que o veículo parava na entrada do cruzamento independentemente da sua rota. A ideia acima relatada não foi desenvolvida pois não sabíamos como informar o ponto de paragem ao

veículo na forma de um "*IRoadPosition*" e como tal nem conseguimos viabilizar nem testar esta primeira ideia.

Como alternativa a isto decidimos tirar partido da comunicação em *Ad-hoc Geobroadcast* e desenvolvemos um método no *RSU* capaz de enviar periodicamente (a cada 1 segundo de simulação) uma mensagem de controlo dando a ordem de parar aos veículos. Esta comunicação é estabelecida com base na área do círculo definido e os veículos que se encontram abrangidos por essa área receberão as mensagens enviadas. No caso, centramos esse círculo na posição do *RSU* (centro do cruzamento) e definimos um raio pequeno de 18 unidades que seria o suficiente para abranger os primeiros veículos de cada via. Nesta fase, e como não conseguimos implementar o método **stop** nos veículos, definimos que quando um veículo recebesse uma mensagem de controlo a dizer para parar, era invocado o método **stopNOW**.

Com esta alternativa implementada obtivemos problemas na sincronização da paragem dos carros, pois por vezes a velocidade era tal que ele não conseguia parar no local onde invocou método **stopNOW**, uma vez que este funciona à semelhança de uma travagem a fundo num carro em ambiente real e por vezes só parava depois da entrada no cruzamento, ficando parado no meio ou na saída deste, o que impedia os outros veículos de passar.

Perante este problema implementamos uma nova funcionalidade no *RSU*, a de enviar mensagens de controlo indicando os veículos que estavam próximos de um cruzamento e como tal deviam abrandar. Este método assemelha-se muito ao do envio das mensagens para parar, no entanto, apesar de centrado no mesmo sítio tem um raio muito maior (50 unidades). Assim, garantimos, à semelhança de um ambiente real, que existe um aviso prévio da chegada a um cruzamento. Os veículos respondem a este pedido com a chamada do método **slowdown** reduzindo a sua velocidade no momento e garantindo que chegam ao cruzamento e conseguem parar antes da entrada no mesmo.

Após esta alteração, conseguimos garantir que os carros paravam antes do cruzamento, porém nem sempre isso acontecia e, quando paravam, notamos que os veículos das vias a vertical paravam depois da linha do semáforo, enquanto que os das vias a horizontal estavam bem mais afastados dessa zona. Este problema surge pois os valores de latitude variam mais do que os de longitude no nosso sistema e a área gerada não acompanha área necessária. Para combater este problema, criamos duas áreas nas quais enviamos mensagens de controlo para o carro parar, uma mais acima em termos de latitude e outra mais abaixo do *RSU* e diminuímos o raio de ambas para 11 unidades. Com esta alteração conseguimos garantir que todos os veículos paravam antes do cruzamento.

Os problemas continuaram na implementação do algoritmo de escolha do nó que deve avançar pois por vezes, principalmente quando o cruzamento estava vazio, era pedido a um carro que avançasse, porém este ainda estaria sob o efeito do **stopNow** e não lhe era possível invocar o método de **resume** voltando assim ao seu comportamento normal e prosseguir a sua rota. Como resolução a este problema, decidimos, ao invés de parar o veículo com o **StopNow**, usar o **changeSpeedWithInterval(0, 0)** o que faz com que o carro fique com ve-

locidade 0 no próprio instante em que o método é chamado e usamos depois o método **getOs().resetSpeed()** para voltar à sua velocidade anterior.

Esta implementação já melhorou bastante o nosso ambiente, porém havia casos onde o primeiro veículo da via avançava porque tinha ordem para avançar e depois o segundo avançava por que não tinha ainda recebido/processado a mensagem de controlo para parar. Resolvemos este problema diminuindo a periodicidade do envio das mensagens de controlo para parar de 1 para 0.5 segundos.

Com esta implementação obtivemos casos onde os carros não avançavam por não terem recebido a mensagem de avançar enviado pelo *RSU* apesar da mesma ter sido enviada, pensamos que talvez por excesso de mensagens na área do cruzamento os veículos não consigam processar todas.

Deste modo implementamos um mecanismo na aplicação dos veículos que, se estes estiverem a circular com velocidades muito baixas o número de saltos possíveis para a retransmissão da mensagem passa de 2 para apenas 1 numa tentativa de diminuir o tráfego e volume de mensagens no cruzamento. Para além disso, aumentamos o intervalo de tempo com que enviamos as mensagens de parar de 0.5 para 0.7 segundos de intervalo.

Ainda assim não conseguimos o cenário ideal pois os veículos nem sempre respondem às mensagens de controlo e, por vezes, a informação não recebida é executada apenas segundos de simulação depois, não percebendo como o carro volta a andar mesmo sem supostamente ter recebido a mensagem do *RSU* para que possa avançar.

11.1 Pontos a melhorar

Visto que esta terceira etapa não ficou funcional, deixamos aqui notas do que gostaríamos de ter implementado caso não tivéssemos tais impasses.

Com todos os problemas que enfrentamos para fazer com que os carros parassem no entroncamento, não demos o devido foco na implementação de um bom algoritmo de escolha do nó que deve avançar. No caso, uma implementação assertiva de um algoritmo para esta fase seria um que conseguisse garantir que veículos na entrada do cruzamento com rotas que não colidam possam avançar em simultâneo, melhorando assim a gestão e o fluxo de tráfego. Para além disso implementar mecanismos de congestão de rede e garantir que as mensagens do *RSU* são recebidas pelos carros com uma prioridade superior às *CAMs* de outros veículos. A utilização da infraestrutura já existente poderia ser melhor reaproveitada e usar os semáforos em conjunto com sensores de movimento para detetar se ainda existem carros à espera, o que seria uma grande ajuda para uma implementação mais eficiente. Isto garantiria que erros como o de o carro não receber a mensagem enviada e esta não ser reenviada pelo *RSU* deixassem de acontecer.

12 Conclusão

As redes *adhoc* emergem como uma solução altamente eficaz para a implementação de um Sistema de Semáforos Inteligentes suportado por comunicações *V2X (Vehicle-to-Everything)* num ambiente com carros inteligentes, uma vez que permitem uma comunicação direta entre os veículos *(V2V)* e entre veículos e infraestruturas *(V2I)*, como os semáforos, sem a necessidade de uma infraestrutura de comunicação preexistente.

Num sistema em ambiente real a integração das comunicações *V2X* permite que os veículos e semáforos compartilhem informações em tempo real, como a velocidade, localização e intenção de movimento, possibilitando uma coordenação eficiente e sincronizada dos semáforos. Isto pode reduzir significativamente os congestionamentos, melhorar o fluxo de tráfego e diminuir o tempo de viagem, ao mesmo tempo em que contribui para a redução de emissões de carbono.

No entanto, e como vimos nesta nossa terceira etapa, nem sempre são suficientes e é importante destacar que a utilização de sensores tanto nos veículos quanto nos semáforos não deve ser negligenciada. Os sensores desempenham um papel vital em coletar dados precisos e na deteção de condições do ambiente que não podem ser facilmente comunicadas através de *V2X.* A combinação de dados provenientes dos sensores com a comunicação *V2X* fortalece a segurança e a eficiência do sistema, permitindo uma visão mais completa e detalhada do cenário de tráfego.

Em suma, a implementação de um Sistema de Semáforos Inteligentes baseado em redes *adhoc* e comunicações *V2X*, complementada pela utilização de sensores, na nossa opinião, oferece uma abordagem mais robusta e abrangente para a gestão do tráfego para ambientes reais, promovendo um tráfego equilibrado e eficiente.