UMA PLATAFORMA PARA INTERNET DOS VEÍCULOS INTELIGENTES

Elton de Souza Vieira* Marianne Batista Diniz da Silva* Daniel Gouveia Costa†
Ivanovitch Medeiros Dantas da Silva*

*Universidade Federal do Rio Grande do Norte
Natal, RN, Brasil

†Universidade Estadual de Feira de Santana
Feira de Santana, BA, Brasil

Email: eltonvs2@gmail.com, mariannedinizsi@gmail.com, danielgcosta@uefs.br, ivan@imd.ufrn.br

Abstract— In the last decade, the growth of the automotive market with the aid of technologies has been notable for the economic, automotive and technological sectors. Alongside this growing recognition, the so-called Internet of Intelligent Vehicles (IoIV) emerges as an evolution of the Internet of Things (IoT) applied to the automotive sector. In this way, the automatic communication between vehicle and devices was facilitated, allowing the accomplishment of several analysis regarding vehicles, such as the identification of a vehicle behavioral pattern through historical usage, fuel consumption checks, maintenance indicators, among others, thus allowing the prevention of critical issues and undesired behaviors. This work proposes a vehicle connection Application Programming Interface (API) using the OBD-II technology, presenting some particular contributions, namely: concurrent execution of commands, implementation of a state machine for prediction of errors and minimization of battery consumption, among others. More precisely, this paper aims to propose an Internet platform for Intelligent Vehicles, capable of collecting and analyzing, through an OBD-II scanner, the sensors made available by vehicles, with the purpose of assisting in the management, prevention and mitigation of different vehicular problems.

Keywords— Internet of Intelligent Vehicles, IoIV, OBD-II, Smart Cities.

Resumo— Na última década, o crescimento do mercado automobilístico com o auxílio de tecnologias tem sido notável para os setores econômico, automobilístico e tecnológico. Alinhada a essa crescente valorização, surge a chamada Internet dos Veículos Inteligentes (IoIV, do inglês, *Internet of Intelligent Vehicles*), como uma evolução da Internet das Coisas (IoT) aplicada ao setor automotivo. Dessa forma, a comunicação automática entre veículo e dispositivos foi facilitada, permitindo a realização de diversas análises a respeito do veículo, tais como a identificação do padrão de comportamento veicular através dos padrões de utilização, verificação do consumo de combustível, indicadores de manutenção, entre outros, permitindo assim, a prevenção de problemas e de comportamentos indesejáveis. Esse artigo propõe uma *Application Programming Interface* (API) para conexão veicular usando a tecnologia OBD-II, apresentando algumas contribuições, a saber: execução concorrente de comandos, implementação de uma máquina de estados para previsão de erros e minimização do consumo de bateria, entre outras. Mais precisamente, esse artigo tem como objetivo propor uma plataforma para Internet dos Veículos Inteligentes, capaz de coletar e analisar, através de um *scanner* OBD-II, os sensores disponibilizados pelos veículos, com o intuito de auxiliar na gestão, prevenção e mitigação de diferentes problemas veiculares.

Palavras-chave— Internet dos Veículos Inteligentes, IoIV, OBD-II, Cidades Inteligentes.

1 Introdução

O mercado automobilístico vem crescendo nos últimos anos, facilitando o surgimento de novos segmentos de mercado. Com o auxilio da Tecnologia da Informação, os automóveis foram sendo melhorados, também disponibilizando uma série de informações computadorizadas.

O número de veículos em todo o mundo aumentou de cerca de 900 milhões em 2006 para mais de 1 bilhão em 2014, devendo chegar a 2 bilhões até 2035 (Yang et al., 2014). Nesse contexto, a maioria desses veículos que surgirão até 2035 serão carros conectados que irão possuir a capacidade de comunicação sem fio (Contreras-Castillo et al., 2017) com vários tipos de dispositivos (sensores, telefones, câmeras) conectados à Internet, utilizando uma variedades de protocolos de comunicação e mídias de transmissão. Além disso, 25 bilhões de dispositivos serão conectados à Internet até 2020 (Yang et al., 2014).

Nesse contexto, sabe-se que os dados oriundos desses sensores e sistemas veiculares podem ser extraídos pelo veículo de forma automatizada, a partir de protocolos de comunicação padronizados para a área veicular. E essas informações podem ser exploradas para interações em um novo contexto de processamento e comunicação, revolucionado a forma como veículos são utilizados. De fato, tais interações ocorrem no contexto da Internet das Coisas (IoT, do inglês *Internet of Things*), que possibilita a troca de informação entre objetos e plataformas por meio da interconexão entre sensores e atuadores (Gubbi et al., 2013). Esse paradigma emergente abrange uma infraestrutura de hardware, software e serviços que conectam objetos físicos, denominados como "coisas", à Internet (Zanella et al., 2014). Esse cenário permite a definição do contexto de IoIV (*Internet of Intelligent Vehicles*) (Dandala et al., 2017), bem como ao conceito de VANETs

(*Veicular Ad Hoc Networks*) que são redes móveis onde são estabelecidas comunicações intra-carros (Chen et al., 2009; Pu et al., 2016), o que determina o foco deste trabalho.

Assim, para coletar uma série de dados dos veículos em tempo real, tais como o status da luz da injeção eletrônica, códigos de diagnóstico de problemas (DTCs, do inglês *Diagnostic Trouble Codes*), rotação do motor, velocidade, temperatura do líquido de arrefecimento, dentre outros, é utilizada a tecnologia OBD (do inglês *On-Board Diagnostics*) que refere-se a um sistema de autodiagnóstico disponível na maioria dos veículos que circulam atualmente (Oliveira et al., 2017). A leitura dos dados é realizada a partir de um dispositivo *"plug and play"* diretamente na ECU (Unidade de controle do motor, do inglês *Engine Control Unit*) do veículo.

Neste contexto, este artigo propõe uma plataforma de 3 módulos para *Internet dos Veículos Inteligentes*, capaz de coletar e analisar, a partir do hardware OBD-II, os dados dos sensores disponíveis nos veículos afim de criar métricas de manutenção, utilização e consumo dos mesmos. Como diferenciais essa plataforma apresenta uma nova API (*Application Programming Interface*), com as seguintes características:

a) Suporte a toda a faixa de comandos disponíveis na tecnologia;

b) Decodificação dos comandos suportados pelo veículo de maneira holística;

c) API permitindo a execução paralela dos comandos OBD;

d) Módulo inteligente para envio de dados baseado na tecnologia de comunicação;

e) Módulo *Fog Computing* na API;

f) Envio de dados transparente para módulos de visualização e análise de dados na nuvem;

g) Máquina de estado para minimizar o consumo de energia do *Global Positioning System* (GPS) e Protocolos de Comunicação.

Como resultados, são apresentados os dados de captura dos sensores coletados por veículos de marcas e fabricantes distintos.

Por fim, este artigo encontra-se organizado da seguinte forma: na Seção 2 são apresentados os trabalhos relacionados. Na Seção 3 é elucidada a arquitetura proposta. Os resultados obtidos são apresentados na Seção 4. Na Seção 5 apresentam-se as considerações finais e alguns trabalhos futuros.

2 Trabalhos Relacionados

Após uma revisão sistemática da literatura, foram encontrados diversos trabalhos que influenciaram a pesquisa realizada e que, portanto, contribuíram com o desenvolvimento da solução proposta. Os autores em (Malekian et al., 2017) realizam o desenvolvimento de um sistema de gerenciamento de frota, com a utilização do OBD-II. O sistema visa medir a velocidade, distância e consumo dos combustíveis dos veículos com o intuito de realizar rastreamento e análise computacionais. Os dados do OBD-II são transmitidos via *Wi-Fi* para um servidor remoto. Um sistema de gerenciamento de banco de dados é implementado no servidor para armazenar os dados, permitindo diferentes análises.

De forma parecida, (Kim et al., 2015) implementam um sistema de disgnóstico móvel que fornece interfaces para o usuário estimar e diagnosticar as condições do motor por meio da comunicação com a ECU. Para o sistema implementado, um novo protocolo foi projetado e aplicado com base no padrão OBD-II para receber valores de dados do motor do ECU desenvolvido. Assim, ele transmite 31 tipos de informações de condições do motor simultaneamente e envia o código de diagnóstico de problemas.

Similarmente, (Yang et al., 2015) apresentam um sistema de diagnóstico para veículos elétrico híbrido e um monitor de dados, utilizando a placa de diagnóstico OBD-II e um *smartphone* Android, utilizando para tanto rede móvel 3G, GPS e a tecnologia *bluetooth*. Já o trabalho em (Meseguer et al., 2015) apresenta uma metodologia para calcular, em tempo real, o impacto ambiental e de consumo de veículos de ignição com centelão e diesel a partir de um conjunto de variáveis, como a taxa de combustível do motor, a velocidade, o fluxo de massa de ar e a carga absoluta, todos obtidos da ECU. A plataforma é capaz de ajudar os motoristas a corrigir seus maus hábitos de condução, oferecendo recomendações úteis para melhorar a economia de combustível.

De fato, percebe-se na literatura que os trabalhos ainda não utilizam os sensores veiculares de forma holística como a plataforma proposta, mas realizam análises com alguns sensores, de forma isolada. Assim, a partir da discussão apresentada anteriormente, é visível que ainda existem lacunas a serem exploradas nessa área, fomentando o desenvolvimento de novas soluções, através da tecnologia OBD-II e do IoIV (*Internet of Intelligent Vehicles*).

3 Arquitetura para os Veículos Inteligentes

A plataforma de IoIV proposta neste trabalho permite que, a partir da integração de dispositivos, seja possível conectar veículos à Internet, fornecendo funcionalidades de armazenamento, visualização e análise de dados em tempo real.

A arquitetura dessa plataforma conta com os seguintes módulos, conforme ilustrado na Figura 1:

a) Módulo de conexão veicular, que visa conectar a ECU do veículo aos outros dispositivos

da plataforma, através de um adaptador OBD-II;

b) Módulo de captura de dados, que por meio de um aplicativo *mobile* conecta-se via *Bluetooth* ao hardware OBD-II e obtém os dados dos sensores veiculares. Isso é feito através de conexão GPRS (*General Packet Radio Service*) ou Wi-Fi com o módulo de armazenamento de dados;

c) Módulo de armazenamento de dados, que possui a responsabilidade de armazenar e servir os dados para as aplicações de análise por intermédio de uma API REST (do inglês, *Representational State Transfer*).

As próximas subseções detalham a estrutura de cada módulo desenvolvido.

3.1 Módulo de Conexão Veicular

A conexão veicular é realizada a partir da tecnologia OBD-II, que é um sistema conectado à ECU do veículo, provendo uma interface de comunicação com o veículo que permite a leitura e transmissão dos dados aferidos pelos sensores e atuadores do veículo em tempo real.

Essa interface permite conexão ao veículo por meio de diferentes protocolos, no entanto, em geral, cada veículo implementa apenas um deles. Assim, o protocolo mais utilizado é o CAN (do inglês, *Controller Area Network*), que foi criado pela *Bosh* em 1980. Sua criação teve como principal objetivo permitir a comunicação entre diferentes ECUs. Além desse protocolo, também são muitos utilizados os protocolos: *SAE J1850 PWM*, *SAE J1850 VPW*, *ISO 9141-2* e *ISO 14230 KWP2000* (Baek and Jang, 2015).

Para acessar os dados fornecidos pelo veículo, existe um conector de interface OBD-II, que é conhecido por DLC (do inglês, *Data Link Connector*). Um DLC possui 16 pinos e é utilizado em praticamente todos os veículos produzidos a partir de 1996. Dessa forma, podem ser utilizados *scanners* OBD (também referidos por "adaptadores"), que são diretamente conectados ao DLC e agem como intermediários, permitindo a comunicação de aplicações externas com o veículo.

A tecnologia OBD-II conta com 10 modos de operação, em que cada um é destinado a uma finalidade específica. Cada um deles possui uma série específica de comandos disponíveis para retornar os dados dos sensores e atuadores presentes no veículo (Oliveira et al., 2017). Para requisitar esses dados são utilizados códigos denominados PIDs (do inglês, *Parameter IDs*). No entanto, as montadoras não são obrigadas a suportar todos os modos de operação em seus veículos, fazendo-se necessária a decodificação dos comandos suportados pelo automóvel em questão. A Tabela 1 sintetiza a finalidade de cada um dos modos de operação do OBD-II.

Tabela 1: Modos de Operação do OBD-II.

Modo	Descrição
01	Retorna os dados da ECU tempo real.
02	Requisita os dados da ECU correspondentes à última falha.
03	Exibe os códigos de erro armazenados no veículo.
04	Limpa os códigos de erro armazenados.
05	Retorna os resultados dos testes dos sensores de oxigênio presentes no veículo.
06	Retorna os resultados dos testes relacionados ao monitoramento não-contínuo.
07	Retorna os resultados dos testes relacionados ao monitoramento contínuo.
08	Requer o controle dos sistemas de bordo.
09	Obtém informações do veículo.
10	Exibe os códigos de erro com status permanente.

3.2 Módulo de Captura de Dados

Esse módulo possui como finalidade capturar, a partir de uma aplicação móvel, os dados do veículo e enviá-los para um banco de dados na nuvem. Para isso, foi desenvolvida uma API para realizar a conexão entre o *scanner* OBD conectado ao veículo e um *smartphone* (com o sistema operacional *Android*).

A API foi desenvolvida utilizando a linguagem Java, com o intuito de ser utilizada como biblioteca em aplicações *Android* e tomou como base a OBD-II Java API[1] disponibilizada por Paulo Pires no site de compartilhamento de código e software GitHub, que teve seu desenvolvimento suspenso no ano de 2017. Durante a criação dessa nova biblioteca, foram analisados todos os repositórios criados a partir da API em que a mesma foi baseada (por meio dos *forks* do *GitHub*), mesclando assim todas as alterações e correções à nova API desenvolvida. Dessa forma, essa API suporta toda a faixa de comandos compreendida nos modos de operação padrão do OBD-II mencionados na Tabela 1.

Para permitir a agregação de comandos foi utilizado o padrão de projeto *composite*, de forma que, na aplicação podem ser criadas listas de comandos a serem executados, ou seja, permitindo um agrupamento de comandos a serem executados no veículo, e não a execução de um comando de maneira isolada. A execução desse grupo de comandos na API pode se dar de forma sequencial ou concorrente, sendo definido por meio da assinatura do método `run`. Assim, a partir da resposta obtida, o processamento (com os cálculos para obter o valor por meio da resposta do OBD) é feita de forma paralela.

Foi utilizado o paradigma *Fog Computing* na API, permitindo a definição da arquitetura que estende a capacidade computacional e armazenamento da nuvem para as camadas de acesso da rede, garantindo que os dados possam ser analisados e transformados em informações ou em ações antes de serem implementados (Mouradian et al., 2018). Assim, utilizou-se o paradigma com

[1] https://github.com/pires/obd-java-api

Figura 1: Visão Geral da Arquitetura

o processamento e decodificação das respostas obtidas pelo veículo. Pelo fato do protocolo CAN funcionar de maneira síncrona, a API foi desenvolvida para suportar a concorrência, garantindo a sincronização na execução dos comandos e permitindo que as operações ocorram de forma concorrente, diminuindo a latência na execução e processamento dos comandos. Para isso, foram utilizados bloqueios (*locks*) como método de sincronização.

Ainda na API, um algoritmo foi implementado para identificação dos comandos suportados pelo veículo. Esse algoritmo é calculado a partir do comando "`01 00`" (modo 01, PID 00). O resultado (retornado na base hexadecimal) possui 4 *bytes* e é convertido para um número inteiro sem sinal de 32 bits. Os PIDs suportados pelo veículo podem ser expressados pela equação:

$$S = \{\, p \in \mathbb{N} \mid 1 \leq p \leq 32 \wedge bitseq(p) = 1 \,\} \quad (1)$$

$$bitseq(n) = number \gg (32 - n)\ \&\ 1 \quad (2)$$

Em que S (Equação 1) é o conjunto de PIDs suportados pelo veículo e $bitseq_p$ é dada a partir da Função 2, no qual *number* é o número dado a partir do resultado (em hexadecimal) retornado pelo veículo. Essa função realiza $(32 - n)$ deslocamentos para a direita e após isso, uma operação lógica de *and* com 1, retornando, dessa forma, o valor do *n-ésimo* bit do número. Esse algoritmo pode ser melhor visualizado na Figura 2 (as colunas mais escuras representam os comandos suportados).

HEX	B				E				...	1				1			
BIN	1	0	1	1	1	1	1	0	...	0	0	0	1	0	0	0	1
PID	01	02	03	04	05	06	07	08		19	1A	1B	1C	1D	1E	1F	20

Figura 2: Decodificação do Comando de PIDs suportados

Outro ponto importante na API desenvolvida para comunicação com o dispositivo OBD-II é de sua implementação agnóstica à tecnologia de transmissão de dados, visto que foi implementada utilizando as classes `InputStream` e `OutputStream`, para a solicitação e resposta dos comandos, respectivamente. Dessa forma, a aplicação não se limita apenas a um protocolo de comunicação ou DLC.

Para a aplicação *mobile*, houve o cuidado de se implementar uma máquina de estados para que houvesse a previsão de erros e economia da bateria, a qual pode ser visualizada na Figura 3. A aplicação possui os seguintes estados:

a) "*Ready*", representa o estado inicial ideal da aplicação e somente a partir deste que as ações do sistema podem ser executadas;

b) "*Not Ready*", é o estado quando um requisito da aplicação não é atendido, bloqueando-a até que os mesmos sejam atendidos. A entrada e saída desse estado são realizadas de forma automática pela aplicação;

c) "*Connecting*", representa o estado quando a conexão é iniciada. Neste estado, é estabelecida a conexão com a interface OBD-II conectada ao veículo e criado um *socket* para comunicação. Caso um erro ocorra durante esse processo, o estado é alterado para "*disconnecting*".

d) "*Connected*", é atingido caso a conexão tenha sido estabelecida corretamente no estado anterior, é onde ocorre a troca de dados por meio do *socket* criado anteriormente;

e) "*Disconnecting*" é o estado utilizado para finalizar a conexão com o veículo. Pode ser alcançado através de uma ação do usuário (desconectar) ou caso um erro ocorra durante a conexão. Nele ocorrem as ações de limpeza e o encerramento da conexão, voltando assim para o estado inicial da aplicação.

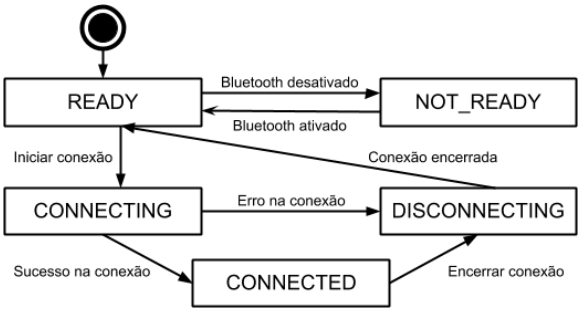


Figura 3: Máquina de estados da aplicação

Outra característica da API foi a implementação de um módulo inteligente de envio de dados, no qual a aplicação decide de forma automática a tecnologia de comunicação a ser utilizada para o envio dos dados ao servidor. Assim, dependendo da conexão e das preferências do usuário (há a possibilidade de desabilitar a utilização de dados móveis na aplicação), será decidido, dentre as tec-

nologias GPRS e Wi-Fi, qual deverá ser utilizada de acordo com a que possuir melhor disponibilidade no momento.

Os dados coletados pelo aplicativo são enviados para o servidor de armazenamento em nuvem utilizando o formato JSON (*JavaScript Object Notation*, por se mostrar um formato otimizado para aplicações Web (Gil and Trezentos, 2011). O objeto enviado segue a estrutura apresentada abaixo.

```
{
  "vehicle_id": "93YBSR7RHEJ2*****",
  "timestamp": "1515511139599",
  "altitude": "45",
  "latitude": "-5.8321709",
  "longitude": "-35.2076702",
  "dadosMap": {
    "AIR_INTAKE_TEMP": "53.0",
    "DISTANCE_TRAVELED_AFTER_CC": "63847",
    "DISTANCE_TRAVELED_MIL_ON": "0",
    "DTC_NUMBER": "0",
    "ENGINE_COOLANT_TEMP": "91.0",
    "ENGINE_LOAD": "47.058823",
    "ENGINE_RPM": "786",
    "INTAKE_MANIFOLD_PRESSURE": "46",
    "LONG_TERM_BANK_1": "-3.90625",
    "PENDING_TROUBLE_CODES": "",
    "SHORT_TERM_BANK_1": "-1.5625",
    "SPEED": "17",
    "THROTTLE_POS": "16.078432",
    "TIMING_ADVANCE": "-28.5",
    "TROUBLE_CODES": ""
  }
}
```

Conforme explicitado no exemplo no formato JSON, 6 campos são enviados para o servidor. O primeiro, denominado "`vehicle_id`", corresponde à identificação do veículo, em que foi utilizado como padrão o VIN (do inglês, *Vehicle Identification Number*) que é conhecido popularmente como "número de chassi", que é uma *string* única de 17 caracteres utilizada para identificar um veículo. O campo "`timestamp`" corresponde à data em que a coleta foi realizada (utilizando o padrão *timestamp*). Já os campos de altitude, latitude e longitude são recuperados pelo GPS do *smartphone* no momento em que a coleta foi realizada. Por último, há o campo "`dadosMap`", que é um objeto contendo uma série de pares nome/valor, em que o nome representa o nome do sensor, e o valor, o dado aferido e processado.

3.3 Módulo de Armazenamento de Dados

Esse módulo da plataforma é responsável por armazenar e distribuir os dados, e pode ser estendido por meio da criação de submódulos. Seu principal submódulo é o "Historiador de dados veiculares", provendo uma API (utilizada pela aplicação *mobile* para a comunicação com esse módulo) em que qualquer subaplicação pode inserir ou consumir os dados armazenados no historiador por meio de seus pontos de acesso.

A ideia do desenvolvimento descentralizado desse módulo é permitir a expansão e o desenvolvimento de subaplicações seguindo o princípio da responsabilidade única. Assim, permite-se que existam submódulos especializados em cada parte da análise, processamento e visualização de dados da plataforma.

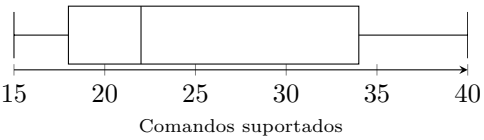
4 Resultados

Para a utilização da API proposta, foram realizados alguns experimentos iniciais. O objetivo desses experimentos é avaliar sua viabilidade prática com relação a captura dos sensores veículares. Assim, apresentam-se como resultados os dados de capturas dos sensores, coletados de veículos com marcas e fabricantes distintos, por meio de conveniência. Dessa forma, conectou-se o OBD nos veículos e depois a API para realizar a captura de dados dos sensores, demostrados na Tabela 2, da qual podem ser realizadas avaliações estatísticas a respeito do suporte a comandos pelos veículos. A distribuição dos resultados pode ser melhor visualizada a partir do *Box Plot* 1.

Tabela 2: Captura dos Sensores por veículos

Modelo	**Ano**	**Sensores**
Chevrolet Celta	2012	20
Chevrolet Onix	2015	39
Fiat Argo	2017	28
Fiat Doblo	2014	18
Fiat Mobi	2017	24
Fiat Siena	2011	18
Fiat Strada	2013	18
Fiat Uno	2013	18
Ford Ka	2013	15
Hyundai Tucson	2010	34
Mitsubishi L200 Triton	2013	28
Renault Sandero	2014	22
Toyota Corolla	2015	40
Volkswagen Fox	2014	18
Volkswagen Gol	2014	39
Volkswagen Up	2015	38

Box Plot 1: Distribuição do número de comandos suportados por veículo



A partir da investigação dos resultados iniciais obtidos, constatou-se um crescimento no tamanho do conjunto de comandos suportados por todos os veículos (intersecção) conforme o ano de fabricação do veículo aumenta. Foi verificado ainda que mesmo em carros de mesma categoria, há uma diferença no número de comandos suportados, como no caso envolvendo o Fiat Mobi e o Volkswagen Up, em que o segundo, mesmo tendo seu ano de fabricação anterior ao primeiro, dispõe de um número superior de sensores. Com essa investigação percebeu também que veículos populares possuem quantidades de sensores similares a carros de luxos, como no caso do Chevrolet Onix, Volkswagen Up e o Corolla, ambos de 2015.

No geral, os testes iniciais realizados são um indicativo dos benefícios da utilização da API proposta para monitoramento veicular inteligente.

5 Considerações Finais

Este artigo apresentou a proposta de uma plataforma para Veículos Inteligentes, que tem como objetivo coletar e analisar, a partir do hardware OBD-II, todos os sensores disponibilizados pelos veículos. Espera-se que a API proposta possa trazer diversos benefícios à pesquisas e desenvolvimentos nessa área.

Durante a utilização da plataforma foram realizados testes em diversos veículos. Com isso, foi percebido que alguns carros (Renault Master Van, Citröen Xsara Picasso, entre outros) produzidos em 2010, mesmo possuindo a interface OBD, não permitiram a conexão com o automóvel. Esse fato ocorreu pois esse foi o ano em que houve a obrigatoriedade da implantação dessa tecnologia nos veículos comercializados no Brasil, e por se tratar de veículos fabricados fora do Brasil, foi utilizado um protocolo próprio de comunicação, impossibilitando a coleta de dados.

Além disso, foram encontradas outras dificuldades no decorrer do desenvolvimento da plataforma, como o tempo para a requisição dos comandos, problema esse solucionado a partir do algoritmo de decodificação dos comandos suportados, que possibilitou a redução desse tempo de 8 segundos para 1.5 segundos. Outra dificuldade foi durante o período de implantação da aplicação em estudo de caso, visto que por não se tratar de um ambiente controlado (como em um simulador), a conexão está propensa a erros, que devem ser tratados.

Como trabalhos futuros destacam-se: a implementação da comunicação com o *scanner* OBD-II por meio do Wi-Fi, fazendo uso da implementação agnóstica da API proposta; a adição do algoritmo de consumo instantâneo de combustível, possibilitando que sejam mensurados níveis de poluição (Oliveira et al., 2017); a utilização de "*Protocol Buffers*" para o envio de dados para o servidor, visto que essa tecnologia possibilita uma economia no uso de dados móveis por parte de aplicações móveis (Sumaray and Makki, 2012); a ampliação do estudo de caso para validação da plataforma proposta; e a utilização de *Big Data* e técnicas de aprendizado de máquina para detecção de padrões a partir dos dados coletados.

Agradecimentos

A pesquisa e desenvolvimento desse artigo teve o suporte do CAPES e do Instituto Metrópole Digital (IMD) por meio do projeto *Smart Metropolis*.

Referências

Baek, S.-h. and Jang, J.-W. (2015). Implementation of integrated obd-ii connector with external network, *Information Systems* **50**: 69–75.

Chen, Y., Xiang, Z., Jian, W. and Jiang, W. (2009). An improved aomdv routing protocol for v2v communication, pp. 1115–1120.

Contreras-Castillo, J., Zeadally, S. and Ibáñez, J. A. G. (2017). A seven-layered model architecture for internet of vehicles, *Journal of Information and Telecommunication* **1**(1): 4–22.

Dandala, T. T., Krishnamurthy, V. and Alwan, R. (2017). Internet of vehicles (iov) for traffic management, *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, pp. 1–4.

Gil, B. and Trezentos, P. (2011). Impacts of data interchange formats on energy consumption and performance in smartphones, *Proceedings of the 2011 workshop on open source and design of communication*, ACM, pp. 1–6.

Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions, *Future generation computer systems* **29**(7): 1645–1660.

Kim, H.-s., Jang, S.-j. and Jang, J.-w. (2015). A study on development of engine fault diagnostic system, *Mathematical Problems in Engineering* **2015**.

Malekian, R., Moloisane, N. R., Nair, L., Maharaj, B. and Chude-Okonkwo, U. A. (2017). Design and implementation of a wireless obd ii fleet management system, *IEEE Sensors Journal* **17**(4): 1154–1164.

Meseguer, J. E., Calafate, C. T., Cano, J. C. and Manzoni, P. (2015). Assessing the impact of driving behavior on instantaneous fuel consumption, *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, IEEE, pp. 443–448.

Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J. and Polakos, P. A. (2018). A comprehensive survey on fog computing: State-of-the-art and research challenges, *IEEE Communications Surveys Tutorials* **20**(1): 416–464.

Oliveira, J., Lemos, J., Vieira, E., Silva, I., Abrantes, J., Barros, D. and Costa, D. G. (2017). Co2catcher: A platform for monitoring of vehicular pollution in smart cities, *2017 IEEE First Summer School on Smart Cities*, IEEE, pp. 37–42.

Pu, L., Liu, Z., Meng, Z., Yang, X., Zhu, K. and Zhang, L. (2016). Implementing on-board diagnostic and gps on vanet to safe the vehicle, pp. 13–18.

Sumaray, A. and Makki, S. K. (2012). A comparison of data serialization formats for optimal efficiency on a mobile platform, *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '12, ACM, New York, NY, USA, pp. 48:1–48:6.

Yang, F., Wang, S., Li, J., Liu, Z. and Sun, Q. (2014). An overview of internet of vehicles, *China Communications* **11**(10): 1–15.

Yang, Y., Chen, B., Su, L. and Qin, D. (2015). Research and development of hybrid electric vehicles can-bus data monitor and diagnostic system through obd-ii and android-based smartphones, *Advances in Mechanical Engineering* **5**: 741240.

Zanella, A., Bui, N., Castellani, A., Vangelista, L. and Zorzi, M. (2014). Internet of things for smart cities, *IEEE Internet of Things journal* **1**(1): 22–32.