



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
TECNÓLOGO EM REDES DE COMPUTADORES

ANTONIO MATEUS DE SOUSA

**AVALIAÇÃO DE DESEMPENHO DE *COMPUTATION OFFLOADING* EM NUVENS
VEICULARES**

QUIXADÁ – CEARÁ

2016

ANTONIO MATEUS DE SOUSA

AVALIAÇÃO DE DESEMPENHO DE *COMPUTATION OFFLOADING* EM NUVENS
VEICULARES

Monografia apresentada no curso de Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de tecnólogo em Redes de Computadores. Área de concentração: Computação.

Orientador: Prof. Msc. Alisson Barbosa de Souza

QUIXADÁ – CEARÁ

2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S696a Sousa, Antonio Mateus de.
Avaliação de desempenho de computation offloading em nuvens veiculares / Antonio Mateus de Sousa. –
2016.
52 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá,
Curso de Redes de Computadores, Quixadá, 2016.
Orientação: Prof. Me. Alisson Barbosa de Souza.

1. Computation offloading. 2. Redes veiculares. 3. Computação em nuvem. I. Título.

CDD 004.6

ANTONIO MATEUS DE SOUSA

AVALIAÇÃO DE DESEMPENHO DE *COMPUTATION OFFLOADING* EM NUVENS
VEICULARES

Monografia apresentada no curso de Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de tecnólogo em Redes de Computadores. Área de concentração: Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Msc. Alisson Barbosa de Souza (Orientador)
Campus Quixadá
Universidade Federal do Ceará – UFC

Prof. Dr. Paulo Antonio Leal Rego
Campus Quixadá
Universidade Federal do Ceará - UFC

Prof. Msc. Marcos Dantas Ortiz
Campus Quixadá
Universidade Federal do Ceará - UFC

Aos meus familiares e amigos.

AGRADECIMENTOS

Agradeço a Deus por me permitir viver este momento. Ao meu orientador, professor Alisson Barbosa de Souza, pela orientação e apoio e aos demais professores que me ajudaram e incentivaram nesta jornada e, por fim, agradeço a minha família e amigos por toda ajuda durante este percurso.

“ O que sabemos é uma gota; o que ignoramos é um oceano”.

(Isaac Newton)

RESUMO

As VANETs (*Vehicular Ad Hoc Networks*), ou redes veiculares, visam fornecer segurança e conforto para os motoristas nas estradas e rodovias. Entretanto, os veículos estão evoluindo rapidamente no quesito de poder computacional, ou seja, os veículos estão saindo de fábrica com dispositivos de alto desempenho mas que permanecem, na maioria do tempo, ociosos ou sendo desperdiçados. VANET-Cloud tenta integrar os conceitos de computação em nuvem em redes veiculares com o objetivo de sanar esse problema de subutilização de recursos por meio do fornecimento dos mesmos como um serviço na nuvem. Tais serviços podem ser fornecidos na forma de acesso à Internet, recursos computacionais, compartilhamento de informações e armazenamento. Neste trabalho, abordamos o *computation offloading* em VANETs como uma técnica de utilização oportunística de recursos ociosos presentes nos veículos para a execução de tarefas complexas.

Palavras-chave: VANETs. Computação em nuvem. Computation offloading.

ABSTRACT

VANETs (Vehicular Ad Hoc Networks), or vehicular networks aim to provide safety and comfort for drivers on the roads and highways. However, vehicles are rapidly evolving in the category of computing power, ie vehicles are leaving the factory with high-performance devices but remain in the majority of time idle or being wasted. VANET-Cloud tries to integrate cloud computing concepts in vehicle networks in order to remedy this underutilized resource problem by supplying the same as a cloud service. Such services can be provided in the form of Internet access, computer resources, information sharing and storage. In this paper we report the *computational offloading* in VANETs as an opportunistic use of technical idle resources present in vehicles to perform complex tasks.

Key-words: VANETs. Cloud computing. Computation offloading.

LISTA DE FIGURAS

Figura 1 – Modelo de comunicação em VANET-Cloud	21
Figura 2 – <i>Computation Offloading</i>	23
Figura 3 – Técnica de <i>offloading</i> em VANETs.	32
Figura 4 – Divisão e envio da tarefa para os <i>surrogates</i>	33
Figura 5 – Modelo Manhattan.	36
Figura 6 – Comparação de execução Local vs execução com Offloading (Matriz 1000x1000)	38
Figura 7 – Comparação de execução Local vs execução com Offloading (Matriz 2000x2000)	39
Figura 8 – Comparação de execução Local vs execução com Offloading (Matriz 5000x5000)	40
Figura 9 – Comparação do uso de CPU na execução Local vs execução com Offloading (Matriz 1000x1000)	41
Figura 10 – Comparação do uso de CPU na execução Local vs execução com Offloading (Matriz 2000x2000)	42
Figura 11 – Comparação do uso de CPU na execução Local vs execução com Offloading (Matriz 5000x5000)	43
Figura 12 – RTTs coletados na simulação no NS-3	44

LISTA DE TABELAS

Tabela 1 – Comparação entre os trabalhos relacionados	30
Tabela 2 – Ambiente de testes	35
Tabela 3 – Ambiente de simulação	36
Tabela 4 – Tempo de execução somado ao RTT	45

LISTA DE ABREVIATURAS E SIGLAS

VANET	<i>Veicular Ad Hoc Network</i>
V2V	<i>Vehicle-to-Vehicle</i>
V2I	<i>Vehicle-to-Infrastructure</i>
OBU	<i>On Board Unit</i>
RSU	<i>Road Side Unit</i>
MANET	<i>Mobile Ad Hoc Network</i>
VCC	<i>Veicular Cloud Computing</i>
VC	<i>Veicular Clouds</i>
VuC	<i>Vehicles using Cloud</i>

SUMÁRIO

1	INTRODUÇÃO	13
2	FUNDAMENTAÇÃO TEÓRICA	15
2.1	<i>Cloud computing</i>	15
2.1.1	<i>Características essenciais</i>	16
2.1.2	<i>Modelos de serviço</i>	17
2.1.3	<i>Modelos de implantação</i>	17
2.2	<i>Vehicular Ad-hoc Networks</i>	18
2.2.1	<i>Padronização em VANETs</i>	18
2.2.2	<i>Roteamento em VANETs</i>	19
2.3	<i>Mobile Cloud Computing</i>	20
2.3.1	<i>VANET-Cloud</i>	20
2.3.2	<i>Taxonomia de VANET-Cloud</i>	22
2.3.3	<i>Modelos de serviço em VCC</i>	22
2.4	<i>Computation offloading</i>	23
2.4.1	<i>Vantagens do Offloading</i>	24
2.4.2	<i>Desvantagens do Offloading</i>	24
2.4.3	<i>Migração de código</i>	25
2.5	<i>Aplicações relacionadas a VCC</i>	25
2.5.1	<i>Otimização de sinais de tráfego</i>	25
2.5.2	<i>Gerenciamento de parques de estacionamento</i>	26
3	TRABALHOS RELACIONADOS	27
4	OBJETIVOS	31
4.1	Objetivo geral	31
4.2	Objetivos específicos	31
5	TÉCNICA DE <i>COMPUTATION OFFLOADING</i> UTILIZADA EM VANETS	32
5.1	Workload utilizado	33
5.2	Ambiente de testes	35
6	RESULTADOS	38
6.1	Tempo de execução	38
6.2	Uso de CPU	41

6.3	<i>Round Trip Time</i>	43
7	CONCLUSÃO	46
8	TRABALHOS FUTUROS	47
	REFERÊNCIAS	48

1 INTRODUÇÃO

Com o passar do tempo os veículos estão se tornando mais sofisticados com capacidades computacionais e dispositivos de armazenamento a bordo, poderosos recursos computacionais, capacidades de comunicação e menores limitações de energia (FLEMING, 2012).

Dessa forma, os serviços fornecidos pelas chamadas *Vehicular Ad hoc Networks* (VANETs), ou redes veiculares vão além do fornecimento de serviços de segurança e acesso à *Internet*. Veículos agora são vistos como detentores de alto poder computacional e fornecedores de serviços como conectividade, armazenamento, coleta de dados de GPS, dentre outros (FAROOQ; PASHA; KHAN, 2014).

Percebendo o poder computacional presente nos veículos, iniciou-se a pesquisa sobre computação em *VANET-Cloud*, ou nuvens veiculares, que permitem que recursos computacionais presentes nos veículos possam ser integrados com o ambiente de nuvem tradicional, que consiste apenas em entidades computacionais estacionárias (BITAM; MELLOUK; ZEADALLY, 2015). Veículos serão tratados como recursos computacionais subutilizados, que poderiam estar fornecendo serviços públicos. Todos os dias muitos veículos gastam horas em engarrafamentos e rodovias (WHAIDUZZAMAN et al., 2014).

Em *VANET-Cloud*, análogo à computação em nuvem, há três principais modelos de serviços, são eles *Network as a service* (NaaS), *Storage as a Service* (StaaS) e *Data as a Service* (DaaS) (FAROOQ; PASHA; KHAN, 2014). Para os consumidores poderem acessar estes serviços, os mesmos precisam utilizar um protocolo de descoberta de serviços que fará uma busca por possíveis veículos que estão aptos a realizar as tarefas solicitadas.

No futuro, motoristas presos em engarrafamentos irão aceitar doar seus recursos computacionais, ou exigir algo em troca, para ajudar entidades externas a executarem simulações complexas que demandam grande processamento como, por exemplo, o reescalonamento de semáforos, ou em casos de emergência como desastres naturais ou evacuações não planejadas, uma federação de nuvens veiculares será formada para efetuar tarefas de cálculo de tempo de viagem e avaliar os recursos disponíveis como água, comida, abrigo e gasolina (WHAIDUZZAMAN et al., 2014).

Uma vez que os recursos presentes nos veículos estejam acessíveis a terceiros poderá ser aplicada a técnica de *computation offloading*, que consiste na divisão e envio de uma tarefa complexa para ser executada em um dispositivo de poder computacional maior ou igual, que neste

caso seria representado por um veículo com recursos computacionais acessíveis e, possivelmente, ociosos ou subutilizados (LI et al., 2014).

Este trabalho tem como objetivo avaliar a realização de um *offloading* de processamento em VANETs. Desta forma, após a conclusão do trabalho, é possível avaliar se a técnica de *offloading* em VANETs é viável e indicada para tarefas computacionalmente complexas.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta Seção apresentamos uma visão geral sobre os conceitos abordados neste trabalho. Na Seção 2.1, introduzimos os conceitos de *Cloud Computing*; na Seção 2.2, abordamos os conceitos de *Vehicular Ad hoc Networks*; na Seção 2.3, explanamos *Mobile cloud computing*; na Seção 2.4, apresentamos os conceitos de *Computation Offloading*; por fim, na Seção 2.5, apresentamos algumas das aplicações relacionadas à *Vehicular Cloud Computing* (VCC).

2.1 *Cloud computing*

Pode-se notar a rápida evolução de tecnologias como armazenamento, processamento e o sucesso da *Internet*. Devido a tais avanços, os recursos tornam-se cada vez mais baratos, poderosos e acessíveis do que nunca. Esse cenário abriu caminho para um novo modelo computacional chamado de *cloud computing*, ou computação em nuvem, que permite o acesso rápido e sob demanda a diversos recursos computacionais (ZHANG; CHENG; BOUTABA, 2010).

De acordo com o NIST, ou *National Institute of Standards and Technology* (MELL; GRANCE, 2011), computação em nuvem é um modelo para permitir o acesso conveniente, sob demanda de rede a um conjunto compartilhado de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com um esforço mínimo de gerenciamento ou de interação com provedor de serviços.

As vantagens oferecidas pela computação em nuvem, como: facilidade de implantação (sem a necessidade em investir em infraestrutura), alta escalabilidade, redução dos riscos de negócios e gastos com manutenção levaram as empresas a migrarem seus sistemas para a nuvem (POLASH; ABUHUSSEIN; SHIVA, 2014).

Tecnologias anteriores a computação em nuvem foram: *Grid computing*, *Utility computing* e *Autonomic computing*, segundo Zhang, Cheng e Boutaba (2010).

- A tecnologia de *Grid computing* consiste em recursos computacionais distribuídos que eram interconectados para alcançar um objetivo computacional em comum. O desenvolvimento desta tecnologia visava, principalmente, a execução de aplicações que exigiam computação intensiva (cálculos complexos, por exemplo). No sentido de empregar recursos distribuídos para alcançar um objetivo em comum, *grid computing* é

semelhante à computação em nuvem. Mas a computação em nuvem é mais eficiente, pois utiliza a tecnologia de virtualização em altos níveis como *hardware* e plataformas de aplicação.

- *Utility computing* é um modelo de fornecimento de serviços sob demanda que cobrava o uso dos recursos utilizados a partir de seu uso. A computação em nuvem utiliza alguns dos conceitos presentes na *utility computing*, como recursos sob demanda e uma forma de pagamento baseada na quantidade de recursos utilizados.
- Já a *Autonomic computing*, criada pela IBM, foi pensada para construir um sistema capaz de se auto gerenciar, reagindo a observações externas sem a intervenção humana. Embora a computação em nuvem possua alguns mecanismos de automação nos serviço de fornecimento de recursos, seu objetivo é reduzir os custos dos recursos e não reduzir a complexidade do sistema.

2.1.1 *Características essenciais*

De acordo com Mell e Grance (2011), o modelo de computação em nuvem possui cinco características essenciais, estas características são apresentadas e elucidadas logo abaixo:

1. Serviço sob demanda - o usuário pode adquirir unilateralmente recurso computacional na medida em que necessite e sem precisar de interação humana com os provedores de cada serviço.
2. Amplo acesso - os recursos computacionais são disponibilizados por meio da *Internet* e acessados através de mecanismos padronizados que permitem o uso por dispositivos móveis e estações de trabalho.
3. *Pooling* de recursos - os recursos computacionais do provedor são utilizados para servir múltiplos usuários usando um modelo multi-inquilino (*multi-tenant*), com diferentes recursos físicos e virtuais, dinamicamente atribuídos e ajustados de acordo com a demanda dos usuários. Estes usuários não precisam ter conhecimento da localização física dos recursos computacionais, podendo somente especificar a localização em um nível mais alto de abstração, tais como o país, estado ou centro de dados.
4. Elasticidade rápida - recursos podem ser adquiridos de forma rápida e elástica, em alguns casos automaticamente, caso haja a necessidade de escalar com o aumento da demanda, e liberados, na retração dessa demanda. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer

momento.

5. Medições de serviço - os sistemas de gerenciamento utilizados para computação em nuvem controlam e monitoram automaticamente os recursos para cada tipo de serviço (armazenamento, processamento e largura de banda). O uso dos recursos deve ser transparente para o provedor do serviço, assim como, para o consumidor do serviço utilizado, normalmente se utiliza o modelo de medição chamado *pay-as-you-go*, que cobra somente o que realmente é utilizado.

2.1.2 Modelos de serviço

A NIST dividiu o modelo de computação em nuvem em três modelos de serviço para sua implantação, são eles: *Software as a service* (SaaS), *Platform as a Service* (PaaS) e *Infrastructure as a Service* (IaaS) (MELL; GRANCE, 2011).

- SaaS: É nesse modelo onde são executadas as aplicações e onde os consumidores acessam estes serviços, hospedados na nuvem, através da *Internet*. O consumidor não possui controle sobre recursos de armazenamento, servidores, sistemas operacionais, armazenamento ou mesmo características individuais da aplicação.
- PaaS: nesse modelo, os consumidores desenvolvem suas aplicações e as lançam na nuvem, a partir de ferramentas, linguagens de programação, bibliotecas e serviços fornecidas pelo provedor da nuvem. O consumidor não tem controle sobre recursos de armazenamento, rede, servidores ou sistemas operacionais possuindo apenas o controle sobre a aplicação.
- IaaS: nesse modelo o cliente tem acesso a infraestrutura fornecida pelo provedor de serviço na nuvem e a recursos computacionais como armazenamento, processamento e rede. Com tais recursos, o consumidor está apto para implantar seus serviços.

2.1.3 Modelos de implantação

Os principais modelos de implantação podem ser divididos em três tipos principais, são eles: nuvens públicas, nuvens privadas e nuvens híbridas (ZHANG; CHENG; BOUTABA, 2010).

- nuvens públicas: é quando uma empresa oferece seus recursos como serviços ao público em geral, podendo cobrar pelo uso dos mesmos.
- nuvens privadas: são nuvens privadas, geralmente projetadas e disponibilizadas para uso de uma organização ou empresa.

- nuvens híbridadas: é a união dos modelos de nuvens privadas com as nuvens públicas como tentativa de sanar as limitações presentes em cada uma.

2.2 *Vehicular Ad-hoc Networks*

Vehicular ad hoc networks, ou redes ad hoc veiculares (VANETs) integram as capacidades da nova geração de redes sem fio aos veículos. Tais redes têm como objetivo fornecer conectividade ubíqua aos veículos enquanto trafegam em rodovias ou estradas, e comunicação veículo para veículo, e veículo para infraestrutura, eficiente de modo a possibilitar a implantação dos chamados *Intelligent Transportation Systems* (ITS), ou sistemas de transportes inteligentes (LI; WANG, 2007).

Além disso, as VANETs são uma sub categoria de *Mobile Ad-hoc NETWORKS* (MANETs) que consistem em veículos com a capacidade de comunicar-se entre si e fornecer serviços baseados em duas categorias: serviços de segurança no trânsito e serviços voltados para o conforto do motorista. A rede presente em VANETs é completamente móvel e necessita de pouca ou nenhuma infraestrutura para operar. As principais aplicações para redes veiculares consistem em uma grande variedade de aplicações, como monitoramento de tráfego cooperativo, controle de fluxo de tráfego, prevenção de colisões, informações sobre serviços próximos, serviços de conectividade com a *Internet*, etc (LI; WANG, 2007).

Veículos são providos de dispositivos chamados *On Board Unit* (OBU), ou unidade de bordo, para realizar a comunicação. A comunicação pode ocorrer de duas formas: *Vehicle-to-Vehicle* (V2V), ou veículo para veículo, e *Vehicle-to-Infrastructure* (V2I), ou veículo para infraestrutura. Na comunicação V2V, o veículo conversa de forma *ad hoc* com outros veículos sem a necessidade de infraestrutura, já na comunicação V2I há uma entidade chamada *Road Side Unit* (RSU), ou unidade de acostamento, que realiza a função de roteador, encaminhando as mensagens proveniente dos veículos. Devido à natureza altamente dinâmica das VANETs, o desenvolvimento de protocolos de roteamento se torna uma tarefa desafiadora e complexa (SINGH; AGRAWAL, 2014).

2.2.1 *Padronização em VANETs*

Com a quantidade crescente de veículos nas ruas, nota-se também um aumento no número de acidentes. Então, há uma necessidade de comunicação entre os veículos para

diminuir este cenário de acidentes. Em virtude disso, a comissão federal de comunicação dos Estados Unidos alocou 75 MHz do espectro de 5,9 GHz para o *Dedicated Short Range Communication* (DSRC), que é um padrão para comunicação para de serviços de médio alcance (BHOI; KHILAR, 2014). O principal objetivo do DSRC e do padrão 802.11p *Wireless Access for Vehicular Environment* (WAVE) é definir um conjunto de regras que possibilitem diminuir o atraso na comunicação entre os veículos, rápido reconhecimento da rede e alta vazão na comunicação (CAMPOLO; MOLINARO, 2011).

O padrão 802.11p define o funcionamento das camadas físicas e de controle de acesso ao meio (MAC) para redes veiculares. Entretanto, a arquitetura WAVE não se limita apenas a camada MAC e física (SOUZA et al., 2013). Essa arquitetura é dividida em cinco documentos distintos:

- Padrão IEEE 1609 define outras camadas de pilhas de protocolos, como camada de rede alternativa à camada IP, camada de segurança, operações em múltiplos canais de comunicação, dentre outros;
- O padrão IEEE 1609.1 é responsável pelo gerenciamento da utilização de recursos como memória e processamento nos *On Board Units* (OBUs), que são os computadores de bordo presente nos veículos, e *Road Side Units* (RSUs), ou unidades de acostamento que atuam como *gateways* para os veículos.
- O padrão IEEE 1609.2 realiza o processamento de forma segura das mensagens, além de definir quando elas devem ser processadas.
- Já o padrão IEEE 1609.3 responde pelos serviços das camadas de rede e transporte, como roteamento e endereçamento. Além disso, esse padrão também é responsável pela configuração e manutenção do sistema.
- Por fim, o padrão IEEE 1609.4 define quais modificações são feitas no padrão IEEE 802.11, para permitir a operação em múltiplos canais. Essa ação é realizada por meio da classificação dos pacotes indicando quais pacotes vão ser comutados no canal de controle ou para um dos canais de serviço.

2.2.2 Roteamento em VANETs

Como já mencionado o roteamento em VANETs é uma tarefa bastante complexa devido à sua mobilidade e topologia dinâmica. Os protocolos de roteamento em VANETs podem ser divididos da seguinte forma: protocolos baseados em topologia, baseado em posicionamento,

baseado em *cluster*, baseado em *geocast* e baseado em *broadcast* (BHOI; KHILAR, 2014).

- O roteamento baseado em topologia é dividido em protocolos de roteamento reativo e pró-ativo. Nos protocolos de roteamento reativo, as tabelas de roteamento são montadas sob demanda, ou seja, somente quando necessário, reduzindo assim o *sobrecarga* na rede. Já os protocolos do tipo pró-ativo atualizam suas tabelas em intervalos regulares de tempo, isto mantém as tabelas sempre atualizadas, mas sobrecarregam a rede.
- Protocolos de roteamento baseados em posicionamento geográfico utilizam informações de localização para selecionar o próximo salto e encaminhar mensagens.
- Protocolos de roteamento baseado em *cluster* são caracterizados por um grupo de nós, onde um nó pertencente a este grupo é escolhido como líder para encaminhar mensagens.
- No roteamento *multicast* os pacotes são enviados de um grupo de zero ou mais nós, onde o endereço de destino é único. Portanto esse tipo de roteamento é o mais eficiente para a comunicação em grupos (SOUZA et al., 2013).
- No roteamento baseado em *geocast* as mensagens são entregues para veículos em uma região por um serviço *multicast*.
- Por fim, no roteamento *broadcast* as mensagens são enviadas para todos os veículos e os RSUs.

2.3 Mobile Cloud Computing

Mobile cloud computing (MCC) é um paradigma emergente onde dispositivos móveis podem ser tanto consumidores quanto fornecedores de serviços (GERLA, 2012). MCC consiste na ideia de que, sem investimento em infraestrutura, empreendimentos podem funcionar alugando a infraestrutura e *software* necessários (WHAIDUZZAMAN et al., 2014).

MCC herda as características intrínsecas à computação em nuvem, como os modelos de serviços, serviços acessíveis globalmente, capacidade de acessar serviços a qualquer hora, sem a necessidade de qualquer planejamento de implantação (WHAIDUZZAMAN et al., 2014).

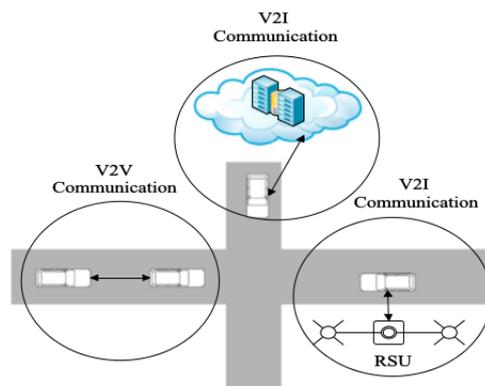
2.3.1 VANET-Cloud

VANET-Cloud tem como principal objetivo utilizar o excesso de poder computacional presente nos veículos. O vasto número de veículos nas ruas e rodovias serão considerados como recursos computacionais subutilizados. No conceito de *VANET-Cloud*, os

donos destes veículos poderiam alugar estes recursos ociosos, semelhante aos *data centers*, e beneficiar-se economicamente. Para cada serviço fornecido, os provedores podem receber uma espécie de pontuação para, futuramente, usar tais pontos na utilização de outros serviços (MERSHAD; ARTAIL, 2013).

De acordo com Whaiduzzaman et al. (2014), a camada dentro do veículo, é responsável pelo monitoramento das condições do motorista como saúde e humor, além de coletar informações sobre o veículo como pressão dos pneus, temperatura, comportamento do motorista, etc. Dessa forma, a camada dentro do veículo pode determinar se o automóvel, ou motorista está apto a continuar a viagem. A camada de comunicação é dividida em duas subcamadas, são elas a comunicação veículo para veículo (*Vehicle-to-Vehicle-V2V*) e veículo para infraestrutura (*Vehicle-to-Infrastructure-V2I*) (Figura 1).

Figura 1 – Modelo de comunicação em VANET-Cloud



Fonte: (BITAM; MELLOUK; ZEADALLY, 2015)

Na comunicação V2V, os veículos conversam entre si de modo *ad hoc*. Enquanto na comunicação V2I o veículo comunica-se com uma unidade de acostamento ou com um dispositivo que permita o acesso à internet (como *smartphones* e *tablets*). Por fim, a camada de nuvem consiste em três subcamadas internas: aplicação, infraestrutura de nuvem e plataforma de nuvem. Na camada de aplicação localizam-se diversos tipos de serviços, que podem ser acessados remotamente pelos motoristas como nível de combustível, reconhecimento de atividade humana, saúde do motorista, entre outros.

A camada de infraestrutura de nuvem consiste possui duas partes principais: armazenamento em nuvem e processamento em nuvem. No armazenamento em nuvem, os dados coletados pela camada de dentro do veículo serão armazenados em um sistema de

armazenamento baseado na aplicação, enquanto a parte de processamento na nuvem é responsável por calcular as tarefas computacionais a fim de se obter um desempenho maior.

2.3.2 *Taxonomia de VANET-Cloud*

A taxonomia de *VANET-Cloud*, de acordo com Hussain et al. (2012), está dividida em três arquiteturas principais, são elas *Vehicular Clouds* (VC), *Vehicles using Clouds* (VuC) e *Hybrid Clouds* (HC). VC é dividida em duas subseções caracterizadas de acordo com seu movimento: (1) *Static clouds* referem-se a veículos estacionados que fornecem serviços de nuvem; (2) nuvens dinâmicas são criadas de maneira *ad hoc* para realizar uma tarefa ou fornecer um serviço, neste caso os veículo podem estar em movimento ou estacionados.

- VC refere-se a formação de uma nuvem de forma dinâmica, em que nós interessados em formar uma nuvem elegem uma entidade autorizada (EA). Esta autoridade irá enviar solicitação para os nós vizinhos, convidando-os para participarem da nuvem, os nós que desejarem participar da nuvem enviarão *acks* para a EA. Alcançado o limite de nós, a EA irá solicitar a permissão de uma autoridade superior para formar uma nuvem e fornecer recursos. Uma vez recebida a permissão os nós participantes irão compartilhar seus recursos em um rico ambiente virtual. Após a realização das tarefas, a nuvem é dissolvida.
- VuC é, basicamente, quando veículos utilizam RSUs que agem como *gateways* para acessarem a nuvem externa.
- HC é a combinação de VC e VuC, onde VC pode ser o fornecedor de serviços e ao mesmo tempo ser um consumidor.

Neste trabalho atuamos na arquitetura de VC, pois os veículos não acessarão RSUs e nem nuvens externas.

2.3.3 *Modelos de serviço em VCC*

Em VCC existem diversos tipos de serviços que um veículo pode fornecer a outro, como *Network as a Service* (NaaS), *Storage as a Service* (STaaS), *Cooperations as a Service* (CaaS), *Information as a Service* (INaaS), *Data as a Service* (DaaS), *Entertainment as a Service* (ENaaS) (WHAIDUZZAMAN et al., 2014). Mas os tipos de serviços fundamentais são NaaS, STaaS e DaaS (MERSHAD; ARTAIL, 2013).

- Rede como um serviço (NaaS), alguns veículos possuem conexão com a Internet, entretanto a maioria dos carros não possuem acesso à Internet. Veículos que possuem acesso podem

compartilhar este recurso com outros veículos utilizando, por exemplo, conexões 3G provenientes de dispositivos dentro do veículo.

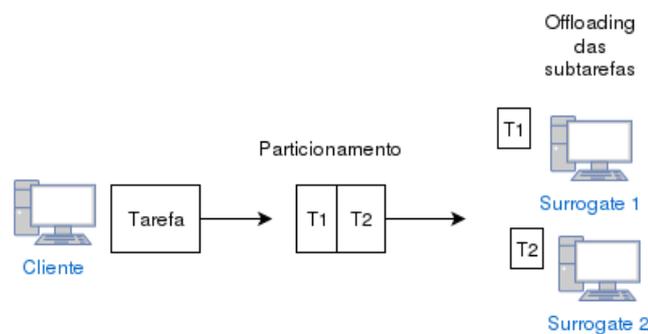
- Armazenamento como um serviço (STaaS), alguns veículos possuem alto poder de armazenamento, mas outros precisam de armazenamento adicional. Veículos detentores de recursos de armazenamento em excesso podem alugar este excesso a outros veículos.
- Dados como um serviço (DaaS), consiste no fornecimento de informações cruciais para o motorista dirigir melhor e com maior segurança, como arquivos de vídeo, mapas da cidade, notícias, condições da estrada, etc.

2.4 Computation offloading

É notável que dispositivos móveis atuais como *smartphones* estão aptos a realizar tarefas de alta complexidade que, por sua vez, sacrificam recursos preciosos do dispositivo, como bateria e largura de banda (SHI et al., 2012; LIN et al., 2015). Uma forma para contornar estas limitações é a técnica de *computation offloading*. *Offloading* refere-se à solução para diminuir a carga de processamento sobre um dispositivo sobrecarregado, por meio da migração da aplicação para uma máquina remota com recursos computacionais disponíveis (KUMAR et al., 2013).

Na Figura 2, podemos notar que o *offloading* computacional é realizado por meio do particionamento de uma aplicação, localizada no cliente, que é dividida em duas subtarefas, que por sua vez serão enviadas aos *surrogates* que executarão as subtarefas. Em seguida, os *surrogates* executam as subtarefas e, após concluir a execução, as reenviam para o cliente que realiza a união dos resultados.

Figura 2 – *Computation Offloading*.



Fonte: Elaborada pelo autor.

Entretanto, em muitos cenários a realização do *offloading* não se dá somente por

motivos de economia de recursos. Ao invés disso, a realização do *offloading* visa questões de desempenho, como execução de tarefas complexas que sobrecarregariam ou apresentariam demora para alcançar um determinado objetivo caso executada em uma única máquina (KUMAR et al., 2013; YANG et al.,).

Em nosso trabalho, utilizamos os conceitos de *computation offloading* para permitir que uma aplicação em execução, que exija grande capacidade computacional, possa ser migrada de um nó para nós que possuam recursos disponíveis que atendam às requisições da tarefa a ser executada.

Para a realização do *offloading*, algumas métricas devem ser escolhidas para determinar qual parte da aplicação será migrada e qual será o *surrogate* mais apropriado para a realização da tarefa. Este processo recebe o nome de *offloading decisions*, ou decisões de *offloading* (KUMAR et al., 2013).

2.4.1 Vantagens do Offloading

De acordo com Khan (2015), a melhora de desempenho de uma aplicação pode ser alcançada a partir da divisão da mesma em vários subprogramas, a cada um desses subprogramas será atribuído a um diferente processador para execução. Com a migração de parte de uma aplicação de um dispositivo de baixo desempenho para outros com recursos computacionais superiores, ou que estão sendo subutilizados, não apenas resolverá o problema da limitação de recursos mas também permitirá utilizar, de forma oportunística, os recursos ociosos nos *surrogates*.

2.4.2 Desvantagens do Offloading

Apesar das diversas vantagens inerentes ao *computation offloading* deve-se considerar alguns pontos importantes antes de realizá-lo, segue abaixo uma listagem desses pontos:

- antes de realizar o *offloading* deve-se avaliar se o mesmo é realmente necessário, ou seja, se aplicação é bastante complexa a ponto de necessitar do *offloading* (FAHIM; MTIBAA; HARRAS, 2013).
- analisar se o *sobrecarga* computacional, no particionamento da aplicação e escolha de *surrogate*, é baixo, moderado ou muito complexo (KOVACHEV, 2012).
- avaliar como a rede será afetada nesse processo.

2.4.3 Migração de código

Migração de código é uma técnica que permite a migração de um código entre os nós presentes na rede que tem como objetivo, em grande parte, a otimização de desempenho na execução de uma determinada aplicação (FUGGETTA; PICCO; VIGNA, 1998).

Há dois modelos principais para migração de código, de acordo com Fuggetta, Picco e Vigna (1998), os mesmos são listados abaixo:

- Mobilidade fraca, consiste na migração de parte do código para uma máquina alvo. Este código, ao chegar na máquina alvo, será executado a partir de seu estado inicial.
- Mobilidade forte, diferente da mobilidade fraca, o código ao chegar na máquina alvo será executado a partir do ponto onde foi suspenso.

Independente do modelo utilizado, outra forma de caracterizar a migração de código, segundo Fuggetta, Picco e Vigna (1998), podem ser as seguintes:

- Migração iniciada pelo emissor, trata-se da migração iniciada na máquina onde o código está atualmente sendo executado. Geralmente este tipo de migração inicia-se quando o emissor realiza o *upload* de um programa para um servidor.
- Migração iniciada pelo receptor, nesse tipo de migração a iniciativa é tomada pela máquina alvo, um notável exemplo desta técnica são os *Java applets*.

2.5 Aplicações relacionadas a VCC

Abordamos nesta Seção algumas das possíveis aplicações em VCC, segundo Whaiduzzaman et al. (2014).

2.5.1 Otimização de sinais de tráfego

Atualmente os sistemas de otimização de sinais de trânsito funcionam de forma isolada operando apenas sobre uma intercessão ou rodovia. Uma desvantagem deste esquema é que o sistema não lida com rápidas mudanças no trânsito. A solução para este problema seria a utilização de *Vehicular Clouds* (VCs) que em conjunto poderiam atuar para melhorar os sistemas de sinais de tráfego.

2.5.2 Gerenciamento de parques de estacionamento

Atualmente nas grandes cidades um dos principais problemas enfrentados pelos motoristas é encontrar uma vaga de estacionamento disponível. Alguns aplicativos presentes em *smartphones* tentam sanar este problema, tais com *Smartpark*, *Parkme* e *ParkMate*. No entanto, estas aplicações necessitam de conexão constante com a *Internet* e, além disso, são limitadas a locais específicos do globo. Com a utilização de esquemas de estacionamento baseado em VANETs, os próprios veículos poderiam fornecer informações sobre as vagas disponíveis em estacionamentos para outros veículos interessados.

3 TRABALHOS RELACIONADOS

Mershad e Artail (2013) propuseram um protocolo de descoberta de serviços em nuvens VANETs chamado CROWN. O protocolo CROWN faz o uso de *Road Side Units* (RSUs), que agem como diretórios de nuvem disponibilizando informações sobre veículos fornecedores de serviços ou *tranSPorTation seRver* (STAR).

Nesse protocolo, os veículos que possuem recursos que estão, possivelmente, sendo desperdiçados podem decidir alugá-los ou até mesmo fornecê-los de forma altruísta. Para tal, o veículo deve registrar-se junto a um RSU, enviando um pacote contendo quais serviços ele oferece e os atributos dos serviços. Esses atributos variam para cada tipo de serviço fornecido em *VANET-Cloud*, além de um tempo estimado de quando o veículo deixará a VANET. O cliente que deseja utilizar os serviços provenientes de um STAR irá enviar um pacote ao RSU. Este pacote irá conter quais serviços o consumidor deseja usar e seus atributos desejados. Daí em diante, o RSU procurará em sua *cache* para determinar qual STAR atende aquela requisição.

Utilizamos uma abordagem semelhante à utilizada pelos autores do protocolo CROWN. Mas, usamos uma técnica descentralizada, sem a dependência de RSUs, onde veículos que desejam realizar o *offloading* se comunicam diretamente com os veículos que estão fornecendo o serviço de processamento de dados.

Grubitzsch e Schuster (2014) utilizaram o já conhecido protocolo XMPP (*eXtensible Messaging and Presence Protocol*) para descoberta de serviços distribuídos em uma nuvem. Na proposta há duas entidades centrais: o *Broker* e o *Runtime*. O *Runtime* é responsável por armazenar e executar as aplicações nele registradas, enquanto o *Broker* indica aos consumidores qual *Runtime* está fornecendo a aplicação que eles desejam utilizar. O *Broker* e o *Runtime* podem ser a mesma entidade. Nesse trabalho, foi acrescentado a técnica de *roster*: onde cada *Runtime* é identificado por um *Jabber Identifier* (JID), que é representada da seguinte forma: *[usernameOfServer].[servicename]@[domain]*. No esquema cada desenvolvedor, cliente e *Runtime* também possui seu próprio JID para garantir sua identidade. Para a validação desse trabalho os autores realizaram um estudo de caso utilizando máquinas virtuais em um laboratório.

Neste trabalho avaliamos apenas o *offloading* computacional, deixamos o protocolo de descoberta e técnica de escolha de substituto para trabalhos futuros.

Já Bravo-Torres et al. (2015) elaboraram um protocolo de roteamento para suportar o *offloading* de dados em *VANET-Cloud*. O protocolo chamado *Virtual Nodes Intersection-Based Routing* (VNIBR) utiliza uma combinação de roteamento topológico e geográfico com três

objetivos principais: endereçar nós pelos IPs, realizar tomada de decisões em interseções e encaminhar pacotes de uma interseção a outra de maneira geográfica. Trabalhos anteriores já introduziram de forma semelhante esta técnica, mas, diferente dos demais, o VNIBR executa sobre uma camada de virtualização que resulta em maior estabilidade nas comunicações.

No trabalho de Li et al. (2014), foi abordado um *framework* para suportar o *offloading*, ou divisão de processamento, em VANET-Cloud. No trabalho, os autores assumiram que os veículos estão distribuídos e movem-se em um plano bidimensional. O *offloading* em VANET-Cloud pode ocorrer de duas formas distintas, são elas: *offloading* entre veículos e *offloading* entre veículos e RSUs. Assim como nesse trabalho, abordamos somente o *offloading* entre veículos, deixando o *offloading* entre veículos e RSUs para trabalhos futuros.

O componente chave e a base para o *framework* de *offloading* é o protocolo de descoberta, segundo os autores. No protocolo criado a busca é feita por meio de inundação da rede, aumentando a probabilidade de encontrar um veículo que satisfaça as métricas. As métricas utilizadas foram as seguintes: o maior tempo de conexão entre um par de veículos, direção, a quantidade de tarefas que o veículo pode realizar.

Por fim, para escolher um veículo para realizar o *offloading*, foram avaliadas quatro estratégias de decisão, são elas: estratégia de seleção aleatória, estratégia de seleção baseada na capacidade computacional, estratégia de seleção baseada na distância e estratégia de seleção multi-atributo. Notou-se que a estratégia de escolha baseada em multi-atributo obteve melhores resultados.

No trabalho acima, os autores utilizaram *flooding*, ou inundação, como técnica de descoberta, onde uma mensagem de descoberta é propagada até encontrar um nó que atenda aos requisitos. Em nossa proposta, limitaremos a área de atuação a um salto para evitar um possível *sobrecarga* na rede e facilitar a operação do envio das subtarefas, uma vez que os *surrogates*, ou substitutos, que são os nós que executarão as subtarefas, estarão próximos o suficiente.

Além disso, em Li et al. (2014) os autores não detalharam o *workload* usado e o algoritmo de particionamento utilizado. Neste trabalho utilizaremos como *workload* a multiplicação de matrizes onde utilizamos um método de particionamento da mesma para dividir seu processamento entre os *surrogates*, ou substitutos.

Em Shi et al. (2012) os autores abordaram o *offloading* em um cenário móvel. Além disso, utilizaram o paradigma chamado *Cirrus Cloud* que utiliza vários recursos computacionais para permitir a continuação da operação em um ambiente de comunicação intermitente. Nesse

trabalho foram realizados testes em dois cenários distintos, no primeiro cenário o dispositivo móvel enfrenta a comunicação intermitente ao tentar acessar os recursos computacionais presentes na nuvem. Já no segundo cenário o dispositivo lida com a comunicação intermitente ao realizar o *offloading* para outro dispositivo. Os autores utilizaram programas para reconhecimento facial e algoritmos complexos como *workload*.

Por fim, os autores concluíram que a proposta utilizando *Cirrus Cloud* obteve bons resultados nos dois cenários, onde reduziu o tempo final de execução da tarefa além de conseguir superar a comunicação intermitente.

No trabalho de Lin et al. (2015), os autores propuseram um *framework* para tomada de decisão na realização do *offloading* que tem como objetivo a redução do tempo de resposta e consumo de energia nos dispositivos móveis. Os experimentos foram executados em diversos modelos de aparelhos celulares, onde realizaram um estudo de caso utilizando multiplicação de matrizes como *workload* em alguns casos. Os experimentos foram comparados com ferramentas já existentes.

Finalmente, os resultados mostraram que o *framework* conseguiu reduzir em torno de 75% do tempo de execução das tarefas e reduzir cerca de 56% o uso de bateria do dispositivo utilizando o *offloading*.

Yang et al. () abordaram o problema do particionamento da aplicação em um ambiente dinâmico, onde os usuários executam aplicações complexas enquanto movem-se pelo cenário. A proposta desse trabalho consiste em um *framework* para a realização do particionamento da aplicação, uma vez que o mesmo deve prever certos pontos no comportamento do ambiente, como *workload* e capacidade de transmissão do enlace.

O *offloading* foi realizado utilizando migração de máquinas virtuais. A execução dos testes desse trabalho utilizaram 23 *access points* na área de teste e usuários movendo-se à velocidades de 0.5 m/s, 10 m/s e 15 m/s. Os resultados mostraram que o *framework* conseguiu diminuir o tempo de execução em cerca de 35%.

A Tabela 1 apresenta um resumo comparativo entre os trabalhos relacionados. Alguns trabalhos não mencionados na tabela utilizaram método de medição usando dispositivos reais e não detalharam o número de nós utilizados e ambientação.

Tabela 1 – Comparação entre os trabalhos relacionados

Trabalhos relacionados	Simulador utilizado	Número de nós	Velocidade max/min
Mershad e Artail (2013)	NS-2	300	15m/s e 30m/s
Bravo-Torres et al. (2015)	NS-3	256, 512, 1024 e 2048	Não informado
Li et al. (2014)	Simulador próprio	Não informado	10Km/h e 30Km/h

Fonte: Elaborada pelo autor.

4 OBJETIVOS

4.1 Objetivo geral

Realizar uma avaliação de desempenho da técnica de *computation offloading* em nuvens veiculares.

4.2 Objetivos específicos

- Identificar problemas relacionados ao *offloading* em nuvens veiculares;
- Analisar:
 - cenários;
 - processamentos;
 - simulações;
 - workload*;
 - algoritmo de particionamento da aplicação;
 - medições.
- Coletar e analisar os dados e resultados;

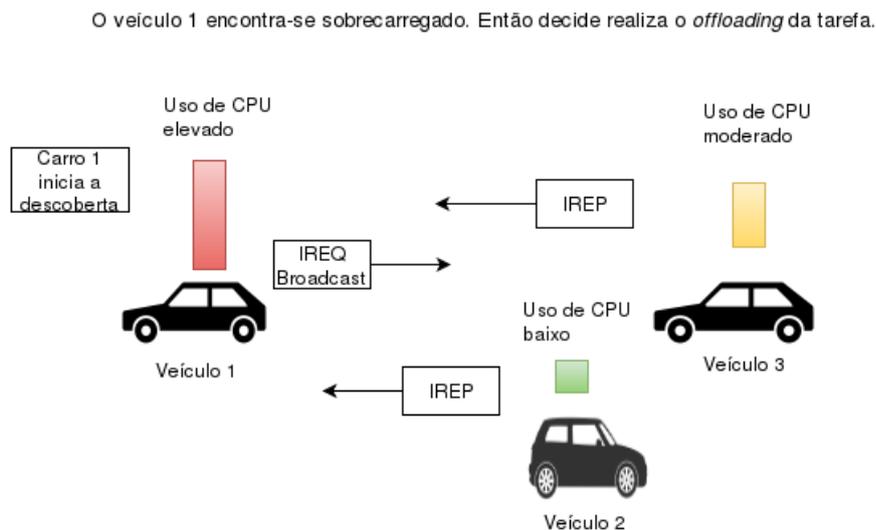
5 TÉCNICA DE COMPUTATION OFFLOADING UTILIZADA EM VANETS

Como explanado anteriormente, o *offloading* computacional pode ser utilizado para reduzir o tempo de processamento de uma determinada tarefa, economizar recursos ou simplesmente para ganhos de desempenho.

Neste trabalho abordamos o *offloading* computacional para ganhos de desempenho por meio da utilização oportunística dos recursos ociosos presentes nos veículos. Uma visão geral do trabalho pode ser apresentada na Figura 3, onde é possível observar que uma determinada aplicação que está sendo executada no veículo 1 elevou seu uso de CPU a um nível crítico, dessa forma o veículo 1 tenta realizar o *offloading* para desafogá-lo ou, simplesmente, diminuir o tempo de execução da tarefa.

Para a avaliação de desempenho do *offloading* em nosso cenário determinamos três métricas a serem avaliadas, são elas tempo de execução das tarefas que corresponde ao tempo necessário para completar uma determinada tarefa, uso de CPU onde avaliamos a utilização dos recursos computacionais pelas técnicas de execução local e *offloading* e, por fim, o (*Round Trip Time*), ou tempo de ida e volta, que é a quantidade de tempo transcorrido entre o momento em que o segmento é enviado e o momento em que é recebido um reconhecimento para o segmento,

Figura 3 – Técnica de *offloading* em VANETs.

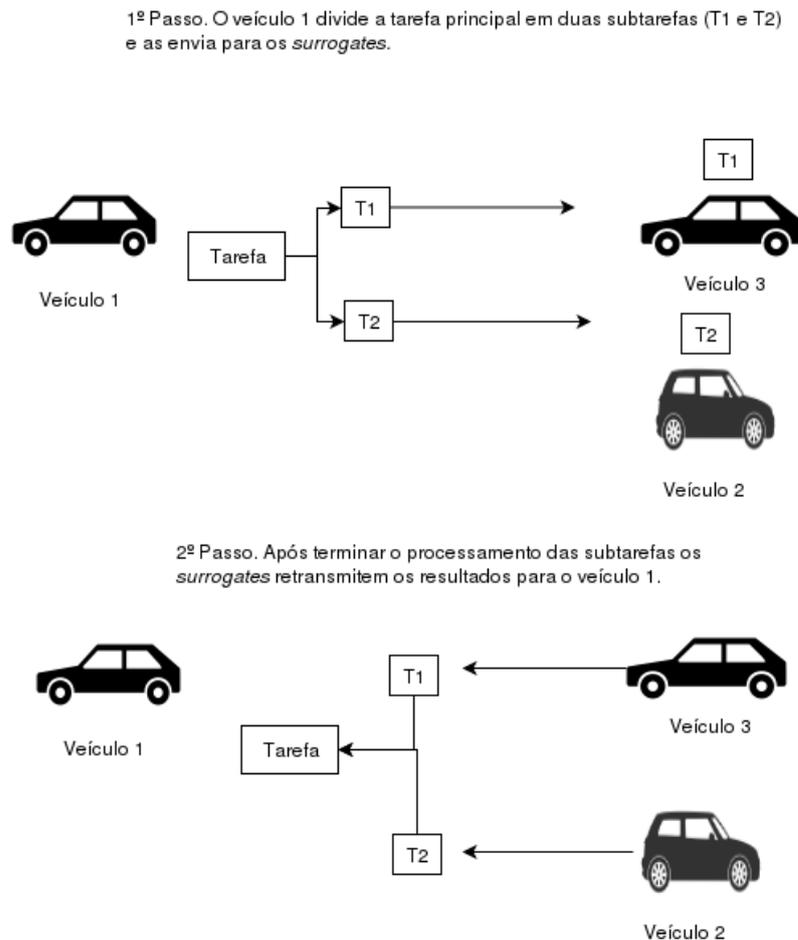


Fonte: Elaborada pelo autor.

Por meio de mensagens *Information REQuest* (IREQ) o cliente irá determinar quais *surrogates* fornecem serviços de processamento de dados a partir de suas mensagens *Information*

REPLY (IREP). Em seguida, o cliente entrará em uma acordo com os *surrogates* e, como mostrado na Figura 4, realizará o particionamento da tarefa resultando em subtarefas que serão enviadas aos *surrogates* escolhidos. Por fim, como mostrado no segundo passo representado na Figura 4, os *surrogates*, após terminarem a execução das subtarefas os *surrogates* retornam os resultados das subtarefas para o cliente que, por sua vez, realizará a união dos resultados.

Figura 4 – Divisão e envio da tarefa para os *surrogates*.



Fonte: Elaborada pelo autor.

5.1 Workload utilizado

Neste trabalho utilizamos uma abordagem semelhante ao trabalho de Shiraz et al. (2014), que utiliza como *workload* a multiplicação de matrizes quadradas, onde implementamos um algoritmo que, basicamente, divide as matrizes em blocos que serão enviados aos *surrogates*. A proposta do algoritmo foi baseada em um algoritmo bastante conhecido em álgebra linear que pode ser encontrado no trabalho de George e Liu (1978). Suponhamos que temos a seguinte

multiplicação de matrizes $N \times N$, onde N são as dimensões da matriz. Logo abaixo apresentamos a operação a ser realizada.

$$MatrizA = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \times MatrizB = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad (1)$$

Além disso, suponhamos que existam dois *surrogates* dispostos a realizar o processamento desta tarefa, então temos as seguintes operações a serem transferidas:

$$\begin{bmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \\ 13 & 14 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \quad (2)$$

A operação (2) será enviada para o primeiro *surrogate*, enquanto a operação (3) será enviada para o segundo *surrogate*.

$$\begin{bmatrix} 3 & 4 \\ 7 & 8 \\ 11 & 12 \\ 15 & 16 \end{bmatrix} \times \begin{bmatrix} 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad (3)$$

Como podemos observar, o particionamento das matrizes é realizado de acordo com o número de *surrogates* disponíveis. Portanto, os blocos resultantes terão as dimensões $N \times NS$ e $NS \times N$, onde N é a dimensão original da matriz e NS será igual a $N \div \text{numeroDeSurrogates}$, dessa forma obtemos uma divisão justa de tarefas.

Por fim, o produto dessas multiplicações resultará em duas novas matrizes com dimensões $N \times N$, que serão retornadas ao cliente que, por sua vez irá somá-los e obter o resultado (4).

$$MatrizA \times MatrizB = \begin{bmatrix} 90 & 100 & 110 & 120 \\ 202 & 228 & 254 & 280 \\ 314 & 356 & 398 & 440 \\ 426 & 484 & 542 & 600 \end{bmatrix} \quad (4)$$

5.2 Ambiente de testes

Para a elaboração deste trabalho foram utilizadas três máquinas, onde duas delas representam os *surrogates*, que encontram-se ociosos, e outra máquina de desempenho inferior representa o cliente sobrecarregado. As configurações de cada uma podem ser vistas na Tabela 2. Utilizamos um rede cabeada de 100Mbps nas medições.

Na elaboração dos *scripts*, tanto cliente quanto servidor, utilizamos a linguagem *python* e o módulo para aplicações científicas chamado *numpy*, que nos permite trabalhar com *arrays* bidimensionais de forma mais simplificada. Os *scripts* estão disponíveis no GitHub¹.

Assim como no trabalho de Wang e Li (2004), utilizamos uma técnica de compressão e decompressão de mensagens, neste caso de *arrays* bidimensionais, para reduzir o tamanho do pacote e auxiliar no envio das matrizes. Para realizar a compressão e decompressão dos *arrays*, utilizamos um método nativo do módulo *numpy* que, além disso, também reduz o tempo de serialização.

Tabela 2 – Ambiente de testes

Tipo	RAM	Processador	SO	Total de núcleos	Velocidade (GHz)
Cliente	4GB	intel celeron	Debian 3.16.36	2	2.5
<i>Surrogate</i>	8GB	intel i5	Ubuntu 14.04	4	3.1
<i>Surrogate</i>	8GB	intel i5	Ubuntu 14.04	4	3.1

Fonte: Elaborada pelo autor.

As medições foram realizadas no Laboratório de Redes de Computadores na Universidade Federal do Ceará - Campus Quixadá. Após executarmos os testes trinta vezes com matrizes de dimensões 1000×1000 , 2000×2000 e 5000×5000 , coletamos os dados de tempo de execução local e em seguida com *offloading*, uso de CPU no cliente e atraso de serialização. De acordo com a literatura, realizamos trinta execuções para termos uma boa amostragem do experimento, tornando possível coletar o intervalo de confiança e ter uma base de informações adequada.

Realizamos trinta repetições em cada cenário pois com este número de execuções obtemos uma amostra confiável para realizar a coleta dos dados, cálculo do intervalo de confiança e aferição.

Para o cálculo do intervalo de confiança dos experimentos utilizamos um nível de confiança de 95%. Para a coleta dos RTTs utilizamos o simulador de rede NS-3 (AL-SULTAN

¹ https://github.com/mateus-n00b/offload_scripts

et al., 2014). As informações das simulações estão presentes na Tabela 3.

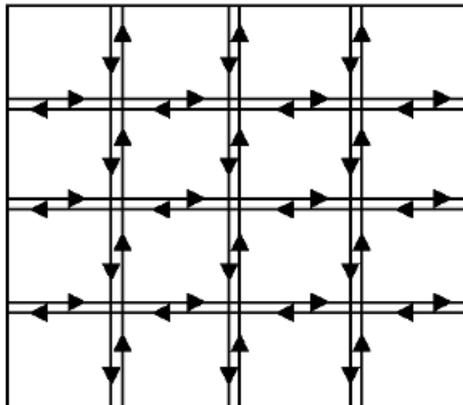
Tabela 3 – Ambiente de simulação

Modelo de perda de propagação	Nakagami
Número de nós	25
Modelo de atraso de propagação	ConstantSpeedPropagationDelayModel
Tempo de simulação	150 segundos
Modelo de mobilidade	Gipps
Cenário	Manhattan
Protocolo de transporte	TCP
Protocolo de enlace	802.11p
Alcance(m)	200

Fonte: Elaborada pelo autor.

Nas simulações utilizamos o modelo de mobilidade Gipps (GIPPS, 1981), onde os nós na simulação trafegam em velocidade máxima, mas esse valor pode mudar se houver possibilidade de colisão com outro veículo. Para o cenário, escolhemos o modelo Manhattan que simula um ambiente urbano em formato de malha que é composto de ruas verticais e horizontais e cada rua possui duas faixas nas direções norte e sul, como pode ser observado na Figura 5.

Figura 5 – Modelo Manhattan.



Fonte: (JAYAKUMAR; GOPINATH, 2008).

Escolhemos o NS-3 como ferramenta de simulação pois o mesmo oferece facilidades relacionadas a simulação de VANETs como, por exemplo, o protocolo 802.11p que é o protocolo padrão criado para VANETs. Além disso, podemos utilizar o *trace* gerado pelo SUMO², que é

² http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

um simulador de mobilidade urbana muito utilizado pela comunidade acadêmica, para gerar a mobilidade adequada a VANETs.

Na simulação havia vinte e cinco veículos, mas somente oito tinham a habilidade de comunicar-se e recursos computacionais disponíveis e dentre esses apenas dois forneciam serviço de processamento de dados que, por sua vez, foram requisitados por um veículo na VANET. Por fim, os demais veículos realizavam trocas de informações, a cada um segundo, simulando mensagens de sinalização.

Na simulação realizamos uma troca de pacotes de tamanhos iguais aos das matrizes utilizadas em nossas medições, onde o cliente envia um pacote para o *surrogate*, pacote este que seria parte da tarefa a ser executada, e em seguida o *surrogate* transmite ao cliente um pacote, que simula a resposta enviada ao cliente. Medimos os RTTs a partir destas trocas de mensagens.

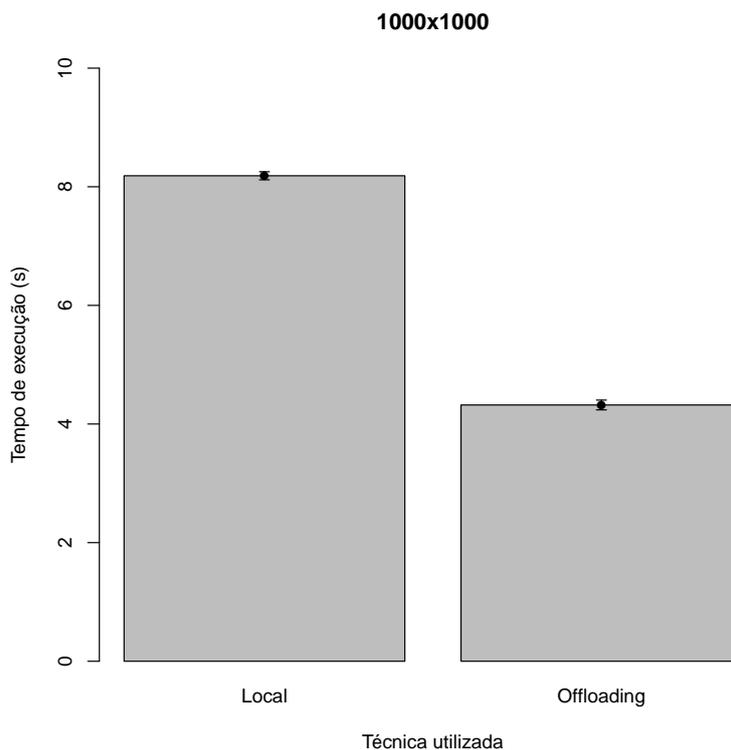
6 RESULTADOS

Nesta Seção apresentamos os resultados sintetizados a partir dos experimentos realizados. Foram realizados experimentos com multiplicações de matrizes com as dimensões 1000×1000 , 2000×2000 e 5000×5000 . Inicialmente, realizamos a multiplicação localmente, em uma máquina de baixo desempenho, logo em seguida realizamos o *offloading* com dois *surrogates* que estavam ociosos.

6.1 Tempo de execução

A seguir apresentamos as comparações dos tempos de execução utilizando as técnicas de execução local e execução com o *offloading*.

Figura 6 – Comparação de execução Local vs execução com Offloading (Matriz 1000x1000)



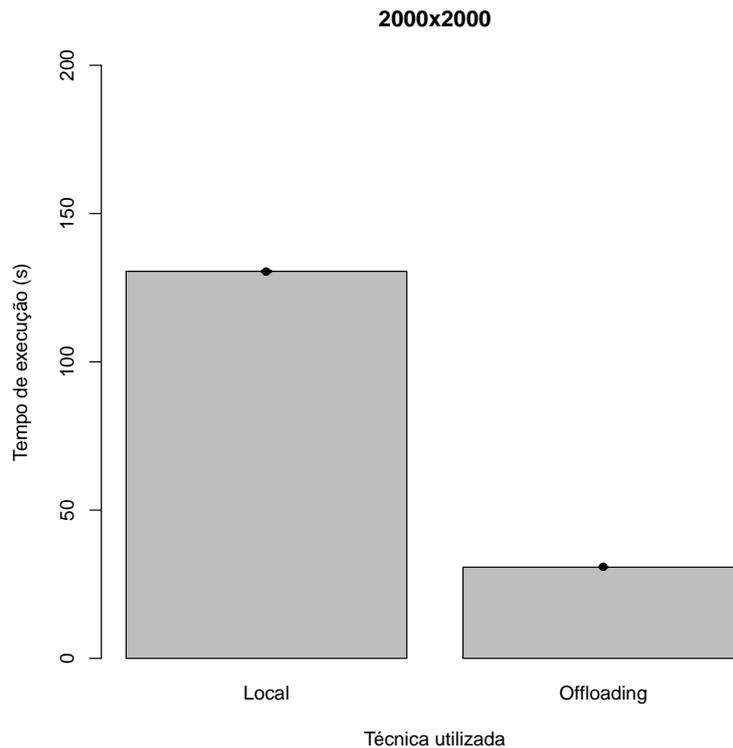
Fonte: Elaborada pelo autor.

Na Figura 6 apresentamos a comparação entre a execução local e a execução com o *offloading* computacional de uma multiplicação de matrizes 1000×1000 para um nível de confiança de 95%. Podemos notar que o tempo de processamento diminuiu cerca de 50% com

o *offloading* a média na técnica de processamento local está distribuída entre 8.252s e 8.118s enquanto na técnica de *offloading* a média gira em torno de 4.402s e 4.238s.

Podemos notar que os limites do intervalo de confiança em ambos os casos encontram-se próximos da média. Em VANETs, o tempo necessário para completar essa tarefa é aceitável.

Figura 7 – Comparação de execução Local vs execução com Offloading (Matriz 2000x2000)



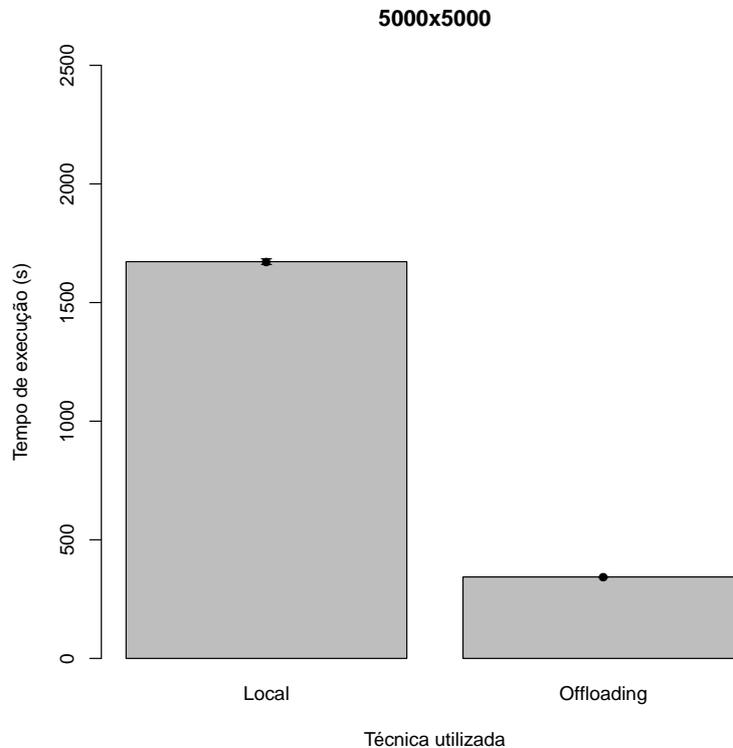
Fonte: Elaborada pelo autor.

Já na Figura 7 é apresentado uma comparação entre o tempo de processamento de uma multiplicação de matrizes 2000×2000 , onde aplicamos a técnica do *offloading* computacional que reduziu em cerca de um minuto o tempo necessário para completar a tarefa. Para os valores da média na execução local encontramos 130.862s e 130.186s, já na execução com o *offloading* a média está distribuída entre 30.849s e 30.649s. Podemos inferir que a diferença entre os intervalos da execução local é bastante expressiva se comparado aos limites encontrados na técnica de *offloading*.

Apesar da evidente redução, o tempo gasto nesse processamento é muito alto para um cenário onde a topologia muda constantemente e os nós, possivelmente, nunca mais se encontrarão novamente. Por outro lado, em um evento de engarrafamento esse *offloading* poderá

ser possível pois haverá muitos nós que possam receber tais tarefas, como afirma Li et al. (2014).

Figura 8 – Comparação de execução Local vs execução com Offloading (Matriz 5000x5000)



Fonte: Elaborada pelo autor.

Por fim, na Figura 8 nos é mostrado a comparação entre as técnicas de execução local e execução com o *offloading* computacional em uma multiplicação de matrizes 5000×5000 , onde podemos observar que a média encontrada utilizando a técnica de execução local está entre 1684.420s e 1660.380s, já a média encontrada ao utilizar o *offloading* está localizada entre 342.663s e 344.249s. É notável a diferença entre os intervalos na execução local se comparada ao *offloading*, isso ocorre devido a variação no nível de ocupação da CPU na máquina cliente, pois a mesma estava executando outras tarefas em paralelo.

No gráfico podemos notar que o tempo necessário para executar a tarefa localmente é muito alto, já com o *offloading* o tempo cai drasticamente. Isso se deve ao fato da máquina cliente ter recursos computacionais inferiores aos recursos dos *surrogates*, portanto é indicado a realização do *offloading* pois os atrasos de transmissão, propagação e RTTs não serão tão significativos no tempo total de processamento como notamos em tarefas anteriores. Em contrapartida, um veículo dificilmente manterá uma conexão por esse período com outro veículo,

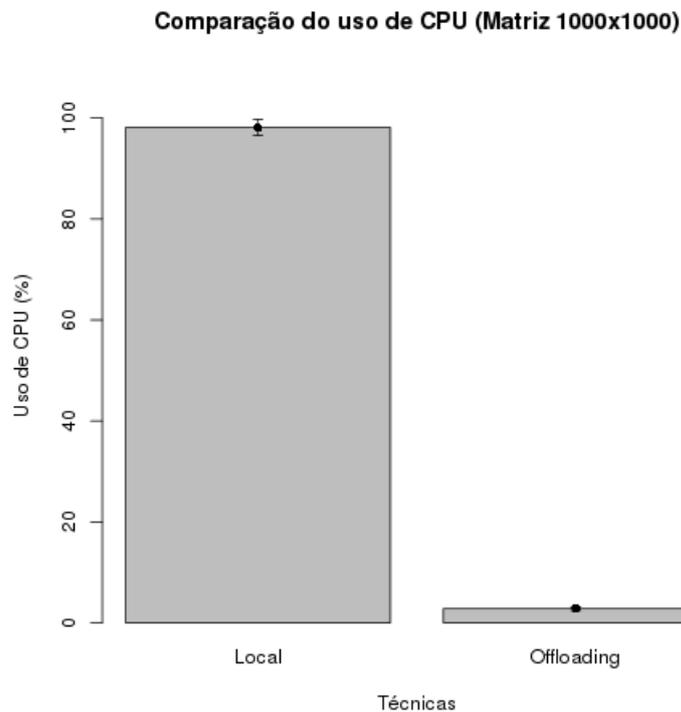
a não ser no caso de um comboio, como uma família viajando ou veículos seguindo a mesma direção.

6.2 Uso de CPU

Nesta Subsecção é apresentado as comparações relacionadas ao uso de CPU, onde foi observado qual técnica utiliza mais recursos computacionais da máquina cliente. Nestas medições avaliamos a quantidade de recursos de CPU utilizado somente pelo processo de *offloading* e da multiplicação de matrizes. Os *surrogates* encontravam-se ociosos, ou seja reservados apenas para executarem as tarefas provenientes do cliente.

Para a coleta destes dados foram realizadas trinta execuções, em cada técnica, e a utilização de um nível de confiança de 95% para a coleta do intervalo de confiança.

Figura 9 – Comparação do uso de CPU na execução Local vs execução com Offloading (Matriz 1000x1000)

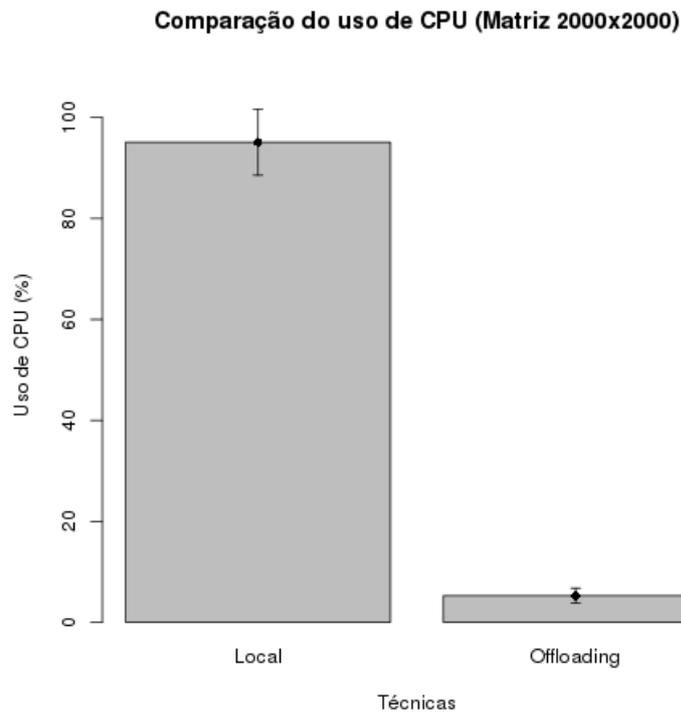


Fonte: Elaborada pelo autor.

Na Figura 9, notamos uma grande diferença se compararmos a execução local com a execução com o *offloading* na operação com a matriz 1000×1000 , onde a média na técnica

de execução local gira em torno de 95.31% e 100.92%. Enquanto na técnica de *offloading* computacional a média fica entre 0.61% e 1.50% de uso de CPU.

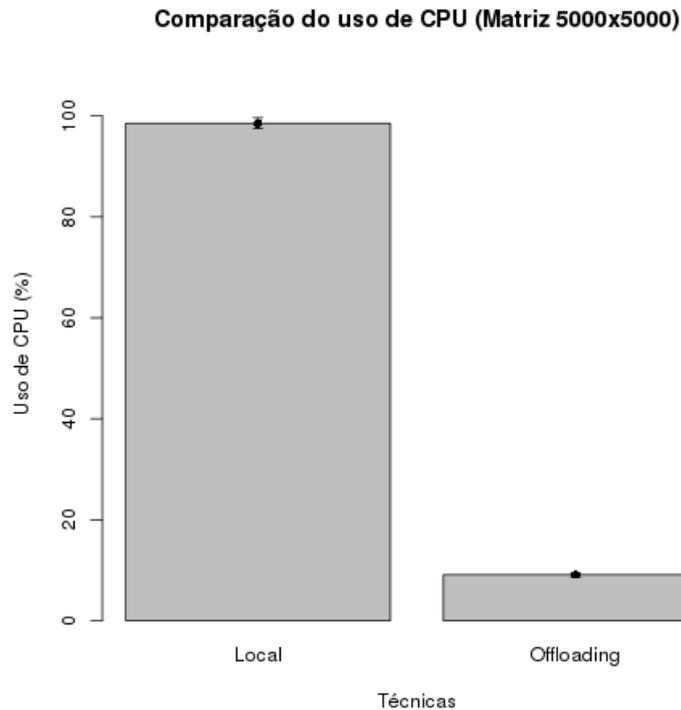
Figura 10 – Comparação do uso de CPU na execução Local vs execução com Offloading (Matriz 2000x2000)



Fonte: Elaborada pelo autor.

Na Figura 10 é apresentado a comparação do uso de CPU para a matriz 2000×2000 , onde a média para execução local está compreendida entre 89.85% e 100.37%, já na execução com o *offloading* a média se apresenta entre 5.06% e 8.05%. Pode-se notar que a técnica de *offloading* computacional reduziu o uso de CPU, entretanto se comparado a matriz 1000×1000 o uso de CPU teve um pequeno aumento. Isso ocorre devido ao aumento do tamanho da matriz que, por sua vez, necessita de mais recursos para serializá-la e enviá-la ao *surrogate*.

Figura 11 – Comparação do uso de CPU na execução Local vs execução com Offloading (Matriz 5000x5000)



Fonte: Elaborada pelo autor.

Por fim, na Figura 11 é apresentado a comparação da utilização da CPU na operação da multiplicação de uma matriz de dimensões 5000×5000 . Pode-se inferir, observando a Figura 11, que a técnica de *offloading* computacional requer menos recursos de CPU, pois as operações complexas são executadas externamente, enquanto a execução local eleva bastante seu uso. A média para a execução local gira em torno de 89.43% e 107.62%, enquanto a média da técnica de *offloading* está entre 1.18% e 1.97%.

Após a realização das medições fica evidente que no cenário abordado todas as operações com o *offloading* utilizam o mínimo de recursos na máquina cliente. Estes resultados ocorrem porque as operações de maior complexidade são migradas para os *surrogates*.

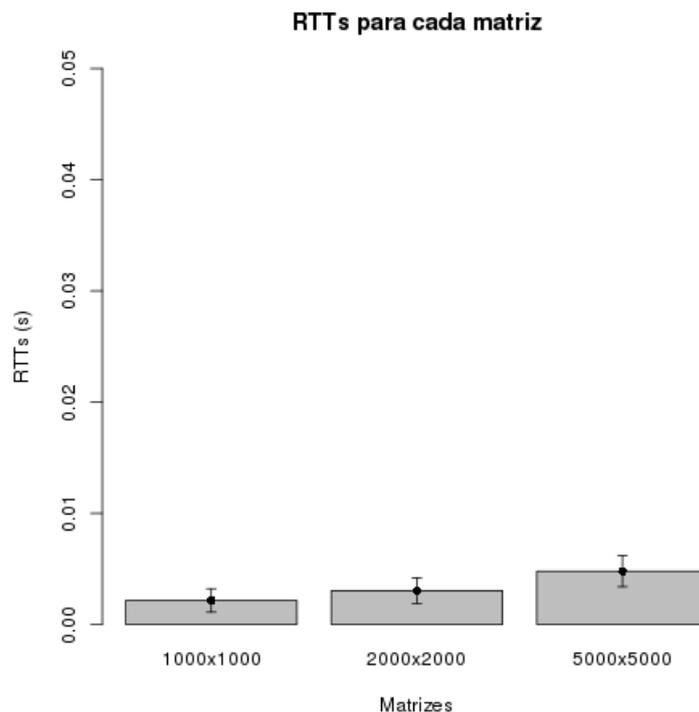
6.3 Round Trip Time

Nesta Subseção apresentamos os RTTs coletados a partir da simulação realizada no simulador de redes NS-3. Como dito anteriormente, a simulação é composta de vinte e um veículos movendo-se à uma velocidades aleatórias. Repetimos o experimento trinta vezes e

coletamos os RTTs a partir do *trace* gerado pelo NS-3. Novamente utilizamos um nível de confiança de 95%.

Para a coleta dos RTTs criamos um cenário igual ao das medições realizadas em laboratório, onde um veículo realiza o *offloading* de uma tarefa para dois veículos. Neste trabalho nos detemos apenas ao *offloading* sem realizar descoberta de serviços e escolha de *surrogate*. Para cada matriz modificamos o tamanho do pacote a ser enviado e o tamanho dos pacotes a serem enviados ao cliente com o objetivo de simular a execução realizada em laboratório.

Figura 12 – RTTs coletados na simulação no NS-3



Fonte: Elaborada pelo autor.

Na Figura 12 nos é apresentado os RTTs obtidos a partir das simulações realizadas, podemos inferir que na matriz 1000×1000 a média encontra-se entre 0.0011s e 0.0032s, na matriz 2000×2000 a média está entre 0.0018s e 0.0041s. Já na matriz 5000×5000 notamos o maior RTT, que está compreendido entre 0.0033s e 0.0061s. O RTT é maior na matriz 5000×5000 devido a quantidade de informações que são enviadas tanto para o *surrogate* quanto para cliente.

Após coletarmos esses resultados, realizamos a soma dos RTTs aos tempos de execuções com *offloading*, pois dessa forma podemos avaliar se o *offloading* consegue manter o

tempo de execução aceitável no ambiente de VANETs. Esses valores são apresentados na Tabela 4.

Tabela 4 – Tempo de execução somado ao RTT

Matriz	Tempo final
1000x1000	4.2391s
2000x2000	30.6531s
5000x5000	344.2461s

Fonte: Elaborada pelo autor.

Como podemos observar na Tabela 4, os tempos finais não sofreram grandes alterações após o acréscimo dos respectivos RTTs, sendo assim, o *offloading* permanece sendo a melhor opção para a realização da tarefa abordada em nosso ambiente de VANETs. É importante salientar que houveram muitas perdas e retransmissões no cenário 5000×5000 . Isso pode ser devido aos veículos passarem pouco tempo dentro dos alcances de comunicação uns dos outros, podendo os veículos até mesmo estarem em direções/sentidos opostas(os).

Importante mencionar os desafios inerentes ao protocolo TCP na troca de mensagens, uma vez que o mesmo precisa manter sessões sem rompimentos e apresenta atraso inicial devido ao estabelecimento da comunicação.

7 CONCLUSÃO

Após a realização dos experimentos em laboratório e, em seguida, a utilização da simulação, utilizando o simulador de redes NS-3 (AL-SULTAN et al., 2014), para simular uma VANET com nós trocando pacotes, que representavam as matrizes, medimos o RTT e inserimos no tempo de processamento final. Concluimos que o *offloading* em nosso cenário de VANET-Cloud resultou em alternativa para a execução de tarefas computacionalmente complexas, pois com o uso da técnica de *computation offloading* conseguimos reduzir o tempo de execução e uso de CPU na máquina cliente.

Para a realização do *offloading* em VANETs deve-se abordar alguns fatores críticos como capacidade computacional do *surrogate*, velocidade, tempo de permanência na área do cliente, segurança dos dados, algoritmos de escolha de *surrogate*, etc. Tais fatores implicam diretamente no processamento das tarefas e no tempo final de execução.

A comunicação entre veículos e a nuvem já existe como a tecnologia OnStar¹ presente em veículos da montadora Chevrolet. A comunicação entre veículos é utilizado por veículos da montadora BMW². Portanto, em um futuro próximo as montadoras de automóveis investirão cada vez mais na comunicação entre veículos tornando o *offloading* computacional possível de ser realizado ambientes reais.

Por fim, podemos inferir, a partir dos resultados dos experimentos, que o *offloading* computacional realizou a execução das tarefas de forma mais rápida se comparado a execução local. No cenário de VANETs as operações de 2000×2000 e 5000×5000 seriam mais desafiadoras devido ao tempo total de execução com o uso da técnica de *offloading* pois os veículos se afastam rapidamente uns dos outros. Entretanto, esse problema poderia ser contornado se os veículos seguissem na mesma direção, formando um comboio, ou em um engarrafamento, dessa forma, conexões seriam mantidas por um período maior de tempo. Além disso, observamos que a técnica de *offloading* computacional reduziu consideravelmente o uso de CPU na máquina cliente, dessa forma possibilitando que o essa máquina possa executar tarefas computacionalmente complexas sem possuir recursos de *hardware* avançados.

¹ <http://www.chevrolet.com.br/onstar.html>

² http://www.bmw.com/com/en/insights/technology/technology_guide/articles/cartocar_communication.html

8 TRABALHOS FUTUROS

Este trabalho se propôs a avaliar a técnica de *offloading* computacional em VANETs utilizando como *workload* uma multiplicação de matrizes quadradas com dimensões variadas.

Para avaliarmos a proposta implementamos um algoritmo de particionamento de matrizes e criamos uma arquitetura composta por um cliente representado por uma máquina com uma tarefa a ser executada, então realizamos o *offloading* para duas máquinas, com recursos computacionais superiores e ociosos, que representaram os *surrogates*.

Neste trabalho abordamos exclusivamente o *offloading*. Os próximos passos seriam a criação de uma espécie de *framework* com suporte ao *offloading* de aplicações presentes em VANETs como reconhecimento facial, rastreamento, identificação do comportamento do motorista, processamento de imagens, seleção de surrogate, protocolo de descoberta, etc.

REFERÊNCIAS

- AL-SULTAN, S.; AL-DOORI, M. M.; AL-BAYATTI, A. H.; ZEDAN, H. A comprehensive survey on vehicular ad hoc network. **Journal of network and computer applications**, Elsevier, v. 37, p. 380–392, 2014.
- BHOI, S. K.; KHILAR, P. M. Vehicular communication: a survey. **Networks, IET**, IET, v. 3, n. 3, p. 204–217, 2014.
- BITAM, S.; MELLOUK, A.; ZEADALLY, S. Vanet-cloud: a generic cloud computing model for vehicular ad hoc networks. **Wireless Communications, IEEE**, IEEE, v. 22, n. 1, p. 96–102, 2015.
- BRAVO-TORRES, J. F.; SAIANS-VAZQUEZ, J. V.; LOPEZ-NORES, M.; BLANCO-FERNANDEZ, Y.; PAZOS-ARIAS, J. J. Mobile data offloading in urban vanets on top of a virtualization layer. In: IEEE. **Wireless Communications and Mobile Computing Conference (IWCMC), 2015 International**. [S.l.], 2015. p. 291–296.
- CAMPOLO, C.; MOLINARO, A. On vehicle-to-roadside communications in 802.11 p/wave vanets. In: IEEE. **2011 IEEE Wireless Communications and Networking Conference**. [S.l.], 2011. p. 1010–1015.
- FAHIM, A.; MTIBAA, A.; HARRAS, K. A. Making the case for computational offloading in mobile device clouds. In: ACM. **Proceedings of the 19th annual international conference on Mobile computing & networking**. [S.l.], 2013. p. 203–205.
- FAROOQ, M. U.; PASHA, M.; KHAN, K. U. R. A data dissemination model for cloud enabled vanets using in-vehicular resources. In: IEEE. **Computing for Sustainable Global Development (INDIACom), 2014 International Conference on**. [S.l.], 2014. p. 458–462.
- FLEMING, B. Recent advancement in automotive radar systems [automotive electronics]. **Vehicular Technology Magazine, IEEE**, IEEE, v. 7, n. 1, p. 4–9, 2012.
- FUGGETTA, A.; PICCO, G. P.; VIGNA, G. Understanding code mobility. **IEEE Transactions on software engineering**, IEEE, v. 24, n. 5, p. 342–361, 1998.
- GEORGE, A.; LIU, J. W. Algorithms for matrix partitioning and the numerical solution of finite element systems. **SIAM Journal on Numerical Analysis**, SIAM, v. 15, n. 2, p. 297–327, 1978.
- GERLA, M. Vehicular cloud computing. In: IEEE. **Ad Hoc Networking Workshop (Med-Hoc-Net), 2012 The 11th Annual Mediterranean**. [S.l.], 2012. p. 152–155.
- GIPPS, P. G. A behavioural car-following model for computer simulation. **Transportation Research Part B: Methodological**, Elsevier, v. 15, n. 2, p. 105–111, 1981.
- GRUBITZSCH, P.; SCHUSTER, D. Hosting and discovery of distributed mobile services in an xmpp cloud. In: IEEE. **Mobile Services (MS), 2014 IEEE International Conference on**. [S.l.], 2014. p. 47–54.
- HUSSAIN, R.; SON, J.; EUN, H.; KIM, S.; OH, H. Rethinking vehicular communications: Merging vanet with cloud computing. In: IEEE. **Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on**. [S.l.], 2012. p. 606–609.

- JAYAKUMAR, G.; GOPINATH, G. Performance comparison of manet protocols based on manhattan grid mobility model. **Journal of Mobile communication**, v. 2, n. 1, p. 18–26, 2008.
- KHAN, M. A. A survey of computation offloading strategies for performance improvement of applications running on mobile devices. **Journal of Network and Computer Applications**, Elsevier, v. 56, p. 28–40, 2015.
- KOVACHEV, D. Framework for computation offloading in mobile cloud computing. **IJIMAI**, Universidad Internacional de La Rioja, v. 1, n. 7, p. 6–15, 2012.
- KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. **Mobile Networks and Applications**, Springer, v. 18, n. 1, p. 129–140, 2013.
- LI, B.; PEI, Y.; WU, H.; LIU, Z.; LIU, H. Computation offloading management for vehicular ad hoc cloud. In: **Algorithms and Architectures for Parallel Processing**. [S.l.]: Springer, 2014. p. 728–739.
- LI, F.; WANG, Y. Routing in vehicular ad hoc networks: A survey. **Vehicular Technology Magazine, IEEE**, IEEE, v. 2, n. 2, p. 12–22, 2007.
- LIN, Y.-D.; CHU, E. T.-H.; LAI, Y.-C.; HUANG, T.-J. Time-and-energy-aware computation offloading in handheld devices to coprocessors and clouds. **IEEE Systems Journal**, IEEE, v. 9, n. 2, p. 393–405, 2015.
- MELL, P.; GRANCE, T. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.
- MERSHAD, K.; ARTAIL, H. A framework for implementing mobile cloud services in vanets. In: IEEE. **2013 IEEE Sixth International Conference on Cloud Computing**. [S.l.], 2013. p. 83–90.
- POLASH, F.; ABUHUSSEIN, A.; SHIVA, S. A survey of cloud computing taxonomies: Rationale and overview. In: IEEE. **Internet Technology and Secured Transactions (ICITST), 2014 9th International Conference for**. [S.l.], 2014. p. 459–465.
- SHI, C.; AMMAR, M. H.; ZEGURA, E. W.; NAIK, M. Computing in cirrus clouds: the challenge of intermittent connectivity. In: ACM. **Proceedings of the first edition of the MCC workshop on Mobile cloud computing**. [S.l.], 2012. p. 23–28.
- SHIRAZ, M.; GANI, A.; AHMAD, R. W.; SHAH, S. A. A.; KARIM, A.; RAHMAN, Z. A. A lightweight distributed framework for computational offloading in mobile cloud computing. **PLoS one**, Public Library of Science, v. 9, n. 8, p. e102270, 2014.
- SINGH, S.; AGRAWAL, S. Vanet routing protocols: issues and challenges. In: IEEE. **Engineering and Computational Sciences (RAECS), 2014 Recent Advances in**. [S.l.], 2014. p. 1–5.
- SOUZA, A. B.; CELESTINO, J.; XAVIER, F. A.; OLIVEIRA, F. D.; PATEL, A.; LATIFI, M. Stable multicast trees based on ant colony optimization for vehicular ad hoc networks. In: IEEE. **The International Conference on Information Networking 2013 (ICOIN)**. [S.l.], 2013. p. 101–106.

WANG, C.; LI, Z. Parametric analysis for adaptive computation offloading. In: ACM. **ACM SIGPLAN Notices**. [S.l.], 2004. v. 39, n. 6, p. 119–130.

WHAIIDUZZAMAN, M.; SOOKHAK, M.; GANI, A.; BUYYYA, R. A survey on vehicular cloud computing. **Journal of Network and Computer Applications**, Elsevier, v. 40, p. 325–344, 2014.

YANG, L.; CAO, J.; TANG, S.; HAN, D.; SURI, N. Run time application repartitioning in dynamic mobile cloud environments. IEEE.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of internet services and applications**, Springer, v. 1, n. 1, p. 7–18, 2010.