Um Algoritmo de *Offloading* Computacional para Nuvens Veiculares

Antonio M. de Sousa*, Alisson B. de Souza*†, Paulo A. L. Rego*, José N. de Souza† e José A. F. de Macedo†

***Resumo*— As VANETs (Vehicular Ad Hoc Networks), ou Redes *Ad Hoc* Veiculares, visam fornecer serviços de segurança e conforto para os motoristas nas estradas e rodovias. Dentre esses serviços, existem tarefas de computação intensiva em aplicações de tempo-real que precisam ser executadas da forma mais rápida possível. Utilizando o paradigma de Nuvens Veiculares (ou VANET-*Cloud*), onde os veículos ofertam seus recursos computacionais como um serviço, e com o objetivo de reduzir o tempo de execução das tarefas, nós propomos um algoritmo de *Offloading* Computacional e avaliamos seu desempenho e viabilidade.**

***Palavras-Chave*— VANETs, *Offloading* Computacional, Nuvens Veiculares, Computação em Nuvem.**

***Abstract*— VANETs (Vehicular Ad Hoc Networks) aim to provide safety and comfort services for drivers on roads and highways. Among these services, there are computation-intensive tasks in real-time applications that need to be executed as quickly as possible. Using the Vehicular Cloud paradigm (or VANET-*Cloud*), where vehicles offer their computing resources as a service, and in order to further reduce the execution time of tasks, we propose a Computational *Offloading* algorithm and analyze its performance and feasibility.**

***Keywords*— VANETs, Computational Offloading, Vehicular Cloud, Cloud Computing.**

I. Introdução

Com o passar do tempo, veículos se tornaram mais sofisticados com capacidades computacionais e dispositivos de armazenamento a bordo, poderosos recursos computacionais, capacidades de comunicação e menores limitações de energia [1]. Dessa forma, os serviços fornecidos pelas chamadas VANETs (*Vehicular Ad hoc Networks*), Redes *Ad Hoc* 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, coleta e armazenamento de dados, dentre outros [2].

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 tanto formarem nuvens isoladas, menores, móveis e *ad hocs* como serem integrados com o ambiente de nuvem tradicional, que consiste apenas em entidades computacionais estacionárias [3].

Como diariamente muitos veículos gastam horas em engarrafamentos e rodovias, os veículos de agora em diante serão tratados como recursos computacionais subutilizados, que podem fornecer serviços públicos. Motoristas presos em engarrafamentos poderão aceitar doar seus recursos computacionais, ou exigir algo em troca, para ajudar entidades externas a executarem simulações complexas que demandam grande processamento [4]. Por exemplo, o reescalonamento de semáforos, posicionamento geográfico, sensores que monitoram a saúde do motorista, jogos cooperativos, reconhecimento facial, ou em casos de emergência (como desastres naturais ou evacuações não planejadas) [5].

Para os consumidores poderem acessar esses recursos computacionais disponíveis, eles precisam utilizar um protocolo de descoberta para buscar veículos que estejam aptos a realizar as tarefas solicitadas.

Uma vez que os recursos presentes nos veículos estejam acessíveis a terceiros, pode-se aplicar a técnica de *offloading* computacional, que consiste na divisão (particionamento) e envio de tarefas complexas (cargas de trabalho ou *workloads*) para serem executadas em outros dispositivos (também chamados de substitutos ou *surrogates*). Neste caso, os substitutos são veículos com recursos computacionais acessíveis e, possivelmente, ociosos ou subutilizados [6]. O *offloading* pode ser utilizado para ganhos de desempenho [7] ou para diminuir a carga de processamento sobre um dispositivo sobrecarregado, por meio da migração de partes de uma aplicação (ou a aplicação inteira) para uma máquina remota [8][9].

Neste contexto, este trabalho tem como objetivo apresentar um algoritmo de *offloading* computacional em VANETs que melhore o desempenho de execução de tarefas computacionalmente complexas, aproveitando os recursos computacionais disponíveis de forma oportunística. Logo em seguida, é possível avaliar o desempenho e a viabilidade do algoritmo proposto em VANETs.

O restante deste trabalho está organizado da seguinte forma: a Seção II apresenta os trabalhos relacionados, já a Seção III apresenta a solução proposta. Na Seção IV, os experimentos e a análise dos resultados são apresentados, e por fim, a Seção V apresenta as conclusões obtidas com a realização deste trabalho.

II. Trabalhos Relacionados

Em [4], os autores 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 podem alugar seus

*Universidade Federal do Ceará, Quixadá-CE, Brasil. E-mails: mateus-sousa@alu.ufc.br, alisson@ufc.br, pauloalr@ufc.br.

†Universidade Federal do Ceará, Fortaleza-CE, Brasil. E-mails: neuman@ufc.br, jose.macedo@lia.ufc.br.

recursos 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. O nosso trabalho, porém, foca apenas no *offloading* entre veículos, deixando o *offloading* entre veículos e RSUs para trabalhos futuros.

No trabalho [6], um *framework* foi desenvolvido para suportar o *offloading* em VANET-Cloud. Os autores assumiram que os veículos estão distribuídos e que se movem em um plano bidimensional, e o *offloading* foi implementado de duas formas distintas: *offloading* entre veículos e *offloading* entre veículos e RSUs. No entanto, os autores não detalharam o *workload* e o algoritmo de particionamento utilizados. Os autores utilizaram inundação da rede na busca pelos veículos substitutos e implementaram seu próprio ambiente de simulação, usando a linguagem Java. Em nossa proposta, ao invés de usar inundação, a área de atuação foi limitada a um salto para evitar uma possível sobrecarga na rede e facilitar a operação de envio das subtarefas, uma vez que os substitutos estarão próximos o suficiente do cliente. Além disso, detalhamos o *workload* e o algoritmo utilizado.

No trabalho de [10], os autores propuseram um *framework* para tomada de decisão de *offloading*, cujo objetivo é reduzir o tempo de resposta e consumo de energia dos dispositivos móveis. Os experimentos foram executados em diversos modelos de aparelhos celulares e foi realizado um estudo de caso utilizando multiplicação de matrizes como *workload*. Finalmente, os resultados mostraram que o *framework* conseguiu reduzir em torno de 75% do tempo de execução das tarefas e reduzir em cerca de 56% o uso de bateria do dispositivo utilizando o *offloading*. Em nosso trabalho, ao invés de usarmos aparelhos celulares, utilizamos o *offloading* para melhorar o desempenho da execução de tarefas complexas em VANETs.

Neste trabalho, escolhemos a multiplicação de matrizes como *workload*. Mas, diferente de [10], utilizamos matrizes de dimensões 1000×1000 e 2000×2000 que apresentam uma complexidade computacional maior. Além disso, para análise de desempenho e viabilidade utilizamos o simulador de redes NS-3 e emulação em máquinas físicas. Por fim, por ser uma área relativamente nova, não existem muitos trabalhos na literatura e, para o melhor de nosso conhecimento, somos os primeiros a apresentar detalhadamente um algoritmo de escolha e o tipo de *workload* utilizado.

III. Solução Proposta

Aplicações de tempo-real precisam ser executadas rapidamente. Por mais que as VANETs possuam cada vez mais poder computacional, às vezes o tempo de processamento dessas tarefas não é baixo o suficientes, por, dentre outros fatores, o veículo já estar com o processamento sobrecarregado. Nesse contexto, esta seção apresenta um algoritmo de *offloading* computacional capaz de diminuir esses tempos de processamento e execução. Para isso, utilizamos o conceito de Nuvens Veiculares ao aproveitarmos os processadores de veículos vizinhos que ofertam seus recursos computacionais como um serviço e que tenham esses recursos ociosos, parcialmente ociosos ou subutilizados.

O algoritmo de *offloading* computacional é apresentado no Algoritmo 1 abaixo.

Algoritmo 1: *Offloading* computacional em VANETs

```
1  Função Solicitação():
2      para cada veículo ∈ alcance faça
3          enviar solicitação
4      fim
5  Retorne
6  Função Resposta():
7      se serviço de offloading estiver em execução então
8          enviar resposta ao solicitante
9      fim
10 Retorne
11 Função EnvioDeOffloading(int nsd):
12     nse = 0
13     para cada resposta recebida faça
14         se tve > (tpw + α) então
15             adicionar veículo à lista de substitutos
16             nse = nse + 1
17         fim
18     fim
19     se nse >= nsd então
20         enviar o workload para nsd veículos da lista de substitutos
21     fim
22 Retorne
23 Função RetornaResultado():
24     retorna o resultado do offloading para o solicitante
25 Retorne
```

Quando o veículo cliente (**i** na Figura 1) precisa executar uma tarefa, ele tem a opção de fazer *offloading*. Para isso, o primeiro passo é a descoberta de substitutos que possam processar partes da tarefa.

As duas primeiras funções (`Solicitação` e `Resposta`) do Algoritmo 1 fazem parte do protocolo de descoberta. Conforme a função `Solicitação` e a Figura 1(a), o veículo cliente envia uma mensagem de solicitação em *broadcast* de um salto para todos os veículos que estão no seu alcance de comunicação, uma vez que o veículo **i** busca veículos que estejam oferecendo um serviço de processamento de dados e que estejam com CPU livre ou parcialmente livre.

Em seguida, conforme a função `Resposta` e Figura 1(b), os veículos que ofertam o serviço de *offloading* e que estejam com CPU disponível retornam com uma mensagem de resposta informando sua posição, velocidade e direção. Posteriormente, conforme função `EnvioDeOffloading` e Figura 1(c), o veículo **i** analisa as respostas recebidas e verifica se $tve > (tpw + \alpha)$, onde tve é o tempo de vida estimado do enlace, tpw é o tempo de processamento do *workload*, nse é o número de substitutos escolhidos e nsd é o número de substitutos desejados. O tve é calculado de acordo com os trabalhos [11] e [12].

O tpw é continuamente atualizado e armazenado em tabelas nos veículos para cada tipo de *workload*, de modo que usamos um histórico para obtê-lo. Dessa forma, é possível identificar se o *workload* conseguirá ser processado pelos substitutos e retornado ao veículo **i** antes que os veículos percam a comunicação. Após esse cálculo, é criada uma lista com os veículos que podem ser substitutos. Em seguida, o algoritmo verifica se temos quantidade suficiente de substitutos ($nse >= nsd$). Em caso positivo, o *workload* é distribuído e enviado para os veículos substitutos iniciarem o processamento da

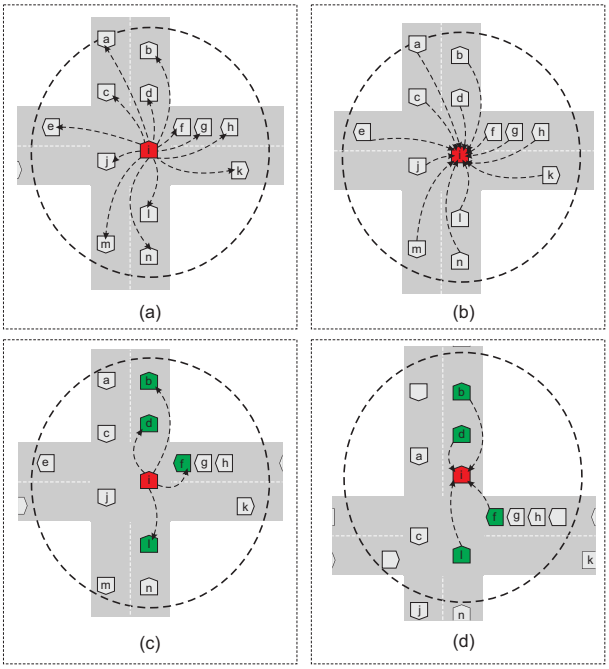


Fig. 1. *Offloading* computacional em Nuvens Veiculares: (a) O veículo **i** envia uma mensagem de solicitação em *broadcast*; (b) Todos os veículos que estão no alcance do veículo **i** respondem; (c) O veículo **i** envia partes do *workload* para o 4 substitutos escolhidos; (d) Os 4 substitutos escolhidos retornam os resultados para o veículo **i**.

tarefa. Por fim, conforme função `RetornaResultado` e Figura 1(d), os veículos substitutos escolhidos retornam os resultados para o veículo **i**. É importante ressaltar que se não houver substitutos suficientes ($nse < nsd$), o *offloading* não é realizado.

IV. Experimentação e Análise de Resultados

Para validar a solução proposta, utilizamos cenários simulados em que a tarefa (multiplicação de matrizes) foi distribuída de um veículo cliente para veículos substitutos. As subseções seguintes apresentam mais detalhes do experimento realizado.

A. Simulação

Dois cenários veiculares foram utilizados: (1) cenário Manhattan, que é uma área urbana de 1.6 x 1.5 quilômetros, feito de *grids*, com duas avenidas horizontais e quatro avenidas verticais e cada avenida contendo duas faixas e tráfego veicular fluindo nos dois sentidos; (2) cenário de rodovia (*Highway*), que é uma pista de 1 quilômetro de comprimento com duas faixas nas duas direções e quando um veículo chega ao final da pista, ele retorna pela direção contrária.

As simulações foram realizadas com o simulador NS-3 (versão 3.25) [13] executando em uma máquina com processador Intel *Core i5*, 4 GB de memória RAM e sistema operacional Debian 8. Alguns parâmetros da simulação são apresentados na Tabela I.

É importante destacar que não foi possível implementar o processamento da carga de trabalho no NS-3. Por isso, nós utilizamos emulação para calcular o tempo de processamento

TABELA I
Parâmetros de simulação do NS-3

Modelo de rádio propagação	Nakagami [14]
Número de nós	25, 50, 75 e 100
Modelo de atraso de propagação	ConstantSpeedPropagationDelay
Tempo de simulação	150 segundos
Modelo de mobilidade	Gipps
Cenário	Manhattan e Highway
Protocolo de transporte	TCP
Protocolo utilizado na descoberta	UDP
Alcance(m)	200
Protocolo de Transporte	UDP (descoberta) e TCP
Protocolo de Enlace e Física	802.11p
Pacote Contendo a Matriz	256B
Tempo de Vida do Enlace	Cálculo baseado em [11] e [12]
Workload Utilizado	Multiplicação de Matrizes [15] [16]
Veículos com CPUs disponíveis	Todos menos o cliente

do *workload*, tamanho do pacotes de requisição e resposta do *offloading*. Os valores resultantes da emulação foram então utilizados na simulação.

B. Emulação

A emulação foi utilizada para a coleta do tempo total de processamento das multiplicações das matrizes, particionamento das tarefas e obtenção do tamanho dos pacotes contendo as subtarefas. Ao término da emulação, os dados foram coletados e os atrasos inerentes à execução em laboratório foram devidamente retirados do tempo final do processamento, de modo a não interferir nas simulações.

Os tempos finais de processamento foram obtidos da emulação utilizando 2, 4 e 6 substitutos, que eram máquinas ociosas, que estavam esperando para executar as subtarefas. Os experimentos foram executados 30 vezes, tanto no cenário com *offloading* quanto no cenário de processamento local. Após as execuções, foi calculada a média do tempo de execução, com um nível de confiança de 95%, para multiplicações com matrizes com as seguintes dimensões: 1000×1000 e 2000×2000.

Os experimentos foram realizados no Laboratório de Redes da Universidade Federal do Ceará - Campus Quixadá, onde foram utilizadas máquinas do laboratório (cada uma com 8 GB de memória RAM e processador Intel *Core i5*), que atuaram como servidores para processar partes do *workload*, e um *notebook* de desempenho inferior (4 GB de RAM e processador *Core i5*), representando o cliente que faria o particionamento da tarefa e enviaria os *workloads* para os servidores.

C. Análise dos Resultados

Nesta seção, foi avaliado o desempenho do algoritmo proposto nos dois cenários descritos anteriormente: urbano e rodovia.

1) Tempo total de realização do offloading: Consiste no tempo transcorrido entre a descoberta de substitutos (solicitação e resposta - respectivamente, Figuras 1(a) e 1(b)), envio do *workload* (Figura 1(c)), tempo para processar o *workload* e o recebimento dos resultados (Figura 1(d)).

Nós calculamos o tempo de execução local da tarefa, bem como o tempo de execução com *offloading*, variando o número de veículos no cenário e o número de subtarefas (i.e., número necessário de substitutos). Além disso, comparamos o algoritmo proposto com um algoritmo que faz seleção dos substitutos de forma aleatória.

O tempo de execução do *workload* localmente (sem *offloading*) foi obtido através de emulação e é apresentado na Tabela II, onde $\overline{x}$ representa a média e $\pm$ representa o intervalo de confiança.

TABELA II
TEMPO TOTAL DE EXECUÇÃO SEM FAZER OFFLOADING

Matrizes	1000×1000		2000×2000	
	$\overline{x}$	$\pm$	$\overline{x}$	$\pm$
Tempo(s)	8.185433	0.1417	130.5244	0.7066

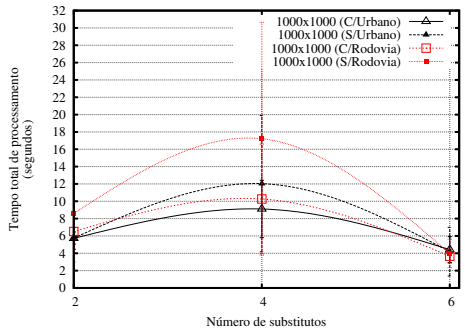


Fig. 2. Tempo total de processamento do *offloading*: matriz 1000×1000.

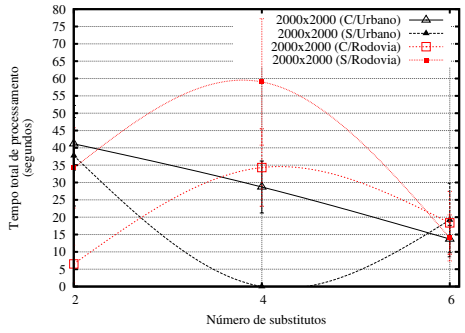


Fig. 3. Tempo total de processamento do *ofloading*: matriz 2000×2000.

As Figuras 2 e 3 apresentam o tempo total de realização do *offloading* desde a fase de descoberta de substitutos até o recebimento da última tarefa processada. Em tais Figuras, o "C" representa os tempos com o algoritmo proposto e o "S" representa os tempos sem o algoritmo proposto (aleatório).

Na Figura 2 são apresentados os tempos relacionados à realização do *offloading* com matrizes 1000×1000 em cenários urbanos e de rodovias. No *offloading* da matrizes 1000×1000 o algoritmo, executado com dois e seis substitutos, conseguiu reduzir o tempo de processamento se comparado à execução local (Tabela II), onde os tempos finais foram menores que os tempos de execução local. No caso anterior o menor tempo de execução foi obtido no ambiente urbano até seis substitutos, onde foi superado pela execução no cenário de rodovia. Entretanto, com quatro substitutos, o *offloading* não conseguiu reduzir o tempo total de processamento da tarefa. Os cenários com 2 substitutos em rodovias sem o algoritmo proposto também não tiveram tempos menores do que na execução local. Entretanto, nos demais cenários o *offloading* com a matriz 1000×1000 conseguiu reduzir o tempo total do processamento.

Já na Figura 3, a multiplicação de matrizes 2000×2000 em cenários urbanos e de rodovias foi executada mais rápido para todos os números de nós, quando comparada à execução local (Tabela II). Tal resultado pode ser explicado pelo fato da tarefa ser computacionalmente complexa, o que requer mais tempo e recursos.

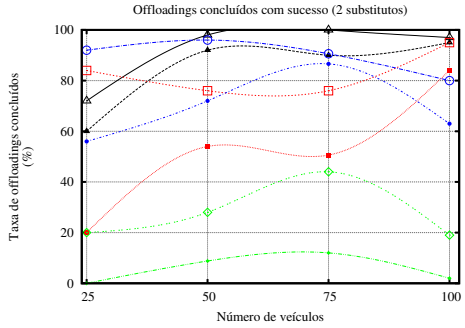
Portanto, o *offloading* pode ser utilizado para ganhos de desempenho no cliente em tarefas mais complexas, como na maioria dos casos de matrizes 1000×1000 e em todos os casos de matrizes 2000×2000.

2) Taxa de offloading concluídos com sucesso: Avaliamos se o algoritmo proposto consegue garantir uma boa quantidade *offloadings* realizados com sucesso no ambiente de VANETs e também se nosso algoritmo de *offloading* é mais eficiente do que fazer *offloading* de forma aleatória. Ressaltamos que para um *offloading* ser realizado com sucesso, precisamos necessariamente encontrar a quantidade de substitutos desejados e os substitutos escolhidos devem conseguir retornar os resultados de seus processamentos.

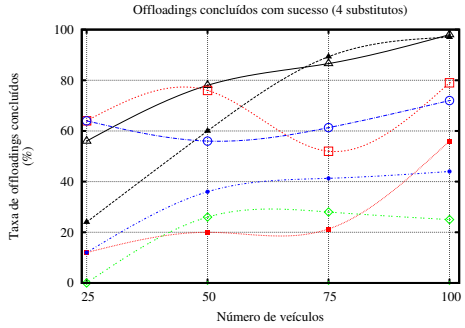
Nas Figuras 4(a), 4(b) e 4(c) (legenda na Figura 4(d)) são apresentadas as taxas de *offloadings* concluídos com sucesso com dois, quatro e seis substitutos. No caso da matriz 1000×1000, notamos que as taxas de sucesso com o algoritmo proposto (C - Rodovia e Urbano - Figura 4(d)), na maioria dos cenários apresentados, são maiores se comparadas as do método aleatório (S - Rodovia e Urbano - Figura 4(d)).

Com a matriz 2000×2000, o algoritmo proposto foi melhor em todos os cenários se comparado ao método de escolha aleatório. Entretanto, notamos que nas Figuras 4(c) e 4(b) com 25 nós e em cenário urbano, para matrizes 2000×2000, não foi possível completar nenhum *offloading* usando ambos os algoritmos (C e S), devido ao cenário ser muito esparso (veículos distantes uns dos outros). Mas à medida que o número de veículos aumenta a taxa de sucesso com o algoritmo proposto tende a crescer consideravelmente. Finalmente, no caso da matriz 2000×2000, em cenário de rodovia, observamos que o algoritmo obteve um número maior de *offloadings* concluídos se comparado ao ambiente urbano. Em contrapartida o algoritmo de escolha aleatório (S) no cenário com 25 veículos e 6 substitutos, Figura 4(c), não conseguiu completar nenhum *offloading* devido ao cenário ser muito esparso. O algoritmo alcançou as maiores taxas de sucesso no caso representado na Figura 4(c), um ambiente de rodovia com 6 substitutos utilizando a matriz 2000×2000, onde a taxa de sucesso alcançou 95% com 100 veículos além de um tempo total de processamento de cerca de 20 segundos.

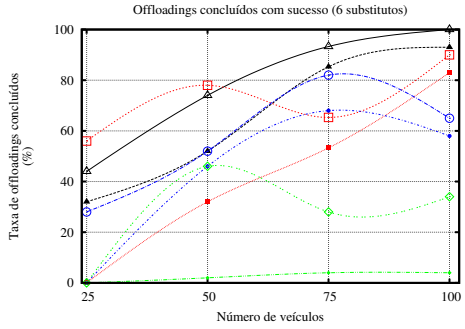
Com os resultados obtidos podemos inferir que o algoritmo proposto consegue escolher os melhores substitutos se comparado ao método de escolha aleatório. Um ponto importante do



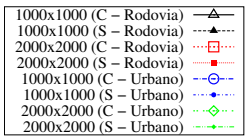
(a) *Offloading* com dois substitutos



(b) *Offloading* com quatro substitutos



(c) *Offloading* com seis substitutos



(d) Legenda

Fig. 4. Taxa de *offloadings* concluídos com sucesso.

comportamento do algoritmo que devemos observar é o fato de que quando o número de veículos aumenta no ambiente de rodovia, diferente do ambiente urbano, a taxa de sucesso dos *offloadings* tende a aumentar. Esse comportamento acontece devido à menor complexidade presente no ambiente de rodovia onde os veículos movem-se em um ambiente mais previsível.

V. Conclusão e Trabalhos Futuros

Neste trabalho apresentamos a análise de um algoritmo de *offloading* computacional em cenários de nuvens veiculares e comparamos com um algoritmo de *offloading* aleatório e com a execução local de uma tarefa. O objetivo é usar o *offloading* computacional em VANETs com a intenção de melhorar o desempenho de aplicações com uso intensivo de processamento e aproveitar os recursos computacionais ociosos presentes nos veículos no fornecimento de serviços de processamento de dados. Em nossa análise apresentada anteriormente avaliamos que a realização *offloading* mostrou-se mais eficiente do que a execução local para tarefas mais complexas. Entretanto, precisamos melhorar a taxa de sucesso desses *offloadings* por diminuir a restrição de só considerar sucesso se necessariamente encontrarmos a quantidade de substitutos desejados. Também iremos variar outros parâmetros nos trabalhos futuros.

Portanto, apesar do cenário de VANETs parecer hostil a essas aplicações devido à sua dinamicidade, os veículos em movimento ou em engarrafamentos, possuem recursos subutilizados, que podem ser aproveitados através do *offloading* computacional para melhorar o desempenho de aplicações.

Referências

[1] B. Fleming, "Recent advancement in automotive radar systems [automotive electronics]," *Vehicular Technology Magazine, IEEE*, vol. 7, no. 1, pp. 4–9, 2012.

[2] M. U. Farooq, M. Pasha, and K. U. R. Khan, "A data dissemination model for cloud enabled vanets using in-vehicular resources," in *Computing for Sustainable Global Development (INDIACom), 2014 International Conference on*. IEEE, 2014, pp. 458–462.

[3] S. Bitam, A. Mellouk, and S. Zeadally, "Vanet-cloud: a generic cloud computing model for vehicular ad hoc networks," *Wireless Communications, IEEE*, vol. 22, no. 1, pp. 96–102, 2015.

[4] K. Mershad and H. Artail, "A framework for implementing mobile cloud services in vanets," in *2013 IEEE Sixth International Conference on Cloud Computing*. IEEE, 2013, pp. 83–90.

[5] M. Whaiduzzaman, M. Sookhak, A. Gani, and R. Buyya, "A survey on vehicular cloud computing," *Journal of Network and Computer Applications*, vol. 40, pp. 325–344, 2014.

[6] B. Li, Y. Pei, H. Wu, Z. Liu, and H. Liu, "Computation offloading management for vehicular ad hoc cloud," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2014, pp. 728–739.

[7] P. A. Rego, P. B. Costa, E. F. Coutinho, L. S. Rocha, F. A. Trinta, and J. N. de Souza, "Performing computation offloading on multiple platforms," *Computer Communications*, 2016.

[8] L. Yang, J. Cao, S. Tang, D. Han, and N. Suri, "Run time application repartitioning in dynamic mobile cloud environments," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp. 336–348, July 2016.

[9] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[10] Y.-D. Lin, E. T.-H. Chu, Y.-C. Lai, and T.-J. Huang, "Time-and-energy-aware computation offloading in handheld devices to coprocessors and clouds," *IEEE Systems Journal*, vol. 9, no. 2, pp. 393–405, 2015.

[11] A. B. Souza, J. Celestino, F. A. Xavier, F. D. Oliveira, A. Patel, and M. Latifi, "Stable multicast trees based on ant colony optimization for vehicular ad hoc networks," in *The International Conference on Information Networking 2013 (ICOIN)*. IEEE, 2013, pp. 101–106.

[12] J. Härri, C. Bonnet, and F. Filali, "Kinetic mobility management applied to vehicular ad hoc network protocols," *Computer Communications*, vol. 31, no. 12, pp. 2907–2924, 2008.

[13] S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti, and H. Zedan, "A comprehensive survey on vehicular ad hoc network," *Journal of network and computer applications*, vol. 37, pp. 380–392, 2014.

[14] W. Galván and A. Fonseca, "O impacto dos modelos de propagação nos protocolos de roteamento para redes veiculares urbanas," *XXXIV Simpósio Brasileiro de Telecomunicações-SBrT*, pp. 1–5, 2016.

[15] M. Shiraz, A. Gani, R. W. Ahmad, S. A. A. Shah, A. Karim, and Z. A. Rahman, "A lightweight distributed framework for computational offloading in mobile cloud computing," *PloS one*, vol. 9, no. 8, p. e102270, 2014.

[16] A. George and J. W. Liu, "Algorithms for matrix partitioning and the numerical solution of finite element systems," *SIAM Journal on Numerical Analysis*, vol. 15, no. 2, pp. 297–327, 1978.