Algoritmo de Decisão para Offloading Computacional em Vehicular Fog Computing com Pedestres

Paulo H. G. Rocha[1], **Alisson B. de Souza**[2], **Francisco A. Silva**[3] **e Paulo A. L. Rego**[1]

[1]Departamento de Computação – Universidade Federal do Ceará (UFC)
Fortaleza – CE – Brasil

[2]Campus Quixadá – Universidade Federal do Ceará (UFC) – Quixadá – CE – Brasil

[3]Universidade Federal de Piauí (UFPI) – Picos – PI – Brasil

paulorocha@great.ufc.br, alisson@ufc.br

faps@ufpi.edu.br, paulo@dc.ufc.br

Abstract. *With a more significant number of devices integrated and connected in a VANET (Vehicular Ad Hoc Networks), there is greater availability and variety of computing resources to run applications. In VANETs, VFC (Vehicular Fog Computing) technology establishes vehicles, edge, and cloud as resource-providing infrastructures. However, the use of VFC as infrastructure for pedestrians is still limited, with few works addressing computational offloading in such a scenario. In this context, we implemented a decision algorithm for the offloading process based on resources provided by VFC to guarantee better offloading and latency rates. The results showed that the implemented algorithm obtained efficiency rates above 90% in the tested scenarios and a reduction of up to 40% in the offloading execution time compared to a random approach tested.*

Resumo. *Com um maior número de dispositivos integrados e conectados em uma VANET (Veicular Ad Hoc Networks), há uma maior disponibilidade e variedade de recursos computacionais para executar aplicações. No campo das redes veiculares, a tecnologia VFC (Vehicular Fog Computing) estabelece veículos, borda e nuvem como infraestruturas provedoras de recursos. No entanto, o uso do VFC como infraestrutura para pedestres ainda é limitado, com poucos trabalhos abordando o offloading computacional nesse cenário. Nesse contexto, um algoritmo de decisão para o processo de offloading foi implementado com base em recursos disponibilizados pela VFC, de modo a garantir melhores taxas de offloading e latência. Os resultados demonstraram que o algoritmo proposto obteve taxas de eficiência na escolha acima de 90% nos cenários testados, além de uma redução de até 40% no tempo de execução do offloading quando comparado à abordagem aleatória testada.*

1. Introdução

Pedestres, ciclistas e motociclistas são chamados VRUs (do inglês, usuários vulneráveis da estrada). A comunicação V2X (Veículo-para-Tudo) engloba a comunicação entre veículos e todas as categorias de VRUs. Ao habilitar V2X para VRUs, eles podem se tornar parte ativa do ITS (do inglês, Sistema de Transporte Inteligente) e possibilitar várias

aplicações de ITS nos campos de segurança, prevenção de acidentes, comunicação, dentre outras [Sewalkar and Seitz 2019]. Trabalhos recentes buscam desenvolver o conceito de P2V (Pedestre-para-Veículo) e P2X (Pedestre-para-Tudo), semelhantes ao V2X, porém com o pedestre atuando como origem ou cliente de serviços [Yoo et al. 2021].

Na comunicação P2X/V2X, os pedestres usam um *smartphone* para estabelecer comunicação por rede *Wi-Fi* ou via rede celular (por exemplo, LTE ou 5G) para se comunicarem com diversos componentes, como a nuvem, servidores de borda e veículos [Khan et al. 2022]. No caso da comunicação celular, os usuários participam de uma rede maior através de um sistema de comunicação móvel. Seguindo a evolução atual de LTE para 5G, a tecnologia traz muitas vantagens para aplicações de segurança em VRU — e também é considerada para comunicação V2X [Chen et al. 2017].

No entanto, um dos maiores problemas enfrentados pelos *smartphones* é a limitação de recursos. Quanto mais informações de contexto e algoritmos complexos são usados, maior é a carga de computação nos dispositivos [Nguyen et al. 2019]. Para lidar com tal problema nos dispositivos móveis, a técnica de *offloading* computacional tem sido utilizada para facilitar a execução de aplicações complexas computacionalmente, através da distribuição de partes da aplicação a outros dispositivos ou infraestruturas, sendo bastante empregado em comunicações veiculares [De Souza et al. 2020].

VFC emprega veículos como infraestrutura para fazer um melhor uso da comunicação veicular e seus recursos computacionais [Hou et al. 2016]. A tecnologia VFC é introduzida para apoiar o ITS, oferecendo recursos computacionais através de mais componentes disponibilizados como servidores e assim melhorar o processamento de aplicações, diminuindo a sobrecarga de processamento e as taxas de atraso na rede no *offloading* computacional[Shi et al. 2020]. A integração de pedestres a uma arquitetura VFC tem potencial de melhorar suas capacidades de computação e diminuir a sobrecarga de rede no processo de *offloading* computacional, com poucos trabalhos na literatura desenvolvendo soluções que tratem do *offloading* de tarefas nesse contexto.

Este trabalho procura integrar VRUs, precisamente pedestres, à tecnologia VFC, criando cenários com diferentes componentes de comunicação, que incluem veículos, pedestres, servidores de borda e servidores de nuvem. O trabalho introduz um algoritmo de decisão para o processo de *offloading* computacional em VFC com o pedestre como origem (P2X), e avalia o desempenho do algoritmo proposto em cenários com suporte a três diferentes categorias de componentes, além de comparar com um algoritmo clássico do tipo FIFO (*First in First out*).

As principais contribuições do presente trabalho são listadas a seguir:

- Criação de ambiente simulado P2X, utilizando tecnologias de acesso de ondas milimétricas (mmwave) 5G para a comunicação entre veículos, pedestres, servidor de borda e Nuvem. Os cenários criados estão disponíveis para a comunidade[1];
- Criação e implementação de um algoritmo de decisão de *offloading* de tarefas em uma rede veicular com VFC e pedestre; e
- Resultados da avaliação de desempenho da solução proposta, mensurando a performance do algoritmo de *offloading* através de quatro métricas: tempo de *offloading*, taxa de execuções bem sucedidas, taxa de recuperação e taxa de execução

[1] https://github.com/PauloHGR/manhattan_cenario_SBRC

local de tarefas.

O presente trabalho está divido da seguinte forma. Na Seção 2, são abordados os trabalhos relacionados. Na Seção 3, é apresentado o algoritmo proposto, bem como detalhes de implementação. A Seção 4 apresenta o processo de realização dos experimentos de avaliação da solução proposta. Na seção 5, os principais resultados alcançados nos experimentos são discutidos. Por fim, na Seção 6, as principais conclusões e os trabalhos futuros são apresentados.

2. Trabalhos Relacionados

Em [Wang et al. 2018], é apresentado um *offloading* computacional em que usuários de dispositivos buscam veículos próximos para processar suas aplicações. O objetivo é utilizar efetivamente os recursos computacionais disponíveis nos veículos inteligentes ao redor dos pedestres, transformando tais veículos em *cloudlets*. Os autores ainda propõem um esquema de retransmissão entre os *cloudlets*, ao invés de executar todas as tarefas em um único *cloudlet*. Além disso, foi desenvolvido um algoritmo para apoiar o processo de decisão com base no melhor tempo de troca entre *cloudlets*. O tempo de troca entre *cloudlets* é tempo para mudar para um novo *cloudlet* e selecionar o melhor como o próximo local de execução ou retornar a execução de volta ao veículo requisitante de *offloading*. Os autores utilizam apenas veículos como componentes do *offloading*, consequentemente seu algoritmo de decisão considera apenas essa entidade.

No trabalho [Lin et al. 2019], os autores implementam uma arquitetura federada com o uso de servidores de borda e veículos como componentes de uma *Fog* Veicular. Os autores desenvolveram um algoritmo guloso iterativo com o intuito de facilitar o processo de decisão de *offloading*, com base nos recursos de cada componente. O algoritmo verifica a disposição dos recursos e faz um balanceamento de quantas tarefas enviar para cada componente. Por fim, eles analisam a redução de custo proporcionada pela solução. Os autores, porém não realizam uma medição do impacto do tempo de *offloading* no cenário e também não usam essa arquitetura como provedor de recursos para pedestres.

Um esquema de recuperação de dados para *offloading* computacional em VEC (*Vehicular Edge Computing*) é proposto por [Boukerche and Soto 2020]. Os autores apresentam um modelo de sistema de *offloading* computacional para avaliar a capacidade de simular os elementos que contribuem para melhorar seu desempenho em um ambiente com computação em borda, veículos e pedestres. O trabalho, porém não envolve o compartilhamento de tarefas entre pedestres e veículos no processo de *offloading*, somente entre veículo/pedestre para RSU (Road-Side Unit).

Alguns trabalhos implementam esquemas de comunicação, com foco no veículo como origem, como em [Nguyen et al. 2020], em que os autores propõem uma abordagem adaptativa para *smartphones*, que podem ser o destino para o *offloading* de tarefas vindas dos veículos. Os autores também permitem a execução local de tarefas, caso não existam opções mais vantajosas. Além disso, o consumo de energia e a latência de cada esquema são investigados. O trabalho utiliza também um servidor de borda, porém os autores não realizam medições da taxa de *offloading* bem-sucedido ou taxa de execução local no processamento das tarefas.

No trabalho [Gonçalves et al. 2019], é implementado um esquema de avaliação da predição de mobilidade de usuários de dispositivos em *Fog Computing* no contexto

de Cidade Inteligente. O objetivo do trabalho é analisar o comportamento do processo de migração de aplicações em diferentes cenários de um ambiente de *Fog Computing*. Contudo, não é considerado no trabalho uma aplicação de *offloading* computacional e uma posterior avaliação de desempenho deste processo nos cenários propostos.

O trabalho [de Almeida et al. 2020] investiga a capacidade do *Wi-Fi Direct* oferecer conectividade no ambiente veicular. Com base nas trocas de informações entre um veículo e um pedestre em uma arquitetura V2P (*Vehicle-To-Pedestrian*), são analisados os resultados de medições reais usando *smartphones* comerciais. Os autores também não consideram cenários de *offloading* computacional em suas avaliações.

O presente trabalho implementa um algoritmo de decisão de *offloading* em VFC, integrado a dispositivos de pedestres. Os trabalhos apresentados utilizam os pedestres em comunicação com servidores remotos como Borda ou veículos, mas nenhum trabalho utiliza VFC como servidor de recursos para pedestres. Os trabalhos que utilizam VFC têm como foco os veículos como origem do *offloading* e não pedestres. Além disso, as métricas analisadas no presente trabalho não foram abordadas em conjunto nos trabalhos citados.

3. Offloading Computacional em VFC com Pedestres

Esta seção apresenta a abordagem proposta em três partes. A primeira parte engloba o processo de descoberta de veículo a ser integrado à comunicação VFC. Na segunda parte, é apresentado o algoritmo de tomada de decisão para o processo de *offloading* em P2x. A última parte trata da configuração do ambiente de rede veicular para suportar o P2X e possibilitar o *offloading* computacional.

3.1. Processo de Descoberta de Veículo

Para inserir veículos na comunicação e, consequentemente, no processo de *offloading*, é preciso selecionar o veículo mais apropriado para cada situação. A Figura 1 ilustra o processo de descoberta.

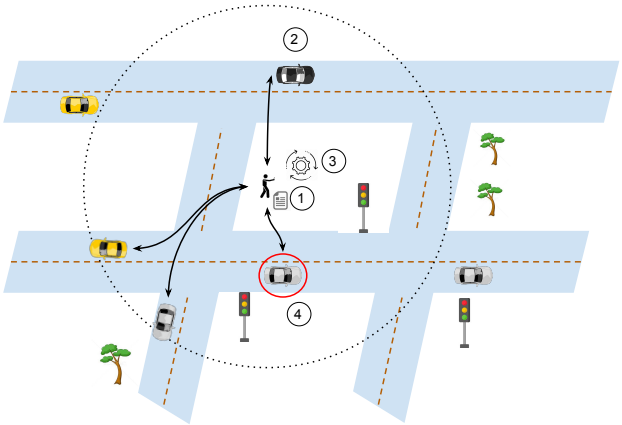

Figura 1. Processo de descoberta de veículos.

O pedestre envia requisições *broadcast* diretamente a todos os veículos no seu raio de alcance (Figura 1 — etapa 1). Os veículos próximos respondem à requisição com algumas informações importantes para o processo de escolha, tais como velocidade,

posição e percentual de uso da CPU do veículo (Figura 1 — etapa 2). Inicia-se então o processo de decisão (Figura 1 — etapa 3) e o primeiro passo é calcular a estimativa do tempo de vida do enlace entre o pedestre(i) e um veículo(j), com base no cálculo LLT(*Link Lifetime*) [Souza et al. 2013]. Os veículos que tiveram uma predição de tempo $LLT_{ij} = 0$ foram descartados do processo de escolha. O segundo passo é verificar a distância entre o pedestre e o veículo. Foi definido um valor máximo de alcance de 250 metros, eliminando qualquer veículo que esteja a uma distância superior. Em seguida, coleta-se o percentual de uso da *CPU* de cada carro, descartando aqueles veículos com o uso total da *CPU*. Por último, o veículo que possui mais CPU livre e um maior tempo de vida do enlace estimado, nessa ordem, é escolhido.

Uma vez escolhido o melhor veículo do ambiente (Figura 1 - etapa 4), a estação base mais próxima do veículo é identificada, um servidor de borda ligado ao gNB é instanciado e o servidor de nuvem também é ativado. A Figura 2 ilustra a topologia de comunicação P2X quando todos os componentes estão disponíveis para o *offloading* computacional.

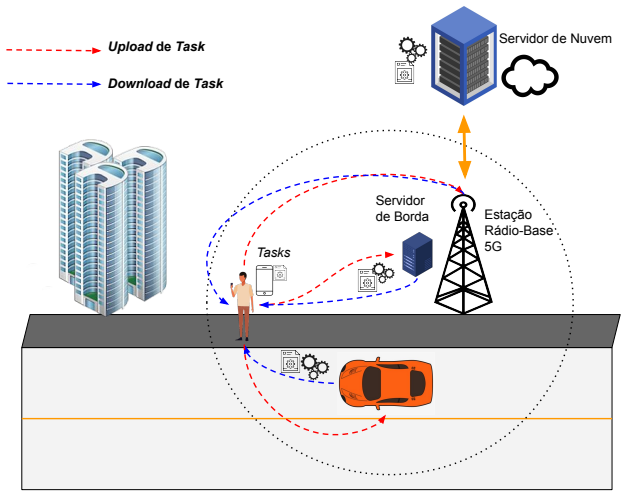

Figura 2. Topologia de comunicação P2X.

A comunicação de acesso entre veículo, pedestre e borda é realizada via rede celular, através do acesso 5G, como explicado na subseção 3.3. Para realizar a comunicação com o servidor na nuvem, foi configurado um *link* ponto a ponto entre a borda e a nuvem.

3.2. Algoritmo de Decisão de *Offloading*

Com todos os componentes identificados, é implementado então o processo de decisão do *offloading* computacional, que está definido no Algoritmo 1. Para iniciar o processo de decisão, o algoritmo recebe as informações de todos os componentes da comunicação, sendo *Edge*, *Vehicle* e *Cloud*. Essas informações são recebidas através de respostas à solicitação de *offloading* realizada pelo pedestre à rede VFC, sendo as respostas armazenadas em $\{DiscoveryPacket\}$. O $\{DiscoveryPacket\}$ contém as informações de cada pacote de resposta do componente, que será chamado servidor remoto.

O conjunto $\{providers\}$ armazena os servidores remotos encontrados pelo pedestre para realizar o *offloading*. Já o conjunto $\{Tasks\}$ armazena as tarefas processadas pelos servidores remotos ou pelo próprio dispositivo do pedestre. O conjunto $\{ST\}$ é

Algorithm 1: Algoritmo de Decisão

Input: {DiscoveryPacket}, Edge, Vehicle, Cloud, numberOfTasks
Output: {Tasks}

```
1  providers ← {}; tasksSelected ← 0;
2  Tasks ← {}; ST ← {}
3  foreach packet in {DiscoveryPacket} do
4  |   {providers} ← getComponent(packet);
5  {providers} ← Sort({providers});
6  foreach provider in {providers} do
7  |   if tasksSelected < numberOfTasks then
8  |   |   cpu ← getCPU(provider);
9  |   |   taskToEachProvider ←
   |   |     balacerTask(provider, cpu, numberOfTasks, tasksSelected);
10 |   |   ST ← {ST} ∪ taskToEachProvider;
11 |   |   tasksSelected ← tasksSelected + taskToEachProvider;
12 if tasksSelected < numberOfTasks then
13 |   localTasks ← numberOfTasks − tasksSelected;
14 |   task ← localExecution(localTasks);
15 |   Tasks ← {Tasks} ∪ task;
16 Tasks ← sendTasks(providers, ST);
17 return {Tasks};
```

responsável por armazenar as tarefas selecionadas para os servidores remotos pelo Algoritmo 2.

O processo então começa com o algoritmo realizando uma verificação de todas as respostas recebidas através de uma iteração (linha 3). É verificado cada pacote por meio da função $getComponent$, que identifica de qual componente veio a requisição, retornando o componente e suas informações para $\{providers\}$ (linhas 4). Os componentes $providers$ são ordenados pela função $Sort$ (linha 5), sendo os componentes mais prioritários aqueles que receberão mais tarefas. Os componentes são ordenados pelo seu tipo e percentual de CPU livre, onde a prioridade dos tipos é definida na seguinte sequência: borda, veículo e Nuvem. Com a verificação terminada, o próximo passo é realizar o balanceamento de tarefas, que é o processo em que se decide quantas tarefas serão processadas em cada componente.

Iterativamente, é verificado se cada servidor remoto *provider* possui tarefas a serem processadas (linhas 6-7). A variável $taskSelected$ armazena a quantidade de tarefas já atribuídas a algum servidor, com o valor inicial começando em zero. Para cada componente pertencente ao *provider* a função *balancerTask* é chamada (linha 9), tendo como retorno o número de tarefas que o componente receberá. Essa função recebe o percentual de uso da CPU do componente, consultado pela função *getCPU*(linha 8), o próprio componente, o número total de tarefas a serem processadas (*numberOfTasks*) e a quantidade de tarefas já atribuídas a outros componentes (*tasksSelected*).

A função *balancerTask* é definida no Algoritmo 2. A função verifica se o uso de CPU pelo *provider* é menor que o percentual necessário para processar uma tarefa ($RA_{provider}$). Se for menor, então o *provider* consegue processar a tarefa, logo a variável (*tasksToProvider*) é incrementada. A variável $RA_{provider}$ atualiza o seu novo estado com mais uma tarefa, passando a ter um maior percentual de CPU ocupada. Por fim, se não

houver mais recursos, o valor de *tasksToProvider* é retornado.

Algorithm 2: balancerTask

```
   Input: provider, cpu, numberOfTasks,tasksSelected
   Output: tasksToProvider
1  tasksToProvider ← 0
2  for i in range(taskSelected,numberOfTasks) do
3  |  if cpu < RA_provider then
4  |  |  tasksToProvider ← tasksToProvider + 1;
   |  |  RA_provider ← RA_provider ∪ tasksToProvider;
5  return tasksToProvider;
```

De volta ao Algoritmo 1, o retorno da função *balancerTask()* é armazenado em $\{ST\}$ na mesma posição em que o componente foi inserido no conjunto $\{providers\}$ (linha 10). A quantidade de tarefas selecionadas é incrementada em *tasksSelected* (linha 11). O processo retorna para verificar se as tarefas selecionadas já atingiram o número total de tarefas que o pedestre solicitou para serem processadas (*numberOfTasks*), encerrando o balanceamento assim que esse valor é alcançado (linha 7).

Caso o número de *tasksSelected* não tenha atingido o número de tarefas solicitadas pelo pedestre (*numberOfTasks*) após o laço de iteração encerrar, é realizada uma última verificação de modo a descobrir se sobraram tarefas a serem processadas (linha 12). Caso hajam tarefas, é realizada uma operação para coletar a quantidade de tarefas remanescentes executá-las localmente no dispositivo do pedestre, através da função *localExecution()* (linha 14). O resultado é armazenado no conjunto $\{Tasks\}$, responsável por receber as tarefas processadas (linha 15).

Ademais, as tarefas são enviadas para serem executadas em outros componentes através da função *sendTasks* (linha 16). A função *sendTasks* recebe o conjunto de servidores e o conjunto $\{ST\}$ e envia para cada *provider* do conjunto de $\{providers\}$ as respectivas tarefas armazenadas no conjunto $\{ST\}$. Na função *sendTasks*, cada *provider* recebe as tarefas atribuídas a ele para processamento, com essa associação sendo realizada pela posição de *index* dos conjuntos $\{Providers\}$ e $\{ST\}$. Os resultados são inseridos no conjunto $\{Tasks\}$ e retornados ao requisitante (i.e., o pedestre).

3.3. Arquitetura de Comunicação

Para desenvolver este trabalho, foi utilizado o simulador de redes ns-3[2], com a ferramenta SUMO[3], responsável por gerar mapas e condições de tráfego. Um cenário urbano foi escolhido para ser o ambiente de tráfego da simulação, gerado com o SUMO, a partir de uma parte da região de Manhattan, na cidade de Nova York, EUA, equivalente a 2 km², como mostra a Figura 3 [Eckermann et al. 2019].

Além do mapa da região, o SUMO gera um arquivo de mobilidade, com veículos, pedestres e rotas, que o ns-3 importa, e com isso os veículos e pedestres são transformados em nós de rede. Posteriormente, estações-base são instaladas no cenário para permitir a comunicação entre os nós, realizada pelo módulo *mmWave* do ns-3, responsável por implementar a comunicação de acesso 5G. A tecnologia mmWave permite a configuração

[2]https://www.nsnam.org/

[3]https://sumo.dlr.de/docs/index.html

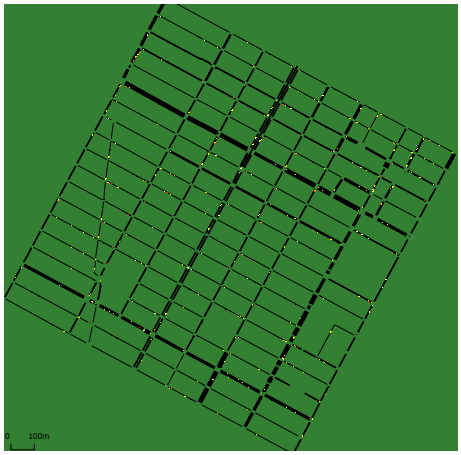

Figura 3. Cenário utilizado neste trabalho — região de Manhattan, Nova York, EUA.

das camadas físicas e de enlace do 5G nos dispositivos utilizados pelos componentes (i.e., pedestre, veículo e servidor de borda). As configurações utilizadas estão resumidas na Tabela 1.

Parâmetro	**Valor**
Ponto a Ponto	
Datarate	1 Gb/s
Delay	0.010 s
MTU	1500 bytes
Acesso 5G	
Datarate	450 Mbps
Modelo de Rádio Propagação	5G mmwave systems [Mezzavilla et al. 2018]
Demais Configurações	
Velocidade máxima dos veículos	16 m/s
Velocidade dos pedestres	1 m/s
Protocolo de Descoberta de Veículo	UDP
Protocolo de Transferência de Tarefas	TCP
Tempo de Simulação	70 s

Tabela 1. Principais Configurações da Simulação.

Para possibilitar tal comunicação, uma estação-base (ou gNB, *Next Generation NodeB*) próxima a um pedestre é escolhida e um servidor de borda é instanciado próximo à gNB. Foram configuradas 16 estações-base no cenário, de modo a garantir cobertura total da rede 5G em todo o cenário.

4. Avaliação Experimental

De modo a validar o algoritmo proposto, é realizada uma série de experimentos para avaliar métricas de desempenho do processo de *offloading* computacional. As métricas abordadas são: taxa de sucesso de *offloading*, taxa de execuções locais, taxa de falhas de *offloading* e tempo total de *offloading* computacional.

A abordagem é avaliada em três diferentes contextos, com o algoritmo proposto em 1 sendo utilizado e adaptado para cada contexto. No primeiro contexto, o algoritmo

considera apenas o servidor de borda interligado com o pedestre, cenário chamado LE. Depois, uma abordagem chamada LEC é avaliada, onde o servidor de borda e o servidor de nuvem estão disponíveis para o pedestre. Por fim, num contexto chamado LECV, é analisado o algoritmo com todos os componentes da VFC, ou seja, com veículos e servidores de borda e nuvem interligados ao pedestre. Esses três contextos são avaliados e comparados ainda com um algoritmo de decisão aleatória de *offloading*, chamado FIFO. No FIFO, assim como no LECV, todos os componentes estão disponíveis, mas a alocação de tarefas é realizada de forma aleatória.

Além dos algoritmos, outros dois fatores foram utilizados nos experimentos, a densidade de veículos/pedestres e o tamanho das tarefas a serem executadas. Com base em [Li et al. 2019] e [Ibrahim and Weigle 2008], foram definidos três diferentes níveis para a densidade: densidade baixa, com 50 veículos e 50 pedestres; densidade média com 276 veículos e 276 pedestres e, por fim, densidade alta com 609 veículos e 510 pedestres. Com relação ao tamanho da tarefa, foram utilizados dois níveis: 558 KB e 1.2 MB [de Souza et al. 2020]. Os fatores e seus respectivos níveis estão resumidos na Tabela 2, que também demonstra as combinações de fatores entre densidade de tráfego e tamanho da tarefa.

Número de Veículos	**Número de Pedestres**	**Tamanho da Tarefa**	**Identificador da Combinação de Fatores**
50	50	558 KB	B1
50	50	1.2 MB	B2
276	276	558 KB	M1
276	276	1.2 MB	M2
609	510	558 KB	D1
609	510	1.2 MB	D2

Tabela 2. Combinação dos Fatores de Densidade e Tamanho da Tarefa.

Com os parâmetros, fatores e níveis definidos, os experimentos foram executados em uma máquina com o ns-3 instalado com o módulo *mmwave* versão 3.29 e com as seguintes configurações: Processador *Xeon* E5645 @2.40GHz com 24 núcleos e 32 GB de *RAM*.

Foram selecionados aleatoriamente o pedestre e o tempo de início da simulação, e as *seeds* foram armazenadas para garantir que as comparações sejam justas e executadas com os mesmos valores. Para cada combinação de fatores (algoritmo, densidade e tamanho da tarefa), foram realizadas 80 execuções da simulação com o parâmetro quantidade de tarefas solicitadas pelo pedestre fixado em 8 (oito).

Para realizar a medição da métrica do tempo de *offloading* computacional, são coletados o tempo desde o envio da tarefa pelo pedestre até chegar ao componente servidor (*upload*), o tempo de processamento da tarefa e o tempo de envio da tarefa processada pelo servidor remoto ao pedestre (*download*). O valor de *upload* é dado pela variável UPL e o valor de *download* é dado pela variável DWL, ambos os valores medidos em segundos, conforme as equações a seguir:

$$UPL_{i,j} = \frac{TT}{DR} + \frac{D_{ij}}{VPM} + DLF \qquad DWL_{i,j} = \frac{TP}{DR} + \frac{D_{ij}}{VPM} + DLF \qquad (1)$$

onde TT refere-se ao tamanho das tarefas selecionadas para envio, DR é o *data-rate*, $D_{i,j}$ é a distância entre o pedestre (i) e o componente servidor (j), VPM refere-se à velocidade de propagação do meio e DLF é o atraso na fila de transmissão ou recepção do envio/recebimento do pacote. No cálculo do *download*, TP é o tamanho do resultado do processamento das tarefas recebidas. A seguir é demonstrado o cálculo do tempo de processamento das tarefas em segundos:

$$Proc_j = \frac{NT * C_t}{RA} \tag{2}$$

onde NT refere-se ao número total de tarefas recebidas pelo servidor remoto, C_t é a quantidade de CPU necessária para processar uma tarefa e RA é a quantidade de recurso disponível no servidor para realizar o processamento [Xiao et al. 2019]. Por fim, é calculado o tempo final de *offloading* (TFO), somando os tempos anteriormente calculados, como demonstrado na Equação 3.

$$TFO = UPL_{i,j} + Proc_j + DWL_{i,j} \tag{3}$$

Dado que o processo de *offloading* foi finalizado, a quantidade de tarefas processadas em uma determinada execução é contabilizada e armazenada em um arquivo de *log*, com a identificação da *seed* configurada, para permitir uma comparação pareada.

O resultado do processo de *offloading*, além da métrica TFO, permite computar outras três métricas relacionadas à eficiência do algoritmo, ao analisar o impacto da tomada de decisão no processo de *offloading*. A primeira é o percentual de tarefas processadas com sucesso e retornadas pelos servidores. A segunda refere-se ao percentual de tarefas executadas localmente no dispositivo do pedestre. E a última é o percentual de recuperações, que se refere à execução local de uma cópia da tarefa enviada a um servidor e que não foi retornada, devido a uma perda de conexão entre cliente e servidor.

5. Resultados

Nesta seção, são apresentados e discutidos os resultados dos experimentos realizados com base nas métricas mencionadas na seção anterior.

5.1. Eficiência do offloading

A Figura 4 apresenta as métricas relacionadas à eficiência da tomada de decisão de *offloading*. Como pode-se observar, o algoritmo LECV é o que apresenta a maior taxa de *offloading* realizado, o que é esperado já que nessa abordagem há todos os componentes da rede veicular VFC no processo de decisão. Nota-se também que o algoritmo LEC, apesar de ter menos componentes disponíveis, é melhor que o algoritmo FIFO. Um reflexo da eficiência na tomada de decisão é a baixa taxa de execuções locais, já que o pedestre envia mais tarefas para outros dispositivos e as escolhas parecem apropriadas, diminuindo as recuperações. Apesar de não utilizar veículos no processo de decisão, o algoritmo LEC é também uma opção melhor do que o algoritmo FIFO.

O algoritmo FIFO apresenta altas taxas de recuperação, reflexos do processo de aleatoriedade de sua escolha, dado que componentes inaptos podem ser escolhidos para o

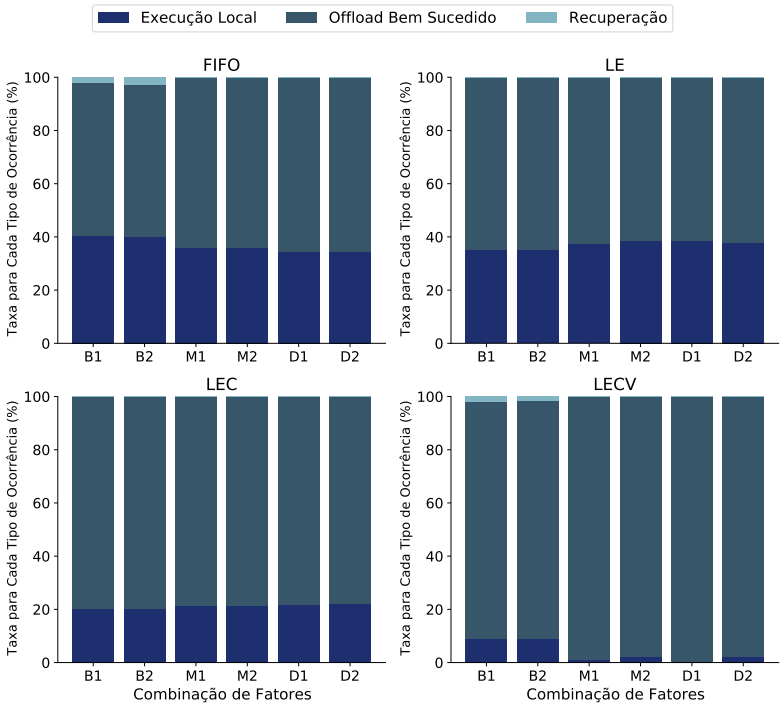

Figura 4. Eficiência do offloading com cada algoritmo.

processamento de tarefas. O algoritmo LE também apresenta altas taxas de recuperação, o que pode ser justificado pela sobrecarga gerada no componente, já que o servidor de borda é o único componente disponível para o processamento de tarefas.

Nas execuções em cenários de baixa densidade de tráfego (B1 e B2) dos algoritmos FIFO e LECV, há uma maior taxa de execuções locais e recuperações devido à maior possibilidade de não haver veículos próximos ao pedestre. Pelo contrário, tal resultado não é percebido em cenários com maiores densidades de tráfego, visto que haviam mais veículos disponíveis no cenário.

É importante ressaltar que o risco de utilizar veículos é maior, pois se aumenta a probabilidade de cliente e servidor perderem a comunicação entre si e, com isso, o pedestre não receber o resultado das tarefas compartilhadas devido às suas características de mobilidade. Entretanto, os resultados mostram que a taxa de recuperação não é grande, o que indica que a adoção da técnica de *offloading* nessa categoria de cenário é segura.

5.2. Tempo de *Offloading*

A Figura 5 apresenta a média do tempo de *offloading* com intervalo de confiança de 95%. Os resultados demonstram que a presença de mais opções para o *offloading*, como nos algoritmos LEC e LECV, ajudam a diminuir o tempo de duração do *offloading*. Um tempo de duração menor indica um processamento de tarefas melhor e, consequentemente, menor latência e melhor experiência para o usuário das aplicações. A abordagem de decisão de *offloading* proposta neste trabalho é melhor em relação à abordagem aleatória, dado que os requisitos de recursos dos dispositivos (CPU) são considerados para ponderar as decisões de *offloading*, resultando em processamentos mais rápidos.

Os resultados do algoritmo LE mostram que a utilização de apenas um servidor de

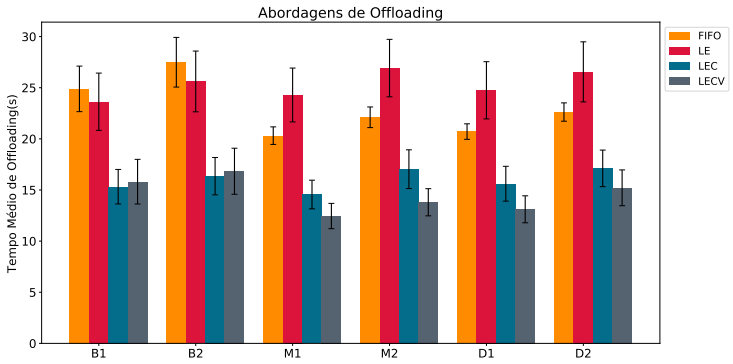

Figura 5. Tempo médio de *offloading* com cada Algoritmo.

borda (com poder de processamento limitado) pode ser insuficiente para melhorar o tempo de execução da aplicação, principalmente por estar sujeito a sobrecargas de requisições, caso seja o único componente a tratar *offloading* computacional.

Foram realizados testes estatísticos adicionais para melhor avaliar os resultados com sobreposição de intervalos de confiança. Inicialmente, testes de *Shapiro-Wilk* foram realizados, onde foi constatado que os dados não são normalmente distribuídos. Então, foram realizadas comparações pareadas entre os resultados usando os testes de *Wilcoxon*. Nos testes, a hipótese nula é que duas soluções comparadas possuem o mesmo desempenho. Se $p < 0,05$, a hipótese nula é rejeitada sendo considerado que as soluções comparadas possuem desempenhos diferentes estatisticamente. A Tabela 3 apresenta os valores de p para os testes executados.

	B1	**B2**	**M1**	**M2**	**D1**	**D2**
LE vs. FIFO	0,521	0,485	0,011	0,002	0,005	0,016
LEC vs. LECV	0,911	0,751	0,045	0,007	0,020	0,035

Tabela 3. Valores de p para testes de *Wilcoxon* pareados.

Com base nos resultados estatísticos, apenas os cenários de média e alta densidade (M1, M2, D1 e D2) apresentam diferenças estatísticas entre os resultados comparados ($p < 0.05$). Dessa forma, nos cenários de baixa densidade, LEC e LECV apresentaram os melhores desempenhos de tempo, não havendo diferença estatística entre eles. Além disso, em cenários de média e alta densidade, a solução LECV apresentou os melhores desempenhos de tempos de *offloading*, superando todas as outras soluções.

6. Conclusão

Este trabalho investigou como melhorar os avanços de comunicação em cenário de redes veiculares, especificamente VFC. Foi implementada uma solução para estender as aplicações de VFC em redes veiculares, trazendo mais dispositivos para o ambiente com a inserção de pedestres, que atuam como clientes do processo de *offloading* computacional. Um algoritmo de decisão de *offloading* foi apresentado e experimentos foram conduzidos para avaliar o desempenho do algoritmo aplicado em três contextos de densidade de tráfego e duas categorias de *workload*. Além disso, uma estratégia de decisão aleatória também foi utilizada nos experimentos.

Como os resultados demonstraram, a abordagem proposta neste trabalho tem potencial para melhorar o tempo de execução e eficiência do *offloading* computacional em ambientes de VFC com pedestres atuando como clientes. Tais melhorias refletem em menos sobrecargas dos servidores de borda e menos erros na escolha dos destinos do *offloading*, diminuindo retransmissões e latência, e consequentemente melhorando a qualidade das aplicações de ITS ou aplicações pessoais dos pedestres. Uma das contribuições deste trabalho é liberar para a comunidade os cenários criados, de modo a facilitar a reprodutibilidade dos experimentos.

Como trabalhos futuros, esperamos estender a solução para considerar pedestres como destinos do *offloading* e trabalhar com tecnologias D2D (Dispositivo-para-Dispositivo), assim como considerar outras métricas de avaliação, como consumo energético.

Referências

Boukerche, A. and Soto, V. (2020). An efficient mobility-oriented retrieval protocol for computation offloading in vehicular edge multi-access network. *IEEE Transactions on Intelligent Transportation Systems*, 21(6):2675–2688.

Chen, S., Hu, J., Shi, Y., Peng, Y., Fang, J., Zhao, R., and Zhao, L. (2017). Vehicle-to-everything (v2x) services supported by lte-based systems and 5g. *IEEE Communications Standards Magazine*, 1(2):70–76.

de Almeida, T. T., Júnior, J. G. R., Campista, M. E. M., and Costa, L. H. M. K. (2020). Uma análise de desempenho do wi-fi direct para comunicações veículo-pedestre. In *Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 253–266. SBC.

De Souza, A. B., Rego, P. A., Carneiro, T., Rodrigues, J. D. C., Rebouças Filho, P. P., De Souza, J. N., Chamola, V., De Albuquerque, V. H. C., and Sikdar, B. (2020). Computation offloading for vehicular environments: A survey. *IEEE Access*, 8:198214–198243.

de Souza, A. B., Rego, P. A. L., Rocha, P. H. G., Carneiro, T., and de Souza, J. N. (2020). A task offloading scheme for wave vehicular clouds and 5g mobile edge computing. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE.

Eckermann, F., Kahlert, M., and Wietfeld, C. (2019). Performance analysis of c-v2x mode 4 communication introducing an open-source c-v2x simulator. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pages 1–5. IEEE.

Gonçalves, D. M., Bittencourt, L. F., and Madeira, E. M. (2019). Análise da predição de mobilidade na migração de aplicações em computação em névoa. In *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 580–593. SBC.

Hou, X., Li, Y., Chen, M., Wu, D., Jin, D., and Chen, S. (2016). Vehicular fog computing: A viewpoint of vehicles as the infrastructures. *IEEE Transactions on Vehicular Technology*, 65(6):3860–3873.

Ibrahim, K. and Weigle, M. C. (2008). Cascade: Cluster-based accurate syntactic compression of aggregated data in vanets. In *2008 IEEE Globecom Workshops*, pages 1–10. IEEE.

Khan, M. J., Khan, M. A., Beg, A., Malik, S., and El-Sayed, H. (2022). An overview of the 3gpp identified use cases for v2x services. *Procedia Computer Science*, 198:750–756.

Li, F., Chen, W., Shui, Y., Wang, J., Yang, K., Xu, L., Yu, J., and Li, C. (2019). Connectivity probability analysis of vanets at different traffic densities using measured data at 5.9 ghz. *Physical Communication*, 35:100709.

Lin, Y.-D., Hu, J.-C., Kar, B., and Yen, L.-H. (2019). Cost minimization with offloading to vehicles in two-tier federated edge and vehicular-fog systems. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pages 1–6. IEEE.

Mezzavilla, M., Zhang, M., Polese, M., Ford, R., Dutta, S., Rangan, S., and Zorzi, M. (2018). End-to-end simulation of 5g mmwave networks. *IEEE Communications Surveys & Tutorials*, 20(3):2237–2263.

Nguyen, Q.-H., Morold, M., David, K., and Dressler, F. (2019). Adaptive safety context information for vulnerable road users with mec support. In *2019 15th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 28–35. IEEE.

Nguyen, Q.-H., Morold, M., David, K., and Dressler, F. (2020). Car-to-pedestrian communication with mec-support for adaptive safety of vulnerable road users. *Computer Communications*, 150:83–93.

Sewalkar, P. and Seitz, J. (2019). Vehicle-to-pedestrian communication for vulnerable road users: Survey, design considerations, and challenges. *Sensors*, 19(2):358.

Shi, J., Du, J., Wang, J., Wang, J., and Yuan, J. (2020). Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(12):16067–16081.

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

Wang, Z., Zhong, Z., Zhao, D., and Ni, M. (2018). Vehicle-based cloudlet relaying for mobile computation offloading. *IEEE Transactions on Vehicular Technology*, 67(11):11181–11191.

Xiao, L., Zhuang, W., Zhou, S., and Chen, C. (2019). Learning while offloading: Task offloading in vehicular edge computing network. In *Learning-based VANET Communication and Security Techniques*, pages 49–77. Springer.

Yoo, S. K., Cotton, S. L., Zhang, L., Doone, M. G., Song, J. S., and Rajbhandari, S. (2021). Evaluation of a switched combining based distributed antenna system (das) for pedestrian-to-vehicle communications. *IEEE Transactions on Vehicular Technology*, 70(10):11005–11010.