Uma Arquitetura Orientada a Serviços com Suporte ao Fatiamento na Borda da Rede para a Internet de Veículos

1st Roseli da R. Barbosa
Centro de Informática (UFPE)
Recife, Brasil
rrb2@cin.ufpe.br

2nd Rhodney A. M. B. K. Simoes
Centro de Informática (UFPE)
Recife, Brasil
rambks2@cin.ufpe.br

3rd Kelvin L. Dias
Centro de Informática (UFPE)
Recife, Brasil
kld@cin.ufpe.br

***Resumo*—Espera-se que o consumo de *streaming* de vídeo, bem como de mapas de alta definição, atualizados em tempo real, para o suporte à tomada de decisão para carros autônomos, aumentem significativamente o tráfego de dados nos sistemas de transporte inteligentes baseados nas redes 5G. Dois componentes-chave das redes 5G auxiliam no atendimento dos requisitos de baixa latência, QoS e mobilidade para tais cenários: fatiamento de rede (*Network Slicing*) e *Multi-access Edge Computing* (MEC). Com o objetivo de fornecer uma solução conjunta que compreenda o fatiamento de rede e MEC para a Internet de veículos, este artigo concebeu uma arquitetura orientada a serviços que viabiliza a instanciação e implantação de cadeias de serviços (SFCs). Para avaliar a proposta, um ambiente experimental foi implementado e análises de desempenho foram conduzidas para avaliar sua efetividade na criação de SFCs para redes veiculares.**

***Palavras-chave*—5G; Fatiamento de Rede; MEC; Internet de Veículos**

I. Introdução

A Internet dos Veículos (IoV) se beneficia de tecnologias de softwarização de rede para o gerenciamento de sua topologia dinâmica, requisitos de QoS, escalabilidade e orquestração de recursos. Para alcançar o gerenciamento flexível e eficiente em cenários de IoV, o fornecimento de serviços de nuvem mais próximos dos usuários finais é de suma importância para a implantação dos futuros serviços para IoVs. Por um lado, a tecnologia NFV (*Network Functions Virtualization*) permite a implementação e o provisionamento de serviços sob demanda. Dessa forma, os serviços de rede podem ser virtualizados e executados como softwares em máquinas virtuais (VNFs - *Virtual Network Functions*), sem a necessidade de utilizar dispositivos de hardware real, além de possibilitar que estas funções de rede sejam encadeadas logicamente em uma plataforma de nuvem. Uma fatia de rede (*Network Slice* - NS) corresponde a um conjunto de recursos dedicados para a instanciação de serviços, que podem ser encadeados e isolados logicamente através de uma rede virtual [1]. Assim, o fatiamento de rede viabiliza inquilinos com requisitos de serviços distintos em uma infraestrutura compartilhada. Por outro lado, a MEC (*Multi-access Edge Computing*) busca reduzir a latência para garantir um bom desempenho de aplicações que são sensíveis a atrasos, garantindo uma melhor experiência do usuário. Tanto o NFV como a MEC foram padronizados em conjunto pelo ETSI (*European Telecommunications Standards Institute*) em um ambiente integrado denominado MEC-NFV [2].

Fatiamento de redes e MEC são, geralmente, aplicados de forma independente como soluções para contemplar os requisitos de Redes Veiculares (VANETs - *Vehicular Ad Hoc Networks*). Com o objetivo de fornecer uma solução conjunta que compreenda o fatiamento de rede e MEC para a Internet de veículos, este artigo concebeu uma arquitetura orientada a serviços em conformidade com a arquitetura de referência do 3GPP (*Third Generation Partnership Project*). Trabalhos relacionados ou apresentam uma visão conceitual sobre fatiamento para redes veiculares ou não apresentam propostas e resultados de desempenho para o gerenciamento do ciclo de vida das VNFs. O artigo concebeu um middleware orientado a mensagens com base no modelo *publish/subscribe*. As camadas da arquitetura do middleware estão alinhadas com 5G SBA (*Service Based Architecture*) e incluem o componente MANO (*Management and Network Orchestration*). A proposta consiste em mecanismos de gerenciamento do ciclo de vida e alocação dinâmica de VNFs e SFCs (*Service Function Chaining*) para lidar com as flutuações do tráfego veicular. Para demonstrar a eficácia da proposta, foi desenvolvido um *testbed* para avaliar a transmissão de vídeos em SD (*Standard Definition*) e 4K em um cenário de rede veicular.

Apesar de existirem vários trabalhos e projetos com estratégias de fatiamento de rede, as soluções atuais carecem de uma visão orientada a serviços como proposto pelo 3GPP, e de uma abordagem dinâmica para lidar com cenários de IoV. Para este fim, destacamos as seguintes questões:

1) Como identificar os requisitos dos consumidores e tratá-los?
2) Considerando o compartilhamento de conteúdos para veículos conectados, como tornar a arquitetura genérica para o uso de diferentes protocolos de transporte (TCP ou UDP)?
3) Quais os benefícios que técnicas de *caching* trariam para um cenário de redes veiculares?

Com base nas questões de pesquisa levantadas, identifica-

mos algumas hipóteses com o desenvolvimento deste artigo:

1) Torna-se necessária a adesão de uma entidade intermediadora que seja capaz de traduzir os requisitos dos usuários em recursos/serviços alocados.
2) Através da adesão de um middleware torna-se possível construir uma camada de infraestrutura genérica, que seja capaz de lidar com diferentes tipos de protocolos de transporte.
3) O uso de estratégias de cache em redes veiculares é crucial por se tratar de ambiente altamente dinâmico e com desconexões frequentes, além de permitir a redução da latência total.

Este artigo está organizado da seguinte forma: a Seção II discute os principais trabalhos relacionados. A proposta de arquitetura orientada a serviços para IoV é apresentada na Seção III. As Seções IV e V descrevem o planejamento para a criação dos cenários de avaliação e as métricas utilizadas. A Seção VI apresenta os resultados da avaliação de desempenho da proposta. Por fim, a Seção VII conclui o artigo com as considerações finais.

II. Trabalhos Relacionados

A. Concepções Arquiteturais

Em [3], os autores propuseram uma arquitetura em camadas para o fatiamento em redes veiculares. Os autores utilizam o Mininet-WiFi para a emulação de fatias de rede via controlador SDN (*Software Defined Networking*) em um cenário veicular. Contudo, não implementam a criação e alocação de VNFs ou encadeamento destas. Além disso, a proposta arquitetural em camadas é conceitual, sem implementação e avaliação, como por exemplo, do orquestrador para gerenciamento de ciclo de vida de VNFs.

Por outro lado, nossa proposta é flexível por meio da concepção de um middleware e a inclusão de uma camada de aplicação, idealizada com base nas seguintes características: (i) O modelo *publish/subscribe* pode ser utilizado em qualquer tipo de aplicação que faça o uso de eventos; (ii) Permite a adequação dos tipos de parâmetros que formarão o corpo da mensagem, de acordo com os requisitos da camada de aplicação; (iii) Todas as mensagens são serializadas antes de serem enviadas; (iv) A camada de infraestrutura do middleware implementado permite o tráfego de dados usando o TCP ou UDP; (v) Através do uso de um *proxy* é fornecida a transparência de acesso e de localização.

Outro modelo arquitetural é apresentado em [4], com um foco mais amplo em serviços móveis para 5G como um todo. O trabalho exibe uma proposta de arquitetura baseada nas especificações do 3GPP, subdividindo em três domínios: rede de acesso, núcleo e transporte. Apesar de apresentar uma avaliação de desempenho com um escopo reduzido, a principal contribuição do trabalho consiste em discutir os desafios de pesquisa dentro do conceito de fatiamento de rede, apontando em cada domínio uma concepção sobre possíveis soluções. Ainda assim, nosso trabalho se assemelha com a abordagem utilizada para provisionamento de fatias de rede, considerando as informações dos usuários.

B. Estratégias de Alocação de Recursos

Os autores de [5] mostram uma estratégia de otimização através de três modelos complementares baseados em ontologia, como uma extensão do trabalho [6]. As ontologias *Network Slice Blueprint* e *Sub-network Blueprint* realizam a tradução dos requisitos dos usuários e identificam os recursos que serão alocados, respectivamente. Já em [7] a estratégia de alocação de recursos é baseada no *framework* implementado pelos autores, com o foco em arquiteturas MEC e foi denominado: JECRS (*Joint Edge and Central Resource Slicer*). A estrutura do *framework* possui uma interface que identifica os requisitos dos usuários e as trata através de um orquestrador de recursos. Também possui um módulo que realiza o monitoramento das fatias de rede, permitindo a criação/atualização/exclusão de tarefas. Assim como em nossa proposta, as informações de cada mapeamento são persistidas em um banco de dados.

Assim como os autores de [5] e [7], em nosso trabalho também consideramos as informações dos usuários para o gerenciamento de recursos, mas com uma abordagem diferente. Cada VNF possui uma capacidade máxima de atendimento de acordo com o *flavor* associado, e sua taxa de utilização é atualizada sempre que for escolhida no processo de seleção. Realizando o gerenciamento dessa forma se avalia não só a quantidade de recursos computacionais alocados, mas também os requisitos necessários para atender cada serviço provisionado.

Artigos recentes utilizam técnicas de inteligência artificial para o aprovisionamento de fatias de rede como em [8], [9], mas focam em aspectos de recursos de rádio e são avaliados via simulação. Particularmente, em [9] a MEC entra na solução como viabilizadora da execução de algoritmos de aprendizado por reforço profundo, mas não implementa fatiamento de rede, como feito por nossa proposta.

III. Orquestração de recursos orientada a serviços para IoV

A concepção da proposta foi baseada na arquitetura de referência MEC-NFV especificada pelo ETSI em [2]. O modelo implementando na arquitetura proposta se baseia na divisão em camadas amplamente utilizado em diversos estudos, a qual foi detalhado em [10]. As principais contribuições de cada camada são descritas nas próximas Seções.

A. Camadas de Aplicação e Serviço

De acordo com [11], faz-se necessária a adesão de uma camada intermediária de middleware para permitir o gerenciamento flexível de conexões externas. Neste trabalho, criamos uma abstração da camada de aplicação por meio de um middleware integrado na camada de serviços, o que permitirá a execução de diferentes tipos de aplicações para cenários de veículos conectados. Para este artigo, nosso foco é em serviços distribuição de conteúdo de *streaming* de vídeo.

1) Middleware orientado a mensagens: A Figura 1 apresenta a organização de cada camada do middleware implementado e os relacionamentos entre suas entidades. O middleware proposto possui quatro elementos principais: eventos, *subscribers*, *publishers* e um servidor de eventos. Um evento pode ser uma informação a ser processada ou até mesmo um conteúdo a ser fornecido, ou seja, um evento é algo de interesse de alguma entidade. Os *subscribers* registram o interesse e consomem eventos, já os *publishers* produzem os eventos que serão disponibilizados. Um servidor de eventos gerencia todas as subscrições e realiza o tratamento das mensagens. Os *publishers* são os produtores de conteúdo e os *subscribers* os consumidores/veículos (Elemento 12 da Figura 1). Seguimos os princípios da padronização *Remoting Patterns* abordado em [12] e reutilizamos os seguintes elementos: *Client Request Handler* e *Server Request Handler* na camada de infraestrutura, e o *Client Proxy*, *Marshaller* e o *Message Broker* na camada de distribuição, conforme descrito a seguir:

1) ***Client Proxy:*** Os clientes (*producers/consumers*) interagem com o *Client Proxy* que realiza a intermediação das requisições, fornecendo assim transparência de acesso e de localização do servidor remoto.
2) ***Client Request Handler:*** Para que todas requisições sejam efetuadas dos clientes para o servidor remoto é papel do *Client Request Handler* abrir e fechar as conexões de rede, bem como receber as mensagens de retorno e enviar para o cliente apropriado.
3) ***Server Request Handler:*** Tem a função de receber solicitações simultâneas e efetuar o gerenciamento e estabelecimento das conexões.
4) ***Marshaller:*** Todas as mensagens trafegadas na rede são serializadas em bytes antes de serem enviadas e deserializadas no recebimento. Tal processo é importante para lidar com a compatibilidade entre o cliente e o servidor. O processo de *Marshaller/Unmarshall* entre as entidades é demonstrado na Figura 1 na cor vermelha.
5) ***Request Server:*** Recebe e trata todas as mensagens recebidas dos *producers* e *consumers*. No ato da subscrição dos *consumers*, são identificados os principais parâmetros no corpo da mensagem e armazenados em uma base de dados (e.g., ID, conteúdo, protocolo e tipo de recurso).

Além dos elementos descritos, outras entidades foram adicionadas ao middleware proposto, tais como:

1) ***StartServer:*** Instancia o servidor e aguarda novas conexões.
2) ***Content:*** Detalha as características dos conteúdos de acordo com as necessidades da camada de aplicação, como nome, tamanho e sua localização.
3) ***Message:*** Específica os parâmetros que serão passados no corpo das mensagens, tais como ID, protocolo, conteúdo, *status* e entre outros. Além disso, o formato das mensagens é do tipo *Reply Packets*.
4) ***IListenerProxy:*** Devido à necessidade de estabelecer múltiplas requisições simultâneas, o *IListenerProxy* tem a função de implementar chamadas de retorno independentes e assíncronas.
5) ***IReceiver:*** Herda os comportamentos do *IListenerProxy* e trata as mensagens de retorno de acordo com as requisições realizadas pelo solicitante.
6) ***ConnectionPool:*** Em toda aplicação que faz o uso de um banco de dados, com inúmeros acessos simultâneos, o tratamento do número de conexões que serão realizadas é essencial. De maneira geral, a cada consulta o servidor abre uma nova conexão com o banco de dados e a fecha em seguida, tornando ineficiente tal utilização em larga escala. Para evitar que isso aconteça e buscando reduzir a latência, um *pool* de conexões foi implementado para reaproveitar as conexões ativas.

É importante salientar que na camada de aplicação pode ser utilizada por qualquer tipo de aplicação entre os *producers* e *consumers*. Para este trabalho foi adotada a transmissão de vídeos. As abstrações do middleware estão presentes na arquitetura da Figura 2 por meio dos seguintes elementos: (i) o *Publisher* e *Subscriber* da arquitetura equivalem ao *pub/sub* presentes na arquitetura do middleware; (ii) o *Message Broker* da arquitetura corresponde às camadas de distribuição e infraestrutura do middleware.

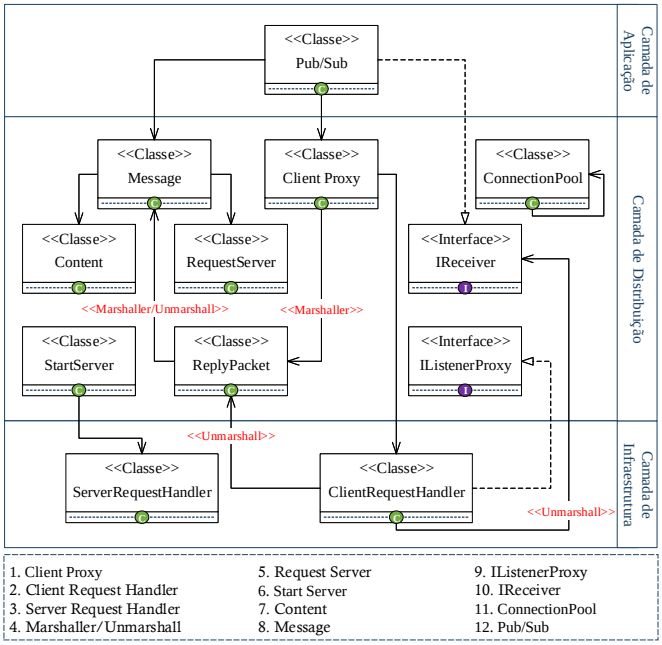

Figura 1. Arquitetura do Middleware

Um *consumer* ao se subscrever passa no corpo da mensagem seus interesses, tais como o nome e a qualidade do conteúdo (SD ou 4K), o protocolo de comunicação para o recebimento do conteúdo (TCP ou UDP) e se deseja ou não compartilhar recursos. Vale lembrar que essa abordagem foi adotada inicialmente e que será aprimorada em trabalhos futuros. Já os *producers* serão notificados quando houver alguma requisição de conteúdos sem saber quem o solicitou. Em seguida, a transmissão do vídeo para o cache é iniciada

após a instanciação completa da fatia de rede correspondente, de acordo com os parâmetros já definidos, como o protocolo de comunicação e a porta para a transmissão.

2) Cache de Conteúdos: Antes dos *publishers* enviarem os conteúdos solicitados para o cache, uma mensagem de *status* com o nome do conteúdo, protocolo de comunicação e seu tamanho é encaminhada para o *Message Broker*. O *Cache Agent* verifica o tamanho atual do cache e se o tamanho disponível suporta o novo conteúdo. Caso não seja possível adicionar o conteúdo, é iniciado o processo de verificação para identificar quais os conteúdos que estão sendo menos utilizados e que não estão sendo transmitidos nos últimos minutos. Se não houver nenhum conteúdo dentro da faixa especificada, o fator de decisão passa a ser apenas a taxa de requisição atual, sendo removidos do cache os conteúdos com menor taxa de requisição até que o novo conteúdo possa ser adicionado e, dessa forma, também é feito o gerenciamento do ciclo de vida dos conteúdos.

B. Camada de Gerenciamento e Orquestração

Na arquitetura proposta, o NFVO (NFV *Orchestrator*) é formado por quatro entidades: (i) *Database*, (ii) *Slicing Orchestrator*, (iii) *NF Agent* e (iv) *Slice Manager*. A seguir são apresentadas as principais características de cada módulo:

- ***Database*:** Persiste todas as informações de redes, VNFs, SFCs e *consumers*, com os relacionamentos necessários e os estados de cada processo. Este componente pode ser distribuído em cada nó da borda.
- ***Slicing Orchestrator*:** Identifica as requisições recebidas pelo *Message Broker* e trata de acordo com o SLA do usuário solicitante e o conteúdo requerido. Caso o conteúdo não esteja em cache, será instanciada uma fatia de rede para o usuário de acordo com o tipo de recurso e protocolo definidos. Após serem instanciadas todas as NFs e criada toda a SFC, o *producer* saberá que já é possível iniciar a transmissão do vídeo através da cadeia correspondente e de acordo com o classificador de fluxo SFC.
- ***NF Agent*:** É de responsabilidade do *NF Agent* identificar quais NFs serão atribuídas para cada fatia de rede instanciada pelo *Slicing Orchestrator*. Em cenários que o usuário solicitante não exige recursos totalmente dedicados, essa decisão é tomada com base no percentual de utilização das NFs já instanciadas. As NFs com as menores utilizações são sempre as primeiras candidatas consideradas no processo de alocação.
- ***Slice Manager*:** O ciclo de vida das VNFs é gerenciado de forma dinâmica com base no *status* das fatias de rede instanciadas. A cada finalização da transmissão dos vídeos, após novos interesses, o *status* da fatia de rede é atualizada, possibilitando ao *Slice Manager* identificar todas as desalocações necessárias (e.g., VNFs, SFCs, redes e entre outros). Para a desalocação, inicialmente é identificado o tipo de fatia de rede (se compartilha recurso ou não) e se cada NF está atendendo a mais alguma solicitação. Essa informação é importante para saber se será removido todo mapeamento criado (e.g., rede, portas, classificadores SFC) e as NFs ou apenas reduzidas suas taxas de utilização.

C. Camada de Infraestrutura

Na camada de infraestrutura propomos a utilização da plataforma OpenStack[1] para atuar como *Virtualized Infrastructure Manager* (VIM). Também propomos uma solução de orquestrador NFV e gerenciador de VNFs (*Slice Manager*) com base no framework ETSI NFV MANO [13]. O orquestrador viabiliza comunicações baseadas em REST - *Representational State Transfer* através da API - *Application Programming Interface* do OpenStack implantado.

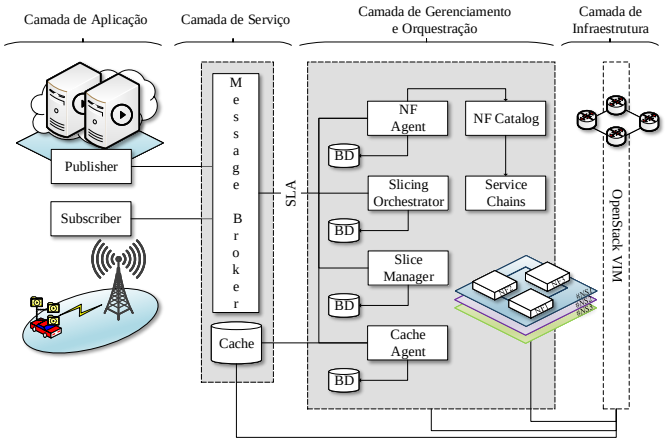

Figura 2. Arquitetura Projetada

IV. Planejamento e Testes Iniciais

Para avaliar a arquitetura proposta, uma nuvem privada foi implantada com a versão *Train* do OpenStack em um servidor com 90 GB de RAM, 300 GB de disco e processador 2 - 8 Core Intel Xeon E5530s. Já a geração de carga foi feita usando o emulador Mininet-Wifi[2] com o simulador de mobilidade urbana SUMO[3], executando em um servidor com 32 GB de RAM, 50 GB de disco e processador 8 Single Core Intel Xeon E5530s. Ambos servidores utilizaram o sistema operacional Ubuntu 18.04.3.

A implantação do OpenStack foi do tipo *All-In-One*, isto é, com a função de *controller* e *compute* no mesmo nó. Foram configurados apenas os serviços essenciais do tipo Core, compostos pelo *Nova, Keystone, Glance, Swift, Cinder* e o *Neutron* com a API de encadeamento de funções de serviço (*networking-sfc*). Tratando-se do ambiente de simulação, utilizamos 10 veículos e 6 RSUs (*Roadside Units*) com um mapa cobrindo parte de Manhattan - *New York* exportado através do OpenStreetMap[4]. O *trace* utilizado na simulação está ilustrado na Figura 3. A mobilidade dos veículos é definida com base

[1]OpenStack: https://www.openstack.org
[2]Mininet-Wifi: https://intrig.dca.fee.unicamp.br/mininetwifi
[3]SUMO: https://sumo.dlr.de/docs/index.html
[4]OpenStreetMap: https://www.openstreetmap.org

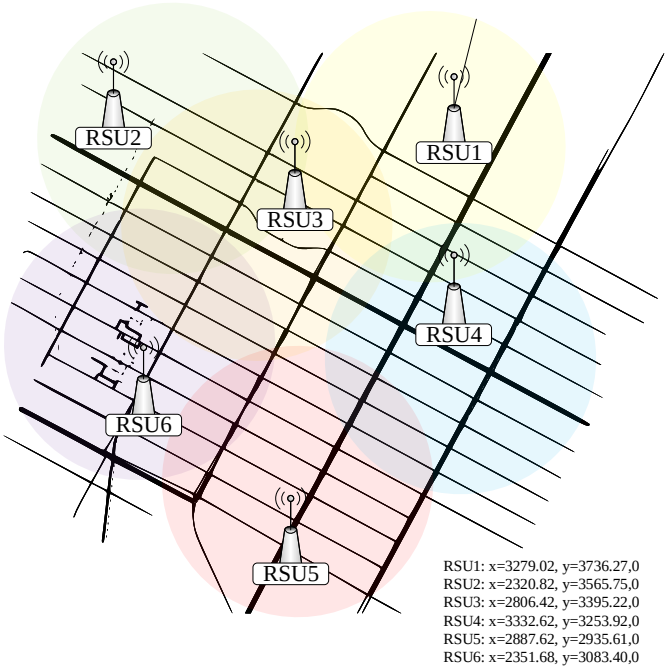

Figura 3. *Trace* com uma região de Manhattan – *New York*

nos arquivos de configuração do SUMO, cuja a velocidade máxima configurada foi de 11,18 m/s. Detalhes adicionais dos arquivos de configuração podem ser visualizados no repositório do Mininet-Wifi em *mn_wifi/sumo/data*.

No Mininet-WiFi é possível ter a comunicação V2V (*Vehicle to Vehicle*) e a V2I (*Vehicle to Infrastructure*) que foi o tipo de comunicação adotada em nosso ambiente de experimentação. Também adicionamos um nó para atuar como NAT para fornecer acesso à Internet aos nós emulados. Em virtude disso, conforme os veículos entram no ambiente de simulação verifica-se se estão associados à alguma antena e se estão com acesso à Internet para efetuar as requisições. Embora a quantidade de veículos seja reduzida no cenário avaliado, cada veículo gera inúmeras requisições durante todo o processo de simulação, utilizando sementes aleatórias com intervalos exponenciais entre cada requisição por meio da biblioteca *gsl-randist*[5]. Buscamos explorar o máximo no número de requisições para mensurar o comportamento da nossa aplicação.

Com os resultados preliminares, foi possível perceber que o número de veículos emulados gerando carga simultaneamente causa um impacto considerável que reflete no tempo de resposta, já que cada requisição corresponde a uma nova thread no servidor onde os nós foram emulados, compartilhando então CPU, memória e o link de transmissão. Por esse motivo, foram utilizados apenas 10 veículos.

V. Projeto dos Experimentos

Diversos experimentos preliminares foram realizados visando mensurar a quantidade de requisições que uma VNF pode atender com recursos mínimos, com a finalidade de definir sua capacidade máxima. Para tal, foram avaliadas utilizações de CPU e de memória aceitáveis de acordo com o número de requisições, durante todo o ciclo de vida das VNFs. Foi possível identificar que uma VNF com 256 MB de RAM, 2 vCPUs e 10 GB de disco pode atender até 5 requisições.

A Figura 4 ilustra todas as variações nos cenários de avaliação, que são: Sem Cache (SC) e Com Cache (CC) na MEC. Para cada fatia de rede é criada uma rede do tipo *self-service* como rede privada, estas são interligadas a um roteador virtual criando assim uma conexão entre as redes internas com a rede externa (*provider*). Na Figura 4, cada rede virtual representa a fatia de rede à qual as VNFs estão associadas, exceto a *net0* que consiste na rede de gerenciamento.

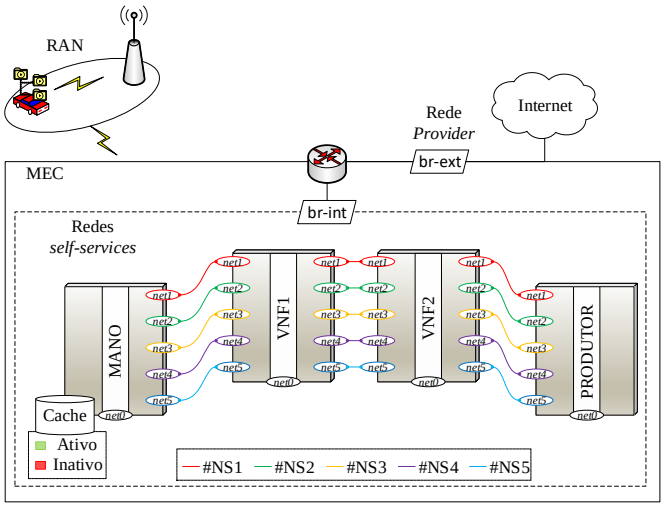

Figura 4. Ambiente de Experimentação

Com o intuito de reduzir o escopo de avaliação, em todos os cenários avaliados foi utilizado apenas um produtor de conteúdo transmitindo vídeos usando o TCP ou o UDP, conforme o tipo de protocolo requerido pelos consumidores. O vídeo escolhido para ser transmitido possui 15 segundos de duração e foi utilizado em duas qualidades, em SD (480p - 1,3 MB) e em 4K (2160p - 10,9 MB).

Para cada ambiente criado, as seguintes métricas foram avaliadas: (i) Utilização de CPU, (ii) Utilização de Memória e (iii) Tempo de Resposta. Tratando-se dos fatores e níveis, consideramos o número de veículos (0-10), quantidade de requisições por veículo (distribuição exponencial), tipo de tráfego (TCP ou UDP), uso do cache (sim ou não), qualidade do vídeo (SD ou 4K) e a quantidade de execuções (30 execuções de 20 minutos cada). Apenas o tipo de recurso compartilhado será avaliado para minimizar o escopo de avaliação. Para cada métrica avaliada os resultados obtidos são apresentados de duas formas: o comportamento individual de apenas uma amostra e o valor médio de todas as amostras com seus respectivos intervalos de confiança, utilizando um nível de confiança de 95%. As subseções a seguir exibem os gráficos gerados e uma análise dos resultados obtidos. Seguem

[5]Biblioteca *gsl-randist*: https://linux.die.net/man/1/gsl-randist

algumas informações importantes para facilitar o entendimento dos gráficos:

- **Utilização de CPU e Memória:** (i) Com uma amostra são apresentados os instantes iniciais e finais de cada SFC, que também expressam seus ciclos de vida (Figuras 6a, 7a, 9a e 10a); (ii) Nos gráficos com a média das amostras, as utilizações são agregadas pelo tempo (Figuras 6b, 7b, 9b e 10b).
- **Tempo de Resposta:** (i) Foi calculado o percentil 90 para todos os resultados e a única diferença consiste em como são exibidos. Com apenas uma amostra são expostos os valores que correspondem a 90% do que foi coletado, removendo assim os *outliers*. As linhas pontilhadas na vertical representam os instantes que os conteúdos ficam disponíveis em cache (Figuras 8a e 11a); (ii) Já nos gráficos com a média das amostras, são indicados apenas os limites obtidos com o percentil 90, através de linhas tracejadas na horizontal. Além disso, cada ponto representa a média obtida com os respectivos intervalos de confiança (expostos com a cor de fundo suavizada), calculados a cada 10 segundos (Figuras 8b e 11b); (iii) As linhas de tendência são representadas de acordo com a cor da fatia de rede correspondente.

VI. Avaliações e Resultados

Para mensurar o tempo para alocação de recursos, criação de SFCs e a definição do roteamento entre as VNFs na plataforma OpenStack, a Figura 5 exibe os valores médios obtidos no ambiente de experimentação deste trabalho, considerando os recursos computacionais utilizados. Em todos os cenários, esse tempo de provisionamento inicial está inserido no tempo de resposta. Os elementos envolvidos na criação de SFCs no OpenStack são: instanciação de VNFs, configuração das tabelas de roteamento e criação de redes virtuais, portas, pares de portas e seus grupos, classificadores de fluxos e as cadeias.

Os tempos de cada elemento descrito anteriormente são indicados através da criação de cinco fatias de rede, cada uma com duas VNFs, na Figura 5a (gráfico à esquerda). Por outro lado, no gráfico à direita, temos as médias que foram obtidas com o provisionamento total, por meio de 30 amostras com as mesmas características do gráfico à esquerda. Também é importante destacar que, os resultados ilustrados na Figura 5b, foram gerados a partir de cenários com compartilhamento de recursos, onde todas as fatias de rede foram alocadas para as VNFs 1 e 2. Tais características de cenários são detalhadas nas próximas Seções.

O módulo *Slice Manager* é acionado durante o início de cada avaliação, logo, é possível observar alguns impactos devido às modificações realizadas nas VNFs durante o processo de desalocação e, por esse motivo, ele deve ser considerado para a análise dos resultados.

A. Sem Cache

De maneira geral, a utilização de CPU apresentou maiores valores em todos os resultados se comparada com a utilização de memória, já que não há aplicações consideradas ”pesadas”

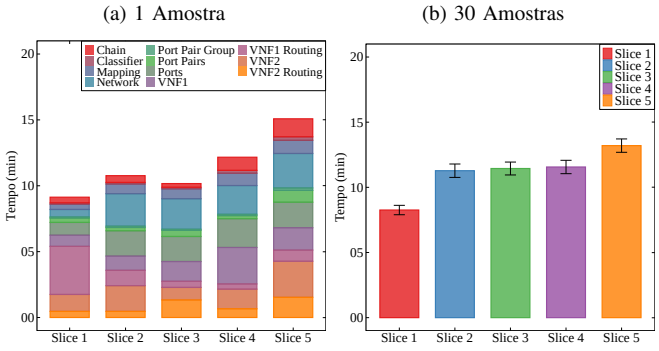

Figura 5. Tempo de Alocação/Provisionamento

operando nas VNFs. Também foi possível perceber que as maiores utilizações, em sua maioria, resultaram nos *setups* iniciais das VNFs, sendo identificado nos gráficos nos instantes iniciais ou quando há apenas uma fatia de rede instanciada, já que também significa que outras quatro fatias de rede ainda estão sendo provisionados simultaneamente.

Todos os resultados de utilização de CPU e memória obtiveram os limites inferiores e superiores próximos à média. Em experimentos SC, a maior média atingida de CPU foi de 86,27% com uma transmissão em 4K por meio do protocolo UDP (Figura 6b). Tal valor representa 10,64% a mais se comparado com a mesma qualidade do vídeo sendo transmitido sobre o TCP (Figura 6b), ambos percentuais foram obtidos pela VNF1. Já as menores utilizações de CPU vieram da VNF2, sendo o menor valor 15,81% correspondente aos resultados expostos na Figura 6b, onde também representa 18,46% a menos em relação à Figura 6b.

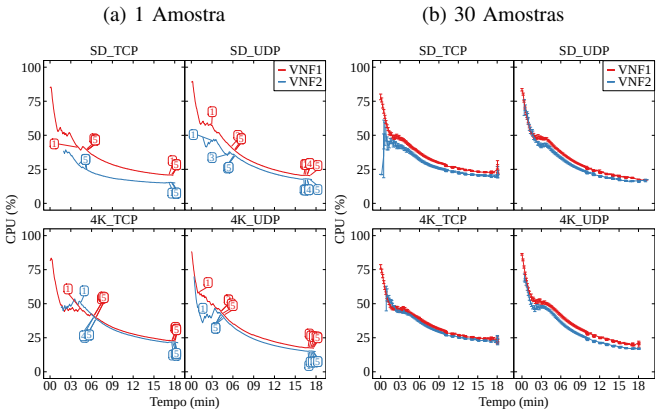

Figura 6. Utilização de CPU - Cenário SC

Diferente dos resultados de utilização de CPU, a maior utilização de memória ocorreu com a transmissão em SD usando o protocolo UDP (Figura 7b), com a média de 43,38%. Todos os resultados obtidos em experimentos SC mostram no máximo uma média de 40,96% de utilização de memória.

O tempo de resposta da Figura 8 refletiu em valores esperados, por tais motivos: (i) O NS1 apresentou melhores

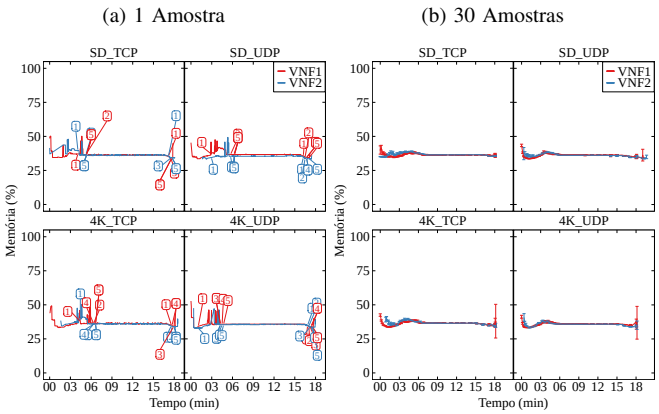

Figura 7. Utilização de Memória - Cenário SC

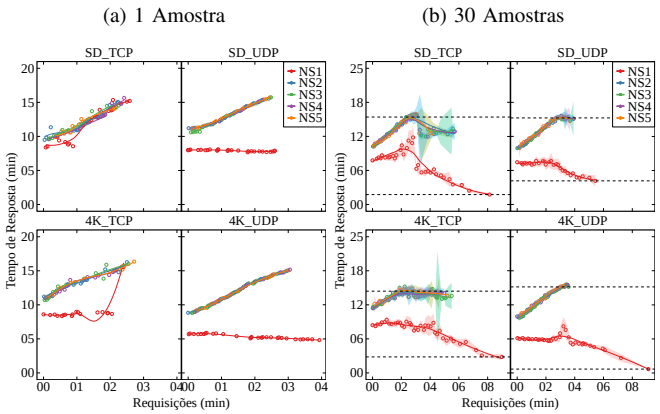

Figura 8. Tempo de Resposta - Cenário SC

resultados se comparado com os demais, já que é o primeiro a ser alocado; (ii) Enquanto as fatias de rede são provisionadas as requisições são enfileiradas para serem atendidas após cada finalização. Portanto, o tempo máximo de aproximadamente 16 minutos que foi obtido na maioria dos resultados pelas fatias de rede NS2 ao NS5, corresponde ao período de provisionamento agregado ao tempo de atendimento; (iii) Como já abordado nas Seções anteriores, o valor relativamente alto de 16 minutos, também está relacionado ao fato da capacidade de transmissão do enlace ser compartilhada, já que se trata de um ambiente de avaliação emulado em um único *host* físico.

Os valores de utilização apresentados, principalmente de CPU, geram compromissos que devem ser discutidos, por exemplo:

1) Mais ou menos 80% de utilização de CPU corresponderia a um valor aceitável ou acarretaria sobrecarga nas VNFs?
2) Até que ponto o uso da elasticidade se faz necessário?
3) Qual o melhor método para realizar elasticidade, horizontal ou vertical?
4) A elasticidade deve ser realizada de forma reativa ou proativa?
5) O tempo para fazer elasticidade pode acarretar em um maior tempo de resposta?

Uma forma de tentar responder tais questionamentos é considerá-los em ambientes reais. Em cenários reais temos de um lado o assinante e do outro o fornecedor de serviços, e as questões levantadas são consideradas de acordo com o que foi contratado por cada assinante, se atendem ou não suas necessidades. Por esse motivo, realizar elasticidade pode ser um trabalho futuro a ser implementado e investigado nos cenários avaliados, utilizando informações contextuais das VNFs por exemplo.

B. Com Cache

Como é possível perceber na Figura 5 e com os resultados preliminares, o *setup* de provisionamento inicial das VNFs pode impactar no tempo de resposta de um determinado serviço. Conforme existe a necessidade de aumentar o número de VNFs e de acordo com o tipo de serviço que será instanciado, esse tempo poderá crescer consideravelmente. Nesta situação, podemos considerar duas abordagens para a redução da latência: aumentar o número de produtores de conteúdos para reduzir o enfileiramento de requisições ou criar uma estratégia de cache de conteúdos. Buscando aproveitar os benefícios proporcionados pela evolução da computação móvel e aprimorados com o conceito de MEC, bem como visando um diferencial em termos de eficácia para a arquitetura proposta neste trabalho, optou-se pela criação de uma estratégia de cache de conteúdos.

Com a inclusão do cache é possível obter ganhos relacionados ao ciclo de vida das VNFs. Como não é mais necessário que o produtor envie o mesmo conteúdo à cada nova requisição, o ciclo de vida das VNFs se torna menor através do gerenciamento realizado pelo módulo *Slice Manager*. Enquanto em experimentos SC, as instâncias ficam ativas em torno de 18 minutos, com CC estas ficam provisionadas por até 14 minutos (conforme os resultados obtidos).

Por outro lado, a utilização de CPU apresentou um aumento significativo de até 52,18% pela VNF1, com a transmissão em SD usando o UDP (Figuras 6b e 9b). Tal aumento está diretamente relacionado ao fato do módulo *Slicing Orchestrator* estar trabalhando ao mesmo tempo que o *Slice Manager*, alocando e desalocando fatias de rede simultaneamente. Já a utilização de memória se manteve constante e não obteve uma diferença expressiva, na comparação com os resultados entre os experimentos SC e CC (Figuras 10a e 10b).

Com a utilização do cache foi possível obter uma redução de até 96,56% no tempo de resposta mínimo, e até 39,84% no tempo de resposta máximo (Figuras 11a e 11b). Além disso, pode-se observar que as menores médias e intervalos de confiança foram obtidos em experimentos usando o UDP. Por não realizar o *handshake*, como o TCP, o UDP consegue se sobressair quando o quesito é tempo de resposta. Apesar do UDP não garantir confiabilidade, ainda assim é amplamente usado quando se trata de fluxos de dados em tempo real ou quando perdas de pacotes são aceitáveis.

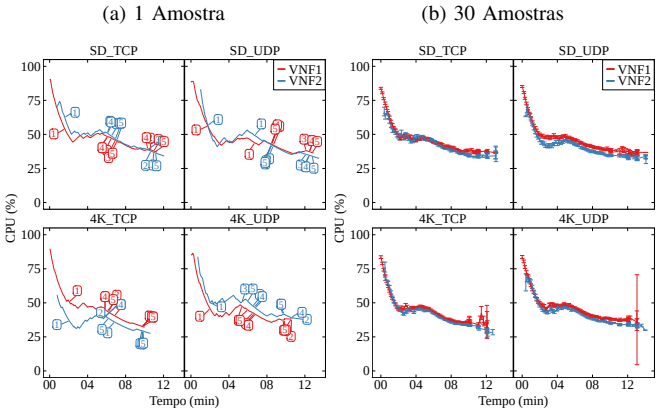

Figura 9. Utilização de CPU - Cenário CC

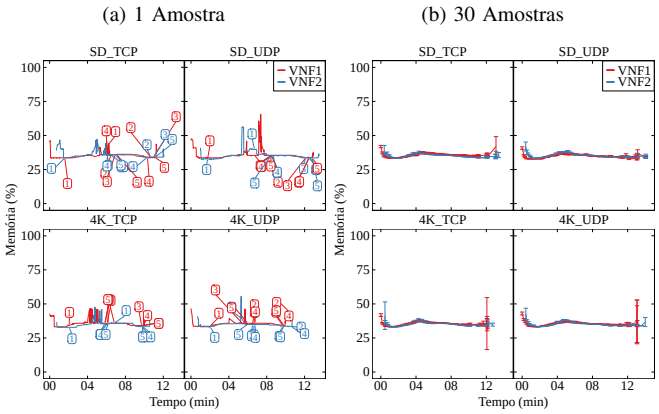

Figura 10. Utilização de Memória - Cenário CC

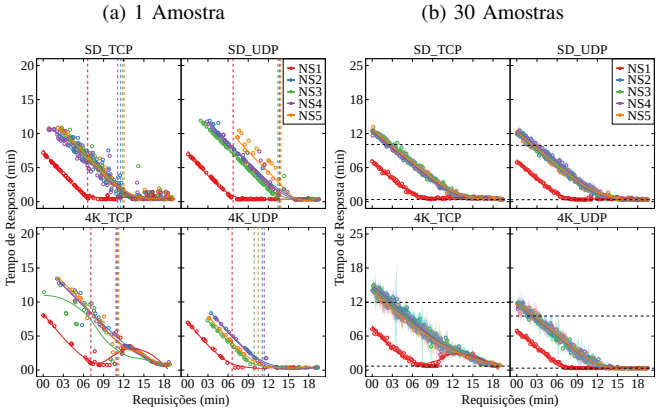

Figura 11. Tempo de Resposta - Cenário CC

VII. Considerações Finais

Neste artigo, concebemos, implementamos e avaliamos uma arquitetura de middleware orientado a serviços que realiza fatiamento de rede no ambiente MEC para Internet de Veículos. Também evidenciamos os benefícios de *caching* de conteúdo. Apesar da melhoria esperada na redução do atraso, desconhecemos que uma solução conjunta, baseada tanto em *caching* quanto em uma arquitetura orientada a serviços para viabilizar fatiamento de rede, tenha sido proposta ou avaliada na literatura para redes veiculares. Como trabalhos futuros, pretendemos avaliar novos cenários com redes heterogêneas e protocolos de transporte (e.g, QUIC) e propor uma solução que leve em conta o *handover* dos veículos e das fatias de rede.

Referências

[1] N. Alliance, "NGMN 5G White Paper," 2016. Disponível em: https://ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf

[2] G. ETSI, "Multi-access Edge Computing (MEC); Framework and Reference Architecture - ETSI GS MEC 003 v2.1.1," 2019. Disponível em: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf

[3] C. Campolo, R. dos Reis Fontes, A. Molinaro, C. Esteve Rothenberg, and A. Iera, "Slicing on the road: Enabling the automotive vertical through 5G network softwarization," *Sensors*, vol. 18, no. 12, p. 4435, 2018.

[4] I. Afolabi, T. Taleb, P. A. Frangoudis, M. Bagaa, and A. Ksentini, "Network Slicing-Based Customization of 5G Mobile Services," *IEEE Network*, vol. 33, no. 5, pp. 134–141, Sep. 2019.

[5] A. H. Celdrán, M. G. Pérez, F. J. Clemente, F. Ippoliti, and G. M. Pérez, "Dynamic network slicing management of multimedia scenarios for future remote healthcare," *Multimedia Tools and Applications*, 2019.

[6] A. H. Celdrán, M. G. Pérez, F. J. G. Clemente, F. Ippoliti, and G. M. Pérez, "Policy-based network slicing management for future mobile communications," in *2018 Fifth International Conference on Software Defined Systems (SDS)*. IEEE, 2018, pp. 153–159.

[7] H.-T. Chien, Y.-D. Lin, C.-L. Lai, and C.-T. Wang, "End-to-end slicing as a service with computing and communication resource allocation for multi-tenant 5G systems," *IEEE Wireless Communications*, vol. 26, no. 5, pp. 104–112, 2019.

[8] J. Mei, X. Wang, and K. Zheng, "Intelligent network slicing for V2X services toward 5G," *IEEE Network*, vol. 33, no. 6, pp. 196–204, 2019.

[9] Z. Mlika and S. Cherkaoui, "Network Slicing with MEC and Deep Reinforcement Learning for the Internet of Vehicles," *IEEE Network*, vol. 35, no. 3, pp. 132–138, 2021.

[10] T. Soenen, R. Banerjee, W. Tavernier, D. Colle, and M. Pickavet, "Demystifying network slicing: From theory to practice," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 1115–1120.

[11] H.-C. Hsieh, C.-S. Lee, and J.-L. Chen, "Mobile edge computing platform with container-based virtualization technology for IoT applications," *Wireless Personal Communications*, vol. 102, no. 1, pp. 527–542, 2018.

[12] M. Voelter, M. Kircher, and U. Zdun, "Remoting Patterns-Foundations of Enterprise, Internet, and Realtime Distributed Object Middleware," 2004.

[13] G. ETSI, "Network Functions Virtualization (NFV): Architectural framework," *ETSI Gs NFV*, vol. 2, no. 2, p. V1, 2013.