VNF-Cache: Um Serviço NFV-COIN de Cache na Rede

Bruno E. Farias, José Flauzino, Elias P. Duarte Jr.

Universidade Federal do Paraná (UFPR), Depto. Informática
Curitiba, PR, Brasil

{bef18,jwvflauzino,elias}@inf.ufpr.br

Resumo. *Com o crescimento exponencial da quantidade de dados disponíveis na Internet, torna-se essencial otimizar o tempo de resposta e o uso dos recursos para acesso aos dados. As caches são uma solução eficaz que aproxima os dados de clientes, dispensando requisições repetitivas nos servidores. Este artigo apresenta a VNF-Cache, um serviço de cache para bancos de dados geograficamente remotos do tipo chave-valor. A VNF-Cache é um serviço NFV-COIN (*Network Function Virtualization-Computing In The Network*), tecnologia em processo de padronização no IETF que possibilita a implementação de serviços arbitrários diretamente na rede. A VNF-Cache intercepta pacotes de rede, trata, armazena e envia valores diretamente para os clientes quando possível. Através de uma implementação para prova de conceito e de experimentos realizados com servidores geograficamente espalhados no Brasil, Estados Unidos e Japão, foram observados tanto redução significativa no tempo de resposta, como aumento na quantidade de requisições processadas por segundo.*

1. Introdução

As caches ainda são uma das principais tecnologias utilizadas para melhorar a eficiência do acesso a dados através da Internet. Este trabalho propõe a VNF-Cache, um serviço de cache para bancos de dados chave-valor baseado na tecnologia de Virtualização de Funções de Rede (*Network Function Virtualization* - NFV). A tecnologia NFV possibilita a implementação de funções de rede em software que pode ser executado em infraestruturas virtualizadas [ETSI 2012]. Como a gama de funções de rede que podem ser implementadas utilizando NFV é muito variada, a arquitetura de referência NFV-MANO (NFV - *MANagement and Orchestration*) surgiu para padronizar as implementações e possibilitar a interoperabilidade entre os sistemas NFV [ETSI 2021]. Para isto, a arquitetura define o gerenciamento do ciclo de vida das *Virtualized Network Functions* (VNFs), que podem inclusive ser disponbilizadas através de marketplaces na Internet [Bondan et al. 2019].

Recentemente, a arquitetura NFV-COIN (NFV - *COmputing In the Network*) foi proposta para possibilitar a implementação de serviços arbitrários e inovadores diretamente na rede, através do paradigma chamado *COmputing In the Network* (COIN) [Venâncio et al. 2022]. A NFV-COIN está no processo de padronização pelo IETF, que já publicou dois *Drafts* com foco nas interfaces para as funções na rede: o primeiro especifica o *problem statement* [Jeong et al. 2025b] e o segundo especifica o *framework* [Jeong et al. 2025a].

A VNF-Cache apresentada neste trabalho é um serviço NFV-COIN. Ela tem como propósito agilizar o acesso de clientes a bancos de dados chave-valor geograficamente distantes. Ao aproximar dados de clientes, a VNF-Cache tem por objetivos reduzir o

tempo para a execução de requisições e aumentar a taxa de requisições executadas por unidade de tempo. Além disso, ao evitar que pacotes de requisição e resposta percorram todo o caminho até o servidor, a VNF-Cache promove um melhor uso da infraestrutura da rede.

Para desempenhar a funcionalidade de *caching* dos conjuntos chave-valor, a VNF-Cache deve estar localizada em algum ponto da rede entre os clientes e os servidores de banco de dados chave-valor. Seu funcionamento inicia com a filtragem de pacotes de rede de interesse, seguida de processamento e armazenamento dos valores das chaves requisitadas pelos clientes. Sempre que possível, a VNF-Cache retorna dados requisitados diretamente para o cliente, caso contrário redireciona o pacote para o servidor e, quando viável, armazena o valores retornados pelo servidor para requisições futuras.

A arquitetura da VNF-Cache é apresentada, bem como um protótipo implementado como prova de conceito. Através da avaliação empírica, foram efetuados experimentos com três cenários diferentes: (A) cliente, VNF-Cache e servidor de banco de dados chave-valor próximos entre si, (B) cliente e VNF-Cache próximos entre si, distanciando o servidor e (C) cliente, VNF-Cache e servidor distantes entre si. Os servidores e VNF-Cache remotos foram instanciados na *Amazon Elastic Compute Cloud*[1], serviço de computação em nuvem da *Amazon Web Services* (AWS)[2]. Foram utilizadas máquinas virtuais para a VNF-Cache e o servidor de banco de dados tanto em Curitiba e São Paulo, no Brasil; Ohio, na costa leste dos Estados Unidos, e em Tóquio, Japão. Os resultados destes experimentos apontam que quando clientes e servidores estão distantes entre si, a utilização de uma VNF-Cache é capaz de reduzir consideravelmente o tempo de resposta das requisições e aumentar o número de requisições processadas por segundo.

O restante deste trabalho está organizado da seguinte maneira. A seção 2 apresenta uma visão geral da tecnologia NFV e NFV-COIN, bem como trabalhos relacionados. A Seção 3 descreve a VNF-Cache, apresentando sua funcionalidade e arquitetura. O protótipo implementado, bem como os resultados experimentais estão na Seção 4. Por fim, a Seção 5 traz as conclusões e lista trabalhos futuros.

2. NFV, NFV-COIN & Trabalhos Relacionados

A Virtualização de Funções de Redes, ou NFV, é uma alternativa concreta para implementar serviços de redes utilizando técnicas de virtualização, como *Virtual Machines* (VMs) e *containers*, que podem ser executadas em hardware de prateleira. Uma grande variedade de serviços de rede podem ser implementados com NFV, como roteadores, *Virtual Private Network* (VPN), análisadores de tráfego, *firewalls*, *Content Delivery Network* (CDN), entre outros [ETSI 2012].

Um dos principais resultados destes esforços para estabelecer padrões e desenvolver uma arquitetura para gerenciar e padronizar a implantação das funções de rede virtualizadas é o modelo *MANagement and Orchestration* (MANO). O MANO surgiu como arquitetura para permitir interoperabilidade entre os sistemas NFV.

A arquitetura NFV-MANO é composta basicamente por três blocos fundamentais e interdependentes: o *Virtualized Infrastructure Manager* (VIM), o NFV *Orchestrator*

[1]https://aws.amazon.com/pt/ec2/?nc2=h_ql_prod_cp_ec2

[2]https://aws.amazon.com/pt/

(NFVO) e o VNF *Manager* (VNFM) [ETSI 2021]. O ciclo de vida das VNFs é gerenciado pelo VNFM e são executadas em infraestruturas de virtualização de funções de redes (os NFVIs). Estas, por sua vez, são gerenciadas pelos VIMs, que são integrados e gerenciados pelo NFVO. Entre as principais atribuições do NFVO, estão inclusos a distribuição de todos os recursos computacionais entre os VIMs [Fulber-Garcia et al. 2024] e também o gerenciamento dos serviços de rede virtualizados. Todos estes elementos geralmente são disponibilizados através das chamadas Plataformas NFV [Tacker 2025, ETSI 2025, Flauzino et al. 2021].

Além de possibilitar a concepção de VNFs básicas com funcionalidades diversas, é possível ainda realizar a composição de VNFs individuais em serviços complexos, formando assim uma *Service Function Chain* (SFC) [Halpern et al. 2015, Fulber-Garcia et al. 2020]. Cada SFC pode ser definida em um escopo centralizado, ou até mesmo abrangendo múltiplos sistemas autônomos, nuvens e orquestradores [Huff et al. 2020]. A arquitetura NFV-COIN [Venâncio et al. 2022] amplia este escopo para serviços arbitrários e inovadores diretamente na rede, através do paradigma *COmputing In the Network* (COIN). Por ser baseada em software e virtualização, a tecnologia NFV-COIN apresenta grande flexibilidade para a implantação de novos recursos nativos das redes. A interface para funções NFV-COIN é realizada através de APIs padronizadas, que permitem seu uso inclusive por usuários finais. Além disso, há um módulo de gerenciamento para implantação, configuração e monitoramento dos serviços. Diversos serviços NFV-COIN já foram propostos, como detectores de falhas de processos [Turchetti and Duarte 2015], consenso [Venâncio et al. 2021] além da difusão confiável e ordenada de mensagens na rede [Venâncio et al. 2019].

Trabalhos Relacionados: NFV & Caches

Nos últimos anos, vários trabalhos foram desenvolvidos envolvendo a união de NFV e de caches. [Zhuang et al. 2019], por exemplo, discutem sobre a possibilidade de se aplicar caches baseadas em NFV para minimizar o tempo de recuperação de conteúdos em sistemas de *Internet-of-Vehicles* (IoV). Para os autores, o uso deste tipo de cache pode facilitar a implantação dos serviços e a disseminação de seus conteúdos, possibilitando uma melhor confiabilidade e eficiência dos serviços IoV.

Já [Clayman et al. 2018] propõem uma arquitetura para *streaming* de vídeo *Server and Network Assisted Dynamic Adaptative Streaming over HTTP* (SAND). Nesta arquitetura, instâncias de caches virtualizadas são criadas conforme a demanda por conteúdo. Além disso, os autores também discorrem sobre os posicionamentos dessas instâncias no grafo da rede, baseando-se em características como a largura de banda dos caminhos, os locais e o número de clientes da rede.

Outro trabalho relevante é o de [Liu et al. 2017], que discutem o ganho de desempenho notável que a aplicação de caches NFV em redes 5G pode causar. Além da flexibilidade, dinamicidade e escalabilidade possibilitadas pelo uso de NFV, os autores ainda destacam a possibilidade de oferecer serviços de cache para provedores de serviços e para operadoras de rede utilizando a mesma infraestrutura.

3. Um Serviço de Cache na Rede para Bancos de Dados Chave-Valor

Esta seção apresenta a VNF-Cache, um serviço de cache voltado para bases de dados do tipo *Key-Value Store* (KVS), ou banco de dados chave-valor. Este é um tipo de banco de

dados não-relacional que realiza a persistência dos dados através da associação de uma única chave para cada dado armazenado [Seeger 2009]. O uso deste tipo de banco de dados permite ao desenvolvedor da aplicação o armazenamento dos dados sem o uso de esquemas, ou seja, sem o tradicional método relacional de linhas e colunas pré-definidas. Desta forma, a flexibilidade no projeto do banco de dados é maior, assim como a qualidade do código de programação correspondente.

Neste sentido, a VNF-Cache visa aproximar os dados do cliente, armazenando as informações diretamente na rede e em locais mais próximos. A VNF-Cache deve estar localizada no caminho entre o cliente e o servidor remoto, realizando o processamento dos pacotes e armazenando os valores das chaves requisitadas pelos clientes. Caso uma chave requisitada esteja válida nesta cache, seu valor é retornado diretamente para o cliente, dispensando a necessidade de reencaminhar os pacotes para o servidor. A Figura 1 ilustra a sequência de passos do funcionamento da VNF-Cache.

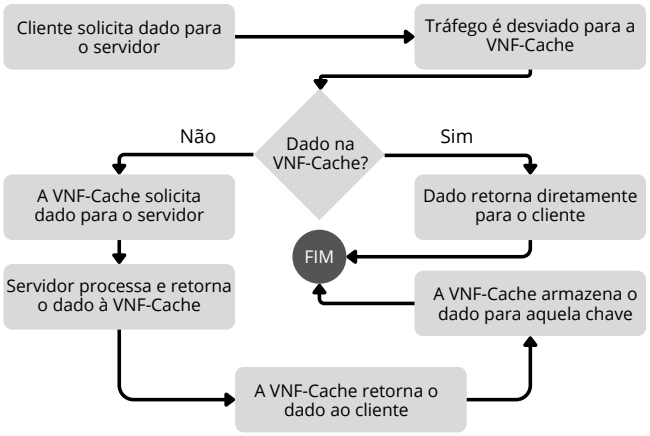

Figura 1. Funcionamento da VNF-Cache.

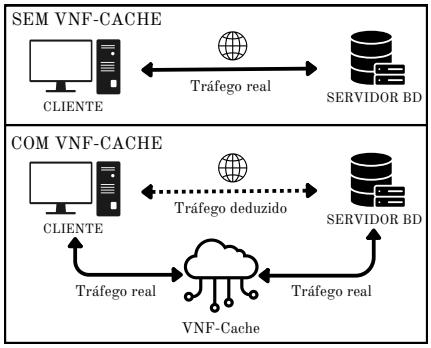

Figura 2. Exemplo de arquitetura de rede sem e com VNF-Cache.

A Figura 2 mostra a rede com e sem a integração de uma VNF-Cache. Na figura, o tráfego deduzido é definido como aquele em que o cliente deduz estar causando na rede, que pode ser diferente do tráfego real presenciado na rede.

A Arquitetura da VNF-Cache

A arquitetura da VNF-Cache é composta por um conjunto de três módulos: VNF-Cache *Filter*, um VNF-Cache *Manager* e o VNF-Cache *Storage*, ilustrados na Figura 3 e descritos a seguir. O VNF-Cache *Filter*, ou apenas *Filter*, é o módulo responsável pela filtragem dos pacotes recebidos pela VNF-Cache, sejam eles enviados pelos clientes (*Client Filter*) ou pelo servidor (*Server Filter*). Estes dois submódulos recebem e filtram os pacotes de rede em três possíveis fluxos: o *Manipulation Flow* (MF), o *Response Flow* (RF) e o *Coordination Flow* (CF), descritos a seguir.

O fluxo de manipulação de dados (MF) é composto pelos pacotes enviados pelos clientes que contenham operações de manipulação de dados, tais como buscas, inserções, atualizações e exclusões. Já o fluxo de respostas RF é composto pelos pacotes enviados pelos servidores e que são respostas para estes pacotes enviados pelos clientes no fluxo MF. Por fim, o fluxo de coordenação (CF) é composto pelos demais pacotes trafegados,

ou seja, aqueles enviados pelos clientes ou pelos servidores e que possuem outros objetivos, como manter a conexão entre clientes e servidores ou realizar o monitoramento de disponibilidade do servidor.

Desta forma, o *Client Filter* é responsável pela filtragem dos pacotes originados pelo cliente, separando-os entre o fluxo MF, que são os pacotes que comunicam as manipulações de dados solicitadas pelos clientes ao banco, e o fluxo CF, que são os pacotes que realizam as comunicações básicas entre cliente e servidor, neste caso no sentido do cliente para o servidor. De forma semelhante, o *Server Filter* filtra os pacotes vindos do servidor, separando-os entre o fluxo RF, que são os pacotes que respondem as solicitações realizadas pelos clientes no fluxo MF, e o fluxo CF, que neste caso é no sentido do servidor para o cliente.

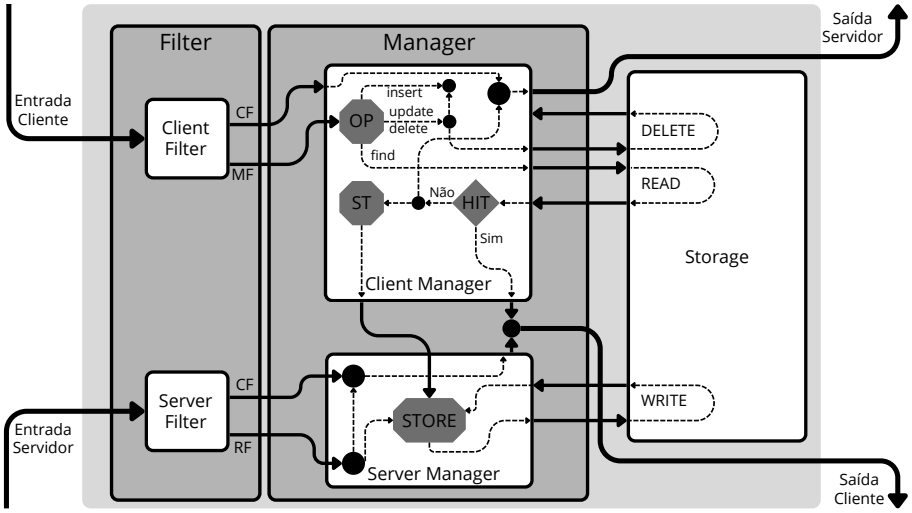

Figura 3. Arquitetura da VNF-Cache.

Por sua vez, o *Manager* é o principal módulo de gerenciamento da VNF-Cache. De forma semelhante ao *Filter*, o *Manager* também é composto por dois submódulos: o *Client Manager* e o *Server Manager*. O *Client Manager* recebe os dois fluxos de pacotes do *Client Filter* e faz o tratamento conforme necessário: o fluxo MF é tratado diretamente com o módulo de armazenamento VNF-Cache *Storage* (que será apresentado a seguir), realizando as leituras, atualizações e exclusões dos dados conforme as operações. Já o fluxo CF é enviado diretamente para a saída da VNF-Cache com destino ao servidor. De forma semelhante, o *Server Manager* recebe os dois fluxos de pacotes do *Server Filter*. Porém, no *Server Manager*, todos os pacotes são enviados para a saída com destino ao cliente, independentemente do fluxo designado pelo *Server Filter*. A diferença entre o processamento dos fluxos no *Server Manager* é que enquanto o fluxo CF é apenas redirecionado para os clientes, os pacotes do fluxo RF passam por um processamento extra, tendo como objetivo o armazenamento dos dados retornados pelo servidor no VNF-Cache *Storage*, conforme as solicitações do *Client Manager*.

Por fim, o módulo *Storage* é o responsável pelo armazenamento das chaves e de seus respectivos valores. Este módulo possui duas funcionalidades principais: (i) retornar o valor de uma chave requisitada pelo *Client Manager* e (ii) armazenar o valor de uma chave capturada pelo *Server Manager* e solicitada pelo *Client Manager*.

Os dois principais fluxos na arquitetura da VNF-Cache ocorrem após a requisição de consulta de uma chave por algum cliente. Quando um cliente solicita uma chave, o *Client Filter* realiza a filtragem deste pacote no fluxo de manipulação de dados, o MF. Em seguida, o *Client Manager* determina o tipo da operação, que no caso é *find*, e faz uma requisição ao *Storage* pelo valor daquela chave. Após o retorno do módulo de armazenamento, podemos ter um cache *miss* ou um cache *hit*. Caso a requisição resulte em um cache *hit*, o *Client Manager* retorna o pacote diretamente para o cliente. Por outro lado, caso a requisição seja um cache *miss*, o *Client Manager* redireciona o pacote para o servidor e envia um sinal ao *Server Manager*, alertando-o que a chave requisitada não está na cache e que seu valor deve, na medida do possível, ser armazenado no *Storage* após a resposta do servidor.

4. Implementação e Resultados Experimentais

A VNF-Cache foi implementada na linguagem Python[3], incluindo diversas das suas bibliotecas como *Scapy* e *PyShark*. Foi utilizado o MongoDB[4], um banco de dados orientado a documentos *JavaScript Object Notation* (JSON). Embora este não seja um banco de dados exclusivamente chave-valor, ele pode ser utilizado como tal ao armazenar os dados em forma de documentos flexíveis. Cada item dos dados é atrelado a um único "índice" gerado automaticamente, que é a chave no contexto chave-valor. É importante destacar que a VNF-Cache pode ser facilmente adaptada para ser utilizada com outros bancos de dados chave-valor, como o Redis, por exemplo. Para realizar a comunicação entre os clientes e os servidores de bancos de dados MongoDB, a biblioteca utilizada foi a PyMongo[5].

Para realizar a comunicação dos clientes com o servidor de banco de dados MongoDB, a biblioteca PyMongo utiliza pacotes de rede com o protocolo de rede IP e transporte TCP. A VNF-Cache filtra apenas pacotes com identificador de operação 2013, que é o tipo padrão para requisições de busca, inserção, exclusão e alteração do PyMongo. Os pacotes do PyMongo possuem um cabeçalho padrão com 25 bytes de tamanho, sendo separados em 7 campos, descritos a seguir.

O primeiro campo (4 bytes) é o ***length***, que contém o tamanho total do pacote TCP. O segundo campo (4 bytes) é o ***request_id***, que contém um identificador único da requisição (o mesmo identificador deve estar presente no pacote de resposta). O terceiro campo (4 bytes) é o ***response_to***, que contém o identificador ao qual aquele pacote se refere (caso este seja a resposta para algum outro pacote enviado anteriormente). O quarto campo (4 bytes) é o ***op_code***, que contém o número de identificação da operação (neste caso focamos apenas na operação de código 2013 – fluxo de manipulação). O quinto campo (4 bytes) contém algumas ***flags*** para comunicação entre PyMongo e MongoDB. O sexto campo (1 byte) é o ***payload_type*** e contém o tipo do conteúdo do pacote PyMongo. Por fim, o sétimo campo (4 bytes) é o ***payload_size***, que contém o tamanho total do documento *Binary JSON* (BSON) contido naquele pacote.

Ao unirmos o conteúdo em binário do ***payload_size*** e o restante do *payload* do pacote, podemos obter o JSON completo que foi enviado pelo PyMongo utilizando a classe ***RawBSONDocument*** da biblioteca ***bson***. Desta forma, foram viabilizados os devidos

[3]https://www.python.org

[4]https://www.mongodb.com

[5]https://pymongo.readthedocs.io/en/stable/index.html

tratamentos dos pacotes dentro da implementação da VNF-Cache, facilitando e tornando eficiente o monitoramento e a filtragem dos pacotes que são de fato importantes. Através dos campos do cabeçalho do pacote PyMongo, a VNF-Cache pode tomar as decisões corretas para cada evento, como reenviar o pacote para o servidor e armazenar a resposta, retornar o valor armazenado diretamente para o cliente, aplicar as políticas de preenchimento e substituição, entre outras.

Na implementação da VNF-Cache, a cache propriamente dita é armazenada em um único arquivo Python. Seu funcionamento é exatamente como o proposto anteriormente, ou seja, o tráfego de pacotes de rede com destino ao servidor de banco de dados é desviado para uma porta específica da cache. Esta, por sua vez, analisa os pacotes recebidos e faz os devidos tratamentos de acordo com a necessidade de cada requisição. Este desvio dos pacotes de rede foi realizado em roteadores de alto desempenho e só pôde ser concretizado devido a uma recente proposta de classificação de pacotes através de roteamento diretamente no plano de controle da rede [Flauzino et al. 2024].

Para realizar o monitoramento dos pacotes recebidos pela cache, um *socket* da biblioteca padrão do Python é aberto na porta especificada e aguarda por requisições de estabelecimento de conexão pelos clientes. Quando este *socket* recebe um pedido de conexão de algum cliente, uma nova *thread* é criada. Em cada *thread* aberta, um novo *socket* é criado para estabelecer a comunicação direta entre a VNF-Cache e o servidor MongoDB. Após o estabelecimento das conexões entre cliente e cache, e cache e servidor, a VNF-Cache aguarda pelos pacotes que serão enviados pelo cliente.

Quando recebe um pacote, o módulo *Filter* lê o cabeçalho e separa pacotes com *op_code* 2013 e que contenham operações de busca, inserção, atualização e exclusão. Outros pacotes são reenviados diretamente para o servidor MongoDB ou para o cliente, como, por exemplo, os pacotes de estabelecimento de conexão do PyMongo, de estatística e de monitoramento de disponibilidade.

Após capturar um pacote com o código de operação 2013 (fluxo de manipulação), a VNF-Cache realiza sua decodificação, identificando a operação de manipulação e em quais dados a manipulação será executada. É necessário determinar o método que está sendo utilizado, qual a chave sendo requisitada e, se for a primeira requisição de uma chave, qual é o valor retornado pelo servidor para aquela chave. Para isso, é realizada a reconstrução do documento JSON que foi enviado pelo PyMongo. Assim, o módulo *Manager* realiza o processamento dos pacotes que contêm operações de buscas, inserções, alterações ou exclusões de dados. As operações de busca de dados possuem o termo ***find*** como chave e a coleção da busca como valor correspondente. De forma semelhante, as operações de inserção, atualização e exclusão possuem os termos *insert*, *update* e *delete*, respectivamente.

Por fim, a chave do dado que está sendo manipulado está presente no campo *filter* do JSON. De acordo com a documentação do MongoDB, através do campo *filter* é possível realizar diferentes combinações, como busca por chaves iguais, maiores ou menores do que um inteiro (caso a chave seja um inteiro), por igualdade ou existência de uma *string* dentro de outra, conjunções, disjunções, entre outros. Para simplificar esta implementação, o foco foi apenas nas operações com chaves únicas. Portanto, nesta implementação da VNF-Cache, a chave da requisição é o valor que está no campo "$eq",

que está localizado no campo *"filter"*. Opcionalmente, pode-se omitir o campo "$eq", deixando a chave buscada ser diretamente o valor referente à chave daquela coleção. Vale destacar que as operações de inserção de dados não são tratadas internamente pela VNF-Cache, já que não influenciam diretamente os dados já armazenados nela.

Ao encontrar um pacote de busca/leitura de uma chave específica, a VNF-Cache verifica primeiramente se este conjunto chave-valor já está no armazenamento local da cache. Na implementação da VNF-Cache, o módulo *Storage* de armazenamento dos dados é realizado em um dicionário Python, utilizando o mesmo par chave-valor do MongoDB. Se a chave requisitada não estiver no dicionário, o *Client Manager* reencaminha o pacote para o servidor MongoDB e aguarda pelo pacote de resposta, e armazena o valor quando retornar. Por outro lado, se a chave requisitada estiver no dicionário, o *Client Manager* reconstrói o pacote de dados e o encaminha diretamente ao cliente. Desta forma, o pacote de requisição da chave enviado pelo cliente sofre um *drop*, ou seja, o pacote é ignorado e não é reencaminhado para o servidor.

A VNF-Cache implementada utiliza a política de *caching Write-Invalidate* [Jacob et al. 2008]. Desta forma, as operações de alteração e exclusão dos dados fazem com que os mesmos sejam retirados do *Storage*, como uma forma de invalidar os dados. Assim, ao receber um pacote que atualiza o dado de uma chave, o respectivo conjunto chave-valor é retirado da cache e o pacote é reencaminhado para o servidor, que realiza as alterações necessárias no banco de dados.

Como existe a possibilidade de múltiplas *threads* estarem em execução ao mesmo tempo e solicitarem leituras e/ou escritas no dicionário da cache, existe a possibilidade de duas ou mais *threads* realizarem modificações no mesmo dado ao mesmo tempo, podendo causar incoerências nas respostas. Para resolver este problema, são utilizadas as primitivas ***acquire()***, que trava o acesso ao dicionário exclusivamente para aquela *thread*, e ***release()***, que libera o acesso para as demais *threads*.

De forma complementar ao funcionamento básico da VNF-Cache, a implementação tem opções de linha de comando para definir os níveis de detalhamento do *log* produzido, o número máximo de itens da VNF-Cache e a geração de arquivos estatísticos, como os de registros de cache *hit* e *miss*, por exemplo.

4.1. Avaliação Empírica

Foram efetuados experimentos com diferentes cenários de aplicações da VNF-Cache, variando sua capacidade de armazenamento e a sua localização em relação ao cliente e ao servidor. As métricas avaliadas incluem o impacto da VNF-Cache no tempo de resposta de uma requisição, que compreende o intervalo entre o envio da requisição pelo cliente e a chegada da resposta enviada pelo servidor, e a quantidade de requisições processadas por unidade de tempo. Foram definidos três cenários diferentes para os experimentos: (A) cliente, VNF-Cache e servidor de banco de dados próximos entre si; (B) cliente e VNF-Cache próximos entre si, e o servidor distante; e (C) cliente, VNF-Cache e servidor distantes entre si. Desta forma, é possível analisar a eficiência das diferentes aplicações da VNF-Cache conforme a distância entre os clientes e os servidores varia.

Para a execução dos experimentos, foram utilizadas uma máquina física e múltiplas combinações de máquinas virtuais, conforme será descrito adiante. A máquina física possui um processador Intel(R) Core(TM) i5-7400 @3.0 GHz x 4, 16 GB de

memória RAM, uma interface de rede de 100 Mb/s e sistema operacional *Ubuntu 20.04.6*. Esta máquina serviu para a coordenação dos testes e execução de algumas das máquinas virtuais. Estas, por sua vez, foram instanciadas tanto localmente utilizando *Kernel-based Virtual Machine* (KVM), quanto remotamente através da *Amazon Elastic Compute Cloud*[6], um serviço de computação em nuvem da *Amazon Web Services*[7] (AWS) que possibilita a instanciação de máquinas virtuais em diferentes localizações do mundo.

Cada experimento consistiu de um cliente enviando 30 lotes de 1000 requisições para chaves inteiras aleatórias, distribuídas de forma uniforme no intervalo de 1 a 100 da coleção *phrases* do banco de dados *randomPhrases* do MongoDB, que é constituído por frases aleatórias. Além disso, estes também realizam a medição do tempo de resposta de cada requisição e do número de requisições processadas por segundo.

Por sua vez, a VNF-Cache foi executada em máquinas virtuais com duas opções de especificações. As máquinas virtuais executadas na máquina física utilizam o sistema operacional *Ubuntu 20.04*, em um processador virtualizado de 3 GHz x 2, memória principal de 2 GB e 15 GB de armazenamento em disco. Já na AWS, as máquinas virtuais executam o mesmo sistema operacional, porém sobre um processador virtualizado de 2.5 GHz x 1, memória principal de 1 GB e 8 GB de armazenamento em disco. Por fim, o servidor de banco de dados MongoDB foi implementado em máquinas virtuais com *Ubuntu Server 20.04*, processador de 1 GHz (no KVM) ou de 2,5 GHz (na AWS), 1 GB de memória RAM e 10 GB de armazenamento em disco.

Experimento 1: Cliente, Servidor e VNF-Cache Próximos

O primeiro experimento foi realizado com cliente, VNF-Cache e servidor de banco de dados próximos entre si. Para isso, três máquinas virtuais foram instanciadas na mesma máquina física. Além disso, foram implementadas políticas de redirecionamento no roteador da rede criada para intercomunicação das VMs. Desta forma, todo pacote do cliente com destino ao banco de dados do servidor é desviado para uma porta específica da VM que executa a VNF-Cache. A Tabela 1 mostra os tempos de resposta, em milissegundos (ms), das requisições para a mesma chave nos experimentos sem e com a VNF-Cache.

Tabela 1. Tempos das requisições (em ms) para a mesma chave e *overhead* causado pela VNF-Cache no cenário A.

	Ordem das Requisições para uma Determinada Chave						
VNF-Cache	1ª	2ª	3ª	4ª	5ª	6ª	...
S/ VNF-Cache	1,12 ms	1,56 ms	2,04 ms	1,26 ms	1,17 ms	1,35 ms	...
C/ VNF-Cache	10,54 ms	5,40 ms	3,87 ms	3,32 ms	4,05 ms	3,93 ms	...
Overhead	9,42 ms	3,84 ms	1,83 ms	2,06 ms	2,88 ms	2,58 ms	...

A tabela mostra que, sem utilizar a VNF-Cache, o tempo de resposta médio para cada requisição (considerando as primeiras 6 requisições) variou entre 1,12 e 2,04 milissegundos (ms). O tempo médio foi de 1,41 ms, com uma taxa de 535 requisições por segundo. A VNF-Cache foi então habilitada com capacidade máxima de 100 conjuntos

[6]https://aws.amazon.com/pt/ec2/?nc2=h_ql_prod_cp_ec2

[7]https://aws.amazon.com/pt/

chave-valor (ou seja, todo o espaço de chaves do banco de dados). Desta vez, considerando o preenchimento da cache, o tempo de resposta médio foi de 5,18 ms, ou seja, aproximadamente 3,5 vezes mais demorado. Já o número médio de requisições por segundo foi de 191, ou seja, uma queda de quase 65%.

Estes resultados possibilitam concluir que em um cenário de proximidade entre cliente e servidor, a VNF-Cache não atinge o seu objetivo de redução do tempo de resposta. Neste caso, isto acontece pois a VNF-Cache apenas adiciona um processamento extra no percurso dos pacotes de rede que, ao invés de trafegarem diretamente entre cliente e servidor, precisam passar pela VNF-Cache antes de chegarem aos seus destinos finais. Como neste caso o tempo de resposta das requisições diretas entre o cliente e servidor já é baixo, o *overhead* causado pela VNF-Cache, não é capaz de justificar sua implementação neste cenário. É possível perceber pela Tabela 1 que na primeira requisição pela chave, quando ocorre um cache *miss* e a VNF-Cache necessita requisitar o servidor, o *overhead* médio causado é de 9,42 ms.

Experimento 2: Cliente e VNF-Cache Próximos, Servidor Distante

Em seguida, os experimentos afastando geograficamente o servidor de banco de dados do cliente e da VNF-Cache foram realizados. Para isso, foram instanciadas na AWS duas máquinas virtuais para o servidor de banco de dados, sendo a primeira delas em Ohio, na costa leste dos Estados Unidos, e a segunda em Tóquio, Japão. O cliente e a VNF-Cache foram executados em máquinas virtuais no KVM da máquina física em Curitiba. A Tabela 2 mostra os tempos de acesso para cada combinação de localizações e capacidades da VNF-Cache. Para uma melhor análise do desempenho da implementação, nas execuções deste cenário variou-se também a capacidade da VNF-Cache entre 10, 30, 70 e 100 conjuntos chave-valor.

Nos experimentos sem utilização da VNF-Cache e com o servidor em Ohio, o tempo médio das requisições diretas entre cliente e servidor foi de cerca de 164 ms, com cerca de 6 requisições processadas por segundo. Já nos experimentos com o servidor em Tóquio, o tempo médio das requisições diretas entre cliente e servidor foi de cerca de 292 ms, com cerca de 3,3 requisições processadas por segundo.

Tabela 2. Tempos médios das requisições (em ms) para cada capacidade da VNF-Cache local e posicionamento do servidor de banco de dados chave-valor.

	Sem	Capacidade da VNF-Cache (em pares chave-valor)			
Localização	VNF-Cache	10	30	70	100
OHIO	164 ms	174,48 ms	138,51 ms	64,66 ms	8,08 ms
JAPÃO	292 ms	303,34 ms	239,35 ms	112,66 ms	11,02 ms

Neste experimento o impacto da VNF-Cache é muito positivo: com capacidade de 100 conjuntos chave-valor, cliente e VNF-Cache em Curitiba e o servidor em Ohio o tempo médio de resposta foi cerca de 8 ms com uma média de 118 requisições processadas por segundo. Já com o servidor em Tóquio, os resultados são ainda mais expressivos: o tempo das requisições caiu de 292 ms (sem VNF-Cache) para uma média de 11,02 ms e o número de requisições processadas por segundo aumentou de 3,3 para uma média de

87. As melhoras são de até aproximadamente 95% com o servidor em Ohio e 96% com o servidor em Tóquio.

A Tabela 2 também mostra o impacto da capacidade da VNF-Cache local em comparação ao espaço de chaves possíveis do banco de dados. Por exemplo, nos experimentos realizados com a cache com capacidade de apenas 10 conjuntos chave-valor, é observado um pequeno aumento no tempo médio de resposta das requisições. Com o servidor em Tóquio e a VNF-Cache de 10 posições na mesma localização do cliente, o tempo médio chegou a piorar em cerca de 3,5%. Já com o servidor localizado em Ohio e a VNF-Cache com a mesma capacidade, o tempo médio de resposta das requisições piorou em cerca de 6%. Ou seja, nestes experimentos da VNF-Cache, caso a capacidade da cache seja muito pequena, o tempo de resposta médio tende a ser pior se comparado ao tempo das requisições diretas.

Experimento 3: Cliente, VNF-Cache e Servidor Distantes

Experimentos afastando a VNF-Cache do cliente também foram realizados. O cliente foi executado em Curitiba, a VNF-Cache em uma máquina virtual na AWS em São Paulo. A ideia é avaliar o benefício de uma cache no país para o acesso de clientes locais a servidores no hemisfério norte. A Tabela 3 mostra que para a VNF-Cache com capacidade de 100 conjuntos chave-valor, o tempo médio das requisições diminuiu em cerca de 87% com o servidor em Ohio e em cerca de 92% com o servidor em Tóquio. Já o número de requisições por segundo aumentou para uma média de 45 para Ohio (no acesso direto são apenas 6 requisições por segundo) e para uma média de 40 em Tóquio (no acesso direto, são apenas 3,3 requisições por segundo).

Outro ponto relevante apresentado na Tabela 3 é o fato de a VNF-Cache já apresentar um resultado benéfico mesmo com a baixa capacidade de 10 conjuntos chave-valor. Neste caso, o tempo de resposta das requisições reduziu em cerca de 17% com o servidor em Ohio e cerca de 12% com o servidor em Tóquio. Esse resultado é melhor que o do Experimento 1, e acreditamos que se deve às limitações da máquina física que executa as máquinas virtuais KVM para cliente e VNF-Cache.

Tabela 3. Tempos médios das requisições para cada capacidade da VNF-Cache em SP e posicionamento do servidor de banco de dados chave-valor.

	Sem	Capacidade da VNF-Cache (em pares chave-valor)			
Localização	VNF-Cache	10	30	70	100
OHIO	164 ms	134,56 ms	108,92 ms	57,88 ms	21,13 ms
JAPÃO	292 ms	257,08 ms	204,31 ms	100,50 ms	22,13 ms

A Figura 4 mostra a quantidade de requisições processadas por segundo durante os primeiros 60 segundos da execução dos lotes de requisições com a VNF-Cache em São Paulo e o servidor em Ohio. Os resultados apontam que, durante o experimento das requisições diretas entre o cliente e o servidor, a média de requisições processadas por segundo se manteve relativamente estável, variando entre 5 e 6. Já com a VNF-Cache em São Paulo e capacidade de 100 conjuntos chave-valor, após algum tempo de envio das requisições dos lotes, a média aumentou consideravelmente, ultrapassando em alguns

momentos a marca de 50 requisições processadas por segundo. Este tempo no qual a média de requisições processadas por segundo ainda é baixa pode ser associado ao tempo de *warm-up* da VNF-Cache, ou seja, o tempo que demora para ela armazenar localmente as cópias dos valores das chaves. Desta forma, quando a cache está cheia, mais valores poderão ser retornados rapidamente para o cliente, aumentando o número de requisições processadas por segundo. Através da figura nota-se ainda que, conforme a capacidade da VNF-Cache aumenta, maior é a quantidade de requisições processadas por segundo.

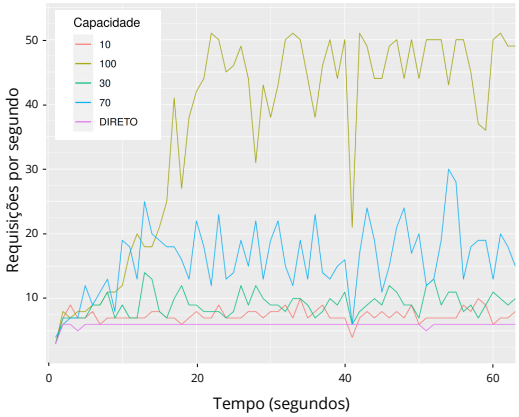

Figura 4. Processamento de requisições.

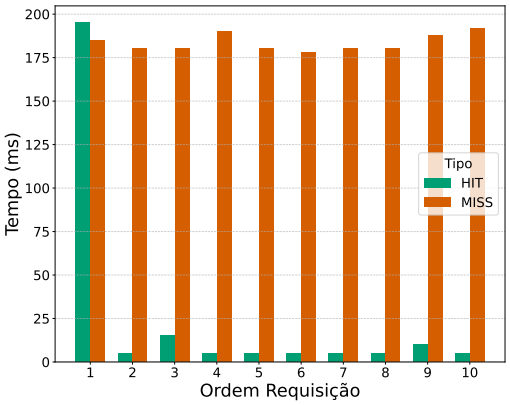

Figura 5. Cache *hit* vs. cache *miss*.

Por fim, considerando o servidor instanciado em Ohio, a Figura 5 demonstra a diferença do tempo de resposta quando a consulta de uma chave na VNF-Cache resulta em cache *miss* e quando resulta em cache *hit*. As barras de cor laranja mostram o tempo de resposta para as 10 primeiras requisições por uma chave que não foi armazenada na cache, enquanto as barras verdes representam o caso em que a chave foi armazenada após a primeira requisição. Para as requisições sem armazenamento em cache, o tempo médio foi de 180 ms. Já no cenário em que ocorre o armazenamento em cache, a primeira requisição apresenta um tempo de resposta similar ao anterior (quase 200 ms), mas a partir da segunda requisição (quando o valor da chave já foi armazenado na cache) o tempo médio reduz para 6,5 ms. Esta redução de cerca de 96% no tempo de resposta ressalta o benefício da VNF-Cache. Além disso, a redução do número de requisições para o servidor também diminui o tráfego de pacotes na rede e a sobrecarga dos servidores.

4.2. Discussão

Os experimentos mostram com clareza que a VNF-Cache reduz o tempo de resposta e aumenta o número de requisições processadas por segundo para bancos de dados chave-valor remotos. Além disso, vale recordar que há uma redução do tráfego de rede implícita, ao evitar que requisições e respostas tenham que percorrer todo o caminho até o servidor. Por fim, devido à sua construção como uma função virtual de rede, é notável a flexibilidade de implantação proporcionada pela VNF-Cache, já que a mesma pode ser instanciada e configurada em diversos pontos da rede de maneira simples e rápida.

5. Conclusão

Este trabalho propôs a VNF-Cache, um serviço de cache para bancos de dados chave-valor implementado como uma função virtual de rede. Através do processamento dos pacotes

de rede enviados entre clientes e servidores de bancos de dados, a VNF-Cache pode realizar o armazenamento de conjuntos chave-valor diretamente na rede, aproximando os dados das aplicações solicitantes. Ao retornar os dados requisitados diretamente para os clientes, a VNF-Cache possibilita uma redução no tempo de resposta, tráfego de dados e uso de recursos da rede. Através da implementação de um protótipo e dos experimentos realizados, foi possível obter uma redução considerável no tempo de resposta das requisições para os servidores de bancos de dados chave-valor geograficamente distantes. Além disso, os experimentos apontam um aumento expressivo do número de requisições processadas por segundo.

Trabalhos futuros incluem a implementação de diversas políticas de preenchimento e substituição de dados em cache. Outra expansão que ampliaria ainda mais o escopo de funcionamento é permitir o *caching* de dados de bancos de dados e coleções variáveis, bem como de múltiplos bancos de dados chave-valor simultaneamente, como o Redis e o Amazon DynamoDB. Uma limitação do protótipo é a necessidade de comunicação segura entre os clientes e os servidores – é necessário garantir o uso de VNF-Caches com segurança. Por fim, outro trabalho futuro relevante é a implementação de métodos de armazenamento dos conjuntos chave-valor em estruturas de dados mais robustas e que possuam um melhor tratamento para manipulações de dados concorrentes.

Agradecimentos

Este trabalho foi parcialmente apoiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Programa de Excelência Acadêmica (PROEX) – Código de Financiamento 001; e pelo CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) - projeto 305108/2025-5.

Referências

Bondan, L., Franco, M. F., Marcuzzo, L., Venancio, G., Santos, R. L., Pfitscher, R. J., Scheid, E. J., et al. (2019). Fende: marketplace-based distribution, execution, and life cycle management of vnfs. *IEEE Communications Magazine*, 57(1):13–19.

Clayman, S., Kalan, R. S., and Sayit, M. (2018). Virtualized cache placement in an sdn/nfv assisted sand architecture. In *2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 1–5. IEEE.

ETSI (2012). Network functions virtualisation – introductory white paper. Standard, European Telecommunications Standards Institute, Darmstadt, Germany.

ETSI (2021). Etsi gr nfv-man 001 v1.2.1 - network functions virtualisation (nfv); management and orchestration; report on management and orchestration framework. Standard, European Telecommunications Standards Institute, Valbonne, France.

ETSI (2025). Open Source MANO. `https://osm.etsi.org`. Accessado em novembro de 2025.

Flauzino, J., Fülber-Garcia, V., Huff, A., Venâncio, G., and Jr., E. D. (2021). Gerência e orquestração de funções e serviços de rede virtualizados em nuvem cloudstack. In *XXVI Workshop de Gerência e Operação de Redes e Serviços*, pages 82–95. SBC.

Flauzino, J., Lyra, C., and Duarte Jr., E. (2024). Utilizando anycast para filtragem de pacotes para funções de rede virtualizadas em roteadores de alto desempenho. In *15o Workshop de Pesquisa Experimental Internet do Futuro (WPEIF)*, pages 31–38. SBC.

Fulber-Garcia, V., Duarte Jr, E. P., Huff, A., and dos Santos, C. R. (2020). Network service topology: Formalization, taxonomy and the custom specification model. *Computer Networks*, 178:107337.

Fulber-Garcia, V., Flauzino, J., Venâncio, G., Huff, A., and Junior, E. P. D. (2024). Breaking the limits: Bio-inspired sfc deployment across multiple domains, clouds and orchestrators. In *2024 IEEE Conference on NFV-SDN*, pages 1–6. IEEE.

Halpern, J. et al. (2015). Service Function Chaining (SFC) Architecture. RFC 7665, IETF.

Huff, A., Venâncio, G., Garcia, V. F., and Duarte, E. P. (2020). Building multi-domain service function chains based on multiple nfv orchestrators. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 19–24. IEEE.

Jacob, B., Ng, S., and Wang, D. (2008). *Memory Systems: Cache, DRAM, Disk*. Elsevier.

Jeong, J. P., Shen, Y., Ahn, Y., Kim, Y., Duarte Jr., E. P., and Yao, K. (2025a). A framework for the interface to in-network functions (i2inf).

Jeong, J. P., Shen, Y., Ahn, Y., Kim, Y., Duarte Jr., E. P., and Yao, K. (2025b). Interface to in-network functions (i2inf): Problem statement.

Liu, Y., Point, J. C., Katsaros, K. V., Glykantzis, V., Siddiqui, M. S., and Escalona, E. (2017). Sdn/nfv based caching solution for future mobile network (5g). In *2017 European Conference on Networks and Communications (EuCNC)*, pages 1–5. IEEE.

Seeger, M. (2009). Key-value stores: a practical overview. *Medieninformatik*.

Tacker (2025). Tacker - OpenStack NFV Orchestration. `https://wiki.openstack.org/wiki/Tacker`. Accessado em novembro de 2025.

Turchetti, R. C. and Duarte, E. P. (2015). Implementation of failure detector based on network function virtualization. In *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, pages 19–25. IEEE.

Venâncio, G., Turchetti, R. C., Camargo, E. T., and Duarte Jr, E. P. (2021). Vnf-consensus: A virtual network function for maintaining a consistent distributed software-defined network control plane. *International Journal of Network Management*, 31(3):e2124.

Venâncio, G., Turchetti, R. C., and Duarte, E. P. (2019). Nfv-rbcast: Enabling the network to offer reliable and ordered broadcast services. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*, pages 1–10. IEEE.

Venâncio, G., Turchetti, R. C., and Duarte Jr, E. P. (2022). Nfv-coin: Unleashing the power of in-network computing with virtualization technologies. *Journal of Internet Services and Applications*, 13(1):46–53.

Zhuang, W. et al. (2019). SDN/NFV-empowered future IoV with enhanced communication, computing, and caching. *Proceedings of the IEEE*, 108(2):274–291.