UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

OTÁVIO AUGUSTO ROCHA DA CRUZ

MICRO-CHAIN: UMA ARQUITETURA PARA GERENCIAR E ORQUESTRAR MICROSSERVIÇOS NDN

Porto Alegre
2024

OTÁVIO AUGUSTO ROCHA DA CRUZ

MICRO-CHAIN: UMA ARQUITETURA PARA GERENCIAR E ORQUESTRAR MICROSSERVIÇOS NDN

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.
Área de concentração: Controle e Automação

ORIENTADOR: Prof. Dr. Carlos Eduardo Pereira

CO-ORIENTADOR: Prof. Dr. Edison Pignaton de Freitas

Porto Alegre
2024

OTÁVIO AUGUSTO ROCHA DA CRUZ

MICRO-CHAIN: UMA ARQUITETURA PARA GERENCIAR E ORQUESTRAR MICROSSERVIÇOS NDN

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Carlos Eduardo Pereira, UFRGS
Doutor pela Universitat Stuttgart, Alemanha

Banca Examinadora:

Prof. Dr. Luciano Paschoal Gaspary, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Angelo Perkusich, UFCG
Doutor pela Universidade Federal da Paraíba – João Pessoa, Brasil

Prof. Dr. Ivan Müller, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Coordenador do PPGEE: _____
Prof. Dr. Jeferson Vieira Flores

Porto Alegre, junho de 2024.

DEDICATÓRIA

Dedico este trabalho a todos que me apoiaram de alguma forma, seja diretamente ou indiretamente. Em especial, a minha tia Maria Cândida Rocha e ao meu tio Dorvalino Getúlio de Oliveira, a quem dedico essa dissertação *in memorian*.

AGRADECIMENTOS

Ao Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) pela oportunidade de realização de trabalhos em minha área de pesquisa.

Ao Prof. Carlos, agradeço o tempo que dedicastes a me orientar e agregar ao trabalho.

Ao Prof. Edison, agradeço por me co-orientar e estar sempre disposto à me ajudar.

Ao senhor Paulo Milheiro Mendes, que por intermédio da Airbus nos auxiliou durante todo o trabalho.

À todas as pessoas que se fizeram presentes nessa etapa da minha vida e me apoiando de alguma forma para a conclusão deste curso.

À CAPES pela provisão da bolsa de mestrado.

RESUMO

Redes Centradas em Informação (do inglês *"Information-Centric Networking"* ou ICN) é um dos conceitos que estão sendo desenvolvidos para melhorar o desempenho das redes, incluindo para a Internet do Futuro. Além disso, a arquitetura de microsserviço surge como uma alternativa à arquitetura monolítica para desenvolvimento de *software*. A junção desses dois conceitos leva ao desenvolvimento de microsserviços ICN, *i.e.*, funções de rede ICN desenvolvidas a partir da arquitetura de microsserviços. Essa abordagem tende a aproveitar as qualidades dos dois conceitos, melhorando principalmente a flexibilidade e escalabilidade da rede. Todavia, lidar com o gerenciamento e orquestração de microsserviços ICN não é trivial. Portanto, este trabalho propõe Micro-Chain, uma arquitetura para gerenciar e orquestrar microsserviços ICN para um conjunto de nós. Este trabalho descreve Micro-Chain a partir de quatro módulos e os seus relacionamentos, além das operações fundamentais. Mais especificamente, este trabalho lida com Rede de Dados Nomeada (do inglês *"Named Data Networking"* ou NDN), uma implementação de ICN.

Para validar a solução, dois estudos de caso são definidos e avaliados. No primeiro, é avaliada a escala automática de um microsserviço NDN com base no consumo de recursos. Além disso, é abordado o problema de posicionamento de microsserviços, onde é verificada uma diferença na performance para diferentes configurações.

No segundo estudo de caso, um contexto militar é abordado. Nesse cenário, uma versão da arquitetura Micro-Chain é utilizada para estabelecer uma rede NDN sob demanda para recuperação de dados de vídeo de um Veículo Aéreo não Tripulado (VANT). Neste sistema, dispositivos com certos limites de recursos podem implantar um microsserviço para armazenamento de conteúdo local em nós intermediários, tal como estabelecido por NDN. Logo, buscando validar o desempenho da rede NDN, comparou-se a recuperação de conteúdo via produtor, simulando o processo que seria realizado pelo Protocolo de Internet (do inglês *"Internet Protocol"* ou IP), e a recuperação de conteúdo a partir de nós intermediários. Os resultados indicam que a recuperação via nós intermediários, mais próximos do consumidor, apresenta Tempo de Ida e Volta 31% menor que a recuperação via produtor.

No estudo de caso militar, a flexibilidade do sistema foi verificada a partir de um

cenário de substituição dinâmica de um VANT com baixo nível de bateria. Nesse experimento, o VANT executa um processo de saída da rede NDN, tendo seus microsserviços NDN excluídos. Por fim, um novo VANT substitui o VANT que saiu, sendo implantados os microsserviços NDN para que ele integre a rede.

Palavras-chave: Arquitetura de Microsserviços, Redes Centradas em Informação, Rede de Dados Nomeada, Automação de Redes.

ABSTRACT

Information-Centric Networking (ICN) is one of the concepts that are being developed to improve network performance, including for the Future Internet. Additionally, microservices architecture emerges as an alternative to monolithic architecture. The combination of these two concepts leads to the development of ICN microservices, *i.e.*, ICN network functions developed from the microservices architecture. This approach tends to take advantage of the qualities of both concepts, mainly improving the flexibility and scalability of the network. Therefore, this work proposes Micro-Chain, an architecture to manage and orchestrate ICN microservices for a set of nodes. This work describes Micro-Chain through four modules and their relationships, along with the main operations. Specifically, this work deals with Named Data Networking (NDN), an implementation of ICN.

To validate the solution, two case studies are defined and evaluated. In the first one, the on-demand scaling of an NDN microservice is evaluated based on resource consumption. Additionally, the problem of positioning microservices is addressed, where a difference in performance for different configurations is verified.

In the second case study, a military context is addressed. In this scenario, a version of the Micro-Chain architecture is used to establish an on-demand NDN network for retrieving video data from an Unmanned Aerial Vehicle (UAV). In this system, devices with certain resource limits can deploy a microservice for storing local content on intermediate nodes, as established by NDN. Thus, to validate the performance of the NDN network, the retrieval of content directly from the producer, simulating the process that would be performed by the Internet Protocol (IP), is compared with the retrieval of content from intermediate nodes. The results indicate that recovery via intermediate nodes, closer to the consumer, has a 31% lower Round-Trip Time than direct retrieval from the producer.

In the case of military study, the system's flexibility was verified through a scenario of dynamic replacement of a UAV with low battery level. In this experiment, the UAV performs an egress process from the NDN network, with its NDN microservices being removed. Finally, a new UAV replaces the exiting UAV, and NDN microservices are deployed for it to integrate into the network.

Keywords: Microservice Architecture, Information-Centric Networking, Named Data Networking, Network Automation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	*Application Programming Interface*
BR	*Backward Router*
CPU	*Central Processing Unit*
CS	*Content Store*
DNS	*Domain Name System*
ETSI	*European Telecommunication Standard Institute*
FIB	*Forwarding Information Base*
HPA	*Horizontal Pod Autoscaler*
HTTP	*Hypertext Transfer Protocol*
ICN	*Information-Centric Networking*
IP	*Internet Protocol*
MANO	*Management and Orchestration*
MEC	*Multi-Access Edge Computing*
NDN	*Named Data Networking*
NFD	*Named Data Networking Forwarding Daemon*
NFV	*Network Functions Virtualization*
NFVI	*NFV Infrastructure*
NFVO	*NFV Orchestrator*
NR	*Name Router*
ONAP	*Open Network Automation Platform*
OSM	*Open Source MANO*
PIT	*Pending Interest Table*
RTT	*Round Trip Time*
SDN	*Software-Defined Networking*
SF	*Strategy Forwarder*
SFC	*Service Function Chaining*

UDP *User Datagram Protocol*

VANT Veículo Aéreo Não Tripulado

VIM *Virtualized Infrastructure Manager*

VM *Virtual Machine*

VNF *Network Functions Virtualization*

VNFM *Virtual Network Function Manager*

VPA *Vertical Pod Autoscaler*

SUMÁRIO

1 INTRODUÇÃO

O Protocolo de Internet (do inglês *"Internet Protocol"* ou IP) é o alicerce sobre o qual a Internet foi construída. Um dos recursos definidos por esse protocolo é o sistema de endereçamento para cada dispositivo conectado. Um pacote IP usa esses endereços para identificar para onde uma mensagem deve ser enviada. No entanto, à medida que as demandas por uma conectividade mais ágil e eficiente crescem, surgem desafios inerentes ao IP, especialmente em sua abordagem centrada no *host*. Essa característica significa que o foco principal está nos dispositivos que se comunicam, e não no conteúdo que está sendo transmitido. Para superar essas barreiras, surgem as Redes Centradas em Informação (do inglês *"Information-Centric Networking"* ou ICN) como uma alternativa promissora (SINGH; UJJWAL, 2020).

Ao contrário do modelo IP, que se concentra nos hosts, o ICN adota um modelo de comunicação solicitação-resposta que utiliza nomes de conteúdo diretamente na camada de rede. Essa abordagem torna a ICN mais adequada para a aquisição de conteúdo na Internet, pois os usuários podem solicitar diretamente os dados que desejam, sem a necessidade de traduzir nomes em endereços IP, como é feito pelo Sistema de Nome de Domínio (do inglês *"Domain Name System"* ou DNS). Além disso, o ICN define uma estratégia nativa para armazenamento temporário de conteúdo em dispositivos da rede, desvinculando o conteúdo de um elemento único. Tal estratégia possibilita a recuperação rápida de conteúdo em nós intermediários mais próximos dos consumidores, o que é especialmente promissor para conteúdos acessados com frequência.

Paralelamente, no campo de desenvolvimento de *software*, a arquitetura monolítica se destaca como o modelo tradicional. Nela, o sistema é implementado como uma única unidade coesa, semelhante a um bloco sólido. Essa abordagem pode ser eficaz para projetos pequenos e simples, mas apresenta desafios para aplicações complexas e de grande escala. Sistemas monolíticos extensos são difíceis de manter e melhorar, pois modificar uma parte do *software* pode afetar outras partes, resultando em erros difíceis de identificar e corrigir. Além disso, a escalabilidade exige a replicação completa do sistema, o que pode ser dispendioso e ineficiente (CERNY *et al.*, 2022).

A arquitetura de microsserviços surge como uma alternativa à arquitetura monolítica.

Nessa arquitetura, o sistema é decomposto em pequenos serviços independentes, cada um com sua própria função específica. Diferente da abordagem monolítica, cada microsserviço pode ser escalonado separadamente de acordo com suas necessidades específicas, otimizando o uso de recursos. Ademais, uma alteração em um microsserviço exige pouca ou nenhuma alteração dos outros, logo, é mais fácil realizar manutenções e incorporar novas melhorias (CERNY *et al.*, 2022).

A implementação de ICN utilizando a arquitetura de microsserviços,denominada aqui como microsserviços ICN, aproveita as vantagens inerentes de ambos os conceitos, proporcionando uma rede escalável e flexível, na qual componentes individuais são desenvolvidos, implantados e gerenciados de forma independente. Além disso, os microsserviços ICN tendem a ser desenvolvidos, mantidos e evoluídos de maneira mais eficaz em comparação com a abordagem monolítica tradicional. Dessa forma, uma rede com essas características se mostra altamente desejável.

No entanto, o gerenciamento e a orquestração de microsserviços apresentam desafios consideráveis. Os microsserviços devem ser implantados, conectados e monitorados em tempo de execução. Além disso, as soluções atuais que utilizam microsserviços geralmente concentram-se no gerenciamento e orquestração de aplicações na nuvem, enquanto microsserviços ICN lidam com operações de rede. Para suportar estes requisitos, é necessário definir uma solução que determine uma organização de entidades e os seus relacionamentos para realizar estas operações.

Buscando aproveitar esse contexto, MARCHAL; CHOLEZ; FESTOR (2018) desenvolveram um conjunto de microsserviços para Rede de Dados Nomeada (do inglês *"Named Data Networking"* ou NDN), uma implementação de ICN. Além disso, para controlar dinamicamente os microsserviços NDN desenvolvidos, os autores também implementaram um gerente responsável por implantar e conectar os microsserviços.

Todavia, o gerenciador desenvolvido por MARCHAL; CHOLEZ; FESTOR (2018) está limitado a uma única máquina, enquanto, em aplicações reais, a implantação de NDN ocorre em uma rede distribuída, com múltiplos nós. Portanto, este trabalho estende o trabalho de MARCHAL; CHOLEZ; FESTOR (2018) para um contexto com múltiplos nós. Para isso, esta dissertação apresenta uma arquitetura modular chamada Micro-Chain para implantar e controlar microsserviços ICN em um cluster de máquinas. As contribuições deste trabalho são:

- Desenvolvimento de uma arquitetura para implantar, orquestrar e monitorar uma rede NDN baseada em microsserviços.

- Desenvolvimento de uma solução para implantar uma rede NDN customizada a partir da quantidade de recursos dos dispositivos, onde dispositivos com poucos recursos implantam apenas recursos cruciais e quando necessário.

- Buscando melhorar a reprodutibilidade e reuso, as implementações das soluções propostas podem ser acessadas online[1]. Essa implementação aproveita ferramentas como Kubernetes e Prometheus, beneficiando-se do suporte robusto da comunidade e atualizações regulares dessas tecnologias.

- Uma discussão sobre o uso da arquitetura de microsserviços para desenvolver funções de rede, mais especificamente de uma rede NDN, identificando desafios e direções futuras.

O principal objetivo deste trabalho é projetar e implementar uma arquitetura para implantar e gerenciar uma rede ICN sob demanda. Para implantar essa rede NDN, são utilizados microsserviços NDN, que são uma implementação do NDN a partir da arquitetura de microsserviços. Esses microsserviços são desenvolvidos "quebrando"o NDN em pequenas funções isoladas que podem ser implantadas e geridas separadamente. Essas funções (ou microsserviços NDN) podem ser encadeadas para implementar determinadas funcionalidades desejadas. Espera-se que a solução projetada seja capaz de instanciar, configurar, monitorar e escalar esses microsserviços em tempo de execução.

Micro-Chain é avaliada através de dois estudos de caso. O primeiro consiste na escala automática de um microsserviço com base no consumo da Unidade Central de Processamento (do inglês *"Central Processing Unit"* ou CPU) e memória. Já o segundo caso, lida com a implantação de uma rede NDN sob demanda em um contexto militar.

Este trabalho é divido da seguinte forma: no capítulo 2 é apresentada a fundamentação teórica sobre NDN, máquinas virtuais e contêineres, arquitetura monolítica e de microsserviços, Kubernetes, Prometheus e Virtualização de Funções de Rede (do inglês *"Network Functions Virtualization"* ou NFV). O capítulo 3 exprime a análise do estado da arte, que abrange principalmente NFV, ICN, NFV desenvolvidos a partir de microsserviços e soluções para gerenciar e orquestrar Funções Virtualizadas de Rede (do inglês *"Virtualized Network Function"* ou VNF). Na sequência, no capítulo 4, é descrita a solução proposta nesse trabalho. No capítulo 5, são apresentados os estudos de caso abordados, resultados e discussões. Finalmente, no capítulo 6, é apresenta a conclusão deste trabalho.

[1] `https://gitfront.io/r/otavio/N7YU2cbfUFrD/micro-chain-dissertation/`

2 FUNDAMENTAÇÃO TEÓRICA

Os principais conceitos utilizados nesta proposta estão descritos neste capítulo. Primeiro, é descrito o funcionamento de NDN. Em seguida, são introduzidos os conceitos de Máquinas Virtuais e contêineres, destacando suas principais diferenças. A arquitetura monolítica e de microsserviços são tratadas na sequência. Então, é apresentado um orquestrador de contêineres chamado Kubernetes, sendo destacada sua arquitetura e operações fundamentais. Posteriormente, é introduzida uma ferramenta para monitoramento, o Prometheus, sendo sua arquitetura também detalhada. Por fim, é apresentado NFV.

2.1 Redes de Dados Nomeada

NDN (ZHANG *et al.*, 2014) é uma implementação do paradigma ICN, onde os dados são acessados e buscados por seus nomes. Em contraste ao paradigma convencional do IP, o NDN atribui um nome único a cada pedaço de informação transmitido na rede, sendo esses nomes usados para identificar e recuperar informações. Portanto, quando um dispositivo deseja acessar um determinado conteúdo, em vez de solicitar o conteúdo com base em um endereço específico de um servidor, o dispositivo envia um pedido contendo o nome do dado desejado.

A comunicação em uma rede NDN é baseada em dois tipos de pacotes: interesse e dados. O pacote de interesse representa a solicitação do consumidor quanto a um certo conteúdo, enquanto o pacote de dados consiste no conteúdo solicitado. Um dispositivo NDN executa 3 funções, Armazenamento de Conteúdo (do inglês *"Content Store"* ou CS), Base de Informações de Encaminhamento (do inglês *"Forwarding Information Base"* ou FIB) e Tabela de Interesses Pendentes (do inglês *"Pending Interest Table"* ou PIT), os quais são definidos a seguir:

- **CS**: um cache temporário de pacotes de dados que serve para satisfazer interesses futuros. CS emprega uma estratégia para escolher qual conteúdo deve ser armazenado localmente. Por exemplo, se o CS estiver cheio, o conteúdo mais antigo pode ser substituído por um conteúdo mais recente.

- **FIB**: uma tabela de roteamento para pacotes de interesse que mapeia nomes para

interfaces de rede. O FIB é preenchido por um protocolo de roteamento baseado em prefixo de nome e pode ter múltiplas interfaces de saída para cada prefixo.

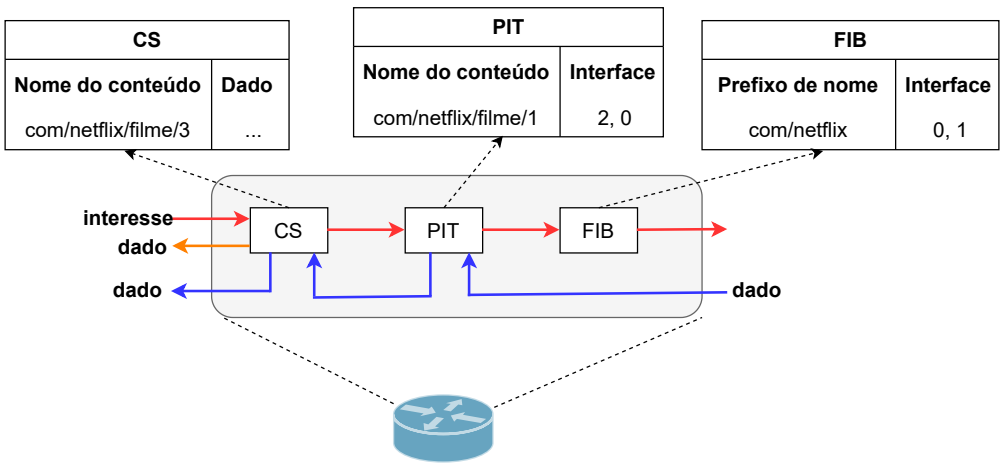
- **PIT**: tabela com todos os interesses encaminhados, mas ainda não atendidos. Cada entrada PIT registra o nome do conteúdo solicitado pelo interesse juntamente com sua(s) interface(s) de entrada para auxiliar no encaminhamento dos pacotes de dados ao longo do caminho reverso realizado pelo pacote do interesse.

A Figura 1 ilustra a operação de um dispositivo NDN em uma rede conforme os tipos de pacotes recebidos. Quando um pacote de interesse (setas vermelhas) chega ao dispositivo, ele verifica se o conteúdo está armazenado na CS. Se estiver, um pacote de dados é enviado em resposta (seta laranja). Essa é a forma mais rápida de recuperar conteúdo em NDN, isto é, a partir de um dispositivo com o conteúdo armazenado localmente.

Se o conteúdo não está na CS, o dispositivo verifica se existe alguma entrada na PIT para o pacote de interesse recebido. Caso exista, significa que o dispositivo já encaminhou um pacote de interesse para esse conteúdo e está à espera da resposta. Nesse caso, não é necessário reencaminhar um novo pacote de interesse, o dispositivo apenas adiciona a interface de entrada na PIT e, quando o pacote de dados correspondente chegar, é realizado um envio multicast para todas as interfaces na PIT, conforme exemplificado para as interfaces 2 e 0 para o conteúdo *com/netflix/filme/1*. Essa abordagem é mais eficiente que a unicast utilizada pelo IP. Por outro lado, se a PIT não possui nenhuma entrada para o pacote de interesse recebido, o dispositivo adiciona uma nova entrada para esse interesse. Na sequência, o pacote de interesse é operado pela FIB, onde é encaminhado para o próximo dispositivo da rede.

Ainda, a qualquer momento que um pacote de dados chega no dispositivo, ele pode escolher armazená-lo localmente antes de encaminhar para o próximo dispositivo.

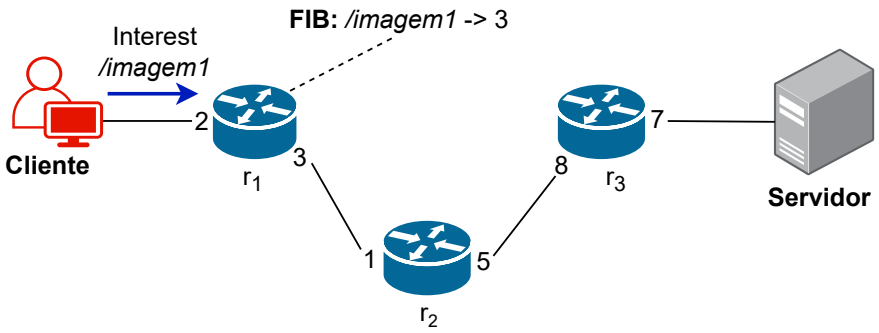
Figura 1 – Ilustração do processamento de pacote em um dispositivo NDN.



Fonte: do autor.

A Figura 2 exemplifica a solicitação para um conteúdo nomeado como */imagem1* em uma rede com 3 roteadores NDN, onde cada roteador possui duas interfaces identificadas, uma de saída e uma de entrada. Assim que a solicitação do conteúdo enviada pelo cliente chega ao roteador r_1, ele executa as operações definidas na Figura 1. Como r_1 não possui cache local do conteúdo e nem uma correspondência na PIT, o conteúdo é encaminhado conforme definido pela FIB, que nesse caso consiste em encaminhar para a porta 3. Esse processo se repete para o roteador r_2 e r_3, quando o interesse é finalmente encaminhado para o servidor.

Figura 2 – Exemplo de uma solicitação de um conteúdo por um cliente em uma rede NDN.



Fonte: do autor.

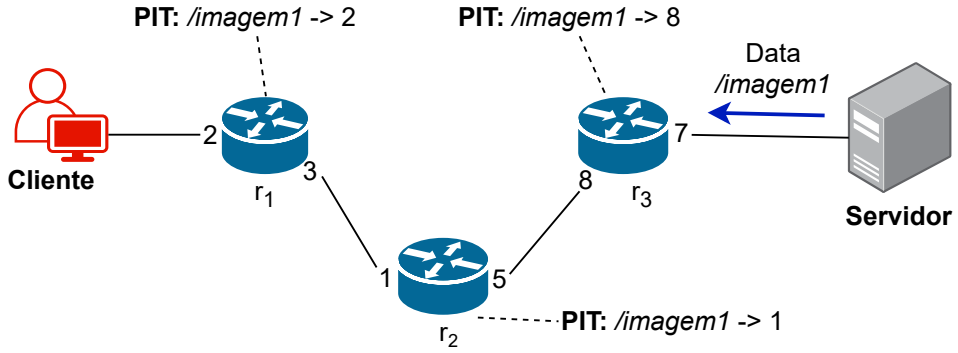
Quando o pacote de interesse chega ao servidor, ele responde com um pacote de dados contendo o conteúdo */imagem1*, Figura 3. O pacote de dados utiliza as entradas na PIT dos roteadores para direcionar o pacote no caminho reverso do pacote de interesse. Conforme o pacote de dados trafega nos roteadores, eles utilizam a política implementada pela CS para decidir se devem armazenar localmente o pacote. Por exemplo, se o roteador r_1 decide armazenar o pacote de dados */imagem1*, quando o cliente ou qualquer outro dispositivo requisitar esse conteúdo a r_1, ele será capaz de responder diretamente com o pacote de dados armazenado.

2.2 Máquinas Virtuais e Contêineres

Uma Máquina Virtual (do inglês *"Virtual Machine"* ou VM) é um ambiente de computação virtualizado que emula um sistema de computador físico. A Figura 4 apresenta os principais componentes de uma máquina física sem VM, à esquerda, e com VMs, à direita. Na representação com VMs, há a inclusão de um componente chamado Hypervisor, mais especificamente um Hypervisor do tipo 2. Esse componente é responsável por criar e gerenciar as VMs (SHARMA *et al.*, 2016).

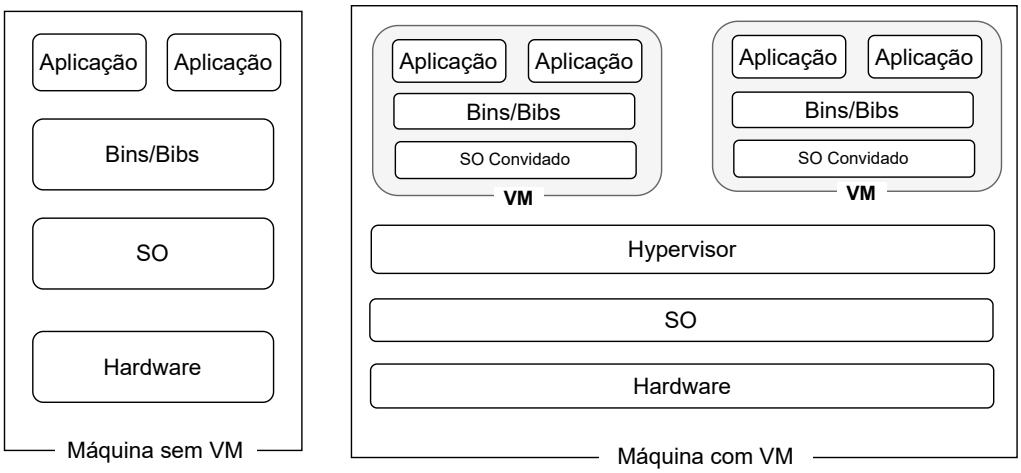
Uma das principais vantagens das VMs é a capacidade de executar vários sistemas operacionais e aplicativos simultaneamente em um único servidor, eliminando a neces-

Figura 3 – Exemplo de recuperação de um conteúdo no servidor em uma rede NDN.



Fonte: do autor.

Figura 4 – Exemplo de um máquina sem VM e com VM.



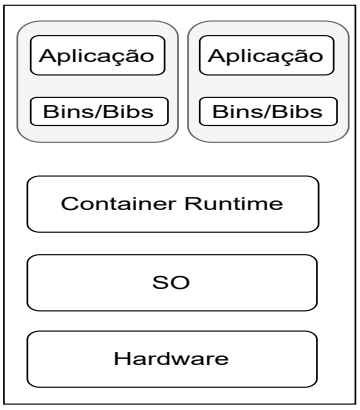
Fonte: do autor.

sidade de investir em hardware dedicado para cada aplicação com sistema operacional diferente. Além disso, as VMs oferecem a conveniência de serem facilmente copiadas, movidas ou restauradas em caso de falhas de hardware ou outras emergências, o que simplifica a implementação de estratégias de backup e recuperação. Com a virtualização, as organizações podem facilmente ajustar suas infraestruturas conforme demandas correntes, adicionando ou removendo VMs conforme necessário, sem interrupção significativa dos serviços essenciais. Essa capacidade de escalabilidade dinâmica torna as VMs uma solução ideal para ambientes dinâmicos.

Outra abordagem para desenvolver e encapsular aplicações são os contêineres. A Figura 5 ilustra os componentes presentes em uma máquina física que implementa aplicações baseadas em contêineres, onde o Container Runtime é o componente responsável por iniciar e gerenciar os contêineres. Diferente das VMs, os contêineres compartilham o Kernel do sistema operacional hospedeiro e utilizam os conceitos de *namespace* e *cgroups* para isolar e limitar recursos e processos. Como os contêineres não precisam iniciar e

manter um sistema operacional como as VMs, eles são mais leves, consomem menos recursos de hardware, e rápidos. Em contrapartida, as VMs oferecem melhor isolamento, com sistemas operacionais independentes (PAHL *et al.*, 2019; CZIVA; PEZAROS, 2017).

Figura 5 – Especificação dos componentes de uma máquina que utiliza contêineres.



Fonte: do autor.

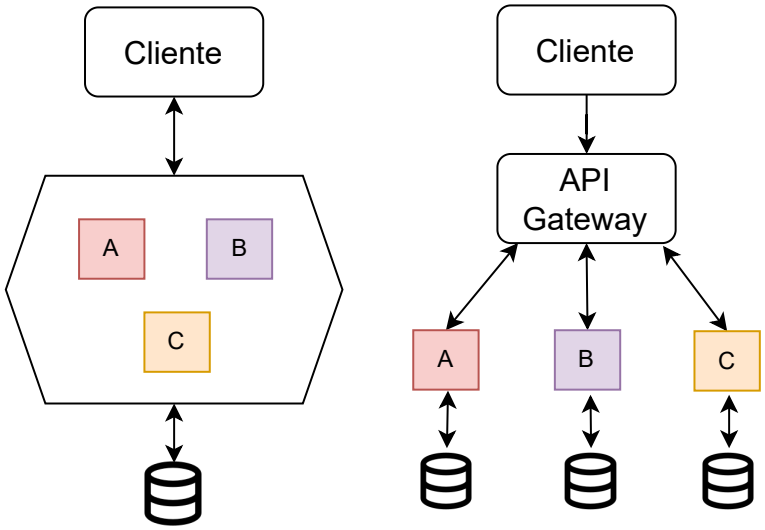
2.3 Arquitetura Monolítica e Arquitetura de Microsserviços

A arquitetura monolítica refere-se a uma maneira de desenvolver aplicativos, onde todos os componentes e funcionalidades estão agrupados e interligados em uma única aplicação. É como ter um grande bloco de construção, onde todas as partes do sistema são desenvolvidas e implantadas juntas.

Como alternativa ao desenvolvimento de *software* baseado na arquitetura monolítica, surge a arquitetura de microsserviços. A Figura 6 apresenta um *software* desenvolvido a partir de uma arquitetura monolítica, à esquerda, e a partir de uma arquitetura de microsserviços, à direita. Na arquitetura de microsserviços, o *software* é desenvolvido como uma coleção de componentes menores e independentes chamados de microsserviços, sendo que cada microsserviço executa uma função específica do sistema (FOWLER, 2016). Além disso, os microsserviços podem ser encadeados para executar um conjunto de funcionalidades. A Figura 6 exemplifica esse contexto, onde os microsserviços utilizam um API Gateway como um ponto de entrada para clientes externos e um intermediador para o encadeamento de microsserviços. Vale ressaltar que essa não é a única maneira de encadear microsserviços, eles podem, por exemplo, possuir conexões diretas entre eles.

Uma das principais diferenças entre a arquitetura monolítica e de microsserviços é a maneira de escalar o sistema. Conforme exemplificado pela Figura 7, quando uma arquitetura monolítica escala, toda a aplicação é replicada. Por outro lado, na arquitetura de microsserviços, partes específicas podem ser escaladas conforme a demanda (FOWLER, 2016). No caso da Figura 7, é considerado que apenas o microsserviço A precisa escalar e

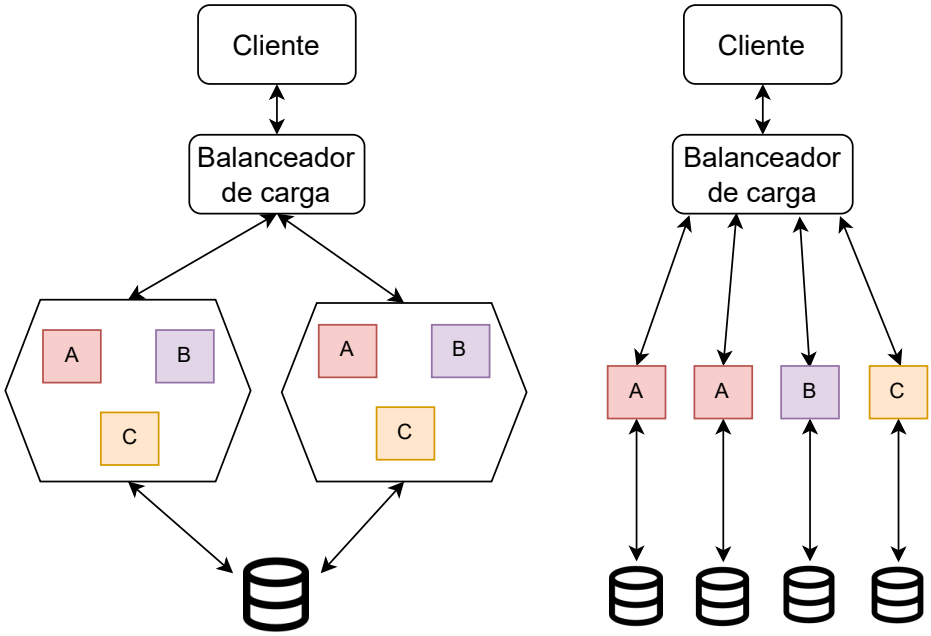
Figura 6 – Arquitetura monolítica e de microsserviços.



Fonte: do autor.

por isso há duas instâncias dele. Um balanceador de carga é utilizado em ambos os casos para gerenciar qual instância do *software* ou microsserviço será executado.

Figura 7 – Ilustração do aumento de escala da arquitetura monolítica e de microsserviços.



Fonte: do autor.

A arquitetura de microsserviços permite que cada microsserviço seja desenvolvido, implantado e dimensionado separadamente, o que a torna potencialmente mais flexível que a arquitetura monolítica. Além disso, o baixo acoplamento entre os microsserviços facilita o processo de atualizações e manutenções do sistema, pois uma alteração em um

microsserviço tende a não afetar diretamente outros microsserviços. Ademais, diferentemente da arquitetura monolítica, cada microsserviço pode ser desenvolvido a partir das tecnologias mais apropriadas para o seu escopo. Por exemplo, um microsserviço pode ser desenvolvido em linguagem Python e outro em C, devido ao suporte de bibliotecas ou desempenho. No entanto, a arquitetura de microsserviços também traz desafios, um deles é a complexidade adicional no gerenciamento e comunicação entre os microsserviços.

Especialmente no contexto de computação em nuvem, microsserviços vêm ganhando popularidade. Nesse cenário, os microsserviços são geralmente encapsulados em contêineres e gerenciados por alguma plataforma de orquestração, como Kubernetes, AWS ECS, Azure Kubernetes Service (AKS) e Google Kubernetes Engine (GKE), para automatizar os processos de implantação, monitoramento, escala e recuperação de falhas.

2.4 Kubernetes

Kubernetes (KUBERNETES, 2023) é uma plataforma de código aberto amplamente utilizada para orquestração de contêineres na nuvem. O Kubernetes gerencia e organiza contêineres, fornecendo recursos para implantação, escalar e balanceamento de carga.

Um cluster Kubernetes consiste em um conjunto de máquinas no qual os componentes do Kubernetes e os aplicativos desenvolvidos em contêineres são implantados e executados. Cada máquina do cluster é um nó Kubernetes, podendo ser categorizado como nó de trabalho ou nó de controle. Os nós de trabalho são responsáveis por executar as aplicações do sistema, enquanto os nós de controles são responsáveis por gerenciar e coordenar as operações no cluster. A Figura 8 especifica os principais componentes dos nós Kubernetes e os seus relacionamentos.

Mais especificamente, o nó de controle implementa os seguintes componentes:

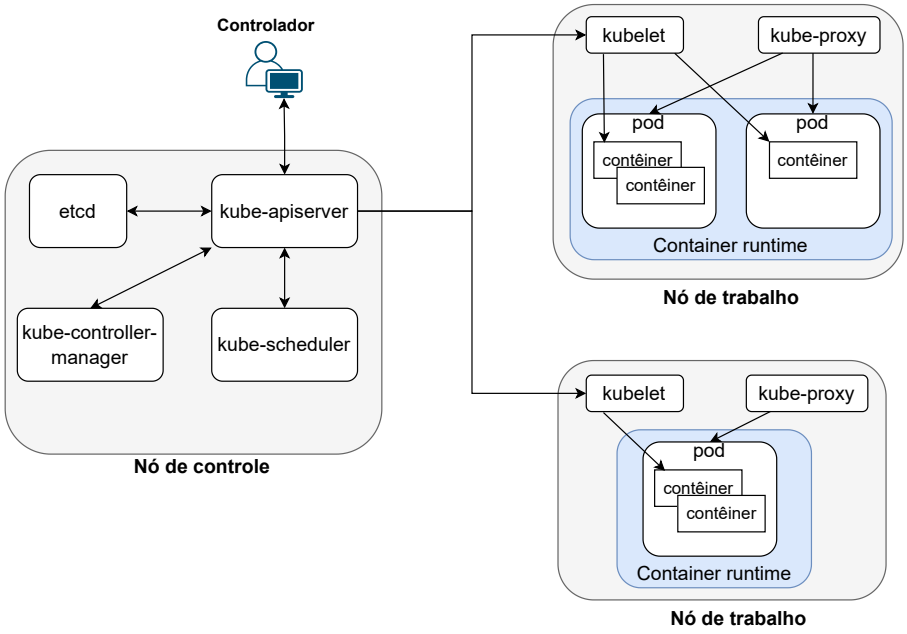
- **Kube-apiserver**: é responsável por expor a Interface de Programação de Aplicação (do inglês *"Application Programming Interface"* ou API) do Kubernetes, permitindo que outros componentes e ferramentas externas interajam com o cluster. Todas as operações no cluster, como implantação, atualização e exclusão de recursos, utilizam esse módulo.

- **etcd**: este componente é um armazenamento do tipo chave-valor distribuído que armazena todos os dados de configuração, segredos e o estado desejado do cluster.

- **Kube-scheduler**: é responsável por determinar o posicionamento de contêineres no cluster;

- **Kube-controller-manager**: é responsável por operar controladores que regulam o estado do cluster de modo a garantir que o estado atual do cluster corresponda ao estado desejado. Existem vários controladores, cada um cuida de um aspecto específico do sistema.

Figura 8 – Arquitetura do Kubernetes.



Fonte: (KAUR; MANGAT; KUMAR, 2022)

Por outro lado, o nó de trabalho implementa os seguintes componentes:

- **Kubelet**: é responsável por gerenciar contêineres. É ele que garante que os contêineres estão em execução e saudáveis.
- **Kube-proxy**: este componente gerencia a conectividade de rede do contêiner. Para isso, ele mantém as regras de encaminhamento de rede para permitir a comunicação entre dispositivos e dispositivos.
- **Container runtime**: lida com a criação, execução e destruição dos contêineres. Kubernetes oferece suporte a vários Containers runtime, como Docker e Containerd.

Outro recurso importante no Kubernetes são os objetos. Eles são entidades persistentes armazenadas no etcd e gerenciadas por meio do kube-apiserver. Os objetos descrevem os aplicativos: quais estão sendo executados, os recursos disponíveis e as políticas de comportamento para cada um. Alguns dos principais objetos do Kubernetes são:

- **Pod**: este objeto é a menor unidade implantável. Ele pode conter um ou mais contêineres que compartilham o mesmo espaço de rede e armazenamento;
- **Service**: É uma abstração que define um conjunto de políticas para acessar um conjunto de objetos, dentro ou fora do cluster. O serviço fornece um terminal estável para acesso, geralmente a partir de um endereço IP e uma porta;

- **ReplicaSet**: é usado para garantir que um determinado número de réplicas de um Pod estejam sempre em execução no cluster.

- **Deployment**: este objeto permite gerenciar declarativamente a implantação de um conjunto de Pods idênticos. É um recurso de nível superior utilizado para gerenciamento de versões e controle de réplicas a partir do ReplicaSet.

O Kubernetes possui várias implementações para atender a diferentes casos de uso e ambientes, como MicroK8s, Google Kubernetes Engine e K3s. Este utiliza K3s, uma implementação com baixo consumo de recursos, que possibilita criar e destruir clusters Kubernetes de maneira rápida e fácil. Essas características tornam K3s ótimo para cenários de aplicações na borda, Internet das Coisas (do inglês *"Internet of Things"* ou IoT) e embarcado.

2.4.1 Gerenciamento de Recursos

No Kubernetes, o gerenciamento de recursos, como uso de CPU e memória, é fundamental para o bom desempenho do cluster. Para isso, o Kubernetes possibilita configurar os limites e as solicitações de recursos durante a implantação de contêineres. A solicitação se refere à quantidade mínima de recursos que um contêiner utiliza, sendo que o Kubernetes aloca esses recursos quando o contêiner é agendado em um nó. E o limite de recursos é a quantidade máxima de recursos que um contêiner pode utilizar. Ambas auxiliam na previsibilidade e no balanceamento de carga no cluster. De modo geral, os conceitos de solicitações e limites devem ser utilizados cuidadosamente, conforme o contexto ao qual a solução está inserida.

Ao definir solicitação de recursos no Kubernetes, o contêiner tem a garantia de uma quantidade miníma de recursos para utilizar. Essa informação também permite que o Kubernetes agende Pods em nós que possuem recursos disponíveis para atender às solicitações definidas. Todavia, solicitações com valores muito altos podem ocasionar recursos ociosos. Além disso, antecipar as necessidades precisas de recursos pode ser desafiador, especialmente para aplicativos com demanda variável ou de difícil previsibilidade.

Já os limites de recursos no Kubernetes evitam que contêineres consumam todos os recursos disponíveis em um nó, o que poderia levar à instabilidade e a possíveis falhas em cascata. Os limites também auxiliam na distribuição mais equilibrada em momentos de alta demanda, evitando situações de contenção. Consequentemente, o gerenciamento adequado desses limites pode resultar em um uso mais eficiente dos recursos do cluster.

Todavia, assim como para as solicitações, determinar os limites de uma aplicação requer um entendimento profundo das necessidades de recursos do aplicativo. Por exemplo, se um contêiner possui limites baixos, ele pode falhar ao tentar iniciar devido à falta de recursos ou precisará escalar com maior frequência, o que gera uma sobrecarga. Ainda, podem ocorrer problemas de subutilização de recursos disponíveis quando o nó possui

recursos livres durante uma alta situacional da demanda, mas um contêiner está limitado a consumir certa quantidade de recursos. Além disso, caso os limites sejam altos, isso limitará a implantação de novos aplicativos no cluster.

2.4.2 Posicionamento

No Kubernetes, o processo de posicionamento de Pods é gerenciado pelo componente kube-scheduler. Esse agendador é responsável por atribuir Pods a nós específicos dentro do cluster. Para isso, são executadas duas etapas, filtragem e cálculo de prioridade. Na filtragem, é verificado quais nós são capazes de executar o Pod aplicando um conjunto de predicados. Alguns predicados suportados são (SANTOS *et al.*, 2019):

- **PodFitsResources:** garante que os Pods tenham recursos de CPU e memória disponíveis para serem alocados.
- **NoVolumeZoneConflict**: garante que, quando um Pod solicita um volume com acesso em zonas específicas, ele só será agendado em nós que estão na mesma zona do volume requisitado.
- **MatchNodeSelector:** controla a distribuição dos Pods com base em sua afinidade ou anti-afinidade com outros Pods ou nós.

Na segunda etapa, cálculo de prioridade, o Kubernetes pontua os nós restantes com base em um conjunto de prioridades. Algumas das prioridades suportadas são (SANTOS *et al.*, 2019):

- **LeastRequestedPriority:** pontua conforme fração livre de CPU e memória, sendo preferível o nó com mais recursos livres.
- **MostRequestedPriority:** o contrário de LeastRequestedPriority, prefere o nó com mais recursos alocados.
- **NodeAffinityPriority:** pontua os nós com base na afinidade definida pelo usuário.

O kube-schedule pode usar um grupo de prioridades de interesse para pontuar os nós. Nesse caso, cada prioridade é ponderada dependendo da importância, sendo a pontuação final de cada nó a soma de todas as pontuações (SANTOS *et al.*, 2019).

2.4.3 Escalabilidade

A escala de Pods no Kubernetes é uma prática crucial para garantir a robustez do sistema. Esse processo permite o ajuste sob demanda de acordo com métricas de interesse. Tradicionalmente, existem duas maneiras de escalar, *Horizontal Pod Autoscaler* (HPA) e *Vertical Pod Autoscaler* (VPA).

O HPA é realizado com base em limites de métricas, como uso de CPU e memória. Quando um Pod atinge esses limites, a quantidade de replicar é ajustada automaticamente. Por exemplo, em um caso de alta demanda, um Pod pode ser estressado ao ponto de atingir o limite de memória estabelecido e, então, uma réplica desse Pod é implantada no sistema. Nesses casos, geralmente utiliza-se um balanceador de carga para dividir o trabalho entre as réplicas.

Por outro lado, no VPA a quantidade de réplicas não é alterada, mas as solicitações e limites de recursos alocados para um Pod são ajustados dinamicamente. Nesse tipo, os Pods passam a possuir solicitações e limites maiores ou menores a depender da necessidade atual.

Para ambas as maneiras de escalar, é necessário definir as métricas relevantes para o contexto no qual o sistema está inserido, sendo que geralmente o consumo de CPU e memória são as métricas utilizadas. Além disso, é necessário definir uma forma de monitorar essas métricas em tempo de execução para atender ao comportamento esperado.

2.5 Prometheus

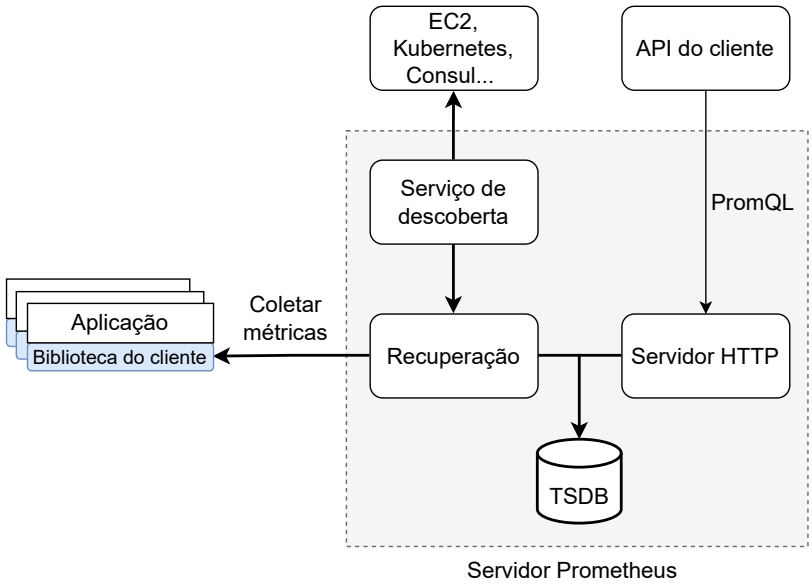
Prometheus (PROMETHEUS, 2023) é um *software* de código aberto que fornece funcionalidades de monitoramento e alerta. A Figura 9 apresenta a arquitetura do Prometheus, onde o Servidor Prometheus contém os módulos implementados por essa ferramenta, são eles: Serviço de Descoberta, Recuperação, *Time Series DataBase* (TSDB) e o servidor Hypertext Transfer Protocol (HTTP). O módulo Recuperação é responsável por extrair as métricas de endpoints. Para isso, os endpoints precisam implementar um servidor e exportar as métricas em um formato específico, o que é facilitado pela biblioteca cliente do Prometheus.

Capturas bem sucedidas realizadas pelo módulo Recuperação são armazenadas no TSDB, um banco de dados de série temporal. As métricas armazenadas em TSDB podem ser acessadas via Servidor HTTP, a partir de Prometheus Query Language (PromQL). PromQL é uma linguagem de consulta que permite selecionar e agregar dados de série temporal. Além disso, em alguns contextos é necessária a descoberta de novos endpoits. Para isso, Prometheus implementa o módulo Serviço de descoberta.

2.6 Virtualização de Funções de Rede

A NFV é uma estratégia disruptiva que revoluciona a concepção, implementação e operação de redes convencionais. Ao contrário do modelo tradicional, que depende de hardware dedicado para cada função de rede, a NFV aproveita a virtualização para executar essas funções em dispositivos de propósito geral. A Figura 10 ilustra a distinção entre a abordagem convencional e NFV para cinco funções de rede, balanceador de carga,

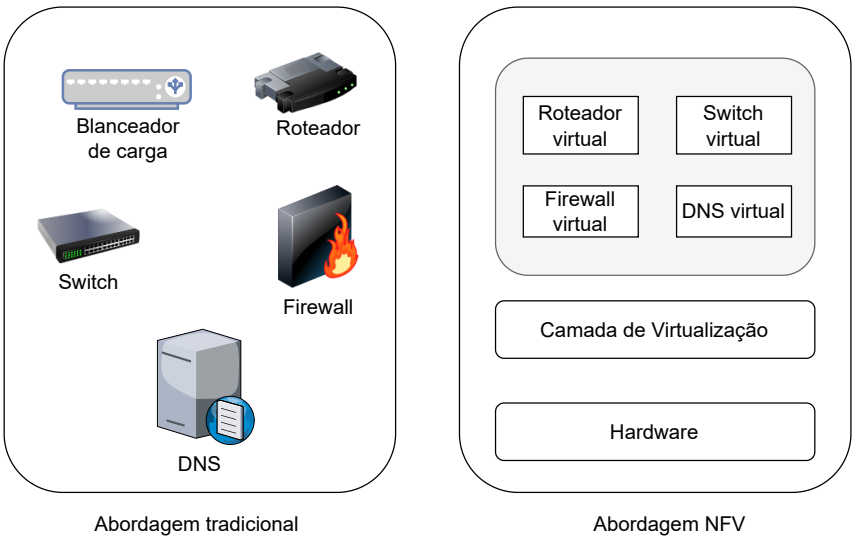
Figura 9 – Arquitetura do Prometheus.



Fonte: do autor.

roteador, switch, firewall e DNS.

Figura 10 – Abordagem tradicional e abordagem baseada em NFV.



Fonte: (KAUR; MANGAT; KUMAR, 2022).

NFV oferece uma série de vantagens às operadoras de rede em comparação à abordagem tradicional, alguns deles são:

- **Flexibilidade**: maior flexibilidade para implantar novos serviços e funcionalidades de rede, sem a necessidade de adquirir e instalar hardware adicional.

- **Escalabilidade**: facilita adicionar ou remover recursos de rede rapidamente, o que impulsiona o ajuste da rede sob demanda.
- **Redução de custos**: minimiza a subutilização de recursos de servidores, podendo reduzir significativamente os custos de capital e operacional da rede.
- **Agilidade**: permite que as operadoras de rede respondam mais rapidamente às mudanças nas demandas do mercado e a novas tecnologias.

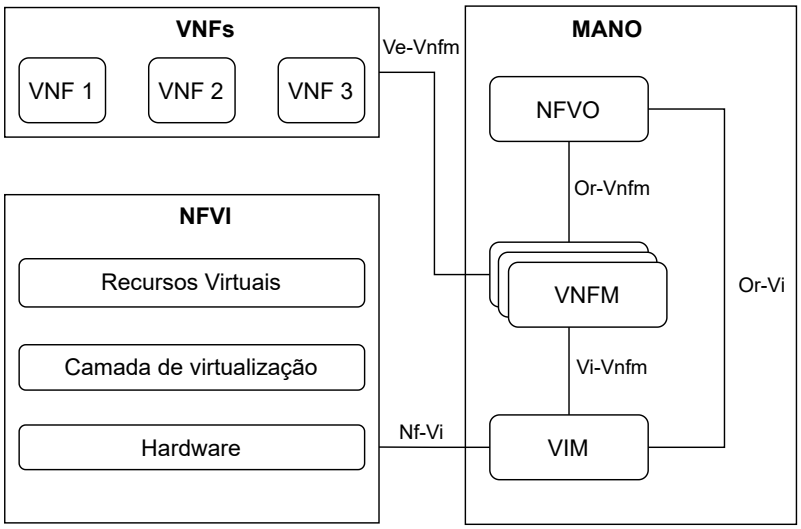
O European Telecommunication Standard Institute (ETSI) desempenha um papel crucial no desenvolvimento e na padronização de NFV. Em 2012, esse instituto lançou um grupo de especificações para NFV visando criar um ambiente interoperável e aberto para a implantação de serviços de rede virtualizados. Desde então, o ETSI tem realizado esforços para definir arquiteturas, interfaces e requisitos para a implementação de NFV. Essas normas proporcionam uma referência comum para os fornecedores e os operadores de rede, o que deve acelerar a adoção do NFV em todo o setor de telecomunicações.

Em uma de suas especificações, o ETSI estabelece uma arquitetura de referencia para NFV (ETSI, 2014). Essa arquitetura é composta por três camadas: VNF, Infraestrutura de Virtualização de Funções de Rede (do inglês *"NFV Infrastructure"* ou NFVI) e Gerenciamento e Orquestração de NFV (do inglês *"NFV Management and Orchestration"* ou MANO). As VNFs são as unidades funcionais de NFV, representando a implementação virtualizada de funções de rede. O NFVI inclui recursos de hardware, camada de virtualização e recursos virtuais para executar VNFs. Por fim, o MANO é o responsável por gerenciar e orquestrar tanto as VNFs quanto a NFVI.

A Figura 11 fornece uma visão geral da arquitetura ETSI NFV, onde os módulos principais de MANO são o Gerenciador de Infraestrutura Virtualizada (do inglês *"Virtualized Infrastructure Manager"* ou VIM), Gerenciador de Funções de Rede Virtual (do inglês *"Virtual Network Function Manager"* ou VNFM) e Orquestrador NFV (do inglês *"NFV Orchestrator"* ou NFVO), os quais são descritos a seguir:

- **VIM**: gerencia os recursos físicos e virtuais da infraestrutura de um domínio. Podem haver vários VIMs. Mais especificamente, o VIM cria, mantém e encerra máquinas virtuais, além de executa o gerenciamento de desempenho e falhas de hardware, *software* e recursos virtuais. Pode haver um único VIM para gerenciar os recursos de um único domínio NFVI ou vários VIMs com cada um gerenciando seu respectivo NFVI.
- **VNFM**: responsável por criar, manter, encerrar e escalar instâncias VNF. Um único VNFM pode gerenciar vários VNFs ou um único VNF.
- **NFVO**: é o elemento central, coordena os recursos de diferentes VIMs e a criação de um serviço ponta a ponta que envolve diferentes domínios VNFMs, podendo instanciar VNFMs quando aplicável.

Figura 11 – ETSI NFV.



Fonte: (KAUR; MANGAT; KUMAR, 2022).

O ETSI também define os protocolos e APIs que fornecem a interface entre os módulos, alguns deles são: Nf-Vi entre o VIM e o NFVI; Or-Vi entre o NFVO e o VIM; Vi-Vnfm entre o VNFM e o VIM; Or-Vnfm entre VNFM e o NFVO; Ve-Vnfm entre as VNFs e VNFM. Algumas dessas interfaces já possuem especificação, outras estão a ser padronizadas (KAUR; MANGAT; KUMAR, 2022).

3 ANÁLISE DO ESTADO DA ARTE

Conforme os objetivos e a motivação deste trabalho, a análise do estado da arte se concentra nos seguintes temas:

- Uso conjunto e separado dos conceitos Virtualização de Funções de Rede, Redes Definidas por Software e Redes Centradas em Informação.
- VNFs desenvolvidos a partir de microsserviços.
- Análise de soluções para gerenciar e orquestrar VNF, uma vez que microsserviços NDN são basicamente funções virtuais de rede.
- Problema de posicionamento.

3.1 Virtualização de Funções de Rede, Redes Definidas por Software e Redes Centradas em Informação

GüR *et al.* (2022) investigaram a integração de ICN e Computação de Borda de Multiacesso (do inglês *"Multi-Access Edge Computing"* ou MEC) em redes 5G e *Beyond 5G*. Eles defendem que o uso conjunto de MEC e ICN oferecerá latência reduzida, cache nas bordas, diminuição do tráfego de rede e mobilidade eficiente de sessões. Sendo esses benefícios úteis para aplicações como cidades inteligentes, casas inteligentes, aplicações multimédia em rede, automóveis de condução autônoma e cuidados de saúde inteligentes.

Quanto à mobilidade, GüR *et al.* (2022) argumentam que o armazenamento local de conteúdo de ICN permite que os usuários recuperem o conteúdo a partir de qualquer nó intermediário, desacoplando o conteúdo do servidor de hospedagem original. Assim, o suporte à mobilidade e ao transporte sem sessão é intrínseco ao ICN. Dessa forma, quando um consumidor móvel se desconecta e se conecta a um novo ponto de conexão, isso é facilmente tratado pela estrutura de comunicação do ICN. GüR *et al.* (2022) concluem que a mobilidade dos consumidores é bem apoiada pela ICN, mas a mobilidade dos produtores e a mobilidade das redes são questões importantes de mobilidade que ainda precisam analisadas.

Devido principalmente ao suporte à mobilidade apresentado por ICN, trabalhos vêm abordando o uso dessa tecnologia para redes de VANTs (SHARIAT; TIZGHADAM; LEON-GARCIA, 2016; LEI *et al.*, 2019; BARKA *et al.*, 2018; KAPETANIDOU; MENDES; TSAOUSSIDIS, 2023). SHARIAT; TIZGHADAM; LEON-GARCIA (2016) desenvolveram um sistema pub/sub para um sistema de transporte inteligente no contexto de cidades inteligentes. Nesse sistema, os VANTs empregam ICN para coletar e compartilhar informações sensoriais visando evitar problemas como o congestionamento de tráfego. Outros trabalhos (LEI *et al.*, 2019; BARKA *et al.*, 2018; KAPETANIDOU; MENDES; TSAOUSSIDIS, 2023) também utilizam VANTs como nós para uma rede ICN, mas com foco em melhorar a segurança, propondo abordagens para mitigar problemas como o envenenamento de conteúdo e autenticidade de dados.

ALDAOUD *et al.* (2023) pesquisaram sobre o uso conjunto de ICN, NFV e Redes Definidas por Software (do inglês *"Software-Defined Networking"* ou SDN). O trabalho conclui que o uso conjunto dessas tecnologias pode melhorar a confiabilidade, escalabilidade, mobilidade, flexibilidade e robustez das redes. ALDAOUD *et al.* (2023) discutem ainda sobre o uso dessas tecnologias no contexto de 5G, 6G e rede de satélites, onde ICN, NFV e SDN têm o potencial para impulsionar o desempenho.

Os trabalhos apresentados nessa seção mostram que o uso das tecnologias NFV, ICN e SDN estão sendo abordados para as mais diversas áreas, o que inclui Internet do Futuro, 5G, *beyond 5G*, 6G e redes de satélites. O uso conjunto dessas tecnologias ocorre principalmente para estabelecer uma rede flexível com suporte a controle de rotas e encaminhamentos, assim como entrega rápida. Em especial, destaca-se que NFV podem impulsionar a implementação de ICN. O interesse de trabalhos recentes em NFV, SDN e ICN, seja de forma conjunta ou separada, indicam a importância dessas tecnologias para redes atuais e futuras. Além disso, destaca-se o uso de ICN para contextos envolvendo mobilidade, especialmente redes baseadas em VANTs.

3.2 Arquitetura de Microsserviços

CHOWDHURY *et al.* (2019) descrevem que um problema fundamental com VNFs monolíticas é o processamento de funcionalidades redundantes em diferentes VNFs, gerando uma sobrecarga de processamento desnecessária quando VNFs são encadeados para formar Encadeamento de Funções de Serviço (do inglês *"Service Function Chaining"* ou SFC). Para solucionar esse problema, CHOWDHURY *et al.* (2019) propõem a restruturação dos VNFs a partir da arquitetura de microsserviços, onde cada componente de função de rede virtual é um microsserviço.

HAWILO; JAMMAL; SHAMI (2019a) também prevê a arquitetura de microsserviços para o desenvolvimento de VNF. Os autores descrevem que essa abordagem: 1) facilita testes de funcionalidades isoladamente; 2) facilita a manutenção com adoção da integra-

ção contínua; 3) aumenta a flexibilidade; e 4) permite escalar componentes de função de rede separadamente, conforme a demanda.

Os trabalhos dessa seção demonstram que a arquitetura de microsserviços é uma opção para o desenvolvimento de VNFs. Essa abordagem concede aos VNFs melhorias na flexibilidade, escalabilidade, manutenção, desenvolvimento e redundância.

3.3 Gerenciamento e orquestração de VNFs

A arquitetura ETSI NFV MANO serve como referência para outros projetos independentes que estão sendo realizados por fornecedores ou por comunidades de código aberto. Entre os esforços que estão sendo despendidos nesse sentido, destacam-se Open Network Automation Platform (ONAP)[1], Open Source MANO (OSM)[2] e Tacker, os quais vem recebendo suporte de grandes empresas do ramo de telecomunicação. Todos implementam VNFM e NFVO da arquitetura ETSI NFV MANO, mas somente ONAP e OSM também implementam VIM. Todavia, ONAP e OSM estão longe de estar completos ou estáveis. Além disso, implementar soluções utilizando essas duas implementações requer experiência nas atuais plataformas de rede de computação em nuvem, o que torna o processo não trivial (YILMA *et al.*, 2020; KAUR; MANGAT; KUMAR, 2022).

CASTILLO LEMA (2019) argumenta que OSM, ONAP e Tacker visam principalmente VNF baseadas em IP. Com isso em mente, é proposta uma arquitetura que lida com a implantação dinâmica de uma rede de função nomeada e serviços de rede para infraestrutura ICN. A abordagem desenvolvida é baseada na linguagem Especificação de Topologia e Orquestração para Aplicativos em Nuvem (do inglês *"Topology and Orchestration Specification for Cloud Applications"* ou TOSCA), que define comportamento das VNF em VMs. Dentre os trabalhos futuros, CASTILLO LEMA (2019) destacam que a solução desenvolvida carece de funcionalidades mais maduras de monitoramento e escala automática.

Cloudify (G.A., 2024) é uma solução que abrange NFVO e VNFM da arquitetura ETSI NFV MANO. Dentre as funcionalidades de cloudify, destaca-se a implantação em ambiente de nuvem, monitoramento, suporte a controladores SDN, detecção de problemas e falhas, correção manual ou automática de ativos e gerenciamento de tarefas de manutenção. No entanto, alguns recursos avançados de cloudify estão disponíveis apenas na edição *premium* (DE CARVALHO; PATRICIA FAVACHO DE ARAUJO, 2020), que é paga.

Em sua revisão sobre gerenciamento e orquestração na arquitetura NFV, (KAUR; MANGAT; KUMAR, 2022) destacaram que a tecnologia de contêineres vem ganhando atenção para desenvolver VNFs, principalmente devido aos ganhos em escalabilidade,

[1] `https://www.onap.org`
[2] `https://osm.etsi.org/`

utilização de recursos e desenvolvimento ágil de *software* apresentado pelos contêineres. Essas características tornam os contêineres mais adequados que VMs quando: 1) os aplicativos foram desenvolvidos a partir da arquitetura de microsserviços; 2) existe a necessidade de minimizar o número de servidores; e 3) os VNFs possuem apenas um sistema operacional. Todavia, (KAUR; MANGAT; KUMAR, 2022) também ressaltam que os contêineres sofrem com problemas de segurança devido ao compartilhamento do Kernel e são uma tecnologia em estágio inicial.

CZIVA; PEZAROS (2017) compararam o desempenho de VMs - XEN dom0, clickOS, KVM virtio, XEN domU, KVMe1000 - e contêineres. Os resultados mostram que os contêineres possuem: 1) o terceiro menor atraso, com KVM e XEN apesentando os menores valores; 2) menor tempo de iniciação, consumindo 10 s para instanciar e iniciar 50 VNFs, enquanto VM XEN consome 40 s apenas para instanciar; 3) o menor consumo de memória, gastando 2,21 MB por VNF, enquanto ClickOS gasta mais que o dobro desse valor. Consequentemente, os contêineres emergem como uma abordagem interessante para cenários com restrição de recursos, como na borda da rede.

Em ambientes onde as VNFs são executadas via contêineres e em um cluster de nós, é comum a utilização de ferramentas como Kubernetes ou Docker Swarm. O Kubernetes é a ferramenta mais utilizada, desempenhando funções relacionadas ao VIM e ao VNFM na arquitetura ETSI NFV MANO (KAUR; MANGAT; KUMAR, 2022). WANG *et al.* (2022) propõem um orquestrador para implantar SFC em um cluster Kubernetes, sendo os SFCs desenvolvidos a partir da arquitetura de microsserviços. Esta solução inclui ainda uma interface Web que possibilita a interação dinâmica de um operador com o sistema, além de um sistema de monitoramento baseado em Prometheu.

MAI *et al.* (2019) apresentam uma proposta para implementar, gerenciar e proteger uma rede NDN, na qual cada nó corresponde a um NFD conteinerizado. A solução é avaliada dinamicamente em um cenário de ataque de envenenamento de conteúdo e escala, este último realizado conforme o tamanho da PIT. Embora o tamanho da PIT forneça uma indicação indireta do estresse de um contêiner, é mais vantajoso, neste caso, escalar com base no consumo de CPU e memória. A escala baseada em CPU e memória é abordagem usual para escalar microsserviços na nuvem.

A presente seção mostrou os avanços em relação ao gerenciamento e orquestração de VNFs. Uma tendência identificada é o uso da arquitetura ETSI NFV MANO para esse contexto, onde OSM e ONAP se destacam. A maioria dos trabalhos de pesquisa anteriores está focada no desenvolvimento de soluções baseadas em VMs. Todavia, vem ganhando atenção o desenvolvimento de VNFs a partir de contêineres. Como os contêineres são mais leves, eles podem ser utilizados em aplicações onde há recursos escassos, como uma rede composta por VANTs. Além disso, as soluções propostas são geralmente direcionadas ou avaliadas no contexto de redes IP, necessitando de ajustes quando se trata de um cenário ICN (CASTILLO LEMA, 2019; MAI *et al.*, 2019).

3.4 μNDN

Concentrando-se nos conceitos de NDN e microsserviços, MARCHAL; CHOLEZ; FESTOR (2018) desenvolveram um conjunto de VNFs NDN implementados a partir da arquitetura de microsserviços, daqui para frente também chamados de microsserviços NDN. Esses microsserviços foram desenvolvidos com base no *software* monolítico Named Data Networking Forwarding Daemon (NFD).

MARCHAL; CHOLEZ; FESTOR (2018) compararam o desempenho dos microsserviços NDN e o monolítico NFD, em que os microsserviços atingiram *throughput* maior e latência menor. Todavia, o desempenho dos microsserviços foi ao custo de um maior consumo de CPU. Sendo assim, há uma compensação que deve ser considerada para o contexto abordado.

Em seu trabalho, MARCHAL; CHOLEZ; FESTOR (2018) também desenvolveram um gerenciador para lidar com operações dinâmicas, as quais se destacam: 1) criar e destruir instâncias de microsserviços; 2) criar e destruir as conexões entre os microsserviços; 3) captura de métricas dos microsserviços; 4) escala horizontal automática baseada no consumo da CPU.

A solução desenvolvida e avaliada por MARCHAL; CHOLEZ; FESTOR (2018) implanta microsserviços em uma máquina local. No entanto, aplicações reais, como Internet, IoT e rede de VANTs, envolvem uma infraestrutura de rede com múltiplos nós. Para resolver esse problema, este trabalho estende o trabalho de MARCHAL; CHOLEZ; FESTOR (2018) para permitir a implantação de microsserviços NDN em um composto por múltiplos nós, superando a restrição de uma única máquina local.

Além disso, este trabalho delineia um novo estudo de caso que envolve uma aplicação de vigilância em uma missão militar, onde os microsserviços NDN são implantados sobre demanda e com conhecimento de recursos.

3.5 Problema de Posicionamento para Execuções Encadeadas

A execução de uma aplicação ou serviço a partir do encadeamento de componentes distribuídos, como no caso de VNFs e microsserviços, sofre com o problema de posicionamento (SANTOS *et al.*, 2020; HAWILO; JAMMAL; SHAMI, 2019b). Esse problema de posicionamento refere-se à decisão de onde implantar um certo componente de modo a maximizar desempenho e minimizar custos, como, respectivamente, latência de comunicação e consumo de recursos físicos.

SANTOS *et al.* (2020) propuseram um método para posicionamento de cadeias de serviços em ambientes de névoa ("fog computing"), especificamente adaptado para casos de uso de cidades inteligentes. O método possui duas políticas, uma com reconhecimento de latência e outra com reconhecimento de local. A política com reconhecimento de latência seleciona o melhor nó candidato utilizando Dijkstra. Por outro lado, a política

de reconhecimento de localização se baseia na minimização da latência dependendo do rótulo de local de destino. Os resultados mostram que a abordagem proposta reduz a latência da rede em até 18% quando comparada ao scheduler padrão do Kubernetes.

Ainda no âmbito do ambiente de névoa, SANTOS *et al.* (2019) propuseram um método de posicionamento com reconhecimento de rede para Kubernetes, onde as variáveis consideradas foram o Tempo de Ida e Volta (do inglês *"Round Trip Time"* ou RTT) e largura de banda requisitada. Comparada ao scheduler padrão do Kubernetes, a solução obteve até 80% de redução na latência da rede.

HAWILO; JAMMAL; SHAMI (2019b) modelaram e desenvolvem uma programação linear inteira mista e uma heurística para posicionamento de VNF. A solução proposta busca minimizar atrasos entre as instâncias VNF e ponta a ponta do SFC. Ambos os métodos obtiveram melhor desempenho quando comparados a uma abordagem gananciosa.

Os trabalhos dessa seção revelam uma lacuna no método de posicionamento padrão do Kubernetes, que não considera parâmetros de rede, podendo acarretar atrasos indesejados. Para resolver esse problema, os autores propuseram métodos para posicionamento que consideram parâmetros como RTT e o tempo de latência entre contêineres e nós.

3.6 Conclusão

Este capítulo resumiu a literatura relevante para essa dissertação. Na seção 3.1, os trabalhos abordados tratam dos conceitos NFV, SDN e ICN, os quais têm uso planejado para aplicações como Internet do futuro, Beyond 5G e 6G. Além disso, foi destacado o uso de ICN para VANTs, contexto que faz parte de um dos estudos de caso abordado por este trabalho. Na sequência, seção 3.2, foi abordado o uso de NFV desenvolvidos a partir da arquitetura de microsserviços, os quais podem promover melhor flexibilidade e escalabilidade.

O foco principal deste trabalho é o gerenciamento e orquestração de microsserviços ICN. Portanto, na seção 3.3, são apresentados alguns trabalhos para gerenciamento e orquestração de VNFs, onde verificou que as soluções são geralmente idealizadas para uso em redes IPs, sendo que abordagens baseadas em IP necessitam ser remodeladas para uso em um contexto ICN. Além disso, foi identificado que contêineres podem ser utilizados para desenvolver VNFs mais leves e rápidas.

Na seção 3.4, é apresentado o uNDN, uma solução para orquestração de microsserviços ICN. Mais especificamente, este trabalho toma como principal referência uNDN, estendendo-o para um contexto com múltiplos nós.

Por fim, trabalhos que abordam a temática de posicionamento foram tratados na seção 3.5. Nessa seção, é discutido o problema do Kubernetes não considerar parâmetros de rede para posicionar Pods, o que pode provocar latência. Assim como os trabalhos dessa seção, o cenário abordado por este trabalho necessita minimizar o atraso na entrega de

mensagens, já que um alto atraso incorre em um *throughput* menor.

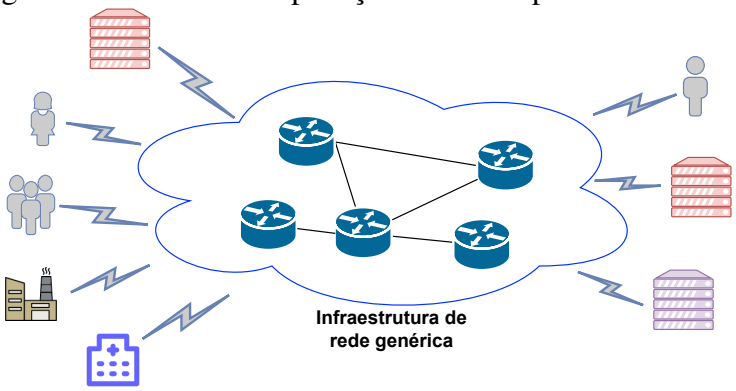
Em suma, a maioria das soluções propostas para implantar e gerir uma rede a partir de funções de rede são direcionadas para redes IP, como OSM, ONAP e Tacker. Devido às diferenças conceituais entre IP e ICN, essas abordagens não podem ser utilizadas para ambos os contextos, consequentemente, ICN fica sem suporte adequado. Além disso, mesmo as soluções com foco em redes ICN, não consideram uma abordagem baseada na arquitetura de microsserviços CASTILLO LEMA (2019). Até onde é do nosso conhecimento, apenas MARCHAL; CHOLEZ; FESTOR (2018) considera tanto ICN quanto microsserviços. No entanto, a solução desenvolvida por MARCHAL; CHOLEZ; FESTOR (2018) é limitada a um único nó. Este trabalho aborda essa lacuna, propondo uma arquitetura para implantar e controlar uma rede ICN em um cluster de nós.

4 SOLUÇÃO PROPOSTA

A Figura 12 ilustra o cenário de aplicação abordado neste trabalho. Em ambos os lados têm-se consumidores e produtores de conteúdo, que são conectados a partir de um conjunto de dispositivos de rede, representados por roteadores. Como abordado anteriormente, ICN traz certas vantagens em relação à abordagem tradicional IP. Nesse sentido, considera-se que o operador da rede procura implantar uma rede ICN a partir dos dispositivos de rede disponíveis.

Considera-se ainda que a demanda por conteúdo é dinâmica e que a rede deve ser capaz de ser ajustada sob demanda. Para que a rede tenha essas características, é necessário manipular a rede e seus recursos de maneira rápida, o que é possível a partir de NFV. Logo, nesse cenário, os dispositivos da rede devem implantar VNFs ICN para estabelecer uma rede NDN.

Figura 12 – Cenário de aplicação abordado por este trabalho.



Fonte: do autor.

O cenário apresentado na Figura 12 é um cenário genérico, o qual consiste no estabelecimento e controle de uma rede NDN. Esse contexto tem potencial para abranger diversas aplicações, como IoT (NOUR *et al.*, 2019; ZHANG *et al.*, 2023; MARS *et al.*, 2019), 5G e 6G (ALDAOUD *et al.*, 2023; GüR *et al.*, 2022), saúde (NOUR *et al.*, 2017; ALOURANI *et al.*, 2023), rede de FANETs (BARKA *et al.*, 2018; KAPETANIDOU; MENDES; TSAOUSSIDIS, 2023), IoT Industrial (FREY *et al.*, 2019) entre outros estudos de caso. Procurando esclarecer essa questão, o capítulo 5 apresenta dois estudos de

caso mais específicos, um voltado para o escala automática e outro para recuperação de dados de vídeo durante uma missão militar.

4.1 Micro-Chain

Este trabalho considera que a rede ICN da Figura 12 é implantada a partir de microsserviços NDN. Um cenário de aplicação como esse exige uma maneira para gerenciar e orquestrar microsserviços NDN de modo a atingir o comportamento esperado. Esse problema é solucionado neste trabalho a partir do planejamento e implementação de uma arquitetura denominada de Micro-Chain.

Uma visão geral da arquitetura Micro-Chain é mostrada na Figura 13, onde estão especificados quatro módulos e os seus relacionamentos, sendo a comunicação entre os módulos realizada por APIs. Mais especificamente, os módulos são:

- **Microsserviços:** são os VNFs desenvolvidos a partir da arquitetura de microsserviços. Na Figura 13 estão representados como retângulos com prefixo *micro*, como micro1 e micro2. Um conjunto de microsserviços em sequência consiste em uma cadeia de microsserviços ou, de maneira equivalente, em um encadeamento de função de rede.

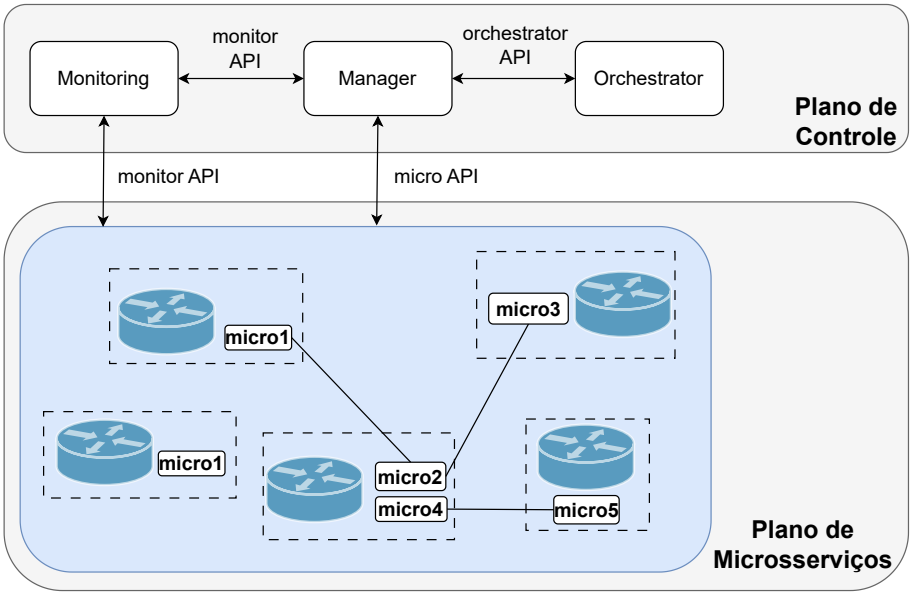
- **Monitoring:** adquire, armazena, processa e expõe métricas de monitoramento por meio de monitor API. Este módulo trata de métricas sem restrições temporais que precisam ser armazenadas. Conforme a necessidade, o módulo Manager pode solicitar os dados capturados pelo Monitoring para utilizar em alguma lógica interna, como a escala de microsserviços.

- **Manager:** é o ponto central da arquitetura, responsável por mediar as operações do sistema. É ele o responsável por implementar a lógica de negócio. Algumas das atribuições desse módulo são: 1) solicitar que o Orchestrator crie/exclua um microsserviço via orchestrator API; 2) definir o posicionamento para microsserviços durante a implantação; 3) configurar os microsserviços, como a conexão entre os microsserviços via micro API; 4) lidar com métricas de monitoramento com requisitos de rápido acesso; e 5) executar as etapas para escalar microsserviços. Além disso, o Manager também possui uma aplicação Web que permite um operador visualizar e modificar os microsserviços.

- **Orchestrator:** é responsável por implantar e garantir a integridade dos contêineres, mesmo diante de falhas.

A descrição acima realizada de de Micro-Chain é conceitual. Para melhor compreensão do funcionamento da arquitetura, as próximas seções explicam melhor cada módulo

Figura 13 – Representação da arquitetura Micro-Chain.



Fonte: do autor.

em termos de implementação. Todavia, vale ressaltar que a arquitetura não se limita a esse modo de implementação, isto é, as tecnologias utilizadas.

4.2 Microsserviços

Foram aproveitados os microsserviços desenvolvidos e disponibilizados online[1] por MARCHAL; CHOLEZ; FESTOR (2018), sendo realizadas modificações na implementação de modo a ajustar as operações para o contexto de Micro-Chain. Mais especificamente, os microsserviços utilizados foram:

- **Name Router (NR)**: faz o papel da FIB do NDN. Quando chega um pacote de interesse, esse microsserviço encaminha em direção ao produtor.
- **Backward Router (BR)**: faz o papel da PIT do NDN. Salva a entrada de pacotes de interesse e utiliza essa informação para encaminhamento reverso de pacotes de dados.
- **Content Store (CS)**: é equivalente a CS do NDN. Armazena pacotes de dados para recuperação posterior.
- **Strategy Forwarder (SF)**: distribui pacotes de interesse para impedir sobrecarga de instâncias de microsserviço. As estratégias utilizadas por SF podem ser balanceamento de carga, *failover* ou outras mais específicas.

[1] `https://github.com/Nayald/NDN-microservices`

Cada microsserviço possui interfaces de ingresso e egresso para a comunicação NDN. As interfaces de ingresso de um microsserviço são as interfaces onde ele é o destino, enquanto as interfaces de egresso correspondem às interfaces onde ele é a fonte. Para melhor compreensão, considere a cadeia de microsserviços da Figura 14, onde as setas representam o sentido para pacotes de interesse. Nesse exemplo, CS1 possui face 1 como interface de ingresso e face 2 como interface de egresso.

Figura 14 – Interfaces do microsserviço CS1.

Fonte: do autor.

Cada microsserviço possui duas características: orientação e cardinalidade. Um microsserviço orientado processa apenas interesse ou dados em suas interfaces, enquanto um não-orientado pode processar ambos. Por outro lado, a cardinalidade corresponde ao número de fontes ou destinos diferentes que um microsserviço pode distinguir ao encaminhar. Uma cardinalidade de "1"significa que o microsserviço encaminha pacotes para todas as suas interfaces de ingresso ou egresso (multicast). Uma cardinalidade igual a "N"significa que um microsserviço é capaz de encaminhar pacotes para a(s) fonte(s) ou destino(s) específicos, o que implica manter alguma informação de rota. A Tabela 1 apresenta a orientação e cardinalidade dos microsserviços NDN.

Tabela 1 – Orientação e cardinalidade dos microsserviços NDN.

Nome	Orientação	Cardinalidade (ingresso/egresso)
Name Router	Sim	1/N
Backward Router	Sim	N/1
Content Store	Não	1/1

Fonte: Modificado de (MARCHAL; CHOLEZ; FESTOR, 2018).

Como forma de exemplificar o funcionamento da orientação e cardinalidade, considere o microsserviço CS. A Figura 15 apresenta o pseudocódigo do comportamento de CS para a interface de ingresso. Se CS recebe um pacote de interesse na interface de ingresso, ele verifica se possui o conteúdo solicitado armazenado e, se sim, responde com um pacote de dados contendo o conteúdo via interface do pacote de interesse (linha 1 à 4). Caso contrário, CS envia o pacote para a interface(s) de egresso (linha 5 à 7). Por outro lado, se CS receber um pacote de dados, ele salva localmente o pacote e encaminha diretamente para a(s) interface(s) de egresso (linha 9 à 12). Para a interface de egresso, as etapas são similares às executadas para a interface de ingresso, mas os encaminhamentos (linha 6 e 11) são realizados para as interfaces de ingresso.

Figura 15 – Etapas executadas ao receber pacote NDN na interface de ingresso.

```
1  se pacote de interesse então
2  |    se conteúdo armazenado localmente então
3  |    |    Encaminha pacote de dados via interface do pacote de interesse
4  |    fim
5  |    senão
6  |    |    Encaminha pacote de interesse via interface(s) de egresso
7  |    fim
8  fim
9  se pacote de dados então
10 |    Salva o pacote localmente
11 |    Encaminha o pacote de dados via interface(s) de egresso
12 fim
```

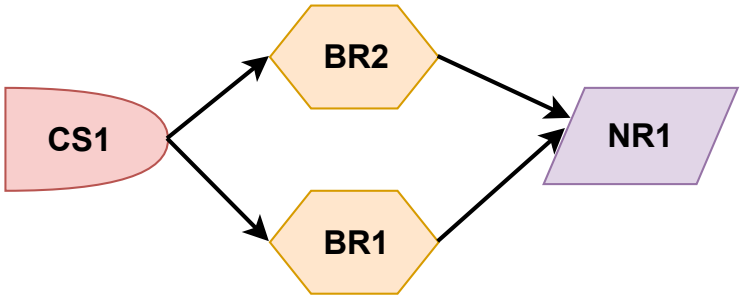
Fonte: do autor.

Conforme exemplificado, CS processa tanto pacotes de interesse quanto de dados em suas interfaces, tornando-o não orientado. Além disso, ele encaminha pacotes para todas as suas interfaces de ingresso ou egresso, o que o classifica como tendo cardinalidade 1/1. Como meio comparativo, BR processa apenas pacotes de interesse nas interfaces de ingresso e pacotes de dados nas interfaces de egresso (orientado). Além disso, o envio via interfaces de ingresso é direcionado para interfaces específicas, conforme entradas na PIT (cardinalidade N). Por outro lado, o envio através das interfaces de egresso é multicast (cardinalidade 1).

Essa dinâmica complexa envolvendo orientação e cardinalidade torna a implantação desses microsserviços não trivial. A Figura 16 apresenta uma configuração errônea que poderia ser implantada para o aumento de escala de BR1. Como CS1 possui cardinalidade para egresso igual a N, um pacote de interesse recebido em uma interface de ingresso por CS1 seria encaminhado para as duas instâncias de BR. Consequentemente, dois pacotes de interesse iguais trafegariam na rede, resultando em desperdício de recursos. Nesse caso, deve ser usado o microsserviço SF para dividir o fluxo de dados. Observe que o estabelecimento e manipulação de uma rede NDN a partir desses microsserviços exige conhecimento adequado do funcionamento dos mesmos. É necessário conhecer o que pode ser ligado com o que para atingir certas características desejadas e em que ordem eles devem ser ligados.

Todos os microsserviços aceitam pelo menos três operações: 1) criação de uma nova interface, utilizada para criar conexões entre os microsserviços; 2) a exclusão de uma interface; 3) alteração de configurações, as quais dependem do tipo de microsserviço. Além disso, os microsserviços podem ter outras operações específicas. O microsserviço NR, por exemplo, aceita operações para configurar o encaminhamento até servidores de

Figura 16 – Configuração de microsserviços para o aumento de escala de BR1 com duplicidade de pacotes.



Fonte: do autor.

conteúdo, o que é utilizado pelo Manager para controlar as rotas.

4.3 Módulo Manager

O Manager e micro API foram desenvolvidos com base em (MARCHAL; CHOLEZ; FESTOR, 2018), também sendo realizadas modificações na implementação para ajustar as operações de Micro-Chain.

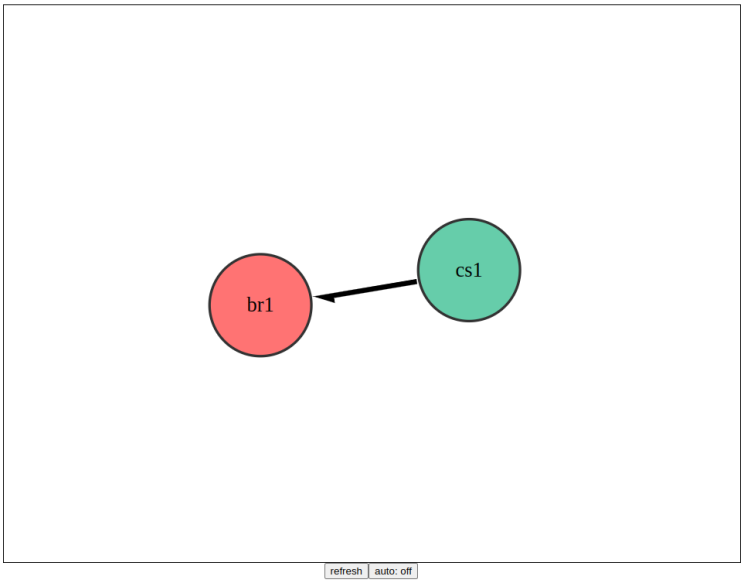
Dentre as capacidades do Manager, destaca-se a criação e manutenção de dois grafos, um com a representação da topologia dos microsserviços e outro com os nós da rede. Cada grafo armazena informações de interesse necessárias para o gerenciamento, como consumo de CPU e memória, nome e rotas. Além disso, o Manager também possui uma aplicação web que permite a representação gráfica e textual de informações armazenadas no grafo de microsserviços. Um operador pode usar essa aplicação para operações manuais, como criar ou conectar microsserviços.

A Figura 17 e Figura 18 apresentam fotos tiradas da aplicação web. A Figura 17 mostra o grafo que o operador pode utilizar para visualizar a cadeia de microsserviços implantada nos dispositivos. Já a Figura 18 mostra outra parte da aplicação web, onde o operador pode visualizar dados textuais e executar operações.

Uma das funcionalidades executadas pelo Manager é a escala de microsserviços, o qual possui algumas limitações. Mais especificamente, um microsserviço deve atender as seguintes condições para escalar: 1) não deve estar na borda da rede, para evitar a quebra de conexões fora da rede gerenciada; 2) deve ser definido como escalável; e 3) deve ter um consumo de CPU e memória acima do limite estabelecido, sendo que cada microsserviço pode ter um limite específico.

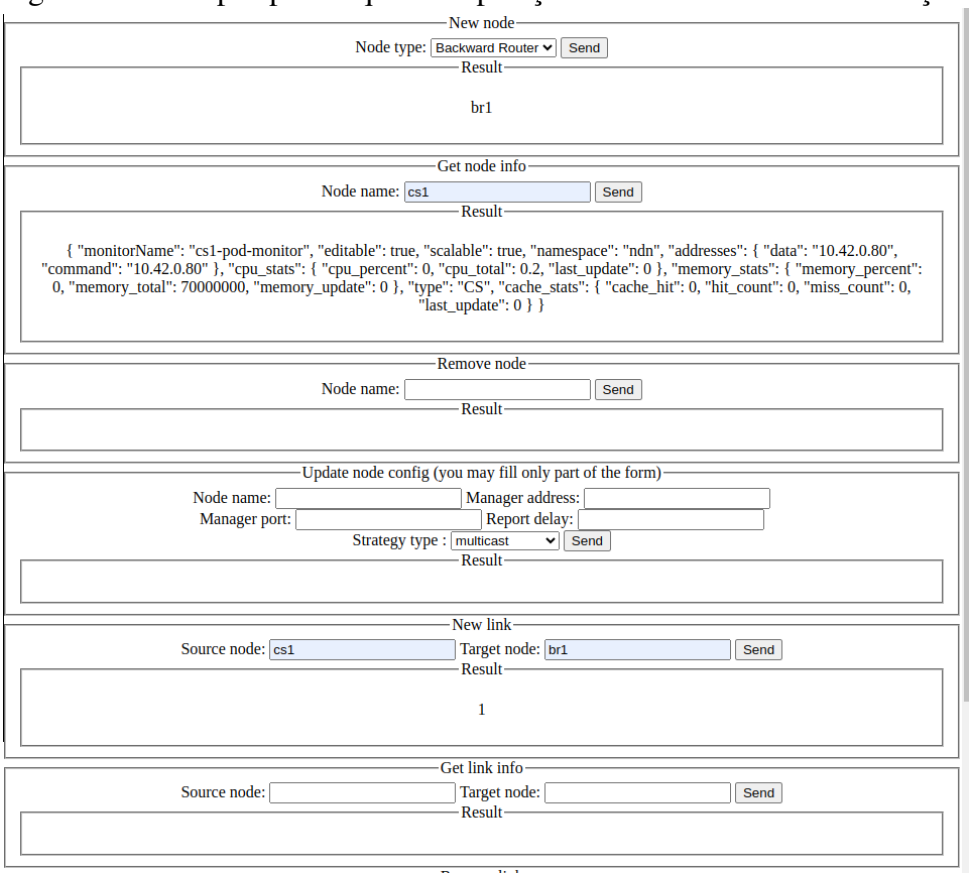
A Figura 19 exemplifica o processo de escala a partir da ampliação de BR1. Durante esse processo, sete operações são realizadas: implantação de SF1 (1); conexão entre SF1 e BR1 (2); conexão entre CS1 e SF1 (3); exclusão da conexão entre CS1 e BR1 (4); implantação de BR2 (5); conexão entre BR2 e NR1 (6); e, por fim, conexão entre SF1 e BR2 (7). Vale ressaltar que a quebra momentânea da cadeia ou duplicação de pacotes

Figura 17 – Grafo de microsserviços.



Fonte: do autor.

Figura 18 – Campos para requisitar operações ou dados dos microsserviços.

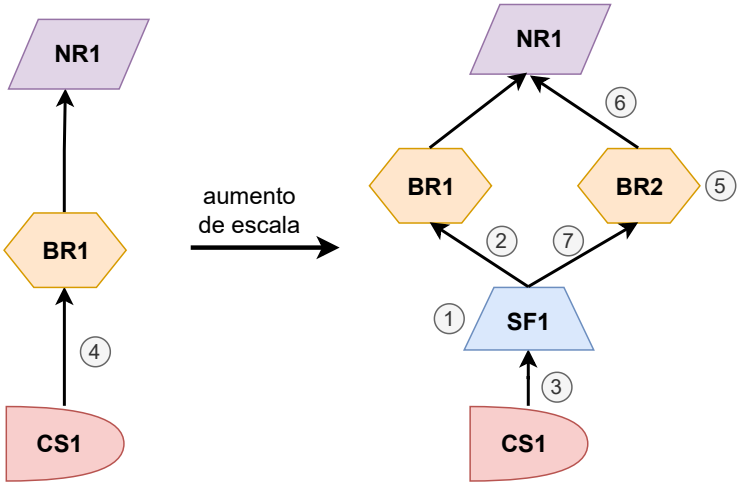


Fonte: do autor.

pode resultar em perda de pacotes durante o processo.

Para a comunicação entre Manager e microsserviços, são utilizadas mensagens no formato JSON, enviadas via UDP (micro API). Todas as mensagens contêm pelo menos

Figura 19 – Etapas para aumento de escala de BR1.



Fonte: Modificado de (MARCHAL; CHOLEZ; FESTOR, 2018).

um campo "action"e "id". O campo "action"é usado para mapear mensagens para funções específicas e o campo "id"é um número inteiro utilizado para identificar a operação. Um microsserviço irá copiar o id em sua resposta e o Manager irá utilizá-lo para saber a qual solicitação essa resposta está associada.

A Figura 20 e Figura 21 exemplificam o formato das mensagens trocadas durante a criação de uma conexão entre microsserviços. Na Figura 20 tem-se o formato da mensagem enviada pelo Manager para cs1, o microsserviço de origem. Essa mensagem contém o endereço e a porta do microsserviço de destino, ao qual cs1 é solicitado a criar uma interface. Enquanto na Figura 21 tem-se a reposta enviada por cs1 para o Manager, onde é informado que a conexão foi estabelecida com sucesso (status).

Figura 20 – Exemplo de mensagem enviada pelo Manager para requisitar uma nova face.

```
{
    "action": "add_face",
    "id": 1,
    "layer": "tcp",
    "address": "172.18.0.5",
    "port": 6363
}
```

Fonte: Modificado de (MARCHAL; CHOLEZ; FESTOR, 2018).

O Manager gerencia centralmente as rotas disponíveis na rede e as muda diretamente para os módulos relevantes, como um controlador SDN. Essa ação é executada sempre que um provedor de conteúdo envia uma solicitação de rota ou sai da rede, ou, ainda, quando conexões são criados.

Figura 21 – Exemplo de mensagem enviada pelo microsserviço cs1 em resposta a requisição do Manager.

```
{
    "name": "cs1"
    "type": reply,
    "action": "add_face",
    "id": 1,
    "status": "success",
    "face_id": 1
}
```

Fonte: Modificado de (MARCHAL; CHOLEZ; FESTOR, 2018).

4.4 Módulos Monitoring e Orchestrator

A implementação de Micro-Chain utiliza Prometheus[2] como módulo Monitoring e um cliente do Prometheus[3] como monitor API. Mais especificamente, o Prometheus utiliza uma aquisição de dados do tipo pull para obter métricas ativamente via requisições periódicas. Para isso, o microsserviço monitorado deve executar um servidor que responde às requisições do Prometheus.

MARCHAL; CHOLEZ; FESTOR (2018) sugerem métricas que podem ser monitoradas pelos microsserviços desenvolvidos por eles. Particularmente, estatísticas de rota para NR; pacotes de dados não solicitados e pacotes de interesse retransmitidos para BR; e contadores de acertos e erros para CS.

Em Micro-Chain, o módulo Monitoring apenas captura, trata e expõe as métricas, isto é, ele não executa nenhuma tomada de decisão a partir das métricas capturadas. Esse tipo de operação é realizada pelo Manager. Considerando esse contexto, em Micro-Chain o processo de monitoramento de métricas pode ser realizado de três maneiras: diretamente pelo módulo Monitoring, diretamente pelo Manager ou usando esses dois módulos. O processo padrão de captura de métricas é via Monitoring. Nesse caso, o Manager, como tomador de decisão, precisa solicitar as métricas capturadas pelo Monitoring para depois executar suas operações, o que intrinsecamente insere latência. Por isso, outra forma de captura é diretamente pelo Manager, o que proporciona uma tomada de decisão mais rápida.

Além de Prometheus, a implementação de Micro-Chain utiliza K3s[4] como módulo Orchestrator e um cliente do Kubernetes[5] como orchestrator API. Em particular, Kubernetes

[2] `https://prometheus.io/`
[3] `https://github.com/prometheus/client_python`
[4] `https://k3s.io/`
[5] `https://github.com/kubernetes-client/python`

e Prometheus possuem uma integração, o que facilita o uso conjunto dessas ferramentas. Além disso, no geral, essas ferramentas apresentam um bom suporte de material didático e atualizações periódicas, consequentemente o desenvolvimento é acelerado.

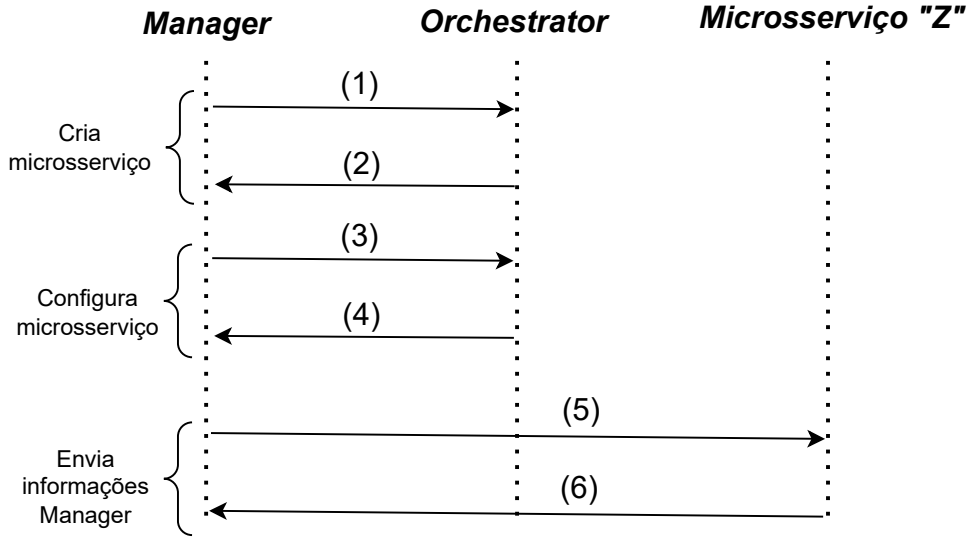
Os módulos Manager, Monitoring e os microsserviços são desenvolvidos em contêineres e implantados dentro do cluster K3s. No futuro, isso permitirá que módulos como Manager e Monitoring sejam dimensionados sob demanda. Além disso, essa abordagem aproveita a robustez de K3s para reiniciar módulos automaticamente em caso de falha.

4.5 Operações Fundamentais

Entre as operações fundamentais que podem ser realizadas a partir de Micro-Chain, se destacam: 1) a implantação e exclusão de microsserviços; 2) a aquisição de métricas; 3) e a criação e exclusão de conexões entre microsserviços, sendo essas operações a base para executar outras operações. A seguir, essas operações são descritas com mais detalhes.

A Figura 22 mostra a operação de implantação de um microsserviço "Z". Para isso, o Manager solicita ao Orchestrator, via orchestrator API, a criação do microsserviço (1). Nessa solicitação, são fornecidas informações como nome da imagem e nome do nó de implantação. Ao receber a solicitação, o Orchestrator executa operações no nó para baixar e configurar o novo microsserviço conforme informações enviadas pelo Manager. Quando este processo é concluído, o Orquestrador salva o estado e comunica o resultado da operação ao Manager (2), que salva o estado em um grafo local. Um processo similar acontece para o processo de destruição de um microsserviço.

Figura 22 – Diagrama com etapas para implantação e configuração de um microsserviço.



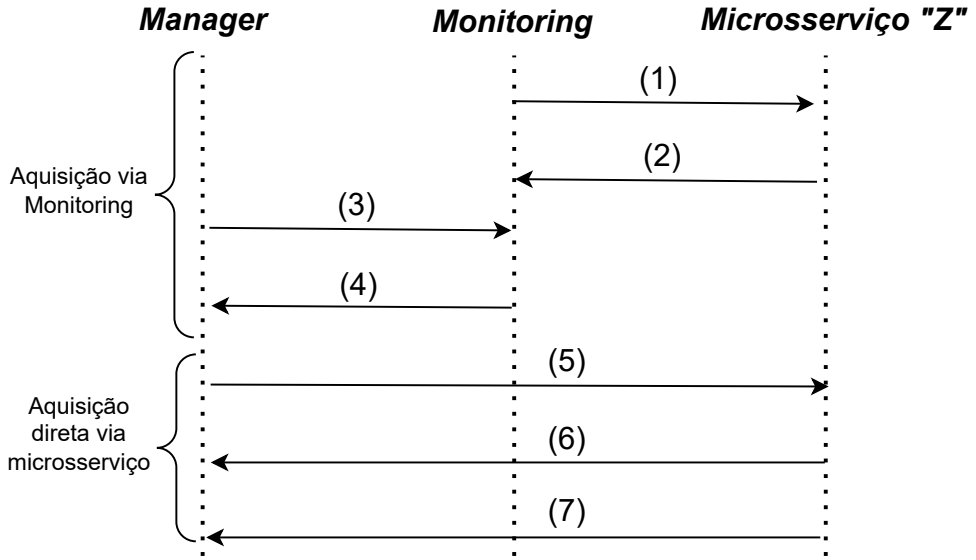
Fonte: do autor.

Após implantar o microsserviço "Z", operações (1) e (2) da Figura 22, o Manager verifica o tipo de microsserviço e configura o Orchestrator com detalhes de monitoramento (3

e 4), como porta e intervalo de captura. Na sequência, o Manager transmite seu endereço IP e porta para o microsserviço via micro API (5), o que permite que o microsserviço inicie uma comunicação com o Manager quando necessário. Por fim, o microsserviço realiza a configuração e responde informando o resultado da operação (6).

A Figura 23 ilustra as etapas realizadas pelo Manager para aquisição de métricas do microsserviço "Z", que podem ser realizadas de duas formas, mediante Monitoring ou diretamente pelo Manager. Na primeira forma, o Monitoring envia solicitações periódicas para capturar as métricas expostas pelo microsserviço (1 e 2). O Manager, por sua vez, adquire as métricas capturadas pelo Monitoring via monitor API (3 e 4).

Figura 23 – Diagrama com etapas para configuração e captura de métricas pelo Manager.



Fonte: do autor.

Na segunda forma para aquisição de métricas, o Manager inicia enviando um comando de configuração via micro API, Figura 23 (5). O comando solicita que o microsserviço reporte uma certa métrica a uma dada taxa. Ao receber essa solicitação, o microsserviço executa a operação e responde com o resultado (6). Se a operação foi bem sucedida, o microsserviço envia periodicamente os valores aferidos via micro API (7). A diferença entre a primeira e a segunda forma é que a primeira forma leva sempre quatro etapas para ser concluída, enquanto a segunda leva uma após a configuração. Essa é uma das justificativas para a aquisição diretamente pelo Manager ser mais rápida.

Para criação de uma conexão entre dois microsserviços, o Manager envia uma mensagem de configuração via micro API para o microsserviço de origem informando o IP e a porta do microsserviço de destino. Ao receber a mensagem, o microsserviço executa a operação a partir dos dados informados. Na sequência, o microsserviço responde com o resultado da operação e salva o estado. O processo de exclusão de uma conexão acontece de forma similar.

5 ESTUDOS DE CASO E RESULTADOS

Para validar a solução proposta, dois estudos de caso são abordados. O primeiro lida com o posicionamento e escala automática de microsserviços. Enquanto o segundo trata de uma aplicação de vigilância baseada em múltiplos VANTs. As subseções a seguir detalham cada um dos estudos de caso, os resultados e uma discussão geral sobre a solução desenvolvida.

5.1 Escala e Posicionamento de Microsserviços

Essa seção aborda o estudo de caso para escala horizontal automática de um microsserviço, similar ao executado por MARCHAL; CHOLEZ; FESTOR (2018). Além disso, esse cenário também aborda o problema de posicionamento de microsserviços.

Uma característica fundamental de soluções baseadas em microsserviços é sua capacidade de escalar dinamicamente. Isso inclui a capacidade de aumento de escala para microsserviços mais lentos e diminuição de escala para microsserviços ociosos. Em Micro-Chain, todos os módulos da arquitetura são utilizados durante uma situação de escala. Sendo assim, considera-se esta situação como um estudo de caso para validar Micro-Chain.

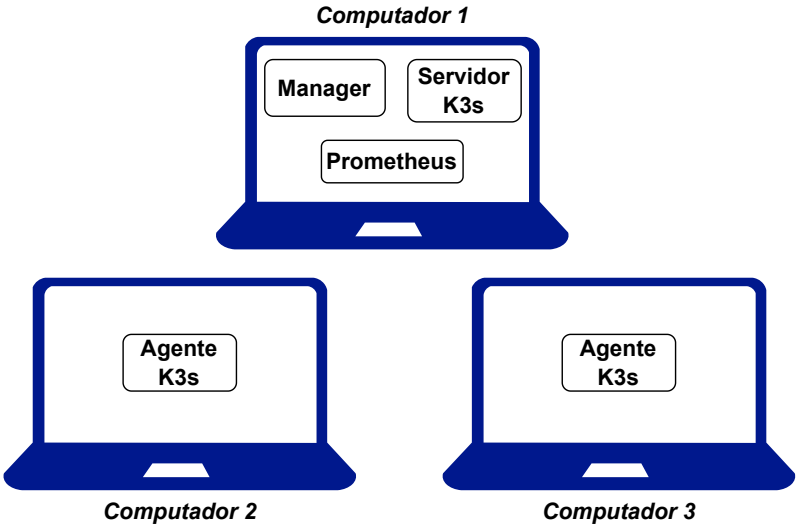
A implantação dos microsserviços enfrenta o problema de posicionamento, onde o posicionamento inadequado pode gerar degradação na qualidade da comunicação. Vale ressaltar que o problema de posicionamento de microsserviços não é foco desse trabalho, logo, ele não é formulado e tratado de maneira formal. Mais especificamente, o objetivo é demonstrar o que acontece com o desempenho da rede quando os microsserviços são posicionados de maneira diferente.

5.1.1 Configuração Experimental

A Figura 24 mostra uma visão geral do ambiente experimental, onde os notebooks representam os nós da rede e os retângulos os recursos de *software* instalados em cada dispositivo. Mais especificamente, o *Computador 1* possui um servidor K3s com Manager e os principais recursos do Prometheus instalados, enquanto o *Computador 2* e

Computador 3 possuem um agente K3s. Ainda, um produtor e um consumidor NDN são instalados em *Computador 2* e/ou *Computador 3* para geração de pacotes NDN. Destaca-se que todos os computadores estão conectados por uma mesma rede local.

Figura 24 – Ambiente experimental, estudo de caso escala de microsserviço.



Fonte: do autor.

A Tabela 2, apresenta as especificações técnicas de cada computador utilizado nos experimentos, sendo que a diferença entre os recursos tem como objetivo demonstrar o funcionamento da arquitetura em um sistema heterogêneo. Destaca-se, ainda, que o baixo nível de recursos do computador 2 é o que justifica ele ser iniciado com apenas um agente K3s instalado.

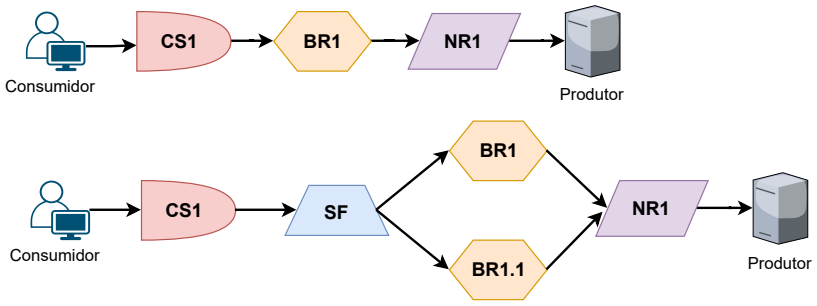
Tabela 2 – Especificação dos nós da Figura 24.

Dispositivo	Descrição
Computador 1	Intel Core i3-6100 com 2 núcleos e 4 threads, frequência base de 3,70 GHz; 8 GB de memória RAM.
Computador 2	Intel Core i5-3210M com 4 núcleos e 2 threads, frequência base de 2,60 GHz; 6 GB de memória RAM.
Computador 3	AMD Ryzen 7 6800H com 8 núcleos e 8 threads, frequência base de 3,2 GHz; 16 GB de memória RAM.

A Figura 25 mostra a configuração esperada inicialmente dos microsserviços na parte superior e a configuração após o aumento de escalada na parte inferior. A configuração inicial possui os microsserviços do tipo CS, BR e NR, onde CS atua exclusivamente como encaminhador, isto é, não responde com conteúdo armazenado localmente. Esses microsserviços foram escolhidos por corresponderem às principais funcionalidades do NDN, nomeadamente CS, FIB e PIT. O experimento se concentra no aumento e diminuição de

escala de BR1 a partir de limites pré-definidos para consumo de CPU e memória. Mais especificamente, CPU = 70% e memória = 70% para aumento de escala, e CPU = 35% e memória = 35% para diminuição de escala. O foco em BR1 se deve a ele ser considerado um gargalo, já que exige o maior consumo de CPU entre os microsserviços utilizados e possui o menor *throughput* (MARCHAL; CHOLEZ; FESTOR, 2018).

Figura 25 – Configuração inicial dos microsserviços para o cenário de escala.



Fonte: do autor.

Esse estudo de caso busca responder às seguintes perguntas: **P1:** O posicionamento dos microsserviços afeta a performance? **P2:** A solução implementada consegue realizar a escala automática de microsserviços? **P3:** Qual é a sobrecarga de comunicação inserida pelo plano de controle de Micro-Chain?

Para responder a essas perguntas, são definidos dois cenários. O primeiro lida com o problema de posicionamento, onde é avaliado o RTT do consumidor para duas configurações de microsserviços com diferentes quantidades de saltos. Enquanto o segundo cenário busca validar o processo de escala por meio do consumo de CPU e memória, além de demonstrar a sobrecarga de comunicação inserida pelo plano de controle durante esse processo.

5.1.2 Resultados

A presente seção apresentou os resultados obtidos para os dois cenários do estudo de caso de escala e posicionamento de microsserviços.

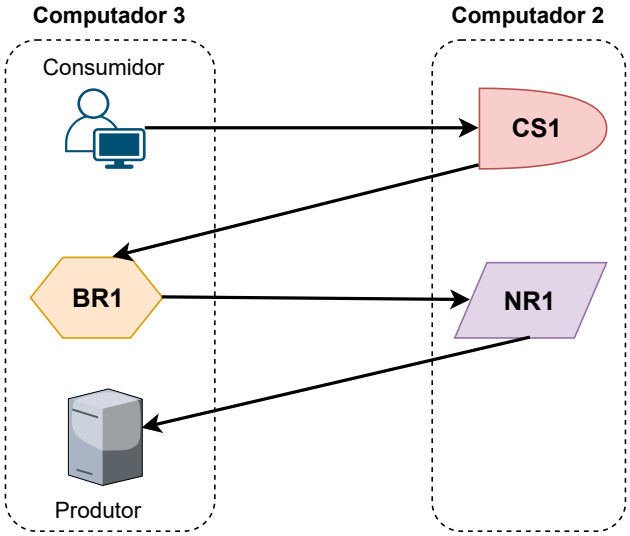
5.1.2.1 Posicionamento de Microsserviços

Para avaliar o problema de posicionamento de microsserviços e seu impacto no desempenho da comunicação, dois contextos são abordados. Para ambos os contextos, são avaliados o *throughput* do consumidor e é considerado que cada computador pode implantar no máximo dois microsserviços.

Para o primeiro contexto, os microsserviços foram posicionados conforme demonstrado na Figura 26. Nessa configuração, os pacotes de interesse e dados do NDN requerem quatro saltos entre dispositivos para trafegarem do consumidor ao produtor. A Figura 27

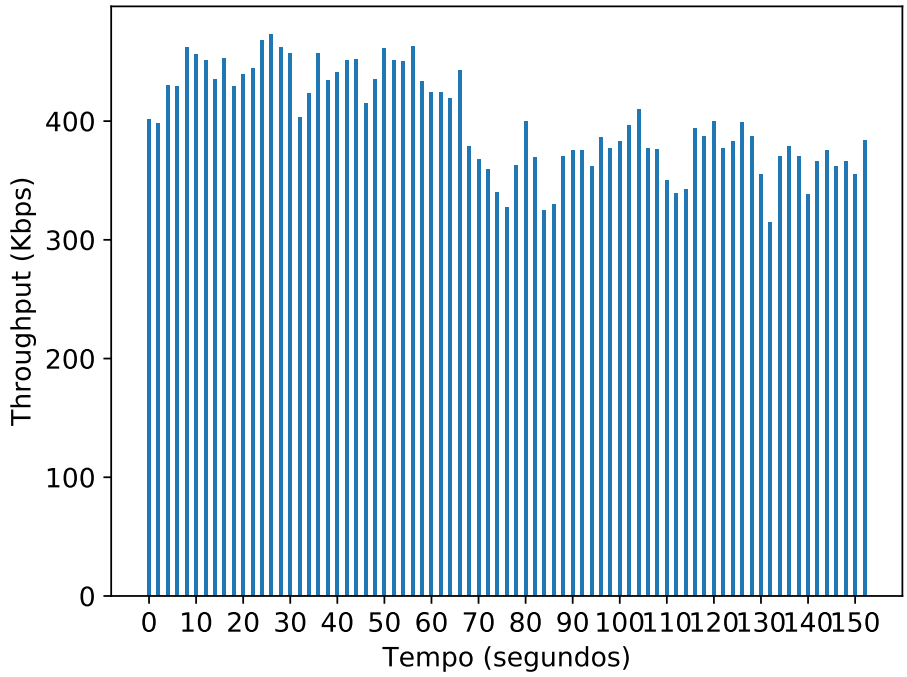
apresenta o *throughput* do consumidor para essa configuração, onde o *throughput* médio é 400,0779 Kbps.

Figura 26 – Ilustra a configuração de microsserviços onde são necessários quatro saltos.



Fonte: do autor.

Figura 27 – *Throughput* do consumidor a cada 2 segundos para o caso com quatro saltos.

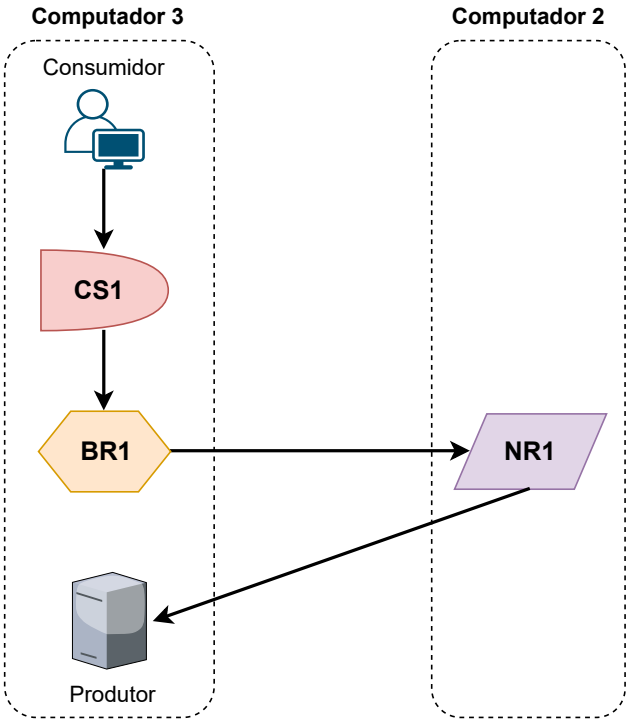


Fonte: do autor.

A Figura 28 apresenta a configuração dos microsserviços para o segundo contexto, onde dois salto entre dispositivos é necessário para uma comunicação entre consumidor e

produtor. O *throughput* para essa configuração é mostrado na Figura 29, onde o *throughput* médio é igual a 707,6875 Kbps. Esse valor é 76% superior ao valor alcançado pela configuração da Figura 27. Logo, respondendo a P1, o posicionamento afeta a performance da rede. Esse resultado pode ser atribuído ao aumento do atraso entre o cliente e o servidor, onde três fatores devem ter contribuído: (1) aumento no número de saltos, (2) inserção de um novo microsserviço na cadeia, o SR, e (3) velocidade de processamento, uma vez que os computadores utilizados possuem recursos heterogêneos.

Figura 28 – Configuração de microsserviços. É necessário dois salto.



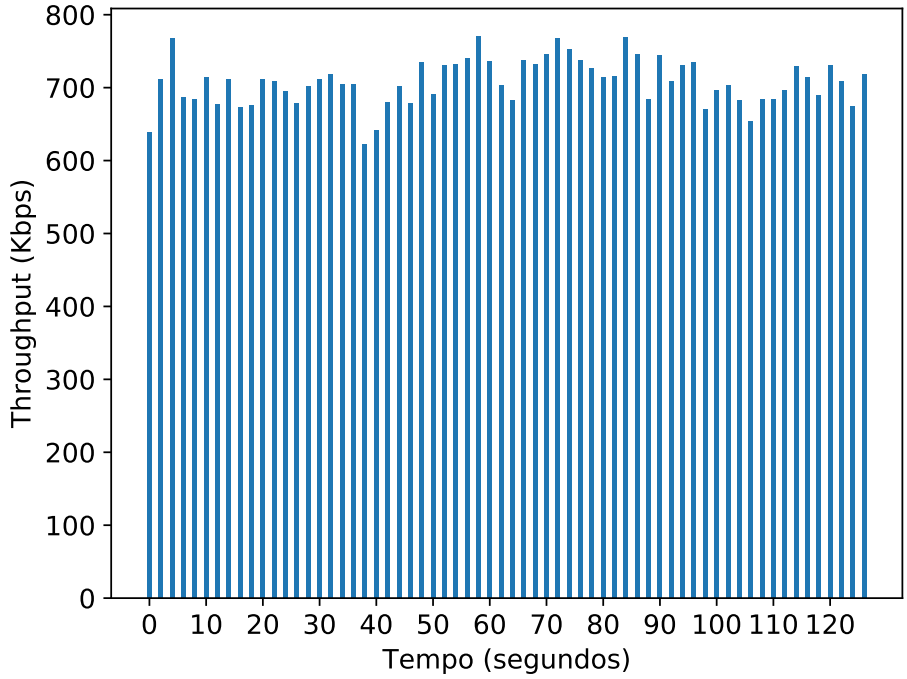
Fonte: do autor.

5.1.2.2 Escala de Microsserviços

A Figura 30 apresenta o consumo de CPU durante o experimento de escala de $br1$ (em verde). Inicialmente, antes de iniciar o consumidor e produtor NDN, $br1$ está com consumo de CPU próximo a 0%. Após serem iniciados, entre 60 e 90 s, o consumo de CPU de $br1$ aumenta para acima do limite estabelecido (70%) em 90 s, sendo iniciado o processo de aumento de escala. Em 120 s é possível verificar o resultado desse processo, onde aparece o consumo da nova instância de $br1$, $br1.1$ (em roxo), e $br1.sr1$ (em laranja), um microsserviço do tipo SF. Nessa configuração, $br1.sr1$ divide o tráfego entre $br1$ e $br1.1$. Mais especificamente, o processo de aumento de escala levou aproximadamente seis segundos para ser executado.

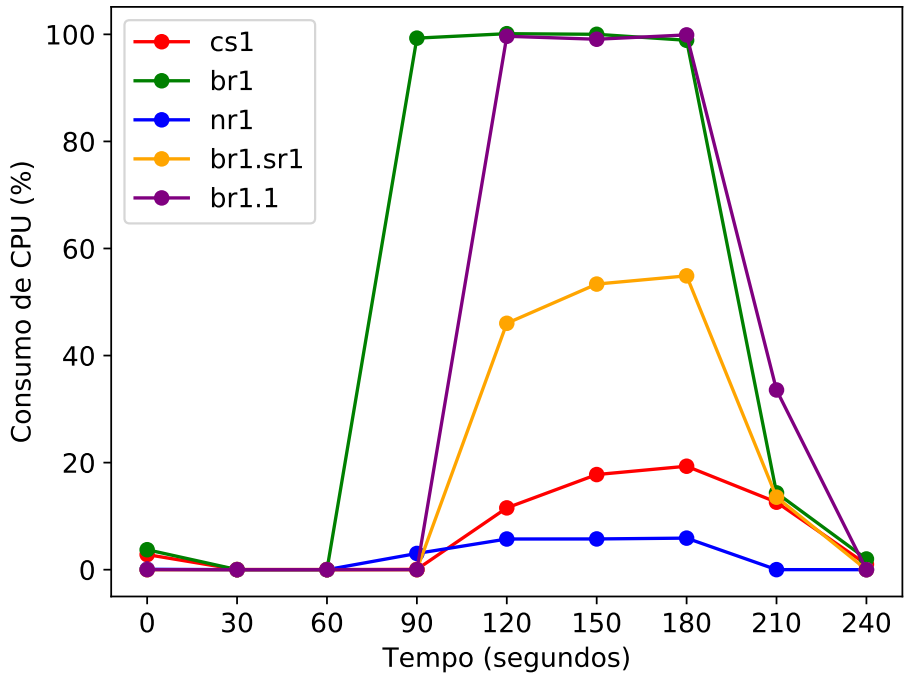
Ainda na Figura 30, entre 150 e 180 s, o consumidor é parado, resultando em uma diminuição no consumo de CPU nos segundos seguintes. Quando o consumo de CPU de

Figura 29 – *Throughput* do consumidor a cada 2 segundos para o caso com dois salto.



Fonte: do autor.

Figura 30 – Consumo de CPU a cada 30 segundos.

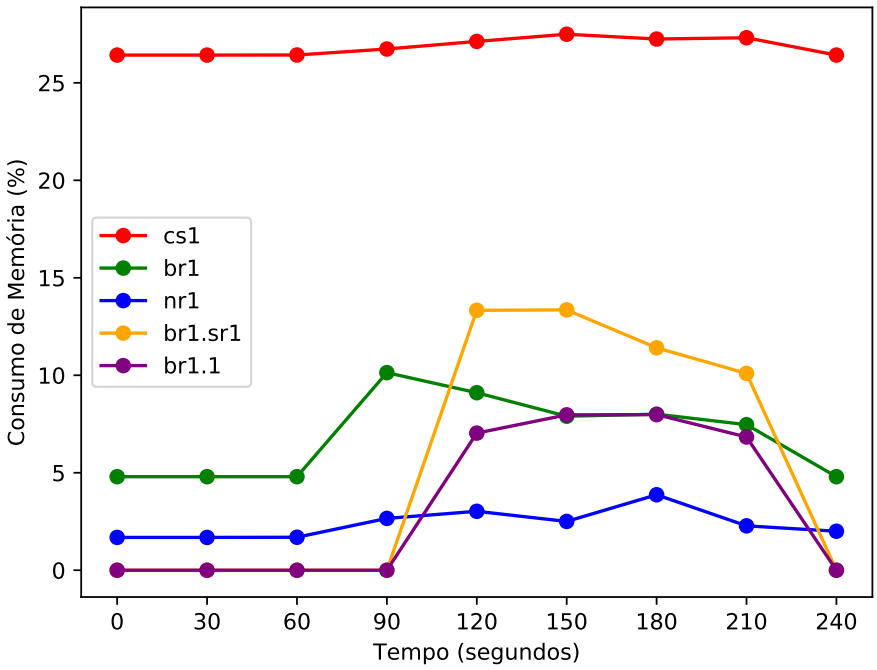


Fonte: do autor.

$br1$ desce para um valor abaixo do limite de 35% o processo de diminuição de escala é executado, resultando na mesma configuração de microsserviços do inicio, em 0 s.

A Figura 31 ilustra o consumo de memória durante o mesmo período do experimento, onde os valores para cada microsserviço permaneceram praticamente inalterados. Devido às baixas flutuações, o sistema não atingiu os critérios de limite necessários para escalar com base no uso de memória. Como o processo de escala ocorreu conforme o esperado, a resposta para P2 é que sim, a solução implementada consegue realizar a escala automática de microsserviços.

Figura 31 – Consumo de memória a cada 30 segundos.

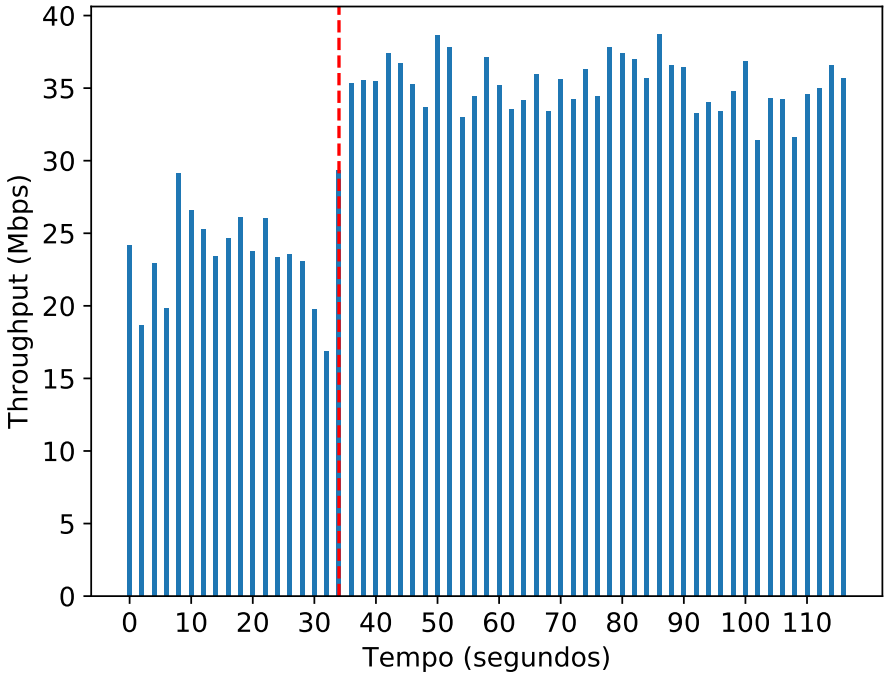


Fonte: do autor.

A Figura 32 mostra o *throughput* verificado pelo consumidor durante o experimento, onde a reta em vermelho indica o momento aproximado do fim do processo de aumento de escala. Vale ressaltar que os dados dessa figura são adquiridos entre o início e o fim do consumidor, logo, a Figura 32 e a Figura 30 não coincidem no tempo. Antes do aumento de escala, Figura 32 entre 20 e 34 s, o *throughput* médio é 23,36 Mbps. Por outro lado, o *throughput* médio é 35,1802 Mbps após o aumento, o que representa um aumento de aproximadamente 50%. Esse resultado demonstra que o aumento de escala está correlacionado com um incremento no *throughput*.

Destaca-se ainda que, durante parte do processo de aumento de escala entre 30 e 34 s, o *throughput* cai devido à momentânea desconstrução da cadeia, ocasionando 101 pacotes de interesse perdidos. Logo, se faz necessária uma abordagem que evite a perda de pacotes durante esse processo. Uma maneira de fazer isso é alterar a ordem de execução das

Figura 32 – *Throughput* a cada 2 segundos.



Fonte: do autor.

operações no Manager, evitando excluir conexões antes que todas as conexões necessárias estejam criadas. Além disso, os microsserviços teriam que ser modificados para alterarem o modo de encaminhamento durante esse período.

Buscando demonstrar sobrecarga decorrente do plano de controle, a Figura 33 mostra a quantidade de comunicação ocorrida entre o Manager e os módulos Monitoring, microsserviços e Orchestrator. A quantidade de comunicação é calculada a partir da quantidade absoluta de mensagens HTTP e UDP que Manager envia e recebe. Na Figura 33 dois picos se destacam. O primeiro pico, próximo a 95 s, se deve ao processo de aumento de escala, onde ocorreram cerca de 44 comunicações, das quais 22 são com o Orchestrator.

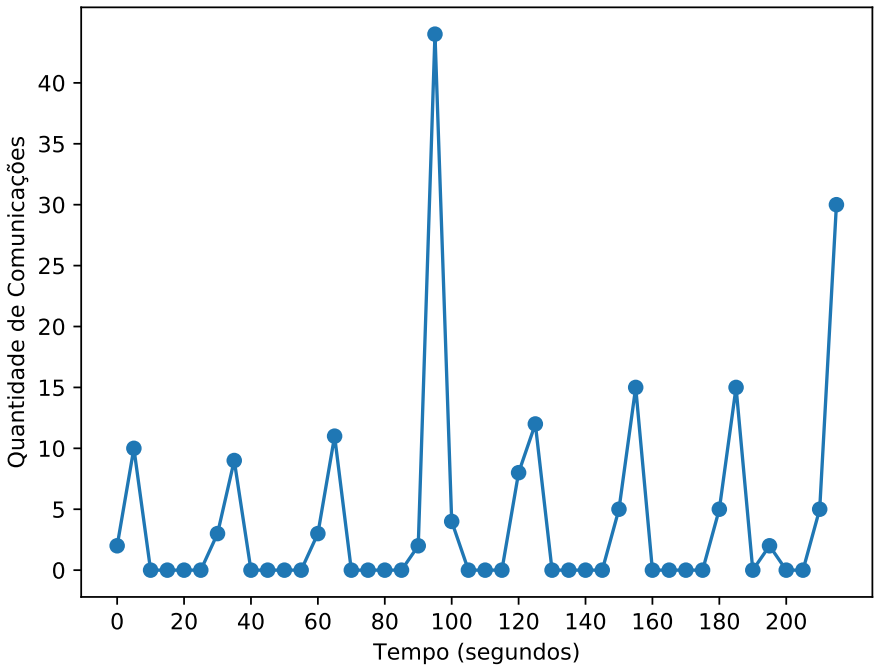
O outro pico da Figura 33, em 215 s, é decorrente do processo de redução de escala. Durante esse período, cerca de 30 comunicações ocorreram. O menor número em relação ao processo de aumento de escala se deve ao menor número de operações executadas.

Procurando responder à pergunta P3: um total aproximado de 29273 pacotes de dados foram recuperados pelo consumidor durante o experimento. Enquanto a quantidade total de comunicações de controle ocorridas é igual a 185, o que representa 0.632% da quantidade de pacotes de dados recuperados.

A Figura 34 apresenta a configuração dos microsserviços utilizada para atingir os resultados demonstrados até aqui. Nessa configuração, foi possível atingir um aumento do *throughput* a partir do aumento de escala. Todavia, esse resultado nem sempre é atingido.

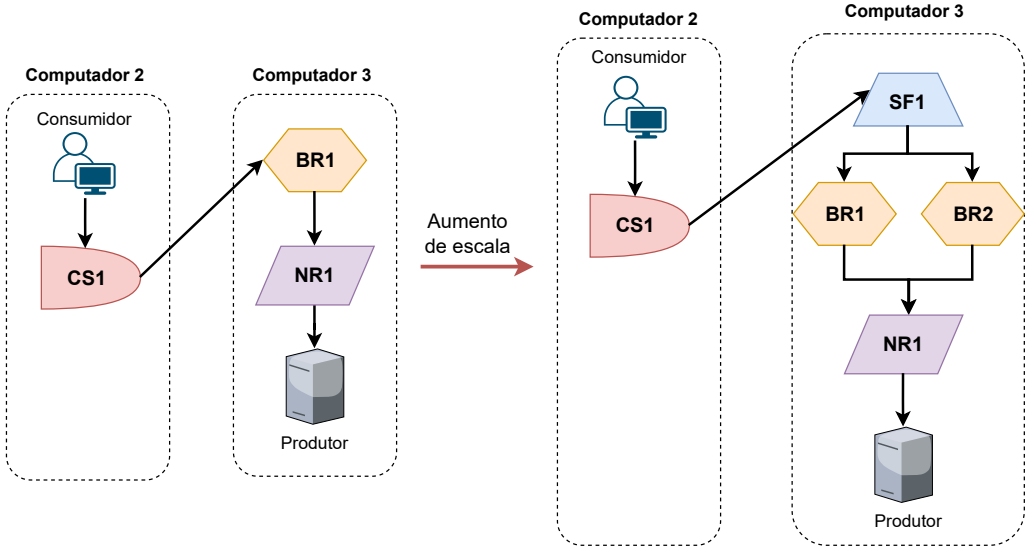
Para demonstrar um caso onde o *throughput* diminui, considere a configuração da

Figura 33 – Quantidade e mensagens entre o Manager e os outros módulos a cada 5 segundos.



Fonte: do autor.

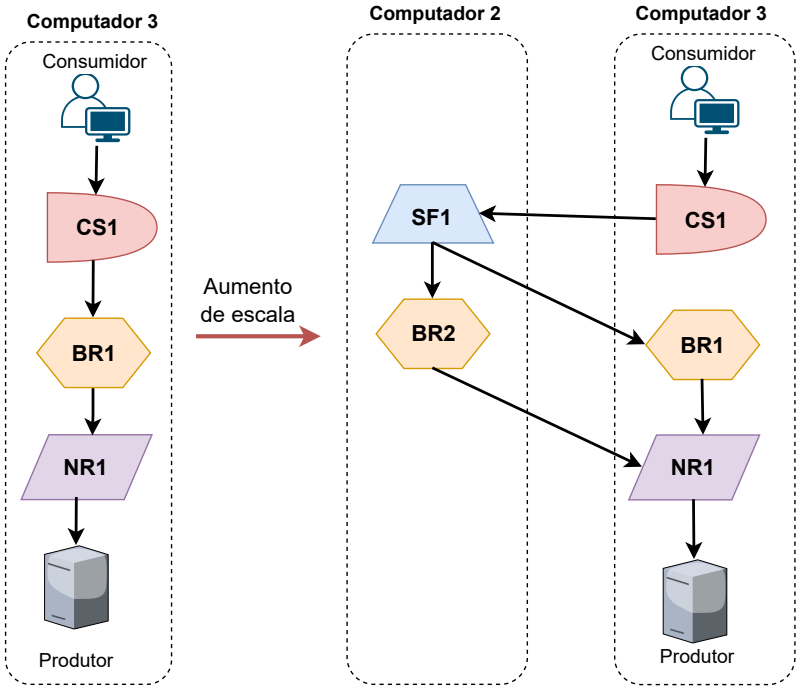
Figura 34 – Posição dos microsserviços para aumento de *throughput*.



Fonte: do autor.

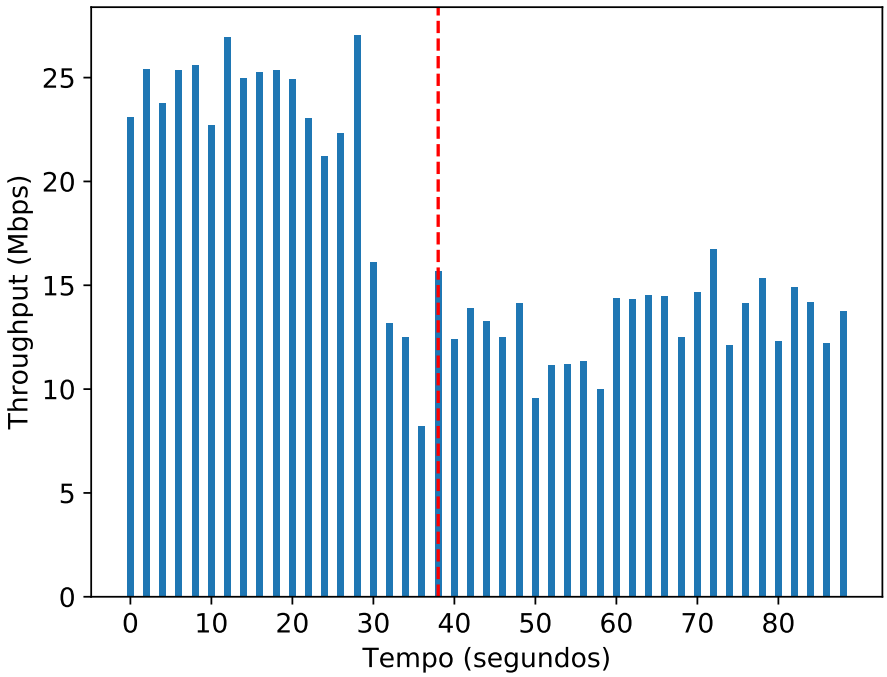
Figura 35. A Figura 36 mostra o *throughput* para essa configuração, onde o *throughput* médio é 21,9436 Mbps antes do aumento de escala e 13,2898 Mbps depois. Isto é, o *throughput* diminui mesmo com o aumento de escala, o que pode ser justificado pelo aumento na quantidade de saltos para 2. Todavia, mais investigações são necessárias.

Figura 35 – Posição dos microsserviços com diminuição de *throughput*.



Fonte: do autor.

Figura 36 – *Throughput* a cada 2 segundos para configuração da Figura 35.

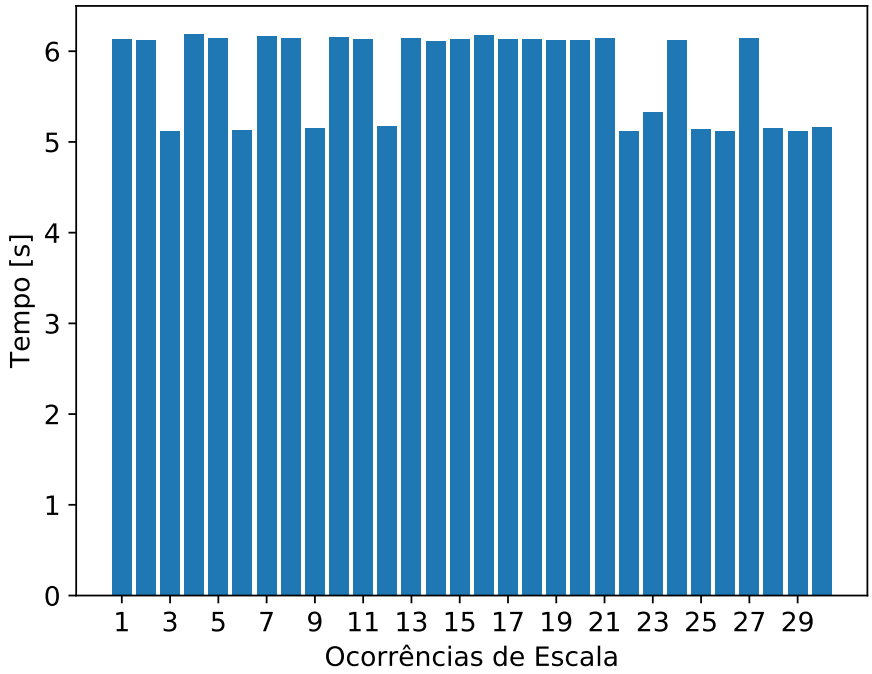


Fonte: do autor.

A Figura 37 e Figura 38 mostram o tempo necessário para realizar 30 processos de aumento e redução de escala. O valores mostrados não consideram o tempo de download

da imagem, uma vez que cada computador possui a imagem dos microsserviços baixada localmente com antecedência. O tempo médio para aumento de escala é de 5,7815 s, enquanto o tempo médio para redução de escala é de 0,0274 s. O maior tempo para o processo de expansão se deve ao maior número de etapas executadas (etapas da Figura 19).

Figura 37 – Tempo necessário para executar o processo de aumento de escala 30 vezes.



Fonte: do autor.

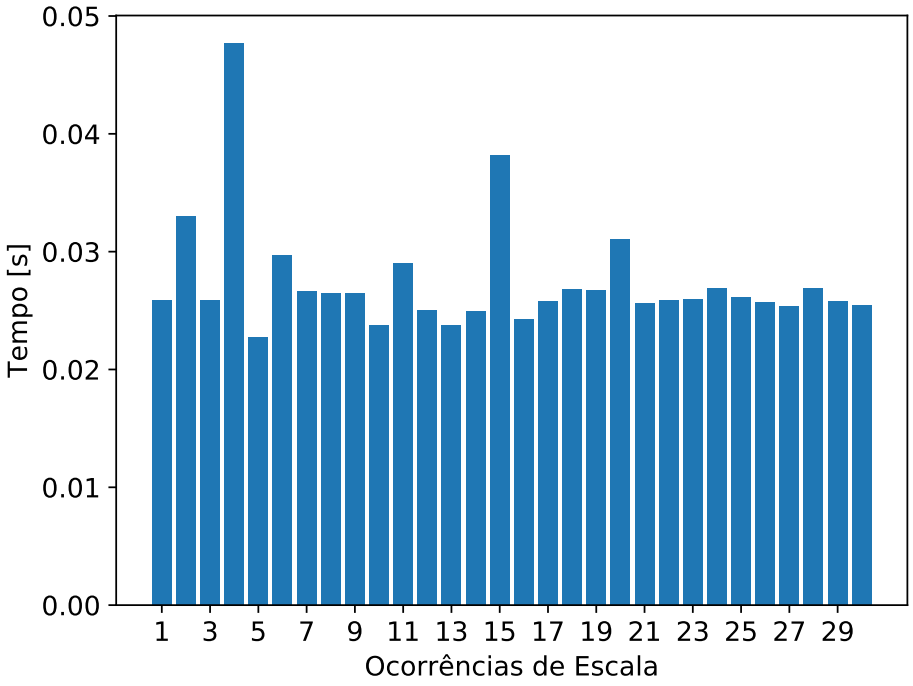
5.2 Aplicação de vigilância em uma missão militar

Semelhante ao conceito de IoT, a Internet das Coisas de Batalha (do inglês *"Internet of Battle Things"* ou IoBT) refere-se à integração de dispositivos físicos, veículos, sensores e qualquer objeto projetado para adquirir, processar e/ou compartilhar dados em um ambiente militar (LEAL *et al.*, 2019).

Em missões militares, especialmente aquelas que envolvem vigilância e monitoramento em tempo real, a obtenção de imagens com baixa latência é importante para o sucesso da missão, o que exige o estabelecimento de uma infraestrutura de rede local para tráfego de dados. Nesse contexto, uma infraestrutura dinâmica e eficiente desempenha um papel crucial no êxito da missão, sendo que essa infraestrutura deve ser robusta, fácil de implantar e reimplantar onde e quando for necessário.

O experimento descrito nessa seção aborda o estabelecimento de uma rede dinâmica para recuperação de vídeo durante vigilância em ambiente IoBT. A Figura 39 apresenta

Figura 38 – Tempo necessário para executar o processo de redução de escala 30 vezes.



Fonte: do autor.

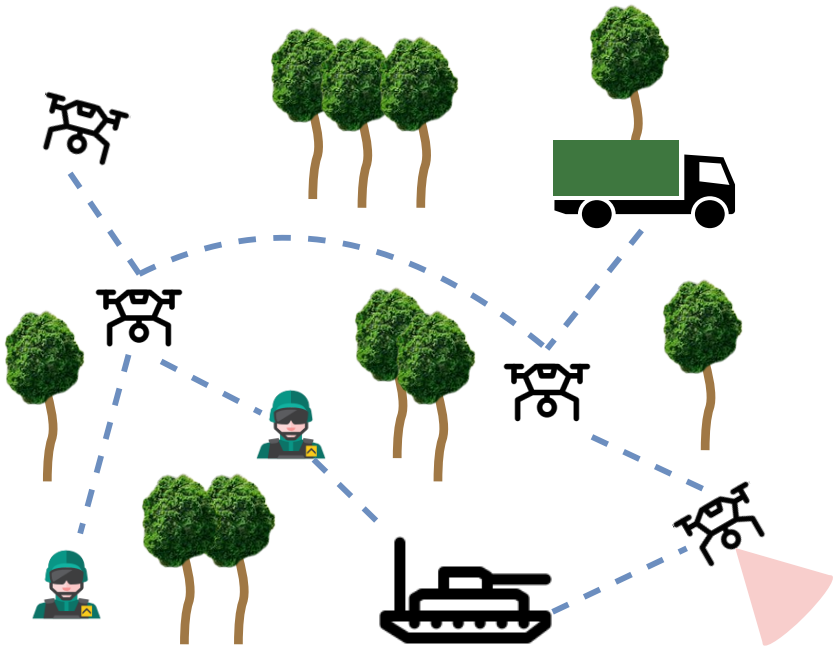
com mais detalhes do contexto abordado, onde o VANT com cone em vermelho é o responsável por adquirir e enviar as imagens de vídeo. Enquanto a rede é composta por VANTs, soldados e tanques conectados (tracejados azuis), os quais operam como roteadores ou consumidores. Apesar do foco nesses dispositivos, esse contexto também pode incluir sensores sem fio em campo, rádios, computadores transportados em solo, veículos militares, entre outros elementos como parte da rede (LEAL *et al.*, 2019; AFANASYEV *et al.*, 2018).

Os nós que constituem a rede IoBT geralmente apresentam capacidades híbridas de processamento, armazenamento e comunicação. Por exemplo, um VANT possui menor capacidade de processamento e armazenamento se comparado a um computador. Logo, as aplicações implementadas, incluindo as de rede, devem considerar essas limitações.

Em suma, o contexto abordado necessita de uma rede resiliente às alterações dinâmicas no ambiente, com baixa latência e com conhecimento do limite de recursos dos dispositivos, sendo que essas características podem ser atingidas a partir de Micro-Chain. Ao utilizar NDN nesse contexto, dados de vídeo podem ser recuperados de nós intermediários mais próximos do consumidor de modo a minimizar a latência.

A flexibilidade de microsserviços NDN permite a implantação dinâmica nos dispositivos de modo que apenas as funcionalidades necessárias sejam implantadas, considerando para isso a quantidade de recursos disponíveis de cada dispositivo. Ainda, os microsserviços podem ser rapidamente excluídos se necessário, o que libera recursos para outras

Figura 39 – Contexto para aplicação de vigilância em IoBT.



Fonte: do autor.

operações. Por exemplo, um dispositivo pode ser habilitado como um dispositivo de rede NDN em uma certa etapa da missão.

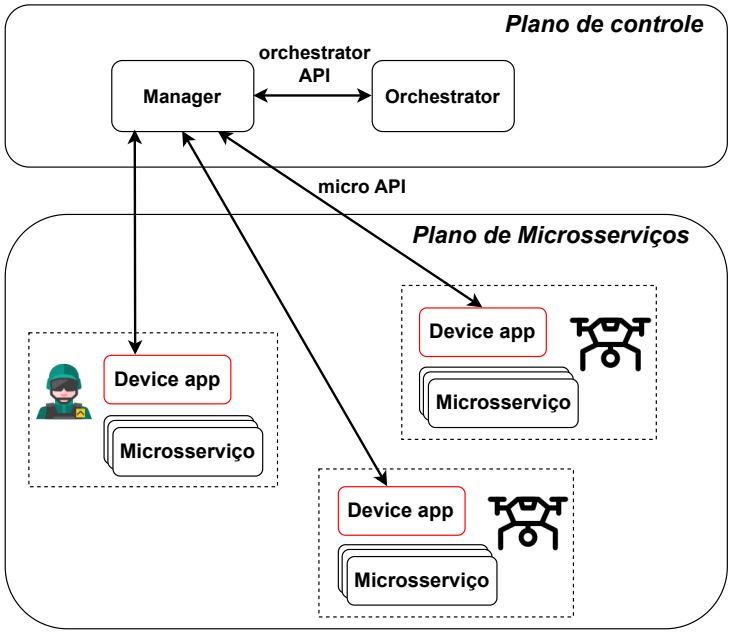
5.2.1 Solução baseada em Micro-Chain

Na presente seção foram apresentadas as alterações realizadas na arquitetura Micro-Chain (Figura 13) para alcançar as características desejadas para a aplicação abordada, conforme delineado na seção anterior. Essas adaptações mostram a flexibilidade da arquitetura para se ajustar a novos contextos.

A Figura 40 ilustra a arquitetura utilizada no experimento realizado, onde o retângulo em vermelho destaca um novo módulo, chamado Device app. Esse módulo é responsável por implementar a lógica do dispositivo para entrada e saída da rede NDN, se relacionando com o Manager a partir de micro API, como estabelecido para os microsserviços anteriormente. Adicionalmente, como esse caso não aborda o monitoramento de métricas, o módulo Monitoring não é utilizado.

De modo geral, a implementação de um plano de controle eficiente demanda uma quantidade considerável de recursos, especialmente em redes de grande escala. Logo, os módulos do plano de controle da Figura 40 devem ser cuidadosamente implantados. No contexto do estudo de caso abordado, esses módulos podem ser instalados em uma estação base localizada em um acampamento próximo à área da missão, equipada com os dispositivos suficientes para processamento e comunicação. Além disso, como a estação está localizada fora do local de combate, há potencialmente uma menor exposição a

Figura 40 – Arquitetura utilizada para uma aplicação de vigilância durante uma operação militar.



Fonte: do autor.

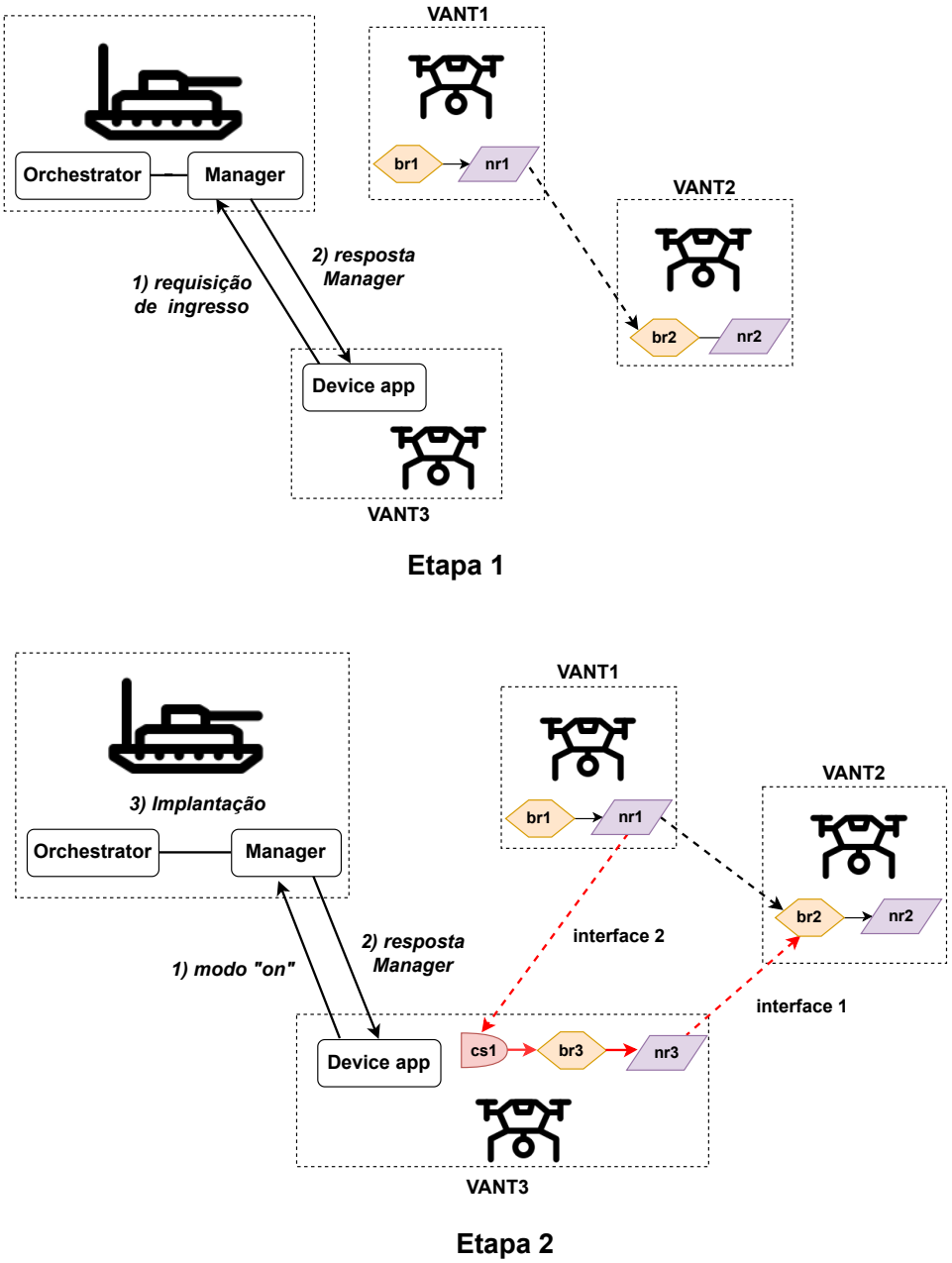
possíveis ataques.

Os dispositivos em campo que compõem a rede NDN, como os VANTs, frequentemente possuem recursos limitados. Por isso, o Manager personaliza a implantação dos microsserviços a partir dos recursos disponíveis. Em particular, é exigido que os dispositivos implantem, no mínimo, os microsserviços BR e NR, visto que desempenham um papel fundamental no encaminhamento de pacotes de interesse e dados dentro da estrutura do NDN. Por outro lado, a implantação do microsserviço CS é condicionada à quantidade de recursos do dispositivo. Dessa forma, dispositivos com recursos acima de um determinado limite realizam a implantação de CS, enquanto os demais se restringem à implantação dos microsserviços BR e NR. Além disso, esse estudo de caso não lida com escala de microsserviços, então, os microsserviços são implantados sem limite de consumo de recursos.

A Figura 41 exemplifica o processo de entrada na rede NDN pelo VANT3 a partir de duas etapas. O sistema é composto VANTs, os quais compõem a rede NDN, e um tanque de guerra que implementa a camada de controle. Na primeira etapa, o Device app de VANT3 envia uma requisição de entrada ao Manager (1). Essa mensagem é enviada via micro API e contém informações sobre o dispositivo. O Manager, por sua vez, verifica as informações recebidas e, então, aceita ou recusa a entrada 2).

Após ter sua entrada aceita, o dispositivo pode mudar seu modo de operação para "on"(ativo), iniciando a implantação dos microsserviços. Essa estratégia permite ao dis-

Figura 41 – Etapas executadas para entrada de um VANT na rede NDN.



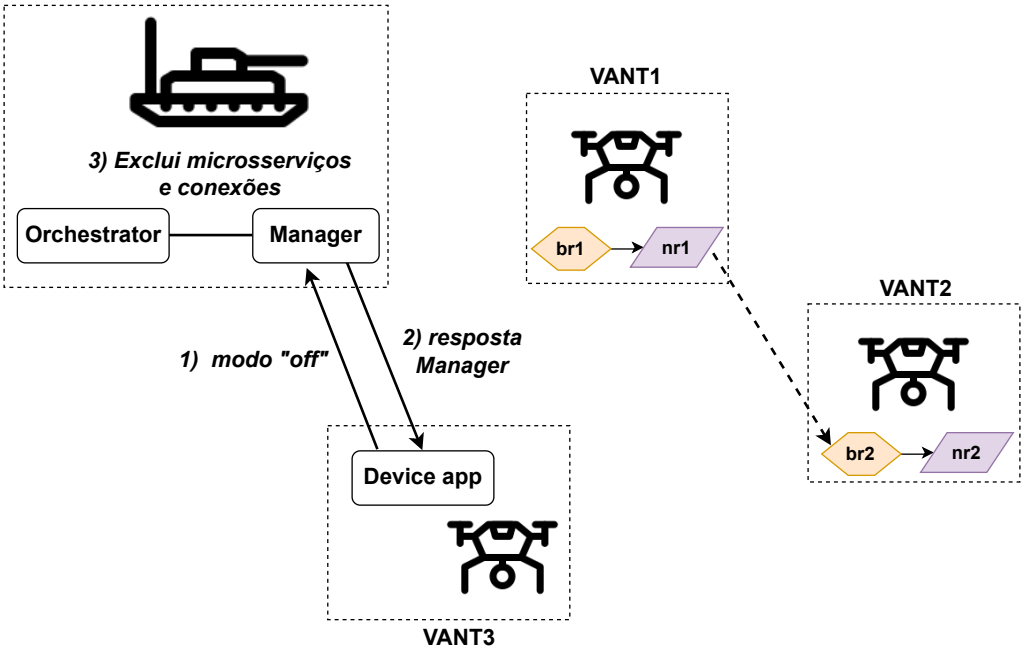
Fonte: do autor.

positivo determinar o momento da implantação. Dessa forma, quando o VANT3 alterar seu modo para "on", conforme mostrado na etapa 2 da Figura 41, ele envia uma nova mensagem ao Manager (1), que confirma o recebimento (2) e inicia a implantação e conexão dos microsserviços (3), indicado pelas setas em vermelho.

Um nó da rede NDN pode, a qualquer momento, efetuar o processo de saída da rede NDN, conforme demonstrado na Figura 42 para o VANT3. O processo começa com o envio de uma mensagem via micro API para o Manager (1), informando o modo de ope-

ração "off"(desligado). Em seguida, o Manager confirma o recebimento (2) e exclui os microsserviços (3), liberando os recursos alocados anteriormente. Para evitar encaminhamento para os microsserviços excluídos, o Manager também executa uma operação para atualizar o encaminhamento dos microsserviços (3), nesse caso excluindo a interface 2 de nr1.

Figura 42 – Processo de saída de um VANT da rede NDN.



Fonte: do autor.

5.2.2 Configuração Experimental

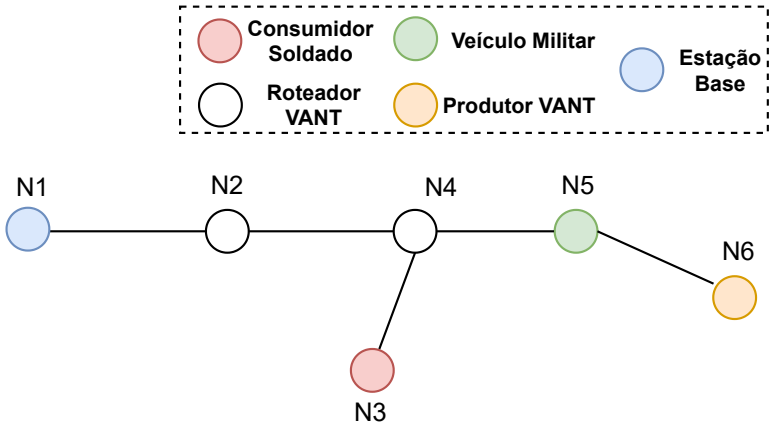
A topologia de rede utilizada no experimento é mostrada na Figura 43, composta por uma estação base, dois VANTs como roteadores, um VANT como produtor, um veículo militar e um soldado. Considera-se que o produtor VANT está equipado com uma câmera que transmite imagens do terreno, sendo o soldado e a estação base consumidores dessas imagens. Além disso, é importante ressaltar que o experimento parte do pressuposto de que o Manager possui conhecimento prévio dessa topologia.

Os consumidores e produtores são executados a partir de ndnping v0.6[1] com NFD v0.6.1[2]. O NFD é um requisito para utilizar o aplicativo ndnping, logo, esses nós utilizam a versão monolítica do NDN e não implantam os microsserviços NDN. Além disso, em resposta a pacote de interesse, o produtor gera pacotes de dados com carga útil contendo 7000 bytes. Como o conteúdo dos dados não é o foco deste trabalho, a carga útil não

[1] https://github.com/named-data/ndn-tools/tree/ndn-tools-0.6/tools/ping

[2] https://github.com/named-data/NFD/tree/NFD-0.6.1

Figura 43 – Topologia dos nós no experimento.



Fonte: do autor.

contém dados reais de vídeo. Nesse sentido, o consumidor que recebe um pacote de dados apenas considera ter recebido corretamente os dados solicitados.

A Tabela 3 apresenta os recursos de cada nó da topologia, sendo os nós implementados a partir de máquinas reais e VMs, isto é, consideramos os nós como computadores. Conforme mencionado, a implantação do microsserviço CS é seletiva. Para os experimentos, considerou-se que apenas nós com no mínimo 3 CPUs e 3000 MB de memória podem implantar esse microsserviço. Logo, espera-se que apenas o veículo militar implemente CS.

Tabela 3 – Recursos dos nós para o experimento.

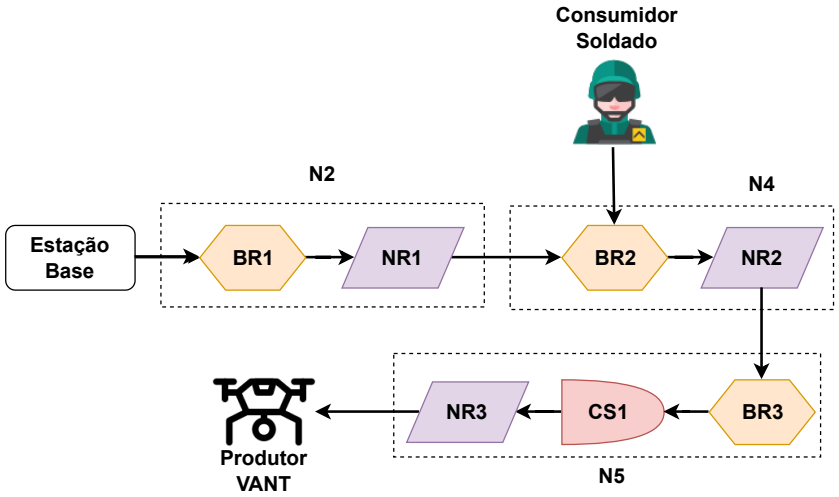
Dispositivos	CPU	Memória (GB)
Estação Base	4	8
Soldado e VANTs	2	2
Veículo Militar	4	4

O experimento é dividido em dois cenários, *(i)* e *(ii)*. O cenário *(i)* verifica o desempenho da comunicação após o estabelecimento da rede NDN. Esse desempenho é avaliado em dois contextos, um para aquisição de dados via produtor, o que simula uma abordagem baseada em IP, e outro para aquisição de dados via microsserviço CS, sendo apresentado o RTT para ambos os casos. Por outro lado, o cenário *(ii)* avalia o processo de entrada e saída de um dispositivo, o qual envolve a configuração dinâmica da rede. Nesse cenário, o VANT representado pelo nó N2 inicia o processo de saída da rede devido ao baixo nível de bateria, seguido por um processo de entrada executado por um VANT substituto. Portanto, considera-se que apenas o nó N2 tem sua conexão alterada durante os experimentos, o restante das conexões da topologia (Figura 43) se mantém inalteradas.

5.2.3 Resultados

Quando o Manager é iniciado, ele utiliza seu conhecimento da topologia para implantar e conectar os microsserviços nos nós. A Figura 44 ilustra a configuração dos microsserviços após a implantação e conexão dos microsserviços. O veículo militar (N5) é o único nó que possui recursos maiores do que os estipulados para implantar CS (3 CPUs e 3000 MB de memória), logo, ele é o único que implementa esse microsserviço.

Figura 44 – Configuração de microsserviços após implantação inicial.



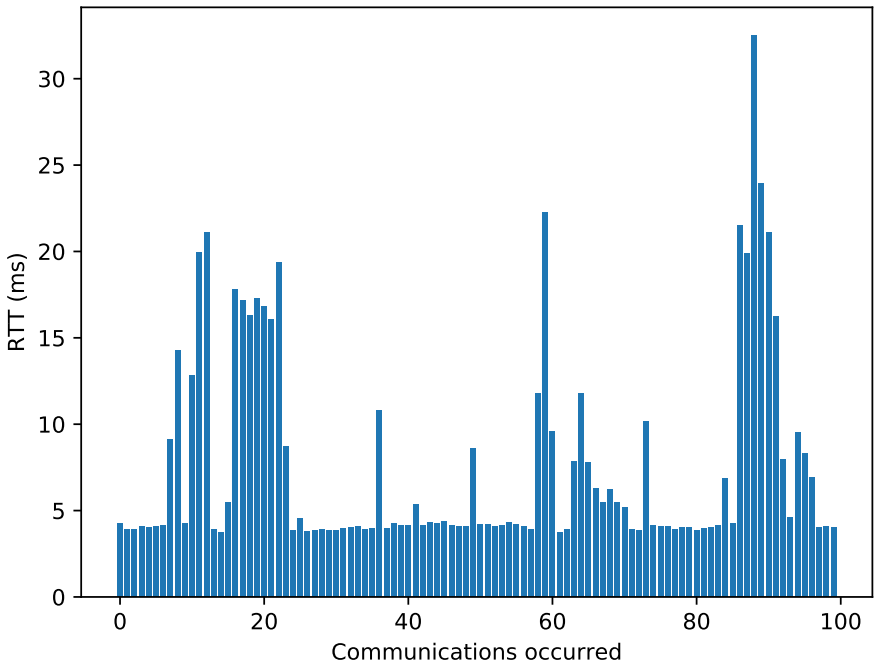
Fonte: do autor.

No cenário *(i)* busca-se avaliar o desempenho da comunicação NDN. A Figura 45 ilustra o RTT do consumidor soldado para quando o conteúdo é recuperado exclusivamente a partir do produtor (N6). Para facilitar a visualização, cada barra corresponde ao RTT médio das últimas 20 recuperações bem-sucedidas. Os resultados para 2000 pacotes transmitidos são: 0% de perda de pacote, RTT médio de 7,6535 ms e RTT máximo de 174,2450 ms.

A Figura 46 ilustra o RTT do consumidor soldado para quando o conteúdo é adquirido a partir do microsserviço CS1 implantado no veículo militar. Para 2000 pacotes transmitidos, os resultados são: 0% de perda de pacote, RTT médio de 2,4104 ms e RTT máximo de 118,4210 ms. Observe que o RTT médio neste cenário é cerca de 31% inferior ao RTT médio do cenário anterior (7,6535 ms), o que indica que o conteúdo é capturado mais rapidamente a partir de CS1. Isso ocorreu porque o soldado adquiriu dados através do veículo militar (N5), o qual está posicionado mais próximo do soldado do que o produtor VANT (N6).

No cenário *(ii)*, são avaliados os processos de saída e entrada na rede NDN. Nesse cenário, considerou-se que o VANT (N2) apresenta um baixo nível de bateria, sendo necessário substituí-lo. Portanto, assim que N2 efetuar o processo de saída da rede (Figura 42), o Manager exclui os microsserviços BR1 e NR1 do nó, resultando na configuração

Figura 45 – RTT do soldado com recuperação de conteúdo via nó N6. Cada barra corresponde na média de 20 amostras.



Fonte: do autor.

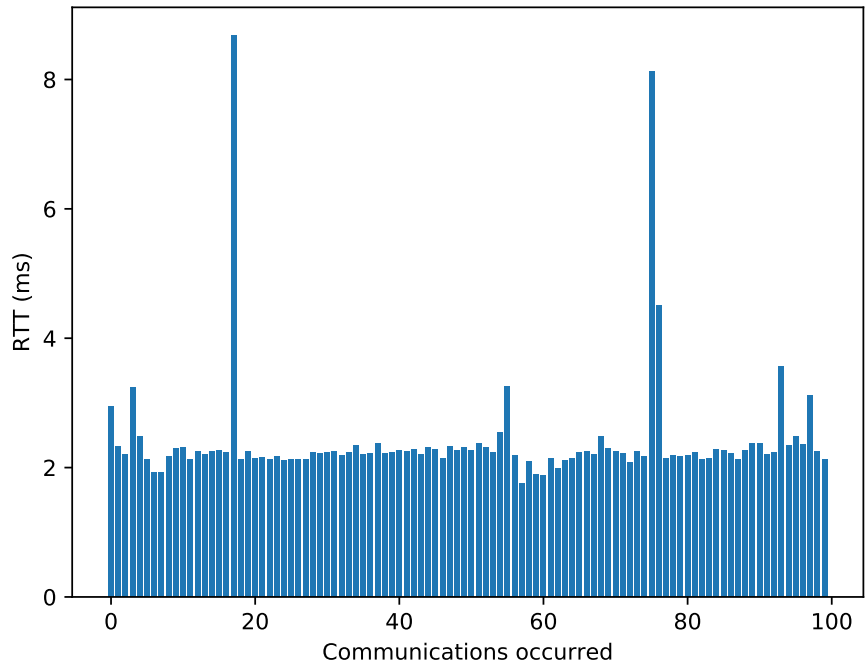
da Figura 47. Na sequência, o VANT substituto realiza o processo de entrada à rede, conforme apresentado na Figura 41. O resultado deste processo é uma nova composição de microsserviços, semelhante à mostrada na Figura 44. Esse cenário mostra que o sistema é capaz de customizar a rede sob demanda, manipulando os microsserviços conforme a necessidade.

5.3 Discussão

Os experimentos demonstraram a maleabilidade da arquitetura de microsserviços, onde partes específicas da aplicação puderam ser dimensionadas separadamente sem a necessidade de dimensionar toda a aplicação como na arquitetura monolítica. Além disso, o sistema pode escolher quais funcionalidades NDN implementar conforme necessário. Em experimentos futuros, é necessário comparar o desempenho de uma implementação monolítica e de microsserviços em um cenário de cluster. Todavia, para garantir uma comparação justa entre essas arquiteturas, é essencial estabelecer um método para escalar NFD.

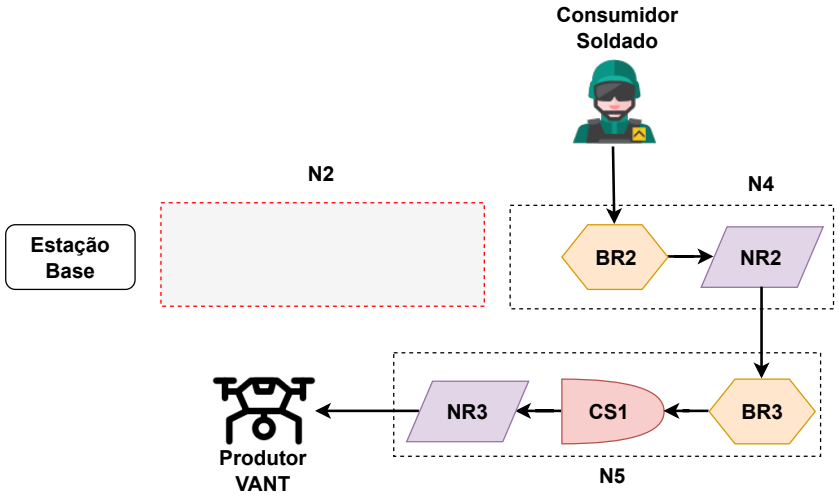
O uso de microsserviços deve facilitar a implementação de melhorias. Por exemplo, novos métodos para decisão de armazenamento na CS podem ser implementados via alteração do microsserviço CS, sem necessitar lidar diretamente com os outros microsser-

Figura 46 – RTT do soldado com recuperação de conteúdo via microsserviço CS1. Cada barra corresponde na média de 20 amostras.



Fonte: do autor.

Figura 47 – Configuração de microsserviços após saída do nó N2.



Fonte: do autor.

viços. Outro exemplo é o desenvolvimento de novas abordagens para pesquisa de nome, a qual pode ser implementada e testada via microsserviço NR.

Vale ressaltar que a arquitetura foi inicialmente projetada para uma abordagem de microsserviços NDN. Ainda assim, a estrutura pode ser estendida para outros cenários

mais genéricos. Além disso, a implementação utiliza Kubernetes e Prometheus como módulos Orchestrator e Monitoring, mas a arquitetura Micro-Chain não se limita a eles. Por exemplo, com ajustes, é possível utilizar outros orquestradores como Docker Swarm, Apache Mesos e Nomad.

Um desafio da implementação de uma arquitetura de microsserviços é determinar o nível apropriado de granularidade, ou seja, o escopo de funcionalidade de cada microsserviço na solução. No contexto dos microsserviços NDN, avaliar e ajustar a granularidade pode reduzir a latência. A princípio, deve-se verificar a combinação das funcionalidades NR e BR em um único microsserviço, conforme também sugerido por MARCHAL; CHOLEZ; FESTOR (2018).

Este trabalho se concentrou em implantar e escalar microsserviços NDN. Todavia, MARCHAL; CHOLEZ; FESTOR (2018) também desenvolveram estratégias para impulsionar a segurança da rede, as quais não foram utilizadas nesse trabalho. Abordagens futuras podem avaliar esse contexto para múltiplos nós.

Comparado à arquitetura ETSI NFV MANO, Micro-Chain possui um Manager que desempenha algumas funções de VNFM e NFVO, enquanto o Orchestrator assume responsabilidades tanto do VIM quanto do VNFM. Logo, não há uma equivalência direta entre os dois sistemas. Além disso, ao contrário do MANO, Micro-Chain define um módulo para monitoramento, Monitoring. O mesmo é valido para a solução de MARCHAL; CHOLEZ; FESTOR (2018), onde o monitoramento é feito unicamente pelo Manager. Essa separação visa oferecer uma maior modularidade e possibilitar o desenvolvimento e manutenção separada desse módulo. Essa abordagem funciona na nuvem, onde *software* de monitoramento vêm sendo desenvolvidos e integrados às soluções.

Os experimentos conduzidos por MARCHAL; CHOLEZ; FESTOR (2018) foram realizados em uma máquina local, o que não permitiu a identificação do problema de posicionamento. Este trabalho aborda esse problema, avaliando a performance para diferentes configurações e discutindo possíveis variáveis que devem ser consideradas para a definição de um método de posicionamento.

Nesse sentido, é fundamental que trabalhos futuros definam um método de posicionamento que considere parâmetros como quantidade de saltos e velocidade de processamento. Além disso, uma solução de Inteligência Artificial poderia ser projetada para prever demandas e preparar a rede de maneira preventiva (MANIAS; SHAMI, 2021).

É importante ressaltar que, durante os experimentos, os dispositivos já tinham as imagens dos microsserviços baixadas. Isso torna o processo de implantação de um microsserviço mais rápido, mas consome memória manter todas as imagens baixadas sem necessariamente utilizá-las. Portanto, se o cenário de aplicação tiver uma limitação de recursos muito restrita, manter localmente as imagens dos microsserviços pode ser inviável. Nesse caso, o método de posicionamento deve priorizar os dispositivos que já possuam as imagens localmente.

A Micro-Chain foi avaliada em um cluster local. Nesse sentido, trabalhos futuros podem este trabalho para suportar operações em um contexto multi-cluster. Uma possível abordagem nesse contexto seria utilizar um Manager para controlar cada cluster e implementar uma forma para que os Managers se comuniquem. Dessa forma, cada Manager precisaria considerar as variáveis locais do seu cluster e as variáveis gerais do conjunto de clusters para a tomada de decisão. Outra alternativa viável consiste em definir um Manager global que controla os Managers locais.

Para cenários geográficos de grande escala com múltiplos dispositivos e cadeias de microsserviços, seria interessante verificar a divisão da rede em regiões com base em parâmetros específicos de interesse. A expectativa é que esta estratégia facilite o posicionamento e controle dos microsserviços.

A arquitetura proposta tem o Manager como responsável por lidar com as políticas de encaminhamento do NDN, a partir do envio de interfaces para os microsserviços NR e criação de conexões entre microsserviços. Em trabalhos futuros, essa função pode ser separada do Manager a partir do desenvolvimento de um módulo controlador SDN para lidar com estratégias de encaminhamento e roteamento dentro da rede NDN. Esse módulo pode estar conectado a Monitoring e Manager para tomada de decisão.

O tempo consumido para efetuar o processo de aumento de escala é em média 5,7818 s, o que é elevado para contextos como o consumo de dados pela Internet, onde a rede precisa se ajustar rapidamente. Ainda mais preocupante, alguns pacotes são perdidos devido à quebra das conexões entre os microsserviços durante o processo de escala. Portanto, antes de utilizar a solução proposta em sistema de comunicação real, o processo de escala deve ser melhorado para eliminar a perda de pacotes durante sua execução. Uma maneira de fazer isso seria configurar os microsserviços para encaminhar os pacotes para um caminho específico durante o processo de escala.

Para o estudo de caso de vigilância, partiu-se do pressuposto de que os links entre os dispositivos são conhecidos pelo Manager para que ele possa criar as conexões corretas entre microsserviços de diferentes nós. Todavia, esse pressuposto é inválido para uso real. Logo, a solução deve ser ajustada com um método para descoberta de conexões em tempo de execução para garantir a manutenção correta das conexões entre microsserviços de diferentes nós.

Ainda considerando uma perspectiva realista para uso da solução proposta no estudo de caso de vigilância, foi assumido que a topologia da rede permanece inalterada durante os experimentos, com exceção do nó N2, que passa por um processo de substituição. Um problema que pode ocorrer nesse sentido é que a mobilidade dos nós pode requerer muitos ajustes nos microsserviços, o que pode não ser atendido pela solução no tempo necessário. Portanto, isso precisa ser verificado em trabalhos futuros.

Durante os experimentos, foram empregados microsserviços encapsulados em contêineres. Embora os contêineres sejam conhecidos por sua leveza, enfrentam desafios de

segurança devido ao compartilhamento do Kernel. Logo, uma abordagem mais atrativa seria estender Micro-Chain e os experimentos para uma abordagem hibrida. Conforme mencionado por KAUR; MANGAT; KUMAR (2022), uma implantação mista de VNFs em contêineres e VMs pode melhorar a escalabilidade e a automação, mas esse ainda é um desafio de pesquisa em aberto.

Semelhante à NFD, os microsserviços utilizados permitem implantar uma rede ICN sob uma rede IP, ou seja, IP e ICN coexistem. Isso é demonstrado pelo uso de monitor API, orchestrator API e micro API, todas desenvolvidas com base em IP. Apenas a recuperação de conteúdo é realizada via ICN. Todavia, essa coexistência não é necessariamente uma limitação, sendo explorada em trabalhos como os de NOUR *et al.* (2019); RAVINDRAN *et al.* (2017); CONTI *et al.* (2020).

A solução desenvolvida possibilita implantar microsserviços unitários e depois conectá-los. Em trabalhos futuros, a arquitetura pode ser entendida para suportar a implantação de serviços de rede a partir de um descritor textual como TOSTA. Cada serviço consistiria em uma cadeia de microsserviço com certas configurações. Isso permitiria, por exemplo, que operadoras de rede implantassem serviços de rede rapidamente, com menos operações manuais.

Com o 5G, fatias NDN podem ser implantadas para suportar certos serviços, como IoT. Nesse sentido, trabalhos futuros devem desenvolver e avaliar uma abordagem de fatiamento baseada em microsserviços NDN no contexto de 5G. Sendo que Micro-Chain pode atuar nesse cenário para gerenciar e orquestrar os microsserviços.

Para impulsionar a generalização da arquitetura, deve ser considerado o desenvolvimento de módulos (ou plugins) que se conectam ao Manager para implementação de lógicas de gerenciamento específicas, similar ao que é estabelecido para VNFM em MANO. Isso permitiria aos desenvolvedores incorporar suas próprias estratégias de gerenciamento, possibilitando, por exemplo, a utilização de diversos métodos de posicionamento e escala para um determinado microsserviço ou cadeia de microsserviço (KOUCHAKSARAEI *et al.*, 2018).

Para impulsionar a modularidade da arquitetura, a aplicação Web pode ser desvinculada do Manager, sendo implementada e implantada separadamente. Essa separação exige o desenvolvimento de uma interface de comunicação entre Manager e aplicação Web. Essa interface pode ser baseada em uma REST API, que encaminha as solicitações da aplicação Web para o Manager, que, por sua vez, responde com o resultado.

6 CONCLUSÃO

Neste trabalho foi proposto Micro-Chain, uma arquitetura para gerenciar e orquestrar microsserviços NDN sob demanda. A arquitetura define quatro módulos, seus interrelacionamentos e operações principais, sendo disponibilizada online para impulsionar extensões e a reprodutibilidade. Um foco proeminente foi dedicado à otimização do consumo de recursos durante a implementação, alcançada por meio do emprego de contêineres e do K3, uma implementação de Kubernetes com baixo consumo de recursos. Essas características impulsionam o uso da solução em cenários com recursos limitados, como industrial IoT e VANETs.

Dois estudos de caso foram definidos e avaliados. O primeiro consiste em um cenário de escala automática em um cluster com três nós. Nesse cenário, a decisão de escalabilidade é executada com base nos limites de uso de CPU e memória. Durante os experimentos foi identificado que o uso de memória permanece praticamente inalterado. Por outro lado, o consumo da CPU apresenta uma grande variação. No geral, os experimentos validam as principais operações propostas pela Micro-Chain em tempo de execução.

Além disso, para o primeiro estudo de caso também abordou o posicionamento de microsserviços. Foi descoberto que o posicionamento inadequado dos microsserviços gera uma queda no *throughput*. Os resultados mostram uma diferença no *throughput* de aproximadamente 76% entre uma configuração e outra. O reforça a necessidade de desenvolver um método de posicionamento, o qual deve considerar variáveis de rede, como saltos, e a velocidade de processamento dos nós.

O segundo estudo de caso consiste no estabelecimento de uma rede NDN sob demanda para distribuição de dados de vídeo durante uma missão militar. Nesse estudo de caso, verificou-se a recuperação de conteúdo a partir do produtor, simulando IP, e a recuperação de conteúdo a partir de um nó intermediário com o conteúdo armazenado. A distribuição do conteúdo nos nós da rede é uma das principais características de NDN. Os resultados mostram que a abordagem com conteúdo armazenado em nós intermediários, mais próximos do consumidor, possui RTT 31% menor.

Ademais, para validar a flexibilidade da solução em gerenciar a rede sob demanda, foi estabelecido um experimento que simula a troca de um VANT com baixo nível de bateria.

Nesse experimento, primeiro o Manager exclui os microsserviços implantados no VANT com baixo nível de bateria e depois reconfigura as rotas dos microsserviços remanescentes. Na sequência, o novo VANT entra na rede e tem microsserviços implantados nele, enquanto os microsserviços dos outros nós são novamente reconfigurados considerando os novos microsserviços.

Como uma solução modular, Micro-Chain delineia uma arquitetura que define um conjunto de módulos com responsabilidades isoladas, o que potencialmente simplifica a compreensão do sistema e a resolução de problemas. Além disso, essa abordagem também impulsiona a atualizações e correções específicas em um módulo sem afetar os demais, promovendo a reutilização de código e facilitando a escalabilidade do sistema para se adaptar a novos requisitos, conforme demonstrado no estudo de caso militar.

Trabalhos futuros devem focam em um contexto com múltiplos clusters já que a solução está limitada a um cluster. Nesse contexto, deve ser desenvolvida uma maneira para comunicação entre os clusters e uma política para tomada de decisão em conjunto. Além disso, inspirado em 5G, pode ser desenvolvida uma lógica para implantar e controlar fatias de rede NDN. Nesse caso, as fatias podem implantar certas funções do NDN conforme as características desejadas. Por exemplo, uma fatia de vídeo necessita de alta velocidade para uma boa qualidade de experiência, logo, pode implantar uma CS robusta. Enquanto uma fatia de IoT pode não implementar uma CS, já que não há uma restrição justa de velocidade.

REFERÊNCIAS

AFANASYEV, A. *et al.* A brief introduction to named data networking. *In*: MILCOM 2018-2018 IEEE MILITARY COMMUNICATIONS CONFERENCE (MILCOM), 2018. **Proceedings [...]** [S.l.: s.n.], 2018. p. 1–6.

ALDAOUD, M. *et al.* Leveraging ICN and SDN for Future Internet Architecture: a survey. **Electronics**, [S.l.], v. 12, n. 7, p. 1723, 2023.

ALOURANI, A. *et al.* Dynamic and Energy Efficient Cache Scheduling Framework for IoMT over ICN. **Applied Sciences**, [S.l.], v. 13, n. 21, p. 11840, 2023.

BARKA, E. *et al.* A trusted lightweight communication strategy for flying named data networking. **Sensors**, [S.l.], v. 18, n. 8, p. 2683, 2018.

CASTILLO LEMA, J. **A generic network function virtualization manager and orchestrator for content-centric networks**. 2019. Tese (Doutorado em Engenharia Elétrica) — Universidade de São Paulo, 2019.

CERNY, T. *et al.* Microservice Architecture Reconstruction and Visualization Techniques: a review. *In*: IEEE INTERNATIONAL CONFERENCE ON SERVICE-ORIENTED SYSTEM ENGINEERING (SOSE 22), 2022. **Proceedings [...]** [S.l.: s.n.], 2022. p. 39–48.

CHOWDHURY, S. R. *et al.* Re-Architecting NFV Ecosystem with Microservices: state of the art and research challenges. **IEEE Network**, [S.l.], v. 33, n. 3, p. 168–176, 2019.

CONTI, M. *et al.* The Road Ahead for Networking: a survey on icn-ip coexistence solutions. **IEEE Communications Surveys & Tutorials**, [S.l.], v. 22, n. 3, p. 2104–2129, 2020.

CZIVA, R.; PEZAROS, D. P. Container Network Functions: bringing nfv to the network edge. **IEEE Communications Magazine**, [S.l.], v. 55, n. 6, p. 24–31, 2017.

DE CARVALHO, L. R.; PATRICIA FAVACHO DE ARAUJO, A. Performance Comparison of Terraform and Cloudify as Multicloud Orchestrators. *In*: IEEE/ACM

INTERNATIONAL SYMPOSIUM ON CLUSTER, CLOUD AND INTERNET COMPUTING (CCGRID), 2020., 2020. **Proceedings [...]** [S.l.: s.n.], 2020. p. 380–389.

ETSI. **Network Functions Virtualization (NFV); Architectural Framework,**. [S.l.: s.n.], 2014.

FOWLER, S. J. **Production-ready microservices**: building standardized systems across an engineering organization. [S.l.]: "O'Reilly Media, Inc.", 2016.

FREY, M. *et al.* Security for the Industrial IoT: the case for information-centric networking. *In*: IEEE 5TH WORLD FORUM ON INTERNET OF THINGS (WF-IOT), 2019., 2019. **Proceedings [...]** [S.l.: s.n.], 2019. p. 424–429.

G.A., C. **Cloudify orchestration project portal**. [S.l.: s.n.], 2024. Disponível em: `https://cloudify.co/` Acesso em: 22 jan. 2024.

GüR, G. *et al.* Integration of ICN and MEC in 5G and Beyond Networks: mutual benefits, use cases, challenges, standardization, and future research. **IEEE Open Journal of the Communications Society**, [S.l.], v. 3, p. 1382–1412, 2022.

HAWILO, H.; JAMMAL, M.; SHAMI, A. Exploring Microservices as the Architecture of Choice for Network Function Virtualization Platforms. **IEEE Network**, [S.l.], v. 33, n. 2, p. 202–210, 2019.

HAWILO, H.; JAMMAL, M.; SHAMI, A. Network function virtualization-aware orchestrator for service function chaining placement in the cloud. **IEEE Journal on Selected Areas in Communications**, [S.l.], v. 37, n. 3, p. 643–655, 2019.

KAPETANIDOU, I. A.; MENDES, P.; TSAOUSSIDIS, V. Enhancing Security in Information-Centric Ad Hoc Networks. *In*: NOMS 2023-2023 IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, 2023. **Proceedings [...]** [S.l.: s.n.], 2023. p. 1–9.

KAUR, K.; MANGAT, V.; KUMAR, K. A review on Virtualized Infrastructure Managers with management and orchestration features in NFV architecture. **Computer Networks**, [S.l.], v. 217, p. 109281, 2022.

KOUCHAKSARAEI, H. R. *et al.* Programmable and Flexible Management and Orchestration of Virtualized Network Functions. *In*: EUROPEAN CONFERENCE ON NETWORKS AND COMMUNICATIONS (EUCNC), 2018., 2018. **Proceedings [...]** [S.l.: s.n.], 2018. p. 1–9.

KUBERNETES. **Kubernetes**: production-grade container orchestration. [S.l.: s.n.], 2023. Disponível em: `https://kubernetes.io/` Acesso em: 28 dez. 2023.

LEAL, G. M. *et al.* Empowering Command and Control through a Combination of Information-Centric Networking and Software Defined Networking. **IEEE Communications Magazine**, [S.l.], v. 57, n. 8, p. 48–55, 2019.

LEI, K. *et al.* Securing ICN-Based UAV Ad Hoc Networks with Blockchain. **IEEE Communications Magazine**, [S.l.], v. 57, n. 6, p. 26–32, 2019.

MAI, H. L. *et al.* Toward Content-Oriented Orchestration: sdn and nfv as enabling technologies for ndn. *In*: IFIP/IEEE SYMPOSIUM ON INTEGRATED NETWORK AND SERVICE MANAGEMENT (IM), 2019., 2019. **Proceedings [...]** [S.l.: s.n.], 2019. p. 594–598.

MANIAS, D. M.; SHAMI, A. The Need for Advanced Intelligence in NFV Management and Orchestration. **IEEE Network**, [S.l.], v. 35, n. 1, p. 365–371, 2021.

MARCHAL, X.; CHOLEZ, T.; FESTOR, O. μ NDN: an orchestrated microservice architecture for named data networking. *In*: ACM CONFERENCE ON INFORMATION-CENTRIC NETWORKING, 5., 2018. **Proceedings [...]** [S.l.: s.n.], 2018. p. 12–23.

MARS, D. *et al.* Using information centric networking in internet of things: a survey. **Wireless Personal Communications**, [S.l.], v. 105, p. 87–103, 2019.

NOUR, B. *et al.* A distributed ICN-based IoT network architecture: an ambient assisted living application case study. *In*: GLOBECOM 2017-2017 IEEE GLOBAL COMMUNICATIONS CONFERENCE, 2017. **Proceedings [...]** [S.l.: s.n.], 2017. p. 1–6.

NOUR, B. *et al.* A survey of Internet of Things communication using ICN: a use case perspective. **Computer Communications**, [S.l.], v. 142, p. 95–123, 2019.

NOUR, B. *et al.* Coexistence of ICN and IP Networks: an nfv as a service approach. *In*: IEEE GLOBAL COMMUNICATIONS CONFERENCE (GLOBECOM), 2019., 2019. **Proceedings [...]** [S.l.: s.n.], 2019. p. 1–6.

PAHL, C. *et al.* Cloud Container Technologies: a state-of-the-art review. **IEEE Transactions on Cloud Computing**, [S.l.], v. 7, n. 3, p. 677–692, 2019.

PROMETHEUS. **Prometheus**: power your metrics and alerting with the leading open-source monitoring solution. [S.l.: s.n.], 2023. Disponível em: `https://prometheus.io/` Acesso em: 28 dez. 2023.

RAVINDRAN, R. *et al.* 5G-ICN: delivering icn services over 5g using network slicing. **IEEE Communications Magazine**, [S.l.], v. 55, n. 5, p. 101–107, 2017.

SANTOS, J. *et al.* Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing Applications. *In*: IEEE CONFERENCE ON NETWORK SOFTWARIZATION (NETSOFT), 2019., 2019. **Proceedings [...]** [S.l.: s.n.], 2019. p. 351–359.

SANTOS, J. *et al.* Towards delay-aware container-based service function chaining in fog computing. *In*: NOMS 2020-2020 IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, 2020. **Proceedings [...]** [S.l.: s.n.], 2020. p. 1–9.

SHARIAT, A.; TIZGHADAM, A.; LEON-GARCIA, A. An ICN-based publish-subscribe platform to deliver UAV service in smart cities. *In*: IEEE CONFERENCE ON COMPUTER COMMUNICATIONS WORKSHOPS (INFOCOM WKSHPS), 2016., 2016. **Proceedings [...]** [S.l.: s.n.], 2016. p. 698–703.

SHARMA, P. *et al.* Containers and virtual machines at scale: a comparative study. *In*: OF THE 17TH INTERNATIONAL MIDDLEWARE CONFERENCE, 2016. **Proceedings [...]** [S.l.: s.n.], 2016. p. 1–13.

SINGH, V. P.; UJJWAL, R. A walkthrough of name data networking: architecture, functionalities, operations and open issues. **Sustainable Computing: Informatics and Systems**, [S.l.], v. 28, p. 100419, 2020.

WANG, Z. *et al.* A Web-based Orchestrator for Dynamic Service Function Chaining Development with Kubernetes. *In*: IEEE 8TH INTERNATIONAL CONFERENCE ON NETWORK SOFTWARIZATION (NETSOFT), 2022., 2022. **Proceedings [...]** [S.l.: s.n.], 2022. p. 234–236.

YILMA, G. M. *et al.* Benchmarking open source NFV MANO systems: osm and onap. **Computer communications**, [S.l.], v. 161, p. 86–98, 2020.

ZHANG, L. *et al.* Named data networking. **SIGCOMM Comput. Commun. Rev.**, [S.l.], v. 44, n. 3, 2014.

ZHANG, Z. *et al.* In-network caching for icn-based iot (icn-iot): a comprehensive survey. **IEEE Internet of Things Journal**, [S.l.], 2023.