Rodrigo Ferraz Azevedo

Slice como Serviço em Edge Computing e Dispositivos IoT com Restrição de Recursos

Sorocaba, SP

01 de Julho de 2021

Rodrigo Ferraz Azevedo

Slice como Serviço em Edge Computing e Dispositivos IoT com Restrição de Recursos

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC-So) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Linha de pesquisa: Engenharia de Software e Sistemas de Computação.

Universidade Federal de São Carlos – UFSCar

Centro de Ciências em Gestão e Tecnologia – CCGT

Programa de Pós-Graduação em Ciência da Computação – PPGCC-So

Orientador: Prof. Dr. Fábio Luciano Verdi

Coorientador: Prof. Dr. Luciano Bernardes de Paula

Sorocaba, SP

01 de Julho de 2021

Azevedo, Rodrigo Ferraz

Slice como serviço em edge computing e dispositivos IoT com restrição de recursos / Rodrigo Ferraz Azevedo -- 2021.
106f.

Dissertação (Mestrado) - Universidade Federal de São Carlos, campus Sorocaba, Sorocaba
Orientador (a): Fábio Luciano Verdi
Banca Examinadora: Edmundo Roberto Mauro Madeira, Yeda Regina Venturini
Bibliografia

1. Slice. 2. IoT. 3. Edge computing. I. Azevedo, Rodrigo Ferraz. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Maria Aparecida de Lourdes Mariano - CRB/8 6979

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências em Gestão e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Rodrigo Ferraz Azevedo, realizada em 30/07/2021.

Comissão Julgadora:

Prof. Dr. Fabio Luciano Verdi (UFSCar)

Prof. Dr. Edmundo Roberto Mauro Madeira (UNICAMP)

Profa. Dra. Yeda Regina Venturini (UFSCar)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

À minha esposa Graziela e filha Mariana;
e aos meus pais Mateus e Izabel.

Agradecimentos

Agradeço,

a minha família, principalmente esposa Graziela e filha Mariana pela compreensão pelos momentos em que estive ausente para a conclusão deste mestrado.

ao meu orientador, Professor Doutor Fábio, pelos ensinamentos e apoio durante todo o período em que passei na UFSCar, principalmente durante o trabalho na minha pesquisa.

ao meu co-orientador, Professor Doutor Luciano, pelo apoio durante a pesquisa.

a todos os meus colegas de universidade, em especial André, Paulo, Leandro, Guilherme e Gustavo.

a todos os meus professores da UFSCar, em especial Professora Doutora Luciana e Professor Doutor Fábio por ensinamentos que levarei para a vida profissional.

além de um agradecimento especial a meu colega André por me ajudar com o entendimento inicial do projeto NECOS.

"Jamais considere seus estudos como uma obrigação, mas como uma oportunidade invejável para aprender a conhecer a influência libertadora da beleza do reino do espírito, para seu próprio prazer pessoal e para proveito da comunidade à qual seu futuro trabalho pertencer."

(Albert Einstein)

Resumo

Um novo conceito que descreve uma estrutura federada que fornece computação (processamento), rede e armazenamento como uma estrutura de fatias virtuais, a *Cloud Network Slicing* (CNS) possibilita novas abordagens para aplicações IoT e estruturação dos recursos na borda da rede. Algumas características dos dispositivos IoT, como conexão intermitente, alimentação por bateria, baixo poder de processamento e outras restrições de recursos requerem novas estratégias para atender este novo perfil de aplicações. Esta pesquisa adota a arquitetura definida no Projeto NECOS e implementa as funções para criação de CNS em dispositivos de borda com restrição de recursos. A implementação foi avaliada em *hardware Single Board Computer* (SBC) usando soluções de virtualização leve (microsserviços) e os resultados atingidos confirmaram a hipótese de suporte à *slice* nestes dispositivos com recursos restritos, mas com limitações de suporte à múltiplas *slices*. Portanto, este trabalho apresenta um breve estudo sobre as limitações do projeto NECOS em ambientes com restrição de recursos, principalmente no contexto IoT, propõe e implementa a extensão de alguns componentes NECOS para suporte à equipamentos com restrição de recursos, e então analisa o desempenho da proposta em *hardwares* reais IoT utilizando SBC.

Palavras-chaves: Cloud Computing, Edge Computing, IoT, Slice, NECOS, Microsserviços, Cloud Network Slice.

Abstract

A new concept that describes a federated framework that provides computing (processing), networking, and storage as a virtual slice framework, Cloud Network Slicing (CNS) enables new approaches to IoT applications and structuring resources at the edge of the network. Some characteristics of IoT devices, such as intermittent connection, battery power, low processing power, and other resource constraints, require new strategies to meet this new application profile. This research adopts the architecture defined in the NECOS Project and implements the functions for creating CNS in resource-constrained edge devices. The implementation was evaluated in Single Board Computer (SBC) hardware using lightweight virtualization solutions (microservices) and the results achieved confirmed the hypothesis of slice support in these devices with restricted resources, but with limitations to support multiple slices. Therefore, this work presents a brief study on the limitations of the NECOS project in resource-constrained environments, mainly in the IoT context, proposes and implements the extension of some NECOS components to support resource-constrained equipment, then analyzes the performance of the proposed on real IoT hardware using SBC.

Key-words: Cloud Computing, Edge Computing, IoT, Slice, NECOS, Microservices, Cloud Network Slice.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CNS	Cloud Network Slice
DC	Data Center
DCSC	DC Slice Controller
eMMC	Embedded Multimedia Card
FaaS	Function as a Service
GRE	Generic Routing Encapsulation
IoT	Internet of Things
IMA	Infrastructure and Monitoring Abstraction
LSDC	Lightweight Slice Defined Cloud
NFV	Network Function Virtualization
NECOS	Novel Enablers in Cloud Slicing
OVS	Open vSwitch
QoS	Quality of Service
SD	Secure Digital Card
SLA	Service Level Agreement
SA	Slice Agent
SDN	Software Defined Networking
VDU	Virtual Deployment Unit
VIM	Virtual Infrastructure Manager
WSC	WAN Slice Controller

Sumário

Introdução

Sistemas IoT são predominantemente compostos por dispositivos baseados em sensores e atuadores distribuídos e possuem conexão de dados intermitente. Estas características fazem com que, em certos cenários, os dispositivos IoT gerem um grande volume de dados que necessitam de processamento e análise na *cloud* para então atuar sobre o mesmo *hardware*, o que aumenta o tráfego de dados entre o dispositivo, o provedor de telecomunicações e o *data center*. Esse fato, aliado a outras características como conexão intermitente, alimentação por bateria, baixo poder de processamento, mobilidade e outras restrições de recursos, impactam diretamente na qualidade da aplicação, exigindo novas estratégias para atender a esse novo perfil de demanda (DONNO; TANGE; DRAGONI, 2019).

Com o objetivo de atender à demanda de redução de tráfego de dados, respostas mais rápidas entre o dispositivo IoT e a *cloud*, além do aumento da segurança, foram propostas novas abordagens como a *edge computing*. Essa abordagem propõe a alocação de serviços que anteriormente estavam localizados na *cloud* para locais mais próximos aos dispositivos que geram dados e consomem serviços durante a operação (DONNO; TANGE; DRAGONI, 2019).

Nesse sentido, combinado com a maior disponibilidade de recursos computacionais distribuídos entre diferentes provedores de infraestrutura, houve também a necessidade de suportar múltiplas tecnologias, de modo que o conceito de fatiamento de recursos ganhou força como solução para uma melhor prestação de serviços de IoT baseados em *slicing* (PASSI; BATRA, 2017).

Slicing não é um conceito novo e já foi mencionado em várias iniciativas, tais como (Y, 2011), (ETSI, 2014), (ALLIANCE, 2015), (TR, 2016), (FLINCK et al., 2017) e (SPRECHER, 2017). No entanto, o surgimento de um novo conceito, chamado *Cloud Network Slicing* (CNS), como em (CLAYMAN et al., 2021), envolve não apenas o conceito de fatiamento de recursos de rede, mas também recursos de computação e armazenamento. Esse conceito permitiu uma nova proposta também para a arquitetura IoT, baseada na abstração de rede de ponta a ponta e isolada (E2E, *end-to-end*), na qual o dispositivo IoT se torna um elemento da rede distribuída que hospeda a CNS.

Entretanto, é conhecido que as soluções baseadas em *edge computing* tipicamente utilizam equipamentos com baixos recursos computacionais, como citado em (XHAFA; KILIC; KRAUSE, 2020). Portanto, demandas como tempo de resposta da aplicação, duração da bateria, largura de banda, além de segurança e privacidade dos dados exigem maior atenção dos responsáveis pelo sistema distribuído. Não obstante, diversos outros

fatores como limitação de banda, *hardware* de alto custo e limitação de acesso ao espaço físico onde se encontram os dispositivos, faz com que *hardwares* precisem ser compartilhados com garantias de isolamento entre aplicações, demonstrando a possibilidade da aplicação do conceito de CNS em IoT.

Adicionalmente, no contexto de CNS, não é recomendável que os dispositivos de borda sejam dedicados à uma única *slice*, principalmente em casos que contemplem *hardwares* escassos em dispositivos com baixo poder computacional, como por exemplo, na presença de recursos como GPUs, que são utilizadas principalmente em aplicações de processamento de imagens. Dessa forma, é interessante que um único dispositivo comporte mais de uma *slice* em casos específicos (XHAFA; KILIC; KRAUSE, 2020).

Uma das recentes arquiteturas que atende a todos esses requisitos é a que foi desenhada no contexto do projeto NECOS (*Novel Enablers for Cloud Slicing*) (CLAYMAN et al., 2021). Assim, a possibilidade de instanciação de CNS em dispositivos de borda com restrição de recursos, através de componentes NECOS, é uma das abordagens que podem tornar os serviços de IoT mais seguros, escaláveis e confiáveis com o uso eficiente de recursos escassos.

No NECOS foi projetada uma arquitetura para CNS e uma implementação minimalista foi desenvolvida no contexto do projeto. Entretanto, a implementação não contemplou cenários com equipamentos de baixo recurso na borda. As *slices* criadas contemplam essencialmente o uso de dispositivos computacionais com alto poder de processamento, tipicamente servidores físicos localizados em *data centers*. Uma vez que a arquitetura não foi instanciada neste tipo de cenário, a sua aplicação neste projeto exigiu a adaptação de algumas funcionalidades ao novo cenário focado em IoT.

Como os trabalhos relacionados presentes na literatura não preveem múltiplas *slices* diretamente no dispositivo restritivo e é acreditado que tal funcionalidade possa reduzir custos de projetos e aumentar qualidade de serviços IoT, espera-se que a aplicação e provável extensão dos componentes do projeto NECOS para suporte a estes dispositivos, eleve o estado da arte em projetos IoT.

Neste documento é descrito o projeto e implementação de um sistema minimalista para a criação de CNS em ambientes de borda de rede com restrição de recursos através de componentes NECOS. É citado como os componentes da arquitetura NECOS foram estendidos através de microsserviços baseados em *containers* para suportar o *hardware* validado através de uma prova de conceito usando como cenário um serviço de coleta de dados, que pode ser usado como exemplo de um sistema de monitoração ambiental.

Foram utilizados *Single Board Computers* (SBC), que se referem a computadores que agregam todos os componentes eletrônicos em uma única placa de circuito impresso, a saber: Dragonboard 410c (processador ARM com 1,2 GHz, memória 1 GB e GPU),

Raspberry Pi 3 (processador ARM com 1,2 GHz, memória 1 GB e GPU), Raspberry Pi 4 (processador ARM com 1,5 GHz, memória 4 GB e GPU) e NVIDIA Jetson Nano (processador ARM com 1,43 GHz, memória 4 GB e GPU).

A avaliação de desempenho dos SBC durante a criação de uma ou mais *slices* foi realizada através de um *software* de monitoramento para a captura do consumo de CPU, consumo de memória e temperatura atingida durante este processo. Os resultados atingidos confirmaram a possibilidade de suporte à múltiplas *slices* nestes dispositivos com recursos restritos. No entanto, verificou-se que instanciar *slices* diretamente nestes dispositivos consome recursos superiores ao ofertado por SBC. Como a *slice* depende de virtualização, além de mecanismos que garantam isolamento entre elas, o suporte a múltiplas *slices* se torna limitado pela capacidade de execução de ambientes de virtualização.

Objetivos e Contribuições

O principal objetivo desta dissertação é apresentar uma solução baseada na arquitetura NECOS para suporte à CNS em dispositivos com restrição de recursos, como dispositivos IoT e outros equipamentos presentes na borda da rede e que seja eficiente e de orquestração automatizada em dispositivos localizados em locais de difícil acesso. Objetiva-se também verificar o suporte da arquitetura NECOS à múltiplas *slices* em dispositivos restritivos através da experimentação por uma prova de conceito.

Em resumo, as principais contribuições desse trabalho são:

- Apresentação de um estudo sobre as limitações do projeto NECOS em ambientes com restrição de recursos, principalmente no contexto IoT;
- Proposta de extensão dos componentes NECOS para suporte à equipamentos com restrição de recursos, principalmente no contexto IoT;
- Construção de uma prova de conceito baseada na proposta;
- Validação através da prova de conceito do suporte à múltiplas *slices*, além da análise do desempenho do proposto em *hardwares* reais IoT utilizando SBC (*Single Board Computer*).

Organização

Essa dissertação está estruturada conforme a seguir.

- No capítulo 1 são descritos os conceitos fundamentais no contexto IoT, incluindo o projeto NECOS, e detalhados seus problemas e principais estudos presentes na literatura para a compreensão do trabalho.;
- No capítulo 2 é descrita a proposta de implementação da arquitetura NECOS e dos componentes estendidos;
- No capítulo 3 são descritos os experimentos realizados e resultados obtidos;
- Finalmente, são apresentadas as principais conclusões e oferecidas sugestões para trabalhos futuros no capítulo final.

1 Fundamentação Teórica

O objetivo deste capítulo é apresentar a fundamentação teórica necessária para compreensão do trabalho. Na seção 1.1 abordaremos resumidamente o conceito de *Edge Computing*. O conceito de Internet das Coisas (IoT) será abordado na Seção 1.2. Já os conceitos de *Network Slicing* e *Cloud Network Slicing* e sua relação serão abordados nas Seções 1.3 e 1.4, respectivamente. Finalmente, na Seção 1.6, apresentamos os principais trabalhos relacionados e suas principais diferenças com relação a nossa proposta.

1.1 Edge Computing

Edge Computing é um paradigma moderno de computação relativo à disponibilização na borda da rede de serviços, os quais anteriormente estavam localizados somente na *cloud*, principalmente serviços sensíveis à latência e qualidade de conexão. Esta camada de rede engloba os dispositivos finais e seus usuários e sua arquitetura estende os serviços da *cloud* para a borda da rede, elevando a qualidade de serviço da aplicação (XHAFA; KILIC; KRAUSE, 2020).

A Figura 1 mostra que parte do processamento de dados coletados dos sensores, e que tradicionalmente são processados na *cloud*, migram para a borda da rede e passam a operar em dispositivos físicos ou virtuais (*edge nodes*) estrategicamente posicionados em locais mais próximos dos sensores e atuadores. Há também casos em que dispositivos agregam sensores fisicamente, sendo possível afirmar que desempenham papéis conjuntos de sensoriamento e processamento ao passo que os dados passam a ser processados diretamente nos dispositivos onde são capturados, em situações específicas.

Como exemplo, é possível citar uma aplicação que captura imagens e as envia para serviços na *cloud* para detecção e reconhecimento facial. Após o processamento dos dados, um dispositivo é acionado no local onde o dado foi originado. Seguindo os princípios de *edge computing*, o serviço de detecção e reconhecimento que, geralmente, gera grande tráfego e consumo de dados, seria alocado na borda da rede ao invés da *cloud*, reduzindo o trânsito de dados e aumentando o tempo de resposta e confiabilidade da aplicação (XHAFA; KILIC; KRAUSE, 2020).

Desta forma, a *edge computing* privilegia aplicações que necessitam de alta garantia de serviço ao passo que entrega processamento mais rápido que aplicações que dependem de processamento na *cloud*, mas o uso de aplicações na borda da rede traz grandes desafios, principalmente no suporte à carga de trabalho e gestão de segurança de equipamentos geralmente localizados em locais remotos e com controle de acesso físico reduzido ou

inexistente.

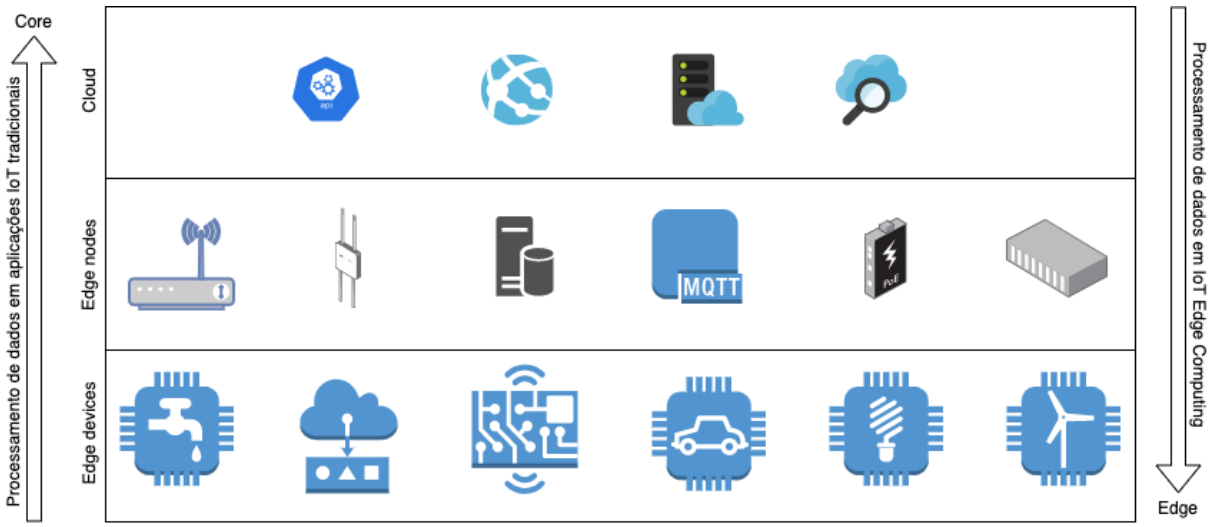

Figura 1 – *Edge Computing*.

1.2 Internet das Coisas

Segundo (DONNO; TANGE; DRAGONI, 2019), o termo Internet das Coisas (*Internet of Things* - IoT) se refere à conexão de objetos do mundo real (coisas) através de sensores e atuadores conectados à Internet. O conceito de IoT tem relação direta com *cloud computing*, já que depende desta como camada de aplicação.

A arquitetura IoT, geralmente, é constituída por 3 camadas:

- Camada de percepção: responsável por coletar dados do meio ambiente através de sensores, transmiti-los para a camada de rede, então realizar interações através de atuadores. Esta camada geralmente é representada pelo dispositivo IoT físico ou virtual.
- Camada de rede: responsável por enviar os dados recebidos dos sensores para a aplicação consumidora que se encontra na camada de aplicação, assim como transmitir a informação no caminho inverso até os atuadores dos dispositivos conectados.
- Camada de aplicação: responsável pelos serviços e aplicações que lidam com os dados advindos dos sensores e retornam ações baseadas nas regras de negócios estabelecidas. Esta camada geralmente é representada pela *cloud*.

Os dispositivos IoT geralmente comportam aplicações sensíveis à latência, com alto consumo de banda de rede, com recursos computacionais limitados, componentes distribuídos geograficamente, dependentes de dados de contexto locais (localização do

usuário, qualidade de conexão etc) e com grandes desafios no gerenciamento de segurança e privacidade.

(STEEN; TANENBAUM, 2007) descreve a operação padrão deste tipo de sistema distribuído composto por dispositivos baseados em sensores e com conexão de dados intermitente como equipamentos que coletam dados e os transferem para um serviço na *cloud*, onde são aplicadas regras de negócios, então instruções são devolvidas para o dispositivo de origem, onde ocorre a execução de ações através de seus atuadores. Estas características fazem com que ocorra um grande tráfego de dados nos dois sentidos entre os dispositivos IoT e a *cloud*, impactando negativamente na performance das aplicações e aumentando os custos envolvidos com o grande volume de dados transitando nos dois sentidos, dos dispositivos para a *cloud* e vice-versa.

1.3 Network Slicing

Slicing é um conceito que se refere à segmentação de recursos físicos em elementos virtuais voltados à eficácia de aplicações e serviços em infraestrutura compartilhada (CLAYMAN, 2018), mudando o paradigma relacionado às redes de computadores com base em recursos físicos.

Entretanto, o conceito é comumente associado à virtualização de redes através do conceito de NFV (*Network Function Virtualization*), provendo múltiplas redes lógicas sobre uma infraestrutura de rede compartilhada (FLINCK et al., 2017), representadas cada uma delas por uma *slice* isolada e com serviços independentes sobre um canal de comunicação único.

Do ponto de vista de um operador de rede, *Network Slicing* implementa rede como serviço, provendo *slices* que oferecem serviços de redes independentes com capacidade de suportar diferentes serviços com garantias de qualidade de serviço (QoS) (ETSI, 2014). Desta forma, este conceito tem sido comumente associado a redes 5G onde um canal de comunicação representado por uma frequência de transmissão é dividido em vários subcanais isolados e independentes (ORDONEZ-LUCENA et al., 2017).

Já do ponto de vista de padronização da definição do termo, este tem sido utilizado contextualmente de diferentes maneiras, embora todos tenham algo em comum, que o conceito engloba virtualização de elementos de rede e a criação de múltiplas instâncias de redes lógicas, provendo rede como serviço, conforme citado em (Y, 2011), (ETSI, 2014), (ALLIANCE, 2015), (TR, 2016), (FLINCK et al., 2017), (SPRECHER, 2017) e (KUKLIŃSKI et al., 2018).

O conceito de *Network Slicing* é comumente associado na literatura à entrega de serviços de rede e a Figura 2 ilustra o fatiamento de rede (*network slicing*) que fornece

serviços comumente associados aos dispositivos físicos de rede.

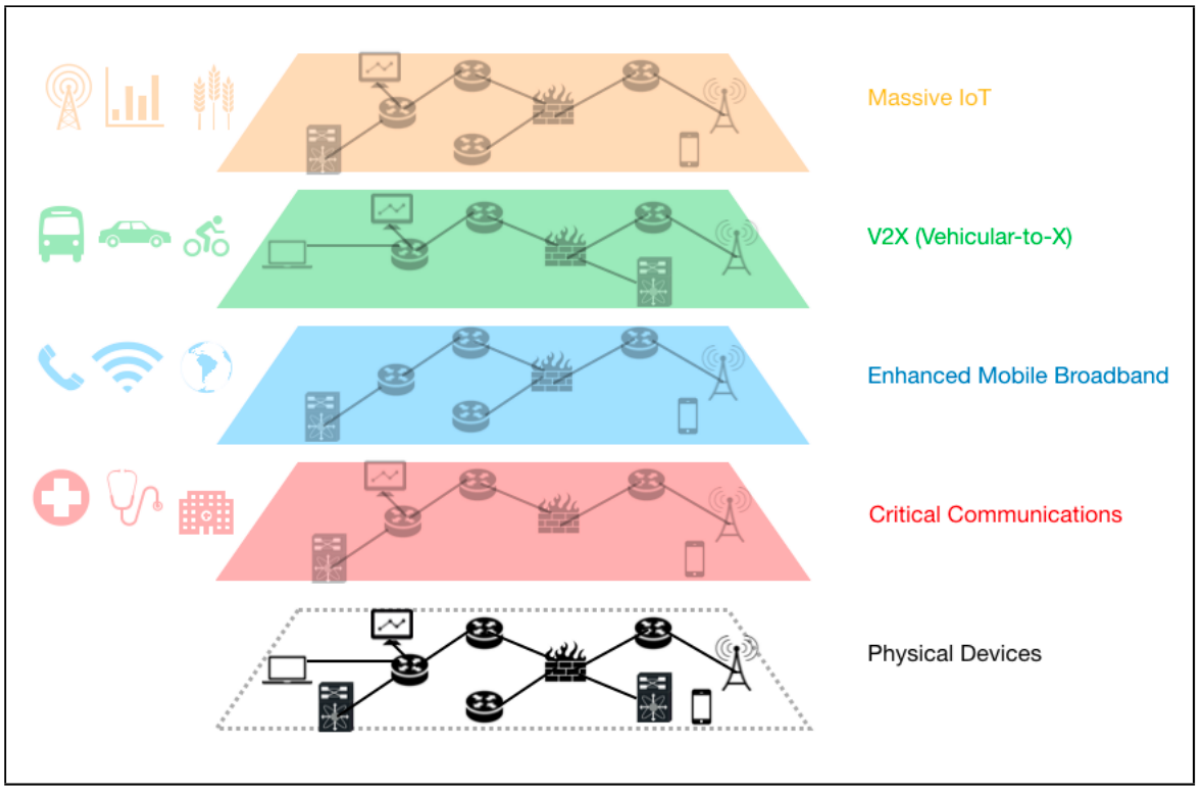

Figura 2 – *Network Slicing* (ALMEIDA; JR; VERDI, 2020).

Neste contexto é importante observar o conceito de *Virtual Deployment Unit* (VDU), que se refere à unidade virtual que comporta todo o *hardware* virtual que suporta as funções virtualizadas do *hardware* de rede (MARINONI et al., 2018).

1.4 Cloud Network Slicing

A *Cloud Network Slicing* (CNS), como citado em (CLAYMAN et al., 2021), envolve não apenas o conceito de fatiamento de recursos de rede, mas também recursos de computação e armazenamento em uma *slice* fim-a-fim. Desta forma é possível afirmar que a CNS entrega serviços integrados de *cloud computing* e *network slicing* em uma unidade lógica fim-a-fim, a *slice*.

Então, a CNS permite a programação de estruturas de rede sob demanda baseadas em NFV e SDN, aliadas ao poder do *cloud computing* para entrega de uma estrutura virtual fim-a-fim entre dispositivos de computação e rede multidomínios, independentemente controlados, gerenciados e orquestrados (ALMEIDA; JR; VERDI, 2020; ROCHA et al., 2019).

Em um contexto de aplicação IoT, esse conceito permite uma nova proposta de arquitetura, baseada na abstração de rede de ponta a ponta e isolada (E2E, *end-to-end*),

na qual o dispositivo IoT se torna um elemento da rede distribuída que hospeda a *slice*.

Conforme (CLAYMAN et al., 2021), a CNS provê recursos controlados, orquestrados e gerenciados independentemente e cada *slice* pode acomodar serviços isoladamente e entre domínios administrativos (*Cross Domain Slice Federation*), o que teoricamente permite o provisionamento de novos serviços e funcionalidades em dispositivos IoT sob demanda e de forma autônoma. A Figura 3 ilustra a CNS provendo a combinação de todas as redes e recursos computacionais, funções e ativos para a entrega de um serviço.

Estas características da CNS permitem novas abordagens computacionais em um cenário onde há crescimento exponencial da quantidade de dispositivos conectados processando informações e impactando toda a infraestrutura, permitindo, por exemplo, a migração sob demanda de processamento de dados da *cloud* para a *edge*, logo transferindo o impacto dessa responsabilidade da *cloud* para os dispositivos conectados conforme citado em (XHAFA; KILIC; KRAUSE, 2020).

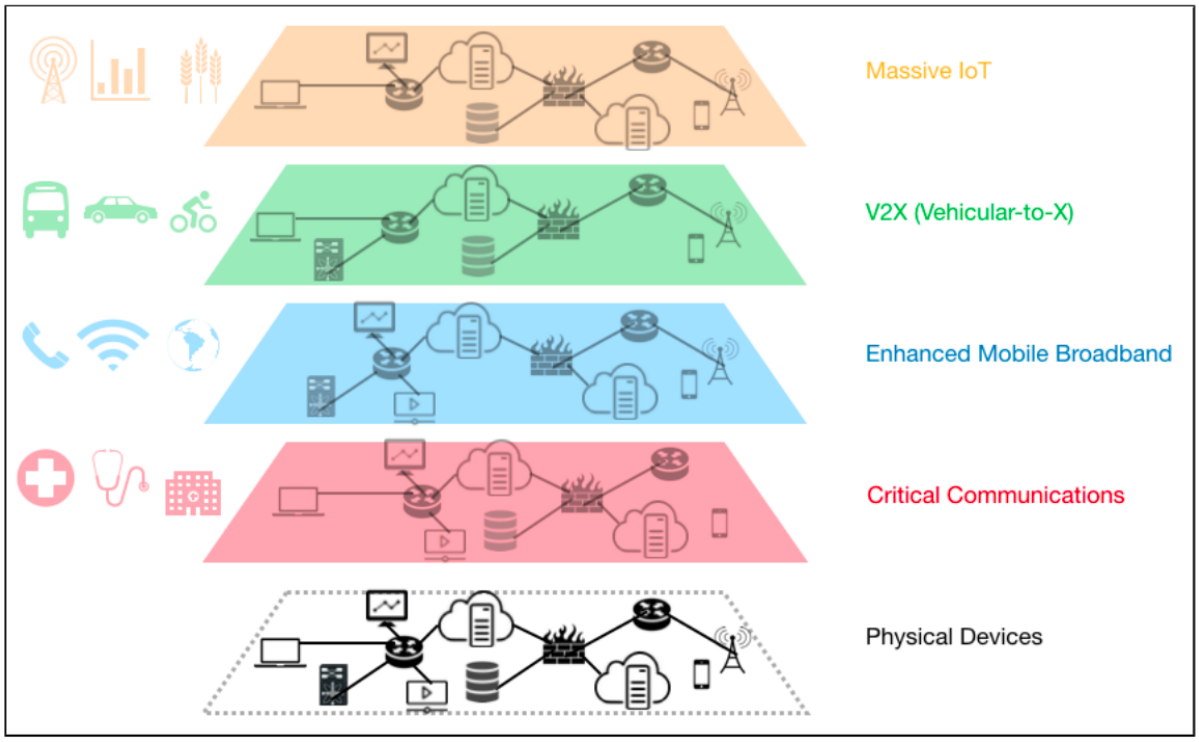

Figura 3 – *Cloud Network Slicing* (ALMEIDA; JR; VERDI, 2020).

No contexto de CNS, VDU se refere à unidade virtual que comporta além das funções virtualizadas de rede, também as funções de armazenamento e processamento (computação). É geralmente representada por máquinas virtuais e *containers* (CLAYMAN et al., 2021).

1.5 NECOS

Frente à demanda crescente de novas abordagens para solução das limitações dos provedores de computação em *cloud*, entre elas a centralização de infraestrutura e a incompatibilidade com serviços heterogêneos, o projeto NECOS, *Novel Enablers for Cloud slicing*, concebeu e implementou uma arquitetura para atender aos novos requisitos de *data centers* de forma inovadora e integrada (CLAYMAN et al., 2021).

O NECOS se baseia no conceito de *Lightweight Software-defined Cloud* (LSDC), uma proposta que estende a virtualização de recursos de *data centers* provendo gerenciamento uniforme através de um alto nível de orquestração, combinando *slice parts* em uma *slice* fim-a-fim para a entrega de uma estrutura virtual composta por rede, computação e armazenamento.

O projeto introduz um novo conceito de *slice-as-a-service*, a *Cloud Network Slicing* (CNS), que engloba rede, armazenamento e computação integrados em uma *slice*, com alocação de recursos otimizada através de orquestração de alto nível baseada em algoritmos de inteligência artificial (CLAYMAN et al., 2021). Através do conceito de CNS, o projeto visa alterar a maneira como os serviços de computação, rede e armazenamento são oferecidos e entregues aos contratantes de serviços de computação em nuvem, permitindo a entrega desses recursos de forma integrada, em uma única unidade, a CNS. Esta unidade agrega recursos de rede, computação e armazenamento, gerando maior automação e maior agilidade na prestação de serviços de comunicação, economia e escalabilidade.

No NECOS a CNS é um conjunto gerenciado de recursos de infraestrutura física e virtual para a entrega através de *slice* como serviço de recursos virtuais de rede, *cloud*, conectividade, computação (processamento), armazenamento e demais serviços que, geralmente, são entregues por *data centers* e provedores de telecomunicações. Esta capacidade faz com que infraestrutura como serviço seja alugada ou compartilhada entre provedores de recursos para melhoria dos serviços ofertados.

As principais características da CNS são:

- Implantação simultânea de várias fatias lógicas independentes, compartilhadas ou particionadas, em uma plataforma de infraestrutura comum;
- Suporte dinâmico a vários serviços, *multi-tenancy* e meios de integração via *market-place*;
- Separação de funções, simplificando o provisionamento de serviços, gerenciamento de rede e integração para suporte de serviços;
- Redução de custos de operadores de serviços e infraestrutura, fomentando a inovação para oferta de serviços personalizados sem alterar a infraestrutura física;

- Capacidade de atualizar, além dos aplicativos, a estrutura virtual de rede, sem alteração física;
- Monitoramento e provisionamento sob demanda de infraestrutura virtual;
- Oferta de serviços de terceiros sob demanda, capacitando parceiros para ofertas aos clientes finais, aumentando a rede da operadora e gerando valor sem aumento de custos.

As funcionalidades da arquitetura NECOS não se aplicam apenas às limitações dos serviços de *cloud computing*, mas podem ser aplicadas com sucesso na solução de desafios de orquestração em *edge computing*. Entretanto, o NECOS apresenta algumas limitações em ambientes compostos por dispositivos de borda com restrição de recursos. A principal limitação do NECOS se refere ao provisionamento da CNS, que geralmente é provisionada com equipamentos não compartilhados, ou seja, um dado elemento físico é dedicado exclusivamente a uma única CNS.

No caso da borda da rede, é comum encontrar limitação ainda maior de recursos que a descrita anteriormente e prevista pelo NECOS, então o provisionamento da CNS necessita ser feito em vários equipamentos. No entanto, alguns recursos IoT são escassos e de difícil acesso, como dispositivos de monitoramento localizados em áreas remotas, situação que possa demandar ainda que um único dispositivo comporte mais de uma CNS.

A dependência de *hypervisor* é outra limitação do NECOS quando se analisa a sua compatibilidade com dispositivos IoT que são geralmente baseados em arquitetura ARM e com recursos insuficientes para suporte a este tipo de virtualização, exigindo uma nova abordagem de virtualização e que garanta o isolamento entre *slices* (CLAYMAN et al., 2021).

A arquitetura funcional do NECOS é descrita na Figura 4, na qual são apresentados os seus elementos principais representados por 4 subgrupos: *Tenant*, LSDC *Slice Provider*, *Resource Marketplace* e *Resource Provider*.

O *tenant*, destacado em branco, refere-se a qualquer organização que necessite de *slices* para seus serviços, como uma CDN, *Content Delivery Network*, por exemplo. O *tenant* também pode atuar como fornecedor de infraestrutura virtual, nos casos de compartilhamento de infraestrutura, ofertando seus recursos ociosos para o NECOS. Este subgrupo é responsável por requisitar a criação e exclusão de *slices*, além da implantação de serviços nas *slices* criadas. Toda a interação é feita através da API do *Slice Provider* descrito a seguir.

No contexto deste trabalho, o *tenant* poderia ser um grupo de pesquisadores que demandem dados de sensores localizados em determinadas regiões e desejem contratar

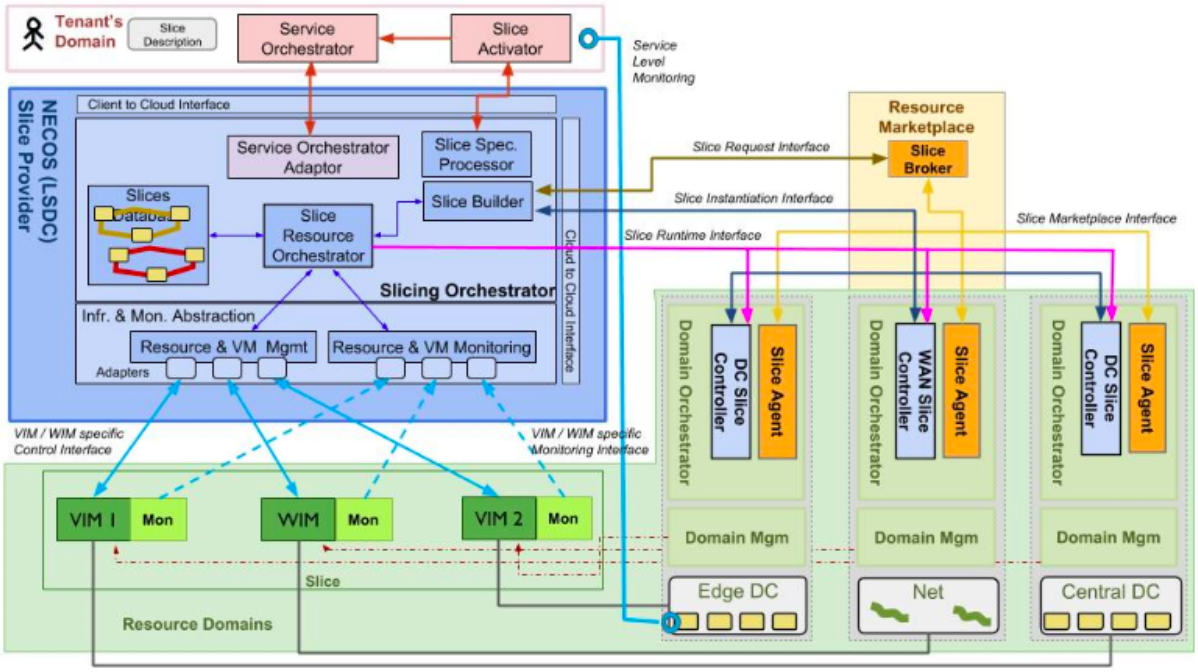

Figura 4 – Arquitetura Funcional NECOS (CLAYMAN, 2018).

serviço de *sensor-as-a-service* ou qualquer empresa que deseje orquestrar sensores e atuadores para aplicações diversas.

O *Slice Provider* (LSDC - *Lightweight Software Defined Cloud*), destacado em azul, é o subsistema que permite a criação de *slices* a partir de *slice parts*. Este subsistema é o responsável por atender as solicitações recebidas do *tenant* e construir a *slice* fim-a-fim solicitada, gerenciando também o seu ciclo de vida, monitorando, alterando ou destruindo a *slice* conforme necessário ou em casos parametrizados pelo *tenant*.

Este subsistema interage com o *Resource Marketplace* para localizar as *slice parts* presentes nos diferentes provedores de recursos que serão combinadas para a montagem da *slice* fim-a-fim. Então a topologia criada é salva em um banco de dados de controle chamado *Slice Database*, através do qual o NECOS pode consultar informações para manutenção da *slice*

O LSDC também apresenta a funcionalidade de implantação de serviços nas *slices* criadas, gerenciando a implantação e atualização dos *softwares* solicitados pelo *tenant*. Neste caso, o NECOS consulta a topologia da *slice* a fim de implantar o serviço da forma solicitada pelo *tenant*.

O *Resource Marketplace*, destacado em amarelo, é um repositório de informações a respeito dos recursos federados disponíveis para serem compostos em *slices*. Os recursos ofertados são listados no *broker* que permite filtro de *slice parts* por especificações desejadas, como por exemplo, por localização geográfica, por custo financeiro, características do *hardware*, entre outros.

Este subgrupo objetiva facilitar a oferta de recursos pelos *Resource Providers* descritos a seguir ao passo que mantém um diretório centralizado através do qual o *Slice Provider* pode consultar por recursos necessários para a composição da *slice* planejada. Estes repositórios podem ser públicos ou privados para o caso de compartilhamento de recursos entre parceiros.

O NECOS prevê a atualização dos dados do *Resource Marketplace* de duas formas, através de modo *push*, no qual o *Slice Broker* busca por recursos em um catálogo de recursos ou através do modo *pull*, no qual o *Slice Broker* consulta cada provedor de recursos para identificar se os recursos desejados estão disponíveis. O componente responsável por esta inteface com o provedor de recursos é o *Slice Agent*, responsável por responder às solicitações do *Slice Broker* sobre os recursos disponíveis na borda da rede e também atualizar o *Resource Marketplace* com esta lista de recursos

O *Resource Provider*, destacado em verde, refere-se as organizações que disponibilizam os recursos físicos e virtuais para o NECOS construir a *slice*, como dispositivos de rede, servidores em *data centers* e, especificamente, no âmbito deste trabalho, os provedores de dispositivos IoT que ofereçam dispositivos sob demanda, de uso exclusivo ou compartilhado. Este subgrupo deve fornecer API para permitir que o *Slice Provider* construa as *slices* solicitadas pelo *tenant*, além de manter interface com o *Resource Marketplace* para manter as informações atualizadas de recursos ofertados.

Este subgrupo possui 2 componentes principais, a saber: *DC Slice Controller* e *WAN Slice Controller*, responsáveis pela implantação e configuração dos ambientes virtuais nos dispositivos (máquinas virtuais, *containers* etc) e conexão entre *slice parts*, respectivamente.

O fluxo de criação de uma *slice*, citando cada componente NECOS envolvido no processo, está destacado na Figura 5 e é descrito a seguir. Primeiramente, o *tenant* descreve as características da *slice* desejada através de uma API do LSDC (I1). Após isso, o *Slice Builder* consulta o *Resource Marketplace* sobre a disponibilidade das *slice parts* com aquelas características entre os diversos domínios de recursos, que por sua vez verifica sua base de dados que contém a lista de *slice parts* listadas (I2) e, caso necessário, consulta o *Slice Agent* de cada provedor de recursos verificando as suas disponibilidades (I3). O *Slice Builder* combina então as *slice parts* em uma *slice* agregada (I4). Para esta tarefa, há suporte do *Slice Resource Orchestrator* que orquestra as *slices*, controlando seu ciclo de vida e todos os serviços das *slice parts* integradas (I5). Finalmente, a topologia da *slice* é salva no *Slice Database* para controle.

A topologia salva servirá como referência futura para que o *tenant* solicite a implantação dos serviços que serão suportados pela *slice*, sendo o fluxo de implantação do serviço descrito a seguir e presente na Figura 5 (I6). Primeiramente, o *tenant* envia a descrição dos passos necessários para implantação do serviço desejado através da API do

LSDC que coordena a implantação do serviço através do componente IMA (*Infrastructure and Monitoring Abstraction*).

O IMA é o componente responsável pelo monitoramento da *slice* criada através da coleta de KPIs, além da implantação do serviço descrito pelo *tenant*.

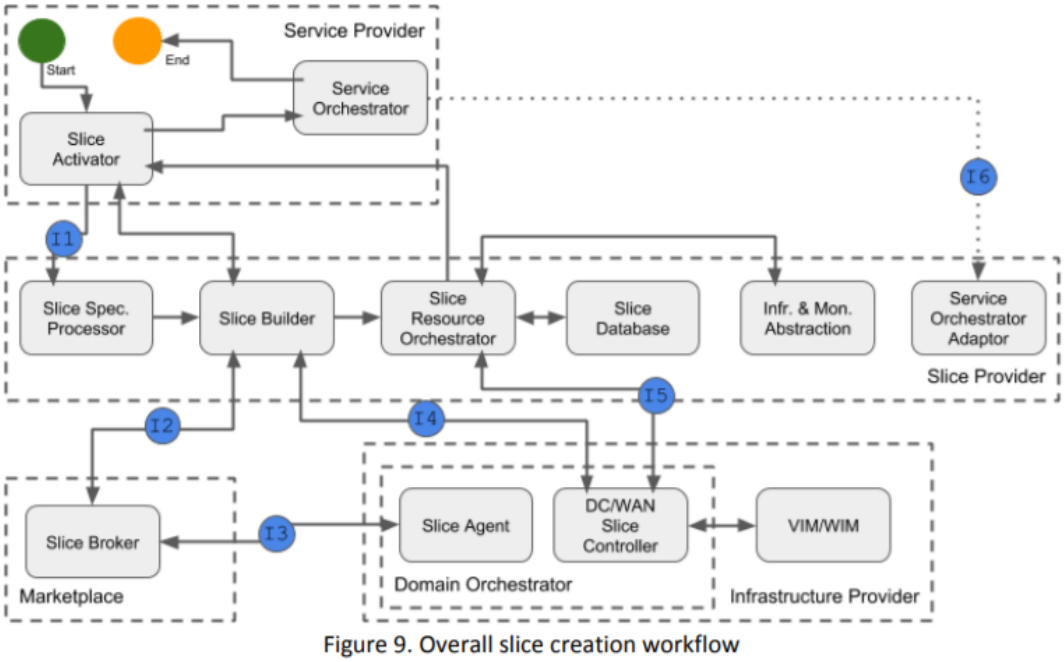

Figura 5 – *Workflow* padrão do NECOS para criação de CNS e implantação de serviço (CLAYMAN, 2018).

Desta forma, observa-se que o NECOS orquestra não somente a *slice* fim-a-fim, mas também implanta o serviço suportado por esta *slice*, além de monitorar seu funcionamento para que em casos específicos esta seja recriada ou destruída.

Cabe destacar que as implementações realizadas dentro do projeto NECOS têm como foco criar *slices* usando como infraestrutura servidores de alto desempenho, geralmente localizados em grandes *data centers*. Tais infraestruturas são cobertas por conexão de dados e alimentação elétrica redundantes, além de acesso relativamente fácil para equipes técnicas de manutenção dos equipamentos físicos, o que pode não ser uma realidade em cenários envolvendo computação na borda com dispositivos de baixo processamento e armazenamento.

O NECOS prevê em seu projeto inicial os ambientes IoT do ponto de vista da *cloud* orquestrada dinamicamente para suportar este tipo de ambiente, mas este trabalho pretende abordar também a orquestração do dispositivo IoT pretendendo, inclusive que este suporte a CNS, não apenas consuma serviços disponibilizados por *slices* na *cloud*.

No capítulo seguinte é descrita a proposta para extensão dos componentes NECOS para suporte aos dispositivos IoT com restrição de recursos

1.6 Trabalhos Relacionados

Em alguns trabalhos foram propostas arquiteturas baseadas em recursos federados e *edge computing*, sendo utilizados como referência para este trabalho. Porém, nenhuma proposta identificada ofereceu orquestração de CNS ou equivalente em dispositivos com restrição de recursos, como as encontradas em dispositivos IoT, utilizando uma plataforma com VIM (*virtual infrastructure manager*) sob demanda e oferta de recursos via *slice broker* como o NECOS.

O trabalho proposto em (ALAM et al., 2018) cita uma arquitetura baseada em *containers* Docker para a redução de latência na comunicação entre dispositivos IoT localizados na *edge* e a *cloud*. A proposta visa contornar a alta latência entre as duas camadas, adicionando-se uma camada intermediária, chamada de *fog*, representada por um *gateway*, que fica em um local da rede mais próximo ao dispositivo IoT e comportando parte do serviço que anteriormente estava na *cloud*. A orquestração deste trabalho é feita através de um *software* orquestrador, Docker Swarm (DOCKER, 2021b), localizado na *cloud*.

Nesta proposta, a *cloud* se responsabiliza pelo armazenamento e visualização de dados históricos, além da orquestração das demais camadas. Já a *fog* é responsável por serviços que geralmente são disponibilizados na *cloud*, como transformação de dados e análises, além da orquestração da rede através de NFV e, por fim, os dispositivos IoT responsáveis por capturar dados de sensores e executar pequenos processamentos. As 3 camadas se comunicam entre si por um *messaging hub publish-subscribe* fixo e já configurado por padrão.

Entretanto, a arquitetura proposta é fixa (IoT, *fog*, *cloud*) e não prevê a orquestração de *slice* no dispositivo IoT que é provisionado em *hardwares* exclusivos, sem compartilhamento de recursos e comunicando-se com o *messaging hub* via API, portanto não permitindo implantação dinâmica de serviço no dispositivo IoT. Nesta arquitetura há 3 níveis: dispositivo IoT (*edge*), *gateway* (*fog*) e *cloud*.

A principal desvantagem desta arquitetura em relação à baseada em CNS é que a arquitetura prevê poucos cenários e a *slice* ocupa todo o dispositivo orquestrado, gerando uso desnecessário de recursos.

Outra proposta, citada em (CHAKRABORTI et al., 2018), propõe uma arquitetura baseada em *edge computing* para ICN (*Information Centric Networking*) utilizando *slicing* em *gateways* para melhoria da qualidade de serviços IoT com a redução de latência em ambientes com restrição de recursos e, também utiliza SBC como dispositivo IoT na prova de conceito.

Esta proposta também utiliza Docker Swarm para orquestração de serviços e *containers*, cujas regras foram criadas através de um aplicativo para *smartphone* Android.

Há apenas serviços fixos baseados em *containers* que podem ser orquestrados no *gateway*, como por exemplo, autenticação de novos dispositivos IoT.

Entretanto, orquestra virtualização leve somente em dispositivos com altos recursos (*gateways*), diferenciando-se também deste trabalho. Nesta proposta, apenas os *gateways* IoT são orquestrados com virtualização leve através de *containers*. Já os dispositivos IoT não utilizam virtualização e já possuem softwares anteriormente implantados para anúncio e comunicação com o *gateway*.

Logo, sem virtualização do dispositivo IoT não é possível executar orquestração deste, somente do *gateway*, eliminando uma das vantagens de *slicing* para ambientes IoT que é a conversão do elemento físico em um elemento da rede distribuída ao passo que os serviços passam a ser providos por hardware virtualizado.

O trabalho realizado em (FERNANDEZ; VIDAL; VALERA, 2019), é o mais próximos ao proposto nesta pesquisa. Nele, os autores apresentam uma proposta de orquestração de *slices* em dispositivos IoT, cuja prova de conceito utiliza SBC, além de um orquestrador de *slices*. Porém, a proposta não prevê múltiplas *slices* por dispositivo e um *broker* de recursos para oferta de *slice parts* por provedores de recursos. A arquitetura é dividida em *cloud* e *edge* com dispositivos orquestrados através do *software* Kubernetes (FOUNDATION, 2020).

É proposto um orquestrador de *slices* na *cloud* e *edge* com o objetivo de lidar com a heterogeneidade das aplicações IoT (diversos tipos de dispositivos, conectividade e protocolos) e reduzir a latência das aplicações através da implantação dinâmica de funções IoT (IoT *gateways*, bancos de dados, *analytics*) em diferentes locais da rede de acordo com a exigência de cada aplicação.

A falta de suporte a múltiplas *slices* faz com que o *hardware* seja subaproveitado em alguns casos ao passo que comporta aplicações exclusivas de um único *tenant*, exigindo que novos hardwares sejam instalados no mesmo espaço físico para suporte de novas aplicações.

No contexto do NECOS o termo *tenant* se refere aos locatários que consomem recursos oferecidos por *data centers* para suporte às suas aplicações. Este ator utiliza e paga pelos serviços ofertados ao passo que os fornecedores dos recursos respeitam os níveis de garantias acordados (SILVA et al., 2018). Logo, no contexto deste trabalho o conceito se aplica ao locatário da *slice* e, por consequência das partes que a suportam, como os SBC, por exemplo.

Logo, a principal vantagem de uma proposta baseada na plataforma NECOS em relação à proposta acima é o uso de componentes considerados estado da arte em orquestração de *slice parts* ofertadas via *slice broker* permitindo o provisionamento de serviços baseados em características controladas dinamicamente, como custo, qualidade de serviço, localização geográfica etc.

Slice parts são recursos federados e localizados em diferentes provedores de infraestrutura, disponíveis para a construção de uma *slice*. No contexto do NECOS estes recursos podem ser ofertados e negociados de forma autônoma através de um *marketplace* (CLAYMAN, 2018).

Já (RIBEIRO et al., 2019) apresenta um trabalho utilizando a plataforma NECOS para a entrega de serviços na borda da rede, porém os dispositivos utilizados são computadores com disponibilidade de recursos computacionais, de armazenamento e de rede, além de possuir alimentação elétrica constante. Tal trabalho propõe um Centro de Dados Itinerante (*Itinerant Data Center* - IDC), localizado em um barco, que hospeda microsserviços sob demanda para levar serviços essenciais a regiões sem ou com pouca infraestrutura, através da migração de serviços da *cloud* para a borda da rede. Esses serviços são alocados em CNS orquestradas pelo NECOS no IDC.

Então, observa-se que o trabalho citado não prevê as restrições de recursos de armazenamento, computação, rede e alimentação ao mesmo tempo como a proposta deste trabalho.

Embora (RIBEIRO et al., 2019) não observe todas as restrições deste trabalho, é um caso interessante de aplicação real do NECOS em projeto de *edge computing* servindo como importante referência em orquestração em ambientes assíncronos, com restrição de conectividade, algo muito comum em serviços IoT

Desta forma, as principais diferenças deste trabalho em relação aos trabalhos descritos acima são: (1) os dispositivos da borda suportam múltiplas *slices* promovendo melhor uso de recursos que podem ser compartilhados por aplicações distintas e isoladas; (2) o conceito de *slice parts* que permite maior dinamicidade na orquestração dos recursos para compor a arquitetura; (3) oferta de *slice parts* através de *marketplace* tornando dinâmica a arquitetura da solução; (4) virtualização dos dispositivos IoT permitindo oferta de sensores e atuadores de forma dinâmica no sistema distribuído.

A Tabela 1 apresenta um comparativo entre as características dos projetos relacionados apresentados nesta seção e a proposta deste trabalho. Os trabalhos estão organizados conforme a seguir: (I) ALAM et al. (2018); (II) CHAKRABORTI et al. (2018); (III) FERNANDEZ et al. (2019); (IV) RIBEIRO et al. (2019); (V) Este trabalho.

Tabela 1 – Comparativo entre trabalhos relacionados.

	Características	Pontos negativos
I	Arquitetura fixa (IoT, *gateway, cloud*) para redução de latência na comunicação entre dispositivos IoT e a *cloud*.	A *slice* ocupa todo o dispositivo da borda com uso desnecessário de recursos.
II	Arquitetura baseada em *edge computing* para orquestração de *slices* em *gateways* IoT.	Apenas os *gateways* IoT são orquestrados através de *containers*, logo os dispositivos IoT não utilizam virtualização.
III	Arquitetura prevê orquestração de *slices* em dispositivos IoT e utiliza SBC na prova de conceito.	Não prevê múltiplas *slices* no dispositivo IoT, além de não ofertar *slices parts* ou similares via *marketplace* de recursos.
IV	Arquitetura para migração de serviços da *cloud* para a borda da rede utilizando orquestração de *slices* em servidores utilizando NECOS.	Prevê somente restrição de comunicação, além de não utilizar dispositivos IoT.
V	Arquitetura para orquestração de CNS em dispositivos com restrição de recursos e suporte à múltiplas *slices*, promovendo melhor uso de recursos que podem ser compartilhados por aplicações distintas e isoladas, além da oferta de *slice parts* via *marketplace* permitindo maior dinamicidade na orquestração dos recursos para compor a arquitetura da solução.	

2 Detalhamento da Proposta

Este capítulo descreve a proposta de implementação e extensão dos componentes da arquitetura NECOS para criação de CNS em ambientes de borda de rede com restrição de recursos, principalmente ambientes IoT baseados em sensores.

As inovações apresentadas pelo NECOS para ambientes de nuvem (*cloud computing*) podem ser estendidas com sucesso para os ambientes da borda da rede (*edge computing*) permitindo a orquestração completa do sistema distribuído contemplando computação (processamento), rede e armazenamento, mesmo em equipamentos com recursos restritos e ambientes heterogêneos.

O NECOS também apresentou alto grau de maturidade em projetos de telecomunicações envolvendo a virtualização de funções de rede, inclusive com casos de uso apresentados com sucesso em alguns trabalhos. Entretanto, pouco foi abordado sobre orquestração em ambientes restritivos como IoT, logo estas restrições demandam prováveis novas abordagens dos componentes NECOS para tal suporte.

Importante citar que algumas características tornam o NECOS confiável para novas propostas para elevação do potencial de atendimento às exigências dos serviços IoT como:

- Independência de fornecedor de *hardware*: independente do dispositivo que comporta a *slice*, o serviço IoT terá os mesmos recursos virtuais e mesmas funcionalidades;
- Redução de custos: *hardwares* mais baratos com serviços transparentes reduzem custo de aquisição e manutenção, além da redução no consumo energético ao passo que menos *hardware* é necessário para tarefas diversas;
- Confiabilidade: redundância, distribuição geográfica e transparência de *hardware* permitem re-provisionamento de uma *slice* em situações de falha do ambiente atual e para manter qualidade mínima de serviço (QoS);
- Flexibilidade: serviços baseados em multidomínios e multiplataformas executados de forma transparente permitem adequações para atendimento de demanda mantendo custo adequado;
- Escalabilidade: o fácil provisionamento da *slice* permite a substituição de *hardware* ou adição de novos para atendimento da demanda;
- Elasticidade: a característica de aumento ou redução dos recursos de uma *slice* permite melhor aproveitamento de recursos, como *hardware* de IoT, que podem ser caros e exclusivos;

- Segurança: isolamento da *slice* em dispositivos compartilhados, além da possibilidade de implantação e atualização automática de serviços podem elevar o nível de segurança de sistemas IoT;

- Eficiência: adaptação do serviço de forma dinâmica a novas redes de comunicação, distribuição geográfica e demais itens já citados podem garantir eficiência do serviço IoT em determinados períodos, além da redução do número de *hardwares* ociosos;

- Ciclo de vida: o controle do ciclo de vida da *slice* através de provisionamento, monitoramento e recuperação de falhas pode garantir níveis de qualidade mínima de serviço previamente estabelecidos (SLA);

- Ambiente de testes: possibilidade de testar uma aplicação IoT em ambientes de testes (*testbeds*) dinamicamente provisionados com custos reduzidos e dimensionado sob demanda e sem aquisição de *hardwares* específicos (*hardware-as-a-service*), reduzindo os riscos inerentes aos projetos IoT;

- Privacidade: todas as vantagens citadas perderiam totalmente a importância se não fosse garantida a privacidade de aplicações que porventura dividam o mesmo *hardware*. Desta forma, é previsto no projeto NECOS o isolamento entre aplicações implantadas em ambientes com múltiplas *slices*.

Entretanto, os componentes especificados e implementados no projeto NECOS focaram na criação e gerência de *slices* em ambientes de infraestrutura farta como médios e grandes *data centers*, cobertos por boa conexão de dados e alimentação elétrica redundantes. Desta forma, no âmbito deste trabalho, alguns componentes necessitaram ser estendidos para a nova realidade de dispositivos distribuídos, com restrição de recursos.

Além de restritivos em recursos, os *hardwares* IoT apresentam heterogeneidade como característica principal, aumentando esforços para comunicação entre dispositivos e para implementação e implantação de *software*, o que pode ser resolvido através da virtualização destes dispositivos.

Embora os *hardwares* sejam restritivos em recursos, em muitos casos são também escassos e de difícil aquisição e substituição, então espera-se que tais equipamentos suportem mais de uma *slice* por dispositivo, prevendo os seguintes cenários:

- Um dispositivo comportando uma única *slice* com recursos exclusivos;

- Um dispositivo comportando múltiplas *slices* e compartilhando recursos;

- Uma *slice* ocupando mais de um dispositivo.

A orquestração deve ser capaz de atender a uma solicitação de criação de *slice* e a posterior implantação do serviço nesta *slice*. Para isso, o *tenant* define os requisitos de hardware necessários para a *slice* assim como as necessidades do serviço a ser implantado utilizando a notação NECOS definida no arquivo YAML para este fim.

Os fragmentos do arquivo YAML abaixo descrevem como exemplo uma orquestração através de uma solicitação em alto nível como a seguinte para uma *slice* em um dispositivo IoT localizado em um local inóspito de difícil acesso: "Crie um serviço de captura de dados ambientais em um dispositivo de borda com sensores localizados no Brasil para captura de dados de temperatura e umidade a cada 10 horas para estudo de determinado fator ambiental com determinada garantia de nível de serviço. Criar também servidores *cloud* localizados na Europa para armazenamento dos dados históricos."

Desta forma, os seguintes itens contidos no YAML apresentam cada parte da solicitação descrita:

Infraestrutura:

O YAML que descreve a *slice* cita a distribuição geográfica das *slice parts* da *slice* fim-a-fim no item *geographic* conforme fragmento abaixo.

```
// slice_definition.yaml
      slice-constraints:
         geographic: [BRAZIL, EUROPE]
         dc-slice-parts: 1
         edge-slice-parts: 3
         net-slice-parts: 3
```

Cada *slice part*, incluindo seu tipo (*type*), localização (*location*) e sistema operacional (*vdu-image*) são descritos conforme fragmento abaixo e são totalmente compatíveis, sem necessidade de alteração do padrão, para ambientes IoT.

```
// slice_definition.yaml
         - dc-slice-part:
             name: edge-slice-brazil
             type: EDGE
             location: AMERICA.BRAZIL
             dc-slice-controller:
                 dc-slice-provider: undefined
                 ip: undefined
                 port: undefined
             VIM:
                 name: container-vim
                 version: undefined
```

```
            vim-shared: false
            vim-federated: false
            vim-ref: undefined
            host-count: 1
            vdus:
                - vdu:
                    id: docker-master
                    name: docker-master
                    description: Node for IoT deployment
                    instance-count: 1
                    hosting: SHARED
                    vdu-image: 'debian'
```

A conexão entre as *slice parts* é descrita no item *net-slice-part* onde é citado o tipo de conexão (WIM) e ligação entre os dispositivos físicos ou virtuais (*links*).

```
// slice_definition.yaml
        - net-slice-part:
            name: dc-spain-to-edge-brazil
            wan-slice-controller:
                wan-slice-provider: undefined
                ip: undefined
                port: undefined undefined
            WIM:
               name: GRE
               version: undefined
               wim-shared: true
               wim-federated: false
               wim-ref: undefined
            links:
               - dc-part1: core-dc
               - dc-part2: edge-slice-brazil
               - requirements:
                    bandwidth-GB: 1
            link-ends:
               link-end1-ip: undefined
               link-end2-ip: undefined
```

Implantação do serviço:

O fragmento abaixo descreve a localização do serviço, ou seja, a *slice part* onde cada serviço deve ser implantado, além dos comandos (*commands*) necessários para tal

implantação.

```
// service_definition.yaml
      id: IoT1_sliced
      slice-parts:
        - dc-slice-part:
          name: core-dc
          vdus:
            - vdu:
              name: core-vm
              VIM: VM
              namespace:
              commands:
                - git clone
                    https://github.com/rodrigoferrazazevedo/IOT-MQTT-Python.git
                - export DEBIAN_FRONTEND=noninteractive; sudo apt-get update;
                    sudo apt-get install mosquitto; sudo apt install -y
                    python-pip
                - sudo pip install -r IOT-MQTT-Python/requirements.txt
                - sudo python IOT-MQTT-Python/subscribe-cloudmqtt.py
        - dc-slice-part:
          name: edge-slice-brazil
          vdus:
            - vdu:
              name: docker-master
              VIM: Docker
              namespace:
              commands:
                - git clone
                    https://github.com/rodrigoferrazazevedo/IOT-MQTT-Python.git
                - export DEBIAN_FRONTEND=noninteractive; sudo apt-get update;
                    sudo apt install -y python-pip
                - sudo pip install -r IOT-MQTT-Python/requirements.txt
                - sudo python IOT-MQTT-Python/publish-cloudmqtt.py
```

O suporte para a criação de *slices* neste tipo de *hardware* restrito e compartilhado deve ser baseado em soluções leves de microsserviços porque virtualização convencional através de *hypervisor* demanda dispositivos com mais recursos disponíveis, além de apresentar incompatibilidade ou instabilidade em dispositivos ARM, arquitetura comum para dispositivos IoT. Os candidatos ideais para este uso são *container* (BERNSTEIN, 2014) e *unikernel* (Kurek, 2019).

Desta forma, os subsistemas *Resource Provider*, *Resource Marketplace* e IMA necessitam de extensão porque nestes dispositivos será necessário interagir com virtualização leve ao invés de máquinas virtuais sobre *hypervisor*. Já os demais componentes NECOS são totalmente compatíveis ao cenário proposto, a saber: LSDC *Slice Provider* e arquivos descritivos YAML enviados pelo *tenant*.

A Figura 6 destaca a arquitetura proposta e a seguir é descrito como cada subsistema NECOS e seus componentes podem ser implementados para suportar um ambiente IoT.

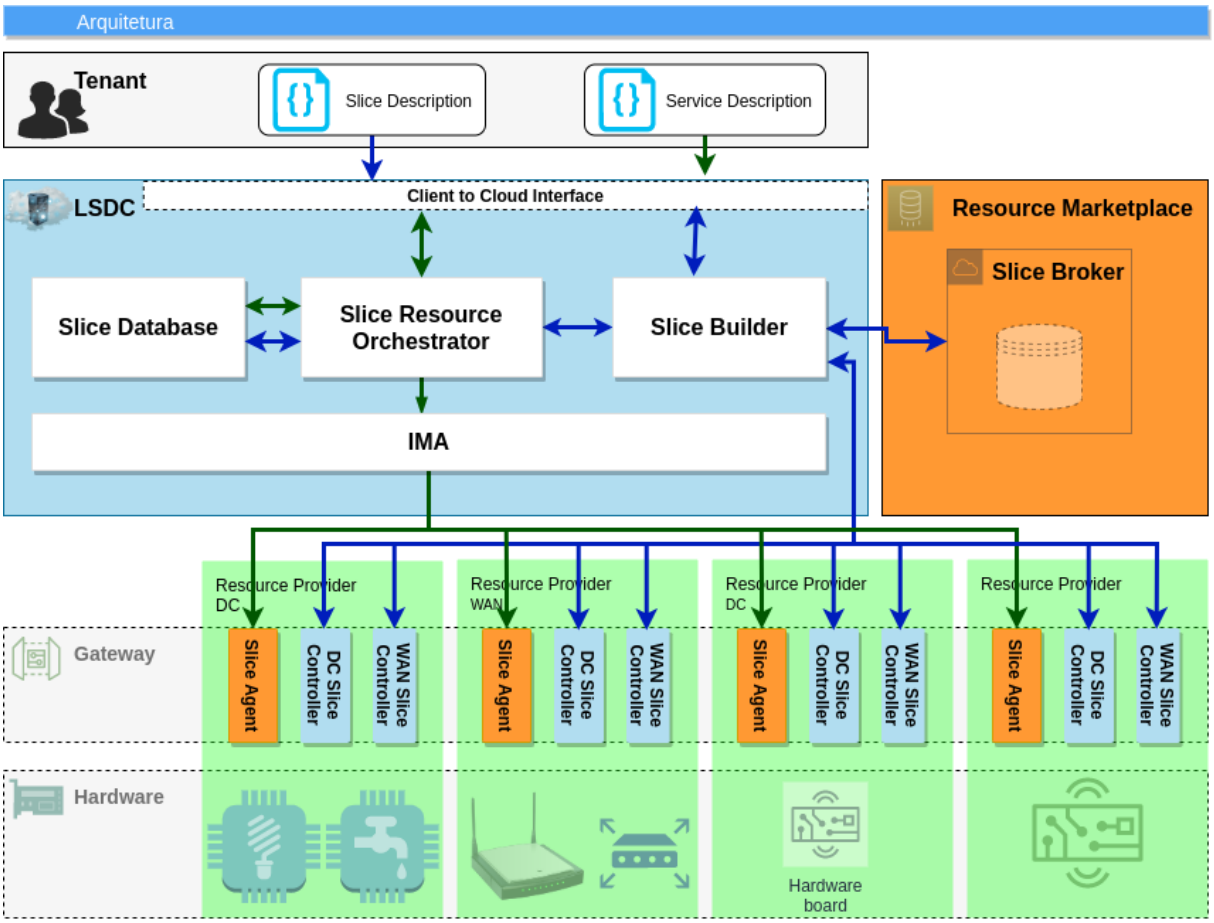

Figura 6 – Proposta da arquitetura de *slice* como serviço para IoT.

No contexto do NECOS, o mesmo dispositivo que comporta a *slice* pode comportar os componentes do NECOS. Porém, em ambientes com recursos muito restritos, como é o caso de ambientes IoT cujos equipamentos apresentam pouco espaço de armazenamento, geralmente utilizando cartões SD, além de quantidade de memória RAM restrita, o ideal é que estes dispositivos não comportem componentes que possam ser instalados em outros *hardwares*. Sendo assim, nesta proposta tais componentes NECOS foram implantados em *gateways* presentes na borda da rede onde se encontram os dispositivos IoT, tal como sugerido por (ALAM et al., 2018).

Desta forma, os componentes dos subgrupos abaixo são instalados em um ou mais *gateways* na borda da rede para gestão das *slices* nos dispositivos IoT com restrição de

recursos na borda da rede, sendo candidatos a *gateways* computadores convencionais ou modelos de *gateways* específicos para este fim (DELL, 2021).

A comunicação entre os componentes NECOS e os dispositivos IoT deve ser feita através de *adapters* já preparados, como SSH para dispositivos IoT, ou ferramentas para automação de serviços *cloud*, como AWS CLI (SERVICES, 2021), por exemplo.

2.1 Resource Marketplace

Com relação aos componentes do *Resource Marketplace*, apenas o *Slice Agent* necessita de extensão nesta proposta para fazer interface fornecendo a lista de recursos disponíveis no provedor de recursos para locação. Este componente deve ser instalado em um *gateway*.

2.1.1 Slice Agent

Os dispositivos disponíveis em cada *Resource Provider* são anunciados ao *Slice Broker* via *Slice Agent* (SA) para que o NECOS disponibilize em seu *Resource Marketplace* os recursos físicos citados.

Nesta proposta, o *Slice Agent* é composto por uma interface e um banco de dados local onde são cadastrados os dispositivos, incluindo os dados de sensores e atuadores, ofertados e estes dados são sincronizados com o *Resource Marketplace* para persistência destas informações no banco de dados centralizado do NECOS.

Desta forma, o SA faz *push* de todos os dados dos dispositivos locais para o serviço localizado no *Slice Broker* que recebe e registra tais informações em seu banco de dados. Vale destacar que o NECOS também permite *pulling* para que o *Slice Broker* solicite a cada provedor a lista de recursos disponíveis em intervalos parametrizados ou no momento da criação da *slice*.

2.2 Resource Provider

Os componentes DC *Slice Controller* e WAN *Slice Controller* também são instalados em um *gateway* que fica responsável pela orquestração das *slice parts* presentes em sua rede.

2.2.1 DC Slice Controller

No contexto de um *data center*, o DC *Slice Controller* (DCSC) é responsável por instanciar máquinas virtuais em cada *slice part* que comportará a *slice*. Estes dispositivos, geralmente, são computadores em data centers.

Já no contexto deste trabalho, em um ambiente IoT, o DCSC é responsável por instanciar um ou mais *containers*, além de criar interfaces virtuais para isolar cada CNS localizada em um ou mais *containers*.

Conforme citado, o DCSC também é responsável por isolar as *slices* na rede local. Então, conforme Figura 7, para cada *slice* criada uma nova interface virtual é criada e conectada aos *containers* que fazem parte daquela SBC garantindo o isolamento.

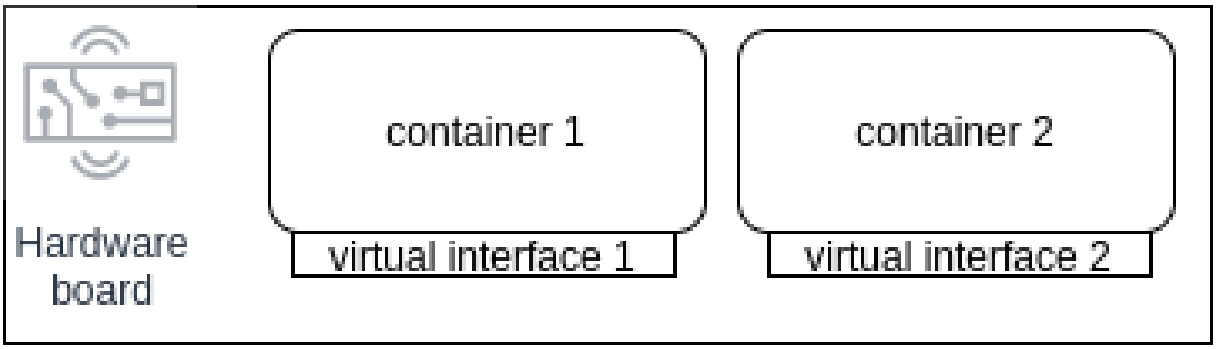

Figura 7 – Proposta de isolamento entre *containers* no SBC.

Cada interface virtual criada pode adotar a nomenclatura equivalente ao ID da *slice* descrita pelo *tenant* no arquivo YAML ou utilizar nomenclatura exclusiva gerada aleatoriamente para melhor controle.

```
// slice_definition.yaml
slices:
   sliced:
      id: IoT1_sliced
      name: IoT1_sliced
```

2.2.2 WAN Slice Controller

O WAN *Slice Controller* (WSC) é responsável por conectar *slice parts* localizadas em diferentes provedores de recursos para garantir a continuidade da *slice* fim-a-fim.

Então, no contexto deste trabalho o WSC deve conectar *containers* localizados nos SBC e em redes distintas através de tunelamento virtual, além do ambiente do *core* localizado na *cloud* e que geralmente utiliza máquinas virtuais.

A Figura 8 destaca a arquitetura proposta para isolamento e conexão entre o SBC e demais *slice parts*. Por padrão, cada *container* é conectado na rede *docker0* que é configurada durante a instalação do Docker, mas neste caso os *containers* compartilhariam a mesma rede, não atendendo ao requisito de isolamento de *slice* do NECOS.

Desta forma, para cada *slice* é criada uma nova interface de rede virtual conectada ao *container* Docker equivalente àquela *slice*. Em seguida, cada uma destas interfaces

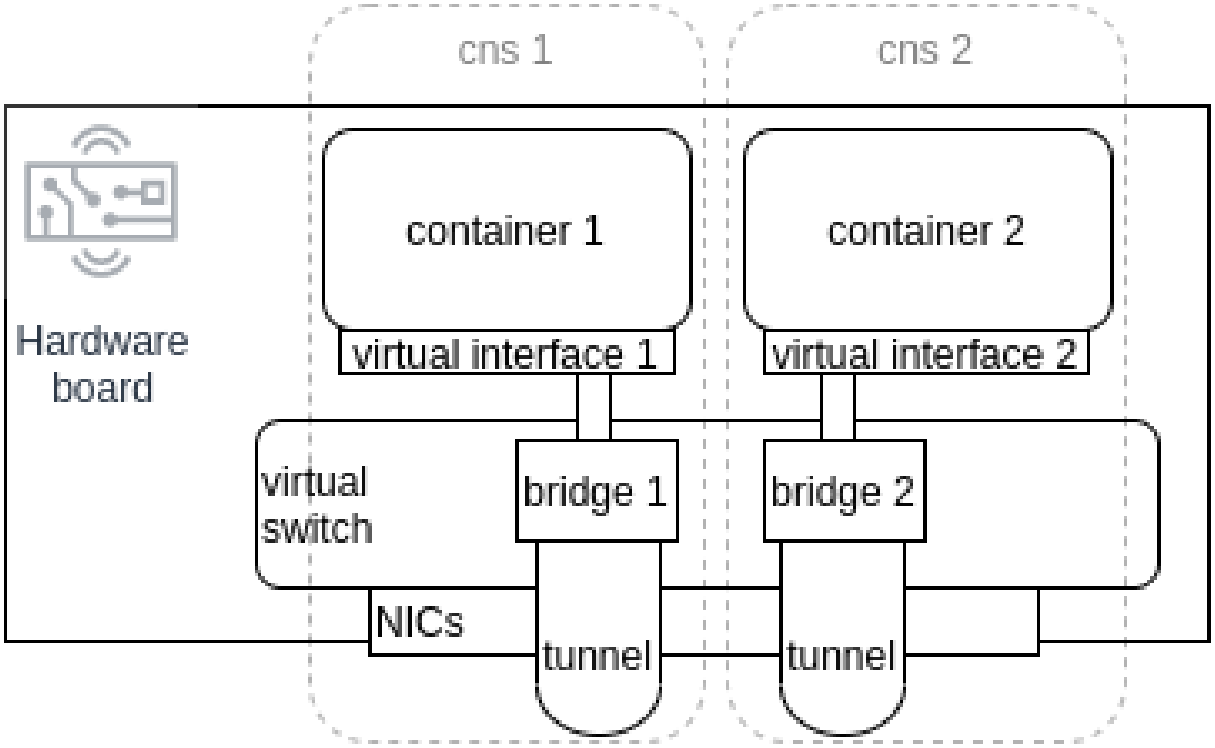

Figura 8 – Proposta de isolamento e conexão entre SBC e demais *slice parts*.

virtuais é conectada a um *switch* virtual (bridge 1 e bridge 2) para conexão com outras *slice parts* utilizando alguma tecnologia de tunelamento. Neste trabalho foi utilizado o tunelamento GRE (FARINACCI et al., 2000).

A criação das interfaces de rede virtuais fica a cargo do DCSC antes de iniciar cada *container*. Já a conexão via *switch* virtual e tunelamento ficam a cargo do WSC, que decide quais *slice parts* conectar baseado na descrição presente no arquivo YAML no item *net-slice-part* conforme fragmento abaixo.

```
// slice_definition.yaml
        - net-slice-part:
            name: dc-spain-to-edge-brazil
            wan-slice-controller:
                wan-slice-provider: undefined
                ip: undefined
                port: undefined undefined
            WIM:
               name: GRE
               version: undefined
               wim-shared: true
               wim-federated: false
               wim-ref: undefined
            links:
```

```
        - dc-part1: core-dc
        - dc-part2: edge-slice-brazil
        - requirements:
            bandwidth-GB: 1
      link-ends:
        link-end1-ip: undefined
        link-end2-ip: undefined
```

2.3 IMA

O subsistema IMA é responsável pela implantação dos serviços na *slice* criada, além do monitoramento das *slice parts* que formam cada slice. Esta proposta atual não prevê o monitoramento dos SBC, apenas a implantação dos serviços.

Sendo assim, após receber a confirmação da infraestrutura criada, o *tenant* pode enviar novo arquivo YAML descrevendo os serviços que deseja implantar, especificando o local para acesso ao código junto com a *slice part* onde deve ser implantado tal serviço.

```
// service_definition.yaml
    id: IoT1_sliced
    slice-parts:
    ...
      - dc-slice-part:
        name: edge-slice-brazil
        vdus:
        ...
            commands:
              - git clone
                  https://github.com/rodrigoferrazazevedo/IOT-MQTT-Python.git
              - export DEBIAN_FRONTEND=noninteractive; sudo apt-get update;
                  sudo apt install -y python-pip
              - sudo pip install -r IOT-MQTT-Python/requirements.txt
              - sudo python IOT-MQTT-Python/publish-cloudmqtt.py
```

2.4 Resumo dos componentes NECOS implementados ou estendidos

A Tabela 2 apresenta um resumo de cada componente NECOS utilizado neste trabalho e seu papel.

Tabela 2 – Descritivo dos componentes NECOS.

Componente	Papel
Slice Builder	Responsável pela requisição ao *Resource Marketplace* das diferentes *slice parts* que serão agregadas em uma *slice* a ser disponibilizada nos dispositivos IoT.
Slice Resource Orchestrator	Responsável pela orquestração das *slices* criadas a partir da agregação de *slice parts* e elementos de serviços.
Slice Database	Responsável por armazenar a topologia de infraestrutura criada e servir como base de informação para alteração e exclusão de cada *slice*, além de implantação de serviços nos dispositivos.
Slice Broker	Responsável por armazenar a lista de recursos ofertados por cada provedor de recursos a fim de permitir filtro por características desejadas pelo *tenant* (preço, localização etc).
IMA	Responsável pela implantação do serviço descrito pelo *tenant* em cada dispositivo.
DC *Slice Controller*	Responsável pela configuração e inicialização de cada unidade virtual (VDU) que compõe a *slice*.
WAN *Slice Controller*	Responsável pela conexão entre *slice parts*.
Slice Agent	Responsável por atualizar o *Slice Broker* com a lista de recursos disponível em cada provedor de recursos.

3 Implementação e Validação

Este capítulo descreve a implementação e avaliação da prova de conceito implementada para validar as novas funcionalidades. Foram utilizados *hardwares* reais e os componentes da arquitetura NECOS para criação de CNS em ambientes de borda de rede com restrição de recursos.

3.1 Implementação

Esta seção descreve a extensão e avaliação dos componentes da arquitetura NECOS para criação de CNS em ambientes de borda de rede com restrição de recursos. Uma prova de conceito foi implementada para validar as novas funcionalidades utilizando *hardwares* reais.

Como a arquitetura NECOS depende de virtualização para suporte à *slice* e SBC são incompatíveis com *hypervisor*, foi proposta neste trabalho a adaptação desta arquitetura para suporte a múltiplas *slices*. Para isso, foi feita uma mudança na forma de virtualização e reorganização de alguns componentes de borda da rede.

Os componentes foram implementados seguindo a especificação arquitetural NECOS, descrita na Figura 4. Entretanto, os componentes do subsistema *Resource Provider* (RP) precisaram ser estendidos para interação com os *containers* localizados nos SBC, a saber o DC *Slice Controller* (DCSC) e WAN *Slice Controller* (WSC). O subsistema de monitoramento (*Resource & VM Monitoring*) não foi implementado neste trabalho, uma vez que o monitoramento, tanto da infraestrutura quanto dos serviços, está além do escopo deste trabalho.

Embora o NECOS permita que toda estrutura da borda seja instanciada de forma virtual no mesmo *hardware* que comporta a *slice*, no contexto IoT este cenário não é recomendável, pois tal situação consumiria recursos, escassos por natureza neste contexto. Dessa forma, neste trabalho foi considerada uma nova camada de *hardware*, em forma de *gateway*, para receber os componentes de gerenciamento do NECOS na borda. Tal abordagem é utilizada em (XHAFA; KILIC; KRAUSE, 2020) para cenários IoT. Tipicamente, como trata-se de *hardware* de baixo processamento, implantar os artefatos na própria arquitetura do dispositivo ocuparia recursos deste com a própria solução, algo não recomendável em face às restrições.

Neste sentido, o *gateway* recebeu os três componentes descritos a seguir, que interagiram com os SBC via SSH e pode ser verificado na Figura 6.

- DCSC: responsável por isolar a rede de cada CNS através de interfaces de redes virtuais no dispositivo, instanciando *containers* Docker e os associando à devida CNS;
- WSC: responsável por realizar a conexão entre as *slice parts* distribuídas, seja na *edge* ou *cloud*, através de tunelamento por rede privada virtual;
- SA: responsável por atualizar o *Slice Broker* com a lista de dispositivos físicos disponíveis para compor a CNS.

A extensão necessária ocorreu nos algoritmos dos componentes DCSC e WSC para suportar a instanciação de *containers* diretamente nos SBC utilizando a arquitetura destacada na Figura 8. O DCSC e WSC também foram movidos para um novo equipamento.

Todos os componentes descritos anteriormente foram implementados utilizando a linguagem de programação Python e o *framework* Flask (PALLETS, 2021) para implementação dos microsserviços e gestão das requisições e respostas RESTful (RODRIGUEZ, 2008) porque os diversos componentes se comunicam entre si através de API REST baseada na especificação arquitetural NECOS. Cada componente foi armazenado em um *container* e as portas de cada microsserviço devidamente configuradas para funcionamento da comunicação da API.

Algumas bases de dados foram necessárias para suportar os três componentes na borda da rede, a saber: base de dados para armazenamento, inserção e consulta da lista de recursos disponibilizadas no provedor de recursos, além da exclusão de componentes indisponíveis. Neste trabalho optou-se por uma única base de dados na borda e compartilhada entre os três componentes por estarem localizados no mesmo local, no *gateway*, mas caso seja necessário armazenar dados em diferentes dispositivos basta replicar esta base de dados.

Além disso, duas bases de dados foram necessárias para suportar o *Slice Database*, para armazenamento da topologia da *slice* e, para suportar o *Slice Broker*, para armazenamento e gestão dos recursos disponíveis nos diversos provedores de recursos que porventura façam parte do sistema.

As informações de *hardware* dos dispositivos IoT oferecidos como recursos foram armazenadas no *Resource Marketplace* para que cada parque de dispositivos fosse listado como domínio de recursos. No caso de dispositivos IoT, os seguinte dados foram listados: localização geográfica, capacidade de processamento, quantidade de GPUs disponíveis e tipo de conexão de dados.

A base de dados utilizada foi o SQLite, que é uma biblioteca em linguagem C que implementa um banco de dados SQL embutido utilizando poucos recursos. O SQLite permite consultas, leituras, atualizações e exclusões através da linguagem SQL de forma

leve e ideal para uso em ambiente de microsserviços que armazenam pequena quantidade de dados (SQLITE, 2021).

A comunicação entre os componentes instalados no *gateway* e os SBC ocorreu por SSH e para tal funcionalidade ser implementada em Python foi utilizada a biblioteca Paramiko (PARAMIKO, 2021) que é bem utilizada no meio acadêmico e possui qualidade em lidar com conexões com e sem criptografia.

Já para a conexão entre as *slice parts* localizadas em diferentes redes foi utilizado um *switch* virtual, o Open vSwitch (OVS) (OPENVSWITCH, 2021a), que foi escolhido por suportar tunelamento GRE e ter capacidade de lidar com milhares de túneis simultâneos, suportando configuração remota para criação, configuração e exclusão de túneis, isolamento e permitindo a conexão de máquinas virtuais e *containers* localizados em diferentes redes de *data centers* (OPENVSWITCH, 2021b).

Para cada *slice* é criada uma interface virtual conectada exclusivamente a um VDU, além de um *switch* virtual exclusivo por *slice*, também um túnel GRE é criado entre *slice parts*. Após os elementos citados serem criados, todos são conectados a portas virtuais do OVS, desta forma o OVS gerencia de forma transparente o tráfego de dados entre todos os dispositivos conectados em suas portas, mantendo o isolamento.

Sendo assim, este trabalho analisou os seguintes cenários: *hardware* específico e dedicado para uma *slice* e *hardware* compartilhado. Estes cenários objetivaram avaliar se estes dispositivos com restrição de recursos comportariam uma ou mais *slices*, além de inferir o *hardware* mínimo e o *hardware* ideal baseado nos cenários analisados.

Dado que alguns *hardwares* de computação de borda tendem a apresentar alto custo de aquisição e manutenção, espera-se ainda que suportem múltiplas *slices*. Consequentemente, aliado à dificuldade em oferecer recursos computacionais mínimos para suporte a *hypervisor*, houve também a necessidade de substituir soluções típicas de virtualização do NECOS. Neste sentido, duas abordagens para virtualização leve foram consideradas neste projeto: *container* (DOCKER, 2021a) e *unikernel* (MADHAVAPEDDY et al., 2013).

No caso de *container* as escolhas iniciais foram o BalenaOS (BALENA, 2020), sistema operacional (SO) especializado em servir *conteiner* em dispositivos IoT ARM (ARM, 2020) com restrição de recursos, que é o caso dos SBC propostos neste trabalho. Já no caso do *unikernel*, foi considerada a utilização do MirageOS (XEN; FOUNDATION, 2020), SO especializado em servir *unikernel*.

No entanto, foram encontradas limitações durante a implementação e implantação de ambas as soluções, demandando pesquisa e teste de alternativas que fossem minimamente compatíveis com dispositivos ARM IoT.

Foram também encontradas limitações durante a implementação e implantação das soluções baseadas em *unikernel* nos dispositivos selecionados.

No caso do BalenaOS, o sistema operacional não se mostrou compatível com o SBC Dragonboard 410c, não permitindo instalação, então optou-se por utilizar Ubuntu Linux e em seguida instalar o Docker para gestão dos *containers* nos SBC.

Já as soluções baseadas em *unikernel* não se mostraram compatíveis com dispositivos ARM, não permitindo compilação direta nestes, além da dependência de *hypervisor* (PROJECT, 2021), inviabilizando sua orquestração. Os *hardwares* utilizados nesta pesquisa também não atenderam aos requisitos mínimos exigidos pelos *hypervisors* KVM (PROJECT, 2021) e Xen (CYTRIX, 2021). Os projetos *unikernel* testados foram MirageOS (MADHAVAPEDDY et al., 2013), OSv (OSV, 2021) e Unikraft (UNIKRAFT, 2021). Sendo assim, tal tecnologia deverá ser analisada em trabalhos futuros, quando as soluções baseadas em *unikernel* estejam mais maduras e aplicáveis a estes tipos de *hardware*.

Sendo assim, tal tecnologia deverá ser analisada em trabalhos futuros, quando as soluções baseadas em *unikernel* estejam mais maduras e aplicáveis a estes tipos de *hardware*.

Desta forma, os componentes estendidos conforme descrito no Capítulo 2 passaram a suportar orquestração de múltiplas *slices* utilizando *containers* para que fosse possível compartilhar dispositivos IoT entre aplicações distintas e isoladas na borda da rede. A Figura 6 descreve a arquitetura proposta, através da qual um *tenant* define as características da *slice* desejada através da API do LSDC. O *Slice Builder* então consulta o *Resource Marketplace* e, com auxílio do SRO, constrói a *slice* IoT que atenda aos requisitos descritos pelo dono do serviço.

O DC *Slice Controller*, por sua vez, implanta um dos seguintes cenários: um dispositivo dedicado exclusivamente a uma única *slice* com recursos exclusivos ou um dispositivo compartilhado com duas ou mais *slices*. A topologia da *slice* é então salva no *Slice Database* para controle.

A responsabilidade de gestão da conexão entre *slice parts* é do WAN *Slice Controller*, então este manipula através de comandos via SSH cada dispositivo.

Para a conexão de cada unidade virtual *container* de forma isolada com outras unidades de diferentes *slice parts*, foi utilizado um *switch* virtual, no caso, o software Open vSwitch, para gestão da conexão e tunelamento GRE entre os *containers*. Cada WAN *Slice Controller* localizado nas bordas da rede envia comandos de configuração do Open vSwitch para a criação de túneis GRE com dispositivos localizados na mesma rede ou externos àquela rede.

Os componentes do subsistema NECOS LSDC foram implementados sem alteração da especificação, com exceção do IMA, que precisou ser estendido para possibilitar a implantação de serviço em *container*.

Cabe relembrar que a implantação do serviço no NECOS ocorre através do envio

de um arquivo YAML que descreve o serviço a ser implantado. Tal implantação é realizada pelo subsistema IMA, que após receber a descrição citada consulta o *Slice Database* para entender a topologia da CNS criada e implantar o serviço nas *slice parts* corretas.

Para a interação com cada *container*, foi utilizado o comando *docker run* para a execução de comandos necessários descritos no YAML enviado pelo *tenant*. Após isso, uma iteração foi codificada para a execução sequencial dos comandos descritos conforme fragmento abaixo que exemplifica o download do código-fonte (*git clone*) do serviço a ser implantado, as configurações do ambiente e por fim a execução do serviço.

```
// service_definition.yaml
        ...
            commands:
              - git clone
                  https://github.com/rodrigoferrazazevedo/IOT-MQTT-Python.git
              - export DEBIAN_FRONTEND=noninteractive; sudo apt-get update;
                  sudo apt install -y python-pip
              - sudo pip install -r IOT-MQTT-Python/requirements.txt
              - sudo python IOT-MQTT-Python/publish-cloudmqtt.py
```

Neste trabalho foi utilizada, como exemplo de aplicação específica de domínio de recursos de IoT, uma prova de conceito representando um parques de sensores conectados presentes em uma reserva ambiental. Esses recursos são disponibilizados para a plataforma NECOS que, por sua vez, entrega-os na forma de *slices* para os *tenants* (uma instituição de pesquisa, por exemplo). Limitações características de dispositivos IoT como falta de garantia de conexão de dados, alimentação por baterias, entre outros fatores, tornam o LSDC do NECOS ideal para este tipo de ambiente se observada a sua capacidade de mapeamento de forma dinâmica dos componentes de serviços na *slice*. Isto é importante para serviços IoT que se encontram em locais de difícil acesso e necessitam ser atualizados ou reconfigurados dinamicamente para novos serviços.

O serviço de monitoração ambiental foi codificado em linguagem de programação Python. Este realiza a captura e tratamento dos dados de temperatura, umidade e altura que o sensor se encontra do chão, simulando um dispositivo conectado em uma floresta para análise ambiental remota ou mesmo servir como plataforma de testes para cientistas.

Neste trabalho, o serviço de monitoramento ambiental consistia em instalar parte do serviço na borda e parte na *core cloud* com o objetivo de capturar dados de sensores presentes nos SBC. O serviço foi composto por um servidor Mosquitto MQTT (FOUNDATION, 2021) implantado no *core* e outra aplicação Python implantada na borda que publicava dados no *core* utilizando o protocolo MQTT.

Como o MQTT é baseado no modelo *publish-subscribe*, o *software* da borda utilizou

o método *publish* para publicar (enviar) cada dado para o servidor implantado no *core*.

Para validar o envio e recebimento dos dados foi implantado manualmente um outro *script* Python no *core* para subscrição ao servidor Mosquitto MQTT e exibição na tela dos dados anteriormente enviados do SBC para o *core* confirmando o funcionamento da comunicação.

À seguir é descrito como a infraestrutura foi instanciada e o serviço foi implantado para validação da *slice* nos dispositivos SBC.

3.2 Validação

A validação deste projeto foi feita a partir da orquestração de uma infraestrutura na borda da rede de forma automatizada através da prova de conceito criada, então foi feita a posterior implantação de um serviço simulando um *software* como serviço de monitoração ambiental.

A primeira parte consistiu no envio de um arquivo YAML descrevendo uma infraestrutura composta por uma máquina virtual na *cloud* conectada aos SBC localizados na *edge*, representada por uma rede IEEE 802.11 (*WiFi*) conectando os SBC, sendo que para os experimentos os *containers* foram previamente armazenados em cada SBC para manter o ambiente o mais homogêneo possível. Como o objetivo da pesquisa é a análise do *hardware*, os dados da *cloud*, como latência da comunicação entre *edge* e *cloud*, foram desconsiderados.

Os seguintes cenários foram testados em *hardwares* SBC:

- Instanciação dinâmica de CNS através dos componentes NECOS estendidos;
- Instalação do serviço de monitoração ambiental através dos componentes NECOS estendidos.

Como apresentado anteriormente, uma *slice part* pode ser composta por diferentes recursos, tanto computacionais (servidores) quanto de redes. Entretanto, no nosso ambiente de testes, cada *slice part* é composta por apenas um SBC. Qualquer outra configuração com mais SBC por *slice part* é possível.

3.2.1 Ambiente de Testes

O ambiente de testes utilizado é composto por duas máquinas físicas, uma máquina virtual (*cloud*) e 4 SBC conectados em rede conforme representado na Figura 9 e descrito a seguir.

Uma máquina física Samsung com as seguintes configurações: processador Intel i7-8550U 8. geração com 1.8 Ghz, 16 GB de RAM e 256 GB de disco rígido. Esta máquina física fez o papel de LSDC e *Resource Maerketplace* onde está sendo executado o IMA, *Slice Builder*, *Slice Database*, *Slice Resource Orchestrator* e *Slice Broker*. O sistema operacional é o Linux Ubuntu 20.04 LTS.

Uma máquina física Samsung com as seguintes configurações: processador Intel i7-8550U 8. geração com 1.8 Ghz, 8 GB de RAM e 1 TB de disco rígido. Esta máquina física fez o papel de *gateway* onde está sendo executado o DC *Slice Controller*, WAN *Slice Controller* e *Slice Agent*. O sistema operacional é o Linux Fedora 32.

Os SBC escolhidos para os experimentos foram: Dragonboard 410c (processador ARM com 1,2 GHz, memória 1 GB e GPU), Raspberry Pi 3 (processador ARM com 1,2 GHz, memória 1 GB e GPU), Raspberry Pi 4 (processador ARM com 1,5 GHz, memória 4 GB e GPU) e NVIDIA Jetson Nano (processador ARM com 1,43 GHz, memória 4 GB e GPU). Estes dispositivos foram escolhidos por serem amplamente utilizados na prototipação de dispositivos IoT (JOHNSTON et al., 2016).

Em todos os SBC o sistema operacional Ubuntu Core foi instalado em SD com o objetivo de padronizar o armazenamento porque o Dragonboard é o único SBC com armazenamento interno disponível no formato eMMC que é mais rápido e estável que SD.

A proposta deste trabalho foi validada no ambiente descrito acima através do envio de um arquivo YAML para a API do LSDC descrevendo a infraestrutura a ser criada. Este primeiro envio é feito através de qualquer *software* ou método compatível com as instruções HTTP, pois é esperado que o *tenant* utilize a interação com o LSDC de forma agnóstica.

```
curl -X POST --data-binary @Tenant_IoT1_slice_specification.yaml -H
    "Content-type: text/x-yaml" http://192.168.1.106:5000/create_slice
```

Para cada *slice* foi criada uma interface de rede virtual em cada SBC que desempenhou o papel de *slice part*, então para prover conectividade entre elas, o WSC foi adaptado para conectar instâncias Docker na *edge* com as *slice parts* através de tunelamento virtual utilizando o protocolo GRE e para esta funcionalidade de tunelamento, foi utilizado o OVS instalado em cada um dos SBC para gestão dos túneis e isolamento entre as *slices* nos ambientes SBC com múltiplas *slices*.

3.2.2 Metodologia dos Testes

As avaliações de desempenho foram executadas instanciando-se a CNS e então apagando-a após sua criação. Em seguida cada teste foi repetido sem a reinicialização do SBC.

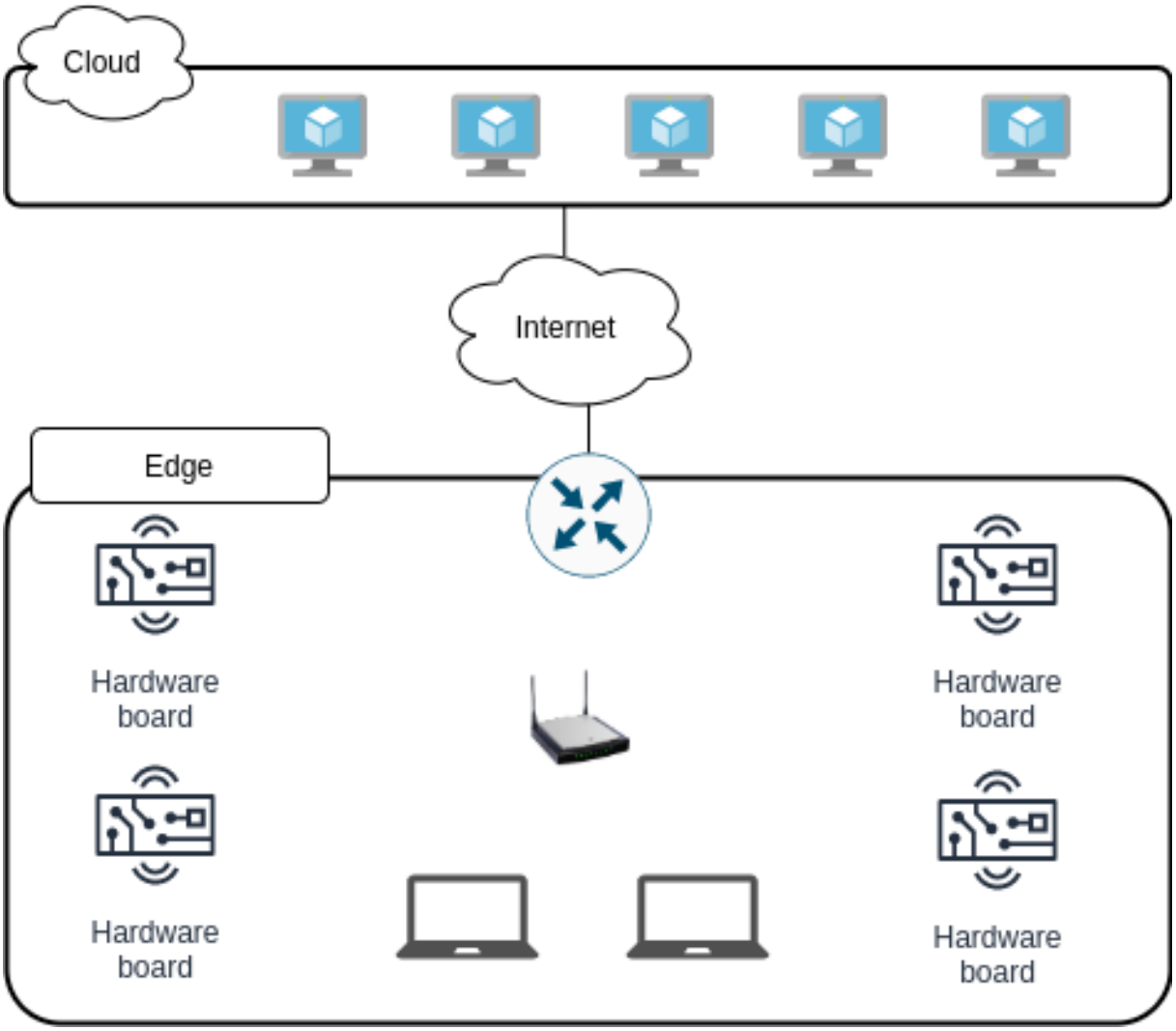

Figura 9 – Ambiente de testes.

O desempenho durante a criação de uma ou mais CNS foi capturado através do *software* de monitoramento Netdata (NETDATA, 2021), obtendo-se o consumo de CPU, consumo de memória e temperatura atingidos durante este processo. Cada resultado foi exportado de forma automática através da API do Netdata logo após cada teste e os resultados obtidos confirmaram a hipótese de suporte a múltiplas *slices* nestes dispositivos com recursos restritos.

Em uma primeira avaliação, foi analisado o consumo médio de CPU nos diferentes equipamentos. Para isso, foi variada a quantidade de CNS e VDU em cada *slice* da seguinte forma: 1 CNS com 1 VDU (Figura 10), 1 CNS com 2 VDU (Figura 11), 1 CNS com 3 VDU (Figura 12), 2 CNS com 1 VDU (Figura 13), 2 CNS com 2 VDU (Figura 14) e, finalmente, 2 CNS com 3 VDU (Figura 15) em cada SBC. O processo foi repetido 5 vezes para cada cenário.

O planejamento inicial para testes previa 30 repetições por cenário, mas devido aos

problemas descritos a seguir em razão às restrições do hardware foi necessária a redução para 5 repetições por cenário.

3.2.3 Resultados

Nesta seção são apresentados os resultados obtidos na realização dos testes executados no ambiente com *hardwares* reais que foram foco desta pesquisa a partir da metodologia descrita acima.

Os resultados foram divididos em três períodos para análise, sendo:

- *Network*: refere-se à grandeza medida equivalente ao período de tempo em que o DCSC fatia a rede local através de interfaces de redes virtuais para isolamento das CNS;
- *Container*: refere-se à grandeza medida no período de tempo em que o DCSC leva para instanciar os *containeres* da CNS nos SBC;
- WAN: refere-se à grandeza medida no período de tempo em que o WSC demanda para conectar aquela *slice part* à CNS.

Os resultados estão apresentados nas Figuras 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26 e 27. Os valores são relativos ao consumo de CPU e memória dos processos do sistema operacional, além da temperatura atingida, para gestão dos *containers* (*Network* e *Container*) e OpenvSwitch (OPENVSWITCH, 2021a) (WAN).

Cada teste foi repetido apenas 5 vezes porque os SBC com menos recursos, ou seja, Raspberry Pi 3 e Dragonboard, não suportaram um número maior de repetições em um curto espaço de tempo e perderam a comunicação com o RP (*Resource Provider*) após superaquecimento.

Interessante observar que o fato de ter que suportar mais de um *software* de suporte à solução que consome recursos consideráveis, ou seja, Docker e Open vSwitch acabam consumindo recursos do SBC que poderiam ser disponibilizados para a CNS.

3.2.3.1 Teste de consumo de CPU

Em uma primeira avaliação, foi analisado o consumo médio de CPU nos diferentes equipamentos. Para isso, foi variada a quantidade de CNS e VDU em cada *slice* da seguinte forma: 1 CNS com 1 VDU (Figura 10), 1 CNS com 2 VDU (Figura 11), 1 CNS com 3 VDU (Figura 12), 2 CNS com 1 VDU (Figura 13), 2 CNS com 2 VDU (Figura 14) e, finalmente, 2 CNS com 3 VDU (Figura 15) em cada SBC. O processo foi repetido 5 vezes para cada cenário.

O número de variações entre CNS e VDU entre os *hardwares* manteve o número máximo de 2 CNS com 3 VDU porque os dispositivos com menor disponibilidade de recursos, no caso o Dragonboard e o Raspberry Pi 3, não suportaram a carga de trabalho, algo perceptível nas Figuras 12 e 15, nas quais é possível observar o consumo elevado de CPU. Como o trabalho objetivou o estudo comparativo entre os *hardwares* foi mantido o padrão de testes entre os 4 dispositivos.

Como o consumo de CPU exibido nas figuras considera somente os processos de gestão de *containers* e Open vSwitch, se observado o consumo de CPU no momento em que cada SBC não está suportando CNS (estado de repouso), os valores justificam a indisponibilidade dos dispositivos na Figura 15:

- Dragonboard 410c: 3,5%;
- Raspberry Pi 3: 4%;
- Raspberry Pi 4: 7%;
- Jetson Nano: 5%.

Observando-se os valores aferidos em repouso é possível observar que o suporte aos *containers* consome demasiadamente o recurso de CPU, impactando os SBC com menos recursos, no caso, o Dragonboard 410c e o Raspberry Pi 3. Na Figura 10 é possível observar que o Raspberry Pi 3 apresenta consumo de CPU bem superior aos demais SBC nos cenários de gestão de *containers* (*Network* e *Container*), mas ainda apresenta menor consumo de CPU que o Dragonboard em WAN.

Esse comportamento já era esperado em *hardwares* com poucos recursos para gestão de ambientes virtuais. Em contrapartida, é possível observar que o Raspberry Pi 4 consumiu menos CPU em praticamente todos os cenários, algo que pode ser justificado pela sua frequência de 1,5 GHz, bem superior a 1,2 GHz dos dois SBC mais restritivos e 1,43 GHz do Jetson Nano. Também foi verificado que o Raspberry Pi 3 apresentou maior consumo de CPU, embora o Dragonboard aumentou consideravelmente seu consumo conforme novas VDU foram adicionadas.

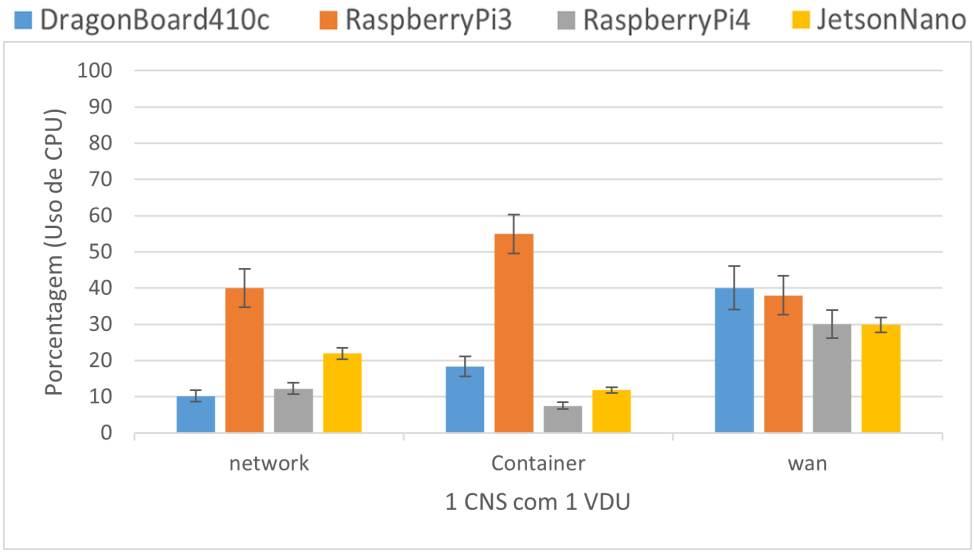

Figura 10 – Consumo de CPU durante a criação de 1 CNS com 1 VDU por SBC.

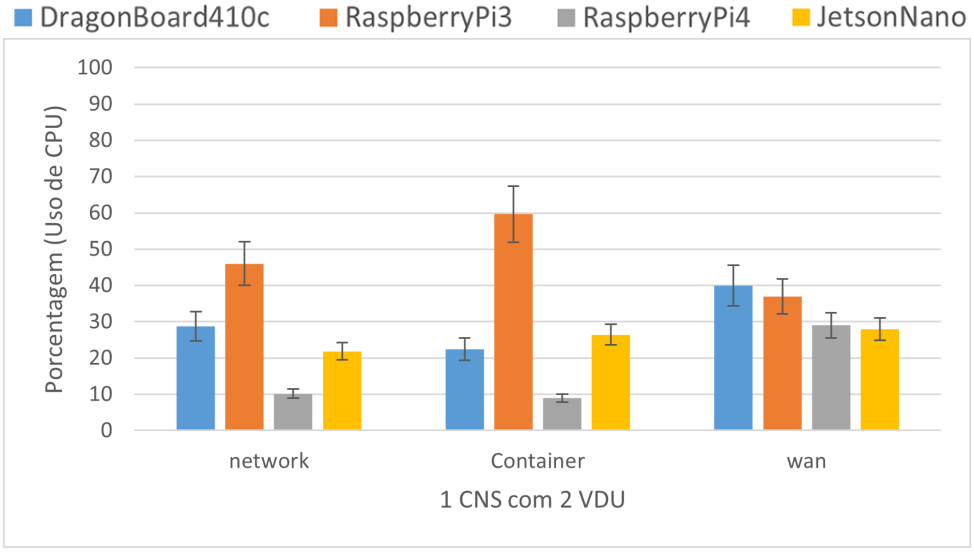

Figura 11 – Consumo de CPU durante a criação de 1 CNS com 2 VDU por SBC.

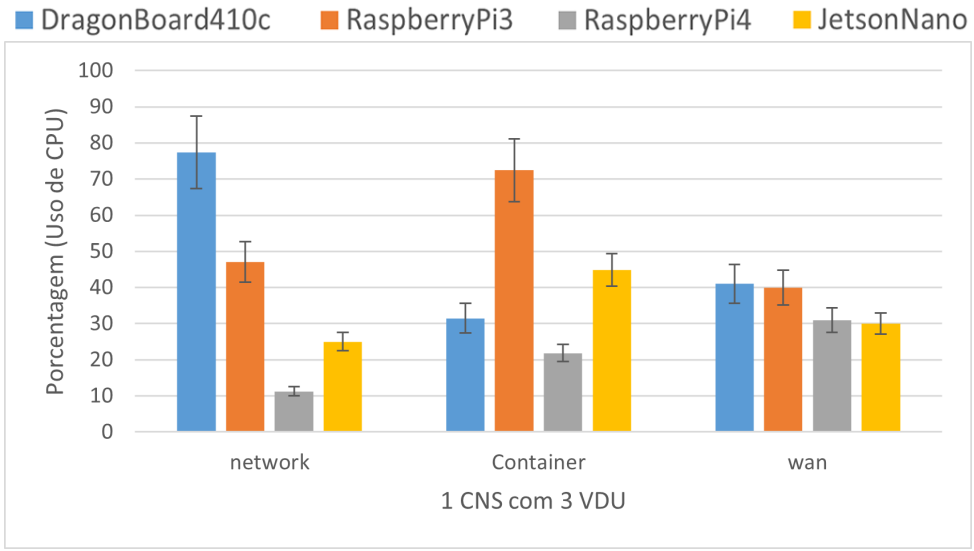

Figura 12 – Consumo de CPU durante a criação de 1 CNS com 3 VDU por SBC.

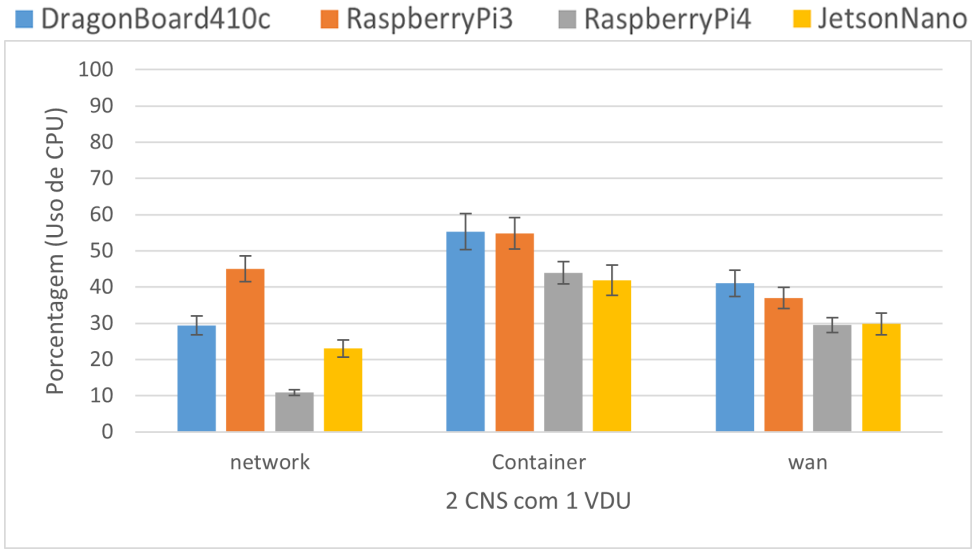

Figura 13 – Consumo de CPU durante a criação de 2 CNS com 1 VDU por SBC.

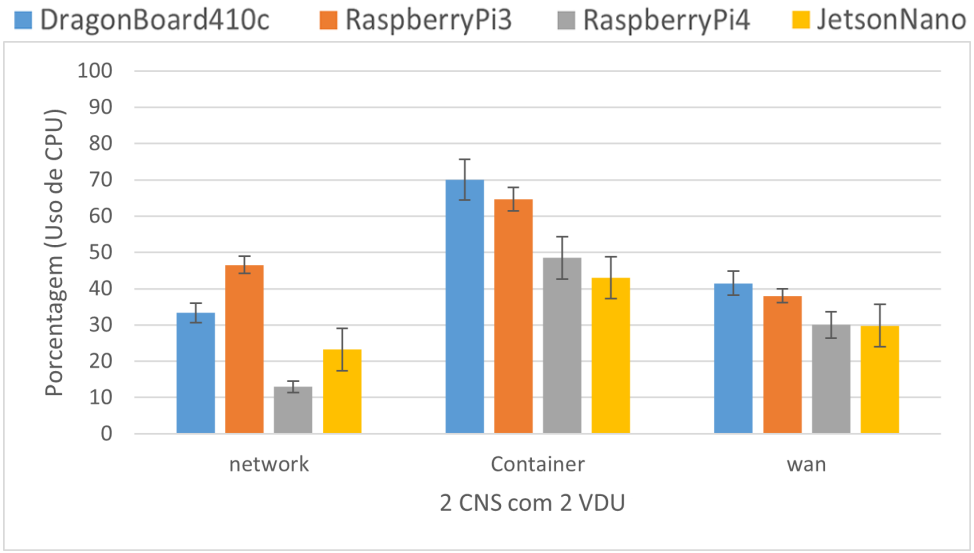

Figura 14 – Consumo de CPU durante a criação de 2 CNS com 2 VDU por SBC.

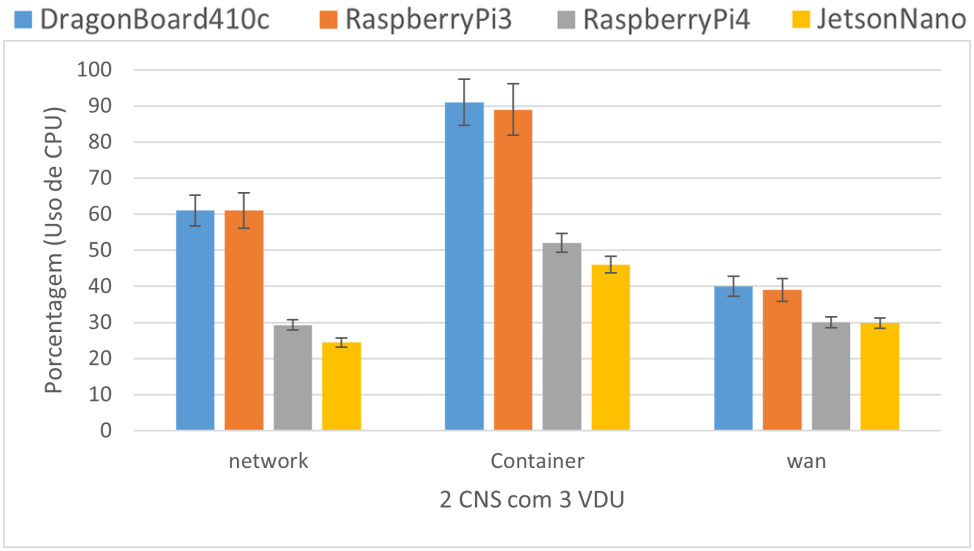

Figura 15 – Consumo de CPU durante a criação de 2 CNS com 3 VDU por SBC.

3.2.3.2 Teste de consumo de memória

Com relação ao consumo de memória, apresentado nas Figuras 16, 17, 18, 19, 20 e 21, é possível observar que o Dragonboard consumiu mais memória ao instanciar as CNS, mas ainda assim o consumo se manteve baixo em relação à quantidade deste recurso disponível em cada SBC. O Raspberry Pi 4 foi o SBC que menos consumiu memória. Já o Jetson Nano utilizou o recurso comparativamente acima do esperado se comparada a sua quantidade de memória disponível em relação aos demais SBC.

O Jetson Nano foi o SBC que apresentou melhor desempenho ao instanciar cada CNS consumindo menos tempo, seguida pelo Raspberry Pi 4, Raspberry Pi 3 e por último

o Dragonboard, o que pode justificar seu uso de memória fora do padrão esperado.

Considerando-se o estado de repouso, ou seja, o momento em que cada SBC não está suportando CNS, os valores iniciais de consumo médio de memória são conforme abaixo:

- Dragonboard 410c: 24% (240 MB);
- Raspberry Pi 3: 26% (260 MB);
- Raspberry Pi 4: 18% (720 MB);
- Jetson Nano: 31% (1240 MB).

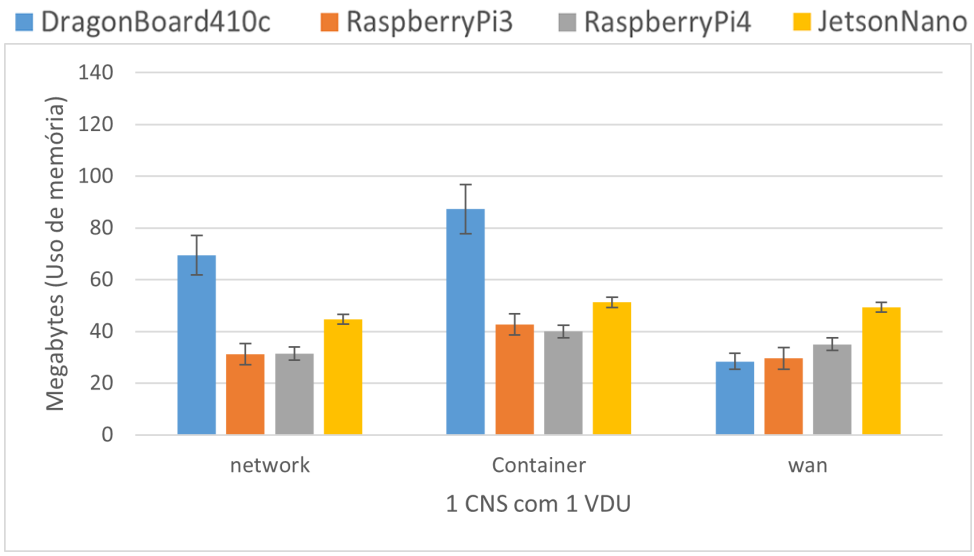

Figura 16 – Consumo médio de memória durante a criação de 1 CNS com 1 VDU por SBC.

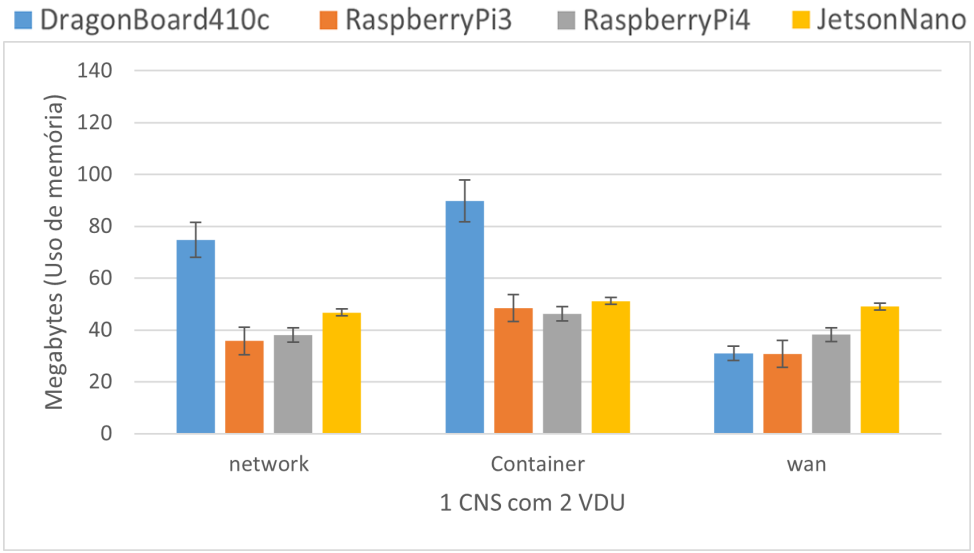

Figura 17 – Consumo médio de memória durante a criação de 1 CNS com 2 VDU por SBC.

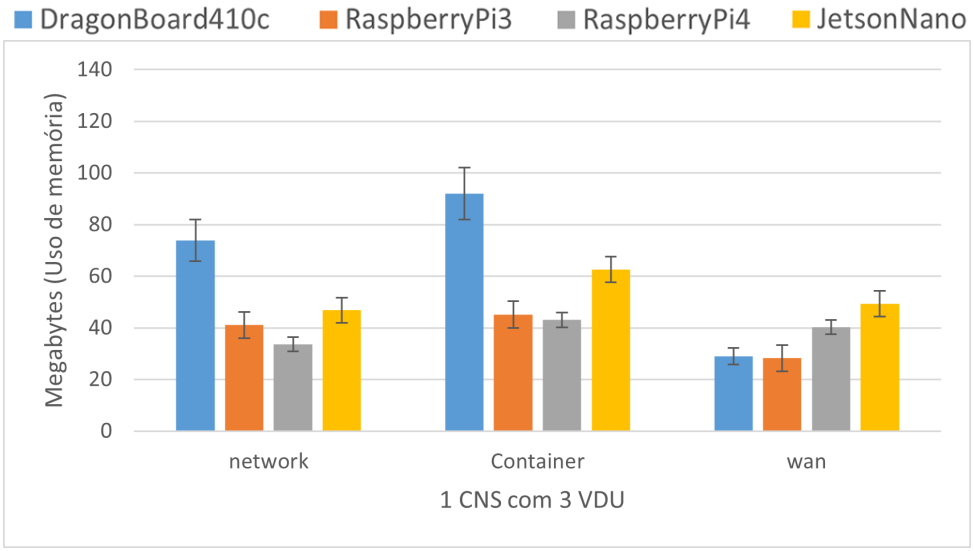

Figura 18 – Consumo médio de memória durante a criação de 1 CNS com 3 VDU por SBC.

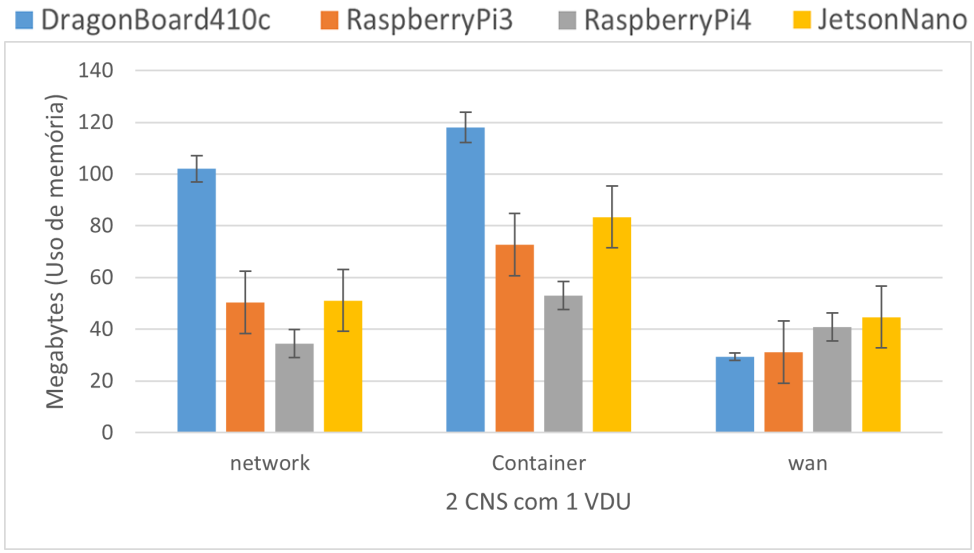

Figura 19 – Consumo médio de memória durante a criação de 2 CNS com 1 VDU por SBC.

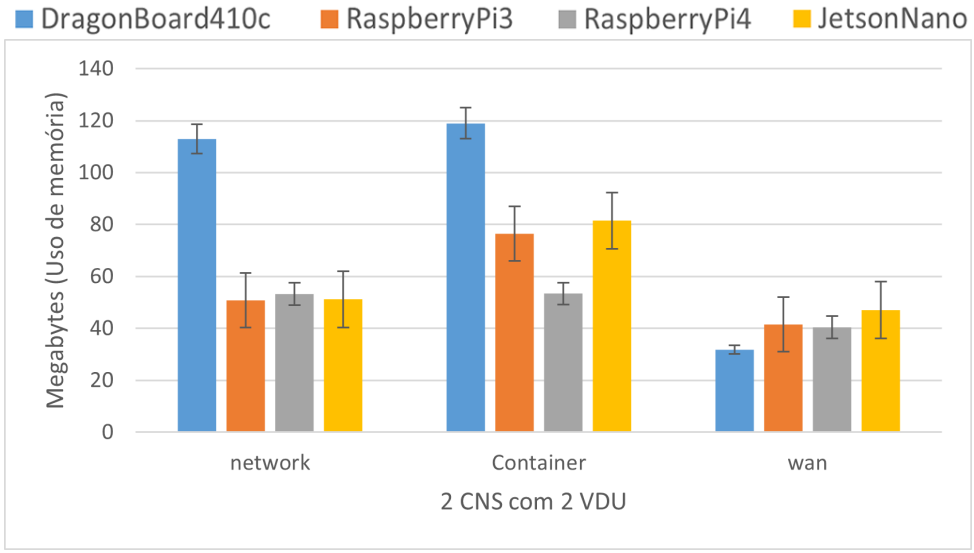

Figura 20 – Consumo médio de memória durante a criação de 2 CNS com 2 VDU por SBC.

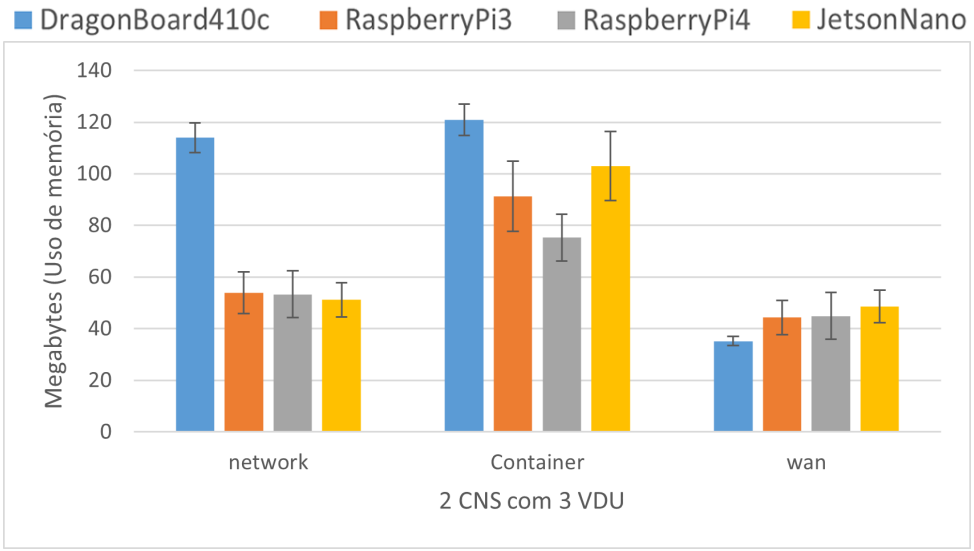

Figura 21 – Consumo médio de memória durante a criação de 2 CNS com 3 VDU por SBC.

3.2.3.3 Teste de temperatura de operação

A temperatura é uma característica limitadora para este tipo de *hardware* devido ao uso de componentes sensíveis. Tanto o Raspberry Pi (PI, 2021) quanto o Dragonboard (96BOARDS, 2021) citam em sua documentação que até 70 graus Celsius é a temperatura ideal de operação. Já o Jetson Nano apresenta em sua documentação oficial o limite de 80 graus Celsius para perfeita operação (NVIDIA, 2021). Dessa forma, nas Figuras 22, 23, 24, 25, 26 e 27 é possível observar que estes limites foram praticamente atingidos, limitando o número de CNS instanciadas e em operação nestes dispositivos.

Considerando-se o estado de repouso, ou seja, o momento em que cada SBC não está suportando CNS, os valores iniciais de temperatura média são conforme abaixo:

- Dragonboard 410c: 45 graus célsius;
- Raspberry Pi 3: 58 graus célsius;
- Raspberry Pi 4: 56 graus célsius;
- Jetson Nano: 40 graus célsius.

Observa-se que o Jetson Nano possui melhor capacidade de operação com alta carga de trabalho e consequente emissão de temperatura, com alta tolerância no consumo deste recurso, no entanto o Raspberry Pi seguido do Dragonboard aparentam possuir um sistema de gestão de temperatura superior aos demais SBC, com pouca variabilidade entre a temperatura inicial de trabalho e a alcançada em estado de *stress*.

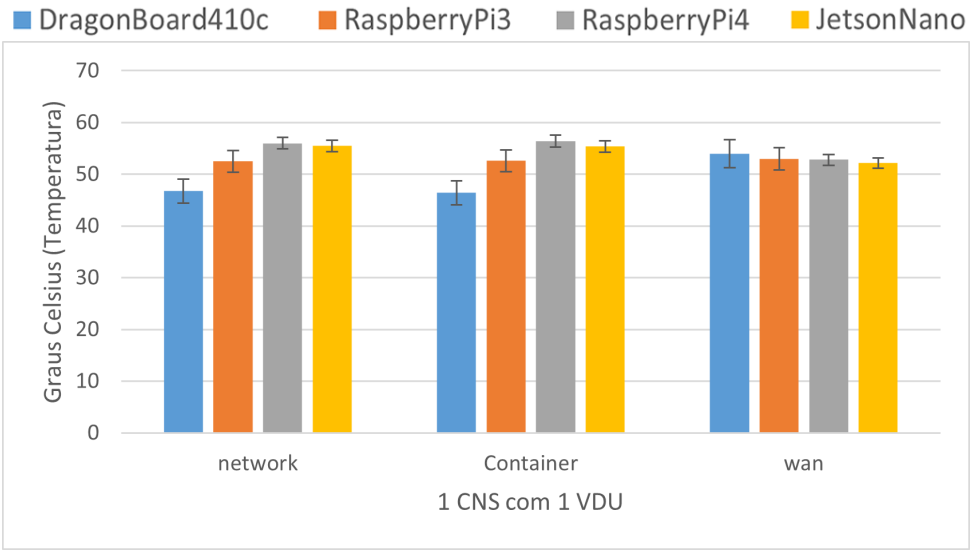

Figura 22 – Temperatura atingida durante a criação de 1 CNS com 1 VDU por SBC.

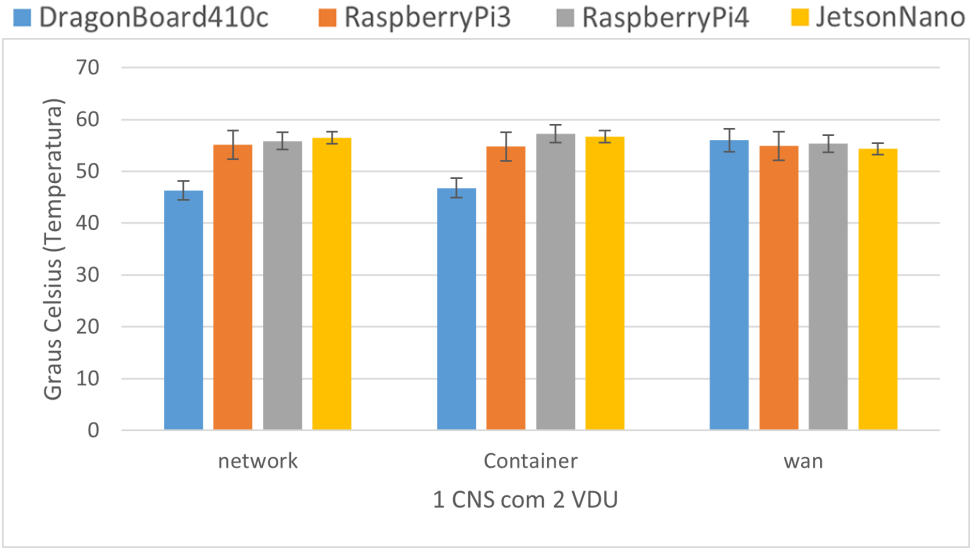

Figura 23 – Temperatura atingida durante a criação de 1 CNS com 2 VDU por SBC.

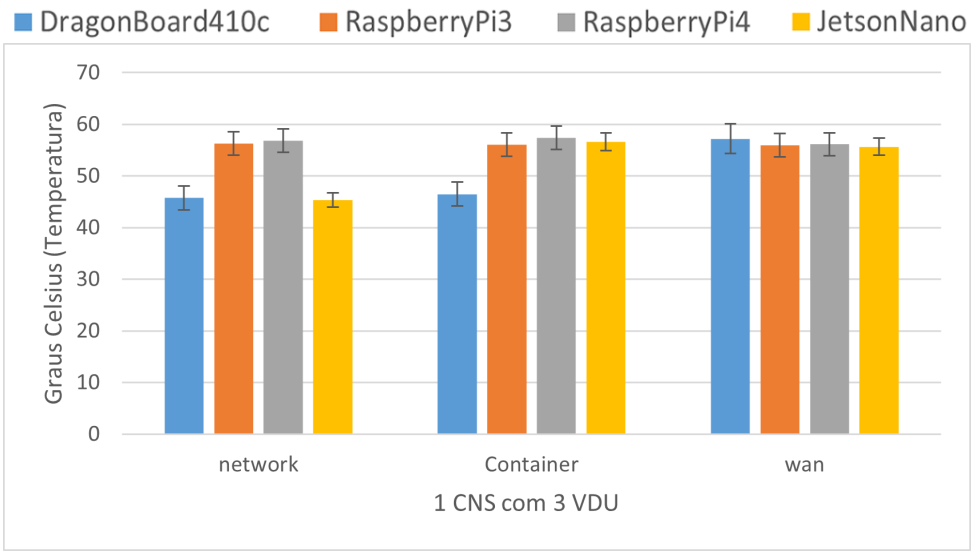

Figura 24 – Temperatura atingida durante a criação de 1 CNS com 3 VDU por SBC.

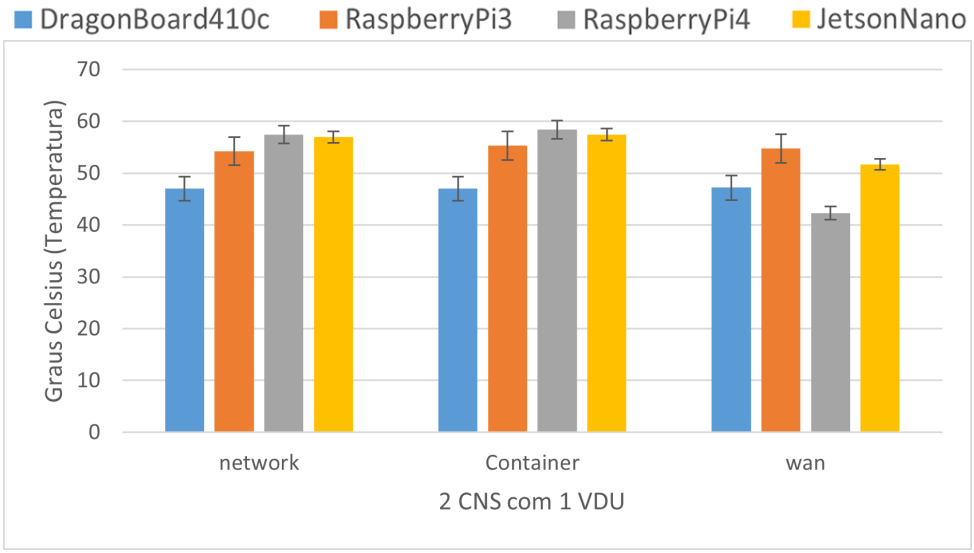

Figura 25 – Temperatura atingida durante a criação de 2 CNS com 1 VDU por SBC.

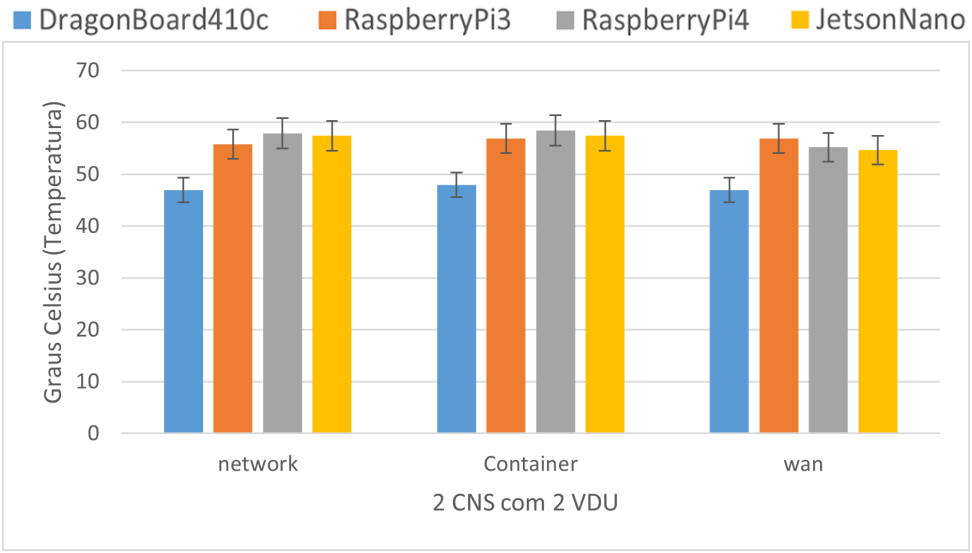

Figura 26 – Temperatura atingida durante a criação de 2 CNS com 2 VDU por SBC.

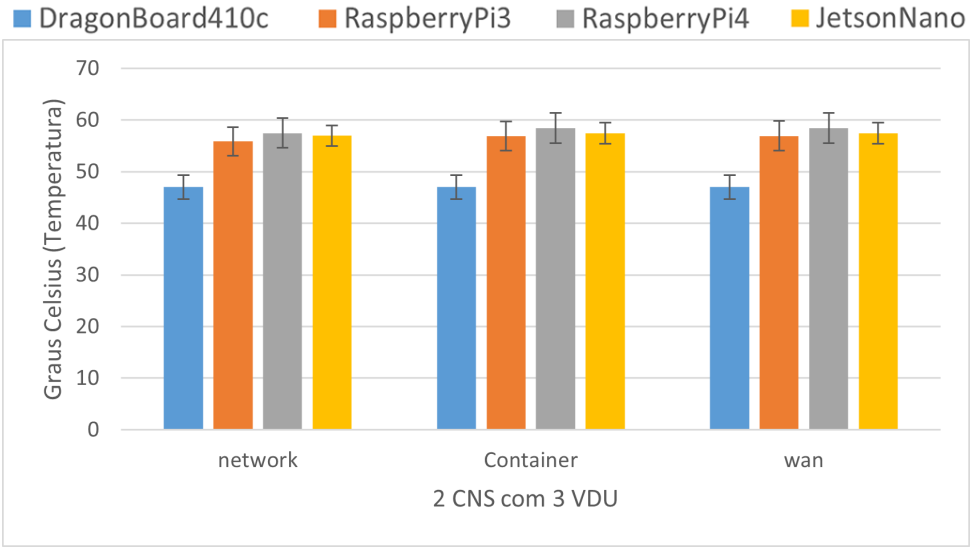

Figura 27 – Temperatura atingida durante a criação de 2 CNS com 3 VDU por SBC.

3.2.3.4 Tempo de instanciação de CNS em cada SBC

O tempo médio de instanciação de 1 CNS em cada SBC se deu conforme a Figura 28 na qual é possível notar a resposta rápida do Jetson Nano, o que justifica seu consumo maior se comparado ao SBC mais próximo em configuração de recursos, o Raspberry Pi 4. Deste modo, o Jetson Nano se mostra como boa opção para para uso em contextos que requisitam instanciação ou recuperação rápida de elementos virtualizados na borda da rede. Já o *hardware* inferior dos demais SBC refletiram negativamente em seus tempos.

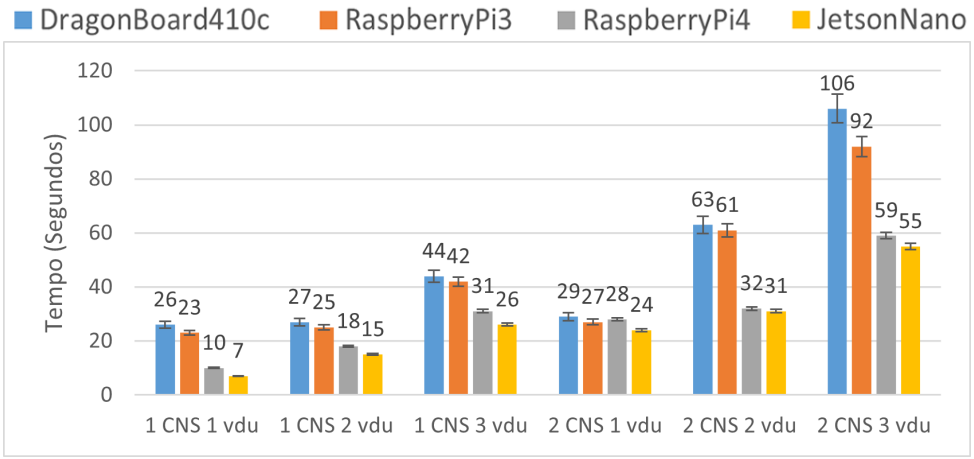

Figura 28 – Tempo médio de instanciação da CNS por SBC.

3.2.3.5 Hardware mínimo para suporte à CNS

Baseado nos resultados e consumo de recursos básicos, como CPU, memória e temperatura atingida durante operação foi confirmado o *hardware* ARM mínimo para operação de múltiplas CNS como sendo dispositivos com as características próximas aos SBC Dragonboard ou Raspberry Pi 3, ou seja, processador com 1,2 GHz e 1 GB de memória RAM. É recomendado também o uso de um dissipador de calor, haja visto a temperatura elevada atingida durante a operação deste tipo de *hardware* e que pode comprometer o seu funcionamento em ambientes com pouca ventilação natural.

Observando-se os problemas encontrados devido à falta de recursos é possível afirmar que dispositivos com características inferiores às citadas acima como mínimo recomendável não suportariam a CNS com qualidade mínima de serviço em ambiente com múltiplas *slices*.

3.2.3.6 Implantação do Serviço

Por fim, cabe destacar que a implantação do serviço no NECOS ocorre através do envio de um arquivo YAML que descreve o serviço a ser implantado. Tal implantação é realizada pelo subsistema IMA, que após receber a descrição citada consulta o *Slice Database* para entender a topologia da CNS criada e implantar o serviço nas *slice parts* corretas.

Neste trabalho, o IMA foi utilizado para implantar um serviço de monitoramento ambiental que consistia em instalar parte do serviço na borda e parte na *core cloud* com o objetivo de capturar dados de sensores presentes nos SBC. O serviço foi composto por um servidor Mosquitto MQTT implantado no *core* e outra aplicação Python implantada na borda que publicava dados no *core* utilizando o protocolo MQTT.

O objetivo do serviço foi capturar temperatura, umidade e altura do sensor em

relação ao chão simulando dispositivos IoT fixados em troncos de árvores e localizados em florestas para pesquisa ambiental com o intuito de executar medição automatizada de árvores e estudos sobre mudanças climáticas em micro regiões.

Este é um cenário hipotético que poderia se beneficiar com orquestração automatizada de dispositivos IoT e oferta de sensores sob demanda em locais de difícil acesso para implantação e manutenção de dispositivos de medição como os citados.

Conclusão

Neste trabalho foi apresentada uma proposta de um sistema minimalista utilizando os componentes do projeto NECOS para fatiamento de recursos e orquestração de dispositivos IoT operando na borda da rede. Os componentes NECOS foram avaliados e implementados para o suporte à criação de CNS em ambiente de borda com recursos restritos utilizando virtualização leve através de *containers*.

Para avaliação, implementou-se uma prova de conceito para validar as novas funcionalidades utilizando *hardwares* reais. O desempenho durante a instanciação de uma ou mais CNS foi avaliado através da captura e análise do consumo de CPU, consumo de memória, tempo de instanciação e temperatura atingida durante este processo e os resultados obtidos confirmaram a hipótese de suporte a múltiplas *slices* nestes dispositivos, no entanto verificou-se limitações no número de *slices* instanciadas em um mesmo *hardware* devido a restrições de recursos em dispositivos IoT. Desta forma a solução apresentada pode atender soluções que demandem *hardware* exclusivo, mas de forma limitada em casos em que o *hardware* precise ser compartilhado entre *tenants*.

Como o suporte a múltiplas *slices* nestes dispositivos foi confirmado com certas limitações de carga, pois a instanciação de *slices* depende de virtualização, mesmo leve, além de mecanismos que garantam isolamento e acabam demandando mais recursos do *hardware*, nós acreditamos que isso possa ser resolvido, ou talvez minimizado, através de novas formas de virtualização, ainda não compatíveis com SBC, como *unikernel* e FaaS (*Function as a Service*) (BALLA; MALIOSZ; SIMON, 2020).

Por fim, a partir dos dados apresentados é possível confirmar o suporte destes SBC à CNS. No entanto, o *hardware* limitado não permite a abordagem tradicional do NECOS que prevê dispositivos sem tais restrições e com conexão constante, como em *data centers*. Verifica-se também que a estrutura minimalista destes *hardwares* IoT os torna suscetíveis a fatores já controlados em outros ambientes, como impacto da temperatura ambiente e de operação do dispositivo nas aplicações que este comporta.

Trabalhos Futuros

As análises realizadas neste trabalho servem como fundamento básico para realização de mais estudos em dispositivos SBC. Usamos uma solução de virtualização leve (microsserviços), mas mesmo assim a instanciação de múltiplas *slices* ficou limitada, dada a necessidade de isolamento requerida pela natureza conceitual das *slices*. Neste sentido, futuras investigações devem avançar na direção de outras soluções mais leves tais como

unikernel e FaaS.

Este trabalho não analisou o desempenho dos *hardwares* na execução dos serviços na *slice*, porém acreditamos ser importante um estudo mais profundo com o intuito de entender o impacto dos serviços IoT mais comumente utilizados em cenários como cidades inteligentes, indústria e saúde, por exemplo.

Já do ponto de vista da rede do sistema distribuído, entendemos que a extensão deste trabalho cobrindo NFV e seus *hardwares* de suporte tornaria o estudo mais completo acerca de um sistema real IoT.

Por fim, recomendamos a realização de testes de estresse nos SBC Raspberry Pi 4 e Jetson Nano com o intuito de verificar o maior número de CNS que estes comportam porque entendemos como limitação deste trabalho a finalização dos testes no momento que os SBC de menor capacidade não suportaram a carga de trabalho e perderam comunicação.

Submissões de artigos

1. Slice como Serviço em Edge Computing e Dispositivos IoT com Restrição de Recursos. SBRC 2021, Simpósio Brasileiro de Redes de Computadores.

2. Slice como Serviço em Edge Computing e Dispositivos IoT com Restrição de Recursos. XXVI WGRS, 2021, Workshop de Gerência e Operação de Redes e Serviços.

Agradecimentos: NECOS (Novel Enablers for Cloud Slicing). 4a Chamada Brasil/União Européia; Sintesoft Tecnologia Ltda.

Referências

96BOARDS. *DragonBoard 410c*. 2021. Https://www.96boards.org/product/dragonboard410c/. Accessed: 2021-02-22. Citado na página 71.

ALAM, M. et al. Orchestration of microservices for iot using docker and edge computing. *IEEE Communications Magazine*, IEEE, v. 56, n. 9, p. 118–123, 2018. Citado 2 vezes nas páginas 39 e 48.

ALLIANCE, N. Ngmn 5g white paper. *Next generation mobile Networks, white paper*, p. 1–125, 2015. Citado 2 vezes nas páginas 25 e 31.

ALMEIDA, L. C. de; JR, P. D. M.; VERDI, F. L. Cloud network slicing: A systematic mapping study from scientific publications. *arXiv preprint arXiv:2004.13675*, 2020. Citado 3 vezes nas páginas 17, 32 e 33.

ARM, I. *Architecting a Smarter World – Arm*. 2020. Https://www.arm.com. Accessed: 2020-04-16. Citado na página 57.

BALENA, I. *balenaOS - Run Docker containers on embedded IoT devices*. 2020. Https://www.balena.io/os/. Accessed: 2020-04-16. Citado na página 57.

BALLA, D.; MALIOSZ, M.; SIMON, C. Open source faas performance aspects. In: *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*. [S.l.: s.n.], 2020. p. 358–364. Citado na página 77.

BERNSTEIN, D. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, IEEE, v. 1, n. 3, p. 81–84, 2014. Citado na página 47.

CHAKRABORTI, A. et al. Using icn slicing framework to build an iot edge network. In: *Proceedings of the 5th ACM Conference on Information-Centric Networking*. [S.l.: s.n.], 2018. p. 214–215. Citado na página 39.

CLAYMAN, S. D3. 1: Necos system architecture and platform specification. v1. *NECOS Project Deliverable*, v. 10, 2018. Citado 5 vezes nas páginas 17, 31, 36, 38 e 41.

CLAYMAN, S. et al. The necos approach to end-to-end cloud-network slicing as a service. *IEEE Communications Magazine*, v. 59, n. 3, p. 91–97, 2021. Citado 6 vezes nas páginas 25, 26, 32, 33, 34 e 35.

CYTRIX. *Xen - System requirements*. 2021. Https://docs.citrix.com/en-us/xenserver/7-1/system-requirements.html. Accessed: 2021-06-05. Citado na página 58.

DELL. *Dell Edge gateways IoT*. 2021. Https://www.dell.com/pt-br/work/shop/gateways-e-computaAccessed: 2021-06-24. Citado na página 49.

DOCKER. *Docker Documentation*. 2021. Https://docs.docker.com/config/containers/resource_constraints. Accessed: 2021-02-22. Citado na página 57.

DOCKER. *Swarm Mode Overview*. 2021. Https://docs.docker.com/engine/swarm/. Accessed: 2021-02-22. Citado na página 39.

DONNO, M. D.; TANGE, K.; DRAGONI, N. Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. *Ieee Access*, IEEE, v. 7, p. 150936–150948, 2019. Citado 2 vezes nas páginas 25 e 30.

ETSI, N. F. V. Network functions virtualisation (nfv). *Management and Orchestration*, v. 1, p. V1, 2014. Citado 2 vezes nas páginas 25 e 31.

FARINACCI, D. et al. *Generic routing encapsulation (GRE)*. [S.l.], 2000. Citado na página 51.

FERNANDEZ, J.-M.; VIDAL, I.; VALERA, F. Enabling the orchestration of iot slices through edge and cloud microservice platforms. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 19, n. 13, p. 2980, 2019. Citado na página 40.

FLINCK, H. et al. Network slicing management and orchestration. *Internet Engineering Task Force, Tech. Rep*, 2017. Citado 2 vezes nas páginas 25 e 31.

FOUNDATION, E. *Eclipse Mosquitto*. 2021. Http://mosquitto.org/. Accessed: 2021-02-22. Citado na página 59.

FOUNDATION, T. L. *Kubernetes*. 2020. Http://www.kubernetes.io. Accessed: 2020-04-16. Citado na página 40.

JOHNSTON, S. J. et al. Applicability of commodity, low cost, single board computers for internet of things devices. In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2016. p. 141–146. Citado na página 61.

KUKLIŃSKI, S. et al. A reference architecture for network slicing. In: IEEE. *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. [S.l.], 2018. p. 217–221. Citado na página 31.

Kurek, T. Unikernel network functions: A journey beyond the containers. *IEEE Communications Magazine*, v. 57, n. 12, p. 15–19, December 2019. ISSN 1558-1896. Citado na página 47.

MADHAVAPEDDY, A. et al. Unikernels: Library operating systems for the cloud. *ACM SIGARCH Computer Architecture News*, ACM New York, NY, USA, v. 41, n. 1, p. 461–472, 2013. Citado 2 vezes nas páginas 57 e 58.

MARINONI, M. et al. Allocation and control of computing resources for real-time virtual network functions. In: *Proc. of the international Symposium on Advances in Software Defined Networking and Network Function Virtualization (SoftNetworking 2018)*. [S.l.: s.n.], 2018. p. 52–57. Citado na página 32.

NETDATA, I. *GitHub - netdata/netdata: Real-time performance monitoring, done right! https://www.netdata.cloud*. 2021. Https://github.com/netdata/netdata. Accessed: 2021-02-02. Citado na página 62.

NVIDIA. *NVIDIA Jetson Nano Thermal Guide*. 2021. Https://developer.nvidia.com/embedded/dlc/jetson-nano-thermal-design-guide-1.3. Accessed: 2021-02-22. Citado na página 71.

OPENVSWITCH. *Open vSwitch*. 2021. Https://www.openvswitch.org/. Accessed: 2021-02-02. Citado 2 vezes nas páginas 57 e 63.

OPENVSWITCH. *Project — Open vSwitch 2.15.90 documentation*. 2021. Https://docs.openvswitch.org/en/latest. Accessed: 2021-02-02. Citado na página 57.

ORDONEZ-LUCENA, J. et al. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *IEEE Communications Magazine*, IEEE, v. 55, n. 5, p. 80–87, 2017. Citado na página 31.

OSV. *OSv - the operating system designed for the cloud*. 2021. Http://osv.io/. Accessed: 2021-02-02. Citado na página 58.

PALLETS. *Flask Documentation*. 2021. Https://flask.palletsprojects.com/en/2.0.x/. Accessed: 2021-07-08. Citado na página 56.

PARAMIKO. *Welcome to Paramiko! Paramiko documentation*. 2021. Https://www.paramiko.org/. Accessed: 2021-06-05. Citado na página 57.

PASSI, A.; BATRA, D. Future of internet of things (iot) in 5g wireless networks. *International Journal of Engineering and Technology*, v. 7, n. 1.5, 2017. Citado na página 25.

PI, R. *Raspberry Pi Documentation*. 2021. Https://www.raspberrypi.org/documentation/hardware. Accessed: 2021-02-22. Citado na página 71.

PROJECT, K. *FAQ - KVM*. 2021. Https://www.linux-kvm.org/page/FAQ. Accessed: 2021-06-05. Citado na página 58.

RIBEIRO, J. B. et al. Slices como serviço sobre um centro de dados itinerante aplicado ao cenário amazônico. In: SBC. *Anais do I Workshop de Teoria, Tecnologias e Aplicações de Slicing para Infraestruturas Softwarizadas*. [S.l.], 2019. p. 15–27. Citado na página 41.

ROCHA, A. L. B. et al. Uma proposta de arquitetura para o monitoramento multidomínio de cloud network slices. In: SBC. *Anais do I Workshop de Teoria, Tecnologias e Aplicações de Slicing para Infraestruturas Softwarizadas*. [S.l.], 2019. p. 42–55. Citado na página 32.

RODRIGUEZ, A. Restful web services: The basics. *IBM developerWorks*, v. 33, p. 18, 2008. Citado na página 56.

SERVICES, A. W. *AWS CLI - Interface de linha de comando*. 2021. Https://aws.amazon.com/pt/cli/. Accessed: 2021-07-08. Citado na página 49.

SILVA, F. S. D. et al. Necos project: Towards lightweight slicing of cloud federated infrastructures. In: IEEE. *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. [S.l.], 2018. p. 406–414. Citado na página 40.

SPRECHER, N. Internet engineering task force h. flinck internet-draft c. sartori intended status: Informational a. andrianov expires: January 4, 2018 c. mannweiler. *Network*, 2017. Citado 2 vezes nas páginas 25 e 31.

SQLITE. *SQLite Documentation*. 2021. Https://www.sqlite.org/docs.html. Accessed: 2021-06-04. Citado na página 57.

STEEN, M. v.; TANENBAUM, A. S. *Distributed systems: principles and paradigms*. [S.l.]: Prentice Hall, 2007. Citado na página 31.

TR, O. 526. *Applying SDN Architecture to 5G Slicing*, S. l.]: 5G PPP, 2016. Citado 2 vezes nas páginas 25 e 31.

UNIKRAFT. *Home - Unikraft*. 2021. Http://unikraft.org/. Accessed: 2021-02-02. Citado na página 58.

XEN; FOUNDATION, L. *MirageOS*. 2020. Https://mirage.io/. Accessed: 2020-04-16. Citado na página 57.

XHAFA, F.; KILIC, B.; KRAUSE, P. Evaluation of iot stream processing at edge computing layer for semantic data enrichment. *Future Generation Computer Systems*, Elsevier, v. 105, p. 730–736, 2020. Citado 5 vezes nas páginas 25, 26, 29, 33 e 55.

Y, I.-T. R. I.-T. *3001, Future Networks: Objectives and Design Goals*. [S.l.]: ITU Geneva, Switzerland:, 2011. Citado 2 vezes nas páginas 25 e 31.

APÊNDICE A – YAML

YAML completo de especificação da *slice*:

```
slices:

   sliced:

      id: IoT1_sliced

      name: IoT1_sliced

      short-name: IoT1_sliced

      description: Slicing for IoT

      vendor: MTC

      version: '1.0'

      created-slice: False


      # logo of slice for visualization purposes

      logo: iot1.png



      slice-constraints:

         geographic: [BRAZIL, EUROPE]

         dc-slice-parts: 1 # Could be a constraint on values.

         edge-slice-parts: 4

         net-slice-parts: 4
```

```
slice-requirements:
   elasticity: false
   reliability:
      description: reliability level
      enabled: true
      value: none  # {path-backup, logical-backup, physical-backup\}


slice-lifecycle:
   description: lifecycle status
   status: construction  # {modification, activation, deletion\}


cost:
   dc-model:
      model: COST_PER_PHYSICAL_MACHINE_PER_DAY
      value-euros: {lower_than_equal: 10}
   net-model:
      model: COST_PER_LINK_PER_DAY
      value-euros: {lower_than_equal: 50}


slice-timeframe:
   service-start-time: {100919: 10 pm}
```

```
    service-stop-time: {101019: 10 pm}



slice-parts:

    - dc-slice-part:
        name: core-dc
        type: DC
        location: EUROPE.SPAIN
        dc-slice-controller:
            dc-slice-provider: undefined
            ip: undefined
            port: undefined


        monitoring-parameters:
           tool: netdata
           measurements-db-ip: <X>
           measurements-db-port: <X>
           granularity-secs: 10
           type: host
           metrics:
```

```
      - metric:
          name: PERCENT_CPU_UTILIZATION
      - metric:
          name: MEGABYTES_MEMORY_UTILIZATION
      - metric:
          name: TOTAL_BYTES_DISK_IN
      - metric:
          name: TOTAL_BYTES_DISK_OUT
      - metric:
          name: TOTAL_BYTES_NET_RX
      - metric:
          name: TOTAL_BYTES_NET_TX


VIM:
    name: xen-vim
    version: XEN-SERVER
    vim-shared: false
    vim-federated: false
    vim-ref: undefined
    host-count: 1
    vdus:
```

```
- vdu:

    id: core-vm

    name: core-vm

    description: load balancer and content services

    instance-count: 1

    hosting: SHARED

    vdu-image: 'core-vm-template'



    # does not require a flavor

    epa-attributes:

       host-epa:

           cpu-architecture: PREFER_X86_64

           cpu-number: 4

           storage-gb: 30

           memory-mb: 8192

           os-properties:

               # host Operating System image properties

               architecture: x86_64

               type: linux

               distribution: ubuntu

               version: 16.04
```

```
                    host-image: indefined


- dc-slice-part:

    name: edge-slice-brazil-1

    type: EDGE

    location: AMERICA.BRAZIL

    dc-slice-controller:

        dc-slice-provider: undefined

        ip: undefined

        port: undefined


    monitoring-parameters:

       tool: netdata

       measurements-db-ip: <X>

       measurements-db-port: <X>

       granularity-secs: 10

       type: host

       metrics:

          - metric:

               name: PERCENT_CPU_UTILIZATION

          - metric:
```

```
            name: MEGABYTES_MEMORY_UTILIZATION
        - metric:
            name: TOTAL_BYTES_DISK_IN
        - metric:
            name: TOTAL_BYTES_DISK_OUT
        - metric:
            name: TOTAL_BYTES_NET_RX
        - metric:
            name: TOTAL_BYTES_NET_TX


VIM:
    name: docker-vim
    version: undefined
    vim-shared: false
    vim-federated: false
    vim-ref: undefined
    host-count: 1
    vdus:
        - vdu:
            id: docker-master
            name: docker-master
```

```
                description: Master node for IoT deployment
                instance-count: 1
                hosting: SHARED
                vdu-image: 'debian'

                # does not require a flavor
                epa-attributes:
                  host-epa:
                      cpu-architecture: PREFER_ARM_64
                      cpu-number: 8
                      storage-gb: 10
                      memory-mb: 4096
                      os-properties:
                          # host Operating System image properties
                          architecture: arm_64
                          type: linux
                          distribution: undefined
                          version: undefined
                      host-image: undefined

    - dc-slice-part:
```

```
name: edge-slice-brazil-2

type: EDGE

location: AMERICA.BRAZIL

dc-slice-controller:

    dc-slice-provider: undefined

    ip: undefined

    port: undefined



monitoring-parameters:

   tool: netdata

   measurements-db-ip: <X>

   measurements-db-port: <X>

   granularity-secs: 10

   type: host

   metrics:

      - metric:

           name: PERCENT_CPU_UTILIZATION

      - metric:

           name: MEGABYTES_MEMORY_UTILIZATION

      - metric:

           name: TOTAL_BYTES_DISK_IN
```

```
      - metric:
          name: TOTAL_BYTES_DISK_OUT
      - metric:
          name: TOTAL_BYTES_NET_RX
      - metric:
          name: TOTAL_BYTES_NET_TX


VIM:
    name: docker-vim
    version: undefined
    vim-shared: false
    vim-federated: false
    vim-ref: undefined
    host-count: 1
    vdus:
      - vdu:
          id: docker-master
          name: docker-master
          description: Master node for IoT deployment
          instance-count: 1
          hosting: SHARED
```

```
                vdu-image: 'debian'


                # does not require a flavor
                epa-attributes:
                  host-epa:
                      cpu-architecture: PREFER_ARM_64
                      cpu-number: 8
                      storage-gb: 10
                      memory-mb: 4096
                      os-properties:
                          # host Operating System image properties
                          architecture: arm_64
                          type: linux
                          distribution: undefined
                          version: undefined
                      host-image: undefined


- dc-slice-part:
    name: edge-slice-brazil-3
    type: EDGE
    location: AMERICA.BRAZIL
```

```
dc-slice-controller:

    dc-slice-provider: undefined

    ip: undefined

    port: undefined



monitoring-parameters:

   tool: netdata

   measurements-db-ip: <X>

   measurements-db-port: <X>

   granularity-secs: 10

   type: host

   metrics:

      - metric:

           name: PERCENT_CPU_UTILIZATION

      - metric:

           name: MEGABYTES_MEMORY_UTILIZATION

      - metric:

           name: TOTAL_BYTES_DISK_IN

      - metric:

           name: TOTAL_BYTES_DISK_OUT

      - metric:
```

```
            name: TOTAL_BYTES_NET_RX
        - metric:
            name: TOTAL_BYTES_NET_TX


VIM:
    name: docker-vim
    version: undefined
    vim-shared: false
    vim-federated: false
    vim-ref: undefined
    host-count: 1
    vdus:
        - vdu:
            id: docker-master
            name: docker-master
            description: Master node for IoT deployment
            instance-count: 1
            hosting: SHARED
            vdu-image: 'debian'

            # does not require a flavor
```

```
                epa-attributes:
                  host-epa:
                      cpu-architecture: PREFER_ARM_64
                      cpu-number: 8
                      storage-gb: 10
                      memory-mb: 1024
                      os-properties:
                          # host Operating System image properties
                          architecture: arm_64
                          type: linux
                          distribution: undefined
                          version: undefined
                      host-image: undefined


- dc-slice-part:
    name: edge-slice-brazil-4
    type: EDGE
    location: AMERICA.BRAZIL
    dc-slice-controller:
        dc-slice-provider: undefined
        ip: undefined
```

```
    port: undefined


monitoring-parameters:
   tool: netdata
   measurements-db-ip: <X>
   measurements-db-port: <X>
   granularity-secs: 10
   type: host
   metrics:
      - metric:
           name: PERCENT_CPU_UTILIZATION
      - metric:
           name: MEGABYTES_MEMORY_UTILIZATION
      - metric:
           name: TOTAL_BYTES_DISK_IN
      - metric:
           name: TOTAL_BYTES_DISK_OUT
      - metric:
           name: TOTAL_BYTES_NET_RX
      - metric:
           name: TOTAL_BYTES_NET_TX
```

```
VIM:
    name: docker-vim
    version: undefined
    vim-shared: false
    vim-federated: false
    vim-ref: undefined
    host-count: 1
    vdus:
        - vdu:
            id: docker-master
            name: docker-master
            description: Master node for IoT deployment
            instance-count: 1
            hosting: SHARED
            vdu-image: 'debian'

            # does not require a flavor
            epa-attributes:
               host-epa:
                   cpu-architecture: PREFER_ARM_64
```

```
                    cpu-number: 8
                    storage-gb: 10
                    memory-mb: 1024
                    os-properties:
                        # host Operating System image properties
                        architecture: arm_64
                        type: linux
                        distribution: undefined
                        version: undefined
                    host-image: undefined

- net-slice-part:
    name: edge-brazil-1-to-dc-core
    wan-slice-controller:
        wan-slice-provider: undefined
        ip: undefined
        port: undefined undefined

    WIM:
       name: GRE
       version: undefined
```

```
        wim-shared: true
        wim-federated: false
        wim-ref: undefined

    links:
        - dc-part1: core-dc
        - dc-part2: edge-slice-brazil-1
        - requirements:
            bandwidth-GB: 1

    link-ends:
        link-end1-ip: undefined
        link-end2-ip: undefined

- net-slice-part:
    name: edge-brazil-2-to-dc-core
    wan-slice-controller:
        wan-slice-provider: undefined
        ip: undefined
        port: undefined undefined
```

```
    WIM:
       name: GRE
       version: undefined
       wim-shared: true
       wim-federated: false
       wim-ref: undefined

    links:
       - dc-part1: core-dc
       - dc-part2: edge-slice-brazil-2
       - requirements:
            bandwidth-GB: 1

    link-ends:
       link-end1-ip: undefined
       link-end2-ip: undefined

- net-slice-part:
    name: edge-brazil-3-to-dc-core
    wan-slice-controller:
        wan-slice-provider: undefined
```

```
            ip: undefined
            port: undefined undefined

        WIM:
           name: GRE
           version: undefined
           wim-shared: true
           wim-federated: false
           wim-ref: undefined

        links:
           - dc-part1: core-dc
           - dc-part2: edge-slice-brazil-3
           - requirements:
               bandwidth-GB: 1

        link-ends:
           link-end1-ip: undefined
           link-end2-ip: undefined

- net-slice-part:
```

```
name: edge-brazil-4-to-dc-core

wan-slice-controller:

    wan-slice-provider: undefined

    ip: undefined

    port: undefined undefined



WIM:

   name: GRE

   version: undefined

   wim-shared: true

   wim-federated: false

   wim-ref: undefined



links:

   - dc-part1: core-dc

   - dc-part2: edge-slice-brazil-4

   - requirements:

        bandwidth-GB: 1



link-ends:

   link-end1-ip: undefined
```

```
            link-end2-ip: undefined
```

YAML completo de especificação do serviço:

```
slices:
  sliced:
    id: IoTService_sliced
    slice-parts:
      - dc-slice-part:
        name: core-dc
        vdus:
          - vdu:
            name: core-vm
            VIM: VM
            namespace:
            commands:
              - git clone
                https://github.com/rodrigoferrazazevedo/IOT-MQTT-Python.git
              - export DEBIAN_FRONTEND=noninteractive; sudo apt-get update;
                sudo apt-get install mosquitto; sudo apt install -y
                python-pip
              - sudo pip install -r IOT-MQTT-Python/requirements.txt
              - sudo python IOT-MQTT-Python/subscribe-cloudmqtt.py
      - dc-slice-part:
        name: edge-slice-brazil-1
        vdus:
          - vdu:
            name: docker-master
            VIM: Docker
            namespace:
            commands:
              - git clone
                https://github.com/rodrigoferrazazevedo/IOT-MQTT-Python.git
              - export DEBIAN_FRONTEND=noninteractive; sudo apt-get update;
                sudo apt install -y python-pip
              - sudo pip install -r IOT-MQTT-Python/requirements.txt
              - sudo python IOT-MQTT-Python/publish-cloudmqtt.py
```