UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Vinícius Marques Rodrigues

Avaliação de Escalabilidade e Desempenho do Plano de Controle em Redes Definidas por Software com OVN

SÃO CARLOS - SP

2025

Vinícius Marques Rodrigues

Avaliação de Escalabilidade e Desempenho do Plano de Controle em Redes Definidas por Software com OVN

Trabalho de conclusão de curso apresentado ao curso de Engenharia de Computação da Universidade Federal de São Carlos, para obtenção do título de bacharel em engenharia de computação.

Orientador: Prof. Dr. Hélio Crestana Guardia

SÃO CARLOS - SP

2025

Resumo:

A adoção de Redes Definidas por Software (SDN) em ambientes de nuvem de larga escala impõe desafios significativos ao plano de controle. Enquanto o plano de dados tem recebido atenção substancial através de tecnologias como DPDK, o desempenho do plano de controle — especificamente a latência de convergência entre a intenção lógica e a configuração física — tornou-se um novo gargalo crítico. No ecossistema OpenStack, o Open Virtual Network (OVN) atua como o componente central de orquestração, traduzindo configurações lógicas (*Northbound*) em fluxos OpenFlow (*Southbound*). Em versões anteriores, o *daemon ovn-northd* realizava cálculos completos do grafo de rede para cada alteração, resultando em uso excessivo de CPU e alta latência em cenários de alta densidade de portas. Através da injeção sintética de recursos de rede (Switches Lógicos e Portas), quantificamos o tempo de convergência e o consumo de recursos computacionais. Os experimentos comparam o desempenho entre versões distintas avaliando o impacto do mecanismo de Processamento Incremental (*Incremental Processing*). Os resultados evidenciam como a otimização algorítmica no plano de controle reduz o tempo de provisionamento de redes em ordens de magnitude, viabilizando a escalabilidade para milhares de instâncias virtuais.

Palavras-chave: Redes definidas por Software, Open Virtual Network, Plano de Controle, Escalabilidade, Processamento Incremental, OpenStack.

Abstract

The adoption of Software-Defined Networking (SDN) in large-scale cloud environments presents significant challenges to the control plane. While the data plane has received substantial optimization through technologies such as DPDK, control plane performance—specifically the convergence latency between logical intent and physical realization—has emerged as a new critical bottleneck. Within the OpenStack ecosystem, Open Virtual Network (OVN) serves as the central orchestration component, translating Northbound logical configurations into Southbound OpenFlow flows. In legacy versions, the ovn-northd daemon performed a full recomputation of the network graph for every state change, leading to excessive CPU consumption and high latency in high-density port scenarios. This work utilizes the synthetic injection of network resources (Logical Switches and Ports) to quantify convergence time and computational resource usage. The experiments compare performance across different versions, evaluating the specific impact of the Incremental Processing mechanism. Results demonstrate that algorithmic optimization in the control plane reduces network provisioning time by orders of magnitude, thereby enabling scalability for thousands of virtual instances.

Keywords: Software-Defined Networks, Open Virtual Network, Control Plane, Scalability, Incremental Processing, OpenStack.

Introdução

A computação em nuvem (*Cloud Computing*) consolidou-se como o paradigma dominante para a infraestrutura de TI moderna, permitindo o provisionamento de recursos de computação, armazenamento e rede sob demanda. Para sustentar a elasticidade e a automação exigidas por essas plataformas, a arquitetura de redes evoluiu de modelos estáticos e baseados em hardware para as Redes Definidas por Software (*SDN - Software-Defined Networking*). Nesse contexto, OpenStack destaca-se como a plataforma de código aberto mais utilizada para orquestração de nuvens privadas e públicas, adotando OVN (*Open Virtual Network*) como sua solução padrão para a virtualização de redes.

OVN estende as capacidades do Open vSwitch (OVS)(PFAFF et al., 2015), fornecendo uma abstração de rede lógica (switches e roteadores virtuais) que é distribuída e traduzida em regras de fluxo nos switches virtuais de cada hipervisor. Embora a indústria tenha dedicado esforços significativos na otimização do plano de dados (*data plane*) — utilizando tecnologias como DPDK (*Data Plane Development Kit*) e aceleração via hardware para garantir a vazão de pacotes — um novo desafio emergiu com o crescimento da escala das nuvens: o desempenho do plano de controle (*control plane*).

Em ambientes de hiperescala ou com alta densidade de contêineres e máquinas virtuais, a velocidade com que a rede "converge" — ou seja, o tempo entre a solicitação de uma nova porta e a efetiva programação dos fluxos que permitem o tráfego — tornou-se um indicador crítico de qualidade de serviço (QoS) e eficiência operacional.

1.1 Problematização

O problema central abordado neste trabalho reside na complexidade computacional envolvida na tradução de fluxos lógicos (*logical flows*) para configurações físicas em ambientes SDN de larga escala. Na arquitetura do OVN, o daemon *ovn-northd* é responsável por traduzir a configuração do banco de dados *Northbound* (lógico) para o banco de dados *Southbound* (físico/lógico intermediário).

À medida que o número de portas lógicas, redes e listas de controle de acesso (ACLs) aumenta, o tempo necessário para recalcular os fluxos lógicos pode crescer exponencialmente. Esse fenômeno cria um gargalo no plano de controle: enquanto o hipervisor pode ativar uma máquina virtual em segundos, a rede pode levar um tempo significativamente maior para se tornar operante.

Esse gargalo não afeta apenas o componente responsável por fazer os cálculos. Mesmo após calcular todo o estado da rede, no caso do *northd*, ainda existe a necessidade de criar uma transação com o banco de dados *southbound* para a entrega do estado calculado. Essa transação depende da disponibilidade de escrita no *southbound*, que pode ser comprometida por operações internas de manutenção do consenso RAFT, impactando o *ovn-controller* que aguarda o estado desse banco de dados (MAXIMETS, 2021).

Esse atraso de convergência impacta diretamente a elasticidade da nuvem, afetando operações de *autoscaling*, recuperação de desastres e a experiência do usuário final, que percebe a latência não como um problema de rede, mas como uma falha na disponibilidade do serviço.

1.2 Proposta

Diante deste cenário, este trabalho propõe uma avaliação de desempenho e escalabilidade do plano de controle do OVN integrado ao *OpenStack*. Esta proposta concentra-se na mensuração da **latência de provisionamento** e no **custo de convergência**.

A metodologia consiste na injeção controlada de recursos sintéticos (redes, sub-redes e portas) diretamente através da API do OpenStack Neutron em um ambiente de produção real. Scripts desenvolvidos orquestram a criação massiva de portas lógicas ("*Fake Ports*") que, embora não estejam conectadas a cargas de trabalho ativas, obrigam todo o plano de controle (*Neutron Server*, *Plugin* OVN, *northd* e *OVN Controller*) a realizar o ciclo completo de alocação de endereços, cálculo de fluxos e propagação de estado.

O objetivo é oferecer uma análise empírica sobre o comportamento do mecanismo de tradução de configuração alto-nível em configuração lógica do *ovn-northd* sob estresse.

1.3 Resultados

A partir da execução dos experimentos de escalabilidade utilizando a metodologia proposta, buscou-se obter os seguintes resultados:

1. **Caracterização da curva de latência:** Demonstrar a correlação entre o número total de elementos lógicos no sistema e o tempo necessário para o provisionamento. Espera-se observar um comportamento de degradação progressiva (linear ou exponencial) da latência à medida que o número de elementos lógicos crescem.
2. **Identificação de gargalos de recursos:** Correlacionar os picos de latência de provisionamento com o consumo de recursos (CPU e Memória) dos processos centrais do OVN (especificamente o *ovn-northd* nos nós de controle), comprovando a hipótese de que o cálculo de fluxos lógicos é uma operação intensiva em CPU.
3. **Análise de viabilidade para alta escala:** Concluir sobre os limites operacionais da versão atual do OVN implantada no ambiente, fornecendo dados sobre os benefícios de atualizações para versões com suporte a Processamento Incremental (*Incremental Processing*) em cenários de maior densidade.

2. Fundamentação Teórica

Este capítulo apresenta os conceitos essenciais para a compreensão dos desafios de escalabilidade em redes de ambientes de computação em nuvem. A discussão parte dos princípios de Redes Definidas por Software (SDN), detalha a arquitetura do *Open Virtual Network* (OVN) e seus componentes de banco de dados, e analisa os gargalos computacionais inerentes à tradução de configurações lógicas em fluxos físicos.

2.1 Redes Definidas por Software (SDN) em Nuvens IaaS

A ascensão da computação em nuvem impôs novos desafios e oportunidades para a arquitetura de redes. Provedores de nuvem devem suportar redes virtuais com alto desempenho e um conjunto rico de funcionalidades — como balanceamento de carga, firewalls, VPNs e proteção contra DoS — operando em escala global. Para atender a esses requisitos, o paradigma de Redes Definidas por Software (*Software Defined Network* - SDN) tornou-se a base da virtualização de redes modernas.

SDN viabiliza a elasticidade da nuvem ao desacoplar o plano de controle (*control plane*), responsável pelas decisões de roteamento e política, do plano de dados (*data plane*), responsável pelo encaminhamento efetivo dos pacotes. Em ambientes de nuvem como o OpenStack, essa separação é crítica, pois permite que o orquestrador crie redes virtuais isoladas e aplique políticas de segurança de forma programática, sem a rigidez da intervenção manual em switches físicos.

Segundo Dalton et al. (2018), no contexto da plataforma *Andromeda* do *Google*, um ambiente de virtualização de rede robusto deve atender a requisitos fundamentais que vão além da conectividade básica:

- Isolamento de desempenho e segurança: a rede deve fornecer a ilusão de que as máquinas virtuais (VMs) residem em sua própria rede privada, isoladas das ações de outros locatários (tenants). Isso inclui garantir que o tráfego de um cliente não impacte a latência ou a largura de banda de outro.

- Escalabilidade do plano de controle: o sistema deve suportar o provisionamento rápido de um grande número de hosts virtuais. Redes de larga escala apresentam desafios únicos, exigindo que o plano de controle gerencie tabelas de roteamento massivas e altas taxas de alteração (*churn*) sem degradar a capacidade de resposta.
- Velocidade de funcionalidades (*Feature Velocity*): a capacidade de introduzir novas funcionalidades de rede e melhorias de segurança sem interromper as cargas de trabalho existentes é essencial. Arquiteturas baseadas em software permitem uma evolução rápida, diferentemente de soluções baseadas estritamente em hardware.
- Desempenho próximo ao hardware: idealmente, a camada de virtualização deve oferecer largura de banda e latência indistinguíveis da infraestrutura física subjacente, minimizando o overhead da CPU do hospedeiro.

Esses requisitos evidenciam que o desafio em nuvens IaaS não é apenas mover pacotes, mas orquestrar a complexidade lógica de milhares de redes virtuais simultâneas. OVN (*Open Virtual Network*), objeto de estudo deste trabalho, implementa esses princípios ao distribuir a inteligência de rede para as bordas (os hipervisores), utilizando uma arquitetura de plano de controle hierárquica para tentar mitigar os gargalos de escalabilidade identificados em sistemas centralizados tradicionais.

2.1.1 O Papel do Open vSwitch (OVS)

OVN é construído sobre o alicerce do *Open vSwitch* (*OVS*) (PFAFF et al., 2015), um *switch* virtual multicamada projetado especificamente para as restrições e requisitos de ambientes virtualizados. Diferente de appliances de rede tradicionais, que possuem hardware dedicado para garantir desempenho de linha no pior caso, *OVS* opera em um ambiente de recursos compartilhados, onde a conservação de CPU do hipervisor é prioritária para não impactar as cargas de trabalho dos usuários.

A arquitetura do OVS, conforme detalhado por Pfaff et al. (2015), é dividida em dois componentes principais que operam em espaços distintos:

- O *Daemon* de usuário (*ovs-vswitchd*): responsável pela lógica complexa de encaminhamento e pela comunicação com controladores externos (como OVN) através do protocolo *OpenFlow*. Este componente implementa o "caminho lento" (*slow path*), tomando decisões sobre o primeiro pacote de novos fluxos e traduzindo regras de alto nível em ações de datapath.
- O módulo de kernel (*Datapath*): projetado para ser simples e rápido, atua como um cache de microfluxos e megafluxos. Ele processa a grande maioria dos pacotes diretamente no espaço do núcleo, sem a necessidade de intervenção do espaço de usuário.

A eficiência desse sistema depende criticamente da capacidade do *OVS* de classificar pacotes e manter um cache efetivo. Inicialmente, o OVS utilizava um modelo reativo baseado em microfluxos (conexões exatas), mas a necessidade de escalabilidade levou à adoção de *Megaflows*. O cache de *Megaflows* permite que uma única entrada no kernel corresponda a múltiplos fluxos de transporte através do uso de máscaras genéricas (*wildcards*), reduzindo drasticamente o número de chamadas ao espaço de usuário.

No contexto de OVN, OVS não é apenas um “encaminhador” de pacotes, mas o ponto de aplicação de políticas lógicas complexas. O controlador (*ovn-controller*) deve traduzir a intenção lógica em tabelas OpenFlow que o *ovs-vswitchd* possa processar e converter em *Megaflows* eficientes. Portanto, a latência de convergência da rede está diretamente ligada à velocidade com que essas regras são computadas e instaladas no OVS diante de mudanças na topologia virtual.

2.2 Arquitetura do Open Virtual Network (OVN)

Open Virtual Network (OVN) representa uma evolução paradigmática na orquestração de redes virtuais, concebido explicitamente para superar as limitações das arquiteturas de SDN de primeira geração. Diferentemente das abordagens anteriores, que dependiam de agentes imperativos enviando configurações para switches individuais, OVN opera sob um modelo declarativo orientado a banco de dados. Nesta arquitetura, o estado armazenado é a "fonte final

da verdade", permitindo que o sistema escale para atender às demandas de sistemas de gerenciamento em hiperescala, como OpenStack e Kubernetes.

A arquitetura do OVN desacopla a intenção lógica da realização física através de um pipeline de tradução dividido em três estágios principais, mediados por dois bancos de dados OVSDB distintos.

2.2.1 O Modelo de Dois Bancos de Dados (NB e SB)

A espinha dorsal do plano de controle do OVN é a separação estrita entre a configuração lógica e o estado de execução:

- ***Northbound Database (NB_DB):*** atua como a interface de alto nível para o CMS (*Cloud Management System*), como o OpenStack Neutron. Este banco de dados armazena os elementos lógicos da rede, contendo tabelas para Switches Lógicos, Roteadores Lógicos, ACLs (*Access Control Lists*) e Balanceadores de Carga. *NB_DB* representa o estado desejado da rede, abstraindo completamente a infraestrutura física subjacente. Por exemplo, uma *ACL* é definida entre grupos de segurança lógicos, sem referência a endereços IP de hipervisores ou túneis.
- ***Southbound Database (SB_DB):*** funciona como a camada de sincronização e distribuição de estado. Contém o "estado realizado" da rede, armazenando *Logical Flows* (fluxos lógicos), *Port Bindings* (associações de portas lógicas a chassis físicos) e *MAC Bindings*. Crucialmente, SB_DB é o ponto de convergência para todos os nós de computação (Chassis). A escalabilidade deste componente é crítica, pois ele deve suportar conexões simultâneas e monitoramento de milhares de instâncias do ovn-controller. Para garantir consistência e alta disponibilidade, ambos os bancos de dados utilizam tipicamente o algoritmo de consenso RAFT.

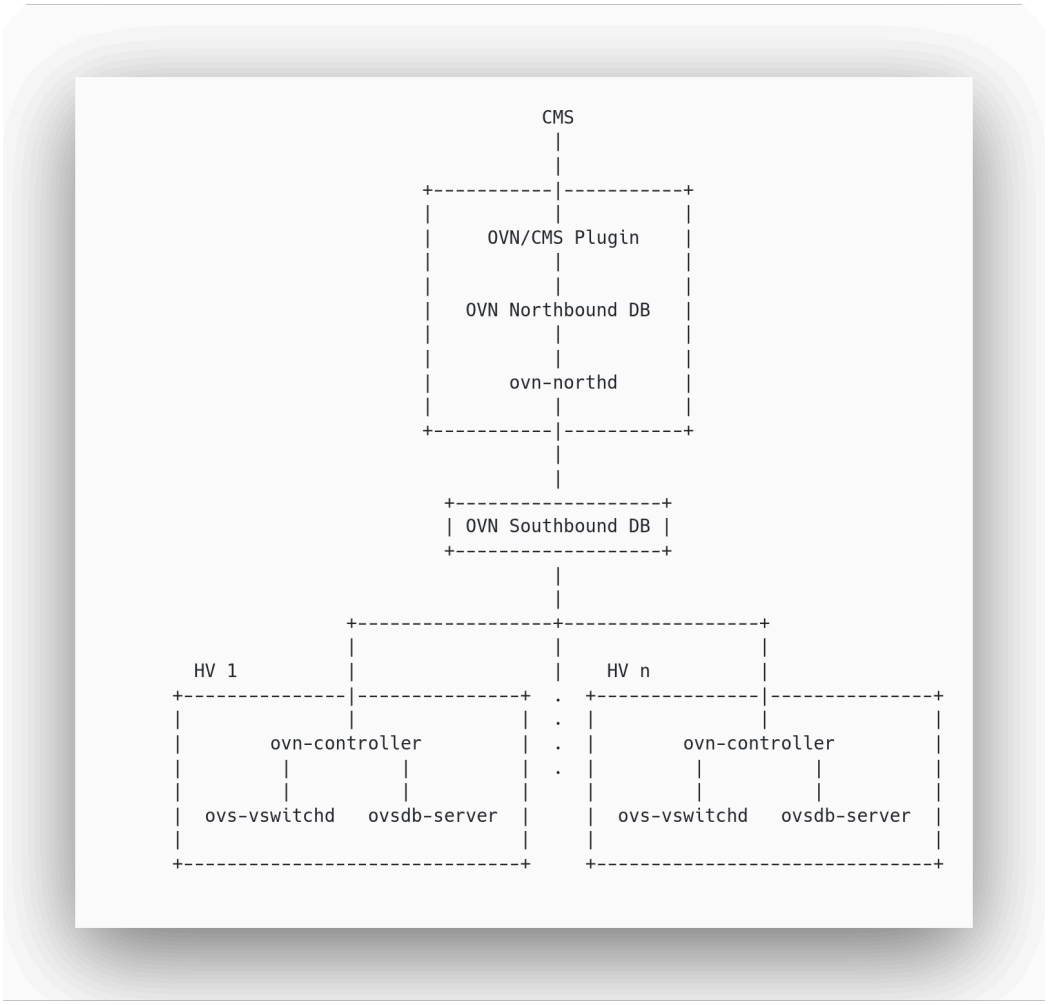
2.2.2 daemon ovn-northd

O *daemon ovn-northd* é responsável pela tradução das configurações de alto nível do OVN (armazenadas em *NB_DB*) para a configuração lógica no *SB_DB*.

2.2.3 daemon ovn-controller

Executado em cada hipervisor (*Chassis*), *ovn-controller* é o agente responsável pela realização física da rede. Ele opera como um cliente *OVSDB* que se conecta "para cima" ao *SB_DB* e "para baixo" ao *vswitch* local (*Open vSwitch*) via *OpenFlow*. Sua maneira de funcionamento depende da conexão *SB_DB* obtendo apenas os Logical Flows relevantes para as portas que estão hospedadas em seu chassi local. Em seguida, ele traduz esses fluxos lógicos (que contêm referências abstratas) em regras OpenFlow físicas (que contêm números de portas reais e IDs de túnel).

Figura 1 - Arquitetura OVN

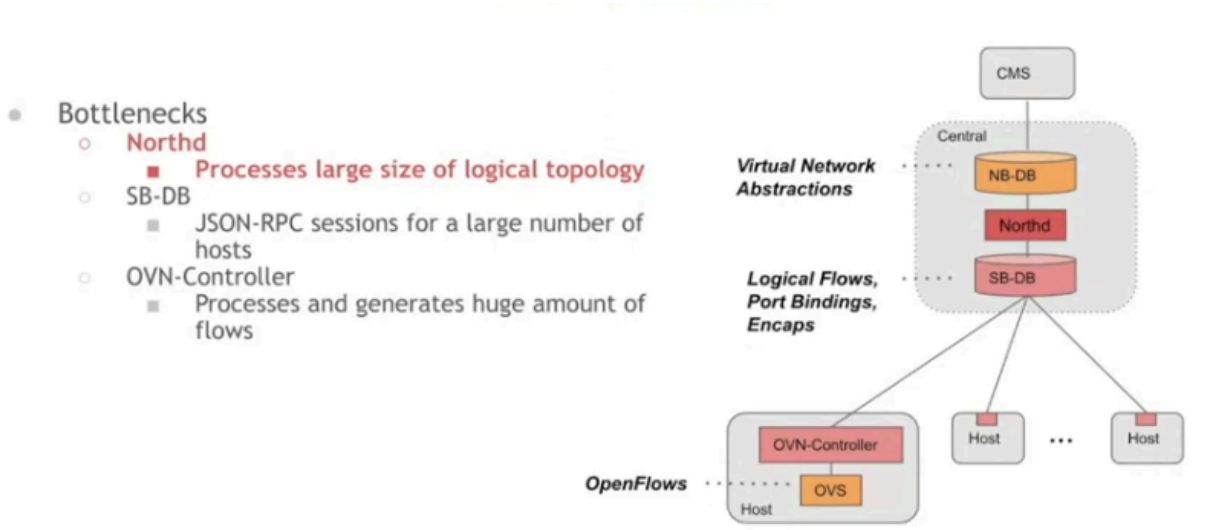


Fonte - *ovn-architecture(7) — Linux manual page*

2.3 Desafios de Consenso e Persistência de Dados (OVSDB)

A escalabilidade do plano de controle do OVN não depende apenas da eficiência algorítmica do ovn-northd, mas também está intrinsecamente ligada ao desempenho da camada de persistência de dados como visto na figura 2. Em ambientes de produção, os bancos de dados Northbound (NB) e Southbound (SB) operam em cluster utilizando o algoritmo de consenso RAFT para garantir alta disponibilidade.

Figura 2 - Principais Gargalos de Escalabilidade nos componentes do OVN



Fonte: Zhou (OVSCON, 2023).

2.3.1 O Algoritmo RAFT e as “Tempestades de Escrita”

O protocolo RAFT exige que todas as operações de escrita (transações) sejam propostas pelo líder e reconhecidas por um quórum de seguidores antes de serem confirmadas (commit). (OVN ARCHITECTURE, 2025).

Em cenários de hiperescala, ocorre o fenômeno das "Tempestades de Escrita" (*Write Storms*). Isso se manifesta quando milhares de instâncias do *ovn-controller* tentam atualizar o banco Southbound simultaneamente (ex: atualização de status de portas após um *reboot* de *cluster*). O líder RAFT deve serializar essas transações. Se o log crescer excessivamente ou a E/S de disco

saturar, o líder pode falhar no envio de *heartbeats*, desencadeando eleições de liderança espúrias que pausam temporariamente o plano de controle (MAXIMETS, 2021).

Além disso, o processo de Compactação de Log historicamente representava uma operação bloqueante. Em bancos de dados grandes, a compactação parava o processamento de novas transações, causando desconexão em massa de clientes (*Thundering Herd*). Otimizações recentes introduziram a compactação paralela para mitigar este risco (OVSDB PERFORMANCE, 2022).

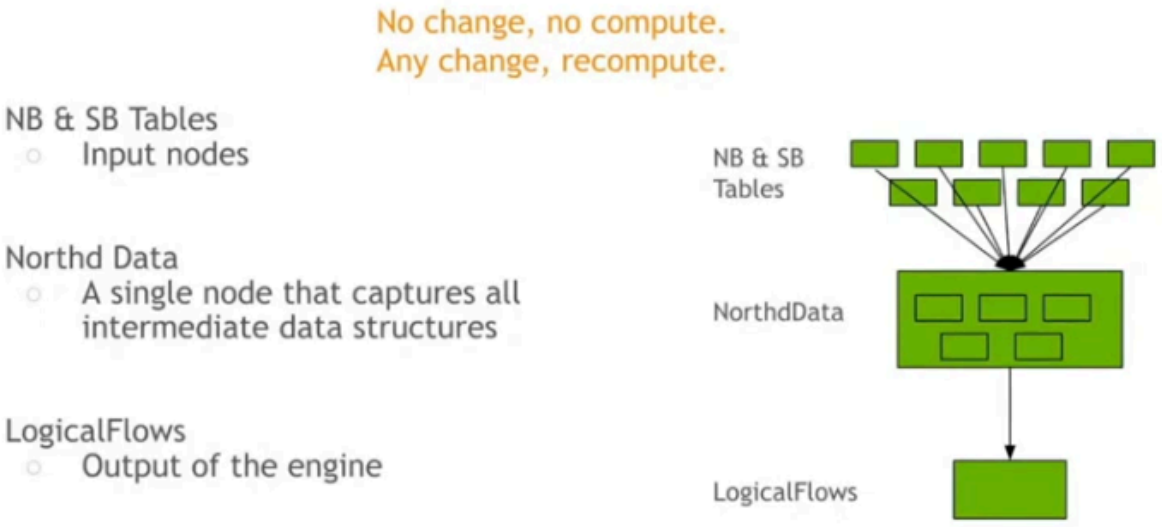
2.4 Evolução Arquitetural: do Recálculo Total ao Processamento Incremental

A maior transformação na engenharia do OVN foi a transição do modelo de recomputação total para o motor de Processamento Incremental (I-P).

2.4.1 Recomputação total

Nas versões iniciais, conforme ilustrado na figura 3, qualquer alteração no NBDB — por menor que fosse — acionava o *ovn-northd* para reler todo o banco, recalcular todos os fluxos lógicos e empurrar o novo estado para o *SBDB*. De maneira semelhante, *ovn-controller* gerava a tabela de OpenFlows inteira sempre que obtinha o conteúdo de *SBDB*. (MICHELSON, 2019)

Figura 3 - Funcionamento inicial do *northd*l



Fonte: Zhou (OVSCON, 2023).

2.4.2 O Motor I-P e o Grafo Acíclico Dirigido (DAG)

Para resolver esse gargalo, foi implementado o motor I-P, que estrutura a lógica interna como um **Grafo Acíclico Dirigido (DAG)** de nós de processamento.

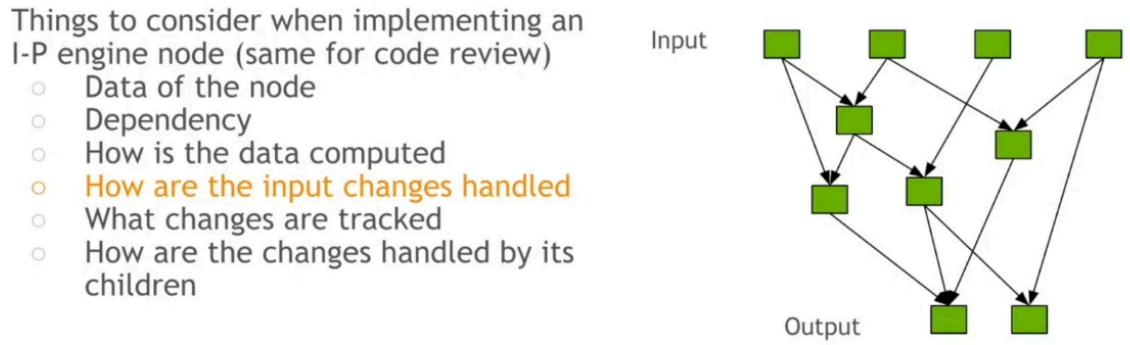
- **Nós do Motor**: componentes como en_northd (input) e en_lflow (output).
- **Manipuladores de Mudança (*Change Handlers*):** funções específicas que processam apenas o delta das alterações. Por exemplo, se uma nova ACL é criada, o motor executa apenas o manipulador dessa ACL, sem reprocessar toda a topologia de roteadores lógicos (ZHOU, 2023).

Conforme apresentado na figura 4, existem diversos fatores a considerar ao implementar um nó de processamento incremental. A evolução do motor I-P se deu através do refinamento da granularidade de seus nós, com a decomposição de estruturas monolíticas em componentes especializados (ZHOU; SIDDIQUE, 2021).

A separação de responsabilidades e o isolamento de recursos voláteis, exemplificado pela criação de nós dedicados para dados de Balanceadores de Carga (*lb_data*), foram fundamentais para

impedir que alterações frequentes na configuração de serviços disparassem recomputações globais na topologia lógica (OPEN VIRTUAL NETWORK, 2024).

Figura 4 - Considerações para implementar processamento incremental



Fonte: Zhou (OVSCON, 2023)

2.5 Avaliação de Desempenho

A comunidade de desenvolvimento do OVN estabeleceu ferramentas padrão para validação de escalabilidade do sistema, focadas em simulação eficiente de cenários massivos:

- **ovn-fake-multinode**: apresentada por Siddique (2019), esta ferramenta utiliza a técnica de containers aninhados (Docker-in-Docker) para instanciar centenas de chassis simulados em um único servidor físico. Ela virtualiza a camada de *underlay*, permitindo testar a convergência do plano de controle sem a necessidade de hardware massivo.
- **ovn-heater**: detalhada por Maximets (2022), trata-se de um gerador de carga desenhado para testar o OVN "mais próximo do mundo real". Ele atua como um cliente sintético do banco *Northbound*, simulando a rotatividade (*churn*) típica de um orquestrador de nuvem (como criação e destruição rápida de portas e balanceadores) e medindo a latência de convergência através de sondas no plano de dados simulado.

Embora eficazes para o desenvolvimento do núcleo do OVN (teste de regressão e otimização de banco de dados), essas ferramentas isolam o componente de rede do orquestrador de nuvem real.

Nesse sentido, a presente pesquisa busca preencher a lacuna da avaliação em cenários integrados, onde a sobrecarga do Neutron e a comunicação RPC também influenciam a performance final.

3. Análise da Implementação de Incremental Processing no northd

A viabilização do Processamento Incremental (I-P) para a criação e exclusão de Logical Switches no ovn-northd não se tratou apenas de uma adição funcional, mas de uma refatoração estrutural profunda, conduzida através da série de contribuições desenvolvida por Lorenzo Bianconi (BIANCONI, 2024). O conjunto de alterações analisado (Patches 1 a 5) pode ser segmentado em duas fases distintas: a modernização das primitivas de dados para suportar dinamismo e a implementação da lógica de produtor-consumidor no grafo de dependências do motor.

3.1 Modernização das Estruturas de Dados e Alocação Dinâmica (Patches 1, 2 e 3)

A arquitetura legada do OVN dependia fortemente de arrays estáticos e redimensionamentos manuais de memória, uma abordagem eficiente para varreduras lineares (típicas de recomputação total), mas proibitiva para o dinamismo exigido pelo processamento incremental. Os três primeiros patches estabelecem a fundação necessária para superar essa rigidez.

O Patch 1: (BIANCONI, 2024a) introduziu a estrutura *dynamic_bitmap* em *lib/ovn-util.h.* Anteriormente, o gerenciamento de mapas de bits — essenciais para associar recursos como Balanceadores de Carga a Switches — era feito de forma ad-hoc. A nova estrutura encapsula a gestão de capacidade e introduz a função *ovn_bitmap_realloc*, que garante a inicialização segura (zero-initialization) de novos bits, prevenindo a corrupção de estado lógico durante a expansão da rede.

Complementarmente, o Patch 3: (BIANCONI, 2024c) substituiu os vetores manuais de ponteiros (*struct ovn_datapath **array*) pela primitiva *struct vector* na definição de *ovn_datapaths*. Esta mudança altera a estratégia de alocação de linear para geométrica (*amortized doubling*), eliminando o custo excessivo de cópias de memória via *xrealloc* em clusters grandes. O uso combinado de vetores e bitmaps permitiu a implementação da função *ods_assign_array_index*, que recicla índices de datapaths deletados (*dynamic_bitmap_scan*), mantendo a densidade e a eficiência da estrutura.

O Patch 2: (BIANCONI, 2024b) aplicou essas novas primitivas ao subsistema de *Load Balancers*, modificando as funções de associação (*ovn_lb_datapaths_add_ls*) para utilizar *dynamic_bitmap_realloc*. Isso tornou o sistema resiliente ao crescimento horizontal dinâmico, eliminando a necessidade de estimativas prévias do tamanho do cluster.

3.2 O Produtor de Eventos: Rastreamento no Nó *en_northd* (Patch 4)

Com a infraestrutura de dados modernizada, o Patch 4: (BIANCONI, 2024d) implementou a lógica central de detecção de mudanças no nó produtor *en_northd*. Este componente deixa de ser *stateless* entre execuções e passa a persistir o estado das alterações através da estrutura *northd_tracked_data*.

A inovação central deste *patch* reside no uso de contêineres de referência (*hmapx*) para implementar um rastreamento "zero-copy". A estrutura *tracked_dps* mantém listas separadas para *datapaths* criados/atualizados (*crupdated*) e deletados (*deleted*).

Para operações de exclusão, o patch introduziu o padrão de Deleção Diferida. Ao deletar um switch lógico na função *northd_handle_ls_changes*, o motor remove o objeto das estruturas de busca globais e libera seu índice no bitmap, mas não libera a memória do objeto imediatamente. O objeto é mantido vivo e adicionado à lista de rastreamento *deleted*. Isso é crucial para que os nós consumidores possam acessar metadados do switch (como seu UUID) para limpar suas próprias dependências antes que o objeto seja destruído definitivamente na função *destroy_tracked_dps*.

A sinalização para o restante do grafo é feita através da nova flag *NORTHD_TRACKED_SWITCHES*, que atua como um contrato de interface, indicando aos consumidores que existem mudanças seguras para serem processadas incrementalmente.

Figura 3 - Pseudocódigo para *northd/northd.c* função *northd_handle_ls_changes*

```
// PSEUDOCÓDIGO - PATCH 4 (Nó en_northd)

Função northd_handle_ls_changes(entradas, dados_northd):

    // 1. Inicializa estrutura de rastreamento
    dados_northd.trk_data.type = NORTHD_TRACKED_NONE

    // 2. Verifica se existem mudanças nas tabelas de Logical Switch
(LS)
    Se (não há LS novos E não há LS deletados E não há LS atualizados):
        Retornar FALSO (Forçar Recompute) // Segurança

    // ------------------------------------------------------------
    // PROCESSAMENTO DE CRIAÇÃO (NEW)
    // ------------------------------------------------------------
    Para cada switch NOVO (new_ls) na lista de entradas:

        // Verifica se o switch é complexo demais para I-P agora
        // (Ex: tem QoS, Forwarding Groups ou Meter Band associados na
criação)
        Se (new_ls tem COPP OU Load Balancer OU DNS Records...):
            Retornar FALSO (Abortar para Recompute Total)

        // Cria o objeto ovn_datapath na memória (Alocação)
        od = ovn_datapath_create(new_ls)

        // Configura índices e bitmaps para este novo datapath
        ods_assign_array_index(dados_northd.ls_datapaths, od)
        init_ipam_info(od) // Inicializa gerenciamento de IP

        // Adiciona este datapath à lista de "Criados/Atualizados" para
rastreamento
        hmapx_add(dados_northd.trk_data.trk_switches.crupdated, od)

    // ------------------------------------------------------------
    // PROCESSAMENTO DE DELEÇÃO (DELETED)
    // ------------------------------------------------------------
    Para cada switch DELETADO (deleted_ls) na lista de entradas:

        // Encontra o objeto ovn_datapath existente na memória
        od = encontrar_datapath_por_uuid(deleted_ls.uuid)

        Se (od não existe):
            Logar Erro "Datapath não encontrado"
            Retornar FALSO (Abortar)

        // Remove das estruturas principais de memória (Hash Map e
Vetor)
        hmap_remove(dados_northd.ls_datapaths, od)
        limpar_
```

Fonte: Autoral

3.3 O Consumidor e a Desvinculação de Fluxos: Nó en_ls_stateful (Patch 5)

O Patch 5: (BIANCONI, 2024e) fechou o ciclo de processamento incremental, alterando o nó consumidor *en_ls_stateful*. Historicamente, este nó agia como uma barreira de escalabilidade, forçando recomputações totais ao detectar qualquer alteração na topologia de switches.

A nova implementação inverteu a lógica de dependência: o manipulador *ls_stateful_northd_handler* passou a aceitar explicitamente eventos onde a flag *NORTHD_TRACKED_SWITCHES* está presente. Ao iterar sobre a lista de switches criados (*crupdated*), o nó gera ou atualiza seus registros de estado internos de forma idempotente, criando o *ls_stateful_record* apenas se ele ainda não existir.

O ganho de desempenho mais significativo ocorre no processamento de deleções. Ao consumir a lista de switches deletados fornecida pelo produtor, o nó remove o registro de sua tabela interna e propaga o evento para o próximo estágio. Finalmente, a função *lflow_handle_ls_stateful_changes* executa a chamada crítica *lflow_ref_unlink_lflows*. Este mecanismo permite identificar e remover cirurgicamente apenas os fluxos lógicos associados ao switch removido, transformando uma operação de complexidade O(N) (regeneração total) em O(1) (remoção pontual), o que explica a redução de latência observada nos testes de escalabilidade.

Figura 4 - Pseudocódigo para *northd/en-ls-stateful.c* função *ls_stateful_northd_handler*

```
// PSEUDOCÓDIGO - PATCH 5 (Nó en_ls_stateful)

Função ls_stateful_northd_handler(node, dados_stateful):

    // 1. Verifica se a mudança veio do northd e contém switches rastreados
    Se (dados_northd tem flag NORTHD_TRACKED_SWITCHES):

        // ------------------------------------------------------
        // CONSUMIR CRIAÇÃO (CRUPDATED)
        // ------------------------------------------------------
        Para cada datapath (od) na lista trk_switches.crupdated:

            // Verifica se já temos um registro de estado para ele
            Se (não existe registro na tabela stateful):
                // Cria um novo registro para gerenciar ACLs/LBs deste switch
                ls_stateful_record_create(dados_stateful.table, od)

        // ------------------------------------------------------
        // CONSUMIR DELEÇÃO (DELETED)
        // ------------------------------------------------------
        Para cada datapath (od) na lista trk_switches.deleted:

            // Encontra o registro de estado associado
            registro = ls_stateful_table_find(od)

            Se (registro existe):
                // Remove da tabela interna de registros stateful
                hmap_remove(dados_stateful.table, registro)

                // Adiciona à lista local de deletados (para o próximo nó:
en_lflow)
                hmapx_add(dados_stateful.trk_data.deleted, registro)

        Retornar HANDLED_UPDATED (Sucesso)

    // Se houver mudanças que não sejam de switches, LBs ou ACLs...
    Retornar UNHANDLED (Recompute)

// ------------------------------------------------------------
// EFEITO COLATERAL CRÍTICO (Limpeza de Fluxos)
// Ocorrem em lflow_handle_ls_stateful_changes (northd.c)
// ------------------------------------------------------------
Função lflow_handle_ls_stateful_changes(entradas, dados):

    // Este passo ocorre no nó seguinte (en_lflow), consumindo a saída acima
    Para cada registro na lista dados_stateful.trk_data.deleted:

        // Desvincula (Unlink) todos os fluxos lógicos associados a este switch
        // Isso remove as entradas da tabela Logical_Flow no SBDB eficientemente
        lflow_ref_unlink_lflows(registro.lflow_ref)

    Retornar VERDADEIRO
```

Fonte: Autoral

4. Metodologia

Para avaliar o impacto das otimizações de plano de controle no Open Virtual Network (OVN), foi adotada uma metodologia experimental baseada na geração controlada de carga sintética em um ambiente OpenStack de produção. Nesta seção são descritos, em detalhe, (i) a arquitetura do ambiente de testes, (ii) scripts de automação desenvolvido para orquestrar os experimentos e (iii) os procedimentos utilizados para coleta e tratamento das métricas de desempenho.

4.1 Ambiente Experimental

Os experimentos foram conduzidos num ambiente de cluster mantido no DC (Cirrus), que trata-se de uma infraestrutura de nuvem privada gerenciada via Juju e MAAS (*Metal as a Service*). O ambiente é composto por nós de controle (onde residem os serviços ovn-northd e bancos de dados OVSDB em cluster RAFT) e nós de computação (chassis).

Para viabilizar a análise aprofundada de desempenho (*profiling*), foi necessário preparar uma compilação customizada dos pacotes OVN (versão atual 25.09 e a release 24.09). Para garantir a máxima fidelidade na análise de perfilamento e eliminar a dependência de bibliotecas dinâmicas do sistema que careciam de símbolos de depuração, optou-se por uma compilação estática e monolítica. Open vSwitch (OVS) foi pré-compilado com as flags --disable-shared -fPIC, e o OVN foi subsequentemente ligado estaticamente a estas bibliotecas utilizando a opção DEB_BUILD_OPTIONS="nostrip".

4.2 Orquestração e Automação

Dada a complexidade de alternar versões de software em um cluster distribuído e a necessidade de garantir um estado inicial consistente para cada bateria de testes, foi desenvolvido um conjunto de scripts de automação. Estes scripts automatizam o gerenciamento de ciclo de vida do experimento, dividido em 2 módulos funcionais. Para garantir a reprodutibilidade da pesquisa, todo o código-fonte desenvolvido encontra-se disponível publicamente no repositório: https://github.com/ViniRodrig/TCC.

4.2.1 Módulo de Orquestração de Versão e Estado (deploy-ovn-version.sh)

A substituição de versões do OVN em um ambiente produtivo gerenciado por Juju apresenta desafios de consistência, especialmente devido à natureza distribuída do banco de dados OVSDB (que opera com algoritmo de consenso RAFT) e à necessidade de compatibilidade estrita entre os binários e os símbolos de depuração.

Para garantir a reprodutibilidade dos cenários, foi desenvolvido o script *deploy-ovn-version.sh*. Este componente atua como um orquestrador de implantação da plataforma (*deployment*), executando uma máquina de estados finita que garante a transição atômica do cluster. O funcionamento do algoritmo é dividido em quatro fases críticas:

1. Fase de Preparação e Limpeza:
    - O script identifica a unidade líder do cluster OVN Central.
    - Para garantir que métricas de versões anteriores não contaminem o novo teste, o serviço é interrompido e os arquivos de banco de dados (/var/lib/ovn/*.db) são removidos (Wipe).
    - São instalados simultaneamente os pacotes binários (.deb).
    - O banco de dados é reinicializado com o comando ovsdb-tool create-cluster, estabelecendo um novo termo de liderança RAFT e reconfigurando os certificados SSL necessários para a comunicação segura com o Neutron.

Figura 5 - Pseudocódigo para *deploy-ovn-version.sh*, fase 1.

```
FASE 1 — Leader (reset + bootstrap do cluster)
  Copiar pacotes .deb para /tmp do leader (juju scp)
  Parar serviços OVN/OVSDB no leader
  Matar processos ovsdb-server remanescentes
  Apagar bancos/estado antigo (/var/lib/ovn/* e /etc/ovn/*.db)
  Criar /var/lib/ovn

  INSTALL_CENTRAL(leader)

  leader_ip ← obter IP do leader (hostname -I)
  Criar novo cluster OVSDB:
    ovsdb-tool create-cluster NB em tcp:leader_ip:6643
    ovsdb-tool create-cluster SB em tcp:leader_ip:6644

  Iniciar serviços no leader:
    systemctl start ovn-ovsdb-server-nb, ovn-ovsdb-server-sb, ovn-northd

  Configurar listeners remotos (SSL) no leader:
    NB: pssl:6641:leader_ip
    SB: pssl:6642:leader_ip
    SB (para Neutron): pssl:16642:leader_ip

  Popular tabela SSL (mTLS) nos bancos NB/SB:
    ovn-nbctl set-ssl (key_host, cert_host, ovn-central.crt)
    ovn-sbctl set-ssl (key_host, cert_host, ovn-central.crt)
```

Fonte: Autoral

2. Fase de Convergência do Cluster:
    - Para as unidades seguidoras (followers), o *script* executa um procedimento de limpeza similar.
    - Diferente do líder, estas unidades não criam um novo cluster, mas executam o comando *ovsdb-tool join-cluster*, conectando-se ao IP do líder recém-provisionado. Isso reconstrói o quórum de alta disponibilidade necessário para o funcionamento do plano de controle.

Figura 6 - Pseudocódigo para *deploy-ovn-version.sh, fase 2*

```
FASE 2 — Followers (wipe + join no cluster)
  Para cada follower em FOLLOWER_UNITS:
    Copiar pacotes .deb para /tmp do follower
    Parar serviços OVN/OVSDB e matar ovsdb-server
    Apagar bancos/estado antigo

    Preparar caminhos/symlinks esperados:
      criar /var/lib/ovn
      linkar /var/lib/ovn/ovnnb_db.db → /etc/ovn/ovnnb_db.db
      linkar /var/lib/ovn/ovnsb_db.db → /etc/ovn/ovnsb_db.db

    INSTALL_CENTRAL(follower)

    local_ip ← obter IP do follower (hostname -I)
    Entrar no cluster:
      ovsdb-tool join-cluster NB:
        arquivo local + nome "OVN_Northbound"
        tcp:local_ip:6643 conectando no leader tcp:leader_ip:6643
      ovsdb-tool join-cluster SB:
        arquivo local + nome "OVN_Southbound"
        tcp:local_ip:6644 conectando no leader tcp:leader_ip:6644

    Iniciar serviços OVN no follower
    Configurar listeners remotos (SSL) no follower:
      NB: pssl:6641:local_ip
      SB: pssl:6642:local_ip
      SB (Neutron): pssl:16642:local_ip
```

Fonte: Autoral

3. Fase de Atualização do Plano de Dados (Chassis Upgrade):
    - O *script* itera sobre todas as unidades de computação (ovn-chassis).
    - Para evitar processos "zumbis" que poderiam corromper a coleta de dados, o script força a parada do ovn-controller e remove processos residuais da memória antes de instalar os novos binários customizados.
4. Fase de Reintegração com o OpenStack:

- Por fim, o serviço neutron-server é reiniciado para forçar um novo handshake SSL e a reconexão com o banco de dados Northbound recém-criado, restabelecendo a gestão da nuvem.
- Este fluxo automatizado reduz o tempo de reconfiguração do ambiente de horas para minutos e elimina inconsistências de configuração que poderiam invalidar a comparação entre as versões do OVN.

Figura 7 - Pseudocódigo para *deploy-ovn-version.sh, fase 3 e 4.*

```
FASE 3 – Chassis (upgrade dos nós de dados)
  Para cada chassis em CHASSIS_UNITS:
    Copiar pacotes .deb (ovn-common, ovn-host, ovn-controller-vtep)

    Parar serviços ovn-host/ovn-controller
    Matar processos ovn-controller “zumbis” (killall -9)

    INSTALL_CHASSIS(chassis)

    Iniciar ovn-host


FASE 4 – Neutron (forçar novo handshake SSL)
  Reiniciar neutron-server (neutron-api/0)

Finalizar:
  Exibir mensagem de sucesso com a VERSION instalada
```

Fonte: Autoral

4.2.2 Injeção de Carga Sintética (measure-scalability.py)

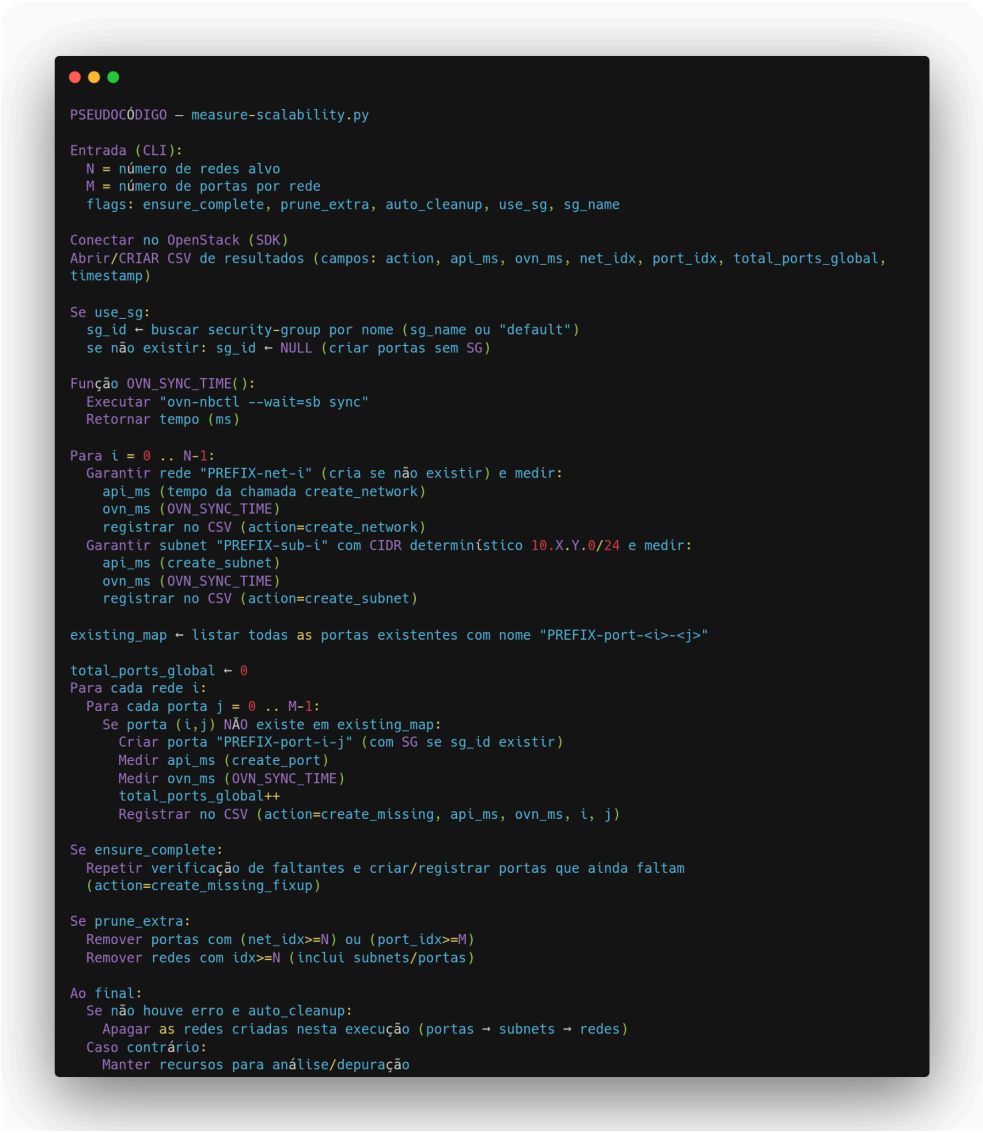
A avaliação de escalabilidade desenvolvida não se baseia em tráfego de dados do usuário (plano de dados), mas sim na pressão exercida sobre o plano de controle. Para isso, foi desenvolvido o *script* measure-scalability.py utilizando o SDK do OpenStack.

O *script* simula um cenário de hiperescala através da criação massiva de recursos lógicos. Em uma execução típica de teste, o *script* provisiona:

- Topologia Lógica: Criação de N Redes Virtuais (Logical Switches).
- Densidade de Portas: Criação de M Portas por rede (ex: 500 redes x 10 portas = 5.000 portas totais).

Embora estas portas não estejam conectadas a VMs ativas (evitando o esgotamento de RAM dos hipervisores), elas forçam o ovn-northd a realizar o ciclo completo de computação: alocação de endereçamento IP, cálculo de fluxos lógicos e atualização do banco *Southbound*.

Figura 8 - Pseudocódigo para *measure-scalabilty.py*



```
PSEUDOCÓDIGO — measure-scalability.py

Entrada (CLI):
  N = número de redes alvo
  M = número de portas por rede
  flags: ensure_complete, prune_extra, auto_cleanup, use_sg, sg_name

Conectar no OpenStack (SDK)
Abrir/CRIAR CSV de resultados (campos: action, api_ms, ovn_ms, net_idx, port_idx, total_ports_global,
timestamp)

Se use_sg:
  sg_id ← buscar security-group por nome (sg_name ou "default")
  se não existir: sg_id ← NULL (criar portas sem SG)

Função OVN_SYNC_TIME():
  Executar "ovn-nbctl --wait=sb sync"
  Retornar tempo (ms)

Para i = 0 .. N-1:
  Garantir rede "PREFIX-net-i" (cria se não existir) e medir:
    api_ms (tempo da chamada create_network)
    ovn_ms (OVN_SYNC_TIME)
    registrar no CSV (action=create_network)
  Garantir subnet "PREFIX-sub-i" com CIDR determinístico 10.X.Y.0/24 e medir:
    api_ms (create_subnet)
    ovn_ms (OVN_SYNC_TIME)
    registrar no CSV (action=create_subnet)

existing_map ← listar todas as portas existentes com nome "PREFIX-port-<i>-<j>"

total_ports_global ← 0
Para cada rede i:
  Para cada porta j = 0 .. M-1:
    Se porta (i,j) NÃO existe em existing_map:
      Criar porta "PREFIX-port-i-j" (com SG se sg_id existir)
      Medir api_ms (create_port)
      Medir ovn_ms (OVN_SYNC_TIME)
      total_ports_global++
      Registrar no CSV (action=create_missing, api_ms, ovn_ms, i, j)

Se ensure_complete:
  Repetir verificação de faltantes e criar/registrar portas que ainda faltam
  (action=create_missing_fixup)

Se prune_extra:
  Remover portas com (net_idx>=N) ou (port_idx>=M)
  Remover redes com idx>=N (inclui subnets/portas)

Ao final:
  Se não houve erro e auto_cleanup:
    Apagar as redes criadas nesta execução (portas → subnets → redes)
  Caso contrário:
    Manter recursos para análise/depuração
```

Fonte: Autoral

O *script* registra o tempo de convergência de cada transação, permitindo a construção de curvas de latência em função do número de recursos existentes, incluindo o tempo de resposta da API no Neutron e o tempo efetivo para as mudanças serem sincronizadas. Através do comando *ovn-nbctl –wait=sb sync* garantimos que as mudanças no *Northbound* tenham sido calculadas pelo *northd* e sido propagadas para o *Southbound*.

Figura 9 - Métricas exportadas

total_ports_global	network_index	port_index	duration_ms	timestamp	action	api_ms	ovn_ms
-1	0	-1	164.2	1765847170.5119772	create_network	2549.65	164.2
-2	0	-1	238.75	1765847173.1571846	create_subnet	2327.83	238.75
-3	1	-1	182.33	1765847175.5978565	create_network	2071.81	182.33
-4	1	-1	219.83	1765847178.7251766	create_subnet	2709.54	219.83
1	0	0	209.11	1765847180.7000513	create_missing	1531.81	209.11
2	0	1	214.78	1765847182.2611997	create_missing	1346.11	214.78
3	1	0	169.46	1765847183.7546334	create_missing	1323.65	169.46
4	1	1	147.31	1765847185.3572607	create_missing	1455.06	147.31

Fonte: Autoral

4.3 Justificativa Metodológica: Teste Integrado vs. Simulação Sintética

A literatura técnica especializada e as atualizações recentes de performance do projeto (MAXIMETS, 2022) destacam o uso de ferramentas como o *ovn-heater* para testes de escala. Diferente de *scripts* simples de injeção de dados, essa ferramenta foi projetada para mimetizar o comportamento de um CMS (*Cloud Management System*), gerando cargas de trabalho sintéticas que simulam a criação de portas e balanceadores diretamente no banco Northbound do OVN.

Embora essa abordagem seja ideal para isolar gargalos internos do OVN (como a compactação do banco de dados ou eleições de liderança RAFT), ela abstrai a complexidade do software de orquestração real. Em ambientes produtivos, o desempenho percebido pelo operador é a soma das latências de toda a pilha: API OpenStack → Processamento do Plugin Neutron → Transação OVSDB Northbound → Northd → Southbound → Chassis.

A metodologia adotada neste trabalho, utilizando o *script* measure-scalability.py sobre um cluster OpenStack completo, difere da abordagem sintética por exercitar o caminho crítico real. Enquanto o ovn-heater avalia o desempenho do OVN sob um "cliente idealizado", a abordagem deste trabalho captura o custo computacional (*overhead*) da tradução de modelos e da comunicação RPC intrínseca ao Neutron antes que a requisição chegue ao plano de controle da rede.

5. Análise de Desempenho e Escalabilidade do Plano de Controle

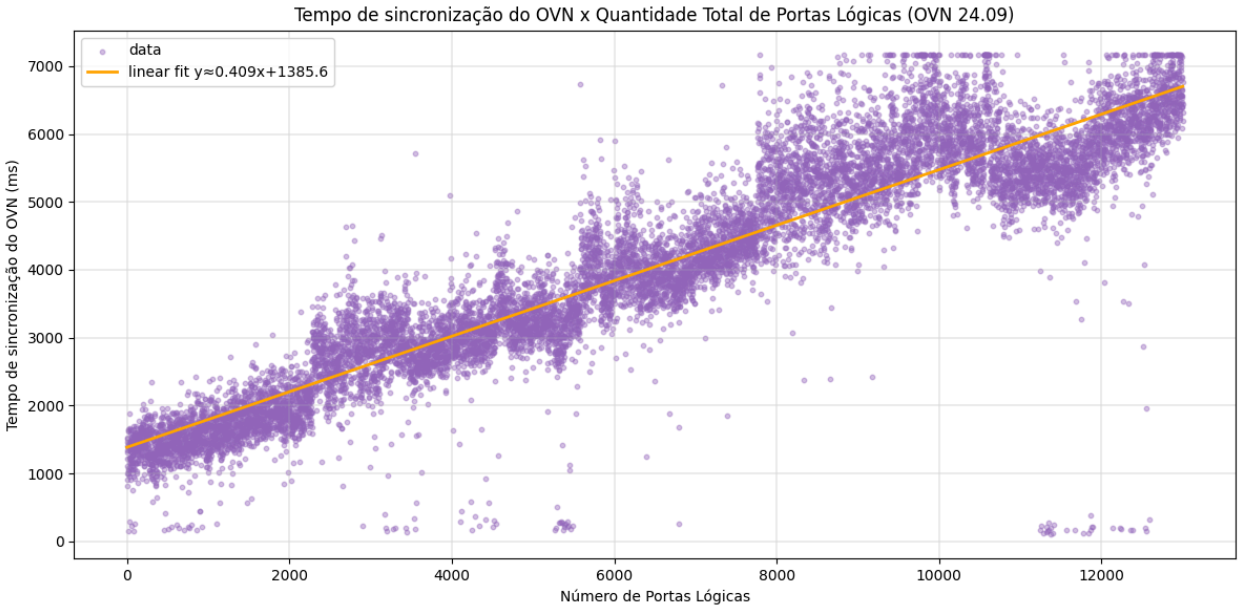
Os experimentos comparativos entre as versões do OVN 24.09.4 e 25.09.90 demonstram comportamentos distintos para as duas fases de provisionamento avaliadas. A análise quantitativa foi realizada através de regressão linear nas amostras de latência efetiva (ovn_ms_eff), permitindo modelar o custo computacional marginal para cada tipo de recurso adicionado.

5.1 Custo Incremental de Densidade de Portas

A quantidade de tempo que o northd demora para realizar seus cálculos está diretamente relacionada à quantidade de elementos lógicos presentes no NBDB.

A figura 10 demonstra uma relação direta entre número de portas e o tempo em que o northd demora para inserir uma nova parte a medida em que mais portas já existem e precisam ser reprocessadas. A figura representa a regressão linear para o tempo de criação na versão 24.09.

Figura 10 - Tempo de Criação x Número Total de Portas (OVN Versão 24.09)

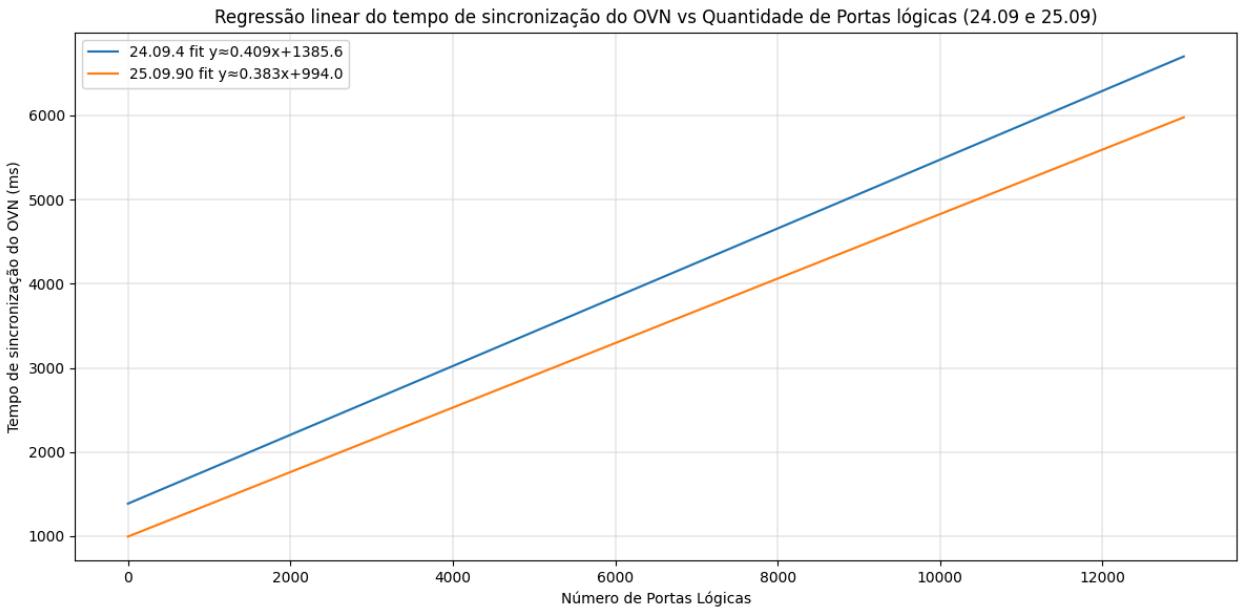


Fonte: Autoral

A Figura 11 apresenta a sobreposição das regressões lineares para a criação de portas nas versões analisadas. Visualmente, o gráfico exibe duas retas praticamente paralelas, o que evidencia uma complexidade algorítmica semelhante entre as versões. Esse comportamento é corroborado pelas equações das retas de tendência:

- Versão 24.09.4: y ≃ 0.409x + 1385.6 (Linha Azul)
- Versão 25.09.90: y ≃ 0.383x + 1385.0 (Linha Amarela)

Figura 11 - Comparação do tempo para criação de novas portas nas duas versões do OVN



Fonte - Autoral

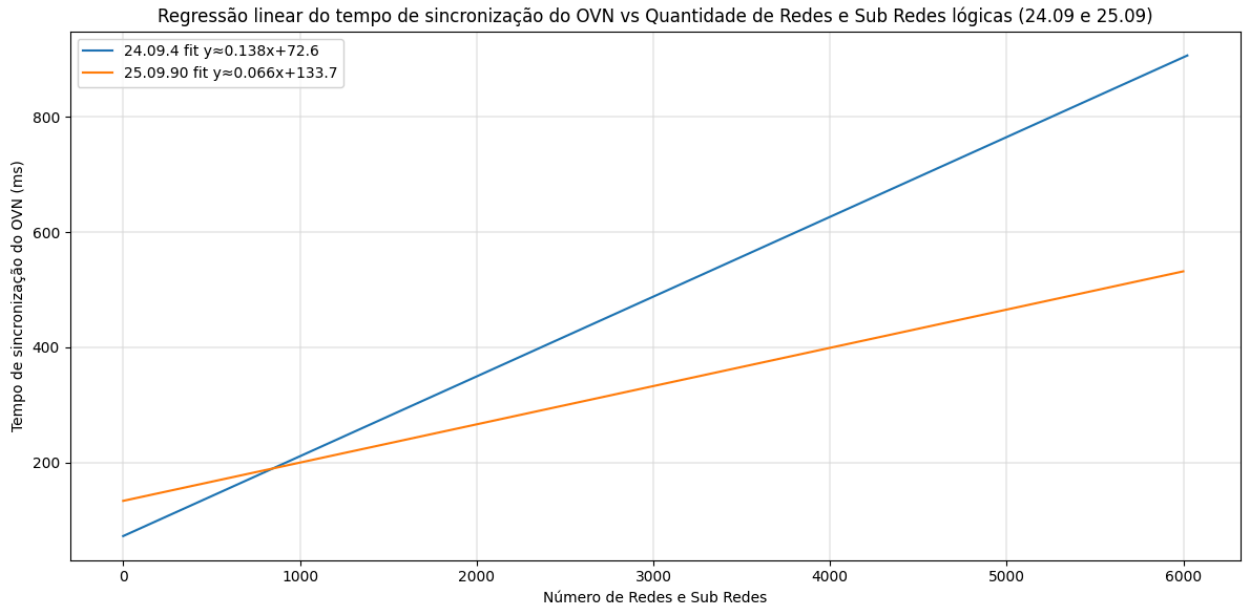
Diferentemente da Fase 1, ambas as curvas mantêm um comportamento linear O(N). A inclinação das retas (coeficiente angular) sofreu apenas uma redução discreta de aproximadamente 6% (0.409 para 0.383 ms/porta). Essa similaridade nos coeficientes angulares confirma que, para o fluxo de trabalho de criação de Logical Switch Ports, não houve alteração algorítmica fundamental; a arquitetura de processamento manteve-se estável, sem a introdução de novos mecanismos de processamento incremental (I-P) que alterassem sua classe de complexidade.

5.2 Fase 2: Convergência na Criação de Topologias Lógicas

A Figura 12 ilustra a latência de convergência durante a criação sequencial de Networks e Subnets, totalizando 6000 componentes lógicos. Diferentemente do cenário anterior, aqui as curvas de tendência se cruzam, revelando uma alteração profunda no comportamento de escalabilidade entre as versões, conforme expresso pelas equações de regressão linear:

- Versão 24.09.4 (Linha Azul): $y \simeq 0.138x + 72.6$
- Versão 25.09.90 (Linha Amarela): $y \simeq 0.066x + 133.7$

Figura 12 - Comparação do tempo para criação de redes e sub-redes nas duas versões do OVN



Fonte: Autoral

Entretanto, observa-se um trade-off nos estágios iniciais. O intercepto da reta aumentou de 72.6 para 133.7, indicando um custo fixo inicial maior na nova versão. Esse comportamento é esperado e justifica a vantagem inicial da versão 24.09.4 (Linha Azul) em cargas baixas; o aumento deve-se ao overhead de inicialização das estruturas de rastreamento necessárias para o motor de Processamento Incremental (I-P). Contudo, esse custo fixo é rapidamente compensado pelo ganho de escala (a inclinação menor da reta).

Essa otimização valida empiricamente a eficácia das alterações introduzidas no código base, especificamente a interação entre dois mecanismos:

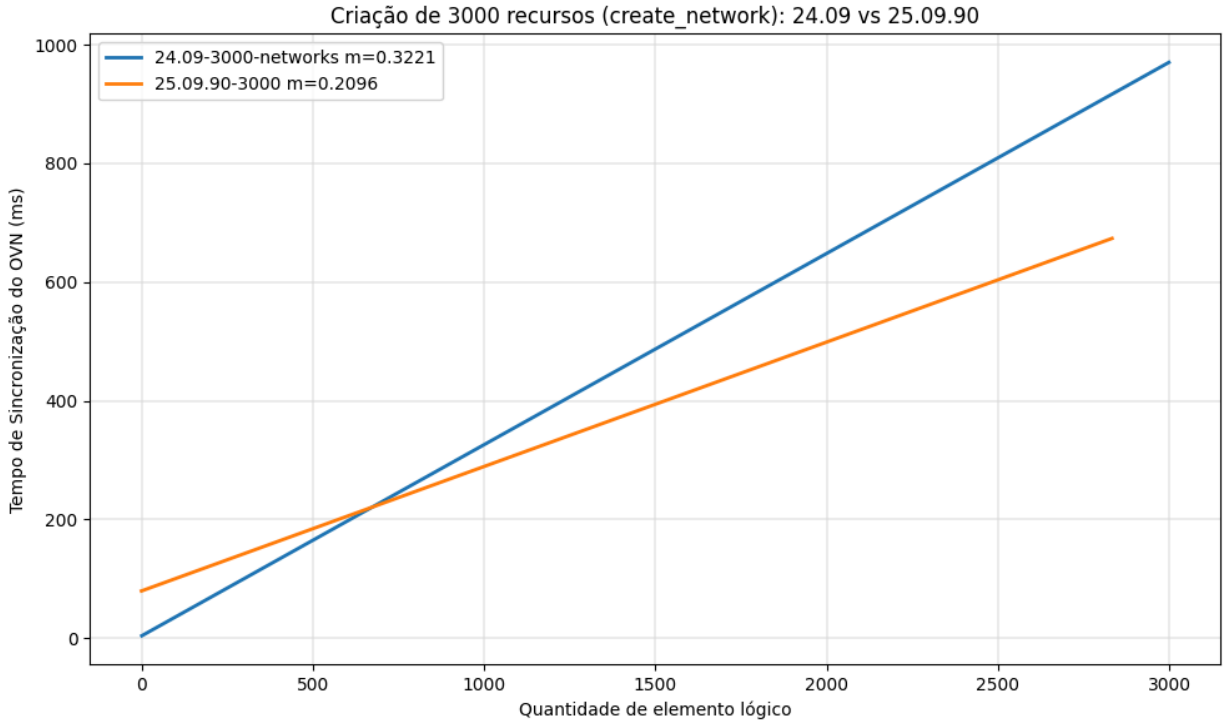
1. **Rastreamento de Mudanças (BIANCONI, 2024d):** Capacitou o nó en_northd a rastrear eventos de criação e exclusão de Logical Switches, substituindo o recálculo (recompute) completo de estruturas internas pela emissão de dados rastreados (tracked data).
2. **Consumo Incremental (BIANCONI, 2024e):** Modificou o nó dependente en_ls_stateful — responsável por gerenciar estados como Load Balancers e ACLs — para consumir apenas as diferenças fornecidas pelo en_northd.

O resultado prático é o "achatamento" da curva de latência observado na Figura 12: o daemon evita a recomputação de objetos não relacionados à nova topologia, garantindo que o tempo de convergência cresça de forma muito mais lenta em relação ao número de redes.

5.3 Fase 3: Impacto Isolado da criação de Networks x Subnets.

Na criação de redes (create_network), como apresentado pela figura 13 a regressão linear de ovn_ms em função do índice lógico de rede mostrou um decréscimo significativo na inclinação ao comparar 24.09 com 25.09.90: o coeficiente angular caiu de aproximadamente 0,32 ms por rede para cerca de 0,21 ms. Isso indica que, na nova versão, o custo incremental de adicionar mais redes cresce de forma mais suave, o que é consistente com os objetivos de escalabilidade do *incremental processing* no ovn-northd. Em termos práticos, ao redor de 3000 redes, a versão 25.09.90 apresenta tempos de convergência menores do que a 24.09 para essa ação.

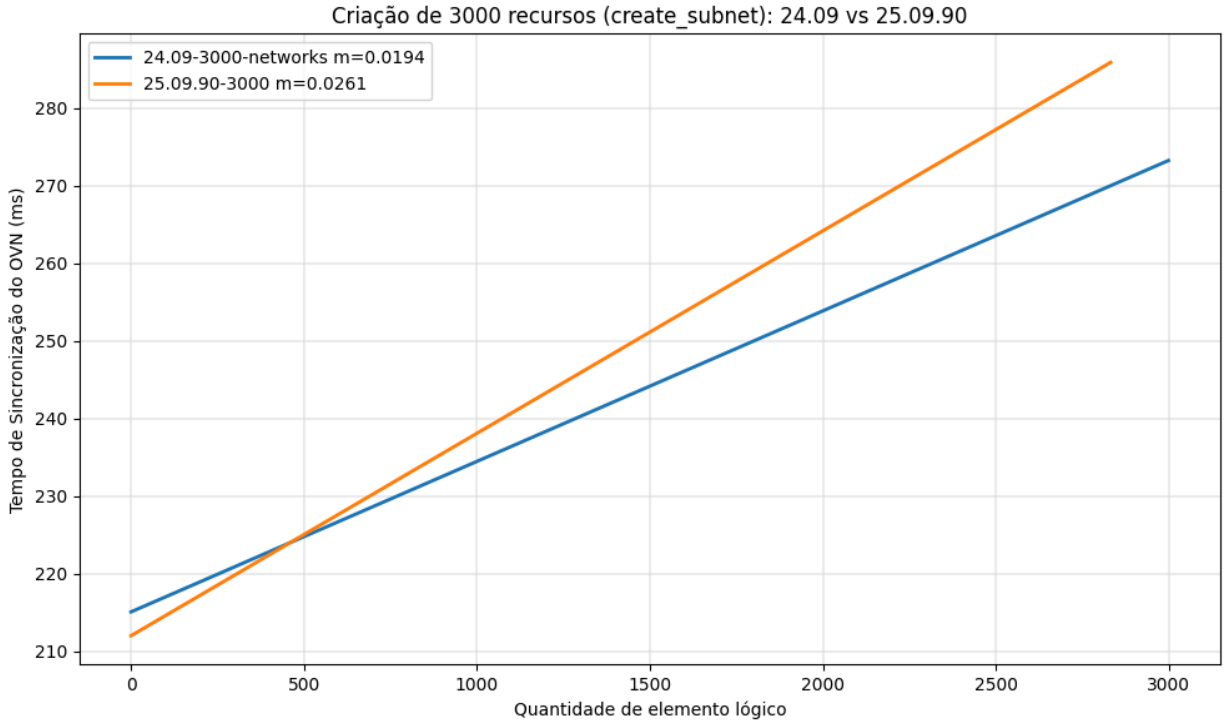
Figura 13 - Comparação do tempo para criação de redes nas duas versões do OVN



Fonte: Autoral

Já na criação de sub-redes (create_subnet), ilustrado pela figura 14 o comportamento é mais equilibrado entre as versões. Os intercepts das regressões são semelhantes (em torno de 210–215 ms), mas a inclinação é levemente maior em 25.09.90 (≈ 0,026 ms por índice, contra ≈ 0,019 ms em 24.09). Isso significa que, para um número muito elevado de sub-redes, a 25.09.90 pode apresentar um crescimento marginalmente mais acentuado do tempo de convergência nessa fase específica. No entanto, essa diferença é pequena em comparação com o ganho observado na criação de redes.

Figura 14 - Comparação do tempo para criação de redes nas duas versões do OVN



Fonte: Autoral

Considerando que a operação de criação de redes é mais custosa e dominante no *pipeline* de configuração da topologia (cada rede envolve mais recomputes e estruturas lógicas do que a sub-rede correspondente), o benefício global da versão 25.09.90 permanece positivo. Em resumo, mesmo com uma leve piora no *slope* de create_subnet, a redução mais expressiva na inclinação de create_network indica uma melhoria geral de escalabilidade na fase de provisionamento de redes.”

5.4 Discussão dos Resultados

Os resultados indicam que, entre as versões avaliadas, a evolução do OVN concentrou-se em reduzir o custo de manipulação da topologia lógica, em especial na criação de Logical Switches (networks). A Fase 2 mostra que, para a criação de redes e subnets em conjunto, a inclinação da

regressão linear caiu de aproximadamente 0,138 ms/rede na versão 24.09.4 para cerca de 0,066 ms/rede na 25.09.90, uma redução em torno de 52% no custo marginal por novo componente lógico. Essa "achatada" na curva é coerente com os commits que introduzem *Incremental Processing* para *Logical Switches* (rastreamento seletivo no en_northd e consumo incremental no en_ls_stateful), permitindo evitar recomputes completos a cada inserção na NBDB.

Quando isolamos o impacto da criação de redes (create_network), a análise de regressão reforça essa conclusão: o coeficiente angular cai de ≃ 0,32 ms por rede (24.09) para ≃ 0,21 ms (25.09.90). Em outras palavras, à medida que a quantidade de networks cresce, o custo incremental da nova versão aumenta mais lentamente, o que resulta em tempos de convergência menores na cauda do teste (por volta de 3000 redes) mesmo com um intercepto ligeiramente mais alto. Já na criação de sub-redes (create_subnet) o comportamento é bem mais estável entre as versões: os interceptos são praticamente equivalentes (≈ 210–215 ms) e a inclinação passa de ≃ 0,019 ms para ≃ 0,026 ms por subnet, um aumento pequeno se comparado ao ganho observado para networks.

Na Fase 1, dedicada à criação de portas (Logical Switch Ports), o comportamento permanece essencialmente linear em função do número total de portas já existentes. As regressões obtidas para as duas versões apresentam coeficientes angulares muito próximos (cerca de 0,409 ms/porta em 24.09.4 contra 0,383 ms/porta em 25.09.90), indicando que não houve mudança estrutural na complexidade algorítmica dessa operação: o padrão O(N) de reprocessamento de portas é preservado. A melhoria aparece sobretudo no intercepto (redução de ~1385,6 ms para ~994,0 ms), sugerindo otimizações em overheads fixos — por exemplo, inicialização de estruturas internas, serialização de dados ou caminhos de RPC — mas não uma alteração fundamental na forma como o northd trata o crescimento do número de portas.

Do ponto de vista operacional, isso significa que a versão 25.09.90 torna o sistema significativamente mais eficiente para cenários com grande número de redes isoladas (ambientes multi-tenant com muitos Logical Switches), pois o custo marginal de expandir a topologia lógica caiu tanto na análise agregada (networks + subnets) quanto na análise isolada de create_network. Ao mesmo tempo, o comportamento para a criação de portas permanece previsível e linear, com apenas ajustes de constante. Em conjunto, os dados sugerem que o *Incremental Processing* foi

efetivo em reduzir o impacto da expansão da topologia (número de LS) sem alterar de forma substancial o modelo de escalabilidade já consolidado para o preenchimento dessas topologias com portas e instâncias.

6. Conclusões

O software discutido na tese é o OVN, que juntamente com OVS e OpenStack, se tornaram o padrão de código aberto para gerenciamento de computação em nuvem. Utilizado por diversas empresas ao redor do mundo, recebe contribuições que otimizam fatores específicos e desenvolvem novas funcionalidades. Os elementos da arquitetura do OVN foram apresentados e questões de escalabilidade foram apresentadas, tanto nos bancos de dados quanto nos componentes que recebem dados dos bancos e fazem traduções a partir deles.

A capacidade de provisionamento de um volume grande de componentes virtuais e o tempo para as mudanças serem refletidas impactam desenvolvedores, pesquisadores e clientes. O comportamento de um dos elementos do ovn (*northd*) foi mensurado na criação de componentes lógicos frequentes (redes, sub redes e portas).

A avaliação comparativa entre a versão estável 24.09 (2024) e a versão em desenvolvimento (25.09) revelou comportamentos distintos dependendo do cenário de teste. Na criação de portas, ambas as versões apresentaram desempenho semelhante; um resultado esperado, visto que o código base responsável por essa tarefa não sofreu alterações estruturais significativas no período.

Em contrapartida, o fluxo de criação de redes (Logical Switches) passou por modificações profundas no código-fonte. A análise concentrou-se nestas alterações para validar a hipótese de melhoria de desempenho, permitindo quantificar o impacto direto da adoção do Processamento Incremental (I-P) na escalabilidade do sistema.

Existem diversos aprofundamentos que podem estender o conteúdo desse trabalho. Enquanto essa tese mensurou apenas a criação de componentes lógicos do ponto de vista do *northd*, outras ações como deleção, atualização podem ser avaliados. A abordagem de processamento incremental também está presente no *ovn-controller* que deve enfrentar desafios semelhantes.

Referência Bibliográfica:

DALTON, Michael et al. Andromeda: performance, isolation, and velocity at scale in cloud network virtualization. In: USENIX SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION (NSDI), 15., 2018, Renton. **Proceedings[...]**. Berkeley: USENIX Association, 2018. p. 373-387. Disponível em: https://www.usenix.org/conference/nsdi18/presentation/dalton. Acesso em: 07 dez. 2025.

PFAFF, Ben et al. The design and implementation of Open vSwitch. In: USENIX SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION (NSDI), 12., 2015, Oakland. **Proceedings[...]**. Berkeley: USENIX Association, 2015. p. 117-130. Disponível em: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff. Acesso em: 07 dez. 2025.

OPEN VIRTUAL NETWORK. Changelog v24.03.0. 2024. Disponível em: https://www.ovn.org/en/releases/changelog_v24.03.0/. Acesso em: 08 dez. 2025.

ZHOU, Han; **SIDDIQUE**, Numan. OVN-Northd Incremental Processing in C: Methodology, Achievements, and Challenges. In: OPEN VSWITCH AND OVN FALL CONFERENCE, 2023. **Proceedings[...]**. [S.l.: s.n.], 2023. Disponível em: https://www.youtube.com/watch?v=pC4EGTg8KQI. Acesso em: 08 dez. 2025.

ZHOU, Han. OVN Controller Incremental Processing. In: OPEN VSWITCH FALL CONFERENCE, 2018. **Apresentação de slides**. [S.l.: s.n.], 2018. Disponível em: https://www.openvswitch.org/support/ovscon2018/6/1350-zhou.pptx. Acesso em: 08 dez. 2025.

MAXIMETS, Ilya. [ovs-dev,v2] ovsdb-raft: Transfer leadership before creating a snapshot**.** Open vSwitch Mailing List Archive, 06 maio 2021. Disponível em: https://patchwork.ozlabs.org/project/openvswitch/patch/20210506124731.3599531-1-i.maximets@ovn.org/. Acesso em: 08 dez. 2025.

OPEN VIRTUAL NETWORK. ovn-architecture: Open Virtual Network architecture. Linux Programmer's Manual. 2025. Disponível em: https://man7.org/linux/man-pages/man7/ovn-architecture.7.html. Acesso em: 08 dez. 2025.

Michelson, Mark. Performance improvements in OVN: Past and future. Red Hat Developer, 02 jan. 2019. Disponível em: https://developers.redhat.com/blog/2019/01/02/performance-improvements-in-ovn-past-and-future. Acesso em: 08 dez. 2025.

MAXIMETS, Ilya. OVSDB performance updates '22: Testing with ovn-heater. In: OPEN VSWITCH AND OVN FALL CONFERENCE, 2022. **Apresentação (Slides)**. Disponível em: https://www.openvswitch.org/support/ovscon2022/slides/ovsdb-perf-updates-22-ovn-heater.pdf. Acesso em: 08 dez. 2025.

SIDDIQUE, Numan. Deploying multi chassis OVN using docker in docker. In: OPEN VSWITCH FALL CONFERENCE, 2019, Westford, MA. **Apresentação (Slides)**. Disponível em: https://www.openvswitch.org/support/ovscon2019/day2/1319-siddique.pdf. Acesso em: 08 dez. 2025.

BIANCONI, Lorenzo. northd: Convert datapath array into vector. 2024c. Repositório oficial do OVN no GitHub. Commit f22a005. Disponível em: https://github.com/ovn-org/ovn/commit/f22a005f89353f19bdc3548b7eae471dcc985627. Acesso em: 11 dez. 2025.

BIANCONI, Lorenzo. northd: I-P for logical switch creation/deletion in en_ls_stateful engine node. 2024e. Repositório oficial do OVN no GitHub. Commit 48e1736. Disponível em: https://github.com/ovn-org/ovn/commit/48e1736f89c6e488e21472bffdb4b80a72719148. Acesso em: 11 dez. 2025.

BIANCONI, Lorenzo. northd: I-P for logical switch creation/deletion in en_northd. 2024d. Repositório oficial do OVN no GitHub. Commit 06e2c1b. Disponível em: https://github.com/ovn-org/ovn/commit/06e2c1bf0ce74524273c1fab8f628af554d61ffc. Acesso em: 11 dez. 2025.

BIANCONI, Lorenzo. northd: Introduce dynamic_bitmap struct. 2024a. Repositório oficial do OVN no GitHub. Commit b0ca75c. Disponível em: https://github.com/ovn-org/ovn/commit/b0ca75c1bfb00a2aa0cdd9bd2add7ce66179f72d. Acesso em: 11 dez. 2025.

BIANCONI, Lorenzo. northd: Reallocate LB nb_lr_map/nb_ls_map if needed. 2024b. Repositório oficial do OVN no GitHub. Commit 3229f77. Disponível em: https://github.com/ovn-org/ovn/commit/3229f776374c3f30da4594b823d4716c32c4c282. Acesso em: 11 dez. 2025.