NEA: Arquitetura de elemento de rede SDN com suporte a MAC definido pela aplicação.

Diego Nunes Molinos

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2022

Diego Nunes Molinos

NEA: Arquitetura de elemento de rede SDN com suporte a MAC definido pela aplicação.

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Pedro Frosi Rosa
Coorientador: Marcelo Barros de Almeida

Uberlândia
2022

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

M723 Molinos, Diego Nunes, 1986-
2022 NEA [recurso eletrônico] : arquitetura de elemento de rede SDN com suporte a MAC definido pela aplicação / Diego Nunes Molinos. - 2022.

Orientador: Pedro Frosi Rosa.
Coorientador: Marcelo Barros de Almeida.
Tese (Doutorado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.te.2022.243
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Rosa, Pedro Frosi,1959-, (Orient.). II. Almeida, Marcelo Barros de,1972-, (Coorient.). III. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. IV. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Tese, 5/2022, PPGCO				
Data:	23 de março de 2022	Hora de início:	08:00	Hora de encerramento:	11:50
Matrícula do Discente:	11613CCP006				
Nome do Discente:	Diego Nunes Molinos				
Título do Trabalho:	NEA: Arquitetura de elemento de rede SDN com suporte a MAC definido pela aplicação				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se, por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Rafael Pasquini - FACOM/UFU, Rodrigo Sanches Miani - FACOM/UFU, Edward David Moreno Ordonez - DCOMP/UFS, Eduardo Coelho Cerqueira - Instituto de Tecnologia/UFPA, Marcelo Barros de Almeida - FEELT/UFU (Coorientador) e Pedro Frosi Rosa - FACOM/UFU orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Edward David Moreno Ordonez - São Cristóvão/SE; Eduardo Coelho Cerqueira - Belém/PA; Rafael Pasquini, Rodrigo Sanches Miani, Marcelo Barros de Almeida e Pedro Frosi Rosa - Uberlândia/MG. O discente participou da cidade de Uberlândia/MG.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Pedro Frosi Rosa, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Doutor.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Rafael Pasquini**, **Professor(a) do Magistério Superior**, em 06/04/2022, às 08:42, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marcelo Barros de Almeida**, **Professor(a) do Magistério Superior**, em 06/04/2022, às 09:31, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Edward David Moreno Ordonez**, **Usuário Externo**, em 06/04/2022, às 09:43, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Eduardo Coelho Cerqueira**, **Usuário Externo**, em 06/04/2022, às 10:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Rodrigo Sanches Miani**, **Professor(a) do Magistério Superior**, em 06/04/2022, às 11:56, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Pedro Frosi Rosa**, **Professor(a) do Magistério Superior**, em 16/05/2022, às 10:34, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **3463624** e o código CRC **CA0B8A63**.

Referência: Processo nº 23117.019501/2022-71 SEI nº 3463624

Este trabalho é dedicado à minha família e amigos,
meus maiores incentivadores.

Agradecimentos

Primeiramente gostaria de agradecer a Deus, pelas oportunidades concedidas. Aos meus pais, que sempre se doaram e renunciaram aos seus sonhos para que eu pudesse realizar os meus. Esta conquista é nossa. A minha esposa Radharani, pela paciência e compreensão com minha ausência durante esta caminhada, pelo respeito, simplicidade e amor durante esses longos anos. Aos irmãos que a pesquisa me trouxe, verdadeiros amigos, que mesmo diante de inúmeras adversidades não perdem o amor pela pesquisa, são tantos, mas gostaria de mencionar o meu irmão Romerson – que esteve sempre comigo desde o início da minha jornada, Marcelo e Natal – pelo inestimável apoio na reta final da escrita, Acrísio, Pedro Damaso, Maurício e Daniel. A CAPES pelo apoio financeiro. Um agradecimento aos professores Rodrigo Sanches Miani, Edward David Moreno Ordoñez e Marcelo Barros de Almeida, pelo apoio na construção deste trabalho com valiosas sugestões durante a qualificação. Ao Professor Daniel Gomes Mesquita, por acreditar em mim, pelos ensinamentos, pela paciência e amizade. Ao Professor Flávio de Oliveira Silva, pelos conselhos, ensinamentos e suporte nessa caminhada. Ao estimado Professor Pedro Frosi Rosa, com muito carinho e admiração, gostaria de expressar meu profundo agradecimento, pela confiança, pelos diversos conselhos, pela sabedoria transmitida, pela humildade e carinho que tem com todos os estudantes e professores, pelo exemplo de ser humano que és, obrigado por me apoiar.

"A ciência é muito mais do que um corpo de conhecimento. É uma maneira de pensar."
(Carl Sagan)

Resumo

A Internet se tornou um fator limitante para sua própria evolução, uma vez que aplicações estão sendo desenvolvidas sobre uma nova perspectiva de utilização da rede, exigindo mais Qualidade de Serviço (QoS) e Qualidade de Experiência do Usuário (QoE). Abordagens que objetivam redesenhar a arquitetura, por exemplo, Redes Definidas por Software (SDN), têm-se tornado populares no campo de estudo das redes de computadores, como tentativa de minimizar as limitações vivenciadas pelo TCP/IP. Em tese, as SDN naturalmente deixam a cargo do plano de controle toda flexibilidade e programabilidade da rede, negligenciando a capacidade do plano de dados, em atuar na melhoria da QoS e QoE percebida por usuários. As tecnologias comumente adotadas nas soluções para o plano de dados, em sua maioria, recorrem a plataformas legadas para materializar as políticas de encaminhamento. Embora seja possível (re)configurar o comportamento de *switches*, o *Media Access Control (MAC)* é único por equipamento, não sendo possível tratar requisitos de aplicações. Diante do exposto, este trabalho objetiva o desenvolvimento de uma arquitetura de elemento de rede, incluindo um protótipo de equipamento, (*Switch*), com o MAC orientado pelos requisitos de aplicações. O *Switch* NEA quando comparado a outras abordagens similares, possui a capacidade de expor, através de uma fina granularidade, a lógica referente às políticas de encaminhamento de baixo nível ao plano de controle por meio de um módulo orquestrador, permitindo (re)programação de forma sistemática e rápida. Os resultados obtidos mostram a capacidade do *Switch* NEA em atuar na melhoria da Quality of Service (QoS), através do controle acurado das regras de encaminhamento no plano de dados e da habilidade de manipular diversos fluxos de dados pela mesma porta física, atribuindo diferentes parâmetros (prioridade e largura de banda) para diferentes fluxos. Ressalta-se também a possibilidade de adição de novas funcionalidades e suporte a novos protocolos.

Palavras-chave: Internet do Futuro. SDN. MAC. Elemento de Rede. Switch Linux-Based. ETArch. QoS. QoE.

Abstract

The Internet has become a limiting factor for its evolution. Applications are being developed from a new perspective, demanding Quality of Service (QoS) and Quality of Experience (QoE). Approaches that aim to redesign the architecture, e.g., Software Defined Networks (SDN), have become popular in the computer networks field to minimize the TCP/IP limitations. In theory, SDN Networks naturally leave all flexibility and programmability of the network to the control plane, neglecting the ability of the data plane to provide QoS and QoE. As a result, the technologies commonly used in data plane solutions often resort to legacy platforms to materialize the forwarding policies. Although those technologies enable reconfiguring the network element's behavior, the MAC remains the same. This work aims at the specification and development of a network element architecture (NEA) and a prototype of a Switch with MAC driven by application for SDN networks. The NEA architecture proposed in this thesis, compared with other similar approaches, can expose, through the fine-grained, the logic referring to low-level forwarding policies to the control plane through an orchestrator module, allowing (re)programming in a systematic and fast way. The results obtained show the ability of *Switch* NEA to improve QoS, through the accurate control of forwarding rules in the data plane and the ability to handle different data flows through the same physical port, assigning parameters like priority and bandwidth to other streams. It is also worth mentioning the possibility of adding new functionalities and support for new protocols.

Keywords: Future Internet. SDN. MAC. Network Element. Switch Linux-Based. ETArch.QoS. QoE.

Lista de ilustrações

Lista de tabelas

Lista de siglas

ASIC Application Specific Integrated Circuit

API Application Programming Interface

ACL Access Control List

ARM Advanced RISC Machine

ARP Address Resolution Protocol

BER Bit Error Rate

CAM Content Addressable Memory

CE Control Element

CBQ Class-Based Queueing

CSMA Carrier Sense Multiple Access

CRC Cyclic Redundancy Check

DTS Domain Title Service

DTSA Domain Title Service Agent

DDoS Distributed Denial of Service

DSMARK Diff-Serv Marker

ETArch Entity Title Architecture

ETSCP Etarch Switch Control Protocol

FDB Forwarding Data Base

FPGA Field Programmable Gate Array

FIFO First in, First out

FSM Finite State Machine

FE Forwarding Element

ForCES Forwarding and Control Element Separation

GPON gigabit passive optical networks

HTB Hierarquical Token Bucket

IP Internet Protocol

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

IPFIX Internet Protocol Flow Information Export

LAN Local Area Network

LFB Logical Functional Block

LACP Link Aggregation Control Protocol

MAC Media Access Control

MIPS Microprocessor Without Interlocked Pipeline Stages

MTU Maximum Transmission Unit

NAT Network Address Translation

NIC Network Card Interface

NFV Network Function Virtualization

ONF Open Networking Foundation

OSI Open System Interconnection

OCS Optical Circuit Switching

OPS Optical Packet Switching

OVS Open vSwitch

OUI Organizationally Unique Identifier

P2P Peer-To-Peer

PIF Protocol Independent Forwarding

P4 Programming Protocol-Independent Packet Processors

PISCES Programmable Protocol Independent Software Switch

PRIO Priority Queue

QoE Quality of Experience

QoS Quality of Service

QSFP Quad Small Form Factor Pluggable

RSPAN Remote Switch Port Analyser

RED Random Early Detection

SDK Software Development Kit

SDN Software Defined Network

SCA Side Channel Attacks

SoC System on a Chip

STP Spanning Tree Protocol

SFQ Stochastic Fair Queueing

SKB Socket Buffer

SD Standard Definition

TC Traffic Control

TCAM Ternary Content-Addressable Memory

TCP Transmission Control Protocol

TBF Token Bucket Flow

UDP User Datagram Protocol

UE User Equipment

VLAN Virtual Local Area Network

VXLAN Virtual eXtensible Local Area Network

VoIP Voice over IP

VNF Virtual Network Functions

VFP Virtual Filtering Plataform

Sumário

Capítulo 1

Introdução

Naturalmente, usuários utilizam a Internet para consumir algum tipo de serviço, podendo ser aplicações *streaming*, redes sociais, tarefas pessoais ou profissionais, e o fato é que, de modo geral, esses usuários não se preocupam com aspectos da infraestrutura da rede e, comumente, conectam-se através de vários dispositivos. De acordo com Oliveira (2019), cada dispositivo conectado, seja qualquer estação de trabalho, *laptop*, *tablet*, *smartphone* ou até mesmo um *smartwatch*, pode hospedar diversas aplicações com diversos requisitos de comunicação.

Para aplicações multimídia (*Streaming ao vivo ou armazenado*), para as quais há requisitos específicos em relação a tempo e perdas de quadros, que são sensíveis a variação de atrasos (*Delay Sensitive/Interarrival Jitter*), não obstante utilizarem *buffers* para armazenamento de dados, é oportuno lembrar que, eventuais perdas podem comprometer a qualidade, degradando a *Quality of Service (QoS)* e *Quality of Experience (QoE)*.

Embora requisitos tais como tolerância a falhas, mobilidade, segurança, controle de banda, QoS e QoE sejam exigidos por aplicações contemporâneas, assume-se que a rede se torna responsável por materializá-los (OLIVEIRA, 2019). Isto contribui para os problemas estruturais e de expansão da Internet, e, de acordo com Bezahaf et al. (2020), são problemas que dificilmente serão resolvidos apenas com alterações na arquitetura de camadas ou protocolos.

A Internet, nas últimas décadas, não passou por significativas modificações nas camadas de Transporte e Rede, mantendo os principais conceitos e protocolos, propostos há mais de 50 anos (BEZAHAF et al., 2020; SILVA et al., 2012). Serviços e aplicações, tais como *Voice over IP (VoIP)*, *Streaming*, Vídeo Conferências e *Peer-To-Peer (P2P)*, contribuíram significantemente para a mudança no contexto de uso da rede (OLIVEIRA, 2019). Dentro deste contexto cita-se algumas aplicações tais como Realidade Virtual e Hologramas que tendem a exigir muitos recursos da rede.

Nota-se que estes novos requisitos tiveram um impacto importante nas camadas Física e de Aplicação, sendo que as outras camadas não acompanharam os avanços, sendo possível observar que algumas necessidades de QoS, QoE, segurança e mobilidade, não

são atendidas pelas camadas intermediárias (OLIVEIRA, 2019).

Em resposta às limitações da arquitetura Internet, existe o paradigma de Internet do Futuro, constituído de abordagens *Clean Slate* e *Evolutionary*, que respectivamente, propõem uma estrutura totalmente nova ou consideram adaptações na atual arquitetura. De acordo com Oliveira (2019), dentro deste paradigma, algumas propostas aparecem como arquiteturas de próxima geração, sendo elas XIA (ANAND et al., 2011), RINA (DAY; MATTA; MATTAR, 2008), SONATE (MÜLLER; REUTHER, 2009), NENA (MARTIN; VöLKER; ZITTERBART, 2011), *Mobility First* (SESKAR et al., 2011), NEBULA (ANDERSON et al., 2013), NDN (ZHANG et al., 2010), ETArch (SILVA, 2013b) e algumas mais recentes como: Novagenesis (ALBERTI et al., 2017), multiFIA (SEOK et al., 2017), FlexNGIA (ZHANI; ELBAKOURY, 2020) e Fi-SWoT (PETRAKIS; KONTOCHRISTOS; RASTSINSKAGIA, 2021).

Nota-se que as propostas citadas, mesmo com diferentes abordagens, convergem para a tratativa de flexibilizar o uso dos recursos de rede, de foma que os requisitos das comunicações sejam classificados em diferentes níveis da rede. Conforme Campbell et al. (1999), uma rede programável se torna diferente das outras redes pelo fato de poder ser programada a partir de um conjunto mínimo de *Application Programming Interfaces (APIs)*, onde cada conjunto pode ser composto de uma gama de serviços.

Estudos como Pan, Paul e Jain (2011) e Bezahaf et al. (2020) afirmam que as soluções desenvolvidas dentro do paradigma de Internet do Futuro não devem possuir a premissa principal de corrigir os problemas da arquitetura atual através de melhoramentos pontuais. Portanto, políticas de encaminhamento, mobilidade e segurança, devem ser uma preocupação tratada no projeto da arquitetura da rede e não isoladamente (OLIVEIRA, 2019).

Em resposta às dificuldades apresentadas pela arquitetura Internet, surgem as *SDNs*, onde a premissa principal é a separação do plano de controle do plano de dados da rede (IETF, 2015; FOUNDATION, 2017). Na filosofia SDN não é previsto complexidade no nível do plano de dados, de forma que qualquer tratativa de maior complexidade é encaminhada para o plano de controle (BREBNER, 2015). Em tese, o plano de dados é constituído por elementos de rede simples para encaminhamento de primitivas.

É possível observar que soluções em SDNs assumem que existe conectividade no nível do Enlace, independentemente de aspectos da infraestrutura e das capacidades dos (*Switches*) (OLIVEIRA, 2019). Essa conectividade vem sendo considerada satisfatória para o funcionamento da rede e, com isto, o potencial de melhoria no plano de dados se encontra inexplorado, negligenciando completamente a possibilidade de atuação deste nível no desempenho da rede e na melhoria de QoS (OLIVEIRA, 2019).

De modo geral, o plano de controle se estabelece no nível de *software* através do controlador, enquanto o plano de dados se estabelece no nível de *hardware* através do *Switch* (ONF, 2014; OLIVEIRA, 2019). No entanto, ressalta-se que *Switches* virtuais,

cada vez mais comuns em estruturas de *cloud computing*, são materializados no nível de *software*.

Em decorrência da separação dos planos de dados e de controle, um distanciamento natural nas funções de gerenciamento da rede acaba se instituindo, fazendo com que o plano de dados se comporte como um único tecido granular da rede (OLIVEIRA, 2019). Este comportamento contribui para que todas as funções nobres, tais como atualização e criação de regras de encaminhamento, estejam alocadas no plano de controle (BREBNER, 2015). O encaminhamento de primitivas no plano de dados é de inteira responsabilidade do plano de controle, sendo que, no plano de dados, o *Switch* é responsável por materializar a comunicação entre entidades (KALJIC et al., 2019; OLIVEIRA, 2019).

Sob a óptica de soluções para o plano de dados nas SDNs, a técnica OpenFlow vem sendo largamente utilizada. Além de implementar a filosofia das redes SDN no que tange a separação dos planos, o OpenFlow oferece a capacidade de configuração de fluxos e políticas de encaminhamento predefinidas (*match actions*). Fato é que, tanto o OpenFlow quanto outras tecnologias, tais como *Programming Protocol-Independent Packet Processors (P4)* e *Open vSwitch (OVS)*, apresentam avanços na gestão de encaminhamento, ofertando a possibilidade de instituir vários fluxos através da mesma porta física e o estabelecimento de parâmetros (*rate* e *burst*) por fluxo, contudo, existem limitações nas tratativas de tráfego orientado por aplicações, principalmente em relação ao ajuste fino de parâmetros tais como prioridade, *burst* e controle de banda (*rate*) definido por aplicação, além de que, todo o tráfego acaba-se desaguando sobre equipamentos onde o *Media Access Control (MAC)* é único.

Contextualizando o MAC único, por exemplo, os *Switches Ethernet* fazem uso do protocolo *Carrier Sense Multiple Access (CSMA)*, que é responsável por organizar a forma que os *frames* são inseridos no canal de transmissão (TANENBAUM, 1996). O protocolo CSMA utiliza uma estrutura em *First in, First out (FIFO)* para controlar a entrega de primitivas no canal, não permitindo qualquer tipo de ordenação nem escalonamento dos *frames*, independente de aplicação. Nesse sentido, pensar em alterar diretamente a forma de atuação do MAC para atender necessidades específicas de QoS de aplicações não é uma tarefa trivial e ainda exige modificações nas *Network Card Interfaces (NICs)*, protocolos e equipamentos de rede. Contudo, atuar na manipulação de parâmetros que impactam diretamente o MAC através de uma camada de *software* torna-se uma abordagem interessante para alcançar flexibilidade e programabilidade no baixo nível da rede.

De forma geral, as soluções utilizadas no plano de dados em SDNs para resolver as ações de encaminhamento foram projetadas para serem simples, observa-se que requisitos como QoS e QoE não foram aprofundados no projeto dessas soluções e acabam dependendo de suporte de plataformas legadas, que naturalmente são limitadas (KALJIC et al., 2019).

De acordo com Kaljic et al. (2019), Michel et al. (2021), o desafio face a planos de dados programáveis está na capacidade de como especificar e reconfigurar a arquitetura

de baixo nível . A configuração/(re)configuração dos chips presentes nos comutadores de dados não é nada trivial e carece de melhorias para se tornar expressiva e flexível frente às necessidades de um plano de dados programável (MICHEL et al., 2021). Isto em parte justifica a adoção por soluções centradas em *software*, onde é possível alcançar flexibilidade através de linguagens de programação de alto nível e do uso de *APIs* para a confecção de uma solução transparente e escalável.

Ante o exposto, como aperfeiçoamento do plano de dados no que tange à melhoria de QoS de usuários, a proposta desta tese é o desenvolvimento de um protótipo de *Switch* baseado em uma arquitetura de elemento de rede que possui a capacidade de manusear e orquestrar políticas de encaminhamento a partir dos requisitos de QoS de aplicações com impacto direto no MAC. Este novo *Switch* institui-se como uma proposta mais flexível e programável em comparação com as soluções atuais com a capacidade de praticar um ajuste fino nos parâmetros de QoS com impacto direto na forma como os *frames* são entregues nas filas de saída. Entende-se por parâmetros valores referente à vazão (largura de banda), prioridade (priorização de tráfego) e controle de *burst* orientado por aplicação e não por fluxo ou porta física.

1.1 Motivação

Não desconstruindo a filosofia em relação à separação das funções da rede em SDN, onde o plano de dados opera sob a supervisão do plano de controle, mas, permitir que o elemento de rede possua autonomia na inspeção e instituição de tarefas inerentes ao plano de dados, são premissas que visam a melhoria de QoS da rede e ainda desoneram o controlador.

Sob a perspectiva do conceito de disponibilidade, concentrar grande parte da gestão da rede no plano de controle, sob a responsabilidade do controlador, naturalmente institui-se um gargalo na rede, visto que, o controlador pode apresentar falhas e travamentos (OLIVEIRA, 2019). Mesmo em propostas que preveem a utilização de agentes distribuídos (tal como *Entity Title Architecture (ETArch)*), uma falha no controlador se propaga para todos os elementos da rede, podendo ocasionar um travamento geral. Diante desse contexto, pensar no *Switch* como um elemento autônomo com capacidade de decisão no plano de dados é uma abordagem que contribui para melhoria da gestão da rede.

Diante da perspectiva da QoS de usuários, o MAC praticado pelos *Switches* atuais, inclusive os utilizados em SDN, não conseguem materializar os requisitos das aplicações atuais e futuras devido a dificuldade em expressar as demandas do controlador no baixo nível.

As aplicações de modo geral possuem diferentes requisitos de comunicações, por exemplo, nas aplicações VoIP a taxa de amostragem utilizada no sinal de voz impacta diretamente no *Maximum Transmission Unit (MTU)*, que pode ocasionar alterações no controle

de banda da aplicação para garantir a QoS. De acordo com Mushtaq, Shahid e Fowler (2012), ainda em relação às aplicações VoIP, os atrasos devem ser algo entre $45ms$ e $90ms$ em um baixo *jitter*, sendo que, atrasos superiores à $400ms$ prejudicam a QoS da aplicação. Embora diferentes requisitos sejam definidos pelas aplicações, a rede acaba-se tornando responsável por materializá-los, no contexto de redes SDNs, o *Switch* é o principal elemento desta infraestrutura.

Diante deste contexto, pensar em atribuir ao *Switch* a capacidade de manipular variáveis com impacto direto no MAC, ofertando melhorias na qualidade das comunicações, através de uma arquitetura que permite materializar as ações de encaminhamento com uma granularidade mais fina[1], torna-se uma abordagem cativante para atender os requisitos de QoS de usuários.

1.2 Hipótese

O MAC (programável) orientado pelos requisitos especificados por aplicações é capaz de atender os requisitos de QoS de usuários.

1.3 Objetivos

Este trabalho visa o desenvolvimento de uma arquitetura para elemento de rede com o MAC definido pela aplicação para redes SDN. Pretende-se especificar e projetar os mecanismos de encaminhamento do elemento de rede (*Switch*) através de uma abordagem *Linux-based*, totalmente centrada em *software*, demonstrando ser possível alcançar flexibilidade no baixo nível da rede através da manipulação e orquestração do MAC já presente nas *NICs*.

Todo o processo é guiado por objetivos específicos que conduzem a estratégia de desenvolvimento e estão elencados como se segue:

1. Estabelecer um conjunto básico e funcional de requisitos de elemento de rede para SDN;

2. Estabelecer um conjunto de requisitos funcionais de elemento de rede a partir da análise do Modelo de Títulos para o estudo de caso;

3. Definir o formato de comunicação entre controlador e elementos de rede;

4. Modelar e implementar uma arquitetura para o elemento de rede com o MAC orientado pelas aplicações;

[1] Neste contexto granularidade fina significa atuar com mais profundidade nas ações de encaminhamento, permitindo gerenciar independentemente vários fluxos por porta, com diferentes requisitos associados por fluxo.

5. Especificar e implementar um protótipo de *Switch* baseado na arquitetura proposta; e

6. Demonstrar a utilização do protótipo utilizando uma arquitetura SDN.

1.4 Contribuições

Nas SDNs, a pesquisa é insuficientemente focada no plano de Dados, as soluções assumem que a infraestrutura de rede irá oferecer o melhor cenário para satisfazer os requisitos de QoS de aplicações (KALJIC et al., 2019; MICHEL et al., 2021). Considerando que as aplicações suscitam novas expectativas, os atuais *Switches* SDNs apresentam dificuldades em garantir QoS pois, possuem limitações em relação à instituição e atualização das capacidades dos fluxos em tempo real, e sem qualquer atuação sobre o MAC. A primeira contribuição deste trabalho se configura no desenvolvimento de um *Switch* SDN baseado em uma arquitetura com o MAC orientado por aplicações que permite manipular parâmetros, tais como: prioridade, controle de banda e *burst*, com uma granularidade mais fina quando comparada à outras soluções, objetivando a garantia da QoS de usuários.

De acordo com o estado da arte, até o momento em que o levantamento sistemático foi realizado para esta tese, não foram encontrados trabalhos que objetivam o desenvolvimento de uma arquitetura para elemento de rede com o MAC (programável) orientado pelos requisitos de aplicações para garantir QoS de usuários. Sendo assim, esta tese se institui como um aperfeiçoamento do plano de dados no que tange à melhoria da QoS de usuários em redes SDN.

A presente tese deixa ainda como contribuição acadêmica o capítulo 2 que se comporta como um arcabouço conceitual para novos estudos focados em explorar a QoS no plano de dados em redes SDN. Destaca-se a contribuição para a indústria nacional, onde toda a modelagem, implementação e código, é direcionada à construção de um produto. Toda a experiência e conhecimento adquirido nesta tese fica como contribuição do trabalho, além do produto final que poderá ser melhorado (aspectos de desempenho) para alcançar aplicação na indústria ou, na comunidade *open-source*.

Além das contribuições citadas, ainda será apresentado no capítulo 8 a produção bibliográfica produzida durante o desenvolvimento desta tese.

1.5 Organização do Documento

Este documento está estruturado da seguinte forma:

1. O Capítulo 1 contextualiza a problemática deste trabalho através de um breve relato das soluções existentes. Neste capítulo também é apresentado os objetivo e as contribuições originais deste trabalho.

2. O Capítulo 2 apresenta o aparato conceitual desta tese, descrevendo o paradigma de Internet do Futuro e também as limitações da atual arquitetura. Ainda neste capítulo é explanado os mecanismos gerais de operação de *Switches* L2. Por último, os principais trabalhos correlatos preconizando as soluções flexíveis no plano de dados em redes SDN.

3. O Capítulo 3 descreve o método deste trabalho, discorrendo detalhadamente todos os passos utilizados na construção do conhecimento.

4. O Capítulo 4 apresenta o levantamento e refinamento de requisitos utilizados para especificação da arquitetura NEA.

5. O Capítulo 5 apresenta detalhadamente a especificação da arquitetura NEA, inicialmente com a *Finite State Machine (FSM)* referente a modelagem da arquitetura, suporte oferecido pelo Kernel do Linux e desenvolvimento da aplicação NEA-SWITCHD.

6. O Capítulo 6 detalha o estudo de caso desta tese. Neste capítulo é apresentado a modelagem e implementação da arquitetura NEA para comportar as características da ETArch.

7. O capítulo 7 discorre sobre a experimentação e resultados desta tese. Este capítulo é dividido em dois momentos, o primeiro referente aos testes funcionais do protótipo e o segundo momento relacionado a testes realizados dentro do estudo de caso.

8. O Capítulo 8, por fim, traz as considerações finais deste trabalho e as propostas futuras.

Capítulo 2

Revisão da Literatura

Este capítulo tem dois enfoques, sendo que na Seção 2.1, é tratado o embasamento conceitual desta tese, e na Seção 2.2, são apresentados trabalhos correlatos com foco em programabilidade e flexibilidade no plano de dados em redes SDN.

2.1 Fundamentação Teórica

Alguns tópicos são essenciais para uma contextualização apropriada desta tese. Primeiramente, cabe entender o paradigma de Internet do Futuro, em especial a filosofia SDN, que é a referência para os mecanismos de encaminhamento do comutador (*Switch*) apresentado nesta proposta. A compreensão dos conceitos básicos de comutadores de dados L2 e o suporte do sistema operacional Linux. Por último, a análise dos trabalhos correlatos com foco na flexibilidade e programabilidade de elementos de rede com vistas a QoS e QoE de usuários.

2.1.1 Internet do Futuro

A Internet como conhecemos foi desenvolvida na década de 70, ainda que, uma grande parte dos requisitos tenham sido instituídos no momento de sua concepção, a rede vem experimentando novas funcionalidades que constantemente exigem correções e alterações em sua estrutura (CLARK et al., 2004; FELDMANN, 2007).

Ao avaliar a evolução da Internet, observa-se o quanto a utilização dos protocolos TCP/IP, princípio da transparência e da comunicação fim-a-fim, auxiliou no avanço da arquitetura. Todavia, atualmente, sua arquitetura limita o progresso da Internet (OLIVEIRA, 2019; PEREIRA, 2012b).

A Internet foi projetada com foco em heterogeneidade na camada de Rede, com núcleo simples e a inteligência alocada nos *hosts*, e estes fatores contribuíram para a sobrecarga de funcionalidade nos sistemas finais e também na dificuldade de implantação de melhorias na gerência da rede (PEREIRA, 2012b). Clark et al. (1991), em 1991 já citavam proble-

mas relacionados a roteamento, endereçamento, segurança e a dificuldade da Internet em prover QoS.

Com relação ao endereçamento IP, além da falta de interoperabilidade entre *Internet Protocol Version 4 (IPv4)* e *Internet Protocol Version 6 (IPv6)*, a sobrecarga semântica de identificação e localização dificulta a implantação de soluções que exijam mobilidade, pois não é possível que dispositivos ativos realizem a transição entre pontos de acessos sem a necessidade de (re)conexão (MOREIRA et al., 2009).

De acordo com Moreira et al. (2009), sempre houve consenso de que requisitos de segurança deveriam estar presentes nos *hosts* da rede. Em sistemas distribuídos, por questões inerentes à arquitetura, torna-se necessário que o núcleo da rede os satisfaça. De acordo com Goel, Williams e Dincelli (2017), os problemas de segurança não se restringem apenas a vulnerabilidades dos usuários, mas também englobam aspectos da infraestrutura da rede (GOEL; WILLIAMS; DINCELLI, 2017).

Conforme Hakiri et al. (2014), atingir requisitos de aplicações não é algo trivial, pois, os protocolos, naturalmente, tendem a ser definidos isoladamente e se destinam a resolver problemas específicos. Além disso, a instituição de novos serviços resulta em reconfiguração de milhares de dispositivos e protocolos de rede, dificultando cada vez a aplicação de novas políticas de QoS e QoE (HAKIRI et al., 2014).

O paradigma de Internet do Futuro visa minimizar os principais problemas vivenciados na Internet através de duas abordagens (REXFORD; DOVROLIS, 2010; GONÇALVES et al., 2014). A primeira delas, apresenta uma visão de substituição em totalidade da atual arquitetura, conhecida como abordagem revolucionária ou *Clean Slate*. De acordo com Rexford e Dovrolis (2010), a abordagem *Clean Slate* apresenta vantagens, por não se limitar à especificação da arquitetura Internet, porém, instituir uma nova arquitetura requer grandes esforços. Em contrapartida, a abordagem conhecida como evolucionária visa aprimorar o que já existe na Internet, tornando menos complexo a instituição de novos serviços, porém, significa trabalhar em um ambiente totalmente limitado (REXFORD; DOVROLIS, 2010).

Entre as abordagens mais expressivas no campo de Internet do Futuro existem as Redes Definidas por Software (SDN[1]) que apresentam uma forma inovadora de gerir as redes de computadores (PEREIRA, 2012a; KALJIC et al., 2019). As tecnologias SDN visam criar uma interface lógica bem definida para o controle da rede, tendo esta camada a função de abstrair a rede e fornecer diversos serviços que permitam reconfigurabilidade (*Stateless*) em tempo de execução (OLIVEIRA, 2019).

2.1.2 Redes Definidas por Software - SDN

De acordo com Kaljic et al. (2019), Oliveira (2019), a singularidade da abordagem SDN reside no fato de fornecer programabilidade para rede através do desacoplamento

[1] *Software Defined Network*

dos planos de Controle e Dados, já no projeto arquitetônico da rede. Em suma, SDN visa ofertar redes compostas de dispositivos simples e flexíveis ao invés de dispositivos complexos (XIA et al., 2014). Na Figura 1 é possível observar a separação dos planos de Dados e Controle nas SDNs, juntamente com alguns aspectos de infraestrutura e aplicação.

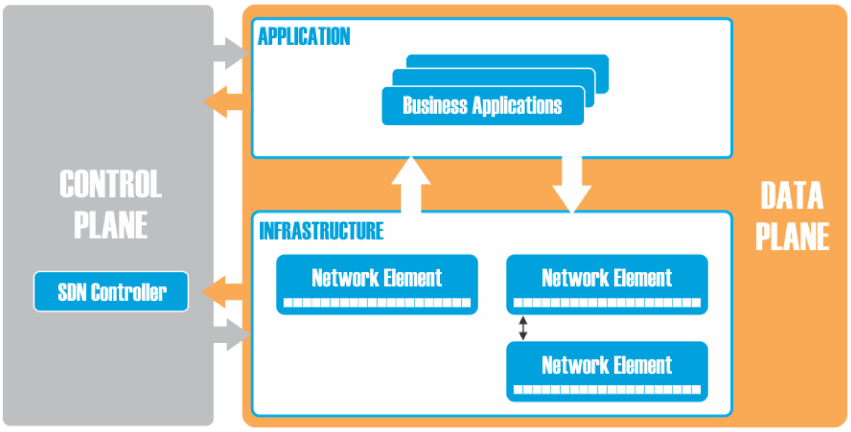


Figura 1 – Arquitetura SDN sob a óptica da separação dos planos de Dados e Controle da rede. Figura adaptada de DeCusatis et al. (2013)

Para Mendiola et al. (2016), o sucesso da abordagem SDN está relacionado com a disponibilidade de controladores *open-source* para fomentar o desenvolvimento de soluções. Ainda que, no plano de Dados, as soluções tenham se concentrado nas tecnologias *Ethernet* e *Internet Protocol (IP)*, existe a necessidade de evoluir a arquitetura para suportar novas tecnologias, tais como: *Optical Circuit Switching (OCS)*, *Optical Packet Switching (OPS)* e *gigabit passive optical networks (GPON)* (KALJIC et al., 2019).

De forma geral, algumas propostas focadas no plano de Dados se concentram em introduzir reconfigurabilidade no baixo nível da rede, enquanto outras optam pela abstração do plano de Dados tornando-o independente da tecnologia (KALJIC et al., 2019).

Independentemente da abordagem, observa-se que pesquisas em SDN são insuficientemente focadas no plano de Dados, contribuindo para que a infraestrutura de rede seja responsável por materializar o melhor cenário para satisfazer os requisitos de QoS e QoE das comunicações (KALJIC et al., 2019; MICHEL et al., 2021).

Neste contexto, chama-se atenção para o modelo definido por Pereira (2012b), que mesmo após 10 anos de sua concepção, oferece melhorias na compreensão semântica dos requisitos de comunicação nas camadas intermediárias e endereçamento na rede, impactando diretamente as capacidades dos elementos de rede (OLIVEIRA, 2019).

2.1.3 Modelo de Títulos para Internet do Futuro e a ETArch

Modelo de Títulos (PEREIRA, 2012b) é uma proposta para Internet de Futuro que visa aproximar semanticamente as camadas superiores e inferiores, permitindo aos requi-

sitos das comunicações serem compreendidos pelas camadas intermediárias. A *ETArch* (SILVA, 2013b) é a materialização do Modelo de Títulos sob uma perspectiva *Clean Slate*.

Aplicações do tipo *VoIP* são exemplos que justificam a utilização do Modelo do Títulos, pois ilustram a dificuldade em atender os requisitos de comunicação *multicast* e *broadcast* (OLIVEIRA, 2019). Na Internet, para que a comunicação seja possível, são necessários diferentes endereços em diferentes níveis no TCP/IP, i.e., endereço MAC (Camada de Enlace), endereço IP (Camada de Rede), endereço de transporte (Camada de Transporte) e endereço do usuário (Camada de Aplicação) (OLIVEIRA, 2019).

Neste contexto, o Modelo de Título prevê a utilização de um identificador único, chamado *Título* com suporte semântico através da camada de Serviço do Modelo para endereçar entidades (PEREIRA, 2012b). O padrão de organização em camadas é adotado para especificação do Modelo de Títulos e a ETArch, assim como no TCP/IP. Na Figura 2 é possível visualizar a ETArch, o Modelo de Título e o TCP/IP sob a perspectiva das camadas.

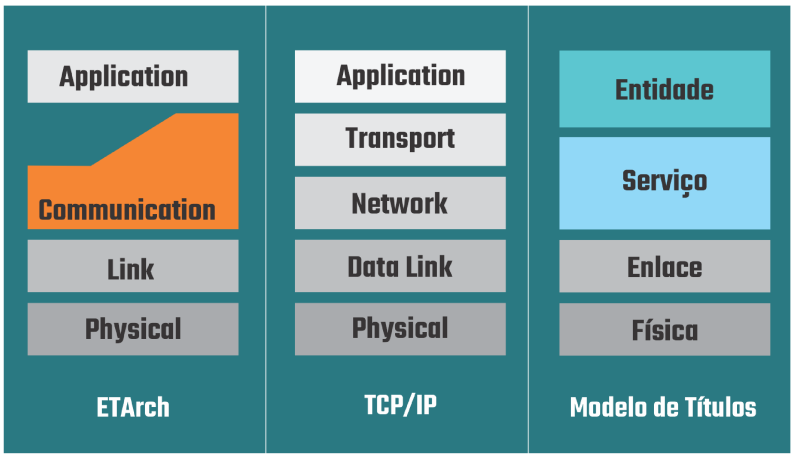


Figura 2 – Camadas TCP/IP e Camadas do Modelo de Títulos. Figura adaptada de Pereira (2012b), Silva (2013b)

De acordo com Oliveira (2019), a camada Entidade do Modelo de Títulos se comunica semanticamente com a camada de Serviço através de Ontologia. A camada de Serviço é responsável por traduzir para as camadas inferiores as necessidades de comunicações.

Sob a perspectiva da ETArch, a camada *Application* se aproxima da camada de Aplicação da arquitetura Internet, sendo que o foco da ETArch encontra-se na camada *Communication*, substituindo as camadas de Rede e Transporte dos protocolos TCP/IP, sendo esta responsável por traduzir para as camadas inferiores as necessidades de aplicações.

Ainda de acordo com a Figura 2, pode ser observado que as camadas inferiores, tanto do Modelo de Títulos quanto na ETArch, não tiveram notáveis modificações em relação ao TCP/IP.

De acordo com (OLIVEIRA, 2019), no Modelo de Título, a comunicação entidade-entidade possui, semanticamente, o mesmo comportamento de uma comunicação cliente-servidor da Internet, onde toda entidade está habilitada para a comunicação. As comunicações, de forma geral, possuem diferentes requisitos em função do contexto, o Modelo de Títulos define a utilização de ontologia para que a camada de Entidade consiga comunicar semanticamente com a camada de Serviço (OLIVEIRA, 2019).

Para uma melhor compreensão dos elementos conceituais do Modelo de Títulos, que também estão presentes na ETArch, faz-se necessário detalhar os termos *Entidade*, *Título*, *Workspace* e *Serviço de Domínio de Título*, que possuem um significado, consideravelmente, diferente dos aplicados na Internet.

De acordo com Pereira (2012b), *Entidades* são elementos de comunicação com requisitos bem definidos; O *Título* representa uma designação única para identificar, de forma não ambígua, uma entidade; O *Workspace* é o barramento lógico criado para suportar comunicações entre entidades e, o *Serviço de Domínio de Título* é um domínio capaz de compreender as instâncias das entidades bem como suas propriedades e requisitos.

Sob a óptica do plano de Dados, as comunicações são materializadas na ETArch através do elemento de rede com suporte à comunicação baseada em *Workspace* e endereçamento baseado em *Títulos*. Neste contexto, o *Workspace* se institui como o principal elo entre as *Entidades* no plano de Dados.

De acordo com Silva (2013a) o *Workspace* é definido pelos seguintes atributos: Título, lista de elemento de rede (NE), lista de capacidades, requisitos, visibilidade (público ou privado) e nível. A criação, gerenciamento e encerramento de um *Workspace* é controlado pelo sistema distribuído de títulos, chamado *Domain Title Service (DTS)*, constituído por agentes denominados *Domain Title Service Agent (DTSA)* (SILVA, 2013a).

Na Figura 3 é ilustrado um cenário característico de um ambiente DTS/ETArch, sendo possível visualizar os principais elementos definidos nesta seção.

Os *Workspaces* podem ser de dados ou controle, operacionalmente, os *Workspaces* de dados são instituídos pelo DTSA a partir de solicitações advindas das *Entidades*, que informam suas necessidades e suas capacidades de comunicação ao controlador da rede (SILVA, 2013a). O DTSA então envia mensagens de controle para o elemento de rede com a premissa de atualizar as políticas de encaminhamento para compreender a(s) nova(s) *Entidade(s)*.

De forma geral, o *Workspace* é instituído a partir de um conjunto de requisitos exigidos pela aplicação. Note que uma simples *Network Function Virtualization (NFV)* deve suportar diversas aplicações. No entanto, pensar que cada aplicação possui requisitos de comunicação específicos, um *Slice*[2] deveria ser associado com a uma aplicação. Neste contexto, entende-se que o *Workspace* oferece uma granularidade mais fina em relação ao

[2] *Network Slicing* trata-se de um conceito para as futuras redes móveis, que visa compartilhar a mesma infraestrutura física com diversas aplicações com diferentes requisitos de QoS.

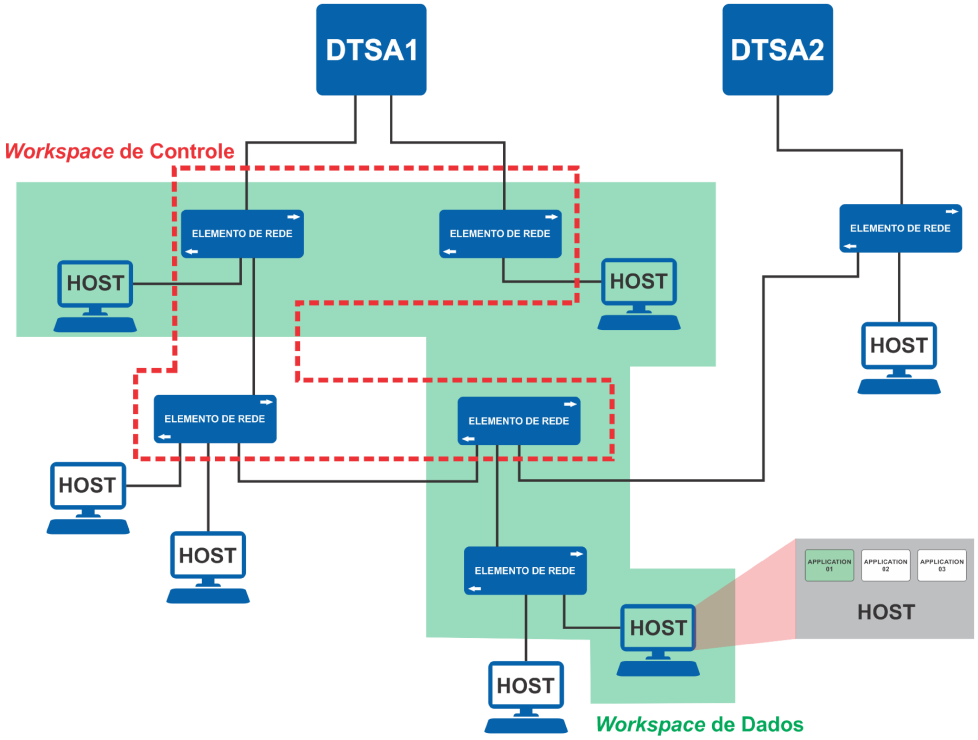


Figura 3 – Cenário típico ETArch com os elementos: DTS, DTSA, Entidade, NE e Workspace. Figura adaptada de Silva (2013a)

gerenciamento de recursos em comparação com o *Slice*.

Diante do contexto apresentado, mesmo com as perspectivas oferecidas pela ETArch, existe a necessidade de um mecanismo que suporte (re)configuração sistemática das políticas de encaminhamento com fina granularidade para atingir alto nível de flexibilidade no plano de Dados, proporcionando melhoria na QoS e QoE de usuários.

2.1.4 Programabilidade e Flexibilidade no Plano de Dados em SDN

A princípio, SDN introduz a capacidade de controlar o plano de Dados através de uma interface bem definida, portanto é possível implementar soluções de encaminhamento exclusivas para diferentes cenários (MICHEL et al., 2021; XIA et al., 2014). A capacidade de (re)programação de elementos de rede é visível, porém, torna-se importante ressaltar que existe uma sutil diferença na interpretação dos termos programabilidade e flexibilidade. Conforme Kaljic et al. (2019), a flexibilidade reflete a capacidade do plano de Dados em atender novas funcionalidades e a programabilidade é o método pelo qual a flexibilidade é alcançada.

De acordo com Zilberman et al. (2015), embora o benefício de SDNs esteja focado na capacidade de (re)programação no plano de Controle, trabalhos enfatizam a importância da reconfiguração dinâmica no plano de Dados (MICHEL et al., 2021) Kaljic et

al. (2019). A flexibilidade no baixo nível da rede implica na capacidade de elementos de redes expressarem suas capacidades e suportarem reconfigurações sistemáticas, rápidas e escaláveis (BIFULCO; RÉTVÁRI, 2018).

Michel et al. (2021) afirmam que o elemento de rede em SDN se tornou mais flexível e dinâmico em comparação com os convencionais, que possuem funcionalidades profundamente enraizadas no *hardware*, por exemplo, um *Switch Ethernet* tradicional não processa os campos da camada de Rede do TCP/IP e um roteador antigo não suporta *IPv6*. Fato é que, a configuração de chips de comutação não é trivial e carece de melhorias para se tornar expressiva e flexível frente as necessidades de um plano de dados programável (MICHEL et al., 2021).

Diante dessas dificuldades, tem se observado que as soluções em SDN assumem que existe conectividade no nível do Enlace, independente de aspectos da infraestrutura e das capacidades dos *Switches*, esta sendo considerada satisfatória para o funcionamento da rede, negligenciando qualquer possibilidade de melhoria (OLIVEIRA, 2019).

No estudo realizado por Kaljic et al. (2019), os autores enfatizam que não existe um consenso para definir flexibilidade no plano de dados. Ao ponto que alguns trabalhos associam flexibilidade ao projeto da rede, outros avaliam sob a óptica da resiliência do sistema. Fato é que a programabilidade é um fator chave para alcançar flexibilidade no plano de dados.

O termo 'Elemento de Rede' é utilizado para expressar a existência de um 'objeto capaz de comutar transmissões de dados' no contexto SDNs. Em relação ao plano de Dados, o elemento de rede é concretizado pelo comutador ou *Switch*.

A compreensão da forma de operação de comutadores de dados de propósito geral torna-se importante, pois, permite avaliar os processos sob a óptica da flexibilidade e programabilidade, contribuindo para os requisitos da arquitetura NEA.

Nas seções 2.1.5 e 2.1.6 serão apresentadas as tecnologias utilizadas para construção de comutadores de dados de propósito geral.

2.1.5 Comutação de Dados em Plataformas Legadas

Sob a visão da organização, a comunicação entre duas entidades se inicia nas camadas Física e de Enlace *Link* do modelo *Open System Interconnection (OSI)* (ISO, 2017) (OLIVEIRA, 2019). O responsável pelo encaminhamento de primitivas entre as entidades neste nível da rede é o elemento de rede[3] (IEEE, 2001).

De acordo com Oliveira (2019), para o desenvolvimento desta tarefa, o elemento de rede possui Tabelas de Chaveamento[4] (*Switching Table*), atualizadas pelo plano de Controle. Nos elementos de rede *Ethernet-based*, esta tabela comumente é chamada *MAC Address*

[3] Doravante, o termo elemento de rede, comutadores de dados e *Switch* possuem o mesmo valor semântico

[4] Neste trabalho também denominada Tabela de Encaminhamento

Table (KUROSE; ROSS, 2006). Independente da nomenclatura, esta tabela possui os seguintes campos: endereço de *hardware* (*MAC Address*) para os dispositivos conectados, a porta física que cada dispositivo se conecta e um *timestamp* para controlar o ciclo de vida do registro (OLIVEIRA, 2019).

Com base nas informações presentes nos cabeçalhos dos *frames*, os dispositivos praticam as políticas de encaminhamento (*forwarding*). Neste contexto, um *frame* pode ser encaminhado para uma porta de saída do equipamento, ou para várias portas (KUROSE; ROSS, 2006).

De acordo com Oliveira (2019), o processo de construção e manutenção das tabelas de encaminhamento é de total responsabilidade do elemento de rede. Inicialmente as tabelas encontram-se vazias e sempre que uma Entidade é conectada em uma determinada porta física, um novo registro é inserido na tabela de encaminhamento (Origem (*SRC-ADD*), Porta (*PORT*), Tempo (*TIME*)). Naturalmente, os elementos de rede possuem rotinas internas de controle para coletar registros inativos que possam ser excluídos, visando o pleno funcionamento do equipamento.

As tabelas de encaminhamento são armazenadas em memórias voláteis, como por exemplo memórias do tipo *Content Addressable Memory (CAM)* (CISCO, 2017). De acordo com Irfan et al. (2022), memórias CAM permitem uma forma de endereçamento divergente das convencionais, isto é, ao invés de endereço numérico, é possível utilizar o endereço MAC como referência e todo o processo custa apenas um ciclo de processamento. Ao prazo que, as memórias do tipo CAM endereçam valores binários (0's e 1's), as *Ternary Content-Addressable Memory (TCAM)* adicionam um terceiro elemento, aumentando sua capacidade de endereçamento. Memórias do tipo TCAM são largamente utilizadas em roteadores pois apresentam alto desempenho para consultas em tabelas de roteamento e *Access Control List (ACL)* (IRFAN et al., 2022).

Naturalmente, a demanda de processamento e encaminhamento no elemento de rede pode ser insuficiente frente à velocidade do canal de transmissão (LANGEMAK, 2015). Para contornar este problema, esses equipamentos recorrem a *buffer packets*[5] para acomodar as filas de *frames*. Por padrão, cada porta física é composta por uma fila de saída e uma fila de entrada. Essas filas são implementadas utilizando o conceito de *FIFO*. A Figura 4 ilustra resumidamente a arquitetura básica de um *Switch* L2.

A comunicação entre *Switch-Switch* é instituída por um barramento chamado *trunk* de alta velocidade, ilustrado na Figura 4 pela porta *Quad Small Form Factor Pluggable (QSFP)*. Portas QSFP são utilizadas para escalar o equipamento em ambientes onde o número de portas físicas não é satisfatório, visando garantir o desempenho da rede.

Swtiches atuais permitem que várias *Local Area Networks (LANs)* possam coexistir no mesmo equipamento físico. Através da técnica de *Virtual Local Area Network (VLAN)* é

[5] Espaço de memória destinado ao armazenamento de pacotes que aguardam transmissão ou são recebidos pela rede.

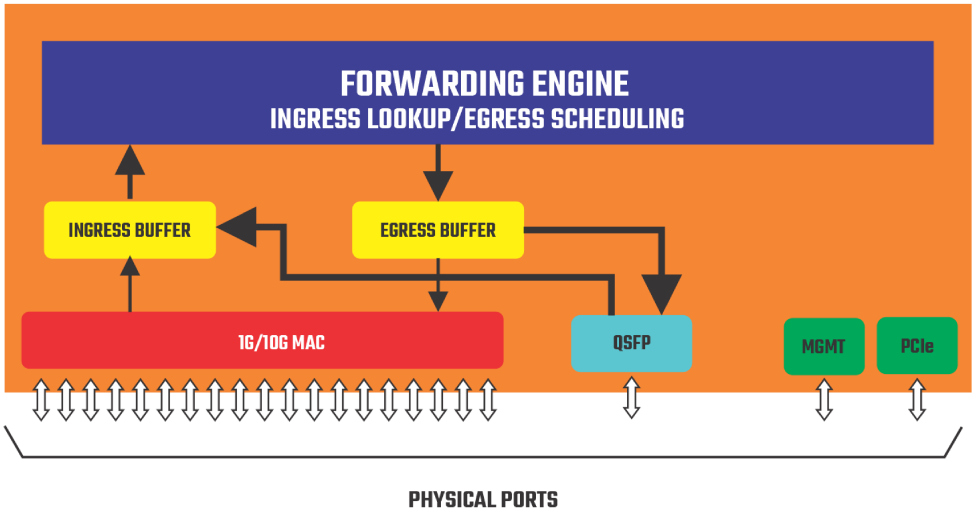


Figura 4 – Visão geral da arquitetura de um *Switch* l2 sob a perspectiva do encaminhamento. Figura adaptada de Ferro (2013)

possível dividir uma rede física em várias redes virtuais, criando domínios de *broadcast* separados. Ainda que básico, VLANs permite que comutadores de dados tenham influência sobre a QoS através da capacidade de priorização de tráfego.

2.1.6 Suporte do Sistema Operacional Linux para Comutação de Dados

A adoção do sistema operacional Linux para implementação de soluções voltadas para redes de computadores tem ganhado atenção nos últimos anos. Projetos como: OpenvSwitch (PFAFF et al., 2015), Rocker Switch (FELDMAN, 2015) e SwitchDev (PIRKO; FELDMAN, 2014) são exemplos de soluções que recorrem ao uso do sistema operacional Linux para compor soluções para encaminhamento de dados.

O sistema operacional Linux permite a utilização de *Bridges* para agregação de fluxos, além do suporte a diversas tecnologias da camada de Enlace, tais como: *Ethernet*, padrão 802.1, inclusive 802.1P/Q, também permite gerenciar diversas interfaces de redes físicas (NIC) através de uma interface lógica, deixando o nível de *hardware* transparente e independente de tecnologia..

Nativamente, o Linux possui suporte para diversas funções de rede L2/L3 através de suas ferramentas *iproute2*, *ifconfig*, *bridge*, *brctl*, *vconf*, *iptables*, *ebtables*, *nftables*, *SwitchDev*, dentre outras.

No contexto desta tese, objetiva-se detalhar a *Bridge*, pois trata-se do princípio básico de comutação de dados através da utilização do sistema operacional Linux.

2.1.6.1 Linux Bridge

A *Bridge* é uma interface de rede lógica composta por pelo menos uma interface de rede física. *Bridges* modernas atuam nas camadas 2 e 3 da arquitetura TCP/IP, manipulando primitivas através do endereço MAC e endereço IP. O funcionamento das *Bridges* é transparente na rede, o que a torna atrativa em aplicações como: equipamentos de comutação L2/L3, *Firewalls* e soluções de monitoramento de tráfego de rede, dentre outras.

Soluções como OpenvSwitch (PFAFF et al., 2015), RockerSwitch (FELDMAN, 2015) e SwitchDev (PIRKO; FELDMAN, 2014) fazem uso de *Bridges* em suas arquiteturas para executar funções L2/L3, segregação de tráfego com VLAN, melhoramento da QoS, dentre outros. De acordo com (VARIS, 2012), um módulo de *Bridge* é dividido em 3 partes, são elas: (i) estrutura de base de dados; (ii) configuração; e (iii) processamento de *frames/packets*.

Na Figura 5 podem ser observados os módulos e submódulos que compõem *Bridges*, bem como as interfaces de comunicação entre *user-space* e *kernel-space* (LIU, 2017).

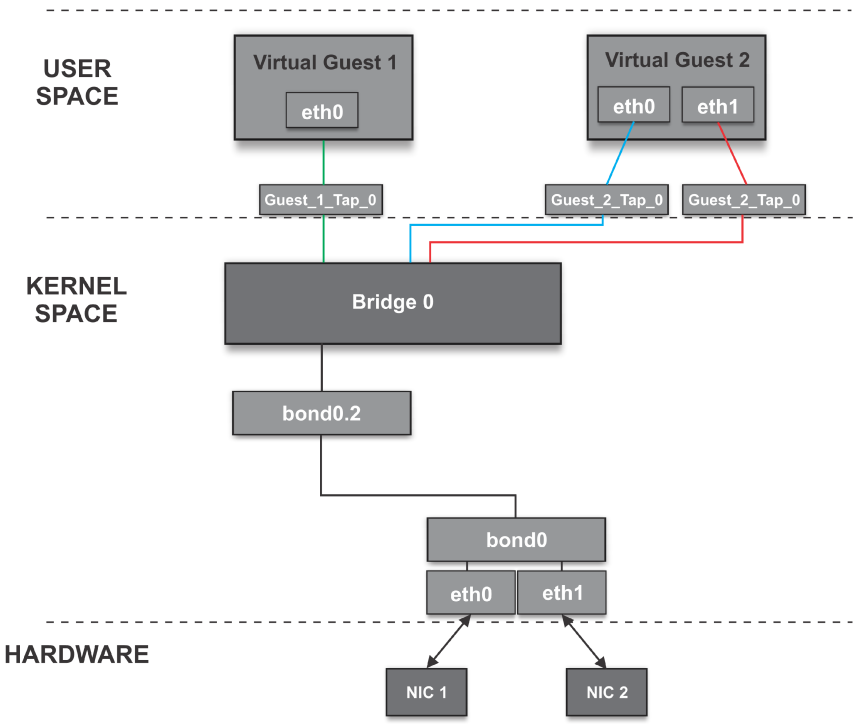


Figura 5 – Linux Bridge de propósito geral - Figura adaptada de Liu (2017)

Do ponto de vista da comutação de dados, torna-se importante compreender o funcionamento da *Bridge* em relação à manipulação de endereços MAC, a estrutura da *Forwarding Data Base (FDB)*, a interface de configuração das *Bridges* e, por último, a compressão do processo de encaminhamento de primitivas dentro da *Bridge*.

2.1.6.2 *Forwarding Database - (FDB)*

A FDB é responsável por armazenar endereços MAC relativos a entidades conectadas nas portas físicas do elemento de rede. Na Figura 6 é possível visualizar o mecanismo de indexação de endereços MAC para a tabela de encaminhamento (FDB) (VARIS, 2012).

De acordo com Varis (2012), internamente, a FDB é estruturada por uma matriz de 256 linhas, onde cada linha é definida como um elemento que se comporta como uma lista simplesmente encadeada. Cada nó dessa lista é uma referência direta para um endereço MAC reconhecido pela *Bridge*. Na Figura 6 é apresentado toda a organização da FDB e a forma de indexação de endereço MAC na *Bridge*.

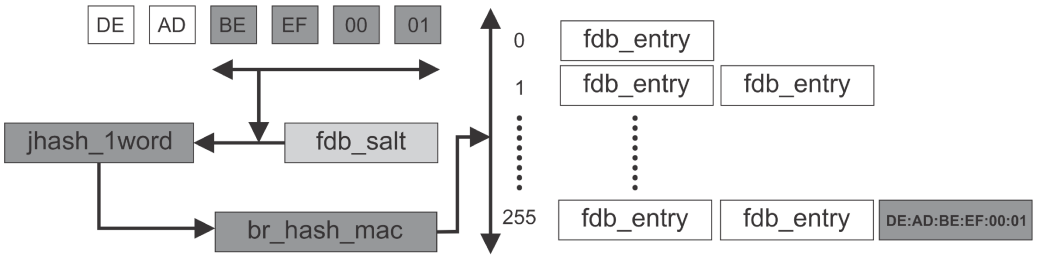


Figura 6 – Organização da FDB na *Bridge*. Figura adaptada de Varis (2012)

Os Endereços MAC são compostos por 48 bits, onde 24 bits são reservados para *Organizationally Unique Identifier (OUI)* e os outros 24 bits especificam um identificador exclusivo no OUI, normalmente referente ao fabricante do *hardware*.

De acordo com Varis (2012), a FDB utiliza um mapeamento de *hashing* para os endereços MAC, e este processo é calculado pela função *br_hash_mac*. De acordo com Varis (2012), este procedimento garante um identificador suficiente exclusivo e, em simultâneo, permite a utilização de algoritmos de *hash*.

A entrada da *FDB* para o endereço MAC de destino é encontrada através da lista encadeada referente ao *hash* gerado anteriormente.

As entradas não utilizadas na FDB são limpas periodicamente pela função *br_cleanup*, acionada pelas funções de controle da *Bridge*.

2.1.6.3 Configuração de um módulo Bridge

De acordo com (VARIS, 2012), *Bridges* possuem duas interfaces separadas de configuração para o nível de usuário do sistema operacional. A primeira delas é *ioctl*, utilizada para criar e remover *Bridges* do sistema operacional e também realizar a inclusão ou exclusão de interfaces de redes nas *Bridges* (VARIS, 2012). A segunda interface de configuração, *sysfs*, é utilizada para manipulação dos parâmetros das interfaces das *Bridges* (VARIS, 2012).

Na Figura 7 é ilustrado o processo de configuração através da interface *iotcl*, que é responsável por criar e inicializar a *Bridge*. Neste procedimento é possível também inserir ou excluir determinadas interfaces de rede (Portas físicas).

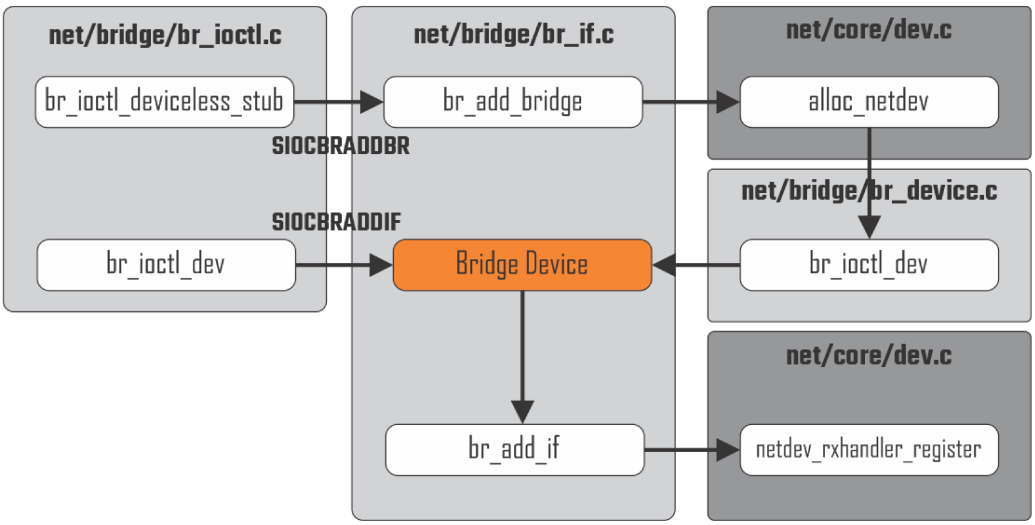


Figura 7 – Interfaces de configuração da *Bridge*. Figura adaptada de Varis (2012)

A criação de novas *Bridges* é iniciada através do comando $SIOCBRADDIF$. As funções *br_ioctl_deviceless_stub* e *br_add_bridge* são responsáveis por criar o objeto *Bridge* internamente no Kernel e eventualmente chamar a função *alloc_netdev* para criação do *netdevice* (VARIS, 2012).

Através do comando $SIOCBRADDIF$ é possível inserir interfaces de rede nas *Bridges*. Essa função é responsável por criar e configurar uma nova porta dentro da *Bridge* para comportar o objeto *net_bridge_port*.

2.1.6.4 Processamento de *Frames* na *Bridge*

O tratamento de primitivas na *Bridge* tem início direto no *socket buffer*[6] (VARIS, 2012). Na Figura 8 é apresentado todo o caminho percorrido por um *frame* dentro da *Bridge*, desde o ingresso até a fila de saída.

Torna-se importante ressaltar que, cada interface adicionada na *Bridge* terá um manipulador de entrada (*rx_handler*) definido como manipulador de entrada da *Bridge* (*br_handle_frame*) (VARIS, 2012).

De acordo com a Figura 8, pode ser observado que o *frame* é inicialmente processado pela função *br_handle_frame*, isto inclui realizar as etapas de conferência e também a separação entre *frames* com destino local ou que deverão ser encaminhados para a *Bridge*. Por padrão, primitivas de controle, como por exemplo Spanning Tree Protocol (STP), são

[6] Porção de memória que implementa o conceito de fila, sendo utilizado entre às NICs e o espaço de memória do processo.

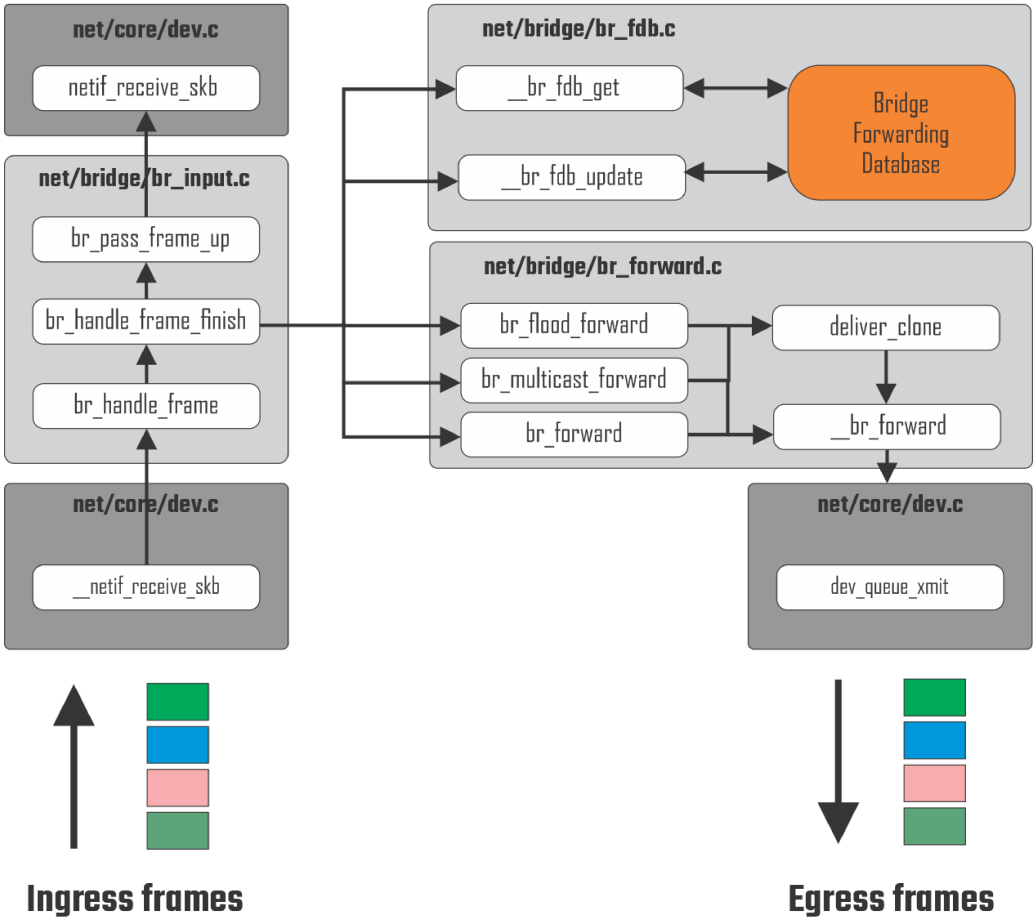


Figura 8 – Arquitetura de encaminhamento *Bridge*. Figura adaptada de Varis (2012)

encaminhadas para as camadas superiores. Por fim, se uma decisão de encaminhamento for tomada, a primitiva é repassada para *br_handle_frame_finish*, onde de fato será realizado o encaminhamento (VARIS, 2012).

De acordo com Varis (2012), a função *br_handle_frame_finish* realiza a atualização da FDB através da função *br_fdb_update* e, com base no endereço MAC de destino, a primitiva alcança a fila de saída.

Frames unicast terão a FDB indexada com o endereço MAC de destino. Através da função *__br_fdb_get*, a FDB será consultada para descobrir qual será o *net_bridge_port* para onde a primitiva será enviada através da função *br_forward* (VARIS, 2012).

Caso o *frame* não tenha nenhuma entrada registrada na FDB, a função *br_flood_forward* será chamada e o *frame* será encaminhado para todas as *net_bridge_ports*.

No caso da comunicação *multicast*, o *frame* é consultado na FDB através da *net_bridge_ports* e na sequência o encaminhamento para todas as portas previstas utilizando a função *br_multicast_forward* (VARIS, 2012). De forma geral, a função *br_flood_forward* será acionada quando o *frame* em questão não possuir uma entrada na FDB.

A lógica de encaminhamento do módulo *Bridge* do Linux é implementada em três funções básicas: (i) *br_flood_forward*, para encaminhamento *broadcast*, (ii) *br_multicast_forward*,

para encaminhamento *multicast* e (iii) *br_forward*, para encaminhamento *unicast*. Todas as três funções de encaminhamento acabam acionando a função *___br_forward* que de fato adiciona o *frame* no *buffer* de saída para ser consumido pelas NICs através da função *dev_queue_xmit* (VARIS, 2012).

Além do foco em *QoS* e *QoE*, esta tese foca nos mecanismos de comutação de dados centrados em *software*. Como já definido anteriormente, atuar no redesenho do MAC requer um grande esforço, não somente em relação a construção de uma nova solução, mas também na alteração de milhares de equipamentos de rede. Na Seção 2.2 será apresentado os trabalhos correlatos sob a óptica da programabilidade e flexibilidade com foco na melhoria da QoS e QoE.

2.2 Trabalhos Relacionados

Embora o OpenFlow, como já citado anteriormente, seja amplamente utilizado como tecnologia SDNs, existem diversas propostas emergentes para alcançar flexibilidade e programabilidade no plano de dados. De forma geral, existem dois tipos de abordagens para soluções no plano de dados em SDNs: as centradas em *software*, onde toda a capacidade de trabalho da solução está concentrada no nível de *software*; e as centradas em *hardware*, que objetivam explorar as características do *hardware* para o desenvolvimento das funções de encaminhamento.

Nesta seção, trata-se especificamente o uso de mecanismos L2 para provimento de alguma programabilidade no nível de Enlace. Ademais, torna-se importante compreender como esses mecanismos e técnicas exploram a flexibilidade tanto ao nível de *software* quanto em *hardware*.

2.2.1 Abordagens Centradas em *Software* para o Plano de Dados em SDN

Explorar flexibilidade nos diferentes níveis de uma arquitetura SDN tem ganhado bastante evidência atualmente. Aspectos de programabilidade podem estar presentes em diferentes níveis da arquitetura e serem materializados através de diversas tecnologias.

Com relação ao plano de dados, de acordo com Kaljic et al. (2019), predominantemente existem duas arquiteturas, quais sejam *Forwarding and Control Element Separation (ForCES)* e o OpenFlow. Embora inúmeras soluções sejam baseadas nessas arquiteturas, formas diferentes são praticadas para alcançar programabilidade.

2.2.1.1 *Forwarding and Control Element Separation (ForCES)*

De acordo com Khosravi e Anderson (2003), a arquitetura descrita do ForCES é composta de várias entidades com funcionalidades bem definidas que cooperam entre si para

alcançar a funcionalidade desejada, como por exemplo roteamento e encaminhamento IP.

A entidade lógica que implementa o protocolo ForCES é denominada (*Forwarding Element (FE)*). Normalmente estas entidades recorrem a uma plataforma de *hardware* para prover a capacidade de processamento de pacotes sendo controladas por *Control Elements (CEs)*, que são entidades lógicas que implementam o protocolo ForCES para gerenciar um ou mais *FEs*.

Sob a óptica da arquitetura do elemento de encaminhamento ForCES, pode-se observar que o *Logical Functional Block (LFB)* é a representação lógica de uma simples funcionalidade. Naturalmente os *Data Paths* são formados pela interconexão de diversos LFBs (HALPERN; SALIM, 2010) (KALJIC et al., 2019). A Figura 9 ilustra a organização dos elementos dentro de uma arquitetura ForCES.

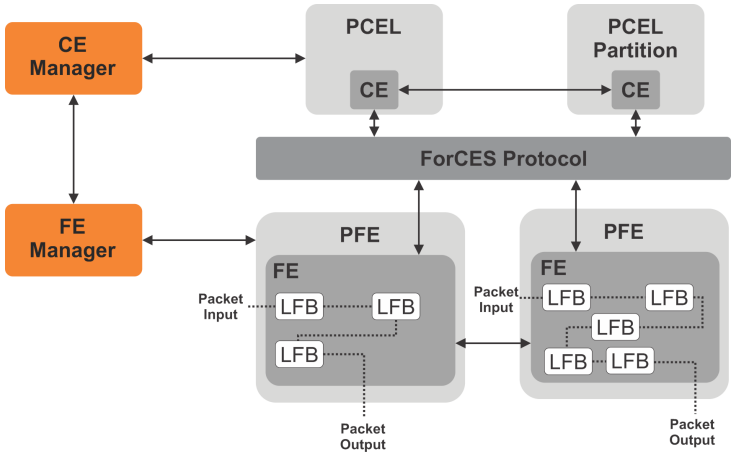


Figura 9 – Visão geral da arquitetura ForCES. Figura adaptada de Mendiola et al. (2016)

Na Figura 9 é possível observar um método para disposição e interligação das LFBs. De acordo com (KALJIC et al., 2019), existem 3 categorias de encadeamentos: (i) sequencial, onde todos os LFBs são interligados de forma sequencial, impondo um cadenciamento no fluxo de pacotes; (ii) Ramificado (*Branches*), que prevê a utilização de condições e fluxos ramificados; e (iii) Híbrido, que implementa ambas as descrições anteriores. Uma cadeia de LFBs apropriada é gerada através de um Agente (*LFB Chain Matching Agent*), que possui o principal papel de combinar uma ou mais LFBs para atender a solicitação.

O ForCES fornece um suporte significativo em várias áreas onde o processamento de pacotes distribuídos nos elementos de rede é necessário. Além disso o ForCES define um modelo flexível e personalizado visto que existe a possibilidade de criação de novas LFBs.

Embora o ForCES tenha sido apresentado como um modelo promissor, cabe ressaltar que sua aplicação não está em um nível satisfatório devido a vários fatores, entre eles: (i) falta de adoção por empresas que atuam no setor de redes e (ii) pouco uso no ambiente acadêmico devido à falta de suporte para trabalhos experimentais na forma de código-fonte aberto utilizável.

2.2.1.2 *Switches OpenFlow*

A tecnologia OpenFlow tem sido utilizada amplamente para o desenvolvimento de soluções em SDN. O OpenFlow oferece uma maneira fácil de desenvolver soluções L2 para suportar cenários SDN por meio da virtualização do nível de infraestrutura de rede. Segundo o McKeown et al. (2008), o OpenFlow permite a criação de fluxos configuráveis a partir de um controlador central desenvolvido em *software*. Ele separa o plano de dados do plano de controle e apresenta um modelo de programação logicamente centralizado para gerenciamento dos elementos da rede. A Figura 10 ilustra a arquitetura OpenFlow.

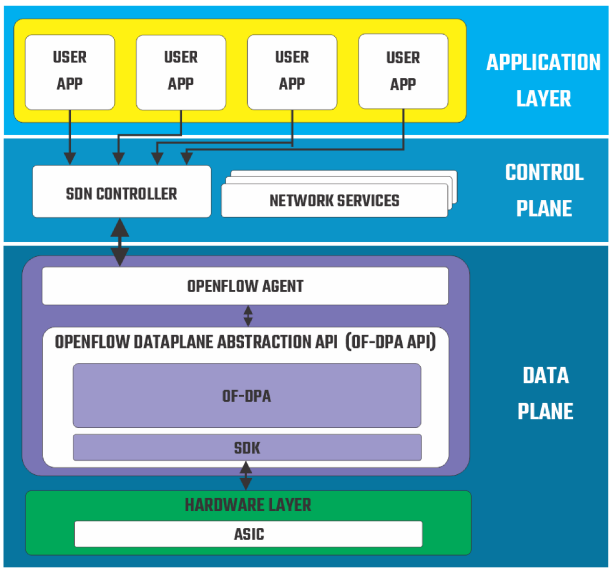


Figura 10 – Visão geral da arquitetura OpenFlow. Figura adaptada de Mehra, Maurya e Tiwari (2019)

Em relação ao encaminhamento de dados nos *Switches* OpenFlow, há uma tabela de fluxos e uma ação associada a cada entrada da tabela, resultando em um limitado conjunto de ações por fluxos ao invés de tratamento individual por aplicação (MEHRA; MAURYA; TIWARI, 2019; OLIVEIRA, 2019). Os principais elementos do OpenFlow são: *Flow Table* (uma ação associada a cada entrada), *Secure Channel* (entre o controlador e o *Switch*) e *OpenFlow Protocol* (padrão de comunicação entre o controlador e o switch).

Conforme McKeown et al. (2008), existem dois tipos de comutadores OpenFlow, são eles:

- *Dedicated OpenFlow switches*: Um comutador cujo encaminhamento de pacotes é definido por um controlador remoto. Os Fluxos e as regras são entradas na tabela, tendo como referência três regras básicas: (i) encaminhar os pacotes de um determinado fluxo para uma determinada porta; (ii) encapsular e encaminhar o fluxo para o controlador, então o controlador decide se o fluxo será adicionado à tabela de fluxo; e (iii) descartar o fluxo (evitando *denial of service*, por exemplo). As entradas

na tabela são identificadas pelo cabeçalho do pacote (que define o fluxo), a ação (o que fazer com o pacote) e estatísticas (número de pacotes, último encaminhamento, etc.).

- *OpenFlow-enabled switches*: São os comutadores/roteadores/*access points* que integraram funcionalidades OpenFlow. Eles precisam separar os fluxos experimental e regular, o que pode ser feito encaminhando os pacotes do fluxo através do pipeline normal do comutador ou definindo através de VLANs.

Os *Switches* OpenFlow operam sobre plataformas legadas, predominantemente *Ethernet-based* e embora permitam fluxos configuráveis, veta-se qualquer possibilidade de modificação no baixo nível (BIFULCO; RÉTVÁRI, 2018).

De acordo com Foundation (2015), *Switches* OpenFlow oferecem um limitado suporte de QoS através de um simples mecanismo de fila. Com relação a priorização de tráfego, no OpenFlow este aspecto é tratado diretamente na tabela de fluxo, ofertando priorização para todo um fluxo, exigindo a (re)configuração de toda entrada na tabela caso algum parâmetro seja alterado. O OpenFlow oferece um simples e limitado controle de banda (*rate*), conforme Foundation (2015), na prática o controle de banda é especifico da implementação, não sendo específica para cada pacote individual mas sim mapeado pela banda. Na prática, para janelas de transmissão curtas (*Windows size*) o controle de banda do OpenFlow é ignorado devido a granularidade do pacote (FOUNDATION, 2015).

De acordo com Kaljic et al. (2019), a programabilidade do OpenFlow é limitada ao nível de conteúdo do fluxo da tabela, o que restringe sua flexibilidade apenas ao aspecto de adaptação de fluxos. Em suma, entende-se que o OpenFlow apresenta uma granularidade grossa[7] em relação às ações de encaminhamento, pois não é permitido manipular fluxos orientados por aplicações, além de possuir uma estrutura fixa de *pipeline* baseada na consulta de tabelas de fluxo (*stateless*).

Conforme Brebner (2015), ao prazo que as redes SDN vieram prevendo um plano de dados simples, o OpenFlow baseava-se nos chips dos *Switches* existentes, de forma que o caminho de dados no nível mais baixo já era predeterminado.

Portanto, há uma necessidade contínua de modificação nas especificações do OpenFlow já que as redes SDN estão tomando espaço e há, então, uma necessidade de evoluir o OpenFlow para ampliar as tecnologias a que ele se destina, os protocolos que suporta, os cabeçalhos reconhecidos e a forma com que pacotes são processados (BREBNER, 2015).

2.2.1.3 *Protocol Independent Forwarding e P4*

Diante da necessidade de evolução do OpenFlow, a *Open Networking Foundation (ONF)* apresenta o projeto *Protocol Independent Forwarding (PIF)* (ONF, 2017). De

[7] Neste contexto granularidade grossa significa maior limitação em relação às ações de encaminhamento, resultando na incapacidade de alterar as características dos fluxos criados em tempo real

acordo com Oliveira (2019), através de elementos de rede programáveis que suportam o PIF, esta abordagem busca definir a arquitetura do *Switch* através de um *firmware* parametrizável.

Todos os esforços em relação ao projeto PIF estão direcionados para o P4 (P4.ORG, 2017). Bosshart et al. (2014) define P4 como uma linguagem de programação de alto nível para configuração de elementos de rede. De forma geral, o P4 é uma resposta às dificuldades impostas pelo OpenFlow em relação a capacidade de se expressar no baixo nível diante das demandas de programação no plano de dados.

De acordo com Kaljic et al. (2019) o P4 está pautado em três objetivos, são eles: (i) permitir configuração e reconfiguração do módulo *parser* e do módulo de processamento de pacotes das plataformas suportadas; (ii) garantir a independência de protocolo através da *match-action table*; e (iii) garantir a independência da plataforma alvo.

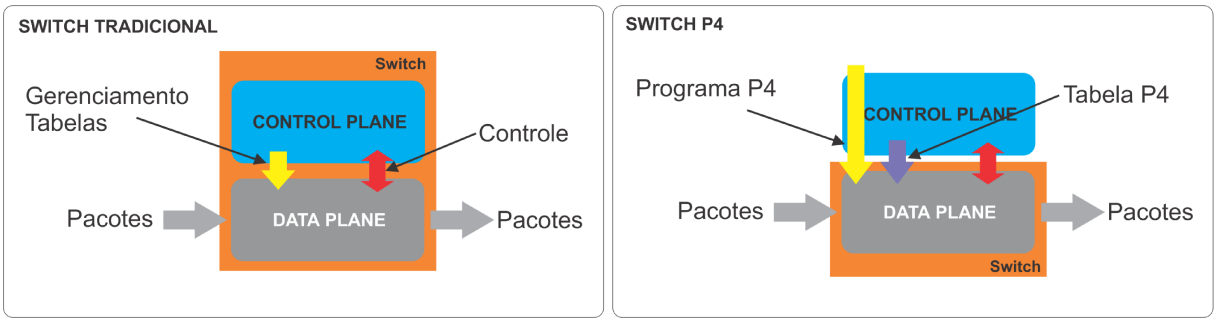


Figura 11 – Visão geral da arquitetura P4. Figura adaptada de Asterfuison (2022)

De acordo com o Asterfuison (2022), nos *Switches* tradicionais, o *Application Specific Integrated Circuit (ASIC)* é quem determina quais funções o plano de dados pode suportar, o plano de controle é responsável por atuar sobre essa infraestrutura física. Neste contexto, pode ser observado a dificuldade do P4 em se expressar diante de um equipamento legado.

Em suma, o P4 visa apresentar um modelo abstrato de encaminhamento com *parser* programável. O *pipeline* de tarefas é composto por múltiplos estágios de *match-action*, que podem ser arranjados de forma paralela se houver suporte do equipamento (KALJIC et al., 2019).

O P4 independe de protocolo, ao contrário dos *Switches* SDNs atuais que operam em sua grande maioria sobre IP, sendo possível descrever em alto nível, o formato do cabeçalho para ser interpretado no *Switching Fabric*, caso exista suporte (ASTERFUISON, 2022).

A desvantagem na utilização do P4 esta na necessidade de uma infraestrutura de tradução seja implementada até o nível do processador, usar ferramentas de síntese de alto nível para gerar *hardware* personalizado pode ser uma abordagem que introduz complexidade adicional no projeto, com resultados ainda não tão satisfatórios.

De acordo com Asterfuison (2022), o *Switch P4* da *Asterfusion X-T* oferece um *Switching Fabric* com suporte ao P4, entregando desempenho e flexibilidade para a solução.

2.2.1.4 *Open vSwitch (OVS)*

O OVS é um *Switch* multicamadas mantido pela Linux Foundation, licenciado sob a Apache 2. O OVS atua como um comutador de dados virtual para ambientes virtualizados. Escrito em linguagem C, atualmente é bastante utilizado em implementações em ambientes SDNs e oferece diversos recursos para encaminhamento de primitivas no nível L2 e L3 (PFAFF et al., 2015; PROJECT, 2016b). Dentre os principais recursos do OVS destaca-se o suporte nativo ao protocolo OpenFlow, *Internet Protocol Flow Information Export (IPFIX)*, *Remote Switch Port Analyser (RSPAN)*, *Link Aggregation Control Protocol (LACP)*, *Virtual eXtensible Local Area Network (VXLAN)* e 802.1ag.

De acordo com Pfaff et al. (2015), o OVS permite operar com dois caminhos de dados diferentes, um deles no nível de usuário do sistema operacional, responsável pela classificação das primitivas conforme a tabela de fluxo e a comunicação entre o plano de dados e o plano de controle. Já o caminho de dados no nível do Kernel é responsável por praticar os encaminhamentos diretamente no baixo nível sem a necessidade de comunicação com nível de usuário (PFAFF et al., 2015).

A Figura 12 ilustra a organização dos componentes do OVS.

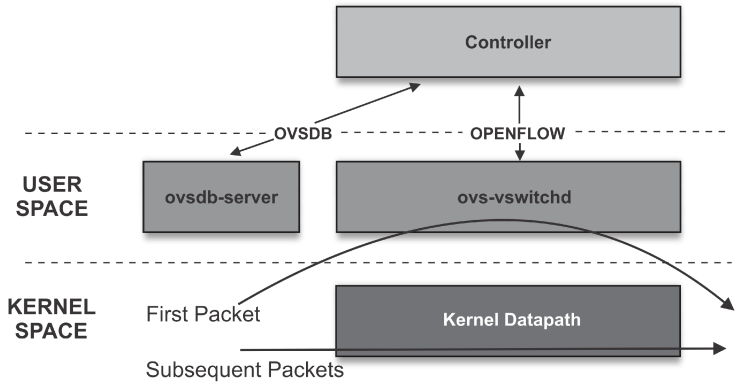


Figura 12 – Componentes e Interfaces do OVS - Figura adaptada de Pfaff et al. (2015)

Como pode ser observado na Figura 12, os fluxos de encaminhamentos que não exigem a participação direta do controlador, seja para inserção ou atualização, são resolvidos diretamente no nível de Kernel sem onerar o nível de usuário e principalmente, sem gerar chaveamento de contexto.

De acordo com Project (2016b), o OVS é totalmente portável devido a maior parte do código-fonte ser escrito em linguagem C, além de oferecer suporte a sistemas operacionais Linux, FreBSD e Microsoft Windows, sendo igualmente adaptado para outras arquiteturas tais como *Microprocessor Without Interlocked Pipeline Stages (MIPS)* e *Advanced*

RISC Machine (ARM), o que permite sua aplicação em sistemas embarcados (PFAFF et al., 2015). A integração entre o OVS e o Linux facilita a implementação de novas funcionalidades devido a sua organização modular e a compatibilidade com o restante do sistema operacional (PFAFF et al., 2015).

2.2.1.5 *Linux SwitchDEV*

A ideia principal da proposta *Ethernet Switch Driver Model* (SwitchDev) é eliminar a dependência de uso de *API* e *Software Development Kit (SDK)* de diversos fabricantes de dispositivos *Switches*, permitindo uma interface de comunicação flexível e simples com os *Switching Fabrics* (PIRKO; FELDMAN, 2014).

De acordo com Pírko e Hat (2015), o SwitchDev não necessita de um processador de alto desempenho para realizar operações do plano de dados, visto que, todo controle da soluções encontra-se no nível de *software*.

O Switchdev é hospedado no Kernel do Linux, especificamente o *netdev*, uma comunidade de projeto de código aberto abarcada pela *Linux Foundation*. As mudanças são feitas e inseridas no Kernel do Linux, seja adicionando novos recursos ou resolvendo correções; sendo assim, a comunidade garante que Switchdev fique ativo na infraestrutura.

A utilização do SwitchDev exige suporte de plataformas de *hardware*. Atualmente, alguns fabricantes se mostram otimistas com a proposta e possuem um linha de equipamentos com suporte ao SwitchDev, são eles: Mellanox Technologies (adquirida pela NVIDIA em 2019), Marvell Semiconductor, Edgecore Networks e Delta Electronics. A empresa Mellanox Techonologies foi a primeira a oferecer modelos de equipamentos com o SwitchDev e hoje possui uma linha de produtos com suporte ao SwitchDev (SITNIKOV, 2018).

Com relação à QoS, o SwitchDev faz uso do *Traffic Control (TC)*, que já está presente nas versões atuais do kernel Linux. Além do TC o SwitchDev utiliza *Virtual Local Area Network (VLAN)* para segregação de tráfego e controle de prioridade, contudo, as ações implementadas no SwitchDev são habilitadas por porta física e não aplicação (QRATOR, 2020), ofertando também um granularidade mais grossa com relação as ações de encaminhamento.

2.2.2 Abordagens Centradas em *Hardware* para SDN

Como forma de agregar desempenho nas tarefas de baixo nível das propostas centradas em *software*, as propostas centradas em *hardware* são promissoras, porém, de notável complexidade. Em geral, essas soluções fazem uso de tecnologias como *Field Programmable Gate Array (FPGA)* e *System on a Chip (SoC)*.

2.2.2.1 *Field Programmable Gate Array* (FPGA) e NetFPGA

Kalyaev e Melnik (2015) apresentam as vantagens de projetar *Switches* utilizando plataformas reconfiguráveis. Os autores justificam que o OpenFlow, por ser a opção mais acessível para implementação de solução em SDN, necessita de especial atenção em relação às capacidades do *hardware* legado, que em determinados casos, não permite atualização, possui capacidade de programação limitada e não possui total compatibilidade com as versões do OpenFlow (OLIVEIRA, 2019). Como máxima do trabalho, os autores propõem corrigir alguns gargalos de processamento com o auxílio de FPGA, sendo o principal deles trazer para o *hardware* as tabelas e os mecanismos de consulta, buscando maior *throughput* através da utilização da técnica de paralelismo (OLIVEIRA, 2019).

Channegowda, Nejabati e Simeonidou (2013) apresentam uma proposta de desenvolvimento de equipamentos de rede programável centrado no FPGA (OLIVEIRA, 2019). O objetivo é manter uma camada de *software* no elemento de rede operando de forma similar a um controlador, instituindo funções de rede (*Virtual Network Functions (VNF)*) e atuando diretamente no *hardware* através doFPGA (OLIVEIRA, 2019). Essa abordagem faz uso de um compilador para descrever o comportamento do *hardware* a partir de linguagens abstratas de alto nível. De acordo com os autores, essa abordagem é mais flexível quando comparada ao OpenFlow e trata-se de um projeto dos laboratórios Xilinx (OLIVEIRA, 2019).

De forma geral, as soluções baseadas em FPGAs e *NetFPGA* são bem sucedidas para pesquisas na área de redes. No entanto, elas possuem limitações ao serem utilizadas para inovação no campo de SDN (ANTICHI et al., 2013). Isso se justifica pois a *NetFPGA-1G*, primeira geração, não fornece recursos suficientes para o processamento de tabelas de fluxo em soluções OpenFlow. A interface PCI de comunicação é o gargalo para descarregar o tráfego para o hospedeiro e o controlador. A segunda geração de *NetFPGA-10G* suporta interface de fibra 10G e contém mais recursos de *hardware*, mas não há design de referência SDN/OpenFlow estável para pesquisas futuras (HU et al., 2014). A Figura 13 ilustra a visão geral da arquitetura de uma *NetFPGA*.

Além das limitações relacionadas ao barramento PCI e também a falta de portabilidade entre as versões, a *NetFPGA* exterioriza algumas dificuldades para explorar a programabilidade no baixo nível da estrutura, como por exemplo, mesmo dispondo de um tecido reconfigurável, a *NetFPGA* possui parte do MAC predefinido, limitando qualquer possibilidade de reconfiguração.

2.2.2.2 *System on a Chip (SoC)*

As abordagens instituídas com processador escalável integrado a plataformas reconfiguráveis com suporte a diversos sistemas operacionais são opções exploradas atualmente, pois visam minimizar os problemas arquiteturais das *NetFPGAs*.

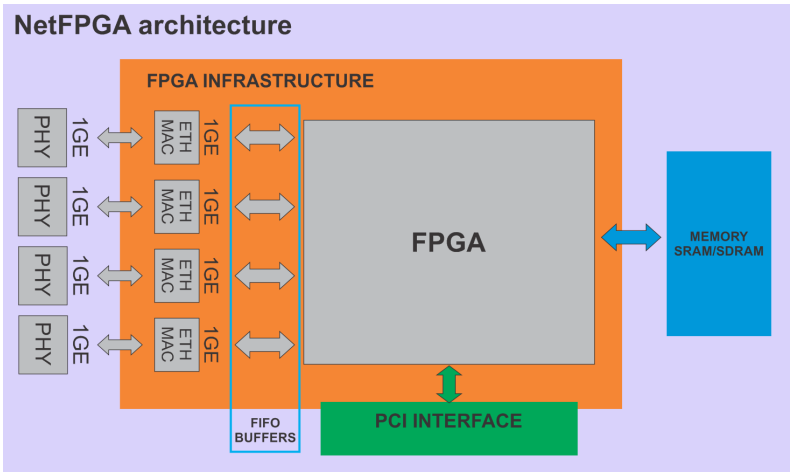


Figura 13 – Visão geral da arquitetura da *NetFPGA*. Figura adaptada de Gibb et al. (2008)

A plataforma *ONetSwitch*, apresentada por Hu et al. (2014), trata-se de um dispositivo totalmente programável de alto desempenho e baixo consumo de energia. A plataforma é composta pelo processador escalável *Xilinx Zynq SoC 7000*, que permite explorar a flexibilidade e programabilidade através de sua arquitetura híbrida *(ARM + FPGA)*.

Apesar do *ONetSwitch* oferecer uma solução frente ao problema de comunicação PCI da NetFPGA, ainda carece de melhorias. Assim como na NetFPGA, o *ONetSwitch* depende exclusivamente de um *Drive Ethernet* (ASIC) para operacionalização do MAC, não permitindo alterações neste nível da solução.

2.2.3 Requisitos de Elementos de Rede em SDN

O *Switch* é o principal elemento de rede no plano de dados nas SDNs, sendo responsável por materializar neste nível da rede todas as ações de encaminhamento definidas pelo controlador.

Observa-se que, a capacidade de reconfiguração ofertada pelos *Switches* OpenFlow está atrelada à capacidade do *hardware* legado em suportar essa demanda. Existe uma grande complexidade nas plataformas legadas em compreender as atualizações apresentadas pelo OpenFlow (BIFULCO; RÉTVÁRI, 2018).

De acordo com Oliveira (2019), dado que os comutadores de dados são construídos sobre plataformas de *hardware* (*ASIC*), a possibilidade de mudança de contexto neste nível é impossível, as configurações pós *bootstrap* são relativas às configurações realizadas no nível de *software*, geralmente através de ferramenta proprietária.

Com o advento dos *Switches* OpenFlow, começa a existir o suporte para a (re)configuração pós *bootstraping*, observa-se que esse suporte se desmembra em duas vertentes: (i) uma forma de descrever requisitos da rede e/ou padrões de processamento de pacotes indepen-

dente de plataforma e (ii) uma interface que traduza ao elemento de rede estas descrições das linguagens de alto nível (OLIVEIRA, 2019).

Esta tendência pode ser observada na proposta *Programmable Protocol Independent Software Switch (PISCES)* (SHAHBAZ et al., 2016), uma modificação do OVS com suporte para o *P4*. O PISCES é uma plataforma *protocol-independent* com a capacidade de alterar o comportamento do *parser*, *match* e *actions* a partir de um código em C gerado pelo compilador P4. De acordo com (SHAHBAZ et al., 2016), essa abordagem tem precedentes em *data centers*, além do fato de reduzir o número de equipamentos físicos, *upgrades* na estrutura física não impactam nos *Switches* virtuais.

Firestone (2017) apresenta em seu trabalho a proposta de um *Switch* virtual SDN para ambiente *Cloud*. Originalmente, o autor desenvolveu um *Virtual Filtering Plataform (VFP)* que foi modificado para um *Virtual Switch*. Como definido pelo autor, o objetivo inicial do projeto era prover uma plataforma programável que habilitaria a utilização de múltiplos controladores de rede. Conforme o autor, a virtualização permite que diversos controladores instaurem suas próprias políticas e regras através de uma plataforma que suporte *deployments* e atualizações sem a necessidade de ser reiniciada.

Huang e Liang (2017) apresenta as dificuldades das propostas de *Switch* SDN em se expressarem sobre plataformas de *hardware* legado. *Switches* SDN devem suportar reconfiguração automática, possuir mecanismos de resposta para lidar com falhas e serem sensíveis a mudança de contexto. Os equipamentos convencionais realizam roteamento de tráfego de forma independente e exigem configurações manuais, ocasionando um gargalo em relação às necessidade das SDN. De forma geral, essas plataformas não conseguem se expressarem de forma efetiva no plano de dados (HUANG; LIANG, 2017).

Sonkoly et al. (2012) de forma especial explana sobre as dificuldades de alcançar *QoS* e *QoE* em soluções baseadas em OpenFlow. A pesquisa relacionada com *QoS* e *QoE* tem tido grande atenção nos últimos anos, porém, os autores acreditam que as arquiteturas fechadas de *Switches* legados tem contribuído significativamente para tornar o provisionamento da QoS um problema permanente.

Ainda sob a óptica de QoS e QoE, Boero et al. (2016) argumenta que a QoS no OpenFlow é bastante limitada. Embora uma ou mais filas podem ser configuradas para cada interface de saída, as entradas de fluxos são mapeadas sempre para uma fila específica e são tratadas de acordo com a configuração predefinida da fila, não podendo, por exemplo, alterar os valores referentes ao parâmetros de QoS.

Fernandes et al. (2020) apresenta a visão de um *Switch* OpenFlow com lógica de operação totalmente alocada no *user-space*. Conforme os autores, abordagens como o OVS podem apresentar dificuldades na adição de novos recursos no elemento de rede, pois toda a capacidade de operação do *Switch* encontra-se provisionada no *kernel-space* e apenas o controle instanciado em *user-space*. Ainda conforme Fernandes et al. (2020), a solução apresenta menor desempenho quando comparadas com outras abordagens, porém,

a adição de novos suportes ao OpenFlow está mais simples.

2.3 Síntese dos Trabalhos Relacionados

Observa-se que os trabalhos relacionados possuem profunda sinergia com a proposta desta tese. São trabalhos que buscam alcançar flexibilidade no plano de dados através do uso de linguagem de programação de alto nível, utilização de mecanismos reconfiguráveis para explorar o paralelismo de tarefas e até mesmo a redistribuição das funções do plano de dados.

Embora nem todas as referências sejam direcionadas para SDN, os trabalhos fizeram surgir importantes comparações e prerrogativa para esta proposta de tese. Dentre elas, as mais expressivas são:

- É notável a complexidade para evoluir o OpenFlow, ampliando o leque de equipamentos, protocolos, cabeçalhos reconhecidos e a forma com que pacotes são processados sem uma contínua modificação em suas especificações. Mesmo o P4, ainda que resolva problemas de escalabilidade do OpenFlow, apresenta dificuldade em se expressar no baixo nível da rede.

- Analisar cabeçalhos *Ethernet*, IP ou até mesmo Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) usando OpenFLow não é um problema. Porém, se for preciso analisar um novo protocolo será necessário adicionar ou aguardar suporte do OpenFLow. Neste contexto o P4 aparece como uma solução elegante e simples.

- As arquiteturas de elemento de rede em SDN não foram projetadas para oferecerem QoS orientado pelas aplicações. Embora diversas soluções ofereçam controle sobre parâmetros tais como prioridade no tráfego e controle de banda, pensar em QoS e no impacto disso no plano de dados não fizeram parte do projeto das soluções.

- O reuso de módulos de *hardware* é uma importante técnica para os cenários de reconfiguração e otimização em soluções baseadas em FPGA, mas a lacuna entre os mecanismos conhecidos de reuso de *software* e reuso de *hardware* para redes ainda é grande.

- O uso de NetFPGA, até certo ponto, habilita a flexibilização no baixo nível. As NetFPGAs podem ser usadas como processador de pacotes num modelo OpenFlow *ethernet-based* e também como plataforma de prototipagem de um *Switch OpenFlow*, porém, devido os recursos de reconfigurabilidade dinâmica serem pouco explorados, a solução acaba se tornando apenas configurável, exigindo reinicialização sempre que houver necessidade de troca de contexto.

- ❑ O conceito de reconfiguração dinâmica em plataformas reconfiguráveis é muito pouco explorado nos trabalhos relacionados (em sua grande maioria apenas a reconfiguração estática é praticada).

Dessa forma pode-se concluir que, mesmo diante de um considerável volume de soluções centradas em *hardware*, utilizando FPGA, NetFPGA e SoC, uma grande tendência na utilização de soluções centradas em *software* é observada. Os fabricantes de *Switches* vem apostando em soluções focadas em *software* para aumentar a capacidade de programabilidade de seus equipamentos exigindo cada vez menos dependência do *hardware* legado.

Observando as soluções centradas em *software*, a principal desvantagem dessas abordagens são os requisitos de QoS não considerados ou parcialmente considerados no projeto da solução. As soluções tendem a implementar ações de encaminhamento baseadas em fluxo ou porta física e não por aplicações.

Devido a Internet fazer uso do protocolo *Ethernet* como um dos principais protocolos de camada 2, todos os fabricantes de NICs produzem interfaces de rede *Ethernet-Based*. Projetar um elemento de rede que não tenha suporte para este padrão significa desenvolver uma solução inelástica, entrando em discordância com os objetivos deste trabalho.

Não fez parte dos objetivos deste trabalho redesenhar o controle de acesso ao meio, contudo, atuar diretamente nas funções já predefinidas do MAC, orquestrando as ações de encaminhamento e não apenas instituindo fluxos de dados, permitiu alcançar elasticidade e flexibilidade no plano de dados em redes SDN.

A arquitetura NEA quando comparada aos trabalhos relacionados, possui a capacidade de expor, através de uma fina granularidade, a lógica referente às políticas de encaminhamento de baixo nível ao plano de controle por meio de um módulo orquestrador. A NEA permite (re)programação de forma sistemática orientada pelos requisitos de aplicação, ofertando melhoria na QoS de usuários.

A Tabela 1 ilustra as principais soluções apresentadas na Seção 2.2 em comparação com a NEA. Através da Tabela 1 é possível observar o tipo da solução,sendo abordagens centradas em *Software*, *Hardware* ou ambas. Além da manipulção dos principais parâmetros que impactam na *Quality of Service (QoS)* (Prioridade, Controle de Banda e *Burst*), foi comparado a atuação da solução em relação às ações de encaminhamento sob a óptica da granularidade (Fluxo, Porta e Aplicação), e por último a capacidade da solução em expressar um MAC programável, permitindo um ajuste fino no controle de inserção de primitivas no canal de transmissão.

Tabela 1 – Visão Geral dos Trabalhos Relacionados

Trabalho	**Ano**	**Abordagem**	**Prioridade**	***Rate***	***Burst***	**Ações Orientadas por:**	**MAC programável**
OpenFlow (FOUNDATION, 2015)	*2015*	Software	*Sim*	*Sim*	*Sim*	*Fluxo*	*Não*
P4 + Asterfusion (ASTERFUISON, 2022)	*2022*	Software + Hardware	*Sim*	*Sim*	*Não*	*Fluxo ou Porta*	*Não*
OVS (PROJECT, 2016a)	*2016*	Software	*Sim*	*Sim*	*Sim*	*Porta*	*Não*
SwitchDev (QRATOR, 2020)	*2020*	Software + Hardware	*Sim*	*Sim*	*Sim*	*Porta*	*Não*
Switch Etarch (OLIVEIRA, 2019)	*2020*	Hardware	*Sim*	*Não*	*Não*	*Workspace*	*Sim*
Switch NEA	*2022*	Software	*Sim*	*Sim*	*Sim*	*Aplicações e Workspace*	*Sim*

Capítulo 3

Método

Este capítulo se destina a detalhar o método de pesquisa adotado para este trabalho, sendo as principais contribuições: (i) descrever a sequência de passos realizados na construção da proposta; e (ii) definição de seu escopo de trabalho.

O método utilizado, conforme Prodanov e Freitas (2013) e Lakatos e Marconi (2003), trata de um método hipotético-dedutivo que se inicia com a percepção de um problema, passando pela formulação da hipótese de pesquisa e instituição de uma solução para testar a predição de ocorrência da referida hipótese.

De acordo com Wazlawick (2009), o objetivo da pesquisa tem notável impacto nas decisões relacionadas ao método, portanto, a definição clara e direta dos objetivos faz-se necessário para construção do método. Dado que este trabalho possui como objetivo principal, o desenvolvimento de uma arquitetura para um elemento de rede com MAC definido pela aplicação juntamente com um protótipo de um *Switch*, o método deve contemplar as etapas de especificação, desenvolvimento e validação desta proposta através de um ambiente de testes, que neste trabalho foi instituído como estudo de caso.

Na Figura 14 é possível observar de forma sistemática o método aplicado neste trabalho, descrevendo visualmente todo o processo de construção da tese.

3.1 Caracterização do Problema

Importante ressaltar que este trabalho possui foco no *Switch*, que é responsável pelo encaminhamento de primitivas no plano de dados, atuando sob gestão do controlador da rede.

3.1.1 Percepção da Incapacidade de Elementos de Rede em Atuar na Garantia da QoS de comunicações

A caracterização do problema se iniciou com a disparidade das funcionalidades da subcamada MAC em relação à capacidade de garantia de QoS nas SDN. Em geral, as

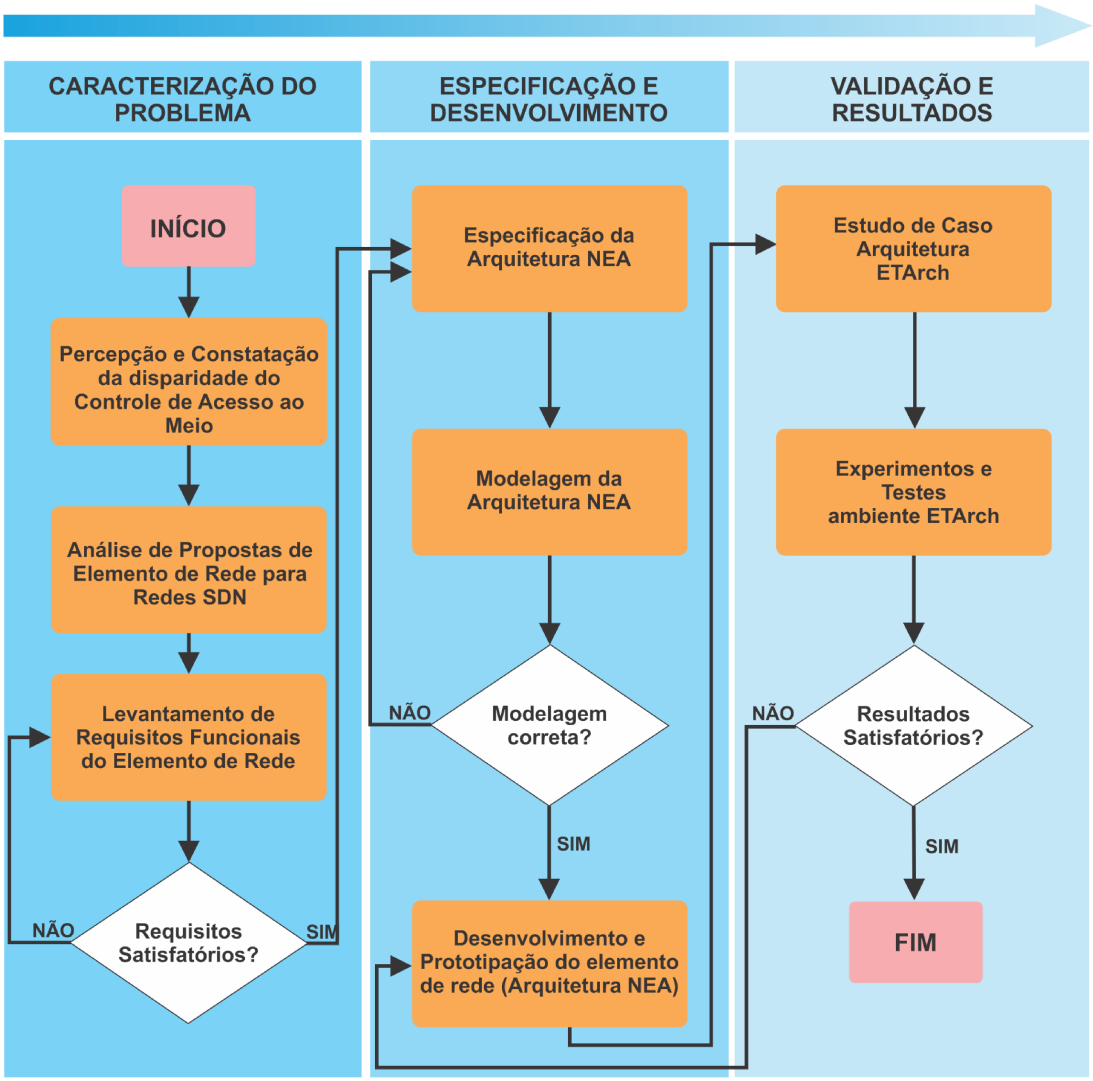


Figura 14 – Método: Fluxograma de Atividades

SDNs assumem que existe uma estrutura física de encaminhamento no plano de dados capaz de compreender e atuar convincentemente sob a supervisão do controlador da rede. Fato é que, este comportamento limita a atuação do plano de dados em relação aos requisitos das comunicações (MICHEL et al., 2021; OLIVEIRA, 2019; KALJIC et al., 2019).

Comumente, as propostas com foco no plano de dados são centradas em *software* e apoiadas em plataformas legadas. Embora seja possível alcançar um certo nível de programabilidade, o MAC é o mesmo. De forma geral, a flexibilidade existe, porém, limitada, ora em relação a impedimentos da própria solução, ora na incapacidade da plataforma em atuar sobre regras de encaminhamento através de uma granularidade mais fina e não simplesmente na instituição de fluxos.

3.1.2 Análise de Propostas de Elemento de Rede para Redes SDN

Esta etapa foi responsável pela análise de propostas expressivas e alinhadas com os objetivos desta tese, ou seja, propostas que buscam flexibilidade e programabilidade no plano de dados com foco na evolução dos elementos de rede em SDN. As análises foram divididas em duas vertentes, soluções baseadas em *software* e *hardware*.

Para servir de direcionamento na análise das propostas, esta etapa envolveu:

- ❏ Análise de soluções implementadas em *software* e *hardware* que objetivam programabilidade no plano de dados em SDN;
- ❏ Compreensão dos aspectos de flexibilidade no plano de dados em SDN;
- ❏ Análise dos mecanismos de encaminhamento de dados em soluções para SDN;

Em suma, esta etapa apresentou notável importância para evolução desta tese, pois, foi possível identificar que as abordagens centradas em *software* detêm grande parte da atenção em relação às implementações de elementos de rede no plano de dados nas SDNs. Foi possível também identificar através desta etapa que as soluções centradas em *hardware*, apesar de promissoras, se posicionam como melhoramentos pontuais em soluções centradas em *software*.

3.2 Levantamento de Requisitos

Embora as aplicações de rede possuam diferentes requisitos de comunicação, foi importante nesta etapa, realizar o levantamento dos requisitos essenciais para que uma comunicação possa, de fato, acontecer integralmente em uma rede SDN. Devido a escolha da ETArch como estudo de caso, faz-se necessário a compreensão do contexto de comunicação em um ambiente baseado em *Workspace*.

Para direcionar a construção do conjunto de requisitos da arquitetura NEA, esta etapa envolveu:

- ❏ Análise operacional de elementos de rede com foco no suporte para estabelecimento de comunicação em SDN.
- ❏ Compreensão das necessidades para instituição e manutenção de fluxos de comunicação em elementos de rede.
- ❏ Definição dos mecanismos essenciais para o funcionamento de elementos de rede;
- ❏ Compreensão das necessidades básicas para comunicação baseada em *Workspace*;

- ❏ Refinamento da análise realizada nos tópicos anteriores com base no Modelo de Títulos;

Esta etapa teve como objetivo a instituição do conjunto de requisitos funcionais da arquitetura NEA, para auxiliar neste levantamento, foi importante compreender, mesmo que de forma primitiva, as atribuições inerentes ao controlador da rede e do elemento de rede em um ambiente SDN.

3.3 Especificação e Desenvolvimento da Arquitetura NEA

Finalizado o levantamento de requisitos, iniciou-se a etapa de especificação e desenvolvimento da arquitetura NEA. Esta etapa foi divida em três momentos: (i) Especificação da arquitetura NEA, (ii) modelagem da solução, e (iii) desenvolvimento de um protótipo de *Switch* baseado na arquitetura NEA.

3.3.1 Especificação da Arquitetura NEA

A especificação da arquitetura NEA compreendeu, inicialmente, um estudo nos principais sub-módulos do Kernel do Linux, tais como: *Bridge*, *Iproute*, *Netfilterqueue*, *iptables*, *ebtables* e *Traffic Control (TC)*, embora estes representem os principais elementos relativos ao encaminhamento de primitivas no Kernel, nem todos foram utilizados na NEA. Este estudo foi importante pois, permitiu obter uma visão clara da organização da arquitetura, contribuindo para a modelagem da NEA.

3.3.2 Modelagem da Arquitetura NEA

Tendo grande aceitação na modelagem de soluções na área da computação, as *FSMs* são sistemas algébricos que podem ser divididos em duas categorias, os Autômatos Finitos (Transdutores) com saída e as Reconhecedoras (Aceitadores) de linguagens, independente da categoria, ambos objetivam assegurar o funcionamento de uma solução (JÚNIOR et al., 2013).

Para a modelagem da arquitetura NEA foi utilizado FSM, especificamente, a Máquina de Mealy (MEALY, 1955), que permitiu compreender todos os processos da arquitetura e suas respectivas interfaces. Embora a Máquina de Mealy seja equivalente à Máquina de Moore (MOORE et al., 1956), para o contexto desta tese a Máquina de Mealy se apresenta como melhor opção para a modelagem pois, as palavras de saída são oriundas das transições e não dos estados, sendo possível para um mesmo estado várias transições com saídas diferentes.

Este processo foi importante pois permitiu instituir uma etapa de validação antes da implementação do protótipo.

3.3.3 Desenvolvimento e Prototipação do *Switch* baseado na Arquitetura NEA - *Switch NEA*

Apesar de ter sido possível desenvolver as etapas de especificação e modelagem da arquitetura concomitantemente, ambas foram realizadas sequencialmente e após o término, deu-se início a ao desenvolvimento da arquitetura NEA. Esta etapa foi finalizada com a confecção do protótipo do *Switch* (*Switch NEA*).

Sob uma perspectiva operacional, o *Switch* baseado na arquitetura NEA atua em duas vertentes, são elas: (i) comunicação com o controlador da rede e (ii) manipulação do MAC.

A comunicação com o controlador foi instituída através de um protocolo de controle. Para este trabalho, foi utilizado o protocolo *ETSCP* (OLIVEIRA, 2019) com adequações no cabeçalho e adição de um novo serviço.

Com relação ao MAC programável, que compreende as ações de encaminhamento, foi instituído uma aplicação que atua como orquestrador instanciado no *user-space* do linux. Esta aplicação possui a capacidade de interagir com os módulos do *kernel-space* e também manusear parâmetros de QoS com impacto direto nas ações de encaminhamento do elemento de rede e por conveniência no MAC.

Para confecção do protótipo foi utilizado um *hardware* computacional tradicional equipado com três interfaces de redes para representar a estrutura do *Switch*. A seguir tem-se a descrição das ferramentas e plataformas utilizadas para a prototipagem:

- *Hardware* Computacional equipado com 2 (duas) interfaces de rede *Offboard* (Placa Rede Gigabit Pci Express Tp Link Tg-3468 10/100/1000), 1 (uma) Interface Gigabit LAN Realtek RTL8111H *Onboard* 10/100/1000, HD sata ADATA 240Gb, Processador Pentium G4400 3.30Ghz, Memória Ram de 8Gb DDR4 2133Mhz Crucial e Sistema Operacional Linux Ubuntu 20.04.

- Módulos habilitados no Kernel do Linux: *Iproute2, Forwarding, Ebtables, 802.1Q, Nftables, Bridge, Netfilterqueue e Traffic Control (TC).*

- Python versão 3.8.5;

- 4 cabos UTP padrão RJ45;

- IDE de programação Visual Code Versão 1.61.0;

3.4 Estudo de Caso

Estudos de Caso são uma metodologia de pesquisa que permite se aprofundar sobre um assunto específico e oferecer subsídios para novas investigações sobre a temática (YIN, 2015). A escolha da ETArch como estudo de caso se deu pela familiaridade da arquitetura com os objetivos desta tese. Dado que a ETArch foi instituída com suporte nativo à requisitos de aplicação e é uma arquitetura justificável para tratar QoS de usuários.

O Estudo de Caso teve notável importância na validação deste trabalho, uma vez que foi possível compreender a capacidade do protótipo baseado na arquitetura NEA em comportar as especificidades de uma arquitetura SDN sem alterar o suporte já existente para arquitetura Internet.

3.5 Experimentos e Validação

Nesta etapa foi instituída a validação do trabalho proposto. A modelagem da arquitetura NEA realizada na macro etapa de desenvolvimento já se apresentou como uma fase de validação, permitindo verificar e redefinir os processos e interfaces de comunicação antes da etapa de implementação.

Nesta etapa do método, os testes foram voltados para o protótipo desenvolvido, sendo divididos em dois momentos. No primeiro momento, avaliou-se a capacidade do protótipo em receber primitivas de controle, compreender e atuar sobre a estrutura de encaminhamento. Neste momento as métricas de avaliação foram pautadas nos requisitos funcionais da NEA.

No segundo momento, verificou-se a capacidade do protótipo em relação à manipulação dos parâmetros de QoS em um ambiente com tráfego real. Neste momento avaliou-se a capacidade do protótipo em relação à priorização e controle de banda de tráfego definidos pelo controlador da rede, constatando a atuação do protótipo na garantia da QoS de aplicações. Na Figura 15 é possível observar o ambiente de validação instituído nesta etapa.

O ambiente proposto para experimentação foi composto de duas estações físicas (*Laptops*), com aplicações distintas representando as *Entidades*. Para o papel de controlador, desenvolveu-se uma versão *lightweight* do *Domain Title Service Agent (DTSA)* capaz de receber e enviar primitivas de controle (ETSCP) para o elemento de rede.

Torna-se importante ressaltar que nesta etapa de testes foram instituídos diversos cenários de experimentação que serão detalhados no Capítulo 7. Toda a etapa de experimentação e testes foi conduzida através de processos e metas bem definidos. Na Tabela 2 é possível observar as metas que foram utilizadas nos testes. O processo (P) é inerente à tarefa executada, a meta (M) trata do resultado esperado.

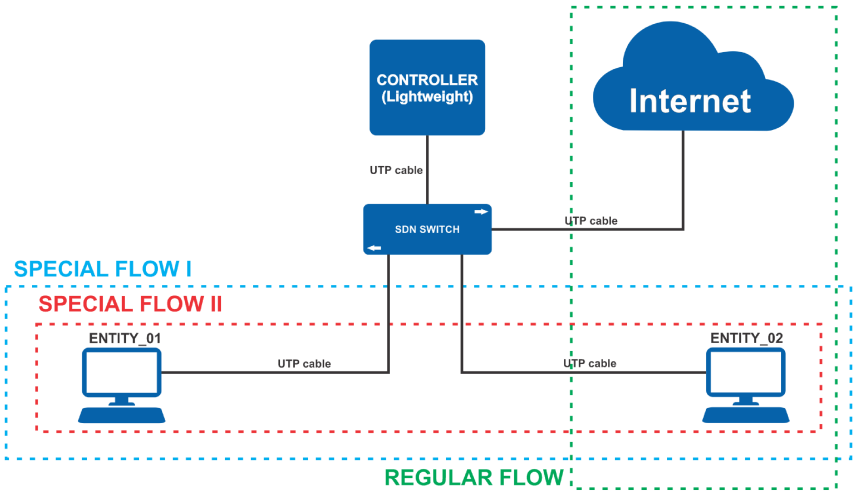


Figura 15 – Ambiente instituído para condução dos experimentos

Antes de descrever o processo de desenvolvimento da NEA, é importante detalhar o levantamento de requisitos necessários para confecção da arquitetura e protótipo.

Tabela 2 – Processos a serem avaliados do elemento de rede

Ident.	**Meta**	**Descrição**
MET1	*P1: Conexão entre o Elemento de Rede e Entidades.* *M1: Tabela de encaminhamento construída e escalonamento instituído.*	Avaliar a capacidade de construção das tabelas de encaminhamento e instituição das políticas de escalonamento baseadas nos requisitos das aplicações.
MET2	*P2: Mensagens ETSCP.* *M2: Mensagens ETSCP do Controlador interpretadas e tratadas.*	Testar recebimento e reconhecimento de mensagens de configuração recebidas do Controlador (DTSA.)
MET3	*P3: Criação e manutenção de rotas baseadas em Workspaces.* *M3: Tabela de Workspace construída/atualizada.*	Avaliar a capacidade do elemento de rede com relação à criação, atualização e exclusão de *Workspaces*.
MET4	*P4: Envio simultâneo de mensagens por diferentes entidades.* *M4: Comunicação entre as entidades instituída.*	Testar transmissão e entrega simultânea de dados entre as entidades conectadas no elemento de rede.
MET5	*P6: Atuar sobre parâmetros de QoS.* *M5: Analisar o impacto da alteração dos parâmetros de QoS nas comunicações.*	Avaliar a capacidade de atuação do elemento de rede em relação a gestão dos parâmetros de QoS.
MET6	*P6: Status do elemento de rede.* *M6: Avaliar o consumo de memória e processador do elemento de rede.*	Avaliar a utilização dos recursos do elemento de rede.

Capítulo 4

Requisitos da Arquitetura NEA

A arquitetura NEA foi materializada em um protótipo de um *switch* capaz de oferecer conectividade física a *Entidades* em uma rede SDN. No plano de Dados em SDNs, o *Switch NEA* é o elemento de rede responsável por operacionalizar regras de encaminhamento especificadas por controladores. Neste contexto, levantar requisitos significa compreender as particularidades das comunicações suportadas por planos de dados.

No contexto de aplicações, comunicações possuem finalidades distintas, com requisitos distintos. Naturalmente, comunicações são suportadas por infraestrutura de encaminhamento de dados, sendo que, aquilo que é necessário, por exemplo em termos de segurança, para uma aplicação bancária é diferente daquilo que é exigido para *download/upload* de imagens em redes sociais.

De forma geral, *switches* são equipamentos que devem promover conectividade a *end points* (notebooks, smartphones, tablets, desktops etc) que hospedam entidades e, diante desta premissa, requisitos como recebimento/envio de *frames* e *forwarding* são serviços essenciais básicos do equipamento (TANENBAUM, 1996). Mesmo em SDN, observa-se que funcionalidades de *switches* permanecem, ainda, inalteradas. Todavia, estes equipamentos vêm experimentando novas demandas, exigindo atualizações.

Com base nos trabalhos correlatos apresentados na Seção 2.2, institui-se os requisitos essenciais para que o *Switch* NEA suporte comunicações em redes SDN. Estes requisitos são organizados e apresentados na Tabela 3.

Além de requisitos básicos, tais como receber/enviar *frames* de portas físicas, o *Switch* NEA deve gerenciar tabelas de encaminhamento em tempo-real, visando garantir QoS/QoE. Em SDN, o *switch* pode ser controlado por um ou mais controladores, que podem enviar primitivas de controles assincronamente ao *switch* controlado. Importante ressaltar que esses requisitos podem ser refinados em diversos processos na fase de desenvolvimento e acabam por definir o que é denominado núcleo básico de funcionalidades do *switch*.

Requisitos como mobilidade, capacidade de tratar a mudança de localidade de *Entidades* de forma transparente; e *self-healing*, capacidade de identificar situações de falhas e buscar a auto-correção, apesar de não fazerem parte do escopo deste trabalho, são re-

Tabela 3 – *Switch* NEA: Requisitos de suporte e manutenção de comunicações em SDN

Identificação	**Requisito**	**Descrição**
R01	*Encaminhamento*	Elemento de rede deve ser capaz de receber, processar e encaminhar dados através da análise do cabeçalho das primitivas.
R02	*Multi-Controlador*	Elemento de rede deve ser capaz de ser supervisionado por vários controladores de rede.
R03	*Qualidade de Serviço*	Elemento de rede deve ser capaz de atuar sobre os parâmetros de controle de tráfego com impacto na QoS e QoE das comunicações.
R04	*Segurança*	Elemento de rede deve implementar mecanismos segurança.
R05	*Auto-cura*	Elemento de rede deve ser capaz de identificar situações de falhas e tentar corrigi-las.
R06	*Mobilidade*	Elemento de rede deve ser capaz de suportar a mudança de localidade de entidades (*User Equipment*) sem prejudicar as regras de encaminhamento.

quisitos que indicam uma evolução no projeto e implementação de *switches*, agregando mais capacidade e autonomia em SDN.

Requisitos de segurança, por se tratar de um tópico extenso e complexo, também não é escopo deste trabalho, ficando para trabalhos futuros.

4.1 Requisitos do *Switch* NEA para a Arquitetura ETArch

A ETArch foi escolhida como estudo de caso deste trabalho e nesta etapa de método contribui para o refinamento dos requisitos do elemento de rede. A escolha da ETArch se deu pelo fato de a mesma ser uma arquitetura que trata de requisitos de QoS orientado por aplicações desde o projeto arquitetônico e mesmo após 10 (dez) anos de sua concepção, ainda apresenta inovações neste quesito. Oliveira (2019), em seu estudo, utiliza a ontologia apresentada por Pereira (2012b) para definição dos requisitos da arquitetura ETArch. De acordo com Oliveira (2019), Pereira (2012b) apresenta a definição de classes e suas hierarquias para a construção da ontologia do Modelo de Título de Entidade. Na Figura 16 é ilustrado o nível mais alto da ontologia, expandindo para subníveis em função da aplicabilidade na arquitetura ETArch.

De acordo com a Figura 16, na parte inferior é possível identificar as subclasses baseadas na ETArch, para a subclasse *Camadas*, existem as subclasses: *Aplicação*, *Comuni-*

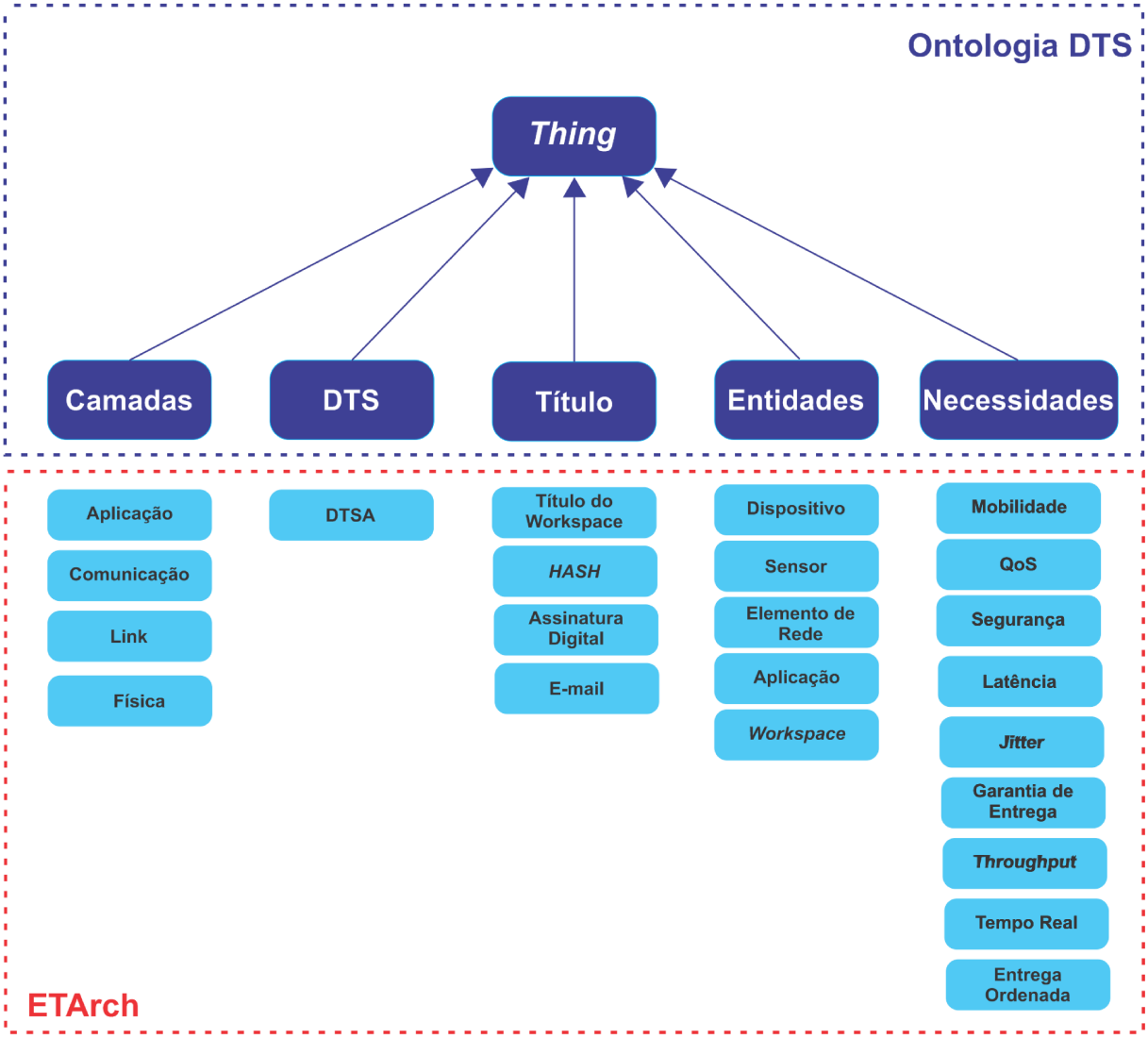


Figura 16 – Definição de Requisitos Ontologia DTS/ETArch. Figura adaptada de Pereira (2012b), Oliveira (2019)

cação, *Link* e *Física*, em relação à subclasse *DTS* existe a subclasse *DTSA* e assim por diante (OLIVEIRA, 2019).

Esta análise se concentrou nas subclasses relacionadas ao *switch*, que são especificadamente as subclasses *Entidades* e *Necessidades*. Lembrando que, de acordo com Pereira (2012b), uma Entidade pode ser instância de sensores, dispositivos de rede, elementos de rede, aplicações em *end-points*, *Workspaces*, dentre outras.

Foi realizado um refinamento nos trabalhos de Pereira (2012b) e Oliveira (2019), para reorganizar a subclasse Necessidades e seus atributos, visando facilitar a compreensão dos requisitos do *switch*. A Figura 16 ilustra requisitos básicos e avançados para o estabelecimento de conexões em *switches*.

Algumas subclasses foram realocadas com o intuito de facilitar a compreensão das

necessidades, como por exemplo, a classe *forwarding*, que trata de atributos básicos de recebimento, processamento e envio de *frames*. Essa redistribuição e organização pode ser visualizada na Figura 17.

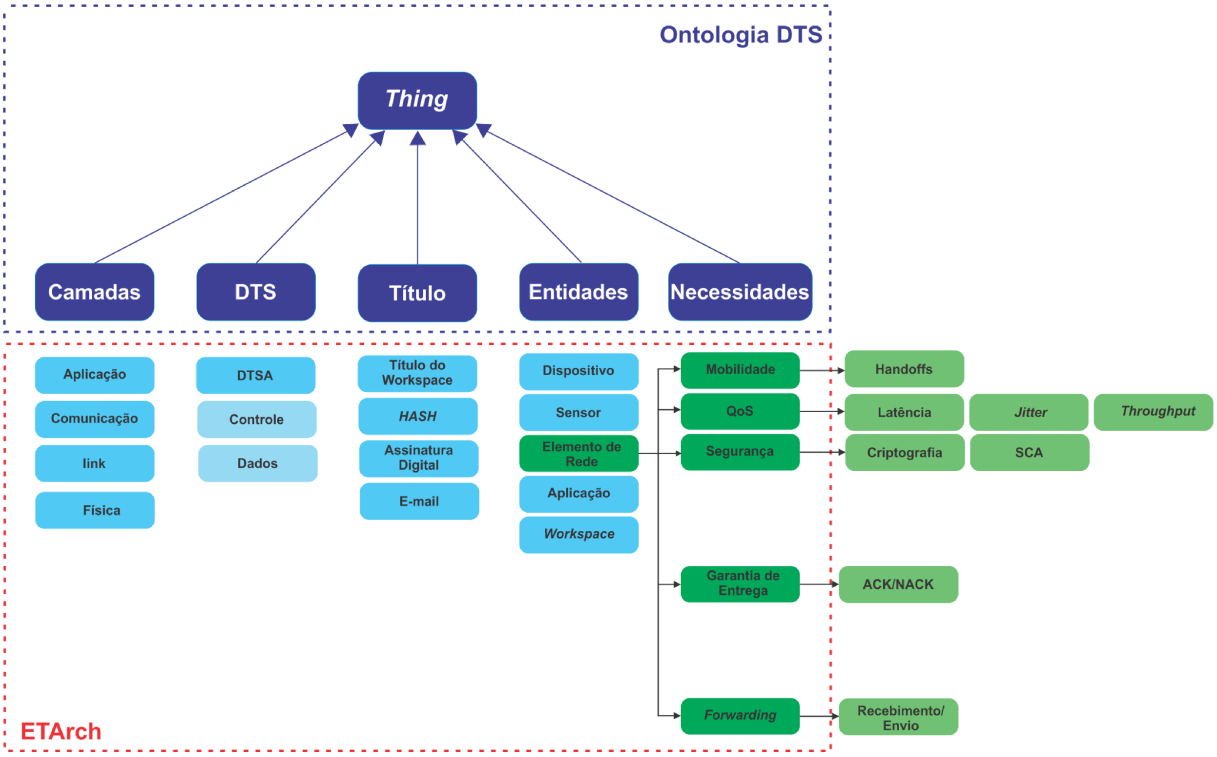


Figura 17 – Requisitos redistribuídos e reorganizados - Figura adaptada de Pereira (2012b), Oliveira (2019)

De acordo com Oliveira (2019), a definição de quais requisitos são básicos e quais são avançados se baseiam no princípio da comunicação. Requisito básico é tudo aquilo que seja o mínimo para garantir a comunicação entre duas entidades. Os demais requisitos, neste contexto, são considerados como avançados. Na Figura 18 pode ser visualizada a organização dos requisitos básicos e avançados em relação às necessidades da arquitetura ETArch.

Muitos requisitos são importantes para a materialização de comunicações em ambientes ETArch e em cenários SDNs. Algumas das capacidades de *switches*, em relação à ETArch, são:

- Diferenciar títulos como identificador único de entidades;
- Oferecer suporte para criação e manutenção de fluxos baseado em *Workspaces*;
- Permitir vários fluxos baseado em *Workspaces* pela mesma porta física;
- Instituir e gerenciar políticas de escalonamento de primitivas;

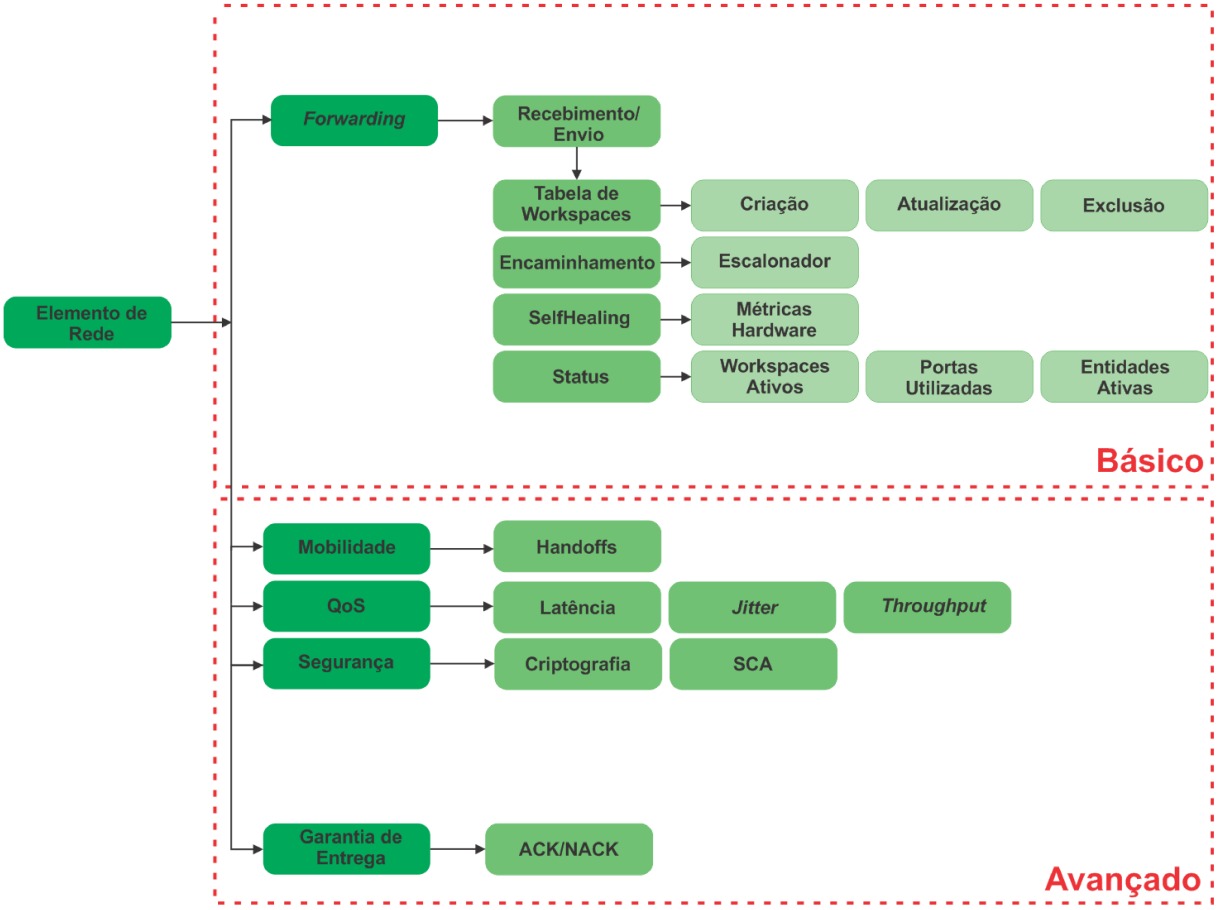


Figura 18 – Requisitos refinados e organizados (Básicos e Avançados). Figura adaptada de Pereira (2012b), Oliveira (2019)

- Registrar a utilização de recursos de *hardware*, tais como processamento e memória; e
- Apresentar *Status* do equipamento com relação a quantidade de *Workspaces* ativos, portas físicas e *Entidades* ativas.

Com relação a requisitos avançados, segue uma lista não exaustiva de possibilidades, tais como:

- Permitir a alteração física de *Entidades* (User Equipment (UE)) de forma transparente;
- Manusear parâmetros de QoS e QoE orientados pela aplicação, tais como prioridade e largura de banda;
- Oferecer mecanismos de segurança, tais como criptografia e métodos para conter *Side Channel Attackss (SCAs)*;
- Garantir a entrega de primitivas em comunicações baseadas em *Workspace*.

4.2 Requisitos da Arquitetura NEA

A análise dos trabalhos correlatos, o levantamento de requisitos advindos das propostas de *switches* SDNs e, também, a (re)avaliação da ontologia DTS/ETArch, culmina com o conjunto de requisitos funcionais e não funcionais do *Switch* NEA.

É importante frisar que não faz parte do escopo deste trabalho, para o protótipo desenvolvido, a implementação de todos os requisitos (funcionais ou não funcionais). Para esta tese, serão implementados os requisitos básicos e somente aqueles avançados relativos a QoS e QoE. Os demais requisitos avançados podem vir a ser implementados em trabalhos futuros.

Para facilitar a análise dos requisitos levantados, eles foram divididos em duas categorias, a saber: (i) Requisitos Funcionais e (ii) Requisitos não funcionais.

Cada requisito especificado representa um conjunto de tarefas a ser implementado na etapa de desenvolvimento. Na Tabela 4 podem ser observados em detalhes os requisitos funcionais e não funcionais da Arquitetura NEA. Para melhorar a compreensão, foram instituídos requisitos de execução, que são desdobramento dos requisitos funcionais e refletem o funcionamento do elemento de rede.

De posse do conjunto de requisitos, foi possível observar de forma clara as funções que o elemento de rede deve desempenhar para satisfazer as necessidades das comunicações em redes SDN. De forma articulada, os requisitos contribuíram para a modelagem dos processos inerentes aos elemento de rede, os quais são apresentados no capítulo 5.

Tabela 4 – Requisitos do *Switch* NEA

Identificador	**Requisito**	**Descrição**
RF01	*Encaminhamento 01*	Elemento de rede deve ser capaz de compreender e processar (*frames*) *Ethernet*.
RF02	*Encaminhamento 02*	Elemento de rede deve ser capaz de compreender e processar (*frames*) ETArch.
RF03	*Encaminhamento 03*	Elemento de rede deve permitir diferentes fluxos orientados à aplicação através da mesma porta física.
RF04	*Encaminhamento 04*	Elemento de rede deve ser capaz de construir tabela de encaminhamento baseada em *Workspace*.
RF05	*Encaminhamento 05*	Elemento de rede deve ser capaz de instituir, atualizar e excluir rotas de encaminhamento baseada em *Workspaces*.
RF06	*QoS 01*	Elemento de rede deve ser capaz de atuar sob parâmetros de QoS e QoE para atender os requisitos de aplicação.
RF07	*QoS 02*	Elemento de rede deve ser capaz de instituir diferentes formas de escalonamentos por porta física do elemento de rede.
RF08	*Connection 01*	Elemento de rede deve ser capaz de emitir *ACK* ou *NACK* para o controlador da rede.
RE01	*Exec 01*	Elemento de rede deve ser capaz de analisar e preservar informações presentes nos cabeçalhos dos *frames*.
RE02	*Exec 02*	Elemento de rede deve ser capaz de identificar e visualizar todos os *frames* para encaminhamento.
RE03	*Exec 03*	Elemento de rede deve ser capaz de aplicar diferentes formas de escalonamento em função das arquiteturas suportadas.
RNF01	*Desempenho 01*	Elemento de rede deve apresentar alto desempenho nas funções de encaminhamento.

Capítulo 5

Desenvolvimento *Switch* NEA

Para facilitar a compreessão, o desenvolvimento é apresentado em três seções, quais sejam: (i) na Seção 5.1 é apresentada a modelagem do *Switch* NEA utilizando FSM; (ii) na Seção 5.2 é apresentada a especificação da arquitetura e o suporte do Sistema Operacional Linux; e (iii) na Seção 5.3 são apresentadas as políticas de escalonamento orientadas pela aplicação.

5.1 Diagrama de Estados da Arquitetura NEA

O conceito de máquinas de estados finitas permite especificar o comportamento de autômatos a partir de estados, transições, eventos e ações. Cada estado presente na modelagem representa um processo no sistema e as transições são associadas a eventos de entrada e saída, para os quais são especificadas ações associadas. Portanto, através da modelagem é possível descrever todas as situações discretas que o sistema possa apresentar. O comportamento do *Switch* NEA pode ser definido formalmente pela expressão M (1), na qual M representa a Máquina de Estados Finita:

$$M = (\Sigma,\ S,\ S\circ,\ \delta,\ F) \tag{1}$$

O símbolo Σ representa o alfabeto de entrada; S representa o conjunto de estados que o sistema pode assumir; $S\circ$ representa o estado inicial; δ representa as funções de transições dos estados; e, por fim, F representa um subconjunto de S que representa os possíveis estados finais da máquina.

Para a construção do diagrama de estados da arquitetura NEA foi utilizado o modelo de máquina de estados finita conhecido como Máquina de Mealy (SHAHBAZ; GROZ, 2009). Neste modelo, uma determinada saída depende do estado atual da máquina e também do valor da entrada, e cada transição é representada pela combinação de entrada(s) e saída(s), na forma $\frac{entrada}{saída}$.

Na Figura 19 é apresentado o diagrama de estados do *Switch* NEA. Neste contexto, os sinais de entrada, em sua maioria, são primitivas de dados, sendo que, em alguns casos,

pode ser uma variável de controle do processo. Os estados descritos representam um comportamento específico em relação à natureza da primitiva.

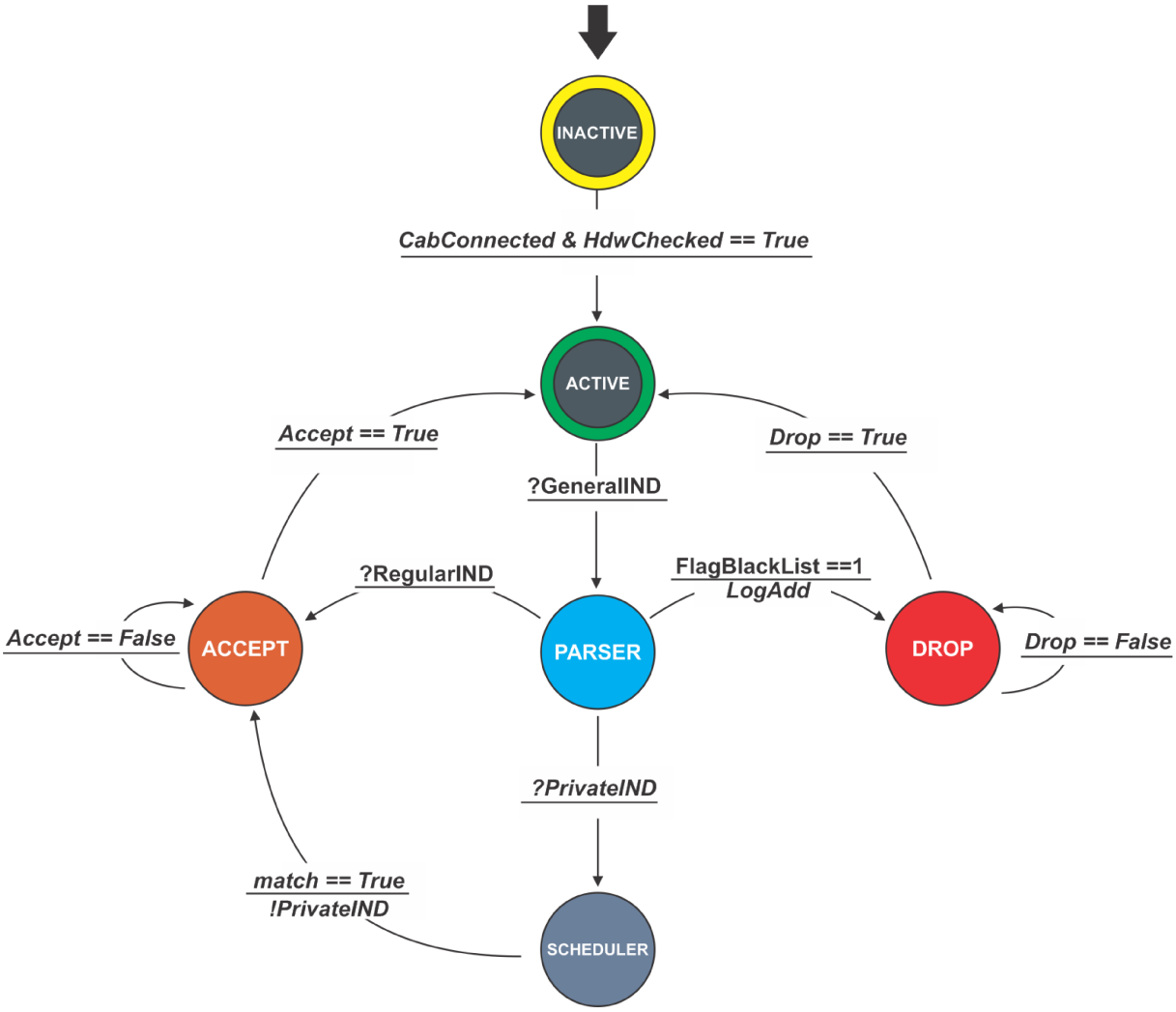


Figura 19 – Diagrama de Estados da Arquitetura NEA

Na Figura 19 o estado *INACTIVE* representa o *Switch* NEA desligado, que quando ligado, verifica o *hardware* e a conexão de um cabo de rede e transita para o estado *ACTIVE*. O protótipo não faz verificações posteriores, uma vez que o objetivo desta tese é provar que as aplicações podem definir os requisitos da QoS e estes são garantidos mesmo que compartilhem a mesma porta física.

Para melhorar a legibilidade do autômato, não estão compreendidas na Figura 19, as transições relativas à inativação da porta física, onde o controlador a qualquer momento pode desabilitar a interface do *Switch*, e as tratativas inerentes às primitivas do plano de Controle. No Capítulo 6 será abordado com mais detalhes o comportamento do *Switch* NEA em função do protocolo do plano de controle *ETSCP*.

A máquina de estados finita do *Switch* NEA pode ser descrita da seguinte forma:

- ❏ Σ: GeneralIND, RegularIND, PrivateIND, match==True, Drop==True, Drop==False, Accept==True, Accept==False, FlagBlackList==1, LogAdd, Cable Connected==True e Hardware Checked==True;
- ❏ S: INACTIVE, ACTIVE, PARSER, DROP, ACCEPT e SCHEDULER;
- ❏ $S\circ$: INACTIVE;
- ❏ δ: $S\ X\ \Sigma \rightarrow S$; e
- ❏ F: ACTIVE.

As boas propriedades devem ser garantidas para evitar mau funcionamento do autômato. *Safety* visa garantir que nenhum estado não desejado seja atingido durante a execução do sistema. *Liveness* garante que um estado desejado deve, eventualmente, ser atingido, ou seja, existe sempre uma sequencia de transições que levem ao estado inicial do sistema. *Bounded* é uma propriedade que está relacionada à essência de autômatos, garantindo que a máquina é finita em termos do número de estados. Ausência de *Deadlock* é uma propriedade que garante que um autômato não atinja um estado a partir do qual não exista transição de saída, ou seja, evita um bloqueio em um determinado estado que resulte no travamento do sistema. Existem outras propriedades de autômatos, mas essas são as de interesse para a presente tese.

Todo estado presente no diagrama da Figura 19 possui ao menos uma transição definida por δ: $S\ X\ \Sigma \rightarrow S$, sendo assim o conjunto de transições definido por δ pode ser especificado da seguinte maneira:

$$\{INACTIVE\,(\text{"}CabConnected\&HdwChecked == True\text{"}) \rightarrow ACTIVE\}$$
$$\{ACTIVE\,(\text{"}GeneralIND\text{"}) \rightarrow PARSER\}$$
$$\{PARSER\,(\text{"}RegularIND\text{"}) \rightarrow ACCEPT\}$$
$$\{PARSER\,(\text{"}PrivateIND\text{"}) \rightarrow SCHEDULER\}$$
$$\{PARSER\,(\text{"}FlagBlackList == 1\text{"}) \rightarrow DROP\}$$
$$\{SCHEDULER\,(\text{"}match == true\text{"}) \rightarrow ACCEPT\}$$
$$\{ACCEPT\,(\text{"}Accept == True\text{"}) \rightarrow ACTIVE\}$$
$$\{ACCEPT\,(\text{"}Accept == False\text{"}) \rightarrow ACCEPT\}$$
$$\{DROP\,(\text{"}Drop == True\text{"}) \rightarrow ACTIVE\}$$
$$\{DROP\,(\text{"}DROP == False\text{"}) \rightarrow DROP\}$$

É importante lembrar que a máquina de estados M é instanciada para cada primitiva recebida pelo *Switch* NEA, permitindo vislumbrar a importância de uma implementação em *hardware* para atingir desempenhos a partir de um certo ponto. Para este protótipo,

não foi feita bi-simulação (*hardware and software codesign*), uma vez que o objetivo principal é provar a capacidade de tratar múltiplos requisitos de aplicações cujos planos de dados passem pelo *Switch* NEA.

A partir do estado *ACTIVE*, ao receber uma primitiva (indicado pelo símbolo ?$GeneralIND$, o autômato transita para o estado *PARSER*, no qual a primitiva será analisada, de acordo a regras estabelecidas na Tabela de Chaveamento, podendo ocorrer o seguinte:

- ❏ Caso haja uma regra de descarte, ela seguirá para o estado *DROP*;
- ❏ Caso haja uma regra de requisitos (QoS/QoE), ela seguirá para o estado *SCHEDULER*, com uma indicação de ?$PrivateIND$; ou
- ❏ Caso não haja requisitos especificados para a corrente primitiva, ela seguirá para o estado *ACCEPT*, numa situação ?$RegularIND$ em que nenhum requisito de prioridade ou largura de banda é requerido.

Observe que, com exceção da transição *PARSER(FlagBlackList==1 -> DROP)* (e as transições presentes no estado *DROP*), o restante das transições alcançam o estado *ACCEPT*, significando que o autômato, em algum momento, realizou tratativas necessárias para a primitiva em questão, estando pronta para ser encaminhada para fila de saída (*pos-routing*).

O estado *SCHEDULER* é responsável por aplicar as ações de encaminhamentos nas primitivas, que compreendem a priorização e controle de banda. Primitivas às quais se apliquem alguma regra de requisito são enviadas para o estado *ACCEPT*, onde serão enviadas para a respectiva fila de saída. Primitivas às quais não se apliquem requisitos (?$RegularIND$), que transitaram do estado *PARSER* para o estado *ACCEPT*, serão encaminhadas para a fila de melhor esforço (*Best Effort*).

As transições ACCEPT(Accept==False -> ACCEPT) e DROP(Drop==False -> DROP) representam eventos de controle, no caso do estado *ACCEPT* o encaminhamento da primitiva para uma determinada fila de saída ainda não foi efetivado e, no estado *DROP* o descarte da primitiva em questão ainda não ocorreu. Contudo as transições ACCEPT(Accept==True -> ACTIVE) e DROP(Drop==True -> ACTIVE) representam o sucesso dos eventos descritos.

5.2 Visão geral da Arquitetura e Organização da NEA

De acordo com Hennessy e Patterson (2011), a arquitetura de um sistema trata de parâmetros visíveis ao desenvolvedor, que deve especificar aspectos estruturais referentes a distribuição espacial dos componentes internos da arquitetura e interconexões. A especificação de uma arquitetura significa distribuir os componentes em diferentes níveis de abstração, integrando-os com os mecanismos de interconexão.

Diferentemente de *switches* tradicionais SDNs, que possuem regras e ações de encaminhamento menos flexíveis, basicamente orientadas por endereços de diferentes camadas, o *Switch* NEA propõe um novo tipo de encaminhamento ajustável a necessidades de aplicações, garantindo que diversos fluxos (comunicações) com diferentes requisitos perpassem o equipamento, inclusive compartilhando a mesma porta física. Desta forma, do ponto de vista das ações de encaminhamento com garantia de qualidade, pode-se dizer que o *Switch* NEA possui capacidade de gerenciar granularidades mais finas em comparação com os atuais *Switches* SDNs. A Figura 20 apresenta uma visão geral da arquitetura NEA e, também, seus principais componentes.

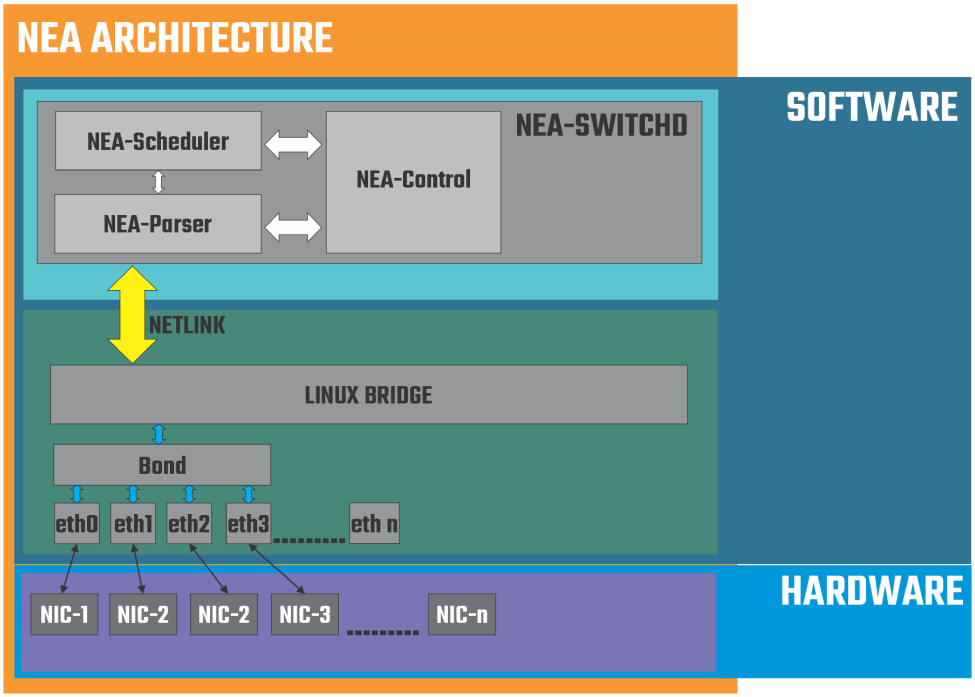


Figura 20 – Visão Geral da Arquitetura NEA

O *Switch* NEA é um protótipo desenvolvido com uma abordagem *linux-based*, projetado para provar a hipótese deste tese. Ambos, *Control Path* e *Data Path* foram desenvolvidos no nível de software.

O *Switch* NEA faz uso de *Bridges* para promover a interligação de interfaces de rede na camada 2 e explorar a flexibilidade através de um módulo de software. A utilização de *Bridges* torna a NEA transparente em relação ao *hardware* e a tecnologia.

Embora a utilização da *Bridge* seja importante para a implementação do *Switch* NEA, é importante ressaltar o suporte do sistema operacional Linux para o seu desenvolvimento.

5.2.1 *Suporte do Sistema Operacional Linux*

Para confecção do *Swtich* NEA, além do módulo *Bridge* explanado no Capítulo 2, é importante citar o *nftables*, *NetfilterQueue* e *Traffic Control (TC)*, responsáveis por:

(i) definir regra(s) de roteamento de primitivas no *kernel*; (ii) enfileirar todas primitivas abarcadas pela regra anterior; e (iii) definir ações de encaminhamento nas filas de saídas do *Switch* NEA.

De acordo com netfilter.org (2021), *nftables* é um *framework* integrado ao *netfilter* que faz reuso de subsistemas, tais como: *Network Address Translation (NAT)*, infraestrutura de *hooks* e *userspace queueing*. O *nftables* é encontrado somente nas versões recentes do *kernel* do Linux – em substituição ao *iptables*, *ebtables* e *arptables* – oferecendo uma interface de programação mais poderosa, permitindo maior profundidade na classificação de primitivas dentro do *kernel*.

NetfilterQueue é um módulo da biblioteca *netfilter*, que provê acesso a todos os *frames*, por meio de regras *nftables*, *iptables*, *ebtables* e *arptables*. O *NetfilterQueue* permite enfileirar primitivas e, através da programação de alto nível, associar ações, tais como: aceitar, descartar, alterar e etiquetar *frames*.

No *Switch* NEA, o módulo *NetfilterQueue* atua em conjunto com o *nftables* na construção da estrutura de enfileiramento de primitivas, redirecionando todo o tráfego do *switch* para a aplicação NEA-SWITCHD no *user space*.

Além da utilização do *nftables* e *NetfilterQueue*, a aplicação NEA-SWITCHD atua na orquestração dos fluxos de saída através da utilização do módulo *Traffic Control (TC)* (TRAINING, 2021; BROWN, 2006). Este módulo faz parte da biblioteca *iproute2* e foi instituído para atuar na melhoria de QoS e QoE, permitindo modificar a forma como as primitivas são recebidas e encaminhadas para a rede.

Para uma melhor compreensão, a seguir será explanado em detalhes os principais submódulos que compõem o *Switch* NEA.

5.3 Aplicação NEA-SWITCHD

NEA-SWITCHD é a principal aplicação do *Switch* NEA. De forma geral, esta aplicação é responsável por tratar todas as primitivas que trafegam pelo *switch*, sendo composta pelos submódulos *NEA-Control*, *NEA-Parser* e *NEA-Scheduler*, que possuem responsabilidades bem definidas e juntos oferecem toda programabilidade e flexibilidade proposta nesta tese.

Do ponto de vista do processamento de *frames*, o *Switch* NEA implementa um *pipeline*, ou seja, independente da origem, todos os *frames* serão enfileirados e tratados um de cada vez, avançando a estágios sucessivos a cada transição.

Em linhas gerais, a aplicação NEA-SWITCHD analisa todos os *frames* que ingressam na *Bridge* e atua sobre as políticas de encaminhamento nas filas de saída. Na Figura 21 é possível observar a aplicação NEA-SWITCHD no contexto do fluxo de primitivas desde a origem até a saída.

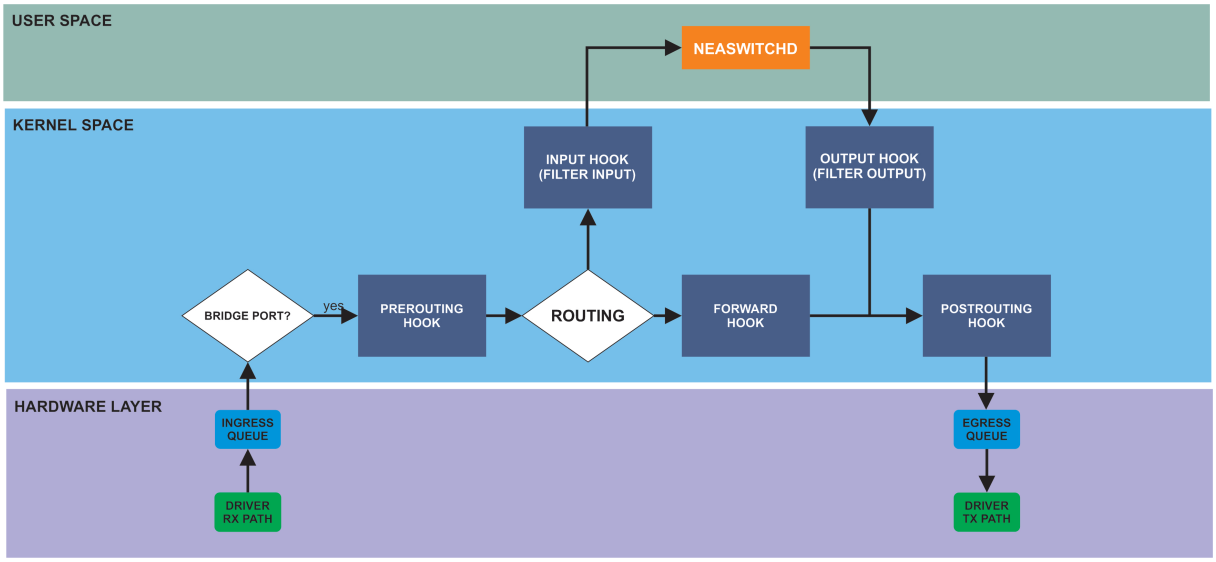


Figura 21 – NEA-SWITCHD e o fluxo de primitvas no *Switch*

5.3.1 Submódulo NEA-CONTROL

Este submódulo é responsável pelos processos de controle, tais como: (i) rotina de *bootstrapping*; (ii) tratamento de primitivas de controle oriundas do *NEA-Parser*; e (iii) construtor de ações de encaminhamento no submódulo *NEA-Scheduler*.

Com relação à rotina de *bootstrapping*, inicialmente, alguns processos externos à aplicação NEA-SWITCHD devem ser executados para o correto funcionamento do *switch*. A rotina de *bootstrapping* é definida pelos processos:

- ❏ INSTITUIÇÃO DA BRIDGE: este processo é responsável pela criação e configuração da *Bridge* no *kernel*, sendo importante selecionar quais interfaces estarão incorporadas na *Bridge* e algumas configurações, como por exemplo endereçamento e NAT;

- ❏ INSTITUIÇÃO DAS POLÍTICAS DE ROTEAMENTO DE PRIMITIVAS DO *kernel*: este processo é responsável pela confecção das regras de pré-roteamento da *Bridge*, através do *framework nftables*, e um detalhe importante é que este processo é definido com prioridade alta no escalonador de processos do sistema operacional (*nice*);

- ❏ INSTITUIÇÃO DO HOOK NETFILTERQUEUE: esta etapa do processo configura um gancho (*Hook*), para desviar o fluxo de *frames* para uma fila pré-determinada, onde posteriormente essas primitivas serão consumidas pelo submódulo *NEA-Parser*.

Na Figura 22 é possível observar um retrato *Switch* NEA após a rotina de *bootstrapping*.

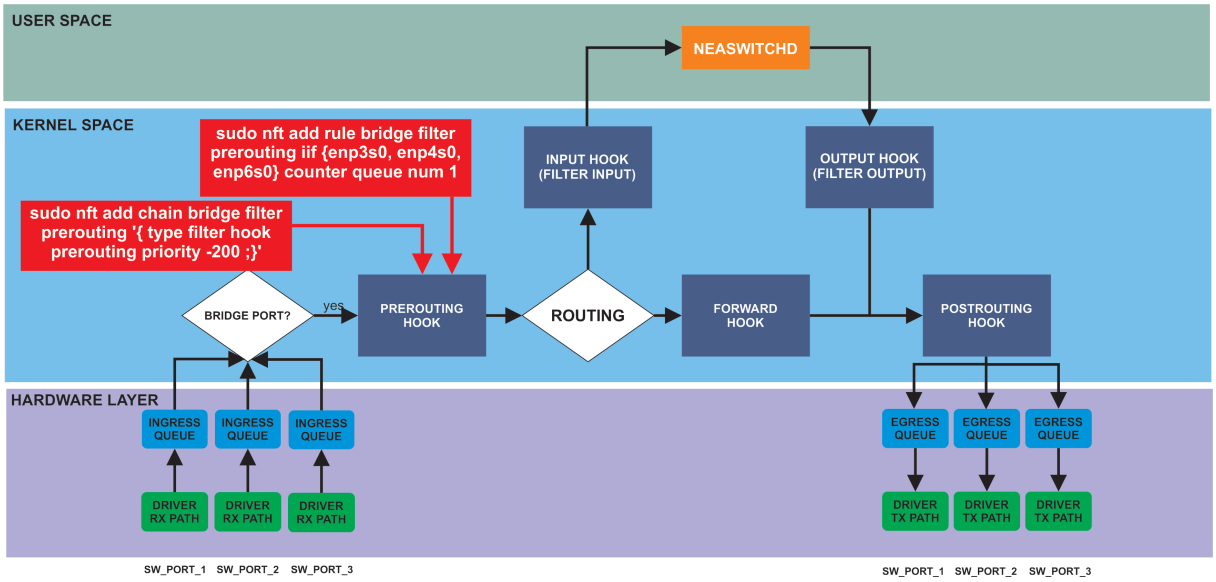


Figura 22 – Integração do módulo NEA-SWITCHD, *nftables* e *netfilterqueue*

De acordo com a Figura 22, a atuação do *nftables* e *NetfilterQueue* acontece diretamente nas políticas de pré-roteamento da *Bridge*. Todas as primitivas que ingressam por interface(s) de rede, independente da quantidade de portas físicas, são submetidas à regra definida pelo *nftables*.

O *nftables* atua diretamente na instância do *Socket Buffer (SKB)*, antes mesmo de qualquer política de roteamento da *Bridge*. Desta forma, garante-se que todas as primitivas serão consumidas pela aplicação NEA-SWITCHD.

O submódulo *NEA-Control* também é responsável por instituir as ações de encaminhamento praticadas pelo submódulo *NEA-Scheduler*. Em tese, as primitivas de controle são identificadas pelo *NEA-Parser* e interpretadas pelo *NEA-Control*, que tem a função de traduzir a mensagem de controle em ações de encaminhamento através do TC. Importante ressaltar que, o *NEA-Scheduler* faz uso das ações implementadas pelo *NEA-Control*.

Na Figura 23 é possível observar a atuação da aplicação NEA-SWITHCD, especificadamente do submódulo *NEA-Control*, em relação a orquestração do *nftables*, *NetfilterQueue*, *Bridge* e *Traffic Control (TC)*.

5.3.2 Submódulo NEA-PARSER

O submódulo *NEA-Parser* é responsável por analisar cabeçalhos de *frames*, em busca de correspondências com regras especificadas na Tabela de Encaminhamento.

A análise é realizada de forma sequencial, inicialmente, retirando *frames* da fila (*NetfilterQueue*) e registrando informações importantes de cabeçalhos, tais como *endereço de destino (MAC)*, *endereço de origem (MAC)* da camada 2, informações das camadas su-

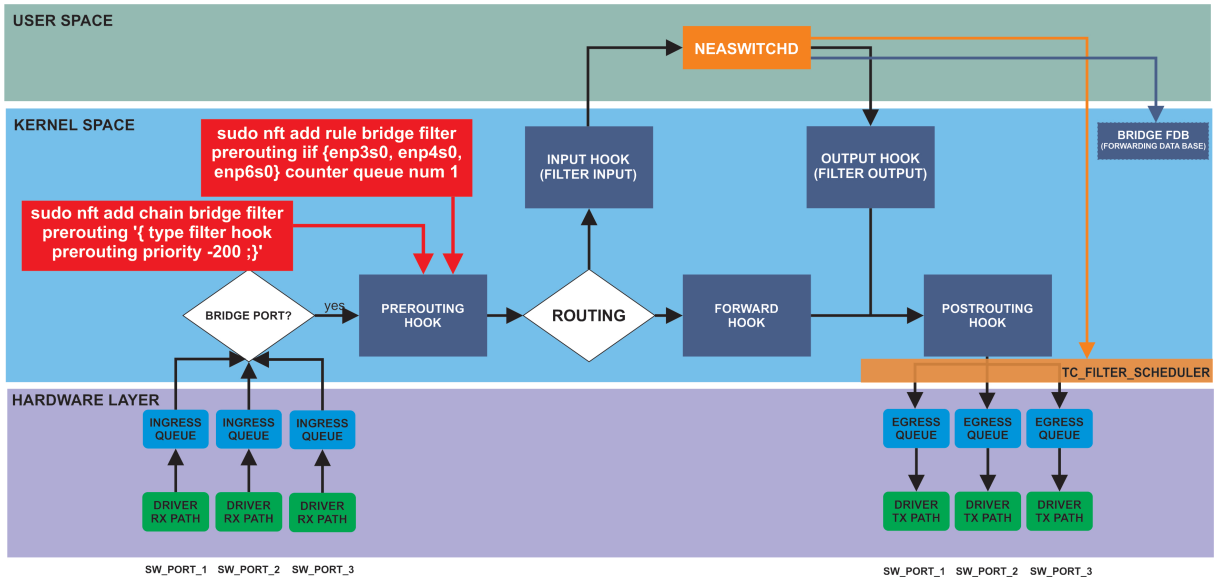


Figura 23 – Integração módulo NEA-SWITCHD, suporte do *kernel* Linux - *nftables*, *netfilterqueue* e *TC*

periores como *endereço IP de origem* e *endereço IP de destino* da camada 3 e, *porta de origem* e *porta de destino* da camada 4.

De acordo com as ações praticadas pelo *NEA-Parser*, a primitiva de dados pode ser encaminhada para o pós-roteamento da *bridge* e, caso já exista uma regra de encaminhamento[1], ela será aplicada à primitiva em tela.

Caso seja uma primitiva de dados, sem nenhuma correspondência no *NEA-Parser*[2], ela será tratada pelo *NEA-Scheduler* como tráfego regular, naturalmente, será tratada na base do melhor esforço, mas sem garantia de banda ou prioridade.

Se, por outro lado, for uma primitiva de controle, ela será encaminhada para o submódulo *NEA-Control*, para instituição ou atualização de políticas de encaminhamento no *NEA-Scheduler*.

É responsabilidade do *NEA-Parser* identificar primitivas que serão descartadas e, portanto, não serão tratadas pelo *switch*.

5.3.3 Submódulo NEA-SCHEDULER

O submódulo *NEA-Scheduler* é responsável por materializar as ações de encaminhamento nas filas de saída. Para realizar esta tarefa este submódulo atua sobre o *Traffic Control (TC)*.

[1] Note-se que, a existência de uma correspondência no *NEA-Parser* decorre de ação de encaminhamento exclusiva praticada pelo *NEA-Scheduler*

[2] Este é o caso de primitivas normais de dados, sem requisitos de qualidade de serviço

O algoritmo de escalonamento atua de forma independente em cada porta física do *switch*, lembrando que o *Switch* NEA permite a instituição de diversos fluxos por porta com tratativas isoladas para cada fluxo, garantindo os respectivos requisitos de qualidade de serviço.

Para uma melhor compreensão do funcionamento do submódulo *NEA-Scheduler* é necessário o entendimento dos elementos seguintes, conforme definido por Brown (2006):

- **Qdisc** – *Queuing Disciplines*: trata-se de um conjunto de algoritmos de escalonamento, responsáveis pela forma como as primitivas são enfileiradas nas filas de saídas;
- **Classes**: trata-se das entidades responsáveis pela classificação das primitivas;
- **Filters**: são elementos utilizados para policiar e classificar as primitivas, atribuindo-as para suas respectivas *classes*;
- **Polices**: são regras que definem os limites associados a cada *filter*.

As ações de escalonamento requerem que cada porta física do *switch* possua obrigatoriamente uma *Qdisc* principal, dado que ela define a regra geral de tráfego e, em especial, das *classes*, associadas à porta física, cuja quantidade é variável e depende da quantidade de tratamentos diferenciados necessários. Por exemplo, caso existam apenas duas categorias de tráfegos, VoIP e Dados, bastaria apenas duas categorias de *classes* de tratamento, uma para cada categoria de tráfego, sendo que, qualquer primitiva que não fosse associada a uma dessas *classes* seria inserida na regra geral da *Qdisc*.

No contexto das políticas de escalonamento, os *filters* atuam como classificadores, encaminhando os *frames* para suas respectivas *classes* e as *policies* definem as capacidades de tráfego das *classes*. Para as *classes VoIP*, por exemplo, define-se um limite de banda de 84Kbps, mas isto depende da técnica de codificação (*codec*).

Queueing Disciplines

Como observado anteriormente, a construção de uma ação para atender um parâmetro de QoS envolve a manipulação das variáveis *classes*, *filters* e *policies*. Contudo, é importante ressaltar que o efeito de uma política de encaminhamento, parte da utilização de algoritmos de escalonamento utilizados pela *Qdisc* (BROWN, 2006).

O subsistema TC permite atuar com 8 tipos diferentes de algoritmos de escalonamento, podendo ser utilizado apenas um para cada *Qdisc* criada (BROWN, 2006). Caso exista a necessidade de troca do algoritmo de escalonamento, isso resultará na (re)configuração das políticas de escalonamento e (re)inicialização da porta física.

De acordo com Brown (2006), os algoritmos de escalonamento podem ser definidos como:

- *First in, First out (FIFO)*: este algoritmo indica que os *frames* serão encaminhadas na ordem em que chegarem na *Qdisc*;
- *Stochastic Fair Queueing (SFQ)*: este algoritmo busca praticar o critério de justiça, equilibrando o encaminhamento de *frames* de maior e de menor prioridade;
- *Diff-Serv Marker (DSMARK)*: este algoritmo visa etiquetar as primitivas para implementação da política *Diffserv*;
- *Token Bucket Flow (TBF)*: trata-se de um algoritmo simples que controla o número de *frames* encaminhados, utilizando-se parâmetros administrativos pré-definidos, sendo permitido alterar esses valores;
- *Random Early Detection (RED)*: este algoritmo prevê o descarte preventivo de *frames* para evitar o aumento excessivo do tamanho da fila;
- *Priority Queue (PRIO)*: este algoritmo prevê que os *frames* serão encaminhados de acordo com o nível de prioridade atribuído para as filas, sendo que enquanto as filas de maior prioridades não forem zeradas, as filas de menor prioridade não serão visitadas;
- *Class-Based Queueing (CBQ)*: este algoritmo prevê um esquema de encaminhamento baseado em hierarquia, no qual, as filas superiores possuem maior prioridade sobre as filas inferiores;
- *Hierarquical Token Bucket (HTB)*: este algoritmo substitui o anterior CBQ, considerado ineficiente, e visa a construção de estrutura no formato de árvore onde os *nós* se comportam como *classes*, oferecendo elasticidade na composição de regras de escalonamento e permitindo estruturar diversas subdivisões de bandas pela concatenação de *classes*, além de permitir explorar diferentes parâmetros de QoS, tais como prioridade e controle de banda.

Para implementação no *Switch* NEA foi utilizado o algoritmo HTB, para todas as *Qdisc*. Todavia, existe flexibilidade para instituir diferentes algoritmos de escalonamento por porta física e, até mesmo, instituir políticas de trocas de algoritmo de escalonamento por porta física, lembrando que, a troca de algoritmo de escalonamento da *Qdisc* resulta na (re)inicialização da porta física.

Hierarquical Token Bucket (HTB)

O algoritmo HTB organiza as *classes* como "nós" de uma estrutura de dados em árvore. Além do nível de prioridade e da largura de banda, utilizados como parâmetro de QoS, no HTB, outras variáveis podem ser exploradas para alcançar mais flexibilidade nas ações de encaminhamento. Os parâmetros das *classes* são:

- *Rate*: banda média garantida para toda a *classe* e também para as *classes* filhas;
- *Ceil*: banda máxima herdada da *classe* pai;
- *Burst*: quantidade máxima, em *bytes*, que pode ser enviada na banda definida pelo parâmetro *ceil*;
- *Cburst*: quantidade máxima, em *bytes*, a ser enviada na banda definida pela *classe*, se não houver limitação por parte do (nó) pai;
- *Priority*: define a ordenação da estrutura de árvore, sendo que as *classes* de maior prioridade (prioridade 0) recebem o excesso de banda primeiro com redução de latência.

O *Switch* NEA desenvolvido neste trabalho atua na camada 2, sendo assim, o endereço MAC presente no cabeçalho *Ethernet* é utilizado como identificador único para as regras de encaminhamento. O uso do cabeçalho *Ethernet* é justificado em função da base instalada de equipamentos com placas de rede dessa tecnologia, sendo que o método de acesso ao meio (MAC) é substituído no *Switch* NEA.

Apesar do TC oferecer notável suporte para a camada 3, é possível adaptá-lo para interpretar o endereço MAC presente no cabeçalho dos *frames* como identificador para as políticas de encaminhamento.

Na figura 24 é possível observar o formato de organização hierárquico praticado pelo *Switch* NEA e também a forma de enfileiramento de *frames* do TC.

De acordo com a Figura 24, nota-se que para cada porta física é instituída uma *Qdisc* juntamente com o algoritmo HTB. Foi utilizada a identificação **1:0**, que trata apenas de uma forma de identificação das *Qdisc*, não apresentando nenhum significado adicional.

Ainda de acordo com a Figura 24, a existência da *class 1:1*, que é a *class* que define os limites de tráfego e prioridade da interface, permite que sejam instituídas diversas *classes* filhas, i.e., *class 1:11*, *class 1:21* e *class 1:31*, herdando as características da *class 1:1* pai. Este formato de organização permite um controle mais detalhado, visto que a *class 1:1* pode ter quantas *classes* filhas forem necessárias.

Uma vez que as *classes* foram criadas, os *filters* devem ser anexados às *classes* para que *frames* possam ser analisados e alocados corretamente. Os *filters* seguem a mesma identificação das *classes*, visto que cada *classe* pode possuir diversos *filters*. Caso seja necessário instituir novas políticas de escalonamento, deve ser criada uma nova *class* e um *filter*.

É importante ressaltar que, os parâmetros definidos para *classes* podem ser alterados em tempo real, sem a necessidade de interrupção do funcionamento do equipamento. A única exceção é com relação a *Qdisc* principal atrelada à porta, responsável por conduzir as políticas de escalonamento. Nota-se que, o algoritmo de escalonamento é vinculado a cada porta física do equipamento, podendo assim utilizar diferentes algoritmos em diferentes

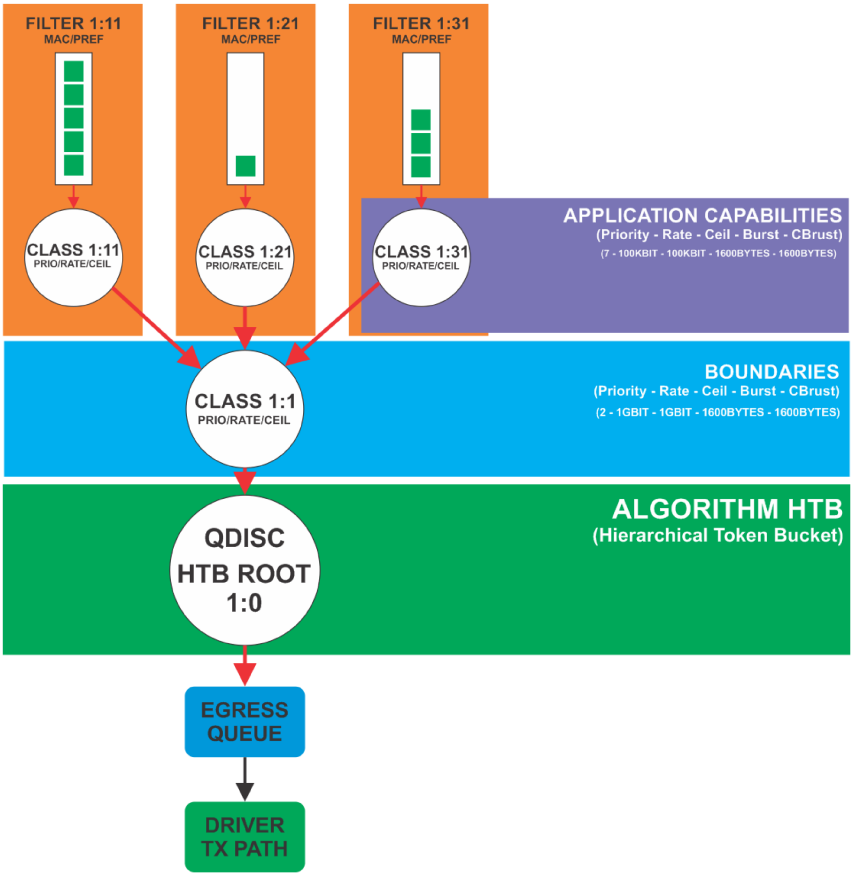


Figura 24 – Visão geral do TC praticado pelo *NEA-Scheduler*

portas. Por conveniência, nesta pesquisa padronizou-se o uso do algoritmo HTB para todas as interfaces.

Paralelamente, a arquitetura NEA permite a criação de diversos fluxos de comunicação com diferentes parâmetros de QoS customizados sob supervisão do controlador da rede.

5.3.4 *NEA DataPath*

De forma geral, o *datapath* define o caminho de cada *frame* durante o processamento, desde de seu ingresso pela porta física de entrada até alcançar a porta de saída. Na figura 25 é possível observar a organização da NEA para tratar as primitivas de dados que trafegam através do elemento de rede.

De acordo com a Figura 25, todas as primitivas oriundas das portas de entrada são enfileiradas e consumidas uma de cada vez pela aplicação.

O submódulo *NEA-Parser* consome a primitiva diretamente do *SKB*, antes mesmo de qualquer política de pré-roteamento da *Bridge*. Este submódulo é responsável por analisar todas as informações presentes no cabeçalho da primitiva, incluindo o suporte para *Ethernet*, IP e TCP/UDP, além da capacidade de suportar novos protocolos. Após a análise pelo *NEA-Parser* a primitiva é encaminhada para o *NEA-Scheduler*.

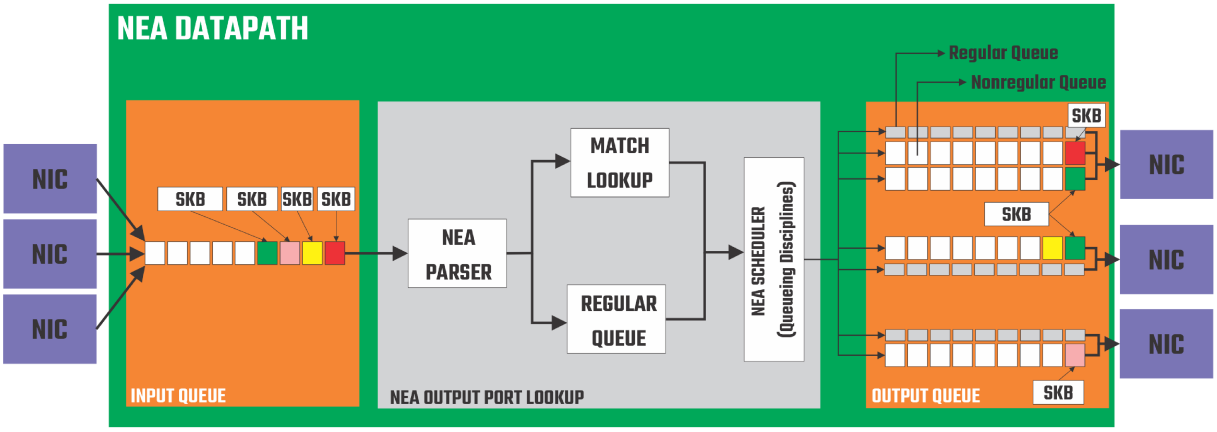


Figura 25 – NEA DataPath

Os processos *Match_LookUp* e *Regular_Queue* fazem parte do submódulo *NEA-Control* e servem para diferenciar primitivas que possuam regras associadas. Do ponto de vista do *datapath*, ambos os processos terminam no escalonador, contudo, o processo *Match_LookUp* é responsável por interceptar as primitivas de controle, aplicar regras de descarte quando necessário, ou simplesmente encaminhar a primitiva para o escalonador.

O submódulo *NEA-Scheduler* é programado através da aplicação NEA SwitchD, especificamente, pelo submódulo *NEA-Control*. A criação de filas de saída juntamente com as regras de escalonamento (baseadas em prioridade e largura de banda) são configuradas através de primitivas de controle, interceptadas pelo processo *Match_LookUp*. As funcionalidades do submódulo *NEA-Control* e *NEA-Scheduler* serão abordadas em detalhes nas seções subsequentes.

Note-se que, de acordo com a Figura 25, a capacidade do *Switch* NEA em promover a segregação de tráfego nas portas físicas de saída. Além da capacidade de instituir diversos fluxos com capacidades distintas pela mesma porta física, o *Switch* NEA permite uma granularidade fina com relação a manutenção das regras de encaminhamento.

Capítulo 6

Estudo de Caso: Arquitetura ETArch

6.1 Arquitetura ETArch

Todo o aparato teórico sobre o Modelo de Títulos e a ETArch necessário para compreensão deste trabalho está disposto no capítulo 2 desta tese. Agora, faz-se necessário revisitar alguns conceitos e explorar a ETArch sob o prisma do plano de dados.

Em um ambiente ETArch, o *Workspace* é um barramento lógico responsável pela interligação das *Entidades*, neste contexto, o *Switch* oferece suporte para o *Workspace* e não diretamente para os (*endpoints*). A criação e manutenção de um *Workspace* é responsabilidade do *DTSA*, que é o controlador SDN no contexto da ETArch.

Com relação ao endereçamento na ETArch, o *Título* é a designação única para garantir a identificação de uma *Entidade* da rede, podendo ser definido por um *Hash*, um identificador numérico, uma sequencia ASCII e até mesmo uma combinação de blocos de dados (PEREIRA, 2012a).

6.1.1 Endereçamento na ETArch

Por definição, neste estudo de caso, o *Título* foi definido com a mesma estrutura de um endereço MAC (Padrão *Ethernet*) para evitar incompatibilidade com a *NICs*. Na Figura 26 pode ser observado o protocolo *Ethernet* com destaque nos campos relativos ao endereço MAC.

De acordo com a Figura 26, o padrão 802.3 (*Ethernet*) possui dois campos relativos ao endereço de origem e destino, *Destination Address* e *Source Address*, com tamanho de 6 *bytes* cada. Os 3 primeiros *bytes* são definidos pelo órgão IEEE, os demais 3 *bytes* pelo fabricante da NIC. Pautado pela norma ISO/IEC 10039 (*LAN MAC Service Definition*), por conveniência esses endereços são representados em codificação hexadecimal.

Endereços MAC podem ser manipulados Universal e Localmente, bem como individualmente ou em grupo (IEEE/SA, 2018). Ao analisar um endereço MAC tem-se a

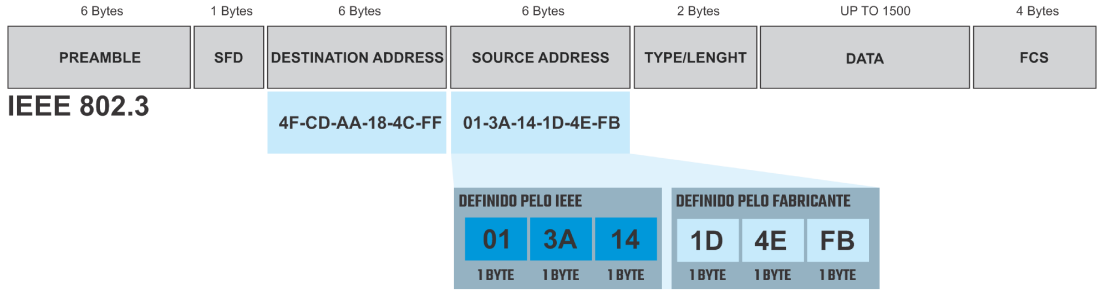


Figura 26 – Padrão IEEE 802.3 - Figura adaptada de Kurose (2013)

sequência, "**xx:xx:xx:xx:xx:xx**", onde cada "**x**" pode ser substituído por um caractere hexadecimal.

Existem 4 possíveis conjuntos de endereços que podem ser utilizados para administração local de endereços, são eles: "**x2:xx:xx:xx:xx:xx**" , "**x6:xx:xx:xx:xx:xx**", "**xA:xx:xx:xx:xx:xx**" e "**xE:xx:xx:xx:xx:xx**" (HONEYWELL, 2017).

A utilização de um padrão reservado de endereço MAC foi necessário para evitar o conflito de endereços no *Switch*. Neste contexto, um *Título* é formado pela combinação da identificação do *Workspace* + identificação da *Entidade*, formando um endereço único sob a estrutura de endereço MAC para a ETArch. Manter o padrão de endereço MAC foi uma decisão de projeto para evitar problemas de compatibilidade entre transmissor, receptor e *Switch*.

Foi fixado neste estudo de caso, para facilitar a compreensão, o seguinte padrão de endereço **FE:xx:xx:Ex:xx:xx**, onde, os 3 *bytes* iniciais são utilizados para compor a identificação do *Workspace*, sendo o primeiro *byte* fixado como (FE) para servir de identificador de range local e os outros 2 *bytes* utilizados para a identificação do *Workspace*. Para os 3 *bytes* finais, segue-se a mesma fundamentação, porém referente a *Entidade*.

6.2 Protocolo de configuração do elemento de rede

O *Etarch Switch Control Protocol (ETSCP)* (OLIVEIRA et al., 2021) é um protocolo de controle para elementos de rede em ambiente ETArch, não tendo utilização na Internet ou qualquer outra arquitetura. Trata-se de um protocolo desenvolvido para operar entre o *Switch* e o controlador da rede (DTSA), orientando a criação, atualização ou exclusão de *Workspaces*.

6.2.1 Formato de mensagem do ETSCP

Para o DTSA configurar um elemento de rede, primitivas ETSCPs são enviadas diretamente para elemento de rede (OLIVEIRA et al., 2021). Na Figura 27 é possível observar o formato de mensagem do ETSCP em comparação com o padrão *Ethernet*.

As alterações mais impactantes foram no *Payload* para comportar informações de controle (*msg_type* e *dst_entity_title*) .

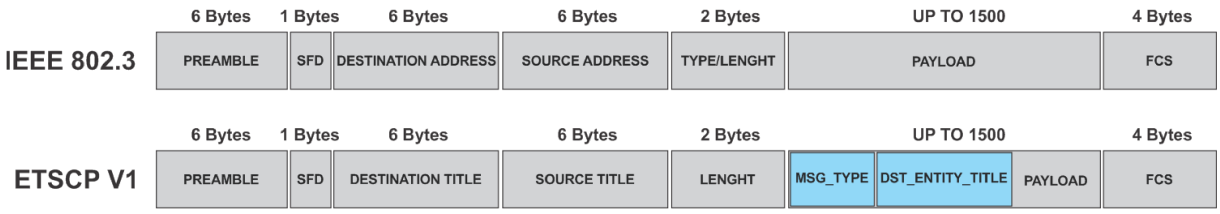


Figura 27 – Formato da mensagem ETSCP. Figura adaptada de Oliveira et al. (2021)

A seguir se apresenta uma breve explanação dos campos do protocolo. De acordo com Oliveira et al. (2021):

1. **DESTINATION_TITLE**: *Título* do *Workspace* de destino;
2. **SOURCE_TITLE**: *Título* da *Entidade* de origem;
3. **LENGTH**: comprimento, em Bytes, do campo de dados;
4. **MSG_TYPE**: indica qual o tipo de mensagem reconhecida pelo vocabulário que aquela primitiva está carregando;
5. **DEST_ENTITY_TITLE**: *Título* da *Entidade* de destino em um Workspace;
6. **PAYLOAD**: Campo de dados para parâmetros das mensagens;
7. **CRC**: soma de verificação para detecção de erro.

A instituição do protocolo ETSCP foi dado em um contexto diferente deste trabalho. O protocolo foi concebido para atuar exclusivamente sobre um ambiente ETArch. Diante das necessidades da NEA, principalmente em relação ao suporte às diversas arquiteturas SDNs foi necessário alterações na estrutura do protocolo ETSCP.

Primeiramente, foi modificado o formato da mensagem do protocolo, expandindo o campo de endereçamento, pois não é coerente manter a divisão de endereço MAC de origem e destino na ETArch. Devido ao endereçamento ser baseado em *Títulos*, entende-se que a unificação dos campos *DESTINATION_TITLE* e *SOURCE_TITLE* do atual ETSCP para um único endereço com duas propriedades semânticas (*Workspace + Entidade*) aumenta a capacidade de endereçamento na arquitetura, permitindo 12 *bytes* de endereçamento total e não somente 6 *bytes* para cada.

Na Figura 28 é possível visualizar a alteração em comparação com a primeira versão do protocolo ETSCP.

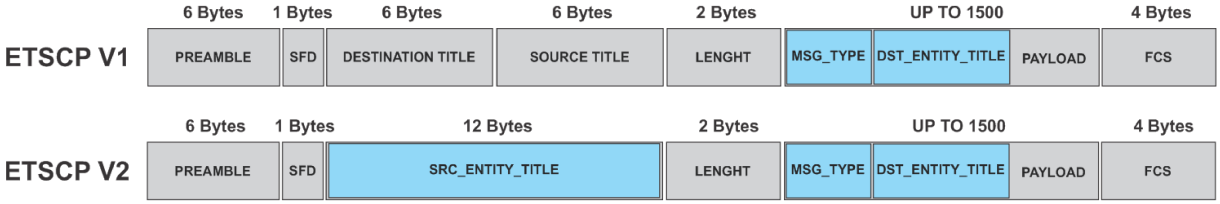


Figura 28 – Formato da mensagem ETSCP atualizada. Figura adaptada de Oliveira (2019)

Em decorrência da alteração descrita, foi importante redefinir os campos do ETSCP. A seguir uma breve explanação dos campos da versão atual do protocolo, utilizada neste estudo de caso.

1. **SRC-ENTITY-TITLE**: identificação não ambígua da *Entidade* de origem, neste campo é utilizado o Título do *Workspace* + Identificação da Entidade;
2. **LENGTH**: comprimento, em Bytes, do campo de dados;
3. **MSG-TYPE**: indica qual a categoria de mensagem reconhecida pelo vocabulário que aquela primitiva está carregando;
4. **DEST-ENTITY-TITLE**: identificação não ambígua da *Entidade* de destino;
5. **PAYLOAD**: campo de dados para parâmetros das mensagens;
6. **CRC**: soma de verificação para detecção de erro.

Além de alterações realizadas no formato da mensagem do ETSCP, foi necessário a inclusão de um novo serviço para atender de forma geral, ações não exclusivas da ETArch. Isto resultou em alterações no vocabulário e nas regras de procedimentos do ETSCP.

6.2.2 Regras de Procedimento do Protocolo ETSCP

É papel das regras de procedimento explicar a dinâmica de troca de mensagens. O protocolo ETSCP é simples em sua concepção e basicamente descreve o comportamento do elemento de rede perante as requisições enviadas pelo controlador da rede, no caso da ETArch, o DTSA (OLIVEIRA, 2019).

Além dos serviços previstos pelo protocolo ETSCP, a saber *Create Workspace*, *Edit Workspace*, *Remove Workspace* e *Change Workspace*, foi necessário instituir um novo serviço para atuar sobre as regras de escalonamento para as comunicações não ETArch.

Além dos serviços já citados, o serviço *Renew General Connection* foi incluído no rol de serviços do protocolo ETSCP e seu comportamento é similar às regras de procedimentos estabelecidas para o serviço *Change Workspace*, porém, sua atuação é exclusiva para

tráfego não ETArch. Este novo serviço atua sobre ações de encaminhamento instituídas para tráfegos não *Workspaces*, normalmente utilizando uma política *Best Effort*.

A nomenclatura do serviço *Change Workspace* foi alterada para *Update Workspace* pois o serviço em questão atua sob múltiplos parâmetros com impacto na QoS.

6.3 *Switch* NEA Aplicado à ETArch

Para o diagrama de estados do *Switch* NEA com suporte à ETArch foi utilizado a máquina de estados finita conhecida como máquina de Mealy (SHAHBAZ; GROZ, 2009). O diagrama de estados apresentado na Seção 5.1 ilustra o comportamento do *Switch* NEA em relação às primitivas de dados que ingressam no elemento de rede. Este comportamento permanece como apresentado, porém, neste estudo de caso fez-se necessário adicionar os serviços de controle inerentes ao ETSCP.

Na Figura 29 é apresentado o diagrama de estados do *Switch* NEA aplicado à ETArch. A grande parte das transições são em relação às primitivas de controle definidas pelo ETSCP e em alguns casos eventos de controle do próprio *Switch*.

Comparado ao diagrama de estados apresentado na Seção 5.1, este diagrama ilustra os serviços inerentes ao protocolo ETSCP, que são: *Create Workspace*, *Edit Workspace*, *Remove Workspace*, *Change Workspace* e *Renew General Connection*. Em suma, estes serviços impactam diretamente na criação ou exclusão de fluxos nas portas físicas do *Switch*, atualização de requisitos de QoS e na instituição de ações de escalonamento.

O diagrama de estados apresentado na Figura 29, representando o *Switch* NEA aplicado à ETArch, pode ser descrito da seguinte forma:

- ❏ Σ: AddWkpIND, EditWkpIND, UpdateWkpIND, RemoveWkpIND, RenewRegConIND, RegisterHit, NoRegister, FlagFull==1, FlagFull==0, FlagBlackList==1, LogAdd, Drop==True, Drop==False, Accept==True, Accept==False, GeneralIND, RegularIND, PrivateIND, AddWkpRESP(+), AddWkpRESP(-), EditWkpRESP(+), EditWkpRESP(-), UpdateWkpRESP(+), UpdateWkpRESP(-), RemoveWkpRESP(+), RemoveWkpRESP(-), UpdateRegConRESP(+), Match==True FlagAdd==True, FlagEdit==True, FlagRemove==True, FlagUpdate==True, FlagRegular==True e CabConnected&HdwChecked==True;
- ❏ *S*: INACTIVE, ACTIVE, PARSER, DROP, ACCEPT, SCHEDULER, LOOKUP-WKP-DATABASE, CHECK-AVAILABILITY, EXEC-ADD-WKP, EXEC-EDIT-WKP, EXEC-UPDATE-WKP, EXEC-REMOVE-WKP, UPDATE-REGULAR-CONNECTION;
- ❏ *S*○: INACTIVE.
- ❏ δ: $S\ X\ \Sigma \rightarrow S$

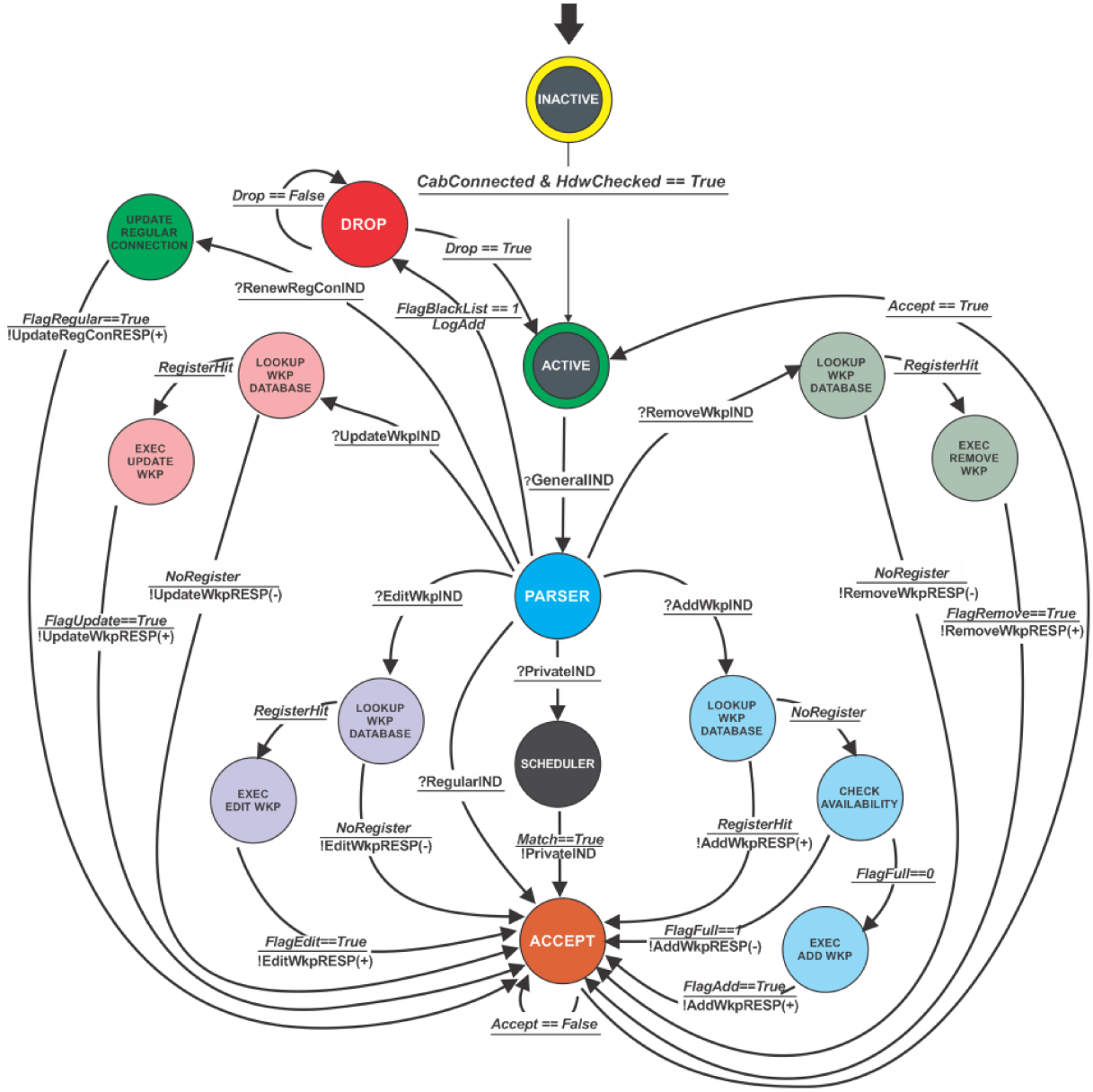


Figura 29 – Diagrama de Estados do *Switch* NEA aplicado à ETArch

❑ F: ACTIVE.

Apesar da presença de novos estados para representar os serviços inerentes ao protocolo $ETSCP$ e, estes serviços tendo maior impacto nas ações de escalonamento, outros ajustes foram necessários no estado $PARSER$ para compreensão das primitivas de controle.

6.3.1 *Parser*

A partir do estado $ACTIVE$, ao receber uma primitiva, indicado pelo símbolo $?GeneralIND$, o autômato transita para o estado $PARSER$, no qual a primitiva será analisada, de acordo com regras estabelecidas na Tabela de Chaveamento, podendo ocorrer o seguinte:

- ❏ Caso haja uma regra de descarte, ela seguirá para o estado *DROP*;
- ❏ Caso não haja requisitos especificados para a corrente primitiva, ela seguirá para o estado *ACCEPT*, numa situação ?$RegularIND$ em que nenhum requisito de prioridade ou largura de banda é requerido;
- ❏ Caso haja uma regra de requisitos (QoS), ela seguirá para o estado *SCHEDULER*, com uma indicação de ?$PrivateIND$;
- ❏ Caso seja uma primitiva, indicado pelo símbolo ?$AddWkpIND$, ela seguirá para o estado *LOOKUP WKP DATABASE* com o objetivo de atingir o estado *EXEC ADD WKP* para executar a inclusão de um novo *Workspace* de dados;
- ❏ Caso seja uma primitiva, indicado pelo símbolo ?$EditWkpIND$, ela seguirá para o estado *LOOKUP WKP DATABASE* com o objetivo de atingir o estado *EXEC EDIT WKP* para executar a edição de um *Workspace* de dados;
- ❏ Caso seja uma primitiva, indicado pelo símbolo ?$RemoveWkpIND$, ela seguirá para o estado *LOOKUP WKP DATABASE* com o objetivo de atingir o estado *EXEC REMOVE WKP* para executar a remoção de um *Workspace* de dados;
- ❏ Caso seja uma primitiva, indicado pelo símbolo ?$UpdateWkpIND$, ela seguirá para o estado *LOOKUP WKP DATABASE* com o objetivo de atingir o estado *EXEC UPDATE WKP* para executar a atualização de parâmetros de QoS de um *Workspace* de dados já existente, ou
- ❏ Caso seja uma primitiva, indicado pelo símbolo ?$RenewRegConIND$, ela seguirá para o estado *UPDATE REGULAR CONNECTION* para atualização de parâmetros de QoS inerentes ao trafego geral do *Switch*.

Pode ser observado que toda transição de saída do estado *PARSER*, inerente às primitivas do ETSCP, seguirá para o estado *LOOKUP WKP DATABSE*, com exceção a primitiva indicada por ?$RenewRegConIND$. A presença de vários estados *LOOKUP WKP DATABSE*, de mesmo valor semântico, se deu para facilitar a visualização e compreensão do fluxo de ações em relação aos serviços de controle.

6.3.2 *Lookup Wkp DataBase*

Este estado representa as consultas realizadas na base de dados de *Workspace* para verificação da existência do fluxo em questão.

O *Switch* NEA faz uso de *DataFrames* para praticar persistência de dados. Os *DataFrames* são estruturas de dados bidimensionais, similar a matrizes, instanciadas em tempo

de execução como um objeto na aplicação, em outras palavras, trata-se de um armazenamento de dados de alta capacidade e desempenho, podendo ser manipulado diretamente pela aplicação, sem intervensões ou aplicações externas.

A estrutura de base de dados utilizada para armazenamento das informações dos *Workspaces* pode ser visualizada conforme Figura 30.

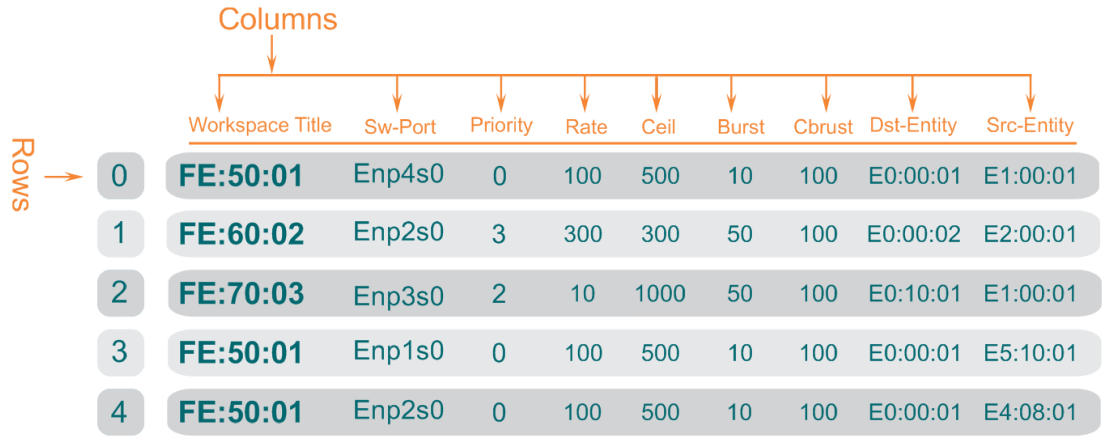


Figura 30 – Estrutura do *DataFrame* para armazenamento das informações dos *Workspaces*.

O *DataFrame* foi composto pela identificação do *Workspace* e pelos parâmetros referente às capacidades do *Workspace*, que são: *Rate*, *Ceil*, *Burst*, *CBurst* e *Priority*. A identificação unívoca das entidades de origem e destino, *Dst-Entity* e *Src-Entity* também fazem parte das informações armazenadas no *DataFrame*. Desta forma, sempre que houver a necessidade de inserção, atualização ou remoção de qualquer *Workspace*, o *DataFrame* será consultado.

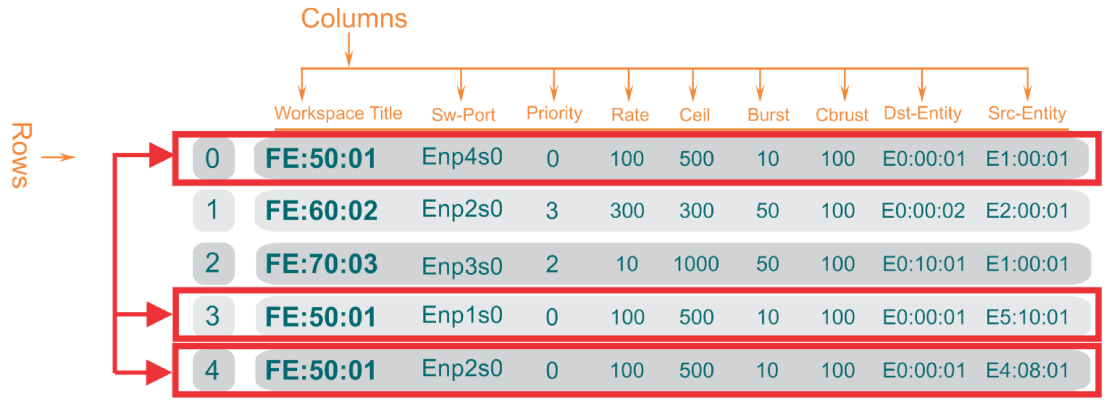


Figura 31 – Estrutura do *DataFrame* com ocorrência de um mesmo *Workspace* para diferentes portas físicas

Na Figura 31 é possível observar a composição do *DataFrame* sob a perspectiva de várias *Entidades* participando de um mesmo domínio de comunicação (*Workspace*), onde

o *Workspace* está operando em diferentes portas físicas.

6.3.3 *Add Workspace*

A solicitação da inserção de um *Workspace* no *Switch* NEA sempre é oriunda do Controlador da rede, no contexto deste estudo de caso, o *Domain Title Service Agent (DTSA)*. O processo de configuração/(re)configuração possui os mesmos passos, iniciando com a interpretação de uma primitiva de controle, indicado pelo símbolo $?AddWkpIND$, emitida pelo controlador e, na sequência, verificando a existência do *Workspace* em questão junto ao *DataFrame*.

Caso o *Workspace* em questão já exista no *DataFrame*, o DTSA será notificado com uma mensagem de confirmação. A inexistência do *Workspace* no *DataFrame* resultará no processo de configuração, que se inicia com a verificação da capacidade do *Switch* em suportar o novo fluxo.

Todo o processo de instituição de um novo fluxo é guiado por mensagens de controle entre o *Switch* NEA e o DTSA. A instituição de um novo *Workspace* no *Switch* NEA inclui a execução dos seguintes passos:

1. *DataFrame Insertion*: inserir as informações do *Workspace* no *DataFrame*;
2. *Scheduler Update*: Atualizar o submódulo *NEA-Scheduler* para operar com o novo *Workspace*. Detalhadamente, isso significa:
   a) instituir um *class* correspondente ao *Workspace* em questão;
   b) definir os parâmetros de QoS conforme estipulado pelo DTSA e,
   c) instituir um *filter* correspondente ao *Workspace*.

6.3.4 *Edit Workspace*

A ação de edição de um *Workspace*, neste contexto, é sempre oriunda do DTSA, requisitada quando houver a necessidade de inserção de *Entidades* do domínio de comunicação. O processo de edição possui os mesmos passos, iniciando com a interpretação de uma primitiva de controle, indicado pelo símbolo $?EditWkpIND$, emitida pelo controlador e, na sequência, verificando a existência do *Workspace* em questão junto ao *DataFrame*.

Torna-se importante reforçar que, um domínio de comunicação (*Workspace*) pode possuir várias entidades participantes e, ainda, cada entidade pode participar de vários domínios de comunicação simultaneamente.

Todo o processo de edição de um fluxo existente é guiado por mensagens de controle entre o *Switch* NEA e o DTSA. A edição de um *Workspace* no *Switch* NEA inclui a execução dos seguintes passos:

1. *DataFrame Lookup*: Verificar a existência do *Workspace* no *DataFrame*;

2. *Scheduler Update*: Atualizar o submódulo *NEA-Scheduler*. Detalhadamente, isso significa:

    a) instituir um *class* correspondente ao *Workspace* em questão na porta onde a *Entidade* está fisicamente conectada;

    b) instituir um *filter* correspondente ao *Workspace*;

    c) definir os parâmetros de QoS conforme estipulado pelo DTSA;

    d) atualizar a tabela de encaminhamento (*Forwarding Data Base (FDB)*) com a inclusão da nova *Entidade* em todas as portas físicas que possuem correspondência com o *Workspace* em questão e,

    e) inserir em todas as *classes* um *filter* relativo a nova *Entidade*.

6.3.5 *Update Workspace*

A ação de *Update* está relacionado com a capacidade do *Switch* NEA alterar as capacidades pré-definidas de um *Workspace* em questão. Para este estudo de caso, conforme já mencionado, são utilizados 5 parâmetros com impacto na QoS, são eles: *Rate*, *Ceil*, *Burst*, *CBurst* e *Priority*. O processo de edição possui os mesmos passos, iniciando com a interpretação de uma primitiva de controle, indicado pelo símbolo $?UpdateWkpIND$, emitida pelo controlador e, na sequência, verificando a existência do *Workspace* em questão junto ao *DataFrame*.

Todo o processo de atualização de um *Workspace* existente é guiado por mensagens de controle entre o *Switch* NEA e o DTSA. A atualização de um *Workspace* no *Switch* NEA inclui a execução dos seguintes passos:

1. *DataFrame Lookup*: Verificar a existência do *Workspace* no *DataFrame*;

2. *Scheduler Update*: Atualizar o submódulo *NEA-Scheduler*. Detalhadamente, isso significa:

    a) mapear a existência do *Workspace* em questão nas portas físicas do elemento de rede e,

    b) alterar os parâmetros de QoS em todos os *classes* em todas as interfaces físicas que possuem correspondência com o *Workpspace* em questão;

Na Figura 32 pode ser observado que as capacidades relacionadas a um determinado *Workspace* são acopladas na *class*, sendo assim, o processo de atualização não envolve nenhuma atualização nos *filters*, apenas nas *classes* relativas ao *Workspace*.

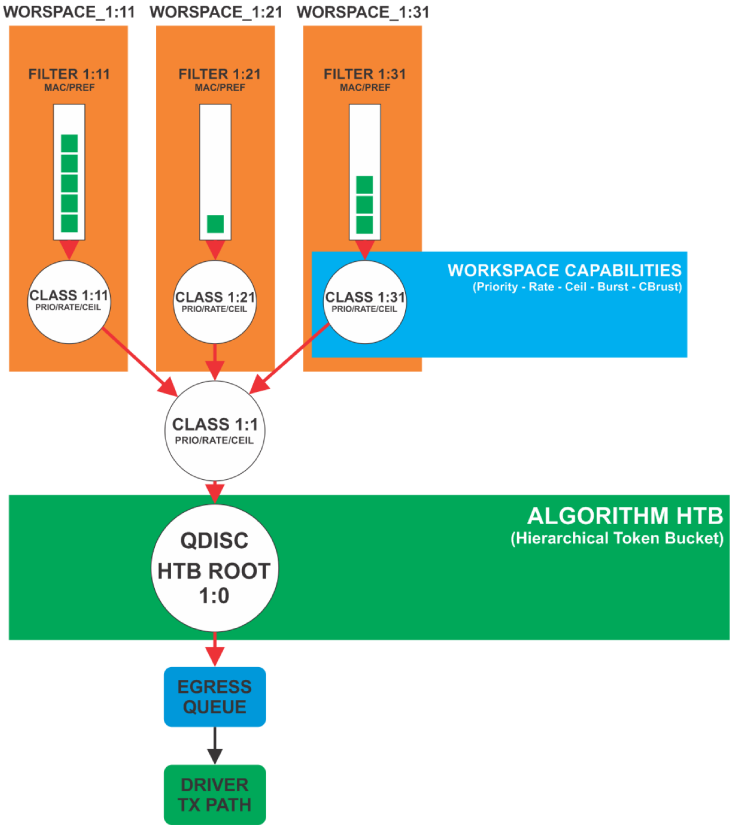


Figura 32 – Ilustração de uma porta física do *Switch* NEA sob a óptica dos filtros, das classes e do algoritmo de escalonamento

6.3.6 *Remove Workspace*

A remoção de um *Workspace* significa excluir todo e qualquer registro do domínio de comunicação no *DataFrame* e em seguida propagar essa atualização para as todas as portas físicas do *Switch* NEA. O processo de edição possui os mesmos passos, iniciando com a interpretação de uma primitiva de controle, indicado pelo símbolo $?RemoveWkpIND$, emitida pelo controlador e, na sequência, verificando a existência do *Workspace* em questão junto ao *DataFrame*.

Todo o processo de remoção de um *Workspace* existente é guiado por mensagens de controle entre o *Switch* NEA e o DTSA. A remoção de um *Workspace* no *Switch* NEA inclui a execução dos seguintes passos:

1. *DataFrame Lookup*: Verificar a existência do *Workspace* no *DataFrame*;

2. *Scheduler Update*: Atualizar o submódulo *NEA-Scheduler*. Detalhadamente, isso significa:

    a) mapear a existência do *Workspace* em questão nas portas físicas do elemento de rede e,

b) remover todos os *filters* correspondentes ao *Workspace* em questão em tordas as portas físicas do *Switch* NEA. Não havendo o *filter* para um *workspace* o escalonador aplica as ações de encaminhamento definidas pela *class*.

6.3.7 *Renew Regular Connection*

No momento da instituição de uma *Queueing Disciplines* para uma determinada porta física, o *Switch* NEA institui uma regra geral para abarcar todo o tráfego que não possui nenhuma ação de encaminhamento associada.

As regras são instituídas no momento da inicialização da aplicação NEA-SWITCHD através do submódulo *NEA-Control*. A ação de atualização das capacidades desta regra geral é responsabilidade do serviço de controle *Renew Regular Connection*. O processo de atualização inicia-se com a interpretação de uma primitiva de controle, indicado pelo símbolo $?RenewRegConIND$, emitida pelo controlador e, na sequência, altera os parametros de QoS do tráfego geral em todas as portas físicas do *Switch* NEA.

Torna-se importante ressaltar que, naturalmente os tráfegos gerais no *Switch* NEA são tratados como *Best Effort*.

6.4 Validação da Arquitetura NEA aplicada à ETArch

A etapa de validação deste estudo de caso já foi apresentada em detalhes no Capítulo 3 deste trabalho.

O objetivo da validação foi demonstrar a capacidade do *Switch* NEA em criar, atualizar e excluir diversos domínios de comunicação baseados em *Workspace* e também de atuar sobre a prioridade e a largura de banda dos tráfegos, ocasionando melhoria da QoS.

Para esta demonstração foi instituído um ambiente de rede local com *Entidades*, *Switch* NEA e uma versão *lightweight* do controlador da rede. Esta versão de DTSA compreende apenas a capacidade enviar primitivas de controle *ETSCP* para o *Switch* NEA, não faz parte do escopo deste trabalhar detalhar as funcionalidades do DTSA.

A Tabela 5 apresenta as metas que foram avaliadas nos testes do *Switch* NEA. Em suma é apresentado o processo (P) que é inerente a tarefa executada, a meta (M), que trata do resultado esperado e a descrição da meta.

De forma geral, os testes realizados demostraram a capacidade do *Switch* NEA em receber e compreender primitivas de controle, no contexto do estudo de caso, primitivas *Etarch Switch Control Protocol (ETSCP)*, conduzir a troca de mensagens entre aplicações (entidades) ETArch e, em outro contexto, ETArch e Internet. O protótipo atuou na criação e atualização dos *Workspaces*, modificando os parâmetros estabelecidos de QoS em tempo real. No Capítulo 7 será apresentado em detalhes os experimentos e resultados obtidos no estudo de caso.

Tabela 5 – Processos a serem avaliados do elemento de rede

Ident.	**Meta**	**Descrição**
MET1	*P1: Conexão entre o Elemento de Rede e Entidades.* *M1: Tabela de encaminhamento construída e escalonamento instituído.*	Avaliar a capacidade de construção das tabelas de encaminhamento e instituição das políticas de escalonamento baseadas nos requisitos das aplicações.
MET2	*P2: Mensagens ETSCP.* *M2: Mensagens ETSCP do Controlador interpretadas e tratadas.*	Testar recebimento e reconhecimento de mensagens de configuração recebidas do Controlador (DTSA.)
MET3	*P3: Criação e manutenção de rotas baseadas em Workspaces.* *M3: Tabela de Workspace construída/atualizada.*	Avaliar a capacidade do elemento de rede com relação à criação, atualização e exclusão de *Workspaces*.
MET4	*P4: Envio simultâneo de mensagens por diferentes entidades.* *M4: Comunicação entre as entidades instituída.*	Testar transmissão e entrega simultânea de dados entre as entidades conectadas no elemento de rede.
MET5	*P6: Atuar sobre parâmetros com impacto na QoS.* *M5: Analisar o impacto da alteração dos parâmetros de QoS nas comunicações.*	Avaliar a capacidade de atuação do elemento de rede em relação à gestão dos parâmetros de QoS.
MET6	*P6: Status do elemento de rede.* *M6: Avaliar o consumo de memória e processador do elemento de rede.*	Avaliar a utilização dos recursos do elemento de rede.

Capítulo 7

Experimentação e Análise de Resultados

Em se tratando de resultados, consideramos que a prototipação *Switch* NEA e, paralelamente, a especificação da arquitetura NEA já são resultados expressivos desta tese. O Estudo de Caso se posicionou como a validação do protótipo desenvolvido.

7.1 Análise das Trocas de Mensagens Referentes aos Serviços de Controle

Para esta seção, ressalta-se os serviços de controle previstos pelo protocolo ETSCP, sendo: *Create Workspace*, *Edit Workspace*, *update Workspace*, *Remove Workspace* e o recém-adicionado, *Renew General Connection*. A validação funcional destes serviços significou conduzir as trocas de mensagens de controle, entre o DTSA e o elemento de rede, observando o comportamento *Switch* NEA.

A seguir serão detalhadas as trocas de mensagens ETSCP entre DTSA e elemento de rede.

7.1.1 Mensagem de Controle *Create Workspace*

A motivação para criação de um *Workspace* nasce da necessidade de uma *Entidade* trocar informações com outra(s) *Entidade(s)*. No momento em que uma *Entidade*, que esteja devidamente conectada ao *Swtich* NEA, envia uma mensagem para um *Workspace* inexistente, o DTSA deve ser notificado para realizar a criação do respectivo *Workspace*. Na Figura 33 é possível observar em detalhes a troca de mensagens entre DTSA e o *Switch* NEA (NE) para a criação de um *Workspace*. Ainda na Figura 33, as setas ilustram a direção e os tipos de mensagens.

De acordo com a Figura 33, ao enviar uma primitiva, para a qual, não existe uma ação de encaminhamento associada no *Switch* NEA, neste contexto representado por

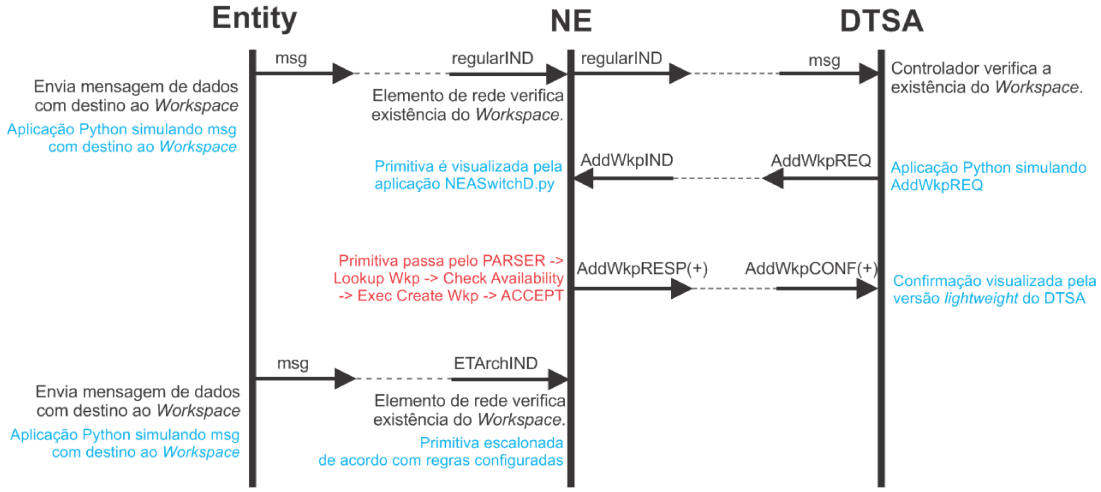


Figura 33 – Troca de mensagens para criação de *Workspace*

Workspaces, a primitiva foi condicionada a uma política *Best Effort*. Paralelamente, este comportamento objetiva encaminhar a primitiva até o alcance do DTSA que inicia o processo de criação do *Workspace*.

Para conduzir o envio de mensagens das *Entidades* e DTSA, foram utilizadas aplicações em linguagem Python escrita pelo autor deste trabalho. Desta forma, foi possível enviar primitivas de controle ETSCP de criação de *Workspace* com destino ao *Switch* NEA.

O *Switch* NEA ao receber e identificar uma mensagem de controle (ETSCP - *Create Workspace*) do *DTSA*, inicia o processo de configuração do *Workspace* conforme descrito no Capítulo 6. Após a criação do *Workspace*, as primitivas enviadas pela *Entidade*, que passam pelo *Switch* NEA, não são mais encaminhadas sob uma política *Best Effort*, mas sim, conforme definidas pelo *Workspace* em questão.

Para facilitar a visualização dos processos de controle no *Swtich* NEA, foram registrados "*Prints*" da interface visual do *Switch* NEA para comprovação do processo avaliado.

Cabe reforçar que a aplicação principal, nomeada *NEASwitchD.py*, foi desenvolvida na linguagem de programação Python, executada e testada em um ambiente totalmente baseado no Linux Ubuntu v.20.04.

O comportamento do *Switch* NEA foi observado através de diferentes elementos, a saber: (i) tabela de encaminhamento (FDB), (ii) *Classes*, *filters* e *Policies* do TC e (iii) interface de filtragem da aplicação NEASwitchD.py. Na Figura 34 observa-se o comportamento inicial da aplicação.

Na Figura 34 é possível observar que a aplicação realizou a captura de todas as primitivas que trafegam pelo *Switch*, analisando uma a uma e registrando informações relativas aos cabeçalhos. Na Figura 35 é possível visualizar as regras de encaminhamento previstas na *Forwarding Data Base (FDB)* da *Bridge* juntamente com as *Queueing Disciplines* ao iniciar a aplicação NEASwitchD.py.

Conforme ilustrado na Figura 35, para cada porta física do *Switch* NEA, neste con-

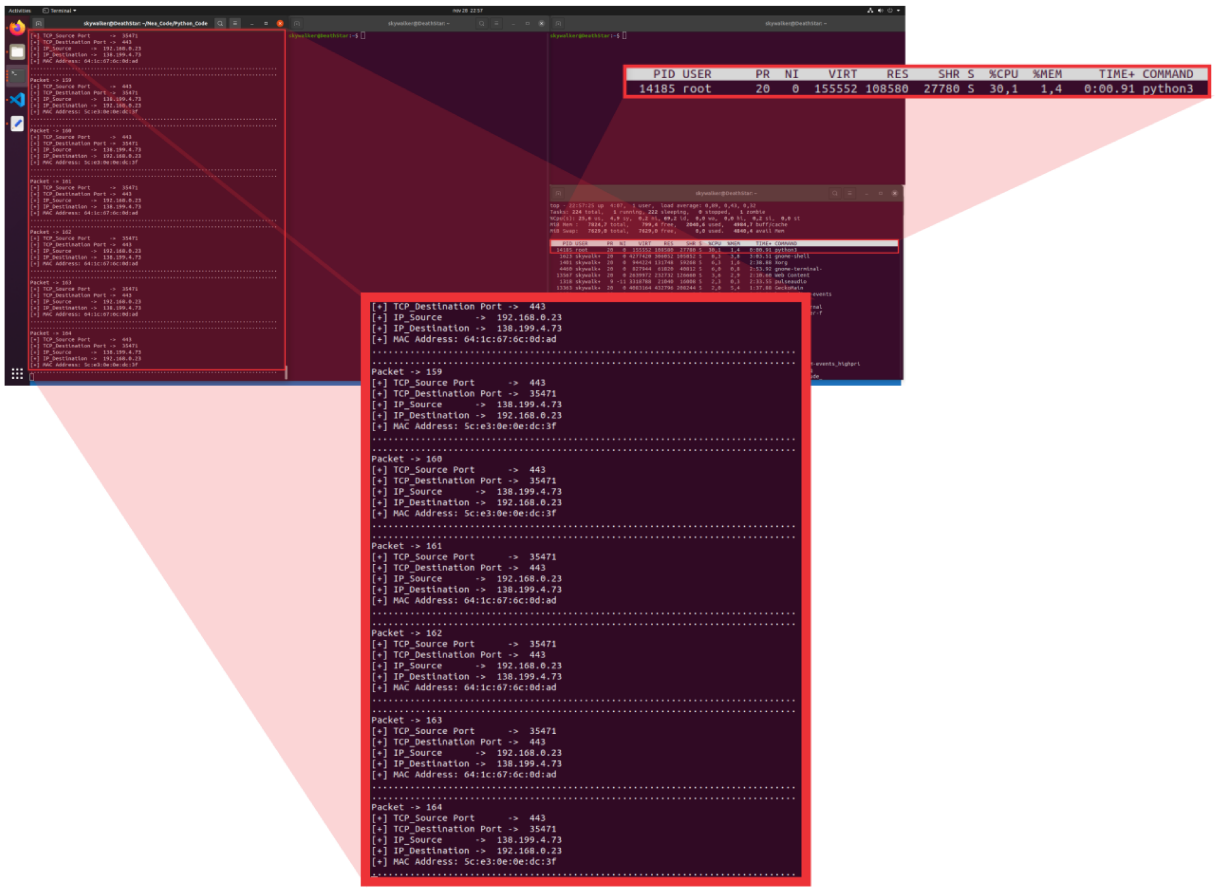


Figura 34 – Resultados registrados pela aplicação NEASwitchD.py

texto, três (3) portas físicas (enp3s0, enp4s0 e enp6s0), criou-se uma *class* geral identificada como *qdisc htb 1: root* e uma *class* filha, identificada por *qdisc sfq 200: parent*, atrelada a *class* pai. A *class* filha trata-se da política *Best Effort* de encaminhamento do *Switch* NEA, que atua em um contexto mais geral, ou seja, sempre que alguma primitiva não possuir uma ação de encaminhamento associada, automaticamente será encaminhada para esta *class*.

A Figura 36 ilustra o recebimento de uma primitiva de controle, neste contexto *AddWkpIND*. Ao identificar uma primitiva de controle ETArch, ou seja, um *frame* com endereço MAC de destino com a seguinte característica: $FE.xx.xx.xx.xx.xx$, a aplicação interpretou o conteúdo presente no cabeçalho do *frame*, identificou qual é o tipo da mensagem, a *Entidade* de origem, os parâmetros de configuração do *Workspace* e a porta física de origem. Após este processo, atualizou-se o *Dataframe* com as informações do *Workspace* para então, instituir as tratativas de escalonamento com o propósito de atender este *Workspace* em questão. Na Figura 37 é possível constatar a atualização da *Forwarding Data Base (FDB)* da *Bridge*, onde adicionou-se uma regra de encaminhamento não somente em relação ao endereço MAC da *Entidade* de origem mas também um caminho

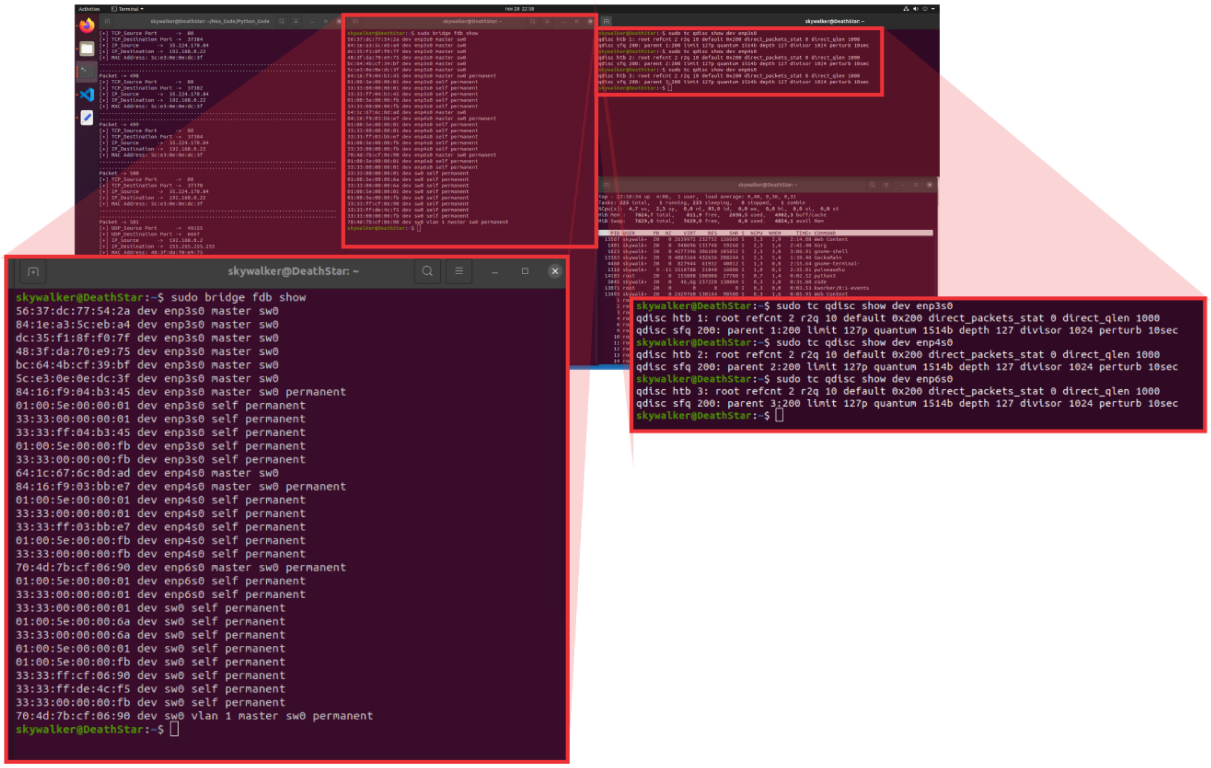


Figura 35 – Resultados registrados pela aplicação NEASwitchD.py - *Forwarding Data Base* e *Classes*

para o *Workspace*.

Conforme já compreendido pelo processo de criação de *Workspace*, torna-se necessário incluir uma entrada de encaminhamento na *Forwarding Data Base (FDB)* da *Bridge* referente ao *Workspace* e não somente em relação à *Entidade*. No momento em que uma *Entidade* ingressa no *Workspace*, ela deve receber todas as primitivas enviadas para o *Workspace* (*multicast*). A princípio pode parecer redundante, mas para materializar o domínio de comunicação, o *Workspace* necessita ter uma identificação única no *Switch* NEA além da *Entidade*.

A Figura 38 ilustra as *classes* e *filters* configurados até o momento.

Na Figura 36, observa-se que o *Workspace* é identificado através de um *Classid*, neste caso trata-se de um identificador utilizado nas tabelas de *classes* e *filters*. Cada *Classid* é gerado randomicamente no momento da criação do *Workspace*. No exemplo, foi utilizado o *Classid* 65.

Com relação ao *Workspace* 65 em questão, até este momento, havia apenas uma *Entidade* ligada participando do *Workspace* em questão, ou seja, apenas na porta física *enp4s0* tem-se a ação de encaminhamento associada ao *Workspace* 65. Conforme novas *Entidades* podem vir a serem inseridas neste *Workspace*, essas mesmas ações são propagadas nas portas físicas das respectivas *Entidades*. A Figura 39 ilustra os parâmetros que foram

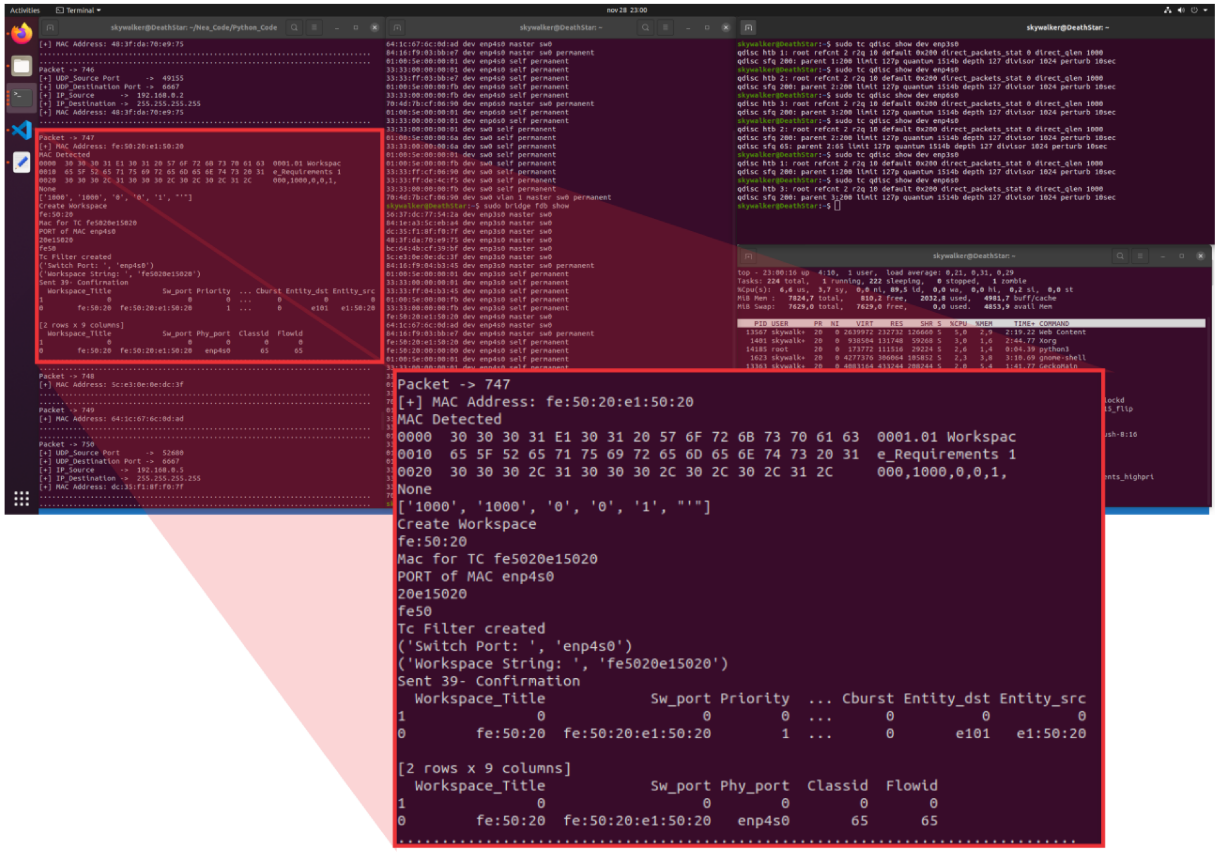


Figura 36 – Resultados registrados pela aplicação NEASwitchD.py - Recebimento da primitiva de controle AddWkpIND.

configurados para as *classes*.

Na Figura 39, observam-se os diferentes níveis de prioridades, *Rate*, *Ceil*, *Burst* e *Cburst* associados às *classes*. Estes parâmetros regulam o tráfego pela *class* naquela determinada porta física. A Figura 40 ilustra os filtros associados às *classes*.

É possível visualizar na Figura 40 os filtros configurados na porta física *enp4s0*. Observa-se que o filtro foi criado mantendo a associação com o *Workspace*, neste contexto, 2:65 (*Flowid*) como regra de *match*, o endereço MAC do *Workspace* e a(s) *Entidade(s)* que fazem parte do domínio de comunicação.

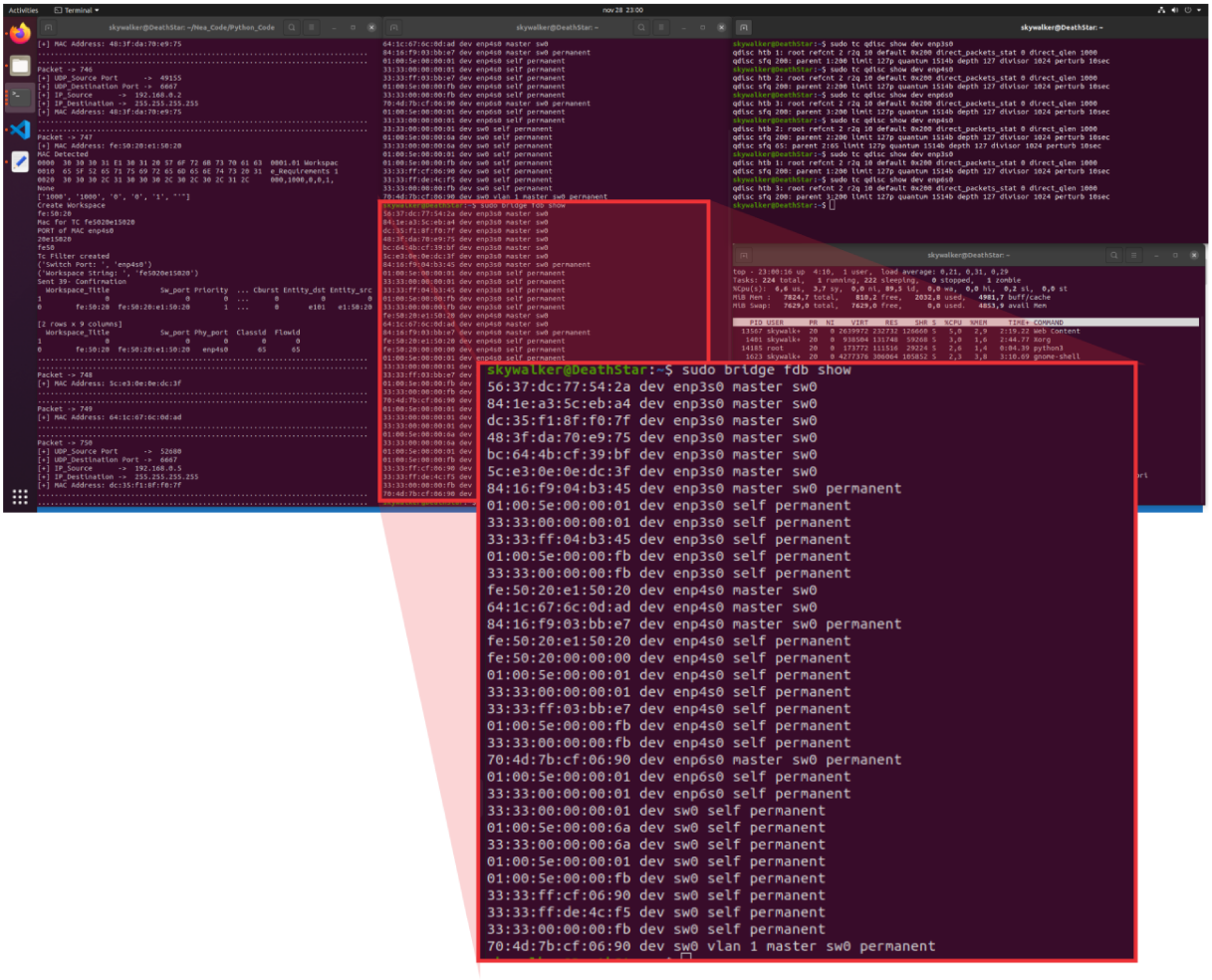


Figura 37 – Resultados registrados pela aplicação NEASwitchD.py - Atualização da *Forwarding Data Base* da *Bridge*

7.1.2 Mensagem de Controle *Edit Workspace*

A edição de um determinado *Workspace* parte da necessidade de inserir uma *Entidade* no domínio de comunicação. A Figura 41 ilustra em detalhe o processo de troca de mensagens entre DTSA e o *Switch* NEA para a edição de um *Workspace*.

Analisando em detalhes a troca de mensagem ilustrada pela Figura 41, inicialmente, quando uma *Entidade* não está inserida em nenhum *Workspace* e transmite uma mensagem que passe pelo elemento de rede, o *Switch* NEA direcionará esta mensagem em questão para DTSA. Neste momento, torna-se necessário editar o *Workspace* para inclusão da nova *Entidade*.

Para o processo de *Edit Workspace*, a troca de mensagens de controle inicia-se com o DTSA emitindo um *EditWkpREQ*, o *Switch* NEA recebe *EditWkpIND* e conforme o diagrama de estados da NEA, o estado *PARSER* direciona a primitiva para o processo de edição de *Workspace*. Conforme já previsto, inicialmente verifica-se a existência do

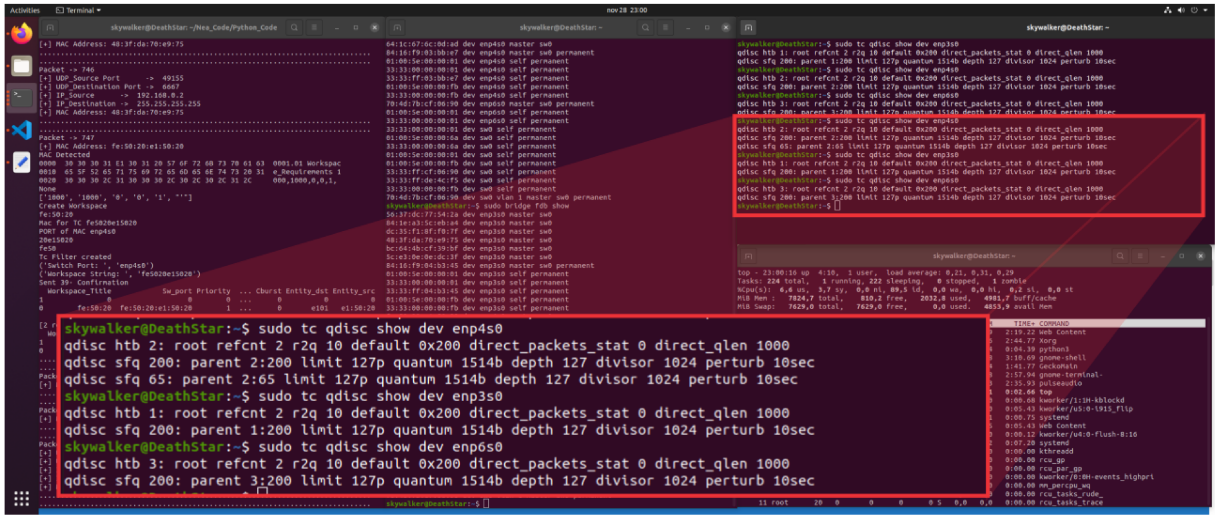


Figura 38 – Resultados registrados pela aplicação NEASwitchD.py - Atualização das *classes*.

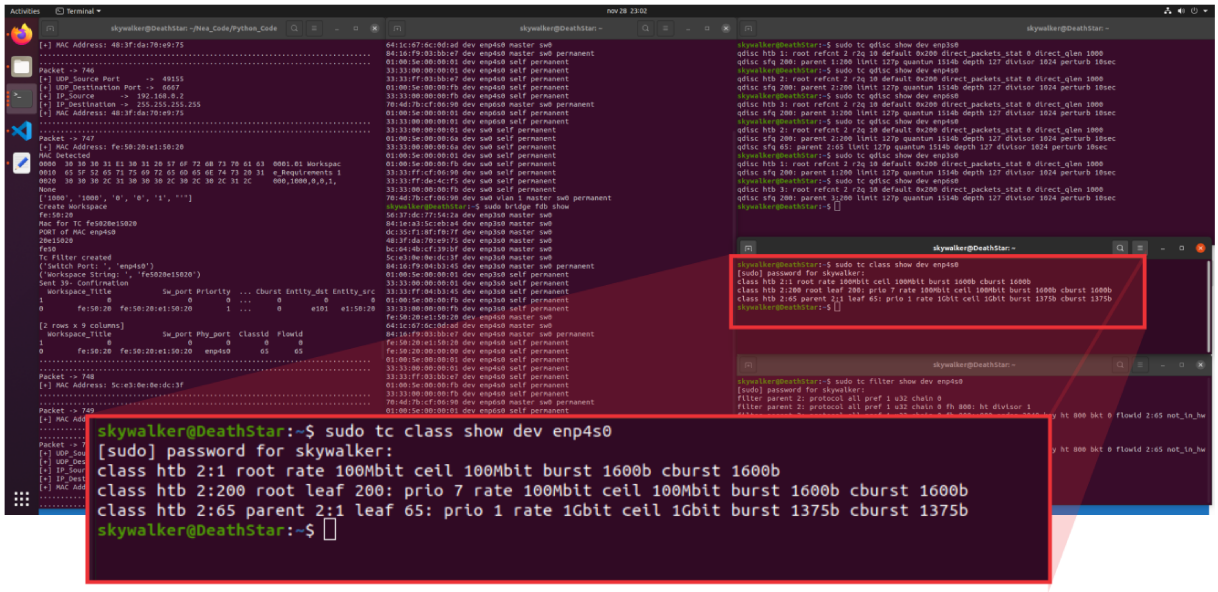


Figura 39 – Resultados registrados pela aplicação NEASwitchD.py - Checagem dos parâmetros

Workspace em questão e na sequência atualiza toda a de base de dados de *Workspaces* e ações de escalonamento.

Importante ressaltar que não existe nenhuma troca de mensagens de controle entre *Entidades* e o *Switch* NEA. Na Figura 41 é possível observar duas mensagem de dados enviadas para o *Switch* em tempos diferentes, a segunda mensagem trata-se de um retransmissão (*timeout*) por parte da *Entidade*.

A Figura 42 ilustra o cenário onde existem dois *Workspaces* configurados no elemento

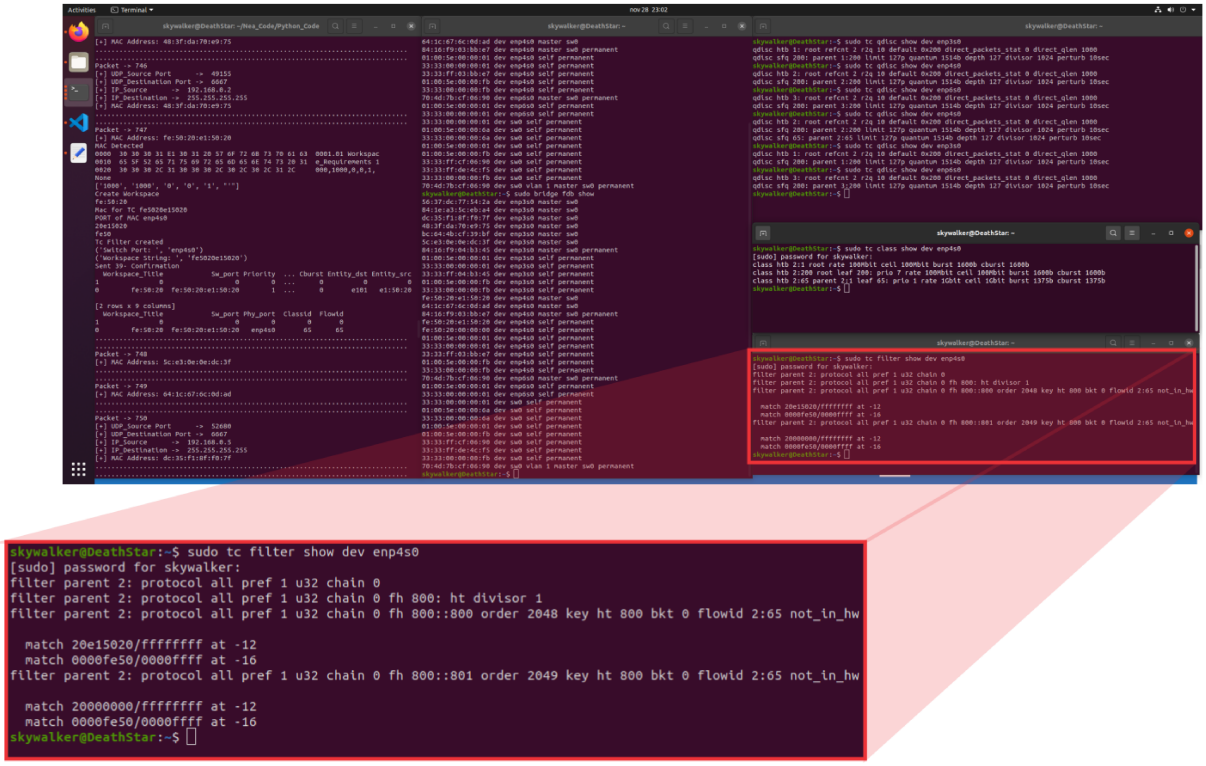


Figura 40 – Resultados registrados pela aplicação NEASwitchD.py - Atualização dos *filters*.

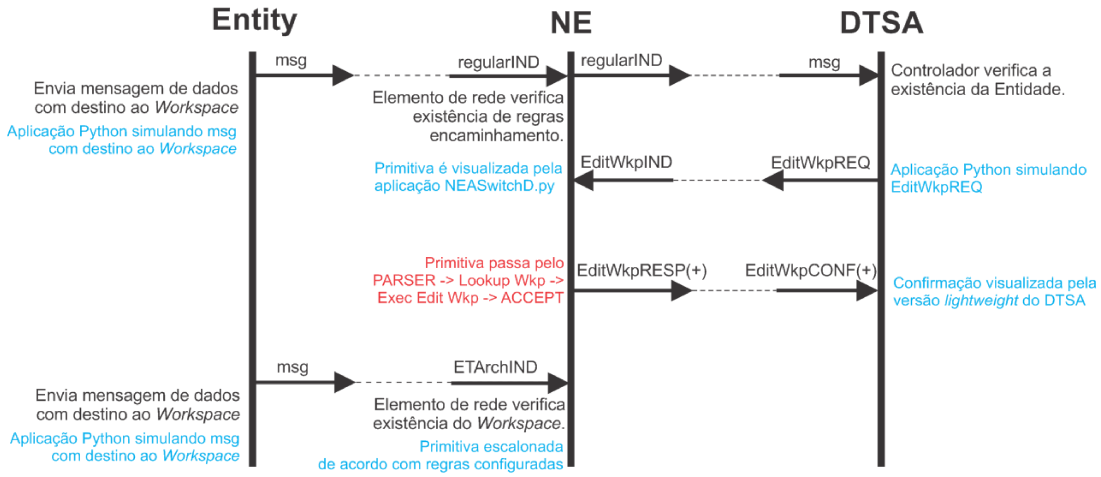


Figura 41 – Troca de mensagens de controle para edição de *Workspace*.

de rede, respectivamente identificados como *FE:50:30* e *FE:50:60*, com *Classid* 23 e 100.

NA Figura 43 observa-se o comportamento do *Switch* NEA ao receber uma primitiva de dados ETArch. O *Switch* NEA identifica o tipo da mensagem pela estrutura do endereço MAC, porém, por não se tratar de uma primitiva de controle e a inexistência de uma ação de encaminhamento associada, o *Switch* NEA apenas direciona a mensagem para a fila

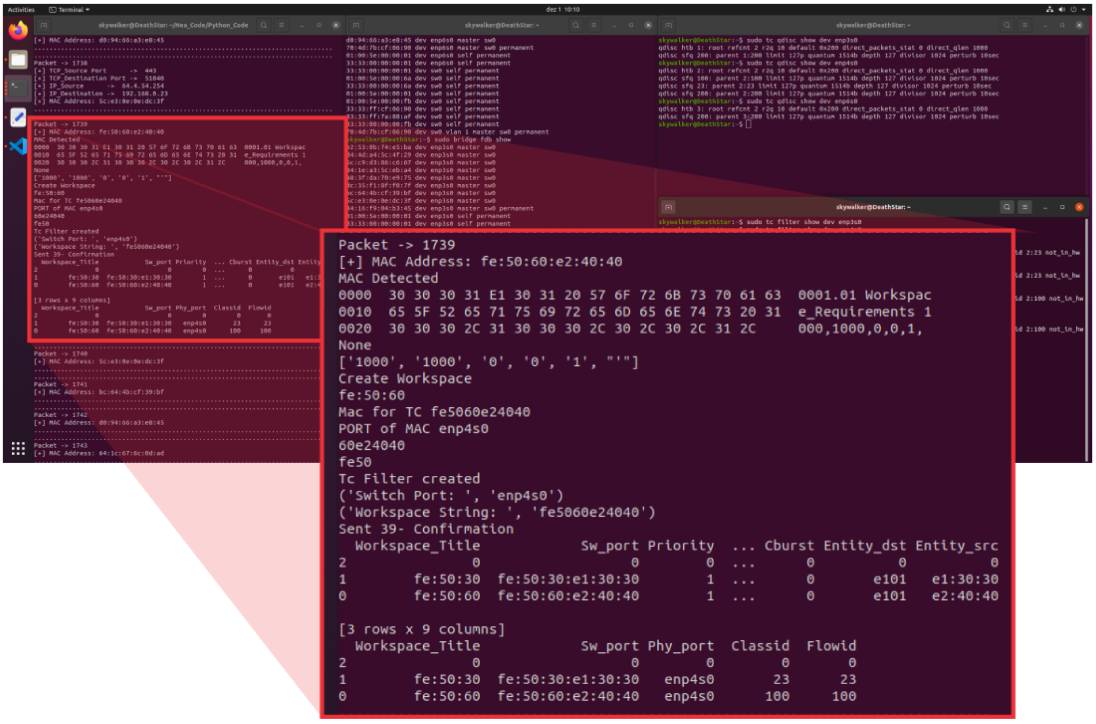


Figura 42 – Resultado registrados pela aplicação NEASwitchD.py - Impressão da tabela de *Workspaces*.

de saída com o objetivo de alcançar o DTSA sob a política *Best Effort*.

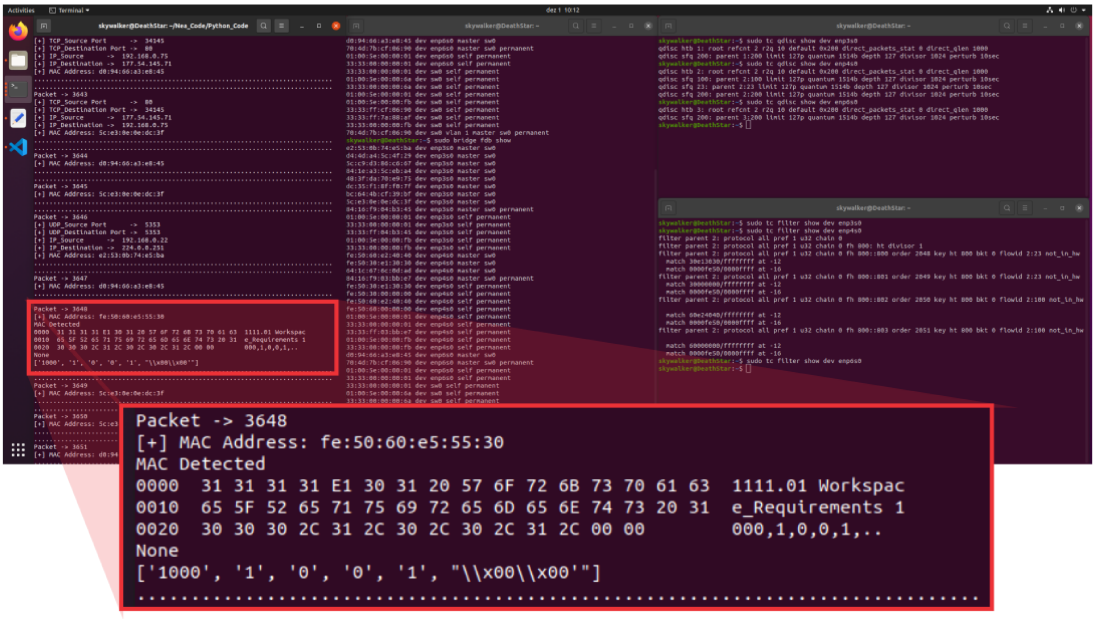


Figura 43 – Resultados registrados pela aplicação NEASwitchD.py - Recebimento de uma primitiva ETArch genérica.

Na Figura 44 observa-se o comportamento do *Switch* NEA ao receber uma primitiva ETSCP *EditWkpIND*. Inicialmente o *Switch* NEA identificou a mensagem de controle e

conforme previsto no diagrama de estados, verificou a existência do *Workspace* no *Dataframe*, na sequência foram extraídas as informações de controle da mensagem. Neste exemplo, foi solicitado pelo DTSA a inserção da *Entidade FE:50:60:E5:55:20* no *Workspace* identificado como *FE:50:60*.

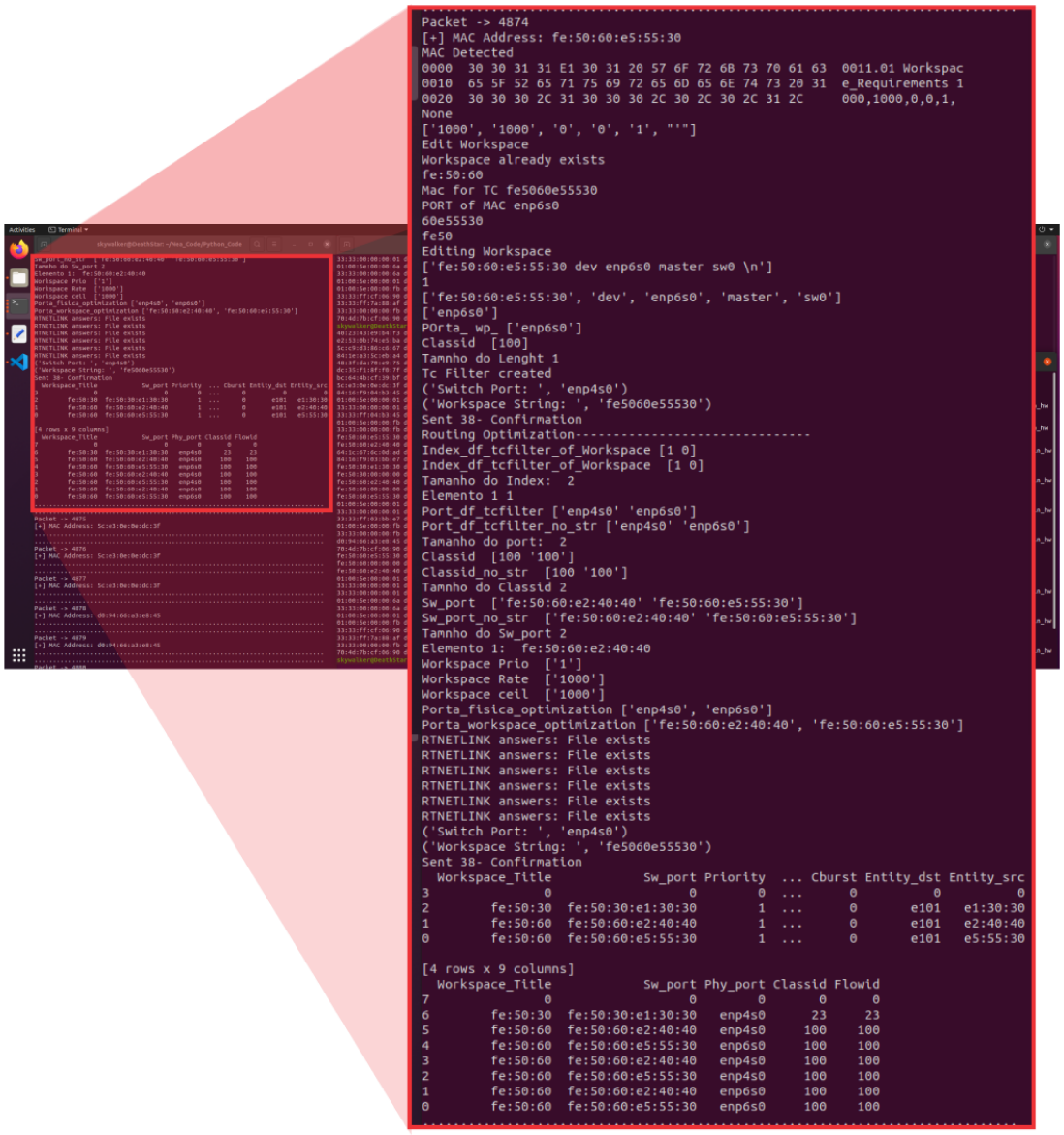


Figura 44 – Resultados registrados pela aplicação NEASwitchD.py - Recebimento de uma primitiva ETSCP de edição de *Workspace*.

O *Switch* NEA sempre faz a verificação na *Forwarding Data Base (FDB)* para identificar em qual porta física a *Entidade* está conectada. Não é possível adicionar uma *Entidade* que não esteja conectada fisicamente ao *Switch* NEA. Faz parte do processo de edição de *Workspace* realizar essa conferência.

Após concluída a verificação descrita, a base de dados de *Workspace* foi atualizada com a inserção da nova *Entidade* e as ações de encaminhamento baseadas no *Workspace*

em questão foram propagadas.

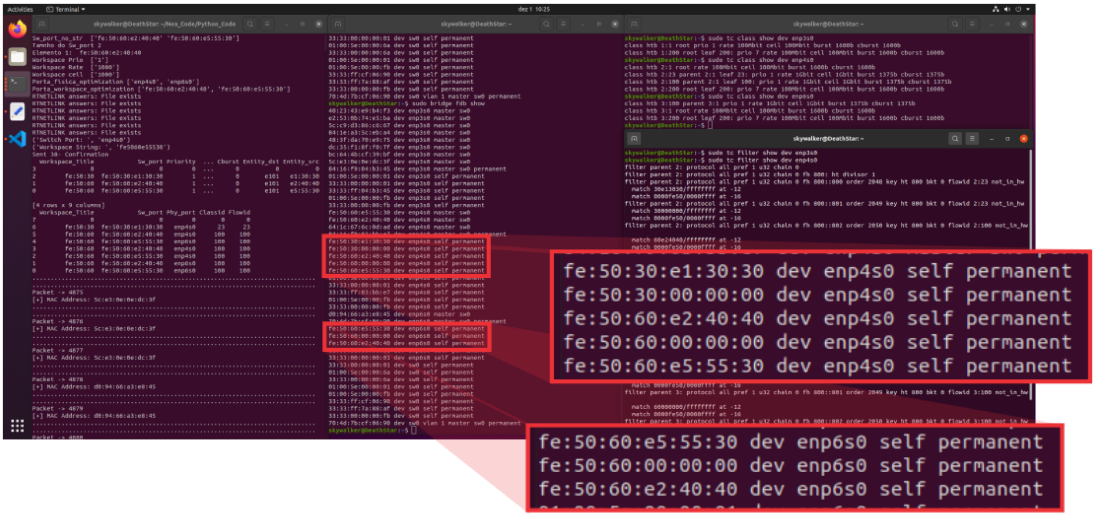


Figura 45 – Resultados registrados pela aplicação NEASwitchD.py - Otimização de rotas no processo de edição de *Workspace*.

Na Figura 45 é possível observar os registros da *FDB*. As entradas possuem o aspecto permanente, pois, de fato é a aplicação *NEASwitchD.py* que controla as inserções e exclusões permanentes.

Por fim, na Figura 46 observa-se a última etapa do processo de edição de *Workspace*, onde foram criadas as *classes*. Nota-se que, uma nova *Entidade* adicionada em um determinado *Workspace* exige a criação de uma nova *class* na porta física atrelada a *Entidade* e, por conveniência, a instituição de novos *filters* herdando as características do *Workspace* em questão.

Conforme a Figura 46, novos *filters* foram adicionados à porta física *enp6s0*, porta onde a *Entidade* em questão está conectada. Foram adicionadas também, políticas de *match* tanto para o *Workspace* quanto para a *Entidade*.

7.1.3 Mensagem de Controle *Remove Workspace*

A remoção de um *Workspace* é um processo conduzido pelo DTSA. Não é objetivo deste trabalho discutir ou apresentar as motivações para exclusão de *Workspaces*. Para a remoção de um *Workspace*, o *Switch* NEA recebe uma mensagem de controle (ETSCP - *Remove Workspace*) com a identificação do *Workspace* que será excluído.

A Figura 47 ilustra a troca de mensagens entre o DTSA e o *Switch* NEA com a finalidade de remover o *Workspace*. Inicialmente, o DTSA emite uma mensagem *DeleteWkpREQ* e o *Switch* NEA recebe *DeleteWkpIND*, o *PARSER* identifica e direciona a mensagem para o processo de remoção de *Workspace*. Caso alguma primitiva relativa ao *Workspace* removido ingresse no *Swtich* NEA após o processo de remoção, a primitiva será tratada sob a política *Best Effort*.

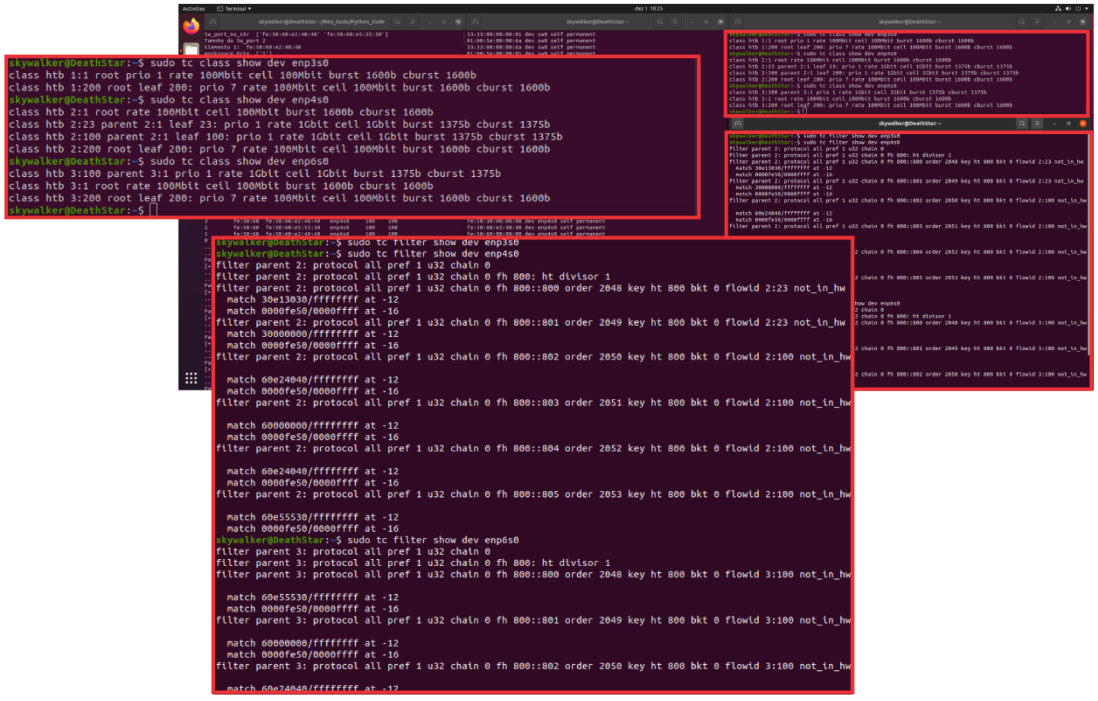


Figura 46 – Resultados registrados pela aplicação NEASwitchD.py - Adição das *Classes* e *Filters* relativos a nova *Entidade* inserida no *Workspace*.

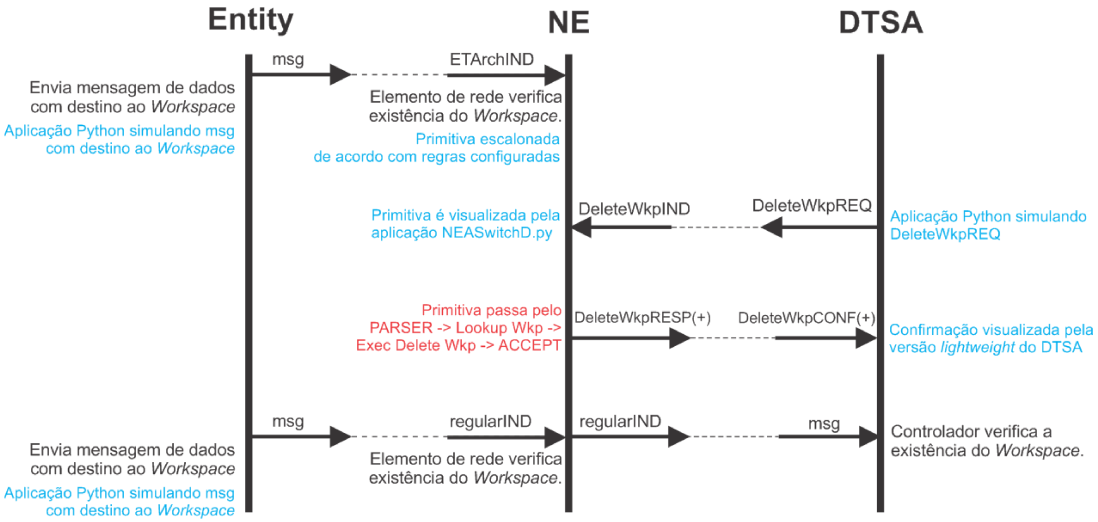


Figura 47 – Troca de mensagens de controle para Remoção de *Workspace*.

Na Figura 48 é possível observar o recebimento da mensagem de controle indicando a remoção de *Workspace*. Primeiramente realizou-se a consulta no *Dataframe* para verificar a existência do *Workspace* em questão, na sequência foi instituída uma estrutura de lista, contendo registro de todas as portas físicas que possuem referência ao *Workspace* em questão.

Neste contexto, onde o *Workspace* a ser excluído possui a identificação *FE:50:60*, a aplicação *NEASwitchD.py* realizou uma varredura em todas as portas físicas com a

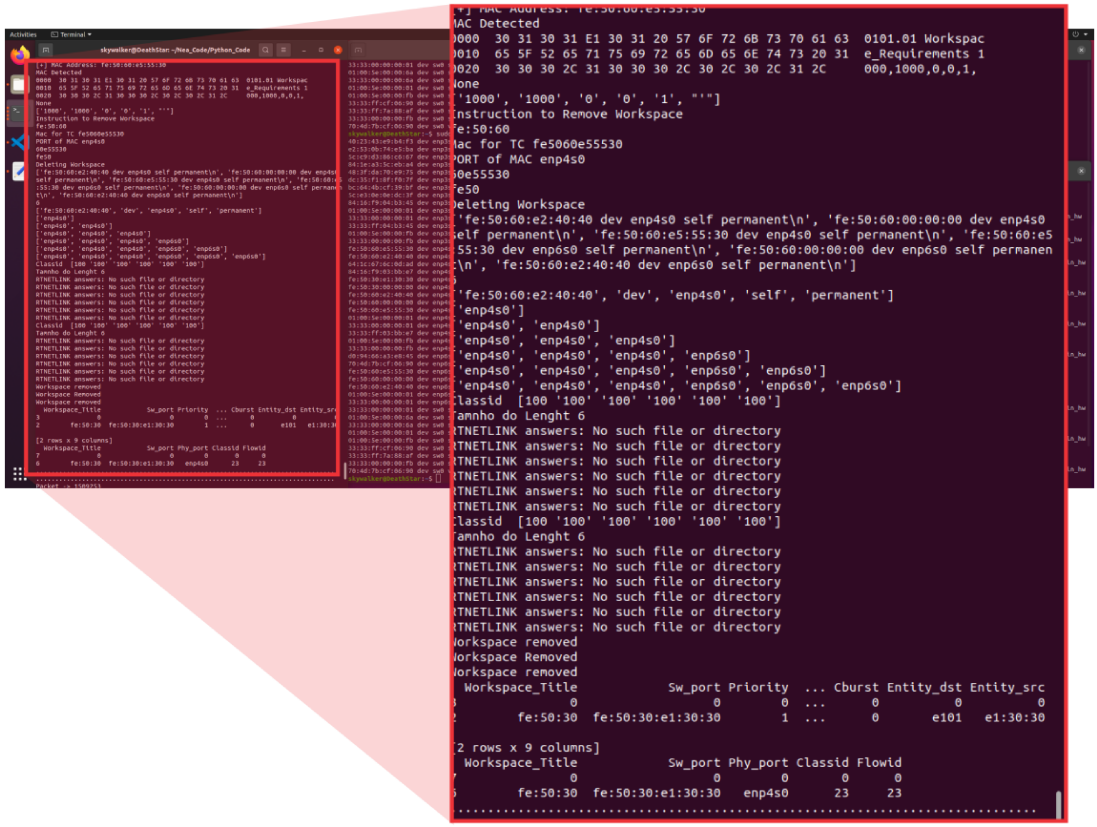


Figura 48 – Resultados registrados pela aplicação NEASwitchD.py - Recebimento da mensagem de remoção de *Workspace*.

intenção de verificar a presença do *Workspace*. Após a varredura, aplicação registrou a identificação da porta física em uma lista estruturada para praticar o processo de exclusão.

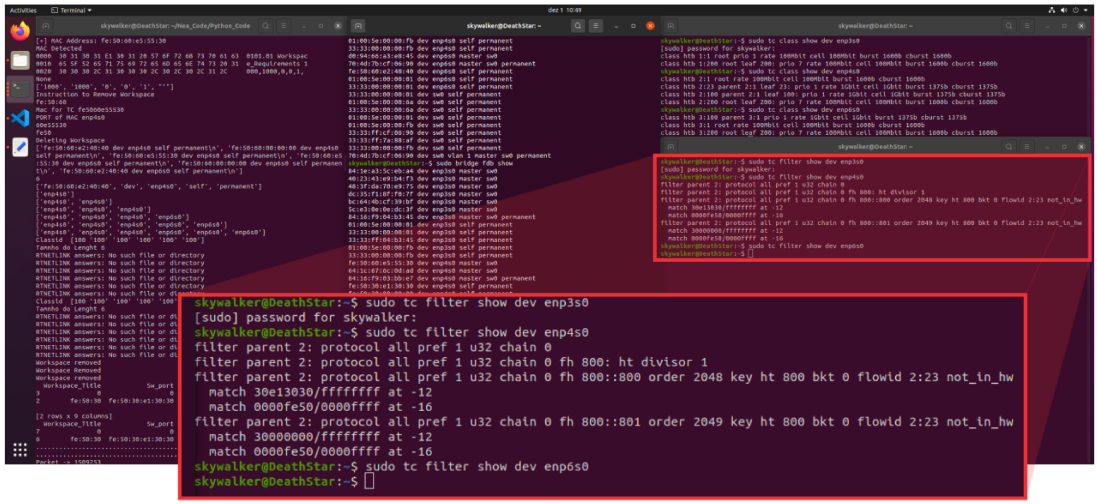


Figura 49 – Resultados registrados pela aplicação NEASwitchD.py - Remoção dos *Filters* relativos ao *Workspace*.

Na Figura 49 observa-se que não existe nenhum *Filter* com *match* para o *Workspace*

FE:50:60, apenas para o *FE:50:30* que permanece ativo no elemento de rede.

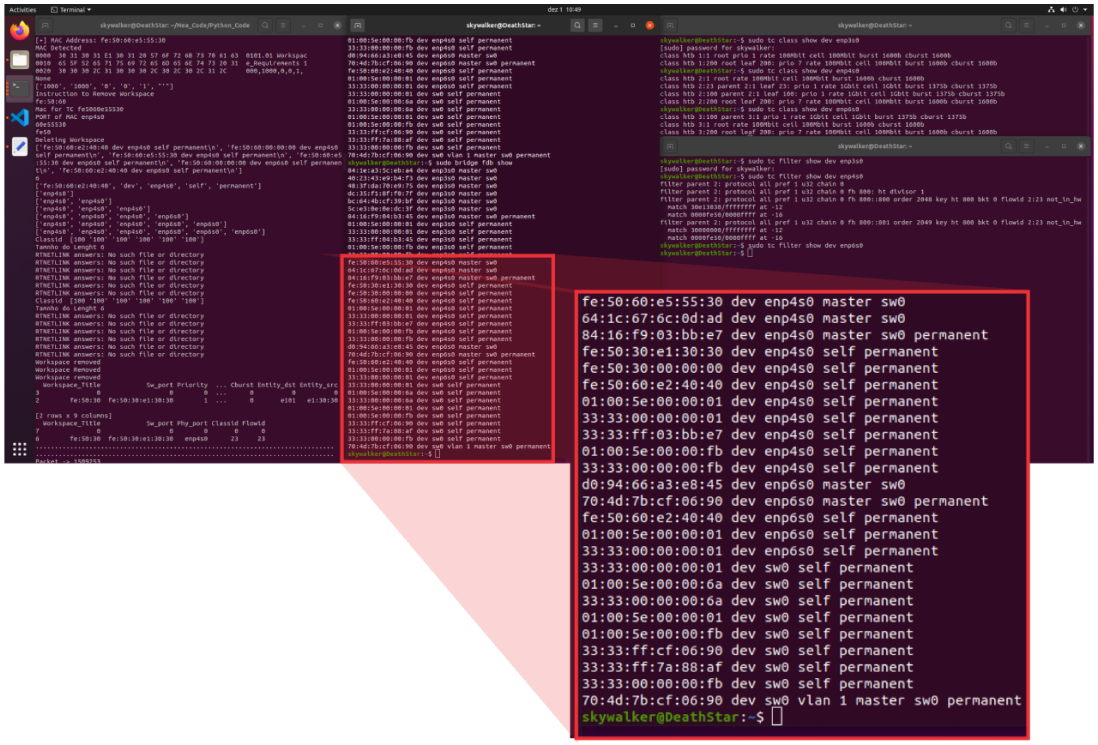


Figura 50 – Resultados registrados pela aplicação NEASwitchD.py - Atualização da *FDB* da *Bridge*.

Na Figura 50 é possível observar a *Forwarding Data Base (FDB)* após o processo de remoção do *Workspace*. Nota-se que não existe nenhuma ação de encaminhamento direcionada para o *Workspace FE:50:60:00:00:00*, restando apenas as rotas das *Entidades* por estarem fisicamente conectadas.

7.1.4 Mensagem de Controle *Update Workspace*

A atualização de um *Workspace* se resume em atualizar os parâmetros de prioridade, largura de banda e *burst* utilizados nas ações de encaminhamento de um determinado *Workspace*. No momento da criação de um determinado *Workspace*, sempre é especificado pelo DTSA quais são as necessidades da comunicação, tais como prioridade de tráfego e banda disponível para utilização. Alterar essas configurações significa ofertar mais ou menos capacidades para o tráfego relativo ao *Workspace*. Para o *Switch* NEA, isso significa redefinir os parâmetros inicialmente definidos.

Na Figura 51 observa-se a troca de mensagens entre o DTSA e o*Switch* NEA. As motivações do DTSA para alteração das capacidades do *Workspace* não fazem parte do escopo deste trabalho, sendo assim, o *Switch* NEA apenas recebe e processa a primitiva de controle *ETSCP - Update Workspace*.

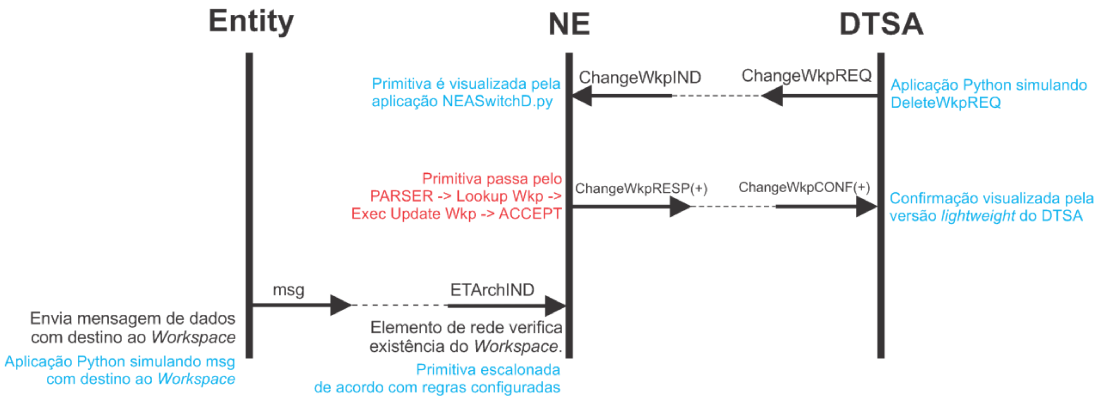


Figura 51 – Troca de mensagens de controle para modificação das capacidades do *Workspace*.

O processo de atualização das capacidades do *Workspace* inicia-se com a verificação da existência do *Workspace* no *Dataframe*. Uma vez confirmada a existência do *Workspace*, são realizadas alterações nas *classes* do *TC* relativas ao *Workspace* em questão.

Em suma, mudar as capacidades do *Workspace* significa alterar os parâmetros configurados nas *classes* relativas ao *Workspace*. Este processo deve se propagar por todas as portas físicas em que o *Workspace* possui correspondência.

Na Figura 52 observa-se o processo de atualização das capacidades do *Workspace Classid* 23, que foi inicialmente instituído com os seguintes parâmetros: *Priority:1*, *Rate:1Gbit*, *Ceil:1Gbit*, *Burst:1375b* e *CBurst:1375b*, alterado para *Priority:7*, *Rate:10Mbit*, *Ceil:10Mbit*, *Burst:1600b* e *CBurst:1600b*.

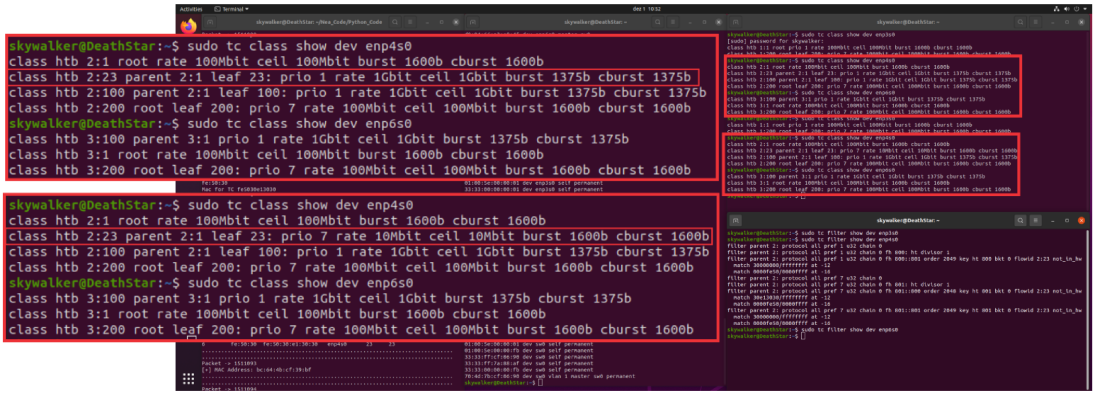


Figura 52 – Resultados registrados pela aplicação NEASwitchD.py - Atualização das capacidades do *Workspace*.

A alteração nas capacidades do *Workspace* não possui interferência nas políticas de filtragem, ou seja, não é realizada nenhuma inserção, exclusão ou modificação nas regras de *match* dos *filters*.

7.2 Análise da Capacidade do *Switch* NEA em Manusear Parâmetros de QoS de Aplicações

Para validar a atuação do *Switch* NEA em relação à capacidade de (re)configuração dos parâmetros de *Quality of Service (QoS)* orientado pela aplicação, foram instituídos 4 (quatro) experimentos, cada um deles com diferentes requisitos de comunicação que envolvem a participação de *Workspaces*.

Não fez parte do escopo desta pesquisa o desenvolvimento de um controlador, contudo para os testes foi implementado uma versão *Lightweight* do DTSA. Esta versão de controlador possui apenas a capacidade de enviar primitivas ETSCP para o *Switch* NEA.

Na Figura 53 é ilustrado a estrutura real utilizada para os experimentos e testes.

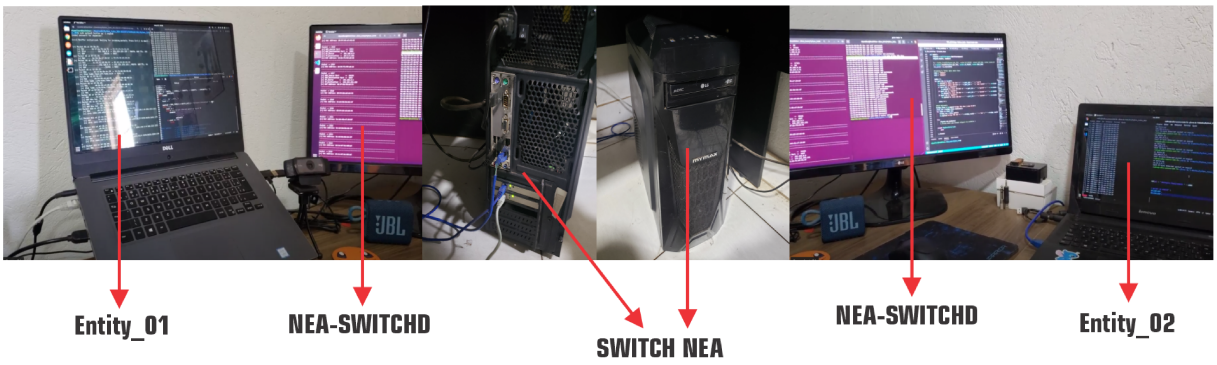


Figura 53 – Ambiente real utilizado nos experimentos.

Na Tabela 6 é apresentado o descritivo das configurações dos equipamentos utilizados nos experimentos e uma breve explanação do comportamento de cada um deles.

Embora o ambiente ilustrado na Figura 53 tenha sido utilizado em todos os experimentos, as capacidades das comunicações foram alteradas em tempo real para cada experimento, permitindo observar a atuação do *Switch* NEA na manutenção da QoS das comunicações.

Os experimentos desenvolvidos objetivaram simular situações reais de tráfego com a atuação sistemática do controlador da rede na criação de *Workspaces* e na atualização dos requisitos das comunicações. Lembrando que, para os experimentos realizados foram utilizados os parâmetros de prioridade (prioridade de tráfego por porta física) e também de controle de banda (*rate*) para garantia da QoS.

Tabela 6 – Descrição dos equipamentos utilizados nos experimentos 1 e 2

Ident.	**Configuração**	**Descrição**
Switch NEA	*Hardware Computacional equipado com 2 (duas) interfaces de rede Offboard (Placa rede Gigabit Pci Express Tp Link Tg-3468 10/100/1000), 1 (uma) Placa de rede Gigabit LAN Realtek RTL8111H Onboard 10/100/1000, HD sata ADATA 240Gb, Processador Pentium G4400 3.30Ghz, Memória Ram de 8Gb DDR4 2133Mhz Crucial e Sistema Operacional Linux Ubuntu 20.04.*	Equipamento que centraliza toda a solução proposta deste trabalho. Foi utilizado a linguagem de programação Python versão 3.8.5. para descrever toda a solução *NEA-SWITCHD*
Entity_01	*Hardware Computacional (Laptop) equipado com 1 (uma) Placa de rede Gigabit LAN RTL8111/8168/8411 Onboard 10/100/1000, HD sata WD 480Gb, Processador i5-7200U 2.50GHz, Memória Ram de 16Gb DDR4 2400Mhz e Sistema Operacional Linux Ubuntu 20.04.*	Equipamento utilizado para representar a *Entity_01*. Foi utilizado a linguagem de programação Python versão 3.8.5. para descrever toda a solução para envio de primitivas
Entity_02	*Hardware Computacional (Laptop) equipado com 1 (uma) Placa de rede Gigabit LAN RTL8111/8168/8411 Onboard 10/100/1000, HD sata WD 240Gb, Processador i3-4005U 1.70GHz, Memória Ram de 8Gb DDR3 1600Mhz e Sistema Operacional Linux Ubuntu 18.04.*	Equipamento utilizado para representar a *Entity_02*. Foi utilizado a linguagem de programação Python versão 3.8.5. para descrever toda a solução para envio de primitivas
Controller	*Hardware Computacional (Laptop) equipado com 1 (uma) Placa de rede Gigabit LAN RTL8111/8168/8411 Onboard 10/100/1000, HD sata WD 240Gb, Processador i3-4005U 1.70GHz, Memória Ram de 8Gb DDR3 1600Mhz e Sistema Operacional Linux Ubuntu 18.04.*	Equipamento utilizado para representar o *Controller* (DTSA). Foi utilizado a linguagem de programação Python versão 3.8.5. para descrever toda a solução para envio de primitivas de controle (*ETSCP*) para o *Switch* NEA

7.2.1 Experimento 1 - *Workspace_01 X Streaming Application*

Na Figura 54 é apresentado o ambiente utilizado para o experimento 1. Observa-se que a *Entity_02* possui, no mesmo dispositivo físico, duas aplicações distintas. Para contextualizar este experimento, a aplicação que faz uso do fluxo regular (*Workspace*

Regular Flow) é uma aplicação de *Streaming*, sendo o *Workspace_01* relativo a uma aplicação básica de envio e recebimento de primitivas de dados.

No contexto do *Workspace_01*, a *Entity_01* envia primitivas de dados de 56 *bytes* de tamanho para a *Entity_02* e a *Entity_02* envia primitivas de dados de 56 *bytes* de tamanho para a *Entity_01*, ambas as transmissões operam numa vazão pré-definida de 25 *frames/s*, resultando em uma velocidade mínima requerida de 11 *Kbit/s*.

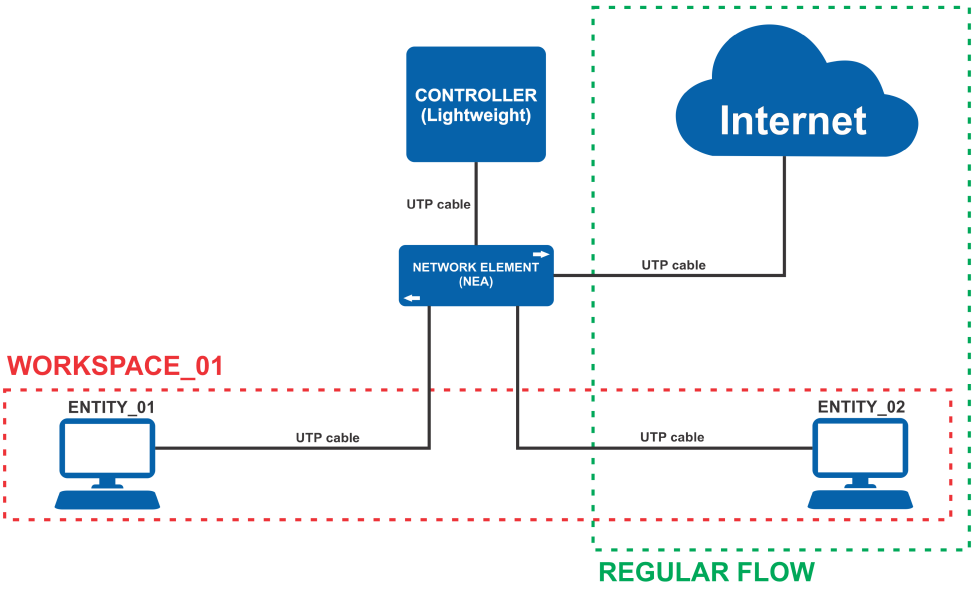


Figura 54 – Ambiente de testes utilizado para o experimento 1.

A sequência de tarefas descritas a seguir definem o processo de teste para o experimento 1.

1. Criar o *Workspace 01*;
2. Inserir duas *Entidades* no *Workspace 01*;
3. Atualizar as capacidades do tráfego regular no *Switch* NEA;
4. Monitorar e registrar os recursos de memória e processador do *Switch* NEA;
5. Monitorar e registrar a quantidade de *frames* processados e encaminhados pelo *Switch* NEA durante o tempo de 1 minutos e 35 segundos.

A seguir aparecem os valores atribuídos aos parâmetros de controle, que neste contexto representam os requisitos das comunicações.

- **Streaming Application**: *Priority: 1 - Rate: 1Gbit - Ceil: 1Gbit - Burst: 1600b - CBurst: 1600b.*

- **Workspace 01**: *Priority: 5 - Rate: 100mbit - Ceil: 100mbit - Burst: 1600b - CBurst: 1600b.*

Na Figura 55 é ilustrado em (a) a vazão no elemento de rede (*frames/s*) referente às aplicações (*Streaming Application e Workspace_01*). Ainda referente as mesmas aplicações, em (b) é ilustrado o quantitativo de *frames* processados e encaminhados pelo *Switch* NEA durante o mesmo período de tempo.

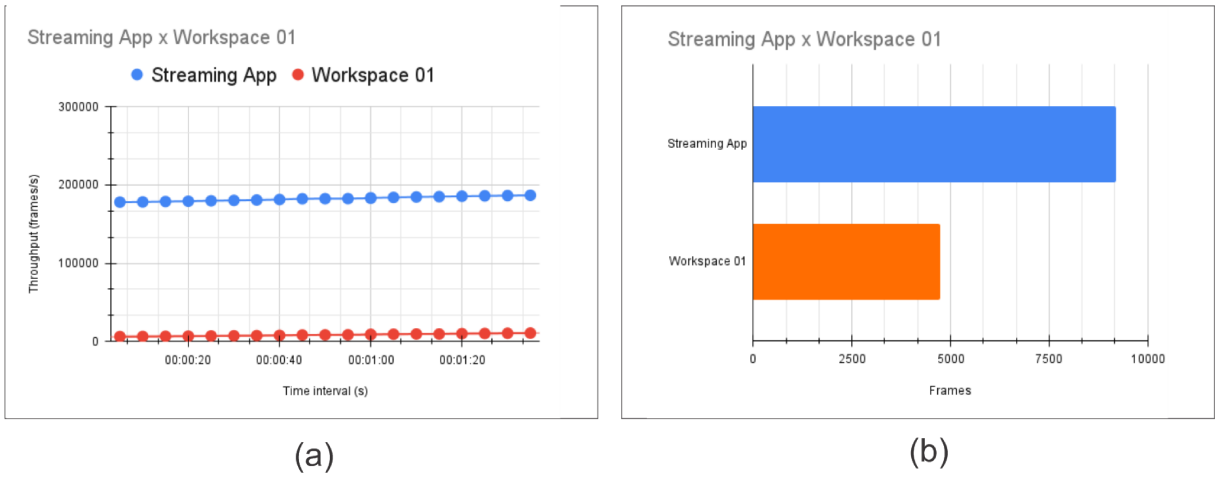


Figura 55 – Resultados relativos ao cenário de testes 1.

De acordo com a Figura 55 (a), observa-se que a aplicação *Streaming* manteve uma vazão (*frames/s*) constante, representada no gráfico pela leve inclinação da reta. De acordo com ZEN (2019), aplicações *Streaming*, operando em *standard definition* (SD), exigem algo em torno de 500 $Kbit/s$ de velocidade mínima, e para aplicações em *High Definition* (HD), algo em torno de 2,5 $Mbit/s$. No caso do experimento 1 foi definido como parâmetro de vazão para a aplicação *Streaming rate: 1 Gbit/s* e *rate: 100 Mbit/s* para o *Workspace*. Ao analisar a velocidade mínima de ambas as aplicações, os parâmetros definidos foram mais que suficiente para garantir a QoS das aplicações.

A Tabela 7 ilustra os dados de vazão coletados no experimento 1. Os dados detalhados em relação ao quantitativo de *frames* observados durante o experimento podem ser visualizados no Apêndice A.

Na Figura 55 (b) é ilustrado o volume de *frames* processados e encaminhados pelo *Switch* NEA durante o intervalo de tempo de 1 minuto e 35 segundos. Para o experimento 1, observou-se que o *Switch* NEA foi capaz de instituir o *Workspace*, adicionar *Entidades* e atuar sobre os parâmetros de prioridade e controle de banda preservando a QoS das aplicações.

Tabela 7 – Taxa de tráfego e configuração de parâmetros de QoS - Experimento 1

Entity	**Vazão (f/s)**	**Parâmetros (QoS)**
Entity_01/Workspace	*24,91 frames/s*	Experimento 01
Entity_02/Workspace	*24,91 frames/s*	Priority.: 5, Rate: 100Mbit, Ceil: 100Mbit, Burst: 1600b.
Entity_02/Streaming	*96,78 frames/s*	Experimento 01 Priority.: 1, Rate: 1Gbit, Ceil: 1Gbit, Burst: 1600b.

7.2.2 Experimento 2 - *Workspace_01 X Streaming Application*

O ambiente para o experimento 2 é idêntico ao utilizado no experimento 1. A alteração entre os experimentos é relativa à configuração dos parâmetros de prioridade e controle de banda das aplicações.

A sequência de tarefas descritas a seguir definem o processo de teste para cenário 2.

1. Atualizar as capacidades do tráfego regular (*Workspace Streaming*) no *Switch* NEA;
2. Monitorar e registrar os recursos de memória e processador do *Switch* NEA;
3. Monitorar e registrar a quantidade de *frames* processados e encaminhados pelo *Switch* NEA durante o tempo de 1 minutos e 35 segundos.

A seguir aparecem os valores atribuídos aos parâmetros de controle, que neste contexto representam os requisitos das comunicações.

- ❏ **Streaming Application**: *Priority: 7 - Rate: 100kbit - Ceil: 100kbit - Burst: 1600b - CBurst: 1600b.*
- ❏ **Workspace 01**: *Priority: 5 - Rate: 100mbit - Ceil: 100mbit - Burst: 1600b - CBurst: 1600b.*

Na Figura 56 é ilustrado em (a) a vazão no *Switch* NEA (*frames/s*) referente às aplicações (*Streaming Application e Workspace_01*). Ainda referente as mesmas aplicações, em (b) é ilustrado o quantitativo de *frames* processados e encaminhados pelo *Switch* durante o mesmo período de tempo.

A Tabela 8 ilustra os dados de vazão coletados no experimento 2. Os dados detalhados em relação ao quantitativo de *frames* observados durante o experimento podem ser visualizados no Apêndice A.

De acordo com a Figura 56 (a), a inclinação da reta, caracterizada pela linha laranja, referente à aplicação definida pelo *Workspace_01*, retratou a vazão determinada pela aplicação, em suma, como não houve alteração entre o experimento 1 e 2 a vazão se manteve.

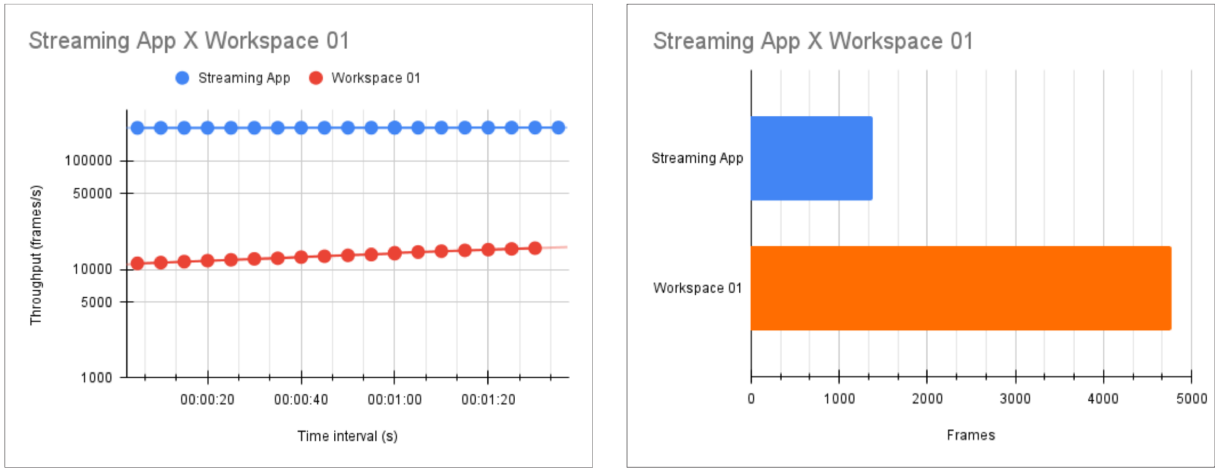


Figura 56 – Resultados relativos ao experimento 2.

Tabela 8 – Taxa de tráfego e configuração de parâmetros de QoS - Experimento 2

Entity	**Vazão (f/s)**	**Parâmetros (QoS)**
Entity_01/Workspace	*25,0 frames/s*	Experimento 02
Entity_02/Workspace	*25,0 frames/s*	Priority.: 5, Rate: 100Mbit, Ceil: 100Mbit, Burst: 1600b.
Entity_02/Streaming	*14,6 frames/s*	Experimento 02
		Priority.: 7, Rate: 100Kbit, Ceil: 100Kbit, Burst: 1600b.

Com relação à aplicação *Streaming*, observou-se que os parametros atualizados não compreendem o mínimo exigido pela aplicação, de acordo com ZEN (2019) para transmissões em Standard Definition (SD) é 500 $Kbit/s$. De acordo com a Figura 56 (b), ficou quase imperceptível a inclinação da reta, caracterizada pela linha azul, retratando a baixa vazão.

Na Figura 56 (b) é ilustrado o volume de *frames* processados e encaminhados pelo *Switch* NEA durante o intervalo de 1 minuto e 35 segundos. A aplicação *Streaming* apresentou uma grande redução na quantidade de *frames* em relação ao experimento 1, o que acabou impactando na QoS da aplicação.

Para o experimento 2, observou-se que o *Switch* NEA foi capaz de atualizar os parâmetros de prioridade e controle de banda em tempo real de execução, sem a necessidade de (re)construir toda a estrutura (*Workspaces* e entidades), preservando a QoS das aplicações, que para este experimento, apenas em relação ao *Workspace_01*, pois na aplicação *Streaming* houve perda da QoS.

Torna-se importante ressaltar que o *Switch* NEA opera sob orientação do controlador da rede, neste contexto o DTSA.

7.2.3 Experimento 3 - *Workspaces X Streaming Application*

Para o experimento 3, não houve alteração física no ambiente de testes, assim como para os outros experimentos. As alterações realizadas foram relacionadas ao quantitativo de *Workspace* e *Entidade* presente no experimento.

No experimento 3 as *Entidades* (*Entity_01 e Entity_02*) fizeram parte de dois *Workspaces* com requisitos diferentes, em suma, houve dois tráfegos com diferentes capacidades diferentes na mesma porta física. Ainda, a *Entity_02* executa uma aplicação *Streaming*.

No contexto do *Workspace_01* e *Workspace_02*, a *Entity_01* enviou primitivas de dados de 56 *bytes* de tamanho para a *Entity_02* e a *Entity_02* enviou primitivas de dados de 56 *bytes* de tamanho para a *Entity_01*, ambas as transmissões trabalharam numa vazão pré-definida de 25 *frames/s*, resultando em uma velocidade mínima requerida de 11 $Kbit/s$.

Na Figura 57 é ilustrado o ambiente utilizado no experimento 3.

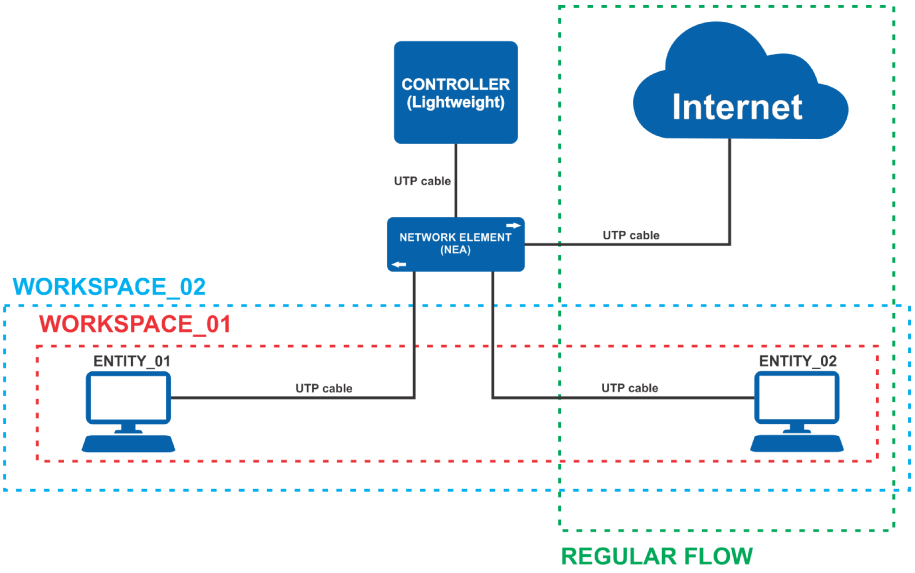


Figura 57 – Ambiente definido para o experimento 3.

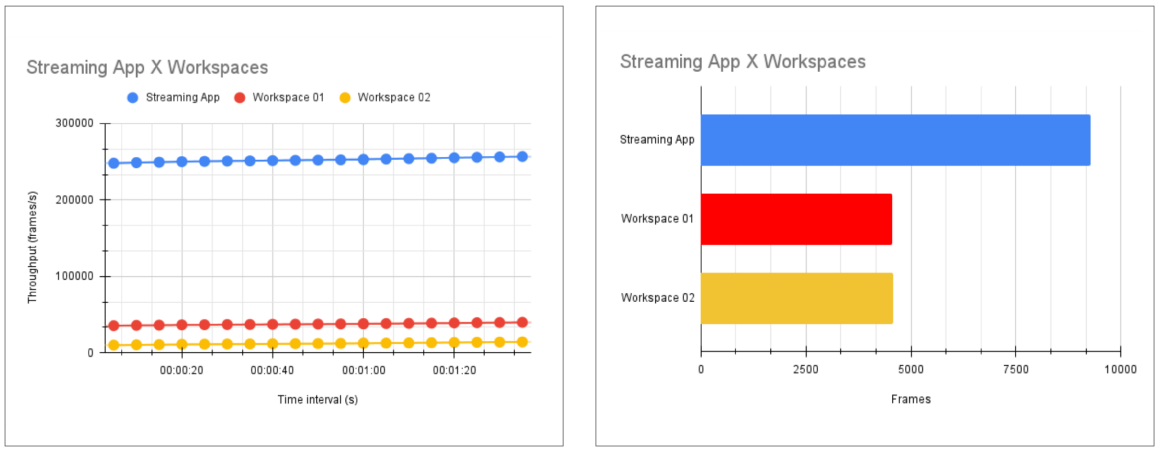
A sequência de tarefas descritas a seguir definem o experimento 3.

1. Criar o *Workspace 2*;
2. Inserir duas entidades no *Workspace 2*;
3. Atualizar as capacidades do tráfego regular no *Switch* NEA;
4. Monitorar e registrar os recursos de memória e processador do *Switch* NEA;
5. Monitorar e registrar a quantidade de *frames* processados e encaminhados pelo *Switch* NEA durante o tempo de 1 minutos e 35 segundos.

A seguir aparecem os valores atribuídos aos parâmetros de controle, que neste contexto representam os requisitos das comunicações.

- **Streaming Application**: *Priority: 7 - Rate: 1mbit - Ceil: 1mbit - Burst: 1600b - CBurst: 1600b.*
- **Workspace 01**: *Priority: 5 - Rate: 1mbit - Ceil: 1mbit - Burst: 1600b - CBurst: 1600b.*
- **Workspace 02**: *Priority: 3 - Rate: 100mbit - Ceil: 100mbit - Burst: 1600b - CBurst: 1600b.*

Na Figura 58 é ilustrado em (a) a vazão no *Switch* NEA (*frames/s*) referente às aplicações (*Streaming Application, Workspace_01 e Workspace_02*). Ainda referente as mesmas aplicações, em (b) é ilustrado o quantitativo de *frames* processados e encaminhados pelo *Switch* NEA durante o mesmo período de tempo.



(a) (b)

Figura 58 – Resultados relativos ao experimento 3.

A Tabela 9 ilustra os dados de vazão coletados no experimento 3. Os dados detalhados em relação ao quantitativo de *frames* observados durante o experimento podem ser visualizados no Apêndice A.

De acordo com a Figura 58 (a), a inclinação da reta, caracterizada pela linha vermelha, referente à aplicação definida pelo *Workspace_01*, retrata a vazão determinada pela aplicação, em suma, como não houve alteração entre o experimento 1, 2 e 3 a vazão se manteve. Apesar da aplicação definida pelo *Workspace_02* apresentar parâmetros diferentes de prioridade e controle de banda em relação ao *Workspace_01*, a vazão é mantida.

Com relação à aplicação *Streaming*, observou-se que os parâmetros atualizados compreendem o mínimo exigido pela aplicação, de acordo com ZEN (2019) para transmissões em SD é 500 $Kbit/s$.

Tabela 9 – Taxa de tráfego e configuração de parâmetros de QoS - Experimento 3

Entity	**Vazão (f/s)**	**Parâmetros (QoS)**
Entity_01/Workspace_01	*23,98 frames/s*	Experimento 03
Entity_02/Workspace_01	*23,98 frames/s*	Priority.: 5, Rate: 1Mbit, Ceil: 1Mbit, Burst: 1600b.
Entity_01/Workspace_02	*24,11 frames/s*	Experimento 03
Entity_02/Workspace_02	*24,11 frames/s*	Priority.: 3, Rate: 100Mbit, Ceil: 100Mbit, Burst: 1600b.
Entity_02/Streaming	*97,71 frames/s*	Experimento 03
		Priority.: 7, Rate: 1Mbit, Ceil: 1Mbit, Burst: 1600b.

Na Figura 58 (b) é apresentado o volume de *frames* processados e encaminhados pelo *Switch* NEA durante o intervalo de 1 minuto e 35 segundos. A aplicação *Streaming* apresentou um aumento de vazão em relação ao experimento 2, o que acabou impactando positivamente na QoS da aplicação.

Para o experimento 3, observou-se que o *Switch* NEA foi capaz de atualizar os parâmetros de prioridade e controle de banda em tempo real de execução, sem a necessidade de (re)construir toda a estrutura (*Workspaces* e entidades), preservando a QoS das aplicações que para este experimento. Foi também possível observar a atuação do *Switch* NEA em diferentes tráfegos pela mesma porta física.

7.2.4 Experimento 4 - *Workspace_ClassId_99 X Streaming Application*

O experimento 4 fez uso da mesma estrutura física utilizada nos outros experimentos.

No experimento 4 instituiu-se um *Workspace* relativo à uma aplicação básica de envio de primitivas e duas *entidades* (*Entity_01 e Entity_02*) e uma aplicação (*Streaming*) denominada *Regular Flow* envolvendo apenas a *Entity_02*.

O objetivo deste experimento, ao contrário dos anteriores, foi realizar sucessivas atualizações nos parâmetros de prioridade e controle de banda das aplicações com o intuito de observar a atuação do *Switch* NEA no recebimento de diversas primitiva de controle e atuação sobre a estrutura de encaminhamento. A sequência de tarefas descritas a seguir definiram o processo de teste para experimento 4.

1. Criar *Workspace*;
2. Inserir duas *Entidades* no *Workspace*;
3. Atualizar as capacidades do *Workspace_Classid_99* 5 vezes durante o tempo de experimento (544 segundos);
4. Atualizar as capacidades do tráfego regular (*Workspace Regular Flow*) 3 vezes durante o tempo de experimento (544 segundos);

5. Monitorar e registrar os recursos de memória e processador do *Switch* NEA;

6. Monitorar e registrar a quantidade de *frames* processados e encaminhados pelo *Switch* NEA durante o intervalo de tempo de 544 segundos.

Na Figura 59 é ilustrado o ambiente para o experimento 4.

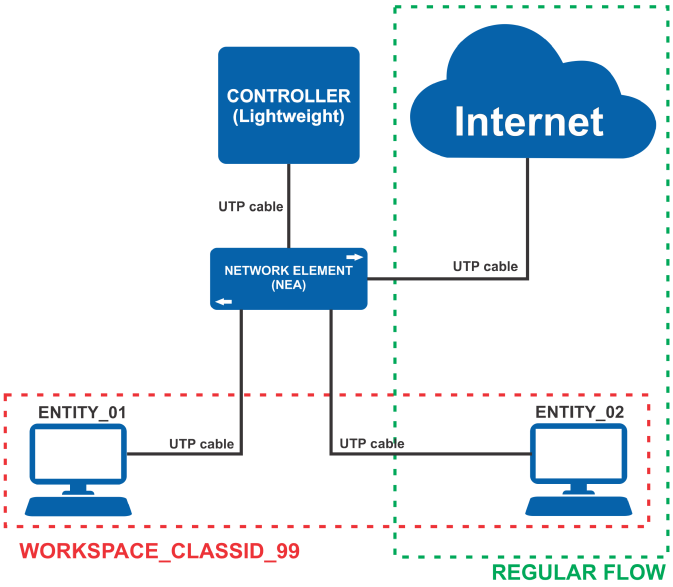


Figura 59 – Ambiente definido para o experimento 4.

Torna-se importante detalhar alguns aspectos no experimento 4 que são diferentes dos experimentos anteriores. Na aplicação definida pelo *Workspace_Classid_99*, a *Entity_02* enviou *frames* de 58 *bytes* de tamanho e paralelamente, a *Entity_01* enviou *frames* de 1253 *bytes* de tamanho. Embora as primitivas tiveram tamanhos diferentes (equivalente ao MTU do *Ethernet*), as ações de encaminhamento foram orientadas por *Workspace*. Neste contexto, a aplicação definida pelo *Workspace_Classid_99* exigiu-se uma velocidade mínima de transmissão de 250 $Kbits/s$ que é referente a taxa de transmissão definida pela aplicação. Neste contexto a aplicação foi programada para enviar 25 *frames/s* de forma ininterrupta.

Na Figura 60 é possível observar a variação da vazão (*frames/s*) em função da alteração dos parâmetros de prioridade e controle de banda aplicados no *Workspace_Classid_99* durante o intervalo de 544 segundos. Para facilitar a visualização, cada alteração nos requisitos do *Workspace_Classid_99* foi sinalizada como um cenário (*Scenario*). Como pode ser observado na Figura 60, cada cenário teve um tempo de duração diferente.

A Tabela 10 ilustra os dados de tempo de duração de cada cenário no experimento 4 juntamente com a vazão mínima exigida pela aplicação definida pelo *Workspace_Classid_99*. Os dados detalhados em relação ao quantitativo de *frames* observados durante o experimento podem ser visualizados no Apêndice B.

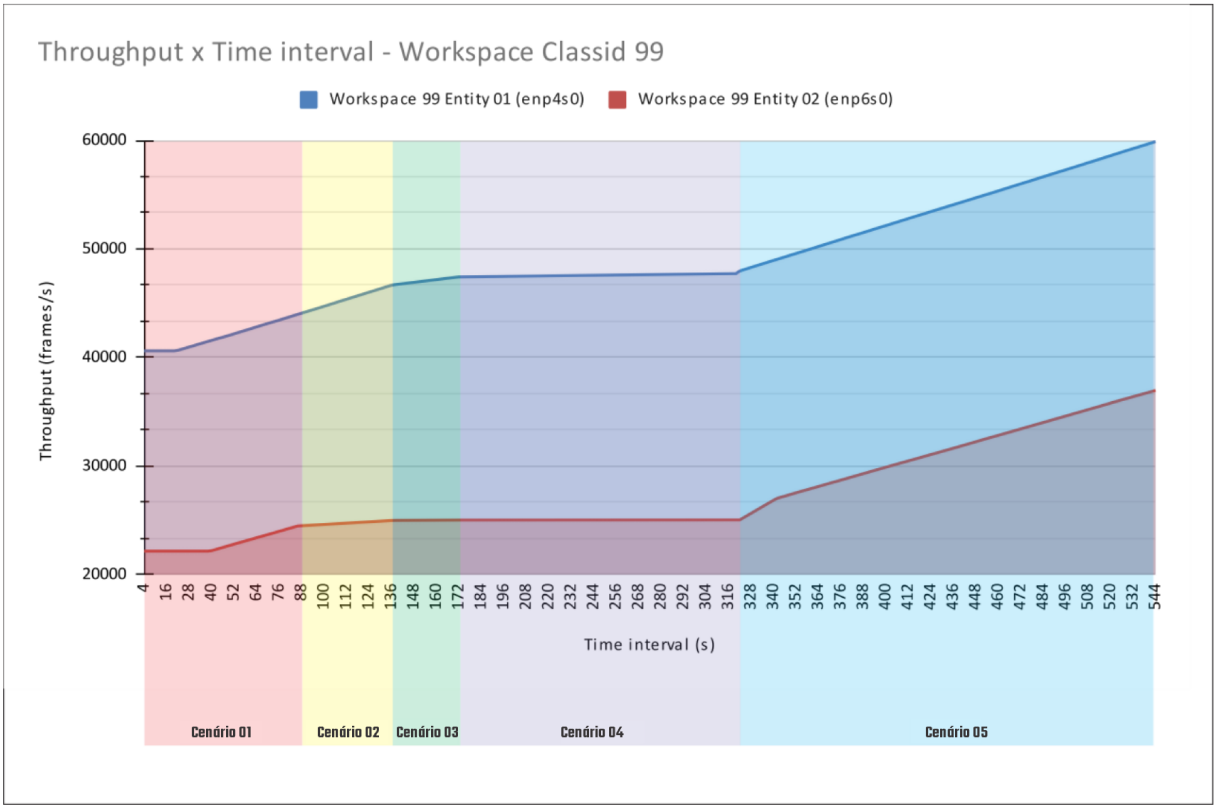


Figura 60 – Resultados relativos ao experimento 4 - *Workspace_Classid_99*

Tabela 10 – Variação de tempo dos cenários do Experimento 4 - *Workspace_Classid_99*

Identificação	**Velocidade mínima exigida**	**Duração (segundos)**
Cenário 01 / Workspace_Classid_01	*250 Kbits/s*	88 s
Cenário 02 / Workspace_Classid_01	*250 Kbits/s*	48 s
Cenário 03 / Workspace_Classid_01	*250 Kbits/s*	36 s
Cenário 04 / Workspace_Classid_01	*250 Kbits/s*	144 s
Cenário 05 / Workspace_Classid_01	*250 Kbits/s*	228 s

Na Figura 61 é possível observar a variação da vazão (*frames/s*) em função da alteração dos parâmetros de prioridade e controle de banda aplicados ao *Workspace Regular Flow* durante o intervalo de tempo de 544 segundos.

A Tabela 11 ilustra o tempo de duração de cada cenário no experimento 4 juntamente com a vazão mínima exigida pela aplicação definida pelo *Workspace Regular Flow*. Os dados detalhados em relação ao quantitativo de *frames* observados durante o experimento 4 podem ser visualizados no Apêndice B.

Apesar dos resultados referente ao experimento 4 terem sido apresentados em dois gráficos diferentes, ambos foram observados concomitantemente durante o mesmo intervalo

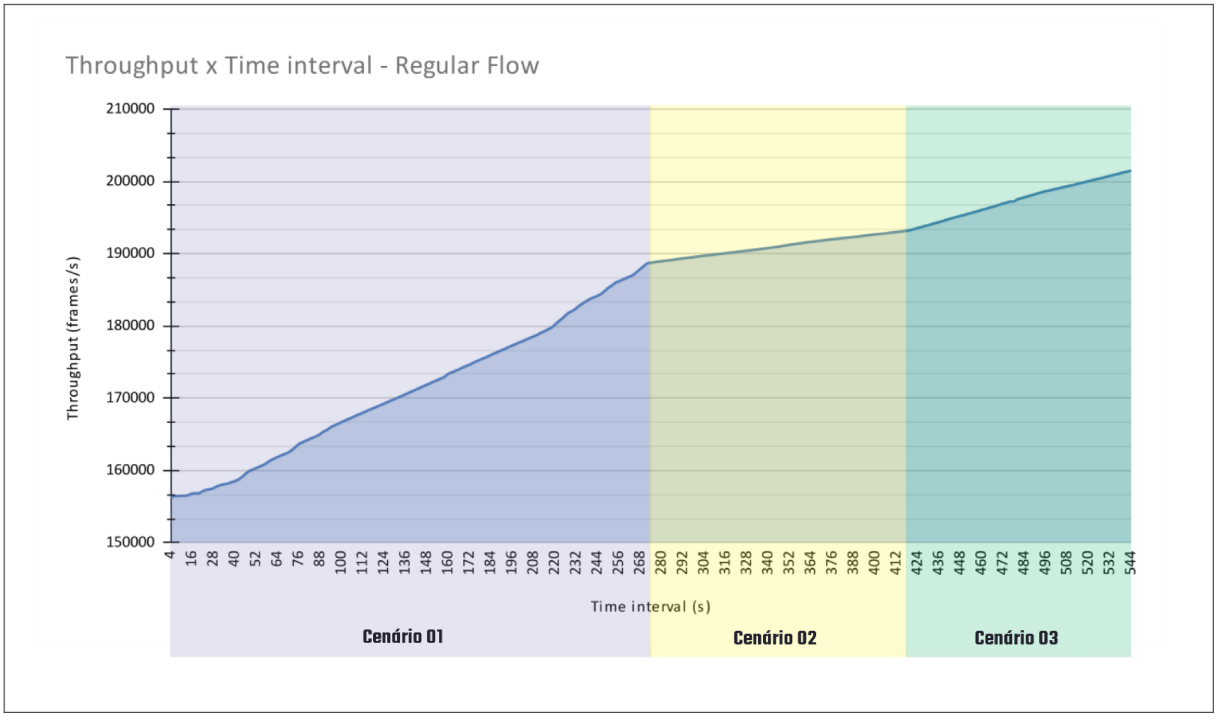


Figura 61 – Resultados relativos ao experimento 4 - *Streaming Application* (Regular Flow).

Tabela 11 – Variação de tempo dos cenários do Experimento 4 - *Workspace Regular Flow* (*Streaming Application*)

Identificação	**Velocidade mínima exigida**	**Duração (segundos)**
Cenário 01 / Regular Flow	*500 Kbits/s*	268 s
Cenário 02 / Regular Flow	*500 Kbits/s*	144 s
Cenário 03 / Regular Flow	*500 Kbits/s*	132 s

de tempo e todos os dados ilustrados nas Figuras 60 e 61 foram coletados em tempo real.

A Tabela 12 oferece uma associação clara entre a vazão e os parâmetros de prioridade e controle de banda utilizados no experimento 4.

Conforme já definido anteriormente, com base nas características das aplicações *Workspace_Classid_99* e *Workspace Regular Flow*, as velocidades mínimas exigidas pelas aplicações foram, respectivamente, $250Kbit/s$ e $500Kbit/s$.

De acordo com a Tabela 12, no contexto de análise da comunicação *Workspace*, observou que no cenário 1 os requisitos definidos (Prioridade, Controle de Banda) foram suficientes para atender o requisito de vazão das aplicações, porém, no cenário 2, 3 e 4 com requisitos de controle de banda reduzidos, a comunicação apresentou degradação em relação à vazão, prejudicando a QoS. No cenário 5, os requisitos foram redefinidos para valores acima do necessário, retomando a vazão ideal.

Tabela 12 – Variação da vazão em função dos parâmetros de prioridade e controle de banda no experimento 4

Entity/Worskpace	**Vazão (f/s)**	**Parâmetros (QoS)**
Entity_01	*26,91 frames/s*	Cenário 01
Entity_02	*39,11 frames/s*	Priority: 2, Rate: 100Mbit, Ceil: 100Mbit, Burst: 1600b.
Entity_01	*10,04 frames/s*	Cenário 02
Entity_02	*53,79 frames/s*	Priority: 7, Rate: 100Kbit, Ceil: 100Kbit, Burst: 1600b.
Entity_01	*1,0 frames/s*	Cenário 03
Entity_02	*10,4 frames/s*	Priority: 7, Rate: 10Kbit, Ceil: 10Kbit, Burst: 1600b.
Entity_01	*0,10 frames/s*	Cenário 04
Entity_02	*2,10 frames/s*	Priority: 7, Rate: 1Kbit, Ceil: 1Kbit, Burst: 1600b.
Entity_01	*53,73 frames/s*	Cenário 05
Entity_02	*56,70 frames/s*	Priority: 2, Rate: 1Mbit, Ceil: 1Mbit, Burst: 1600b.
Entity/Streaming	**Vazão (f/s)**	**Parâmetros (QoS)**
Entity_02	*119,86 frames/s*	Cenário 01 Priority: 6, Rate: 100Mbit, Ceil: 100Mbit, Burst: 1600b.
Entity_02	*30,84 frames/s*	Cenário 02 Priority: 7, Rate: 100Kbit, Ceil: 100Kbit, Burst: 1600b.
Entity_02	*66,41 frames/s*	Cenário 03 Priority: 4, Rate: 300Kbit, Ceil: 300Kbit, Burst: 1600b.

De acordo com a Tabela 12, no contexto de análise da comunicação *Streaming*, observou-se que o controle de banda abaixo de $500Kbit/s$, principalmente em torno de $100Kbit/s$ apresentou degradação da QoS, ocasionando o travamento da aplicação. Contudo, valores acima de $100Mbit/s$ garantiu a QoS da aplicação.

Através experimento 4 foi possível também identificar a segregação de tráfego no elemento de rede. Na Figura 62 é ilustrado o quantitativo de *frames* observado por porta física no elemento de rede durante o experimento 4.

O tráfego da aplicação definida pelo *Workspace_Classid_99* foi instituído nas portas *Enp4s0* e *Enp6s0* do *Switch* NEA, onde as *Entidades* estão conectadas fisicamente. Além do *Workspace_Classid_99* presente na porta física *Enp6s0* esteve presente outro tráfego referente a aplicação definida pelo *Workspace Regular Flow - Streaming*. Neste experimento a porta *Enp3s0* representou a conexão com a Internet, não existindo nenhum *Workspace* vinculado.

De acordo com a Figura 62 foi observado que o tráfego orientado pelo *Workspace_Classid_99* na porta *Enp3s0* é nulo, isto se deu pelo fato que não foi instituído nenhum *Workspace* nesta porta, sendo assim, qualquer *frame* endereçado para a porta *Enp3s0* será tratado com a política *Best Effort*. A porta física *Enp3s0* foi conectada diretamente ao roteador da subrede.

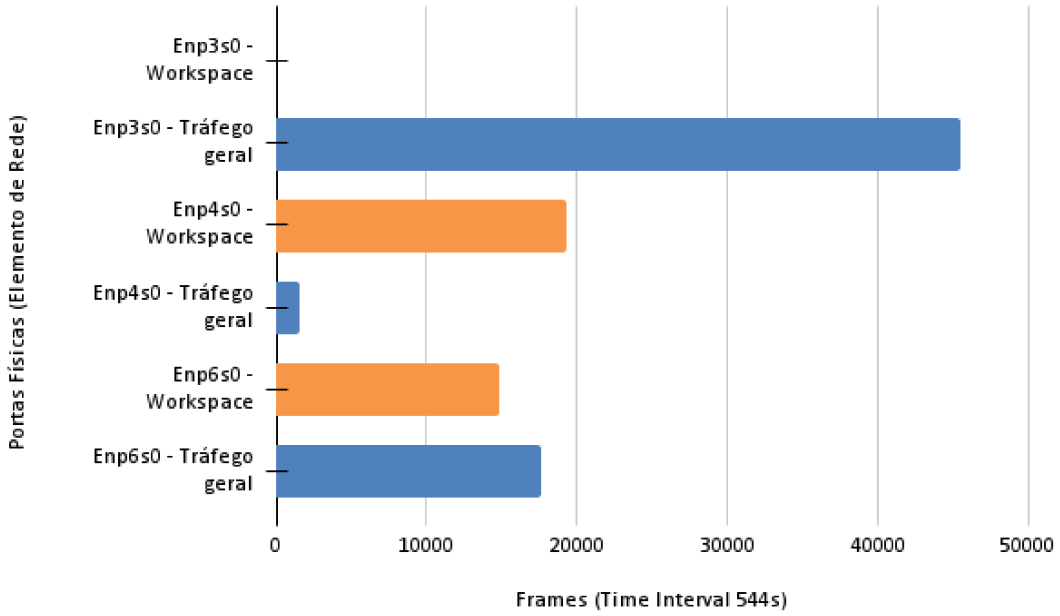


Figura 62 – Quantitativo de *frames* registrados nas portas físicas do elemento de rede

Com relação a porta física *Enp4s0*, observou-se um pequeno volume relacionado ao tráfego geral, este volume é justificado devido à outras aplicações instituídas no mesmo *host* da *Entidade* em questão e também em relação à primitivas de controle da rede, como por exemplo pacotes do tipo *Address Resolution Protocol (ARP)*.

Na porta *Enp6s0*, foi instânciado uma aplicação definida pelo *Workspace_Classid_99*, e uma outra aplicação definida pelo *Workspace Regular Flow - Streaming*, sob a óptica do *Switch* NEA, tratou-se de tráfegos distintos e apesar de terem sido instânciados na mesma porta física, foram tratados separadamente, obedecendo os parâmetros definidos e atualizados durante o experimento.

De foram geral, foi possível observar a capacidade do *Switch* NEA em atuar na segregação de tráfego através de uma mesma porta física.

Conforme já mencionando, o dados mensurados nos cenários 1, 2 e 3, podem ser visualizados em detalhes no apêndice A. Já os dados mensurados para o cenário 4 podem ser visualizados no ApêndiceB.

7.2.5 Consumo de Memória e Processamento

Durante os experimentos 1, 2, 3 e 4 foi observado o consumo de memória e processador do *Switch* NEA. Os valores foram capturados durante os experimentos. Contudo, os experimentos 1, 2 e 3 não apresentaram notáveis variações em termos de utilização de processador e consumo de memória devido às poucas atualizações nos parâmetros de prioridade e controle de banda.

Todavia, no experimento 4 houve diversas atualizações, sendo possível observar variações no consumo para as tarefas de atualização dos parâmetros. Na Figura 63 é apresentados os valores coletados durante o experimento 4.

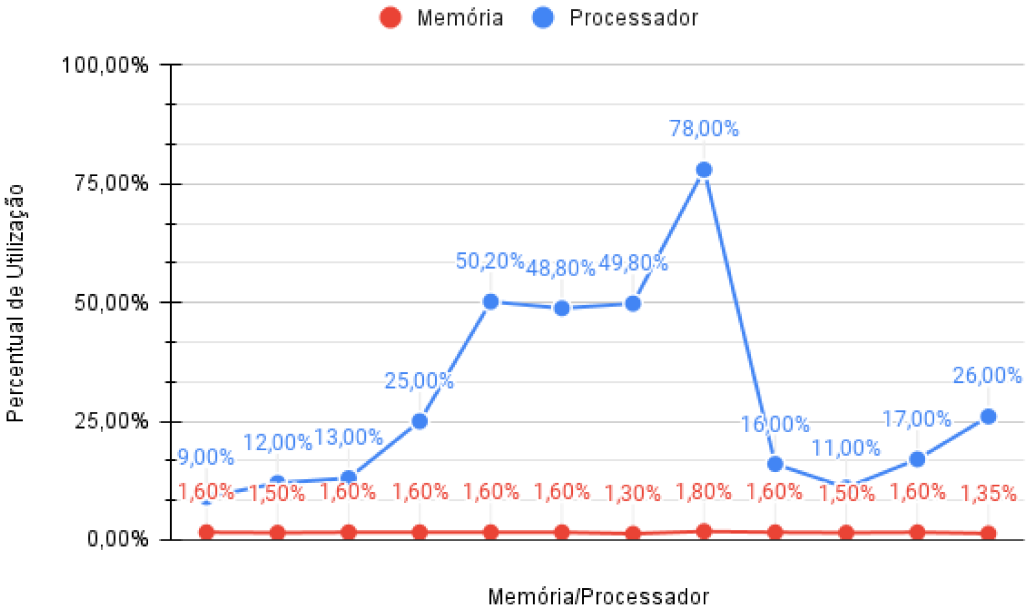


Figura 63 – Consumo de Memória e Processador do protótipo para o experimento 4.

De acordo com a Figura 63, foi observado que os picos são atingidos no momento que o *Switch* NEA necessita (re)configurar os parâmetros de controle da comunicação (*Workspace*). Observou-se que a quantidade de *Workspace* tem notável relação com esse consumo, devido a propagação das atualizações em todas as portas físicas onde existe correspondência do *Workspace* em questão. Contudo, nota-se que o consumo de memória não apresentou grandes variações em comparação com o consumo do processador.

Entende-se que o consumo de memória foi baixo devido a aplicação não necessitar constantemente de alocação de memória, todos os recursos que são alocados em tempo de execução são liberados após o uso. O processamento apresentou picos de consumo em momentos onde houve a necessidade de leitura nas estrutura de dados e atuação através de submódulos do Kernel, como por exemplo, a atualização dos parâmetros de controle de tráfego nas ações de escalonamento através do TC.

7.3 Análise Geral dos Testes e Resultados

Embora todos os experimentos terem sido realizados sob a óptica da ETArch, a NEA foi projetada para suportar qualquer arquitetura SDN. A escolha da ETArch se deu em função da relação da arquitetura com os objetivos desta pesquisa, principalmente aqueles

direcionados à melhoria da QoS. Por se tratar de uma solução totalmente centrada em *Software*, a implementação de novas funcionalidades e protocolos é menos complexa, além de ser uma solução independente de *Hardware*.

Através dos resultados apresentados, observou-se a capacidade do *Switch* NEA em instituir e gerenciar domínios de comunicações ETArch (*Workspaces*). Os requisitos de aplicações foram materializados na estrutura de encaminhamento do *Switch* através de variáveis de controle de banda, prioridade de tráfego e controle de *burst*. Com o *Switch* NEA foi possível instituir um escalonador de *frames* único em cada porta física do *Switch*, ofertando maior autonomia no gerenciamento das ações de encaminhamento, permitindo que cada porta física do equipamento possa atuar com diferentes fluxos em diferentes cenários (requisitos de aplicação), permitindo atualizações *on-the-fly* nos parâmetros de controle (prioridade, controle de banda e *burst*) em toda a estrutura do *Switch*.

A NEA se difere de outras soluções para plano de dados em SDN justamente no fato de ser uma arquitetura que considerou aspectos de QoS desde a fase de projeto. O *Switch* NEA foi desenvolvido sobre uma abordagem totalmente centrada em *Software* que buscou atuar o mais próximo possível do nível *Hardware* (MAC), preservando a programabilidade e flexibilidade para a solução. Diferentemente de outras soluções, onde toda a lógica de encaminhamento acontece em *Software* e a materialização das ações de encaminhamento fica restrita à capacidade de equipamentos legados (*Hardware*).

O *Switch* NEA opera, imediatamente, sobre o *Socket Buffer (SKB)* da *NIC*, atuando diretamente no MAC em cada interface do *Switch*. As plataformas legadas, as quais as soluções em SDN se apoiam, não permitem que a lógica de encaminhamento seja exposta até o nível do MAC devido ao uso de chips (ASICs), os quais não são programáveis, apenas configuravéis e com o MAC único, mesmo em algumas propostas que fazem uso de FPGA, a reconfiguração dinâmica não é explorada, tornando o FPGA um *Hardware* configurável e não programável.

Em suma, os experimentos realizados permitiram validar os objetivos definidos neste trabalho. O *Switch* NEA foi capaz de interpretar primitivas de controle do tipo *ETSCP* advindas do controlador e atuar sobre a estrutura de encaminhamento. Foi avaliado a capacidade do *Switch* NEA em realizar sucessivas alterações *on-the-fly* para manter a QoS das aplicações em questão, tendo os resultados satisfatórios para os experimentos.

Durante os experimentos foram observados perdas de *frames*, porém, não foi instituído um método de avaliação para perda de dados. De acordo com Burgess (2004), *frames* podem ser perdidos ou descartados por diversas razões, incluindo erros, atrasos, dentre outras motivações. Para avaliar o que seria uma taxa de erro do *Switch* NEA, será necessário instituir uma metodologia de testes voltada para verificação de perda de dados, ou seja, avaliar os *frames* que estão sendo descartados pelo *Switch* em decorrência do controle de banda, atrasos e erros.De forma geral, o *Swtich* quando encontra um erro, por exemplo, sinalizado pelo *Cyclic Redundancy Check (CRC)*, todo o *frame* é descartado,

tornando complexo definir uma métrica do tipo *Bit Error Rate (BER)*.

Capítulo 8

Conclusão

Este trabalho apresentou o desenvolvimento de um protótipo de *switch* SDN com suporte a MAC definido pela aplicação. Isto é uma inovação por dois motivos: i) as tecnologias de *switches* atuais definem um mesmo MAC para todas as portas, isto é, o MAC é definido para o equipamento; e ii) o *Switch NEA* permite que o MAC seja definido para cada instância de entidades comunicantes. Observe que (ii) define que uma mesma porta suporte 1 ou mais MACs, concomitantemente.

Foi observado que nas SDNs o plano de dados não é tão explorado quanto o plano de controle e, naturalmente, as soluções assumem que existe conectividade no nível de Enlace, negligenciando qualquer possibilidade de atuação do plano de dados na melhoria da *Quality of Service (QoS)* e *Quality of Experience (QoE)*. Fato é que, grande parte das soluções centradas em software fazem uso da tecnologia *OpenFlow* para resolver o plano de dados. Contudo, *switches OpenFlow* carecem de inúmeras melhorias para conseguir atender os requisitos das atuais aplicações, como por exemplo, suporte a novos protocolos. Soluções centradas em hardware aparecem com um complemento para as soluções em software, ora propondo melhorias através da reutilização de blocos de hardware, ora fornecendo suporte para as funções de alto nível.

Foi observado que grande parte das soluções exploradas no plano de dados é direcionada para promover programabilidade e flexibilidade no que tange a ações de encaminhamento e na adição de novas funcionalidades. Essas soluções não foram projetadas para atuar na melhoria da QoS no plano de dados e apresentam dificuldades em materializar essas necessidades no plano de dados.

O *Switch* NEA, quando comparado a outras soluções, possui a capacidade de expor, através de uma fina granularidade, a lógica referente às políticas de encaminhamento de baixo nível ao plano de controle por meio de um módulo orquestrador. O *Switch* NEA permite (re)programação de regras de encaminhamento orientadas pela aplicação, ofertando melhoria na QoS.

O protótipo desenvolvido é uma solução *Linux Based*, totalmente centrada em *software*. Atuar neste nível significa manipular funções de baixo nível do *Kernel* com impacto

direto no Media Access Control (MAC), o módulo orquestrador (NEASwitchD.py) permite, através de uma linguagem de alto nível, flexibilidade na implementação de novas funcionalidades.

Para validação do *Switch* NEA, foi escolhida uma arquitetura SDN que possua coerência com os propósitos deste trabalho. A escolha da ETArch se deu pelo suporte nativo a requisitos de aplicação, sendo uma arquitetura justificável para tratar QoS e QoE.

Os testes foram divididos em dois momentos, primeiramente foi avaliada a capacidade do protótipo em receber primitivas de controle (ETSCP) e conseguir criar, atualizar ou remover *Workspaces* quando necessário. No segundo momento foi verificada a capacidade do protótipo em instituir regras de encaminhamento baseadas em requisitos de *Workspaces* e também, manipular os parâmetros de controle de tráfego com impacto direto na QoS e QoE.

Os resultados obtidos mostram a capacidade do protótipo em atuar na melhoria de QoS e QoE, através do controle acurado das regras de encaminhamento no plano de dados. O protótipo apresentou a habilidade de manipular diversos fluxos de dados pela mesma porta física, atribuindo diferentes parâmetros (prioridade e largura de banda) para diferentes fluxos. Ressalta-se também a possibilidade de adição de novas funcionalidades e suporte a novos protocolos através da (re)programação do submódulo *NEA-Parser*. Embora os experimentos e resultados tenham sido realizado sob a ETArch, reforça-se que a arquitetura NEA foi projetada para suportar diversas arquiteturas SDNs inclusive, suportar o próprio o OpenFlow.

Algumas limitações foram encontradas ao longo do desenvolvimento deste trabalho e merecem atenção, a saber (i) a quantidade de interfaces físicas disponíveis foram suficientes para validar as metas desta pesquisa, porém, em um ambiente de maior demanda de conexão, um hardware computacional doméstico pode não ser suficiente; (ii) com relação aos experimentos, quando o protótipo foi submetido a um tráfego intenso (várias aplicações *Streaming* e *Workspaces*), em alguns momentos foi possível observar perda de dados (*frames*), nos experimentos isso foi corrigido aumentando a prioridade do tráfego em questão, porém, conforme explanado no Capítulo 7, torna-se necessário instituir uma metodologia de testes para avaliar as perdas; e (iii) para validação dos testes no ambiente ETArch, todas as interfaces de rede foram configuradas no modo *promiscuo* devido ao tipo de endereçamento praticado na arquitetura.

8.1 Principais Contribuições

De modo específico, pode-se afirmar que as principais contribuições deste trabalho são:

1. Protótipo: Especificação e desenvolvimento de um protótipo de elemento de rede (*Switch*) baseado na arquitetura NEA para redes SDN com MAC orientado pela aplicação.

2. Produção Bibliográfica: Capítulo de livro, que engloba:

   - ❏ Arquitetura NEA: motivação no desenvolvimento de uma nova arquitetura de elemento de rede com suporte a MAC definido pela aplicação para redes SDNs.
   - ❏ Especificação e prototipação de um *Switch* para redes SDN com MAC definido pela aplicação com estudo de caso aplicado à ETArch.

Contudo, existem outras produções bibliográficas onde fragmentos desta tese foram aplicados. A análise de diversos trabalhos correlatos, resultando no conjunto de requisitos da arquitetura NEA pode vir a auxiliar futuros trabalhos na compreensão das necessidades do plano de dados frente a requisitos de *Quality of Service (QoS)*.

A fundamentação teórica deste trabalho se posiciona como uma contribuição para estudos em *Software Defined Network (SDN)* com foco no plano de dados. Foi apresentado todo o aparato conceitual em relação a programabilidade e flexibilidade no plano de dados juntamente com as principais abordagens utilizadas atualmente. De modo geral, isto contribuirá para redução do espaço exploratório de novas pesquisas focadas no plano de dados.

O estudo de caso proporcionou a realização de uma análise detalhada da arquitetura *ETArch* com foco no plano de dados. Foi possível (re)avaliar e refinar os requisitos da arquitetura em relação ao elemento de rede, alcançando melhorias no protocolo *ETSCP*. Estas contribuições auxiliarão novas pesquisas do grupo MEHAR.

8.2 Trabalhos Futuros

Visto que a arquitetura NEA foi especificada e implementada em sua totalidade, juntamente com um protótipo de elemento de rede, entende-se que novos trabalhos podem ser conduzidos a partir deste.

A proteção intelectual por meio de uma patente está sendo analisada e, oportunamente, será aplicada junto ao INPI (Instituto Nacional de Proteção Intelectual).

Como contribuições de curto prazo, a realização de estudos comparativos com outras tecnologias de mesmo propósito, como por exemplo o *Switch* OpenFlow, poderá conduzir um novo conjunto de testes focado em métricas de desempenho, visto que neste trabalho, performance foi definido como requisito não funcional.

Ainda como meta de curto prazo, adicionar suporte na NEA para contenção de ataques Distributed Denial of Service (DDoS). A manipulação de *frames* na NEA acontece de forma individual e sequencial, tendo início antes do processo de *pré-routing* na *Bridge*, possibilitando instituir um algoritmo de análise de tráfego para atuar diretamente no *Netfilterqueue*, antes mesmo das primitivas serem roteadas pelo *Kernel*.

Além das metas apresentadas, para dar continuidade a esta pesquisa, sugere-se para desenvolvimento futuro:

- ❏ Implementar os requisitos avançados que não foram definidos no escopo desta tese;
- ❏ Replicar os experimentos para outras arquiteturas SDNs e outros controladores de rede;
- ❏ Replicar os experimentos com situações reais multimídias, tais como VoIP e críticas, objetivando avaliar a capacidade do *Switch* NEA em garantir QoS de aplicações;
- ❏ Desenvolver uma versão *Lightweight* do *Kernel* do Linux para suportar a arquitetura NEA visando consumir menos recursos de hardware;
- ❏ Adicionar suporte para o P4;
- ❏ Especificar e confeccionar uma plataforma de hardware baseada em processador escalável com FPGA integrado para suportar a arquitetura NEA;

8.3 Contribuições em Produção Bibliográfica

Como já mencionado, este trabalho visou a especificação de uma arquitetura de elemento de rede com foco na melhoria de *QoS* para redes SDN juntamente com um protótipo de *Switch*. Ao longo do desenvolvimento desta tese, diversos assuntos foram estudados, em particular as soluções atuantes no plano de dados em SDNs e a arquitetura ETArch, em especial as comunicações baseadas em *Workspaces*.

Os resultados destes estudos podem ser verificados em trabalhos submetidos e publicados em conferências e capítulos de livros, concluindo-se que o tema abordado neste trabalho e seus diversos aspectos têm tido boa aceitação pela comunidade acadêmica. Nesta seção são listadas as contribuições em produção bibliográfica resultantes deste trabalho até o momento.

8.3.1 Artigos e Capítulos de Livros Pulicados

Artigos submetidos e publicados em conferências.

- ❏ OLIVEIRA, R. D. ; FREITAS, M. S. ; MOLINOS, D. N. ; ROSA, P. F. ; MESQUITA, D. G. . ETSCP: Flexible SDN data plane configuration based on bootstrapping of in-band control channels. In: IFIP/IEEE International Symposium on Integrated Network Management, 2021, Bordeaux. 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2021.
- ❏ MOLINOS, D. N.; OLIVEIRA, R. D. ; FREITAS, M. S. ; ALMEIDA, M. B. ; ROSA, P. F. ; SILVA, F. O. . NEA: An SDN Switch Architecture Suitable for Application-Oriented MAC. In: International Conference on Advanced Information

Networking and Applications AINA 2021, 2021, Toronto. Networks and Systems, 2021. v. 227.

- SILVA FREITAS, MARCELO ; OLIVEIRA, ROMERSON ; MOLINOS, DIEGO ; MELO, JULIANO ; ROSA, P. F. ; DE OLIVEIRA SILVA, FLAVIO . ConForm: In-band Control Plane Formation Protocol to SDN-Based Networks. In: 2020 International Conference on Information Networking (ICOIN), 2020, Barcelona. 2020 International Conference on Information Networking (ICOIN), 2020. p. 574.

- FREITAS, MARCELO ; OLIVEIRA, ROMERSON ; MOLINOS, DIEGO ; SILVA, FLAVIO; ROSA, P. F. Workspace-based Virtual Networks: A Clean Slate Approach to Slicing Cloud Networks. In: 9th International Conference on Cloud Computing and Services Science, 2019, Heraklion. Proceedings of the 9th International Conference on Cloud Computing and Services Science, 2019. p. 464.

8.3.2 Artigo Submetido e Aceito

Artigos submetidos e aceitos em conferências, porém, ainda não publicados.

- MOLINOS, D. N. ; OLIVEIRA, R. D. ; FREITAS, M. S. ; N. V. DE SOUZA NETO ; ALMEIDA, M. B. ; SILVA, F. O ; ROSA, P. F. . Designing and Prototyping of SDN Switch for Application-Driven Approach. In: International Conference on Advanced Information Networking and Applications (AINA 2022), 2022, Sydney.

Referências

ALBERTI, A. M. et al. Naming and name resolution in the future internet: Introducing the novagenesis approach. **Future Generation Computer Systems**, Elsevier, v. 67, p. 163–179, 2017. Disponível em: <https://doi.org/10.1016/j.future.2016.07.015>.

ANAND, A. et al. XIA: An architecture for an evolvable and trustworthy internet. In: **Proceedings of the tenth ACM Workshop on Hot Topics in Networks HotNets-X**. Cambridge, MA. USA.: [s.n.], 2011. Disponível em: <https://doi.org/10.1145/2070562.2070564>.

ANDERSON, T. et al. The nebula future internet architecture. In: _____. **The Future Internet: Future Internet Assembly 2013: Validated Results and New Horizons**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 16–26. ISBN 978-3-642-38082-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-38082-2_2>.

ANTICHI, G. et al. From 1g to 10g: code reuse in action. In: **Proceedings of the first edition workshop on High performance and programmable networking**. [s.n.], 2013. p. 31–38. Disponível em: <https://doi.org/10.1145/2465839.2465844>.

ASTERFUISON. **What you should know about P4 programming language P4 programmable switch**. 2022. Available at: <https://cloudswit.ch/blogs/what-you-should-know-about-p4-programming-language-p4-switch/>. Access on 28/03/2022.

BEZAHAF, M. et al. Internet evolution: Critical issues. **IEEE Internet Computing**, IEEE, v. 24, n. 4, p. 5–14, 2020. Disponível em: <https://doi.org/10.1109/MIC.2020.3001519>.

BIFULCO, R.; RÉTVÁRI, G. A survey on the programmable data plane: Abstractions, architectures, and open problems. In: IEEE. **2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)**. 2018. p. 1–7. Disponível em: <https://doi.org/10.1109/HPSR.2018.8850761>.

BOERO, L. et al. Beaqos: Load balancing and deadline management of queues in an openflow sdn switch. **Computer Networks**, Elsevier, v. 106, p. 161–170, 2016. Disponível em: <https://doi.org/10.1016/j.comnet.2016.06.025>.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 3, p. 87–95, 2014. Disponível em: <http://doi.acm.org/10.1145/2656877.2656890>.

BREBNER, G. Programmable hardware for software defined networks. In: **2015 European Conference on Optical Communication (ECOC)**. [s.n.], 2015. p. 1–3. Disponível em: <https://doi.org/10.1109/ECOC.2015.7341956>.

BROWN, M. A. Traffic control howto. **Guide to IP Layer Network**, v. 49, p. 36, 2006.

BURGESS, N. Rfc 2544 testing of ethernet services in telecom networks. **Canada: Agilent Technologies**, 2004.

CAMPBELL, A. T. et al. A survey of programmable networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 29, n. 2, p. 7–23, apr 1999. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/505733.505735>.

CHANNEGOWDA, M.; NEJABATI, R.; SIMEONIDOU, D. Software-defined optical networks technology and infrastructure: Enabling software-defined optical network operations [invited]. **IEEE/OSA Journal of Optical Communications and Networking**, v. 5, n. 10, p. A274–A282, Oct 2013. ISSN 1943-0620. Disponível em: <https://doi.org/10.1364/JOCN.5.00A274>.

CISCO. **CAM (Content Addressable Memory) VS TCAM (Ternary Content Addressable Memory)**. 2017. URL: <https://goo.gl/orPSen>. Acesso em: 07.07.2020.

CLARK, D. et al. **New arch: Future generation internet architecture**. [S.l.], 2004.

_____. **RFC1287: Towards the Future Internet Architecture**. RFC Editor, 1991. URL: <https://dl.acm.org/doi/pdf/10.17487/RFC1287>. Disponível em: <https://doi.org/10.17487/rfc1287>.

DAY, J.; MATTA, I.; MATTAR, K. Networking is ipc: A guiding principle to a better internet. In: **Proceedings of the 2008 ACM CoNEXT Conference**. New York, NY, USA: ACM, 2008. (CoNEXT '08), p. 67:1–67:6. ISBN 978-1-60558-210-8. Disponível em: <http://doi.acm.org/10.1145/1544012.1544079>.

DECUSATIS, C. et al. Dynamic, software-defined service provider network infrastructure and cloud drivers for sdn adoption. In: IEEE. **2013 IEEE International Conference on Communications Workshops (ICC)**. 2013. p. 235–239. Disponível em: <https://doi.org/10.1109/ICCW.2013.6649235>.

FELDMAN, S. Rocker: switchdev prototyping vehicle. **Proceedings of Netdev 0.1**, 2015.

FELDMANN, A. Internet clean-slate design: what and why? **ACM SIGCOMM Computer Communication Review**, ACM New York, NY, USA, v. 37, n. 3, p. 59–64, 2007. Disponível em: <https://doi.org/10.1145/1273445.1273453>.

FERNANDES, E. L. et al. The road to bofuss: The basic openflow userspace software switch. **Journal of Network and Computer Applications**, Elsevier, v. 165, p. 102685, 2020. Disponível em: <https://doi.org/10.1016/j.jnca.2020.102685>.

FERRO, G. **What's Happening Inside an Ethernet Switch ? (Or Network Switches for Virtualization People)**. 2013. Available at: <https://etherealmind.com/whats-happening-inside-an-ethernet-switch-or-network-switches-for-virtualization-people/>. Access on 10/08/2020.

FIRESTONE, D. {VFP}: A virtual switch platform for host {SDN} in the public cloud. In: **14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)**. [S.l.: s.n.], 2017. p. 315–328.

FOUNDATION, O. N. **OpenFlow Switch Specification**. 2015. Available at: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>. Access on 15/06/2020.

_____. **ONF**. 2017. URL: <https://www.opennetworking.org/>. Acesso em: 07.07.2017.

GIBB, G. et al. Netfpga—an open platform for teaching how to build gigabit-rate network switches and routers. **IEEE Transactions on Education**, IEEE, v. 51, n. 3, p. 364–369, 2008. Disponível em: <https://doi.org/10.1109/TE.2008.919664>.

GOEL, S.; WILLIAMS, K.; DINCELLI, E. Got phished? internet security and human vulnerability. **Journal of the Association for Information Systems**, v. 18, n. 1, p. 2, 2017. Disponível em: <https://doi.org/10.17705/1jais.00447>.

GONÇALVES, M. A. et al. Etarch: projeto e desenvolvimento de uma arquitetura para o modelo de título com foco na agregação de tráfego multicast. Universidade Federal de Uberlândia, 2014.

HAKIRI, A. et al. Software-defined networking: Challenges and research opportunities for future internet. **Computer Networks**, Elsevier, v. 75, p. 453–471, 2014. Disponível em: <https://doi.org/10.1016/j.comnet.2014.10.015>.

HALPERN, J. M.; SALIM, J. H. **"Forwarding and control element separation (ForCES) forwarding element model," RFC 5812**. 2010. Available at: <https://www.rfc-editor.org/rfc/rfc5812.txt>. Access on 10/04/2020. Disponível em: <https://doi.org/10.17487/rfc5812>.

HENNESSY, J. L.; PATTERSON, D. A. **Computer architecture: a quantitative approach**. [S.l.]: Elsevier, 2011.

HONEYWELL. **Technical Support - Locally Administered MAC addresses**. 2017. Available at: <https://honeywellaidc.force.com/supportppr/s/article/Locally-Administered-MAC-addresses>. Access on 10/04/2021.

HU, C. et al. Design of all programable innovation platform for software defined networking. In: **Open Networking Summit 2014 ({ONS} 2014)**. [S.l.: s.n.], 2014.

HUANG, M.; LIANG, W. Incremental sdn-enabled switch deployment for hybrid software-defined networks. In: IEEE. **2017 26th International Conference on Computer Communication and Networks (ICCCN)**. 2017. p. 1–6. Disponível em: <https://doi.org/10.1109/ICCCN.2017.8038498>.

IEEE. Ieee standard for local and metropolitan area networks: Overview and architecture. **IEEE Standard 802-2001**, Feb 2001.

IEEE/SA. **Standard Group MAC Addresses: A Tutorial Guide**. 2018. Available at: <https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/macgrp.pdf>. Access on 10/04/2021.

IETF. **Software-Defined Networking (SDN): Layers and Architecture Terminologyl**. [S.l.]: IETF, 2015. URL <https://tools.ietf.org/html/rfc7426#ref-OF08>. (Request for Comments, 7426).

IRFAN, M. et al. Reconfigurable content-addressable memory (cam) on fpgas: A tutorial and survey. **Future Generation Computer Systems**, Elsevier, v. 128, p. 451–465, 2022. Disponível em: <https://doi.org/10.1016/j.future.2021.09.037>.

ISO. **Open systems interconnection - OSI**. [S.l.]: International Organization for Standardization, 2017. URL: <https://www.iso.org/ics/35.100/x/>. Acesso em: 07.07.2020.

JÚNIOR, C. et al. Máquinas de estados finitos aplicados a jogos eletrônicos. **Revista Eletrônica da Faculdade Metodista Granbery**, 2013.

KALJIC, E. et al. A survey on data plane flexibility and programmability in software-defined networking. **IEEE Access**, IEEE, v. 7, p. 47804–47840, 2019. Disponível em: <https://doi.org/10.1109/ACCESS.2019.2910140>.

KALYAEV, A.; MELNIK, E. Fpga-based approach for organization of sdn switch. In: **2015 9th International Conference on Application of Information and Communication Technologies (AICT)**. [s.n.], 2015. p. 363–366. Disponível em: <https://doi.org/10.1109/ICAICT.2015.7338580>.

KHOSRAVI, H. M.; ANDERSON, T. A. **Requirements for separation of IP control and forwarding, RFC 3654**. 2003. Available at: <https://www.rfc-editor.org/rfc/rfc3654.txt>. Access on 10/04/2020. Disponível em: <https://doi.org/10.17487/rfc3654>.

KUROSE, J. Computer networking: A top-down approach /james f. kurose, keith w. ross. 2013.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a Internet: uma abordagem top-down**. Pearson Addison Wesley, 2006. ISBN 9788588639188. Disponível em: <http://books.google.com.br/books?id=i5WwAAAACAAJ>.

LAKATOS, E. M.; MARCONI, M. A. **Fundamentos de Metodologia Científica**. 5. ed. São Paulo, SP, Brasil: Atlas, 2003. ISBN 8522433976.

LANGEMAK, J. **What's the effect of network switch buffer in a data center?** 2015. Available at: <https://searchitoperations.techtarget.com/answer/Whats-the-effect-of-network-switch-buffer-in-a-data-center#:~:text=Sufficient%20network%20buffers%20prevent%20the,ports%20during%20times%20of%20congestion.> Access on 10/08/2021.

LIU, H. **The netfilter.org "nftables"project**. 2017. Available at: <https://netfilter.org/projects/nftables/>. Access on 10/08/2020.

MARTIN, D.; VöLKER, L.; ZITTERBART, M. A flexible framework for future internet design, assessment, and operation. **Computer Networks: The International Journal of Computer and Telecommunications Networking.**, Elsevier North-Holland, Inc., New York, NY, USA, v. 55, n. 4, p. 910–918, mar. 2011. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2010.12.015>.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, ACM, v. 38, n. 2, p. 69–74, 2008.

MEALY, G. H. A method for synthesizing sequential circuits. **The Bell System Technical Journal**, Nokia Bell Labs, v. 34, n. 5, p. 1045–1079, 1955. Disponível em: <https://doi.org/10.1002/j.1538-7305.1955.tb03788.x>.

MEHRA, M.; MAURYA, S.; TIWARI, N. K. Network load balancing in software defined network: A survey. **International Journal of Applied Engineering Research**, v. 14, n. 2, p. 245–253, 2019.

MENDIOLA, A. et al. A survey on the contributions of software-defined networking to traffic engineering. **IEEE Communications Surveys & Tutorials**, IEEE, v. 19, n. 2, p. 918–953, 2016. Disponível em: <https://doi.org/10.1109/COMST.2016.2633579>.

MICHEL, O. et al. The programmable data plane: Abstractions, architectures, algorithms, and applications. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 54, n. 4, p. 1–36, 2021. Disponível em: <https://doi.org/10.1145/3447868>.

MOORE, E. F. et al. Gedanken-experiments on sequential machines. **Automata studies**, Princeton, v. 34, p. 129–153, 1956. Disponível em: <https://doi.org/10.1515/9781400882618-006>.

MOREIRA, M. D. et al. Internet do futuro: Um novo horizonte. **Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC**, v. 2009, p. 1–59, 2009.

MÜLLER, P.; REUTHER, B. Future internet architecture-a service oriented approach. **IT-Information Technology**, v. 50, n. 6, p. 383–389, 2009. Disponível em: <https://doi.org/10.1524/itit.2008.0510>.

MUSHTAQ, M. S.; SHAHID, A.; FOWLER, S. Qos-aware lte downlink scheduler for voip with power saving. In: IEEE. **2012 IEEE 15th International Conference on Computational Science and Engineering**. 2012. p. 243–250. Disponível em: <https://doi.org/10.1109/ICCSE.2012.41>.

NETFILTER.ORG. **VLAN filter support on bridge**. 2021. Available at: <https://developers.redhat.com/blog/2017/09/14/vlan-filter-support-on-bridge/>. Access on 10/08/2021.

OLIVEIRA, R. D. **Um novo método de acesso ao meio baseado em Workspaces para suporte a arquiteturas de Internet do Futuro implementado em FPGA.** Tese (Doutorado) — Universidade Federal de Uberlândia„ 2019. Access on 20/06/2020.

OLIVEIRA, R. D. et al. Etscp: Flexible sdn data plane configuration based on bootstrapping of in-band control channels. In: IEEE. **2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.], 2021. p. 711–715.

ONF. **OpenFlow-enabled Transport SDN**. [S.l.]: Open Networking Foundation, 2014. URL <https://goo.gl/KzwYWV>. (ONF Solution Brief).

_____. **Protocol Independent Forwarding**. 2017. URL: <http://opensourcesdn.org/projects/pif-open_source_intermediate_representation_for_datapaths/>. Acesso em: 07.07.2020.

P4.ORG. **P4**. 2017. URL: <http://p4.org/>. Acesso em: 07.07.2020.

PAN, J.; PAUL, S.; JAIN, R. A survey of the research on future internet architectures. **IEEE Communications Magazine**, v. 49, n. 7, p. 26–36, July 2011. ISSN 0163-6804. Disponível em: <https://doi.org/10.1109/MCOM.2011.5936152>.

PEREIRA, J. H. d. S. **Modelo de título para a próxima geração de Internet.** Tese (Doutorado) — Universidade de São Paulo, 2012. Access on 15/06/2020.

PEREIRA, J. H. de S. **Modelo de título para a próxima geração de Internet**. Tese (Doutorado) — Universidade de São Paulo, São Paulo, SP, 2012. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3142/tde-13062013-113304/pt-br.php>. Acesso em: 7.7.2017.

PETRAKIS, E. G.; KONTOCHRISTOS, I.; RASTSINSKAGIA, K. Fi-swot: Secure federated service oriented architecture for the semantic internet of things. In: **AINA (2)**. [s.n.], 2021. p. 200–214. Disponível em: <https://doi.org/10.1007/978-3-030-75075-6_16>.

PFAFF, B. et al. The design and implementation of open vswitch. In: **12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)**. [S.l.: s.n.], 2015. p. 117–130.

PIRKO, J.; FELDMAN, S. **Ethernet switch device driver model (switchdev)**. 2014. Available at: <https://www.kernel.org/doc/Documentation/networking/switchdev.txt>. Access on 10/08/2020.

PÍRKO, J.; HAT, R. Hardware switches-the open-source approach. In: TECHNICAL REPORT, NETDEV 0.1, THE TECHNICAL CONFERENCE ON LINUX NETWORKING .... [S.l.], 2015.

PRODANOV, C. C.; FREITAS, E. C. D. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2ª Edição**. [S.l.]: Editora Feevale, 2013.

PROJECT, L. F. C. **Quality of Service (QoS)**. 2016. Available at: <https://docs.openvswitch.org/en/latest/faq/qos/>. Access on 10/08/2020.

_____. **What Is Open vSwitch?** 2016. Available at: <https://docs.openvswitch.org/en/latest/intro/what-is-ovs/>. Access on 10/08/2020.

QRATOR. **Linux Switchdev the Mellanox way**. 2020. Available at: <https://blog.qrator.net/en/linux-switchdev-mellanox-way_104/>. Access on 15/03/2021.

REXFORD, J.; DOVROLIS, C. Future internet architecture: clean-slate versus evolutionary research. **Communications of the ACM**, ACM, v. 53, n. 9, p. 36–40, 2010. Disponível em: <https://doi.org/10.1145/1810891.1810906>.

SEOK, S.-J. et al. multifia multi-dimensional future internet architecture. In: IEEE. **2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)**. 2017. p. 87–92. Disponível em: <https://doi.org/10.1109/ICUFN.2017.7993753>.

SESKAR, I. et al. Mobilityfirst future internet architecture project. In: **Proceedings of the 7th Asian Internet Engineering Conference**. New York, NY, USA: ACM, 2011. (AINTEC '11), p. 1–3. ISBN 978-1-4503-1062-8. Disponível em: <http://doi.acm.org/10.1145/2089016.2089017>.

SHAHBAZ, M. et al. Pisces: A programmable, protocol-independent software switch. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. [s.n.], 2016. p. 525–538. Disponível em: <https://doi.org/10.1145/2934872.2934886>.

SHAHBAZ, M.; GROZ, R. Inferring mealy machines. In: SPRINGER. **International Symposium on Formal Methods**. 2009. p. 207–222. Disponível em: <https://doi.org/10.1007/978-3-642-05089-3_14>.

SILVA, F. d. O. **Endereçamento por título: uma forma de encaminhamento multicast para a próxima geração de redes de computadores.** Tese (Doutorado) — Universidade de São Paulo, 2013. Access on 10/06/2020.

SILVA, F. de O. **Modelo de título para a próxima geração de Internet**. Tese (Doutorado) — Universidade de São Paulo, Uberlandia, MG, 2013. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3142/tde-22092014-111409/pt-br.php>. Acesso em: 7.7.2017.

SILVA, F. de O. et al. On the analysis of multicast traffic over the entity title architecture. In: **2012 18th IEEE International Conference on Networks (ICON)**. [S.l.: s.n.], 2012. p. 30–35. ISSN 1531-2216.

SITNIKOV, S. **Mellanox Spectrum Linux Switch Powered by SwitchDev**. 2018. Available at: <https://blog.mellanox.com/2018/12/mellanox-spectrum-linux-switch-powered-by-switchdev/>. Access on 10/08/2020.

SONKOLY, B. et al. On qos support to ofelia and openflow. In: IEEE. **2012 European Workshop on Software Defined Networking**. 2012. p. 109–113. Disponível em: <https://doi.org/10.1109/EWSDN.2012.26>.

TANENBAUM, A. S. **Computer Network**. 3th. ed. [S.l.]: Prentice Hall, PTR, 1996.

TRAINING, L. system programming. **tc(8) — Linux manual page**. 2021. Available at: <https://man7.org/linux/man-pages/man8/tc.8.html>. Access on 10/08/2021.

VARIS, N. Anatomy of a linux bridge. In: **Proceedings of Seminar on Network Protocols in Operating Systems**. [S.l.: s.n.], 2012. p. 58.

WAZLAWICK, R. S. **Metedologia de Pesquisa para Ciência da Computação**. 2. ed. Rio de Janeiro, RJ, Brasil: Elsevier, 2009. ISBN 9788535266436. Disponível em: <http://books.google.com.br/books?id=ZLbCKKWQ6OQC>.

XIA, W. et al. A survey on software-defined networking. **IEEE Communications Surveys & Tutorials**, IEEE, v. 17, n. 1, p. 27–51, 2014. Disponível em: <https://doi.org/10.1109/COMST.2014.2330903>.

YIN, R. K. **Estudo de Caso-: Planejamento e métodos**. [S.l.]: Bookman editora, 2015.

ZEN. **The minimum internet speeds for streaming video in 2019**. 2019. Available at: <https://www.zen.co.uk/blog/posts/zen-blog/2019/04/18/the-minimum-internet-speeds-for-streaming-video-in-2019#:~:text=YouTube%3A%20Minimum%20Speed%20Requirements,need%20at%20least%204%20Mbps.> Access on 28/03/2022.

ZHANG, L. et al. **Named Data Networking Tech Report 001**. 2010. (Technical Report).

ZHANI, M. F.; ELBAKOURY, H. Flexngia: A flexible internet architecture for the next-generation tactile internet. **Journal of Network and Systems Management**, Springer, v. 28, n. 4, p. 751–795, 2020. Disponível em: <https://doi.org/10.1007/s10922-020-09525-0>.

ZILBERMAN, N. et al. Reconfigurable network systems and software-defined networking. **Proceedings of the IEEE**, IEEE, v. 103, n. 7, p. 1102–1124, 2015. Disponível em: <https://doi.org/10.1109/JPROC.2015.2435732>.

APÊNDICE **A**

Quantitativo de Dados Mensurados nos Experimentos 1, 2 e 3

Na Figura 64 é possível observar o quantitativo de dados mensurados nos testes dos experimentos 1, 2 e 3. Esses dados foram capturados através de uma aplicação que monitorou os tráfegos nas respectivas portas físicas onde estavam instanciados os *Workspaces* em questão e o tráfego geral. A aplicação instituída utilizou basicamente os comandos do *Traffic Control (TC) sudo tc class show dev <interface>* e *sudo tc filter show dev <interface>* durante intervalos de tempo predefinidos.

Experimento 1

Streaming: Prio: 1 Rate: 1Gbit Ceil: 1Gbit Burst: 1600b Cburst: 1600b

Workspace 01: Prio: 5 Rate: 100mbit Ceil: 100mbit Burst: 1600b Cburst: 1600b

	00:00:00	00:00:05	00:00:10	00:00:15	00:00:20	00:00:25	00:00:30	00:00:35	00:00:40	00:00:45	00:00:50	00:00:55	00:01:00	00:01:05	00:01:10	00:01:15	00:01:20	00:01:25	00:01:30	00:01:35	Total Frames
Streaming	178092	178335	178847	179255	179812	180287	180762	181377	182351	182451	182523	183036	183513	184076	184665	185051	185612	186090	186420	186803	187287		**9195**
Workspace 01	6247	6360	6612	6803	7060	7287	7516	7797	8275	8445	8560	8793	9026	9285	9566	9566	10025	10241	10504	10745	10979		**4732**

Experimento 2

Streaming: Prio: 7 Rate: 100kbit Ceil: 100kbit Burst: 1600b Cburst: 1600b

Workspace 01: Prio: 5 Rate: 100mbit Ceil: 100mbit Burst: 1600b Cburst: 1600b

	00:00:00	00:00:05	00:00:10	00:00:15	00:00:20	00:00:25	00:00:30	00:00:35	00:00:40	00:00:45	00:00:50	00:00:55	00:01:00	00:01:05	00:01:10	00:01:15	00:01:20	00:01:25	00:01:30	00:01:35	Total Frames
Streaming	202577	202631	202701	202766	202821	202886	202945	203045	203117	203188	203257	203333	203394	203462	203535	203603	203676	203743	203825	203891	203964		**1387**
Workspace 01	11365	11555	11791	12028	12251	12498	12673	12984	13236	13493	13720	13995	14223	14469	14726	14979	15215	15464	15709	15959	16139		**4774**

Experimento 3

Streaming: Prio: 7 Rate: 1mbit Ceil: 1mbit Burst: 1600b Cburst: 1600b

Workspace 01: Prio: 5 Rate: 1mbit Ceil: 1mbit Burst: 1600b Cburst: 1600b

Workspace 02: Prio: 3 Rate: 100mbit Ceil: 100mbit Burst: 1600b Cburst: 1600b

	00:00:00	00:00:05	00:00:10	00:00:15	00:00:20	00:00:25	00:00:30	00:00:35	00:00:40	00:00:45	00:00:50	00:00:55	00:01:00	00:01:05	00:01:10	00:01:15	00:01:20	00:01:25	00:01:30	00:01:35	Total Frames
Streaming	248101	248956	249515	250269	250636	251059	251262	251647	251970	252276	252627	252910	253268	253604	254162	254685	255281	255865	256392	256861	257384		**9283**
Workspace 01	35621	36033	36304	36680	36864	37071	37169	37363	37522	37671	37850	37987	38157	38325	38589	38856	39147	39425	39704	39936	40179		**4558**
Workspace 02	10105	10517	10788	11167	11353	11560	11658	11850	12011	12158	12335	12472	12641	12809	13074	13342	13635	13918	14195	14427	14686		**4581**

Figura 64 – Refinamento dos dados dos arquivos de testes.

APÊNDICE **B**

Quantitativo de Dados Mensurados no Cenário 4

No experimento 4, além da quantidade de atualizações realizadas nos parâmetros de controle de tráfego em relação aos experimentos 1, 2 e 3, o tempo de observação é maior. As Figuras 65, 66, 67, 68 e 69 ilustram o quantitativo de dados coletados diretamente das portas físicas do elemento de rede em relação aos tráfegos *Workspace* e geral.

Esses dados foram capturados através de uma aplicação que monitorou os tráfegos nas respectivas portas físicas onde estavam instanciados os *Workspaces* e o tráfego geral. A aplicação instituída utilizou basicamente os comandos do *Traffic Control (TC) sudo tc class show dev <interface>* e *sudo tc filter show dev <interface>* durante intervalos de tempo predefinidos.

enp3s0 (Streaming)																																							
time (s)	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52														
packets																																									
Scenario 1 - prio 7 rate 100Kbit ceil 100Kbit burst 1600b cburst 1600b																																								
156053	156287	156448	156458	156486	156499	156583	156784	156836	156840	157132	157298	157400	157530	157791	157974	158080	158195	158339	158517	158731	159093	159504	159895	160109	160324														
54	56	58	60	62	64	66	68	70	72	74	76	78	80	82	84	86	88	90	92	94	96	98	100	102	104														
160538	160743	161029	161377	161611	161847	162055	162259	162477	162794	163206	163641	163887	164108	164320	164536	164753	165038	165390	165620	165984	166226	166432	166682	166918	167122														
106	108	110	112	114	116	118	120	122	124	126	128	130	132	134	136	138	140	142	144	146	148	150	152	154	156														
167337	167562	167776	167981	168206	168411	168619	168832	169044	169259	169476	169684	169897	170110	170318	170532	170760	170986	171197	171417	171650	171864	172077	172305	172513	172733														
158	160	162	164	166	168	170	172	174	176	178	180	182	184	186	188	190	192	194	196	198	200	202	204	206	208														
172962	173360	173570	173780	173990	174217	174443	174622	174900	175105	175327	175540	175751	175984	176202	176413	176629	176840	177070	177279	177501	177714	177918	178139	178346	178555														
210	212	214	216	218	220	222	224	226	228	230	232	234	236	238	240	242	244	246	248	250	252	254	256	258	260														
178766	179033	179245	179524	179752	180182	180621	181013	181460	181861	182101	182454	182860	183165	183486	183756	183972	184177	184439	184853	185267	185592	185974	186182	186398	186621														
262	264	266	268	270	272																																			
186830	187050	187470	187879	188278	188657																																			
Scenario 2 – prio 7 rate 100Kbit ceil 100Kbit burst 1600b cburst 1600b																																								
274	276	278	280	282	284	286	288	290	292	294	296	298	300	302	304	306	308	310	312	314	316	318	320	322	324														
188741	188823	188919	188966	189026	189077	189144	189220	189286	189350	189411	189473	189528	189595	189659	189730	189783	189831	189885	189944	190005	190057	190113	190170	190231	190291														
326	328	330	332	334	336	338	340	342	344	346	348	350	352	354	356	358	360	362	364	366	368	370	372	374	376														
190349	190412	190470	190525	190595	190655	190720	190780	190846	190914	190980	191079	191150	191239	191292	191376	191444	191510	191579	191636	191703	191760	191816	191875	191937	191994														
378	380	382	384	386	388	390	392	394	396	398	400	402	404	406	408	410	412	414	416	418																			
192048	192099	192159	192208	192259	192307	192360	192422	192483	192548	192621	192669	192716	192770	192808	192882	192935	193001	193053	193125	193182																			
Scenario 3 - prio 4 rate 300Kbit ceil 300Kbit burst 1599b cburst 1599b																																							
420	422	424	426	428	430	432	434	436	438	440	442	444	446	448	450	452	454	456	458	460	462	464	466	468	470														
193269	193424	193546	193693	193822	193950	194097	194237	194373	194513	194672	194817	194959	195097	195235	195360	195499	195634	195768	195909	196064	196196	196337	196478	196622	196792														
472	474	476	478	480	482	484	486	488	490	492	494	496	498	500	502	504	506	508	510	512	514	516	518	520	522														
196937	197064	197225	197225	197525	197669	197811	197951	198093	198239	198381	198528	198644	198753	198869	198978	199096	199207	199321	199427	199543	199676	199793	199917	200036	200159														
524	526	528	530	532	534	536	538	540	542	544																														
200280	200396	200507	200639	200769	200884	201001	201140	201263	201379	201504																														

Figura 65 – Tabela de dados experimento 4. Dados coletados do tráfego na porta enp3s0.

enp4s0 (Streaming)																										
time (s)	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52
packets	Scenario 1 - prio 6 rate 100Mbit ceil 100Mbit burst 1600b cburst 1600b																									
	12704	12710	12714	12721	12724	12727	12733	12737	12741	12745	12749	12754	12758	12766	12767	12773	12779	12783	12789	12800	12807	12811	12830	12836	12838	12843
	12848	12850	12857	12861	12870	12889	12893	12897	12901	12913	12917	12928	12939	12942	12947	12954	12956	12962	12966	12969	12983	12988	12991	13012	13017	13020
	13024	13029	13032	13040	13045	13046	13053	13058	13062	13067	13074	13076	13082	13087	13090	13095	13099	13112	13118	13123	13140	13145	13150	13154	13158	13163
	13165	13193	13200	13203	13209	13214	13217	13224	13238	13242	13254	13260	13266	13271	13283	13287	13290	13297	13300	13303	13309	13313	13318	13322	13326	13330
	13335	13343	13346	13351	13356	13357	13367	13370	13373	13378	13381	13388	13392	13396	13399	13403	13411	13413	13418	13423	13433	13443	13447	13452	13457	13461
	13466	13474	13502	13504	13508	13516																				
	Scenario 2 - prio 7 rate 100Kbit ceil 100Kbit burst 1600b cburst 1600b																									
	13523	13530	13534	13535	13549	13556	13561	13575	13585	13591	13595	13602	13604	13610	13617	13619	13625	13629	13632	13639	13642	13646	13650	13656	13661	13665
	13671	13680	13688	13695	13710	13720	13724	13732	13738	13743	13750	13753	13757	13763	13768	13774	13777	13780	13786	13792	13796	13800	13806	13810	13815	13821
	13823	13828	13833	13836	13843	13846	13851	13865	13873	13876	13881	13885	13889	13894	13900	13902	13908	13913	13915	13921	13924					
	Scenario 3 - prio 4 rate 300Kbit ceil 300Kbit burst 1599b cburst 1599b																									
	13934	13943	13947	13961	13965	13972	13982	13987	13993	13995	14001	14007	14008	14016	14020	14023	14029	14033	14038	14042	14047	14050	14055	14062	14063	14070
	14074	14076	14083	14087	14093	14097	14101	14107	14111	14118	14120	14124	14131	14134	14139	14143	14147	14151	14156	14162	14164	14169	14174	14177	14184	14188
	14191	14197	14200	14207	14211	14217	14222	14226	14232	14234	14239															

Figura 66 – Tabela de dados experimento 4. Dados coletados do tráfego na porta enp4s0.

enp4s0 (Workspace99)																																							
time (s)	Scenario 01 - prio 2 rate 100Mbit ceil 100Mbit burst 1600b cburst 1600b																																						
packets	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52													
40612	40612	40612	40612	40612	40612	40612	40612	40612	40612	40662	40763	40863	40966	41063	41169	41273	41375	41476	41577	41681	41778	41874	41968	42073	42180													
42287	42395	42496	42601	42709	42818	42924	43032	43136	43239	43337	43438	43548	43654	43760	43871	43976																							
44079	44184	44292	44394	44503	44609	44716	44826	44932	45043	45155	45263	45366	45476	45583	45688	45796	45906	46013	46120	46227	46334	46443	46550	46661															
46715	46757	46799	46841	46883	46924	46966	47009	47050	47093	47134	47176	47218	47260	47302	47344	47386																							
47422	47426	47431	47435	47439	47443	47447	47452	47456	47460	47464	47468	47472	47477	47481	47485	47489	47493	47498	47502	47506	47510	47514	47519	47523	47527															
47531	47535	47540	47544	47548	47552	47557	47561	47565	47569	47573	47578	47582	47586	47590	47594	47599	47603	47607	47611	47615	47620	47624	47628	47632	47636															
47641	47645	47649	47653	47657	47662	47666	47670	47674	47678	47682	47687	47691	47695	47699	47703	47708	47712	47716	47720	47724	47729	47733																	
47973	48078	48190	48298	48409	48516	48621	48728	48837	48945	49047	49153	49258	49370	49477	49580	49687	49793	49900	50007	50115	50226	50335	50442	50549	50658															
50763	50873	50983	51093	51200	51301	51406	51512	51620	51729	51837	51945	52049	52152	52259	52368	52476	52582	52690	52798	52907	53014	53121	53227	53332	53442															
53548	53652	53760	53864	53973	54077	54179	54288	54394	54500	54607	54716	54822	54927	55032	55134	55242	55350	55460	55568	55677	55788	55895	56002	56109	56215															
56322	56434	56546	56652	56759	56868	56973	57083	57186	57292	57402	57509	57617	57726	57836	57947	58052	58158	58269	58378	58490	58598	58707	58816	58925	59038															
59143	59250	59354	59463	59567	59677	59788	59896																																	

Scenario 2 - prio 7 rate 100Kbit ceil 100Kbit burst 1600b cburst 1600b																																							
88	90	92	94	96	98	100	102	104	106	108	110	112	114	116	118	120	122	124	126	128	130	132	134	136															
44079	44184	44292	44394	44503	44609	44716	44826	44932	45043	45155	45263	45366	45476	45583	45688	45796	45906	46013	46120	46227	46334	46443	46550	46661															

Scenario 3 - prio 7 rate 10Kbit ceil 10Kbit burst 1600b cburst 1600b																																							
138	140	142	144	146	148	150	152	154	156	158	160	162	164	166	168	170																							
46715	46757	46799	46841	46883	46924	46966	47009	47050	47093	47134	47176	47218	47260	47302	47344	47386																							

Scenario 4 - prio 7 rate 1Kbit ceil 1Kbit burst 1600b cburst 1600b																																							
172	174	176	178	180	182	184	186	188	190	192	194	196	198	200	202	204	206	208	210	212	214	216	218	220	222														
47422	47426	47431	47435	47439	47443	47447	47452	47456	47460	47464	47468	47472	47477	47481	47485	47489	47493	47498	47502	47506	47510	47514	47519	47523	47527															
47531	47535	47540	47544	47548	47552	47557	47561	47565	47569	47573	47578	47582	47586	47590	47594	47599	47603	47607	47611	47615	47620	47624	47628	47632	47636															
47641	47645	47649	47653	47657	47662	47666	47670	47674	47678	47682	47687	47691	47695	47699	47703	47708	47712	47716	47720	47724	47729	47733																	
47973	48078	48190	48298	48409	48516	48621	48728	48837	48945	49047	49153	49258	49370	49477	49580	49687	49793	49900	50007	50115	50226	50335	50442	50549	50658															
50763	50873	50983	51093	51200	51301	51406	51512	51620	51729	51837	51945	52049	52152	52259	52368	52476	52582	52690	52798	52907	53014	53121	53227	53332	53442															
53548	53652	53760	53864	53973	54077	54179	54288	54394	54500	54607	54716	54822	54927	55032	55134	55242	55350	55460	55568	55677	55788	55895	56002	56109	56215															
56322	56434	56546	56652	56759	56868	56973	57083	57186	57292	57402	57509	57617	57726	57836	57947	58052	58158	58269	58378	58490	58598	58707	58816	58925	59038															
59143	59250	59354	59463	59567	59677	59788	59896																																	

Scenario 5 - prio 2 rate 1Mbit ceil 1Mbit burst 1600b cburst 1600b																																							
322	324	326	328	330	332	334	336	338	340	342	344	346	348	350	352	354	356	358	360	362	364	366	368	370	372														
374	376	378	380	382	384	386	388	390	392	394	396	398	400	402	404	406	408	410	412	414	416	418	420	422	424														
426	428	430	432	434	436	438	440	442	444	446	448	450	452	454	456	458	460	462	464	466	468	470	472	474	476														
478	480	482	484	486	488	490	492	494	496	498	500	502	504	506	508	510	512	514	516	518	520	522	524	526	528														
530	532	534	536	538	540	542	544																																

Figura 67 – Tabela de dados experimento 4. Dados coletados do tráfego na porta enp4s0.

enp6s0 (Streaming)

time (s) / packets	Scenario 1 - prio 6 rate 100Mbit ceil 100Mbit burst 1600b cburst 1600b																									
	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52
127461	127905	128184	128197	128214	128223	128397	128741	128803	128807	129309	129413	129416	129440	129760	129867	129872	129884	129923	129945	129957	130256	130626	130990	130994	131005	
131013	131014	131167	131451	131472	131487	131491	131496	131516	131727	132138	132555	132594	132607	132616	132624	132640	132815	133082	133103	133366	133432	133435	133451	133485	133487	
133489	133504	133510	133517	133532	133534	133541	133546	133551	133562	133573	133578	133587	133597	133599	133606	133627	133632	133639	133649	133653	133657	133666	133696	133699	133720	
133746	133946	133956	133966	133971	133990	134007	134025	134045	134048	134051	134053	134062	134081	134086	134091	134100	134113	134135	134140	134153	134159	134164	134172	134175	134178	
134183	134246	134254	134319	134343	134666	134962	135321	135728	136114	136156	136434	136850	137047	137285	137399	137407	137410	137517	137943	138355	138570	138920	138923	138929	138948	
138959	138964	139340	139758	140134	140450																					

time (s) / packets	Scenario 2 - prio 7 rate 100Kbit ceil 100Kbit burst 1600b cburst 1600b																									
	274	276	278	280	282	284	286	288	290	292	294	296	298	300	302	304	306	308	310	312	314	316	318	320	322	324
140455	140487	140566	140573	140596	140621	140650	140684	140716	140750	140779	140800	140823	140858	140891	140935	140959	140989	141026	141058	141090	141110	141130	141155	141180	141205	
141229	141257	141280	141302	141328	141354	141377	141400	141435	141473	141521	141592	141627	141704	141742	141796	141833	141865	141899	141921	141961	141989	142015	142052	142080	142103	
142122	142142	142167	142181	142201	142220	142244	142276	142309	142364	142425	142449	142473	142500	142529	142575	142598	142638	142659	142700	142724						

time (s) / packets	Scenario 3 - prio 4 rate 300Kbit ceil 300Kbit burst 1599b cburst 1599b																									
	420	422	424	426	428	430	432	434	436	438	440	442	444	446	448	450	452	454	456	458	460	462	464	466	468	470
142766	142825	142848	142898	142927	142958	143013	143071	143122	143178	143236	143291	143345	143409	143474	143528	143584	143639	143692	143750	143816	143869	143924	143980	144032	144095	
144159	144213	144291	144356	144416	144470	144531	144593	144648	144703	144755	144796	144803	144810	144816	144818	144832	144836	144850	144859	144860	144877	144885	144898	144906	144914	
144919	144933	144943	144954	144980	144988	144992	145031	145039	145042	145049																

Figura 68 – Tabela de dados experimento 4. Dados coletados do tráfego na porta enp6s0.

enp6s0 (Workspace99)																																					
time (s)	Scenario 01 - prio 2 rate 100Mbit ceil 100Mbit burst 1600b cburst 1600b																																				
packets	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52											
	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22164	22208	22305	22406	22509	22606	22708	22807											
	54	56	58	60	62	64	66	68	70	72	74	76	78	80	82	84	86																				
	22908	23003	23102	23198	23296	23398	23496	23590	23688	23785	23886	23986	24085	24186	24285	24384	24479																				
	Scenario 2 - prio 7 rate 100Kbit ceil 100Kbit burst 1600b cburst 1600b																																				
	88	90	92	94	96	98	100	102	104	106	108	110	112	114	116	118	120	122	124	126	128	130	132	134	136												
	24502	24522	24542	24562	24582	24602	24622	24642	24663	24683	24703	24723	24743	24763	24783	24803	24823	24843	24863	24884	24904	24924	24944	24964	24984												
	Scenario 3 - prio 7 rate 10Kbit ceil 10Kbit burst 1600b cburst 1600b																																				
	138	140	142	144	146	148	150	152	154	156	158	160	162	164	166	168	170																				
	24993	24995	24997	24999	25001	25003	25005	25007	25009	25011	25013	25015	25017	25019	25021	25023	25025																				
	Scenario 4 - prio 7 rate 1Kbit ceil 1Kbit burst 1600b cburst 1600b																																				
	172	174	176	178	180	182	184	186	188	190	192	194	196	198	200	202	204	206	208	210	212	214	216	218	220	222											
	25027	25028	25028	25028	25028	25028	25029	25029	25029	25029	25029	25030	25030	25030	25030	25030	25031	25031	25031	25031	25031	25032	25032	25032	25032	25032											
	224	226	228	230	232	234	236	238	240	242	244	246	248	250	252	254	256	258	260	262	264	266	268	270	272	274											
	25033	25033	25033	25033	25033	25034	25034	25034	25034	25034	25035	25035	25035	25035	25035	25036	25036	25036	25036	25036	25037	25037	25037	25037	25037	25038											
	276	278	280	282	284	286	288	290	292	294	296	298	300	302	304	306	308	310	312	314	316	318	320														
	25038	25038	25038	25038	25039	25039	25039	25039	25039	25040	25040	25040	25040	25040	25041	25041	25041	25041	25041	25042	25042	25042	25042														
	Scenario 5 - prio 2 rate 1Mbit ceil 1Mbit burst 1600b cburst 1600b																																				
	322	324	326	328	330	332	334	336	338	340	342	344	346	348	350	352	354	356	358	360	362	364	366	368	370	372											
	25042	25243	25444	25645	25846	26049	26250	26450	26651	26852	27034	27135	27231	27333	27435	27537	27637	27734	27828	27925	28022	28121	28220	28320	28419	28517											
	374	376	378	380	382	384	386	388	390	392	394	396	398	400	402	404	406	408	410	412	414	416	418	420	422	424											
	28611	28708	28805	28907	29008	29107	29206	29304	29403	29504	29604	29699	29798	29896	29993	30094	30190	30286	30380	30479	30572	30669	30767	30868	30968	31065											
	426	428	430	432	434	436	438	440	442	444	446	448	450	452	454	456	458	460	462	464	466	468	470	472	474	476											
	31158	31256	31357	31451	31549	31646	31743	31844	31943	32043	32141	32241	32346	32445	32544	32640	32738	32833	32933	33030	33128	33227	33324	33423	33526	33622											
	478	480	482	484	486	488	490	492	494	496	498	500	502	504	506	508	510	512	514	516	518	520	522	524	526	528											
	33720	33821	33922	34016	34115	34217	34318	34414	34511	34611	34708	34811	34910	35005	35105	35208	35305	35400	35503	35605	35703	35805	35905	36005	36103	36195											
	530	532	534	536	538	540	542	544																													
	36292	36390	36485	36582	36673	36771	36870	36971																													

Figura 69 – Tabela de dados experimento 4. Dados coletados do tráfego na porta enp6s0.