Arthur Vieira de Assis Moreira

GERENCIAMENTO DE FILAS E POLICIAMENTO DE INGRESSO EM REDES SDN

Belo Horizonte

2025

Arthur Vieira de Assis Moreira

GERENCIAMENTO DE FILAS E POLICIAMENTO DE INGRESSO EM REDES SDN

Monografia apresentada durante o Seminário dos Trabalhos de Conclusão do Curso de Graduação em Engenharia Elétrica da UFMG, como parte dos requisitos necessários à obtenção do título de Engenheiro Eletricista.

Universidade Federal de Minas Gerais – UFMG

Escola de Engenharia

Curso de Graduação em Engenharia Elétrica

Orientador: Prof. Dr. Luciano de Errico

Belo Horizonte

2025

Agradecimentos

Em primeiro lugar, agradeço a Deus, fonte de toda sabedoria, força e inspiração. Sem Sua presença constante, este trabalho e todo o percurso acadêmico até aqui teriam sido muito mais árduos.

Agradeço profundamente à minha família, que sempre foi meu alicerce. Em especial, à minha mãe, por seu amor incondicional, por cada gesto de apoio, por acreditar em mim mesmo nos momentos em que duvidei, e por ser o exemplo de dedicação e coragem que me inspira diariamente.

Aos amigos, meu sincero reconhecimento. Os momentos compartilhados, as conversas, os estudos em grupo e o apoio mútuo foram essenciais para tornar esta jornada mais leve e significativa.

Aos colegas do LabCOM, meu muito obrigado pela convivência enriquecedora, pela troca de conhecimentos, pela parceria nas pesquisas e pelos inúmeros aprendizados. Foi um privilégio dividir esse espaço com pessoas tão comprometidas e colaborativas.

Finalizo a graduação com gratidão por cada pessoa que, de alguma forma, contribuiu para minha formação pessoal e acadêmica ao longo desses anos na UFMG.

"Combati o bom combate,
terminei a corrida,
guardei a fé."
(Bíblia Sagrada, 2 Timóteo 4:7)

Resumo

Este trabalho tem como objetivo investigar a aplicação de Redes Definidas por Software (SDN) na provisão de Qualidade de Serviço (QoS), por meio da implementação de políticas de policiamento de ingresso e gerenciamento de filas. A proposta é analisada por meio de uma topologia de rede composta por dois switches OpenvSwitch, a priori com quatro hosts, a posteriori com seis hosts em cenários distintos de tráfego. Foram desenvolvidos e executados cinco experimentos com scripts em Python, variando desde a ausência de QoS até a aplicação combinada de gerenciamento de filas e policiamento de entrada. Os resultados demonstram que, mesmo em situações de competição por recursos, a utilização conjunta dessas técnicas possibilita maior controle sobre o fluxo de dados, estabilidade na taxa de transferência e melhoria na alocação de recursos da rede. A abordagem evidencia a efetividade das SDNs como plataforma flexível e programável para suporte à QoS, com potencial para aplicações em redes que demandam gerenciamento dinâmico e granular do tráfego.

Palavras-chave: SDN. Qualidade de Serviço. Policiamento de Ingresso. Gerenciamento de Filas. OpenFlow.

Abstract

This work investigates the application of Software-Defined Networks (SDN) to provide Quality of Service (QoS) through the implementation of ingress policing and queue management policies. The proposal is analyzed using a network topology composed of two OpenvSwitch switches, a priori with four hosts, a posteriori with six hosts, in different traffic scenarios. Five experiments were developed and executed with Python scripts, ranging from no QoS to the combined application of queue management and ingress policing. The results demonstrate that, even in situations of resource competition, the combined use of these techniques enables greater control over data flow, stable throughput, and improved network resource allocation. The approach highlights the effectiveness of SDN as a flexible and programmable platform for supporting QoS, with potential for applications in networks that require dynamic and granular traffic management.

Keywords: SDN. Quality of Service. Ingress Policing. Queue Management. OpenFlow.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

5G	Quinta geração de redes móveis
ACK	Acknowledgment
AQM	Active Queue Management
API	Application Programming Interface
ARPA	Advanced Research Projects Agency
ARPANET	Advanced Research Projects Agency Network
BBR	Bottleneck Bandwidth and Round-trip Propagation Time (em referência à variante do TCP)
BGP	Border Gateway Protocol
CAKE	Common Applications Kept Enhanced
CoDel	Controlled Delay
CoS	Class of Service
CUBIC	Congestion Control Algorithm (em referência à variante do TCP)
DDoS	Distributed Denial of Service
DNS	Domain Name System
DQN	Deep Q-Network
DSCP	Differentiated Services Code Point
ECMP	Equal-Cost Multi-Path
FIFO	First-In, First-Out
FQ	Fair Queuing
FQ-CODEL	Fair Queuing with Controlled Delay
FTP	File Transfer Protocol
HTB	Hierarchical Token Bucket
HTTP	Hypertext Transfer Protocol

ICMP	Internet Control Message Protocol
IFB	Intermediate Functional Block
IoT	Internet of Things (Internet das Coisas)
ISO	International Organization for Standardization
kB	Kilobytes
kbps	Kilobits por segundo
LAN	Local Area Network
LSP	Label Switched Paths
LTS	Long-Term Support
Mbps	Megabits por segundo
MPLS	Multiprotocol Label Switching
NFV	Network Functions Virtualization
ONF	Open Networking Foundation
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OvS	Open vSwitch
PIE	Proportional Integral Controller Enhanced
QoE	Quality of Experience (Qualidade da Experiência)
QoS	Quality of Service (Qualidade de Serviço)
QMIP	Queue Management with Ingress Policing
RED	Random Early Detection
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
RSVP-TE	Resource Reservation Protocol-Traffic Engineering
SFC	Service Function Chaining
SLA	Service Level Agreement

SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
TBF	Token Bucket Filter
TC	Traffic Control
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TSO	TCP Segmentation Offload
UDP	User Datagram Protocol
VoIP	Voice over IP
WAN	Wide Area Network
WFQ	Weighted Fair Queuing

Sumário

1 Introdução

O desenvolvimento das redes de computadores remonta à década de 1960, com a criação da ARPANET, a primeira rede baseada no paradigma de comutação de pacotes, financiada pela ARPA (Advanced Research Projects Agency) dos EUA. Em 1969, a ARPANET já conectava universidades e centros de pesquisa (TANENBAUM; WETHERALL, 2011). Na década de 1970, a padronização de protocolos culminou no modelo OSI (Open Systems Interconnection) pela ISO em 1984 e na adoção da pilha TCP/IP, oficializada pelo Departamento de Defesa dos EUA em 1983 (KUROSE; ROSS, 2017).

Nos anos 1990, as redes locais (LANs) e o uso de switches e roteadores tornaram-se populares, com a Ethernet e a escalabilidade das redes empresariais. Com o início dos anos 2000, surgiram limitações na arquitetura das redes tradicionais. O aumento da complexidade das redes e a necessidade de maior flexibilidade geraram novos desafios, como a dificuldade em lidar com o tráfego heterogêneo. Em resposta, pesquisadores da Universidade de Stanford propuseram o conceito de Redes Definidas por Software (SDN) em 2008, com o projeto Ethane (CASADO et al., 2007), seguido pela criação do protocolo OpenFlow (MCKEOWN et al., 2008), que se tornou um marco importante para a evolução da arquitetura de redes.

1.1 Paradigma SDN

As Redes Definidas por Software dissociam o plano de controle do plano de dados, possibilitando maior programabilidade, abstração e automação no gerenciamento da infraestrutura de rede. Desde sua popularização, o SDN tem sido adotado em data centers, redes de operadoras e na nuvem, com o apoio da Open Networking Foundation (ONF). Estima-se que o mercado de SDN atinja US$ 70 bilhões até 2030 (MARKETS AND MARKETS, 2023), impulsionado pela adoção em 5G, IoT e edge computing, com previsões de crescimento notáveis nos próximos anos.

A literatura tem explorado questões como segurança, QoS, desempenho em ambientes de virtualização e integração com tecnologias emergentes. A pesquisa de Kreutz et al. (2015) apresenta uma análise da arquitetura SDN e seus desafios, como a escalabilidade, confiabilidade e a necessidade de padronização. Outro aspecto importante discutido na literatura envolve as implicações do SDN na implementação de novas políticas de gerenciamento e controle de tráfego, que têm o potencial de transformar significativamente a forma como as redes são projetadas e gerenciadas.

1.2 Motivação e justificativa

A complexidade das redes modernas e o aumento do tráfego heterogêneo exigem soluções para garantir a entrega eficiente e confiável de dados. Aplicações críticas demandam QoS, como controle de latência, jitter e banda mínima. Sob essa perspectiva, a implementação de QoS em redes tradicionais é complexa, devido às limitações de equipamentos legados. Assim, o SDN surge como uma solução ao permitir o controle programático de políticas de tráfego de forma centralizada e adaptativa, oferecendo maior flexibilidade para o gerenciamento da rede.

Este estudo justifica-se pela necessidade de compreender os efeitos dessas políticas sobre o desempenho da rede, especialmente em cenários com múltiplos fluxos concorrentes e limitação de recursos. A pesquisa busca avaliar se a adoção de políticas de QoS em ambientes SDN pode melhorar a eficiência da distribuição de largura de banda, minimizando congestionamentos e otimizando a entrega de dados.

1.3 Objetivos

O presente trabalho tem como foco principal a análise do impacto da aplicação de políticas de gerenciamento de filas e policiamento de ingresso sobre a qualidade de serviço (QoS) em redes SDN.

1.3.1 Objetivos específicos

Intrinsecamente ligados ao propósito principal, os seguintes aspectos delineiam as abordagens essenciais para seu cumprimento:

a) comparar o desempenho da rede com e sem QoS, observando a taxa de transferência e estabilidade em cenários com tráfego concorrente;

b) testar o comportamento das políticas sob diferentes condições, como priorização de fluxos, competição por fila e controle de banda;

c) validar os resultados com métricas de desempenho, identificando a efetividade, limitações e possíveis melhorias das abordagens adotadas.

1.4 Organização

Este trabalho está estruturado em cinco capítulos. O Capítulo 1 apresenta a introdução, contextualizando o surgimento das redes SDN, motivações e objetivos. O Capítulo 2 reúne as revisões bibliográficas utilizadas. O Capítulo 3 descreve a metodologia, detalhando as ferramentas e procedimentos propostos. O Capítulo 4 apresenta e analisa

os resultados obtidos a partir dos experimentos realizados, destacando o comportamento observado e as implicações dos achados. Por fim, o Capítulo 5 apresenta a conclusão, discutindo as principais contribuições do trabalho, suas limitações, as dificuldades encontradas durante o desenvolvimento e possíveis direções para trabalhos futuros.

2 Revisão bibliográfica

Este capítulo apresenta um descritivo abrangente sobre os principais conceitos relacionados às redes de computadores, iniciando pela análise dos protocolos de comunicação TCP e UDP, que são fundamentais para a transmissão de dados entre dispositivos em diferentes aplicações. Em seguida, aborda-se a arquitetura das Redes Definidas por Software, ressaltando a importância do controlador como elemento central que separa o plano de controle do plano de dados, proporcionando maior flexibilidade e programabilidade à rede. O protocolo OpenFlow é destacado como um dos principais padrões que permitem essa comunicação entre controlador e dispositivos de rede. Além disso, o capítulo discute os conceitos de Engenharia de Tráfego, enfatizando sua relevância na otimização do desempenho das redes e na garantia da QoS, especialmente em ambientes com grande demanda e diversidade de aplicações. Por fim, são apresentadas considerações sobre os desafios e vantagens da aplicação das tecnologias SDN no contexto atual, estabelecendo a base teórica essencial para o desenvolvimento de soluções inovadoras no gerenciamento e controle de redes modernas.

2.1 Protocolos de transporte: TCP e UDP

Os protocolos TCP e UDP constituem os pilares da camada de transporte no modelo de referência da Internet. Embora compartilhem o mesmo nível hierárquico na pilha de protocolos, suas arquiteturas, funcionalidades e aplicações são fundamentalmente distintas. O TCP é um protocolo orientado à conexão, que oferece mecanismos robustos de confiabilidade, controle de fluxo, controle de congestionamento e ordenação dos dados. Já o UDP, por sua vez, é um protocolo não orientado à conexão, cuja leveza estrutural o torna ideal para aplicações que demandam baixa latência e toleram perda ocasional de pacotes bem como eventual quebra na ordem de transmissão.

O TCP destaca-se por seu modelo de comunicação confiável, estabelecido por meio de um processo conhecido como *three-way handshake*, que sincroniza os estados entre emissor e receptor antes do início da transferência de dados. A confiabilidade é assegurada através da numeração sequencial dos bytes transmitidos, da verificação de integridade por somas de verificação (checksums), do reconhecimento positivo (ACKs) e do mecanismo de retransmissão de segmentos perdidos. Além disso, incorpora algoritmos avançados de controle de fluxo, como o *sliding window*, e controle de congestionamento, como o *slow start*, *congestion avoidance* e *fast retransmit*, que ajustam dinamicamente a taxa de envio com base nas condições da rede. Essas características fazem do TCP a escolha preferencial para aplicações que exigem entrega íntegra dos dados, como HTTP,

FTP, SMTP e SSH (TANENBAUM; WETHERALL, 2011).

Em contraste, o UDP oferece um serviço de transporte simples, com mínima sobrecarga e sem qualquer garantia de entrega, ordem ou integridade além da aplicação básica de checksum. Seu cabeçalho fixo de apenas 8 bytes reflete essa simplicidade, contendo apenas os campos de porta de origem, porta de destino, comprimento e checksum. A ausência de estabelecimento de conexão e de mecanismos de controle torna o UDP ideal para aplicações em tempo real, onde a latência é crítica e a eventual perda de pacotes é aceitável, como transmissões de vídeo em tempo real, VoIP, DNS e jogos online. A leveza do UDP permite que os dados sejam entregues com menor atraso e maior eficiência de banda em relação ao TCP (KUROSE; ROSS, 2017).

Para fins comparativos, a Tabela 1 resume as principais diferenças entre os dois protocolos, destacando tanto aspectos qualitativos quanto dados numéricos estruturais.

Tabela 1 – Comparativo entre os protocolos TCP e UDP.

Característica	**TCP**	**UDP**
Orientação à conexão	Sim	Não
Confiabilidade	Alta	Baixa
Controle de fluxo	Sim	Não
Controle de congestionamento	Sim	Não
Verificação de integridade	Checksum (16 bits)	Checksum (16 bits)
Retransmissão de pacotes	Sim	Não
Tamanho do cabeçalho	20–60 bytes	8 bytes
Overhead	Alto	Baixo
Aplicações típicas	HTTP, FTP, SMTP	VoIP, DNS, Streaming

Fonte: Adaptado de Postel (1980), Postel (1981).

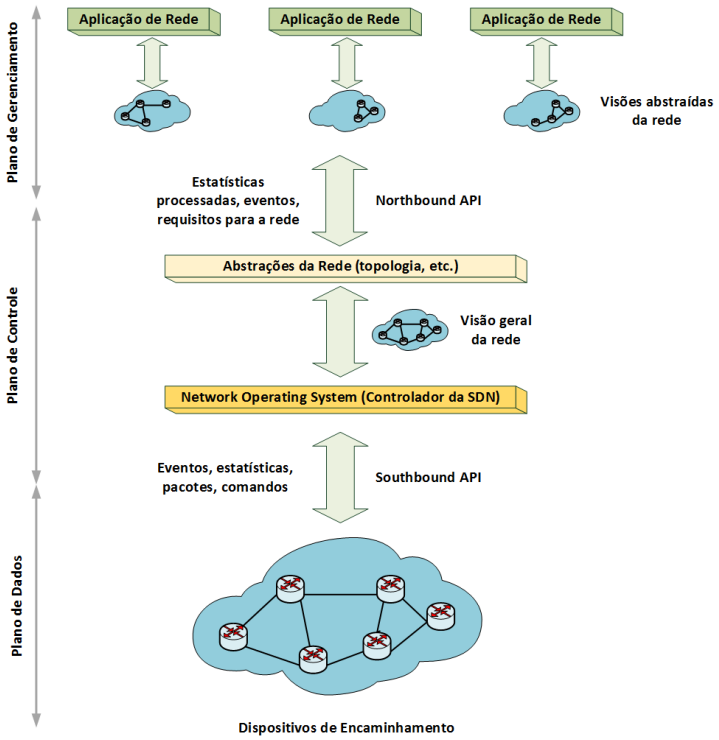
A análise da Tabela 1 revela que o TCP oferece confiabilidade por meio de mecanismos de controle, o que é vantajoso em redes instáveis, mas resulta em maior overhead e latência, podendo reduzir a vazão da aplicação. Em contrapartida, o UDP, ao abrir mão dessas garantias, proporciona um transporte mais leve e eficiente, ideal para aplicações que exigem continuidade e baixa latência. Ambos protocolos coexistem no ambiente da Internet, sendo utilizados conforme as demandas da aplicação. Protocolos mais recentes, como o QUIC, ilustram uma tendência atual de combinar a eficiência do UDP com mecanismos de confiabilidade implementados na própria aplicação, promovendo abordagens híbridas e mais adaptáveis na camada de transporte.

2.2 SDN e OpenFlow: arquitetura e funcionamento

A arquitetura de uma rede SDN é composta por três camadas principais: o controlador, os switches e o protocolo de comunicação entre eles. No entanto, uma visão

mais abrangente da arquitetura, conforme ilustrado na Figura 1, inclui também os planos de gerenciamento, controle e dados, além das interfaces *northbound* e *southbound*, que facilitam a comunicação entre essas camadas (GRUPO DE TELEINFORMÁTICA E AUTOMAÇÃO - UFRJ, 2015).

Figura 1 – Arquitetura de uma rede SDN



Fonte: (GRUPO DE TELEINFORMÁTICA E AUTOMAÇÃO - UFRJ, 2015).

O controlador SDN é o componente central da arquitetura, responsável por fornecer uma visão global e centralizada da rede. Ele coleta informações em tempo real sobre o estado da rede e utiliza esses dados para tomar decisões de encaminhamento, aplicar políticas de QoS, realizar balanceamento de carga e gerenciar a alocação de largura de banda de forma programática. Essa centralização do controle, consoante Kreutz et al. (2015), permite uma gestão mais eficiente e flexível dos recursos da rede, além de facilitar a implementação de novas funcionalidades e serviços. A comunicação com os switches ocorre por meio de protocolos *southbound*, como o OpenFlow, que permite a instalação de regras de encaminhamento e o monitoramento do desempenho da rede (SALMAN et al., 2017).

Além disso, o controlador pode ser programado para implementar políticas de segurança, como detecção de intrusões e mitigação de ataques de negação de serviço (DDoS), aumentando a resiliência da rede. Como mencionado por Nunes et al. (2014), a capacidade de adaptar dinamicamente a rede às condições operacionais é uma das principais vantagens da abordagem SDN, proporcionando maior agilidade na implementação de novas

políticas.

Os switches SDN operam com base em tabelas de fluxo que determinam como os pacotes devem ser tratados. Eles são dispositivos programáveis que recebem instruções diretamente do controlador, permitindo reconfigurações dinâmicas conforme a necessidade, como a priorização de tráfego sensível à latência ou a limitação de banda. Essa separação entre o plano de controle (controlador) e o plano de dados (switches) é uma característica fundamental do SDN, garantindo que a rede seja mais adaptável e responsiva a alterações nas condições de tráfego (GRUPO DE TELEINFORMÁTICA E AUTOMAÇÃO - UFRJ, 2015).

O plano de gerenciamento, por sua vez, atua como a interface entre os administradores de rede e o controlador. Por meio de APIs *northbound*, é possível desenvolver aplicações que interagem com o controlador de maneira abstrata, para monitoramento e controle da rede. Já as APIs *southbound*, como o OpenFlow, estabelecem a comunicação entre o controlador e os dispositivos de rede.

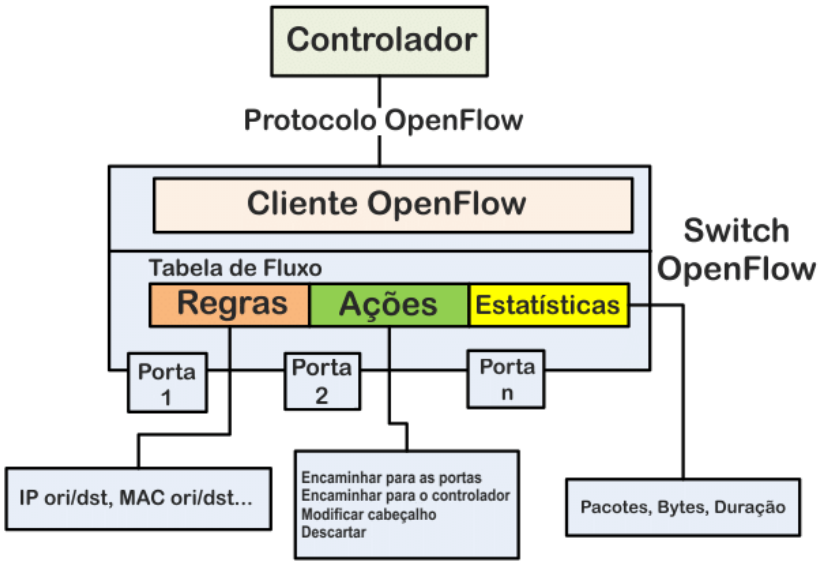
Por fim, a arquitetura SDN também suporta linguagens de programação de alto nível, como a *NetCore*, que permitem a definição declarativa de políticas de rede. Essas políticas são traduzidas em regras simples que podem ser executadas diretamente pelos switches, enquanto decisões mais complexas permanecem sob a responsabilidade do controlador.

2.2.1 Funcionamento do OpenFlow

Quando um pacote chega a um switch, ele é comparado com as entradas nas tabelas de fluxo. Se uma correspondência for encontrada, o switch executa a ação associada, como encaminhar o pacote para uma porta específica ou modificá-lo antes de encaminhá-lo. Caso não haja uma correspondência, o pacote é enviado ao controlador, que decide como o pacote deve ser tratado e pode atualizar a tabela de fluxo do switch conforme necessário. Isso permite que a rede seja programada e adaptada em tempo real, proporcionando uma flexibilidade que não é possível nas redes tradicionais (MCKEOWN et al., 2008), (SEEGER et al., 2019).

A Figura 2 ilustra o funcionamento do protocolo OpenFlow, com o controlador programando os switches e gerenciando as tabelas de fluxo, permitindo a flexibilidade e adaptabilidade das redes SDN. Sob essa ótica, a separação entre o plano de controle e o plano de dados permite uma gestão eficiente e dinâmica da rede, e o protocolo OpenFlow possibilita que o controlador configure os switches e adapte a rede em tempo real. Essa abordagem tem se mostrado ideal para ambientes de alta demanda, como data centers e redes corporativas, onde a capacidade de ajustar rapidamente a infraestrutura de rede é crucial para garantir o desempenho e a experiência do usuário.

Figura 2 – Esquemático do funcionamento do protocolo OpenFlow.



Fonte: Adaptado de Nunes et al. (2014).

2.2.1.1 Tabelas de fluxos no OpenFlow

As tabelas de fluxos são a principal estrutura utilizada pelos switches OpenFlow para realizar o encaminhamento de pacotes. Cada entrada nas tabelas de fluxo consiste em um conjunto de critérios de correspondência, como endereços IP, portas de origem e destino, tipo de protocolo, entre outros. Quando um pacote chega ao switch, ele é comparado com as entradas na tabela de fluxo. Se uma correspondência for encontrada, as instruções associadas são executadas, que podem incluir ações como encaminhar o pacote para uma porta específica, modificar o pacote ou enviar o pacote de volta ao controlador.

Por sua vez, pode ser configurada pelo controlador para priorizar determinados fluxos de tráfego, aplicar políticas de QoS ou realizar outras ações baseadas nas necessidades da rede. Além disso, os switches OpenFlow podem ter múltiplas tabelas de fluxo, permitindo uma filtragem e processamento mais refinados dos pacotes, e possibilitando o uso de pipelines de processamento para tratar pacotes de maneira eficiente e escalável (KREUTZ et al., 2015), (MCKEOWN et al., 2008).

2.2.1.2 Filas no OpenFlow

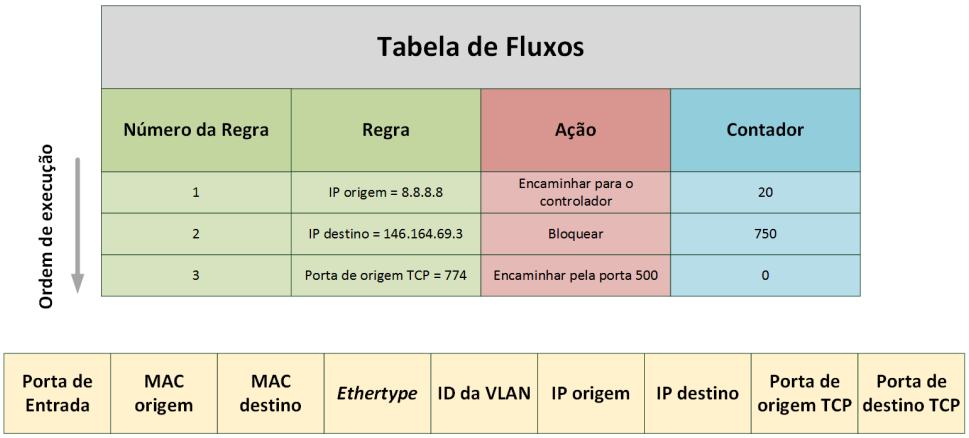
As filas são utilizadas pelos switches OpenFlow para gerenciar o tráfego e aplicar políticas de QoS. As filas podem ser configuradas para controlar a taxa de transmissão de pacotes para diferentes tipos de tráfego, garantindo que fluxos de alta prioridade, como voz e vídeo, recebam a largura de banda necessária, enquanto fluxos menos prioritários, como transferências de arquivos, podem ser alocados com menor largura de banda. O

controlador pode programar os switches para aplicar diferentes políticas de QoS, como limitar a largura de banda ou priorizar certos fluxos de tráfego.

As filas são associadas às portas dos switches e podem ser configuradas para atender aos requisitos de QoS de diferentes tipos de tráfego. Isso permite que o SDN otimize o uso da largura de banda disponível, garantindo que aplicações críticas, como chamadas VoIP e videoconferências, recebam a largura de banda necessária para funcionar corretamente, mesmo em redes congestionadas (MCKEOWN et al., 2008), (KREUTZ et al., 2015).

A Figura 3 ilustra os campos do cabeçalho usados pelo OpenFlow para identificar e controlar fluxos, permitindo ao controlador definir regras que direcionam o encaminhamento dos pacotes nos switches.

Figura 3 – Tabela e campos do cabeçalho utilizados na especificação de fluxos do OpenFlow



Fonte: (GRUPO DE TELEINFORMÁTICA E AUTOMAÇÃO - UFRJ, 2015).

2.2.1.3 Versões do OpenFlow

Desde sua criação, o protocolo OpenFlow passou por diversas versões que introduziram melhorias significativas em termos de flexibilidade, desempenho e suporte a funcionalidades. A versão 1.0, lançada em 2009, estabeleceu os fundamentos do modelo de comutação baseado em fluxo, com uma única tabela de fluxos e suporte limitado a ações e campos de cabeçalho (MCKEOWN et al., 2008). A versão 1.1 introduziu múltiplas tabelas de fluxo e o conceito de grupos, permitindo uma maior modularidade nas decisões de encaminhamento (OPEN NETWORKING FOUNDATION, 2011). Já a versão 1.3, amplamente adotada, expandiu o suporte a novos protocolos, incluindo IPv6 e MPLS, e incorporou contadores mais sofisticados, além de suporte a correspondência em campos adicionais (OPEN NETWORKING FOUNDATION, 2012). A partir da versão 1.4, recursos como controle de fluxo baseado em metadados e eventos síncronos foram adicionados, ampliando o controle sobre o plano de dados. Por fim, a versão 1.5 trouxe refinamentos

como o suporte a instruções de pipeline mais detalhadas e maior controle sobre tabelas de grupo (OPEN NETWORKING FOUNDATION, 2015a). Essas evoluções tornaram o OpenFlow mais robusto e adaptável a diferentes cenários, desde data centers até redes WAN e ambientes virtualizados.

2.3 Engenharia de Tráfego

A Engenharia de Tráfego constitui uma disciplina crítica na operação de redes modernas, visando a otimização da performance da rede por meio da análise, previsão e controle do fluxo de dados. Diferente de abordagens puramente estáticas, considera a topologia da rede, a alocação de recursos e os padrões de demanda para redirecionar dinamicamente os fluxos de tráfego com o objetivo de minimizar a perda de pacotes, o atraso, a variação de latência (jitter) e o uso ineficiente da largura de banda (AWDUCHE et al., 2002).

Tradicionalmente, foi implementada com base em protocolos como o MPLS (Multiprotocol Label Switching), que permite a definição de caminhos explícitos e o roteamento baseado em rótulos, promovendo controle detalhado sobre a utilização dos enlaces e a qualidade de serviço (FARREL, 2003).

2.3.1 Aspectos fundamentais

A Engenharia de Tráfego pode ser decomposta em cinco componentes técnicos principais, conforme apresentado em Li et al. (2007):

a) coleta e medição de tráfego: envolve a obtenção de métricas como volume de tráfego, taxa de pacotes, latência e perdas, frequentemente com ferramentas como NetFlow, sFlow ou sondas baseadas em OpenFlow;

b) modelagem de rede: consiste na abstração matemática da topologia e das capacidades de enlace, permitindo a análise de caminhos e gargalos. Modelos baseados em grafos direcionados e otimizadores convexos são frequentemente utilizados;

c) previsão de demanda: técnicas estatísticas e de aprendizado de máquina são aplicadas para estimar padrões futuros de tráfego com base em históricos e sazonalidades (ZHANG et al., 2003);

d) cálculo de roteamento ótimo: aplicação de algoritmos de otimização para balanceamento de carga e minimização de custo, como Linear Programming, Fattree, ECMP (Equal-Cost Multi-Path), além de heurísticas evolutivas;

e) reconfiguração dinâmica execução controlada de políticas de redirecionamento de tráfego, geralmente com o suporte de SDN Controllers (ex.: ONOS, Ryu), para alterar rotas sem causar instabilidade de convergência;

2.3.2 Multiprotocol Label Switching (MPLS)

Embora abordagens modernas baseadas em SDN tenham ganhado destaque por sua flexibilidade e centralização, o *Multiprotocol Label Switching* (MPLS) continua sendo uma tecnologia amplamente consolidada para Engenharia de Tráfego em redes operadoras e de backbone. O MPLS insere uma camada de comutação baseada em rótulos (*labels*), permitindo a criação de circuitos virtuais com encaminhamento determinístico e independente do endereço IP, promovendo assim uma maior previsibilidade e controle sobre os caminhos de tráfego (ROSEN; VISWANATHAN; CALLON, 2001).

A principal contribuição do MPLS para a Engenharia de Tráfego reside em sua capacidade de estabelecer caminhos explícitos através da rede — denominados *Label Switched Paths* (LSPs) — que podem ser definidos com base em políticas de otimização como menor custo, menor atraso ou maior capacidade disponível. Essa característica torna o MPLS especialmente eficiente para o balanceamento de carga interdomínio e a reserva de recursos, sobretudo quando combinado com o protocolo RSVP-TE (*Resource Reservation Protocol - Traffic Engineering*) (LEWIS, 2004).

Além disso, o MPLS possibilita a aplicação de diferentes classes de serviço (*Class of Service* - CoS) ao tráfego de rede, utilizando campos como o *EXP* (*experimental bits*) do rótulo MPLS para implementar políticas de QoS, como priorização de pacotes sensíveis a atraso. Essa abordagem tem sido particularmente valorizada em ambientes onde é necessário garantir *Service Level Agreements* (SLAs) rigorosos, como redes de operadoras, datacenters interligados e infraestruturas críticas de missão (OSBORNE; SIMHA, 2002).

2.3.3 Engenharia de Tráfego em SDN

Com a separação entre os planos de controle e de dados, a Engenharia de Tráfego em SDN supera limitações clássicas dos protocolos de roteamento distribuído, como OSPF e BGP, ao permitir decisões centralizadas, holísticas e baseadas em políticas de alto nível (ALVIZU et al., 2016). O controlador SDN, com visão completa da rede, pode aplicar algoritmos de otimização global, reconfigurando os fluxos conforme requisitos de SLA (Service-Level Agreement), detecção de falhas ou mudanças súbitas na carga.

A flexibilidade do SDN também permite a integração nativa com funções de gerenciamento de filas, shaping e policing de tráfego, promovendo um plano de dados ativo e inteligente. Em cenários de Engenharia de Tráfego baseados em QoS, filas com priorização diferenciada são fundamentais para garantir o desempenho de fluxos críticos em

coexistência com tráfegos elásticos.

2.3.4 Métricas de desempenho e objetivos de otimização

As métricas tradicionalmente consideradas na Engenharia de Tráfego incluem:

a) utilização de enlace: maximização da eficiência de uso da banda disponível sem sobrecarga;

b) delay de fim-a-fim: minimização da latência acumulada entre origem e destino;

c) perda de pacotes: redução do descarte de pacotes em enlaces críticos;

d) jitter: controle da variação do tempo entre pacotes consecutivos, essencial para fluxos sensíveis a tempo;

e) robustez: capacidade da rede em reconfigurar rotas diante de falhas ou degradações.

A Engenharia de Tráfego frequentemente formula esses objetivos como problemas de otimização sob restrições (e.g., de capacidade ou de SLA), resolvidos via técnicas como programação linear, algoritmos genéticos ou aprendizado por reforço (CHENG et al., 2018).

2.4 Gerenciamento de filas e QoS em redes de computadores

O gerenciamento de filas em redes de computadores desempenha um papel fundamental na garantia da eficiência, estabilidade e desempenho das redes, principalmente em ambientes de alto tráfego. Este processo envolve a organização e o controle do tratamento de pacotes de dados, determinando a ordem de sua transmissão com o objetivo de otimizar a utilização da largura de banda e minimizar problemas como a latência excessiva e a perda de pacotes, características críticas para a QoS (MORREALE; UBALDI; SANTORO, 2019; GUNDOGAN; HABIB, 2018).

QoS refere-se ao conjunto de técnicas utilizadas para fornecer garantias de desempenho aos fluxos de tráfego em uma rede. A implementação de QoS é essencial em cenários onde múltiplos serviços competem por recursos de rede, como voz sobre IP (VoIP), vídeo sob demanda e aplicações críticas de dados (MOEYERSONS et al., 2019). Entre os mecanismos utilizados para garantir QoS estão a classificação de pacotes, o enfileiramento, o policiamento de ingresso e a marcação de tráfego (OH et al., 2018).

2.4.1 Classificação de pacotes para QoS

A classificação de pacotes consiste no processo de inspeção dos cabeçalhos dos pacotes de dados a fim de determinar a qual classe de tráfego cada um pertence. Essa

classificação pode se basear em múltiplos critérios, incluindo endereços IP de origem e destino, números de porta e tipo de protocolo. Além disso, muitas redes utilizam marcações explícitas como o campo Differentiated Services Code Point (DSCP) presente no cabeçalho IP cuja função indica a classe de serviço que o pacote deve receber dentro do modelo DiffServ, permitindo que roteadores e switches apliquem diferentes níveis de prioridade, garantias de desempenho e políticas de tratamento ao tráfego. (GUNDOGAN; HABIB, 2018).

Em arquiteturas de QoS modernas, a classificação é o primeiro estágio antes da aplicação de políticas de gerenciamento de tráfego. Uma vez classificados, os pacotes podem ser submetidos a diferentes filas de prioridade, tratados com níveis distintos de largura de banda, ou mesmo descartados, dependendo do nível de congestionamento da rede (OH et al., 2018). A Tabela 2 apresenta um comparativo entre os principais métodos de classificação de pacotes, destacando suas bases, complexidade e granularidade.

Tabela 2 – Comparativo entre métodos de classificação de pacotes.

Método	**Base de Classificação**	**Complexidade**	**Granularidade**
Por Porta	TCP/UDP (Portas 80, 443, etc.)	Baixa	Média
Por IP	IP de origem/destino	Média	Média
Por Protocolo	ICMP, TCP, UDP, etc.	Baixa	Baixa
Por DSCP	Campo DSCP no IP	Baixa	Alta
Inspeção Profunda	Conteúdo do payload	Alta	Muito Alta

Fonte: Adaptado de Gundogan e Habib (2018).

2.4.2 Técnicas clássicas de gerenciamento de filas

Em redes convencionais, técnicas como FIFO (First-In, First-Out), Weighted Fair Queuing (WFQ) e Random Early Detection (RED) são amplamente utilizadas (LIU; ZHAO, 2017; PORWAL; YADAV; CHARHATE, 2008). FIFO é simples, mas ineficiente em redes congestionadas, pois não prioriza pacotes urgentes (MORREALE; UBALDI; SANTORO, 2019). WFQ distribui largura de banda de forma justa entre fluxos, priorizando aplicações críticas (LIU; ZHAO, 2017), enquanto RED evita congestionamentos descartando pacotes de forma aleatória antes que a fila atinja sua capacidade máxima, prevenindo o fenômeno de bufferbloat (BERG; BRORSSON, 2014). A Tabela 3 apresenta um comparativo entre essas técnicas de enfileiramento, destacando suas características em termos de priorização de tráfego, justiça (*fairness*) na alocação de recursos e complexidade.

Tabela 3 – Comparativo entre técnicas de enfileiramento.

Técnica	**Prioriza Tráfego?**	**Fairness**	**Complexidade**
FIFO	Não	Baixa	Baixa
WFQ	Sim (por peso)	Alta	Média
RED	Indiretamente (por descarte)	Moderada	Alta

Fonte: Adaptado de Liu e Zhao (2017), Porwal, Yadav e Charhate (2008), Berg e Brorsson (2014).

2.4.3 Gerenciamento de filas em SDN

A arquitetura SDN permite configurar múltiplas filas por porta nos switches, como o Open vSwitch (OvS), e priorizar fluxos de acordo com sua criticidade (MCKEOWN et al., 2008). Esse gerenciamento centralizado é possível graças ao protocolo OpenFlow, que permite ao controlador modificar as tabelas de fluxo dos switches e aplicar políticas de QoS (SEEGER et al., 2019; SALMAN et al., 2017). O uso de múltiplas filas e alocação dinâmica de largura de banda aumenta significativamente a eficiência da rede (ZHANG; LIU; ZHAO, 2018).

2.4.4 Policiamento de ingresso em redes SDN

O policiamento de ingresso, responsável por controlar a taxa de entrada de tráfego na rede, ganha novos recursos em SDN. O controlador pode aplicar limites dinâmicos de banda nos pontos de entrada, evitando sobrecargas e protegendo fluxos de alta prioridade (SEEGER et al., 2019; LI et al., 2018). Essa funcionalidade é fundamental em cenários onde múltiplos fluxos competem pela mesma largura de banda, assegurando que aplicações críticas mantenham sua performance (SALMAN et al., 2017).

2.4.5 QoS e desafios de implementação em SDN

Apesar das vantagens, a implementação de QoS em SDN enfrenta desafios. A interoperabilidade entre dispositivos de diferentes fabricantes, a ausência de padronizações amplas e as limitações de escalabilidade ainda representam entraves relevantes (ZHANG; LIU; ZHAO, 2018). Além disso, a natureza centralizada do controle impõe a necessidade de controladores robustos, capazes de processar grandes volumes de dados com baixa latência (LI et al., 2018; OH et al., 2018).

2.5 Estado de arte em Redes SDN e QoS

A crescente demanda por aplicações sensíveis à latência e com altos requisitos de desempenho tem impulsionado pesquisas voltadas à otimização da QoS em Redes

Definidas por Software. Nesse contexto, duas estratégias se destacam: o gerenciamento eficiente de filas e o policiamento de ingresso. Esses mecanismos são fundamentais para garantir que diferentes tipos de tráfego sejam tratados de forma diferenciada, respeitando requisitos como latência, perda e jitter.

Nessa perspectiva, o algoritmo QMIP (Queue Management with Ingress Policing) aloca dinamicamente filas com base nas características do tráfego de entrada, utilizando informações em tempo real coletadas pelo controlador SDN, proporcionando ganhos significativos de desempenho, como a redução de até 42% na latência média e a melhora na utilização de banda em ambientes heterogêneos (ABDELGHANY; MUTHANNA; SALAH, 2023).

Complementando essa abordagem, Kattepur, Dwarakanath e Krishnamurthy (2021) exploraram o uso de aprendizado por reforço baseado em modelo para configurar filas de portas de roteadores, otimizando políticas de priorização e controle de congestionamento. De forma similar, foi proposto um agente de aprendizado por reforço profundo (DQN) para encadeamento de funções de serviço (SFC) em redes 5G com SDN/NFV, considerando tanto QoS quanto QoE na formulação das políticas (CHEN et al., 2018).

Um mecanismo dinâmico de policiamento de ingresso, que ajusta políticas proativamente com foco em prevenção de congestionamentos e ataques, foi desenvolvido por Guo, Wang e Liu (2022). O sistema demonstrou uma melhoria de até 35% na taxa de entrega de pacotes em comparação com abordagens estáticas.

O uso de aprendizado de máquina também tem sido eficaz na configuração de políticas de QoS. A arquitetura MLQoS emprega classificadores para mapear padrões de tráfego a configurações de filas ideais, melhorando em até 28% a satisfação de SLA em aplicações VoIP e de vídeo sob demanda (BARI; HUQ; RAYCHOUDHURY, 2022).

Do ponto de vista da interoperabilidade, foram propostos gateways capazes de traduzir políticas SDN para redes legadas, permitindo que dispositivos tradicionais também se beneficiem de controle refinado de tráfego (LUO; ZHANG; WANG, 2023). Outro trabalho relevante propôs um esquema adaptativo de buffers em redes SDN-IoT, que ajusta o tamanho de filas conforme carga e criticidade, com impacto positivo na latência e jitter (ZHANG; LI; WANG, 2024).

Outros frameworks especializados também foram desenvolvidos. Manzanares-Lopez, Garcia-Almiñana e Badia (2018) introduziram o DEDCA, mecanismo de controle dinâmico de acesso ao canal em redes sem fio com SDN. Xia, Yang e Huang (2022) apresentaram uma abordagem com gerenciamento de filas multinível e seleção de caminhos baseada em DiffServ, resultando em melhorias na aceitação de fluxos em redes celulares SDN.

Além disso, foi proposta uma estratégia de roteamento com consciência de congestionamento, que prioriza pacotes importantes e garante confiabilidade em redes inteligentes

(ALI; KHAN; REHMAN, 2022). Em cenários mais desafiadores, soluções baseadas em aprendizado por reforço para QoS e segurança foram apresentadas por Liu, Zhao e Zhang (2024) e Song et al. (2024), mostrando-se eficazes mesmo sob ataques como DDoS.

Essas pesquisas evidenciam a sinergia entre controle programável, monitoramento em tempo real e inteligência artificial como pilares do futuro da QoS em redes SDN. A combinação de gerenciamento de filas, policiamento de ingresso e segurança adaptativa se mostra essencial para atender às crescentes exigências das aplicações críticas.

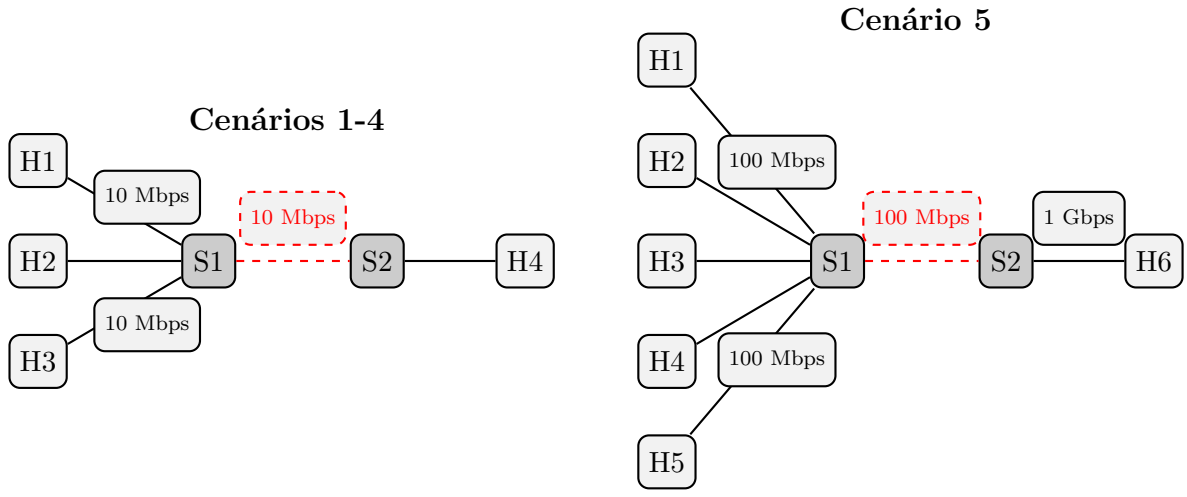
3 Metodologia

Este capítulo descreve a metodologia utilizada para avaliar os efeitos do gerenciamento de filas e do policiamento de ingresso na provisão de Qualidade de Serviço em Redes Definidas por Software. Utilizando uma topologia de rede controlada, foi possível demonstrar a eficiência das técnicas mencionadas, por meio de medições empíricas realizadas em diferentes cenários de tráfego.

3.1 Topologia da Rede

A topologia experimental primária foi composta por quatro hosts e dois switches Open vSwitch, configurados no ambiente de emulação Mininet. Três dos hosts (H1, H2 e H3) estavam conectados ao switch S1, que, por sua vez, conectava-se ao switch S2, ao qual estava conectado o host H4. Essa configuração permitiu a análise de diferentes padrões de tráfego e contenção entre fluxos concorrentes. Para o último cenário, ela foi modificada, com um número de hosts maior (H1, H2, H3, H4, H5 e H6), sendo os cinco primeiros conectados a S1 e o último a S2, as velocidades dos links variam. Ambas são ilustradas na Figura 4.

Figura 4 – Topologias utilizadas nos experimentos.



3.2 Ferramentas e Tecnologias

Neste trabalho, foram utilizadas diversas ferramentas essenciais para a implementação e avaliação de políticas de QoS e controle de tráfego. Cada uma delas teve um papel específico na simulação de redes, controle de tráfego e avaliação do desempenho da rede, conforme descrito a seguir.

3.2.1 Mininet

Mininet foi o emulador de rede utilizado para a criação de topologias SDN complexas em um ambiente virtualizado, permitindo a prototipagem rápida e a realização de testes sem a necessidade de um ambiente físico dispendioso. Proporcionou a simulação de redes com componentes como switches, hosts e controladores (MININET, 2022), todos executados em contêineres no mesmo sistema físico. Nesse sentido, foi escolhido devido à sua flexibilidade e capacidade de simular redes SDN com um alto grau de realismo, permitindo a manipulação de tabelas de fluxo e a configuração de políticas de QoS programáveis em tempo real. O uso do Mininet possibilitou a replicação de cenários de tráfego competitivo, essenciais para testar os efeitos de diferentes políticas de gerenciamento de filas (LANTZ; HELLER; MCKEOWN, 2010).

3.2.2 Open vSwitch (OvS)

O Open vSwitch foi o switch virtualizado de código aberto utilizado, projetado para suportar SDN, implementando o protocolo OpenFlow e proporcionando flexibilidade na configuração de regras de encaminhamento e gerenciamento de tráfego de forma programável (LINUX FOUNDATION COLLABORATIVE PROJECT, 2016). Como discutido por Kreutz et al. (2015), o OvS permitiu a implementação de múltiplas filas em portas de switches, o que possibilitou o controle detalhado sobre a prioridade e a alocação de largura de banda para diferentes fluxos de tráfego. No contexto deste trabalho, o OvS foi utilizado para configurar as filas de tráfego e as políticas de controle de taxa, permitindo que diferentes tipos de tráfego (como voz, vídeo e dados) fossem tratados conforme suas necessidades específicas de QoS (PFAFF et al., 2015).

3.2.3 Ryu

O Ryu foi o controlador SDN programável escrito em Python utilizado para gerenciar a comunicação entre os switches e implementar as políticas de rede em tempo real. O controlador SDN foi a peça central que orquestrou o tráfego na rede, permitindo a programação de regras dinâmicas para o roteamento e o policiamento de ingressos. O Ryu foi escolhido por sua compatibilidade com o protocolo OpenFlow 1.3, que ofereceu suporte a recursos avançados, como múltiplas filas por porta e controle preciso sobre a alocação de largura de banda, aspectos essenciais para a implementação das técnicas de QoS no contexto de SDN (MCKEOWN et al., 2008).

3.2.4 OpenFlow

O protocolo OpenFlow, conforme descrito por McKeown et al. (2008), foi fundamental para a operação das redes SDN, pois permitiu a separação do plano de controle

e do plano de dados. Por meio do OpenFlow, foi possível configurar switches de forma programática, o que facilitou a implementação de políticas de QoS. A principal vantagem do OpenFlow foi sua capacidade de manipular as tabelas de fluxo dos switches, permitindo que o tráfego fosse direcionado de acordo com as regras definidas no controlador. Além disso, o OpenFlow possibilitou o uso de múltiplas filas em cada porta dos switches, permitindo que diferentes fluxos de tráfego fossem tratados de forma distinta, de acordo com suas necessidades de latência e largura de banda (OPEN NETWORKING FOUNDATION, 2015b).

3.2.5 iPerf

O iPerf foi a ferramenta de benchmark de rede utilizada para medir o desempenho da rede, especificamente a largura de banda entre dois hosts (IPERF, 2025). Através do iPerf, foi possível gerar tráfego de rede entre os hosts e medir a taxa de transferência, o que foi essencial para a avaliação da eficácia das políticas de QoS implementadas. O iPerf suportou testes de tráfego TCP e UDP, sendo útil para simular diferentes cenários de tráfego e avaliar o impacto de políticas de controle de tráfego, como o policiamento de ingresso, sobre o desempenho da rede (SILVA; JúNIOR, 2014).

3.2.6 Xterm

O Xterm foi utilizado para facilitar a interação com os hosts virtuais durante os testes. Essa ferramenta permitiu a execução de múltiplos scripts de maneira simultânea em diferentes janelas, possibilitando o monitoramento em tempo real das métricas de tráfego e a coleta de logs durante os experimentos (DICKEY, 2023). O uso do Xterm facilitou a operação dos testes em conjunto com o Mininet, Open vSwitch e Ryu, proporcionando um ambiente integrado para a análise do desempenho da rede e das políticas de QoS aplicadas.

3.2.7 Linux

Todos os experimentos descritos neste trabalho foram conduzidos utilizando o sistema operacional Linux em sua distribuição Ubuntu 22.04.5 LTS. Caracterizou-se pela sua arquitetura modular, estabilidade e ampla adoção em servidores e ambientes de pesquisa de redes. A escolha do Linux para este trabalho deveu-se ao seu suporte nativo às ferramentas de emulação de rede (Mininet) (LANTZ; HELLER; MCKEOWN, 2010), controladores SDN (como o Ryu), e ao subsistema avançado de Controle de Tráfego (`tc`) (ALMESBERGER, 2001), que foi essencial para a implementação das políticas de QoS avaliadas. A vasta documentação e comunidade ativa facilitaram a configuração e gestão do ambiente experimental.

3.2.7.1 Mecanismos de QoS no Linux: tc, HTB, FQ-CODEL e TBF

A implementação prática de políticas de Qualidade de Serviço (QoS) apoiou-se no subsistema de Controle de Tráfego (`tc`) do kernel Linux. Esta ferramenta permitiu a configuração detalhada de disciplinas de enfileiramento (*qdiscs*), classes e filtros para gerenciar como os pacotes foram tratados e transmitidos por uma interface de rede (ALMESBERGER, 2001). O `tc` ofereceu uma vasta gama de algoritmos para modelagem (*shaping*), policiamento (*policing*), priorização e agendamento de pacotes, sendo amplamente utilizado tanto em sistemas Linux autônomos quanto em conjunto com controladores SDN para aplicar políticas de rede dinâmicas.

Dentre as disciplinas de enfileiramento hierárquicas disponíveis, o *Hierarchical Token Bucket* (HTB) destacou-se por permitir a criação de estruturas de classes com garantias de banda e prioridades distintas (DEVERA, 2002). O HTB possibilitou definir uma taxa mínima garantida (`rate`) para cada classe de tráfego, um teto máximo (`ceil`) que podia ser utilizado caso houvesse banda ociosa (empréstimo de banda), e uma prioridade (`prio`) que influenciava a alocação da banda excedente. Sua natureza hierárquica foi particularmente útil para implementar políticas de QoS complexas com subdivisões de banda (DEVERA, 2002).

Para o gerenciamento ativo de filas (AQM) e promoção de justiça entre fluxos, a disciplina *Fair Queuing with Controlled Delay* (FQ-CODEL) combinou duas abordagens modernas (HOEILAND-JOERGENSEN et al., 2018). O componente *Fair Queuing* (FQ) visou garantir a equidade na distribuição de banda entre fluxos concorrentes, isolando-os para prevenir que fluxos agressivos prejudicassem os interativos. Já o *Controlled Delay* (CoDel) foi um algoritmo AQM projetado para combater o *bufferbloat* – o fenômeno de latência excessiva induzida por buffers de rede muito grandes (GETTYS, 2011; NICHOLS; JACOBSON, 2012). O CoDel monitorou o tempo de permanência dos pacotes na fila e induziu o descarte ou marcação ECN quando a latência excedia um limiar, mantendo as filas curtas e o atraso controlado (HOEILAND-JOERGENSEN et al., 2018).

Para impor limites estritos de taxa na entrada ou saída de uma interface, a disciplina *Token Bucket Filter* (TBF) foi utilizada como um policiador. Diferente dos moladores que suavizam o tráfego atrasando pacotes, o TBF permitiu rajadas (*bursts*) limitadas pelo tamanho do "balde de tokens", mas descartou imediatamente pacotes que excediam a taxa configurada quando não havia tokens disponíveis (ALMESBERGER, 2001). Essa característica o tornou adequado para garantir que o tráfego não ultrapassasse limites contratuais ou de capacidade na borda da rede.

A capacidade de configurar programaticamente essas diversas disciplinas via `tc`, seja manualmente ou através de controladores SDN que interagiam com switches virtuais ou físicos, foi fundamental para a implementação e avaliação de políticas de QoS

diferenciadas em redes modernas.

3.3 Cenários de teste

Este trabalho avaliou o impacto de diferentes técnicas de QoS na gestão de tráfego em redes definidas por software, com foco na mitigação da contenção por recursos em enlaces compartilhados. Foram definidos quatro cenários experimentais que simularam situações de competição por banda, permitindo observar e comparar o comportamento da rede em diferentes configurações: sem QoS, com priorização simples, com múltiplos fluxos em classes de serviço distintas e com a aplicação conjunta de policiamento e enfileiramento. Através desses testes, foi possível evidenciar a importância do controle de tráfego para garantir desempenho previsível, estabilidade e equidade no uso dos recursos da rede.

3.3.1 Cenário 1: Competição sem Qualidade de Serviço (QoS)

No primeiro cenário, o objetivo foi estabelecer uma linha de base para o comportamento de uma rede sem qualquer mecanismo de Qualidade de Serviço ativo. Este cenário de controle representou a competição pura por recursos e serviu como base de comparação para os cenários subsequentes.

3.3.1.1 Implementação

O ambiente experimental utilizou a topologia da Figura 4, com H1 e H3 atuando como clientes enviando tráfego para o servidor H4. Para garantir que o descarte de informações fosse prontamente observável em momentos de sobrecarga, o espaço de buffer temporário (fila) em cada interface dos switches foi configurado para o mínimo de 10 (dez) pacotes (via parâmetro `max_queue_size=10`). Esta configuração assegura que o excesso de dados seja descartado rapidamente, tornando a perda de pacotes uma manifestação clara do congestionamento, em vez de apenas um atraso induzido por buffer. Foi utilizado o *ryu-manager* com a aplicação `simple_switch_13`, que implementa um switch OpenFlow básico. Este controlador é responsável por garantir o encaminhamento correto dos pacotes entre os hosts através dos switches. O comando utilizado para iniciá-lo foi:

```
ryu-manager simple_switch_13
```

Os testes de tráfego, conduzidos com a ferramenta `iperf` por um período de 15 segundos, foram registrados a cada segundo. Para o TCP, foi gerado tráfego bulk (padrão do `iperf`), que tenta maximizar a utilização da banda. Para o UDP, foram enviadas taxas de dados constantes ($\approx 90\%$ da largura de banda no teste individual e $\approx 80\%$ da largura de banda nos testes de competição).

Inicialmente, realizou-se uma avaliação da qualidade da rede na ausência de tráfego de aplicação, medindo o tempo que os dados levavam para ir e voltar, a variação desse tempo (*jitter*) e a ocorrência de perdas. Essa medição de base serviu como referência para os cenários subsequentes. Em seguida, foram conduzidos testes específicos para cada protocolo. Para o TCP, foram simulados dois cenários: um em que apenas um fluxo de dados utilizava o caminho de comunicação (H1 → H4) e outro em que dois fluxos competiam simultaneamente (H1 → H4 vs H3 → H4). O mesmo padrão foi aplicado ao UDP. No cenário de competição para ambos os protocolos, os hosts foram instruídos a enviar dados em taxas que, somadas, excediam significativamente a capacidade do link. Ambos os hosts (H1 e H3) enviam 80% da largura de banda (8 Mbps), totalizando uma demanda de 16 Mbps sobre o link de 10 Mbps. Durante os testes de competição, as informações geradas pelos medidores de tráfego foram exibidas em tempo real e armazenadas para uma análise mais detalhada, permitindo a observação direta das reações de cada protocolo ao congestionamento.

3.3.2 Cenário 2: Priorização de Tráfego com Filas

Já no segundo cenário, o intuito foi demonstrar a eficácia de um mecanismo de filas de prioridade para proteger um fluxo de tráfego crítico (H1→H4) da contenção gerada por um fluxo de menor importância (H3→H4).

3.3.2.1 Implementação

O experimento utilizou a topologia da Figura 4 e todos os seus links configurados com largura de banda de 10 Mbps. O tráfego foi gerado por H1 (alta prioridade) e H3 (baixa prioridade) para o servidor H4. A duração total da observação foi de 40 segundos: o fluxo H3 iniciou em T = 0 s por 40 s, e o fluxo H1 iniciou em T = 10 s por 20 s (terminando em T = 30 s). Para o tráfego UDP, ambos os hosts tentaram enviar 15 Mbps (`-b 15m`) para forçar o congestionamento e testar os limites das filas. Para o TCP, foi usado tráfego bulk.

A infraestrutura de QoS, composta por duas filas de prioridades distintas, foi configurada diretamente na interface de saída do switch `s1` (`s1-eth4`) utilizando o utilitário de controle de tráfego do Linux, o `tc`. Essa abordagem foi escolhida por sua robustez e confiabilidade em ambientes de simulação. A inteligência para classificar e direcionar os pacotes para as filas corretas, no entanto, foi centralizada no controlador Ryu.

A configuração de QoS foi projetada para criar uma divisão de banda garantida de 80/20, com prioridades distintas.

a) Fila 1 (alta prioridade): Destinada ao tráfego de H1, foi configurada com uma banda garantida (`rate`) de 8 Mbps e prioridade (`prio`) alta (1). O teto (`ceil`)

foi mantido em 10 Mbps, permitindo que este fluxo utilize a capacidade ociosa do link.

b) Fila 2 (baixa prioridade): Destinada ao tráfego de H3, foi configurada com uma banda garantida (`rate`) de 2 Mbps e prioridade (`prio`) baixa (2). O teto foi definido em 10 Mbps, permitindo que este fluxo também ocupasse a totalidade da banda, mas apenas depois que as necessidades do fluxo de alta prioridade fossem atendidas.

O trecho de código na *Listing* 3.1 ilustra a criação dessa política:

Listing 3.1 – Comandos `tc` para configuração das filas de QoS.

```
# Cria a hierarquia de filas (qdisc) do tipo HTB
tc qdisc add dev s1-eth4 root handle 1: htb default 20

# Define a capacidade total do link
tc class add dev s1-eth4 parent 1: classid 1:1 htb rate 10mbit

# Fila 1 (H1): Garante 8Mbit, prioridade ALTA.
tc class add dev s1-eth4 parent 1:1 classid 1:10 htb rate 8mbit
   ceil 10mbit prio 1

# Fila 2 (H3): Garante 2Mbit, prioridade BAIXA.
tc class add dev s1-eth4 parent 1:1 classid 1:20 htb rate 2mbit
   ceil 10mbit prio 2
```

O controlador Ryu implementa uma lógica de switch L2 reativo, que inspeciona o primeiro pacote de cada novo fluxo. Ao detectar um pacote com origem em H1 (`10.0.0.1`) e destino H4 (`10.0.0.4`), ele instala uma regra OpenFlow no switch `s1` que instrui o switch a encaminhar todos os pacotes subsequentes desse fluxo para a Fila 1 (`OFPActionSetQueue(1)`). Similarmente, o tráfego de H3 para H4 é direcionado para a Fila 2. Ademais, para contornar limitações de performance do TCP em ambientes virtualizados, a funcionalidade de *TCP Segmentation Offload (TSO)* — um recurso que delega à placa de rede a tarefa de dividir grandes blocos de dados em segmentos TCP menores — foi desativada em todos os hosts, pois em ambientes virtualizados essa segmentação ocorre fora do controle do sistema operacional, fazendo com que ferramentas de monitoramento registrem tamanhos de pacotes incorretos e resultem em medições imprecisas de desempenho.

3.3.3 Cenário 3: Competição e Priorização de Múltiplos Fluxos

O terceiro cenário teve como fim analisar a competição entre três fluxos concorrentes (H1→H4, H2→H4 e H3→H4) e a eficácia da QoS em proteger um fluxo prioritário, enquanto os outros dois compartilhavam uma mesma fila de baixa prioridade.

3.3.3.1 Implementação

Este experimento também utilizou a topologia da Figura 4 com seus links configurados em 10 Mbps. O objetivo foi analisar a competição entre três fluxos concorrentes (H1 → H4, H2 → H4, H3 → H4) e a eficácia da QoS em proteger H2 (alta prioridade), enquanto H1 e H3 (baixa prioridade) competiam entre si. A duração total foi de 50 segundos: H1 iniciou em T = 0 s por 50 s; H2 iniciou em T = 10 s por 30 s; H3 iniciou em T = 20 s por 20 s. Todos os fluxos UDP buscaram transmitir 15 Mbps para saturar as filas, ao passo que, para o TCP, foi usado tráfego bulk.

Ao receber o primeiro pacote de um novo fluxo (*packet-in*), o controlador inspeciona os cabeçalhos da camada 3. Especificamente para o tráfego que atravessa o link de contenção (da porta de entrada de `s1` para a porta de saída 4, conectada a `s2`), ele implementa uma lógica de classificação baseada no endereço IP de origem. Conforme a política de QoS estabelecida, o tráfego originado em H2 (`10.0.0.2`) é identificado como prioritário. O controlador, então, instala proativamente uma regra de fluxo no switch `s1` instruindo-o a encaminhar todos os pacotes subsequentes deste fluxo para a Fila 1 (alta prioridade), utilizando a ação OpenFlow `OFPActionSetQueue(1)`. Similarmente, os fluxos originados em H1 (`10.0.0.1`) e H3 (`10.0.0.3`) são classificados como de baixa prioridade e direcionados para a Fila 2. Esta abordagem reativa-proativa garante que apenas o primeiro pacote de cada fluxo cause sobrecarga no controlador, com o restante do tráfego sendo processado na velocidade do hardware do switch.

3.3.4 Cenário 4: Policiamento de Ingresso com Priorização de Egresso

O quarto cenário validou um modelo de QoS mais robusto, combinando a limitação de taxa na entrada da rede (policiamento) com a priorização hierárquica na saída (enfileiramento), a fim de garantir uma performance estável e previsível.

3.3.4.1 Implementação

O experimento utilizou a topologia da Figura 4 e seus respectivos links configurados em 10 Mbps. Primeiramente, na porta de ingresso `s1-eth1`, conectada a H1, foi aplicado um policiador de tráfego. Utilizando um dispositivo virtual *Intermediate Functional Block* (IFB), todo o tráfego de entrada de H1 foi redirecionado e submetido a uma disciplina `tbf` (Token Bucket Filter) que limita a taxa de transmissão a **2 Mbps**. Qualquer tráfego

de H1 que excedesse este limite era descartado imediatamente na entrada, antes mesmo de ser processado para enfileiramento na saída.

Em segundo lugar, na porta de egresso `s1-eth4`, foi implementada uma hierarquia de filas com `tc htb`. O tráfego (já limitado) vindo de H1 foi direcionado a uma fila de alta prioridade (`prio 1`) com 2 Mbps garantidos. O tráfego de H2 foi direcionado a uma fila de média prioridade (`prio 2`) com 6 Mbps garantidos, e o tráfego de H3 a uma fila de baixa prioridade (`prio 3`) com os 2 Mbps restantes. O teste consistiu em iniciar o fluxo policiado de H1 e, em seguida, introduzir os fluxos competitivos de H2 e H3 para verificar a estabilidade e a eficácia da hierarquia de serviço.

A duração total foi de 50 segundos, com H1 ativo o tempo todo, H2 ativo de T = 10 s a T = 40 s, e H3 ativo de T = 20 s a T = 40 s. Os fluxos UDP buscaram transmitir 15 Mbps e os TCP, tráfego bulk.

3.3.5 Cenário 5: Comparativo de Políticas de QoS em Rede Corporativa Simulada

Por fim, o quinto cenário foi desenhado para simular uma rede corporativa sob condições de congestionamento, permitindo uma avaliação comparativa direta entre três políticas de enfileiramento distintas aplicadas a um ponto de gargalo comum: FIFO (First-In, First-Out) como linha de base, HTB (Hierarchical Token Bucket) representando a priorização explícita, e FQ-CODEL (Fair Queuing with Controlled Delay) como uma abordagem moderna de gerenciamento automático. O objetivo é quantificar o impacto de cada política sobre diferentes tipos de fluxos (dados, interativos, tempo real) sob contenção de banda.

3.3.5.1 Implementação

A metodologia envolveu a criação de uma topologia com cinco hosts clientes (H1 a H5) conectados a um switch (S1) via links de 100 Mbps. S1 conecta-se a um segundo switch (S2) através de um link de 100 Mbps. S2, por sua vez, conecta-se a um servidor central (H6) com um link de 1 Gbps. A topologia é ilustrada na Figura 4.

Um mix de tráfego é gerado pelos clientes em direção a H6 utilizando `iperf`, simulando uma rede corporativa real:

a) H1 (elephant): fluxo TCP bulk de longa duração.
b) H2 (misto): um fluxo TCP persistente + fluxos TCP curtos (mice flows).
c) H3 (tempo real): dois fluxos UDP simultâneos: VoIP (128 Kbps) e Vídeo (5 Mbps).
d) H4 (elephant): segundo fluxo TCP bulk.
e) H5 (mice): fluxos TCP curtos e intermitentes.

A política de QoS em teste (FIFO, HTB ou FQ-CODEL) foi aplicada na interface de egresso de S1 para S2 (`s1-eth6`), ponto onde a soma do tráfego dos clientes (até 500 Mbps) excedia a capacidade do link (100 Mbps). O controlador Ryu atuou como um switch L2 padrão. As métricas foram coletadas dos logs do `iperf` no servidor H6 durante os 60 segundos do experimento.

3.4 Justificativa e Fundamentação

A adoção do protocolo OpenFlow 1.3 justificou-se pela necessidade de suporte a múltiplas filas de saída nos switches, um requisito fundamental para a implementação efetiva de políticas de Qualidade de Serviço diferenciadas. Versões anteriores do protocolo não contemplavam esse recurso de forma adequada, limitando a granularidade do controle de tráfego (SEEGER et al., 2019). A capacidade de manipular múltiplas filas permitiu a priorização de fluxos, o isolamento de classes de serviço e a garantia de requisitos específicos de largura de banda, atraso e perda, elementos centrais na provisão de QoS ponta a ponta em ambientes heterogêneos.

Nesse contexto, o paradigma de Redes Definidas por Software representou uma evolução significativa em relação ao modelo tradicional baseado em planos de controle distribuídos. A centralização do controle viabilizada por controladores como o Ryu ofereceu uma abstração lógica unificada da topologia e do estado da rede, permitindo a aplicação de políticas globais e a resposta adaptativa a eventos de rede em tempo real. Como apontado por Casado et al. (2007), essa centralização rompeu com as limitações dos protocolos tradicionais de roteamento e comutação, enquanto Kreutz et al. (2015) reforçaram a importância dessa abordagem para a construção de redes programáveis, dinâmicas e gerenciáveis de forma escalável.

A integração entre o controle centralizado e os mecanismos de enfileiramento e policiamento no plano de dados permitiu a definição e a aplicação de políticas de QoS orientadas por aplicações, fluxos ou usuários, o que se alinhou diretamente às demandas contemporâneas de redes multisserviço e ambientes com requisitos rigorosos de desempenho, como datacenters, redes 5G e infraestruturas críticas (SALMAN et al., 2017). Portanto, a escolha por OpenFlow 1.3, associada ao uso do Ryu e à configuração de políticas de QoS no plano de dados, configurou-se como uma decisão tecnicamente fundamentada e alinhada às melhores práticas na engenharia de redes orientada a desempenho.

4 Resultados e discussões

Este estudo aprofunda a compreensão sobre o comportamento dos protocolos de transporte TCP e UDP em redes de computadores, especialmente sob condições de congestionamento. Através de um experimento controlado, foi possível simular cenários de tráfego variado, desde a utilização plena de um recurso de rede até situações de severa sobrecarga, permitindo uma análise comparativa de suas reações e implicações para a qualidade da comunicação.

4.1 Cenário 1: Competição sem Qualidade de Serviço (QoS)

4.1.1 Tráfego UDP

A análise do UDP revelou sua natureza não orientada à conexão e sem controle de congestionamento. Em um único fluxo (H1 enviando com uma taxa-alvo de aproximadamente 90% da largura de banda (`udp_rate_no_comp = f'int(LINK_BANDWIDTH * 0.9)M'`), a vazão foi de 9,44 Mbps, com um jitter de 0,023 ms e sem perdas. Esta pequena variação entre a taxa de envio configurada e a vazão aferida é comum em testes `iperf` sobre UDP, sendo atribuída a pequenas diferenças nos intervalos de medição e arredondamentos da própria ferramenta. Contudo, no cenário de competição, com H1 e H3 transmitindo em média 80% da largura de banda cada um (a título exemplar, `udp_rate_comp_h1 = f'int(LINK_BANDWIDTH * 0.8)M'`), o protocolo continuou sem redução, resultando em descarte massivo devido à sobrecarga e à fila limitada.

Detalhadamente, no cenário de competição, o host H1 obteve uma vazão efetiva de 3,46 Mbps, com jitter de 15,123 ms e uma perda de 58% (6216 pacotes perdidos de 10701 enviados). H3, por sua vez, alcançou 6,21 Mbps de vazão efetiva, com jitter de 0,286 ms e perda de 26% (2774 pacotes perdidos de 10700 enviados). A soma das vazões recebidas (aproximadamente 9,67 Mbps) indica a saturação do link, porém com alta perda. As Tabelas 4 e 5 ilustram o envio de dados por intervalo, enquanto as Tabelas 6 e 7 detalham as informações do servidor sobre perdas de pacotes e datagramas enviados.

Tabela 4 – Vazão e transferência enviada por H1 no cenário de competição para o UDP.

Intervalo (s)	**Transferência**	**Vazão Média (Mbps)**
0-15	15,0 MBytes	8,39

Tabela 5 – Vazão e transferência enviada por H3 no cenário de competição para o UDP.

Intervalo (s)	**Transferência**	**Vazão Média (Mbps)**
0-15	15,0 MBytes	8,39

Tabela 6 – Relatório do Servidor H4 para o tráfego recebido de H1 no cenário de competição para o UDP.

Intervalo (s)	0-15
Transferência recebida	6,29 MBytes
Vazão recebida (Mbps)	3,46
Jitter (ms)	15,123
Pacotes Perdidos	6216/10701
Perda (%)	58

Tabela 7 – Relatório do Servidor H4 para o tráfego recebido de H3 no cenário de competição para o UDP.

Intervalo (s)	0-15
Transferência recebida	11,1 MBytes
Vazão recebida (Mbps)	6,21
Jitter (ms)	0,286
Pacotes Perdidos	2774/10700
Perda (%)	26

4.1.2 Tráfego TCP

Em contraste, ao analisar o comportamento do TCP, a natureza adaptativa e cooperativa do protocolo diante do congestionamento tornou-se evidente. No cenário em que apenas um fluxo de dados TCP era transmitido, o host H1 conseguiu utilizar quase a totalidade da largura de banda disponível, atingindo uma vazão de 9,71 Mbps. Este resultado demonstra a eficiência do TCP em explorar e aproveitar ao máximo os recursos da rede quando não há concorrência. Quando os dois fluxos TCP foram iniciados concomitantemente, ambos buscando o mesmo caminho de comunicação, o protocolo demonstrou a eficácia de seu mecanismo no controle de congestionamento. O host H1 obteve uma vazão média de 6,21 Mbps, enquanto o host H3 alcançou 3,56 Mbps. A soma dessas vazões (aproximadamente 9,77 Mbps) corresponde à capacidade total do link.

Diferente do UDP, o `iperf` não reportou perdas de pacotes (0%) ou jitter significativo para o TCP, o qual induz o aumento da latência como sinal para seu mecanismo de controle de congestionamento e, porventura, ocasionando as retransmissões de segmentos, um evento gerenciado internamente pelo protocolo para garantir a entrega confiável e não reportado como perda final pela aplicação. A métrica principal do TCP é a vazão adaptativa cujo ajuste dinâmico preenche o link sem o descarte massivo visto no UDP. As Tabelas 8 e 9 ilustram o envio de dados ao longo do intervalo de teste.

Tabela 8 – Desempenho de H1 no cenário de competição para o TCP.

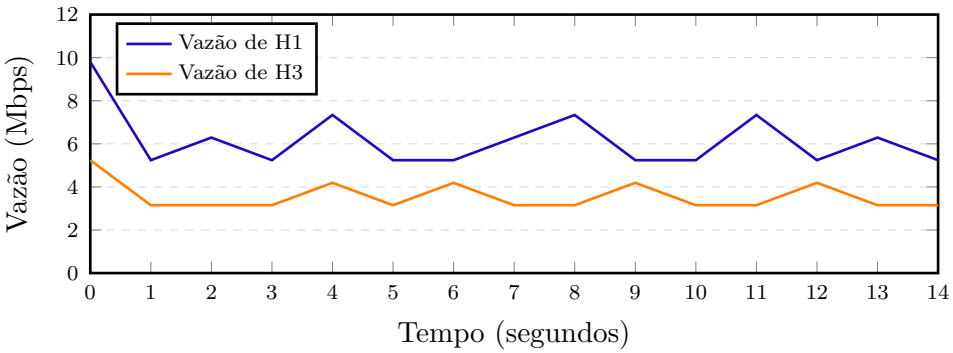
Transferência Total	**Vazão Média (Mbps)**
11,1 MBytes	6,21

Tabela 9 – Desempenho de H3 no cenário de competição para o TCP.

Transferência Total	**Vazão Média (Mbps)**
6,38 MBytes	3,56

Essa adaptação é fundamental para o TCP, pois ele prioriza a entrega confiável dos dados, ajustando sua taxa para evitar perdas e manter a estabilidade da rede. A Figura 5 evidencia a divisão equitativa da vazão entre H1 e H3 ao longo do tempo.

Figura 5 – Vazão de H1 e H3 ao longo do tempo em cenário de competição para o TCP.



Em síntese, a comparação entre TCP e UDP sob as mesmas condições de congestionamento ilustra suas filosofias intrinsecamente distintas. O TCP comporta-se com cooperatividade na rede; ele detecta o congestionamento e reduz proativamente sua taxa de envio, priorizando a entrega confiável e a justiça no uso dos recursos, mesmo que isso signifique uma vazão individual menor. Essa abordagem visa evitar o colapso da rede e garantir que todos os fluxos possam coexistir de forma estável. Por outro lado, o UDP opera de maneira não responsiva; ele prioriza a velocidade e a baixa latência, enviando dados continuamente na taxa solicitada pela aplicação, sem mecanismos internos para detectar ou reagir ao congestionamento. Em um ambiente sobrecarregado, essa agressividade resulta em um descarte massivo de pacotes, comprometendo a integridade da informação, mas mantendo a tentativa de alta vazão.

Este experimento fornece uma base empírica sólida para compreender as implicações da escolha do protocolo de transporte. Os resultados obtidos, que demonstram claramente a redução adaptativa da vazão no TCP e a perda explícita de pacotes no UDP em condições de sobrecarga, são fundamentais para o desenvolvimento de sistemas de rede robustos. Eles ressaltam a necessidade de considerar minuciosamente os requisitos de confiabilidade e sensibilidade ao atraso de cada aplicação, e reforçam a importância de

implementar mecanismos de QoS para gerenciar o tráfego de protocolos não adaptativos como o UDP em redes propensas a congestionamento.

4.2 Cenário 2: Priorização de Tráfego com Filas

4.2.1 Tráfego UDP

A análise dos dados coletados no experimento com tráfego UDP demonstra a capacidade do sistema, implementado via `tc htb` com filas `pfifo` de tamanho limitado, de impor uma hierarquia estrita em um cenário de contenção severa. Diferentemente do TCP, que possui mecanismos intrínsecos de controle de fluxo e congestionamento, o protocolo UDP envia datagramas à taxa especificada pela aplicação, independentemente da capacidade da rede. Essa característica torna a atuação do agendador de pacotes ainda mais crítica, pois ele se torna o único mecanismo responsável por gerenciar o congestionamento, decidindo ativamente quais pacotes descartar.

Nos primeiros 10 segundos do teste, apenas o fluxo de baixa prioridade (H3 → H4), configurado para transmitir a 15 Mbps, estava ativo. Os resultados, detalhados na Tabela 10, mostram que o fluxo alcançou uma vazão média de aproximadamente 9,72 Mbps, consumindo praticamente toda a capacidade do link de 10 Mbps. Isso evidencia a propriedade de "empréstimo" de banda do *Hierarchical Token Bucket* (HTB), disciplina de enfileiramento do Linux para controle de tráfego, que permite a um fluxo exceder sua garantia (`rate`) quando há capacidade ociosa na rede. Contudo, como a taxa de envio (15 Mbps) era superior à capacidade do link, o excedente foi descartado, resultando em uma perda de pacotes média de 35%, valor consistente com a perda teórica esperada de (15-10)/15 ≈ 33,3%. Simultaneamente, a latência da rede, medida pelo *ping*, denotada na Tabela 12, permaneceu extremamente baixa, na casa de 0,08 ms, indicando que o pequeno buffer não estava sendo sobrecarregado a ponto de gerar atrasos significativos.

Tabela 10 – Fluxo de baixa prioridade (H3 → H4) recebido no servidor H4.

Fase (Intervalo s)	**Vazão Média (Mbps)**	**Jitter Médio (ms)**	**Perdidos**	**Total**	**Perda (%)**
1 (0-10)	9,72	0,330	4483	12754	35
2 (10-30)	1,94	0,320	22199	25529	87
3 (30-40)	9,71	0,398	4517	12905	35

Tabela 11 – Fluxo de alta prioridade (H1 → H4) recebido no servidor H4.

Fase (Intervalo s)	**Vazão Média (Mbps)**	**Jitter Médio (ms)**	**Perdidos**	**Total**	**Perda (%)**
2 (10-30)	7,78	0,101	12284	25501	48

A segunda fase, iniciada no instante 10 s com a ativação do fluxo de alta prioridade (H1 → H4), também enviando a 15 Mbps, desencadeou uma contenção agressiva, com

Tabela 12 – Medições de latência de H1 para H4.

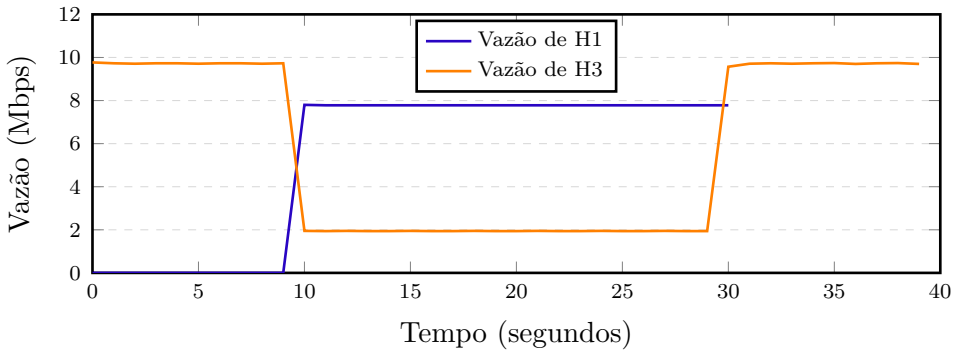
Sequência ICMP	**Fase do Teste**	**Latência (ms)**
1-10	Sem Contenção	0,044 - 0,168 (média ≈ 0,08)
11-30	Com Contenção	14,7 - 15,1 (média ≈ 15,0)
31-45	Sem Contenção	0,051 - 0,113 (média ≈ 0,07)

uma demanda agregada de 30 Mbps sobre o link de 10 Mbps. A resposta da política de QoS foi imediata e precisa, conforme os dados das Tabelas 10 e 11. O fluxo prioritário (H1) foi protegido e estabilizou-se em uma vazão média de 7,78 Mbps, muito próxima de sua garantia de 8 Mbps. A perda de pacotes para este fluxo foi de aproximadamente 48%, refletindo o descarte do tráfego que excedia sua classe de serviço ((15-8)/15 ≈ 46,6%). Em contrapartida, o fluxo de baixa prioridade (H3) sofreu uma queda severa e controlada, sendo confinado à sua banda garantida de 2 Mbps. Sua vazão caiu para 1,94 Mbps, e a perda de pacotes cresceu para 87%, em linha com a perda teórica esperada de (15-2)/15 ≈ 86,6%. O aspecto mais revelador desta fase, contudo, foi o impacto na latência. Os dados da Tabela 12 mostram que o tempo de resposta do *ping* saltou de frações de milissegundos para uma média de 15 ms. Este é um exemplo clássico do fenômeno de *bufferbloat*: a intensidade do tráfego era tão alta que os pacotes de *ping* foram enfileirados e atrasados, aguardando o envio dos pacotes UDP.

A terceira e última fase começou no instante 30 s, com o término do fluxo de H1. A Tabela 10 ilustra a alocação dinâmica de recursos do HTB: a vazão média de H3 saltou instantaneamente de 1,94 Mbps para 9,71 Mbps, recuperando toda a capacidade do link. A perda de pacotes retornou ao patamar inicial de aproximadamente 35%, e a latência (Tabela 12) caiu imediatamente para os níveis nominais de 0,07 ms, demonstrando que o congestionamento e o *bufferbloat* cessaram assim que a contenção foi resolvida.

A Figura 6 ilustra o comportamento temporal das vazões de H1 e H3 ao longo do experimento.

Figura 6 – Vazão de H1 e H3 ao longo do tempo.



A questão do dimensionamento do buffer é crucial para a performance de redes. O valor utilizado (`pfifo limit 10`), que corresponde a aproximadamente 15 KB, foi intencionalmente pequeno para exacerbar o efeito da perda de pacotes para fins demonstrativos. Um buffer mais "próximo da realidade" deve ser dimensionado com base no *Bandwidth-Delay Product* (BDP). Utilizando os dados observados no próprio experimento:

$$BDP = \text{Largura de Banda} \times \text{Atraso de Ida e Volta (RTT)} \tag{4.1}$$

Substituindo os valores, temos:

$$BDP = (10 \times 10^6 \text{ bits/s}) \times (15 \times 10^{-3} \text{ s}) \tag{4.2}$$

$$BDP = 150000 \text{ bits} \tag{4.3}$$

Convertendo para bytes:

$$150,000 \text{ bits}/8 = 18,750 \text{ bytes} \approx 18,3KB \tag{4.4}$$

Este cálculo demonstra que o buffer de 15 KB utilizado foi propositalmente menor que o BDP, o que garantiu a visibilidade das perdas de pacotes. A adoção de um buffer de 60 KB (aproximadamente 40 pacotes), por exemplo, resultaria em uma menor porcentagem de perdas, ao custo de um aumento na latência, ilustrando o trade-off fundamental no gerenciamento de filas.

Em suma, o experimento valida que a política de QoS implementada é capaz de gerenciar tráfego UDP de forma eficaz, aplicando uma hierarquia de serviço que garante a performance de fluxos críticos através do descarte controlado de pacotes de fluxos não prioritários, ao mesmo tempo que revela a relação intrínseca entre o tamanho do buffer, a perda de pacotes e a latência induzida.

4.2.2 Tráfego TCP

A transição do experimento para o protocolo TCP revela a interação entre a política de QoS da camada de rede e os mecanismos de controle de congestionamento da camada de transporte. Enquanto a disciplina `tc htb` continua a impor as mesmas regras de priorização e limitação de banda, a reação do TCP ao congestionamento é fundamentalmente diferente da do UDP, substituindo a perda massiva de pacotes por uma adaptação dinâmica da taxa de transmissão.

Na primeira fase do teste (0-10 s), o fluxo de baixa prioridade (H3 → H4) operou sem contenção. Conforme os dados da Tabela 13, o fluxo rapidamente utilizou toda a

capacidade disponível do link, estabilizando-se em uma vazão média de 9,55 Mbps. Este comportamento espelha o do UDP, confirmando que o agendador HTB permite que um fluxo aloque banda ociosa até o máximo da classe pai. A latência, medida pelo *ping* e sintetizada na Tabela 15, permaneceu em um nível nominal de aproximadamente 0,09 ms, indicando uma rede sem congestionamento.

Tabela 13 – Fluxo de baixa prioridade (H3 → H4) recebido no servidor H4.

Fase (Intervalo s)	**Transferência (MBytes)**	**Vazão Média (Mbps)**
1 (0-10)	11,38	9,55
2 (10-30)	4,56	1,91
3 (30-40)	11,33	9,51

A segunda fase (10-30 s) introduziu o fluxo de alta prioridade (H1 → H4), gerando contenção no link. A resposta da política de QoS foi, novamente, imediata e precisa. O fluxo prioritário (H1), detalhado na Tabela 14, manteve uma vazão notavelmente estável de 7,66 Mbps, consistente com sua garantia de 8 Mbps. Em contrapartida, o fluxo de baixa prioridade (H3) foi confinado à sua garantia, com a vazão caindo para 1,91 Mbps. A diferença crucial em relação ao UDP é a ausência de perda de pacotes reportada pelo `iperf`. Em vez de descartar pacotes, o algoritmo de controle de congestionamento do TCP detectou o preenchimento do buffer (evidenciado pelo aumento do RTT) e reduziu proativamente a janela de envio para se adequar à capacidade alocada pela QoS. O impacto na latência, no entanto, foi similar ao do UDP, com o RTT do *ping* saltando para uma média de 16,5 ms, evidência de *bufferbloat*, em que os pacotes de ambos os fluxos e do *ping* são enfileirados, aumentando o tempo de trânsito.

Na fase final (30-40 s), com o término do fluxo de H1, o TCP do fluxo H3 detectou a disponibilidade de banda e, através de seu mecanismo de *slow start* e *congestion avoidance*, rapidamente expandiu sua janela de transmissão. A Tabela 13 mostra que a vazão retornou ao patamar de ≈ 9,51 Mbps. A latência, por sua vez, voltou imediatamente aos níveis nominais, confirmando o fim do congestionamento.

Tabela 14 – Fluxo de alta prioridade (H1 → H4) recebido no servidor H4.

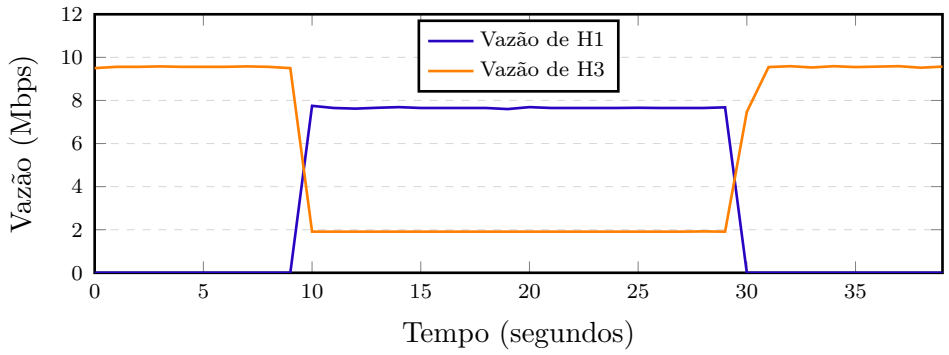
Fase (Intervalo s)	**Transferência (MBytes)**	**Vazão Média (Mbps)**
2 (10-30)	18,66	7,66

Tabela 15 – Medições de latência de H1 para H4.

Sequência ICMP	**Fase do Teste**	**Latência (ms)**
1-10	Sem Contenção	0,058 - 0,260 (média ≈ 0,09)
11-31	Com Contenção	12,5 - 19,4 (média ≈ 16,5)
32-45	Sem Contenção	0,034 - 0,085 (média ≈ 0,07)

A Figura 7 ilustra o comportamento temporal das vazões de H1 e H3 ao longo do experimento.

Figura 7 – Vazão de H1 e H3 ao longo do tempo.



A comparação direta dos resultados obtidos nos dois cenários de teste expõe as diferenças intrínsecas entre os protocolos da camada de transporte e como eles interagem com uma política de QoS na camada de rede. A distinção mais evidente reside na forma como cada protocolo lida com o congestionamento. O UDP, sendo um protocolo não orientado à conexão e sem garantia de entrega, transmite datagramas à taxa definida pela aplicação. Quando o buffer do agendador `pfifo` se esgota, a única ação possível é o descarte de pacotes, o que se reflete diretamente em uma alta taxa de perda nos relatórios do `iperf`. O TCP, por outro lado, é um protocolo confiável e orientado à conexão que implementa algoritmos sofisticados de controle de congestionamento. Ao detectar o aumento do RTT - um indicativo de que os pacotes estão sendo enfileirados -, o TCP interpreta isso como um sinal de congestionamento iminente e reduz sua janela de transmissão, diminuindo a taxa de envio para se adequar à capacidade real da rede. O resultado é a ausência de perda de pacotes do ponto de vista da aplicação, ao custo de uma vazão dinamicamente ajustada.

Sob a mesma política de QoS, as métricas-chave de performance divergem. Para o UDP, os indicadores críticos foram a vazão e a perda de pacotes. A QoS atuou permitindo que uma certa vazão passasse e descartou o restante. Para o TCP, a métrica principal foi a vazão. A QoS atuou informando ao TCP qual era a velocidade segura, e o protocolo se ajustou a ela. A "perda" para o TCP se manifesta como retransmissões de segmentos, um evento gerenciado internamente pelo protocolo e não reportado como perda final pela aplicação.

Ambos os experimentos demonstraram de forma conclusiva que o *bufferbloat* é um fenômeno da camada de rede, que afeta indiscriminadamente os protocolos da camada de transporte. Tanto no cenário TCP quanto no UDP, a latência do *ping* aumentou em mais de 150 vezes durante a contenção (de $\approx$ 0,1 ms para $\approx$ 16 ms). Isso ocorre porque os pacotes ICMP do *ping*, sendo pequenos e de baixa frequência, são enfileirados atrás da

"massa" de pacotes de dados (TCP ou UDP) que saturam o buffer, independentemente do protocolo que os gerou.

Em conclusão, os testes validam que a política de QoS é eficaz em impor a hierarquia de serviço para ambos os protocolos. Não obstante, a interação com o protocolo de transporte define o resultado final para a aplicação: o UDP sacrifica a confiabilidade (perda de pacotes) para manter uma taxa de envio constante, enquanto o TCP sacrifica a taxa de envio (vazão) para garantir a confiabilidade (entrega sem perdas).

4.3 Cenário 3: Competição e Priorização de Múltiplos Fluxos

4.3.1 Tráfego UDP

4.3.1.1 Divisão de Banda em Fila Única

O primeiro teste com UDP estabelece uma linha de base para o comportamento de múltiplos fluxos não confiáveis em uma rede congestionada e sem prioridades. Uma única fila de 10 Mbps foi configurada, e três fluxos UDP, cada um tentando transmitir a 15 Mbps, foram introduzidos progressivamente.

Os resultados revelam um comportamento instável. Na primeira fase (0-10 s), H1, sozinho, saturou o link, alcançando 9,75 Mbps com uma perda de pacotes previsível de 35%, conforme visto na Tabela 16. A entrada de H2 na segunda fase (10-20 s) não resultou em uma divisão justa; em vez disso, H1 manteve sua vazão, enquanto H2 sofreu perdas elevadas (chegando a 98% no primeiro segundo), demonstrando a ineficácia de uma fila FIFO simples em gerenciar múltiplos fluxos UDP, observado na Tabela 17. A terceira fase (20-40 s), com três competidores, intensificou o não determinismo, com as vazões de todos os fluxos se tornando altamente instáveis e as perdas de pacotes crescendo para mais de 90% para os fluxos mais novos (Tabela 18). Este cenário demonstra que, sem QoS, a competição entre fluxos UDP resulta em uma performance imprevisível e degradada para todos os envolvidos.

Tabela 16 – Fluxo H1 → H4 recebido em H4 (competição sem QoS).

Intervalo (s)	**Vazão (Mbps)**	**Jitter (ms)**	**Perdidos**	**Total**	**Perda (%)**
0-10	9,75	0,404	4482	12767	35
10-20	9,72	0,402	4489	12756	35
20-29	6,84	0,252	7073	12758	55
29-30	1,20	0,039	1174	1276	92
30-40	1,32	0,052	11634	12781	91
40-50	9,71	0,404	4492	12753	35

Tabela 17 – Fluxo H2 → H4 recebido em H4 (competição sem QoS).

Intervalo (s)	**Vazão (Mbps)**	**Jitter (ms)**	**Perdidos**	**Total**	**Perda (%)**
10-11	5,35	0,128	24360	24815	98
12-20	5,51	0,145	7287	11482	63
20-40	5,52	0,115	807	1276	63

Tabela 18 – Fluxo H3 → H4 recebido em H4 (competição sem QoS).

Intervalo (s)	**Vazão (Mbps)**	**Jitter (ms)**	**Perdidos**	**Total**	**Perda (%)**
20-40	2,97	0,094	20465	25511	80

4.3.1.2 Priorização de Fluxo com QoS

Este teste introduz a política de QoS para gerenciar os mesmos três fluxos UDP. H2 foi designado como alta prioridade com 6 Mbps garantidos, enquanto H1 e H3 foram designados como baixa prioridade, compartilhando os 4 Mbps restantes.

Os resultados demonstram um controle rigoroso. Na primeira fase, H1 (baixa prioridade) utilizou todo o link (9,73 Mbps com 35% de perda), vislumbrado na Tabela 19. Com a entrada de H2 na segunda fase, a QoS imediatamente protegeu o fluxo prioritário, consoante Tabela 20: H2 alcançou sua garantia de 5,83 Mbps (com 61% de perda, pois tentava enviar 15 Mbps) e H1 foi rebaixado para a banda restante de $\approx$ 3,9 Mbps (com 74% de perda). A terceira fase revelou o fenômeno de inanição de fluxo (*flow starvation*), visto na Tabela 21, para H3: 0 Mbps de vazão e 100% de perda de pacotes. O fluxo foi completamente suprimido pela contenção extrema dentro de sua classe de baixa prioridade, isso porque o buffer já estava cheio com os pacotes enviados por H2, denotando que a QoS para UDP, sob forte congestionamento, não apenas prioriza, mas pode erradicar fluxos não prioritários.

Tabela 19 – Fluxo H1 → H4 recebido em H4 (baixa prioridade).

Intervalo (s)	**Vazão (Mbps)**	**Jitter (ms)**	**Perdidos**	**Total**	**Perda (%)**
0-10	9,73	0,396	4464	12758	35
10-20	3,92	0,194	9387	12754	74
20-40	3,89	0,215	18898	25523	74
40-50	9,72	0,442	4799	12751	38

Tabela 20 – Fluxo H2 → H4 recebido em H4 (alta prioridade).

Intervalo (s)	**Vazão (Mbps)**	**Jitter (ms)**	**Perdidos**	**Total**	**Perda (%)**
10-40	5,83	0,372	23394	38266	61

Tabela 21 – Fluxo H3 → H4 recebido em H4 (baixa prioridade).

Intervalo (s)	**Vazão (Mbps)**	**Jitter (ms)**	**Perdidos**	**Total**	**Perda (%)**
20-40	0,00	0,000	12756	12756	100

4.3.2 Tráfego TCP

4.3.2.1 Divisão Justa de Banda

A análise dos resultados evidencia a natureza cooperativa do protocolo. Na primeira fase (0-10 s), H1 atingiu 9,57 Mbps. Na segunda fase (10-20 s), com a entrada de H2, a contenção fez com que ambos os fluxos convergissem para uma divisão equitativa de aproximadamente 4,78 Mbps cada. A terceira fase (20-40 s), com a entrada de H3, levou os três fluxos a uma divisão justa de aproximadamente 3,20 Mbps cada. Finalmente, H1 reocupou todo o link após o término dos competidores. A Figura 8 ilustra a dinâmica de compartilhamento estável e previsível, também descritas nas Tabelas 22, 23 e 24.

Tabela 22 – Fluxo H1 → H4 recebido em H4 (competição justa).

Intervalo (s)	**Transferência (MBytes)**	**Vazão (Mbps)**
0-10	11,40	9,57
10-20	5,83	4,78
20-40	7,77	3,19
40-50	10,83	9,08

Tabela 23 – Fluxo H2 → H4 recebido em H4 (competição justa).

Intervalo (s)	**Transferência (MBytes)**	**Vazão (Mbps)**
10-20	5,84	4,79
20-40	7,77	3,19

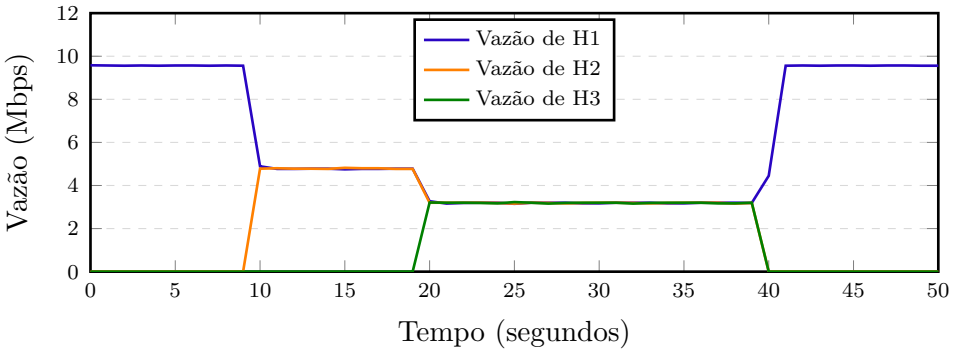
Tabela 24 – Fluxo H3 → H4 recebido em H4 (competição justa).

Intervalo (s)	**Transferência (MBytes)**	**Vazão (Mbps)**
20-40	7,79	3,20

4.3.2.2 Priorização de Fluxo com QoS

Por fim, aplica-se a mesma política de QoS (6 Mbps para H2, 4 Mbps para H1/H3) ao tráfego TCP. Os resultados demonstram uma interação eficaz entre a QoS da rede e o controle de congestionamento do TCP. Na primeira fase, H1 utilizou todo o link (9,57 Mbps). Com a entrada de H2 na segunda fase, a QoS garantiu a performance de H2 em 5,74 Mbps, enquanto H1 se ajustou para a banda restante de 3,83 Mbps. Na terceira fase, com H3, H2 permaneceu protegido, enquanto H1 e H3 cooperativamente dividiram a banda

Figura 8 – Vazão dos fluxos TCP no cenário de divisão justa.



de baixa prioridade - o primeiro alocou 1,91 Mbps e o segundo 1,96 Mbps. Notadamente, não houve perda de pacotes; o TCP reduziu sua velocidade em resposta aos sinais de congestionamento criados pela QoS, os resultados são denotados nas Tabelas 25, 26 e 27 bem como ao longo do tempo na Figura 9.

Tabela 25 – Fluxo H1 → H4 recebido em H4 (baixa prioridade).

Intervalo (s)	**Transferência (MBytes)**	**Vazão (Mbps)**
0-10	11,40	9,57
10-20	4,67	3,83
20-40	4,67	1,91
40-50	11,20	9,39

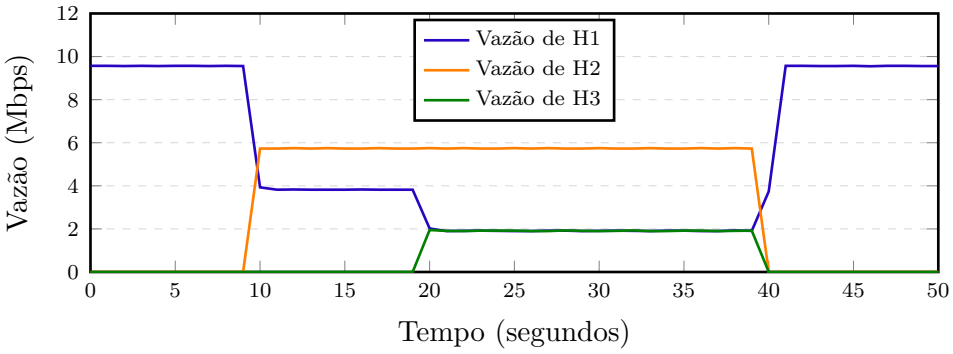
Tabela 26 – Fluxo H2 → H4 recebido em H4 (alta prioridade).

Intervalo (s)	**Transferência (MBytes)**	**Vazão (Mbps)**
10-40	20,90	5,74

Tabela 27 – Fluxo H3 → H4 recebido em H4 (baixa prioridade).

Intervalo (s)	**Transferência (MBytes)**	**Vazão (Mbps)**
20-40	4,88	1,96

Figura 9 – Vazão dos fluxos TCP no cenário com priorização de QoS.



A comparação direta entre as quatro conjunturas propostas expõe as diferenças intrínsecas entre os protocolos da camada de transporte e como eles interagem com uma política de QoS. A distinção mais evidente reside na forma como cada protocolo lida com o congestionamento. O UDP, sendo um protocolo não orientado à conexão, transmite datagramas à taxa definida pela aplicação. Quando o buffer do agendador se esgota, a única ação possível é o descarte de pacotes (*tail-drop*), o que se reflete em uma alta taxa de perda. No cenário *fair*, essa abordagem resultou em um comportamento imprevisível, onde a vazão dependia de um processo estocástico de um pacote encontrar espaço no buffer, invalidando qualquer noção de justiça. Mesmo que se aumentasse o tempo de transmissão de um fluxo, ele não ganharia prioridade, pois a fila FIFO não possui memória ou estado para tal; a instabilidade apenas persistiria.

O TCP, por outro lado, é um protocolo confiável que implementa algoritmos sofisticados de controle de congestionamento. Ao detectar o aumento do RTT, o TCP interpreta isso como um sinal de congestionamento e reduz sua janela de transmissão, diminuindo a taxa de envio para se adequar à capacidade da rede. No cenário *fair*, isso resultou em uma convergência estável e previsível para uma divisão de banda equitativa entre todos os competidores. O resultado, do ponto de vista da aplicação, é a ausência de perda de pacotes, ao custo de uma vazão dinamicamente ajustada.

A introdução da política de QoS demonstrou sua eficácia em impor uma hierarquia de serviço para ambos os protocolos, mas através de mecanismos distintos. Para o UDP, a QoS impôs os limites de banda, descartando ativamente os pacotes que excediam os limites de cada fila para garantir a banda do fluxo prioritário, chegando ao extremo de causar a inanição completa de um fluxo. Para o TCP, a QoS funcionou como um mecanismo de sinalização; ao limitar a banda, ela aumentou o RTT dos fluxos de baixa prioridade, sinalizando para seus algoritmos de controle de congestionamento que eles deveriam reduzir a velocidade, o que fizeram de forma cooperativa.

Em conclusão, os testes validam que a política de QoS é eficaz em impor a hierarquia de serviço para ambos os protocolos. Não obstante, a interação com o protocolo de transporte define o resultado final para a aplicação: o UDP sacrifica a confiabilidade (perda de pacotes) para tentar manter uma taxa de envio constante, resultando em desordenamento sem QoS; enquanto o TCP sacrifica a taxa de envio (vazão) para garantir a confiabilidade (entrega sem perdas), resultando em uma divisão justa, mas não necessariamente desejável, sem QoS. A combinação de SDN e QoS permite, portanto, gerenciar ambos os comportamentos, alinhando a alocação de recursos da rede com as prioridades operacionais.

4.4 Cenário 4: Policiamento de Ingresso com Priorização de Egresso

4.4.1 Tráfego UDP

Os resultados validam perfeitamente o funcionamento da arquitetura de QoS de duas etapas, revelando a diferença crucial entre as técnicas de policiamento e modelagem.

O fluxo de H1, submetido ao policiamento de ingresso, teve sua vazão rigidamente limitada a 1,94 Mbps, conforme a Tabela 28. Como o host tentou transmitir a 15 Mbps, a grande maioria dos seus pacotes foi descartada na entrada do switch, o que é refletido na taxa de perda de pacotes de 87% reportada pelo receptor. A performance de H1 permaneceu estável e completamente isolada, não sendo afetada pela entrada dos fluxos concorrentes, provando a eficácia do policiamento em conter um fluxo na borda da rede.

Os fluxos de H2 e H3 foram gerenciados pelo modelador de tráfego (*shaper*) na saída. Na segunda fase do teste (10-20 s), com H1 e H2 ativos, a QoS de egresso alocou os recursos de forma hierárquica: H1 recebeu seus 2 Mbps prioritários, e H2 pôde utilizar a banda restante, alcançando 7,78 Mbps. Com a entrada de H3 na terceira fase (20-40 s), a contenção total foi estabelecida: H1 manteve seus 2 Mbps, H2 foi confinado à sua garantia de 5,83 Mbps e H3 recebeu os 1,95 Mbps restantes.

O aspecto mais notável, detalhado nas Tabelas 29 e 30, é a taxa de 0% de perda de pacotes para H2 e H3. Este comportamento, que contrasta fortemente com o de H1, ocorre porque a modelagem de saída não descarta pacotes imediatamente. Em vez disso, ela os enfileira e os libera a uma taxa controlada. Este enfileiramento gera contrapressão (*backpressure*) que se propaga pela rede até o host remetente, cujo buffer de envio do kernel se esgota. Consequentemente, a própria aplicação `iperf` em H2 e H3 é desacelerada, sendo impedida de injetar mais pacotes na rede do que a fila de saída consegue processar. Assim, do ponto de vista do receptor, os pacotes que foram enviados chegaram sem perdas, pois o "descarte" efetivo ocorreu no próprio host emissor, que foi forçado a reduzir sua taxa de envio. A distribuição temporal dos resultados é vista na Figura 10.

Tabela 28 – Fluxo H1 → H4 recebido em H4 (policiado em 2 Mbps, alta prioridade).

Intervalo (s)	**Vazão (Mbps)**	**Jitter (ms)**	**Perdidos**	**Total**	**Perda (%)**
0-50	1,94	8,011	55477	63777	87

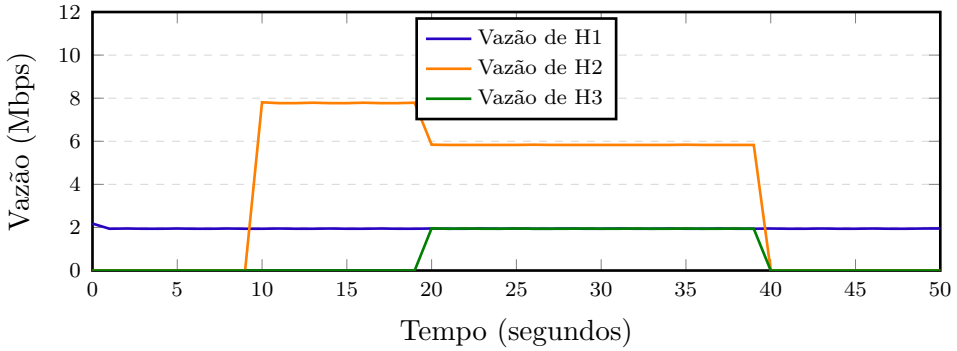
Tabela 29 – Fluxo H2 → H4 recebido em H4 (média prioridade).

Intervalo (s)	**Vazão (Mbps)**	**Jitter (ms)**	**Perdidos**	**Total**	**Perda (%)**
10-20	7,78	2,768	0	6616	0
20-40	5,83	4,220	0	9922	0

Tabela 30 – Fluxo H3 → H4 recebido em H4 (baixa prioridade).

Intervalo (s)	**Vazão (Mbps)**	**Jitter (ms)**	**Perdidos**	**Total**	**Perda (%)**
20-40	1,95	9,708	0	3357	0

Figura 10 – Vazão dos fluxos UDP no cenário de Policiamento e Priorização.



4.4.2 Tráfego TCP

O fluxo de H1, ao encontrar o policiador de 2 Mbps na entrada, sofreu uma perda inicial de pacotes. Seu algoritmo de controle de congestionamento interpretou isso como um sinal de congestionamento severo e rapidamente reduziu sua taxa de envio para se adequar ao limite. Como resultado, a Tabela 31 mostra que a vazão de H1 convergiu e se estabilizou em 1,92 Mbps por todo o teste. Do ponto de vista da aplicação, não há perda de pacotes, pois o TCP garantiu a retransmissão de tudo o que foi descartado pelo policiador.

Os fluxos de H2 e H3, gerenciados pela fila de saída, também se adaptaram de forma inteligente. Na segunda fase (10-20 s), a contrapressão da fila de saída sinalizou ao TCP de H2 que o caminho disponível era de aproximadamente 8 Mbps (10 Mbps total - 2 Mbps de H1), e ele se ajustou para 7,66 Mbps. Na terceira fase (20-40 s), com a contenção total, a hierarquia de QoS foi imposta: H1 manteve seus 2 Mbps; a contrapressão na fila de H2 o forçou a se ajustar à sua garantia de 6 Mbps, resultando em uma vazão de 5,74 Mbps; e a contrapressão na fila de H3 o forçou a se ajustar à banda restante, estabilizando em 1,91 Mbps. As Tabelas 32 e 33 detalham essa alocação, ao passo que a Figura 11 a denota no tempo.

Tabela 31 – Fluxo H1 → H4 recebido em H4 (policiado em 2 Mbps, alta prioridade).

Intervalo (s)	**Vazão Média (Mbps)**
0-50	1,92

Sob essa perspectiva, pode-se perceber que ambas as arquiteturas de QoS foram eficazes em impor a hierarquia de serviço, mas a interação com o protocolo de transporte definiu a qualidade e a natureza da comunicação.

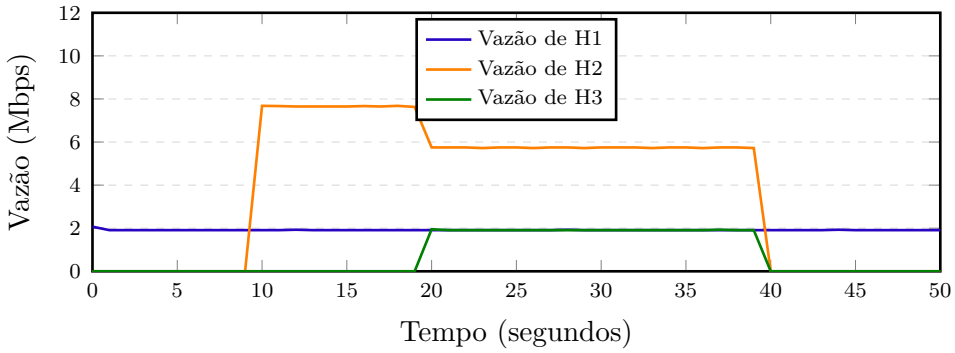
Tabela 32 – Fluxo H2 → H4 recebido em H4 (média prioridade).

Intervalo (s)	**Vazão Média (Mbps)**
10-20	7,66
20-40	5,74

Tabela 33 – Fluxo H3 → H4 recebido em H4 (baixa prioridade).

Intervalo (s)	**Vazão Média (Mbps)**
20-40	1.91

Figura 11 – Vazão dos fluxos TCP no cenário de Policiamento e Priorização.



O teste com UDP demonstrou a aplicação direta e restritiva do controle de tráfego. O policiamento em H1 resultou em uma perda de pacotes explícita e massiva (87%), uma consequência direta do descarte na entrada. A modelagem em H2 e H3, por outro lado, resultou em 0% de perda de pacotes, não porque o tráfego não foi limitado, mas porque a limitação ocorreu de forma implícita através da contrapressão, que forçou a desaceleração da aplicação no próprio host emissor. A QoS, para o UDP, atua como um mecanismo limitante que descarta ou atrasa pacotes.

O teste com TCP revelou uma interação muito mais coordenada. O TCP do H1, ao encontrar o policiador, interpretou a perda de pacotes como congestionamento e cooperativamente reduziu sua taxa de envio para se adequar ao limite, garantindo a entrega confiável sem perdas para a aplicação. Similarmente, os TCPs de H2 e H3 responderam à contrapressão do modelador de saída, ajustando suas janelas de congestionamento para corresponder exatamente às suas bandas garantidas. A QoS, para o TCP, atua como um mecanismo de sinalização, informando ao protocolo qual é a taxa de envio segura e o protocolo se adapta a ela.

Em conclusão, este cenário valida que a combinação de policiamento de ingresso e modelagem de egresso é uma arquitetura de QoS poderosa e granular. O policiamento é ideal para impor limites rígidos na borda da rede, enquanto a modelagem hierárquica é ideal para gerenciar o compartilhamento de recursos no núcleo. A escolha do protocolo de transporte determina o resultado final: com UDP, a QoS garante a banda através do

descarte, comprometendo a confiabilidade; com TCP, a QoS guia o protocolo para que ele mesmo garanta a banda, preservando a confiabilidade.

4.5 Cenário 5: Comparativo de Políticas de QoS em Rede Corporativa Simulada

4.5.0.1 FIFO (First-In, First-Out)

A análise do cenário FIFO, utilizando `pfifo limit 100`, estabeleceu a linha de base. Conforme esperado, a ausência de gerenciamento ativo da fila resultou em competição direta entre os fluxos.

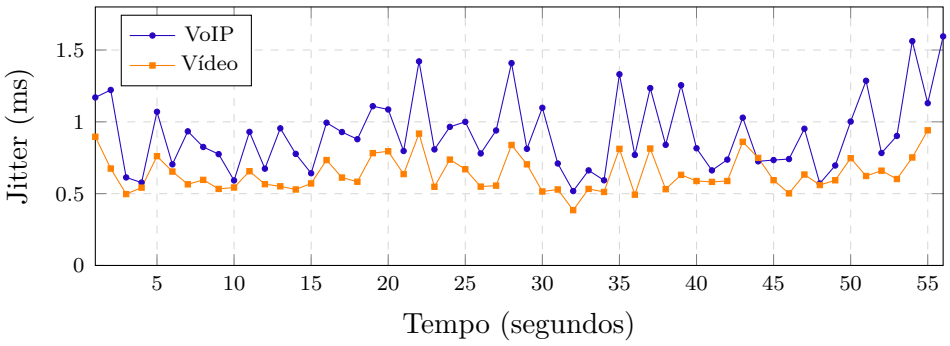
4.5.0.1.1 Tráfego UDP

Os resultados exibidos na Tabela 34 mostram 0% de perda de pacotes, indicando que o buffer não transbordou nesta execução. No entanto, o jitter foi significativo e variável, atingindo picos de 1,595 ms para VoIP e 0,941 ms para Vídeo, como visualizado na Figura 12. Este comportamento decorre do enfileiramento indiscriminado: pacotes UDP sensíveis à latência esperavam atrás de rajadas TCP, introduzindo atrasos variáveis (*bufferbloat*).

Tabela 34 – FIFO: Variação do *jitter* UDP (H3) recebido em H6.

Fluxo UDP	**Perda (%)**	**Jitter Médio (ms)**	**Jitter Máximo (ms)**
H3 (VoIP)	0%	≈ 0,9	1,595
H3 (Vídeo)	0%	≈ 0,6	0,941

Figura 12 – FIFO: Variação temporal do *jitter* UDP (H3).



4.5.0.1.2 Tráfego TCP

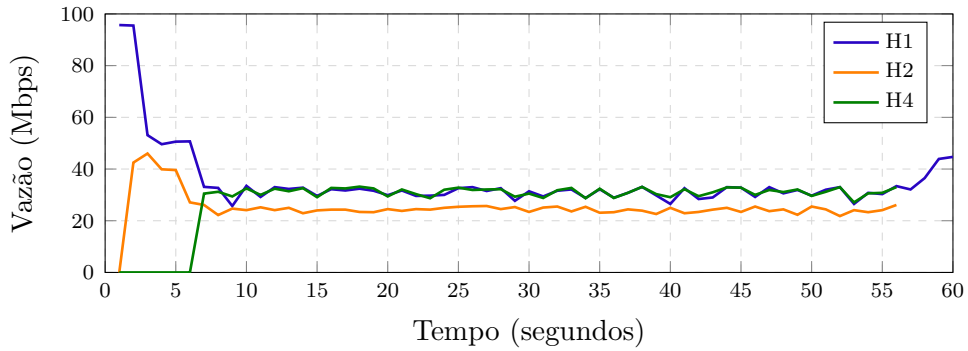
A Figura 13 ilustra a competição pela banda. Com a entrada de H2 e H4, a banda foi dividida de forma instável, com vazões médias finais de 35,0 Mbps (H1), 25,6 Mbps

(H2), e 31,8 Mbps (H4), conforme Tabela 35. Fluxos curtos (H5) sofreram alta latência. A análise via *wireshark* confirmou a ausência de retransmissões significativas, indicando degradação via latência.

Tabela 35 – FIFO: Vazão média TCP recebida em H6.

Fluxo TCP	**Vazão Média (Mbps)**
H1 (Bulk)	35,0
H2 (Persist.)	25,6
H4 (Bulk)	31,8

Figura 13 – FIFO: Vazão temporal dos fluxos TCP.



4.5.0.1.3 Considerações

Demonstrou ser inadequado para tráfego misto, falhando em proteger aplicações de tempo real (alto jitter) e penalizando fluxos interativos curtos (alta latência).

4.5.0.2 HTB (Hierarchical Token Bucket)

O cenário HTB implementou prioridades explícitas: VoIP (1 Mbps, prioridade 1), Vídeo (10 Mbps, prioridade 2), e TCPs (89 Mbps, prioridade 3), com FQ-CODEL gerenciando as filas internas.

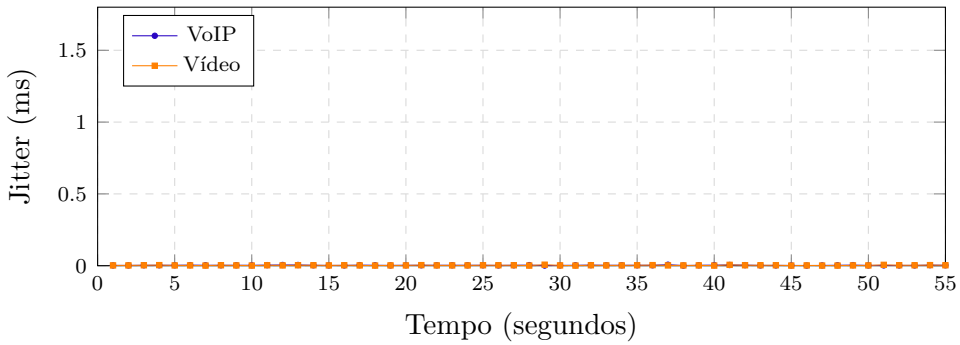
4.5.0.2.1 Tráfego UDP

Os resultados (Tabela 36) foram altamente positivos. Registrou-se 0% de perda e um jitter máximo de apenas 0,007 ms para ambos os fluxos. A Figura 14 mostra linhas essencialmente retas em zero, comprovando a eficácia da priorização.

Tabela 36 – HTB: Variação do *jitter* UDP (H3) recebido em H6.

Fluxo UDP	**Perda (%)**	**Jitter Médio (ms)**	**Jitter Máximo (ms)**
H3 (VoIP)	0%	0,003	**0,007**
H3 (Vídeo)	0%	0,002	**0,007**

Figura 14 – HTB: Variação temporal do *jitter* UDP (H3).



É mister destacar que o jitter observado decorre principalmente do intervalo de medição utilizado pelo `iperf`. Na prática, dada a baixa ordem de grandeza registrada - na faixa de microssegundos - esse valor é tão pequeno que não chega sequer a demandar tempo perceptível de processamento da CPU.

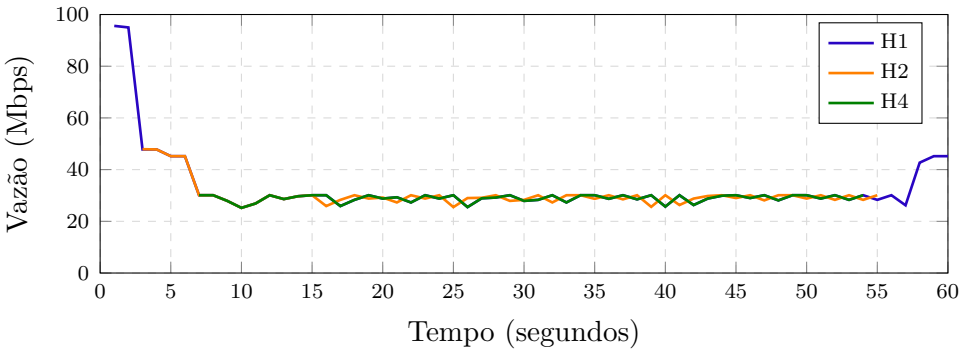
4.5.0.2.2 Tráfego TCP

Confinados à classe de baixa prioridade, os fluxos TCP competiram pela banda restante (≈ 94 Mbps). A Figura 15 mostra que H1, H2 e H4 dividiram essa capacidade de forma justa e estável, com vazões médias de 33,1 Mbps, 30,1 Mbps e 29,8 Mbps, respectivamente (Tabela 37).

Tabela 37 – HTB: Vazão média TCP recebida em H6.

Fluxo TCP	**Vazão Média (Mbps)**
H1 (Bulk)	33,1
H2 (Persist.)	30,1
H4 (Bulk)	29,8

Figura 15 – HTB: Vazão temporal dos fluxos TCP.



4.5.0.2.3 Considerações

Provou sua eficácia em impor garantias de QoS, protegendo de forma efetiva o tráfego prioritário ao custo de limitar o tráfego restante.

4.5.0.3 FQ-CODEL (Fair Queuing with Controlled Delay)

O cenário FQ-CODEL aplicou o algoritmo diretamente na interface, sem classes explícitas.

4.5.0.3.1 Tráfego UDP

Os resultados foram notavelmente bons, conforme a Tabela 38. Registrou-se 0% de perda e jitter médio extremamente baixo (0,074 ms VoIP, 0,039 ms Vídeo), com máximo de 0,084 ms. A Figura 17 mostra jitter baixo e estável. O Fair Queuing (FQ) isolou automaticamente os fluxos UDP dos TCPs agressivos.

Tabela 38 – FQ-CODEL: Variação do *jitter* UDP (H3) recebido em H6.

Fluxo UDP	**Perda (%)**	**Jitter Médio (ms)**	**Jitter Máximo (ms)**
H3 (VoIP)	0%	**0,074**	0,074
H3 (Vídeo)	0%	**0,039**	0,084

Figura 16 – FQ-CODEL: Variação temporal do *jitter* UDP (H3).

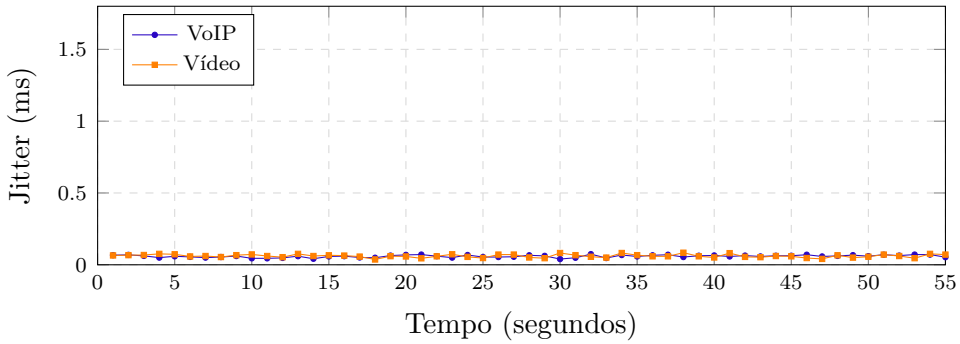


Figura 17 – Variação do Jitter UDP sob FQ-CODEL.

4.5.0.3.2 Tráfego TCP

A Figura 19 denota como os fluxos H1, H2 e H4 dividiram a banda de forma altamente equitativa, com vazões médias de 32,9 Mbps, 30,0 Mbps e 29,5 Mbps, respectivamente (Tabela 39). O CODEL manteve as filas curtas, beneficiando os fluxos *mice* (H5), que apresentaram altas vazões instantâneas (18-22 Mbps), indicando baixa latência.

Tabela 39 – FQ-CODEL: Vazão média TCP recebida em H6.

Fluxo TCP	**Vazão Média (Mbps)**
H1 (Bulk)	32,9
H2 (Persist.)	30,0
H4 (Bulk)	29,5

Figura 18 – FQ-CODEL: Vazão temporal dos fluxos TCP.

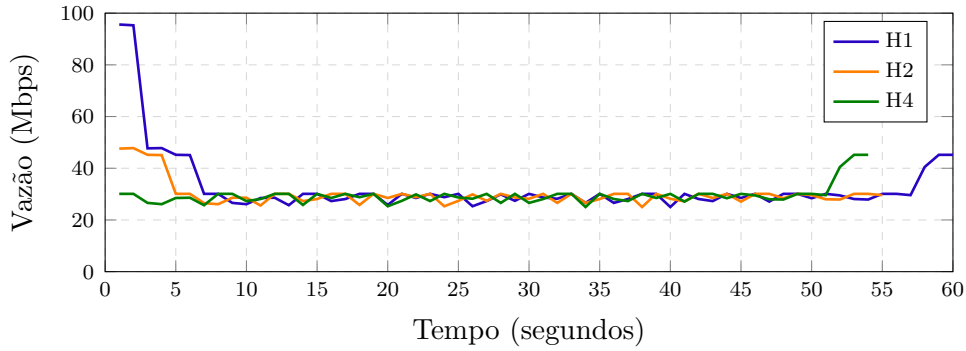


Figura 19 – Vazão TCP sob FQ-CODEL.

4.5.0.3.3 Considerações

Apresentou bom equilíbrio, oferecendo proteção de latência comparável ao HTB para UDP e partilha justa entre TCPs, de forma automática.

4.5.1 Comparativo Final

A análise experimental comparativa das disciplinas de enfileiramento validou os comportamentos teóricos esperados sob condições de congestionamento induzido. A implementação padrão FIFO, servindo como linha de base, exibiu deficiências significativas na gestão de tráfego heterogêneo. A ausência de mecanismos de isolamento ou priorização resultou em latência de enfileiramento substancial e variável (jitter) para os fluxos UDP em tempo real (VoIP e Vídeo), atingindo picos de 1,595 ms, característico do fenômeno de *bufferbloat*. Embora a perda de pacotes tenha sido evitada nesta instância específica, presumivelmente devido à resposta do controle de congestionamento TCP, a variabilidade de atraso e a penalização implícita a fluxos TCP curtos (*mices*) confirmam sua inadequação para redes convergentes modernas.

Em contraste, HTB demonstrou a eficácia do gerenciamento hierárquico baseado em classes e prioridades explícitas. Ao alocar os fluxos UDP em classes de alta e média prioridade com garantias de banda, o HTB os isolou completamente da contenção, resultando na eliminação quase total do jitter (máximo observado de 0,007 ms). O tráfego TCP remanescente foi confinado à classe de baixa prioridade, onde compartilhou a capacidade residual de forma equitativa ($\approx$ 30-33 Mbps por fluxo *elephant*), gerenciado internamente pelo FQ-CODEL anexado à classe. Este comportamento determinístico valida o HTB como a solução preferencial para ambientes que exigem garantias estritas de desempenho e cumprimento de Acordos de Nível de Serviço (SLAs).

Por fim, FQ-CODEL apresentou uma abordagem robusta de gerenciamento ativo de filas (AQM) sem a necessidade de configuração explícita de classes. Seu mecanismo in-

trínseco de enfileiramento justo (FQ) proveu isolamento efetivo entre os fluxos, resultando em *jitter* consistentemente baixo para o tráfego UDP (médias de 0,074 ms e 0,039 ms, máximo $\leq$ 0,1 ms). Adicionalmente, o FQ promoveu uma partilha de banda equitativa entre os fluxos TCP concorrentes ($\approx$ 30-33 Mbps cada). O componente de atraso controlado (CODEL) contribuiu para manter a latência geral baixa, beneficiando implicitamente os fluxos TCP interativos curtos (*mice*).

Em conclusão, a experimentação quantificou as vantagens substanciais do HTB e FQ-CODEL sobre o FIFO em cenários de rede congestionada. A escolha entre HTB e FQ-CODEL é determinada pelos requisitos operacionais: HTB oferece controle preciso e garantias explícitas mediante configuração detalhada, enquanto FQ-CODEL proporciona robusto desempenho geral, baixa latência e justiça de forma adaptativa e com mínima sobrecarga de configuração, tornando-o uma alternativa interessante para a maioria dos ambientes com tráfego misto.[1]

[1] Os códigos desenvolvidos e utilizados neste estudo estão disponíves no repositório: <https://github.com/arthurvdeassis/redes-sdn>.

5 Conclusão

Este trabalho apresentou uma análise experimental detalhada sobre o comportamento dos protocolos TCP e UDP em redes de computadores sob condições de congestionamento, avaliando o impacto de diferentes mecanismos de Qualidade de Serviço (QoS) implementados em um ambiente controlado utilizando Mininet e o controlador SDN Ryu. Através da simulação de múltiplos cenários, variando desde a competição direta sem QoS até configurações com filas hierárquicas (HTB), gerenciamento ativo de filas (FQ-CODEL) e combinação de policiamento com modelagem de tráfego, foi possível quantificar as reações distintas dos protocolos e a eficácia das políticas de QoS aplicadas.

5.1 Contribuições do Trabalho

As principais contribuições deste estudo podem ser resumidas na validação empírica das diferenças protocolares, demonstrando quantitativamente a natureza adaptativa do TCP em contraste com a não responsiva do UDP sob congestionamento (Cenário 1). Analisou-se a interação protocolo-QoS com HTB, evidenciando como esta política impõe garantias de banda e prioridades de forma distinta para UDP (descarte) e TCP (redução de taxa via RTT), validando também a ocorrência de *bufferbloat* (Cenário 2). Confirmou-se o comportamento com múltiplos fluxos, mostrando a tendência do TCP à divisão justa e a instabilidade/inanição do UDP sem QoS, além da capacidade do HTB de gerenciar prioridades estritamente, podendo suprimir fluxos UDP de baixa prioridade (Cenário 3). Clarificou-se experimentalmente a distinção entre policiamento de ingresso (descarte explícito) e modelagem de egresso (limitação implícita via contrapressão), observando seus diferentes impactos em UDP e TCP (Cenário 4). Realizou-se um comparativo entre FIFO, HTB e FQ-CODEL, quantificando a superioridade dos dois últimos em gerenciar tráfego misto, destacando a garantia estrita do HTB e a eficácia adaptativa do FQ-CODEL em prover isolamento e baixa latência (Cenário 5). Por fim, a utilização do ambiente controlado com Mininet e Ryu permitiu a replicação e análise granular dos cenários, fornecendo uma base sólida para as conclusões.

5.2 Limitações do Estudo

Apesar dos resultados obtidos, é importante reconhecer algumas limitações inerentes a este estudo. O uso de um ambiente de simulação (Mininet), embora robusto, pode introduzir diferenças em relação a redes físicas, como evidenciado pela necessidade de desativar o TSO. Os parâmetros de QoS testados foram específicos, e o desempenho pode

variar com outras configurações, incluindo o dimensionamento intencionalmente pequeno do buffer em alguns testes. A geração de tráfego com `iperf` representa fluxos contínuos ou de taxa fixa, que diferem da complexidade e variabilidade de aplicações reais. As topologias de rede, ainda que variadas, foram simplificadas em comparação com redes de produção. Adicionalmente, a lógica do controlador SDN utilizada limitou-se à aplicação de regras estáticas, sem explorar controles dinâmicos avançados.

5.3 Trabalhos Futuros

Com base nas conclusões e limitações, diversas direções para pesquisas futuras podem ser propostas. Sugere-se a exploração de topologias mais complexas e a utilização de geradores de tráfego que simulem um mix de aplicações reais mais diversificado. Uma investigação sistemática do impacto da variação dos parâmetros das políticas de QoS (HTB, FQ-CODEL, etc.) e a comparação com outros algoritmos AQM (CAKE, PIE) seriam valiosas. O desenvolvimento e a avaliação de aplicações SDN mais sofisticadas, que ajustem dinamicamente as políticas de QoS baseadas em monitoramento em tempo real, representam uma área promissora. A validação dos resultados em um testbed físico complementaria os achados da emulação. A análise de como diferentes variantes do TCP (CUBIC, BBR) interagem com as políticas de QoS implementadas também enriqueceria o entendimento.

5.4 Considerações Finais

De forma crítica, este trabalho demonstrou que as técnicas de gerenciamento de filas e policiamento de ingresso, embora conceitualmente consolidadas, apresentam desafios práticos de parametrização e integração. A efetividade dessas técnicas depende fortemente do perfil do tráfego, do dimensionamento dos buffers e da correta priorização das classes. Observou-se que, embora o HTB proporcione controle rigoroso e previsibilidade, sua configuração pode ser trabalhosa e sensível a ajustes incorretos. Já o FQ-CODEL, por automatizar o controle de latência, mostrou-se mais viável em cenários dinâmicos, mas com menor previsibilidade de alocação de banda. O policiamento de ingresso, por sua vez, provou ser eficiente para controle de borda, mas pode resultar em perdas abruptas se mal dimensionado, exigindo calibração cuidadosa para evitar impactos severos em fluxos sensíveis.

As dificuldades práticas enfrentadas ao longo do desenvolvimento reforçam a complexidade de aplicar QoS de forma realista. A principal delas foi a elaboração dos códigos e scripts de automação, desenvolvidos integralmente do zero, o que exigiu uma compreensão aprofundada dos utilitários nativos do Linux (como `tc` e `iptables`) e das particularidades do Mininet. Foi necessário compreender o funcionamento interno das filas e ajustar manu-

almente parâmetros como o tamanho de buffer, de modo a reproduzir perdas e atrasos de forma observável, já que o buffer padrão do Linux é elevado e tende a mascarar o congestionamento. Além disso, a integração entre o controlador Ryu e as políticas de QoS exigiu ajustes finos e depuração constante, evidenciando o grau de dificuldade técnica envolvido na implementação experimental de QoS em ambientes SDN.

Em síntese, o trabalho reforça as diferenças fundamentais no comportamento dos protocolos TCP e UDP sob congestionamento e evidencia as particularidades de cada técnica de gerenciamento de filas e policiamento. Políticas como HTB e FQ-CODEL mostraram-se eficazes na mitigação do *bufferbloat* e na manutenção da estabilidade da rede, sendo a escolha entre elas dependente dos requisitos de previsibilidade (HTB) ou adaptabilidade (FQ-CODEL). A compreensão dessas interações, viabilizada pelo ambiente SDN/Mininet, é essencial para o projeto e otimização de redes modernas. Espera-se que os resultados obtidos sirvam como base para futuras pesquisas e para a implementação de soluções de QoS mais eficientes e realistas em redes corporativas e de alta demanda.

Referências

ABDELGHANY, Ahmed; MUTHANNA, Ammar; SALAH, Khaled. Qmip: Queue management with ingress policing for sdn-based 5g networks. *Computer Networks*, Elsevier, v. 229, p. 1–14, 2023. Citado na página 29.

ALI, Zahid; KHAN, Sheraz; REHMAN, Sami Ur. Congestion-aware routing in sdn for smart grid communications. *Sustainable Computing: Informatics and Systems*, Elsevier, v. 35, p. 1–10, 2022. Citado na página 30.

ALMESBERGER, Werner. *Linux Network Traffic Control — Implementation Overview*. 2001. Documentação Técnica. Disponível em: <http://www.lartc.org/lartc.pdf>. Citado 2 vezes nas páginas 33 e 34.

ALVIZU, Ricardo; BRUGNOLI, Alejandro; PAGANI, Alejandro; MATA-DíAZ, Jorge; CERVELLó-PASTOR, Jorge. Machine learning techniques for traffic engineering in sdn: A survey. *Computer Networks*, v. 106, p. 212–228, 2016. Citado na página 25.

AWDUCHE, Daniel O.; CHIU, Alden; ELWALID, Anwar; WIDJAJA, Indra; XIAO, XiPeng. *Overview and Principles of Internet Traffic Engineering*. 2002. <https://datatracker.ietf.org/doc/html/rfc3272>. RFC 3272. Citado na página 24.

BARI, Md Faisal; HUQ, Khandaker Asifuzzaman; RAYCHOUDHURY, Vishal. Mlqos: Machine learning-based qos configuration in sdn. In: IEEE. *2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. [S.l.], 2022. p. 1–6. Citado na página 29.

BERG, J. Van den; BRORSSON, M. Random early detection for congestion control in high-speed networks. *Computer Networks*, v. 74, p. 45–56, 2014. Citado 2 vezes nas páginas 27 e 28.

CASADO, Martin; FREEDMAN, Michael J.; PETTIT, Justin; LUO, Jianying; MCKEOWN, Nick; SHENKER, Scott. Ethane: Taking control of the enterprise. In: *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*. [S.l.: s.n.], 2007. p. 1–12. Citado 2 vezes nas páginas 15 e 40.

CHEN, Ming; YANG, Lei; ZHAO, Yan; ZHANG, Qiang. Deep-q learning based service function chaining in 5g network. *IEEE Access*, IEEE, v. 6, p. 30564–30573, 2018. Citado na página 29.

CHENG, Xue; WU, Chuan; JI, Yusheng; LI, Bo. Joint optimization of qos and energy in sdn-based traffic engineering. *IEEE Transactions on Network and Service Management*, v. 15, n. 3, p. 1170–1184, 2018. Citado na página 26.

DEVERA, Martin. *HTB Linux queuing discipline manual — usage and internals*. 2002. Documentação Online. Disponível em: <http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm>. Citado na página 34.

DICKEY, Thomas E. *xterm – terminal emulator for the X Window System*. 2023. <https://invisible-island.net/xterm/>. Citado na página 33.

FARREL, Adrian. *The Internet and Its Protocols: A Comparative Approach*. [S.l.]: Morgan Kaufmann, 2003. Citado na página 24.

GETTYS, Jim. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, v. 15, n. 3, p. 95–96, May 2011. Citado na página 34.

GRUPO DE TELEINFORMÁTICA E AUTOMAÇÃO - UFRJ. *Arquitetura SDN (Software Defined Networking)*. 2015. Disponível em: <https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2015_2/SDN/architecture.html>. Citado 3 vezes nas páginas 20, 21 e 23.

GUNDOGAN, Ekin Budak; HABIB, Umer. Quality of service in sdn: A survey. *IEEE Access*, v. 6, p. 65123–65144, 2018. Citado 2 vezes nas páginas 26 e 27.

GUO, Jian; WANG, Xiaoming; LIU, Hao. Dynamic ingress policing for qos-aware sdn. *Journal of Network and Computer Applications*, Elsevier, v. 199, p. 1–13, 2022. Citado na página 29.

HOEILAND-JOERGENSEN, T.; MCKENNEY, P.; TAHT, D.; GETTYS, J.; DUMAZET, E. RFC, *FQ-CoDel Packet Scheduler and Active Queue Management*. RFC Editor, 2018. Disponível em: <https://www.rfc-editor.org/info/rfc8290>. Citado na página 34.

IPERF. *iPerf – The TCP, UDP and SCTP network bandwidth measurement tool*. 2025. <https://iperf.fr/>. Citado na página 33.

KATTEPUR, Ajit; DWARAKANATH, Arun; KRISHNAMURTHY, Sujatha. Model-based reinforcement learning for queue management in sdn routers. In: IEEE. *2021 IEEE Global Communications Conference (GLOBECOM)*. [S.l.], 2021. p. 1–6. Citado na página 29.

KREUTZ, Diego; RAMOS, Fernando M. V.; VERISSIMO, Paulo; ROTHENBERG, Christian Esteve; AZODOLMOLKY, Siamak; UHLIG, Steve. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, 2015. Citado 6 vezes nas páginas 15, 20, 22, 23, 32 e 40.

KUROSE, James F.; ROSS, Keith W. *Redes de Computadores e a Internet: Uma Abordagem Top-Down*. 6. ed. [S.l.]: Pearson, 2017. Citado 2 vezes nas páginas 15 e 19.

LANTZ, Bob; HELLER, Brandon; MCKEOWN, Nick. A network in a laptop: Rapid prototyping for software-defined networks. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*. [S.l.]: ACM, 2010. p. 1–6. Citado 2 vezes nas páginas 32 e 33.

LEWIS, Mark. *MPLS: Implementing the Technology*. [S.l.]: McGraw-Hill, 2004. Citado na página 25.

LI, Lixin; WANG, Yuan; ZHANG, Mingwei; HUANG, Min. An overview of network traffic engineering. *IEEE Communications Surveys & Tutorials*, v. 10, n. 1, p. 28–36, 2007. Citado na página 24.

LI, Zhiqiang; WU, Jiayi; LIU, Rong; ZHANG, Hongyu. Policing ingress traffic in sdn: Design and implementation. *IEEE Journal on Selected Areas in Communications*, v. 36, n. 12, p. 2654–2665, 2018. Citado na página 28.

LINUX FOUNDATION COLLABORATIVE PROJECT. *Open vSwitch – Production Quality, Multilayer Virtual Switch*. 2016. <https://www.openvswitch.org/>. Citado na página 32.

LIU, H.; ZHAO, Y. Weighted fair queuing for efficient traffic management in sdn. *Journal of Computer Networks*, v. 85, p. 112–128, 2017. Citado 2 vezes nas páginas 27 e 28.

LIU, Han; ZHAO, Yu; ZHANG, Ping. Online qos/qoe-aware service function chain orchestration in sdn/nfv. *Future Generation Computer Systems*, Elsevier, p. 1–10, 2024. Citado na página 30.

LUO, Wei; ZHANG, Ning; WANG, Jie. Qos policy translation in hybrid networks with sdn-aware gateways. *IEEE Transactions on Network and Service Management*, IEEE, v. 20, n. 1, p. 65–76, 2023. Citado na página 29.

MANZANARES-LOPEZ, Pilar; GARCIA-ALMIñANA, Jordi; BADIA, Rosa M. Dynamic enhanced distributed channel access for sdn-based wireless networks. In: IEEE. *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. [S.l.], 2018. p. 1–6. Citado na página 29.

MARKETS AND MARKETS. *Software Defined Networking Market - Global Forecast to 2030*. 2023. Acesso em: 13 abr. 2025. Disponível em: <https://www.marketsandmarkets.com/>. Citado na página 15.

MCKEOWN, Nick; ANDERSON, Tom; BALAKRISHNAN, Hari; PARULKAR, Guru; PETERSON, Larry; REXFORD, Jennifer; SHENKER, Scott; TURNER, Jonathan. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, v. 38, n. 2, p. 69–74, 2008. Citado 6 vezes nas páginas 15, 21, 22, 23, 28 e 32.

MININET. *Mininet Overview*. 2022. <http://mininet.org/overview/>. Citado na página 32.

MOEYERSONS, Jeroen; TAVERNIER, Wouter; COLLE, Didier; PICKAVET, Mario. Enabling emergency flow prioritization in sdn networks. *2019 15th International Conference on Network and Service Management (CNSM)*, IEEE, p. 1–8, 2019. Citado na página 26.

MORREALE, P.; UBALDI, M.; SANTORO, R. Policing traffic in sdn for optimal bandwidth allocation. *IEEE Transactions on Network and Service Management*, v. 16, n. 3, p. 1–7, 2019. Citado 2 vezes nas páginas 26 e 27.

NICHOLS, Kathleen; JACOBSON, Van. Controlling queue delay. In: *ACM SIGCOMM Computer Communication Review*. [S.l.]: ACM, 2012. v. 42, n. 5, p. 42–50. Citado na página 34.

NUNES, B. A. A.; MENDONCA, M.; NGUYEN, X. N.; OBRACZKA, K.; TURLETTI, T. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, v. 16, n. 3, p. 1617–1634, 2014. Citado 2 vezes nas páginas 20 e 22.

OH, Byung-Ho; VURAL, S.; WANG, Ning; TAFAZOLLI, R. Priority-based flow control for dynamic and reliable flow management in sdn. *IEEE Transactions on Network and Service Management*, v. 15, n. 4, p. 1720–1732, 2018. Citado 3 vezes nas páginas 26, 27 e 28.

OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification Version 1.1.0*. 2011. <https://opennetworking.org>. Citado na página 23.

OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification Version 1.3.0*. 2012. <https://opennetworking.org>. Citado na página 23.

OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification Version 1.5.1*. 2015. <https://opennetworking.org>. Citado na página 24.

OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification Version 1.5.1*. 2015. Disponível em: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>. Citado na página 33.

OSBORNE, Eric; SIMHA, Ajay. *Traffic Engineering with MPLS*. [S.l.]: Cisco Press, 2002. Citado na página 25.

PFAFF, Ben; PETTIT, Justin; KOPONEN, Teemu; AMIDON, Keith; CASADO, Martin; GROSS, Jesse. The design and implementation of open vswitch. In: *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. [S.l.]: USENIX Association, 2015. p. 117–130. Citado na página 32.

PORWAL, Manoj; YADAV, Rajeev K.; CHARHATE, Shailendra. Engineering traffic in networks: A qos-based approach. *International Journal of Computer Science and Network Security*, v. 8, n. 4, p. 319–326, 2008. Citado 2 vezes nas páginas 27 e 28.

POSTEL, Jon. *User Datagram Protocol*. 1980. <https://tools.ietf.org/html/rfc768>. RFC 768. Citado na página 19.

POSTEL, Jon. *Transmission Control Protocol*. 1981. <https://tools.ietf.org/html/rfc793>. RFC 793. Citado na página 19.

ROSEN, Eric C.; VISWANATHAN, Arun; CALLON, Ross. *Multiprotocol Label Switching Architecture*. RFC Editor, 2001. RFC 3031. Disponível em: <https://datatracker.ietf.org/doc/html/rfc3031>. Citado na página 25.

SALMAN, Ola; ELHAJJ, Imad H.; CHEHAB, Ali; KAYSSI, Ayman. Software defined networking: State of the art and research challenges. *Computer Networks*, Elsevier, v. 75, p. 164–181, 2017. Citado 3 vezes nas páginas 20, 28 e 40.

SEEGER, Jan; BRöRING, Arne; PAHL, Marc-Oliver; SAKIC, Ermin. Semantic qos configuration for iot applications in sdn environments. In: IEEE. *2019 IEEE International Conference on Communications (ICC)*. [S.l.], 2019. p. 1–7. Citado 3 vezes nas páginas 21, 28 e 40.

SILVA, Pedro Henrique Diniz da; JúNIOR, Nilton Alves. Ferramenta IPERF: geração e medição de tráfego TCP e UDP. *Notas Técnicas*, v. 4, n. 2, p. 1–13, 2014. Disponível em: <https://revistas.cbpf.br/index.php/nt/article/view/25>. Citado na página 33.

SONG, Yu; QIAN, Xusheng; ZHANG, Nan; WANG, Wei; XIONG, Ao. Qos routing optimization based on deep reinforcement learning in sdn. *Computers, Materials & Continua*, Tech Science Press, v. 79, p. 3007–3021, 2024. Citado na página 30.

TANENBAUM, Andrew S.; WETHERALL, David J. *Redes de Computadores*. 5. ed. [S.l.]: Pearson, 2011. Citado 2 vezes nas páginas 15 e 19.

XIA, Jun; YANG, Weijia; HUANG, Tao. Service-aware transport layer for qos in sdn cellular networks. *IEEE Transactions on Mobile Computing*, IEEE, v. 21, n. 9, p. 3456–3469, 2022. Citado na página 29.

ZHANG, T.; LIU, H.; ZHAO, Y. Software-defined networking (sdn) for traffic management and qos. *Journal of Network and Systems Management*, v. 26, n. 2, p. 452–471, 2018. Citado na página 28.

ZHANG, Yixin; LI, Cheng; WANG, Rong. Adaptive queue and buffer management for sdn-iot. *IEEE Internet of Things Journal*, IEEE, p. 1–12, 2024. Citado na página 29.

ZHANG, Yin; ROUGHAN, Matthew; DUFFIELD, Nick; GREENBERG, Albert. Fast accurate computation of large-scale ip traffic matrices from link loads. In: *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. [S.l.]: ACM, 2003. p. 206–217. Citado na página 24.