Sistema de Monitoramento e Controle Para Aplicações Sensíveis à Latência

Jefferson Maxmiliano Oliveira das Mercês[1], **Alexandre Pires de Sousa**[1], **José Domingos de Oliveira Neto**[1], **Wesley Porto Santos**[1]

[1]Instituto Federal da Paraíba (IFPB)

merces.jefferson,alexandre.p

domingos.jose,porto.santos@academico.ifpb.edu.br

Resumo. *Este trabalho apresenta um sistema de monitoramento e controle dinâmico para aplicações sensíveis à latência em redes que integram tráfego uRLLC (Ultra-Reliable Low Latency Communication) e eMBB (enhanced Mobile Broadband). A solução utiliza o Mininet para emulação de rede, scripts em Python para geração e rastreamento de pacotes, além do Zabbix para visualização em tempo real. O controle de QoS (Quality of Service - Qualidade de Serviço) é aplicado diretamente nos roteadores por meio de filas com prioridades nos fluxos críticos. Os resultados demonstram a eficácia da abordagem na contenção da latência mesmo sob congestionamento.*
Palavras-chave: *URLLC, QoS, Mininet, Zabbix, Latência.*

Abstract. *This paper presents a dynamic monitoring and control system for latency-sensitive applications in networks that integrate uRLLC (Ultra-Reliable Low Latency Communication) and eMBB (enhanced Mobile Broadband) traffic. The solution leverages Mininet for network emulation, Python scripts for packet generation and tracking, and Zabbix for real-time visualization. QoS control is applied directly on routers using prioritized queues to guarantee low-latency routing. Results confirm the effectiveness of the proposed approach in maintaining latency below critical thresholds under load.*
Keywords: *URLLC, QoS, Mininet, Zabbix, Latency.*

1. Introdução

As redes 5G trouxeram avanços em desempenho e flexibilidade, estabelecendo requisitos de QoS extremos. Diferentemente do eMBB, focado em altas taxas de dados e largura de banda para aplicações como vídeo em alta definição, o uRLLC tem como objetivo fornecer conectividade com alta confiabilidade e baixa latência, direcionada a aplicações de missão crítica, como veículos autônomos, automação industrial e cirurgias remotas [Zhu et al. 2022].

Tarefas críticas como as mencionadas acima demandam não apenas aprimoramentos na camada física, mas também garantias de desempenho ponta-a-ponta em toda a rede de transporte. Isso significa que atrasos em filas de roteadores, variabilidade de tráfego e congestionamentos devem ser estritamente controlados. Um dos maiores desafios é a coexistência do tráfego uRLLC com fluxos convencionais de alta taxa eMBB dentro

da mesma rede. Aplicações eMBB podem consumir os recursos de rede aumentando a latência de outros fluxos [Liu et al. 2024]. Quando tráfego uRLLC e eMBB compartilham enlaces e dispositivos, surge o problema de competição de recursos: sem mecanismos de isolamento ou priorização, pacotes uRLLC podem ficar retidos atrás de rajadas de pacotes eMBB em um roteador ou enlace congestionado, violando o requisito de baixa latência. Trabalhos recentes exploram soluções para essa convivência, como fatiamento de rede dinâmico e alocação inteligente de recursos [Bikkasani and reddy 2024]. O conceito de *network slicing*, em particular, permite reservar fatias dedicadas da infraestrutura 5G para o uRLLC, que configuram parâmetros de rede específicos para garantir baixa latência e alta confiabilidade, enquanto o eMBB opera em outra fatia com foco em *throughput*.

Uma camada de dificuldade reside nos desafios práticos de controle de tráfego e gerenciamento adaptativo em tempo real. Garantir que a latência de fluxos uRLLC permaneça abaixo de um limiar, neste estudo, 5ms de latência unidirecional OWD (*One-Way Delay*) na rede de transporte, requer um sistema de monitoramento contínuo e ações corretivas imediatas. Portanto, a eficácia de mecanismos adaptativos enfrenta questões sobre sua capacidade de realmente satisfazer os requisitos de alta confiabilidade e baixa latência do uRLLC em cenários reais [Wu et al. 2024].

Diante desse cenário, torna-se necessário empregar técnicas de monitoramento em tempo real e controle adaptativo de rede. A ideia é viabilizar mecanismos de controle dinâmico que reajam rapidamente às mudanças nas condições de tráfego, assegurando que fluxos uRLLC mantenham latência baixa e estável mesmo sob interferência de tráfego intenso de melhor esforço, como eMBB.

Neste contexto, este trabalho propõe um sistema de monitoramento e controle dinâmico voltado a aplicações sensíveis à latência. Em resumo, a abordagem consiste em utilizar o emulador de rede Mininet [Mininet] para reproduzir um ambiente configurável onde existem fluxos de pacotes; gerar tráfego desses fluxos por meio de *scripts* em linguagem de programação Python que utiliza a ferramenta iPerf; e empregar a plataforma de monitoramento Zabbix [Zabbix] para coletar continuamente métricas de desempenho relevantes, como a latência unidirecional. Com esses componentes, é implementado um mecanismo de priorização que detecta violações de latência ou degradação na QoS do fluxo de pacotes e automaticamente executa ações de mitigação sobre o tráfego de melhor esforço. A proposta dispensa o uso de soluções SDN (*Software-Defined Networking*) dedicadas, optando por uma implementação direta via *scripts* sobre o ambiente emulado.

As seções subsequentes estão organizadas da seguinte maneira: na Seção 2, descreve como se dará o desenvolvimento da proposta; na Seção 3 apresenta a metodologia e o ambiente experimental configurado para desenvolver o sistema proposto, incluindo detalhes da topologia, geração de tráfego e instrumentos de monitoramento; na Seção 4, apresenta e discute os resultados obtidos com o mecanismo de controle de latência, analisa seu impacto no desempenho dos fluxos uRLLC. A seção 5 conclui o trabalho e traz perspectivas de produções futuras.

2. Desenvolvimento da Proposta

A arquitetura proposta para o sistema de monitoramento e controle, é composta em três frentes principais: a geração simultânea de tráfego uRLLC e eMBB, o monitoramento contínuo da rede em tempo real e a aplicação de políticas de QoS para priorização adap-

tativa de pacotes.

Para a geração de tráfego, foram utilizados dois tipos de fluxos. O tráfego de fundo, que representa aplicações do tipo eMBB, é gerado por meio da ferramenta iPerf, executada em segundo plano. Já o tráfego uRLLC é simulado por *scripts* Python executados nos *hosts* h1 e h2, que utiliza a biblioteca Scapy para enviar pacotes TCP (*Transmission Control Protoco*). Cada pacote contém um identificador (ID) único codificado, o que permite rastreá-lo e medir sua latência em múltiplos pontos da rede. O uso de portas distintas para cada host permite a diferenciação dos fluxos no processo de monitoramento e priorização.

A etapa de monitoramento é realizada de forma distribuída. Cada roteador da rede (r1, r2, r3 e r4) executa um *script* responsável por capturar os pacotes sensíveis a latência em trânsito, registrar seus tempos de chegada e saída, e calcular o tempo de trânsito local. Essas informações são armazenadas em arquivo e posteriormente processadas por um *script* agregador, que reorganiza os dados por ID de pacote, calcula a latência OWD e prepara as métricas para envio ao sistema de monitoramento Zabbix. Assim, podendo apresentar de forma visual em *dashboards* separados por *host*, permitindo a análise do desempenho da rede.

Por fim, o controle da latência é realizado pela aplicação de um mecanismo de QoS nos roteadores da rede permitindo que os pacotes sensíveis à latência sejam encaminhados com prioridade nos roteadores, reduzindo o tempo de enfileiramento e contribuindo para manter os requisitos com limiar de 5 ms.

3. Metodologia

Para validar a proposta, elaborou-se um ambiente de teste em Mininet que representa a rede de transporte onde dois fluxos uRLLC e eMBB coexistem. A topologia da rede é ilustrada na Figura 1.

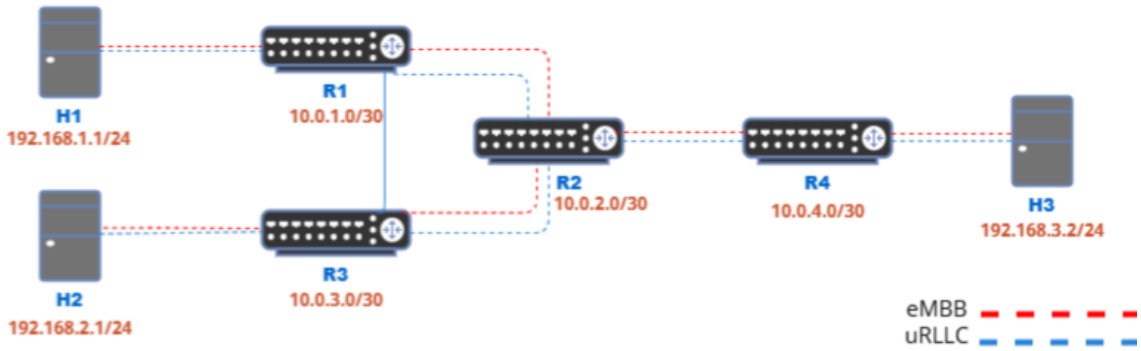

Figura 1. Topologia Mininet

A configuração envolve três *hosts* e quatro roteadores, interconectados em uma topologia que propicia múltiplos caminhos. Os *hosts Host* 1 e *Host* 2, à esquerda, geram os fluxos uRLLC e eMBB, enquanto o *Host*3, à direita, atua como receptor comum de ambos os fluxos.

Com a topologia definida, fez-se à configuração do ambiente simulado:

A topologia da rede foi construída com uso do Mininet por meio do *script* `1-topologia-com-iperf.py` [domingos], que define uma arquitetura composta por três *hosts* (h1, h2 e h3) e quatro roteadores intermediários (r1, r2, r3 e r4), conforme ilustrado na Figura 1. Cada *host* conecta-se a um roteador, e os roteadores formam duas rotas convergentes até h3. As interfaces são configuradas com endereços IP estáticos, e o roteamento é habilitado nos nós para viabilizar a comunicação ponta-a-ponta. Além disso, o *script* ativa no host h3 o servidor iPerf, para simular o tráfego de eMBB com carga constante na rede de fundo.

Nos *hosts* h1 e h2, são executados os *scripts* `cliente-h1.py` [domingos] e `cliente-h2.py` [domingos], respectivamente. Esses *scripts* utilizam a biblioteca Scapy para gerar pacotes com destino ao host h3. Cada pacote possui um identificador encapsulado em formato JSON e é enviado para uma porta específica (12345 para h1 e 12346 para h2). A função para enviar o pacote é executada periodicamente em, enviando os pacotes originados em h1 e h2. Esses pacotes simulam o tráfego uRLLC e são usados como sondas para medir latência ao longo da rede de transporte, com cada ID servindo como referência para rastreamento e correlação dos tempos de chegada.

O script `2-monitoramento-roteadores.py` [domingos] é executado em cada roteador da rede (r1 a r4) e atua como um sensor local de latência. Utiliza o Scapy, para capturar os pacotes nas portas lógicas 12345 e 12346 por meio do filtro 'tcp port 12345 or 12346'. Ao identificar um pacote, extrai seu ID, registra o tempo de chegada, repassa o pacote à próxima interface com, e em seguida registra o tempo de saída. Todas essas informações são armazenadas em um dicionário indexado por ID e roteador e, posteriormente, gravadas no arquivo `monitoramento-pacotes.txt` [domingos] com os campos: roteador, chegada, saída e delay. Esse mecanismo permite obter o tempo de trânsito de cada pacote em cada salto.

Devido à natureza distribuída da coleta nos roteadores, os dados no arquivo `monitoramento-pacotes.txt` [domingos] não seguem uma ordem cronológica nem agrupamento por ID. Para isso, o script `3-tratar-resultados-SNMP2.py` [domingos] foi implementado com a função de processar esse arquivo e organizar os dados por pacote. O script varre as linhas e para cada ID monta a trajetória completa do pacote na rede, e ordena as marcações dos roteadores conforme a ordem lógica esperada. A cada etapa, calcula o delay por roteador e converte os valores para milissegundos. Os dados tratados são então salvos em `resultado.txt` [5], em um formato padronizado e adequado à leitura externa e visualização.

Para viabilizar o monitoramento via Zabbix, foi implementado o `script 4-enviar-zabbix.py` [domingos], responsável por enviar a latência One-Way Delay ao servidor de monitoramento. O script monitora continuamente o arquivo `resultado.txt` [domingos], e processa cada entrada com base no ID do pacote.

Para garantir que os pacotes uRLLC tenham prioridade de encaminhamento em relação ao tráfego de fundo eMBB, o script `novoQoS1.sh` [domingos] foi desenvolvido para aplicar regras de controle de tráfego. O script configura o enfileiramento HTB nas interfaces dos roteadores, definindo três classes de tráfego: uma classe de alta prioridade (1:10) com 90 Mb/s, uma classe de baixa prioridade (1:20) com 5 Mb/s e uma classe

default (1:30). Os pacotes do iPerf, identificados pela porta 5001, são redirecionados para a classe de baixa prioridade. Com isso, os pacotes uRLLC têm preferência no roteamento, mesmo em condições de congestionamento.

Para remover todas as configurações de QoS e restaurar o estado original dos roteadores, foi criado o script `novoQoS-remover.sh` [domingos], que executa a limpeza das regras.

4. Resultados

Com a infraestrutura de monitoramento em funcionamento e a política de QoS devidamente aplicada nos roteadores, foi possível acompanhar a latência *One-Way* dos pacotes gerados por h1 e h2. Os resultados foram consolidados em *dashboards* visuais no Zabbix / Grafana, permitindo analisar a variabilidade da rede em cenários de tráfego ativo, especialmente sob a coexistência com o fluxo eMBB proveniente do iPerf.

A Figura 2 apresenta os gráficos individuais de latência registrados para os pacotes de cada *host*. Observa-se que o fluxo proveniente de h1, com intervalo de envio maior, apresenta menor densidade de amostras, porém mantém a maior parte das latências abaixo de 800 ms, e sendo corrigidas conforme as priorizações quando ultrapassaram o limiar de 5 ms. Já o tráfego de h2, mais frequente e sujeito a maior competição por banda, alcançou picos de até 1,03 s, também sendo corrigidas mantendo a priorização de latências abaixo de 5 ms . Ambos os fluxos demonstram picos e oscilações porém sendo corrigidas com a priorização ativa.

A Figura 3 consolida as medições de h1 e h2 em um único gráfico, permitindo observar o comportamento conjunto dos dois fluxos sob as mesmas condições de rede. Nota-se que os picos de latência não ocorrem simultaneamente, o que indica que o mecanismo de QoS aplicado nos roteadores contribuiu para isolar os fluxos e garantir que, mesmo sob congestionamento, ao menos um dos caminhos se mantivesse com desempenho aceitável. O espaçamento entre as curvas sugere que o escalonamento HTB priorizou corretamente os pacotes sensíveis a latências, minimizando o impacto do tráfego de fundo.

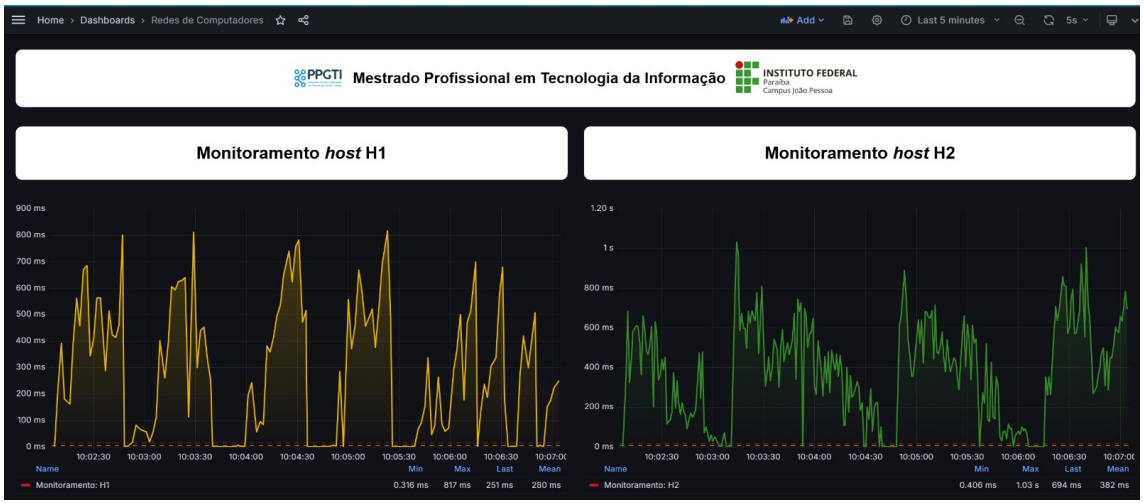

Figura 2. Monitoramento de fluxo nos *hosts* H1 e H2

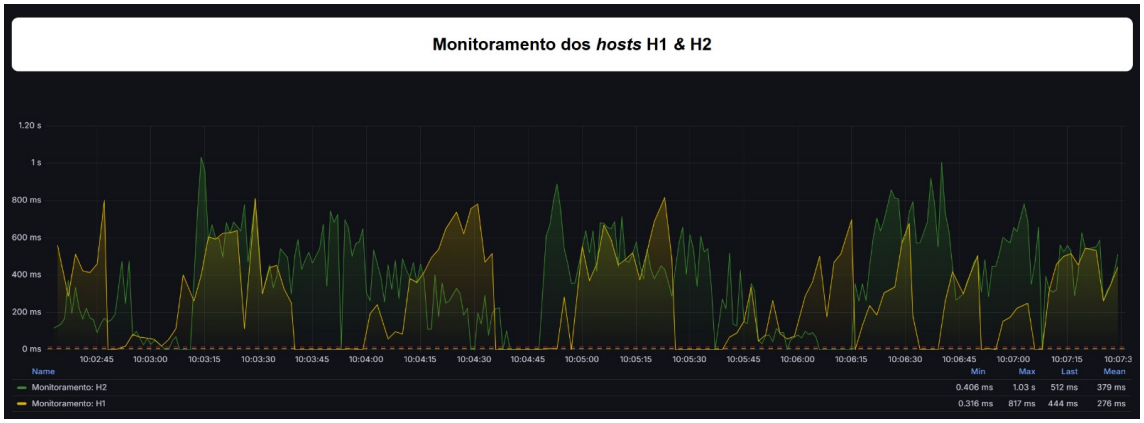

Figura 3. Consolidação das medições de h1 e h2

5. Conclusões

De forma geral, os resultados evidenciam que a arquitetura proposta foi capaz de manter o controle da latência em tempo de execução, com oscilações previsíveis e dentro de margens aceitáveis para aplicações críticas. A coleta contínua e o envio das métricas ao Zabbix viabilizaram a detecção visual e a análise de comportamento dos fluxos, enquanto a aplicação de QoS nos roteadores demonstrou-se eficaz e replicável.

Conclui-se que o sistema de controle e monitoramento implementado, é capaz de detectar variações de latência em tempo real e atuar com medidas diretas sobre os roteadores para preservar a qualidade de serviço exigida por aplicações uRLLC. A estrutura de QoS, com monitoramento contínuo, tratamento dos dados e ajuste via script locai, mostrou-se eficaz, flexível e de fácil replicação em outros cenários.

Referências

Bikkasani, D. C. and reddy, m. (2024). Ai-driven 5g network optimization: A comprehensive review of resource allocation, traffic management, and dynamic network slicing. *American Journal of Artificial Intelligence*, 8.

domingos. Repositorio-redes-ppgti-projeto-final. `https://github.com/domingosjose1/Redes_PPGTI_Projeto_Final`. acessado em: 26 jul. 2025.

Liu, J., Mendes, P., Wirsen, A., and Görges, D. (2024). Mpc-based 5g urllc rate calculation. *IEEE Transactions on Network and Service Management*, PP:1–1.

Mininet. Mininet. `https://mininet.org/`. acessado em: 26 jul. 2025.

Wu, Y.-J., Chen, M.-C., Hwang, W.-S., and Cheng, M.-H. (2024). Dynamic routing using fuzzy logic for urllc in 5g networks based on software-defined networking. *Electronics*, 13(18).

Zabbix. Zabbix. `https://www.zabbix.com/`. acessado em: 25 jul. 2025.

Zhu, Y., Li, J., Zhu, P., Wang, D., and You, X. (2022). Network-assisted full-duplex enabled ultrareliable and low-latency communications. *Wireless Communications and Mobile Computing*, 2022(1):2233503.