

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise do Protocolo QUIC: Uma Comparação Entre Implementações no Espaço do Kernel e Espaço de Usuário

Bruno Barros Xavier

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Marcelo Antonio Marotta

Brasília
2025



Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise do Protocolo QUIC: Uma Comparação Entre Implementações no Espaço do Kernel e Espaço de Usuário

Bruno Barros Xavier

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Marcelo Antonio Marotta (Orientador)
CIC/UnB

Prof. Dr. Priscila América Solís Mendez Barreto
CIC/UnB

Prof. Dr. Edison Ishikawa
CIC/UnB

Prof. Dr. Guilherme Novaes Ramos
Coordenador do Curso de Engenharia da Computação

Brasília, 21 de fevereiro de 2025

Dedicatória

Dedico aos incansáveis disseminadores de saber, que generosamente compartilham seu conhecimento e transformam o complexo em acessível e o desconhecido em familiar. A vocês, cuja missão é transformar vidas e abrir horizontes, minha eterna gratidão e admiração.

Agradecimentos

Primeiramente, gostaria de agradecer ao professor Dr. Marcelo Antonio Marotta, cuja orientação e sabedoria foram fundamentais para a condução e o sucesso deste projeto. Nossas reuniões semanais possibilitaram troca de ideias, assistência na elaboração desse texto e aprofundamento de discussões que enriqueceram significativamente esse trabalho.

Também gostaria de agradecer ao corpo docente dos Departamentos de Ciência da Computação e de Engenharia Elétrica da Universidade de Brasília, cuja dedicação e compartilhamento de conhecimento foram essencias para a formação acadêmica.

Por fim, gostaria de agradecer aos meus familiares e amigos, por darem apoio, lerem o trabalho diversas vezes e também darem comentários valiosos.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Este trabalho tem o foco de comparar implementações do protocolo QUIC feitas no kernel do linux em relação a implementações no espaço de usuário, visando avaliar as diferenças de desempenho em termos de *goodput* e uso de recursos do sistema operacional. A validação foi conduzida por meio de experimentos em contêineres Docker[1], configurados para simular interações cliente-servidor em uma variedade de condições de rede através do ns-3[2], de maneira similar aos procedimentos empregados no QUIC Interop Runner[3]. Melhorias futuras incluem a incorporação de chamadas de sistema, como *sendfile* e *splice*, além da implementação do descarregamento de pacotes no kernel.

Palavras-chave: QUIC, Linux, Espaço de kernel, Espaço de usuário

Abstract

This work focuses on comparing linux kernel-space implementations of the QUIC protocol with user-space implementations, aiming to evaluate the performance differences in terms of *goodput* and operating system resource usage. Validation was conducted through experiments in Docker containers [1], configured to simulate client-server interactions under a variety of network conditions through ns-3[2], similar to the procedures employed in the QUIC Interop Runner [3]. Future improvements include the incorporation of system calls, such as *sendfile* and *splice*, and the implementation of packet offloading on the kernel.

Keywords: QUIC, Linux, Kernel space, User space

Sumário

1 Introdução **1**

2 Trabalhos Relacionados **3**

2.1 The QUIC Transport Protocol: Design and Internet-Scale Deployment . . 3

2.2 Implementation and Performance Evaluation of the QUIC Protocol in Linux Kernel 4

2.3 Beyond socket options: making the Linux TCP stack truly extensible . . . 5

2.4 eQUIC Gateway: Maximizing QUIC Throughput using a Gateway Service based on eBPF + XDP 6

2.5 Automating QUIC Interoperability Testing 7

2.6 QUIC on the Highway: Evaluating Performance on High-rate Links 8

2.7 Evaluating QUIC Performance Over Web, Cloud Storage and Video Workloads 9

2.8 Managing Connections by QUIC-TCP Racing: A First Look of Streaming Media Performance Over Popular HTTP 3 Browsers 11

2.9 Characterization and Prediction of QUICs Performance Under Different Network Conditions 12

2.10 Performance Evaluation of QUIC Vs. TCP for Cloud Control Systems . . . 13

2.11 Síntese 13

3 Metodologia **15**

3.1 Cenário 15

3.2 Métricas 16

3.3 Protótipo 17

3.3.1 Implementação do QUIC no Espaço de Usuário 18

3.3.2 Implementação do QUIC no Kernel 19

3.3.3 Testes e Medições 20

4 Resultados **23**

4.1 Taxa de transferência (Goodput) 23

4.2 Chamadas de sistema 25

5 Trabalhos Futuros **27**

6 Conclusão **28**

Referências **30**

Lista de Figuras

1.1 Pilha do protocolo HTTP/2 em relação a pilha do protocolo HTTP/3 baseada no QUIC. 2

3.1 Configuração do cenário 16
3.2 Comando bash para download das imagens docker do *msquic* e *ngtcp2*. . . 19
3.3 Bash script para download, compilação e instalação do módulo da implementação do QUIC no kernel do linux, quanto montagem da imagem docker da implementação, usando Ubuntu 24.04 como sistema operacional. 19
3.4 Pseudo código da leitura e envio de um arquivo. 20
3.5 Pseudo código do uso da função *sendfile*. 20
3.6 Script utilizado no *bpftrace* para contar o número de chamadas feitos por um programa. 21

4.1 Taxa de transferência de cada implementação para diferentes taxas de perda. 24
4.2 Número de chamadas de sistema feitas no decorrer de cada implementação para diferentes taxas de perda. 26

Lista de Tabelas

4.1 Porcetagem de diferença da taxa de transferência entre as taxas de perda de 1%, 5% e 10%, comparadas ao cenário com 0% de perda. 24
4.2 Porcentagem adicional de chamadas de sistema realizadas nos cenário com taxas de perda de 1%, 5% e 10%, comparadas ao cenário com 0% de perda. 25

Lista de Abreviaturas e Siglas

0-RTT Zero Round-Trip Time.

ACK Acknowledgement.

CDNs Content Delivery Networks.

CPU Central Processing Unit.

eBPF Extended Berkeley Packet Filter.

HTTP/2 Hypertext Transfer Protocol 2.0.

HTTPS Hypertext Transfer Protocol Secure.

IETF Internet Engineering Task Force.

LAN Local Area Network.

QIR Quic Interop Runner.

QoE Quality of Experience.

QoS Quality of Service.

QUIC Quick UDP Internet Connections.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

TTA Time To Arrive.

UDP User Datagram Protocol.

VM Virtual Machine.

XDP Express Data Path.

Capítulo 1

Introdução

Por muito tempo, o Transmission Control Protocol (TCP) [4] tem sido fundamental na comunicação da internet, sendo um dos protocolos mais amplamente adotados devido à sua garantia de confiabilidade, ordem e integridade dos dados transmitidos, o que é crucial para muitas aplicações. Por ser um protocolo bem estabelecido, muitas redes são otimizadas exclusivamente para o protocolo TCP, o que dificulta a introdução de melhorias ou alternativas mais eficientes, contribuindo para a ossificação da rede [5]. Qualquer alteração significativa no TCP ou a adoção de novos protocolos exige mudanças profundas na infraestrutura de rede, que muitas vezes são evitadas devido ao custo e complexidade envolvidos. Para superar essas dificuldades, surgiu o QUIC, um protocolo baseado no UDP, que aproveita a simplicidade e a ampla compatibilidade do UDP com diversas plataformas e sistemas operacionais.

O QUIC é um protocolo de transporte desenvolvido inicialmente pelo Google e, atualmente, padronizado pela Internet Engineering Task Force (IETF) [6]. Sua integração com o UDP mostrado na Figura 1.1 proporciona várias vantagens em relação ao TLS/TCP, como criptografia nativa que reduz significativamente a latência ao eliminar a necessidade do handshake de três vias. Além disso, o QUIC permite independência entre múltiplos fluxos dentro de uma única conexão, o que evita o problema de bloqueio total dos fluxos quando um segmento é perdido. O protocolo também oferece suporte ao Zero Round-Trip Time (0-RTT), permitindo o envio de dados logo no começo do estabelecimento de conexão. Isso possibilita que dados de aplicativos sejam transmitidos já no primeiro pacote de handshake, reduzindo consideravelmente a latência durante a troca de redes, como ao alternar entre redes Wi-Fi e telefonia móvel.

Como o QUIC pretende oferecer diversas vantagens sobre a pilha tradicional TLS + TCP, as desvantagens do protocolo precisam ser levadas em consideração e gerenciadas. Uma das principais desvantagens que afeta o QUIC seria sua implementação no espaço do usuário, sem modificações no kernel do sistema operacional, apenas usando o protocolo

UDP para permitir implantação e atualização rápidas. Essa abordagem, embora facilitadora para atualizações frequentes e implantação flexível, pode introduzir uma sobrecarga adicional devido à troca constante de contexto entre o espaço do usuário e o kernel, o que pode impactar negativamente o desempenho. Além disso, a implementação no espaço do usuário pode limitar a capacidade do QUIC de tirar proveito de otimizações de baixo nível que estão disponíveis para protocolos implementados diretamente no kernel, como acesso mais eficiente aos recursos de rede e gerenciamento de *buffer*. Ainda que existam implementações do QUIC no kernel e no espaço de usuário, não há uma comparação profunda dos reais ganhos de desempenho em relação às duas implementações.

Este trabalho propõe comparar a implementação do protocolo QUIC diretamente no kernel do Linux para avaliação do seu desempenho em comparação a implementações no espaço de usuário. A implementação no kernel busca explorar otimizações de baixo nível que não são possíveis no espaço de usuário, como o acesso mais eficiente aos recursos de rede e a redução da sobrecarga causada pela troca constante de contexto entre o espaço de usuário e o kernel.

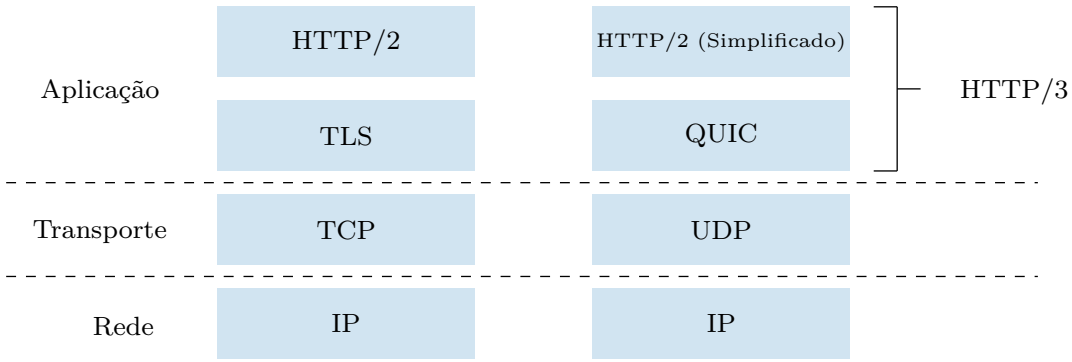


Figura 1.1: Pilha do protocolo HTTP/2 na esquerda em relação a pilha do protocolo HTTP/3 baseada no QUIC na direita.

Capítulo 2

Trabalhos Relacionados

Os trabalhos relacionados a seguir focam no desenvolvimento, implementação e análise de desempenho do protocolo QUIC em comparação com protocolos tradicionais como o TCP. Diversos estudos apontam melhorias em termos de latência, segurança e escalabilidade, sobretudo em cenários que demandam alta performance na transmissão de dados.

2.1 The QUIC Transport Protocol: Design and Internet-Scale Deployment

O QUIC foi projetado para melhorar o desempenho do tráfego HTTPS e permitir uma rápida implantação e evolução dos mecanismos de transporte. O protocolo substitui grande parte da pilha tradicional HTTPS, incluindo HTTP/2, TLS e TCP, e foi implementado globalmente pela Google em milhares de servidores, atendendo a diversos clientes, como o navegador Chrome e o aplicativo de transmissão de vídeo YouTube. O artigo *The QUIC Transport Protocol: Design and Internet-Scale Deployment* [7] trata do desenvolvimento e implantação em larga escala do protocolo QUIC. Dentro do artigo são detalhados as motivações para a criação do QUIC, os princípios de projeto que o guiaram, e os processos que foram utilizados para aperfeiçoar o protocolo em escala global.

Entre as contribuições do artigo, destacam-se a descrição detalhada do projeto e implementação do QUIC, incluindo aspectos como estabelecimento de conexão, criptografia, multiplexação de fluxos e controle de congestionamento e perdas. O artigo também compartilha lições aprendidas sobre os protocolos de transporte e o ecossistema da Internet a partir da implantação do QUIC. Além disso, são apresentados estruturas de experimentação utilizados no desenvolvimento do QUIC, permitindo testes contínuos em larga escala e ajustes baseados em dados coletados do uso real em produção. Isso possibilitou melho-

rias incrementais e uma evolução rápida do protocolo, garantindo ganhos cumulativos de desempenho.

Os resultados apresentados no artigo mostram que o QUIC oferece melhorias significativas em diversas métricas de desempenho. Por exemplo, o QUIC reduz a latência de busca e a latência de reprodução de vídeo, além de diminuir as taxas de *rebuffering* em comparação com o uso de TLS/TCP. Em média, o QUIC reduz a latência de respostas de pesquisa do Google em 8% para usuários de desktop e 3,6% para usuários móveis. Esses benefícios são mais evidentes em condições de alta latência e maior perda de pacotes, o protocolo reduz as taxas de *rebuffering* em reproduções de vídeo em 18% para usuários de desktop e 15,3% para usuários móveis. A análise também discute a eficiência do QUIC em termos de utilização da CPU do servidor e analisa o impacto interno do desempenho do protocolo. Dentro do Google, o QUIC já representa mais de 30% do tráfego total de saída, indicando uma adoção significativa do protocolo.

Enquanto o artigo [7] foca na implementação em escala global do protocolo, esse trabalho irá explorar as diferenças e possíveis vantagens de uma implementação no espaço do kernel em relação ao espaço do usuário. A abordagem feita no artigo [7] é útil para entender as vantagens e desvantagens da abordagem no espaço do usuário do protocolo QUIC, em termos de latência e eficiência em escala global.

2.2 Implementation and Performance Evaluation of the QUIC Protocol in Linux Kernel

Como o QUIC é um protocolo projetado para melhorar a performance de conexões HTTPS, oferecendo funcionalidades como criptografia de pacotes, multiplexação de fluxos e migração de conexões, que são desafios para o modelo de serviço TCP. O artigo *Implementation and Performance Evaluation of the QUIC Protocol in Linux Kernel* [8] explora a implementação e avaliação do desempenho do protocolo QUIC em relação ao TCP, ambos implementados no kernel do Linux, comparando os dois protocolos com testes em máquinas virtuais e redes sem fio.

Para avaliar o desempenho, é feita uma implementação do protocolo QUIC diretamente no kernel do Linux, com as modificações necessárias no código-fonte e a estrutura funcional da nova implementação. A partir disso, são feitas avaliações empíricas sobre os testes em máquinas virtuais e em um ambiente de rede WiFi para comparar o desempenho do QUIC com o do TCP. Todo o código-fonte, detalhes de implementação e arquivos de teste estão disponíveis em um repositório online, facilitando a replicação dos resultados e encorajando futuras implementações e otimizações de próximas versões do QUIC.

Nos testes realizados em máquinas virtuais, o QUIC demonstrou menor latência e maior eficiência na utilização da largura de banda, particularmente em cenários com alta taxa de perda de pacotes. O QUIC atinge seu pico de taxa de transmissão muito mais rápido que o TCP, embora ambos usem o mesmo algoritmo de controle de congestionamento. Nos testes realizados em redes sem fio, o QUIC manteve uma performance superior em comparação ao TCP, demonstrando maior resiliência e estabilidade. Os experimentos indicaram que o QUIC é capaz de manter uma taxa de transferência justa em relação ao TCP, mesmo quando ambos os protocolos operam simultaneamente em uma largura de banda limitada. Em condições de baixa latência e alta perda de pacotes, o QUIC se destacou significativamente, enquanto o TCP ainda apresentou um desempenho competitivo em cenários de baixa perda de pacotes e alta latência. Mostrando a robustez do QUIC em diversos ambientes de rede, reforçando sua potencial vantagem sobre o TCP em situações de rede mais desafiadoras.

Dessa forma, o artigo fornece toda a base para entender as vantagens e desvantagens da implementação do QUIC no kernel do Linux, como também oferece testes em diferentes condições de rede. Além disso, a disponibilidade do código-fonte e dos detalhes de implementação servem como um recurso prático para o desenvolvimento deste trabalho, completando o que já foi feito com testes em comparação às implementações feitas no espaço de usuário.

2.3 Beyond socket options: making the Linux TCP stack truly extensible

Para ser extensível, o protocolo TCP inclui no cabeçalho uma parte para adicionar novas opções. No entanto, a implementação de uma nova opção de TCP leva tempo e precisa ser aceita pela IETF para então ser usada nas principais pilhas. O artigo *Beyond socket options: making the Linux TCP stack truly extensible* [9] discute a necessidade de aumentar a extensibilidade da pilha TCP do Linux usando uma estrutura leve baseada em Extended Berkeley Packet Filter (eBPF) permitindo adicionar facilmente suporte para novas opções TCP na pilha do Linux. Os autores mostram as limitações de estender as implementações dentro do kernel e das opções de soquete atuais, que não permitem políticas específicas por conexão.

As principais contribuições incluem a proposta e implementação de uma estrutura leve baseada no eBPF que permite aos usuários adicionar suporte a novas opções TCP na pilha TCP do Linux. Foram demonstrados vários casos de uso no kernel, onde o código eBPF é injetado em execução para atualizar ou ajustar a implementação do TCP. Foram incluida opções para tempo limite no TCP, opção que permite ao cliente solicitar

ao servidor o uso de um esquema específico de controle de congestionamento, opção TCP que define a janela de congestionamento inicial, ajuste da estratégia de reconhecimento (ACK). Uma das inovações notáveis é a capacidade de ajustar dinamicamente o esquema de controle de congestionamento com base nas características da rede do cliente, algo que melhora significativamente a performance sem a necessidade de mudanças complexas na estrutura do kernel. Essa abordagem não apenas proporciona flexibilidade, mas também é implementada de forma eficiente e com baixo custo, sem a necessidade de recompilar o kernel.

Os resultados apresentados no artigo mostram que a estrutura proposta pode ser eficaz para implementar novas opções TCP e adaptar a pilha TCP a diferentes condições de rede. Por exemplo, a taxa de transferência média caiu de 9,41 Gbps para 9,38 Gbps em todos os três cenários usando eBPF. Essa pequena diminuição da taxa de transferência é principalmente devido ao tamanho do cabeçalho TCP ter aumentado. Além disso, existe no pior caso cerca de 10% a mais de sobrecarga de utilização da CPU com o uso de eBPF. Essas melhorias foram alcançadas sem aumentar significativamente a carga de processamento ou memória, destacando a eficiência e viabilidade da solução proposta.

O uso de eBPF para injetar código customizado no kernel pode ser explorado para suportar novas funcionalidades do QUIC de forma dinâmica. Com essa abordagem, são satisfeitas as necessidades de maior flexibilidade e extensibilidade nas pilhas de protocolo de rede. Enquanto o artigo utiliza eBPF para adicionar novas opções TCP de forma eficiente, esse trabalho pode explorar diferentes abordagens para comparar a eficiência em relação ao código diretamente no kernel. Portanto, a metodologia e as ferramentas apresentadas são relevantes para entender possíveis extensões ou otimizações para o QUIC no Linux.

2.4 eQUIC Gateway: Maximizing QUIC Throughput using a Gateway Service based on eBPF + XDP

O artigo *eQUIC Gateway: Maximizing QUIC Throughput using a Gateway Service based on eBPF + XDP* [10] aborda a criação e avaliação de um serviço de *gateway* que utiliza o Extended Berkeley Packet Filter (eBPF) e o Express Data Path (XDP) para otimizar a taxa de transferência de conexões QUIC. O serviço proposto atua como um ponto de entrada num sistema distribuído, controlando quais pacotes do protocolo QUIC podem passar para aplicação. O módulo de kernel se encarrega do processamento de filtragem de pacotes com base em informações compartilhadas em tempo real com a aplicação no espaço

do usuário, permitindo maior controle e eficiência na filtragem de pacotes diretamente no kernel.

As contribuições do artigo incluem uma nova arquitetura que divide a aplicação em duas partes: um módulo de kernel que implementa um *gateway* utilizando eBPF e XDP, e uma biblioteca em espaço de usuário que permite a comunicação com o módulo de kernel, permitindo a comunicação e o controle eficiente do tráfego de entrada. Para testar a aplicação, foram feitos experimentos que compararam o *gateway* no espaço de kernel em relação a um feito no espaço de usuário. Os testes foram feitos utilizando um servidor HTTPS que sempre manda requisições com 256 kilobytes de texto.

Os resultados dos testes indicam melhorias em vários aspectos de desempenho. A implementação no espaço do kernel aumentou a taxa de transferência em 30,9%, reduziu em 65% a duração média de uma única requisição HTTPS e diminuiu o tempo de CPU necessário para filtrar pacotes em 26,4%. Além disso, houve uma redução de 89% no número de chamadas de sistema executadas durante os testes. Esses resultados demonstram como essa arquitetura pode evoluir para se tornar uma solução de rede definida por software no espaço de kernel.

A criação de um serviço de filtragem de pacotes no espaço do kernel oferece notáveis benefícios de desempenho. Entretanto, este trabalho se concentra na comparação do protocolo QUIC diretamente no espaço do kernel. Porém, utilizar eBPF para injetar código customizado no kernel permite suportar novas funcionalidades do QUIC de forma dinâmica, sem a necessidade de alterar o kernel. Dessa forma, a metodologia e as ferramentas apresentadas são altamente relevantes para futuras implementações do protocolo QUIC, fornecendo dados e testes para a condução de avaliações e experimentos adicionais.

2.5 Automating QUIC Interoperability Testing

Devido ao QUIC poder ser inicialmente implantado sem exigir uma nova infraestrutura, foram surgindo variedades de implementações baseadas em diferentes linguagens de programação e paradigmas. O artigo *Automating QUIC Interoperability Testing* [3] trata do desenvolvimento e implementação do Quic Interop Runner (QIR) [11], uma interface de testes automatizados e sob demanda para a integração entre diferentes implementações do protocolo QUIC. A interface simula diversas condições utilizando o ns-3 [2], que executa uma série de casos de teste entre implementações do QUIC containerizadas, validando os resultados por meio do histórico e rastreamentos de pacotes. A automação oferecida pelo QIR visa superar as limitações dos métodos de teste manuais, como a escalabilidade limitada e a falta de testes precisos.

O QIR utiliza contêineres como blocos de construção básicos, permitindo que os implementadores mantenham e atualizem suas próprias implementações de forma independente. Isso facilita a inclusão de implementações de código fechado e permite testes contínuos e sob demanda. A interface já é suportado por dez implementações do QUIC e se integrou ao fluxo de desenvolvimento de diversos implementadores, demonstrando a necessidade e a eficácia da automação nos testes de interoperabilidade.

Os resultados e dados apresentados no artigo mostram que o QIR reduziu significativamente o tempo e o esforço necessários para configurar e executar testes de interoperabilidade. A capacidade de realizar testes de desempenho sob diferentes condições de rede e a execução de testes contínuos permitem a criação de uma matriz de integração, atualizada com frequência para auxiliar na depuração e no desenvolvimento do protocolo QUIC. Além disso, o artigo discute planos futuros para expandir a interface com novos casos de teste, como testes de alta largura de banda, latência de handshake e cenários mais complexos que simulam redes WiFi e celulares.

A infraestrutura descrita no artigo, incluindo o uso de contêineres e simulação de condições de rede com o ns-3, pode ser adaptada para criar cenários de teste específicos que avaliem o desempenho, a resiliência e a conformidade das implementações do QUIC. Como QIR se concentra na integração entre várias implementações no espaço de usuário, esse trabalho pode construir uma implementação no espaço do kernel para integrar ao que já existe no QIR. Portanto, as ferramentas e ideias sobre Quic Interop Runner (QIR) são extremamente úteis para estruturar os testes e análises neste trabalho, garantindo uma avaliação abrangente e robusta das duas abordagens (espaço do kernel e espaço do usuário).

2.6 QUIC on the Highway: Evaluating Performance on High-rate Links

O artigo *QUIC on the Highway: Evaluating Performance on High-rate Links* [12] analisa o desempenho de diferentes implementações do QUIC em conexões de alta taxa de transmissão. Para isso, os autores desenvolveram uma plataforma de testes baseada no *QUIC Interop Runner* [3, 11] que permite a utilização de diferentes implementações do QUIC. A plataforma automatiza a configuração dos servidores, coleta diversas estatísticas e facilita a transformação e avaliação dos resultados. A pesquisa se concentra principalmente nas implementações LSQUIC da LiteSpeed Tech [13] e quiche da Cloudflare [14], avaliando os fatores que limitam o desempenho em conexões de 10 Gbit/s.

O estudo mostra diferenças no comportamento das implementações lsquic e quiche, como parâmetros padrão divergentes e abordagens distintas quanto à taxa de envio de

confirmações (ACKs). Entre as contribuições do artigo, destaca-se a constatação de que o desempenho das implementações varia significativamente conforme a configuração do sistema e o hardware utilizado. Testes adicionais indicam que aumentar a área de recepção e aplicar otimizações específicas para as arquiteturas de CPU podem aumentar substancialmente o desempenho das implementações do QUIC.

Um dos resultados importantes foi a descoberta de que a área de memória do UDP é por padrão muito pequena, resultando em perda de pacotes no lado do receptor e consequentemente na redução de transferência. Implementações como LSQUIC e quiche apresentaram diferenças de desempenho devido a variações nos parâmetros padrão e na gestão de recursos, como o *buffer* de recepção e o processamento da CPU. Aumentar o tamanho do *buffer* é essencial para melhorar o desempenho em cenários de alta taxa de transferência. A análise também revela que o QUIC atinge transferências similares com as cifras AES-NI e ChaCha20, sendo o principal gargalo ainda a entrada e saída de pacotes. Além disso, as técnicas de descarregamento de segmentação de pacotes no Linux não impactam tanto a taxa de transferência do QUIC, ao contrário do TCP, que se beneficia largamente do descarregamento. Essas informações são importantes para a implementação eficiente do QUIC em redes de alta velocidade, sugere que ajustes específicos podem mitigar perdas de desempenho e maximizar a utilização da largura de banda disponível.

O artigo analisado oferece um meio de partida para o desenvolvimento da comparação entre as implementações do protocolo QUIC no espaço do kernel e no espaço do usuário. Primeiramente, a interface aprensentada é particularmente útil para entender as nuances de diferentes implementações do QUIC e explorar como ajustes específicos podem influenciar o desempenho em diversos ambientes de rede. Além disso, a capacidade de automatizar e padronizar testes de desempenho entre diferentes implementações permite uma análise de gargalos e oportunidades de melhoria mais consistente, podendo a interface ser adaptada para medir e comparar o desempenho de novas implementações do protocolo, como uma implementação no espaço do kernel sem a necessidade de reimplementar toda a interface. Assim, o artigo não só destaca fatores que impactam o desempenho do QUIC que podem ser explorados mais a fundo, mas também fornece um ponto de partida para a análise das implementações no espaço do kernel e espaço de usuário.

2.7 Evaluating QUIC Performance Over Web, Cloud Storage and Video Workloads

O artigo *Evaluating QUIC Performance Over Web, Cloud Storage and Video Workloads* [15] analisa o desempenho do protocolo QUIC em comparação com o TCP, focando especificamente em cargas de trabalho da web, armazenamento em nuvem e streaming de

vídeo. O estudo foi realizado usando testes especificamente desenvolvidos para medir a latência e a taxa de transferência das conexões, além de outros indicadores de desempenho. A análise foi realizada em redes tanto na Alemanha quanto na Índia, para capturar uma variedade de cenários, com larguras de banda e tempos de resposta diferentes.

Os experimentos foram conduzidos utilizando três bancos de teste (*quic_perf*, *tls_perf* e *video*), localizados na Alemanha e na Índia, e com dois tipos de ambientes: um ambiente baseado em VM representando uma rede quase ideal e um ambiente com uma placa Raspberry Pi 3 que permite realizar medições em redes residenciais. Cada banco de testes foi configurado para replicar diferentes condições reais de rede, variando a largura de banda e a latência para simular diferentes ambientes de uso. Além disso, os testes de vídeo avaliaram a qualidade de transmissão, observando eventos como interrupções e a utilização de CPU durante a reprodução. A coleta de dados foi realizada ao longo de três anos (2018-2021), com o objetivo de acompanhar a evolução e identificar tendências.

Os resultados mostrados por Tanya *et al.* [15] revelam que o QUIC pode superar o TLS/TCP em várias métricas de desempenho, especialmente em ambientes onde a latência e o tempo de ida-e-volta são altos. Em todos os testes, o tempo de conexão foi reduzido em média 140% em comparação com TLS/TCP. Para cargas de trabalho de armazenamento em nuvem, o QUIC apresentou maior taxa de transmissão para arquivos menores que 20 Megabytes. Porém, para arquivos maiores que 20 Megabytes, o TCP foi superior e o QUIC apresentou maior utilização de CPU. Apesar dessas limitações, o QUIC demonstrou uma melhor entrega de conteúdo de vídeo, com menos eventos de parada e menores durações de interrupção, cerca de 10% a menos comparado ao TLS/TCP. No entanto, segundo Tanya *et al.* [15] os resultados variam dependendo do sistema que serve o conteúdo, revelando a influência dos Content Delivery Networks (CDNs) no desempenho do QUIC.

Embora o foco do artigo seja na comparação do QUIC com o TLS/TCP, as técnicas utilizadas para medir e analisar o desempenho do QUIC são diretamente aplicáveis ao estudo entre implementações no espaço do kernel e do usuário do protocolo QUIC, e podem ser adaptadas no futuro. Além disso, os dados coletados ao longo de três anos, como a latência, o *throughput*, e a utilização da CPU, podem ser usadas para comparação do trabalho atual, mostrando possíveis vantagens da abordagem do QUIC no kernel.

2.8 Managing Connections by QUIC-TCP Racing: A First Look of Streaming Media Performance Over Popular HTTP 3 Browsers

O artigo *Managing Connections by QUIC-TCP Racing: A First Look of Streaming Media Performance Over Popular HTTP 3 Browsers* [16] explora o desempenho da transmissão de mídia em navegadores que suportam HTTP/3, focando na análise das diferenças de performance entre conexões QUIC e TCP durante a reprodução de vídeo em navegadores web. O objetivo principal é entender como a troca frequente entre esses protocolos afeta a experiência do usuário, especialmente em condições de rede variáveis. Para isso, os autores exploram como funciona a troca de protocolos nos navegadores Chromium e Mozilla Firefox e tentam entender a causa do problema.

Entendendo como funciona a troca de protocolos nos navegadores, os autores identificaram que a troca desnecessária de conexões entre QUIC e TCP, ocorre frequentemente em condições de rede com alta latência ou variabilidade em parte devido a *middleboxes* que bloqueiam ou limitam o tráfego UDP. Para resolver essa limitação, os autores propõem uma modificação no navegador Chromium para desabilitar a troca frequente, permitindo que as conexões QUIC sejam mantidas por mais tempo. Além disso, é sugerido uma solução dinâmica que ajuda a distinguir entre perdas de pacotes causadas por redes ruins e aquelas causadas por firewalls, aprimorando ainda mais a estabilidade das conexões.

Os resultados apresentados mostram que a versão modificada do navegador Chromium, que desabilita a troca de protocolos, apresenta melhorias significativas em relação à versão original em termos de taxas de bits médio, variação da taxas de bits e *rebuffering*. Os testes realizados confirmam que a modificação permite que as conexões QUIC sejam mantidas por mais tempo, resultando em uma melhor experiência de streaming em mais de 60% dos casos analisados. Além disso, a implementação de uma solução que decide dinamicamente quando habilitar a troca de conexões mostrou-se ainda mais eficaz, melhorando a qualidade de experiência em mais de 70% dos casos ao identificar quando o desempenho de rede é baixo ou quando um *middlebox* está limitando o tráfego.

As metodologias e métricas de avaliação utilizadas, como bitrate médio e rebuffering, são diretamente úteis para a análise do trabalho atual. O artigo fornece uma visão dos desafios e benefícios de usar QUIC em navegadores modernos, o que é relevante para entender o comportamento em diferentes condições de rede e possíveis otimizações a nível de sistema, podendo no futuro explorar mais a fundo o uso de uma implementação do kernel do QUIC em navegadores.

2.9 Characterization and Prediction of QUICs Performance Under Different Network Conditions

Segundo *Siag et al.* [17], o QUIC permanece limitado pois a maior parte dos trabalhos existentes se concentram na comparação em relação ao TCP em cenários específicos. O artigo *Characterization and Prediction of QUICs Performance Under Different Network Conditions* [17] destaca a necessidade de entender melhor o desempenho do QUIC para tornar a compreensão do protocolo mais abragente em diferentes ambientes. Para isso, os autores analisam os efeitos de diversos parâmetros de rede do QUIC e identificam seus pontos fortes e fracos.

Para estimar o desempenho do QUIC são utilizados como parâmetros: perda de pacotes, reordenação de pacotes (*jitter*), latência, fluxos paralelos por conexão e largura de banda. Além de avaliar esses efeitos, é desenvolvido um modelo preditivo capaz de estimar o desempenho do QUIC para diferentes cenários de rede. Esse modelo tem como objetivo prever métricas como transmissão efetiva de dados (*goodput*) e o tempo para download de arquivos (TTA). Todos os experimentos são realizados em um ambiente de teste controlado, para evitar interferência de fatores externos.

Os resultados do estudo mostram que a perda de pacotes e o *jitter* têm impactos significativos no desempenho do QUIC. Por exemplo, o *goodput* do QUIC cai drasticamente com pequenas perdas de pacotes, e o *jitter* leva a uma redução acentuada no desempenho devido à chegada fora de ordem dos pacotes. *Siag et al.* [17] ressaltam a necessidade de aprimorar o QUIC para lidar melhor com a reordenação de pacotes, sugerindo que futuras melhorias poderiam aumentar a robustez e a previsibilidade do protocolo.

Enquanto o foco do artigo está na caracterização do desempenho do QUIC em condições adversas de rede, o trabalho a ser desenvolvido pode expandir essa análise para verificar como as diferentes implementações respondem a essas mesmas condições. Os modelos preditivos e as métricas de desempenho desenvolvidos no artigo, como *goodput* e tempo de download, podem ser aplicados para avaliar o desempenho de diferentes implementações do protocolo, por exemplo, a potencial vantagem de uma implementação no kernel em termos de latência e eficiência.

2.10 Performance Evaluation of QUIC Vs. TCP for Cloud Control Systems

O artigo *Performance Evaluation of QUIC Vs. TCP for Cloud Control Systems* [18] explora a comparação de desempenho entre os protocolos de transporte QUIC e TCP em sistemas de controle em nuvem. O objetivo é determinar se o QUIC pode atender aos requisitos de Quality of Service (QoS) necessários para sistemas de controle em nuvem. Além de entender qual protocolo oferece melhor desempenho em termos de latência, largura de banda, perda de pacotes e estabilidade em diferentes condições de rede.

A pesquisa se concentra em dois ambientes de nuvem distintos: um *cluster* de borda privado e uma infraestrutura de nuvem pública, utilizando diferentes cenários de rede como 5G, Ethernet LAN e Internet. Explorando o comportamento do QUIC nesses diversos cenários, busca se compreender se o protocolo QUIC pode oferecer vantagens significativas sobre o TCP em termos de latência, *jitter* e estabilidade de conexão. Além disso, são investigados se a presença de firewalls ou NATs influencia o desempenho do QUIC, que foi projetado para contornar alguns dos problemas enfrentados pelo TCP.

Em termos de volume de dados, o QUIC gerou menos dados por solicitação devido ao TCP ter um tamanho de cabeçalho maior que o UDP e o QUIC. Além disso, o QUIC apresentou tempos de resposta ligeiramente superiores ao TCP, mas com menor variação de jitter. Segundo *Haorui et al.* [18], o desempenho dos protocolos varia conforme as características do sistema de controle e o ambiente de rede. A análise sugere que o QUIC pode ser uma melhor escolha em cenários onde a latência e a variação de tempo são críticas, como no 5G. Entretanto, o TCP ainda mostrou desempenho superior em termos de tempo de resposta em cenários com menor perda de pacotes, como no ambiente AWS.

Embora o foco do artigo seja principalmente na análise de desempenho em redes de controle em nuvem, as metodologias utilizadas, como a medição de latência, jitter e tempo de resposta, podem ser aplicadas diretamente ao passo que o trabalho desenvolvido trata das implicações do protocolo a nível do sistema operacional. Além disso, os dados apresentados no artigo, como os cenários de teste e as configurações de rede, servem como base ou ponto de referência para desenvolver experimentos similares.

2.11 Síntese

No geral, embora o QUIC demonstre várias vantagens sobre o TCP, particularmente em condições de rede sensíveis à latência e de alta perda, ainda há espaço para otimização, especialmente em relação a implementações que visam conectar o kernel e o espaço do usuário. A motivação para desenvolver uma análise comparativa entre a implementação

do kernel e outras implementações do protocolo QUIC originou-se tanto do artigo *Automating QUIC Interoperability Testing* [3], quanto do *QUIC on the Highway: Evaluating Performance on High-rate Links* [12]. Ambos os estudos concentram-se em testar e comparar diversas implementações do QUIC no espaço do usuário. Porém, nenhum trabalho compara uma integração do protocolo QUIC no kernel em relação a uma implementação no espaço de usuário.

Os artigos [16, 15, 17, 18] fornecem extensas comparações de desempenho entre o protocolo QUIC e o TCP em diversos ambientes. O artigo [16] discute o impacto dos mecanismos de corrida de conexão em navegadores, que degradam a Qualidade da Experiência (QoE) em aplicativos como streaming de vídeo. O artigo [15] destaca a latência mais baixa do QUIC, mas identifica maior uso de CPU do que o TCP, atribuindo a vantagem do TCP às otimizações do kernel. Além disso, os artigos [17, 18] exploram o comportamento do QUIC sob condições de rede variáveis. O artigo [17] desenvolve um modelo de desempenho preditivo para o QUIC, destacando parâmetros cruciais para otimizar o desempenho do protocolo. O artigo [18] avalia a adequação do QUIC para sistemas de controle de nuvem, sugerindo que otimizações em nível de kernel podem fornecer benefícios de desempenho consistentes para ambientes sensíveis a Quality of Service (QoS).

Além das comparações entre QUIC e TCP em diferentes cenários, outros estudos focam especificamente em implementações dentro do kernel, destacando vantagens em desempenho e eficiência. O artigo [8] demonstra a implantação do QUIC no kernel do Linux, destacando o desempenho do QUIC ao comparar com o TCP. O artigo [10] apresenta o eQUIC Gateway, um módulo do kernel usando eBPF para filtragem de pacotes, que aproveita o *offloading* de pacotes no espaço do kernel, apresentando melhor rendimento, latência reduzida e menor consumo de CPU, particularmente sob condições de ataque. Adicionalmente, outros artigos serviram de base para entender a tecnologia eBPF e seu potencial. O artigo [9] apresenta a flexibilidade da adaptabilidade do kernel através do eBPF para aprimoramentos do TCP, permitindo ajustes e melhorias dinâmicas ao protocolo no kernel. Esses artigos são essenciais para compreender os benefícios e as limitações práticas das implementações no kernel.

Dessa forma, os trabalhos analisados destacam aspectos fundamentais do protocolo QUIC, como desempenho, eficiência, interoperabilidade e o potencial da integração direta ao kernel do Linux. Esses estudos fornecem uma base sólida para justificar a relevância da comparação a ser realizada neste trabalho entre implementações no kernel e implementações no espaço de usuário, além de evidenciar lacunas ainda não abordadas atualmente.

Capítulo 3

Metodologia

A principal hipótese que orienta esse trabalho é de que a implementação no kernel possa apresentar um desempenho superior a uma implementação feita no espaço de usuário, sobretudo em termos de uma vazão mais elevada na transmissão de dados reais. Essa suposição baseia-se na premissa de que a integração em nível de kernel potencializa o uso de recursos do sistema e reduz o *overhead* de chamada de funções.

Nas seções seguintes, será abordado o detalhamento e o planejamento para a condução dos experimentos, a fim de realizar uma análise aprofundada da comparação entre implementações do QUIC. Esta abordagem visa fornecer evidências que possam confirmar ou refutar a hipótese inicial, esclarecendo o potencial das implementações em nível de kernel frente às implementações no espaço do usuário.

3.1 Cenário

O cenário proposto utiliza contêineres Docker [1], baseados nos testes de interoperabilidade descritos no artigo *Automating QUIC Interoperability Testing* [3]. Três contêineres são configurados: um servidor, uma rede e um cliente, os quais são interconectados para simular uma típica rede de comunicação cliente-rede-servidor. Cada contêiner com uma implementação do QUIC pode ser executado como um servidor ou como um cliente, dependendo do seu papel. Para a gestão e configuração do ambiente dos contêineres, utiliza-se o Docker Compose [19]. O cenário modelado foi elaborado de modo a permitir adaptações a várias condições de rede que incluem limitação de banda, inserção de erros e atraso usando o ns-3 [2]. Os componentes e recursos que compõem o cenário são detalhados a seguir.

- **Cliente** é o dispositivo que inicia as conexões e solicita a transferência de dados ao servidor, como o download de arquivos e recebimento dos pacotes de resposta.

- **Rede Emulada** é a ponte que fornece ligação entre o cliente e o servidor. Tem a função de monitorar os experimentos, coletar dados e introduzir perdas usando o ns-3.
- **Servidor** é o dispositivo que responde às requisições do cliente, armazenando e transferindo os arquivos solicitados pelo cliente.

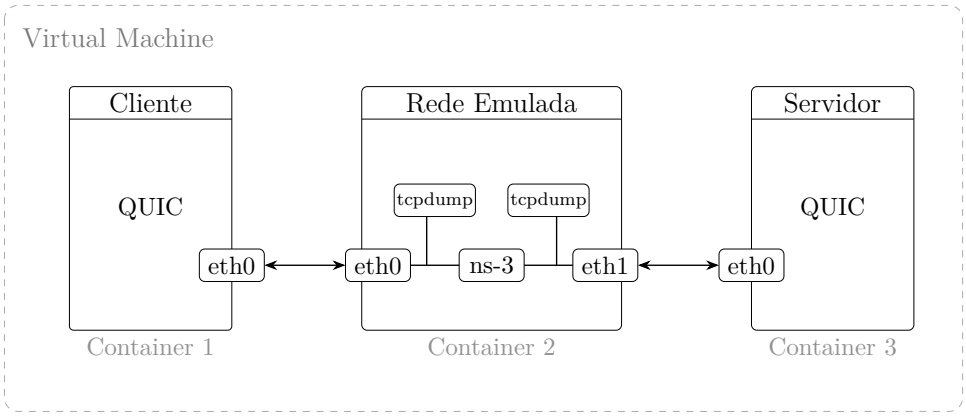


Figura 3.1: Configuração do cenário.

Na Figura 3.1, o Container 1 representa um cliente QUIC, que se comunica com o Container 3, um servidor QUIC, através do Container 2, que emula a rede e aplica condições como perda de pacotes e atraso. A comunicação ocorre via as interfaces de rede (eth0 no cliente e servidor, eth0/eth1 na rede emulada), garantindo que todo o tráfego passe pelo simulador ns-3, onde a ferramenta *tcpdump* [20] captura pacotes para futura análise. Os testes foram realizados dentro de uma máquina virtual para garantir o isolamento da rede da máquina principal, evitando interferências externas e assegurando maior consistência e reprodutibilidade dos resultados. Adicionalmente, o cenário foi ampliado com a integração de ferramentas que analisam os dados coletados e monitoram chamadas de sistema.

3.2 Métricas

Para analisar de forma clara e objetiva o desempenho e a eficácia da proposta, é indispensável o uso de métricas bem definidas. Ao estabelecer indicadores alinhados aos objetivos do estudo, assegura-se maior transparência nos métodos e confiabilidade nos resultados quantificados. Logo, a partir das metodologias empregadas e dos objetivos pretendidos, foram estabelecidas 2 principais métricas, conforme descrito em detalhe a seguir.

1. Taxa de Transferência mede a quantidade de dados transmitidos por unidade de tempo. Uma maior taxa de transferência indica uma implementação capaz de transmir maior volume de dados. Para medir a taxa de transferência calcula-se o número total de bytes transferidos com sucesso dividido pelo tempo total gasto na transferência. Uma taxa de transferência elevada sugere o maior uso da rede, o que é desejado para maximizar o uso da rede.

2. Número de Chamadas de Sistema mede a quantidade de chamadas feitas por um programa ao sistema operacional. Um número reduzido de chamadas de sistema indica uma menor latência, resultante da diminuição na frequência de trocas de contexto entre a aplicação e o sistema operacional. Para aferir essa métrica, é preciso monitorar em tempo de execução cada chamada de sistema feita pela aplicação ao kernel. Essa métrica é relativamente útil porque quanto menor a troca de contexto, menos recursos e tempo são consumidos, tornando a execução da aplicação mais eficiente.

O uso dessas métricas é justificado pela necessidade de avaliar com clareza os principais aspectos de desempenho das implementações do protocolo QUIC. A Taxa de Transferência indica a eficiência na transmissão de dados, enquanto o Número de Chamadas de Sistema reflete a eficiência interna da aplicação em relação ao uso dos recursos do sistema operacional. Dessa forma, essas duas métricas são combinadas para fornecer uma análise abrangente tanto da performance do uso da rede quanto do uso de recursos do sistema das implementações avaliadas.

3.3 Protótipo

Neste experimento visa-se observar o impacto da perda de pacotes no desempenho das métricas estabelecidas, serão desenvolvidos quatro cenários distintos, todos configurados com uma largura de banda constante de 10 Mbps, um atraso de transmissão de 15 ms e uma fila com capacidade para 25 pacotes, diferenciando-se cada cenário pela taxa de perda de pacotes em 0%, 1%, 5% e 10%. O limite de banda de 10 Mbps foi escolhido para realizar simulações em diferentes computadores sem sobrecarregar seus recursos e permite analisar como diferentes aplicações se comportam sob restrições de largura de banda, simulando cenários reais de redes com conexões lentas.

Para garantir a simplicidade dos testes, será utilizado um esquema de solicitação-resposta simplificado baseado no protocolo HTTP/0.9, operando sobre os fluxos do QUIC. Em cada fluxo, será enviada apenas uma solicitação HTTP/0.9, na qual o cliente requisi-

tará o download de um arquivo específico. O servidor, por sua vez, responderá enviando sempre um arquivo de 10 megabytes.

Além disso, observou-se que as interações entre cliente e servidor podem variar quando são utilizadas diferentes implementações no cliente e servidor, uma vez que os modos de operação e características das implementações podem influenciar de maneiras não uniformes. Por esse motivo, para investigar o funcionamento do protocolo de forma mais controlada, optou-se por realizar os testes utilizando implementações idênticas tanto no cliente quanto no servidor.

Os testes serão baseados em situações reais de uso, como transferência de arquivos, estabelecimento de conexões e resolução de perdas de pacotes. Para comparar as implementações do protocolo QUIC, são utilizadas três implementações: duas versões com a implementação do QUIC no espaço do usuário (*msquic*[21] e *ngtcp2*[22]) e uma versão do QUIC para o kernel do Linux.

A rede emulada foi baseada no teste de interoperabilidade descrito no artigo *Automating QUIC Interoperability Testing* [3, 23] e configurada para permitir a injeção de perda de pacotes. Com essa abordagem, foi possível avaliar o funcionamento do protocolo em condições que variam de cenários moderados a extremos, possibilitando investigar como diferentes níveis de perda influenciam a transferência de dados e a estabilidade das conexões.

Por fim, foram realizadas múltiplas execuções dos testes para garantir consistência e reprodutibilidade dos resultados. Replicações de 30 experimentos foram realizadas para cada aspecto comparativo. Para cada replicação realizada, foram calculadas médias com intervalos de confiança variáveis a um nível de confiança de 95%. Todas as implementações avaliadas foram executadas em um ambiente homogêneo, mantendo-se a mesma configuração de hardware e software, de modo a reduzir interferências externas no desempenho medido.

3.3.1 Implementação do QUIC no Espaço de Usuário

As imagens utilizadas referentes às implementações *msquic* [21] e *ngtcp2* [22] foram obtidas diretamente do DockerHub [24]. Inicialmente, é necessário identificar as imagens oficiais correspondentes às versões das implementações de QUIC a serem testadas. Em seguida, foi utilizado o comando *docker pull* com as tags específicas, depois é feito o download dessas imagens para o ambiente de testes, garantindo que todas as dependências e configurações pré-definidas fossem mantidas conforme os parâmetros estabelecidos pelos desenvolvedores. Após o download, as imagens foram armazenadas localmente e utilizadas para os testes de cliente e servidor, assegurando a padronização e a reprodutibilidade dos experimentos.

```
docker pull ghcr.io/microsoft/msquic/qns:main
docker pull ghcr.io/ngtcp2/ngtcp2-interop:latest
```

Figura 3.2: Comando bash para download das imagens do *msquic* e *ngtcp2*.

As implementações *msquic* [21] e *ngtcp2* [22] do protocolo QUIC foram escolhidas para os testes devido às suas características distintas. A biblioteca *msquic*, desenvolvida pela Microsoft, foi escolhida por conta de sua conformidade com as mais recentes especificações do QUIC e sua capacidade de alcançar altas taxas de transferência, oferecendo otimizações e desempenho em cenários de alta demanda. Por outro lado, a biblioteca *ngtcp2* foi escolhida por causa de sua implementação simples e amplo uso em aplicações conhecidas, como é o caso da ferramenta *curl* [25]. Optou-se por limitar a escolha por essas duas implementações para manter o escopo dos testes gerenciável e evitar a complexidade excessiva que poderia ocorrer de incorporar múltiplas implementações, assegurando assim que os resultados fossem significativos e práticos para a análise.

3.3.2 Implementação do QUIC no Kernel

Para usar o container com a versão do QUIC no kernel do Linux, foi necessário compilar o módulo para o kernel do Linux [26] e instalar no sistema operacional da máquina virtual, só para assim montar o contêiner para os testes. Para isso, foi preciso instalar o Docker [1] e o Docker Compose [19] na máquina virtual, e executar o script em bash a seguir.

```
sudo apt install make autoconf automake libtool \
    pkg-config gnutls-dev linux-headers-$(uname -r) -y
git clone https://github.com/lxin/quic \
    && cd quic/
./autogen.sh
autoreconf -i
./configure --prefix=/usr
make
sudo make install
cd ..

git clone https://github.com/quic-interop/quic-network-simulator \
    && cd quic-network-simulator/
cp -r ../quic/tests/interop linuxquic
CLIENT="linuxquic" SERVER="linuxquic" docker compose build
```

Figura 3.3: Bash script para download, compilação e instalação do módulo da implementação do QUIC no kernel do linux, e montagem da imagem docker da implementação, usando Ubuntu 24.04 como sistema operacional.

Normalmente, para enviar arquivos de um servidor para um cliente, é necessário copiar os dados do arquivo para a aplicação (espaço do usuário) usando a chamada *read* e, em seguida, transferir os mesmos dados de volta através do soquete para o espaço do kernel com a chamada *sendmsg*.

```
while(nao_terminou_de_ler)
{
    char buffer[BUFSIZE];
    int n = read(filefd, buffer, BUFSIZE);
    sendmsg(sockfd, buffer, n, 0);
}
```

Figura 3.4: Pseudo código da leitura e envio de um arquivo, *filefd* representa o descritor do arquivo a ser enviado no socket *sockfd*.

Entretanto, essa abordagem é ineficiente, pois requer que o kernel copie os dados para o espaço do usuário e, em seguida, copie os mesmos dados novamente para o kernel. A chamada de sistema *sendfile* elimina essas duas etapas, permitindo que o kernel leia os dados diretamente do arquivo e os envie para o soquete em uma única operação. Contudo, em conexões criptografadas, os dados precisam ser criptografados antes de serem transmitidos e, na maioria das vezes, o kernel não realiza esse processamento.

```
while(nao_terminou_de_ler)
{
    sendfile(sockfd, filefd, NULL, FILESIZE)
}
```

Figura 3.5: Pseudo código do uso da função *sendfile*.

Por outro lado, em uma implementação do protocolo QUIC no kernel, esse procedimento se torna mais direto, pois o kernel efetua a cópia e a criptografia de maneira conjunta. No entanto, apesar da criptografia estar integrada na implementação no kernel do Linux [26], a versão utilizada para os testes ainda não implementou chamadas de sistema, como a chamada *sendfile*, devido ao seu recente desenvolvimento.

3.3.3 Testes e Medições

Serão realizadas dois tipos de medições para investigar o desempenho da implementação no kernel e compará-lo com as versões em espaço de usuário. Inicialmente, os testes ocorrerão na rede emulada, onde a ferramenta *tcpdump* [20] será utilizada para capturar e registrar todos os pacotes durante os experimentos. Todos os parâmetros de configuração

relevantes para a implementação cliente/servidor, como endereço IP, porta e arquivos de certificado, são passados por meio de variáveis de ambiente.

Além disso, durante a execução dos testes, a ferramenta *bpftrace* [27] será utilizada para monitorar e contabilizar as chamadas de sistema realizadas por cada implementação do QUIC. A ferramenta *bpftrace* [27] permite criar scripts que definem pontos de rastreamento (probes) para interceptar eventos específicos do kernel, como chamadas de sistema, interrupções, eventos de rede e outros. Para isso, o *bpftrace* será executado na máquina virtual, e o nome do processo será informado de modo que somente as chamadas associadas à implementação em questão sejam registradas.

A Figura 3.6 apresenta o script utilizado pelo *bpftrace* para monitorar chamadas de sistema. Por meio do ponto de rastreamento "raw_syscalls:sys_enter", o script intercepta cada chamada de sistema e, para cada evento, verifica se o nome do processo corresponde ao processo da implementação do QUIC. Caso essa condição seja satisfeita, o contador associado a esse processo, armazenado em um mapa associativo *@ncalls*, é incrementado, registrando o número de chamadas realizadas. Ao final do teste, no bloco "END", imprime o conteúdo do mapa e o limpa, mostrando o número total de chamadas de sistema feitas pela implementação do QUIC.

```
tracepoint:raw_syscalls:sys_enter {
    if (comm == str($1)) {
        @ncalls[pid, comm]=count();
    }
}

END {
    print(@ncalls);
    clear(@ncalls);
}
```

Figura 3.6: Script utilizado no bpftrace para contar o número de chamadas feitos por um programa.

Ao final dos testes, as ferramentas de monitoração são interrompidas, os dados coletados são copiados dos containers utilizando o comando *docker cp* e verifica-se se todos os arquivos foram transferidos com sucesso. Em seguida, a ferramenta *TShark* [28] será empregada para aplicar filtros e extrair os pacotes relevantes, que servirão de base para o cálculo do *goodput*, que é calculado a partir do tamanho do arquivo enviado dividido pelo

tempo entre o início e o término do envio do arquivo. No final, são feitos os cálculos das médias e dos erros das métricas.

Por fim, visando garantir a transparência, reprodutibilidade e facilitar futuras extensões deste estudo, todos os scripts utilizados para a execução dos testes e coleta de dados mencionados estão disponíveis publicamente em um repositório no *GitHub*[29]. Dessa forma, qualquer interessado pode acessar, revisar e reproduzir facilmente os experimentos realizados a seguir.

Capítulo 4

Resultados

O principal objetivo deste experimento é comparar os resultados das métricas obtidas na implementação do protocolo QUIC no kernel com as versões executadas no espaço do usuário, em cenários com diferentes taxas de perda de pacotes na rede. Para os testes, foi utilizada a plataforma de virtualização Oracle VirtualBox [30], configurando-se uma máquina virtual com 4 núcleos Intel Xeon E5-2667v4 e 8 GB de RAM. O sistema operacional escolhido foi o Ubuntu 24.04, que proporciona um ambiente de desenvolvimento compatível com as principais bibliotecas e ferramentas necessárias para a avaliação das implementações.

4.1 Taxa de transferência (Goodput)

Neste experimento, o foco foi medir a taxa de transferência de cada implementação do QUIC em uma rede com limite de banda e diferentes níveis de perda de pacotes. A Figura 4.1 destaca no eixo vertical do gráfico, a taxa de transferência em kilobits por segundo de três diferentes implementações do protocolo QUIC (kernel, msquic e ngtcp2), sob diferentes taxas de perda de pacotes no eixo horizontal. As barras no gráfico indicam a média aritmética da taxa de transferência, calculada a partir de 30 repetições do mesmo experimento, enquanto as barras de erro ilustram o coeficiente de variação percentual, refletindo a variabilidade nos dados observados.

Ao analisar os dados exibidos na Figura 4.1, observa-se que na ausência de perda de pacotes, a implementação do QUIC no kernel conseguiu em média alcançar 88.3% da taxa de transferência máxima de 10 Mbps, enquanto as implementações *msquic* e *ngtcp2* registraram uma taxa de transferência média de 89.3% e 74.5% de 10 Mbps, respectivamente. É possível ver que o *msquic* apresenta um desempenho inicial comparável ao kernel em 0% de perda, mas sofre uma redução mais acentuada à medida que a taxa de perda aumenta, especialmente evidente no salto para 5% de perda, onde o desempenho

começa a degradar ao ponto de não conseguir enviar mais dados quando a perda é de 10%. O *ngtcp2*, por outro lado, tem baixo desempenho mesmo com baixas taxas de perda, deteriorando-se significativamente à medida que a perda de pacotes cresce. Porém, apesar do *ngtcp2* não possuir uma alta taxa de transferência, observa-se que a taxa de transferência não cai drasticamente ao ponto de não conseguir enviar dados.

Essa característica pode ser vantajosa em cenários onde a continuidade da transmissão é essencial, mesmo que isso signifique operar com uma taxa menor. Em suma, a escolha da implementação ideal dependerá do equilíbrio desejado entre performance máxima e robustez sob condições adversas. A implementação no kernel surge como uma solução para aplicações críticas que exigem alta disponibilidade e desempenho constante.

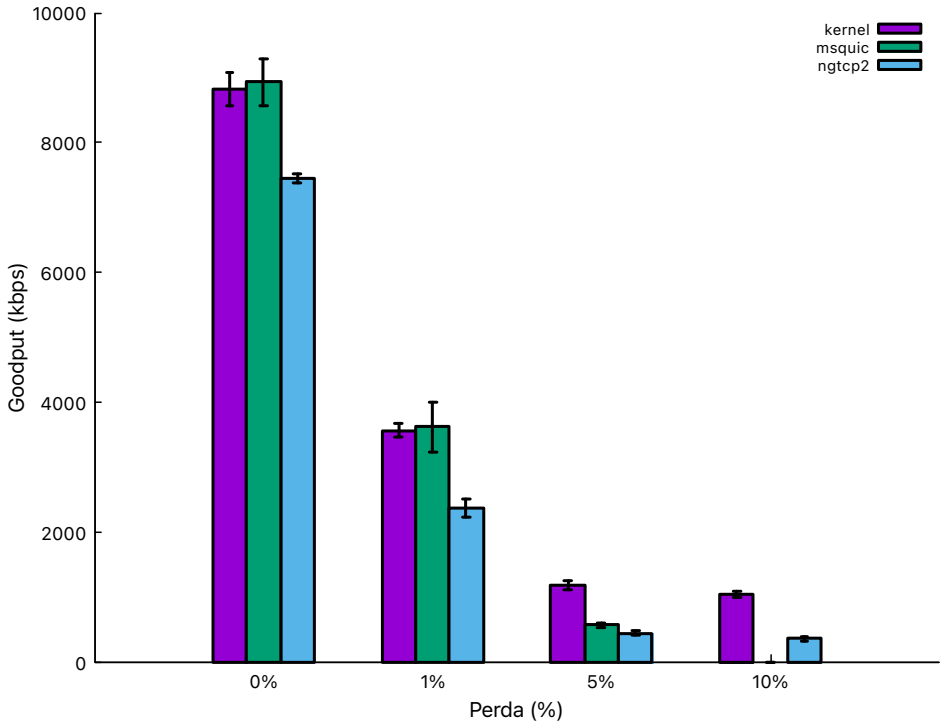


Figura 4.1: Taxa de transferência de cada implementação para diferentes taxas de perda.

Tabela 4.1: Porcetagem de diferença da taxa de transferência entre as taxas de perda de 1%, 5% e 10%, comparadas ao cenário com 0% de perda.

	1%	5%	10%
ngtcp2	-68.02%	-93.91%	-95.1%
msquic	-59.43%	-93.54%	-100%
kernel	-59.51%	-86.5%	-88.2%

4.2 Chamadas de sistema

Este teste tem como objetivo avaliar a eficiência de diferentes implementações do protocolo QUIC, medindo o número de chamadas de sistema realizadas por cada implementação, sob diferentes taxas de perda de pacotes. No eixo horizontal do gráfico, representam-se as taxas de perda de pacotes, enquanto o eixo vertical indica o número total de chamadas de sistema, com os dados sendo a média de 30 repetições para cada condição testada. As barras de erro ilustram o coeficiente de variação percentual, refletindo a variabilidade nos dados observados.

A partir dos dados da Figura 4.2, à medida que a taxa de perda aumenta, a implementação baseada no espaço do usuário sofrem um crescimento no número de chamadas de sistema. Observa-se que a implementação *ngtcp2* consumiu 11,6 vezes mais chamadas de sistema que a versão no kernel, chegando até 13,3 vezes mais chamadas nos testes com 10% de perda. Em contrapartida, a implementação no kernel mantém um número mais estável, evidenciando um número significativamente menor de chamadas de sistema à medida que a taxa de perda aumenta. Além disso, em comparação com a biblioteca *msquic*, a implementação no kernel tem aproximadamente 35% menos chamadas de sistema em um ambiente sem perdas, chegando a 55% menos chamadas de sistema em condições com perdas de 10%.

Uma das razões iniciais para incluir número de chamadas de sistema no conjunto de teste foi avaliar a eficiência das diferentes implementações do QUIC em termos de sobrecarga do sistema operacional. Ao medir as chamadas de sistema, é possível identificar quão intensivamente cada implementação interage com o kernel do sistema operacional. É notável que a implementação do QUIC no kernel resulta em um número reduzido de chamadas de sistema em comparação às implementações msquic e ngtcp2, especialmente em cenários de maior perda de pacotes. Isto indica uma integração mais profunda e otimizada do protocolo QUIC com o sistema operacional, resultando em um menor *overhead* para gerenciamento de pacotes perdidos.

Tabela 4.2: Porcentagem adicional de chamadas de sistema realizadas nos cenário com taxas de perda de 1%, 5% e 10%, comparadas ao cenário com 0% de perda.

	1%	5%	10%
ngtcp2	-1.94%	+6.7%	+14.5%
msquic	-1.83%	-6.88%	-18.2%
kernel	-6.57%	-27.19%	-43.30%

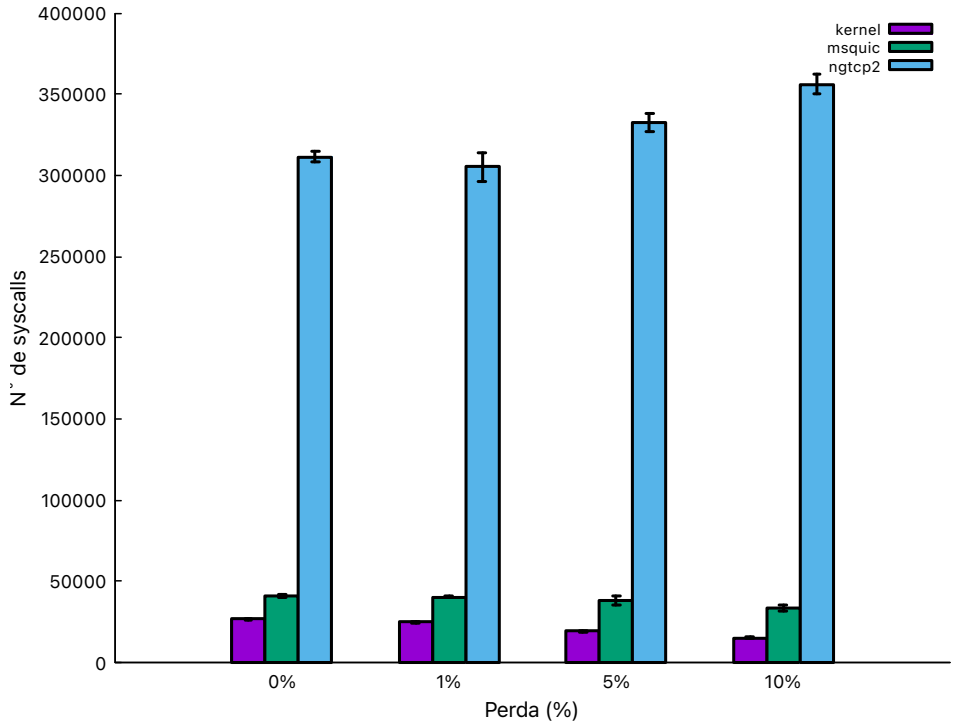


Figura 4.2: Número de chamadas de sistema feitas no decorrer de cada implementação para diferentes taxas de perda.

Capítulo 5

Trabalhos Futuros

Existem ainda possíveis otimizações para implementar na implementação do QUIC no kernel. Conforme descrito na Capítulo 3.3.3, o design atual do módulo de kernel está faltando algumas das funções desejadas como *sendfile* e *splice*. Com as chamadas de sistema *sendfile* e *splice*, é removida a constante troca de contexto, e o kernel lê diretamente os dados do arquivo e os envia para o socket de uma só vez. O trabalho futuro pode incluir essas chamadas de sistema e comparar os testes feitos com os resultados desse trabalho.

Além disso, a falta de descarregamento de pacotes pelo NIC na implementação do kernel representa uma vantagem não aproveitada em relação a uma aplicação executada no espaço do usuário. Como demonstrado no artigo *QUIC on the Highway: Evaluating Performance on High-rate Links* [12], essa abordagem reduz a sobrecarga associada às cópias de dados entre o espaço do usuário e o espaço do kernel, aumentando consequentemente a taxa de transferência em protocolos como o TCP. Se um trabalho futuro fosse feito na parte de descarregamento de pacotes, essas funcionalidades poderiam ser adicionadas.

Adicionalmente, investigações futuras podem se concentrar na incorporação de mecanismos de monitoramento e ajuste dinâmico dos parâmetros de desempenho do protocolo diretamente no kernel. Além de incorporar novas métricas de desempenho, como jitter, uso de CPU e uso de memória. Possibilitando a identificação em tempo real de gargalos e a adaptação automática das configurações de transmissão, contribuindo para uma maior resiliência e eficiência, mesmo sob condições instáveis ou voláteis da rede, como aquelas encontradas em redes móveis ou Wi-Fi. Em resumo, existe ainda um potencial de aprimoramento da implementação do QUIC no kernel. Investigações futuras poderão explorar novas abordagens e técnicas, podendo ampliar esse trabalho à medida que novas funcionalidades são também adicionadas ao protocolo QUIC.

Capítulo 6

Conclusão

Neste trabalho, foi realizada uma comparação entre três implementações do protocolo QUIC: duas que operam no espaço de usuário (*msquic* e *ngtcp2*) e uma que funciona diretamente no kernel do Linux. As implementações foram avaliadas em um ambiente experimental configurado com contêineres Docker [1], executados em uma máquina virtual. Esse cenário foi estruturado em uma rede que atua como ponte, conectando uma aplicação cliente a um servidor. A partir deste ambiente, foi examinado qual implementação se destacou em termos de taxa de transferência (*goodput*) e número de chamadas de sistema através das ferramentas *tcpdump* e *bpftrace*, além de investigar como a perda de pacotes afeta o desempenho dessas métricas. Por fim, os dados coletados foram analisados para determinar qual implementação apresenta o maior custo em termos de recursos e desempenho.

As avaliações envolvendo as três implementações permitiram descobrir que o protocolo QUIC no kernel do Linux tem uma taxa de transferência similar ou superior às implementações em espaço de usuário. Porém, quando existe uma alta perda de pacotes, a implementação no espaço de kernel se destaca com a maior taxa de transferência. Além disso, entre todas as implementações testadas, a implementação no kernel foi a que teve o menor número de chamadas de sistema, mantendo o número constante mesmo com grande perda de pacotes, chegando a ter 93% menos chamadas de sistema em comparação a outras soluções.

Durante o desenvolvimento do trabalho, surgiram inúmeras ideias que se aprimoraram ao longo do processo e geraram novas perspectivas. Inicialmente, muitos dos conceitos propostos não puderam ser implementados, como o caso do cálculo da latência, que não foi utilizado por conta da elevada variação dos dados. Além disso, diversas dificuldades foram encontradas, como os erros de compilação do módulo do kernel e problemas envolvendo incompatibilidade de versões de programa, os quais exigiram esforços significativos para serem solucionados.

Porém, mesmo diante dos desafios, o trabalho revelou-se extremamente enriquecedor e educativo, contribuindo para uma compreensão mais aprofundada do protocolo QUIC e de seu desenvolvimento no kernel. As percepções e descobertas obtidas acerca do desempenho do protocolo evidenciam o potencial da implementação no kernel e possíveis áreas de aprimoramento. Podendo inspirar futuras pesquisas e colaborações, promovendo o aperfeiçoamento contínuo do protocolo QUIC no kernel do Linux.

Referências

[1] Docker: *Containerization of applications with docker*, 2021. `https://github.com/docker`, acesso em 29-07-2024. v, vi, 15, 19, 28

[2] *The ns-3 network simulator*. `https://www.nsnam.org/`, acesso em 29-07-2024. v, vi, 7, 15

[3] Seemann, Marten e Jana Iyengar: *Automating quic interoperability testing*. Em *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, EPIQ '20, página 8–13, New York, NY, USA, 2020. Association for Computing Machinery, ISBN 9781450380478. `https://doi.org/10.1145/3405796.3405826`. v, vi, 7, 8, 14, 15, 18

[4] *Transmission Control Protocol*. RFC 793, setembro 1981. `https://tools.ietf.org/info/rfc793`. 1

[5] Honda, Michio, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley e Hideyuki Tokuda: *Is it still possible to extend tcp?* Em *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, página 181–194, New York, NY, USA, 2011. Association for Computing Machinery, ISBN 9781450310130. `https://doi.org/10.1145/2068816.2068834`. 1

[6] Iyengar, Jana e Martin Thomson: *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-draft draft-ietf-quic-transport-27, Internet Engineering Task Force. `https://datatracker.ietf.org/doc/draft-ietf-quic-transport/27/`, Work in Progress. 1

[7] Langley, Adam, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan Teh Chang e Zhongyi Shi: *The quic transport protocol: Design and internet-scale deployment*. Em *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, página 183–196, New York, NY, USA, 2017. Association for Computing Machinery, ISBN 9781450346535. `https://doi.org/10.1145/3098822.3098842`. 3, 4

[8] Wang, Peng, Carmine Bianco, Janne Riihijärvi e Marina Petrova: *Implementation and performance evaluation of the quic protocol in linux kernel*. Em *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of*

Wireless and Mobile Systems, MSWIM '18, página 227–234, New York, NY, USA, 2018. Association for Computing Machinery, ISBN 9781450359603. https://doi.org/10.1145/3242102.3242106. 4, 14

[9] Tran, Viet Hoang e Olivier Bonaventure: *Beyond socket options: making the linux tcp stack truly extensible*. Em *2019 IFIP Networking Conference (IFIP Networking)*, páginas 1–9, 2019. 5, 14

[10] Pantuza, Gustavo, Marcos A. M. Vieira e Luiz F. M. Vieira: *equic gateway: Maximizing quic throughput using a gateway service based on ebpf + xdp*. Em *2021 IEEE Symposium on Computers and Communications (ISCC)*, páginas 1–6, 2021. 6, 14

[11] Seemann, Marten: *Quic interop runner*, 2020. https://interop.seemann.io/, acesso em 29-07-2024. 7, 8

[12] Jaeger, Benedikt, Johannes Zirngibl, Marcel Kempf, Kevin Ploch e Georg Carle: *Quic on the highway: Evaluating performance on high-rate links*. Em *2023 IFIP Networking Conference (IFIP Networking)*, páginas 1–9, 2023. 8, 14, 27

[13] Tech, LiteSpeed: *lsquic*, 2024. https://github.com/litespeedtech/lsquic, acesso em 29-07-2024. 8

[14] Cloudflare: *quiche*, 2024. https://github.com/cloudflare/quiche, acesso em 29-07-2024. 8

[15] Shreedhar, Tanya, Rohit Panda, Sergey Podanev e Vaibhav Bajpai: *Evaluating quic performance over web, cloud storage, and video workloads*. IEEE Transactions on Network and Service Management, 19(2):1366–1381, 2022. 9, 10, 14

[16] Chaudhary, Sapna, Naval Kumar Shukla, Prince Sachdeva, Sandip Chakraborty e Mukulika Maity: *Managing connections by quic-tcp racing: A first look of streaming media performance over popular http/3 browsers*. IEEE Transactions on Network and Service Management, 21(3):2962–2976, 2024. 11, 14

[17] Siag, Lior, Gil Einziger, Wuji Liu e Chase Wu: *Characterization and prediction of quic's performance under different network conditions*. Em *ICC 2023 - IEEE International Conference on Communications*, páginas 3824–3829, 2023. 12, 14

[18] Peng, Haorui, William Tärneberg, Emma Fitzgerald e Maria Kihl: *Performance evaluation of quic vs. tcp for cloud control systems*. Em *2023 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, páginas 1–6, 2023. 13, 14

[19] Docker: *Docker compose [software]*, 2023. https://docs.docker.com/compose/, acesso em 29-07-2024. 15, 19

[20] *The tcpdup group*. http://www.tcpdump.org, acesso em 29-07-2024. 16, 20

[21] MICROSOFT: *msquic*, 2024. https://github.com/microsoft/msquic, acesso em 29-07-2024. 18, 19

[22] NGTCP2: *ngtcp2*, 2024. https://github.com/ngtcp2/ngtcp2, acesso em 29-07-2024. 18, 19

[23] *Quic network simulator*. https://github.com/quic-interop/quic-network-simulator, acesso em 29-07-2024. 18

[24] Docker: *DockerHub*, 2023. https://hub.docker.com/, acesso em 29-07-2024. 18

[25] STENBERG, Daniel: *curl - command line tool and library for transferring data with urls*, 2025. https://curl.se/, acesso em 21-02-2025. 19

[26] Xin Long: *Quic in linux kernel*. https://github.com/lxin/quic, acesso em 28-01-2025. 19, 20

[27] *bpftrace*. https://github.com/iovisor/bpftrace, acesso em 29-07-2024. 21

[28] Wireshark Foundation: *Tshark*. https://www.wireshark.org/docs/man-pages/tshark.html, acesso em 29-07-2024. 21

[29] Xavier, Bruno: *Comparison between kernel and userspace quic implementations*. https://github.com/BrunoBarrosXavier/QUIC-kernel-vs-userspace, acesso em 21-02-2025. 22

[30] Oracle Corporation: *Oracle VM VirtualBox*. https://www.virtualbox.org/, acesso em 29-07-2024. 23