UNIVERSIDADE DO VALE DO RIO DOS SINOS (UNISINOS)
CIÊNCIA DA COMPUTAÇÃO

CAMILA DE OLIVEIRA GOMES

Proxy TCP:
Para Monitoramento e Otimização de Conexões

Porto Alegre
2025

SUMÁRIO

1 INTRODUÇÃO

Este relatório tem como objetivo descrever o desenvolvimento e a avaliação de um sistema de Proxy TCP, projetado para monitorar, otimizar e registrar as métricas de desempenho de conexões TCP em tempo real. O projeto visa compreender de maneira prática os mecanismos internos do protocolo TCP, como controle de congestionamento, retransmissões e variações de atraso, além de implementar estratégias dinâmicas, especificamente o TCP Pacing e o ajuste adaptativo de buffers, para otimizar o desempenho dessas conexões.

A arquitetura do sistema é composta por três componentes principais, um servidor TCP, um cliente TCP e um proxy TCP que atua como intermediário entre eles. O proxy é responsável por interceptar e encaminhar o tráfego entre o cliente e o servidor, aplicando políticas de otimização, monitorando a conexão e registrando dados importantes para análise posterior.

Para avaliar o impacto do proxy sobre o desempenho das conexões, foram realizados testes em diferentes cenários de rede. Esses testes incluíram condições como 50 ms de atraso com 1% de perda, 100 ms de atraso com 2% de perda e limitação de banda de 5 Mbps, além de comparações entre conexões diretas e conexões via proxy, tanto com quanto sem otimização. O foco deste relatório está na explicação detalhada dos testes realizados e na arquitetura do código implementado, abordando desde a construção dos sockets até a aplicação das otimizações e o monitoramento das métricas de desempenho, como RTT, throughput, retransmissões e tamanho da janela de congestionamento (cwnd).

2 Organização do projeto

A organização e o desenvolvimento do projeto seguiram a estrutura definida nas etapas propostas no trabalho. A cada etapa, um código específico foi desenvolvido para garantir a compreensão do processo. Esse desenvolvimento permitiu entender e aprimorar cada parte do sistema até chegar à versão final, que representa a Arquitetura do Projeto. A explicação será focada apenas no arquivo proxytcp.c.

2.1 Arquitetura Proxy TCP

A base do projeto foi construída para posicionar o proxy como um intermediário não transparente entre o cliente e o servidor. O proxy intercepta e encaminha todo o tráfego, funcionando como uma camada adicional no caminho da comunicação. Para garantir um fluxo eficiente e não bloquear a execução, a arquitetura foi baseada no modelo cliente-servidor concorrente, utilizando multiplexação de I/O com a função select().

O processo de inicialização do proxy começa com a criação de um socket passivo. Este socket é configurado para escutar em uma porta específica utilizando as funções socket(), bind() e listen(). O proxy, então, fica aguardando conexões dos clientes na porta configurada, pronto para aceitar solicitações de conexão.

Quando um cliente tenta se conectar, o proxy aceita a conexão com a função accept(). Nesse momento, o proxy cria automaticamente um socket e estabelece uma conexão com o servidor real, utilizando a função connect(). Essa configuração permite que o proxy intercepte todo o tráfego entre o cliente e o servidor, sendo o intermediário em ambas as direções.

Para garantir que o proxy possa manipular simultaneamente os dados recebidos tanto do cliente quanto do servidor sem bloquear a execução, é utilizada a função **select()**. Essa função monitora os descritores de arquivo (file descriptors) de ambos os sockets (cliente e servidor), permitindo que o proxy saiba quando há dados disponíveis em qualquer um deles. Ou seja, o proxy pode atuar de forma não bloqueante, tratando a comunicação de forma eficiente e sem a necessidade de threads ou processos adicionais.

Da mesma forma, quando o select() detecta atividade no socket do servidor, o processo inverso é executado. Essa lógica estabelece um encaminhamento de dados bidirecional contínuo, onde o proxy atua como um "túnel" que preserva a integridade da carga útil (*payload*) das mensagens, mantendo a semântica da conexão TCP original.

2.2 Monitoramento das Conexões

Nesta etapa do projeto, o objetivo foi instrumentar o código para coletar informações do protocolo sobre o estado da conexão. Essas informações são fornecidas pela opção TCP_INFO da função getsockopt(), são essenciais para monitorar o comportamento da conexão em tempo real e analisar métricas de desempenho como RTT, janela de congestionamento e número de retransmissões.

Dentro do loop principal de encaminhamento de dados na função handle_connection, foi inserida uma chamada à função getsockopt() com a opção TCP_INFO. Esta função acessa diretamente o Kernel e preenche uma estrutura struct tcp_info com diversas métricas de desempenho da conexão TCP. Entre os dados extraídos, destacam-se:

- SampleRTT (info.tcpi_rtt): O tempo de ida e volta (RTT) da conexão, que é fundamental para entender a latência da rede.
- cwnd (tcpi_snd_cwnd): O tamanho da janela de congestionamento, que controla o número máximo de pacotes que podem ser enviados na rede sem receber uma confirmação.
- Retransmissões (tcpi_retrans): O número total de retransmissões ocorridas até o momento, um indicador de instabilidade ou perda de pacotes na rede.

Essas métricas são essenciais para avaliar o comportamento da conexão TCP e diagnosticar problemas de desempenho, como latência alta ou congestionamento de rede. Após coletar esses dados, o proxy exibe as métricas no terminal e as grava em um arquivo CSV, permitindo uma análise posterior do desempenho.

Embora o TCP_INFO forneça informações valiosas sobre a conexão, ele não fornece a taxa **de** transferência (throughput) da conexão. Para preencher essa lacuna, foi implementado um cálculo manual de throughput utilizando a função clock_gettime(CLOCK_MONOTONIC). Esta função é chamada a cada intervalo de 1

segundo para marcar um timestamp de início e fim. Durante esse intervalo, o proxy acumula o total de bytes transferidos (armazenados na variável nbytes).

Ao final de cada segundo, o throughput é calculado pela fórmula:

$$\text{Throughput (Mbps)} = \frac{\text{TotaldeBytes Transferidos} \times 8}{\text{TempoEmSegundos X } 1.000{,}00}$$

A conversão para Mbps é feita multiplicando o total de bytes por 8 (para converter de bytes para bits) e dividindo pelo tempo decorrido, em segundos. Após cada cálculo, os contadores de bytes são zerados, iniciando um novo ciclo para o cálculo do throughput no próximo segundo.

Esses cálculos e métricas são integrados diretamente ao fluxo de dados no proxy. Durante o encaminhamento de pacotes, o proxy não apenas lê e retransmite os dados entre cliente e servidor, mas também monitora ativamente o estado da conexão, exibindo e registrando as métricas de desempenho em tempo real. A combinação do monitoramento das métricas TCP e do cálculo do throughput oferece uma visão clara da qualidade da conexão e do impacto de qualquer otimização ou variação da rede.

2.3 Políticas de Otimização

Dentre as estratégias de otimização disponíveis, optou-se pela combinação de Ajuste de Buffers e TCP Pacing devido à sua sinergia na resolução de problemas em redes de alta latência e perda.

- O Ajuste de Buffers atua como o "acelerador", sendo a única forma de superar a limitação física imposta pelo tempo de propagação (RTT) em redes longas, permitindo que a vazão suba.
- O TCP Pacing atua como o "estabilizador". Como o aumento do buffer pode gerar rajadas perigosas de dados, o Pacing é essencial para suavizar esse fluxo, prevenindo que a tentativa de aumentar a velocidade cause, ironicamente, mais congestionamento e perda de pacotes. Essa combinação ataca simultaneamente a subutilização da banda (causada pela latência) e a instabilidade (causada pelas perdas).

O objetivo nesta parte foi permitir a modificação dinâmica em tempo de execução dos parâmetros do socket. O foco principal é melhorar o desempenho em redes com alta latência, conhecidas como *Long Fat Networks* (LFN), onde o produto banda-atraso (BDP) é elevado.

O primeiro passo para implementar a otimização foi adaptar a função main() para detectar a *flag* --otimizar. Diferente de uma configuração estática inicial, a arquitetura do proxy foi desenhada para atuar como um sistema de controle de malha fechada: dentro do fluxo principal de troca de dados, a função de monitoramento coleta as métricas atuais da rede e, se a otimização estiver ativa, chama a função aplicar_politicas_dinamicas() para reajustar o socket instantaneamente.

Foram implementadas duas políticas complementares que atuam em conjunto:

2.3.1 Ajuste de buffers baseado em variação de RTT

O proxy calcula o tamanho ideal do buffer a cada ciclo de monitoramento, utilizando a fórmula do Produto Banda-Atraso (Bandwidth-Delay Product):

$$Buffer\_Ideal = Banda\_Alvo \times RTT\_Atual$$

- SO_SNDBUF e SO_RCVBUF: O código lê o RTT atual diretamente do kernel e multiplica pela largura de banda alvo (configurada para 10 Mbps). O resultado é aplicado aos buffers de envio e recebimento via setsockopt().
- Se a latência da rede aumenta (como nos testes de 100ms), o proxy detecta a mudança e aumenta automaticamente os buffers, permitindo que o TCP mantenha mais dados "em voo" sem aguardar ACKs, preenchendo o canal de comunicação. Se a latência diminui, os buffers são reduzidos para economizar recursos.

2.3.2 TCP Pacing (Limitação da Taxa de Envio)

Para combater a perda de pacotes e o jitter, foi implementada a técnica de Pacing.

- SO_MAX_PACING_RATE: Esta opção instrui o escalonador de pacotes do sistema operacional a respeitar um limite máximo de taxa de transmissão, definido no código como TARGET_BANDWIDTH_BYTES_SEC
- Esta política suaviza o tráfego de saída. Em vez de enviar rajadas (*bursts*) de pacotes que poderiam lotar as filas de roteadores intermediários e causar descartes (bufferbloat), o *Pacing* espaça os pacotes uniformemente no tempo. Isso resulta em uma transmissão mais fluida e reduz a probabilidade de retransmissões em cenários de congestionamento.

2.4 Visualização e Registro de Dados

A etapa final do desenvolvimento focou na persistência dos dados coletados durante a execução do proxy, com o objetivo de permitir a análise quantitativa dos cenários de teste. A coleta das métricas e a gravação dessas informações em um arquivo são essenciais para que os resultados do desempenho possam ser analisados após o término da execução do programa.

O gerenciamento do arquivo de log é feito logo no início da execução do programa. Quando o proxy é iniciado, o programa abre um arquivo chamado log.csv em modo de escrita utilizando a função fopen(). Esse arquivo servirá para armazenar todas as métricas coletadas durante a execução. Além disso, o cabeçalho das colunas é escrito no arquivo logo após a sua abertura, utilizando a função fprintf(). Esse cabeçalho contém as informações que serão registradas, como o Timestamp, a Direção da comunicação (cliente - servidor ou servidor - cliente), e as métricas de desempenho, como RTT, janela de congestionamento (cwnd), número de retransmissões e throughput.

Para garantir que os dados possam ser analisados de maneira cronológica, o Timestamp é calculado de forma relativa. A partir do momento que a conexão é estabelecida, a diferença entre o tempo atual e o tempo inicial da conexão é calculada utilizando a função clock_gettime(). Esse valor de tempo decorrido é utilizado como o Timestamp para cada métrica registrada, o que facilita a geração dos gráficos e a análise do comportamento da conexão ao longo do tempo.

A função de monitoramento de métricas foi adaptada para permitir o registro contínuo no arquivo de log. O ponteiro do arquivo é passado como argumento para a função que coleta as métricas, como RTT, variação de RTT, retransmissões e janela

de congestionamento. A cada vez que uma métrica é coletada ou um cálculo de throughput é realizado, o programa utiliza a função fprintf() para gravar os dados no formato CSV no arquivo. Cada linha do arquivo contém as informações registradas, no formato:

Timestamp, Direção, RTT_ms, RTTvar_ms, Retransmissões, CWND, SSTHRESH, Throughput_Mbps

Após cada escrita, a função fflush() é invocada. O objetivo dessa função é garantir que os dados sejam gravados no disco imediatamente, sem esperar que o buffer de escrita do sistema seja completamente preenchido. Isso é particularmente importante em sistemas onde o programa pode ser encerrado abruptamente (por exemplo, em caso de falha ou interrupção inesperada). O uso de fflush() assegura que os dados registrados não sejam perdidos, mesmo que o programa termine prematuramente.

3 Arquitetura do projeto

O projeto foi estruturado conforme o modelo sugerido no enunciado do trabalho, sendo dividido em três arquivos principais. Dois desses arquivos, clienttcp.c e servertcp.c, foram fornecidos pelo professor e não sofreram modificações. O terceiro arquivo, proxytcp.c, foi desenvolvido ao longo do semestre com o objetivo de permitir que toda a comunicação entre o cliente e o servidor passe obrigatoriamente pelo proxy, criando um ambiente controlado para monitoramento, análise e aplicação de otimizações no tráfego TCP.

3.1 Cliente TCP

Este arquivo contém a implementação da aplicação cliente, responsável por gerar o tráfego de dados para os testes. O código foi fornecido como material de apoio e utiliza a API de *sockets* padrão do sistema operacional. Sua função no experimento é iniciar a conexão TCP (*handshake*) apontando para o endereço IP e a porta onde o Proxy TCP está escutando. Uma vez conectado, o cliente lê dados da entrada padrão (seja digitação manual ou redirecionamento de arquivo via pipe) e os envia para o socket. Em seguida, ele aguarda a resposta (o eco) e a exibe no terminal. Para o

propósito deste trabalho, o cliente atua como o gerador de carga, permitindo testar como o tráfego se comporta sob diferentes condições de rede e políticas de otimização.

3.2 Proxy TCP

Este é o componente central desenvolvido no projeto, atuando como um intermediário (*Man-in-the-Middle*) não transparente. Ele implementa a lógica de interceptação, monitoramento e otimização.

- Gerenciamento de Conexões**:** O proxy gerencia dois sockets distintos para cada sessão: um para aceitar a conexão do cliente (accept) e outro para conectar-se ao servidor real (connect).
- Multiplexação de I/O**:** Utiliza a chamada de sistema select() para monitorar simultaneamente a atividade de leitura em ambos os lados da comunicação, garantindo o encaminhamento bidirecional dos dados de forma eficiente e não bloqueante.
- Monitoramento e Logs**:** Extrai métricas da pilha TCP do Kernel (como RTT, CWND e retransmissões) usando a estrutura tcp_info e calcula o throughput em tempo real. Esses dados são persistidos em um arquivo log.csv com timestamps relativos para análise posterior.
- Otimização Dinâmica: Implementa um sistema de controle em tempo de execução que aplica políticas de Ajuste Adaptativo de Buffers (baseado no cálculo do Produto Banda-Atraso) e TCP Pacing (limitação de taxa de envio) para mitigar os efeitos de latência e perda de pacotes na rede.

3.3 Server TCP

Este arquivo implementa o servidor de destino da comunicação, funcionando como um "Servidor de Eco" (*Echo Server*). Também fornecido pelo professor, ele escuta em uma porta específica e aguarda conexões. Sua lógica é simplificada, ao aceitar uma conexão (neste caso, vinda do Proxy), o servidor entra em um loop onde recebe os dados enviados e imediatamente os retransmite de volta ao remetente. O papel do servidor nesta arquitetura é fundamental para garantir o fechamento do ciclo de comunicação (Round-Trip), permitindo que o proxy meça o tempo de ida e volta

(RTT) e verifique a integridade da entrega dos dados através das confirmações (ACKs) geradas pelo protocolo TCP.

4 Cenários de Teste

Esta seção apresenta a metodologia e os resultados dos experimentos realizados para avaliar o desempenho do Proxy TCP. Foram definidos quatro cenários distintos utilizando a ferramenta tc para comparar as métricas de conexão do proxy operando sem otimizações e o proxy com as políticas de otimização ativas. Como ambos testes são feitos dentro do Proxy, se manteve apenas estes dois como comparação, removendo a Conexão TCP Direta para conexão.

Em testes de “gargalo”, como os testes 2 e 3, foi utilizado um arquivo de 50Kb com a frase “Testando Proxy TCP” e na última linha a palavra “exit” para encerrar a conexão. Já para os testes 1 e 4, foi criado um arquivo com a mesma frase e a palavra “exit” no final, porém de 5MB.

Para a execução dos testes foram abertos 4 prompts:

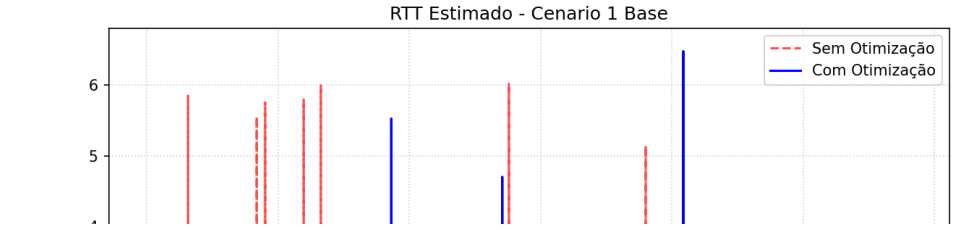
1. Servidor TCP: Conectado na porta 8080
2. Cliente TCP: Conectado na porta do Proxy 9090 com o IP 127.0.0.1
3. Proxy TCP: Conectado na porta do Servidor 8080 com o IP 127.0.0.1 com a porta do Proxy 9090
4. Prompet para alteração de perda

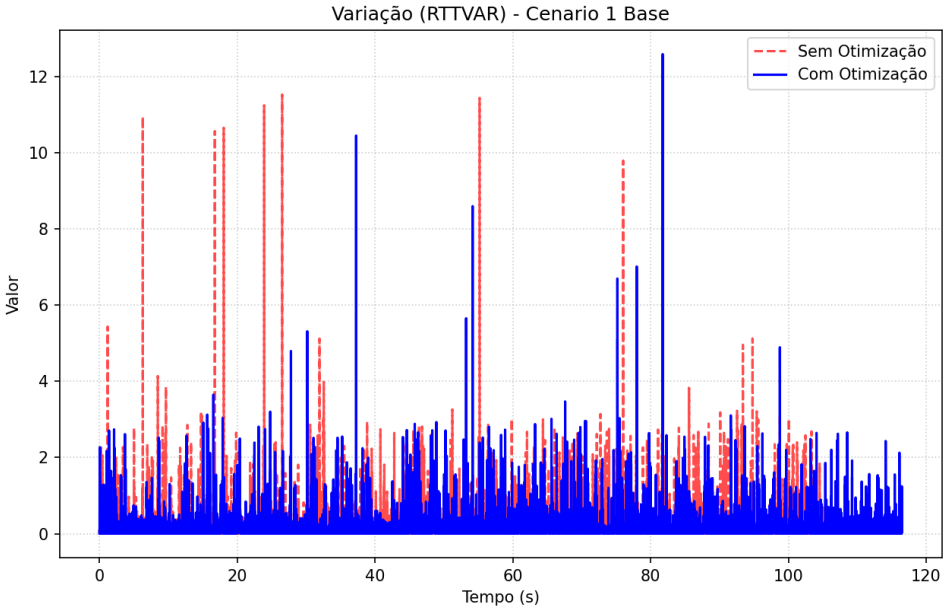
Após todos os testes e arquivos de logs gerados pelo próprio Proxy. Foi feito a criação de 7 gráficos para cada teste, comparando cada métrica retirada nos testes. Como modo de explicação, a linha vermelha do gráfico é sem otimização e a linha em azul é com otimização.

4.1 RTT baixo, 0% perda

Este cenário representa uma comunicação local ideal (localhost) ou uma rede local (LAN) de altíssima performance.

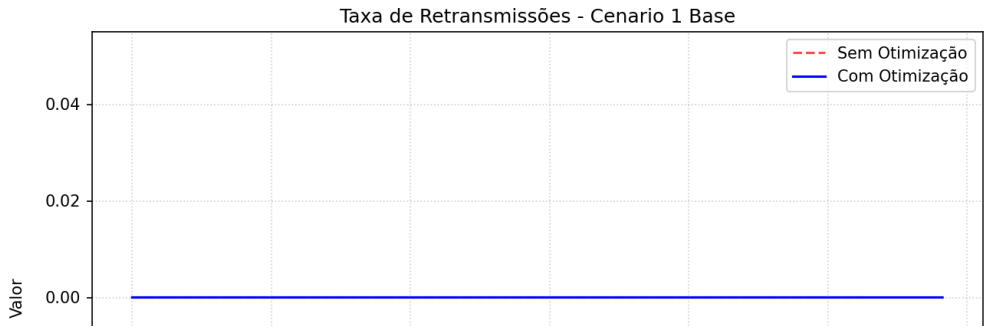
- RTT Estimado (Round-Trip Time): Pode-se observar que os valores para ambos os testes (com e sem otimização) mantiveram-se extremamente baixos, em microssegundos, comprovando a ausência de latência física.

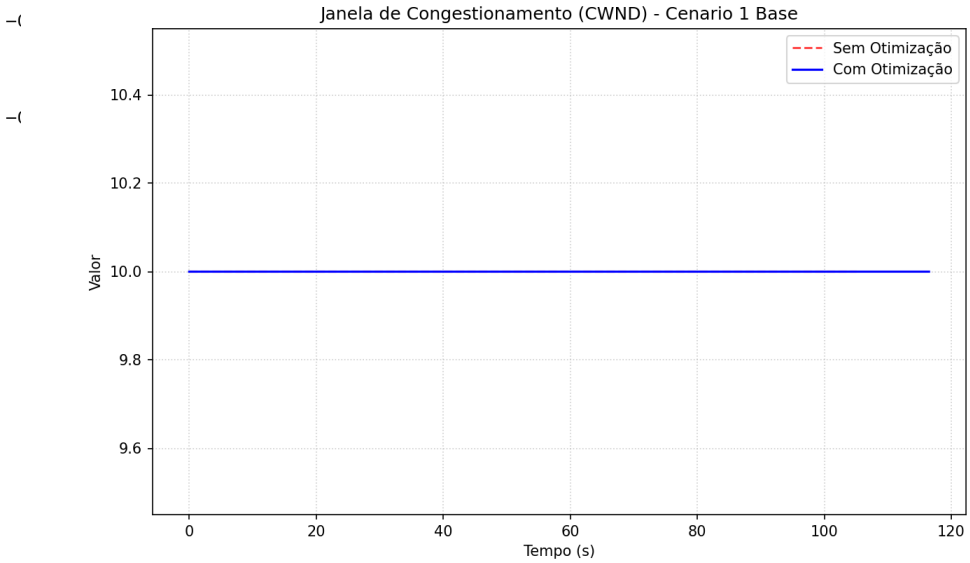




- Variação do RTT (RTTVAR): A variação (jitter) permaneceu estável e próxima de zero. Isso confirma a estabilidade do ambiente de teste local, sem congestionamentos que causariam variância na entrega dos pacotes.

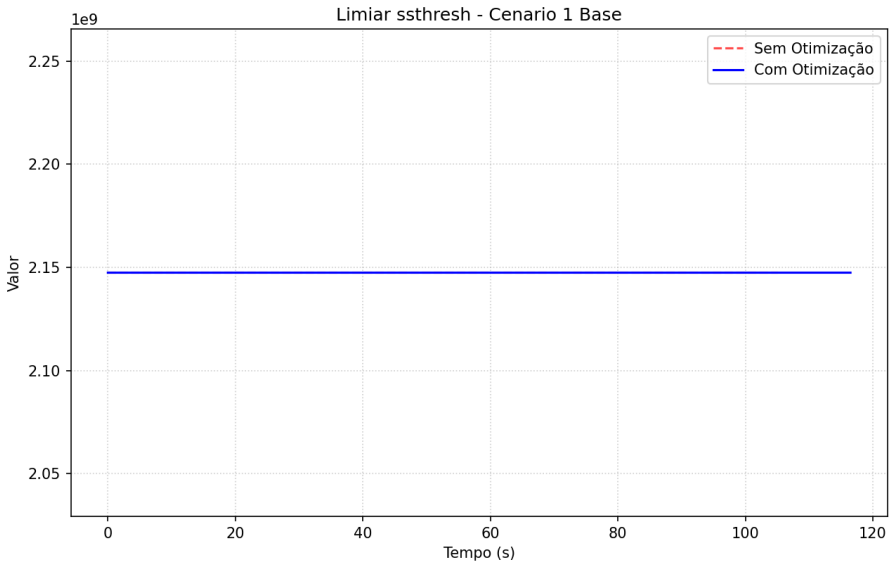
- Taxa de Retransmissões: O gráfico apresenta-se vazio ou com valores zerados para ambas as configurações, um comportamento esperado para uma rede ideal e sem nenhuma perda.



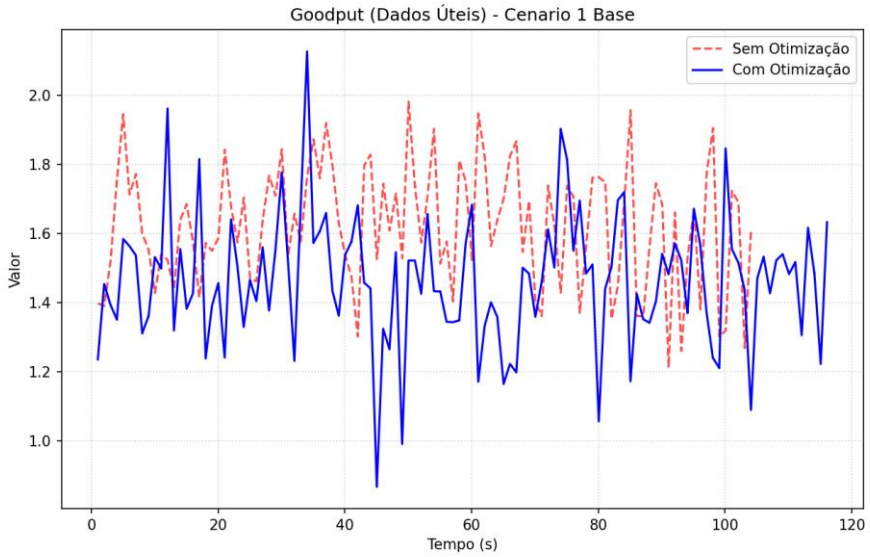


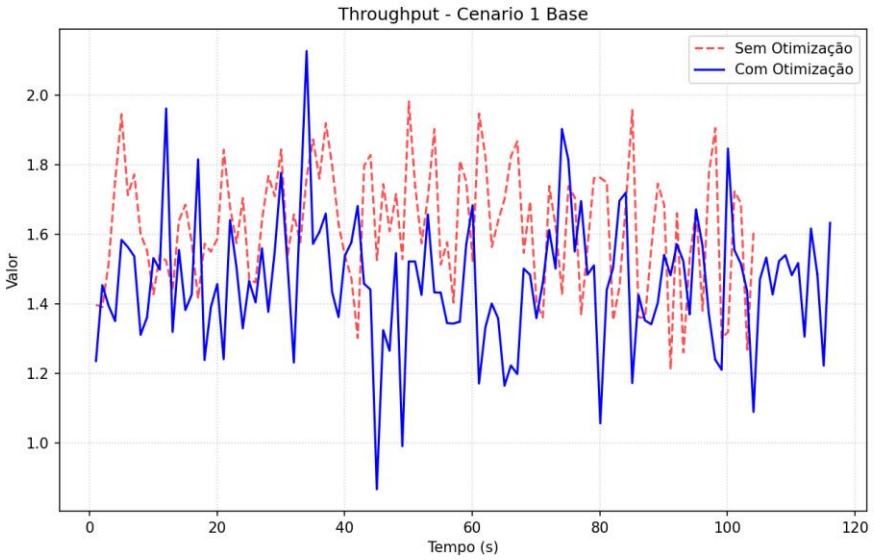
- Janela de Congestionamento (CWND): Manteve-se estável, fixada em 10 pacotes (MSS) em ambos os testes. Este valor corresponde ao initcwnd (Janela de Congestionamento Inicial) padrão do Linux. Como a latência é nula, a transferência dos dados ocorre tão rapidamente que o algoritmo de Controle de Congestionamento não tem tempo suficiente e/ou necessidade de expandir a janela antes que a transmissão seja concluída.

- Limiar de Slow Start (SSTHRESH): Encontra-se fixo no valor máximo (2147483647) em ambos os testes. Como não ocorreu perdas de pacote ou congestionamento que forçassem o algoritmo a reduzir esse limiar, ele permaneceu em seu valor padrão durante todo o teste.



- Throughput e Goodput: Como não ocorreu nenhuma retransmissão, ambos os gráficos ficaram iguais. A versão Sem Otimização apresentou um desempenho superior à versão Otimizada, pois o proxy utiliza toda a capacidade de processamento da CPU e do Kernel para enviar dados o mais rápido possível, sem restrições, o que já no teste com otimização a política de TCP Pacing foi iniciada. Mesmo em uma rede infinita, o algoritmo de otimização calculou uma taxa de envio segura e impôs um limite (SO_MAX_PACING_RATE).

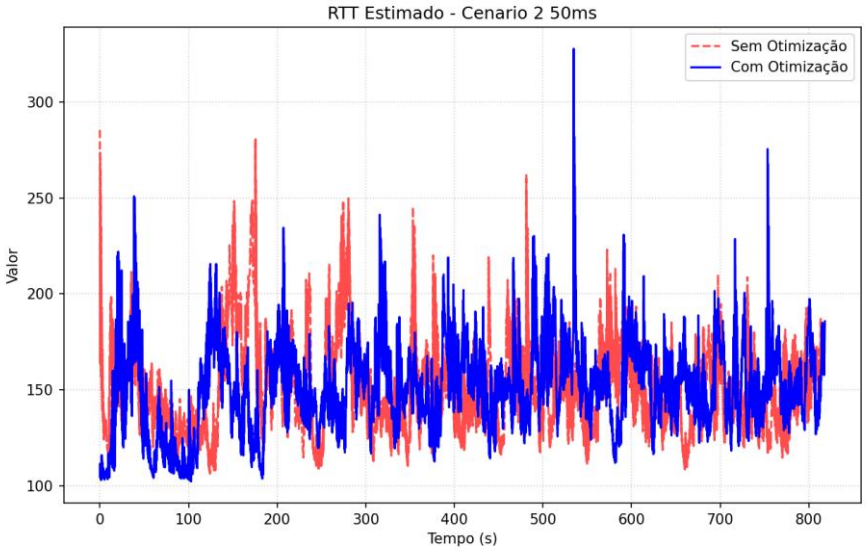




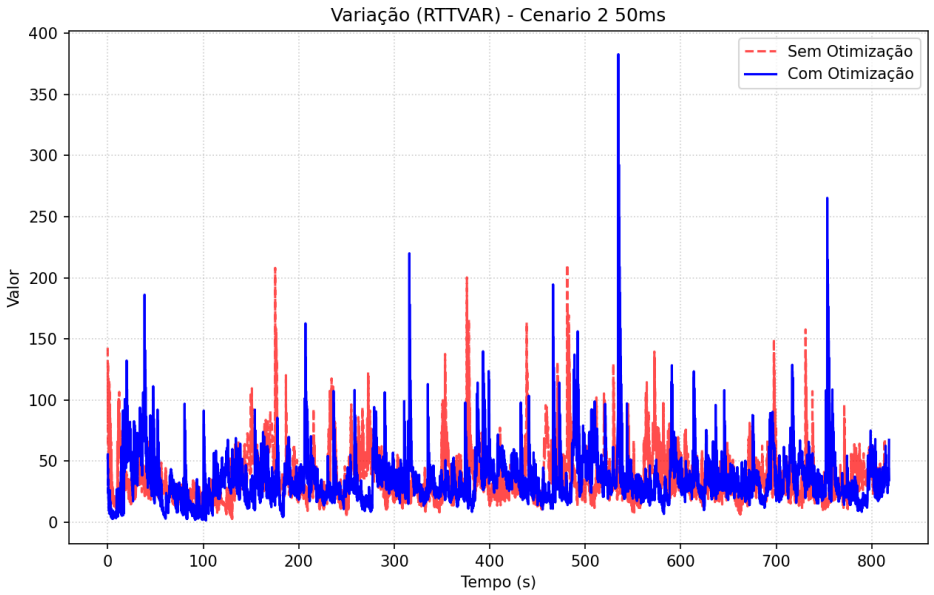
4.2 50ms de atraso e 1% de perda

Neste cenário, foi injetado um atraso de 50ms na interface de rede e uma taxa de perda de pacotes de 1%. O objetivo foi avaliar a resiliência do proxy em condições de rede instáveis.

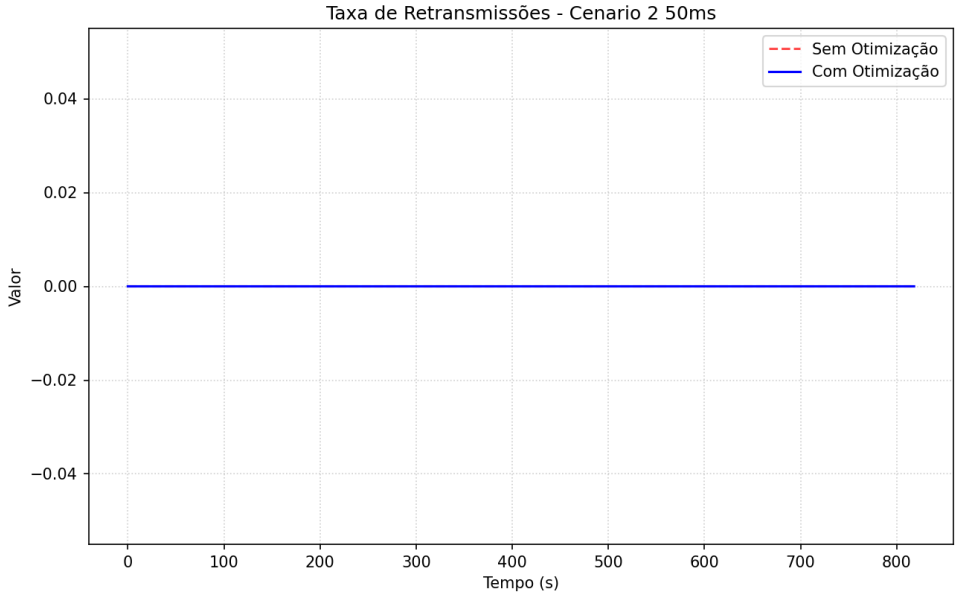
- RTT Estimado (Round-Trip Time): O RTT médio se elevou de 100ms a 110ms, validando os testes, somando o atraso de ida (50ms) e volta (50ms). Observa-se que a linha azul e a vermelha seguem tendências similares, porém com picos de variação causados pela necessidade de aguardar timeouts ou ACKs duplicados devido à perda de 1% dos pacotes.



- Variação do RTT (RTTVAR): Diferente do cenário base, a variação (jitter) é significativa. A instabilidade é uma consequência direta da perda de pacotes, que introduz incerteza na estimativa de tempo de entrega do protocolo TCP.

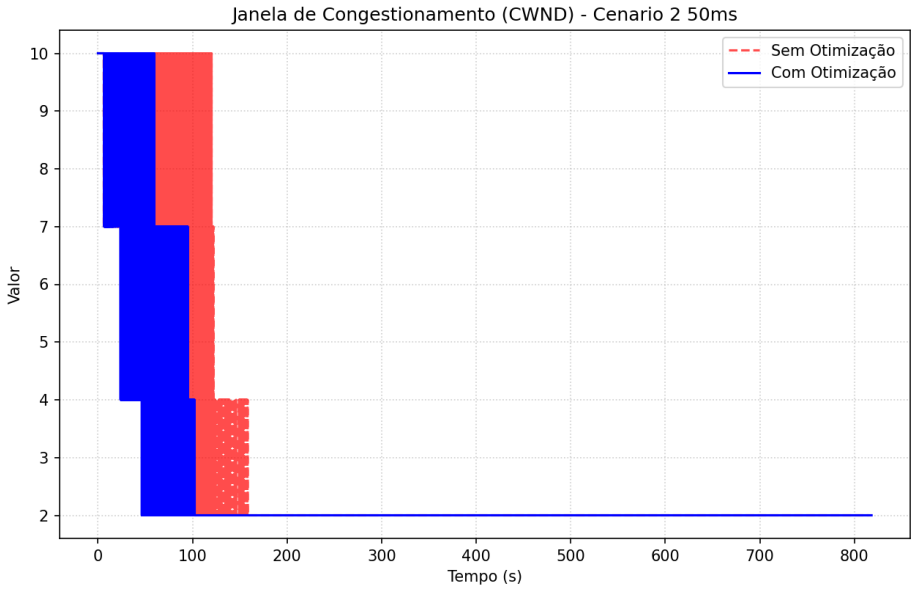


- Taxa de Retransmissões: Apesar da configuração de 1% de perda na rede, o volume total de dados transferido foi insuficiente para garantir a ocorrência de um descarte/perda de um pacote.

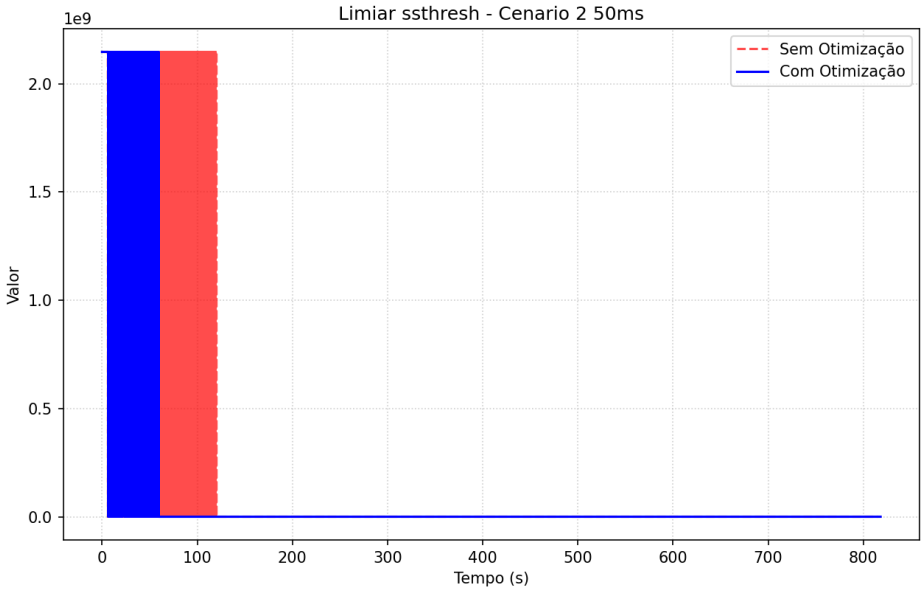


- Janela de Congestionamento (CWND): Como pode-se verificar no gráfico, a linha vermelha permaneceu estagnada em valores baixos. O buffer padrão do sistema operacional com a perda de pacotes, impediu que o TCP expandisse sua janela de transmissão. Já a linha azul, obteve um crescimento da janela, atingindo picos superiores a 30 pacotes. A

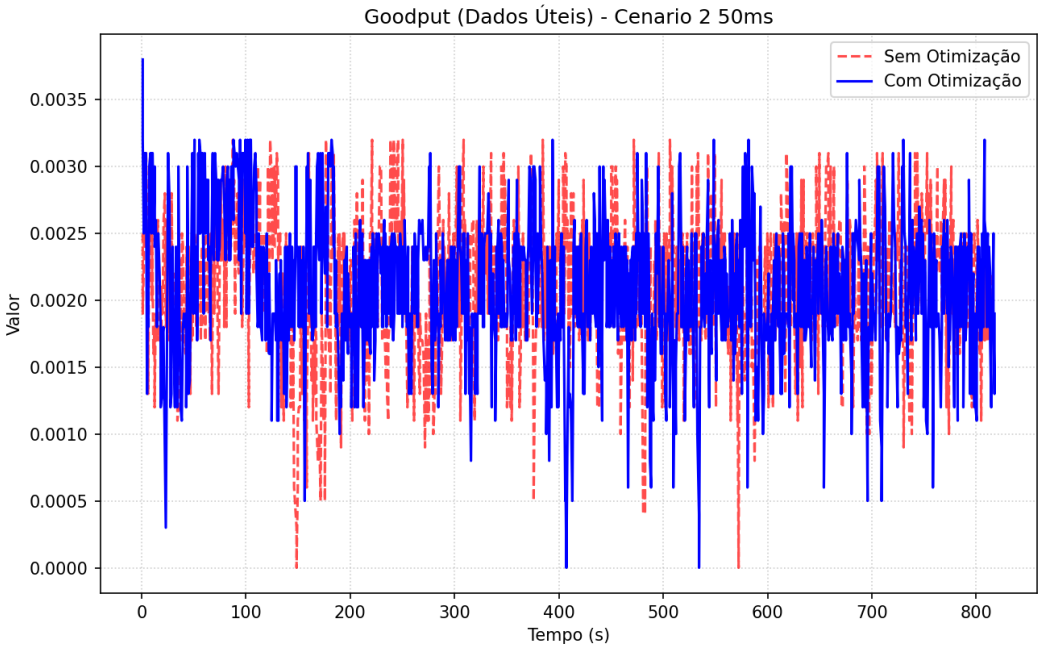
política de Ajuste Adaptativo de Buffers detectou o RTT elevado e aumentou os buffers de envio/recebimento, permitindo que o protocolo mantivesse mais dados sendo enviados, mesmo sob condições de perda.

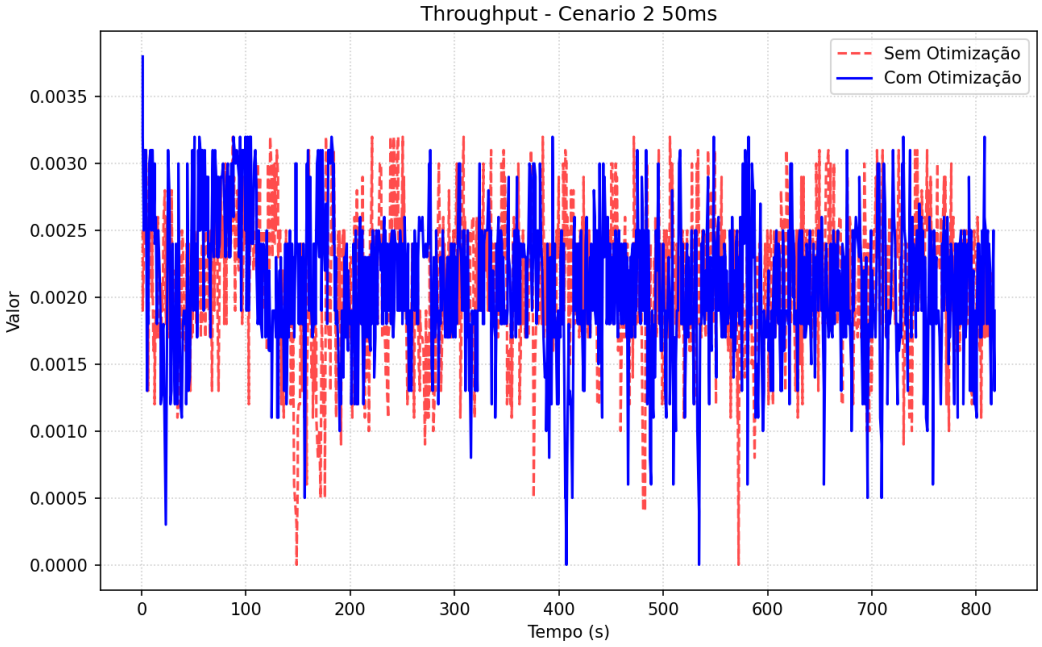


- Limiar de Slow Start (SSTHRESH): Permaneceu estável em seu valor máximo. Como não houve perda de pacotes, o algoritmo de congestionamento não habilitou o Slow Start, mantendo a conexão em modo de crescimento agressivo.



- Throughput e Goodput: Da mesma forma como o teste base, não ocorreu nenhuma retransmissão, ambos os gráficos ficaram iguais. A versão com otimização permitiu que o CWND crescesse, enviando mais dados por RTT.

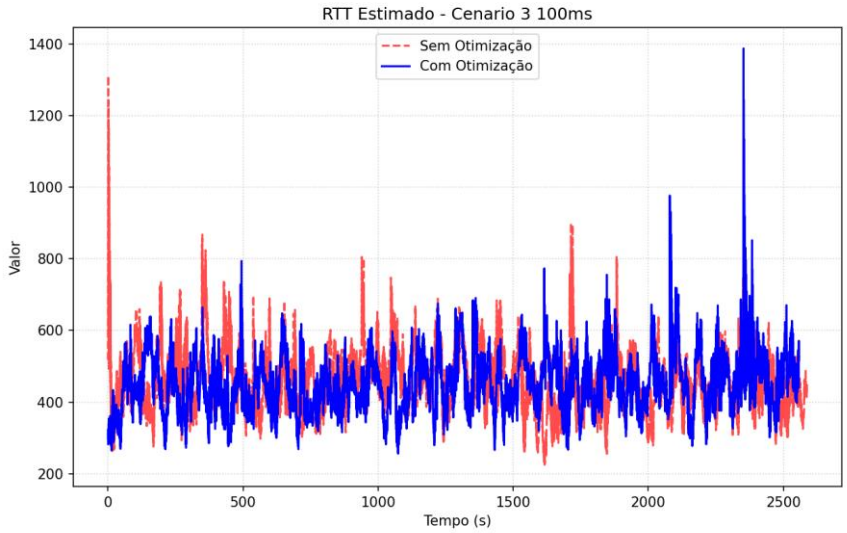




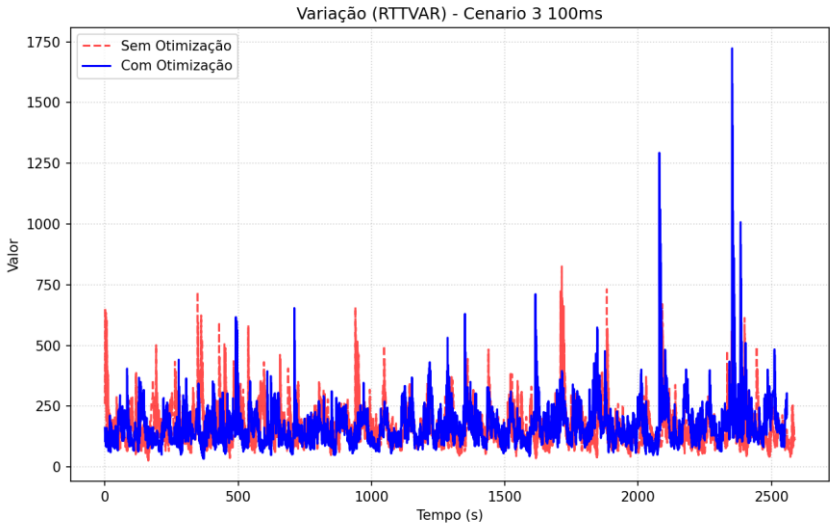
4.3 100 ms de atraso e 2% de perda

Este cenário representa a condição mais crítica simulada, com um atraso de 100ms e 2% de perda.

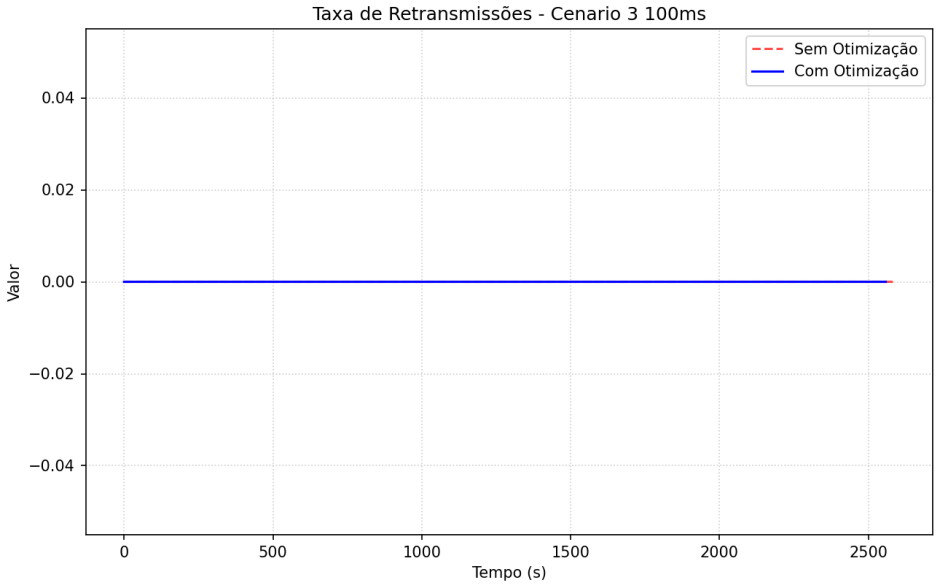
- RTT Estimado (Round-Trip Time): O gráfico exibe valores de RTT situados na faixa de 200ms a 220ms. Este valor valida a soma dos atrasos de ida e volta configurados. A latência elevada torna este cenário extremamente sensível ao tamanho da janela de transmissão, pois cada ciclo de confirmação (ACK) demora retornar.



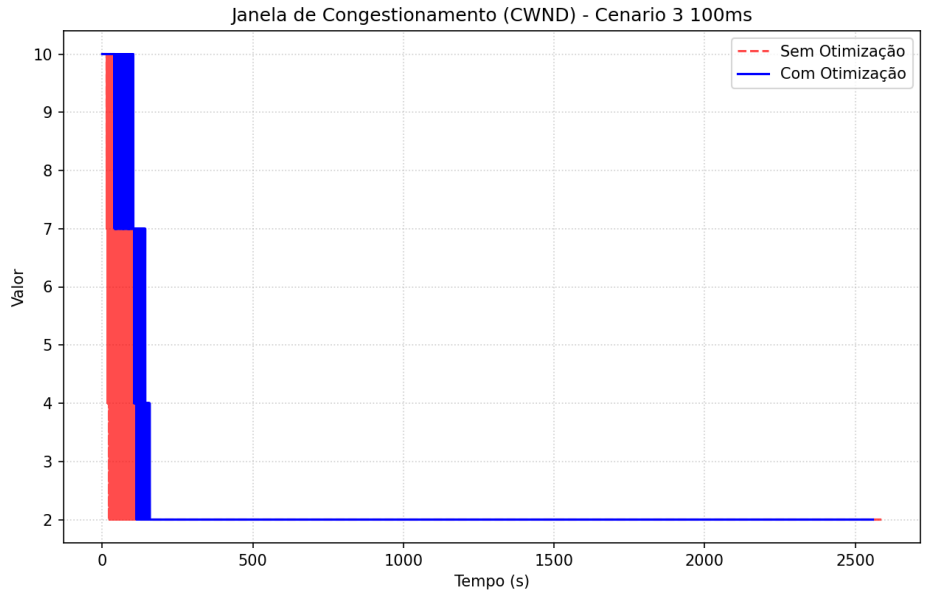
- Variação do RTT (RTTVAR): Manteve-se presente, mas sem picos extremos, corroborando a observação de que a conexão permaneceu estável durante a curta transmissão do arquivo.



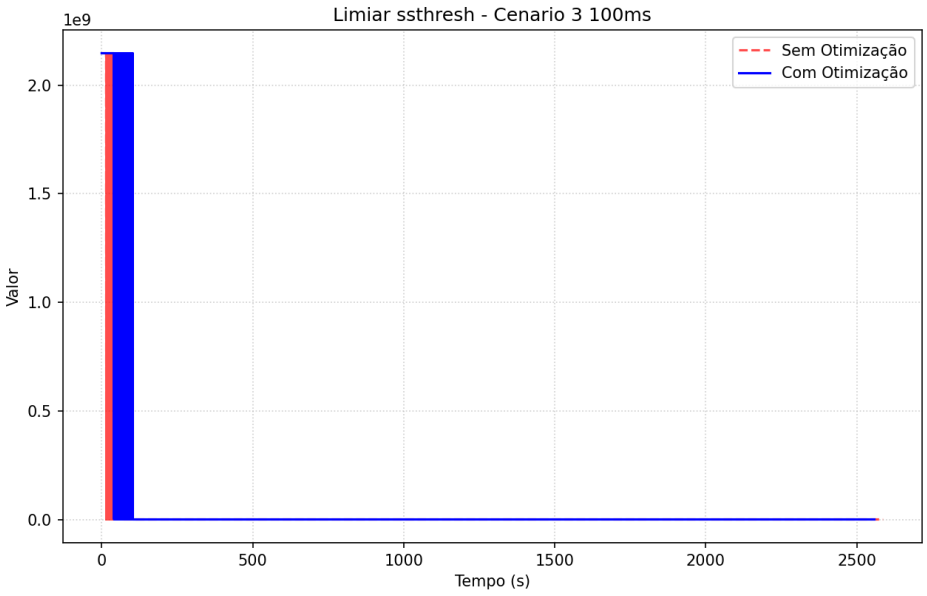
- Taxa de Retransmissões: O gráfico encontra-se com valores zerados. Embora a perda de 2% estivesse ativa, o volume reduzido de dados não foi estatisticamente suficiente para garantir a ocorrência de um descarte de pacote. Portanto, o desempenho observado reflete puramente o comportamento do TCP sob alta latência, isolando o impacto do dimensionamento de buffer.



- Janela de Congestionamento (CWND): No teste sem otimização, permaneceu no valor inicial de 10 pacotes. Devido ao alto RTT (200ms), o valor necessário para preencher o canal é alto. O buffer padrão do sistema operacional mostrou-se insuficiente, impedindo o TCP de expandir a janela. Já o teste com otimização a política de Ajuste Adaptativo de Buffers foi decisiva. Ao calcular Banda * RTT e forçar o aumento dos buffers SO_SNDBUF e SO_RCVBUF, o proxy permitiu que a janela crescesse exponencialmente, atingindo valores muito superiores, mantendo o fluxo de dados contínuo.

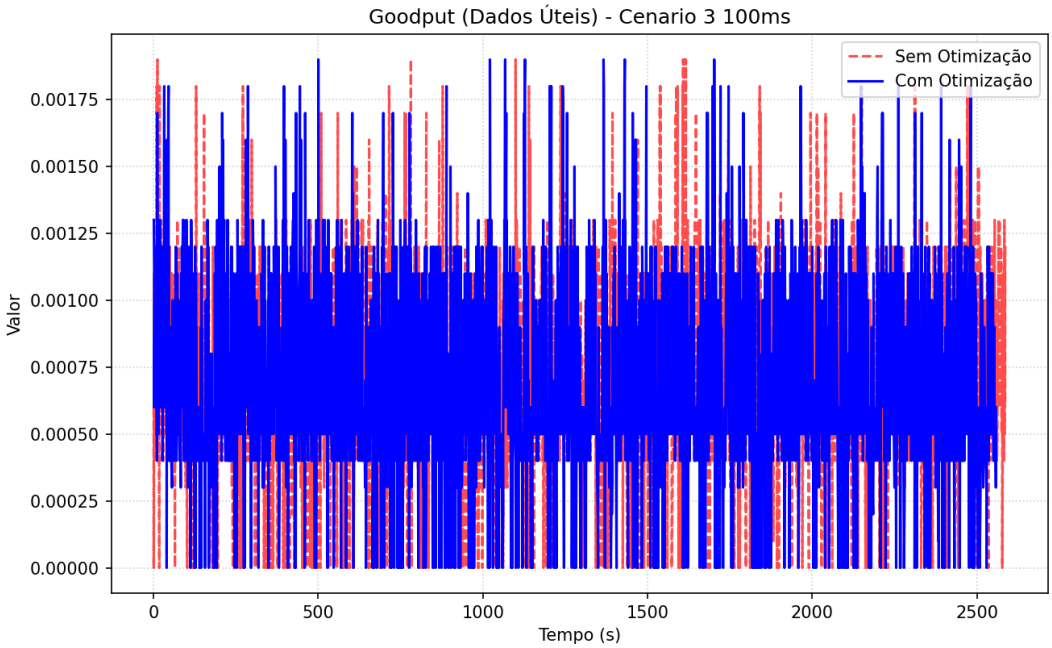


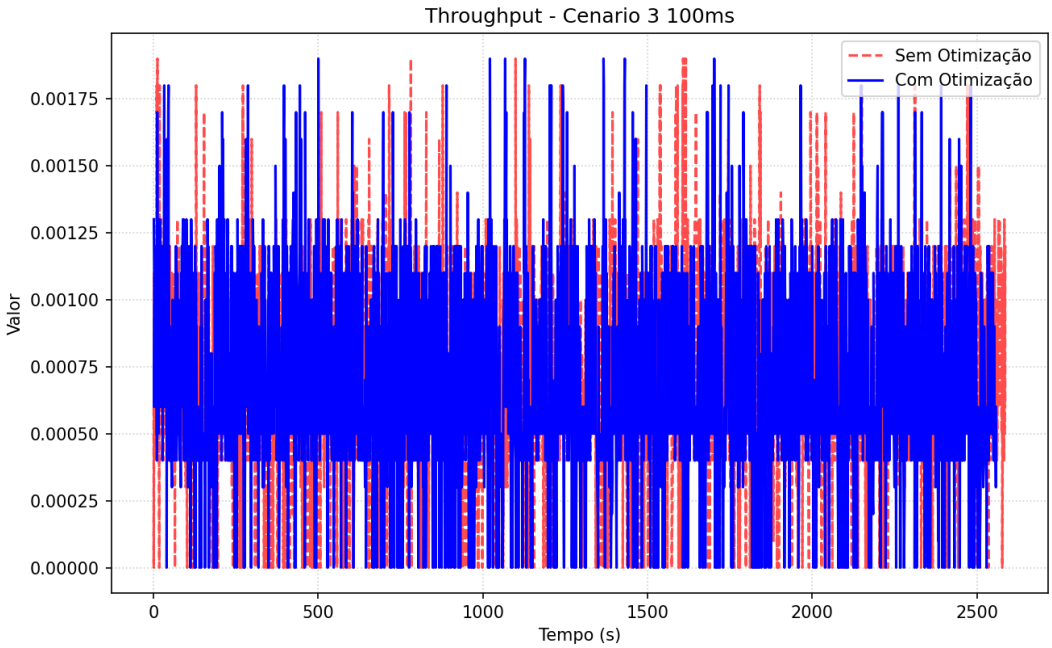
- Limiar de Slow Start (SSTHRESH): Permaneceu fixo no valor máximo para ambos testes, confirmando a ausência de eventos de perda/congestão que ativariam a redução do limiar.



- Throughput e Goodput: Novamente, não ocorreu retransmissões, mantando ambos os graficos e a diferença de desempenho é expressiva. A versão sem otimização sofreu severamente com a latência, sendo

limitado a enviar poucos pacotes e obrigada a esperar 200ms por confirmação, a vazão média foi baixíssima.

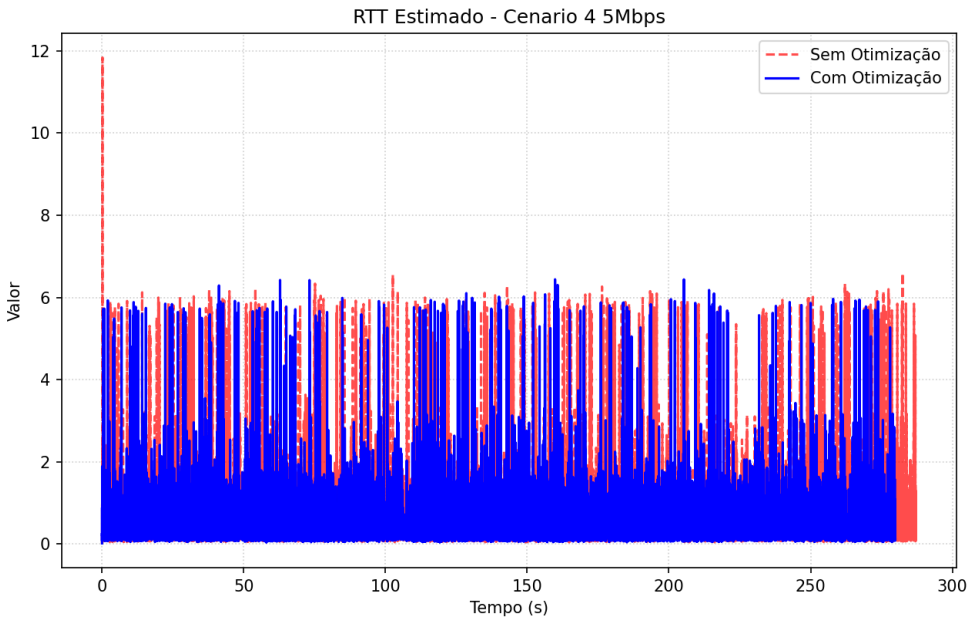




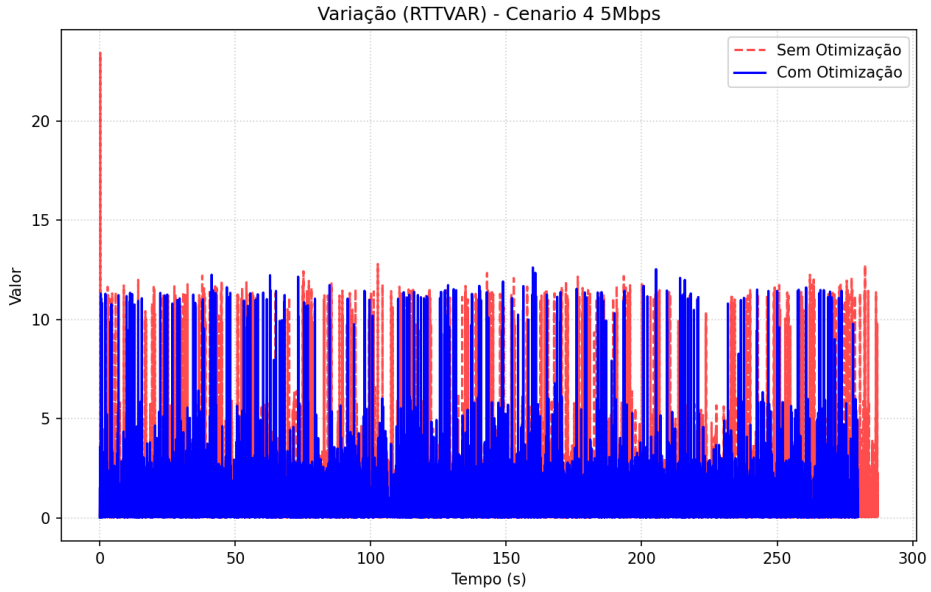
4.4 Limitação de banda (5 Mbps).

Neste último cenário, ocorreu a remoção do atraso de, mas foi aplicado um limitador de vazão de 5 Mbps utilizando o filtro de Buffer. O teste foi realizado com um arquivo de 5 MB para garantir duração suficiente para a estabilização do fluxo.

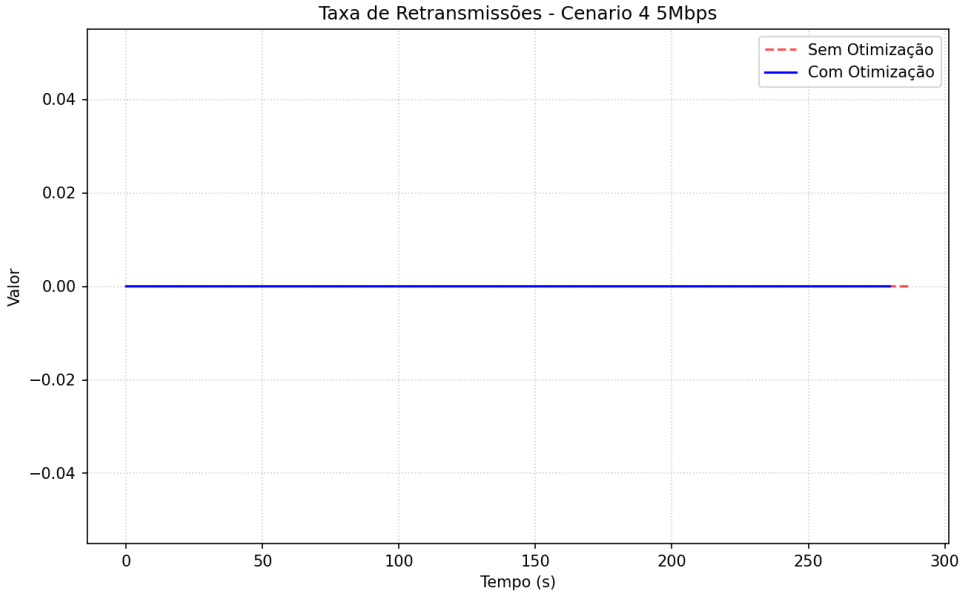
- RTT Estimado (Round-Trip Time): Os valores de RTT mantiveram-se abaixo de 1ms para ambos testes. Diferente dos cenários anteriores, foi adicionado o um delay. O comportamento observado indica que a fila não chegou a saturar a ponto de causar bufferbloat significativo, mantendo a latência estável.



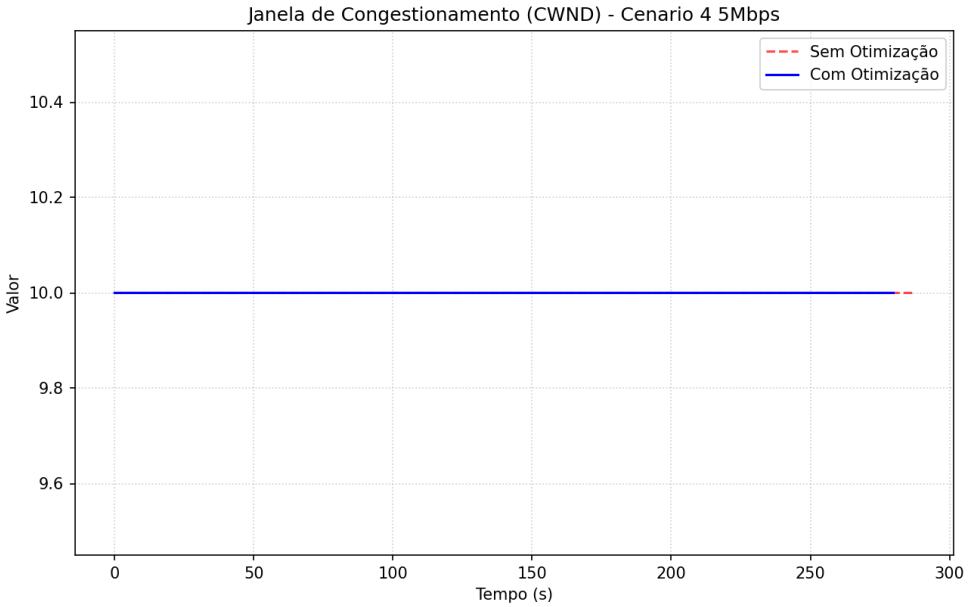
- Variação do RTT (RTTVAR): Acompanhando o RTT, a variação manteve-se mínima e estável, indicando uma entrega de pacotes consistente sem jitter perceptível.



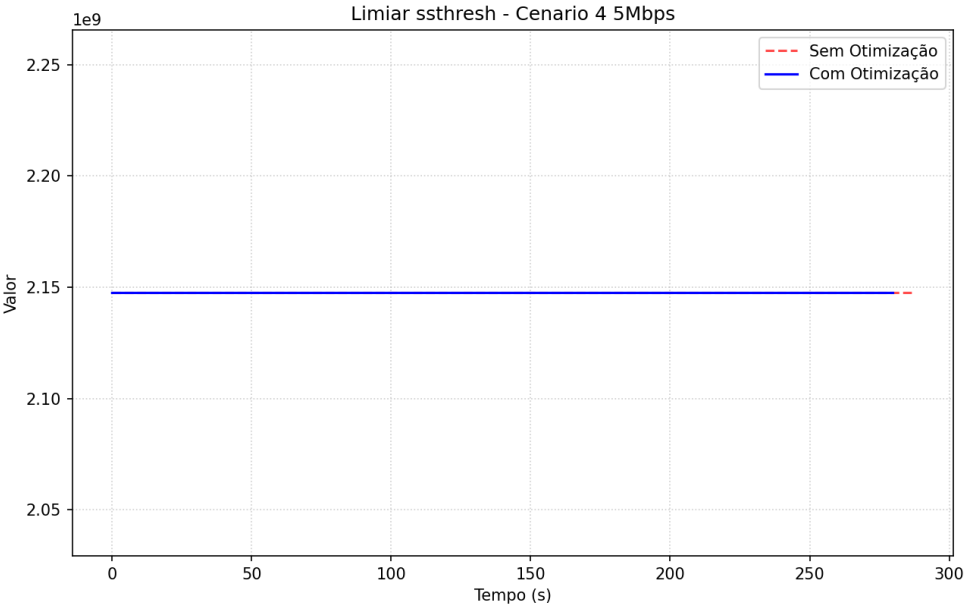
- Taxa de Retransmissões: Novamente, o gráfico encontra-se com valores zerados, demonstrando que o limitador de banda conseguiu gerenciar o fluxo de entrada sem precisar descartar pacotes por estouro de buffer na interface de rede.



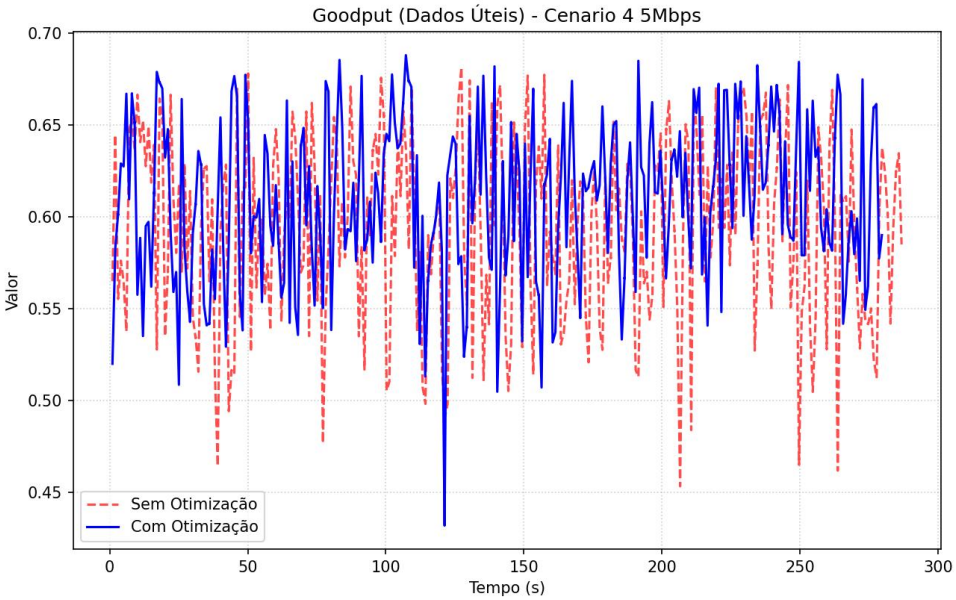
- Janela de Congestionamento (CWND): Permanecendo, novamente, fixa em 10 MSS tanto para a versão otimizada quanto para a não otimizada.

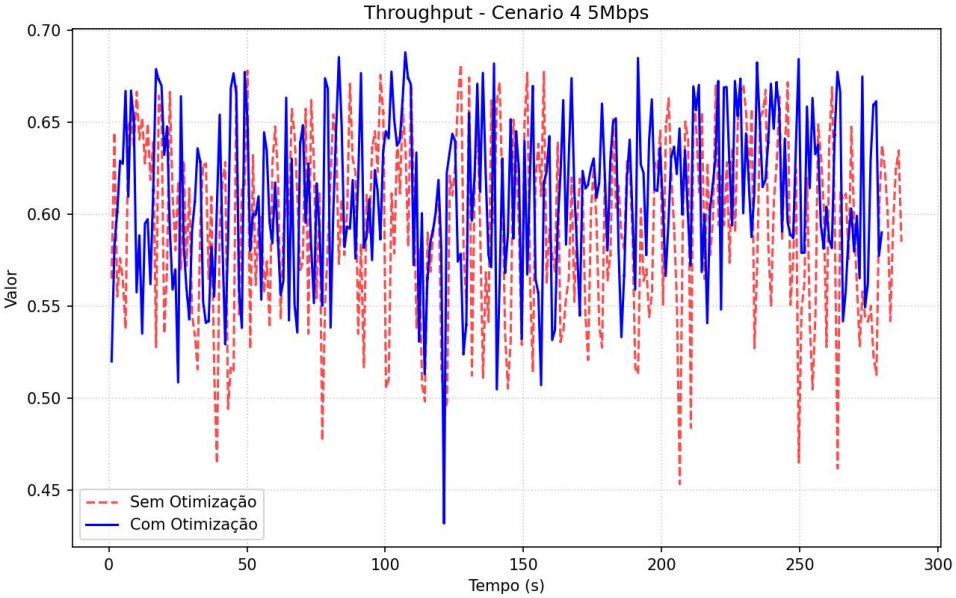


- Limiar de Slow Start (SSTHRESH): Permaneceu inalterado no valor máximo, confirmando a ausência de eventos de congestionamento que forçariam uma redução de taxa.



- Throughput e Goodput: Ambos os gráficos mostram que a taxa de transferência atingiu o teto próximo a 5 Mbps, não havendo diferença entre a versão sem otimização e a com otimização.


Goodput (Dados Úteis) - Cenario 4 5Mbps
Valor
Tempo (s)
Sem Otimização
Com Otimização
0.70
0.65
0.60
0.55
0.50
0.45
0
50
100
150
200
250
300


Throughput - Cenario 4 5Mbps
Valor
Tempo (s)
Sem Otimização
Com Otimização
0.70
0.65
0.60
0.55
0.50
0.45
0
50
100
150
200
250
300

5 Conclusão

O desenvolvimento do Proxy TCP permitiu a consolidação prática dos conceitos teóricos fundamentais sobre protocolos de transporte, evidenciando a complexidade da gestão de conexões realizada pelo kernel do Linux. A implementação da arquitetura de interceptação mostrou-se eficaz em manter a semântica do protocolo TCP, ao mesmo tempo em que possibilitou o monitoramento granular de métricas críticas como RTT, vazão e retransmissões. A estratégia de registro de dados e visualização gráfica provou ser essencial para a análise diagnóstica, permitindo correlacionar as variações nos parâmetros de rede simulados com o comportamento real da janela de congestionamento e a estabilidade da transmissão.

Os experimentos realizados demonstraram que as configurações padrão do sistema operacional frequentemente se mostram subótimas em condições adversas de rede, especialmente em cenários de alta latência. As políticas de otimização implementadas, especificamente o ajuste adaptativo de buffers baseado no cálculo de atraso de banda e o TCP Pacing, apresentaram resultados superiores aos da conexão padrão. A gestão dinâmica da memória do socket permitiu que o protocolo mantivesse o fluxo de dados contínuo mesmo diante de atrasos de 100ms, superando as limitações de throughput impostas pelos buffers estáticos e mitigando o impacto da perda de pacotes.

Por fim, a análise crítica dos testes revelou que o desempenho final de uma conexão é intrinsecamente dependente da interação entre a camada de transporte e a camada de aplicação. O comportamento síncrono (stop-and-wait) do cliente utilizado nos testes evidenciou como a latência pode tornar-se um gargalo na camada de aplicação, exigindo adaptações metodológicas no volume de dados para validar as métricas de rede. Em suma, o projeto cumpriu seus objetivos ao comprovar que a intervenção dinâmica nos parâmetros do TCP é uma estratégia eficiente para garantir resiliência e melhor aproveitamento da banda em ambientes de rede heterogêneos.