Universidade Federal de São João del Rei
Campus Tancredo Neves - Ciência da Computação

TRABALHO PRÁTICO 3

Oscar Alves Jonson Neto
Wallace Venancio Rosa
Guilherme Garcia de Oliveira

São João del-Rei
2025

Sumário

1. Introdução

O presente trabalho prático tem como objetivo a implementação, na linguagem de programação C, de um protocolo de comunicação baseado no protocolo UDP, incorporando mecanismos de confiabilidade por meio da técnica *Stop-and-Wait*. A proposta consiste em simular um ambiente de rede sujeito à perda de pacotes, exigindo que o protocolo desenvolvido assegure a entrega correta e completa de um arquivo entre um cliente e um servidor. Para garantir tal confiabilidade, serão utilizados mecanismos como confirmações de recebimento (*ACKs*), retransmissão de pacotes, controle de sequência e temporização (*timeout*). O sistema deverá ser capaz de lidar de forma robusta com perdas de pacotes e assegurar que o arquivo recebido seja fielmente idêntico ao original transmitido.

2. Protocolo Stop-and-Wait

O protocolo **Stop-and-Wait** é um dos métodos mais simples de controle de fluxo e confiabilidade utilizados em transmissões de dados em redes de computadores. Sua lógica baseia-se no envio de um único quadro (ou pacote) por vez, aguardando-se a confirmação de recebimento (*ACK*) por parte do receptor antes que o próximo quadro seja transmitido. Esse mecanismo garante que cada pacote seja entregue e processado corretamente antes de continuar a transmissão.

2.1 Funcionamento

O funcionamento do protocolo pode ser descrito da seguinte forma:

1. O emissor envia um pacote ao receptor e inicia um temporizador (*timeout*).

2. Ao receber o pacote corretamente, o receptor envia uma confirmação (*ACK*).

3. Se o emissor receber o *ACK* dentro do tempo esperado, ele envia o próximo pacote.

4. Caso o *ACK* não seja recebido dentro do intervalo definido, o emissor retransmite o pacote.

5. Cada pacote é identificado por um número de sequência (geralmente 0 ou 1, de forma alternada) para que o receptor possa detectar duplicatas em caso de retransmissões.

2.2 Vantagens

- **Simplicidade de implementação**: A lógica é direta, o que facilita sua aplicação e depuração.

- **Confiabilidade**: Em redes com perda ou corrupção de pacotes, garante a entrega correta de cada unidade de dados por meio de retransmissões.

2.3 Limitações

Apesar de sua confiabilidade, o protocolo Stop-and-Wait apresenta algumas limitações importantes, especialmente em contextos de redes modernas:

- **Baixo aproveitamento da largura de banda**: Como apenas um pacote pode estar "em trânsito" a qualquer momento, a capacidade total da rede pode ser subutilizada, especialmente em redes de alta latência ou alta taxa de transmissão.

- **Alta latência efetiva**: O tempo de espera entre o envio de um pacote e o recebimento do *ACK* pode causar lentidão significativa, principalmente em conexões de longa distância.

- **Desempenho limitado em comparação com protocolos mais avançados**: Protocolos como *Go-Back-N* utiliza janelas deslizantes que permitem o envio de múltiplos pacotes antes de receber os ACKs, melhorando significativamente o desempenho.

Por conta dessas limitações, o Stop-and-Wait é mais adequado a ambientes controlados.

3. Implementação

3.1 Cliente:

Inicialização e Configuração

A aplicação aceita argumentos de linha de comando que permitem:

- Especificar o caminho do arquivo a ser enviado;
- Ativar o modo detalhado de log (-v ou --verbose);
- Definir a probabilidade de perda simulada de pacotes (-l <prob>).

Após o processamento dos argumentos, é criado um socket UDP (SOCK_DGRAM) e configurado um *timeout* com setsockopt() para controlar o tempo máximo de espera por ACKs.

Envio do Pacote de Início (START)

Antes da transmissão dos dados, o cliente envia um pacote de controle do tipo PKT_START contendo o nome do arquivo a ser transmitido. Esse pacote é enviado de forma confiável:

- Um laço com número limitado de tentativas é utilizado para retransmitir o pacote caso o ACK não seja recebido dentro do tempo limite.

- O cliente verifica a integridade do pacote por meio de um *checksum* e aguarda a confirmação (PKT_ACK) do servidor antes de prosseguir.

Transmissão dos Dados (DATA)

A transferência dos dados é feita por meio da leitura do arquivo em blocos de tamanho fixo (MAX_PAYLOAD_SIZE), encapsulados em pacotes PKT_DATA. O cliente aplica a lógica do Stop-and-Wait:

- Cada pacote de dados é enviado com um número de sequência (0 ou 1 alternado).

- Após o envio, o cliente aguarda um ACK com o mesmo número de sequência.

- Se o ACK correto não for recebido no tempo esperado ou for perdido (simulação de perda), o pacote é retransmitido até o número máximo de tentativas.

Esse processo garante que cada segmento de dados seja recebido corretamente e em ordem.

Encerramento da Transmissão (EOT)

Após o envio de todos os dados, um pacote especial do tipo PKT_EOT (End of Transmission) é enviado para indicar o fim da transmissão:

- Este pacote também segue o mesmo mecanismo confiável com retransmissão até o recebimento de um ACK correspondente.

- Caso o ACK não seja recebido mesmo após várias tentativas, a transmissão é encerrada com uma advertência, mas o cliente finaliza normalmente.

Simulação de Perda de Pacotes

A função simulate_loss() é utilizada para simular a perda de pacotes e ACKs com base na probabilidade configurada pelo usuário. Isso permite testar a robustez da implementação frente a falhas comuns em redes reais.

Relatório de Estatísticas

Ao final da execução, o cliente exibe estatísticas como:

- Tempo total de transferência;
- Total de pacotes enviados;
- Número de retransmissões;
- Taxa de retransmissão (indicador da confiabilidade da rede simulada).

3.2 Servidor

Inicialização do Servidor

O servidor cria um socket UDP e o associa à porta definida (12345), permitindo receber pacotes de qualquer endereço IP. Em seguida, abre um arquivo para escrita binária, no qual os dados recebidos serão armazenados. O servidor também inicializa a variável expected_sequence, que representa o número de sequência do próximo pacote de dados válido que se espera receber (inicialmente 0).

Laço Principal de Recepção

O servidor entra em um laço contínuo, onde aguarda a recepção de pacotes do cliente usando a função recvfrom(). Os pacotes são armazenados em um buffer e desserializados para uma estrutura Packet, que contém os campos: tipo, número de sequência, comprimento, checksum e dados (*payload*).

Verificação de Integridade

Antes de qualquer processamento, o servidor calcula o *checksum* dos dados recebidos e compara com o valor enviado no cabeçalho. Caso o pacote esteja corrompido, ele é descartado sem gerar ACK, e o cliente será responsável por retransmiti-lo ao detectar a ausência de resposta.

Processamento de Pacotes DATA

Ao receber um pacote do tipo PKT_DATA:

- Se o número de sequência recebido corresponde ao valor esperado (expected_sequence), o pacote é considerado válido:
    - Os dados são escritos no arquivo de saída.
    - O número de sequência esperado é alternado (0 → 1 ou 1 → 0).
- Caso contrário, o pacote é considerado **duplicado** ou fora de ordem:
    - Nenhum dado é escrito no arquivo.
    - Um contador de pacotes duplicados é incrementado.

Em ambos os casos, o servidor envia um *ACK* contendo o número de sequência do pacote recebido, informando ao cliente que este pacote (válido ou duplicado) foi processado ou que o ACK anterior foi perdido.

Encerramento da Transmissão (EOT)

Quando o servidor recebe um pacote do tipo PKT_EOT, ele interpreta como sinal de fim da transmissão:

- Um ACK é enviado como confirmação do recebimento do EOT.

- O servidor encerra o laço de recepção e fecha o arquivo de saída.

Simulação de Perda

A função simulate_loss() é utilizada tanto para simular a perda de pacotes de entrada quanto de ACKs enviados. Isso permite testar a tolerância da implementação em cenários realistas com falhas de comunicação.

Estatísticas Finais

Ao final da execução, o servidor exibe um resumo contendo:

- O número total de pacotes de dados recebidos.
- A quantidade de pacotes duplicados descartados.
- A quantidade de pacotes corrompidos.

4. Resultados Obtidos

Iremos abordar e discutir sobre o funcionamento e desempenho dos códigos nos testes feitos.

4.1 Testes com o primeiro arquivo:

Especificação do arquivo:

Formato do Arquivo: .txt

Peso: 10 KB

Teste 1 – Sem Perda de Pacotes (0%)

- **Tempo total de transferência:** 1 segundo
- **Pacotes enviados:** 12 (incluindo START e EOT)
- **Retransmissões:** 0

- **Taxa de retransmissão:** 0%

Este teste representa um cenário ideal, sem qualquer perda de pacotes ou ACKs. A transferência foi realizada de forma contínua, sem interrupções, e todos os pacotes foram reconhecidos na primeira tentativa. O número de pacotes foi suficiente para transmitir os dados sem redundância, e o tempo de conclusão foi mínimo.

Teste 2 – Perda Simulada de 10% (Cliente)

- **Tempo total de transferência:** 6 segundos
- **Pacotes enviados:** 19
- **Retransmissões:** 7
- **Taxa de retransmissão:** 36,84%

Neste cenário, foi simulada uma perda de 10% nos pacotes enviados pelo cliente. Como consequência:

- Houve necessidade de retransmissões frequentes, especialmente nos pacotes de dados e até mesmo no pacote START.

- O número total de pacotes enviados aumentou em mais de 50% em relação ao cenário ideal (de 12 para 19).

- A eficiência do protocolo foi comprometida, já que a taxa de retransmissão atingiu quase 37%, e o tempo total de transferência aumentou em 6 vezes.

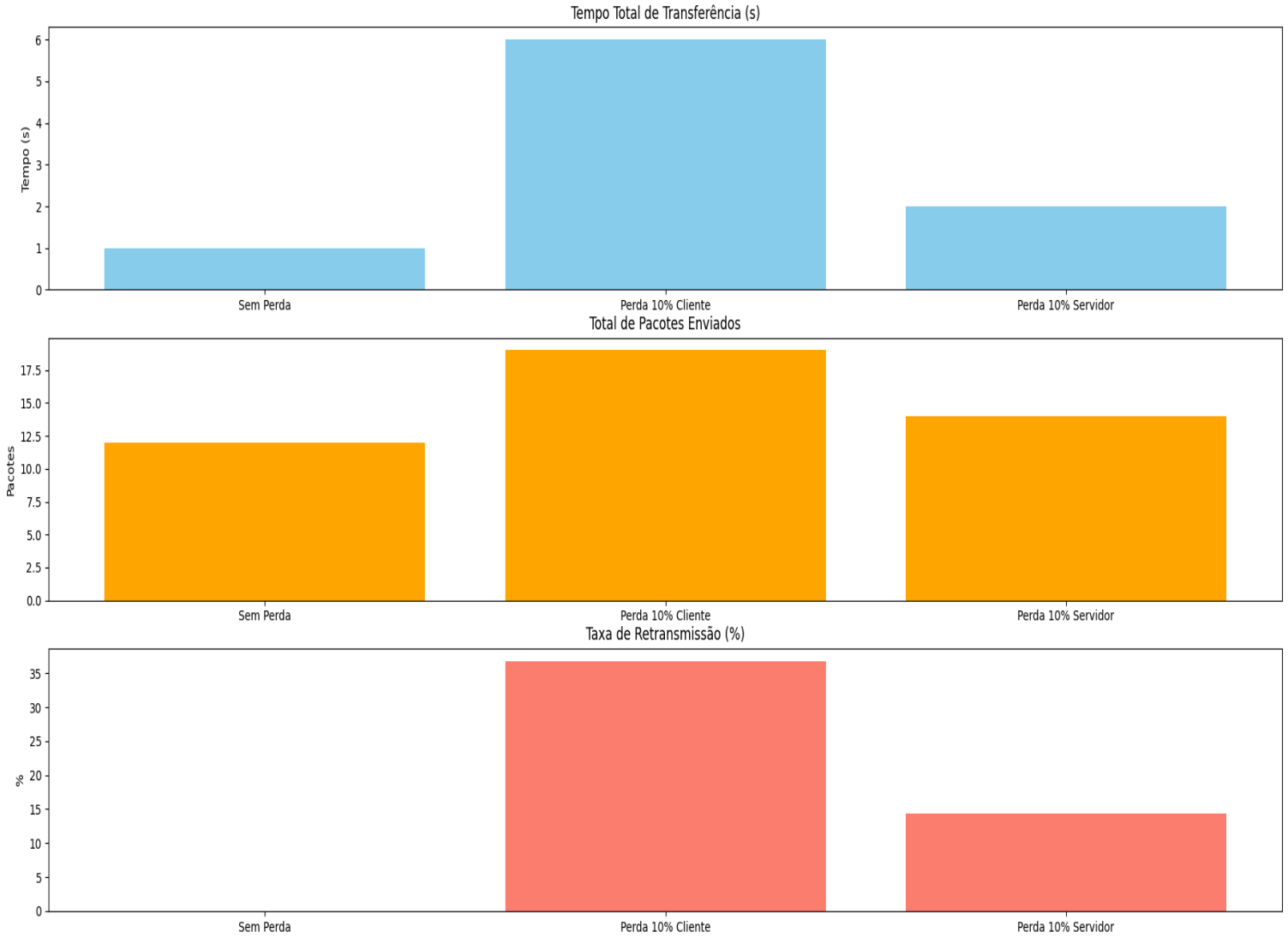
- O servidor ainda conseguiu tratar adequadamente os **pacotes duplicados**, mantendo a integridade do arquivo recebido

Teste 3 — Perda de Pacotes no Servidor (10%)

- **Tempo total de transferência:**2 segundos
- **Pacotes enviados:** 14

- **Retransmissões:** 2
- **Taxa de retransmissão:** 14,29%

Este teste demonstra que a implementação do protocolo Stop-and-Wait é **robusta a perdas de até 10% no servidor**, mantendo uma boa taxa de entrega com baixa retransmissão e tempo de transmissão aceitável. A perda no lado do servidor causa menos impacto que no cliente, o que era esperado, já que o cliente controla o fluxo e retransmite sempre que necessário.



4.2 Testes com o segundo arquivo:

Especificação do arquivo:

Formato do Arquivo: .jpeg

Peso: 1.087 KB

Teste 1 – Sem Perda Configurada

- **Tempo total:** 1 segundo
- **Pacotes enviados:** 1.089
- **Retransmissões:** 0
- **Taxa de retransmissão:** 0,00%

Neste teste, como a simulação de perda estava desativada, a transferência ocorreu de forma ideal. Todos os pacotes (inclusive os de controle: START e EOT) foram entregues corretamente e confirmados com seus respectivos ACKs na primeira tentativa. O protocolo *Stop-and-Wait* demonstrou eficiência e confiabilidade em ambientes sem perdas, conseguindo transferir o arquivo completo em apenas um segundo, sem qualquer necessidade de retransmissão.

Teste 2 – Perda Configurada em 10% no Lado do Cliente

- **Tempo total:** 101 segundos
- **Pacotes enviados:** 1.303
- **Retransmissões:** 214
- **Taxa de retransmissão:** 16,42%

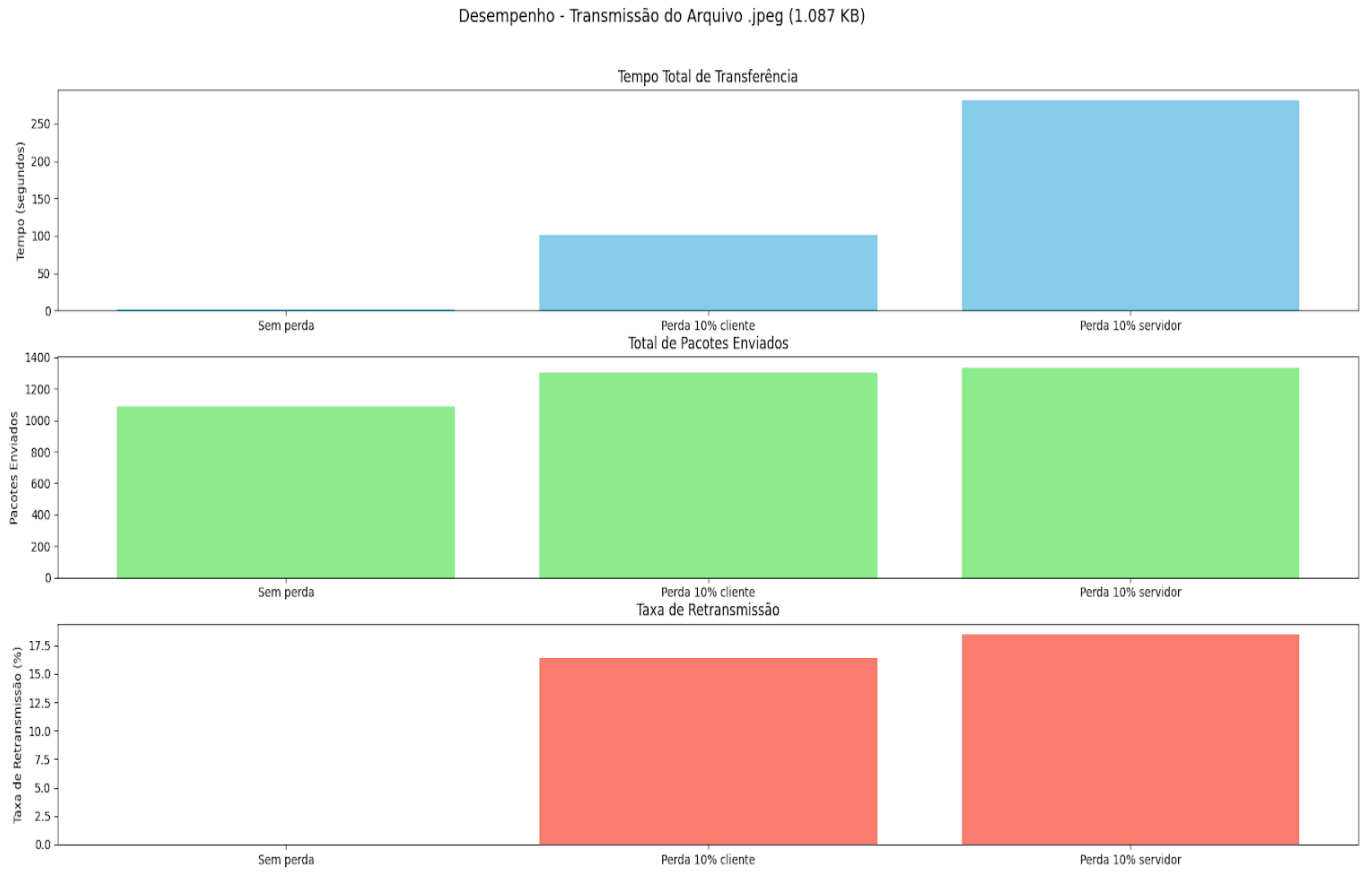
Com a simulação de perda ativada no lado do cliente (ou seja, pacotes de dados podendo ser perdidos antes de chegar ao servidor), o desempenho caiu significativamente. O tempo de transferência aumentou drasticamente para 101 segundos, e houve uma taxa expressiva de retransmissões (16,42%). Isso ocorre porque, no *Stop-and-Wait*, o cliente só envia o próximo pacote após receber o ACK anterior — qualquer perda de pacote ou de ACK implica em timeout e retransmissão,

o que afeta diretamente o tempo total da transferência. Mesmo assim, o sistema foi capaz de concluir a transmissão com sucesso.

Teste 3 – Perda Configurada em 10% no Lado do Servidor

- **Tempo total:** 281 segundos
- **Pacotes enviados:** 1.336
- **Retransmissões:** 247
- **Taxa de retransmissão:** 18,49%

Neste teste**,** a perda de pacotes simulados no lado do servidor impactou ainda mais o desempenho da transferência. Quando o servidor perde um pacote, o cliente não recebe o ACK correspondente e é forçado a retransmitir o mesmo pacote após o timeout. Esse cenário gerou a maior taxa de retransmissão (18,49%) e o maior tempo de transferência (281 segundos), mesmo com uma quantidade de pacotes totais semelhante ao teste anterior. Este resultado destaca como a perda no receptor afeta diretamente a eficiência do protocolo, mesmo quando o cliente está funcionando corretamente.



4.2 Testes com o terceiro arquivo:

Especificação do arquivo:

Formato do Arquivo: .pdf

Peso: 1.484 KB

Teste 1 – Sem Perda Configurada

- **Tempo total:** 1 segundo
- **Pacotes enviados:** 1.486
- **Retransmissões:** 0
- **Taxa de retransmissão:** 0,00%

O desempenho neste cenário foi excelente. Como não havia perda configurada, a transferência foi concluída de forma eficiente e rápida, com todos os pacotes sendo reconhecidos corretamente na primeira tentativa. Esse teste confirma que o protocolo *Stop-and-Wait*, apesar de sua simplicidade, funciona bem em ambientes ideais, com latência baixa e nenhum tipo de perda ou erro de transmissão.

Teste 2 – Perda Configurada em 10% no Lado do Cliente

- **Tempo total:** 183 segundos
- **Pacotes enviados:** 1.833
- **Retransmissões:** 347
- **Taxa de retransmissão:** 18,93%

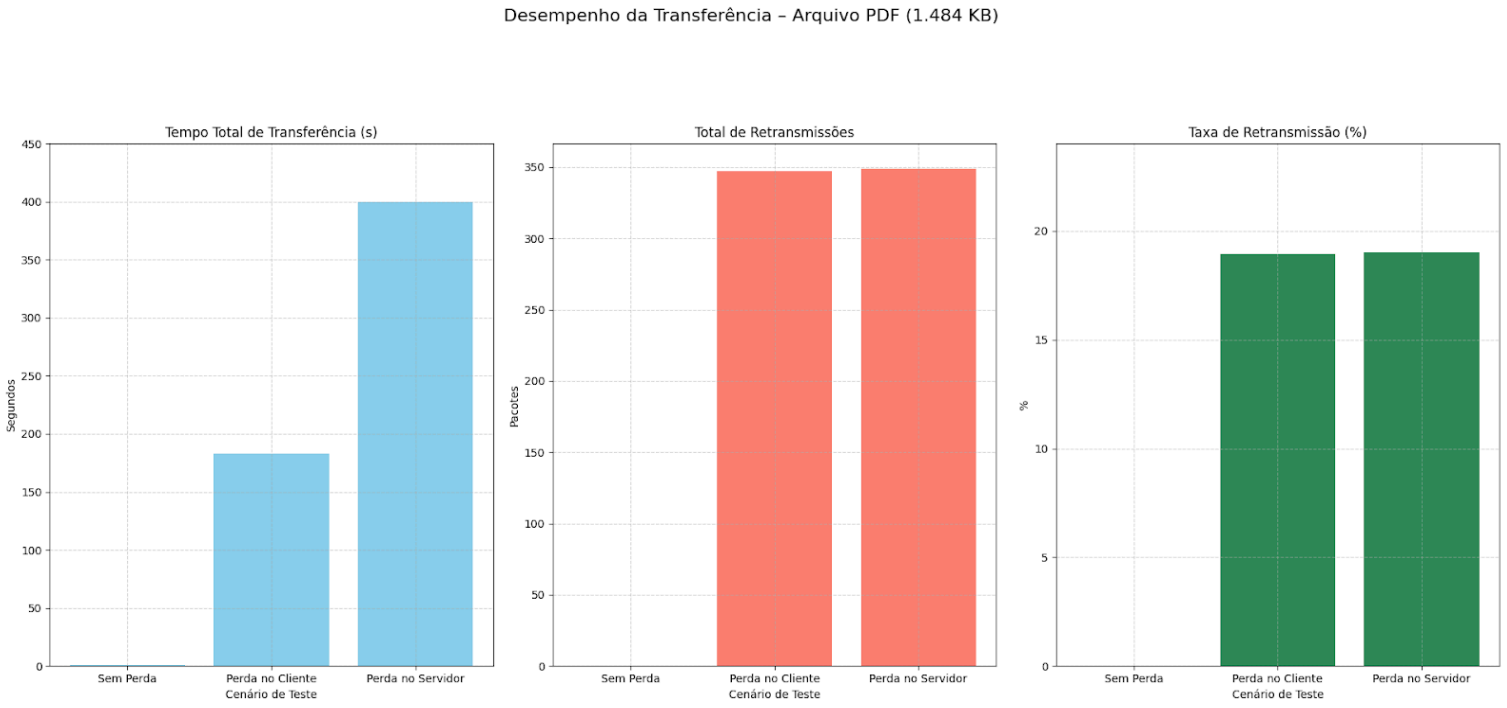
Neste cenário, a perda foi simulada no envio dos pacotes pelo cliente. Isso gerou um impacto expressivo na performance, elevando a taxa de retransmissão para quase 19% e aumentando significativamente o tempo de transferência (de 1 para 183 segundos). Como o protocolo exige o recebimento do ACK antes de prosseguir, qualquer perda de pacote exige retransmissão — o que se torna custoso em

arquivos maiores. Ainda assim, a confiabilidade do protocolo garantiu a entrega correta, mesmo com penalidade no tempo.

Teste 3 – Perda Configurada em 10% no Lado do Servidor

- **Tempo total:** 400 segundos
- **Pacotes enviados:** 1.835
- **Retransmissões:** 349
- **Taxa de retransmissão:** 19,02%

A perda simulada no lado do servidor teve o maior impacto até agora. Aqui, os pacotes enviados pelo cliente nem sempre foram processados ou confirmados devido à simulação de perda no receptor. Como consequência, o cliente precisou retransmitir muitos pacotes, elevando a taxa de retransmissão para 19,02% e aumentando drasticamente o tempo total da transferência para 400 segundos. Isso demonstra claramente que a perda no destino (ou seja, no lado do servidor) é ainda mais prejudicial para a eficiência do *Stop-and-Wait*, especialmente quando se trata de arquivos grandes.



5. Conclusão

Os testes realizados permitiram avaliar o comportamento do protocolo de parada e espera (Stop-and-Wait) sobre UDP, com simulação de diferentes condições de perda de pacotes. O objetivo foi verificar o impacto da perda (tanto de dados quanto de confirmações) na integridade e no desempenho da transferência de arquivos de tamanhos distintos.

Durante os testes, a simulação de perda foi configurada para representar dois cenários distintos: perda no lado do cliente (transmissor) e no lado do servidor (receptor). É importante destacar que a **simulação de perda considera tanto a perda de pacotes de dados quanto a perda de pacotes de confirmação (ACK)**, o que permite observar situações reais de timeout e necessidade de retransmissão.

Nos casos sem perda configurada, as transferências foram concluídas com eficiência, em tempo reduzido e **sem retransmissões**. Por outro lado, ao introduzir uma **probabilidade de perda de 10%**, houve um aumento significativo no número de pacotes enviados, no tempo total de transferência e na taxa de retransmissão. Esse impacto foi mais perceptível com arquivos maiores, como imagens JPEG e documentos PDF, devido à maior quantidade de pacotes envolvidos.

Além disso, observou-se que a perda de ACKs, mesmo com pacotes de dados sendo corretamente entregues, força o cliente a retransmitir pacotes desnecessariamente, contribuindo para o aumento da sobrecarga. O protocolo implementado conta com um limite de tentativas definido por MAX_RETRIES, e caso esse limite seja excedido em retransmissões consecutivas para o mesmo pacote, o cliente encerra a conexão automaticamente, assumindo falha irreversível na comunicação.

Em resumo, os testes mostram a eficácia do protocolo em redes estáveis e sua vulnerabilidade a perdas, principalmente quando envolvem confirmações. O mecanismo de retransmissão garante a integridade dos dados, mas também

introduz latência adicional. Portanto, em ambientes com perdas significativas, são recomendadas estratégias adicionais, como janelas deslizantes ou códigos de correção de erro, para melhorar o desempenho da transmissão.

6. Referências

Jim Kurose Homepage. Disponível em: <https://gaia.cs.umass.edu/kurose_ross/index.php>.

GEEKSFORGEEKS. Stop and Wait ARQ. Disponível em: <https://www.geeksforgeeks.org/computer-networks/stop-and-wait-arq/>.