

Pontifícia Universidade Católica do Rio Grande do Sul

Laboratório de Redes de Computadores

Engenharia de Software

**Carolina Ferreira, Felipe Freitas, Luiza Heller e Mateus
Caçabuena**

Relatório da Aplicação e Análise de Tráfego do Trabalho 1

Porto Alegre

2024

Sumário

| | |
|---------------------------------|----|
| Sumário | 2 |
| 1. Introdução | 3 |
| 2. Aplicação Implementada | 4 |
| 2.1. Server UDP | 4 |
| 2.2. Server TCP | 6 |
| 2.3. Client UDP | 8 |
| 2.4. Client TCP | 10 |
| 3. Análise de Tráfego..... | 14 |
| 4. Conclusão | 21 |

1. Introdução

O presente trabalho de laboratório tem como objetivo desenvolver um protocolo de aplicação para um chat utilizando a arquitetura cliente/servidor, utilizando-se dos protocolos de transporte TCP/IPv4 e UDP/IPv4. O foco deste trabalho é proporcionar uma análise comparativa do desempenho de ambos os protocolos em condições normais e adversas da rede, visando avaliar suas características e impactos em uma aplicação de troca de mensagens e arquivos de texto entre usuários.

Através desta implementação, pretende-se explorar o funcionamento prático dos protocolos de transporte TCP e UDP, verificando suas diferenças em termos de confiabilidade, controle de fluxo e volume de tráfego gerado na rede. Para tanto, foi desenvolvido um protocolo de aplicação específico que permite operações básicas como registro de usuários, envio de mensagens e transferência de arquivos entre clientes, com o servidor atuando como hub central para gerenciar e intermediar as comunicações.

Adicionalmente, será realizada uma análise detalhada do tráfego de rede gerado pela aplicação, utilizando ferramentas como o Wireshark para monitorar o comportamento dos pacotes enviados e recebidos. Além disso, utilizamos o Clumsy para simular distúrbios na rede, introduzindo condições adversas na rede, como latência e perda de pacotes, para verificar como o TCP e o UDP reagem a esses cenários. O trabalho culmina com a comparação detalhada entre os dois protocolos em termos de volume de tráfego, eficiência e integridade dos dados transmitidos.

Dessa forma, o trabalho não apenas visa o desenvolvimento prático de uma aplicação de chat, mas também a compreensão aprofundada dos impactos e diferenças entre as abordagens de comunicação baseadas em TCP e UDP.

2. Aplicação Implementada

2.1. Server UDP

Este código implementa o lado do **servidor** de uma aplicação de chat baseada em **UDP**. Ele gerencia a comunicação entre os clientes, permitindo o envio de mensagens e arquivos de texto, além de gerenciar o registro e a remoção de clientes conectados. O servidor utiliza o protocolo UDP para receber mensagens e responder adequadamente conforme o comando enviado pelos clientes.

Resumo das principais funções e componentes:

1. Bibliotecas Importadas:

- socket: Utilizado para configurar a comunicação via **UDP**.
- clients: Lista de clientes conectados ao servidor (definida em outro arquivo).
- config: Importa diversas constantes e parâmetros (como mensagens de ACK e NACK, tamanho máximo da mensagem, prefixos de comandos, e o endereço do servidor).

2. Configuração do Servidor:

- server_socket: Cria o socket UDP e o liga ao endereço especificado (server_udp), por padrão, localhost:13000.

3. Função main:

- Exibe uma mensagem de prontidão do servidor.
- Aguarda a recepção de mensagens dos clientes, processando-as com a função handle_message.

4. Função **handle_message**:

- Lida com as mensagens recebidas e decide qual ação tomar com base no comando enviado (prefixo da mensagem). Os comandos suportados são:
 - /REG: Registra o cliente.
 - /WHOAMI: Retorna o nome do cliente, seu host e porta.
 - /MSG: Envia uma mensagem para todos os clientes conectados.
 - /FILE: Envia um arquivo para os clientes.
 - /QUIT: Remove o cliente da lista de conectados.
- Caso o comando seja inválido, responde com um NACK_INVALID.

5. Função **register**:

- Adiciona o cliente (nickname e endereço) à lista de clientes conectados.

6. Função **unregister**:

- Remove o cliente da lista de conectados ao servidor, exibindo uma mensagem de desconexão.

7. Função **send_message**:

- Se a mensagem começar com @, a mensagem é enviada como privada para o destinatário especificado.
- Caso contrário, a mensagem é enviada para todos os outros clientes conectados.

8. Função **send_file**:

- Funciona de maneira semelhante ao envio de mensagens, porém com arquivos de texto.

- Se o nome do arquivo começar com @, o arquivo será enviado a um destinatário específico.

9. Mecanismos de Erro e Encerramento:

- O servidor pode ser parado com um **KeyboardInterrupt** (Ctrl+C), ou pode lidar com exceções, imprimindo erros relevantes antes de fechar o socket.

Este servidor implementa a lógica básica de registro, envio de mensagens e arquivos via UDP, com tratamento de clientes e operações comuns de chat.

2.2. Server TCP

Este código implementa o lado **servidor** de uma aplicação de chat baseada em **TCP**. Ele gerencia a comunicação com os clientes de forma síncrona, utilizando a função `select` para monitorar vários sockets ao mesmo tempo. Assim, é possível lidar com múltiplas conexões simultâneas, permitindo que o servidor envie e receba mensagens e arquivos de texto entre os clientes.

Resumo das principais funções e componentes:

1. Bibliotecas Importadas:

- `socket`: Usado para configurar a comunicação via **TCP**.
- `select`: Permite ao servidor monitorar vários sockets simultaneamente, ajudando a gerenciar múltiplos clientes sem ter que se preocupar em ficar fechando e reabrindo as portas.
- `clients`: Lista de clientes conectados (definida em outro arquivo).
- `config`: Importa constantes e parâmetros importantes, como mensagens de confirmação (ACK), tamanho máximo de mensagens, e prefixos de comandos (e.g., /REG, /MSG, etc.).

2. Configuração do Servidor:

- O servidor cria um **socket TCP**, o associa ao endereço e permite reutilização da porta (SO_REUSEADDR), o que facilita a reinicialização.
- O servidor entra em modo de escuta com listen() e suporta até MAX_SERVER_CONNECTIONS simultâneas, por padrão, 5.

3. Função main:

- Exibe uma mensagem de prontidão do servidor.
- Usa select.select() para monitorar os sockets de clientes e o socket do servidor.
- Aceita novas conexões de clientes, recebendo as mensagens e repassando para o tratamento adequado via handle_message.

4. Função handle_message:

- Lida com as mensagens recebidas dos clientes, realizando ações com base nos comandos (prefixos) enviados:
 - /REG: Registra o cliente.
 - /WHOAMI: Retorna o nome e detalhes do cliente.
 - /MSG: Envia uma mensagem a todos os clientes conectados.
 - /FILE: Recebe um arquivo e envia a outros clientes.
 - /QUIT: Remove o cliente da lista de conectados.
- Em caso de comando inválido, o cliente recebe uma mensagem de erro (NACK_INVALID).

5. Função receive_message:

- Lê e decodifica a mensagem enviada por um cliente via TCP.

6. Funções de Registro e Desconexão (register e unregister):

- A função register adiciona o cliente à lista de conectados.

- A função `unregister` remove o cliente, desregistrando-o da aplicação.

7. Função `send_message`:

- Envia mensagens para os demais clientes conectados. Se a mensagem for privada (começa com `@`), é enviada apenas ao destinatário específico.

8. Funções de Arquivos (`receive_file` e `send_file`):

- `receive_file`: Recebe um arquivo de um cliente e o salva localmente.
- `send_file`: Envia o arquivo para um cliente específico ou para todos, conforme o comando.

9. Tratamento de Erros e Finalização:

- O servidor captura exceções e, em caso de erro ou interrupção manual, fecha o socket e encerra a aplicação de forma adequada.

Este servidor implementa as funcionalidades básicas de chat e troca de arquivos utilizando **TCP**, permitindo que múltiplos clientes se conectem simultaneamente e troquem mensagens ou arquivos em tempo real.

2.3. Client UDP

Este código implementa o lado **cliente** de um aplicativo de chat baseado em **UDP**. Ele permite que o cliente envie mensagens e arquivos para um servidor UDP e receba respostas. O cliente pode se registrar, enviar mensagens, transferir arquivos, e sair do servidor.

Resumo das principais funções e componentes:

1. Bibliotecas Importadas:

- `socket`: Usado para configurar a comunicação via **UDP**.

- argv: Permite acessar argumentos passados via linha de comando (como o número da porta).
- Outros módulos locais (como config, print) fornecem constantes e funções auxiliares para lidar com a comunicação e exibição.

2. Função main:

- Função principal que inicia a interação do cliente com o servidor.
- **Processo Principal:**
 1. O cliente exibe as opções de comandos disponíveis com print_options.
 2. Espera por uma entrada do usuário, que pode ser uma mensagem ou um comando especial.
 3. Caso a mensagem seja o comando de sair (/QUIT), o cliente envia a mensagem para o servidor, espera a confirmação de desligamento (ACK_UNREG) e encerra.
 4. Se for outra mensagem, ela é enviada ao servidor, e o cliente espera pela resposta.
- As respostas do servidor são tratadas por funções de exibição, como get_print, que escolhem a melhor maneira de apresentar o resultado ao usuário.

3. Função send_message:

- Envia uma mensagem ao servidor. Se o comando for para envio de arquivo (/FILE), chama a função send_file.

4. Função send_file:

- Lida com o envio de arquivos ao servidor.
- **Processo de Envio:**
 1. Lê o arquivo especificado no comando.
 2. Envia o nome do arquivo ao servidor.

3. Envia o conteúdo do arquivo em pedaços, com o tamanho máximo permitido (`MESSAGE_MAX_SIZE_UDP`), por padrão, 1024 bytes.
4. Quando todo o conteúdo é enviado, envia o marcador de finalização do arquivo (EOF).

5. Tratamento de Erros:

- O código lida com várias exceções possíveis:
 - **FileNotFoundError**: Para o caso de o arquivo não ser encontrado quando o cliente tenta enviá-lo.
 - **KeyboardInterrupt**: Para parar o cliente de forma controlada quando o usuário interrompe a execução.
 - **Argumentos Inválidos**: Verifica se o número da porta foi passado e se ele é válido.

Fluxo de Execução:

1. O cliente é iniciado com a porta como argumento.
2. Exibe as opções de comando e espera por uma entrada.
3. De acordo com o comando, envia uma mensagem ou arquivo ao servidor.
4. O cliente recebe a resposta e a exibe.
5. Se o cliente quiser sair, envia o comando `/QUIT`, recebe a confirmação e encerra.

Este cliente UDP oferece uma interface simples para comunicação com um servidor, permitindo envio de mensagens e arquivos de maneira eficiente.

2.4. Client TCP

Este código implementa o lado **cliente** de um sistema de comunicação baseado em **TCP**. O cliente pode se conectar a um servidor, também TCP, enviar

mensagens ou arquivos e receber respostas. Ele também pode desconectar-se voluntariamente por meio de um comando específico.

Resumo das principais funções e componentes:

1. Bibliotecas Importadas:

- socket: Usada para criar a conexão **TCP** com o servidor.
- Outros módulos locais, como config e print, fornecem constantes e funções auxiliares para manipulação de mensagens e exibição no terminal.

2. Função main:

- É a função principal que estabelece a conexão com o servidor e lida com a comunicação.
- **Processo Principal:**
 1. O cliente tenta se conectar ao servidor no endereço e porta especificados pela constante `server_tcp`.
 2. Exibe as opções disponíveis para o usuário com a função `print_options`.
 3. Coleta a entrada do usuário, que pode ser uma mensagem ou um comando.
 4. Se o comando for o de sair (`/QUIT`), ele envia a mensagem ao servidor e, após a confirmação, encerra a conexão.
 5. Para outras mensagens, o cliente envia o conteúdo ao servidor e aguarda uma resposta, que é exibida no terminal.
- A função utiliza blocos `try-except-finally` para gerenciar erros e garantir que a conexão seja fechada corretamente em qualquer cenário.

3. Função `send_message`:

- Envia a mensagem para o servidor.

- Se a mensagem for o comando de envio de arquivo (/FILE), chama a função `send_file` para tratar a transferência.

4. Função `send_file`:

- Lida com o envio de arquivos para o servidor.
- **Processo de Envio:**
 1. Extrai o nome do arquivo a partir do comando /FILE.
 2. Envia o nome do arquivo ao servidor.
 3. Lê o arquivo em pedaços, utilizando o tamanho máximo de mensagem (`MESSAGE_MAX_SIZE_TCP`), e envia cada pedaço ao servidor.
 4. Após enviar todo o conteúdo do arquivo, envia o marcador EOF para indicar o fim da transferência.
- Se o arquivo não for encontrado, captura a exceção `FileNotFoundError` e exibe uma mensagem de erro apropriada.

5. Tratamento de Erros:

- O código lida com vários tipos de erros:
 - **`FileNotFoundError`:** Quando o arquivo especificado para envio não é encontrado.
 - **`KeyboardInterrupt`:** Para permitir que o cliente seja interrompido manualmente (Ctrl + C) de forma segura.
 - **`Exception`:** Captura outros erros e exibe mensagens detalhadas sobre o problema.

Fluxo de Execução:

1. O cliente tenta se conectar ao servidor usando o endereço e a porta definidos em `server_tcp`.
2. O usuário recebe opções de comandos disponíveis e escolhe o que deseja fazer.

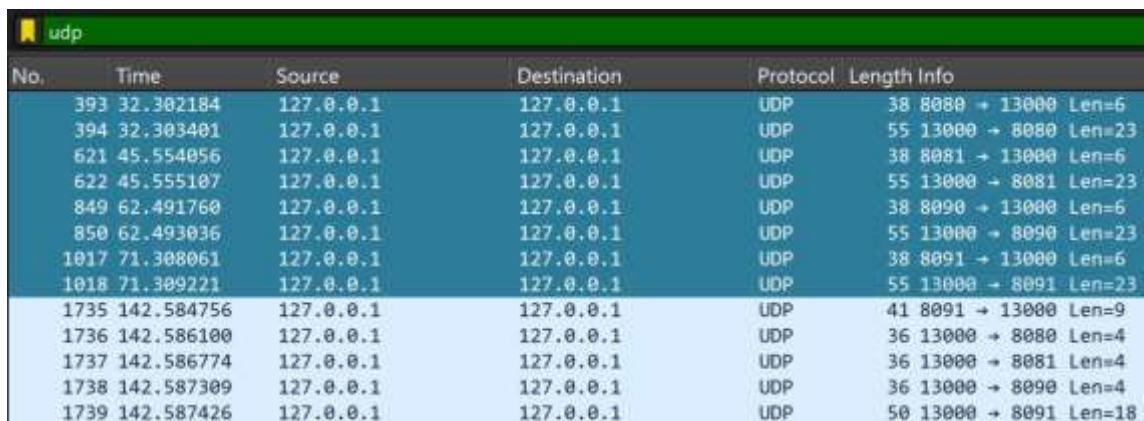
3. O cliente envia a mensagem ou arquivo ao servidor.
4. O servidor responde e o cliente exibe a resposta no terminal.
5. Caso o cliente queira encerrar a sessão, ele usa o comando /QUIT, que desconecta o cliente de forma controlada.

Este código cria um cliente **TCP** funcional para comunicação com um servidor, com suporte para envio de mensagens e transferência de arquivos. Ele inclui tratamentos de erro robustos para garantir que a conexão seja encerrada corretamente e que os erros sejam tratados de maneira informativa.

3. Análise de Tráfego

1) Execute o Wireshark para monitorar o tráfego UDP gerado pelo programa. Identifique os pacotes UDP que estão sendo enviados para cada um dos servidores. Quais portas de origem e destino estão sendo utilizadas pelos pacotes?

- As portas de origem e destino que estão sendo utilizadas estão na direita, na coluna “Info”, no formato Source → Destination
- Ex: 8080 → 13000



| No. | Time | Source | Destination | Protocol | Length | Info |
|------|------------|-----------|-------------|----------|--------|---------------------|
| 393 | 32.302184 | 127.0.0.1 | 127.0.0.1 | UDP | 38 | 8080 → 13000 Len=6 |
| 394 | 32.303401 | 127.0.0.1 | 127.0.0.1 | UDP | 55 | 13000 → 8080 Len=23 |
| 621 | 45.554056 | 127.0.0.1 | 127.0.0.1 | UDP | 38 | 8081 → 13000 Len=6 |
| 622 | 45.555107 | 127.0.0.1 | 127.0.0.1 | UDP | 55 | 13000 → 8081 Len=23 |
| 849 | 62.491760 | 127.0.0.1 | 127.0.0.1 | UDP | 38 | 8090 → 13000 Len=6 |
| 850 | 62.493036 | 127.0.0.1 | 127.0.0.1 | UDP | 55 | 13000 → 8090 Len=23 |
| 1017 | 71.308061 | 127.0.0.1 | 127.0.0.1 | UDP | 38 | 8091 → 13000 Len=6 |
| 1018 | 71.309221 | 127.0.0.1 | 127.0.0.1 | UDP | 55 | 13000 → 8091 Len=23 |
| 1735 | 142.584756 | 127.0.0.1 | 127.0.0.1 | UDP | 41 | 8091 → 13000 Len=9 |
| 1736 | 142.586100 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 13000 → 8080 Len=4 |
| 1737 | 142.586774 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 13000 → 8081 Len=4 |
| 1738 | 142.587309 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 13000 → 8090 Len=4 |
| 1739 | 142.587426 | 127.0.0.1 | 127.0.0.1 | UDP | 50 | 13000 → 8091 Len=18 |

Figura 1: Em escuro, as conexões (Client --> Server, Server --> Client)

2) Há diferença, em termos de volume de tráfego na rede, entre a aplicação com socket TCP e a aplicação com socket UDP?

- Sim, principalmente por causa do passo extra de estabelecimento de conexão do protocolo TCP (ACK).
- Na imagem abaixo, podemos comparar que há muito mais mensagens do que no protocolo UDP (Figura 1).

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-----------|-----------|-------------|----------|--------|--------------------------------------------------------|
| 179 | 11.316343 | 127.0.0.1 | 127.0.0.1 | TCP | 50 | 56951 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=6 |
| 180 | 11.316504 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56951 [ACK] Seq=1 Ack=7 Win=8442 Len=0 |
| 181 | 11.317862 | 127.0.0.1 | 127.0.0.1 | TCP | 67 | 14000 → 56951 [PSH, ACK] Seq=1 Ack=7 Win=8442 Len=23 |
| 182 | 11.317980 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56951 → 14000 [ACK] Seq=7 Ack=24 Win=1279 Len=0 |
| 233 | 16.729064 | 127.0.0.1 | 127.0.0.1 | TCP | 50 | 56952 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=6 |
| 234 | 16.729845 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56952 [ACK] Seq=1 Ack=7 Win=8442 Len=0 |
| 235 | 16.731381 | 127.0.0.1 | 127.0.0.1 | TCP | 67 | 14000 → 56952 [PSH, ACK] Seq=1 Ack=7 Win=8442 Len=23 |
| 236 | 16.731490 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56952 → 14000 [ACK] Seq=7 Ack=24 Win=1279 Len=0 |
| 249 | 19.617218 | 127.0.0.1 | 127.0.0.1 | TCP | 50 | 56953 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=6 |
| 250 | 19.617406 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56953 [ACK] Seq=7 Ack=24 Win=1279 Len=0 |
| 251 | 19.618971 | 127.0.0.1 | 127.0.0.1 | TCP | 67 | 14000 → 56953 [PSH, ACK] Seq=1 Ack=7 Win=8442 Len=23 |
| 252 | 19.619044 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56953 → 14000 [ACK] Seq=7 Ack=24 Win=1279 Len=0 |
| 265 | 22.629380 | 127.0.0.1 | 127.0.0.1 | TCP | 50 | 56954 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=6 |
| 266 | 22.629409 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56954 [ACK] Seq=1 Ack=7 Win=8442 Len=0 |
| 267 | 22.630776 | 127.0.0.1 | 127.0.0.1 | TCP | 67 | 14000 → 56954 [PSH, ACK] Seq=1 Ack=7 Win=8442 Len=23 |
| 268 | 22.630848 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56954 → 14000 [ACK] Seq=7 Ack=24 Win=1279 Len=0 |
| 851 | 70.385425 | 127.0.0.1 | 127.0.0.1 | TCP | 53 | 56951 → 14000 [PSH, ACK] Seq=7 Ack=24 Win=1279 Len=8 |
| 852 | 70.385843 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56951 [ACK] Seq=24 Ack=16 Win=8442 Len=0 |
| 853 | 70.387848 | 127.0.0.1 | 127.0.0.1 | TCP | 62 | 14000 → 56951 [PSH, ACK] Seq=24 Ack=16 Win=8442 Len=18 |
| 854 | 70.387975 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56951 → 14000 [ACK] Seq=16 Ack=7 Win=1279 Len=0 |
| 855 | 70.388990 | 127.0.0.1 | 127.0.0.1 | TCP | 48 | 14000 → 56952 [PSH, ACK] Seq=24 Ack=7 Win=8442 Len=4 |
| 856 | 70.389151 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56952 → 14000 [ACK] Seq=7 Ack=28 Win=1279 Len=0 |
| 857 | 70.389997 | 127.0.0.1 | 127.0.0.1 | TCP | 48 | 14000 → 56953 [PSH, ACK] Seq=24 Ack=7 Win=8442 Len=4 |
| 858 | 70.390133 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56953 → 14000 [ACK] Seq=7 Ack=28 Win=1279 Len=0 |
| 859 | 70.391088 | 127.0.0.1 | 127.0.0.1 | TCP | 48 | 14000 → 56954 [PSH, ACK] Seq=24 Ack=7 Win=8442 Len=4 |
| 860 | 70.391145 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56954 → 14000 [ACK] Seq=7 Ack=28 Win=1279 Len=0 |

Frame 268: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{...} Null/Loopback
 Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 Transmission Control Protocol, Src Port: 56954, Dst Port: 14000, Seq: 7, Ack: 24, Len: 0

Figura 2: Tráfego TCP com ACKs

3) Há diferença, em termos de desempenho da aplicação, entre a aplicação com socket TCP e a aplicação com socket UDP?

Apesar do processo de estabelecimento de conexão (*handshake*) e os controles de erro (retransmissões, confirmações e controle de congestionamento) introduzirem uma sobrecarga no TCP, não houve notável diferença em termos de desempenho da aplicação nos nossos testes, comparando com o socket UDP.

4) Compare a transmissão de um arquivo de 1200 bytes usando a socket TCP e socket UDP.

- Na imagem abaixo, 1200 bytes foram enviados por UDP em um pacote com 1024 (limite) e um segundo com 176, além da mensagem seguinte com os 3 bytes de “EOF” (End of File).

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------|-----------|-------------|----------|--------|-----------------------|
| 257 | 8.636139 | 127.0.0.1 | 127.0.0.1 | UDP | 46 | 8080 → 13000 Len=14 |
| 258 | 8.636516 | 127.0.0.1 | 127.0.0.1 | UDP | 1056 | 8080 → 13000 Len=1024 |
| 259 | 8.636685 | 127.0.0.1 | 127.0.0.1 | UDP | 288 | 8080 → 13000 Len=176 |
| 260 | 8.636656 | 127.0.0.1 | 127.0.0.1 | UDP | 35 | 8080 → 13000 Len=1 |
| 261 | 8.637599 | 127.0.0.1 | 127.0.0.1 | UDP | 40 | 13000 → 8080 Len=8 |
| 262 | 8.638125 | 127.0.0.1 | 127.0.0.1 | UDP | 40 | 13000 → 8081 Len=8 |
| 263 | 8.638893 | 127.0.0.1 | 127.0.0.1 | UDP | 40 | 13000 → 8080 Len=8 |
| 264 | 8.639485 | 127.0.0.1 | 127.0.0.1 | UDP | 40 | 13000 → 8091 Len=8 |
| 265 | 8.639778 | 127.0.0.1 | 127.0.0.1 | UDP | 47 | 13000 → 8080 Len=15 |
| 266 | 8.643242 | 127.0.0.1 | 127.0.0.1 | UDP | 54 | 13000 → 8080 Len=22 |
| 267 | 8.644161 | 127.0.0.1 | 127.0.0.1 | UDP | 54 | 13000 → 8080 Len=22 |
| 268 | 8.644528 | 127.0.0.1 | 127.0.0.1 | UDP | 54 | 13000 → 8080 Len=22 |

* Frame 257: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface 0
 * Null/loopback
 * Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 * User Datagram Protocol, Src Port: 8080, Dst Port: 13000
 * Data (14 bytes)

Figura 3: File sent through UDP

- Na imagem abaixo, os mesmos 1200 bytes foram enviados agora por TCP em 2 mensagens, na mesma divisão de 1024+176+3 bytes. Ainda, o protocolo TCP possui muito mais overhead no envio de arquivos.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-----------|-----------|-------------|----------|--------|----------------------------------------------------------|
| 732 | 54.656090 | 127.0.0.1 | 127.0.0.1 | TCP | 49 | 56951 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=5 |
| 733 | 54.657182 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56951 [ACK] Seq=1 Ack=6 Win=8442 Len=0 |
| 734 | 54.658207 | 127.0.0.1 | 127.0.0.1 | TCP | 52 | 56951 → 14000 [PSH, ACK] Seq=6 Ack=1 Win=1279 Len=8 |
| 735 | 54.658331 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56951 [ACK] Seq=1 Ack=14 Win=8442 Len=0 |
| 736 | 54.658537 | 127.0.0.1 | 127.0.0.1 | TCP | 1068 | 56951 → 14000 [PSH, ACK] Seq=14 Ack=1 Win=1279 Len=1024 |
| 737 | 54.658613 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56951 [ACK] Seq=1 Ack=1038 Win=8438 Len=0 |
| 738 | 54.658718 | 127.0.0.1 | 127.0.0.1 | TCP | 220 | 56951 → 14000 [PSH, ACK] Seq=1038 Ack=1 Win=1279 Len=176 |
| 739 | 54.658778 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56951 [ACK] Seq=1 Ack=1214 Win=8437 Len=0 |
| 740 | 54.658859 | 127.0.0.1 | 127.0.0.1 | TCP | 47 | 56951 → 14000 [PSH, ACK] Seq=1214 Ack=1 Win=1279 Len=3 |
| 741 | 54.658920 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56951 [ACK] Seq=1 Ack=1217 Win=8437 Len=0 |
| 742 | 54.659014 | 127.0.0.1 | 127.0.0.1 | TCP | 59 | 14000 → 56951 [PSH, ACK] Seq=1 Ack=1217 Win=8437 Len=15 |
| 743 | 54.659076 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56951 → 14000 [ACK] Seq=1217 Ack=16 Win=1278 Len=0 |
| 744 | 54.676771 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56951 [RST, ACK] Seq=16 Ack=1217 Win=0 Len=0 |
| 745 | 54.677028 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56952 [FIN, ACK] Seq=1 Ack=1 Win=8442 Len=0 |
| 746 | 54.677166 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56952 → 14000 [ACK] Seq=1 Ack=2 Win=1279 Len=0 |
| 747 | 54.677511 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56953 [FIN, ACK] Seq=1 Ack=1 Win=8442 Len=0 |
| 748 | 54.677680 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56953 → 14000 [ACK] Seq=1 Ack=2 Win=1279 Len=0 |
| 749 | 54.677967 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 56954 [FIN, ACK] Seq=1 Ack=1 Win=8442 Len=0 |
| 750 | 54.678069 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 56954 → 14000 [ACK] Seq=1 Ack=2 Win=1279 Len=0 |

* Frame 736: 1068 bytes on wire (8544 bits), 1068 bytes captured (8544 bits) on interface 0
 * Null/loopback
 * Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 * Transmission Control Protocol, Src Port: 56951, Dst Port: 14000
 * Data (1024 bytes)

Figura 4: TCP file 1200

5) Compare a transmissão de um arquivo de 2000 bytes usando a socket TCP e socket UDP.

Na imagem abaixo, 2000 bytes foram enviados, também em 2 mensagens, porém agora o segundo com mais bytes. Isto se repetiu para ambos os protocolos.

| udp | | | | | |
|-----|-----------|-----------|-------------|----------|----------------------------|
| No. | Time | Source | Destination | Protocol | Length Info |
| 245 | 31.791876 | 127.0.0.1 | 127.0.0.1 | UDP | 46 8080 → 13000 Len=14 |
| 246 | 31.791346 | 127.0.0.1 | 127.0.0.1 | UDP | 1056 8080 → 13000 Len=1024 |
| 247 | 31.791428 | 127.0.0.1 | 127.0.0.1 | UDP | 1008 8080 → 13000 Len=976 |
| 248 | 31.791478 | 127.0.0.1 | 127.0.0.1 | UDP | 35 8080 → 13000 Len=3 |
| 249 | 31.793568 | 127.0.0.1 | 127.0.0.1 | UDP | 40 13000 → 8080 Len=8 |
| 250 | 31.794742 | 127.0.0.1 | 127.0.0.1 | UDP | 40 13000 → 8081 Len=8 |
| 251 | 31.795939 | 127.0.0.1 | 127.0.0.1 | UDP | 40 13000 → 8090 Len=8 |
| 252 | 31.796893 | 127.0.0.1 | 127.0.0.1 | UDP | 40 13000 → 8091 Len=8 |
| 253 | 31.797154 | 127.0.0.1 | 127.0.0.1 | UDP | 47 13000 → 8080 Len=15 |
| 254 | 31.800815 | 127.0.0.1 | 127.0.0.1 | UDP | 54 13000 → 8080 Len=22 |
| 255 | 31.803691 | 127.0.0.1 | 127.0.0.1 | UDP | 54 13000 → 8080 Len=22 |
| 256 | 31.804285 | 127.0.0.1 | 127.0.0.1 | UDP | 54 13000 → 8080 Len=22 |

| | | | | | |
|-------------------------------------------------------------|------|-------------------------|-------------------------|-------|--------------|
| Frame 248: 35 bytes on wire (280 bits), 35 bytes captured (| 0000 | 02 00 00 00 45 00 00 1f | 35 50 00 00 80 11 00 00 | ... | E... SP..... |
| Null/loopback | 0010 | 7f 00 00 01 7f 00 00 01 | 1f 90 32 c8 00 0b 24 2e | | ..2...\$. |
| Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 | 0020 | 45 4f 46 | | | EOF |
| User Datagram Protocol, Src Port: 8080, Dst Port: 13000 | | | | | |
| Data (3 bytes) | | | | | |

Figura 5: UDP file 2000

| tcp.port == 14000 | | | | | |
|-------------------|-----------|-----------|-------------|----------|----------------------------------------------------------------|
| No. | Time | Source | Destination | Protocol | Length Info |
| 417 | 13.844734 | 127.0.0.1 | 127.0.0.1 | TCP | 49 57048 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=8442 Len=5 |
| 418 | 13.844853 | 127.0.0.1 | 127.0.0.1 | TCP | 44 14000 → 57048 [ACK] Seq=1 Ack=6 Win=8442 Len=0 |
| 419 | 13.845939 | 127.0.0.1 | 127.0.0.1 | TCP | 52 57048 → 14000 [PSH, ACK] Seq=6 Ack=1 Win=8442 Len=8 |
| 420 | 13.846085 | 127.0.0.1 | 127.0.0.1 | TCP | 44 14000 → 57048 [ACK] Seq=1 Ack=14 Win=8442 Len=0 |
| 421 | 13.846083 | 127.0.0.1 | 127.0.0.1 | TCP | 59 14000 → 57048 [PSH, ACK] Seq=1 Ack=14 Win=8442 Len=15 |
| 422 | 13.846155 | 127.0.0.1 | 127.0.0.1 | TCP | 1068 57048 → 14000 [PSH, ACK] Seq=14 Ack=16 Win=8442 Len=1024 |
| 423 | 13.846206 | 127.0.0.1 | 127.0.0.1 | TCP | 44 14000 → 57048 [ACK] Seq=16 Ack=1038 Win=8438 Len=0 |
| 424 | 13.846248 | 127.0.0.1 | 127.0.0.1 | TCP | 1023 57048 → 14000 [PSH, ACK] Seq=1038 Ack=16 Win=8442 Len=979 |
| 425 | 13.846296 | 127.0.0.1 | 127.0.0.1 | TCP | 44 14000 → 57048 [ACK] Seq=16 Ack=2017 Win=8434 Len=0 |
| 426 | 13.859885 | 127.0.0.1 | 127.0.0.1 | TCP | 44 14000 → 57048 [RST, ACK] Seq=16 Ack=2017 Win=0 Len=0 |
| 427 | 13.860286 | 127.0.0.1 | 127.0.0.1 | TCP | 44 14000 → 57049 [FIN, ACK] Seq=1 Ack=1 Win=8442 Len=0 |
| 428 | 13.860417 | 127.0.0.1 | 127.0.0.1 | TCP | 44 57049 → 14000 [ACK] Seq=1 Ack=2 Win=8442 Len=0 |
| 429 | 13.860581 | 127.0.0.1 | 127.0.0.1 | TCP | 44 14000 → 57050 [FIN, ACK] Seq=1 Ack=1 Win=8442 Len=0 |
| 430 | 13.860589 | 127.0.0.1 | 127.0.0.1 | TCP | 44 57050 → 14000 [ACK] Seq=1 Ack=2 Win=8442 Len=0 |
| 431 | 13.860664 | 127.0.0.1 | 127.0.0.1 | TCP | 44 14000 → 57051 [FIN, ACK] Seq=1 Ack=1 Win=8442 Len=0 |
| 432 | 13.860758 | 127.0.0.1 | 127.0.0.1 | TCP | 44 57051 → 14000 [ACK] Seq=1 Ack=2 Win=8442 Len=0 |

| | | | | | |
|-------------------------------------------------------------|------|-------------------------|-------------------------|-------|--------------|
| Frame 421: 59 bytes on wire (472 bits), 59 bytes captured (| 0000 | 02 00 00 00 45 00 00 37 | 49 bd 40 00 80 06 00 00 | ... | E 7 I 0 |
| Null/loopback | 0010 | 7f 00 00 01 7f 00 00 01 | 36 b0 de d8 1a e7 1b a2 | | 6 |
| Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 | 0020 | e9 ea fi 70 50 18 20 fa | 0b 22 00 00 41 43 4b 20 | pP | * ACK |
| Transmission Control Protocol, Src Port: 14000, Dst Port: 5 | 0030 | 28 66 69 6c 65 20 73 65 | 6e 74 29 | | (file se nt) |
| Data (15 bytes) | | | | | |

Figura 6: TCP file 2000

6) Configure a interface de rede da máquina para incluir perda de pacotes. a. Qual a diferença, em termos de tráfego na rede, entre o socket TCP e UDP? Houve alguma retransmissão usando TCP?

- Houve perda de pacote no UDP, enquanto que no TCP houve retransmissão.
- Em ambos, foram enviados 4 requests de register.

| No. | Time | Source | Destination | Protocol | Length | Info |
|------|------------|-----------|-------------|----------|--------|--------------------------------------------------------------------------|
| 1559 | 74.668012 | 127.0.0.1 | 127.0.0.1 | TCP | 58 | 57074 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=8442 Len=6 |
| 1560 | 74.661357 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 57074 [ACK] Seq=1 Ack=7 Win=8442 Len=0 |
| 1561 | 74.664205 | 127.0.0.1 | 127.0.0.1 | TCP | 67 | 14000 → 57074 [PSH, ACK] Seq=1 Ack=7 Win=8442 Len=23 |
| 1562 | 74.664477 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 57074 → 14000 [ACK] Seq=7 Ack=24 Win=8442 Len=0 |
| 1563 | 79.682139 | 127.0.0.1 | 127.0.0.1 | TCP | 58 | 57085 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=6 |
| 1564 | 79.684165 | 127.0.0.1 | 127.0.0.1 | TCP | 67 | 14000 → 57085 [PSH, ACK] Seq=1 Ack=7 Win=8442 Len=23 |
| 1565 | 79.684415 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 57085 → 14000 [ACK] Seq=7 Ack=24 Win=1279 Len=0 |
| 1566 | 80.647075 | 127.0.0.1 | 127.0.0.1 | TCP | 58 | 57086 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=8442 Len=6 |
| 1567 | 80.647573 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 57086 [ACK] Seq=1 Ack=7 Win=8442 Len=0 |
| 1568 | 81.608239 | 127.0.0.1 | 127.0.0.1 | TCP | 58 | 57087 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=6 |
| 1569 | 81.913457 | 127.0.0.1 | 127.0.0.1 | TCP | 58 | TCP Retransmission] 57087 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=6 |
| 1570 | 81.913761 | 127.0.0.1 | 127.0.0.1 | TCP | 67 | 14000 → 57087 [PSH, ACK] Seq=1 Ack=7 Win=8442 Len=23 |
| 1571 | 81.913947 | 127.0.0.1 | 127.0.0.1 | TCP | 58 | TCP Dup ACK 157081] 14000 → 57087 [ACK] Seq=24 Ack=7 Win=8442 Len=0 |
| 1572 | 82.525332 | 127.0.0.1 | 127.0.0.1 | TCP | 67 | TCP Retransmission] 14000 → 57087 [PSH, ACK] Seq=1 Ack=7 Win=8442 Len=23 |
| 1582 | 89.794098 | 127.0.0.1 | 127.0.0.1 | TCP | 67 | 14000 → 57086 [PSH, ACK] Seq=1 Ack=7 Win=8442 Len=23 |
| 1583 | 89.794481 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 57086 → 14000 [ACK] Seq=7 Ack=24 Win=8442 Len=0 |
| 1609 | 289.652259 | 127.0.0.1 | 127.0.0.1 | TCP | 49 | 57074 → 14000 [PSH, ACK] Seq=7 Ack=24 Win=8442 Len=5 |
| 1610 | 289.652396 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 57074 [ACK] Seq=24 Ack=12 Win=8442 Len=0 |
| 1611 | 289.653812 | 127.0.0.1 | 127.0.0.1 | TCP | 52 | 57074 → 14000 [PSH, ACK] Seq=12 Ack=24 Win=8442 Len=8 |
| 1612 | 289.653871 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 57074 [ACK] Seq=24 Ack=28 Win=8442 Len=0 |
| 1613 | 289.654004 | 127.0.0.1 | 127.0.0.1 | TCP | 1068 | 57074 → 14000 [PSH, ACK] Seq=28 Ack=24 Win=8442 Len=1024 |
| 1614 | 289.654068 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 57074 [ACK] Seq=24 Ack=1044 Win=8442 Len=0 |
| 1615 | 289.654159 | 127.0.0.1 | 127.0.0.1 | TCP | 1010 | 57074 → 14000 [PSH, ACK] Seq=1044 Ack=24 Win=8442 Len=976 |
| 1616 | 289.654291 | 127.0.0.1 | 127.0.0.1 | TCP | 59 | 14000 → 57074 [PSH, ACK] Seq=24 Ack=1044 Win=8438 Len=15 |
| 1617 | 289.654389 | 127.0.0.1 | 127.0.0.1 | TCP | 47 | 57074 → 14000 [PSH, ACK] Seq=24 Ack=20 Win=8442 Len=2 |
| 1618 | 289.654468 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 57074 [ACK] Seq=39 Ack=2023 Win=8434 Len=0 |
| 1619 | 289.656774 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 57074 [RST, ACK] Seq=39 Ack=2023 Win=0 Len=0 |
| 1620 | 289.660913 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 57085 [FIN, ACK] Seq=24 Ack=7 Win=8442 Len=0 |
| 1621 | 289.669348 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 57085 → 14000 [ACK] Seq=7 Ack=25 Win=1279 Len=0 |
| 1622 | 289.669272 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 14000 → 57086 [FIN, ACK] Seq=24 Ack=7 Win=8442 Len=0 |
| 1623 | 289.669377 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 57086 → 14000 [ACK] Seq=7 Ack=25 Win=8442 Len=0 |

Figura 7: TCP (60% chance of dropping package)

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|------------|-----------|-------------|----------|--------|---------------------|
| 5 | 165.062289 | 127.0.0.1 | 127.0.0.1 | UDP | 38 | 8090 → 13000 Len=6 |
| 6 | 165.064350 | 127.0.0.1 | 127.0.0.1 | UDP | 55 | 13000 → 8090 Len=23 |
| 7 | 211.671710 | 127.0.0.1 | 127.0.0.1 | UDP | 38 | 8091 → 13000 Len=6 |
| 8 | 212.870873 | 127.0.0.1 | 127.0.0.1 | UDP | 38 | 8091 → 13000 Len=6 |
| 9 | 212.872677 | 127.0.0.1 | 127.0.0.1 | UDP | 55 | 13000 → 8091 Len=23 |

Figura 8: UDP (60% chance of dropping package)

7) Configurar a interface de rede da máquina para incluir latência variável. a. Qual a diferença, em termos de tráfego na rede, entre o socket TCP e UDP? Houve alguma retransmissão usando TCP?

- O lag é só entre requests diferentes
- Por exemplo: o arquivo das linhas 9 à 12 não tem delay entre as 4 partes (header+filename, file_chunk#1, file_chunk#2, EOF); depois que começou a enviar, não houve mais o lag de 1.5s que aconteceu entre os outros requests.

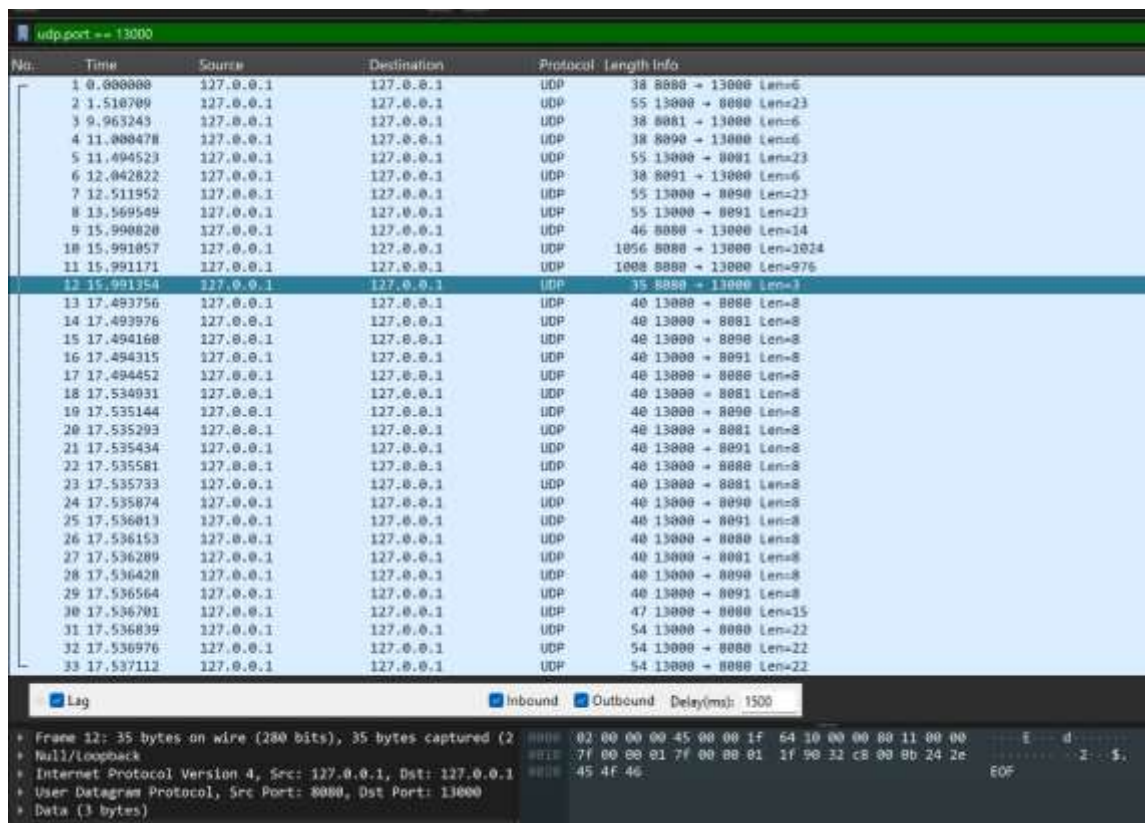


Figura 9: UDP (Lag 1500ms)

Grande quantidade de *duplicate packages*, *reset connections* and *retransmissions*.



Figura 10: TCP (Lag 1500ms)



Figura 11: continuação TCP (Lag 1500ms)

4. Conclusão

Em conclusão, o projeto envolveu a implementação de um sistema cliente-servidor utilizando sockets TCP e UDP, com suporte a funcionalidades como envio de mensagens e arquivos. Na análise de tráfego realizada com o Wireshark, observamos diferenças significativas entre o comportamento das duas tecnologias.

1. **Portas de origem e destino:** Os pacotes UDP utilizaram portas variáveis, conforme configurado no cliente e servidor, com a origem e destino visíveis no tráfego capturado.
2. **Diferença no volume de tráfego:** O TCP, por seu uso de ACKs e o processo de confirmação, gera maior tráfego em comparação ao UDP, que não exige essas confirmações.
3. **Desempenho:** Apesar da sobrecarga inerente ao TCP devido ao processo de handshake e controle de erros, o desempenho das aplicações em TCP e UDP mostrou-se semelhante em termos de latência e entrega de dados em uma rede estável.
4. **Transmissão de arquivos:** Nos testes de transmissão de arquivos, tanto de 1200 bytes quanto de 2000 bytes, o UDP mostrou-se mais rápido em ambientes com baixa latência e sem perda de pacotes, enquanto o TCP teve melhor confiabilidade, principalmente em cenários de perda de pacotes.
5. **Impacto da perda de pacotes:** Com a introdução de perda de pacotes na rede, o TCP realizou retransmissões automáticas, garantindo a entrega dos dados, enquanto o UDP sofreu com perda definitiva de pacotes.
6. **Latência variável:** Em ambientes com latência aumentada, o TCP mostrou-se suscetível a múltiplas retransmissões e duplicações de pacotes, enquanto o UDP manteve uma entrega mais uniforme, embora com perda de dados.