Universidade Federal
de São João del-Rei

REDES DE COMPUTADORES

RELATÓRIO DO TRABALHO PRÁTICO - PROTOCOLOS UDP E TCP

João Antônio Lucas Costa Lucas Emanuel

Novembro de 2024

1 Introdução

Este trabalho tem como foco a implementação de uma aplicação de rede utilizando os protocolos TCP (Transmission Control Protocol) e UDP (User Datagram Protocol), que se destacam entre os diversos protocolos de comunicação, oferecendo características e aplicações distintas em ambientes de rede. O TCP é um protocolo orientado à conexão, que garante a entrega ordenada e confiável dos dados, enquanto o UDP é um protocolo não orientado à conexão, que permite uma comunicação mais rápida, porém sem garantir a entrega ou a ordem dos pacotes.

O objetivo deste trabalho é explorar as diferenças entre esses dois protocolos ao desenvolver uma aplicação de rede que envolva o envio de arquivos, utilizando tanto o TCP quanto o UDP. A aplicação será composta por um cliente e um servidores — um para cada protocolo — que permitirão medir a taxa de download e a perda de pacotes no caso do UDP. O cliente, ao se conectar ao servidor, será responsável por realizar as medições da taxa de transferência de dados para ambos os protocolos, além de detectar a perda de pacotes durante a comunicação via UDP.

Este estudo proporcionará uma compreensão prática das vantagens e limitações de cada protocolo em termos de desempenho, confiabilidade e aplicação em diferentes cenários de rede.

2 Diferenças entre TCP e UDP

1. **Confiabilidade**

   - **TCP:** Estabelece uma conexão confiável entre o cliente e o servidor antes de transmitir os dados, garantindo que todos os dados sejam entregues na ordem correta. Possui retransmissão de pacotes perdidos, o que faz com que a entrega seja confiável e ideal para aplicações que precisam da integridade das informações, como transferências de arquivos, e-mails, HTTP, FTP e SMTP.
   - **UDP:** Não estabelece uma conexão e não garante a entrega de pacotes, podendo haver perda, duplicação ou recebimento fora de ordem. Não há controle de fluxo ou confirmação de entrega, o que deixa ele mais suscetível a erros, mas adequado para aplicações que podem ter perdas, como streaming de vídeo, chamadas VoIP, e jogos online.

2. **Desempenho**

   - **TCP:** Apresenta maior overhead devido à estrutura mais complexa do cabeçalho e ao gerenciamento de conexão, controle de fluxo e retransmissão de pacotes, o que resulta em um desempenho mais lento. Isso deixa o protocolo mais confiável, mas também mais exigente em termos de recursos e tempo.

- **UDP:** Tem overhead mínimo, já que não implementa controle de fluxo ou congestionamento e não garante a entrega de pacotes. Isso permite um desempenho mais rápido e eficiente, sendo ideal para aplicações que exigem baixa latência e alta velocidade, independentemente das condições da rede.

3 Implementação dos Sockets

A implementação dos sockets, tanto para TCP quanto para UDP, foi realizada utilizando as bibliotecas `arpa/inet.h` e `unistd.h`. Essas bibliotecas fornecem métodos para manipulação de endereços IP e conversões de formato na rede, bem como funções como `read`, `write` e `close`, usadas para entrada e saída de dados nos sockets.

A criação do socket é feita com a função `socket(int domain, int type, int protocol)`, onde:

- `domain`: especifica a família de endereços (ex.: `AF_INET` para IPv4).
- `type`: define o tipo do socket, como `SOCK_STREAM` (TCP) ou `SOCK_DGRAM` (UDP).
- `protocol`: identifica o protocolo específico (geralmente `0` para usar o padrão do tipo escolhido).

Após criar o socket, usa-se a função `bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)` para associá-lo a um endereço e porta. Em seguida, os métodos `recvfrom` e `sendto` são utilizados para, respectivamente, receber e enviar dados por meio do socket. A estrutura `sockaddr_in` foi empregada para armazenar os endereços IP e portas necessários à comunicação.

Para o TCP, antes de enviar ou receber dados, é necessário estabelecer uma conexão entre cliente e servidor. Para isso: o servidor utiliza as funções listen (para aguardar conexões) e accept (para aceitar uma conexão de um cliente). O cliente utiliza a função connect para se conectar ao servidor. As funções send e recv são utilizadas diretamente para envio e recepção de dados em uma conexão já estabelecida. Os dados são enviados de forma sequencial e garantida, eliminando a necessidade de implementar verificações adicionais de ordem ou perda de pacotes, comuns no UDP. Após a transferência dos dados, a conexão é encerrada explicitamente com o fechamento do socket usando close. No UDP, o socket é simplesmente fechado sem a preocupação com o encerramento de conexão, pois não há estado de conexão.

4 Resultados

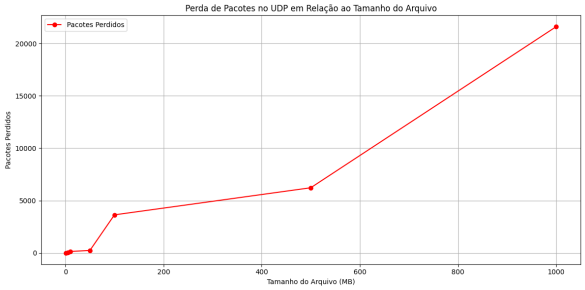

Figura 1: Gráfico de perda de pacotes no UDP em relação ao tamanho do arquivo.

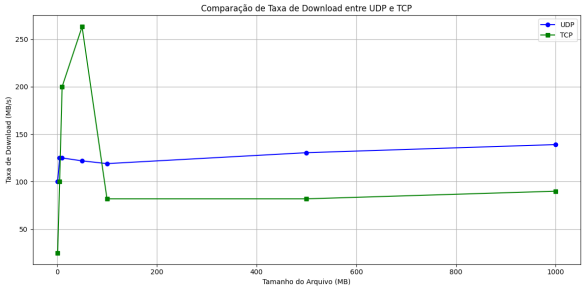

Figura 2: Comparação da taxa de download entre os protocolos UDP e TCP.

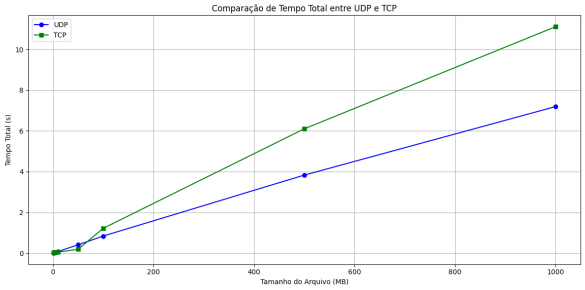

Figura 3: Comparação do tempo total de transferência entre os protocolos UDP e TCP.

Os resultados obtidos confirmam as teorias previamente discutidas. A perda de pacotes no UDP era um comportamento esperado, devido à ausência de mecanismos internos de confiabilidade no protocolo. Observou-se que, para tamanhos menores de arquivo, a taxa de download variou de forma não linear, reflexo tanto das características inerentes aos protocolos quanto de fatores externos, como o desempenho da máquina e as condições da rede durante os testes. Os valores registrados são baseados em uma média de algumas execuções, o que ajuda a minimizar variações extremas. Além disso, o UDP apresentou maior rapidez em comparação ao TCP, como era esperado, devido à sua

eliminação de etapas como confirmação de recebimento e controle de congestionamento, tornando-o mais eficiente em termos de velocidade, ainda que menos confiável.

5 Conclusão

A análise comparativa entre os protocolos TCP e UDP evidencia as diferenças fundamentais em desempenho, confiabilidade e consumo de recursos, confirmando seus papéis distintos em redes. O TCP, com seu controle rigoroso de transmissão e mecanismos de confiabilidade, é ideal para aplicações que exigem integridade e entrega ordenada de dados, como transferências de arquivos e comunicações críticas. No entanto, essa confiabilidade vem com a desvantagem de maior exigência de recursos e tempo, devido ao overhead introduzido pelo estabelecimento de conexão, controle de fluxo e retransmissões, o que pode impactar o desempenho em cenários de alta demanda ou redes com restrições de capacidade. Por outro lado, o UDP, com sua leveza e baixa latência, é mais eficiente em aplicações como streaming e jogos online, onde pequenas perdas são aceitáveis. Contudo, as perdas de pacotes em redes baseadas em UDP podem comprometer significativamente o desempenho, especialmente em redes instáveis. Os resultados apresentados nos gráficos de taxas de download e perdas confirmam essas características, demonstrando a consistência do TCP e a variabilidade do UDP, reforçando que a escolha entre os protocolos deve considerar as demandas específicas do ambiente e da aplicação.