Análise Comparativa de Desempenho dos Protocolos TCP e UDP Utilizando Sockets em Python

Alysson Nogueira R. Alves, João Pedro G. Viana de Souza, Victor Gopfert F. Figueiredo

Universidade do Estado do Rio de Janeiro (UERJ)

Redes de Computadores I – 2025.2

Resumo. *Este artigo apresenta uma análise experimental e comparativa dos protocolos da camada de transporte, TCP e UDP. O objetivo é demonstrar empiricamente as diferenças de desempenho e sobrecarga entre a comunicação orientada à conexão do TCP e a abordagem sem conexão do UDP. Para isso, foi desenvolvida uma aplicação cliente-servidor de eco em Python, utilizando Sockets. O tráfego de rede foi capturado e analisado com a ferramenta Wireshark, focando em três métricas: quantidade de mensagens, volume de dados e tempo da transação. Os resultados quantitativos obtidos confirmam a teoria: o TCP, com seu mecanismo de confiabilidade, apresenta uma sobrecarga significativamente maior, enquanto o UDP se mostra mais leve e direto.*

1. Introdução

A comunicação em redes de computadores é um processo complexo, organizado em camadas de abstração, conforme definido por modelos como o OSI e o TCP/IP. Dentre estas, a camada de transporte ocupa uma posição central, sendo responsável pela comunicação lógica entre processos de aplicações em diferentes hospedeiros. Os dois protocolos mais proeminentes desta camada são o Transmission Control Protocol (TCP) e o User Datagram Protocol (UDP).

Enquanto o TCP oferece um serviço confiável, ordenado e orientado à conexão, o UDP provê um serviço simples, rápido e sem conexão, sem garantias de entrega. A escolha entre um e outro é uma decisão fundamental no desenvolvimento de aplicações de rede e depende inteiramente dos requisitos da aplicação.

Este artigo visa formalizar os conceitos da pilha de protocolos, com destaque para a camada de transporte, e materializar as diferenças teóricas entre TCP e UDP através de um experimento prático. Utilizando a linguagem Python e sua biblioteca de Sockets, foi implementado um serviço de rede simples, testado com ambos os protocolos. O tráfego resultante foi capturado e analisado para quantificar as diferenças de desempenho, oferecendo uma visão clara das implicações práticas de cada escolha.

2. Camada de Transporte: TCP e UDP

A camada de transporte é o elo entre a camada de aplicação e as camadas inferiores da rede. Ela provê a segmentação dos dados e o controle necessário para a comunicação entre processos.

O TCP é um protocolo orientado à conexão, o que significa que antes de qualquer troca de dados, uma conexão estável deve ser estabelecida entre cliente e servidor. Este processo é conhecido como three-way handshake (aperto de mão de três vias). Suas principais características são a confiabilidade, o controle de fluxo e o controle de congestionamento. Essa confiabilidade tem um custo em termos de overhead, como será demonstrado.

O UDP é um protocolo muito mais simples e sem conexão. Ele não estabelece um canal de comunicação prévio; apenas encapsula os dados em datagramas e os envia. Por não possuir mecanismos de controle, handshake ou confirmações, o UDP é extremamente leve e rápido, sendo ideal para aplicações onde a velocidade é mais crítica que a confiabilidade, como streaming de vídeo, jogos online e serviços de DNS.

3. Sockets em Python

Um Socket é um ponto de extremidade (endpoint) em uma comunicação de rede. A API de Sockets provê um conjunto de funções que permite que uma aplicação envie e receba dados através da rede. Em Python, a comunicação via sockets é gerenciada pela biblioteca padrão socket. A implementação difere fundamentalmente dependendo do protocolo de transporte escolhido, utilizando socket.SOCK_STREAM para TCP e socket.SOCK_DGRAM para UDP.

A Figura 1 ilustra o fluxo geral de chamadas de sistema para estabelecer uma comunicação via socket, que é a base do cenário proposto.

Figura 1. Fluxograma do processo de comunicação via Sockets.

4. Cenário Proposto

Para a análise prática, foi desenvolvido um cenário de comunicação cliente-servidor. Foi criada uma aplicação de "eco", onde um cliente envia uma mensagem de texto e o servidor a devolve intacta. O código foi desenvolvido em Python[1], com versões para TCP e UDP. A comunicação foi realizada localmente (localhost, IP 127.0.0.1), eliminando variáveis de latência e perda de pacotes da rede externa.

[1] O código-fonte da aplicação está disponível no repositório GitHub: https://github.com/JoaoVianaSouza/Client_Server_Redes

Para a coleta de dados, foi utilizado um sniffer de rede. Um sniffer, ou analisador de pacotes, é uma ferramenta que intercepta e exibe o tráfego que passa por uma interface de rede, permitindo uma análise detalhada de cada pacote. No experiment realizado, a ferramenta utilizada foi o Wireshark, que nos permitiu registrar todas as transações, contar os pacotes e medir seus tamanhos e tempos de envio/recebimento. Para cada protocolo, o teste foi executado 10 vezes para garantir a consistência dos dados e permitir o cálculo de uma média confiável.

As métricas analisadas foram: a quantidade de mensagens trocadas (total de pacotes), o volume de dados (soma do tamanho em bytes de todos os pacotes) e o tempo (decorrido entre o primeiro e o último pacote da transação).

5. Resultados e análise

A análise dos dados capturados em 10 testes para cada protocolo revelou diferenças quantitativas claras e consistentes. A tabela a seguir apresenta a média dos dados brutos capturados via protocolo UDP e TCP

Resumo (Médias)			
Protocolo	**Média de Mensagens**	**Média de Volume (Bytes)**	**Média de Tempo (s)**
UDP	2	92	0,0001733
TCP	11	538	0,0010505

Tabela 1. Resumo dos dados médios coletados em 10 testes.

A partir desses dados, foi levantado um gráfico comparativo de barras (Gráfico 1) para melhor visualização desses resultados, comparando as respostas dos do protocolo UTP com o protocolo TCP.

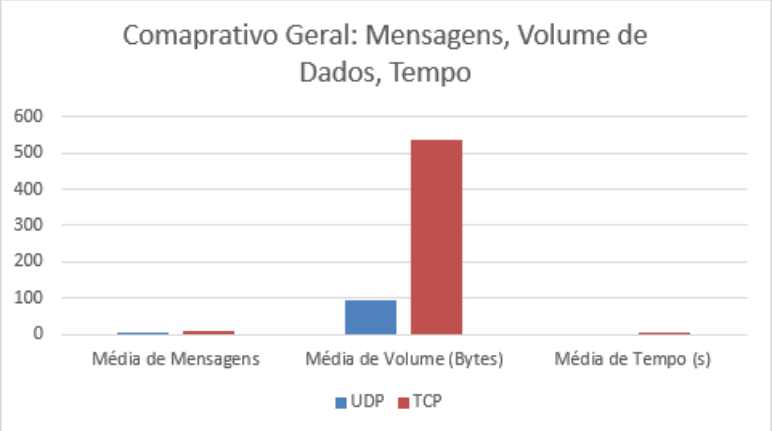

Gráfico 1. Comparativo dos dados médios coletados em 10 testes.

A comunicação UDP foi extremamente previsível, consistindo em exatamente 2 pacotes por transação (Figura 2), refletindo sua natureza direta.

Ethernet | IPv4 · 1 | IPv6 | TCP | UDP · 11

Address	Port	Packets	Bytes	Total Packets	Percent Filtered	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
127.0.0.1	53320	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	53321	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	53322	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	53742	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	54082	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	56776	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	57559	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	62197	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	64785	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	65153	2	92 bytes	2	100.00%	1	46 bytes	1	46 bytes
127.0.0.1	65431	20	920 bytes	20	100.00%	10	460 bytes	10	460 bytes

Figura 2. Detalhe dos pacotes de uma transação UDP (Fonte: Wireshark).

O tráfego TCP, em contraste, demonstrou uma complexidade muito maior, com cada transação consistindo em 11 pacotes (Figura 3). Esse *overhead* é explicado pela divisão da comunicação em três fases distintas e obrigatórias, que garantem a confiabilidade da entrega:

- **Estabelecimento da Conexão (Handshake):** Primeiramente, três pacotes (SYN, SYN/ACK, ACK) são trocados para estabelecer um canal de comunicação confiável antes que qualquer dado da aplicação seja enviado.

- **Troca de Dados:** Com a conexão estabelecida, os dados são enviados em segmentos, e cada segmento recebido é confirmado com um pacote de ACK. Isso garante que nenhuma informação seja perdida ou chegue fora de ordem.

- **Encerramento da Conexão:** Ao final, quatro pacotes (FIN e ACK de ambos os lados) são utilizados para encerrar a conexão de forma organizada, assegurando que toda a comunicação foi finalizada.

Essa metodologia robusta justifica o maior número de pacotes e o maior tempo de transação do TCP, que são o "preço" a ser pago pela sua confiabilidade.

Address	Port	Packets	Bytes	Total Packets	Percent Filtered	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
127.0.0.1	53639	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	53642	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	53671	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	53675	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	53678	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	53681	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	53684	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	53688	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	53689	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	53691	11	538 bytes	11	100.00%	6	291 bytes	5	247 bytes
127.0.0.1	65432	110	5 kB	110	100.00%	50	2 kB	60	3 kB

Figura 3. Detalhe dos 11 pacotes de uma única transação TCP (Fonte: Wireshark).

Para visualizar essas diferenças, foram gerados dois gráficos de barras. O Gráfico 2 compara o *overhead* de cada protocolo em termos de mensagens e volume de dados.

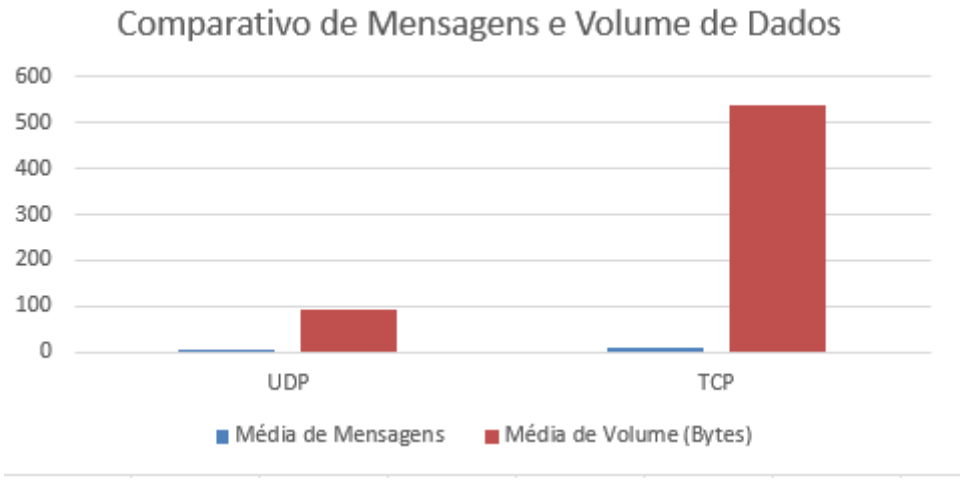

Gráfico 2. Gráfico comparativo de Mensagens e Volume de Dados.

O gráfico ilustra a disparidade de recursos. A comunicação TCP exigiu 5.5 vezes mais pacotes e um volume de dados 5.8 vezes maior que a comunicação UDP para realizar a mesma tarefa de eco. Isso evidencia o "custo" da confiabilidade do TCP.

O Gráfico 3, por sua vez, isola a métrica de tempo para permitir uma análise de desempenho com a escala correta.

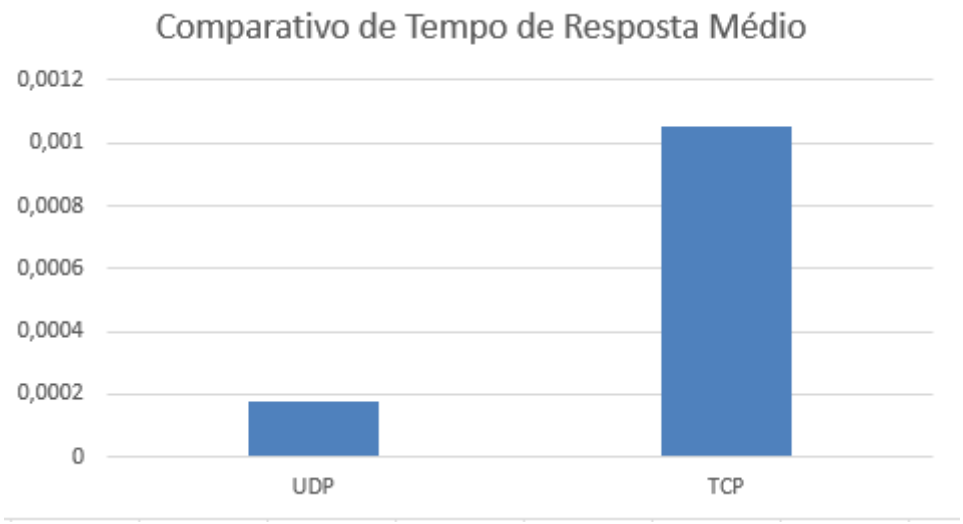

Gráfico 3. Gráfico comparativo de Tempo de Resposta Médio.

O segundo gráfico demonstra que, mesmo em uma rede local de altíssima velocidade, o processo mais complexo do TCP resultou em um tempo de transação médio aproximadamente 6 vezes maior que o do UDP.

6. Conclusão

Este trabalho realizou uma comparação prática e quantitativa entre os protocolos TCP e UDP. Através de um cenário controlado utilizando sockets em Python e análise de tráfego com Wireshark, foi possível validar empiricamente os conceitos teóricos da camada de transporte.

Os resultados demonstraram de forma inequívoca o trade-off fundamental entre confiabilidade e desempenho. O UDP se provou extremamente eficiente, com uma sobrecarga mínima. Em contrapartida, o TCP exigiu um volume de pacotes e dados muito superior para realizar a mesma tarefa, um reflexo direto de seus mecanismos de controle. Conclui-se que a escolha do protocolo de transporte tem um impacto direto e mensurável no desempenho de uma aplicação de rede.

Referencias

Avast (2023). "TCP vs. UDP: qual é a diferença?". Disponível em: https://www.avast.com/pt-br/c-tcp-vs-udp-difference. Acesso em: 19 de set. de 2025.

Controle.net (s.d.). "Sniffer - O que é um analisador de protocolos". Disponível em: https://www.controle.net/faq/sniffer-o-que-e-um-analisador-de-protocolos.Acesso em: 19 de set. de 2025.

SOUZA, J. P. V. et al. (2025). *Código-fonte para análise comparativa de TCP e UDP*. Repositório de Software. Disponível em: https://github.com/JoaoVianaSouza/Client_Server_Redes. Acesso em: 19 de set. de 2025.

Kurose, J. F. e Ross, K. W. (2013). Redes de Computadores e a Internet: Uma Abordagem Top-Down. 6. ed. São Paulo: Pearson Education.

Tanenbaum, A. S. e Wetherall, D. J. (2011). Redes de Computadores. 5. ed. São Paulo: Pearson Education.

Python Software Foundation. socket — Low-level networking interface. Disponível em: https://docs.python.org/3/library/socket.html. Acesso em: 18 de set. de 2025.