

Chat_TCP: Relatório do Trabalho Prático de Sockets em Redes de Computadores I (CSI301)

Luccas Vinicius P. A. Santos Carneiro¹

¹Universidade Federal de Ouro Preto (UFOP)

Curso de Sistemas de Informação — Redes de Computadores I (CSI301)

Matrícula: 20.1.8015

luccas.carneiro@aluno.ufop.edu.br

Abstract. This report describes the development of Chat_TCP, a cross-platform TCP chat system. The server handles multiple connections, announces itself via UDP, and supports both broadcast and private messages. It also keeps a live list of users and provides commands like /lista, /status and /desconectar. Clients for Windows (WinForms) and Linux/macOS (Avalonia) connect easily, opening private chats automatically and showing online users in real time. As an extension, a Java Spring Boot REST API was added for remote control, together with a React webapp so the chat can also run directly in the browser, including on mobile. The system was tested with Wireshark to validate network behavior.

Resumo. Este relatório descreve o desenvolvimento do Chat_TCP, um sistema de chat TCP multiplataforma. O servidor gerencia várias conexões, se anuncia via UDP e permite mensagens em broadcast e privadas. Ele também mantém a lista de usuários em tempo real e comandos como /lista, /status e /desconectar. Os clientes em Windows (WinForms) e Linux/macOS (Avalonia) se conectam de forma simples, abrindo janelas privadas automaticamente e mostrando quem está online. Como extensão, foi criada uma API REST em Java Spring Boot para controle remoto e um webapp em React que permite usar o chat direto no navegador, inclusive no celular. O funcionamento foi validado com análises no Wireshark.

1. Visão Geral

O projeto conseguiu cumprir bem a proposta: criar um chat TCP robusto e que funciona em diferentes plataformas. O servidor é responsável por gerenciar as conexões, divulgar seu endereço via UDP e tratar tanto mensagens em grupo quanto privadas. Os clientes (WinForms e Avalonia) funcionam bem nesse cenário, abrindo janelas de conversa quando necessário e mostrando em tempo real quem está online.

2. Arquitetura

O sistema foi dividido em algumas partes principais:

- **Servidor TCP/UDP:** aceita várias conexões, faz broadcast de presença e gerencia usuários.
- **Clientes:** WinForms (Windows) e Avalonia (Linux/macOS).
- **API REST (Spring Boot):** para monitorar e administrar remotamente.
- **Webapp React:** permite acessar o chat pelo navegador, inclusive no celular.

3. Funcionamento

O funcionamento básico segue este fluxo:

1. O servidor se anuncia via UDP.
2. Os clientes descobrem e se conectam via TCP.
3. Após conectados, podem enviar mensagens em broadcast ou privadas.
4. O servidor mantém e atualiza a lista de usuários ativos.
5. Comandos administrativos permitem consultar ou derrubar conexões.

4. Diagrama do Projeto

A Figura 1 mostra o desenho geral do sistema.

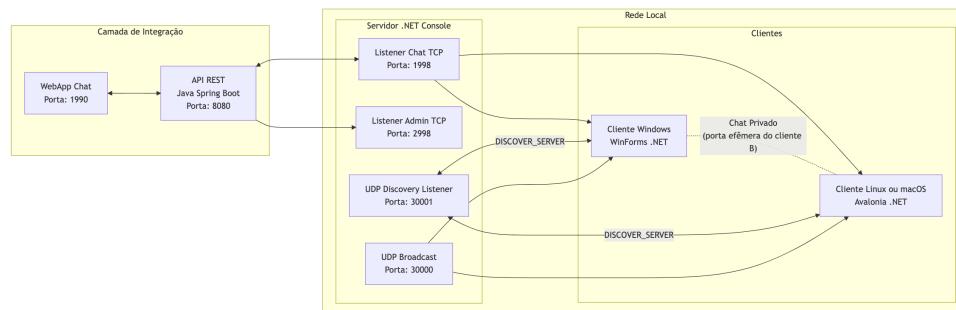


Figura 1. Arquitetura do Chat.TCP.

5. Testes

O sistema foi testado como um todo em três sistemas operacionais distintos: Windows, Linux e macOS. Os objetivos principais foram:

- Verificar a conectividade TCP e a troca de mensagens (broadcast e privadas).
- Confirmar a descoberta de servidor via UDP.
- Validar o comportamento do servidor sob múltiplas conexões simultâneas.
- Analisar o tráfego capturado via Wireshark.

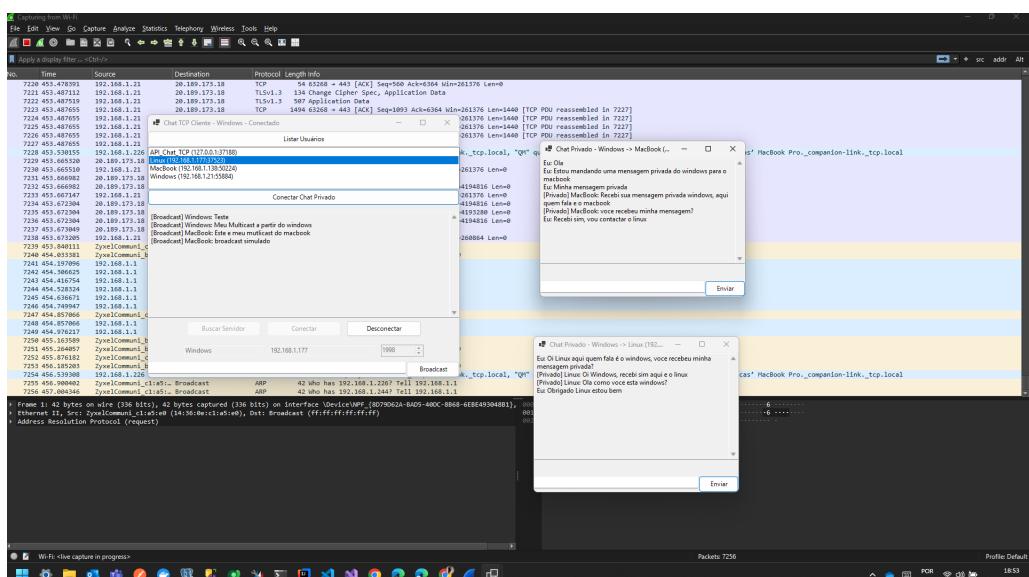


Figura 2. Exemplo do ClienteChat no Windows

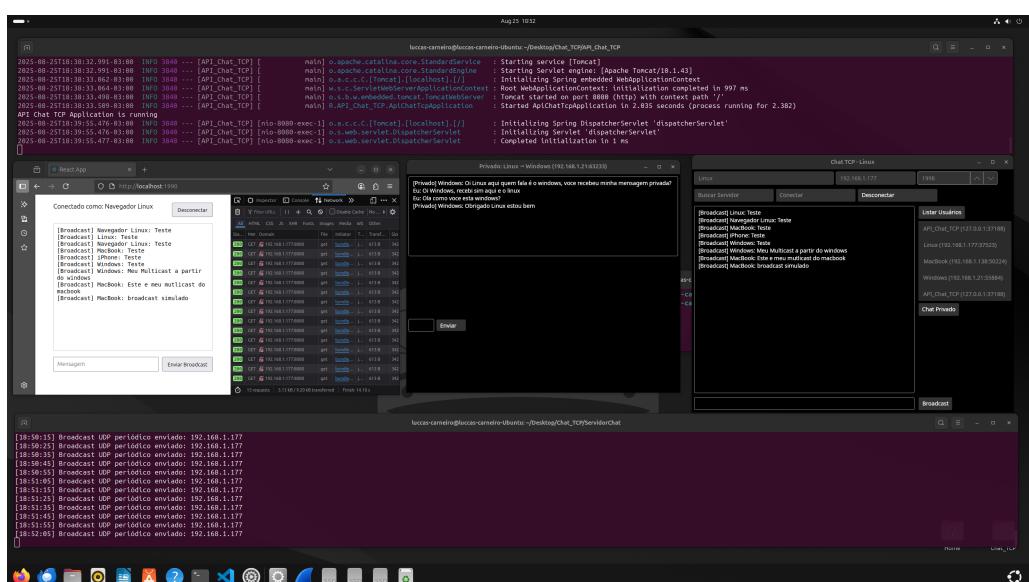


Figura 3. Exemplo do Servidor, API, Web React e ClienteChatLinux no Linux.

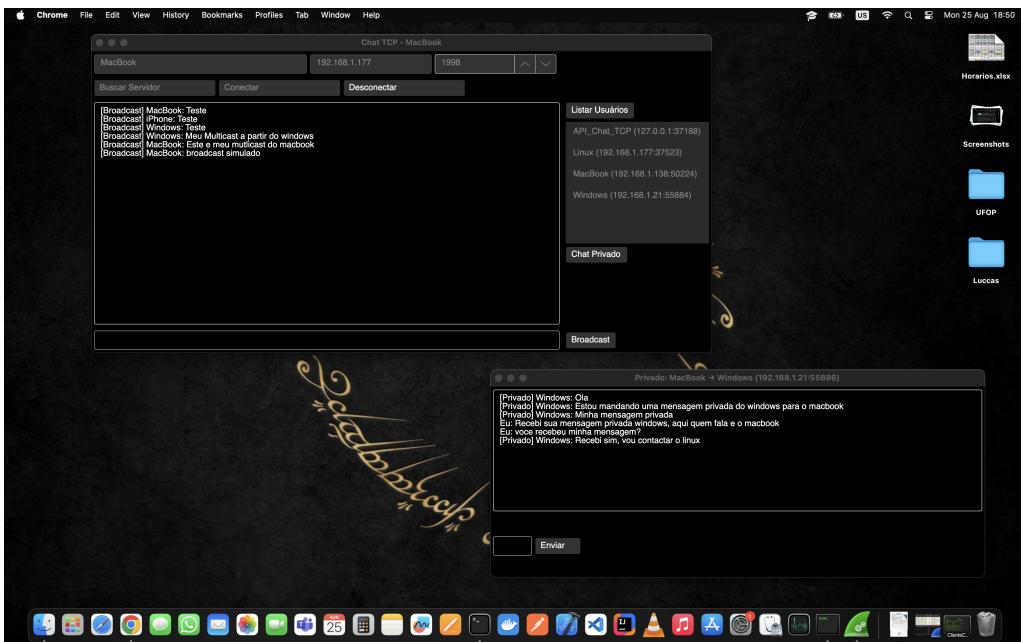


Figura 4. Exemplo do ClienteChatLinux no macOS.

5.1. Execução dos Testes

O fluxo de testes seguiu os seguintes passos:

1. Inicialização do servidor, que anuncia sua presença via broadcast UDP.
2. Conexão de múltiplos clientes em diferentes plataformas.
3. Envio e recepção de mensagens em broadcast e privadas.
4. Execução de comandos administrativos como /lista e /status.
5. Captura do tráfego de rede para validação no Wireshark.

5.2. Resultados

Os testes confirmaram que:

- O handshake TCP (SYN, SYN-ACK, ACK) foi corretamente estabelecido.
- O broadcast UDP do servidor foi recebido e interpretado pelos clientes.
- As mensagens privadas foram entregues corretamente entre pares.
- O servidor manteve a lista de usuários consistente durante as sessões.

Diversos exemplos de execução estão documentados nas Figuras 2 a 21 e nas capturas de Wireshark apresentadas. Todos os **prints** e **traces** completos estão disponíveis para download no repositório do projeto.

6. Análises no Wireshark

O Wireshark foi usado para confirmar que tudo estava funcionando. Foi possível ver:

- O handshake TCP (SYN, SYN-ACK, ACK) ao conectar.
- As mensagens UDP de broadcast do servidor.
- O tráfego de mensagens entre clientes.

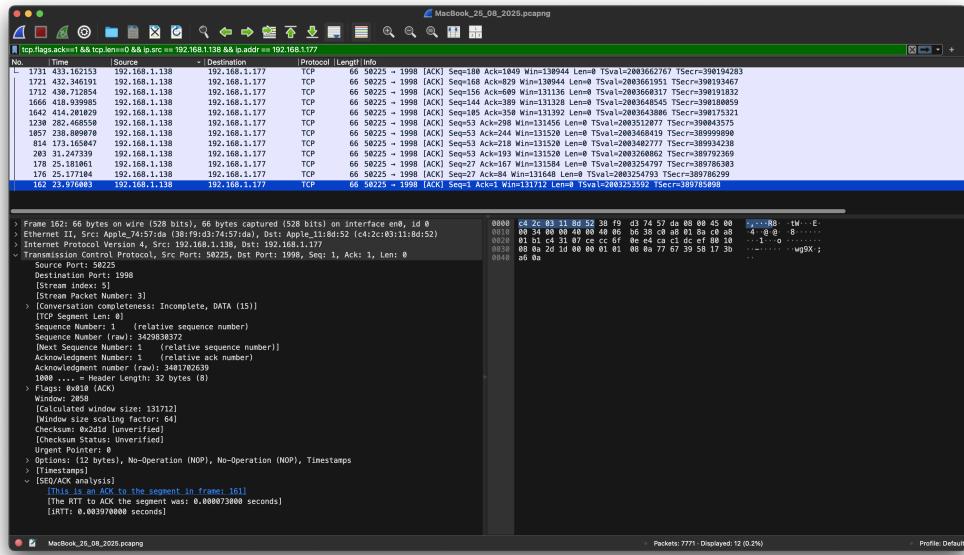


Figura 5. Exemplo de handshake TCP capturado no Wireshark.

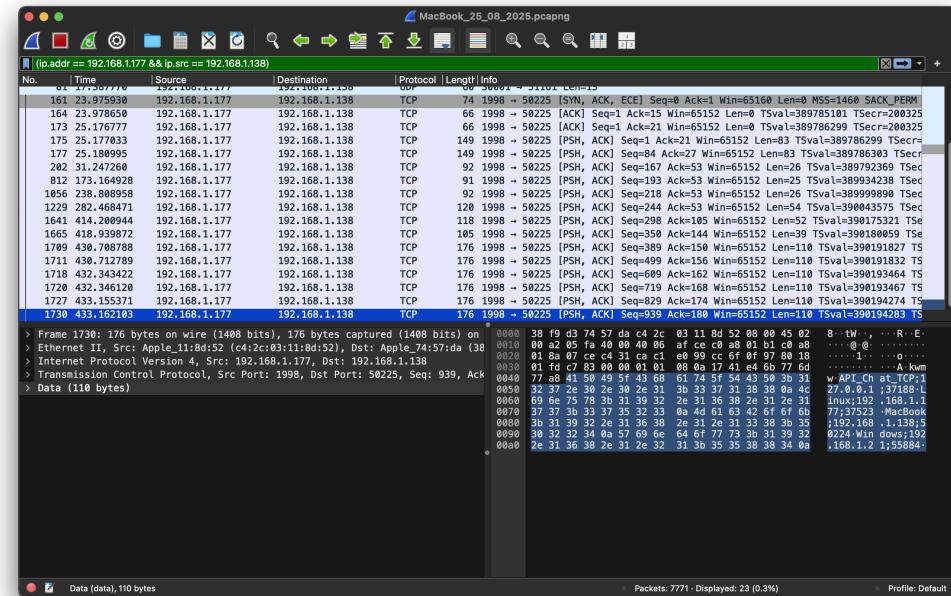


Figura 6. Executando comando de Listar Usuários

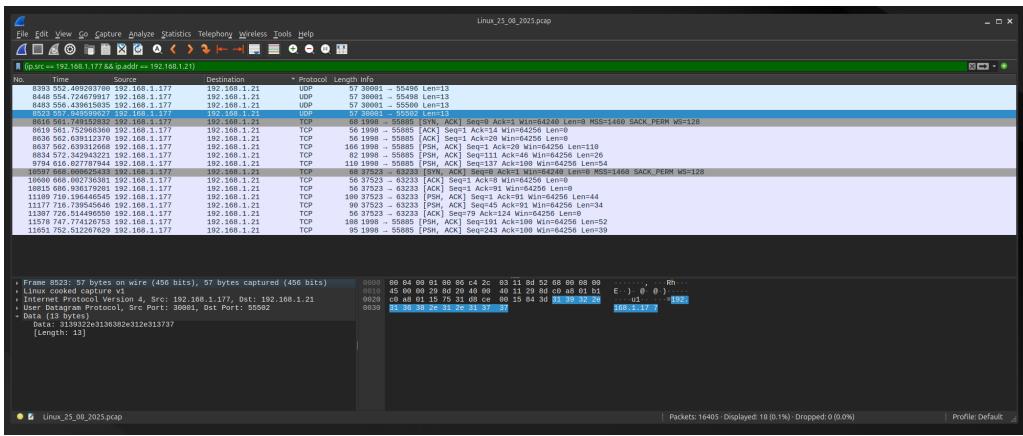


Figura 7. Respondendo busca por servidor

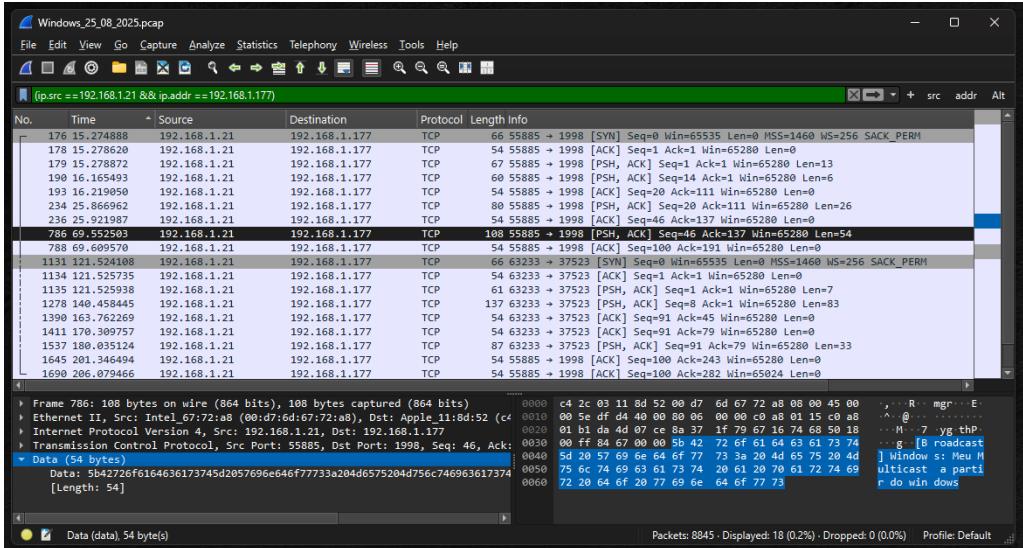


Figura 8. Broadcast simulado

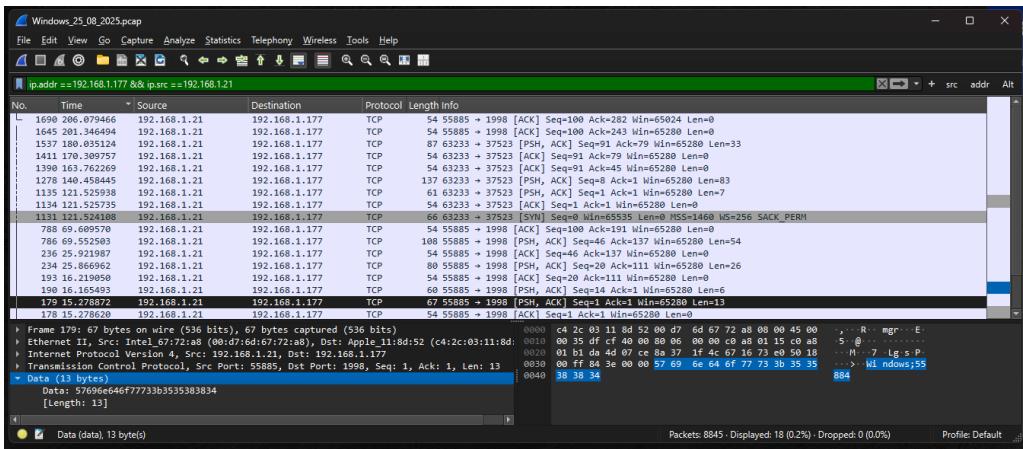


Figura 9. Windows se conecta ao servidor enviando apelido e porta privada.

6.1. Chat Privado

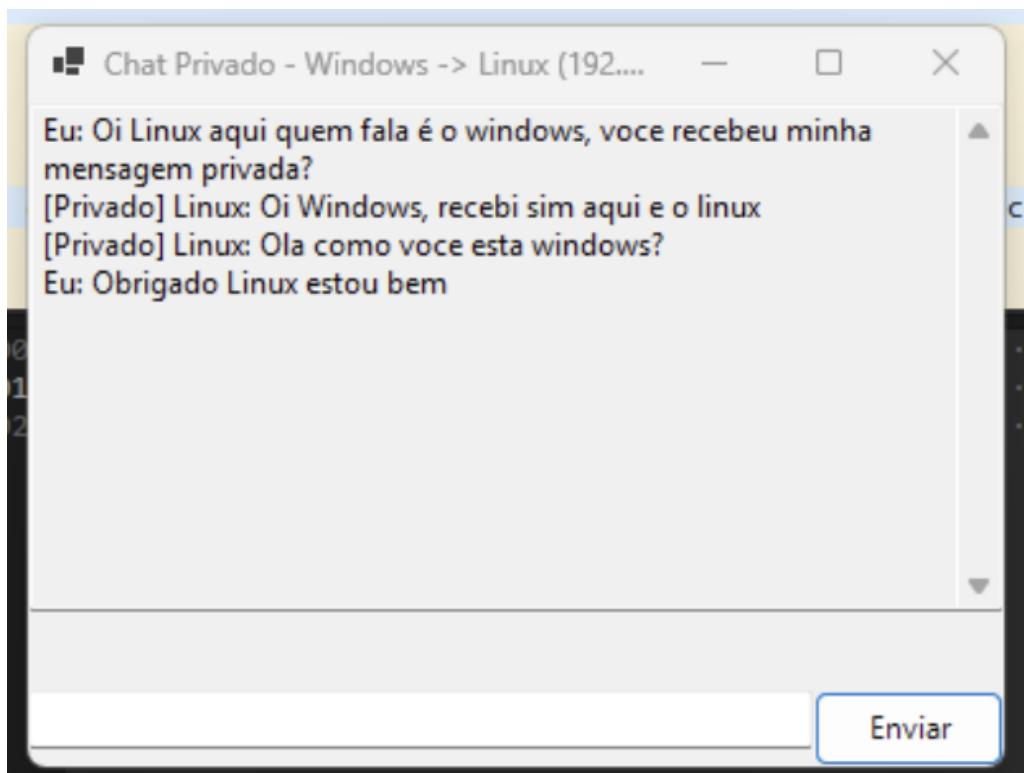


Figura 10. Chat Privado Windows - Linux

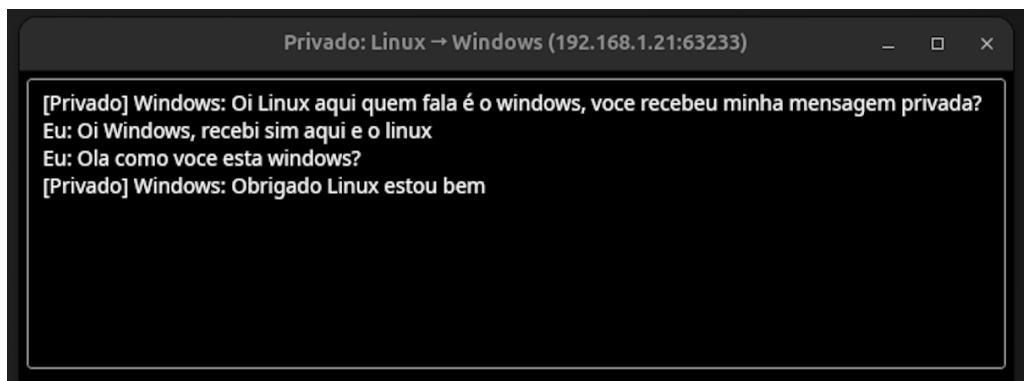


Figura 11. Chat Privado Linux - Windows

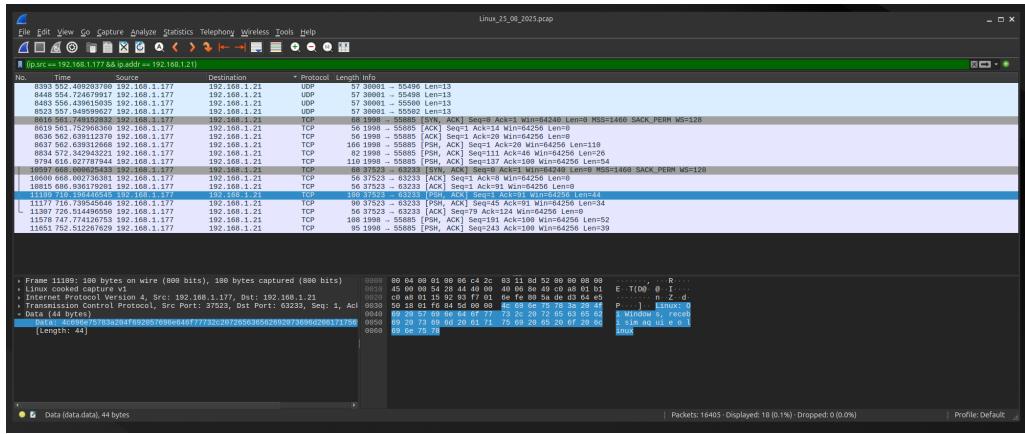


Figura 12. Chat Privado Linux - Windows - Wireshark

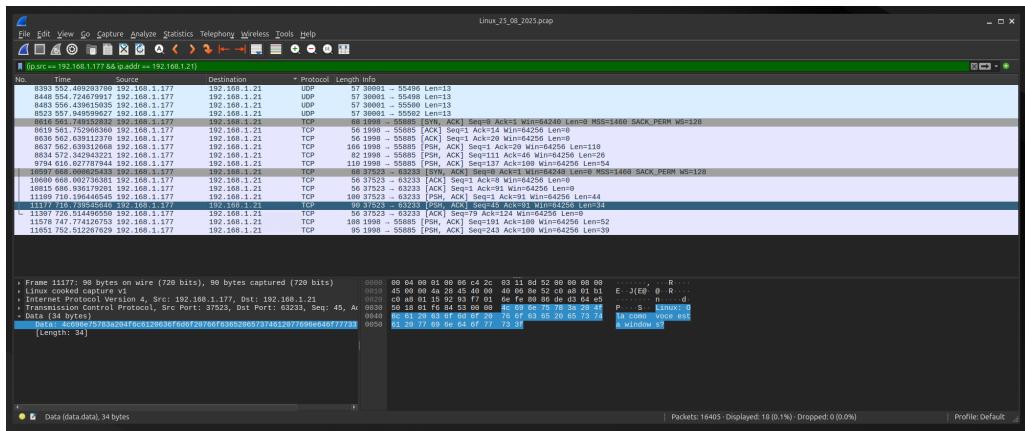


Figura 13. Chat Privado Linux - Windows - Wireshark

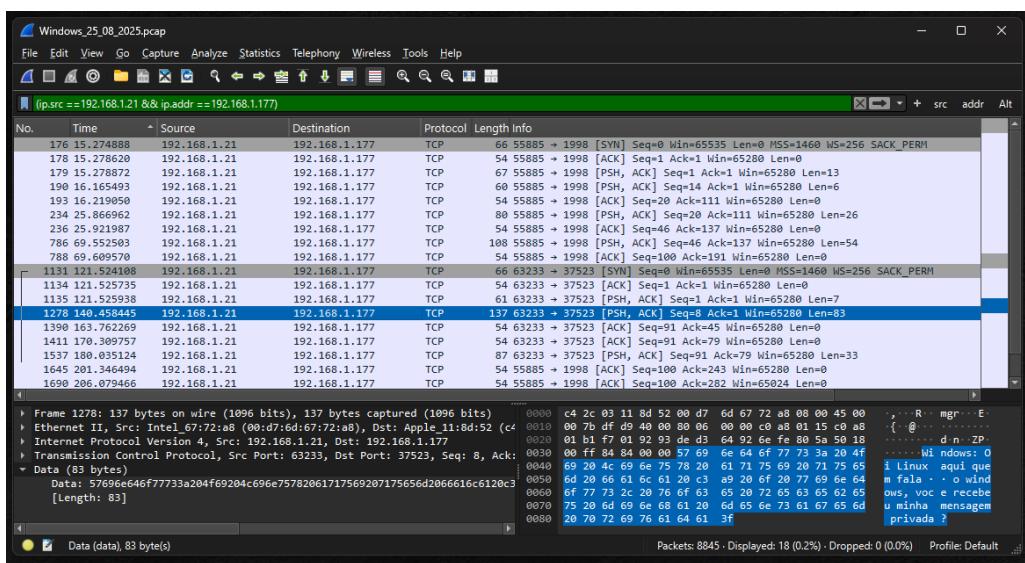


Figura 14. Chat Privado Windows - Linux - Wireshark

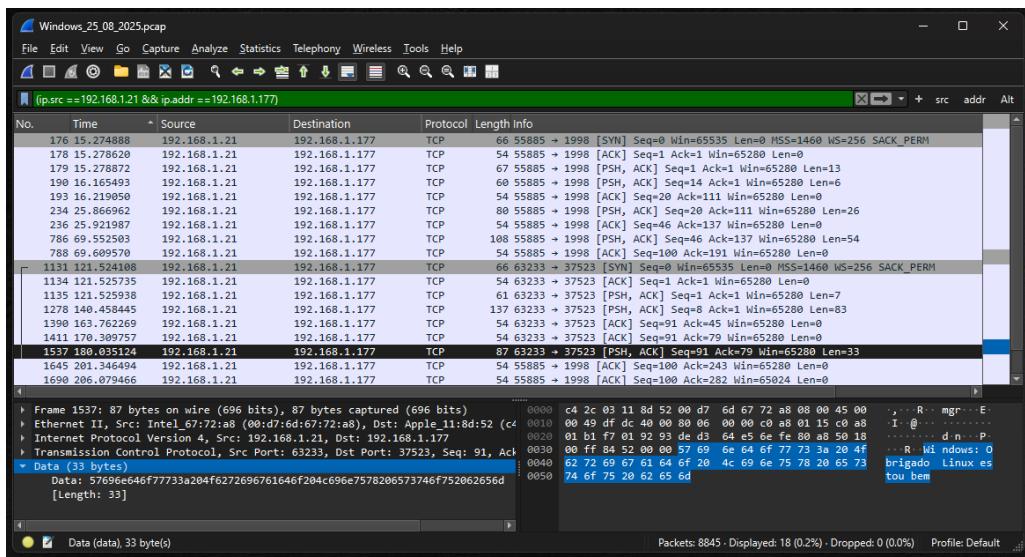


Figura 15. Chat Privado Windows - Linux - Wireshark

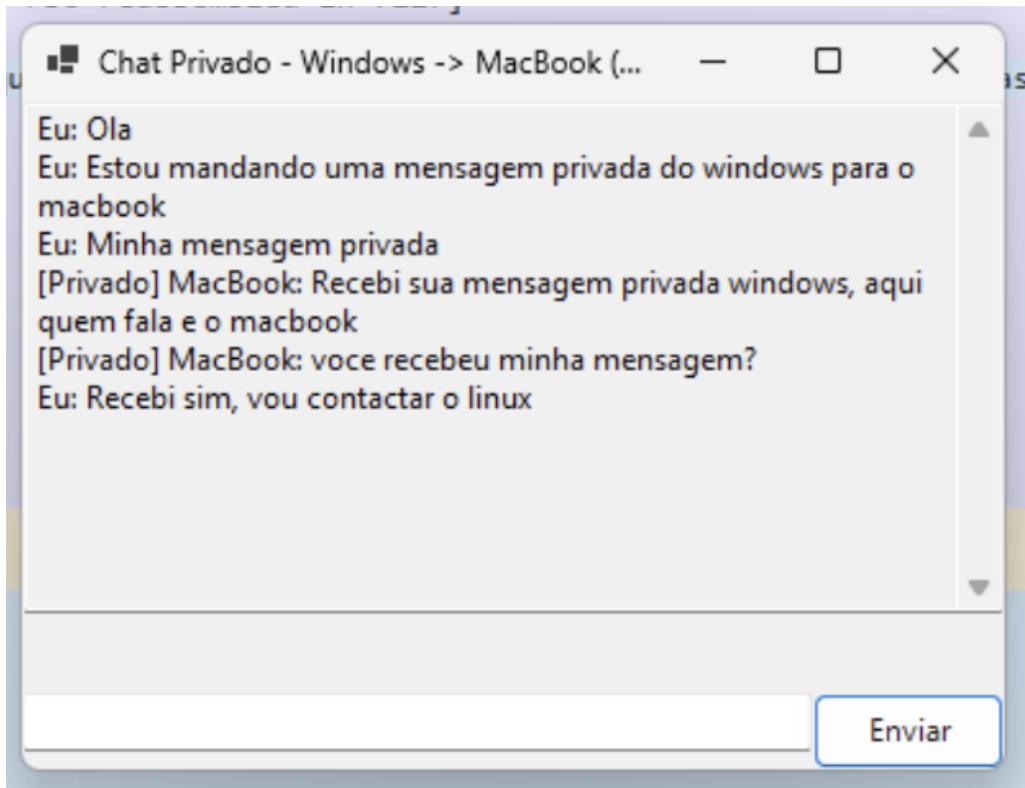


Figura 16. Chat Privado Windows - MacOS

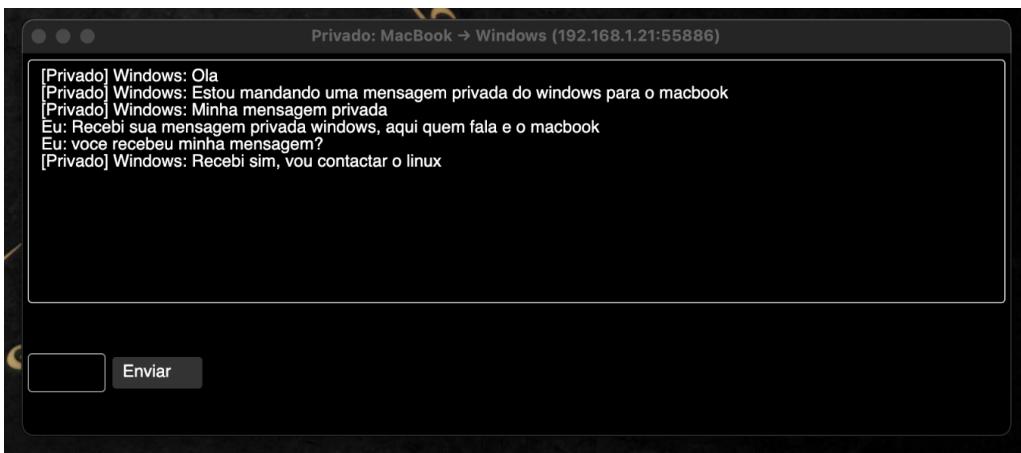


Figura 17. Chat Privado MacOS - Windows

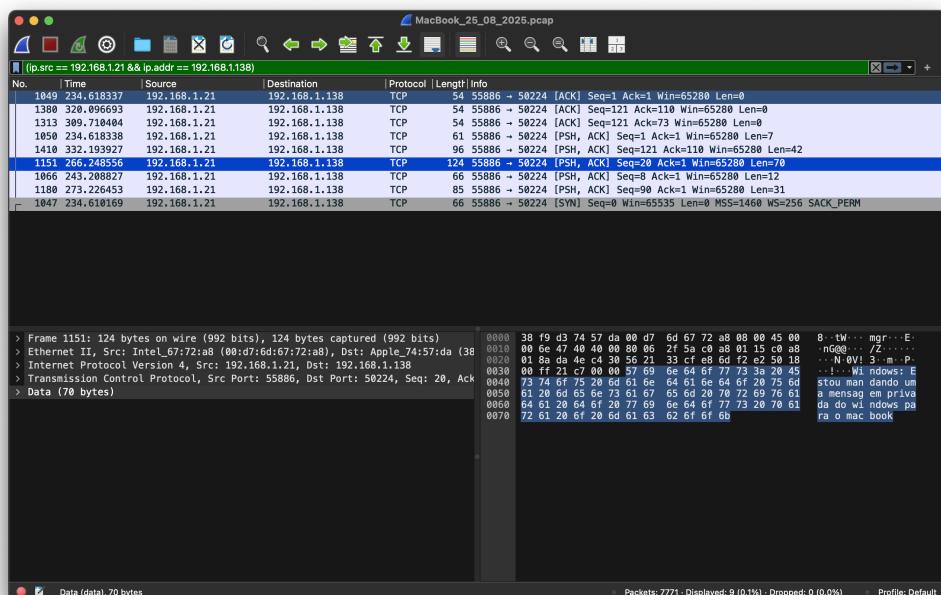


Figura 18. Chat Privado MacOS - Windows - Wireshark

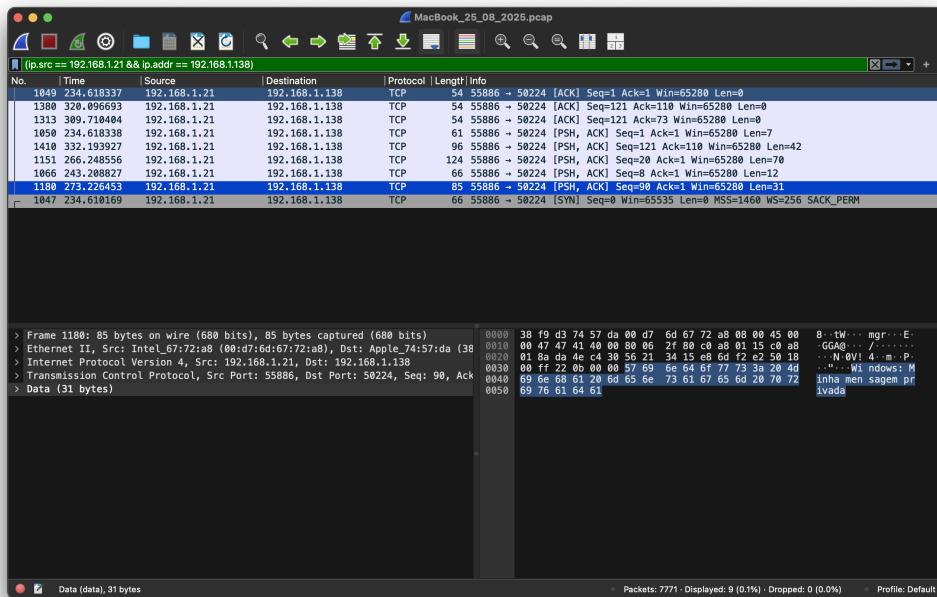


Figura 19. Chat Privado MacOS - Windows - Wireshark

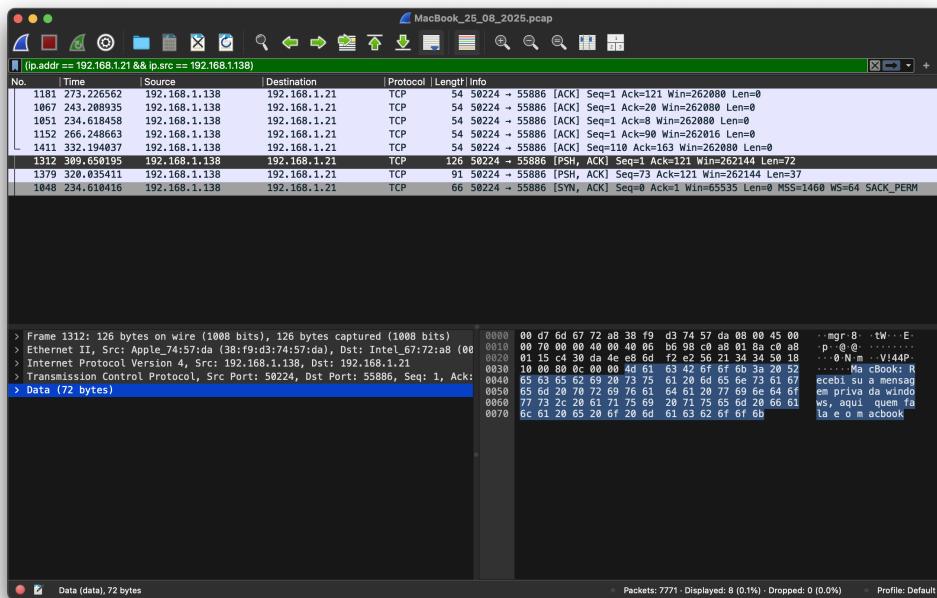


Figura 20. Chat Privado MacOS - Windows - Wireshark

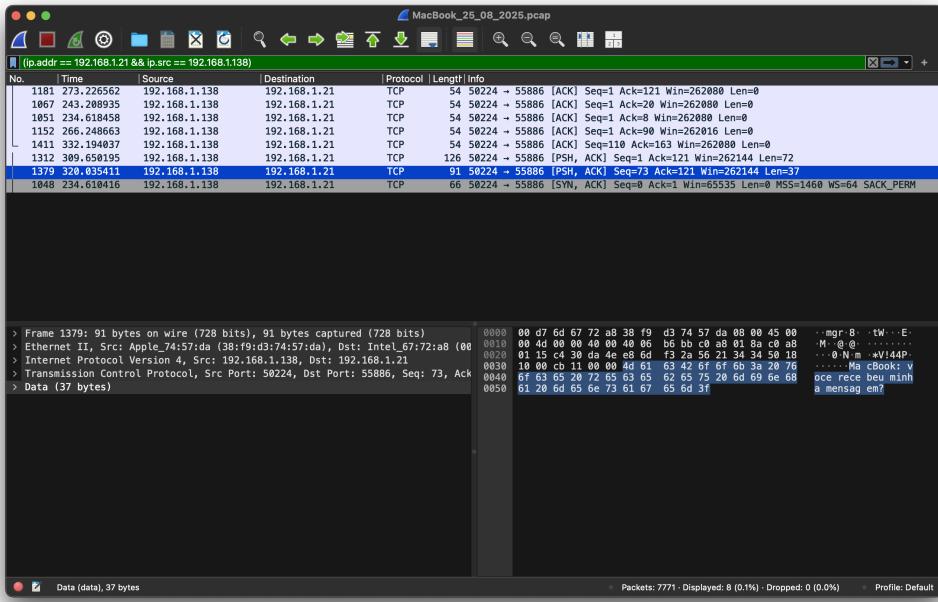


Figura 21. Chat Privado - MacOS- Windows - Wireshark

prints e traces estao disponiveis para download no repositorio

7. Conclusão

Foi um trabalho bem interessante de fazer. No começo tive dificuldade com alguns pontos de sockets e configuração de rede, mas depois que entendi melhor, o desenvolvimento fluiu. O mais legal foi ver diferentes linguagens, frameworks e sistemas conversando entre si. Além de funcionar bem, o projeto ainda ganhou a extensão com API e webapp, deixando a solução bem completa. Essa experiência de integrar várias tecnologias deve ser muito útil nos próximos estudos.

Referências

- [1] Notas de aula da disciplina CSI301 – Redes de Computadores I, Universidade Federal de Ouro Preto (2025).
- [2] TANENBAUM, A. S.; WETHERALL, D. *Redes de Computadores*. 5. ed. Pearson, 2011.
- [3] Oracle. *Class Socket (Java Platform SE 8)*. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html>. Acesso em: 15 ago. 2025.
- [4] Microsoft. *System.Net.Sockets.Socket Class*. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/api/system.net.sockets.socket?view=net-8.0>. Acesso em: 15 ago. 2025.
- [5] Microsoft. *Socket Services in .NET*. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/fundamentals/networking/sockets/socket-services>. Acesso em: 15 ago. 2025.