

Chat em Grupo

Projeto de Aplicação de Chat em grupo desenvolvido na disciplina de Redes de Computadores

Emerson Luiz Cruz Junior
231003531

Isabela Souza Sisnando de Araujo
231018884

Lucas Gabriel de Oliveira Lima
231003406

Abstract—Esse documento é um relatório do Primeiro Projeto de Aplicação da Disciplina de Redes da Universidade de Brasília. Com o objetivo de colocar em prática o conhecimento teórico a respeito de Programação com Sockets, Paradigma de Cliente e Servidor e Sistemas de Redes, o grupo desenvolveu uma Aplicação de Chat em grupo utilizando esses conceitos.

Abstract—This document is a report on the First Application Project of the Networks Class, from University of Brasília. With the purpose of putting into practice knowledge on Socket Programming, Client-Server Paradigm, Protocols and Network Systems, the group has developed a Group Chat Application using those concepts.

Index Terms—socket, client-server, packages, protocol

I. INTRODUÇÃO

O projeto consiste em construir uma aplicação de chat em grupo, a partir de conceitos relacionados com a disciplina de Redes de Computadores. É necessário desenvolver protocolo para transmissão de mensagens e o uso de programação com sockets para estabelecer a conexão entre Cliente e Servidor. A aplicação permite a troca de mensagens entre usuários, a criação de grupos, além de permitir o envio de áudio e imagens.

II. FUNDAMENTAÇÃO TEÓRICA

Ao longo deste projeto, tivemos como fundamentação teórica conceitos que foram desenvolvidos em cima das camadas, principalmente, de aplicação e de transporte da internet. Nesse sentido, colocamos em prática uma aplicação seguindo o paradigma cliente-servidor, trabalhando com a abstração de sockets na linguagem Python, que nos permite fazer a comunicação de diferentes clientes a um servidor passando o endereço IP e a porta em que essa nossa aplicação está ativa.

Esses sockets seguiam o Transfer Control Protocol (TCP) para o transporte. Escolhermos esse protocolo de transporte por nos parecer mais aconselhado ter a garantia de que um pacote de dados fosse enviado e recebido de uma fonte a um destino.

A troca de mensagens se dava por um formato que especificamos para cada ação de requisição de um serviço de um cliente para um servidor, em que passávamos um código dessa ação e no servidor ele a decodificava e passava a resposta dessa ação para o cliente.

Além disso, nosso servidor mantém ativo os sockets de conexão com os clientes, até que ele saia da aplicação, fazendo

certas restrições ao envio de mensagens de acordo com os grupos que ele está participando, inspirado no conceito de WebSockets que surgiu em versões avançadas do protocolo HTTP.

III. AMBIENTE EXPERIMENTAL E ANÁLISE DE RESULTADOS

A. Descrição do Cenário

A aplicação foi desenvolvida utilizando a linguagem de programação Python, e alguns pacotes desenvolvidos com a mesma linguagem. Para a comunicação entre clientes e o servidor, utilizamos *sockets*, fornecidos pela biblioteca *socket*, e configurado com especificações para seguir o protocolo TCP de transporte. Por conta de alguns processos da aplicação precisarem estar funcionando simultaneamente a outros, precisamos utilizar *threads*, por meio da biblioteca *threading*. Na parte do cliente, utilizamos também o Tkinter para fazer a parte principal da interface, além do auxílio do *Pillow*, para fazer a renderização das imagens em um chat, e do *PyGame*, para poder tocar áudios, e a biblioteca *re* para algumas validações por meio de expressões regulares (como a de e-mail). Além disso, optamos por utilizar a programação orientada a objetos, tanto no cliente como no servidor, para encapsular melhor as funcionalidades comuns a um usuário, um grupo e etc.

• Interface Gráfica

Levando em conta o tempo que tínhamos para a conclusão do trabalho e a intersecção de conhecimentos que tínhamos, optamos por construir a parte do cliente usando principalmente a biblioteca Tkinter [3], principalmente por ela ter sido feita em Python. Ela nos permitiu implementar todos os requisitos que eram necessários no lado do cliente.

Uma das características do Tkinter é que uma tela funciona como um loop, isso nos trouxe um problema na hora de construirmos o chat, porque da forma que projetamos, o serviço de recebimento das mensagens deveria estar funcionando também em um loop, o que nos levou ao uso de threads [4] no lado do cliente. Para o funcionamento adequado do chat, acabamos tendo que refatorar algumas vezes o chat até encontrar uma forma que alinhasse o loop principal da tela do Tkinter e o recebimento dos diferentes tipos de mensagem.

As imagens da interface do cliente estão a seguir:

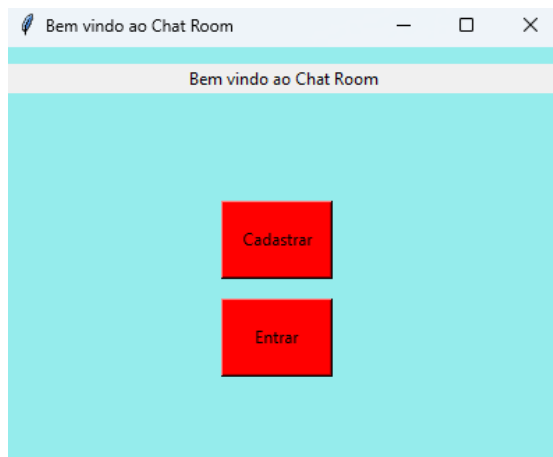


Fig. 1: Home Page

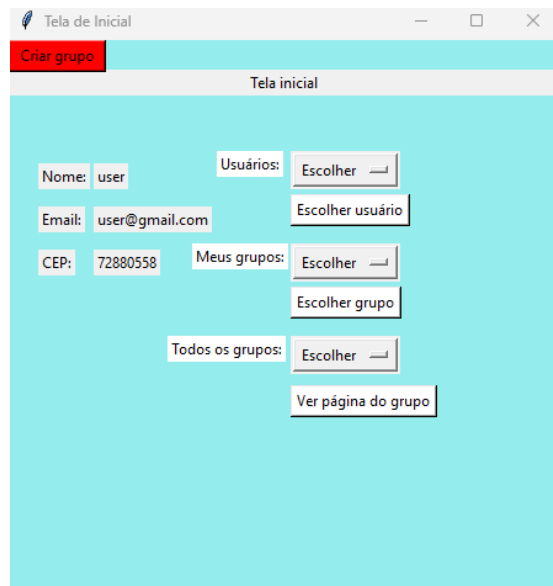


Fig. 4: Página o Usuário - nela pode procurar todos os usuários, grupos ativos e os grupos que o usuário participa

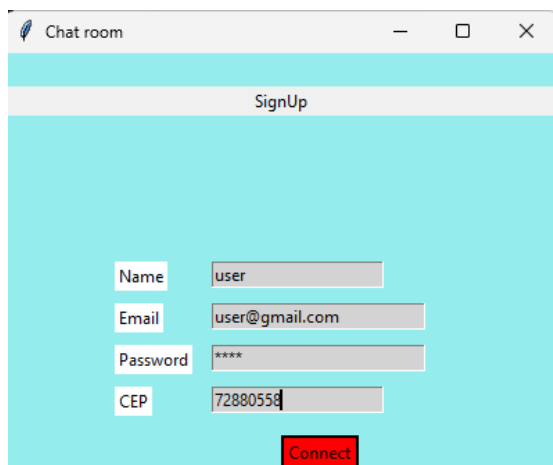


Fig. 2: Sign Up - cadastro de usuários

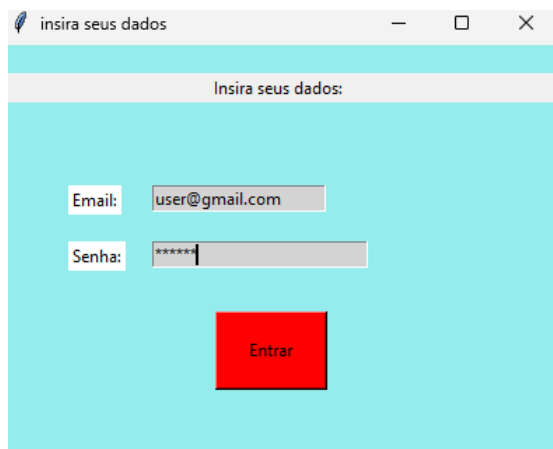


Fig. 3: Login - entrar no sistema



Fig. 5: Perfil do usuários



Fig. 7: Página de Informações do Grupo - nela um usuário pode pedir para entrar no grupo e, se já participar, pode ir para o chat do grupo ou sair desse grupo

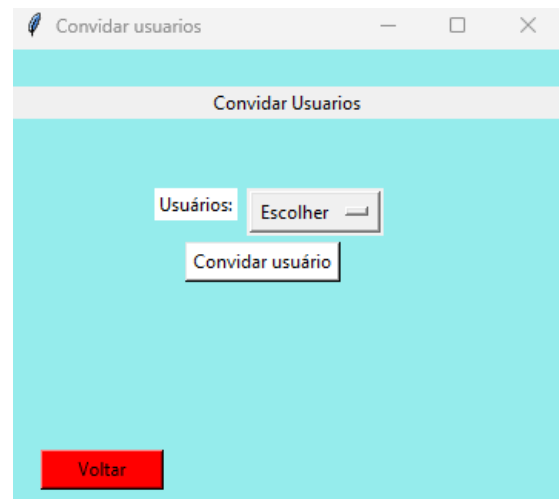


Fig. 8: Página de envio de convites - onde um admin envia convites para entrar em um de seus grupos

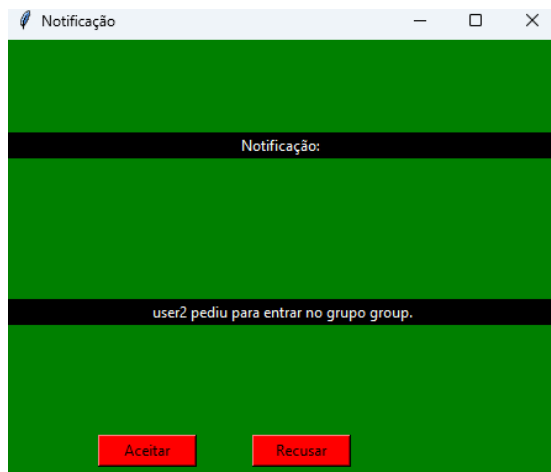


Fig. 6: Notificação - aparece para o admin quando alguém solicita a ele para entrar em um grupo

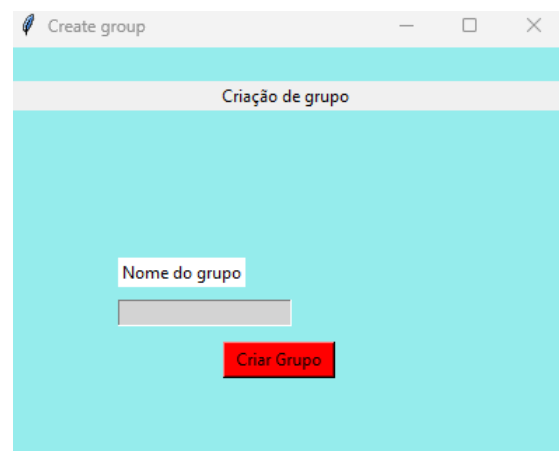


Fig. 9: Página de Criação Grupo

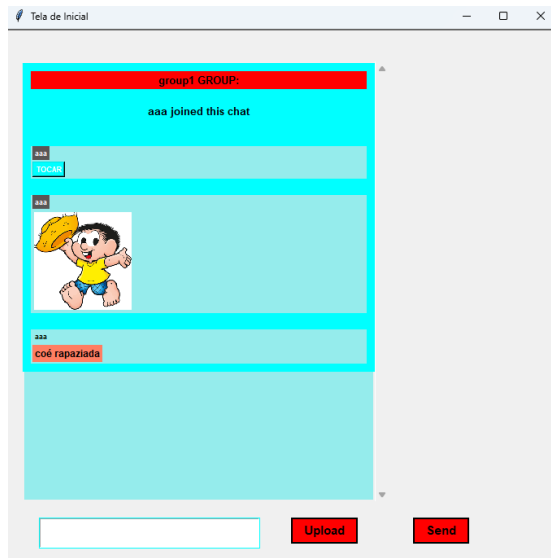


Fig. 10: Chat de um grupo existente - no chat, pode-se enviar imagens, áudios ou mensagens de texto

• Configurações do Cliente

Uma vez iniciada a aplicação, inicializamos também um socket com o **AF_INET**, para requisitar à API do sistema operacional conexões usando **IPv4**, e com o **SOCK_STREAM** para usar o **TCP** nesse socket. Conectamos com o servidor e mantemos essa conexão, instanciando um objeto da classe *ClientUser* para armazenar as informações desse usuário no lado do cliente.

A classe *ClientUser* apresenta alguns métodos direcionados para a comunicação com o servidor, como abrir ou fechar a conexão com um chat, enviar mensagens, etc.

• Configurações do Servidor

Com o servidor rodando a aplicação, ele espera uma nova conexão em seu socket (definido com os mesmos parâmetros do socket no cliente). Uma vez inicializada a conexão, o servidor trata a requisição do cliente, atualizando um objeto *Usuario* já cadastrado ou instanciando outro objeto da classe *Usuario* e lançando uma thread para um loop principal localizado no objeto *Usuario*. Esse loop e a criação de uma thread têm como objetivo evitar problemas com o envio simultâneo de requisições para o servidor vindas de diferentes usuários.

Além disso, a classe *Usuario* tem métodos que tratam o envio de mensagens para o cliente associado a esse usuário (armazenando as mensagens de um determinado canal com outros usuários), o envio de mensagens do cliente associado a esse usuário para outros usuários ou grupos (encontrando o destino e usando o método destinado a receber mensagens nesse destino) e trata convites e pedidos para entrar em grupos, bem como suas notificações.

O servidor apresenta também outra classe para o grupo, que armazena os usuários participantes e o administrador, além de propagar as mensagens para os usuários que participam desse grupo, adicionar e retirar usuários de sua lista.

• Validações de usuário

Na parte do sign-up, é possível fazer o cadastro de um usuário, em que os campos não podem ser vazios, e usamos regex para validar o formato de e-mail e vemos também se o CEP tem um formato válido. Atendendo essas especificações, primeiramente procuramos se existe algum usuário com o mesmo nome ou email fornecido, se encontrarmos, se o sistema encontrar, ele não permite a criação do usuário, caso contrário, o sistema cria esse usuário e o redireciona a sua página inicial.

Na parte de login, um usuário já existente pode acessar a aplicação passando um email existente no sistema e a senha de acesso correta.

• Funcionamento do Chat

Dividimos o funcionamento do chat em chat em grupo (para grupos de usuários) e chat em direto (para mensagens em direto entre usuários), que apesar de suas interfaces serem a mesma, o funcionamento têm algumas diferenças sutis.

Para o chat em grupo, após um usuário ser adicionado à lista de usuários desse grupo, ele tem a permissão de ingressar o chat desse grupo. Quando ele entra no chat, no servidor, guardamos nesse usuário um atributo indicando que ele está conectado a esse grupo, que é o que permite que o servidor envie mensagens à esse usuário por sockets. Após isso, carregamos todas as mensagens antigas desse chat e enviamos para o cliente, onde a interface se encarregará de organizar essas mensagens e deixá-las apresentáveis no chat, no caso de mensagens de texto ou áudio, envia se apenas uma referência de qual arquivo que deveria ser enviado e o tamanho dele, e então o cliente faz a requisição desse arquivo específico ao servidor, e após carregá-lo completamente, ele o insere no chat. Ao enviar uma mensagem de texto, chegará ao servidor, será adicionada ao histórico de mensagens, e então, um método se encarregará a enviar essa mensagem a todos os usuários do grupo que estão ativos e conectados ao grupo, ou seja, o atributo que marca a qual grupo ele está conectado indica que é o grupo dessa mensagem e que o socket de conexão do cliente com o servidor estará ativo. Para mensagens de áudio ou texto, primeiramente é enviado uma mensagem de texto indicando o tamanho e nome do conteúdo que está sendo enviado, após isso, o servidor vai receber dados do tamanho indicado por essa mensagem e vai armazenar esse arquivo [6], e fará o mesmo processo de ver os usuários conectados ao grupo para reenviar esse arquivo, no entanto, ele fará o mesmo processo de indicar que é um arquivo, e o tamanho desse arquivo para o cliente, e então enviará os dados desse arquivo. Para os chat em direto, o funcionamento é parecido em quase todos os quesitos, a diferença é que, ao enviar uma mensagem, quando o servidor vê que o tipo da conexão desse usuário é de um canal em direto, ele apenas envia essa mensagem para o remetente (usuário que enviou a mensagem) e destinatário (o usuário do outro lado desse canal em direto), seguindo as mesmas técnicas para envio de mensagem de áudio e imagem e para o recarregamento do histórico de mensagens. Para renderizar a imagem, utilizamos a biblioteca Pillow [8], com auxílio do método ImageTk, que permite que uma imagem seja

renderizada em um objeto Tkinter. Para tocar áudios, usamos a biblioteca PyGame [7], que conta com um mixer que nos permite dar o play em um áudio ou pausá-lo apenas com um clique em cima do botão de tocar o áudio.

• Formatação de mensagens

Como mencionado anteriormente, especificamos um formato para as mensagens enviadas pelo cliente. Para a conexão inicial do socket, o servidor disponibiliza dois códigos: "0" para cadastrar um novo usuário e "1" para efetuar o "login" de um usuário já existente.

- 0 | email | nome | senha | cep
- 1 | email | senha

Após esse momento inicial, como mencionado anteriormente, atualizamos as informações de um usuário existente ou instanciamos um novo usuário e iniciamos uma thread para o loop principal desse usuário. Para manter a identificação das requisições do cliente, novos formatos e códigos são definidos. Os códigos apresentam as seguintes funcionalidades:

- 0 → abre conexão
- 1 → fecha conexão
- 2 → mensagem para grupo ou usuário (essa distinção é feita a partir qual o tipo de conexão o usuário está fazendo no momento)
- 2U → mensagem no formato de áudio ou imagem
- 2S → requisição de um arquivo de imagem ou áudio específico
- 3 → "logout" (feito ao fechar a aplicação)
- 4 → adiciona um novo usuário
- 5 → manda convite para um grupo
- 6 → pede para entrar em um grupo
- 7 → aceita o convite ou pedido para entrar
- 8 → cria um grupo
- 9 → sai de um grupo
- 10 → pede os grupos registrados no servidor
- 10S → pede os grupos que o usuário está participando
- 11 → pede os usuários registrados no servidor
- 12 → pede as informações pessoais de determinado usuário
- 13 → pede as notificações armazenadas para esse usuário
- 14 → pede as informações de um grupo
- 15 → pergunta se o usuário já está no grupo
- 16 → rejeita convite ou pedido para entrar

Elas são enviadas para o servidor no seguinte formato:

- 0 | tipo de conexão | email ou nome
- 1
- 2 | nome do destino | nome da origem | mensagem
- 2U | tipo de conexão | nome do destino | nome da origem | nome do arquivo | tamanho do arquivo
- 4 | nome do usuário que será adicionado
- 5 | nome do grupo | nome do usuário que foi convidado
- 6 | nome do grupo | nome do usuário que quer entrar no grupo
- 7 | nome do grupo | nome do usuário adicionado
- 8 | nome do grupo | nome do administrador
- 9 | nome do grupo | nome do usuário que saiu

- 10
- 10S
- 11
- 12 | nome do usuário que o cliente deseja obter as informações
- 13
- 14 | nome do grupo que o cliente deseja obter as informações
- 15 | nome do grupo que o cliente quer checar
- 16 | nome do grupo | nome do usuário rejeitado ou que rejeitou o convite

Para cada pedido, o servidor irá tratá-lo seguindo esse formato e realizará as ações correspondentes ao pedido.

• Funcionamento do grupo

Um dos requisitos do projeto era permitir que um usuário fosse convidado para um grupo e que um usuário pudesse solicitar a entrada em determinado grupo. Para atender a isso, seguindo o formato definido acima, o servidor receberá a informação do pedido, processará essa informação e armazenará a notificação no objeto **Usuario** do administrador. A mesma coisa é feita para o convite, com a diferença de que a notificação será armazenada no objeto **Usuario** do convidado e não no do administrador.

Para diferenciar as notificações de um convite ou pedido para entrar, definimos os seguintes formatos e códigos para essas notificações:

Código:

- 3 → convite para um grupo
- 4 → pedido para entrar em um grupo

Formato:

- 3@ nome do grupo
- 4@ nome do grupo @ nome do usuario que quer entrar

Essas notificações aparecem para o usuário ao entrar novamente na aplicação, restando a ele a opção de aceitar ou recusar o que foi solicitado na notificação.

Por simplicidade, definimos que o administrador não poderá sair do grupo, evitando problemas relacionados à definição de um novo administrador; porém, qualquer outro participante poderá sair.

As mensagens enviadas para um grupo são propagadas para todos os presentes na lista de usuários do objeto **Grupo** desse grupo, conforme explicado acima sobre o chat.

• Funcionamento da aplicação

Para explicitar o funcionamento da aplicação, foi feito um **vídeo** [9], no qual é possível visualizar as funcionalidades presentes, bem como os resultados obtidos para o experimento proposto nos requisitos do projeto. Além disso, disponibilizamos também o link para o repositório com o código fonte da aplicação no **GitHub** [10].

B. Analise de Resultados

No roteiro do relatório, foi passado 6 questões, às quais responderemos a seguir:

- A

Aplicação chat em grupo, versão 1.0.0.

```
Notebook@DESKTOP-KPD5LVN MINGW64 /c/codas/Group_Chat_Project/server (main)
$ py main.py
Group chat version 1.0.0 is running on port 3300
```

Fig. 11: Versão do servidor da aplicação

B

De alguns pacotes teste, o IP do cliente (Source) era 192.168.0.4, e o do servidor (Destination) era 192.168.0.4.

Source	Destination
192.168.0.4	192.168.0.4
192.168.0.4	192.168.0.4
192.168.0.4	192.168.0.4
192.168.0.4	192.168.0.4

Fig. 12: Endereços IP de clientes e servidor

C

Esses mesmos pacotes seguem o protocolo TCP, como definidos nos sockets dos clientes e do servidor da aplicação

Protocol

TCP
TCP
TCP
TCP
TCP

Fig. 13: Protocolos dos pacotes transmitidos

D

Escolhendo um dos pacotes de dados, vemos que a porta do servidor é a 3300, como estabelecido em sua configuração. Já a do cliente é a porta 61613.

```
Transmission Control Protocol, Src Port: 61613, Dst Port: 3300,
Source Port: 61613
Destination Port: 3300
```

Fig. 14: Porta do cliente e do servidor

E

O tamanho de dados de alguns pacotes observados podem ser vistos na figura a seguir. O tamanho do pacote inteiro é um pouco maior, por conta dos header das outras camadas da pilha de protocolos, essa informação é apenas do dado (em bytes) enviado pela camada de aplicação. Como se pode observar, os valores são condizentes ao funcionamento esperado da aplicação, porque, no envio e no recebimento de mensagens, estabelecemos o padrão de 1024 bytes a serem enviados por pacote. Essas definições foram feitas pelos métodos "send()" e "recv()" fornecidos pela biblioteca "socket".

```
Data (132 bytes)
Data [truncated]: fffe000030000007c000000610e
[Length: 132]
```

Fig. 15: Tamanho em bytes do pacote de dados

F

Analisando os dados brutos de um pacote escolhido, podemos fazer a análise dos headers em cada camada

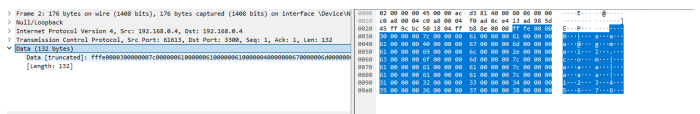


Fig. 16: Dados bruto do conteúdo da mensagem (indicado em azul)

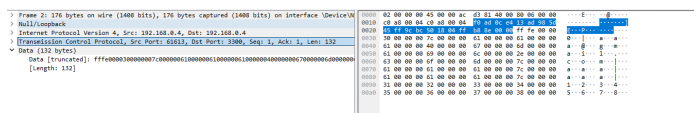


Fig. 17: Dados bruto do header da camada de transporte (indicado em azul)

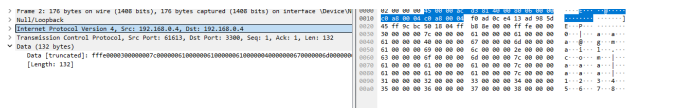


Fig. 18: Dados bruto do header da camada de rede(indicado em azul)

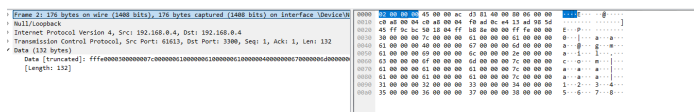


Fig. 19: Dados bruto do header da camada de enlace (indicado em azul)

Ao observar o dado bruto que chega da camada de enlace e é repassado para a camada seguinte, percebe-se o que acontece na pilha de protocolos. Em cada camada, dependendo se o pacote está sendo enviado ou recebido, são adicionados ou retirados dados, conhecidos como cabeçalhos. O papel desses cabeçalhos é fornecer, ao enviar, informações para a camada seguinte que permitam que o protocolo entregue a mensagem ao destino correto. Ao receber, essas informações são processadas com o mesmo objetivo: levar os dados ao destino apropriado, passando camada por camada até que a mensagem chegue ao seu destino final.

IV. CONCLUSÃO

Concluímos este trabalho implementando todos os requisitos apresentados no roteiro para o chat em grupo. Ao longo do processo, adquirimos muitos aprendizados, entre os

quais se destacam: a utilização e configuração de sockets seguindo o protocolo **TCP**, o funcionamento da comunicação no paradigma cliente-servidor e o desenvolvimento de interfaces de usuário. Além disso, com o projeto, tivemos a oportunidade de aplicar na prática conceitos aprendidos em aula que são utilizados diariamente em aplicações.

REFERENCES

- [1] Gabriel Carver, Repositorio no GitHub destinado a disciplina de Redes, 2023
- [2] Documentação da biblioteca *socket.py*, 2024.
- [3] Documentação da biblioteca *tkinter*, 2024.
- [4] Como usar threads no *tkinter*, 2020.
- [5] "CodeLoop - By Ritik", Aplicação de chat em python usando *tkinter*, 2021.
- [6] Malay Bhavsar, Transferência de arquivos usando programação com sockets em python, 2020.
- [7] Documentação da biblioteca *pygame*, 2024.
- [8] Documentação da biblioteca *Pillow*, 2024
- [9] Lucas Gabriel de Oliveira Lima, Emerson Luiz Cruz Junior, Isabela Souza Sisnando de Araujo Gravação do funcionamento da aplicação, 2024.
- [10] Lucas Gabriel de Oliveira Lima, Emerson Luiz Cruz Junior, Isabela Souza Sisnando de Araujo Repositório com o código-fonte do projeto, 2024.