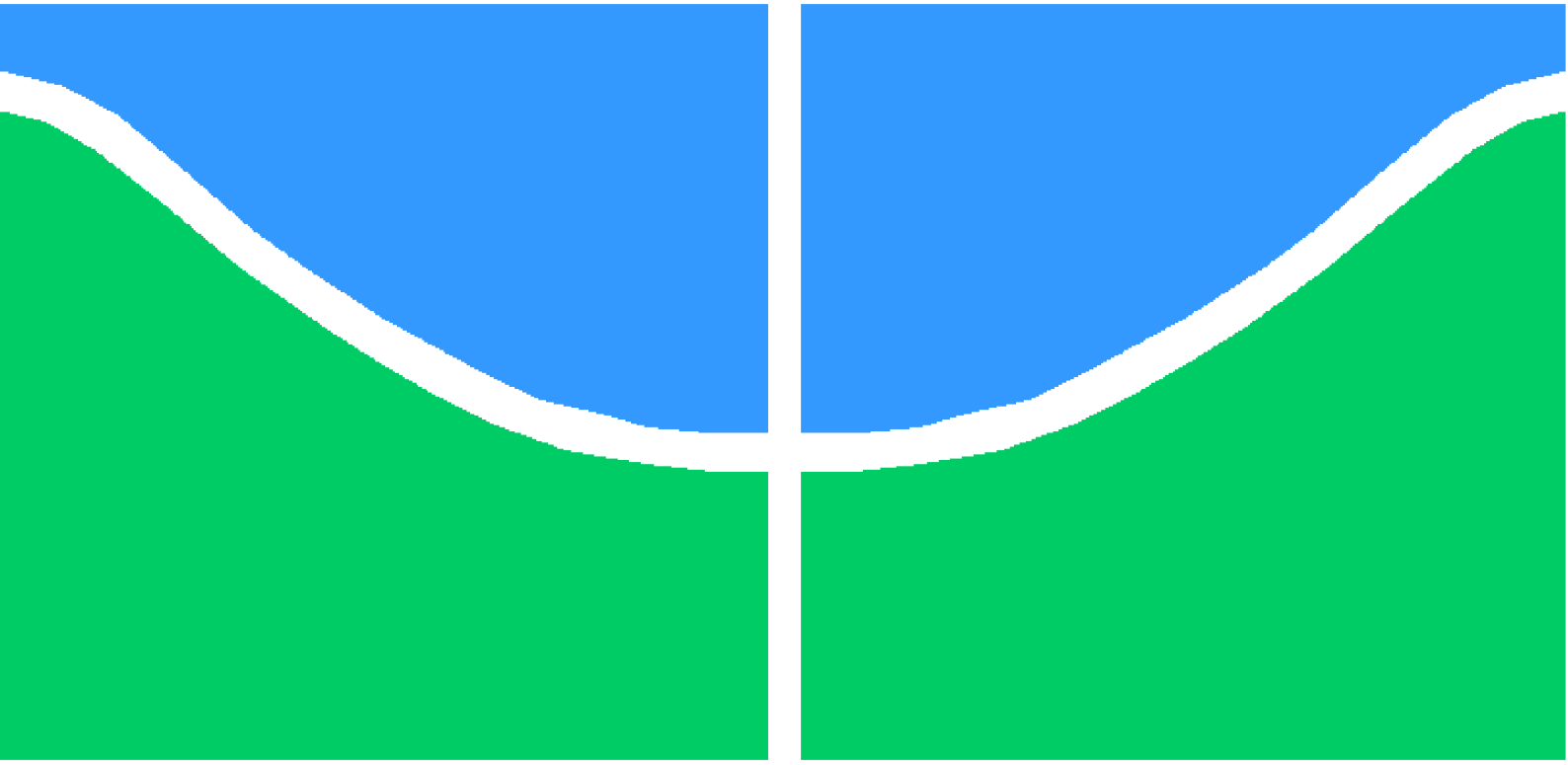

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia Software

Projeto de pesquisa - Criando ambientes virtuais de conversação com uso de sockets

Autores: Antônio Aldísio de Sousa Alves Ferreira Filho 202028211, Carla Rocha Cangussú 170085023, Guilherme Verissimo Cerveira Braz 180018159

Professor: Fernando W. Cruz

Brasília, DF

2022

Sumário

1 Introdução

O presente trabalho tem como objetivo a compreensão da arquitetura de aplicações de rede, segundo a arquitetura TCP/IP, envolvendo a gerência de diálogo. Para isso, foi construído uma aplicação que disponibiliza salas de bate-papo virtuais, nas quais os clientes podem ingressar e interagir (TANENBAUM, 2011).

A sigla TCP significa *Transmission Control Protocol*(Protocolo de Controle de Transmissão) e o IP, *Internet Protocol*(Protocolo de Internet). O modelo de arquitetura TCP/IP foi desenvolvido como uma solução real para a transmissão de dados pela rede e por esse motivo é amplamente utilizado. Para isso, possui quatro camadas (como visto abaixo), que são conhecidas como pilha TCP/IP.

- camada de aplicação;
- camada de transporte;
- camada de internet;
- camada de enlace
- camada de interface de redes;

Para que uma aplicação possa se comunicar com outra sem ter que se preocupar com os detalhes da pilha TCP/IP que gere a rede abaixo dessa aplicação são utilizados *sockets*. Esses têm o papel de abstrair a camada de rede, facilitando assim o processo de transmissão de dados.

Como o protocolo TCP/IP é utilizado para a comunicação segura entre máquinas de uma mesma rede, é muito recomendado para chats. O presente projeto é a aplicação desse modelo com o uso de *sockets* para a construção de salas de bate-papo virtuais. Cada sala receberá um nome e terá o seu limite de participantes determinado ao ser criada. O ingresso de clientes só será permitida em uma sala existente por meio do uso de um identificador, e sempre respeitará o seu limite admitido. Também será possível a saída de clientes de uma sala em que estavam participando. Dessa maneira, os clientes da sala poderão se comunicar (TANENBAUM, 2011).

Para facilitar a entendimento da aplicação do protocolo TCP/IP em uma gerência de diálogo este projeto seguirá a seguinte ordem, Introdução, Metodologia, Descrição da solução, Conclusão e Referências bibliográficas.

Para o desenvolvimento desse trabalho seguimos como base um tutorial encontrado no *youtube* que tinha desenvolvido um chat entre dois clientes (NEURALNINE, 2020)

2 Metodologia

A metodologia aplicada consiste principalmente na pesquisa bibliográfica em livros, portais de periódicos nacionais e internacionais.Também foram realizadas busca de vídeos sobre o tema na plataforma YouTube. Foram utilizados materiais que tratam sobre redes de computadores, modelo de arquitetura de transmissão de dados por redes. Quanto às ferramentas, serão utilizados editores de código-fonte, motores de buscas científicas e formuladores de tabelas.

Para elaboração deste projeto pode ser acompanhada pela a tabela 1.

Tabela 1 – Cronograma de atividades desenvolvidas

Datas	**Atividades realizadas**
26/04/22	Definir linguagem a ser utilizada no projeto
27/04/22	Apresentação de pesquisa sobre o tema e Inicio do desenvolvimento em pares (trio)
29/04/22	Continuação do desenvolvimento
30/04/22	Resolução de *bugs*
02/05/22	Organização do relatório e apresentação
03/05/22	Resolução de *bugs* Finalização do relatório e apresentação

Para elaboração dos artefatos de código estão localizados em um repositório do github (https://github.com/redes2021-2/Trabalho-Final). Por outro lado, os artefatos de relatório e apresentação foram desenvolvido no overleaf e no canvas, respectivamente.

3 Descrição da solução

A solução consiste na criação de dois arquivos um **server.py** e outro **client.py** para que haja a comunicação em grupos concorrentemente entre vários clientes. Além disso, os mesmos terão apelidos e os grupos limite de pessoas especificados em sua criação. Outra funcionalidade disponível sera ver a lista de clientes em sua sessão de chat. Antes de rodar a solução é necessário atender alguns pré-requisitos:

- Python => 3.9
- Tkinter

Para rodar a solução é necessário seguir os seguintes passos:

1. python server.py
2. python client.py

3.0.1 Server.py

Esse arquivo é escrito, em *Python 3*, toda a parte de servidor com *sockets* utilizando o protocolo TCP/IP. Visto isso, foram criadas 3 funções principais, que serão detalhadas abaixo:

- *receive*
- *handle*
- *adcast*

3.0.1.1 Receive

Nessa função ocorre a conexão com o cliente, e a troca das informações iniciais para a conexão, como o *nickname* e o nome do grupo que o cliente deseja ingressar. Além disso, na *Receive* começa uma *thread* para que o servidor possa rodar vários clientes simultaneamente.

```python
def receive():
    while True:
        print('Lista de clientes:')
        print(nicknames)
        print('Lista de grupos:')
        print(groups)
        print('Lista de pares:')
        print(pairs)

        # aceitar conexão e receber endereços
        client, adress = server.accept()
        print(f'Conectado em {str(adress)}')

        # pegar apelido dp cliente e adicionar a lista de apelidos
        client.send("NICK".encode('utf-8'))
        nickname = client.recv(1024).decode('utf-8')
        print(f'Apelido: {nickname}')
        nicknames.append(nickname)
        clients.append(client)

        # pegar grupo do cliente e adicionar a lista de grupos
        client.send("GROUP".encode('utf-8'))
        group = client.recv(1024).decode('utf-8')
        print(f'Grupo: {group}')
        # checar se o grupo existe
        if group not in groups:
            groups.append(group)

        # check se o par existe
        if (client, group) not in pairs:
            # adicionar par cliente/grupo a lista de pares
            pairs.append((client, group))

        global currentGroup  # declarar como global      You, 3 days ago • Atualizando par
        currentGroup = group

        print(f'Usuario {nickname} foi adicionado a lista')  # print log
        # enviar mensagem de entrada para todos os clientes
        broadcast(f'{nickname} entrou no chat!'.encode('utf-8'))

        # criar thread para tratar mensagens
        thread = threading.Thread(target=handle, args=(client,))
        thread.start()
```

Função *Receive*

3.0.1.2 Handle

A *Handle* é responsável pelo o tratamento da mensagem. A mensagem é recebida e seu *broadcast* é feito para os clientes do mesmo grupo. Caso haja erro o cliente, e grupo são removidos das listas.

```
def handle(client):
    while True:
        try:
            message = client.recv(1024)     # receber mensagem
            print(message)                  # print log da mensagem

            # enviar mensagem para todos os clientes
            broadcast(message)
        except:
            # se o cliente sair, remove-lo da lista

            index = clients.index(client)   # pegar indice do cliente
            clients.remove(client)          # remover cliente da lista
            client.close()                  # fechar conexão
            nickname = nicknames[index]     # pegar apelido do cliente
            nicknames.remove(nickname)      # remover apelido da lista

            # achar par do cliente e remover
            for pair in pairs:
                if pair[0] == client:
                    pairs.remove(pair)
            # checar se tem grupo sem par
            for group in groups:
                if group not in [pair[1] for pair in pairs]:
                    groups.remove(group)

            break
```

função *Handle*

3.0.1.3 Broadcast

Na função *Broadcast* foi implementada a parte de encaminhar a mensagem recebida para todos que estão no grupo desse cliente que enviou a mesma.

```
def broadcast(message):
    # enviar mensagem para todos os clientes do grupo
    print(f'Enviando mensagem para todos os clientes do grupo {currentGroup}')
    if currentGroup in groups:
        for pair in pairs:
            if pair[1] == currentGroup:
                pair[0].send(message)
```

função *Broadcast*

3.0.2 Client.py

Nesse arquivo é feita toda a parte de **client** com *sockets* e utilizando o protocolo TCP/IP. Além disso, é utilizada uma biblioteca de terceiros a *'tkinter'*, a qual é necessária a instalação da mesma, para que o serve gere uma *GUI* (General User Interface) para a aplicação de chat. Nesse arquivo então foi criada uma classe *Client* com 5 funções principais e o *init* da classe:

- Write

- Receive
- Guiloop
- Stop
- Init
- Iniciate

3.0.2.1 Write

Nessa parte da solução é feito o envio do mensagem para o -se tiver link *linkar* com a *subsubsection handle -handle* do servidor.

```
def write(self):
    # pegar mensagem do input
    message = f"{self.nickname}: {self.input_text.get()}"
    # enviar mensagem
    self.socket.send(message.encode('utf-8'))
    # limpar campo de texto
    self.input_text.delete(first=0, last='end')
```

Função *Write Client*

3.0.2.2 Receive

Essa função é responsável pelo o recebimento das mensagens do servidor. No servidor são tratados os casos iniciais feitos no *receive do server*, e principalmente ele mostra as mensagens na interface.

```
def receive(self):
    while self.running:
        try:
            message = self.socket.recv(1024).decode('utf-8')
            if message == 'NICK':
                self.socket.send(self.nickname.encode(
                    'utf-8'))  # enviar nickname
            elif message == 'GROUP':
                self.socket.send(self.group.encode(
                    'utf-8'))   # enviar grupo
            else:
                if self.gui_done:
                    # adicionar mensagem ao chat
                    self.chat_text.config(state='normal')
                    self.chat_text.insert('end', message + '\n')
                    self.chat_text.yview('end')
                    self.chat_text.config(state='disabled')
        except ConnectionAbortedError():
            break
        except:
            print("Error")
            self.sock.close()   # fechar socket
            break
```

Função *Receive Client*

3.0.2.3 GuiLoop

O *GuiLoop* faz toda a parte da *GUI* da aplicação, os detalhes de posicionamento, tamanho e estilo da mesma.

```
# sem funcionalidade so a interface front-end
def gui_loop(self):
    # criar janela
    self.win = tkinter.Tk()
    self.win.configure(bg="lightblue")
    self.win.title("Chat")
    self.win.geometry("400x400")

    # input text
    self.chat_label = tkinter.Label(
        self.win, text=self.group, bg='lightblue')
    self.chat_label.configure(font=("Courier", 12))
    self.chat_label.pack(padx=20, pady=5)

    # campo de texto
    self.chat_text = tkinter.scrolledtext.ScrolledText(
        self.win, width=40, height=10)
    self.chat_text.pack(padx=20, pady=5)

    # label para input de mensagem
    self.input_label = tkinter.Label(
        self.win, text="Mensagem", bg='lightblue')
    self.input_label.pack(padx=20, pady=5)

    # campo de texto de entrada
    self.input_text = tkinter.Entry(self.win, width=40)
    self.input_text.configure(font=("Courier", 12))
    self.input_text.pack(padx=20, pady=5)

    # botao para enviar mensagem
    self.send_button = tkinter.Button(
        self.win, text="Enviar", command=self.write)
    self.send_button.pack(padx=20, pady=5)

    self.gui_done = True

    # ao fechar janela chamar stop
    self.win.protocol("WM_DELETE_WINDOW", self.stop)
    self.win.mainloop()
```

Função GuiLoop

3.0.2.4 Stop

Nessa parte do código é realizada o fechamento do *socket* e do processo em geral desse cliente caso a janela da *GUI* seja fechada.

```
def stop(self):
    self.running = False    # parar loop
    self.win.destroy()      # fechar janela
    # stop thread
    # self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.socket.close()     # fechar socket
    exit()
```

função *Stop*

3.0.2.5 Init

Adicionalmente existe a função *init* da classe, que nada mais é que a função realizada em sua instanciação.É no *Init* que são realizadas as etapas essenciais para que as outras funções funcionem corretamente. São feitas a conexão e criação da *socket*, o pedido input do usuário para o *nickname* e grupo para a criação da *GUI* e são feitas as *threads* na função *guiloop e receive*.

```
def __init__(self, host, port):
    # Etapa de conexão
    self.socket = socket.socket(
        socket.AF_INET, socket.SOCK_STREAM)     # criar socket
    # conectar ao host
    self.socket.connect((host, port))

    msg = tkinter.Tk()
    msg.withdraw()

    # pedir nickname
    self.nickname = simpledialog.askstring(
        "Nickname", "Escolha seu apelido:")
    # pedir grupo
    self.group = simpledialog.askstring(
        "Group", "Escolha seu grupo:")

    self.gui_done = False
    self.running = True
```

função *Init Clinet*

3.0.2.6 Iniciate

Nessa função é realizada a instanciação da classe Client

```
def iniciate():
    client = Client(HOST, PORT)
```

função Iniciate Client

4 Conclusão

Neste relatório foi apresentado a aplicação do protocolo TCP/IP no desenvolvimento de salas virtuais de bate-papo. Para isso, foi feito uma breve contextualização do TCP/IP e do uso de *socketes*. Também foram detalhados os encontros feitos pela a equipe para a elaboração do projeto. E por fim a descrição da solução construída.

Como podemos ver, o TCP/IP é um protocolo usado amplamente por garantir a transmissão de dados segura em uma rede. Deste modo, é excelente para gerenciamentos de diálogo entre máquinas, como ocorre em um chat. Isso só é possível devido a presença de um servidor e dos clientes presentes na rede. Esse processo de diálogo é facilitado com a presença de *socketes*.

O presente projeto teve como requisitos, o uso da linguagem de programação Python, permitir a criação de salas virtuais de bate-papo com nome da sala e limite de participantes; Permitir ingresso de clientes, com um identificador, em uma sala existente , de acordo com o limite admitido na sala; Saída de clientes de uma sala em que estava participando; Diálogo entre os clientes da sala. Além disso, foi desenvolvida uma interface gráfica para tornar o processo de interação mais amigáveis entre os clientes.

4.1 Comentário dos integrantes

4.1.1 Antônio Aldísio

Ter um projeto final com esse tema foi de grande experiência, pois tive contato com a linguagem Python e durante o desenvolvimento do trabalho conseguiu entender melhor como funciona o protocolo TC/IP na pratica. Esse projeto tem uma infinidade de opções de melhorias, pois temos diversos tipos de aplicativos de mensagem no dia atual, acho que seria interessante pegar a funcionalidade de criar chat privado com uma senha para as partes.

Sobre minha participação no trabalho foi de forma ativa e participativa. Como adotamos programação em trio para podemos participar de todas as etapas de programação juntos. Na construção dos artefatos também aconteceu uma boa divisão de tarefas. Por fim, acredito que meu aprendizado sobre esse trabalho esta atrelado ao bom trabalho em grupo desenvolvido.

4.1.2 Carla Cangussú

O projeto final me proporcionou entender melhor o funcionamento de protocolo TCP/IP na prática e também dos sockets. Foi interessante ver que não é tão complicado a criação d chat básico, como eu imaginava que seria. Além disso, tive mais contato com a linguagem de programação Python que não tenho tanto conhecimento. Como melhoria seria interessante ter mais tempo de desenvolvimento, uma vez que tivemos prova no mesmo período e os dois foram divulgados juntos. Dessa maneira, tivemos que dividir o tempo entre os dois.

Sempre me mostrei presente e disponível para a realização do trabalho. Também busquei estar atenta para que a carga de trabalho entre os participante estivesse equilibrada e ninguém ficasse sobre carregado. Como todos os integrantes sempre estiveram comprometidos com o trabalho, esse ocorreu de maneira tranquila e fluida para todos, resultando assim em uma boa entrega.

4.1.3 Guilherme Braz

O projeto final foi uma ótima tarefa para poder testar na pratica os conhecimentos adquiridos na matéria, principalmente com relação ao protocolo TCP/IP, o qual foi usado para realizar o mesmo. Ao codificar as sockets e montar as funções para troca de mensagem entre servidor e cliente pode se analisar bem as dinâmicas do protocolo, ainda mais na hora de corrigir erros. Também gostaria de acrescentar que fazendo a parte do relatório e ter de explicar o código ajuda muito também a analisar e correlacionar os conhecimentos teóricos e práticos da matéria.

Com relação a minha participação no trabalho, considero que ajudei bastante o grupo e sempre estive disponível para eles, assim como eles também. Visto isso, acredito que meu grupo foram uma das razões para os quais aprendi bastante com esse projeto.

4.1.4 Tabelas de notas

Tabela 2 – Nota dos integrantes do grupo

Alunos	**Notas**
Antônio Aldiso	SS
Carla Cangussú	SS
Guilherme Braz	SS

Referências

NEURALNINE. *Simple GUI Chat in Python*. YouTube, 2020. Disponível em: <https://www.youtube.com/watch?v=sopNW98CRag>. Citado na página 2.

TANENBAUM, A. S. *Redes de computadores*. [S.l.: s.n.], 2011. ISBN 9788576059240. Citado na página 2.