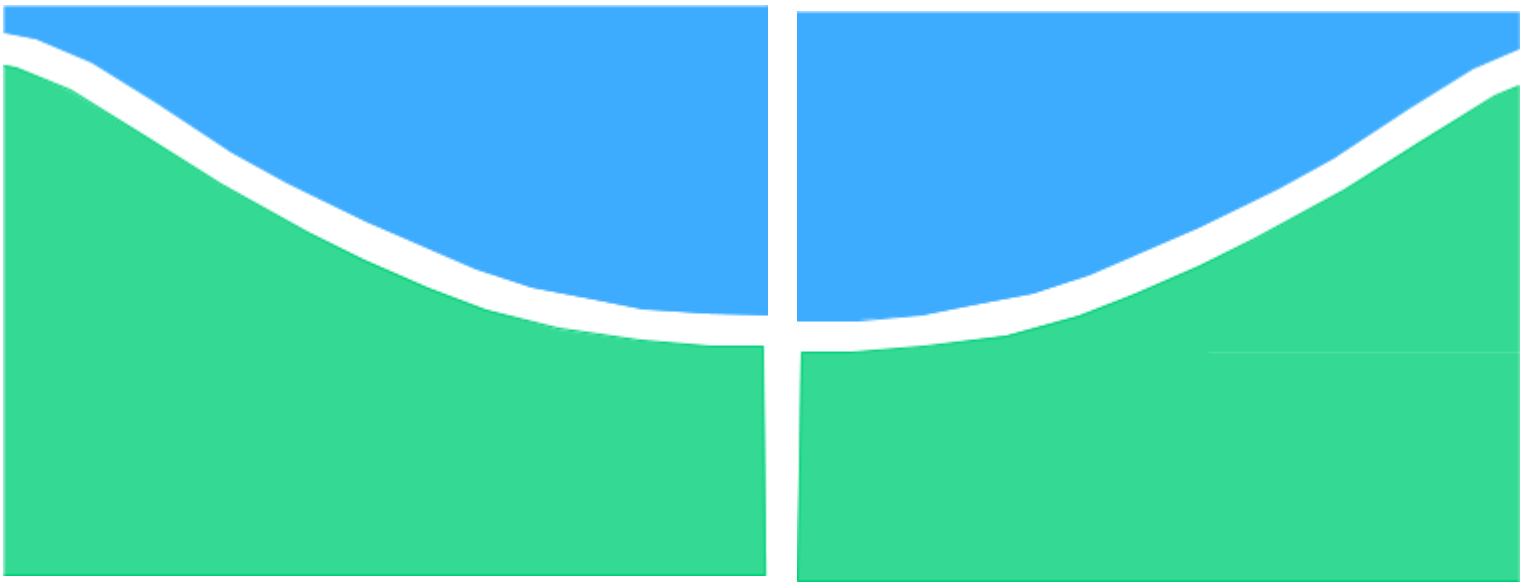


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA

PROGRAMAÇÃO PARA SISTEMAS PARALELOS E DISTRIBUÍDOS

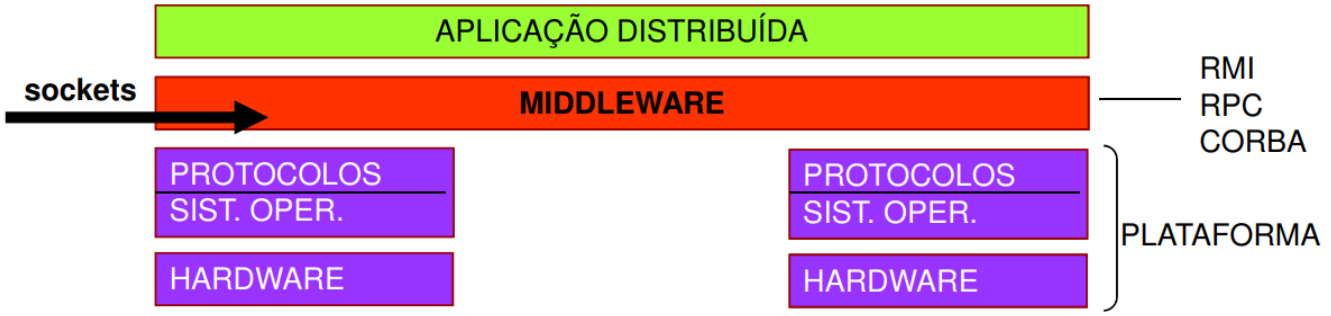
Integrantes:
Jackes da Fonseca, RA: 190030291

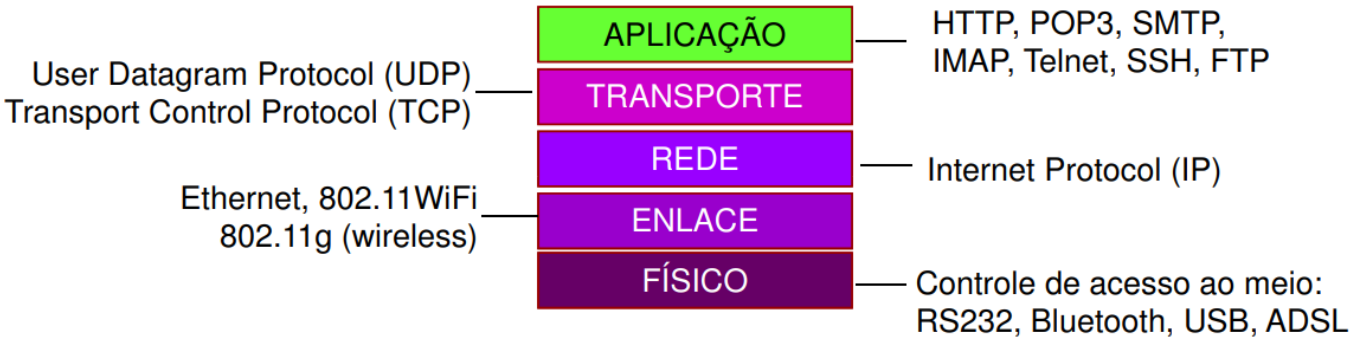
Turma: 2021.2

Brasília, DF
2022

Introdução

Os sockets UDP e TCP são a interface provida pelos respectivos protocolos na interface da camada de transporte.





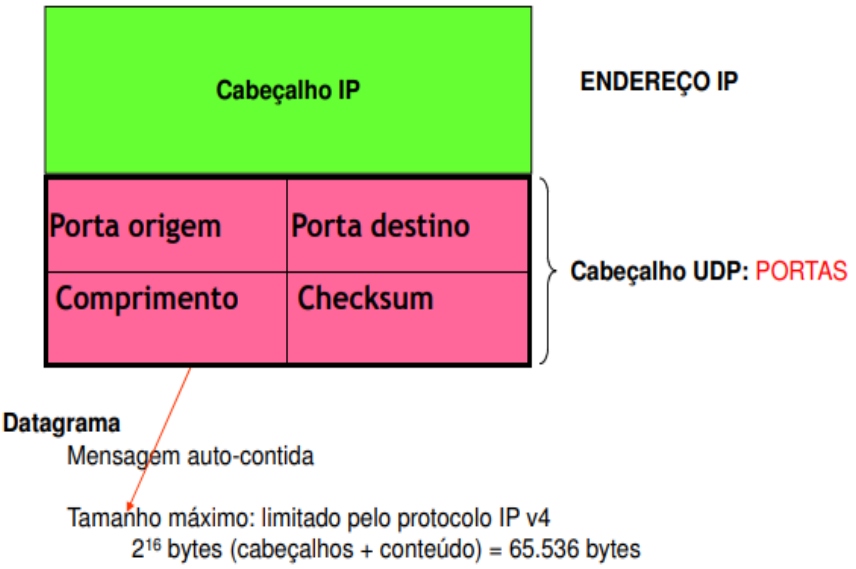
Utilização:

- comunicação interprocessos distribuídos
- Aplicações cliente-servidor

1 - Sockets UDP

Protocolo UDP: canal não-confiável

- Não garante entrega dos datagramas
- Pode entregar datagramas duplicados
- Não garante ordem de entrega dos datagramas
- Não tem estado de conexão (escuta, estabelecida)



Programação com sockets UDP

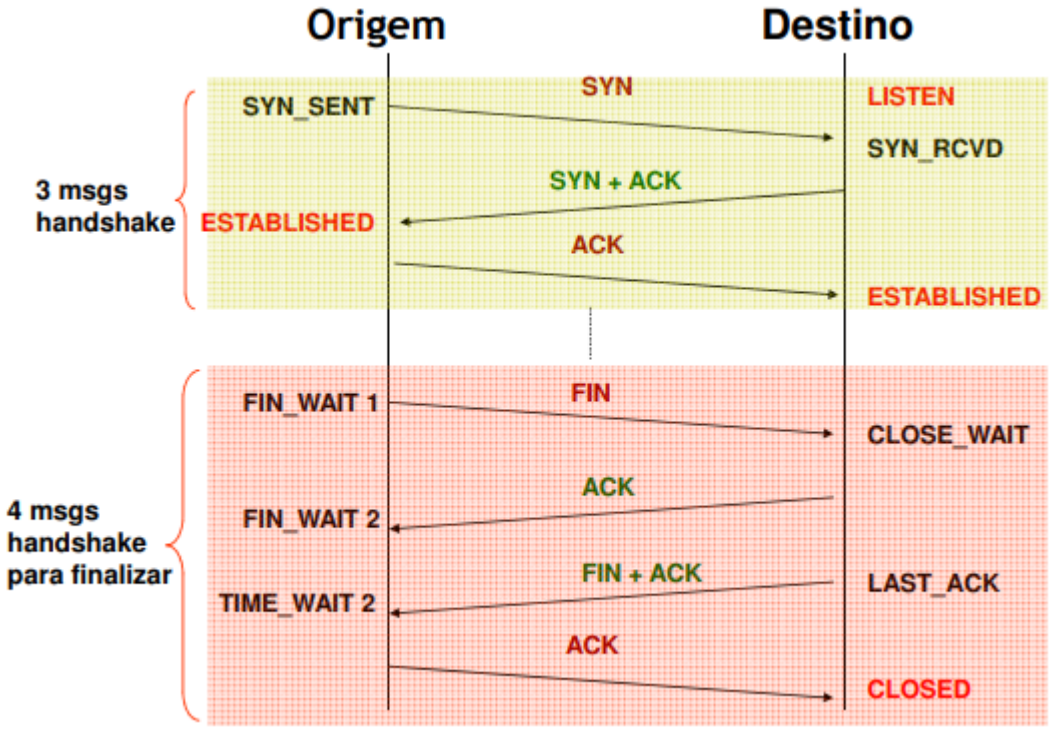
- Criar socket
  - DatagramSocket s = new DatagramSocket(6789);
- Receber um datagrama
  - s.receive(req);
- Enviar um datagrama
  - s.send(resp);
- Fechar um socket
  - s.close();
- Montar um datagrama para receber mensagem
  - new DatagramPacket(buffer, buffer.length);
- Montar um datagrama para ser enviado
  - new DatagramPacket(msg, msg.length, inet, porta);
- Buffer e msg são byte[]

2 - Sockets TCP

Protocolo TCP: implementa um canal confiável.

- Do ponto de vista do desenvolvedor: fluxo contínuo (stream)
- São fragmentados pelo TCP em segmentos
- Garante a entrega dos segmentos
- Não há duplicação
- Garante ordem de entrega dos segmentos
- Possui conexão e, portanto, controla o estado de conexão (escuta, estabelecida, fechada)

- Ponto-a-ponto: um sender:um receiver – sockets são conectados
- Controle de congestionamento: TCP controla o sender quando a rede congestiona.
- Controle de fluxo: Controla o sender para não sobrecarregar o receiver



Programação com Sockets TCP

- Servidor cria socket de escuta numa porta (ex. 6789)
  - ServerSocket ss = new ServerSocket(6789);
- Servidor aceita uma conexão e cria novo socket para atendê-la
  - Socket a = ss.accept();
- Cliente cria socket de conexão
  - Socket s = new Socket("localhost", 6789)
- Fecha o socket
  - s.close()
- Cliente escreve no stream de saída do socket
  - Criar um stream de dados – getOutputStream retorna uma classe abstrata
  - DataOutputStream sai = new DataOutputStream(sc.getOutputStream());
  - sai.writeUTF("mensagem para o servidor");
- Cliente lê o stream de entrada do socket
  - DataInputStream ent = new DataInputStream(sc.getInputStream())
  - String recebido = ent.readUTF();
- Leitura e escrita são similares no servidor, mas são feitas usualmente no socket retornado pelo método accept

Soluções

Utilizando 1 Worker

1- Entre na pasta que contém os arquivos

$ cd /letra\ a/

2 - Execute o comando make para compilar o cliente e o servidor

$ make

3 - Inicie o servidor no seguinte formato
./tcpServer [ip_servidor] [porta_servidor]
$ ./tcpServer 172.17.0.1 5000

4 - Inicie o cliente no seguinte formato
./tcpServer [ip_servidor] [porta_servidor] [tamanho vetor] [tamanho_vetor]
$ ./tcpServer 172.17.0.1 5000 100000

Utilizando múltiplos Workers

1- Entre na pasta que contém os arquivos

$ cd /letra\ b/

2 - Execute o comando make para compilar o cliente e o servidor

$ make

3 - Inicie o servidor no seguinte formato
./tcpServer [ip_servidor] [porta_servidor]
$ ./tcpServer 172.17.0.1 5000

4 - Inicie o cliente no seguinte formato
./tcpServer [quantidade_servidores] <[ip_servidor] [porta_servidor]> [tamanho vetor] [tamanho_vetor]
$ ./tcpServer 3 172.17.0.1 5000 172.17.0.1 5001 172.17.0.1 5002 100000