

**Disciplina:** Conectividade de Sistemas Ciberfísicos

**Curso:**

**Nome:**

## **Trabalho Final**

**Tema:** Sistema Cliente-Servidor com Múltiplos Clientes e Integração com Dispositivo Físico (ESP32)

### **Objetivo:**

Desenvolver um **sistema completo de comunicação cliente-servidor**, com suporte a múltiplos clientes simultâneos e integração real com um **dispositivo ESP32**, consolidando os conceitos de **sockets, threads, protocolos TCP/UDP e IoT**.

### **Descrição Geral do Sistema**

O trabalho consiste em criar um **sistema de comunicação distribuído** onde:

- Um **servidor central (Python)** gerencia conexões simultâneas;
- **Clientes em Python** trocam mensagens via terminal;
- Um **ESP32** atua como cliente físico, enviando dados simulados e respondendo a comandos.

O sistema deve permitir **interação entre todos os nós** (clientes ↔ servidor ↔ ESP32), tanto com **TCP** quanto com **UDP**.

## Etapas e Requisitos do Projeto

### Etapa 1 – Servidor TCP

Crie o arquivo `server_tcp.py` que:

- Aceite **múltiplos clientes simultâneos** (mínimo 3).
- Utilize **Threads** para gerenciar as conexões.
- Retransmita mensagens recebidas de um cliente para todos os demais.
- Mostre no terminal logs com **conexões, mensagens e desconexões**.

**Aceite:** o servidor deve permitir trocas de mensagens entre três ou mais clientes sem travamentos.

### Etapa 2 – Servidor UDP

Crie o arquivo `server_udp.py` que:

- Receba mensagens de clientes UDP e **retransmita manualmente** para todos os endereços conectados.
- Permita que cada cliente se identifique com um nome.

**Aceite:** funcionamento correto com ao menos três clientes, com troca de mensagens via UDP.

### Etapa 3 – Clientes Python

Crie `client_tcp.py` e `client_udp.py` que:

- Permitam interação via terminal com o servidor.
- Mostrem o nome do remetente antes das mensagens.
- Implementem os comandos:
  - `/nick <nome>` → define o nome do cliente;
  - `/sair` → encerra a conexão.

**Aceite:** clientes se comunicam entre si via servidor, com nomes visíveis e desconexão correta.

## Etapa 4 – Integração com ESP32

Crie o código `esp32_client.ino` que:

- Conecte o ESP32 via Wi-Fi ao servidor **TCP (porta 5000)**;
- Envie a cada 2 segundos dados simulados, por exemplo:  
`{"type": "data", "from": "esp32", "payload": {"temp": 25.3, "hum": 60}}`
- Receba comandos vindos dos clientes Python:
  - `led_on` → acende o LED do ESP32;
  - `led_off` → apaga o LED.

**Aceite:** o ESP32 envia dados periodicamente e responde corretamente aos comandos recebidos.

## Etapa 5 – Teste de Desempenho (Benchmark)

Adicione o comando `/bench <bytes>` nos clientes Python (TCP e UDP) para enviar grande volume de dados (ex.: 10 MB) e medir o **tempo total de envio**.

O resultado deve ser mostrado no terminal. **Justificar o motivo da diferença de tempo, caso tenha.**

**Aceite:** o teste deve ser executado corretamente nos dois protocolos, exibindo diferença de tempo entre TCP e UDP.

#### **Etapa 6 – Vídeo Explicativo (até 5 minutos)**

Grave um vídeo com duração **máxima de 5 minutos**, mostrando:

1. **Explicação resumida** da arquitetura do sistema (servidor, clientes e ESP32);
2. **Demonstração prática** do sistema em funcionamento:
  - Conexão de múltiplos clientes;
  - Envio e recebimento de mensagens;
  - Comunicação com o ESP32 e acionamento do LED;
  - Execução do teste de desempenho (/bench).
3. **Comentário final** com dificuldades e aprendizados.

 O vídeo pode ser entregue em formato .mp4 ou link (Drive/YouTube não listado).

## Critérios de Avaliação

A nota será composta por duas partes:

- **50% — Funcionamento prático:** o sistema deve funcionar corretamente conforme os requisitos.
- **50% — Explicação técnica:** o aluno deve demonstrar domínio do código e **responder às perguntas do professor** durante a avaliação.

Critério	Descrição	Peso total	Funcionamento (50%)	Explicação (50%)
<b>Servidor TCP</b>	Aceita múltiplos clientes, retransmite mensagens e mantém estabilidade	<b>20%</b>	10%	10%
<b>Servidor UDP</b>	Comunicação via datagramas e reenvio manual	<b>15%</b>	7,5%	7,5%
<b>Clientes Python</b>	Interface funcional e comandos corretos (/nick, /sair)	<b>10%</b>	5%	5%
<b>Integração com ESP32</b>	Envio de dados e resposta a comandos led_on / led_off	<b>20%</b>	10%	10%
<b>Teste de desempenho</b>	Execução e comparação entre TCP e UDP	<b>15%</b>	7,5%	7,5%
<b>Vídeo explicativo</b>	Clareza, demonstração completa e tempo dentro do limite	<b>15%</b>	15,00%	
<b>Organização e código</b>	Estrutura, clareza, comentários e boas práticas	<b>5%</b>	5,00%	
<b>Total</b>		<b>100%</b>	<b>50%</b>	<b>50%</b>

## Estrutura Recomendada de Pastas

```
trabalho_final/
├── tcp/
│   ├── server_tcp.py
│   └── client_tcp.py
├── udp/
│   ├── server_udp.py
│   └── client_udp.py
└── esp32/
    └── esp32_client.ino
video/
    └── demonstracao.mp4
```

## Lideres de grupos, com ESP32

- **Santiago - Esp32**
- **Gabriel Schwerdt - Esp32**
- **Victor Henrico - Esp32**
- **Spisila - Esp32**
- **Camyllle - Esp32**
- **Nicolas Letti - Esp32**
- **Heitor - Esp32**
- **Leonardo Simioni - Esp32**
- **Rafael Engel - Esp32**
- **Isabella - Esp32**
- **Gabriel Alberto Mota - Esp32**
- **Samuel - Esp32**