

**UNIVERSIDADE DE SÃO PAULO**

Sistemas de Informação - EACH

DAVI FERNANDES MENESES DA SILVA, N. USP: 14760241

GABRIEL GONÇALVES DE SOUZA RIBEIRO, N. USP: 14691679

# Desenvolvimento de Sistemas de Informação Distribuídos

Relatório da parte II do trabalho semestral

SÃO PAULO, SP

2025

## 1. Introdução

Nesta segunda parte do trabalho, foram realizadas melhorias no sistema de troca de mensagens entre peers, com o objetivo de garantir consistência na informação de status distribuído utilizando o relógio de Lamport. Além disso, foi implementada a funcionalidade de busca e download de arquivos entre os peers ativos.

## 2. Alterações no funcionamento do relógio local

O relógio lógico foi adaptado para funcionar conforme o algoritmo de Lamport:

- O relógio inicia com valor 0.
- Antes de enviar qualquer mensagem, incrementa-se o relógio local em 1.
- Ao receber uma mensagem, o relógio local é atualizado para  $\max(\text{clock\_local}, \text{clock\_msg})$  e, em seguida, incrementado em 1.
- Toda atualização de relógio imprime no terminal:  
=> Atualizando relógio para <valor>

## 3. Alterações no gerenciamento de peers

Foi adicionada uma estrutura de dicionário para armazenar, para cada peer conhecido:

- Seu status (ONLINE ou OFFLINE)
- Seu relógio lógico mais recente conhecido

As regras implementadas foram:

- Sempre que se recebe uma mensagem direta (HELLO, BYE, etc.), o status do remetente é atualizado para ONLINE (ou OFFLINE se for BYE) e o relógio só é atualizado se for maior.
- Quando uma lista de peers (PEER\_LIST) é recebida, o peer atualiza sua base de conhecimento apenas se o relógio fornecido for maior que o já armazenado.

## 4. Comando Buscar

O comando [4] Buscar arquivos permite ao peer buscar arquivos compartilhados nos peers ONLINE. O funcionamento é:

1. Envia-se a mensagem LS para todos os peers ONLINE.
2. Cada peer responde com LS\_LIST contendo <nome>:<tamanho> de seus arquivos.
3. Uma lista compilada de arquivos disponíveis na rede é apresentada ao usuário.
4. O usuário seleciona um arquivo da lista para download.
5. O peer envia uma mensagem DL para o peer que possui o arquivo.
6. O peer destino responde com FILE <nome> 0 0 <conteudo\_em\_base64>.
7. O conteúdo é decodificado e salvo localmente com o mesmo nome.
8. O sistema exibe:  
Download do arquivo <nome> finalizado.

## 5. Decisões de Projeto e Refatorações

- O dicionário de peers foi refatorado de {peer: status} para {peer: (status, relógio)}.
- A função tratar\_mensagem() foi expandida para lidar com os novos tipos: LS, LS\_LIST, DL e FILE.
- Utilizou-se Base64 para codificar arquivos binários em mensagens de texto.
- Funções auxiliares como buscar\_arquivos() foram adicionadas para modularidade e legibilidade.

## 6. Testes realizados

- Três instâncias foram abertas com diretórios e arquivos diferentes.
- Mensagens HELLO, GET\_PEERS e PEER\_LIST foram usadas para validar sincronismo e status.
- O comando [4] Buscar arquivos foi testado com arquivos de conteúdo e arquivos vazios.
- O sistema lida corretamente com a ausência de conteúdo base64, exibindo mensagem explicativa em caso de erro.

## 7. Dificuldades enfrentadas

- Ajustar a ordem correta de atualização de relógio e status por tipo de mensagem.
- Simular a troca de mensagens em ambientes assíncronos e com múltiplas instâncias em execução simultânea.

## 8. Instruções de execução

Para executar corretamente o sistema distribuído com múltiplos peers, foi adotada uma estrutura de pastas e arquivos individualizada por instância. Cada peer possui:

- Um diretório exclusivo com seus arquivos compartilhados;
- Um arquivo .txt contendo a lista de vizinhos com os quais pode se comunicar.

|— main.py

|— diretorio/      → Arquivos do peer 1

|— diretorio2/     → Arquivos do peer 2

|— diretorio3/     → Arquivos do peer 3

|— directorio4/      → Arquivos do peer 4

|— vizinhos\_peer1.txt   → Vizinhos do peer 1

|— vizinhos\_peer2.txt   → Vizinhos do peer 2

|— vizinhos\_peer3.txt   → Vizinhos do peer 3

|— vizinhos\_peer4.txt   → Vizinhos do peer 4

### **Como executar:**

Abra um terminal separado para cada peer e execute os comandos:

# Terminal 1

**python main.py 127.0.0.1:9001 vizinhos\_peer1.txt directorio**

# Terminal 2

**python main.py 127.0.0.1:9002 vizinhos\_peer2.txt directorio2**

# Terminal 3

**python main.py 127.0.0.1:9003 vizinhos\_peer3.txt directorio3**

# Terminal 4

**python main.py 127.0.0.1:9004 vizinhos\_peer4.txt directorio4**