

# RCI-Mini-Projecto1

## Fluxo de Execução das Tarefas

---

### Legenda

- **(depende de: X)** indica tarefas que precisam estar concluídas antes.
  - Ordem é **estritamente sequencial**, mas permite paralelização interna por módulo.
- 

## Fase 0 — Preparação

### Tarefa 0.1 — Configurar ambiente de desenvolvimento

- Criar Vagrantfile, preparar toolchain e todas ferramentas necessárias, ambiente do laboratório.
- *(depende de: nada)*

### Tarefa 0.2 — Criar estrutura inicial do projecto

- Pastas e scripts automáticos (se necessário)
  - *(depende de: 0.1)*
- 

## Fase 1 — Servidor de Peers (UDP)

### Tarefa 1.1 — Implementar servidor UDP

- Processar: **REG**, **UNR**, **PEERS**
- Manter tabela de peers registados (IP, porta, seqnumber).

- *(depende de: 0.2)*

## Tarefa 1.2 — Testar servidor UDP

- Testes manuais com `nc`, `telnet`, ou pequenos clientes de teste.
  - *(depende de: 1.1)*
- 

## Fase 2 — Cliente UDP do Peer

### Tarefa 2.1 — Implementar cliente UDP

- Enviar: `REG`, `UNR`, `PEERS`
- Processar: `SQN`, `LST`, `OK`, `NOK`
- *(depende de: 1.2)*

### Tarefa 2.2 — Integrar cliente UDP com comando `join`

- Registrar peer → obter seqnumber → pedir lista → devolver ao módulo TCP.
  - *(depende de: 2.1)*
- 

## Fase 3 — Ligação TCP entre Peers (Overlay)

### Tarefa 3.1 — Implementar servidor TCP

- Aceitar conexões para LNK e FRC.
- Validar N- (vizinhos internos).
- Responder `CNF` ou fechar.
- *(depende de: 2.2)*

### Tarefa 3.2 — Implementar cliente TCP

- Estabelecer sessões LNK e FRC.
- Validar: só conectar a peers com seqnumber menor.
- *(depende de: 3.1)*

### **Tarefa 3.3 — Implementar gestão de vizinhos internos/externos**

- Actualizar tabelas de vizinhos após ligações LNK/FRC.
- *(depende de: 3.2)*

### **Tarefa 3.4 — Implementar reconexão automática**

- Quando perde vizinho externo → pedir lista PEERS → tentar novos LNK/FRC.
- *(depende de: 3.3)*

### **Tarefa 3.5 — Implementar comandos:**

- `show neighbors`
  - `release seqnumber`
  - Parte TCP de `leave` e `exit`
  - *(depende de: 3.4)*
- 

## **Fase 4 — Identificadores (Camada de Aplicação)**

### **Tarefa 4.1 — Implementar estrutura local de identificadores**

- Lista, vetor, ou hashset simples.
- *(depende de: nada; pode ser feita paralelamente com Fase 3)*

### **Tarefa 4.2 — Implementar comandos:**

- `post identifier`
  - `unpost identifier`
  - `list identifiers`
  - *(depende de: 4.1)*
- 

## Fase 5 — Protocolo de Pesquisa Distribuída (QRY/FND/NOTFND)

### Tarefa 5.1 — Implementar receção/envio de QRY

- Validar hopcount
- Reencaminhar para vizinhos
- Gerar respostas corretas (FND / NOTFND)
- *(depende de: 3.5 e 4.2)*

### Tarefa 5.2 — Integrar com comando `search identifier`

- Se peer não conhecer → emitir QRY
  - Se receber FND → guardar identificador
  - Se esgotar hopcount → NOTFND
  - *(depende de: 5.1)*
- 

## Fase 6 — Testes e Validação

### Tarefa 6.1 — Testes unitários por módulo

- Testar cliente UDP isolado
- Testar servidor TCP isolado
- Testar post/unpost/local IDs

- *(depende de: Fases 1–5)*

## **Tarefa 6.2 — Testes integrados com múltiplos peers**

- join → ver conectividade
  - leave → ver reacção de reconexão
  - pesquisa de identificadores com hopcount > 1
  - cenários com FRC
  - simulação de falhas
  - *(depende de: 6.1)*
- 

## **Fase 7 — Finalização**

### **Tarefa 7.1 — Organizar repositório GitHub**

- Docs
- Vagrantfile
- Estrutura coerente
- ETC
- *(depende de: 6.2)*

### **Tarefa 7.2 — Elaborar relatório (modelo oficial)**

- Preencher todas as secções: introdução, realizações, arquitectura, etc.
  - *(depende de: 7.1)*
- 

## **Fluxo Final (visão linear)**

1. 0.1 → configurar ambiente
2. 0.2 → estrutura do projecto

- 3. 1.1 → servidor UDP
  - 4. 1.2 → testes servidor
  - 5. 2.1 → cliente UDP
  - 6. 2.2 → comando join
  - 7. 3.1 → servidor TCP
  - 8. 3.2 → cliente TCP
  - 9. 3.3 → gestão de vizinhos
  - 10. 3.4 → reconexão automática
  - 11. 3.5 → comandos TCP (show, release, leave, exit)
  - 12. 4.1 → estrutura de identificadores
  - 13. 4.2 → comandos post/unpost/list
  - 14. 5.1 → protocolo QRY/FND/NOTFND
  - 15. 5.2 → comando search
  - 16. 6.1 → testes unitários
  - 17. 6.2 → testes integrados
  - 18. 7.1 → preparar repositório
  - 19. 7.2 → relatório final
- 

## **Distribuição Inteligente das Tarefas por Membros**

NOTA: Para cada tarefa a ser realizado devem ser criadas anotações sobre o que foi realizado para facilitar criar o relatório final.

---

### **Membro A — Infraestrutura + Servidor de Peers (UDP) + Cliente UDP**

Este membro constrói toda a base do sistema: ambiente, servidor de peers e o módulo cliente que será usado pelos outros.

## **Responsabilidades principais**

1. Configuração do ambiente (Vagrant, Makefile, estrutura inicial)
2. Servidor UDP:
  - REG
  - UNR
  - PEERS
  - Geração de seqnumber e tabela de peers
3. Cliente UDP no peer
4. Integração completa com comando `join`
5. Integração UDP de `leave` e `exit`
6. Testes independentes (com nc / telnet)
7. Documentação das estruturas do servidor no relatório

## **Justificação**

O servidor de peers é central, mas simples relativamente aos outros módulos. Uma única pessoa controla tudo o que é UDP, eliminando conflitos.

---

# **Membro B — Rede Sobreposta TCP (Overlay) + Gestão de Ligações**

Este membro é responsável pela parte mais crítica do projecto: a overlay P2P.

## **Responsabilidades principais**

1. Servidor TCP para aceitar LNK e FRC
2. Cliente TCP para enviar LNK e FRC
3. Gestão de vizinhos internos/externos (N+, N-)

4. Mecanismo de reconexão automática quando perde vizinhos externos
  - Solicitar PEERS (usa o cliente do Membro A)
  - Tentativas LNK/FRC
5. Implementação dos comandos:
  - show neighbors
  - release seqnumber
  - Parte TCP de leave
  - exit (componente TCP)
6. Tratamento robusto de erros TCP (CNF, fechos inesperados)
7. Construção dos diagramas de protocolos no relatório
  - LNK
  - FRC
  - CNF
  - Fecho e reconexão

## **Justificação**

Esta parte exige maior atenção à lógica, estados e consistência.

Concentrar numa só pessoa evita inconsistências entre cliente/servidor TCP e vizinhos.

---

## **Membro C — Identificadores + Pesquisa Distribuída + Interface da Aplicação**

Este membro implementa toda a camada lógica de aplicação que corre por cima da overlay.

### **Responsabilidades principais**

1. Estrutura local de armazenamento dos identificadores

## 2. Comandos:

- post identifier
- unpost identifier
- list identifiers

## 3. Protocolo de pesquisa distribuída (QRY, FND, NOTFND)

- Emissão de QRY com hopcount
- Encaminhamento para vizinhos
- Respostas FND / NOTFND

## 4. Comando `search identifier`

## 5. Integração completa com o módulo TCP (Membro B)

## 6. Testes funcionais da pesquisa com hopcount > 1

## 7. Secção do relatório sobre o protocolo de pesquisa

## Justificação

A lógica de identificadores e pesquisa depende da overlay, que o Membro B constrói.

Cuidar separadamente da camada de aplicação dá isolamento e permite trabalho paralelo.

## Visão Geral do Mapeamento

Membro	Área Principal	Tarefas
<b>A</b>	UDP + Ambiente	Servidor de peers, Cliente UDP, join, leave UDP, infra
<b>B</b>	TCP + Overlay	Cliente/Servidor TCP, vizinhos, FRC, reconexão, comandos overlay
<b>C</b>	Aplicação	Identificadores, post/unpost/list, QRY/FND/NOTFND, search

# Visão de dependências entre membros (SSOT — Single Source of Truth)

- **Membro A** entrega primeiro o servidor UDP e o cliente UDP.  
*Membros B e C dependem deste módulo para prosseguir.*
  - **Membro B** constrói a overlay logo após A disponibilizar UDP.  
*Membro C depende da overlay para começar a pesquisa.*
  - **Membro C** pode começar a parte local (post/unpost/list) independentemente.  
Depois integra com a overlay do Membro B.
- 

## Equilíbrio de carga

- Membro A: lógica moderada + IO UDP + infra + comandos simples.
- Membro B: parte mais exigente do projecto (TCP + topologia).
- Membro C: camada de aplicação, recursão P2P, hopcount, pesquisa distribuída.

A carga é equilibrada porque o módulo TCP é mais complexo, enquanto UDP + aplicação têm complexidade equivalente quando somadas.