Protocolo P2P Distributed CD Tester

Grupo 120482-121497

junho 2025

1 Introdução

Este documento descreve o protocolo de comunicação **P2P** utilizado entre os nós da Rede de Testes Distribuída. O protocolo define como os nós se descobrem, trocam tarefas, monitoram falhas e realizam eleições para recuperação automática.

2 Visão Geral

A comunicação entre nós é realizada via **UDP**, utilizando mensagens serializadas em **JSON**. Cada nó escuta em uma porta UDP específica e envia mensagens diretamente para os peers conhecidos.

3 Tipos de Mensagens

A tabela abaixo lista os principais tipos de mensagens trocadas entre os nós:

Tipo	**Descrição**
CACHE_UPDATE	Atualização do cache de estado da rede
HEARTBEAT	Sinaliza que o nó está ativo
CONNECT	Solicita entrada em uma rede existente
CONNECT_REP	Resposta à solicitação de conexão
TASK_ANNOUNCE	Anúncio de disponibilidade de tarefas
TASK_REQUEST	Solicitação de tarefa
TASK_SEND	Envio de tarefa para execução
TASK_CONFIRM	Confirmação de receção de tarefa
PROJECT_ANNOUNCE	Anúncio de novo projeto
RECOVERY_ELECTION	Mensagem de candidatura para eleição de nó de recuperação
RECOVERY_ELECTION_REP	Resposta de participação na eleição de recuperação
EVALUATION_RESPONSIBILITY_UPDATE	Atualização de responsável por projeto/avaliação
RECOVERY_ELECTION_RESULT	Resultado da eleição de recuperação

4 Formato das Mensagens

Todas as mensagens são enviadas em formato JSON, com os seguintes campos principais:

Listing 1: Formato geral de mensagem

```
{
  "cmd": "HEARTBEAT",
  "data": { ... },
  "ip": "192.168.1.10",
  "port": 8001,
  "timestamp": 1710000000.0
}
```

- **cmd**: Tipo da mensagem (ver tabela acima)
- **data**: Dados específicos do comando
- **ip**, **port**: Endereço do nó remetente
- **timestamp**: Momento do envio (em segundos desde epoch)

5 Fluxos de Comunicação

5.1 a) Entrada de um novo nó

1. O nó envia uma mensagem `CONNECT` para um nó conhecido.
2. O nó existente responde com `CONNECT_REP`, informando peers e atribuindo um ID.
3. O novo nó passa a enviar/receber heartbeats e participar da rede.

5.2 b) Distribuição de tarefas

1. O nó com tarefas envia `TASK_ANNOUNCE`.
2. Um nó ocioso responde com `TASK_REQUEST`.
3. A tarefa é enviada via `TASK_SEND`.
4. O resultado é devolvido via `TASK_RESULT`.

5.3 c) Detecção de falhas

1. Nós enviam `HEARTBEAT` periodicamente.
2. Se um nó não recebe heartbeat de outro por tempo limite, considera-o falho.

5.4 d) Eleição de nó de recuperação

1. Nós detectam falha e enviam `RECOVERY_ELECTION`.
2. Os peers respondem com `RECOVERY_ELECTION_REP`.
3. O resultado da eleição é divulgado via `RECOVERY_ELECTION_RESULT`.
4. O nó eleito assume as tarefas do nó falho e envia `EVALUATION_RESPONSIBILITY_UPDATE`.

6 Exemplos de Mensagens

HEARTBEAT

```
{
  "cmd": "HEARTBEAT",
  "data": {
    "id": "123456789",
    "peers_ip": ["192.168.1.11:25000"]
  },
  "ip": "192.168.1.10",
  "port": 8001,
  "timestamp": 1710000000.0
}
```

TASK_SEND

```
{
  "cmd": "TASK_SEND",
  "data": {
    "project_ID": "172927292791",
    "module": "test_mod1.py",
    "project_name": "nome_projeto",
    "api_port": 5001,
    "eval_id": "1234567890",
    "url": "https://github.com/...",
    "token": "hjsjh1ka019",
    "type": "url"
  },
  "ip": "192.168.1.10",
  "port": 8001,
  "timestamp": 1710000000.0
}
```

RECOVERY_ELECTION

```
{
  "cmd": "RECOVERY_ELECTION",
  "data": {
    "candidate_id": "123456789",
    "failed_node": "987654321",
    "extra": "..."
    "failed_node_addr":"192.73.81.43:8002"
    "timestamp": 1710000000.0
  },
  "ip": "192.168.1.10",
  "port": 8001,
  "timestamp": 1710000000.0
}
```

EVALUATION_RESPONSIBILITY_UPDATE

```
{
  "cmd": "EVALUATION_RESPONSABILITY_UPDATE",
  "data": {
    "project_id": "123456789",
    "new_node": "987654321",
    "eval_id": "68611"
  },
  "ip": "192.168.1.11",
  "port": 8001,
  "timestamp": 1710000001.0
}
```

CACHE_UPDATE

```
{
  "cmd": "CACHE_UPDATE",
  "data": {
     "addr": "192.73.82.43:8002",
     "peers_ip": get_peers_ip(),
     "evaluations": self.network_cache["evaluations"],
     "projects": self.network_cache["projects"],
     "stats": self._get_status(),
  },
  "ip": "192.168.1.10",
  "port": 8001,
  "timestamp": 1710000002.0
}
```

7 Portas e Protocolo

- **Comunicação entre nós**: UDP, portas 8001, 8002, etc (uma por nó)
- **API HTTP**: TCP, portas 5001, 5002, etc (uma por nó)

7.1 Endpoints Internos

O sistema de testes distribuídos utiliza dois endpoints internos críticos para comunicação entre nós: `/file` e `/task`. Estes endpoints não são destinados à interação direta com usuários, mas são fundamentais para o funcionamento interno da rede distribuída.

7.1.1 Endpoint `/file`

O endpoint `/file/{file_id}` implementa um mecanismo eficiente para transferência de arquivos entre nós, permitindo a distribuição de projetos de teste pela rede.

- **Método:** GET
- **Parâmetros:** `file_id` - Identificador único do projeto ou diretório
- **Comportamento:** Quando um nó recebe uma tarefa, mas não possui localmente o projeto necessário, ele utiliza este endpoint para solicitar os arquivos ao nó responsável pelo projeto.

Este endpoint implementa dois comportamentos distintos dependendo do tipo de recurso solicitado:

1. **URLs de GitHub:** Se o `file_id` corresponde a um URL de projeto no GitHub, o sistema responde com o URL original, permitindo que o nó solicitante baixe diretamente da fonte.
2. **Arquivos locais:** Se o `file_id` corresponde a um diretório local, o sistema compacta automaticamente o diretório em um arquivo ZIP e o retorna, permitindo a transferência eficiente de projetos pela rede.

A implementação no servidor Flask inclui limpeza automática dos arquivos temporários após a transferência:

```
@after_this_request
def remove_file(response):
    try:
        os.remove(zip_file)
    except Exception:
        pass
    return response
```

7.1.2 Endpoint `/task`

O endpoint `/task` é essencial para o mecanismo de distribuição de resultados de testes entre os nós da rede.

- **Método:** POST
- **Conteúdo:** Objeto JSON contendo resultados de execução de testes
- **Comportamento:** Após a execução de um módulo de teste, o nó executor envia os resultados para o nó responsável pelo projeto via este endpoint.

O payload típico enviado para este endpoint inclui:

```
{
    "task_id": "project_123::test_module.py",
    "passed": 15,
    "failed": 2,
    "time": 1.45,
    "project_id": "project_123",
    "module": "test_module.py",
    "ip": "192.168.1.5",
    "port": 8003
}
```

Este endpoint é crucial para a tolerância a falhas do sistema, pois implementa um mecanismo de encaminhamento: se o nó que recebe os resultados não é mais o responsável pelo projeto (devido a uma reorganização após falha), ele automaticamente encaminha os resultados para o novo nó responsável:

```
if responsible_node and not is_responsible:
    url = f"http://{responsible_node[0]}:{responsible_node[1]}/task"
    await asyncio.get_event_loop().run_in_executor(
        None,
        partial(self.node_context.send_http_post, url, info)
    )
```

Estes dois endpoints trabalham em conjunto para garantir a eficiente distribuição de tarefas e a coleta de resultados em uma rede dinâmica onde nós podem entrar, sair ou falhar a qualquer momento.

8 Observações

- Todos os nós implementam o protocolo acima e aceitam mensagens de qualquer peer ativo na rede.
- O sistema é tolerante a falhas e redistribui tarefas automaticamente em caso de falha de um nó.
- Para detalhes de cada uma das mensagens, consulte o script em `src/network/message.py`.