Relatório - Sistema de Chat P2P - TP3 - Redes

Nome : ***Carlos Ferreira dos Santos Junior***

Visão Geral

O sistema de chat P2P foi implementado em Python utilizando a classe principal P2PChat que gerencia todas as funcionalidades do protocolo. A arquitetura é baseada em um modelo híbrido onde cada peer atua simultaneamente como cliente e servidor, realizando conexões com outros nós da rede. O sistema utiliza threads dedicadas para gerenciar conexões simultâneas, garantindo que cada peer conectado tenha sua própria thread para processamento de mensagens, evitando bloqueios durante a comunicação. O sistema implementa um mecanismo de consenso distribuído através de mineração de mensagens com validação por hash MD5.

Mecanismos de Rede

O sistema utiliza codificação binária estruturada com o módulo struct do Python para serialização das mensagens de rede. O protocolo define quatro tipos de mensagens identificadas por códigos específicos: PeerRequest (0x1) para solicitação de lista de peers contendo apenas um byte de tipo, PeerList (0x2) para resposta com peers conhecidos incluindo um byte de tipo seguido por quatro bytes indicando o número de peers em formato big-endian e quatro bytes para cada endereço IP representando os octetos, ArchiveRequest (0x3) para solicitação de histórico contendo apenas um byte de tipo, e ArchiveResponse (0x4) para envio do histórico completo com um byte de tipo seguido por quatro bytes indicando o número de chats e os dados serializados dos chats. Cada chat é serializado incluindo um byte com o tamanho do texto, os bytes do texto da mensagem em ASCII, dezesseis bytes do código de verificação e dezesseis bytes do hash MD5.

Recebimento de Mensagens

O recebimento é implementado através de:

1. **Socket TCP**: Utiliza `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`
2. **Threads dedicadas**: Cada peer conectado tem uma thread para processar mensagens
3. **Leitura estruturada**: Função `_recv_exact()` garante leitura correta dos tipos de mensagem e dados
4. **Timeout de conexão**: Para evitar bloqueios indefinidos

```
def _recv_exact(self, sock: socket.socket, size: int) -> bytes:
    """Lê exatamente size bytes do socket"""
    data = b''
    while len(data) < size:
        chunk = sock.recv(size - len(data))
        if not chunk:
            raise ConnectionError("Conexão fechada durante leitura")
        data += chunk
    return data
```

Envio de Mensagens

O envio de mensagens utiliza conexões TCP persistentes que são mantidas abertas com todos os peers conectados. O código possui locks para proteger o acesso concorrente às estruturas de dados compartilhadas durante operações de envio. Para distribuição de atualizações, o programa realiza broadcast enviando mensagens simultaneamente para todos os peers conectados na rede. O tratamento de erros garante que peers com falhas de comunicação sejam automaticamente desconectados e removidos das estruturas de dados, mantendo a integridade da rede.

Descoberta de Peers

O sistema implementa descoberta de peers através de conexão inicial que aceita um peer como parâmetro de linha de comando e conecta automaticamente se fornecido. A descoberta automática executa um loop a cada 5 segundos enviando mensagens PeerRequest para todos os peers conectados e recebendo PeerList com novos peers, conectando automaticamente aos peers descobertos.

Hash MD5

O sistema utiliza a biblioteca padrão `hashlib` do Python para cálculo do hash MD5:

```
import hashlib

def _calculate_chat_hash(self, history: List[Chat]) -> bytes:
    """Calcula o hash MD5 para validação do último chat"""
    # ... preparação da sequência ...
    return hashlib.md5(sequence).digest()
```

O hash é calculado sobre os últimos 20 chats do histórico (ou todos se houver menos de 20), utilizando dados serializados de cada chat e construindo uma sequência binária concatenando todos os dados. O algoritmo obtém os chats a serem processados, inicializa uma sequência vazia e itera sobre cada chat, onde para mensagens anteriores à última inclui todos os dados serializados completos, mas para a última mensagem exclui o hash MD5 incluindo apenas o tamanho do texto empacotado em struct, o texto codificado em ASCII e o código de verificação. Ao final, retorna o hash MD5 calculado sobre toda a sequência construída usando hashlib.md5(sequence).digest().

Processo de Mineração

O processo de mineração implementa um sistema de Proof of Work simplificado através de um algoritmo que executa um loop infinito gerando códigos de verificação aleatórios de 16 bytes usando random.randint(0, 255). Para cada tentativa, o algoritmo cria um chat temporário com a mensagem, código de verificação gerado e hash zerado, calcula o hash MD5 do histórico atual incluindo esta nova mensagem e verifica se o hash resultante inicia com dois bytes zero. Quando encontra um hash válido que satisfaz o critério, o algoritmo retorna o chat final com o código de verificação correto e o hash calculado, representando uma dificuldade computacional de 1 em 65.536 tentativas.

Execução

Requisitos

- **Python 3.6+**
- **Sistema operacional**: Linux (desenvolvido em Ubuntu 22), Windows (Não testado)
- **Rede**: Acesso TCP na porta 51511

Primeiro peer (servidor inicial):

```
python3 p2p_chat_base.py
```

Peer adicional conectando ao primeiro:

```
python3 p2p_chat_base.py <IP_DO_PRIMEIRO_PEER>
```

Especificando IP de bind do servidor inicial:

```
python3 p2p_chat_base.py <IP_DO_PRIMEIRO_PEER> <IP_LOCAL_BIND>
```

Exemplo de Uso

Cenário: Conectando ao servidor do TP3

```
python3 p2p_chat_base.py pugna.snes.dcc.ufmg.br
```

Ou

```
python3 p2p_chat_base.py
> connect pugna.snes.dcc.ufmg.br
```

Cenário: Dois computadores na mesma rede

```
# Computador 1 (IP: 192.168.1.100)
python3 p2p_chat_base.py

# Computador 2 (IP: 192.168.1.101)
python3 p2p_chat_base.py 192.168.1.100
```

Comandos Disponíveis

Após iniciar o programa, os seguintes comandos estão disponíveis:

- `help` - Mostra lista de comandos
- `peers` - Lista peers conectados
- `hist` - Mostra histórico de mensagens
- `req` - Solicita histórico de todos os peers
- `connect <ip>` - Conecta manualmente a um peer
- `chat <mensagem>` - Envia uma nova mensagem (incluindo o processo de mineração)
- `val` - Valida o histórico atual
- `status` - Mostra status detalhado do sistema
- `debug` - Ativa/desativa modo de mensagens de debug

Mensagem de verificação do TP

Como exigido na especificação, enviei uma mensagem no chat para fins de verificação do TP. O histórico no momento da verificação possuia a seguinte mensagem como mais recente:

```
74: Grupo: Selene Melo e Laura Godinho
```

codigo de verificação: 'c92a75811a9e6fbcbaa8faffe38af4c3'
hash md5: '00000795cab693dd315c28a9e0acb9a5'

A mensagem enviada para a verificação enviada foi a seguinte:

```
75: aluno carlos ferreira
```

codigo de verificação: 'c1979f508a55a4391ca9f89c90d8453a'
hash md5: '000082baf3264e3aa36c1375376a790f'