TP 3 Um Sistema Peer-to-peer de Armazenamento de vídeo

Pedro Andrade Militão

1. Introdução

O objetivo deste trabalho prático é introduzir o aluno a uma Network Peer-to-Peer, normalmente chamada de Torrent. O aluno deve colocar em prática o sistema, utilizando bibliotecas básicas de uma das linguagens: C, C++, Java, Python, Rust e Go, neste caso ele foi implementado em python

Para a execução deste trabalho o comando foi executado da seguinte forma para o Peer e o Cliente, respectivamente:

- `python peer.py ip:porta key-values-files_peer[id] <ip>:<porta> <ip>:<porta>...`
  - `Exemplo: python peer.py 127.0.0.1:5001 key-values-files_peer1 127.0.0.1:5003 127.0.0.1:5002`
- `python cliente.py ip:porta <chunks_desejados>`
  - `Exemplo: python cliente.py 127.0.0.1:5001 1,3,4,5,9`

É recomendada a versão 3.9.1 do Python para executar o trabalho.

2. Arquitetura

- Bibliotecas

Neste trabalho foram utilizadas poucas bibliotecas visto que o problema em si é todo voltado à socket. Elas são:

```
import socket
import struct
import sys
```

Struct: Esta biblioteca executa conversões entre valores Python e estruturas C representadas como objetos de bytes, para a transferência na rede. Foi utilizada ao invés de “tobytes”, e tem duas funções recorrentes no programa:

- pack(*`format`*, *`v1`*, *`v2`*, *`...`*): Retorna um objeto de bytes contendo os valores v1, v2, ... compactados de acordo com o formato da string de formato.
- unpack(*`format`*, *`buffer`*): Descompacte do buffer (presumivelmente compactado por *pack(format, ...)*) de acordo com o formato da string de format.

Socket: A biblioteca principal do trabalho, ela abstrai a camada de redes do programa e implementa a estrutura de dados chamada de socket. Com ela foram utilizadas duas funções:

- sendto(): envia uma struct de bytes com as mensagens do protocolo para o endereço especificado
- recvfrom(): recebe as structs enviadas pelo sendto()

Funções recorrentes:

- encode() e decode(): codifica e decodifica strings para bytes com o formato especificado (‘utf-8’ foi utilizado)
- operações com list comprehension em python

3. Peer

Peers recebem como parâmetro de entrada seu ip:porta no qual vai escutar mensagens, uma key-values-file e a lista de “Vizinhos”

Os peers foram implementados de forma completamente “reativa”, sempre fica esperando uma mensagem chegar escutando no seu “Ip:Porta”, quando ela chega, verifica se é um Hello, Query ou Get e trata de forma correspondente

Peers possuem uma lista de chuncks que ele possui e pode enviar para os clientes, assim como uma lista de “Vizinhos” para os quais ele deve enviar o Query quando este deve ser repassado, respeitando a regra do TTL.

- **Hello:** Ao receber um Hello o Peer envia seu chunkInfo para o cliente e logo em seguida o query para seus vizinhos com TTL começando em 3
- **Query:** Ao receber um Query, o Peer envia pro cliente o seu chunkInfo informando quais chunks ele possui e pode repassar. Em seguida cria diminui em 1 o TTL e repasse para seus vizinhos os desejos do Cliente.
- **ChunkInfo:** O chunkInfo simplesmente informa ao cliente a lista de chunks que ele possui
- **Get:** Ao receber o get do cliente, o peer identifica na lista quais chunks ele deseja receber (da lista que ele tem) e imediatamente encaminha a(s) Response(s) com as informações do(s) chunk(s), junto com o(s) chunk(s) em si.
- **Response:** Mensagem final que realmente envia um chunk do dado que o cliente quer.

4. Cliente

    O Cliente recebe como parâmetro de entrada na linha de comando o Ip:Porta do peer que é a “interface” de entrada deste cliente para o sistema, junto com a lista de chunks que ele deseja receber dos peers.

    O cliente começa enviando um Hello para seu peer de contato e logo depois fica aguardando diversas mensagens de ChunkInfo, armazenando-as em uma lista para mandar o get quando terminar o timeout de 5 segundos.

    Terminada a fase de receber os chunkinfos o cliente itera na lista de ChunkInfos procurando os chunks que ele quer, se um peer tem um ou mais chunks de interesse, o cliente entra em um loop para enviar Gets e receber responses desses gets, processando o response e produzindo parte do arquivo “Log” final logo no recebimento. Terminado um peer o cliente vai para o próximo e repete o processo. Depois de um timeout de 5 segundos o Cliente termina o programa.

5. Discussão

    O sistema de torrent é uma alternativa interessante ao “Download” comum, e de fato é muito mais rapido e eficiente do que a forma comum de comunicação 1 para 1.

    Uma falha deste sistema é que ele no momento não recebe várias responses simultaneamente, ele recebe “Linearmente” essas responses ao invés de recebê-las em paralelo que seria mais eficiente, logo esta seria uma boa melhoria para o futuro.