Trabalho Prático 3
Sistema Peer-to-peer

Arthur Souto Lima

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

arthursl@ufmg.br

1. Introdução

O projeto consiste em implementar um sistema de compartilhamento de arquivo de vídeo segundo um paradigma peer-to-peer por meio de soquetes. O protocolo que rege todo o processo está definido na especificação, de forma que o as mensagens sempre trafegam por soquete UDP. Usam-se paradigmas como alagamento, por exemplo, para que o cliente consiga todas as chunks do vídeo que ele precisa. O trabalho foi implementado em Python, utilizando sobretudo a biblioteca *socket*, bem como outras bibliotecas-padrão para amparar funcionalidades auxiliares.

2. Implementação

2.1. Execução

O programa foi implementado em Python 3 (3.8). No código-fonte entregue, além dos módulos do cliente (*cliente.py*) e peer (*peer.py*), também há dois módulos auxiliares, o *common.py* e o *messages.py*. É importante ressaltar que tanto o cliente quanto o peer usam funções de ambos os módulos, então há a necessidade de cuidado na divisão em pastas, por exemplo no momento dos testes, a fim de evitar problemas com *imports*. Uma solução é copiar não apenas o *cliente.py* (ou o *peer.py*), mas também os dois módulos para cada pasta de cliente ou de peer.

Tal como requerido pela especificação, podemos iniciar o programa usando os comandos abaixo. Convém ressaltar que pode ser necessário fornecer permissão de execução para esses dois scripts principais.

```
./peer.py 10.0.0.3:5000 key-values-files_peer3 10.0.0.2:5002 10.0.0.1:5001
./cliente.py 10.0.0.3:5000 1,5,7
```

Como definido na especificação, o cliente recebe o par IP:porta do peer que servirá de ponto de contato, bem como uma lista de índices das chunks que ele irá requerer da rede. Por sua vez, o peer recebe o IP:porta que ele deve inicializar seu soquete UDP, um arquivo no qual há quais as chunks que ele possui, bem como seus nomes respectivos e, finalmente, uma lista de IP:porta dos outros peers dos quais ele é vizinho. Essa lista de vizinhos pode ser vazia.

2.2. Resumo do Projeto

O passo inicial do projeto foi adaptar o sistema básico feito no Trabalho Prático 2, tendo em vista o exposto em [Cunha 2020], para a realidade de um sistema peer-to-peer. Como recomendado pela especificação, o trabalho foi construído por partes. Inicialmente, as mensagens básicas do protocolo definido na especificação foram implementadas e testadas. Em seguida, a comunicação entre cliente e peer, bem como entre peers foi construída utilizando tais mensagens enviadas pelo soquete UDP. Finalmente, o envio efetivo das chunks foi implementado. Além disso, com intuito de assegurar o funcionamento correto, tal como indicado na especificação, testou-se o sistema com auxílio do Mininet.

3. Arquitetura

A seguir são discutidas as implementações feitas, bem como suas funcionalidades requeridas e soluções adotadas.

3.1. Protocolo e Mensagens

Todas as mensagens do protocolo descritas na especificação foram implementadas via funções auxiliares. Assim, os detalhes de como cada datagrama era composto eram abstraídos do cliente e do peer, de forma que estes apenas chamavam funções de *encode* e *decode* para cada tipo de mensagem para que uma função auxiliar, por exemplo, retornasse o datagrama montado e pronto para enviar (*encode*) ou então retornasse as informações importantes contidas naquele pacote (*decode*). Para manejar os bytes das mensagens, utilizou-se a estrutura *ByteArray*, além de diversas conversões entre tipos primitivos e bytes. Também foi importante usar a conversão entre IPv4 sua representação em bytes, com o auxílio da biblioteca *socket*.

3.2. Cliente

O Cliente é a entidade que de fato inicia o protocolo descrito na especificação. Assim, após processar os argumentos passados, sabe-se o endereço do peer de contato e a lista dos índices de chunks que necessitamos. Cria-se o soquete UDP e envia-se a mensagem hello para este peer, contendo essas chunks, no formato especificado. É importante ressaltar que, tal como as mensagens, essa formatação da lista dentro do datagrama foi abstraída do programa principal, sendo apenas uma função auxiliar que recebe uma lista e produz sua representação binária.

A partir disso, as mensagens chunks_info dos peers irão eventualmente chegar dos peers que possuem chunks de interesse para esse cliente e foram atingidos pelo alagamento iniciado pelo peer de contato.

Quando alguma mensagem de *chunk_info* chegar, o cliente precisa decidir se vai requisitar alguma chunk desse peer ou não. Para tanto, a heurística utilizada foi a seguinte: o peer cuja primeira *chunk_info* que chegar no cliente com alguma das chunks de interesse receberá a mensagem get. Note que caso essa *chunk_info* primeira possua mais de uma chunk que interesse, mais de um get será emitido para o mesmo peer, um para cada chunk. A decisão de enviar a requisição apenas de uma chunk por mensagem get foi uma decisão de projeto.

Cada chunk que chega é salva na pasta raiz do cliente, com o nome *chunk* seguido do índice dela. O formato do arquivo, como definido na especificação é o *m4s*. Convém ressaltar, como discutido nos fóruns da disciplina, que cada cliente é executado em uma pasta separado, logo não há risco de clientes diferentes utilizando a rede peer-to-peer misturarem as chunks que eles conseguiram.

Por fim, é importante destacar, também, que na especificação é descrito que é possível assumir que não há perda de pacotes nem quaisquer outros problemas de rede, logo, se após um certo tempo, o cliente ainda não recebeu algum *chunk_info* com alguma das chunks que ele precisa, é porque ou essa chunk não está disponível na rede ou essa chunk está em algum peer que não pôde ser alcançado com o alagamento de TTL apenas 3. Para lidar com essa situação, na criação do soquete UDP, foi definido um timeout de 5s de inatividade. Assim, após esse tempo, o soquete levanta uma exceção que é captada pelo programa do cliente, o qual então insere essas chunks faltantes no *output-IP.log* com o IP "0.0.0.0", como definido na especificação, e encerra sua execução.

3.3. Peer

O Peer foi concebido como uma entidade reativa, visto que toda mensagem que ele precisava enviar era causada pelo recebimento de uma outra. Isso aliado ao fato de que UDP é orientado a datagramas e que as operações de processamento de mensagens são praticamente instantâneos nos permite gerenciar várias mensagens advindas de clientes e peers diferentes sem utilizar o multithreading.

A execução do peer consiste inicialmente em processar os argumentos passados. Assim, obtém-se IP e porta locais daquele peer, um dicionário com o índice de cada chunk e seu respectivo arquivo e finalmente um outro com o IP e a porta de cada vizinho. Cria-se o soquete UDP as informações passadas e fica-se aguardando a chegada de alguma mensagem num loop infinito. Esse laço seria finalizado caso houvesse uma mensagem para encerrar a execução do peer, mas essa mensagem não compõe o protocolo nem está na especificação. Assim a única forma de encerrar um peer é usando o CTRL+C no terminal que o iniciou.

Neste loop que aguarda a vinda das mensagens, quando chega uma mensagem hello do cliente, o peer envia uma mensagem *chunk_info* de volta para o cliente, caso o peer possua alguma chunk de interesse para o cliente, e, em seguida, faz o alagamento para os seus vizinhos, espalhando mensagens *query* pela rede, inicialmente com TTL de 3, como definido pela especificação. Por fim, chegando um *get*, o peer carrega o arquivo referente à chunk requisitada e envia uma mensagem *response* para o cliente, com essa chunk.

Por fim, chegando uma mensagem *query*, temos que enviar uma *chunk_info* para o cliente, caso esse peer possua alguma chunk de interesse do cliente, e depois seguir com o alagamento, enviando *query* a seus vizinhos, mas com TTL uma unidade menor. Naturalmente, o peer não envia a *query* para aquele outro que lhe enviou a *query*, se este for vizinho do peer atual. É importante lembrar que caso o TTL chegue a zero, essa mensagem *query*, que continuaria o alagamento da rede, não será criada nem enviada.

4. Conclusão

A experiência de desenvolvimento deste último trabalho prático foi mais tranquila do que nos anteriores. Além de já estarmos acostumados com toda a lógica da programação em *sockets* e em redes de uma forma geral, já possuíamos muito código que poderia ser reaproveitado, com leves modificações, do TP2. Isso sem citar também todo o fluxo de trabalho já consolidado com a realização dos últimos dois trabalhos, que facilitou a implementação deste terceiro. É evidente que, ao fim deste ciclo de trabalhos práticos nesta disciplina de redes, estes não apenas contribuíram para a fixação de conceitos e conteúdos vistos em aula mas também para o engrandecimento das habilidades de todos nós como desenvolvedores, tendo em vista que, num mundo conectado e globalizado de hoje, o conhecimento de redes de computadores é imprescindível.

References

Cunha, I. (2020). Introdução à programação em redes. [1]. Playlist no Youtube, Acesso em: 04 jan 2021.

[1] `https://www.youtube.com/playlist?list=PLyrH0CFXIM5Wzmbv-lC-qvoBejsa803Qk`