TP2 - Engenharia de Serviços em Rede

Serviço Over-the-top para entrega de multimédia

João Vale PG55951 **Filipe Pereira PG55941** **João Lopes PG55964**

Introdução

Este trabalho é realizado no âmbito da unidade curricular de Engenharia de Serviços de Rede. Consiste em criar um protótipo de entrega de vídeo com requisitos de tempo real, a partir de um servidor de conteúdos para um conjunto de clientes, sendo que para tal devemos dividir a nossa topologia numa rede CDN, onde formamos uma rede *overlay* aplicacional e numa rede de acesso.

Arquitetura da Solução

Vamos começar por explicar os pontos principais da nossa arquitetura mais sucintamente, sem mencionar protocolos e tecnologias, sendo que entraremos em mais detalhe quando falarmos da implementação.

A nossa solução envolve a utização de um *bootstrapper* que é o próprio servidor de conteúdos. Este *bootstrapper* necessita de um ficheiro de configuração, construído manualmente, onde especificamos a rede *overlay*, ou seja, os endereços IP dos nós, os seus vizinhos, se é um ponto de presença e também quantos desses vizinhos se encontram entre o respetivo nó e o servidor.

Os nós da rede *overlay* começam por receber a sua lista de vizinhos do *bootstrapper* e depois conectam-se a esses vizinhos. O servidor também é considerado como um nó da rede *overlay*, pelo que também se conecta aos seus vizinhos.

Os clientes começam por receber a lista de vídeos disponíveis para *streaming* do servidor e depois enviam, também para o servidor, o nome do vídeo que pretendem ver, sendo que o servidor responde com o ponto de presença mais adequado ao cliente.

Os melhores caminhos para cada ponto de presença são calculados quando a rede *overlay* está completa, sendo calculado o *Round-Trip Time* de cada ligação entre nós da rede *overlay*, formando um grafo com pesos que o servidor utiliza para encontrar os melhores caminhos.

De modo a encontrar o melhor ponto de presença para um cliente, são feitas medições do *RTT* entre cada ponto de presença e o cliente.

As conexões de cada nó com os seus vizinhos são utilizadas para enviar informação sobre a transmissão de vídeo, de modo a cada nó saber se tem de enviar um certo vídeo, para onde o tem de enviar e de quem o vai receber.

Um cliente informa o ponto de presença de quem recebe vídeo que não quer continuar a ver a transmissão e para de aceitar os pacotes de vídeo, cabe depois ao ponto de presença verificar se pode parar de transmitir o vídeo e informar os nós anteriores a ele na rede *overlay*, se tal for necessário. Temos portanto, um corte de fluxos de transmissão de vídeo feito através de *backtracking*, com início no ponto de presença.

Especificações dos Protocolos

Utilizamos três protocolos para diferentes funções, sendo eles o **videoProtocol**, **controlProtocol** e **requestProtocol**.

O **videoProtocol** tem a seguinte definição:

```
videoProtocol(srcAddr, videoName, seqNumber, payload)
```

Este protocolo é enviado exclusivamente por **UDP**, contém um pedaço de vídeo, o endereço de origem do vídeo, o nome do vídeo e um número de sequência que permitiria avaliar perdas e envios fora de ordem de pacotes de vídeo.

Pacotes deste protocolo têm sempre origem no servidor de conteúdos e são enviados para nós da rede *overlay* através de caminhos específicos até chegarem ao cliente que quer assistir à respetiva transmissão.

O **requestProtocol** tem a seguinte definição:

```
requestProtocol(origin, payload, srcAddr="")
```

Este protocolo também é enviado exclusivamente por **UDP**, contém um campo que indica se o pacote originou de um cliente, nó da rede *overlay* ou do próprio servidor, de modo a permitir um encaminhamento correto de pacotes dentro de cada programa, visto que precisamos que cada *thread* em execução receba apenas os pacotes que necessita para a sua função. Contém também um campo com o endereço de origem e também um campo para eventuais dados que possam ser necessários dependendo do propósito para o qual este protocolo for utilizado.

É este o protocolo utilizado pelos nós da rede *overlay* para obter a sua lista de vizinhos, também é utilizado pelos clientes para comunicar com o servidor e pontos de presença e pelos pontos de presença para obter métricas de acesso a clientes.

O **controlProtocol** tem a seguinte definição:

```
controlProtocol(type, srcAddr, destAddr, payload = "")
```

Este protocolo é enviado exclusivamente por **TCP** através das conexões dos nós da rede *overlay* com os seus vizinhos. Contém os endereços de origem e destino, eventuais dados que possam ser necessários dependendo do contexto da utilização deste protocolo e um campo que indica o propósito do pacote.

Este protocolo é utilizado para confirmar a formação completa da rede *overlay*, obter métricas dentro da rede *overlay* e coordenar o envio de vídeo pelos nós *overlay*, sendo utilizado para informar os nós necessários sobre um novo cliente e tambem para informar que já não é necessário continuar o envio de pacotes de uma certa transmissão. Serve portanto como o suporte principal da nossa abordagem.

Implementação

Implementamos a nossa solução com *Python*, fazendo uso da biblioteca *socket* para programação com *sockets*. Para serialização e desserialização de mensagens usamos principalmente a biblioteca *pickle* e também a biblioteca *struct*. Utilizamos a biblioteca *threading* para criação de *threads* e gestão de acessos a pontos críticos do código e a biblioteca *queue* para obter uma estrutura de dados *thread-safe* que utilizamos para encaminhar pacotes para as *threads* corretas. Utilizamos também *ffmpeg*, *ffprobe* e *ffplay*, para converter, obter dados e visualizar vídeos. Para calcular os melhores caminhos utilizamos a biblioteca *networkx*.

Vamos agora especificar melhor alguns pontos que mencionamos na arquitetura da nossa solução.

4.1. Formação da rede *overlay*

Todos os nós *overlay* aguardam um pacote enviado por **UDP** do servidor onde se encontra a lista de vizinhos do nó, se o nó é um ponto de presença e quantos dos seus vizinhos estão entre o nó e o servidor. Todo este processo é sequencial, pelo que apenas depois do servidor enviar todas as listas de vizinhos é que inicia conexões **TCP** com os seus vizinhos. Os vizinhos do servidor, ao aceitarem a conexão do servidor, iniciam conexões **TCP** com os seus restantes vizinhos e assim sucessivamente. Ou seja, cada nó *overlay* sabe quantas conexões tem de aceitar, pois é o número de vizinhos que estão entre o nó e o servidor, e sabe portanto quantas tem de iniciar. Isto porque as conexões originam no servidor e são feitas em cadeia até aos pontos de presença, isto faz com que o servidor apenas inicie conexões e os pontos de presença apenas aceitem conexões, sendo apenas os nós intermédios a fazer os dois, aceitando conexões dos seus vizinhos que estão entre o próprio nó e o servidor e iniciando conexões com os restantes.

Quando um ponto de presença aceita as conexões de todos os seus vizinhos, ele envia um pacote **TCP** a informar que está conectado a todos os seus vizinhos, estes vizinhos adicionam o seu endereço IP a este pacote e reencaminham para os seus vizinhos que se encontram entre o próprio nó e o servidor, eventualmente todos este pacotes chegam ao servidor que necessita de obter todos os endereços IP dos nós da rede *overlay* pelo menos uma vez de modo a prosseguir com a execução do programa.

No final deste processo, existe uma rede baseada em conexões **TCP** montada entre os nós da rede *overlay*, sendo o servidor considerado um nó, em que cada nó tem várias *threads* responsáveis por comunicações com os seus vizinhos e cada nó sabe também quais dos seus vizinhos fazem parte de algum caminho que leva ao servidor.

4.2. Cálculo do melhor caminho até um cliente

Quando a rede *overlay* está completa, o servidor envia um pacote **TCP** a todos os seus vizinhos que enviam uma resposta de modo a ser possível medir o **RTT** da ligação com aquele vizinho. Tal como o inicio das conexões **TCP** entre vizinhos, a medição do **RTT** entre cada ligação é feita com início no servidor, em que cada nó ao receber um pacote para medição do **RTT**, para além de responder a esse pacote, envia também um pedido de medição da métrica aos seus vizinhos que não estão entre o próprio nó e o servidor, pois esses nós iniciaram o pedido para o nó atual.

Sempre que um nó termina uma medição, é enviado um pacote **TCP** para os seus vizinhos que levam ao servidor com o resultado da medição, de modo a atualizar o servidor com os dados de uma certa ligação. Os nós vão reencaminhando este tipo de pacotes até chegarem ao servidor.

Eventualmente, o servidor deverá receber um número de pacotes de atualização igual ao número de ligações na rede *overlay*, excluíndo aquelas em que ele está diretamente envolvido, pois o cálculo do **RTT** para essas ligações é feito diretamente pelo próprio servidor.

Com métricas para todas as ligações, o servidor cria um grafo com pesos e aplica o algoritmo de **Dijkstra** para descobrir o melhor caminho entre o próprio servidor e cada ponto de presença, sendo o melhor caminho aquele com o menor **RTT** total.

Quando um cliente pede para ver alguma transmissão, o servidor envia aos seus vizinhos um pedido para obter o **RTT** entre os pontos de presença e o respetivo cliente. Os nós *overlay* reencaminham este pacote até chegar aos ponto de presença que enviam um pacote **UDP** ao cliente e aguardam a resposta para medir o **RTT**. Os pontos de presença de seguinda enviam os resultados aos seus vizinhos que reencaminham os pacotes até ao servidor.

Depois de receber as medições de todos os pontos de presença, o servidor soma o **RTT** de cada ponto de presença ao cliente com o **RTT** total do melhor caminho até esse ponto de presença e escolhe o menor resultado como caminho até ao cliente.

4.3. Gestão e Reprodução de Vídeos

Decidimos utilizar o formato **MPEG-TS** para os vídeos que o servidor de conteúdos disponibiliza, devido a ser um formato propício a envio na rede, já que organiza o vídeo em pacotes de 188 *bytes* que podem ser enviados facilmente.

Utilizamos *ffmpeg* para converter os vídeos para **MPEG-TS** e *ffprobe* para calcular o atraso entre o envio de pacotes de vídeo de modo a manter uma velocidade adequada de visualização. Este atraso é calculado tendo em conta a *bitrate* do vídeo em questão e o tamanho de blocos de vídeo que decidimos enviar. No nosso caso enviamos blocos de 940 *bytes*, ou seja, cinco pacotes **MPEG-TS** de 188 *bytes*, pelo que ao dividir o tamanho de um bloco pela *bitrate* do vídeo em *bytes*, obtemos o intervalo de tempo que cada envio tem de cumprir para no fim de um segundo termos a *bitrate* adequada.

Um cliente inicia um processo de *ffplay* para visualizar uma transmissão, sendo que o *ffplay* recebe blocos de vídeo no *stdin* do seu processo através de um *pipe*, o processo principal vai recebendo pacotes de vídeo e enviando por esse *pipe* para o *ffplay* reproduzir o vídeo.

4.4. Envio de Vídeo

Um cliente começa por pedir a lista de transmissões disponíveis ao servidor, também pede para ver uma certa transmissão e recebe respostas diretamente do servidor, com a lista de vídeos e o ponto de presença que vai enviar a transmissão. Todas estas comunicações são feitas através de **UDP**.

Tendo o servidor calculado o melhor caminho até ao cliente, inicia a leitura do ficheiro **MPEG-TS** correspondente ao vídeo e envia pacotes **UDP** ao vizinho que se encontra nesse melhor caminho. Também utiliza a conexão **TCP** com esse vizinho para o informar que deve aceitar pacotes do vídeo

e deve encaminhar esses pacotes para o próximo destino no caminho, que também foi enviado no pacote **TCP**. Os nós que recebem um pedido de reencaminho de vídeo, para além de reencaminharem os pacotes **UDP** para o próximo destino no caminho, devem também enviar o pacote **TCP** que originou no servidor para o próximo nó do caminho, para no final todo os nós no melhor caminho até um cliente estarem informados de que têm de reencaminhar essa transmissão.

Cada nó da rede *overlay* guarda a informação de que vídeos está a receber, para onde os tem de enviar e de qual nó está a receber o vídeo.

Evitamos envios repetidos de pacotes de vídeo para o mesmo destino ao utilizar um *set* para guardar os vizinhos a quem enviar um certo vídeo, a natureza desta estrutura de dados impossibilita elementos repetidos, garantido que mesmo que um nó receba dois pedidos para reencaminho do mesmo vídeo, se o próximo elemento no caminho for o mesmo, ele apenas vai enviar uma vez os pacotes. Um pacote é enviado várias vezes somente quando um nó tem vários destinos diferentes para o mesmo vídeo.

O cliente envia um pacote **UDP** ao ponto de presença para parar de assistir a uma transmissão, isto faz com que o ponto de presença remova o endereço do cliente dos destinos de um certo vídeo. Se o ponto de presença não tiver mais destinos para um certo vídeo, informa o vizinho de quem está a receber tal vídeo por **TCP**, o nó remove o endereço do ponto de presença e repete o processo, verificando se ainda tem destinos para esse vídeo e caso não tenha informa o nó de quem está a receber tal vídeo. Ou seja, o corte de fluxos começa no ponto de presença e vai percorrendo o caminho do vídeo até ao servidor até que algum nó *overlay* continue a necessitar do vídeo, pelo que esse nó já não informa o nó de quem recebe o vídeo para parar de transmitir.

É de notar que uma transmissão termina quando não tem mais clientes a assistir ou quando o servidor termina a leitura do ficheiro e que quando um cliente quer ver uma transmissão que está a decorrer, vai começar a ver a partir do ponto em que o servidor se encontra e não do início.

Testes e Resultados

A nossa solução é capaz de transmitir vários vídeos para diferentes clientes pelos melhores caminhos até esses clientes. No entanto, ao testar deparamo-nos com alguns erros raros, tal como o vídeo começar com uma velocidade diferente do esperado para depois regularizar e continuar normalmente. Contudo, na maioria dos testes que fizemos, a solução funcionou corretamente.

É de notar que a nossa topologia possui atrasos na rede mas não possui perdas, isto porque não chegamos a conseguir implementar um método para lidar com perdas de pacotes **UDP**. Para pacotes de vídeo perdas não são um problema mas seriam críticas para comunicações entre clientes e a rede *overlay*.

Conclusões e Trabalho futuro

Concluído o projeto, acreditamos que conseguimos uma solução que cumpre com o material lecionado na unidade curricular. Existem no entanto vários aspetos a melhorar, tal como, implementar um método para lidar com perdas de pacotes **UDP**, lidar com recuperação de falhas, tanto da morte de um nó overlay como da desconexão inesperada de um cliente e também um método de obtenção de métricas mais fiável que é executado mais do que uma vez.