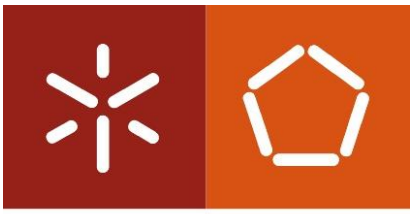

Universidade do Minho

Mestrado em Engenharia Informática

Engenharia de Serviços em Rede

TP2 - Serviço *Over the Top* para entrega de multimédia

PL14

Gonçalo Pereira (PG53834) Paulo Oliveira (PG54133)

Santiago Domingues (PG54225)

Ano Letivo 2023/2024

Índice

Capítulo 1

Introdução

O presente relatório visa apresentar o trabalho prático nº2 da Unidade Curricular de Engenharia de Serviços em Rede.

Neste projeto pretende-se conceber um **protótipo de entrega de áudio/vídeo/texto com requisitos de tempo real**, a partir de um servidor de conteúdos para um conjunto de N clientes, usando o emulador *CORE* como bancada de teste. A abordagem para a implementação deste serviço passa pela eleição de um *Rendezvous Point* (*RP*), sendo este responsável por receber em *unicast* o conteúdo a distribuir, propagando-o por um conjunto de nodos que atuam como intermediários, formando entre si uma árvore de distribuição partilhada, cuja criação e manutenção deve estar otimizada para a missão de entregar os conteúdos de forma mais eficiente, com o menor atraso e a largura de banda necessária.

Sabe-se que se num serviço deste tipo, não existindo *overlay* aplicacional, cada cliente tem de abrir uma conexão para um dos servidores, e esse servidor tem de gerar um fluxo de dados por cada cliente. Esta solução tem problemas óbvios de escalabilidade, pois a partir de um determinado número de clientes, o servidor e/ou a rede deixam de conseguir suportar os fluxos sem perda de qualidade. Uma solução mais eficiente pode ser conseguida adicionando nodos de uma rede *overlay* construída sobre a rede *underlay* de suporte. Assim sendo, as ligações na rede *overlay* são na realidade conexões de transporte (TCP ou UDP), que atravessam uma ou mais ligações na rede física. Para visualizar um certo conteúdo, o cliente deverá enviar um pedido ao vizinho mais próximo, e este pedido será propagado pela árvore até alcançar um nodo que já esteja a difundir este conteúdo, ou, em último caso, o *RP*. Ao receber um pedido, o *RP* deverá avaliar as condições de cada um dos servidores que possua o conteúdo, e selecionar o mais indicado.

Este documento contém a explicação detalhada de todo o processo de desenvolvimento do projeto, desde decisões à implementação, mostrando também os testes realizados e os respetivos resultados, finalizando com as conclusões acerca do mesmo.

Capítulo 2

Arquitetura da Solução

No que toca à arquitetura da solução desenvolvida, como descrito no enunciado e também abordado na introdução, o sistema é composto por nodos que podem ser de quatro tipos: servidor, cliente, nodo intermediário ou *RP* (apenas um). Numa primeira fase, foi necessário decidir a forma de como cada nodo sabe quais os nodos da rede aos quais se tem de ligar (os seu vizinhos). Então, o grupo de trabalho optou por utilizar ficheiros de configuração. Para uma perceção mais intuitiva, estes ficheiro são do formato **JSON** e a sua estrutura é a seguinte:

```
{
    "host": [
      "ip_nodo:porta"
      (...)
    ],
    "nb": [
      "ip_vizinho1:porta_vizinho1"
      (...)
    ]
  }
```

Em relação à comunicação entre os nodos, a seleção do **UDP** (*User Datagram Protocol*) foi feita com base naquilo que consideramos o mais apropriado. Obviamente, não se trataria de uma decisão se todas as opções envolvidas não tivessem os seus prós e contras, e apesar do protocolo escolhido não ser confiável, ou seja, não garante que os pacotes chegam ao seu destino, contrariamente ao TCP (por exemplo, que se trata de um protocolo orientado à conexão), o grupo priorizou a baixa latência que o UDP oferece, não dando grande importância à perda de pacotes, se e só se esta acontecer de forma muito ocasional. Ou seja, assumindo que se está a trabalhar sobre uma rede estável, a eficiência e simplicidade deste protocolo pareceu a melhor opção.

Assim, ao iniciar o programa após a ativação das conexões entre os nodos, um pacote do tipo *init* é enviado por todos os nodos para os seus vizinhos, assegurando, dessa forma, que a topologia está operacional. No que diz respeito às comunicações que envolvem pedido/envio do vídeo da *stream*, a lógica por detrás destes é a lógica referida na introdução: quando um cliente pretende ver o vídeo, o seu pedido percorre a árvore até encontrar um nodo que já esteja a difundir o vídeo. Em último caso, se nenhum destes nodos intermédios estiver a difundir o vídeo, o pedido chega ao *RP*, e este, uma vez que comunica com o servidor, trata de receber o vídeo e enviá-lo. No capítulo 3 são abordadas de uma forma bastante mais aprofundada estas comunicações entre nodos, na secção das interações.

Capítulo 3

Especificação do(s) protocolo(s)

3.1 Formato das mensagens protocolares

O formato das mensagens protocolares definido pelo grupo está apresentado na tabela que se segue.

Campo	**Descrição**
`id`	Identificador do pacote
`source`	Fonte do pacote
`dateTime`	Tempo de criação do pacote
`jumps`	Número de saltos
`filename`	Nome do vídeo
`currentHop`	Salto atual
`streamID`	Identificador da *stream*
`streamPort`	Porta da *stream*
`path`	Caminho do pacote
`ips`	Lista de IPs dos nodos do caminho
`lock`	Objeto para controlo de concorrência

Tabela 3.1: Formato dos Pacotes

Em relação ao identificador do pacote, os tipos de identificador são:

- **init** - Confirmação inicial de conexão
- **LR** e **LA** - *Latency Request* e *Latency Answer*
- **VR** - *Video Request*
- **SP** e **ACK_SP** - *Setup Request* e *Setup Acknowledgment*
- **PLAY**, **PLAY_** e **ACK_PLAY** - *Play Request*, *Play Again Request* e *Play Acknowledgment*
- **PAUSE** - *Pause Request*
- **TEARDOWN** - *Teardown Request*

3.2 Interações

Conectividade

De forma a tratar da conexão entre os nodos no momento do início do programa, é feito o *parse* dos mencionados ficheiros de configuração de cada um dos nodos. Uma vez criados os *sockets*, é enviado um pacote do tipo *init* por cada nodo para todos os seus vizinhos.

Latência

Para obter a latência de determinado caminho, de forma a saber qual a melhor opção para pedir o vídeo, o grupo desenvolveu o seguinte algoritmo: primeiro, é criado um pacote com o ID **LR** (explicado acima) pelo *RP*, e enviado para todos os seus vizinhos. De seguida, cada nodo que recebe esse pacote verifica se ele próprio é um servidor ou não. No caso de não ser servidor, envia novamente para todos os seus vizinhos, exceto àquele que lhe enviou o pacote, e assim sucessivamente. No caso do nodo que recebe ser um servidor, este muda o ID do pacote para **LA**, e envia-o de volta, percorrendo o caminho inverso. O pacote contém este caminho (*path*) pois ao longo do *flood*, os nodos em que ele passou foram adicionados ao mesmo. No caso do pacote nunca chegar a um servidor, vai chegar a um cliente, uma vez que estas são as duas extremidades possíveis na topologia. Se isto acontecer, o cliente apenas ignora o pedido.

A latência é depois guardada numa lista de forma ordenada, juntamente com o IP do servidor em questão e o caminho percorrido (lista dos IPs)

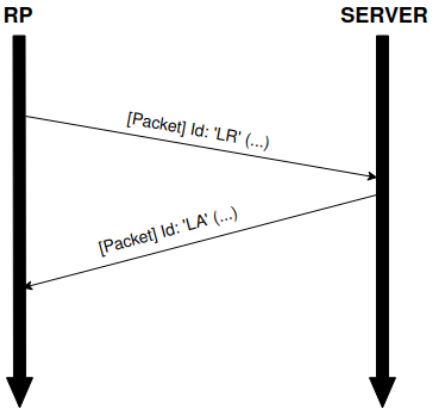

Figura 1: Diagrama de troca de mensagens para o teste de latência.

Streaming

Relativamente ao *streaming*, a troca de mensagens neste processo é a seguinte: Quando um cliente se conecta, sendo este o primeiro cliente a pedir *stream*, envia um pacote de id **VR** (*Video Request*) para o *RP*. De seguida, o *RP* verifica se é o primeiro pedido desse cliente e, se for, acede ao dicionário de forma a saber qual o caminho com latência mais baixa, enviando para esse caminho um pacote do tipo **SR** (*Setup Request*). Quando este pacote chega ao servidor, o servidor envia de volta ao *RP* um **ACK_SP** (*Setup Acknowledgment*). É importante notar que quando este *acknowledgment* percorre o caminho desde o servidor até ao *RP*, para cada nodo que passa é criado um *sender* e um *receiver*. Estes servirão para o encaminhamento dos pacotes do vídeo. Uma vez recebido o **ACK_SP** por parte do *RP*, são enviados de volta um **PLAY** para o servidor e um **ACK_PLAY** para o cliente que efetuou o pedido do vídeo. Se o cliente que se conectar para pedir o vídeo não for o primeiro, e se antes do seu pedido chegar ao *RP* passar por um nodo que já esteja a difundir o vídeo, vai ser deste nodo que o cliente vai consumir.

A partir deste momento, o nodo vizinho do cliente (vizinho esse que recebeu os pacotes do vídeo do servidor, passando pelo *RP*) que solicitou o vídeo está a enviar os pacotes do vídeo e surge no ecrã a janela de transmissão do vídeo porém, como o cliente ainda não selecionou o botão 'Play', ainda não está a receber. A partir do momento que o cliente selecionar o 'Play' da janela, está à escuta e começa a receber os pacotes do vídeo, transmitindo-o.

O diagrama que se segue demonstra a troca de mensagens/pacotes entre os diferentes componentes da topologia quando é feito um pedido de um vídeo por parte do cliente.

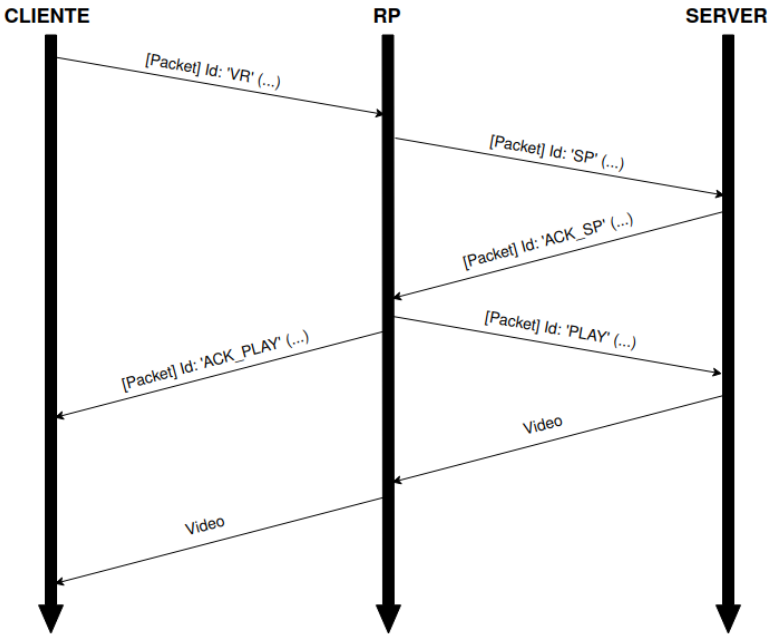

Figura 2: Diagrama de troca de mensagens para a *stream*.

Capítulo 4

Implementação

4.1 Detalhes, parâmetros, bibliotecas de funções, etc.

Como referido no capítulo 2, o grupo decidiu criar ficheiros de configuração para os nodos da rede terem forma de saber quem são os seus vizinhos. Estes ficheiros têm a extensão **JSON** e estão na diretoria *config*. Assim, quando o programa inicia, é feito o *parse* do ficheiro por cada nodo e a informação é guardada num dicionário. Desta forma, para estabelecer a conexão com os restantes, o nodo em questão executa a ação *host* nos IPs da lista *host*, e executa *connect* nos IPs da lista *nb* (o nome *nb* surge como abreviatura da palavra inglesa para vizinhos - *neighbors*). Para passar o ficheiro de configuração ao programa, o utilizador insere o sue caminho como argumento ao executar o programa.

Sendo assim, o comando para iniciar cada nodo da rede é:

```
python3 oNode.py ../config/<nome_nodo>.json
```

4.1.1 Classes

Nesta secção será feita uma breve abordagem acerca das classes mais importantes do código. O grupo criou classes de modo a ter uma boa modularidade, sendo assim mais fácil a criação de novas funcionalidades e a sua manutenção. Todos os objetos aqui apresentados são classes, à exceção do primeiro.

oNode

O ficheiro *oNode* é o único nesta secção que não é uma classe, porém é também importante abordá-lo. Este que contém a função *run*, servindo esta como uma espécie de função *main*. Quando o programa começa a sua execução, este executa o *parse* dos ficheiros de configuração e trata de chamar o teste de latência no caso do *RP*, ou o pedido de vídeo no caso do servidor. No *parse*, para cada *host* é criado um *UDPreceiver*, e para cada *nb* um *UDPsender*.

UDPsender e UDPreceiver

UDPsender é a classe que funciona como o remetente das mensagens. Recebe na sua criação um *socket* (para o envio da mensagem/pacote), o seu nome (ex: *s1* se for o servidor), a lista dos seus próprios IPS (com os quais executa *host*) e é chamada para todos os IPs dos vizinhos no ficheiro de configuração (pois

são os IPs com os quais vai executar *connect*), no momento em que acontece o *parse*, recebendo também esse mesmo IP do vizinho. Ao contrário desta, a classe *UDPreceiver* funciona como o destinatário das mensagens. Esta cria também um *socket* (para receber a mensagem/pacote) e tal como a anterior recebe também a lista dos próprios IPS (os *host IPs*) e o seu nome do nodo. Recebe também a lista dos *senders*, ou seja, a lista dos remetentes, que são criados anteriormente utilizando a classe acima. Uma vez iniciadas as classes *UDPsender* e *UDPreceiver* para cada par de nodos ligados entre si, fica então possível a comunicação, pois a sua ligação fica ativada.

Packet

Como o próprio nome indica, esta classe representa os pacotes, sendo que os seus atributos já foram explicados no capítulo 3. Aqui estão também presentes as funções de *encode* e *decode*, de forma a poder enviá-los no formato correto.

packetHandler

Aqui é feito todo o tratamento dos pedidos, ou seja, estão presentes as funções que implementam os algoritmos de envio/receção de pacotes. Quando o ficheiro *oNode* "chama"o teste de latência ou o pedido de vídeo, por exemplo, estas funções estão presentes nesta classe, entre muitas outras mais, representantes dos outros tipos de pacotes.

Cache

A classe *Cache* funciona como uma espécie de armazenamento dos dados. Esta foi a decisão que o grupo decidiu tomar para guardar os dados que têm de ser partilhados entre as instâncias ou *threads*, para cada nodo. São guardados por exemplo, os pedidos de vídeo, os caminhos, etc.

unicastSender e unicastReceiver

Estas duas funcionam de uma forma bastante semelhante às classes *UDPsender* e *UDPreceiver*, porém apenas para a transmissão dos pacotes do vídeo, e não dos restantes pacotes. Quando o servidor começa a enviar os pacotes do vídeo, o nodo seguinte que o recebe cria um *unicastReceiver* para receber esses pacotes e um *unicastSender* para os enviar para o próximo, e assim sucessivamente até os pacotes chegarem ao cliente que os pediu.

Outras classes

Por uma questão de objetividade no presente documento, as outras classes não serão abordadas da mesma forma que as anteriores, porém não são menos importantes. Entre elas estão principalmente as fornecidas pelos docentes para a transmissão do vídeo, que foram adaptadas ao código do grupo, e também o *Parser*.

Capítulo 5

Limitações da Solução

Após a análise das etapas propostas pelos docentes no enunciado, torna-se necessário abordar de forma crítica as limitações que o projeto desenvolvido contém. Este capítulo visa identificar e analisar as limitações que surgiram durante a realização projeto. A apresentação destas limitações mostra também que algumas delas podem também ser encaixadas no tópico dos trabalhos futuros.

Em primeiro lugar, uma funcionalidade crucial que o grupo não conseguiu implementar devido a restrições temporais foi o *multicast*. No serviço implementado pelo grupo, quando um cliente quer assistir a uma *stream*, o vídeo começa sempre do início. Outra limitação é o facto de não ter sido feita uma definição de um método de recuperação de falhas e entradas de nodos adicionais. Uma vez que o grupo se focou principalmente nas cinco primeiras etapas, deixou um pouco de parte a etapa adicional e, tal como o *multicast*, as restrições temporais não permitiram a realização desta etapa.

Capítulo 6

Testes e resultados

De forma a testar a solução, o grupo utilizou a seguinte topologia:

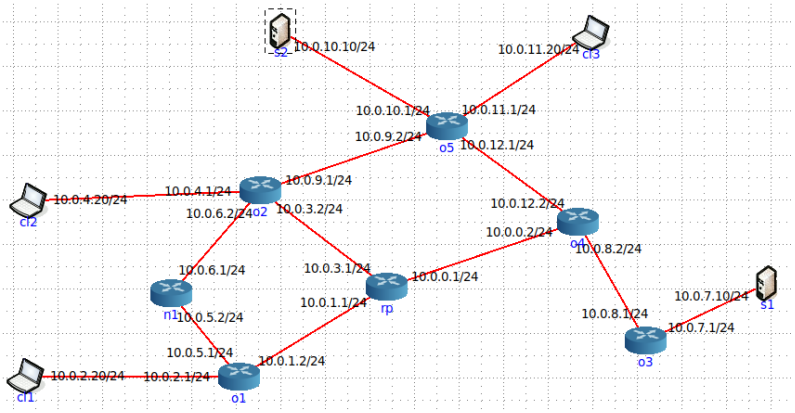

Figura 3: Topologia de teste.

Como é possível observar, a topologia apresenta dois servidores (s1 e s2), três clientes (cl1, cl2 e cl3) e sete nodos intermédios, sendo apenas um de *underlay* e os restantes seis de *overlay*. A figura seguinte apresenta os vídeos dos três clientes em reprodução, juntamente com as linhas de comando de todos os nodos.

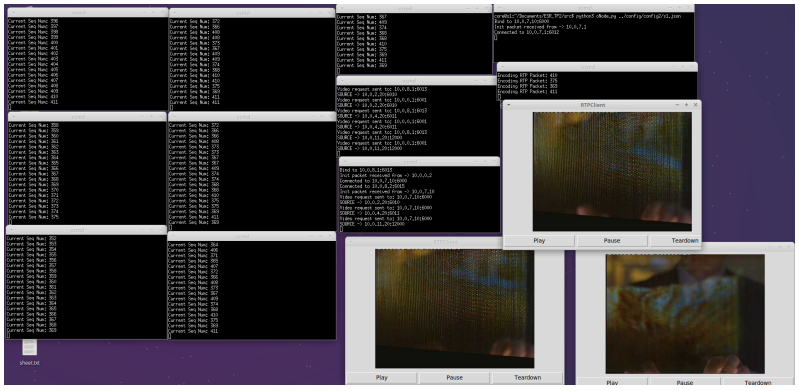

Figura 4: Topologia de teste.

Capítulo 7

Conclusões e trabalho futuro

Com a realização deste projeto, exploramos de forma minuciosa aquilo que é a base de um serviço de *streaming*. Foi bastante importante passar pelas dificuldades encontradas, mais concretamente no controlo dos caminhos dos pacotes (servindo este como exemplo por ser o mais ocorrente), na medida em que fizeram com que o grupo analisasse com espírito crítico, a um nível mais geral, as várias opções relativas a arquiteturas e, dentro da arquitetura, a forma de como passar informação de um nodos para os outros, mostrando-se este como um dos maiores desafios. Foi também muito útil para aplicar os conceitos de *unicast* e *multicast* abordados nas aulas desta unidade curricular, assim como perceber que a gestão de um sistema deste nível se pode tornar complicada devido ao nível de complexidade do código e dos algoritmos e que, se isto acontece a relativamente baixa escala (que é o que está a ser utilizado neste projeto), é possível perceber como pode ser com a escala das grandes aplicações que contem milhões de utilizadores.

No que toca às ferramentas utilizadas, o grupo já tinha tido contacto com o emulador CORE em unidades curriculares anteriores, também, obviamente, dentro do tema das redes de computadores, servindo este projeto para consolidar a experiência com o mesmo. O IDE utilizado para a produção do código em *python* foi o *Visual Studio Code*, sendo este também já bastante familiar para o grupo.

Concluindo, o grupo considera que concluiu com sucesso os principais desafios do projeto, mostrando-se apenas descontente devido à falta da implementação do *multicast*, pois sabe-se que é uma característica crucial num serviço deste tipo.

Para trabalho futuro, a prioridade é, sem dúvida, a implementação do *multicast*, sendo uma função crucial num serviço de *streaming*. O grupo também considera interessante a implementação da possibilidade de ser possível fazer o *stream* de vídeos com outros formatos (e não só em MJPEG), uma vez que esta funcionalidade torna o serviço bastante mais abrangente (em termos de quantidade de vídeos disponíveis para *stream*). Outro objetivo é também melhorar a parte estética. Para isto, alterar a janela *pop-up* de modo a ficar mais apelativa e com mais funcionalidades disponíveis. Por último, era também interessante encontrar uma solução de forma a automatizar parcial ou totalmente o processo de colocar o programa em execução, uma vez que para o fazer neste momento é necessário abrir as linhas de comando de cada um dos nodos da rede *overlay*, e colocar um conjunto de nodos. Este método torna o *setup* da execução do programa lento e fastidioso.