

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

Miguel Pinto, Orlando Palmeira, and Pedro Martins

¹ Universidade do Minho, 4710-057 Braga, Portugal

² Engenharia de Serviços em Rede

³ {pg54105, pg54123, pg54146}@alunos.uminho.pt

1 Introdução

Este relatório descreve o desenvolvimento de um serviço *Over the Top* (OTT) para entrega de conteúdos multimédia, no âmbito da unidade curricular de Engenharia de Serviços em Rede. O objectivo do trabalho consiste em conceber e prototipar um serviço de entrega de áudio/vídeo/texto em tempo real, utilizando um *Rendezvous Point* (RP) e uma rede *overlay*.

O trabalho envolveu a construção da topologia *overlay*, implementação do serviço de *streaming*, monitorização dos servidores de conteúdos, construção dos fluxos para entrega de dados e adicionalmente, a definição de uma estratégia de recuperação de falhas e entradas de nós adicionais.

O relatório apresenta a descrição de cada etapa de desenvolvimento, incluindo a especificação conceptual, implementação, testes, resultados, conclusões e trabalho futuro.

2 Arquitetura da solução

Neste trabalho pretende-se criar um sistema de **entrega multimédia em tempo real**, a partir de um **servidor de streaming** para um conjunto de **clientes**. Para isso, será escolhido um **Rendezvous-Point (RP)**, encarregado de receber o conteúdo proveniente dos servidores de *streaming* via *unicast* e propagando-o por um conjunto de nós 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 largura de banda necessária. A seguinte figura demonstra uma visão geral de como é que os componentes se posicionam perante este sistema de entrega multimédia.

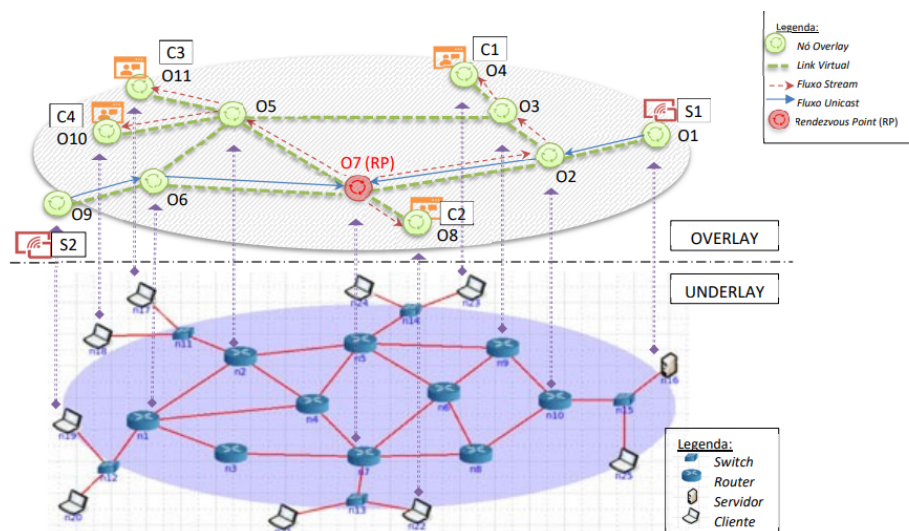


Figura 1 - Visão geral de um serviço OTT sobre uma infraestrutura

Figura 1. Arquitetura do sistema

Para a implementação do trabalho prático recorreremos à linguagem de programação **Python**, visto que é uma linguagem que proporciona um bom nível de abstração e fornece uma ampla gama de bibliotecas que facilitam a manipulação de *sockets* e *threads*, sendo também a linguagem que apresenta um maior grau de familiaridade e conforto aos membros da equipa.

A estratégia que decidimos implementar para a **construção da rede *overlay*** passa pela utilização de um **ficheiro de configuração *JSON*** que irá conter a informação devida ao respetivo componente. Nas seguintes secções apresentamos os argumentos necessários a cada componente, bem como a estrutura dos ficheiros de configuração esperada.

2.1 *oNode*

Quando um nó (*oNode*) da rede *overlay* começa a sua execução, é preciso que ele saiba quais são os seus vizinhos e essa informação estará presente num ficheiro de configuração que contém uma lista dos endereços dos seus vizinhos. O componente pode ser invocado através do seguinte comando `python3 onode.py <config_file>`.



```
{
  "vizinhos": [
    "10.0.14.1",
    "10.0.20.2",
    "10.0.11.1"
  ]
}
```

Figura 2. Exemplo de ficheiro de configuração de um *oNode*

2.2 *RP*

O nodo **RP** (*Rendezvous Point*), terá que ser inicializado com uma lista dos endereços dos *servidores de streaming* que serão a fonte original dos conteúdos que serão transmitidos pela rede *overlay*. O componente pode ser executado através do seguinte comando: `python3 rp.py <config_file>`

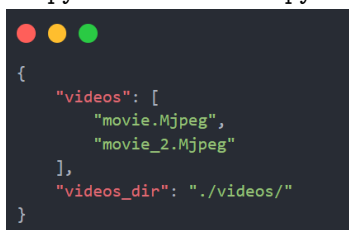


```
{
  "servidores": [
    "10.0.0.10",
    "10.0.2.10"
  ]
}
```

Figura 3. Exemplo de ficheiro de configuração do *RP*

2.3 *Servidores de Streaming*

Os **servidores de streaming** terão de receber como input um ficheiro de configuração que lista os vídeos que tem disponíveis, e a respetiva localização dos ficheiros multimédia. A indicação do campo "videos_dir" é opcional, sendo assumido o valor padrão como `./videos/`. O componente pode ser invocado através do seguinte comando `python3 servidor.py <config_file>`



```
{
  "videos": [
    "movie.Mjpeg",
    "movie_2.Mjpeg"
  ],
  "videos_dir": "./videos/"
}
```

Figura 4. Exemplo de ficheiro de configuração do *Servidor*

2.4 Cliente

Um **cliente** quando começa o seu arranque precisa de saber qual o seu nodo de acesso à topologia e o filme que está a requisitar. A definição do seu vizinho é feita através do ficheiro de configuração e a indicação do filme que pretende visualizar é indicada através da linha de comando da seguinte forma, `python3 cliente.py <config_file> <video_name>`.



Figura 5. Exemplo de ficheiro de configuração de um *Cliente*

3 Especificação dos protocolos

A escolha do **protocolo aplicacional para o streaming** foi o **RTP** (Real-time Transport Protocol), aproveitando uma implementação que adaptamos, fornecida pelos docentes. Este protocolo permite o envio de mensagens utilizando o protocolo de transporte **UDP**, que oferece **simplicidade e baixa latência**, essencial para transmissões em tempo real. Além disso, é ideal para cenários em que há **tolerância a perdas de pacotes**, proporcionando uma abordagem eficaz para a natureza do nosso projeto.

A transmissão dos conteúdos para os vários clientes, através dos diversos componentes da nossa infraestrutura, exigiu a concepção de um **protocolo aplicacional de controlo** que permitisse a gestão do envio dos conteúdos pela rede. Dada a natureza do projeto, consideramos que a escolha mais apropriado seria utilizar o **protocolo de transporte UDP** para a implementação deste protocolo aplicacional de controlo, alinhando-se com a decisão anterior de optar por **RTP** que usa **UDP** devido à sua simplicidade e baixa latência.

Nesse sentido, o protocolo base que sustenta a funcionalidade primordial (transmissão de conteúdos) do nosso serviço é composto por três passos principais:

1. Adquirir informações acerca dos locais que detêm o conteúdo desejado.
2. Efetuar a transmissão do conteúdo.
3. Concluir a transmissão do conteúdo.

Além do protocolo encarregue da transmissão de conteúdos, o nosso serviço incorpora outros protocolos responsáveis pela monitorização dos servidores de conteúdos, tratamento de erros e falhas, assim como a gestão de alterações na topologia da rede, tanto na entrada como na saída de nós.

3.1 Formato das mensagens protocolares

As mensagens envolvidas na comunicação entre os diversos componentes possuem os seguintes campos:

- Tipo de mensagem (identificado por um número inteiro):
 - **START_VIDEO** (1): Solicitação de inicialização de *streaming* de conteúdos
 - **METRICA** (2): Utilizado em pacotes prova trocados entre o RP e os servidores, permitindo ao RP obter informações das condições de serviço de cada um dos servidores de conteúdos.
 - **STOP_VIDEO** (3): Solicitação de paragem de consumo de conteúdos.
 - **CHECK_VIDEO** (4): Pedido de informações acerca da localização de conteúdos.
 - **RESP_CHECK_VIDEO** (5): Resposta a um pedido de localização de conteúdos.
 - **ADD_VIZINHO** (6): Para um nó indicar a um componente adjacente a sua entrada na rede *overlay*.
 - **RMV_VIZINHO** (7): Para um nó indicar a um componente adjacente a sua saída da rede *overlay*.
 - **ALIVE_RECEPTOR** (8): Utilizado para um nó que está a receber conteúdo, avisar os seus fornecedores que ainda se encontra ativo.

- **id**: Número inteiro aleatório (entre 0 e 10^6) que serve para identificar a mensagem enviada. Este identificador é utilizado em conjunto com o ip do cliente (campo origem) essencialmente no momento de recepção de pedidos `CHECK_VIDEO` em que o RP ou um nodo intermédio verifica se já respondeu à mensagem que recebeu.
- **origem**: *String* com o endereço IP do remetente original da mensagem. É utilizado essencialmente para informar a proveniência de um pedido de um cliente (quando envia um `CHECK_VIDEO`), bem como a localização de um vídeo na rede (quando envia um `RESP_CHECK_VIDEO`).
- **timestamp**: Marca o instante em que um componente envia a mensagem. É essencialmente utilizado nas comunicações entre o **RP** e os **Servidores** para o cálculo da métrica.
- **dados**: Este campo permite colocar qualquer tipo de dados que possam ser trocados entre cada componente. Na explicação das interações iremos descrever quais os dados colocados neste campo em cada tipo de interação estabelecida entre os diversos componentes.

3.2 Interações

Nesta secção, apresentaremos vários exemplos de interações entre os diversos componentes do sistema, inerentes à realização das suas funcionalidades.

3.2.1 Monitorização dos Servidores de conteúdos

A monitorização dos servidores de conteúdo tem como finalidade permitir obter um conhecimento das condições de entrega de conteúdo de cada um dos servidores disponíveis. Com esta informação o RP conseguirá determinar qual dos servidores será a melhor escolha para pedir um determinado vídeo.

De modo a obter esta informação, o RP quando arranca pede informação acerca dos vídeos que cada servidor tem, através do envio de uma mensagem `CHECK_VIDEO`. O servidor responderá com uma lista dos vídeos no campo *dados* da mensagem.

Para além, da informação dos vídeos que os servidores tem disponível, o RP irá fazer uma análise às condições de serviço que os servidores proporcionam através do envio de 10 mensagens do tipo `METRICA`, sem nenhum dado adicional, e que os servidores irão responder com uma timestamp imediatamente antes de enviar a resposta.

Para cada servidor, o RP conta o número de respostas que retornaram calculando uma percentagem de retorno bem-sucedido (*sucesses*), e analisa o tempo que demorou a chegar fazendo a diferença do tempo em que recebeu a mensagem com o tempo que vem na timestamp da mensagem, obtendo uma média dos valores que calculou vindo de cada pacote recebido (*avg_delivery_time*). Com estes dois valores criamos uma métrica que junta os dois valores através da seguinte fórmula:

$$\text{final_metric} = 0.5 * (1 / \text{avg_delivery_time}) + 0.5 * (\text{sucesses}/\text{num_of_requests})$$

Através desta métrica, podemos inferir o grau de qualidade das condições que um servidor tem para oferecer, permitindo fazer de forma mais informada e determinística, a escolha do servidor para o qual pedir o *streaming* de um vídeo.

3.2.2 Pedido da localização de um vídeo

O pedido de localização de um vídeo (`CHECK_VIDEO`) ocorre quando um cliente deseja visualizar um vídeo e solicita ao seu nó adjacente que procure a localização desse vídeo na rede. Quando o nó adjacente recebe essa solicitação, podem ocorrer duas situações: (i) O nó adjacente já está a transmitir o conteúdo pretendido pelo cliente, e neste caso, ele responde directamente ao cliente, indicando que possui o conteúdo; ou (ii) o nó adjacente envia aos seus nós vizinhos a solicitação do cliente para que estes procurem o vídeo na rede ou indiquem se já o possuem.

Nesta interacção, um mesmo nó pode receber o mesmo pedido (`CHECK_VIDEO`) mais do que uma vez devido à replicação desse pedido nos nós que não possuem o vídeo pretendido. Portanto, para prevenir este problema de repetições, recorre-se aos campos *id* e *origem* da mensagem. Cada nó regista o *id* e a *origem* da mensagem para verificar se esta já foi respondida por ele. Se um nó confirmar, através do identificador e da origem, que a mensagem já foi respondida, o nó ignora a mensagem e não prossegue com a sua replicação pela rede.

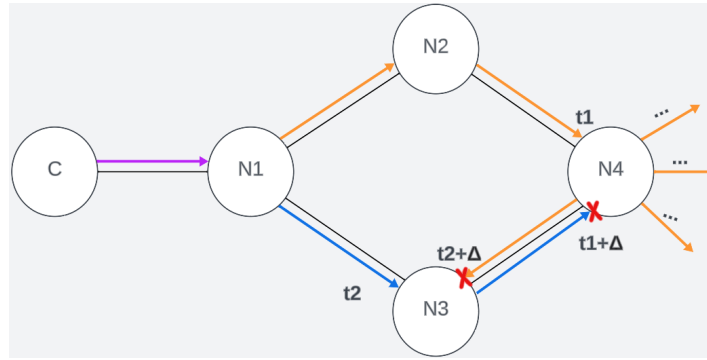


Figura 6. Ilustração do pedido de localização de um vídeo

A título de exemplo, na figura acima nenhum dos nós apresentados tem o vídeo e um cliente envia o pedido de localização (**CHECK_VIDEO**) de um vídeo ao nó N1, que por sua vez propaga o mesmo pedido para os seus nós adjacentes (N2 e N3). Posteriormente, esses nós propagam o pedido até N4.

Em N4, como recebe o pedido de N2 primeiro (t_1), então o pedido feito por N3 ($t_1 + \Delta$) será ignorado. Assim, N4 propaga o pedido de N2 para todos os seus vizinhos, incluindo o N3. O N3, como já tinha recebido este pedido de N1 em t_2 , então ignorará o pedido de N4 feito em $t_2 + \Delta$.

Quando o nó adjacente obtém a localização do vídeo (**RESP_CHECK_VIDEO** enviado pelo nó/RP que tem o vídeo), envia a resposta ao cliente, indicando no campo **origem** o endereço IP do nó (ou RP) que possui o conteúdo pretendido.

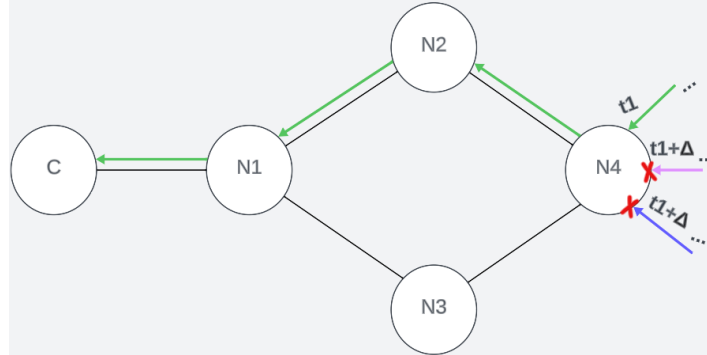


Figura 7. Ilustração da resposta ao pedido de localização do vídeo

Na figura acima, ilustramos o momento em que N4 recebe as respostas aos pedidos de localização de vídeo que propagou no exemplo da figura 6. Neste caso, N4 captura a resposta mais rápida (recebida em t_1) e ignora as respostas que vieram depois dela.

Na receção das mensagens **RESP_CHECK_VIDEO**, os nós intermédios utilizam o campo **origem** (ip do nó/RP que tem o vídeo) dessas mensagens para povoar uma tabela de encaminhamento que servirá para encaminhar a solicitação de um determinado vídeo. A entrada que será inserida nessa tabela será um par (ip de quem tem o vídeo, ip do vizinho pelo qual recebeu a resposta). Desta forma cada nó do trajeto terá a informação para qual vizinho irá reencaminhar a solicitação do vídeo, tendo em conta o nó por onde recebeu a resposta, consequentemente, levando o pedido para onde o vídeo está.

3.2.3 Transmissão de um vídeo

Após o cliente saber a localização de um vídeo, obtida através do pedido de localização do vídeo (**CHECK_VIDEO**), este envia ao nó adjacente uma mensagem (**START_VIDEO**) a indicar que pretende receber o vídeo. No campo **dados** dessa mensagem são incluídos o nome do vídeo solicitado e o endereço IP do componente que possui o vídeo.

Se o nó adjacente ao cliente for o nó que possui o vídeo, este fornece directamente o vídeo ao cliente. Caso contrário, encaminha o pedido do cliente para o próximo nó (obtido através da tabela de encaminhamento que remeterá a mensagem para o caminho correto) e aguarda a transmissão do vídeo pelo próximo nó. Isto ocorre sucessivamente até que o pedido do cliente alcance um nó com o vídeo ou até atingir o RP.

No momento em que o pedido do cliente chega a um nó que possui o vídeo (ou ao RP), esse mesmo nó inicia a transmissão do vídeo para o nó adjacente de onde o pedido veio, enviando pacotes RTP cujo *payload* são os *frames* do vídeo. Por sua vez, o nó que agora está a receber a transmissão encarrega-se de encaminhá-la até ao próximo nó que lhe solicitou o vídeo. Este processo repete-se sucessivamente até que os *frames* do vídeo cheguem ao cliente que o solicitou.

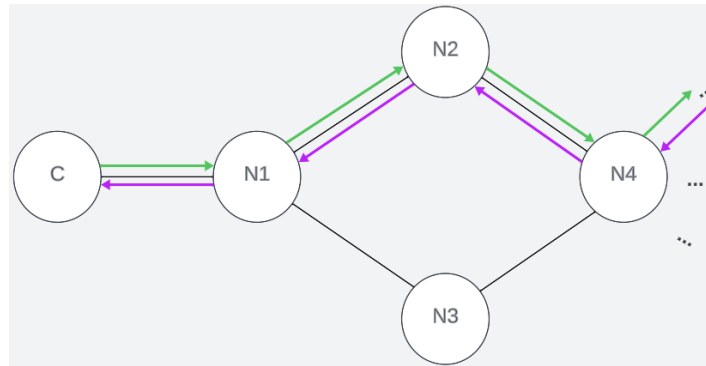


Figura 8. Ilustração de uma transmissão de vídeo

Na figura acima, ilustramos o caso em que o cliente efectua um pedido de visualização de um vídeo (continuação do exemplo da figura 7). Neste caso, tanto o pedido de começo de vídeo (**START_VIDEO**) assinalado com setas verdes, como a transmissão do vídeo assinalado com setas roxas, seguem uma rota bem definida devido ao povoamento da tabela de encaminhamento previamente realizado através da resposta ao pedido de localização de vídeo ilustrado na figura 7

Durante esta interacção, os nós que estão a receber uma transmissão de um vídeo enviam regularmente ao nó que lhe fornece a transmissão uma mensagem a indicar que ainda se encontra em funcionamento (**ALIVE_RECEPTOR**) e que está a consumir a transmissão. Desta forma, caso o receptor deixe de enviar estas mensagens, o nó que fornece a transmissão consegue saber se o seu receptor teve uma interrupção inesperada e assim parar essa transmissão para não ocupar recursos da rede desnecessariamente.

3.2.4 Paragem de transmissão de vídeo

No momento em que um cliente/nó pretende interromper a visualização de um vídeo, envia um pedido (**STOP_VIDEO**) ao nó que lhe está a transmitir o vídeo. Neste pedido, o campo **dados** contém o nome do vídeo que estava a ser reproduzido. Em resposta, o nó adjacente interrompe a transmissão do vídeo ao cliente/nó, removendo o seu IP do registo de nós que estão a consumir o vídeo. Se não houver mais clientes (ou nós) a receber o vídeo, o nó que estava a transmitir emite também um pedido **STOP_VIDEO** ao nó que lhe fornecia o vídeo. Isto ocorre uma vez que não existe mais a necessidade de continuar a transmitir o vídeo.

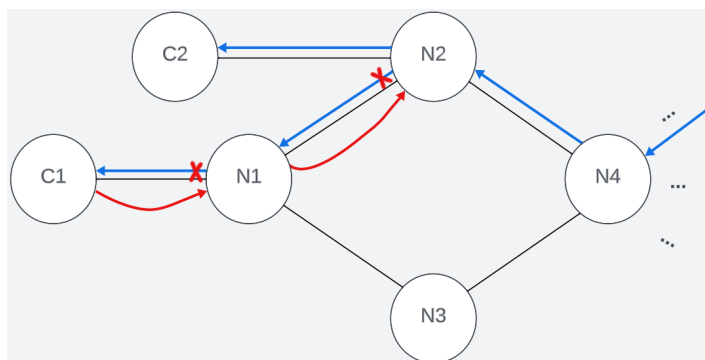


Figura 9. Ilustração do pedido de paragem de transmissão

Na figura acima ilustramos o caso em que temos mais do que um cliente a consumir o mesmo vídeo. Tal como foi explicado, no momento em que o cliente C1 solicita a paragem de transmissão do vídeo (seta vermelha), o nó N1, para além de parar de fornecer o vídeo a C1, vai também solicitar a paragem de transmissão ao nó que lhe estava a fornecer o vídeo (N2), uma vez que C1 era o único consumidor do vídeo em N1. O nó N2, por sua vez, como ainda tem o cliente C2 a consumir o vídeo, apenas irá parar a transmissão do vídeo para o nó N1.

No caso do RP receber um pedido de paragem de transmissão e não existirem mais nós a consumir o vídeo, o RP interromperá a transmissão do vídeo e enviará um pedido ao servidor que lhe fornecia o vídeo, solicitando também a cessação da transmissão.

Existe um caso particular que consiste na situação em que há apenas um cliente a consumir um certo vídeo. Neste caso, quando o cliente solicita ao seu nó adjacente a paragem da transmissão, a mensagem de interrupção irá percorrer todos os nós envolvidos na transmissão até atingir o RP. O RP, por sua vez, também interrompe o consumo do vídeo vindo do servidor.

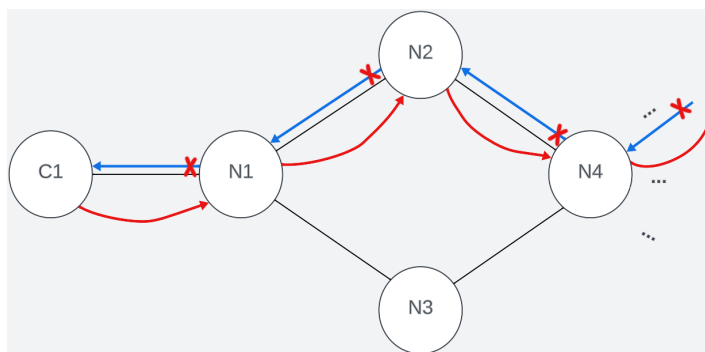


Figura 10. Ilustração do pedido de paragem de transmissão para um cliente sozinho

Na figura acima ilustramos o caso em que apenas um cliente está a consumir um vídeo. Nesse caso, quando o cliente solicita a paragem de transmissão do vídeo, todos os elementos envolvidos na transmissão irão parar a transmissão desse vídeo.

3.2.5 Entrada de um novo nó

No momento em que um novo nó é inicializado na rede, é-lhe fornecido um ficheiro de configuração que contém informações sobre os seus nós vizinhos. Com base nessas informações, o nó envia uma mensagem (`ADD_VIZINHO`) a todos os nós mencionados no ficheiro para indicar a sua presença.

Ao receberem este pedido/mensagem (**ADD_VIZINHO**), os nós registam na sua base de dados o endereço IP de onde provém o pedido, considerando-o como um vizinho. Em seguida, enviam uma mensagem de confirmação a esse nó com o campo **dados="ACK"** para lhe indicar que o registaram como vizinho. Caso o nó não confirme a receção da notificação dentro de um prazo de 2 segundos, o novo nó irá tentar notificar novamente o nó que não lhe respondeu, enviando a mesma mensagem mais uma vez e caso não seja capaz de se confirmar a receção, o nó desiste de o notificar.

3.2.6 Saída de um nó

No momento em que um nó sai da rede, este deve indicar a sua saída aos seus nós vizinhos. Para isso, o nó envia aos seus nós vizinhos uma mensagem (**RMV_VIZINHO**) a indicar a sua saída da rede.

Quando os nós recebem esta mensagem retiram o endereço IP, de onde veio o pedido, dos vizinhos da sua base de dados, cancelando as transmissões que têm como destino este endereço.

4 Implementação

4.1 Etapa 1: Construção da Topologia Overlay com Árvore Partilhada

A construção da topologia *overlay* na nossa solução consiste no fornecimento de um ficheiro de configuração JSON a cada componente da topologia (servidores, RP, nós intermédios e clientes). Nesse ficheiro estão presentes as informações necessárias para cada componente da rede *overlay* e que serão guardadas numa base de dados própria de cada componente, para ser utilizada nas interações.

A topologia *overlay* da nossa solução é dinâmica, o que significa que está preparada para eventuais entradas e saídas de nós.

No momento do arranque de um novo nó, também é fornecido um **ficheiro de configuração** a indicar os seus vizinhos. O novo nó utilizará os dados deste ficheiro para enviar uma mensagem para a porta 3005 dos componentes mencionados no ficheiro a indicar a sua entrada na rede *overlay*. Por sua vez, os nós que recebem esta mensagem registarão o novo nó como um vizinho seu.

No que diz respeito à saída de nós, cada nó terá a responsabilidade de indicar a sua saída aos seus vizinhos. Para isso o nó envia uma mensagem de *teardown* a indicar a sua saída para as **portas 3006**. Posteriormente, os nós que recebem a mensagem de *teardown*(**RMV_VIZINHO**) eliminam o endereço do nó que saiu do seu registo de vizinhos.

4.2 Etapa 2: Serviço de Streaming

O serviço de *streaming* da nossa solução utiliza o código fornecido pela equipa docente. Esse código é utilizado numa classe **ServerWorker** que é criada no servidor de conteúdos para cada vídeo que este está a transmitir. A classe **ServerWorker** utiliza a classe **VideoStream** para ler o ficheiro de vídeo e retornar os *frames* que serão colocados num pacote RTP enviados num *socket* UDP para o RP.

A transmissão de um vídeo através do RP consiste na receção dos *frames* vindos do servidor de conteúdo que posteriormente são reencaminhados para os nós que fazem parte do trajeto do vídeo.

A transmissão de um vídeo através de um nó intermédio é bastante semelhante à transmissão pelo RP em que a única diferença é que a proveniência dos *frames* dos vídeos é o RP ou outro nó intermédio.

4.3 Etapa 3: Monitorização dos Servidores de conteúdos

Os servidores de conteúdos são monitorizados regularmente pelo RP (a cada 2 minutos). Esta monitorização consiste no envio de 10 mensagens de prova que posteriormente os servidores retornam de volta ao RP. Assim, o RP utilizará essas mensagens de retorno para determinar (i) a percentagem de pacotes que foram retornados com sucesso pelo servidor e (ii) a média da latência do percurso das mensagens desde o Servidor ao RP.

Com esses valores, o RP calculará o valor da métrica que será utilizado na escolha de um servidor no momento da solicitação da transmissão de um vídeo.

Ao utilizar esta estratégia de enviar 10 mensagens a cada 2 minutos para os servidores, acrescentamos 5 mensagens por minuto de overhead de controlo à rede, por cada servidor disponível. Consideramos que este valor encontra-se num bom equilíbrio entre frequência da verificação de qualidade dos servidores e a quantidade de overhead imposto na rede.

4.4 Etapa 4: Construção dos Fluxos para Entrega de Dados

A construção dos fluxos para a entrega de dados é realizada mediante a solicitação da localização de um vídeo pelo cliente (`CHECK_VIDEO`). Esse pedido é enviado ao nó adjacente do cliente para a porta 3001, que envia a solicitação aos seus vizinhos, caso o vídeo ainda não esteja a passar por esse nó. Quando a solicitação atinge um nó que possui o vídeo, este responde com uma confirmação (`RESP_CHECK_VIDEO`), juntamente com o seu endereço IP.

Como podem existir vários nós a transmitir o vídeo, os nós que realizaram a solicitação da localização do vídeo poderão receber várias respostas com várias localizações diferentes. Nesta situação, o nó irá utilizar a localização cuja resposta foi recebida em primeiro lugar, isto é, a que teve menor latência, uma vez que no contexto de transmissão de conteúdo em tempo real a latência é dos factores mais importantes.

Caso um nó não tenha o vídeo e não tenha mais por onde difundir o pedido, não é enviada nenhuma mensagem, fazendo com que o timeout notado pelos nós que estão à espera, seja o indicador da inexistência do vídeo.

À medida que os nós recebem uma localização de um vídeo (`RESP_CHECK_VIDEO`), registam essa localização associada ao IP do vizinho que a encaminhou, numa tabela de encaminhamento. É essa tabela que é responsável por estabelecer as rotas de transmissão dos vídeos para os clientes, uma vez que quando um cliente solicitar a transmissão do vídeo (`START_VIDEO` pela porta 3002), os nós irão consultar essa tabela para saber para onde encaminhar o pedido.

4.5 Etapa Adicional: Recuperação de falhas e entradas de nós adicionais

Para além das funcionalidades básicas, implementamos também mecanismos de recuperação de falhas e de entradas de nós adicionais.

As entradas e saídas de nós adicionais são simples mensagens que utilizam o tipo de Mensagem `ADD_VIZINHO` e `RMV_VIZINHO`, que foram explicadas na secção das Interações.

Caso o nó que esteja de saída faça parte de um trajeto que propague uma *stream*, existem dois casos a ter em conta.

O primeiro é quando este nó que saiu sem aviso, enviava a *stream* a um determinado nó. Este nó recetor irá detetar que devido ao encerramento do nó fornecedor, não está a receber pacotes de vídeo, despoletando um timeout que irá fazer uma análise às informações que tem do fornecedor. Como não houve nenhum aviso de encerramento, o nó continua a pensar que o nó mantém-se ativo, dando mais 2 oportunidades ao nó (timeouts de 1 segundo) para receber algum *frame* do conteúdo. Quando as oportunidades se esgotam o nó irá assumir que o seu nó fornecedor, não está acessível, tratando de arranjar um novo fornecedor. Para tal, o nó passa a atuar como um cliente, no sentido em que volta a fazer um `CHECK_VIDEO` para procurar o vídeo na rede e o consequente `START_VIDEO`, para que receba o vídeo novamente e continue a fornecer aos respetivos consumidores. De modo a que estes consumidores, que podem também ser nós da topologia, não comecem a reorganizar o fornecedor por conta própria dada a ausência do recebimento de pacotes, o nó vizinho ao que foi encerrado e que faz parte da rota de transmissão, encarregar-se-á de mandar *dummy* packets para que os outros nós, não detetem nenhum timeout, deixando que este nó vizinho seja o responsável por reorganizar o fornecedor.

Com esta saída silenciosa de um nó, surge outro caso que é necessário tratar, relativo a um nó que está a fornecer conteúdo ao nó que encerrou. Sem haver nenhuma espécie de aviso, este nó não terá maneira de saber que está a enviar conteúdo para um componente encerrado, dado que o protocolo *UDP* não fornece mecanismos embutidos para informar um cliente sobre a desconexão de um servidor. Para ter em conta este caso, foi implementado um novo serviço nos *oNodes* que notifica os fornecedores ativos, mandando-lhes uma mensagem `ALIVE_RECEPTOR` de 20 em 20 segundos, uma espécie de mecanismo de "heartbeat". Os fornecedores ao receberem esta mensagem, tomam nota do endereço do remetente

e uma timestamp a indicar quando a receberam. Com este registo, o fornecedor é capaz de fazer uma verificação da última mensagem que recebeu de um certo consumidor. Caso seja detetado que um consumidor deixou de enviar estas mensagens, o consumidor é removido da base de dados como consumidor de conteúdo, deixando de lhe ser enviada a stream e poupando a rede do envio de pacotes desnecessários.

5 Testes e resultados

Para os testes apresentados nesta secção, foi utilizada seguinte topologia (com os elementos e as ligações da rede *overlay* em destaque):

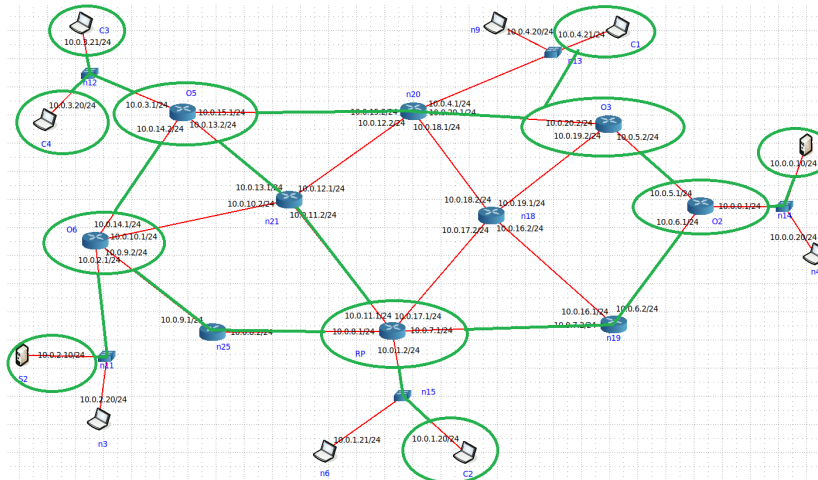


Figura 11. Topologia para os testes

5.1 Transmissão de vídeo para mais do que um cliente

No teste abaixo, apresentamos a transmissão do mesmo vídeo para dois clientes C1 e C3, cujos vizinhos são O3 (10.0.20.2) e O5 (10.0.3.1), respectivamente (o título de cada terminal possui o nome do componente associado).

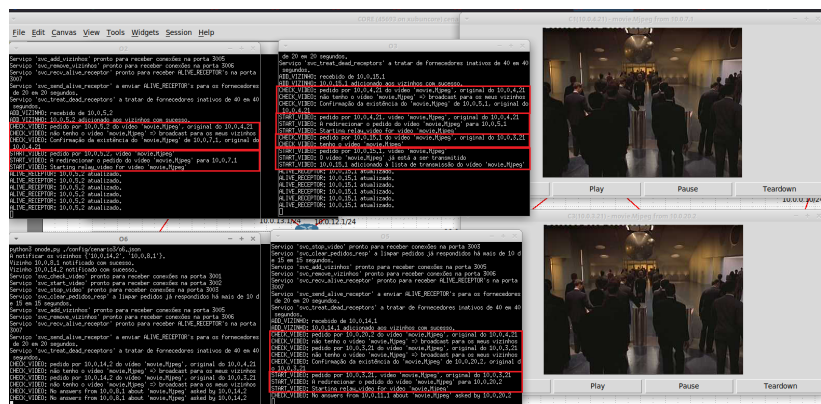


Figura 12. Transmissão para dois clientes

Pelos *logs* dos nós intermédios na figura acima, conseguimos verificar em O3 a solicitação do vídeo (CHECK_VIDEO) *movie.Mjpeg* vinda do cliente C1 (10.0.4.21). O nó O5, por sua vez, foi o primeiro a obter a localização do vídeo (neste momento no RP devido a ser a primeira solicitação em toda a rede) e encaminha-a para o O3 que informa o cliente C1. Seguidamente, o cliente C1 solicita de imediato a O3 para iniciar a transmissão do vídeo. Podemos ver agora que O3 solicita a O2 a transmissão do vídeo e como O2 não tem o vídeo, encaminha o pedido ao RP (10.0.7.1). Ao receber a transmissão do RP, o nó O2 encaminha-a para O3. Por sua vez, O3 inicia a transmissão para o cliente C1.

No que diz respeito à transmissão para o cliente C3, conseguimos observar o pedido de localização do vídeo desse cliente a O5. Como O5 não tem o vídeo, então encaminha o pedido do cliente a todos os seus vizinhos. A primeira resposta que O5 obteve foi de O3 a indicar que tem o vídeo, sendo esta a informação que o cliente C3 irá receber. Assim sendo, C3 envia uma solicitação de transmissão de vídeo a O5 cujo destino é o nó O3. No momento em que O3 recebe esta solicitação, inicia de imediato a transmissão do vídeo (uma vez que já o está a transmitir) a O5 que por sua vez transmite ao cliente C3.

5.2 Saída de clientes

Neste teste apresentamos o que ocorre no momento em que todos os clientes que estavam a consumir um vídeo saem da transmissão. Este teste é a continuação do teste do teste anterior e pretende demonstrar que toda a transmissão, desde o cliente até ao servidor, é parada quando não existem mais clientes a visualizar um certo vídeo.

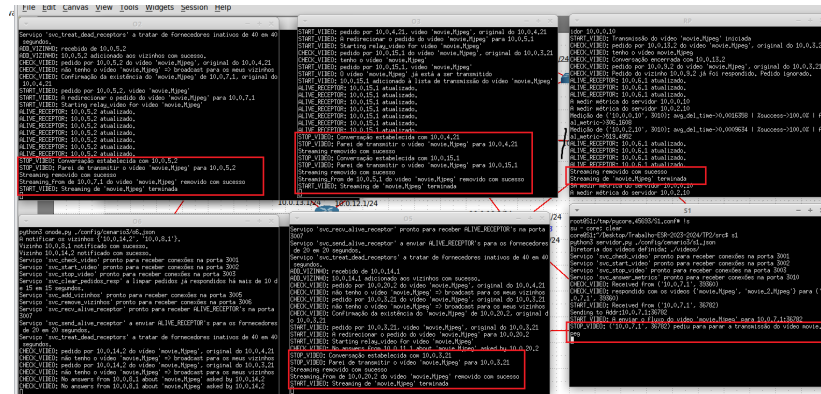


Figura 13. Saída de todos os clientes de uma transmissão

Neste teste, o nó O3 estava a transmitir o mesmo vídeo para os clientes C1 e C3 (teste anterior). No momento em que o cliente C1 solicita a paragem de transmissão de vídeo a O3, o nó O3 deixa de transmitir o vídeo ao cliente C1, mas ainda continua a transmitir para O5 que transmite ao cliente C3.

Quando o cliente C3 solicita a paragem de transmissão, o nó O3 verifica que já não existem mais clientes a consumir o vídeo. Desta forma, irá solicitar a O2 que pare de transmitir. Por sua vez O2, também não tem mais nós a consumir o vídeo e solicita ao RP que pare de transmitir. No final, o RP informa o servidor S1 para parar de transmitir o vídeo.

5.3 Escolha do melhor servidor

Neste teste é pretendido apresentar o modo como o RP actua no momento da escolha de um servidor para realizar a transmissão de um vídeo. Para testar isso, forçamos um atraso de 100 ms na ligação do servidor S1 ao RP.

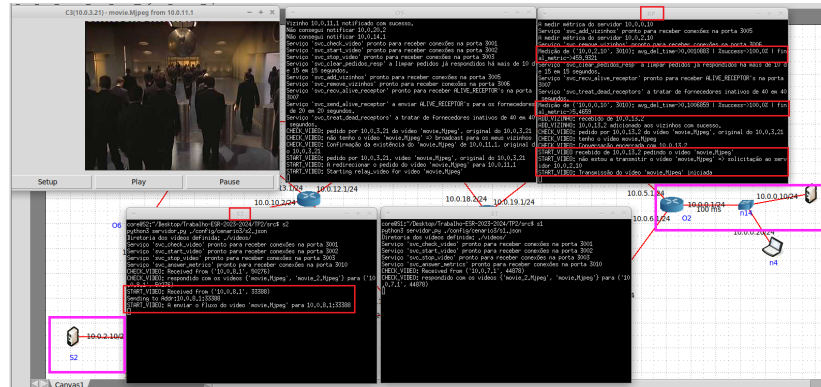


Figura 14. Escolha do melhor servidor

Como podemos ver nos *logs*, o RP solicitou a transmissão do vídeo ao servidor S2 dado que este tinha um valor da métrica maior.

5.4 Extra: Tratamento de falhas

Neste teste apresentamos o tratamento de falhas na stream, quando existe uma falha sem aviso de um nó que está a participar no encaminhaamento de uma stream de vídeo.

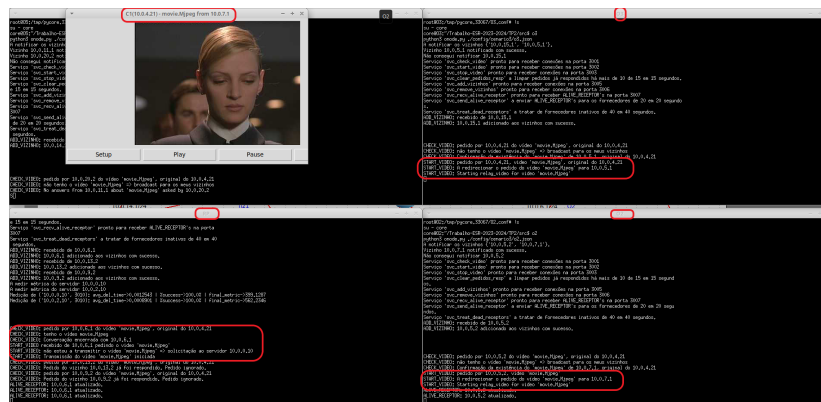


Figura 15. Antes do tratamento de falhas

Como podemos ver nesta última figura, uma rota de streaming foi estabelecida do RP → O2 → O3 → C1. Após o estabelecimento desta rota iremos desativar o nodo O2, sem despoletar nenhuma espécie de mensagem de aviso, que pode ser feito através do atalho (CTRL+\\).

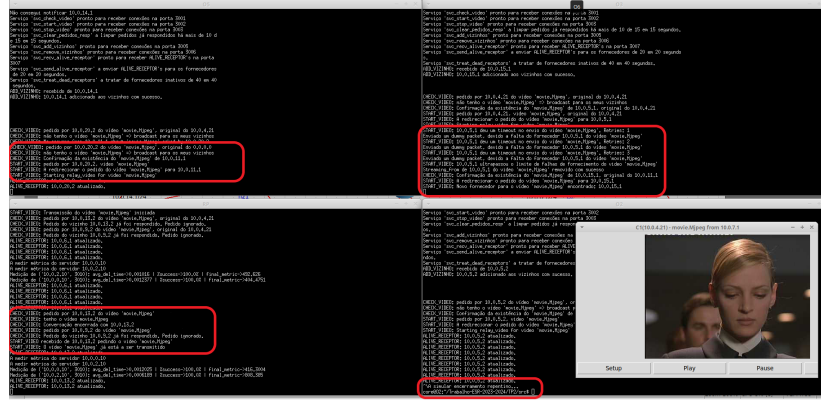


Figura 16. Tratamento de falhas executado

Nesta imagem podemos ver que o O3 detetou um timeout no fornecimento do vídeo por parte do O2, tendo esperado pelo sistema de retries passar o seu limite. Após o limite ter sido ultrapassado, ele procura um novo fornecedor para o vídeo através do envio de um `CHECK_VIDEO` que obtém a informação que o RP tem esse vídeo através do O5, ou seja pela seguinte rota $RP \rightarrow O5 \rightarrow O3 \rightarrow C1$. Podemos também reparar na imagem que o O3 envia um *dummy packet* para os seus consumidores, para que não notem nenhum timeout, fazendo com que a responsabilidade de arranjar um novo fornecedor seja sempre do nodo mais próximo ao nodo que falhou.

6 Conclusões e trabalho futuro

No fim da elaboração do presente projecto, pretendemos apresentar uma avaliação crítica do nosso trabalho.

No que diz respeito aos aspectos positivos, queremos destacar o facto de todas as funcionalidades terem sido implementadas bem como algumas funcionalidades sugeridas na etapa adicional do enunciado. O sistema implementado é capaz de transmitir conteúdo para vários clientes, bem como ajustar-se a alterações e/ou adversidades na topologia. O sistema é capaz de se ajustar à saída voluntária de nós bem como quando os nós são interrompidos repentinamente sem avisar.

Ao nível dos aspectos a melhorar, consideramos que os *logs* apresentados pelos componentes pudessem ser mais configuráveis como por exemplo, indicar um maior grau de detalhe ao indicar *timestamps*, indicar que tipo de *logs* que não são apresentados no terminal, para não atrapalhar na visualização de algum processo em específico, entre outras opções. Poderíamos também ter abordado a ideia de haver mais do que um RP na rede *overlay*. Apesar de considerarmos que seria possível, tendo em conta a nossa implementação final, não fomos capazes de testar a ideia na prática devido a falta de tempo.

No fim deste trabalho, acreditamos ter implementado um bom projecto, na medida em que todos os requisitos do enunciado foram cumpridos e, além disso, também fomos capazes de abordar aspectos adicionais que contribuem para valorizar o trabalho.