Trabalho Prático Nº2 – Serviço Over the Top para entrega de multimédia

Eduardo Francisco Longras Figueiredo *pg52679@uminho.pt*

Gonçalo Nuno Pereira Senra *pg52683@uminho.pt*

Henrique Miguel da Silva Costa *pg52684@uminho.pt*

1 Introdução

No decorrer da unidade curricular (UC) de Engenharia de Serviços em Rede (ESR) da Universidade do Minho, foi proposto implementar um serviço Over the Top para entrega de multimédia, que cada vez mais têm sido utilizados no dia a dia das pessoas, principalmente no que toca a streaming de vídeos fruto da eminente subida da utilização de serviços fornecidos por empresas como por exemplo a *Netflix*.

Este serviço de entrega de multimédia, está assente sob comunicações entre routers intermediários denominados de ***oNodes*** de modo a melhorar a eficiência das transmissões de vídeos entre clientes e servidores, sendo que existe um ***Rendezvous Point*** (RP) que desempenha um papel importante na organização das rotas e distribuições de vídeos.

Deste modo, com base nas tarefas necessárias à execução deste trabalho, ao longo deste relatório iremos destacar temas como a **arquitetura da solução**, **especificações do(s) protocolo(s)**, **implementações**, **limitações da solução**, **testes**, entre outros detalhes fundamentais à vitalidade do programa.

2 Arquitetura da solução

Neste projeto, foi desenvolvido um serviço Over the Top para entrega de multimédia utilizando a linguagem de programação Java e o protocolo de comunicação UDP.

Com base na rede overlay proposta, foram implementados *oNodes* para viabilizar a comunicação bidirecional entre servidores e clientes através da construção e manutenção de uma árvore de distribuição partilhada. A arquitetura desenvolvida é baseada numa rede de *overlay*, que consiste num posicionamento estratégico dos *oNodes* numa rede de *underlay* já existente. Estes *oNodes*, assentes em ficheiros de configuração de arranque com os *ips* da sua vizinhança, são capazes de comunicar entre si de forma a propagar *chunks* de vídeo, de forma a satisfazer as necessidades de qualquer cliente, e manter a árvore de distribuição partilhada atualizada.

No que toca ao serviço de *streaming*, o servidor, quando solicitado, envia pacotes numerados para o RP (*Rendezvous Point*) que os redireciona até ao cliente. No caso de um *oNode* já possuir a *stream*, requisitada pelo cliente, de modo a evitar tráfegos e fluxos de transmissão elevados, esse mesmo nó suporta o pedido pretendido.

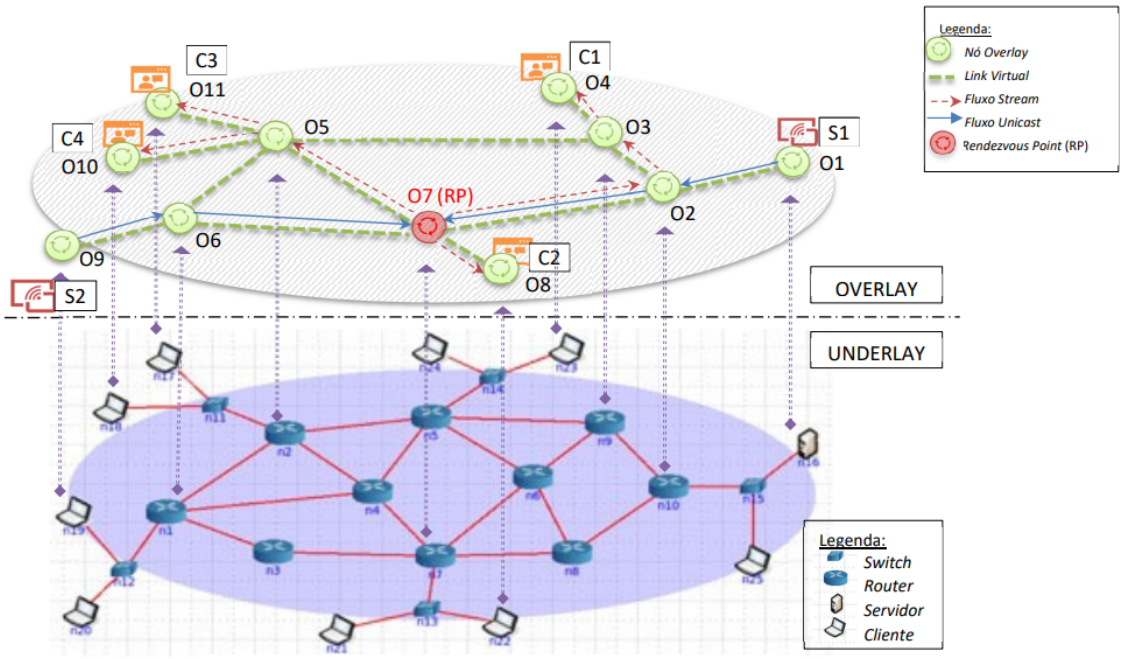

Figure 1: Visão geral de um serviço OTT sobre uma infraestrutura.

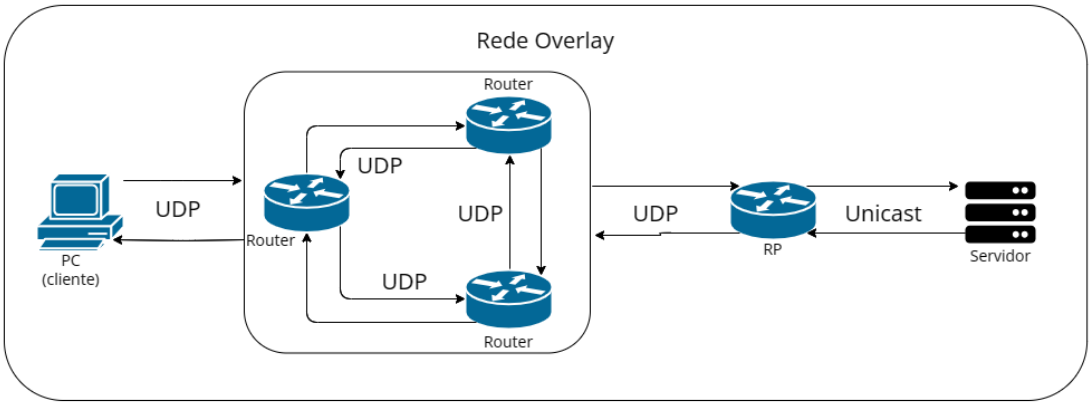

Figure 2: Diagrama da arquitetura da solução.

3 Especificação do(s) protocolo(s)

No âmbito dos requisitos presentes no enunciado, evidenciou-se que seria fulcral a criação de protocolos de streaming e de controlo do fluxo de dados e, deste modo, de maneira a suportar estas necessidades, desenvolvemos as classes ***Packet*** (utilizada para determinar a rota mais eficiente para que o cliente encontre um *oNode* que contenha a *stream*, alem disso é usada para estabelecer a conexão entre servidor e *RP*) e ***RTPpacket*** (usada para transmitir o vídeo do servidor até ao(s) cliente(s)).

3.1 Formato das mensagens protocolares

As mensagens protocolares foram estruturadas em pacotes que consoante a porta de receção do ator em questão (*oNode*, cliente ou servidor), desencadeariam reações diferentes, sejam elas de auxílio de construção e manutenção da árvore, procura, envio e cessamento de *streams*, medição de métricas e conexão a servidores.

Deste modo, a classe ***Packet*** está estruturada da seguinte forma:

- **data**: conteúdo do pacote, com informações relevantes às funções desejadas;
- **hops**: número de ligações por onde já passou (saltos) na rede overlay;
- **path** e **pathInv**: listas que indicam as interfaces dos *oNodes* pelos quais o pacote passou na ida e na volta, respetivamente;
- **networks** e **prevNetworks**: listas que indicam as redes pelas quais o pacote já passou e as redes vizinhas do *oNode* do último salto, respetivamente;
- **aux**: determina se um pacote é proveniente do cliente ou do *oNode*;
- **info**: lista com as *streams* que um servidor possui.

Por outro lado, a classe ***RTPpacket*** está estruturada conforme o que a equipa docente forneceu, sendo que foi necessária a inclusão do seguinte parâmetro:

- **videoName**: variável que contém o nome da *stream* à qual o pacote pertence.

Esta inclusão será aprofundada na secção 4.

3.2 Interações

Do ponto de vista operacional, existem 3 interações evidentes :

3.2.1 cliente $\longleftrightarrow$ *oNode folha*

- Cliente envia um pacote a solicitar a *stream*;
- Cliente escolhe o melhor caminho de acordo com as métricas implementadas e envia de volta um pacote a atualizar o caminho dos *oNodes* envolvidos;
- Cliente recebe a *stream*;
- Cliente recebe um pacote tipo "ping" e envia um do tipo "pong".

3.2.2 oNode $\longleftrightarrow$ *oNode*

- *oNode* recebe um pacote tipo "ping" e envia um do tipo "pong";
- *oNode* redireciona o pacote de origem do cliente até encontrar a *stream*;
- *oNode* envia *stream* do *RP* ao(s) cliente(s);
- *oNode* cancela *stream* do *RP* ao(s) cliente(s) e vice-versa;
- *oNode* envia pacotes tipo "ping" para adicionar outro *oNode* à topologia, ou caso hajam *timeouts* para remover *oNodes* da topologia.

3.2.3 oNode ⟷ *servidor*

- Servidor recebe um pacote tipo "ping" e envia um do tipo "pong", para conexão e medição de métricas;
- Servidor envia *chunks* da stream via *unicast* para o *RP*;
- *oNode* envia pacote a solicitar uma determinada *stream*.

4 Implementação

4.1 Construção da Topologia Overlay com Árvore Partilhada

Na nossa implementação da construção da rede *overlay* com uma topologia de árvore compartilhada (estratégia 2), cada *oNode* é configurado inicialmente com um arquivo que contém informações sobre seus vizinhos, incluindo os endereços *IP* e, respetivamente, as máscaras de rede. Depois da inicialização, cada nó envia pacotes com detalhes sobre a sua rede para todos os vizinhos, utilizando a porta **8500**. Essa comunicação é essencial para atualizar um dicionário que disponibiliza as informações sobre o estado dos *oNodes* vizinhos, tendo como chave uma classe implementada por nós, *IPWithMask*, que guarda o *IP*, a mascara da rede e a network, e contém como valores se esse *oNode* está ativo ou desativo.

Para garantir a fiabilidade da rede, cada nó tem uma thread unicamente dedicada para ouvir pacotes recebidos na porta **8000** e tem um tempo estimado de um segundo para responder. Se não houver uma resposta dentro desse tempo, o *oNode* tenta reenviar o pacote por três vezes antes de considerar o vizinho desativado. Essas atualizações na topologia acontecem a **cada dois segundos**, tornando a rede overlay dinâmica.

A implementação foi pensada para minimizar a perda de pacotes e para desativar os vizinhos apenas após várias tentativas de comunicação sem sucesso. Essa abordagem procura implementar uma rede *overlay* robusta e dinâmica, capaz de lidar com mudanças no estado dos *oNodes*.

4.2 Serviço de Streaming

A nossa visão para o serviço de *streaming* baseou-se na criação de uma classe *Server* cuja função é distribuir os seus conteúdos através do *RP*, para a árvore de difusão. No que toca à comunicação entre o *Server* e o *RP*, o *ip* do *RP*, é passado por parâmetro ao *Server*, tal como os caminhos de todos os vídeos que o *Server* pretende distribuir. Nesta classe, fazemos uso do código disponibilizado pelos docentes e criamos uma instância da classe *ServerRTP* por cada conteúdo que estiver disponível no *Server*. Desta forma, cada objeto do tipo *ServerRTP* tem a capacidade de dividir o vídeo atribuído em pacotes numerados, prontos a serem enviados para o *RP* e consequentemente, difundidos na árvore de distribuição. Durante o desenvolvimento foi necessário adicionar uma variável ao *header* dos pacotes *RTP*, com o nome *videoName* (nome do conteúdo ao qual o pacote pertence), para que os *oNodes* durante a distribuição e obrigatório desempacotamento, possam distinguir os pacotes que recebem, e encaminhá-los para os *oNodes* corretos.

Para que o serviço de *streaming* comece, é necessário que haja no mínimo um cliente que solicite um determinado conteúdo, sendo esta ação despoletada através da função *bestPath*, após escolher o melhor caminho até ao *RP* ou *oNode* mais próximo com esse conteúdo, (mais detalhes sobre a função *bestPath* e todos os detalhes de construção do fluxo para entrega de dados na subsecção 4.4). Após a escolha do melhor caminho, um pacote é enviado para atualizar as rotas dos *oNodes* desse caminho, e quando o mesmo chega ao *oNode* destino, a *stream* começa a ser encaminhada pelos *oNodes* até chegar ao cliente. O cliente estará preparado para receber a *stream*, através do controlo efetuado pela função *actionPerformed* da classe *clientTimerListener* (adaptada do código fornecido pelos docentes) que tem um *socket* à escuta na porta **9000**.

No lado dos *oNodes*, depois dos caminhos possíveis chegarem ao cliente, o *oNode* folha recebe um pacote do cliente com a intenção de começar a receber a *stream* através do melhor caminho, (este pacote é recebido apenas no nó folha, na função *updateBestPath* que tem um socket (*BestPathSocket*) à escuta na porta **7000**. Este *oNode*, atualiza a lista de *streams* que o mesmo difunde, e os dicionários *bestPath* e *bestPathInv* que associam as *streams* aos *ips* que pretendem receber a *stream* e o caminho inverso, respetivamente. Após

estas atualizações fulcrais para o bom encaminhamento do fluxo de *streams*, o pacote enviado pelo cliente e posteriormente recebido pelo *oNode* adjacente, é reenviado para o próximo *oNode* do melhor caminho, que por sua vez, efetuará as mesmas ações do *oNode* anterior e assim por diante. Estas ações, quando propagadas até ao *oNode* destino, desbloqueiam a *stream*, fazendo-a chegar ao cliente.

Para além das funcionalidades supra mencionadas, é necessário referir que existem mecanismos de cancelamento de *stream*, ou seja, quando um cliente é terminado o *oNode* folha deteta o seu termino através da função *pingClient* que envia pacotes tipo *ping* ao cliente com um intervalo de 2 segundos, (pacotes que são recebidos e respondidos pelo cliente na função *pongRouter*). Posto isto, quando o pacote enviado pelo *oNode* não é correspondido, o mesmo prepara o cancelamento da *stream*. Este cancelamento processa-se da seguinte forma: caso o *oNode* tenha mais *ips* para onde tenha de enviar o conteúdo, então o *cliente* que cancelou a subscrição da *stream* fica com o seu *ip* removido do dicionário de difusão (bestPath), e o cancelamento não é propagado; no caso do *oNode* em questão ter apenas esse *ip* como único destino de um determinado conteúdo então, para além desse *ip* ser apagado do dicionário *bestPath*, a *stream* é removida da lista *streams*, e o cancelamento é propagado para o *oNode* que lhe fornecia a *stream* (todos os mecanismos mencionados, são controlados pelas funções *cancelStream* e *cancelStreamClient*). Esta reação em cadeia faz com que, por exemplo, se em toda a rede houver apenas um cliente a consumir uma *stream*, o cancelamento seja propagado, e nenhum conteúdo seja difundido na árvore, nem do servidor para o *RP*, já que, nenhum cliente estaria a solicitar uma *stream*.

4.3 Monitorização dos Servidores de conteúdos

Uma vez que as comunicações entre servidor e *RP* são *Unicast* e com vista a manter uma atualização periódica das condições de entrega de cada servidor, foi implementado um método em cada um destes provedores. Do lado do *RP*, através do uso de uma *thread*, o método *pingServer* envia um pacote do tipo *ping* para cada servidor ativo à escuta na porta **5001**, de **dois em dois segundos**, ficando à espera de uma resposta. Já do lado do servidor, existe uma *thread* à escuta na porta **5000**, para receção dos pacotes enviados diretamente pelo *RP*.

No momento em que o *RP* recebe a resposta, é realizada uma atualização da métrica **latência** para cada servidor que efetuou a resposta (isto significa que se encontra ativo). No entanto, caso após três tentativas de conexão (*retries*), ainda não tenha ocorrido a receção de uma resposta, o servidor é considerado como inativo. Deste modo, se o servidor removido era o que estava a realizar a transmissão do vídeo, são procurados outros servidores ativos que possuam o mesmo vídeo, priorizando o que tiver menor latência. No infortúnio de não haver servidores ativos ou com o vídeo desejado, o cliente deixa de receber a *stream*, até que volte a existir um servidor que satisfaça as necessidades do mesmo.

4.4 Construção dos Fluxos para Entrega de Dados

A construção dos Fluxos para Entrega de Dados passa por quatro etapas:

1. Envio de Pacote pelo Cliente:

   Quando um cliente envia um pacote do tipo *Packet* com o nome da *stream* desejada no campo *data* e um campo *aux* igual a 1 (indicando que é enviado por um cliente), este vai pela porta **6000**. Quando chega a um *oNode* a sua função é verificar se o endereço *IP* do cliente está na lista *ip_clients*, que mantém os clientes associados a esse *oNode*. Este *oNode*, periodicamente envia pacotes do tipo *ping* para os clientes nessa lista na porta **2000** de **dois em dois segundos**, esperando uma resposta na porta **2001** com um limite de tempo de **um segundo**. Se não houver resposta após três tentativas, o cliente é removido da lista.

   À chegada do pacote ao *oNode*, ele altera o campo *aux* para 0(pois, agora quem envia o pacote é um *oNode*), o campo *hops* é incrementada em mais um, e o *IP* de origem é adicionado à lista dentro do pacote nomeado de *path* que corresponde ao caminho de volta do pacote, o *oNode* também adiciona as redes dos *oNodes* vizinhos à lista *prevNetworks* do pacote e verifica se já está redirecionando a *stream* solicitada. Se não estiver, o pacote é encaminhado para os *oNodes* vizinhos.

Ao receber o pacote, um *oNode* vizinho realiza tarefas semelhantes ao *oNode folha*, exceto que não altera o campo *aux* para 0, pois o pacote já vem de um *oNode*. Ele adiciona as redes que estão no pacote no campo *prevNetworks* na lista *networks* do pacote, para **impedir a geração de loops descontrolados**, visto que, ao enviar para os *oNodes* vizinhos não queremos enviar para os vizinhos com redes que já estejam contidas no campo *networks*. Se o pacote atingir um *oNode* com a *stream* ou se for um *RP*, verifica se há um servidor com a *stream* desejada. Se a *stream* for encontrada, o pacote é enviado de volta pelo lista de *IPs* que foi acumulando no campo *path* pela porta **6001**. Ao fazer o caminho inverso, ele apenas vai adicionar o *IPs* de onde recebeu esse pacote ao campo *pathinv*.

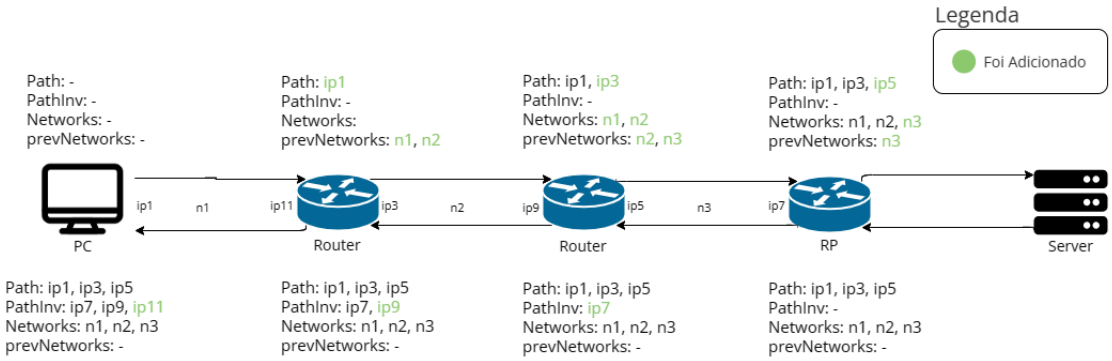

Figure 3: Diagrama do funcionamento das listas *Path*, *PathInv*, *Networks* e *prevNetworks*.

2. Receção e Armazenamento de Respostas pelo Cliente:

   De seguida, o cliente aguarda pacotes de resposta indicando possíveis caminhos até um *oNode* com a *stream* desejada. Esses pacotes são armazenados em uma lista denominada *pacotes*, com um *timeout* de **dois segundos e meio**.

3. Seleção do melhor caminho para o cliente:

   Com base em **métricas** predefinidas pelo grupo, no qual achamos bem em **primeiro ser o número de saltos e , em casos de empate, a latência**, o cliente seleciona o melhor caminho disponível na lista denominada de *pacotes*. Na ausência de caminhos, o cliente aguarda dois segundos e depois tenta localizar a *stream* novamente.

4. Atualização do caminho com partida no cliente:

   Quando o cliente determina o melhor caminho disponível, ele encaminha um pacote, pela porta **7000**, para o *RP* ou o local mais próximo contendo a string desejada, pois este pacote inclui informações sobre o percurso de ida e volta , para atualizar os *oNodes*. No caso de ser *RP*, este ainda terá que enviar uma mensagem ao servidor adequado, para que este lhe comece a fornecer a *stream* desejada.

   Os *oNodes* processam o pacote inversamente, seguindo o caminho inverso(PathInv), e colocam a *stream* contida no pacote à lista *stream* de cada *oNode*. Além disso, eles atualizam o dicionário *bestPath* usando como chave o nome da *stream* e como valores a lista de destinos para encaminhar o conteúdo, presentes no campo *path* do pacote. Simultaneamente, é adicionada à lista *bestPathInv* para especificar a origem do conteúdo, ou seja, qual *oNode* fornecerá esse conteúdo. Essas informações também são extraídas do pacote.

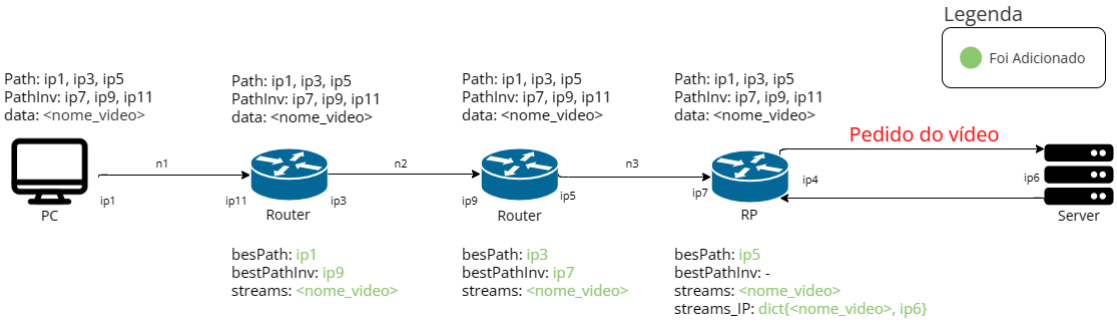

Figure 4: Diagrama de ativação da *stream*.

Após estes passos, o sistema constrói os fluxos necessários para a entrega de dados.

4.5 Definição de um método de recuperação de falhas e entradas de nós adicionais

A recuperação de falhas é um ponto fundamental neste serviço de *streaming*. No nosso ponto de vista, seria impensável que um cliente ficasse sem *stream*, apenas porque um *router* tenha sido repentinamente desligado. Tendo isto em conta, caso um *oNode* que esteja a difundir um conteúdo seja terminado, os *oNodes* adjacentes irão perceber que o mesmo foi terminado através dos pacotes tipo *ping* que serão enviados, mas não desencadearão uma resposta. Desta forma, os *oNodes* fazem uma verificação de dois tipos: caso o *router* terminado estivesse a enviar uma *stream* para o *oNode* em questão, então o *router* propaga um sinal de cancelamento até ao cliente, indicando que o caminho foi interrompido e a necessidade de recalculo de uma nova rota de encaminhamento, (de notar que quando o sinal é propagado, os *oNodes* por onde passa esse pacote, removem os *ips* de encaminhamento dos seus dicionários *bestPath* e *bestPathInv*); se o *router* terminado estivesse a receber uma *stream* de outro, então o *oNode* terminado é removido do dicionário de encaminhamento, e caso fosse o único *router* a receber uma determinada *stream*, então uma mensagem de cancelamento é enviada até ao *RP*, ou até a um *oNode* que tenha mais do que um *ip* no seu *bestPath*. Deste modo, o mecanismo de ativação de rotas mencionado nas secções 4.2 e 4.4, é novamente acionado, e consequentemente, a *stream* volta a chegar ao cliente. Para além dos casos referidos, existe também a possibilidade de um *oNode* adjacente ao cliente terminar inesperadamente, caso isso aconteça, a função *pong* do cliente, que fica à espera de um pacote de reconhecimento do *oNode* folha, dá *timeout*, e entra num ciclo de *retries* com intervalo de 5 segundos, até que o *router* volte a recuperar do termino abrupto.

Outra funcionalidade importantíssima para a escalabilidade de um serviço deste género, é a adição de novos *routers* à rede de *overlay*, com esta funcionalidade conseguimos que hajam novas possibilidades de rotas, rotas estas, que poderão ser mais convenientes para um determinado conjunto de clientes. Assim a rede ficará menos congestionada, e as *streams* chegarão aos clientes em menos tempo.
Como foi mencionado anteriormente, os *routers* são inicializados com ficheiros contendo os *ips* da vizinhança, com esta estratégia, quando um novo *router* é adicionado à rede, o mesmo envia pacotes do tipo *ping*, para os *ips* do seu ficheiro de configuração. Desta forma, os *oNodes* vizinhos recebem esse pacotes e verificam se o *ip* está contido no dicionário *activeRouters*, caso não esteja, o *ip* é imediatamente adicionado. Deste modo, o *oNode* passa a fazer parte da rede de *overlay* e passa a poder participar ativamente na difusão dos conteúdos.

5 Limitações da Solução

Embora o servidor tenha a capacidade de suportar múltiplos arquivos reproduzíveis, a reprodução de vídeos neste sistema é destacada pela eficiência com vídeos codificados no formato *Motion JPEG (Mjpeg)*.

Com base neste precedente, focamos os nossos testes no *movie.Mjpeg* de modo a os simplificar devido a tais conveniências. Além disso, a obtenção de outros conteúdos poder envolver questões legais, reforçando a nossa escolha pelo material fornecido.

A ausência da funcionalidade do botão *Pause* é determinada pela constatação que um cliente que deseje obter uma *stream* não tem a necessidade de recorrer a um botão deste cariz, na nossa opinião. Isto é fundamentado, pelo facto de quando a visualização de uma *stream* é parada e posteriormente retomada, existe um período temporal em que o conteúdo transmitido é perdido e nesse caso, o cliente pode simplesmente sair e voltar a entrar que tem o mesmo efeito.

No entanto, podemos especificar uma maneira de como implementar esta funcionalidade simplista.

Com base no código realizado, acrescentaria-se um novo botão com um *listner* e uma variável booleana, de modo a controlar ação de quando premido e o estado da *stream*, respetivamente.

A partir disto, existem duas possibilidades de implementação:

- Cliente possui um *buffer* que vai armazenando os pacotes da *stream* recebidos, efetuando limpezas quando o mesmo fica cheio;
- Cliente envia um pacote para o*oNode* folha para que este, corte o envio da *stream* para o cliente em questão, podendo este corte, eventualmente ser propagado até ao *RP*, devido ao caminho utilizado, apenas estar a ser usado para satisfazer este mesmo cliente. No caso de retoma, o processo normal de procura de *stream* explicado e fundamentado na secção 4, aconteceria.

Deste modo, viabilizando a existência deste botão equacionado.

6 Testes e resultados

O sistema foi testado em vários cenários, desde casos de utilização com um cliente até situações mais complexas com múltiplos clientes e servidores. Estes testes foram realizados num ambiente de simulação utilizando máquinas virtuais e em computadores físicos ligados à mesma rede local. O sistema demonstrou um bom desempenho em termos de distribuição de conteúdos multimédia, conseguindo lidar com a escalabilidade e dinâmica da rede *overlay* proposta.

Foram efetuados testes de robustez, onde foram simuladas situações de falha de *oNodes*, adição de novos *oNodes*, e situações de sobrecarga da rede. O sistema mostrou-se resiliente a falhas, sendo capaz de recuperar e continuar a entregar conteúdos multimédia mesmo em cenários adversos.

A medição de métricas, como latência e número de saltos, foi realizada durante os testes para avaliar o desempenho do sistema em diferentes cenários. Observou-se que o sistema foi capaz de otimizar a entrega de conteúdos, escolhendo rotas com menor número de saltos e menor latência.

A funcionalidade de *streaming* foi testada com sucesso em diferentes vídeos. O sistema conseguiu lidar eficientemente com a transmissão de *chunks* de vídeo, garantindo uma experiência de visualização fluida para os clientes.

Em resumo, os testes realizados validaram a robustez, eficiência e escalabilidade da solução proposta para entrega de multimédia através de uma rede *overlay*.

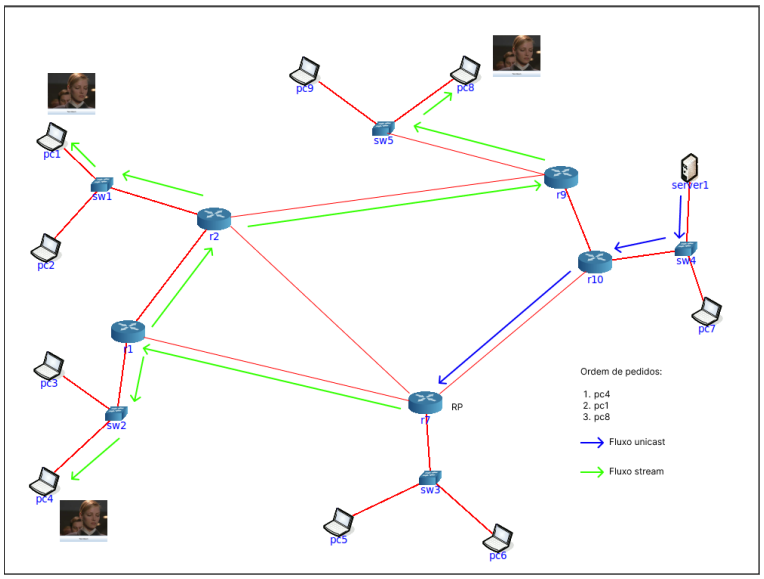

Figure 5: Topologia de teste de conectar 3 clientes.

Neste teste que realizamos, ligamos a rede *overlay* inteira e começamos por pedir *stream* no *pc4*, onde estabeleceu uma conexão com o *RP*, no qual este teve que pedir a *stream* ao servidor que contivesse aquela *stream* e que tivesse as melhores métricas, neste caso, a latência. Seguimos ligando o *pc1*, que dentro dos caminhos possíveis,[r2,r1],[*r2,rp*] ou [*r2,r5,rp*] escolheu o que tinha o menor número de saltos e o que tinha uma menor latência. Por último, pedimos a *stream* a partir do *pc8* que tendo dois caminhos possíveis com o mesmo número de saltos, escolheu o caminho com menor latência, neste caso [r9,r2].

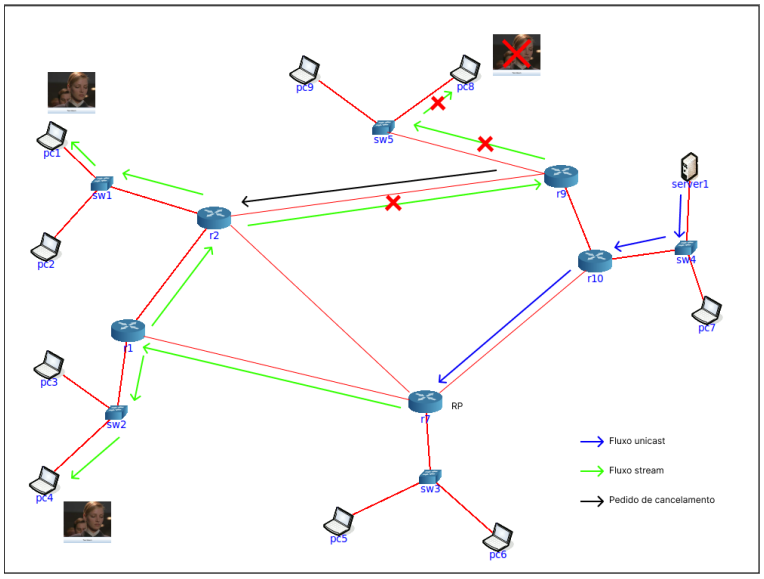

Figure 6: Topologia de teste de cancelar *stream* em um cliente.

Depois de ter os três clientes a receber *stream*, simulamos que o cliente do *pc8*, fechou a stream, logo o cliente deixa de responder aos "*pings*", o que fará que o *oNode* folha tentará comunicar com ele três vezes, e senão responder na última tentativa, ele retirará o seu *IP* da lista *ip_clients* e mandará um pacote para o *IP* que estiver no BestPathInv com a chave do conteúdo que ele estava a receber para cancelar o envio de *chunks* do vídeo para esse *oNode*, pois ele só estava a fornecer *stream* a esse cliente.

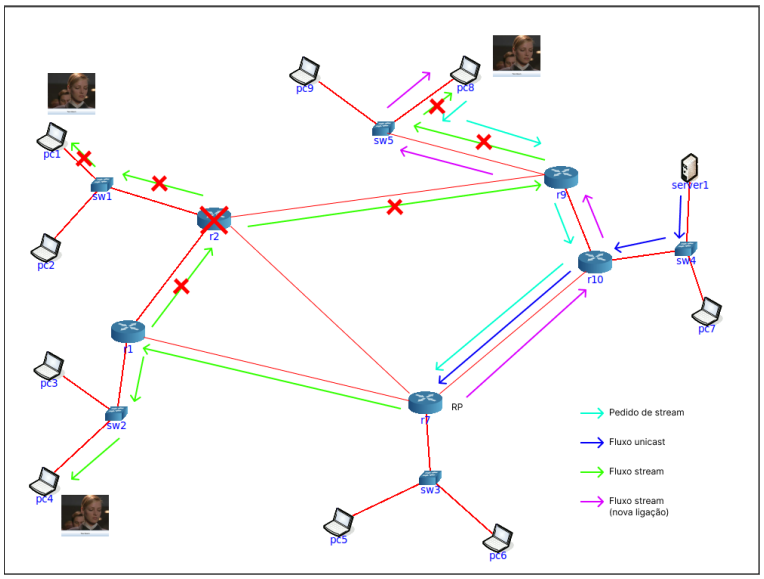

Figure 7: Topologia de teste de um *oNode* terminar inesperadamente.

Elaboramos este cenário de teste para testar a falha de um oNode, ou seja, quando um onde vai a baixo, neste caso o oNode *r2*, o *r1* deixa de enviar *stream* para o *r2*, pois a resposta dos pacotes tipo "ping" deram *timeout* e no caso do *pc1*, enquanto o *oNode r2* não se conectar novamente, ele não consegue receber a *stream*, quando este tornar a ficar ativo ele tornara a receber *stream*, pois este tentará achar uma **nova rota de 5 em 5 segundos** se deixar de responder o *oNode*. Para o caso do cliente *pc8*, este terá uma pausa na *stream* , pois ele terá que recalcular uma nova rota, onde esta será agora pela única possível.

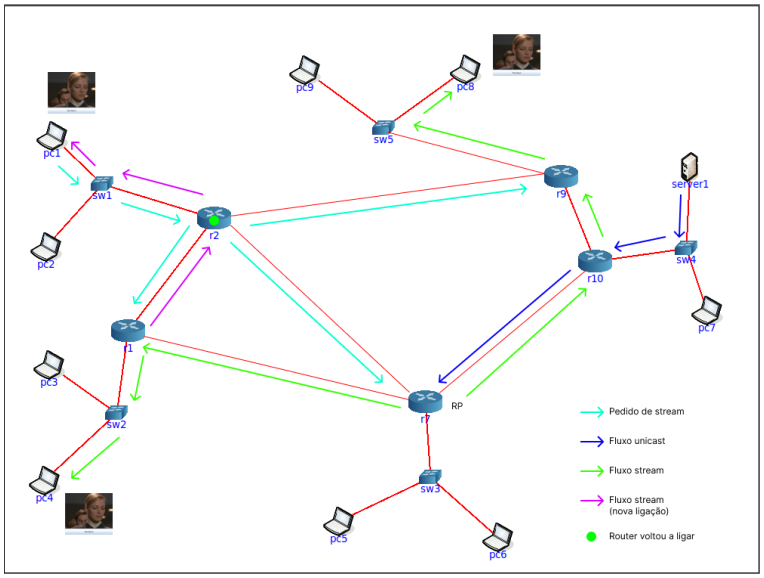

Figure 8: Topologia de teste de ligar *oNode* novamente.

Já neste cenário o objetivo foi testar a reconexão de um *oNode*, nomeadamente o *onode r2*, no qual foi o que foi testado anteriormente a ir abaixo, agora a reconectar, ele vai fazer com que o cliente *pc1* consiga achar um caminho novamente para obter *stream*, e como já expectar, ele continuou a receber *stream* a partir do *chunk* em que o conteúdo está a ser facultado aos outros *oNodes*.

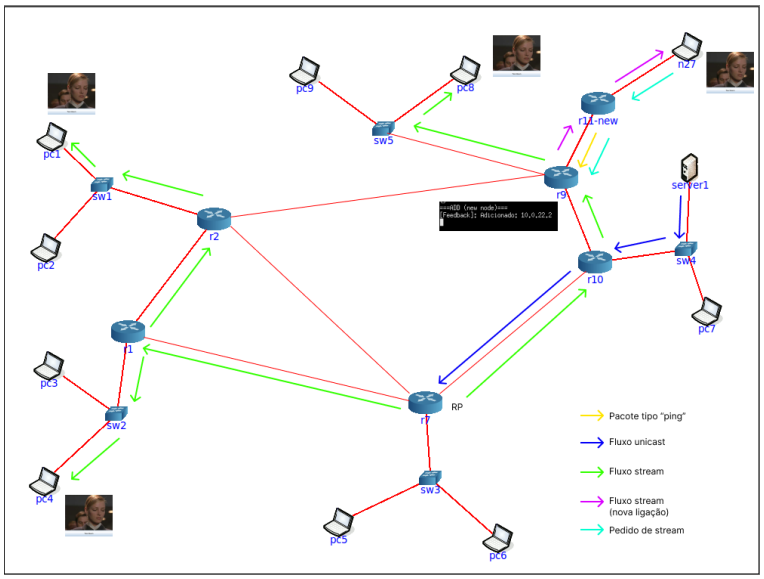

Figure 9: Topologia de teste de adicionar um novo *oNode* à overlay.

Neste cenário de teste, adicionamos um novo *oNode* ('r11-new') à rede de *overlay*. Para que o mesmo possa participar ativamente na árvore de difusão, tal como podemos observar na figura, é enviado um pacote tipo *ping* para o(s) vizinho(s). Desta forma o(s) vizinho(s) poderão adicionar o *ip* do novo *router* à lista de *ips* vizinhos, e consequentemente viabilizar a entrega e receção de conteúdos.

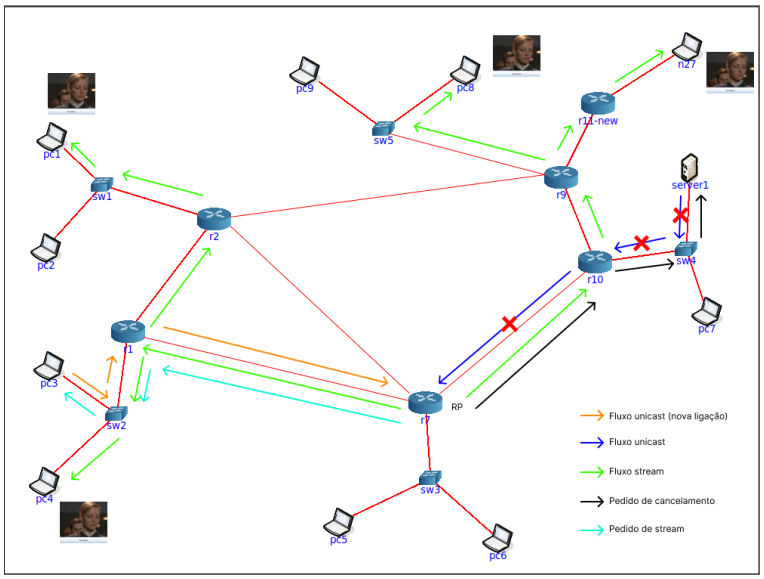

Figure 10: Topologia de teste de monitorizar os servidores.

Este cenário de teste foi construído com o objetivo de testar a monitorização de servidores por parte do *RP*. Neste caso, como podemos verificar, o *servidor1* é terminado inesperadamente, assim sendo, já que o *server2* ('pc3'), tem o mesmo conteúdo que o *server1* estava a transmitir, o *RP* envia um pacote para pedir a *stream*. Este pacote, desbloqueia o fluxo de *stream unicast* até ao *RP*, e consequentemente toda a difusão para a árvore, fazendo com que o clientes voltem a ter acesso ao conteúdo.

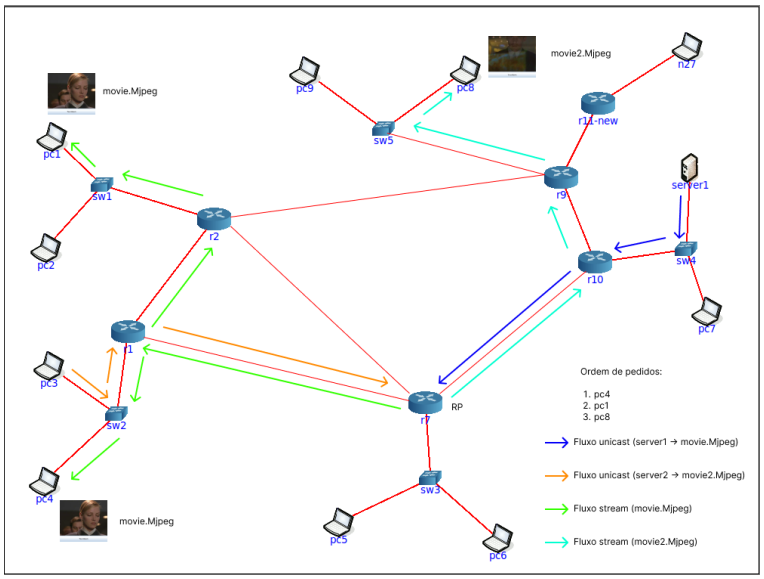

Figure 11: Topologia de teste de fornecer dois conteúdos.

Por último, achamos necessário testar um cenário em que temos múltiplos clientes ativos a pedir *streams* distintas. Neste caso, os clientes 'pc1' e 'pc4' estão a solicitar a *stream movie.Mjpeg*, e o cliente 'pc8' está a pedir a *stream movie2.Mjpeg*. Como podemos observar, na figura acima, os fluxos de *stream* são independentes, já que, os conteúdos também são diferentes. Assim, como podemos verificar este serviço de *streaming* suporta vários conteúdos diferentes simultaneamente.

7 Conclusões e trabalho futuro

Com base no trabalho prático realizado, de modo a num trabalho futuro, melhorar este serviço, achamos que seria interessante explorar a expansão do suporte a formatos de vídeos, aprimorar a interface do utilizador, ou até reforçar a segurança com criptografia.

Em suma, os testes intensos e regulares ao sistema podem tornar o processo de desenvolvimento um pouco cansativo. No entanto quando a fase inicial do trabalho já se encontra consolidada, o trabalho começa a fluir mais naturalmente, gerando um maior proveito das horas dedicadas ao projeto e à matéria teórica lecionada.

Para concluir, agradecendo à docente da aula prática e ao docente da aula teórica, gostaríamos de ressaltar que este trabalho permitiu aprender e consolidar temas teóricos bastante importantes que foram abordados durante o todo semestre.

8 Referências

Livro de apoio fornecido pelos docentes.

Material teórico da UC.