Engenharia de Serviços em Rede
Trabalho Prático Nº2
Serviço Over the Top para entrega de multimédia

Rodrigo Rodrigues, Rui Moreira, and Rui Monteiro

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
email: {pg50726,pg50736,pg50739}@alunos.uminho.pt

Abstract. A utilização de serviços Over-the-top (OTT) tem vindo a crescer ao longo do século XXI, de tal forma que os mais populares como os fornecidos pela Netflix, Amazon Prime Video e Hulu, não são estranhos ao ouvido de qualquer pessoa. Os serviços OTT fazem distribuição de conteúdos pela *internet*, permitindo contornar os problemas de congestão e limitação de recursos da rede de suporte, entregando em tempo real e sem perda de qualidade os media diretamente ao cliente final.

Keywords: *Over the Top* · Services · *multicast* · *overlay* · .

1 Introdução

No âmbito do segundo trabalho prático da UC Engenharia de Serviços em Rede foi concebido, prototipado e elaborado um programa com o objetivo de efetuar *multicast* aplicacional promovendo a eficiência e a otimização de recursos para melhor qualidade de experiência do utilizador. Deste modo, foi necessário considerar diferentes etapas para suportar toda a lógica do programa. Inicialmente, foi construída uma topologia *overlay* a partir da rede *underlay*, onde cada nodo recebe informação do servidor acerca dos seus vizinhos. Uma vez construído o *overlay*, foi necessário identificar os melhores caminhos para o fluxo de dados, desde o servidor aos clientes, e realizar o *streaming* dos dados. Ao longo deste relatório serão detalhadamente descritos todos os passos do desenvolvimento do projeto na linguagem de programação JAVA, bem como os testes realizados no emulador Core.

2 Estratégia da solução

Perante a proposta de implementar um protocolo de *streaming*, decidimos utilizar a linguagem de programação **Java** e o protocolo de transporte escolhido para ser usado no overlay é o UDP *User Datagram Protocol*. O objetivo é que os nós da rede overlay sejam capazes de entregar os conteúdos de forma mais eficiente, isto é, com o menor atraso possível e largura de banda necessária. Para tal, o grupo decidiu implementar os seus protocolos aplicacionais, tanto o de controlo como como o de *streaming*, sob o **protocolo de transporte UDP**.

Isto deve-se ao facto de que o UDP é o protocolo mais apropriado para streaming em tempo real pois, como o UDP é um protocolo simples e não oferece garantia na entrega das mensagens, ao contrário do TCP que é bastante fiável, então não irão ocorrer atrasos para fazer a retransmissão de pacotes.

Apesar de haver risco de perda de informação nos pacotes, UDP apresenta maior rapidez de transferência de dados, porque entrega os dados de forma simples, sem qualquer análise de estado, dando ao programador maior controlo sobre certos aspetos como a decisão de retransmitir pacotes. Considera-se que as eventuais perdas não serão muito significativas e a previsão é que não irão afetar em grande escala a qualidade do streaming.

2.1 *Bootstrapper*

A abordagem escolhida é baseada num controlador, onde se indica apenas um nó *bootstrapper* para arranque da rede, sendo fundamental que o servidor tenha conhecimento de todo o overlay, permitindo determinar quais são os vizinhos do nó considerado como *bootstraper*.

Foi criado um ficheiro de configuração com informação sobre as ligações entre os nodos da rede, ao qual o servidor tem acesso.

2.2 Entidades do programa

Nas classes *Servidor*, *Cliente* e *Router* encontram-se os dados, variáveis e métodos que cada uma destas entidades deve utilizar e/ou armazenar.

Tanto o Cliente como o Router são nodos da tipologia pelo que ambos herdam de uma classe abstrata chamada *Node*.

Relativamente à classe VideoStream, esta permite ao servidor de streaming dividir o ficheiro multimédia a ser enviado em frames e enviar cada frame para a rede. Por sua vez, o protocolo permite ao Cliente unir os frames recebidos de forma sequencial e exibir o resultado final.

3 Especificação dos protocolos

Para concretizar os requisitos estabelecidos no enunciado, foram criadas as classes *Pacote* e *PacoteStreaming* para os protocolos de controlo e de *streaming*, respetivamente.

3.1 Mensagens do protocolo de Controlo (Pacote)

Na presença de uma topologia desta natureza, é importante controlar o fluxo de mensagens de controlo e de dados que a percorrem.

tipoMensagem	
fileInfo	
origem	destino
numSaltos	
ipNodoAnterior	
dados	
startTime	

Table 1. Estrutura do Pacote de Controlo

Onde **fileInfo** indica as ações que um cliente pode realizar sobre o vídeo (iniciar, pausar, ...), indica metadados como o número de sequência que representa a posição de uma frame no vídeo, lista/número de clientes a assistir, etc. Por sua vez, **origem** e **destino** indicam os nomes do primeiro nodo que enviou a mensagem e o nome do destinatário, respetivamente. O campo **numSaltos** indica quantos saltos existem no caminho entre a origem e o nodo onde a mensagem se encontra. O campo de **dados** corresponde ao *payload* (quando um pacote é enviado com o campo de *payload* vazio, o tamanho de *payload* possui o valor zero). Finalmente, startTime é uma métrica temporal utilizada nas mensagens de tipo 1 e mensagens de tipo 6 para calcular o atraso da chegada das mensagens (custo das rotas).

Foi utilizado o formato *Pacote* que lida com diferentes situações que podem ser identificadas pelo campo do **tipoMensagem** de mensagem protocolar:

- **tipo -1**: mensagens de **fim de conexão**, ou seja, quando um nodo da rede se pretende desligar da rede terá que enviar uma mensagem de fim de conexão;
- **tipo 0**: mensagens de **início de conexão**, ou seja, quando um nodo se pretende conectar à rede enviam mensagem ao nodo *bootstrapper*; mensagem de início de conexão ao *bootstrapper*;
- **tipo 1**: mensagens de **routing**, ou seja, após um cliente se conectar à rede, envia uma mensagem de *routing*, propagada por todos os *routers*, com destino ao servidor para descobrir a rota com menor custo até ao *bootstrapper*;
- **tipo 2**: mensagens de **ativação de rota**, ou seja, ativar a rota com o menor custo, entre servidor e cliente;
- **tipo 3**: mensagens de **streaming**, ou seja, mensagens com todo o conteúdo necessário para a transmissão de conteúdo multimédia e pedidos de início e fim do *streaming*;
- **tipo 4**: mensagens de **ativação de vizinhos**, ou seja, informar os vizinhos que o nodo se conectou à topologia;

- **tipo 5**: mensagens de **Ping**, ou seja, um pacote com uma mensagem no campo *payload* para enviar de um nodo para outro;
- **tipo 6**: mensagens de **Ping Periódico**;

3.2 Mensagens do protocolo de *Streaming* (PacoteStreaming)

Para tratar do envio de mensagens do tipo *streaming* multimédia (com identificador 3 no protocolo de controlo), foi utilizado o mesmo pacote do protocolo de controlo. É, no entanto, importante destacar a utilização do campo 'metadados', que só é utilizado neste tipo de mensagens. Como já foi referido, neste campo estão incluídos o tipo de mensagem de *streaming*, o número de sequência do pacote, o número de destinos e o id de cada um desses mesmos destinos.

O tipo pode ser:

- **tipo 0**: mensagens de **início de *stream***, ou seja, quando o cliente clica "play", é enviada uma mensagem ao servidor para iniciar o envio dos dados da *stream*.
- **tipo 1**: mensagens de **pausar *stream***
- **tipo 2**: mensagens de ***streaming* de dados**

3.3 Interações

Fluxo de interações:

1. O Cliente conecta-se e manda um pacote do tipo 0 para o servidor.
2. O Servidor responde com um pacote do tipo 0 com a lista de vizinhos para o Cliente.

3. O Cliente avisa os seus vizinhos ativos que se conectou com um pacote do tipo 4 e envia um pacote do tipo 1 a todos os vizinhos para calcular a sua rota mais rápida até ao servidor.
   (a) Se fosse um *router* a conectar-se, ele apenas avisaria os seus vizinhos ativos que se conectou através dum pacote de tipo 4.

4. O vizinho ativo que recebeu o pacote do tipo 4 volta a enviar um pacote de tipo 1 a todos os vizinhos, de modo a calcular a rota mais rápida até ao servidor.
   (a) Se o vizinho que recebe este pacote tipo 1 for um *router*: Envia um pacote do tipo 1 ao *routers* vizinhos, ou se o seu destino for seu vizinho envia ao destino.
   (b) Se for um servidor, envia um pacote do tipo 2 a todos os seus vizinhos.

5. O *Router* recebe pacote do tipo 2, ativa interface e volta a enviar pacote do tipo 2 para o próximo salto. Se for cliente a receber pacote de tipo 2, mete que já tem rota gerada e não faz nada.
6. O Cliente pressiona o botão de *Play* envia uma mensagem a todos os vizinhos (Pacote de tipo 3 e tipo *streaming multimédia* 0).

7. O *Router* recebe esta mensagem e consulta a tabela de encaminhado de modo a encaminhar a mensagem para o próximo salto.
8. Se for mensagem de *Play*, ele adiciona o IP do Cliente à lista de clientes a transmitir, se for *Pause* retira o IP dessa lista.
9. O Servidor vai enviando a *stream* para os vizinhos enquanto tiver alguém na sua lista de clientes a transmitir.

4 Implementação

4.1 Construção da Topologia Overlay

A primeira etapa crucial para permitir o *streaming* foi a criação da rede *overlay*. Esta rede é construída a partir de um ficheiro de configuração com informações acercas das ligações entre os nodos da rede e cuja leitura é efetuada pelo servidor.

Em cada linha encontra-se o nome do nodo, os seus vizinhos e os correspondentes endereços das interfaces às quais se conecta. A partir deste ficheiro o servidor irá ter conhecimento de toda a rede *overlay*, guardando os dados em memória.

A informação é armazenada num objeto da classe Interface que foi criada com o intuito de conter informação do nome do nodo, o seu estado(ativo/inativo) e o endereço IP da sua interface. Os objetos são guardados num *Map* (nome do nodo para lista de Interface que representa os vizinhos do nodo na rede *overlay*).

4.2 Construção das rotas para o fluxo e monitorização da rede *Overlay*

Depois da estrutura da rede *overlay* estar criada, podemos avançar para a construção das rotas para os fluxos.

A estratégia escolhida para a criação de rotas parte da iniciativa do próprio nodo que se conecta à rede e que deve ser responsável por inundar a rede, enviando uma mensagem de routing (tipo 1) aos seus vizinhos *Router* ativos. Nesta estratégia, as entradas da tabela de encaminhamento dos *Routers* são constituídas pelo id do nodo destino, o ip do próximo salto, o número de saltos e o estado da ligação. Por sua vez, o pacote (tipo 1) contém um campo *startTime* indicativo do instante temporal em que foi enviado, para que depois aquando da receção deste pacote, se possa calcular o tempo que demorou a chegar por um determinado caminho. Tendo a métrica temporal implementada, à medida que os pacotes vão passando pelos *Routers* da rede, estes podem utilizar o tempo para consultarem as suas tabelas de encaminhamento e decidirem qual a melhor entrada, desativando as restantes opções. O tempo representa uma mais-valia em termos de métricas, assegurando melhores resultados do que se obteriam apenas tendo em conta o número de saltos que levou até chegar a cada nodo. É importante notar que existe também um *Ping* periódico a cada intervalo fixo de tempo e que, caso o valor do tempo exceda a média em 100ms, será invocada uma função *gera_rota* para calcular nova rota. Os *routers* devem adicionar novas entradas às suas tabelas de encaminhamento onde o destino será o *idNodo* do nodo

de origem, o *next hop* é o endereço do nodo anterior que enviou a mensagem, o número de saltos é igual ao do pacote recebido já com o valor incrementado e, finalmente, o estado da ligação marcado como inativo. Se a tabela de encaminhamento possuir uma entrada com o destino e *next hop* iguais, escolhe-se a entrada de menor custo temporal.

Em casos de *Routers* que não são vizinhos do servidor, estes devem propagar a mensagem recebida para os seus vizinhos *routers* ativos e sem entradas na tabela de *routing* para o destino em questão.

Em casos de *Routers* que são vizinhos do servidor, estes devem esperar a receção de mensagens de *routing* de todos os seus vizinhos para averiguarem qual a mensagem/caminho com menor custo e comunicarem-no ao servidor.

Tendo inserido novas entradas nas tabelas de encaminhamento dos *Routers* da rede, passamos à ativação de rotas. Para tal, deve-se enviar uma mensagem de ativação de rotas (etiquetada com o *tipoMensagem* 2)no sentido servidor-nodo, percorrendo o caminho determinado anteriormente como o mais curto e, à medida que passa pelos *Routers*, escolhe o próximo nodo cujo custo é menor, passando o estado da entrada corresponde na tabela de encaminhamento a estar "ativo". O processo de construção de rota termina quando o nodo receber o pacote.

4.3 Streaming

A última etapa da realização deste trabalho foi o *streaming* dos dados por parte do servidor. Para a realização da mesma tomamos por base o código fornecido pela equipa docente, aplicando as adaptações necessárias para que o servidor fosse capaz de ler o ficheiro de vídeo e enviá-lo a um cliente que, por sua vez, é capaz de receber esses pacotes e reproduzir o vídeo numa janela.

O acionamento do botão '*play*' na interface gráfica do Cliente desencadeia um pedido ao servidor para que este lhe comece a enviar os dados da *stream* (mensagem de *streaming* tipo 0). Ao receber um pedido desta natureza, o servidor adiciona o id do cliente à lista de clientes que estão a receber conteúdos.

A cada intervalo fixo de tempo, é invocado o método *actionPerformed* que obtém a próxima frame do vídeo, avança o apontador do ficheiro e, caso existam clientes a receber conteúdos, enviar um pacote de streaming (mensagem tipo 3) com o *numSequencia* (número da frame), dados da frame (*payload*) e uma lista de destinos a enviar, ou seja, os clientes que estão a visualizar a *stream*.

Aquando da receção deste pacote, os *routers* da rede devem direcioná-lo para o destino menos custoso segundo indicação da tabela de encaminhamento. Havendo a possibilidade de existir mais do que um destino, será enviado um pacote para cada vizinho correspondente cujos destinos tenham o mesmo próximo salto na tabela de encaminhamento, evitando tráfego redundante.

A receção da transmissão da imagem por parte do Cliente deve ser feita pela ordem que o número de sequência indica e não pela sua ordem de chegada. Para concretizar este requisito usa-se um contador de mensagens recebidas, *imageCounter*, e uma variável indicativa da imagem atual, *currentImage*. A variável *currentImage* é inicializada com o valor do número de sequência do primeiro

pacote recebido e à medida que é mostrada a *frame* respetiva ao utilizador o seu valor é incrementado.

É feita a utilização da estrutura de dados *Map* (numSequencia -> Pacote) para guardar os pacotes. Desta forma para obter a próxima *frame* é apenas necessário obter o pacote correspondente à chave *currentImage*.

Por fim, quando um utilizador pretende parar de ver a *stream*, é enviado um pacote com tipo de mensagem de *streaming* -1 ao servidor e este retira da lista *clientStreaming* o id do cliente respetivo.

4.4 Servidor Alternativo

Foi implementada ainda a funcionalidade do servidor alternativo no nosso serviço de entrega de *streaming*. Essencialmente, os servidores vão enviando *Pings* Periódicos aos clientes a cada intervalo de tempo de 5s, e os clientes guardam os valores dos atrasos para efeitos de cálculo de médias. O Cliente pode então comparar os valores das médias de ambos os servidores e passar a utilizar o servidor alternativo, caso este apresente melhores valores.

É de notar também que, esta troca de servidores acontece automaticamente (sem ser preciso pressionar o botão de pause e play como acontecia na defesa), ficando assim esta mudança transparente ao cliente.

5 Testes e Resultados

5.1 Cenário 2 - Início, Monitorização e Fim de conexão

Na imagem abaixo encontra-se a rede *overlay* utilizada para o teste de início de conexão.

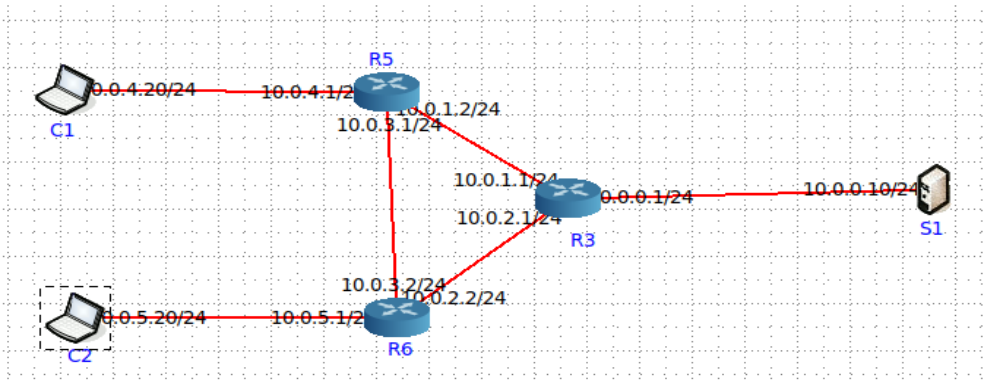

Fig. 1. Cenário 2 - Overlay com 6 nós (um servidor, dois clientes, 3 nós intermédios)

Na figura abaixo é possível observar o processo de início de conexão em que os nodos solicitam ao *bootstrapper* conexão à rede *overlay*, recebem uma resposta do mesmo com os seus vizinhos e enviam uma mensagem a notificar a sua conexão àqueles que se encontram ativos. Os clientes, além de iniciarem a conexão, também requerem a construção da sua nova rota.

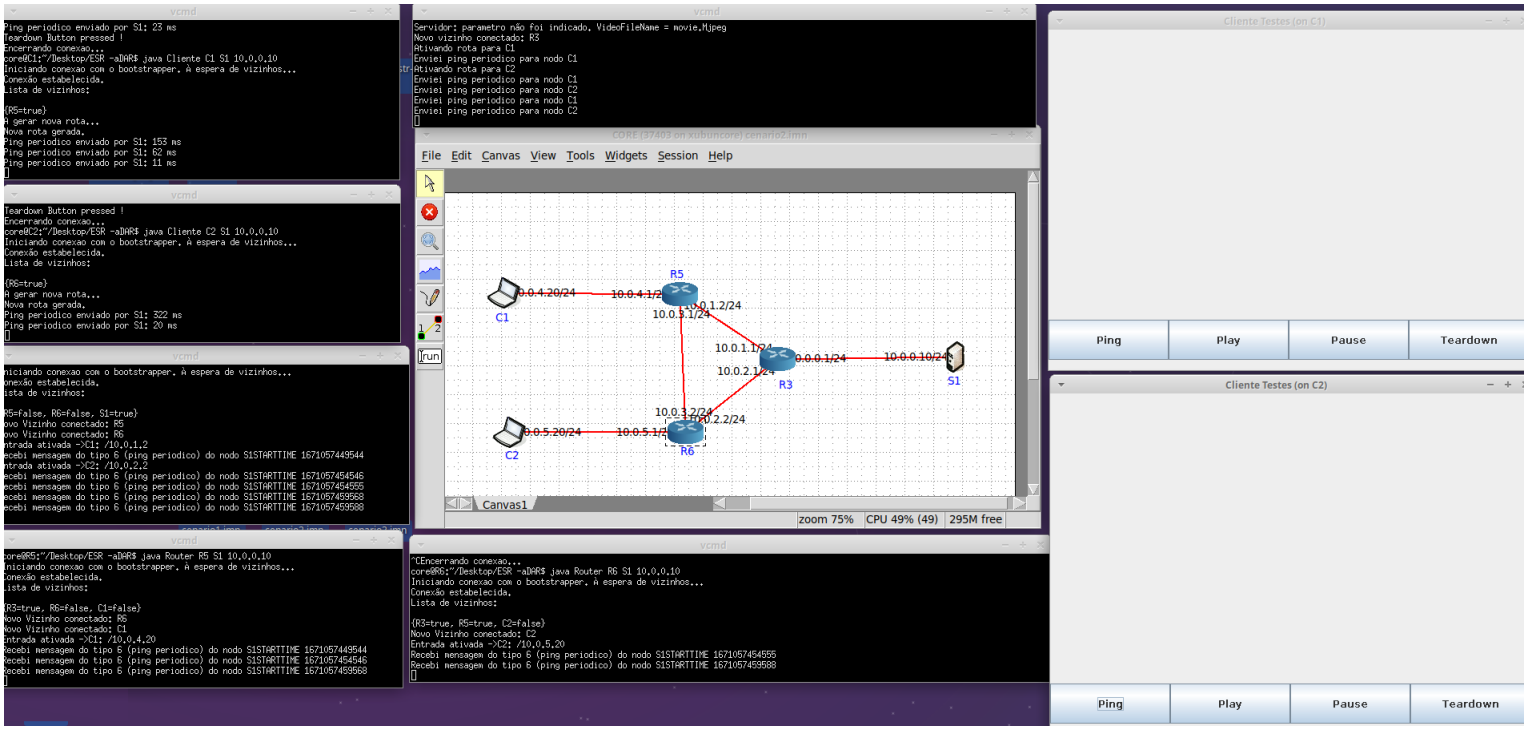

Fig. 2. Teste de início de conexão

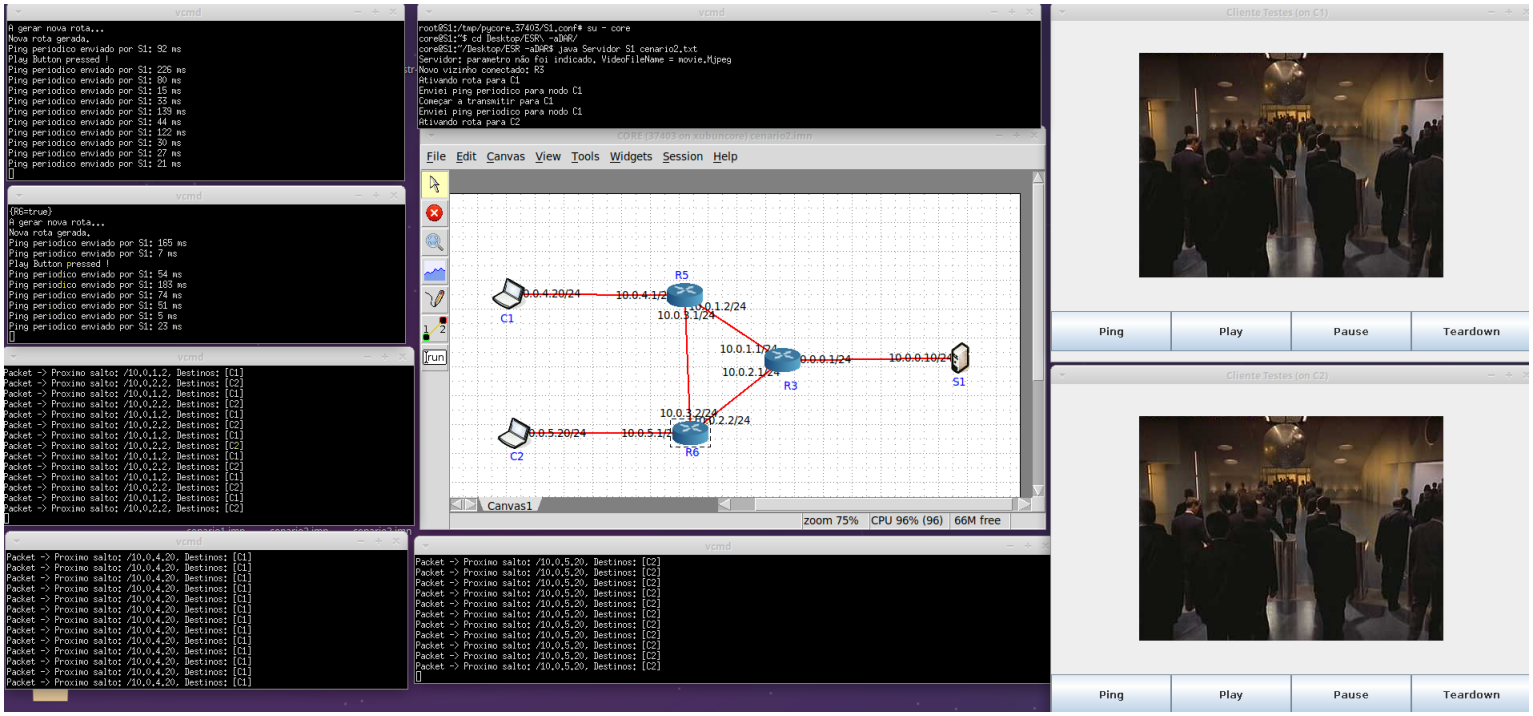

Fig. 3. Teste sem atrasos nas ligações da topologia

No teste acima, os dados dirigidos a ao cliente C1 passam nos *Routers* R3 e R5. No próximo teste de monitorização da rede, ilustrado abaixo, introduziu-se um atraso (500ms) na rota de R3 para R5. Face a este atraso, é possível observar que acontece uma adaptação da rede pois foi gerada uma nova rota para o cliente C1 com menor custo temporal, passando agora por R3 -> R6 -> R5 -> C1.

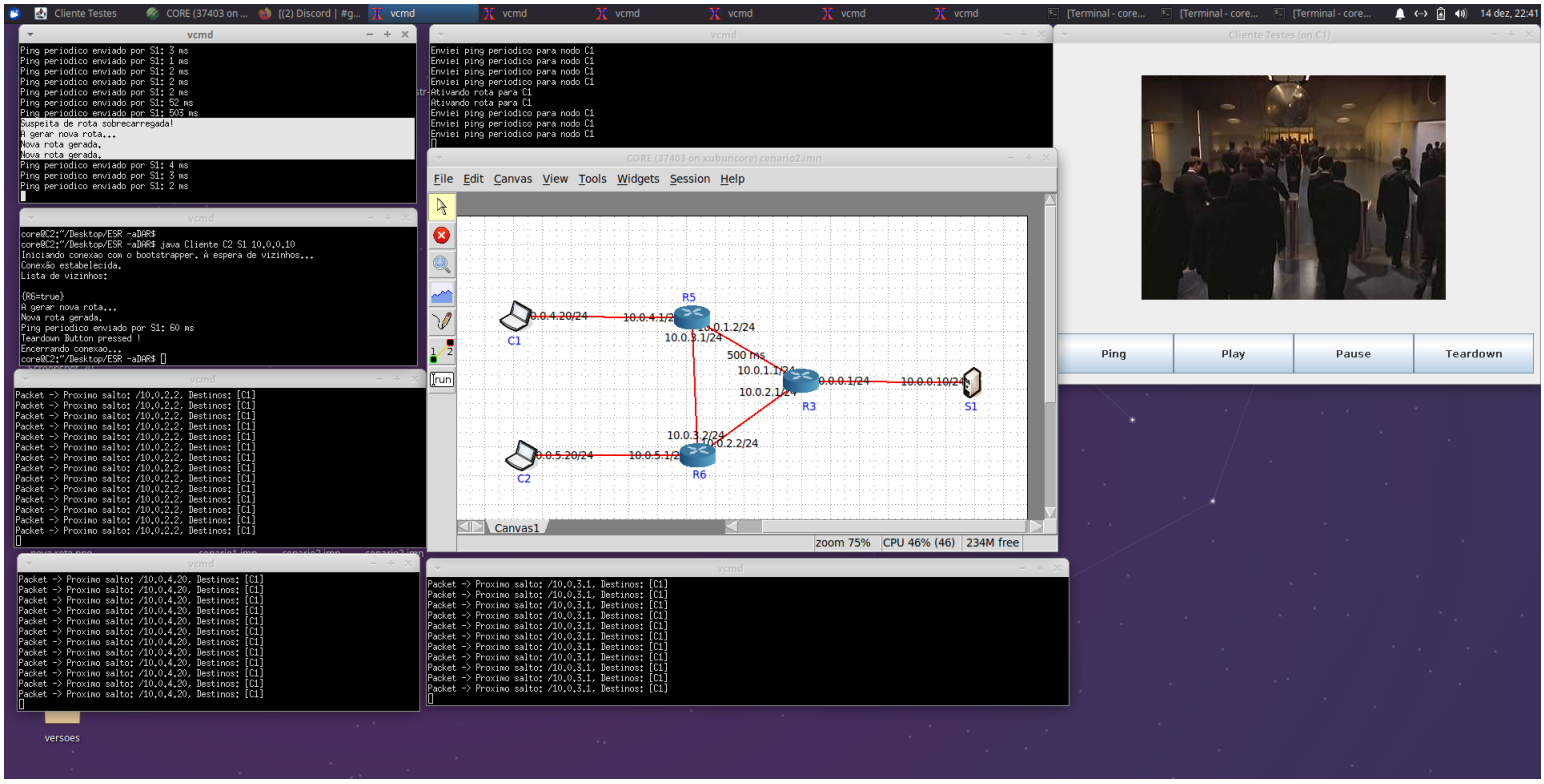

Fig. 4. Teste de monitorização de rede

5.2 Cenário 3 - Streaming de Multimédia

Como se pode verificar na figura abaixo, está a ser feita a transmissão do vídeo para todos os clientes sem haver tráfego redundante a passar nas ligações intermédias.

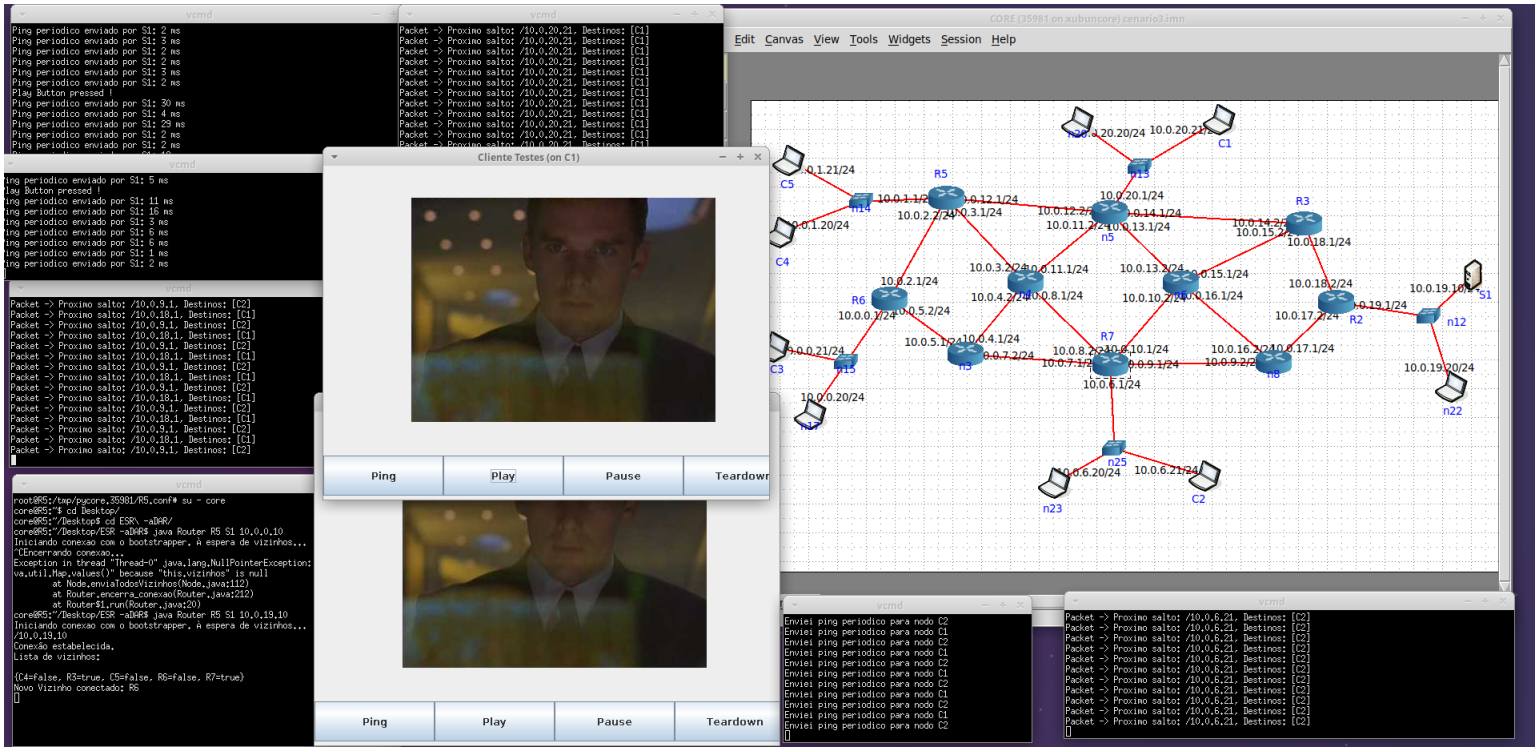

Fig. 5. Teste de streaming de multimédia no cenário 3

6 Conclusões e trabalho futuro

A realização deste trabalho prático permitiu ao grupo consolidar todo o conhecimento referente a protocolos e serviços de *streaming* tanto multimédia como *multicast*, lecionado ao longo da unidade curricular.

Destacamos como fatores positivos a geração de melhores rotas com base na métrica temporal, monitorização e adaptação a alterações na topologia.

Em termos de melhorias no serviço de *streaming*, consideramos que poderia ser implementado um mecanismo de controlo de perdas, de modo a conseguir monitorizar se as mensagens chegam ao destino através da utilização de um sistema de mensagens com o auxilio de ACK's (acknowledgements).