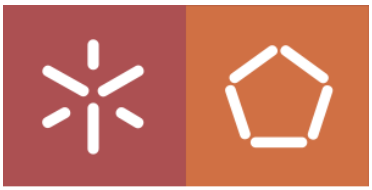


Universidade do Minho
Escola de Engenharia

MESTRADO EM ENGENHARIA INFORMÁTICA

ENGENHARIA DE SERVIÇOS EM REDE

TRABALHO PRÁTICO Nº 3

SERVIÇO *OVER THE TOP* PARA ENTREGA DE MULTIMÉDIA

Grupo 6 – PL1

ANA FILIPA PEREIRA - PG46978

CAROLINA SANTEJO - PG47102

RAQUEL COSTA - PG47600

ÍNDICE

I. INTRODUÇÃO

Neste terceiro trabalho prático da unidade curricular de Engenharia e Serviços de Rede foi pedido ao grupo que desenvolvesse um protótipo de entrega de ficheiros de vídeo, sendo estes enviados por um servidor para um dado conjunto de clientes. Isto forma uma rede *overlay* aplicacional na qual existem nodos intermédios responsáveis por reenviar os dados. Desta forma, pretende-se um envio eficiente dos ficheiros, sem que haja *delays* consideráveis ou uso elevado de recursos tais como largura de banda. Além disto, pretende-se que não seja gerado tráfego redundante, ou seja, caso vários clientes peçam o mesmo ficheiro, se na rota entre o servidor e esses clientes existirem ligações comuns, então nelas apenas pode passar um único fluxo de dados.

Por outro lado, o grupo considera que o *bootstrapper* trata-se do servidor de *streaming* e que este recorre a um ficheiro de configuração para ter conhecimento inicial da rede *overlay*.

É também de realçar que nesta rede *overlay* é necessário determinar os melhores caminhos para o fluxo de dados, desde o servidor até aos clientes. Para isto o grupo considerou o número de saltos como métrica.

Além disto, é de salientar que o grupo, no desenvolvimento do projeto, utilizou a linguagem de programação Java.

Ao longo deste relatório serão detalhadamente descritos todos os passos do desenvolvimento do projeto bem como os testes realizados no emulador Core.

II. ARQUITETURA DA SOLUÇÃO

A. MODELAGEM DA SOLUÇÃO

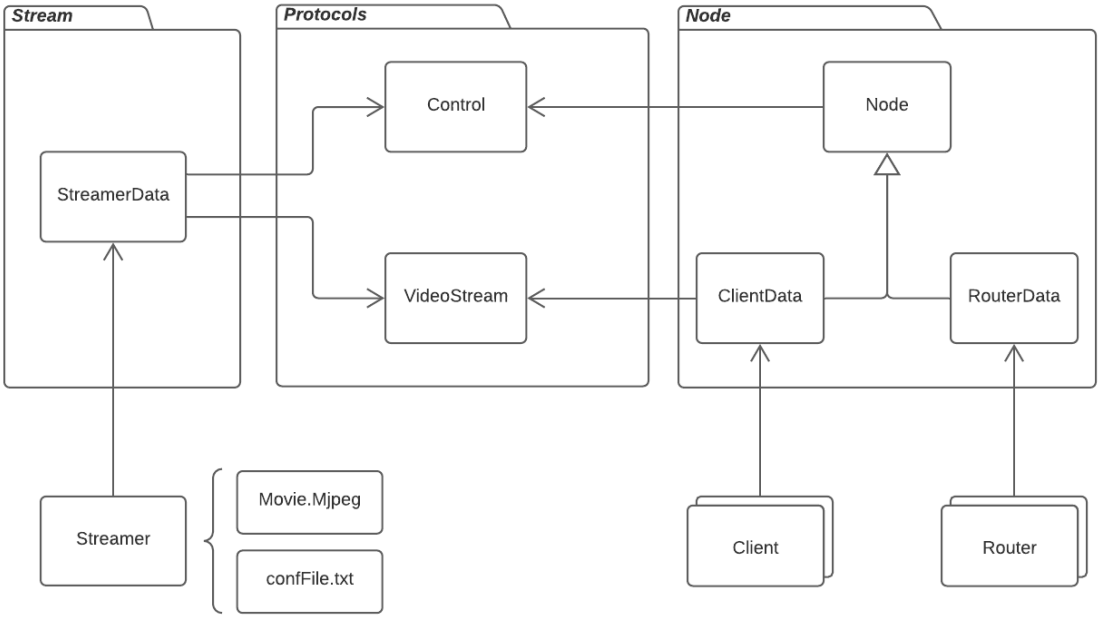


FIGURA 1 - DIAGRAMA GERAL DA SOLUÇÃO

A solução final obtida pelo grupo tem por base as várias estratégias apresentadas pela equipa docente no decorrer das aulas tal como aquelas apresentadas no enunciado do presente trabalho prático. A Figura 1 - Diagrama Geral da Solução*Figura 1* tem como objetivo representar um nível mais abrangente da solução desenvolvida através de um diagrama simples e conciso.

Primeiramente, como foi decidido optar por uma abordagem baseada num controlador, isto é, indicar apenas um nó (*bootstraper*) como contacto para arranque da rede, é fundamental que o servidor tenha conhecimento de todo o *overlay.* Desta forma, ele irá conseguir determinar quais são os vizinhos do nó considerado como *bootstraper*. Assim sendo, foi necessário criar manualmente um ficheiro de configuração que contenha toda a informação de quem se deve ligar a quem, ao qual o servidor tenha acesso. Exemplo: *confFile.txt.* Além disso, o servidor/*streamer* irá ter acesso também ao ficheiro de vídeo que deverá transmitir aos clientes.

É de notar que tanto o Servidor como o Cliente e o Router têm acesso às respetivas classes do tipo “Data”, isto acontece de forma a organizar o código desenvolvido, separando os dados, as variáveis que cada um deve armazenar, e os vários métodos utilizados, da parte que corre a aplicação no nodo da rede. Além disso, tal como está representado no diagrama, existem múltiplos clientes e routers de acordo com a topologia que está a ser utilizada.

Tanto o cliente como o router são nodos da tipologia e, portanto, a nível aplicacional há coisas que ambos têm em comum, daí a criação de uma classe abstrata chamada *Node* e tanto a classe *ClientData* como a *RouterData* herdarem essa mesma classe.

De seguida, é imprescindível destacar as duas classes que retratam tanto o protocolo aplicacional de controlo como o protocolo de *streaming,* e essas são a *Control* e a *VideoStream*, respetivamente.

A classe *Control*, responsável pelo protocolo aplicacional de controlo, é importada tanto pela classe *Node* como pela classe *StreamerData*, uma vez que cada nodo da rede *overlay* deverá correr este protocolo de modo a ser possível enviar/receber pacotes e mensagens, permitindo também a construção de rotas para os fluxos.

No caso da classe *VideoStream*, por um lado, é também importada pela classe *StreamerData*, e por outro, é apenas importada pela classe *ClientData*. Isto acontece, porque este protocolo tem como objetivo dividir o ficheiro multimédia a ser enviado por *frames*, e então o servidor de *streaming* através do protocolo efetua essa divisão e vai enviando para a rede cada *frame*. Por parte do cliente, o protocolo permite unir os *frames* recebidos de forma sequencial e exibir o resultado final.

B. PROTOCOLO DE TRANSPORTE

Um serviço *Over the Top* para entrega de multimédia, é um serviço responsável por fazer *streaming* sobre a rede IP pública. Para tal, é formada uma rede *overlay* própria, assente em cima de um protocolo transporte como o UDP ou TCP, e/ou aplicacional da Internet (HTTP).

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, uma vez que consegue enviar pacotes a uma taxa constante, independentemente de congestões na rede ou da capacidade da aplicação de os conseguir receber, ao contrário do que acontece com o TCP. Além disso, é de notar que o *multicast* de IP reduz significativamente os requisitos de largura de banda, algo que o TCP impede de usar e o UDP não.

Sendo assim, 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. Estas características fazem dele um protocolo adequado para aplicações que decorram em tempo real.

Em última nota, é fundamental referir que é necessário considerar certas características para os protocolos aplicacionais de *streaming* que são construídos sob UDP, e que estes mesmos têm de ser capazes de executar as seguintes tarefas: Fornecer os comandos de reproduzir e parar; Fornecer os meios para entrega de múltiplos *streams*, detetando possíveis pacotes perdidos; Sincronizar diferentes *streams* de *media* numa base de tempo partilhada em tempo real e, além disso, conseguir sequenciar os pacotes.

III. ESPECIFICAÇÃO DOS PROTOCOLOS APLICACIONAIS

A. PROTOCOLO DE CONTROLO

1. FORMATO DAS MENSAGENS PROTOCOLARES

De forma a ser possível enviar dados ao longo da rede foi necessário criar uma classe *packet* responsável por essa transmissão de informação. É de realçar que um pacote transmite um dado tipo de mensagem sendo que é esta que determina os campos que esse pacote irá possuir. Todos os campos que um *packet* pode ter estão representados no diagrama da figura abaixo.

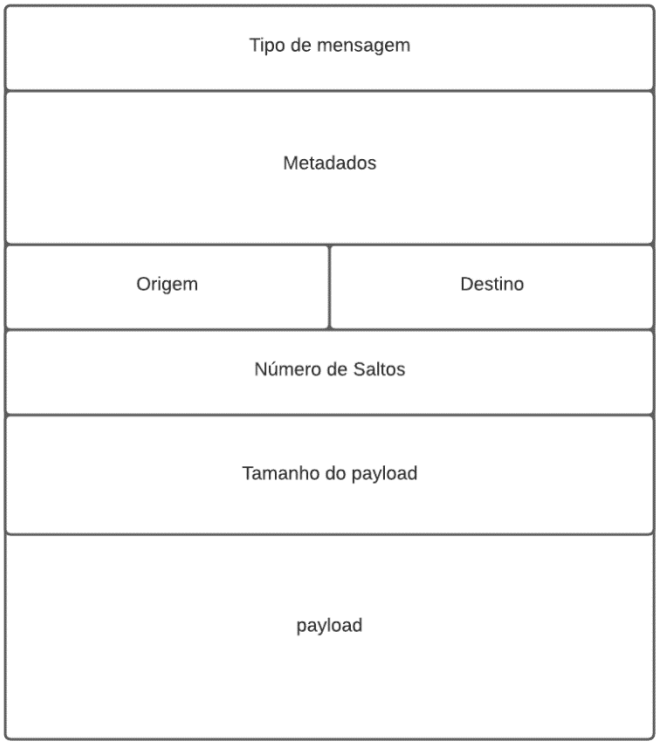


FIGURA 2 - ARQUITETURA PACKET

Em primeiro lugar temos o campo do tipo de mensagem. Neste campo é passado um número inteiro que representa o tipo da mensagem que se pretende enviar. Uma mensagem pode ser um dos seguintes tipos: início de conexão, fim de conexão, mensagem de *routing*, ativação de *routing*, *streaming* de multimédia, ativação de vizinhos ou *ping*.

O campo Origem indica o nome do primeiro nodo que enviou a mensagem e o campo Destino indica o nome do destinatário dessa mesma mensagem. Por outro lado, o campo Número de Saltos indica quantos saltos existem no caminho entre a origem e o nodo atual, ou seja, o nodo onde a mensagem se encontra. Já o campo Metadados possui informações relativas às ações que um cliente poderá ter sobre o vídeo, tais como pausar e iniciar. É também neste campo que se guarda um *sequence number* que simboliza a posição de uma *frame* no vídeo. Além disto, nos Metadados existe uma lista (e o seu respetivo tamanho) com todos os clientes

que estão a assistir à *stream* atual. É de salientar que esta lista apenas está preenchida quando a mensagem é do tipo *streaming* de multimédia. Para outros tipos de mensagem não é relevante ter esta informação.

Por fim, num pacote, existe também o campo que indica o tamanho do *payload*, e o campo do próprio *payload*, no qual é enviado diversos tipos de informações pertinentes. É de realçar que quando um pacote é enviado com o campo de *payload* vazio, o tamanho de *payload* possui o valor zero.

2. INTERAÇÕES (COM O PROTOCOLO)

Como já foi referido existem vários tipos de mensagem, os quais serão definidos ao longo deste tópico.

- **Início de conexão**

  Quando um cliente ou um router se pretende conectar à rede terão que enviar uma mensagem de início de conexão ao *bootstrapper*. Num dado pacote que envia esta mensagem, o início de conexão é representado pelo inteiro 0.

- **Fim de conexão**

  Quando um nodo da rede se pretende desligar da rede terá que enviar uma mensagem de fim de conexão, sendo que os destinatários dependem se o nodo a desconectar é um cliente, router ou o próprio servidor. Este tipo de mensagem é identificado pelo inteiro -1.

- **Mensagem de *Routing***

  Após um cliente se conectar à rede, este envia uma mensagem de *routing*, que será propagada por todos os routers, e com destino ao servidor de forma a descobrir a rota com menor número de saltos entre esse cliente e o *bootstrapper*. Este tipo de mensagem é identificado pelo inteiro 1.

- **Ativação de Rota**

Quando uma mensagem de *routing* chega ao servidor significa já foram preenchidas todas as tabelas de *routing* dos routers, ou seja, já é possível obter uma rota para o cliente que se pretende conectar. Deste modo, é assim enviada uma mensagem com o identificador 2, que será responsável por ativar a rota com o menor número de saltos, tanto no sentido servidor -> cliente como no sentido cliente -> servidor.

- ***Streaming* de multimédia**

Para o envio de mensagens relacionadas com o *streaming* de multimédia é utilizado o identificador 3. Estas mensagens terão todo o conteúdo necessário para a transmissão de conteúdo multimédia e pedidos de início e fim do *streaming*.

- **Ativação dos vizinhos**

Quando um nodo se conecta na rede, é-lhe indicado pelo servidor todos os seus vizinhos incluindo a informação de aqueles que estão conectados na rede. No entanto, para que os vizinhos do nodo em questão saibam que este já se conectou é necessário que lhes seja

dada essa informação. Desta forma, é utilizada a mensagem com o identificador 4 para dizer que um nodo vizinho foi ativado na rede.

- ***Ping***

Para ser possível realizar alguns testes, implementou-se um tipo de mensagem *ping*. Este tipo de mensagem é identificado pelo inteiro 5 e o pacote que a envia possui no campo *payload* uma mensagem que se pretende enviar de um nodo ao outro.

B. PROTOCOLO STREAMING

1. FORMATO DAS MENSAGENS PROTOCOLARES

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.

2. INTERAÇÕES (COM O PROTOCOLO)

- **Iniciar *stream***

Quando o utilizador pressiona o botão 'play' da sua interface gráfica é enviada uma mensagem com identificador 0 (tipo de mensagem de *streaming*) destinada ao servidor, com o objetivo de iniciar o envio dos dados da *stream* que está a ser transmitida pelo mesmo.

- **Pausar *stream***

Quando o utilizador pressiona o botão 'pause' da sua interface gráfica é enviada uma mensagem com identificador 1 (tipo de mensagem de *streaming*) destinada ao servidor, com o objetivo de parar o envio dos dados da *stream* que está a ser transmitida pelo mesmo.

- ***Streaming* de dados**

Quando o servidor envia os dados referentes à sua transmissão de multimédia, envia pacotes com o identificador 3 (tipo de mensagem de *streaming*), para o(s) cliente(s) aos que até ao momento pretendem assistir a transmissão.

IV. IMPLEMENTAÇÃO

A. CONSTRUÇÃO DA TOPOLOGIA *OVERLAY*

Na primeira etapa do desenvolvimento da solução, foi necessário começar por construir a topologia *overlay*. A estratégia seguida pelo grupo começou por definir um ficheiro de configuração com a informação sobre quem se deve ligar a quem, seguindo a estrutura do seguinte exemplo:

S1: n3,10.0.0.1;
n3: S1, 10.0.0.10; n6, 10.0.2.2; n5, 10.0.1.2;
n5: c1, 10.0.4.20; n3, 10.0.1.1; n6, 10.0.3.2;
n6: c2, 10.0.5.20; n3, 10.0.2.1; n5, 10.0.3.1;
c2: n6, 10.0.5.1;
c1: n5, 10.0.4.1;

Tal como se pode verificar em cada linha está representado o nome do nodo e à frente 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 em memória através de um *HashMap,* os dados indicados neste mesmo ficheiro.

Para armazenar essa informação foi primeiramente criada uma classe *Interface* que guarda o nome do nodo, o endereço IP da respetiva interface, e, ainda, o estado desse nodo, podendo o nodo estar ativo ou inativo. Assim sendo, a chave do *HashMap* será o nome do nodo e o valor será uma lista de Interfaces que representam os vizinhos desse nodo na rede *overlay*.

B. CONSTRUÇÃO DAS ROTAS PARA OS FLUXOS

Na segunda etapa do desenvolvimento da solução, foi necessário definir uma estratégia para a construção das rotas entre os nodos da rede e o servidor. A métrica utilizada para definir a escolha da melhor rota foi o número de saltos. Para perceber o funcionamento desta funcionalidade é importante referir o formato da tabela de *routing* de um router da rede *overlay*. Cada entrada desta tabela é constituída pelo id do nodo destino, o ip do próximo salto, o número de saltos e o estado da ligação.

Em primeiro lugar, para não sobrecarregar o servidor, foi definida uma estratégia em que é o próprio nodo que se conecta que será responsável por inundar a rede com uma mensagem de routing (tipo 1). Esta mensagem (constituída pelos campos 'tipo de mensagem', 'origem', 'destino' e 'número de saltos') será enviada a todos os seus vizinhos (ativos) que sejam routers. Por sua vez, todos os routers que recebem esta mensagem vão, em primeiro lugar, incrementar o campo 'número de saltos'. De seguida vão atualizar a sua tabela de routing adicionando uma nova entrada em que o destino é o id do nodo 'origem', o próximo salto é o ip do nodo anterior que enviou a mensagem (obtido a partir do datagrama UDP), o número de saltos será o correspondente ao do pacote recebido já incrementado e por fim o estado desta ligação que para já estará inativo. Caso já exista uma entrada com o mesmo destino e próximo salto será escolhida aquela cujo número de saltos for menor. Por fim, caso o router em questão não seja vizinho do servidor propaga a mensagem recebida com o novo número de saltos a todos

os routers vizinhos desde que estes estejam ativos e não haja nenhuma entrada na tabela de *routing* para destino requerido com o seu ip (para evitar ciclos). Se o router for o vizinho do servidor, este ficará à 'espera' de que cheguem as mensagens de *routing* de todos os seus vizinhos, e no final vai entregar a mensagem com o menor número de saltos ao servidor que é o destino final.

Neste momento estão preenchidas as tabelas de *routing* de todos os routers com destino ao nodo que conecta, no entanto não existe nenhuma rota ativa para o mesmo. Deste modo, é enviada no sentido contrário (servidor->nodo) uma mensagem do tipo 4 (ativação de rota), que irá percorrer agora o caminho mais curto o servidor para o nodo escolhendo para cada router o próximo nodo cujo custo é menor. Em cada router por onde esta mensagem passa é alterado o estado da respetiva entrada da tabela de *routing* para ativo. Quando o nodo recebe o pacote está terminada a construção da sua rota.

C. STREAMING

Na última etapa do desenvolvimento da solução foi pedido o desenvolvimento de um servidor capaz de ler o vídeo de um ficheiro e o enviar em pacotes, numerados, para a rede *overlay* e um cliente que seja capaz de receber esses mesmos pacotes e reproduzir o vídeo numa janela. Desta forma, foi necessário adaptar o código java fornecido pelos docentes (constituído pela interface gráfica e segmentação do ficheiro em *frames*) ao que já tinha sido feito nas etapas anteriores.

Quando um cliente pressiona o botão 'play' da sua interface gráfica, é enviado um pedido ao servidor, para que este lhe comece a enviar os dados da *stream*. Este pedido é feito utilizando uma mensagem de streaming com o identificador 0 com destino ao servidor. Quando o servidor recebe um destes pedidos adiciona o id do cliente respetivo a uma lista com todos os clientes que estão a visualizar a *stream (clientStreaming)*.

Consequentemente, o servidor terá um objeto do tipo *Timer,* que será responsável por, a cada intervalo fixo de tempo (tempo de uma frame), invocar um método (*actionPerformed*). Este método irá obter a próxima frame do vídeo, avançar o apontador do ficheiro e, caso existam clientes na lista *clientStreaming,* enviar um pacote de streaming com o identificador 3 (tipo de mensagem de streaming) com o número da *frame* (número de sequência), os dados relativos à *frame* lida (payload), e a lista de todos os destinos aos quais deve ser enviado (*clientStreaming*).

De seguida, cada router que receba este pacote deve direcioná-lo para o destino mais perto segundo a sua tabela de *routing.* Assim sendo, como neste caso pode existir mais do que um destino será enviado para cada vizinho correspondente, um pacote cujos destinos tenham o mesmo próximo salto na tabela de *routing*. Desta forma é evitado o tráfego redundante e a ineficiência.

Por fim, quando o pacote chega ao cliente, é necessário que a transmissão da imagem recebida seja pela ordem indicada pelo número de sequência e não pela sua ordem de chegada. Desta forma, foram definidas duas variáveis (*imageCounter* e *currentImage*) onde a primeira é um contador que indica quantas mensagens foram recebidas e a segunda indica o número da imagem atual. O *currentImage* será inicializado 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. Todos os pacotes serão guardados num *Map* em que a chave é o número de sequência e o valor é o pacote associado. Desta forma para obter a próxima *frame* é apenas necessário obter o pacote correspondente à chave *currentImage*. Para evitar que este identificador seja incrementado até valores que não correspondem aos pacotes enviados pelo servidor, é utilizado o contador *imageCounter* e, após 20 mensagens obtidas do map, o valor da *currentImage* não é incrementado, mas sim igualado novamente ao número de sequência do próximo pacote recebido.

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.

D. BIBLIOTECAS DE SUPORTE USADAS

- ***java.net.DatagramSocket*** : Canal UDP utilizado
- ***java.net.InetAddress*** : Resolução de nomes e IP's
- ***java.awt.event*** : Criação de *handlers* e *listeners* de acordo com os eventos especificados

V. TESTES E RESULTADOS

A. INÍCIO DE CONEXÃO

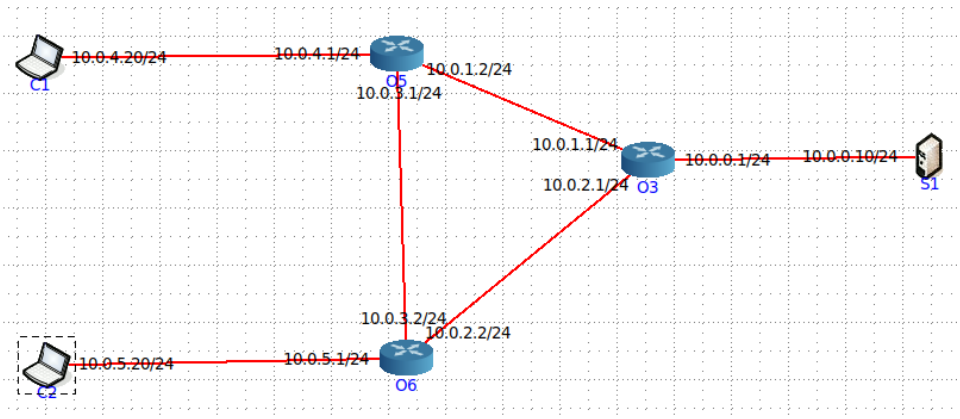
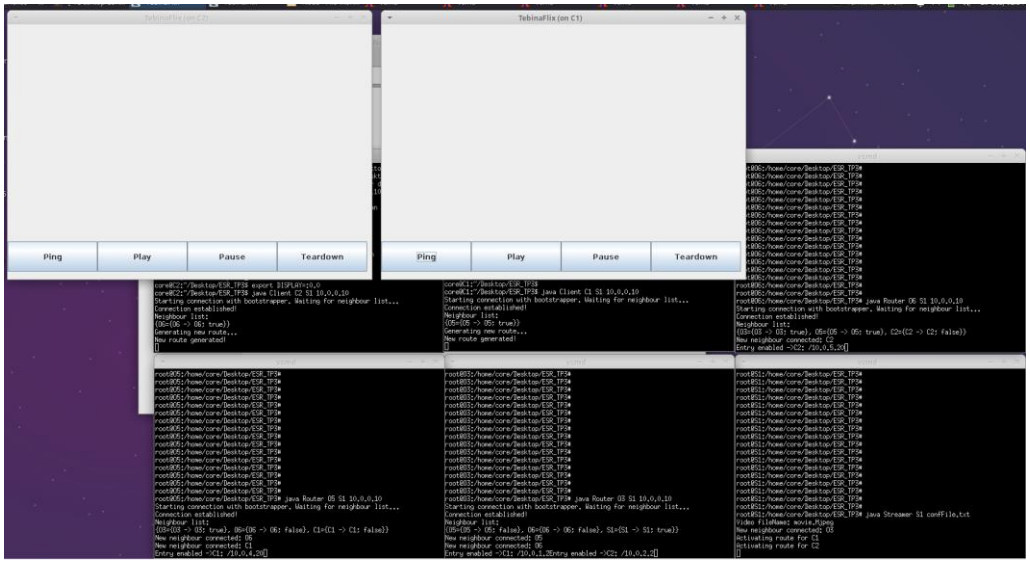


FIGURA 3 - CENÁRIO 2: OVERLAY COM 6 NÓS (UM SERVIDOR, DOIS CLIENTES, 3 NÓS INTERMÉDIOS)

Para efetuar este teste foi utilizada uma rede *overlay* com os nodos da imagem acima.



Como se pode verificar na imagem todos os nodos vão solicitar a conexão à rede *overlay* ao *bootstrapper*, recebem os respetivos vizinhos e enviam uma mensagem, aos que se encontram ativos, a avisar da sua conexão. No caso dos clientes além do início de conexão é também construída a sua nova rota.

B. STREAMING DE MULTIMÉDIA

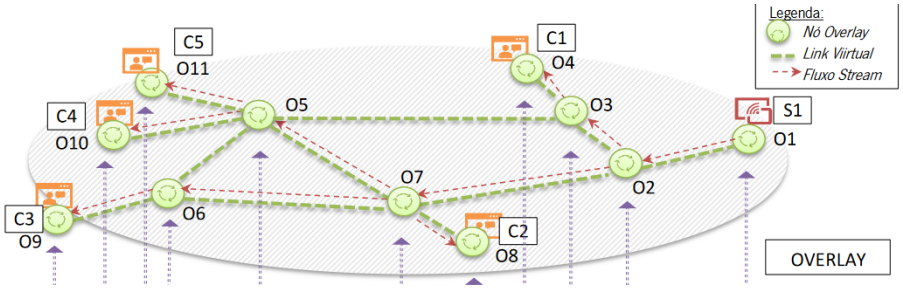
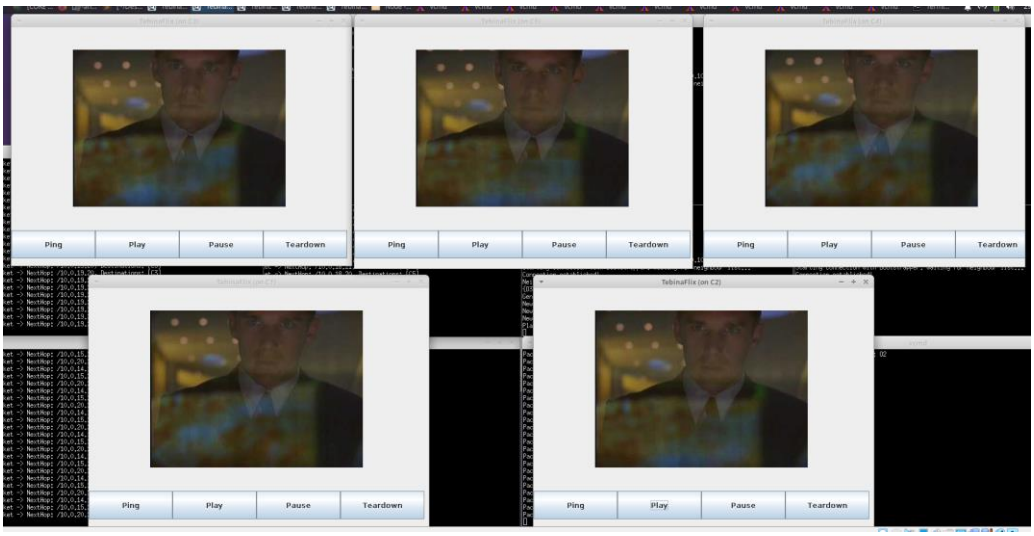
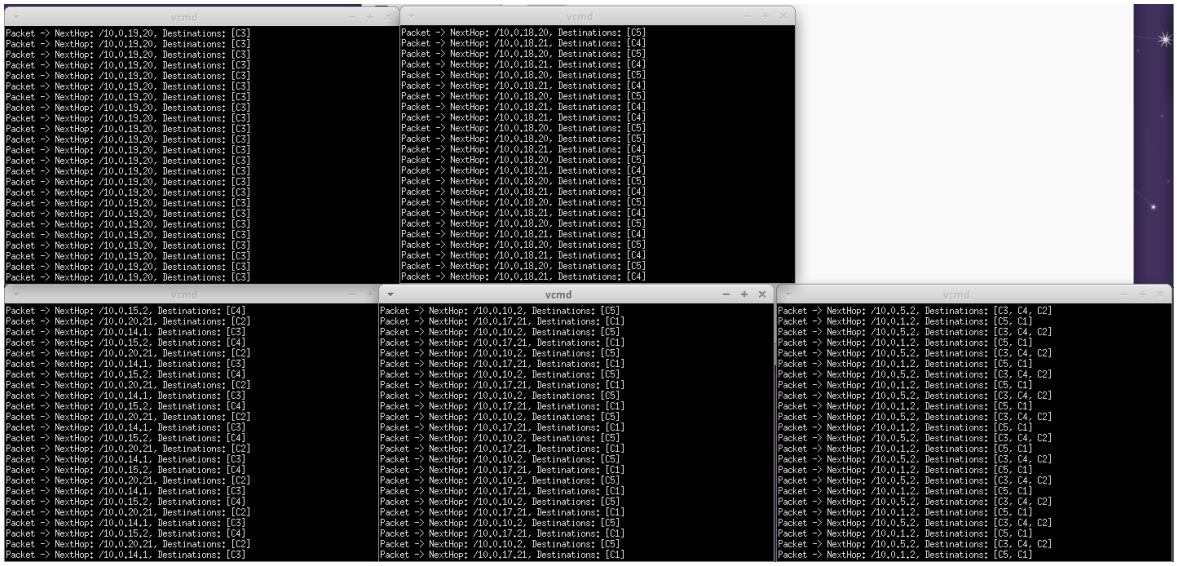


FIGURA 4 - CENÁRIO 3: OVERLAY COMPLEXO

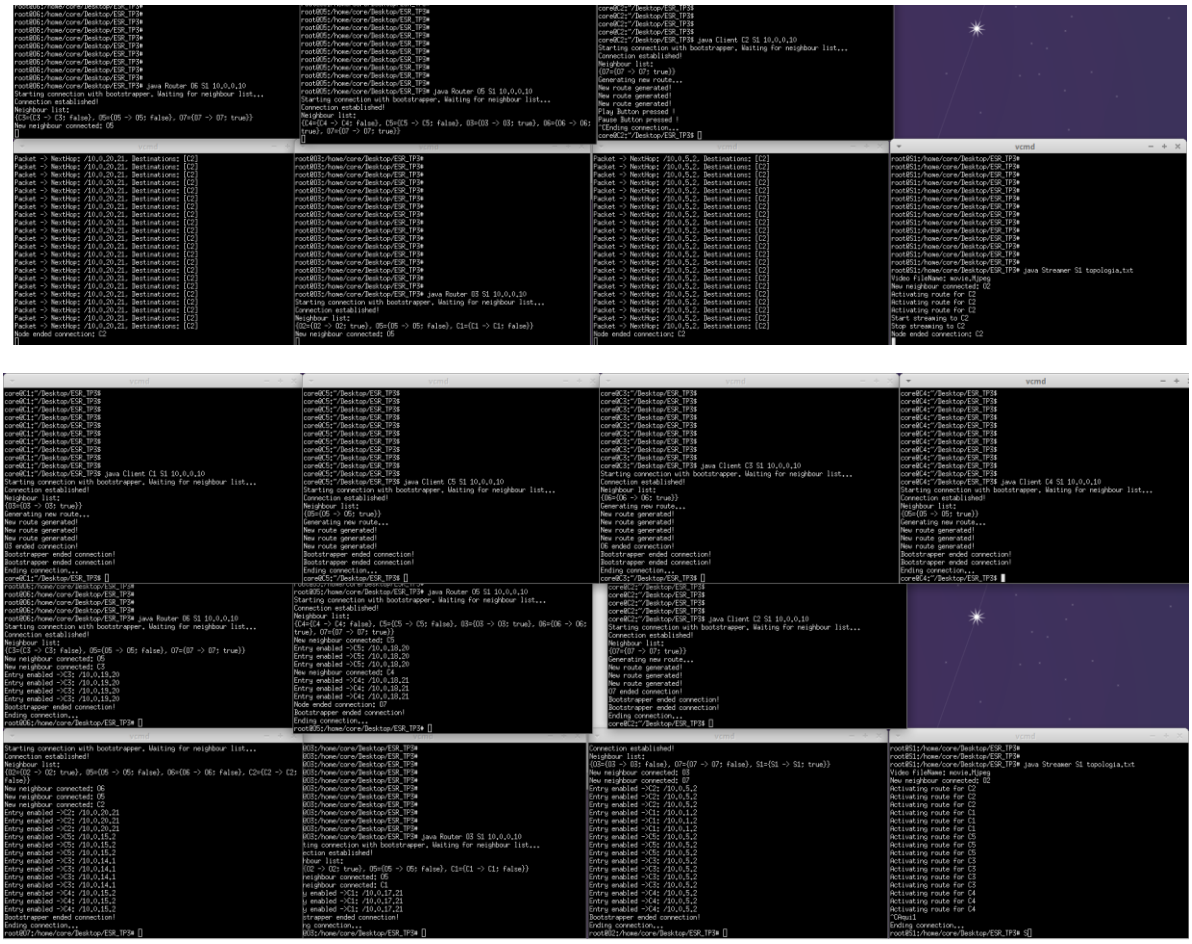
Para efetuar ambos os testes de *streaming* e fim de conexão foi utilizada a rede *overlay* da imagem acima.





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

C. FIM DE CONEXÃO



Como se pode verificar nas imagens acima quando um nó se desconecta da rede *overlay* avisa todos os seus vizinhos e todos os nodos incluídos na sua rota, incluindo o servidor. Por outro lado, quando o servidor de desconecta da rede *overlay,* todos os seus nós serão desligados.

VI. 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.

Durante o desenvolvimento do projeto, o grupo sentiu algumas dificuldades das quais se realçam a construção das rotas de envio dos dados e em garantir que a rede *overlay* fosse dinâmica ou seja que a qualquer momento um nodo se possa conectar ou desconetar. Apesar disto, o grupo conseguiu encontrar uma solução e ultrapassar as adversidades encontradas.

Assim sendo, o grupo considera que fez um bom trabalho tendo implementados os requisitos definidos no enunciado. No entanto, há espaço para melhorias, e como trabalhos futuros seria interessante implementar mecanismo de controlo de perdas, uma vez que o protocolo de transporte utilizado é o UDP e este é não fiável.