# Engenharia de Serviços em Rede
# Trabalho Prático Nº3
# Serviço *Over the Top* para entega de multimédia

Ana Luísa Carneiro, Ana Rita Peixoto, and Luís Miguel Pinto

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {pg46983,pg46988,pg47428}@alunos.uminho.pt

## 1 Introdução

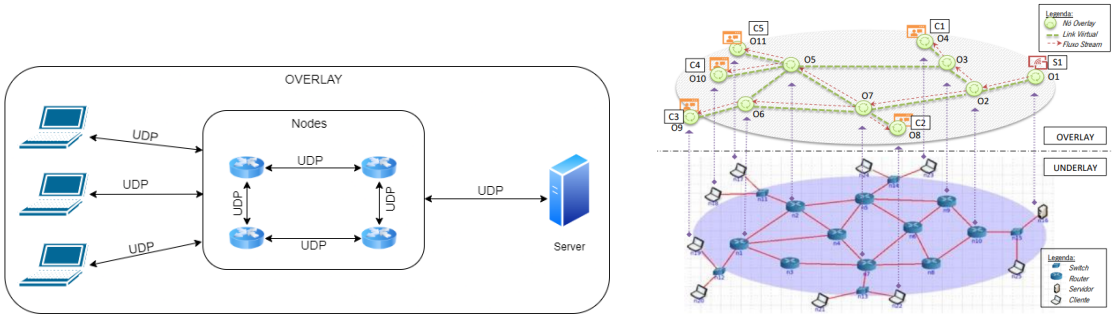
No âmbito do trabalho prático 3 da UC Engenharia de Serviços em Rede foi elaborado um programa com o objetivo de efetuar ***multicast*** **aplicacional** da forma mais eficiente possível. Deste modo, foi necessário considerar diferentes etapas para suportar toda a lógica do programa. Inicialmente, foi construída uma topologia *overlay* a partir da rede *underlay*, onde cada nodo recebe informação do servidor acerca dos seus vizinhos. De seguida, procedeu-se à construção das diferentes tabelas de encaminhamento e de custos em cada nodo, com o propósito de calcular o melhor caminho entre os nodos. Esta rota recorrendo ao protocolo UDP irá permitir a comunicação entre todas as entidades na rede. Após a construção da rede *overlay* e das tabelas de *routing*, é possível efetuar o *streaming* do conteúdo a partir do servidor, de modo a transmitir a informação para o cliente. É de notar que ao longo da realização do trabalho tivemos em consideração monitorização (*pings*), métricas de robustez (alterações na topologia *overlay*) e de eficiência (replicação da carga nos nodos).

## 2 Arquitetura da Solução

Para a realização deste trabalho prático utilizamos a linguagem **Java**, dada a familiarização e facilidade de trabalhar com a mesma. Além disso, dada a natureza do projeto e a necessidade de implementar um protocolo de *streaming*, consideramos conveniente efetuar as comunicações utilizando o protocolo de transporte UDP. Optamos por este protocolo pela sua rapidez no envio de pacotes, quando comparado com o protocolo TCP. É de notar que, ao utilizar UDP, pode haver perda de pacotes. No entanto, estas perdas não são muito significativas e a previsão é que não irão afetar em grande escala a qualidade do *streaming*.

Na figura 1 a) está apresentado um diagrama ilustrativo da arquitetura da solução. É possível observar que existem 3 entidades principais para o programa: **clientes**, **nodos** e **servidor**. É importante referir que tanto os clientes como o servidor também são nodos da rede e possuem as suas próprias tabelas de encaminhamento. Assim, consideramos a existência de um único servidor que será encarregue de distribuir informação para as diferentes entidades, e ainda efetuar o *streaming* de um dado conteúdo. Os clientes são nodos que irão consumir o serviço de *streaming* e, portanto, irão ativar e receber o fluxo a partir do servidor. Os nodos que não são nem clientes nem servidor, tem como objetivo encaminhar as mensagens que recebem do nodo servidor ou do nodo cliente.

Na figura 1 b) é possível observar uma topologia de rede exemplificativa, onde se efetua a criação de uma rede *overlay* a partir da rede *underlay*. A base do programa baseia-se na lógica de possuir vários clientes e um servidor nos extremos da rede. Por conseguinte, a comunicação entre estas duas entidades terá que recorrer aos nodos intermédios e às suas tabelas de encaminhamento, e os pacotes transmitidos terão que ser encaminhados por diferentes nodos, até concluir o percurso entre o servidor e o cliente.




(a) Arquitetura da solução (b) Redes *overlay* e *underlay* do programa

Figura 1: Ilustração de Arquiteturas

## 3 Especificação do(s) protocolo(s)

De acordo com os requisitos estabelecidos no enunciado do projeto, foi necessário criar protocolos aplicacionais de controlo e um protocolo de *streaming*. Deste modo surgem as classes *Packet* e *RTPpacket* que, como iremos ver de seguida, implementam as mensagens protocolares de controlo e *streaming*, respetivamente.

Em particular, podemos associar os protocolos aplicacionais de controlo a protocolos para gestão de processos de : criação da camada de *overlay*, criação da árvore de distribuição de conteúdos, monitorização de nodos, atualização de rotas de encaminhamento e custos de encaminhamento, resposta à criação e/ou remoção de nodos, entre outros. Ao passo que para o protocolo de transmissão de dados do *streaming* entre servidor, nodos e clientes, usamos principalmente o *RTPpacket*.

### 3.1. Formato das mensagens protocolares.

#### Packet - Controlo Aplicacional

De modo a conseguir controlar o fluxo das mensagem de controlo e de dados que atravessam a topologia foi necessário usar o formato *Packet*. Esta mensagem protocolar apresenta um cariz bastante diversificado uma vez que lida com diferentes situações, neste sentido surge o campo ***Tipo*** que serve para identificar o tipo da mensagem protocolar, podendo este ser :

- **tipo 0** : caso seja uma mensagem de **atualização de *overlay***, ou seja, quando um nodo entra na topologia *overlay* tem de notificar os seus vizinhos que entrou na topologia;
- **tipo 1** : caso seja uma mensagem de **flooding**, ou seja, quando estamos a povoar as tabelas de encaminhamento e de custos de cada nodo;
- **tipo 2** : caso seja uma mensagem de **activate**, ou seja, quando um cliente se junta à topologia temos de ativar recursivamente as rotas até ao servidor, caso estas se encontram desativadas;
- **tipo 3** : caso seja uma mensagem de **overlay**, ou seja, quando um nodo entra na topologia tem de pedir ao servidor os seus vizinhos ;
- **tipo 4** : caso seja uma mensagem de ***ping* Cliente**,ou seja, enquanto o cliente estiver vivo notifica o nodo adjacente para demonstrar o seu interesse em receber conteúdos;
- **tipo 5** : caso seja uma mensagem de ***ping* Cliente Pruning**, ou seja, quando um cliente se desconecta da topologia tem de haver um mecanismo de *pruning* que vá recursivamente, enquanto possível, desligando as rotas que antigamente estavam ativadas;

- **tipo 6** : caso seja uma mensagem de ***ping* Router**, ou seja, enquanto o nodo estiver vivo notifica os seus nodos adjacentes. Esta funcionalidade serve para monitorizar os nodos;
- **tipo 7** : caso seja uma mensagem de **resposta do servidor com vizinhos Router**.

O campo ***Dados*** surge com o propósito de enviar as informações corretas de acordo com o tipo da mensagem protocolar, podendo este ser preenchido com : um valor de custo (superior a 1) conforme o número de saltos até aquele nodo - caso tipo 1 ou 0 para indicar que o campo é irrelevante, caso seja outro tipo de mensagem. O campo ***Vizinhos*** serve apenas o propósito de enviar a lista dos IPs dos vizinhos do nodo que acaba de se juntar a rede, ou seja, este campo apenas será preenchido quando a mensagem protocolar for do tipo 7. Na tabela 1 conseguimos observar os diversos campos que constituem um *Packet*.

**RTPpacket - Streaming**

Em adição foi ainda usado o *RTPpacket* para a transmissão de conteúdos de *streaming* pela topologia. Tal como sugere o nome este funcionará como um *Real-time Transport Protocol*, algo muito usado hoje em dia em aplicações de tempo real como, por exemplo, entrega de dados áudio ponto-a-ponto, como Voz sobre IP. Esta classe foi fornecida pela equipa docente de forma a apoiar a realização do trabalho prático. A sua estrutura em nada foi alterada, mantendo-se assim todas as funcionalidades originais. Na tabela 2 conseguimos observar os diversos campos que constituem um *RTPpacket*.

| **Tipo** | **Dados** | **Vizinhos** |
|---|---|---|

Tabela 1: Estrutura *Packet*

| **Version** | **Padding** | **Extension** |
|---|---|---|
| **Marker** | **PayloadType** | **SequenceNumber** |
| **Ssrc** | **header** | **payload_size** |
| **HEADER_SIZE** | | |

Tabela 2: Estrutura *RTPpacket*

## 3.2. Interações.

Na figura 2 podemos observar num exemplo muito simples uma possível ordem do envio das mensagens protocolares *Packet* na topologia. Este exemplo é apenas ilustrativo, pelo que devemos ter em conta que num caso real haveriam mais Nodos e que haveria um maior fluxo de mensagens. Posto isto, podemos agora começar por descrever a figura seguinte. Numa fase inicial apenas existe o servidor, quando um novo nodo entra na topologia terá de pedir ao servidor os seus vizinhos (*Packet* tipo 3), ao qual o servidor responderá (*Packet* tipo 7). Estando o nodo "vivo"este envia duas mensagens, uma para indicar aos seus vizinhos da topologia que ele acabou de se juntar à rede (Packet tipo 0) e outra de monitorização que se iria repetir periodicamente com o objetivo de indicar aos seus vizinhos que ele se mantém vivo (Packet tipo 6). Em seguida, um cliente entra na rede *overlay*, tal como ocorre para um novo nodo que entra na rede, ele também terá de pedir os seus vizinhos e obter a resposta do servido (*Packet* tipo 3 e 7, respetivamente), notificar os vizinhos que entrou na topologia (Packet tipo 0) e periodicamente enviar ao seu nodo adjacente mensagens a indicar o seu interesse em continuar a receber conteúdo (Packet tipo 4). Em adição, visto que são os pedidos dos clientes que ativam as rotas das tabelas de encaminhamento, terá de enviar uma mensagem de ativação para o seu nodo adjacente (*Packet* tipo 2) que por sua vez enviará uma mesma mensagem de ativação de rotas para o servidor (*Packet* tipo 2). Finalmente, o nodo que está a monitorizar o cliente irá notar que este se desconectou e deste modo envia uma mensagem ao servidor a indicar para este desativar a sua rota de streaming (*Packet* tipo 5).

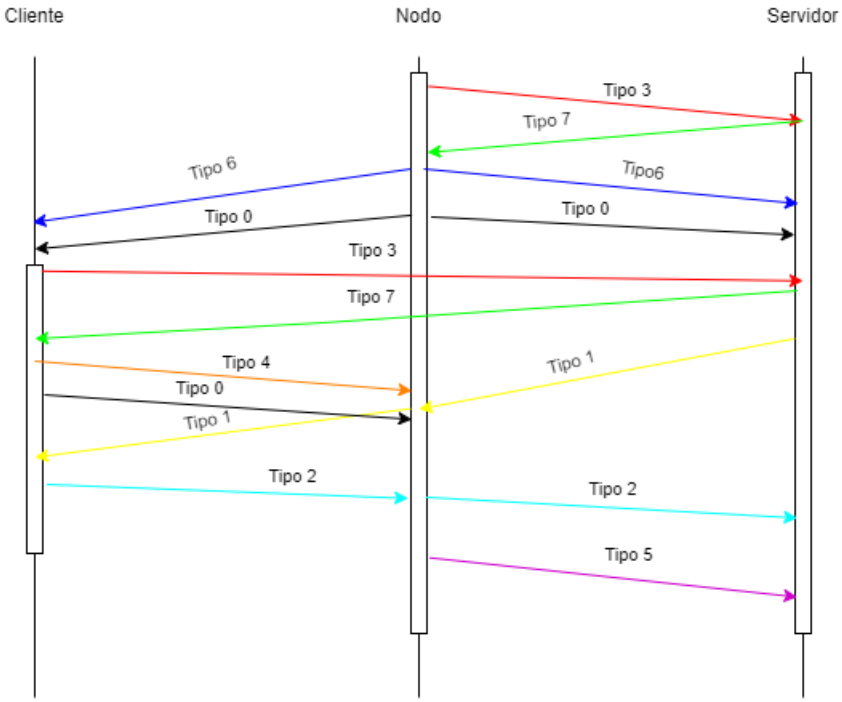



Figura 2: Interações Packet

# 4 Implementação

## 4.1. Etapa 1 - Criação do *Overlay*.

A primeira etapa crucial para permitir o *streaming* foi a criação da rede *overlay*. Esta rede é construída a partir de uma rede *underlay* onde são selecionados alguns nodos. O ponto de partida é a existência de um servidor que começa por efetuar a leitura de um ficheiro *bootstrap* que contém informações relevantes acerca da estrutura do *overlay*. Este ficheiro contém todos os nodos da rede e os respetivos vizinhos, e ainda identifica quais os nodos necessários para arrancar com o programa (nodos importantes). De seguida, procede-se à execução do programa nos outros nodos (enquanto nodo ou cliente) efetuando conexão com o servidor através do envio de uma mensagem de registo de forma a obter os seus vizinhos. Após receber esta mensagem, o servidor envia a informação dos nodos adjacentes a cada nodo que efetuou o pedido. O servidor apenas avança para a etapa 2 (construção das rotas) quando tiver todos os nodos importantes conectados. É de notar que os nodos estão ativamente à escuta de mensagens de outros nodos para o caso em que novos nodos sejam adicionados à rede e/ou ao ficheiro *bootstrap*. Neste caso, pode ser necessário que certos nodos tenham de ser notificados da existem de um novo vizinhos. Essa notificação será enviada pelo novo vizinho aos seus nodos adjacentes. Além disso, o servidor efetua o *flood* periodicamente, para garantir que as rotas estão sempre atualizadas e operacionais.

Por fim, a estratégia utilizada para monitorizar os nós e a possibilidade de abandonarem a rede, baseou-se no envio de mensagens periódicas (*pings*) de cada nó para os seus vizinhos. Quando um nó (cliente ou nodo) deixa de receber mensagens de *ping* de algum dos seus vizinhos, considera que este vizinho abandonou a rede e todas as rotas associadas a esse nodo são desativadas. Caso o nodo removido seja o nodo anterior (origem) de algum outro nodo, então esse nodo deve atualizar o seu nodo anterior, de modo a encontrar uma nova rota até ao servidor atualizando, consequentemente, a sua tabela de *routing*.

### 4.2. Etapa 2 - Contrução das rotas para os fluxos.

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 servidor, que inunda a rede para definir os melhores caminhos para o fluxo dos dados, e utiliza o número de saltos que levou até chegar a cada nodo como métrica. De seguida, caso seja a primeira mensagem de *flood* recebida por um nodo, ou caso ele receba uma mensagem cujo custo seja menor do que o custo que recebeu anteriormente, envia mensagem de *flood* a todos os seus vizinhos (exceto ao nodo que lhe enviou mensagem - nodo anterior) com informação dos novos custos. Ao enviar estas mensagens de *flood* para a construção das rotas, as tabelas de custos são preenchidas e possuem informações acerca dos nodos e custos associados.

Quando um cliente se conecta à rede, este envia uma mensagem para o seu nodo adjacente indicando ativação da rota. De seguida, cada nodo envia uma mensagem ao seu nodo anterior na rota de fluxo de modo a ativar toda essa rota, até chegar ao servidor. Quando o servidor recebe mensagem de ativação, começa o *streaming* do conteúdo que depois será encaminhado por todas as rotas ativas da rede. Caso não existam clientes, as rotas existem nas tabelas de encaminhamento, mas estão desativadas.

### 4.3. Etapa 3 - Teste Percursos *Overlay*.

Na criação das rotas da rede *overlay* foram realizados alguns testes para garantir que o cálculo das rotas estava correto. O primeiro teste realizado foi aquando da construção das rotas e consistiu em imprimir no ecrã as mensagens recebidas por cada nodo no *flood*, de modo a conhecer o custo de ir de um nodo até outro. De seguida, foram efetuados testes após a ativação de um cliente e com recurso a *prints* de modo a verificar se os dados estavam a ir corretamente de nodo em nodo até ao cliente, seguido a rota ótima.

### 4.4. Etapa 4 - *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. Deste modo, no momento do *streaming* o servidor começa por ler o ficheiro de vídeo e envia-o em pacotes numerados para os nodos presentes na sua tabela de encaminhamento cujas rotas estejam ativadas. Estes pacotes são enviados de nodo em nodo, seguindo as rotas ativas da rede, até chegarem ao cliente que pediu o *streaming*. Caso um nodo possua mais do que uma rota ativa, este irá replicar os pacotes e enviá-los por cada uma das rotas, o que permite aumentar a eficiência na transmissão e evitar redundância. É de notar que o servidor efetua um *streaming* contínuo dos dados, enviando o vídeo em *loop* até todos os clientes abandonarem a rede. Nesse caso as rotas estão desativadas e por isso os pacotes são descartados antes de chegar ao cliente.

## 5 Testes e Resultados

Nesta secção apresentamos alguns testes realizados na topologia do core de forma a testar todas as funcionalidades implementadas, analisando se o resultado obtido em cada teste foi de acordo com o que era pretendido. Na figura 3 encontra-se a topologia *underlay* que será usada em todos os testes. Nesta topologia consideramos que os nodos que serão utilizados na rede *overlay* vão do n1 ao n9, desta forma incluímos o servidor, alguns *routers* e todos os clientes.

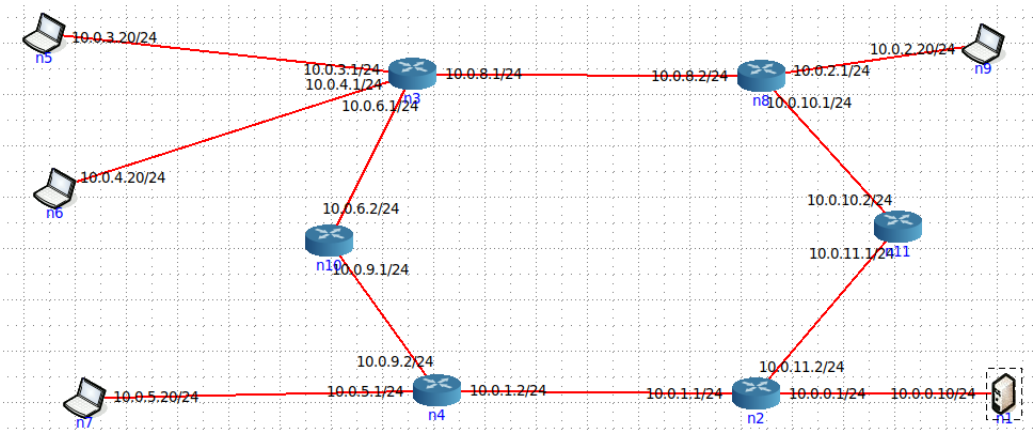



Figura 3: Rede *underlay* de teste

Após o programa ser executado nos nodos que consideramos mais importantes (n1,n2,n3,n4 e n8), isto é, os nodos fundamentais para iniciar a execução do sistema na rede obtemos a rede de *overlay*. Após o *flood* que vai preencher todas as tabelas de encaminhamento obtemos a rede na figura 4.

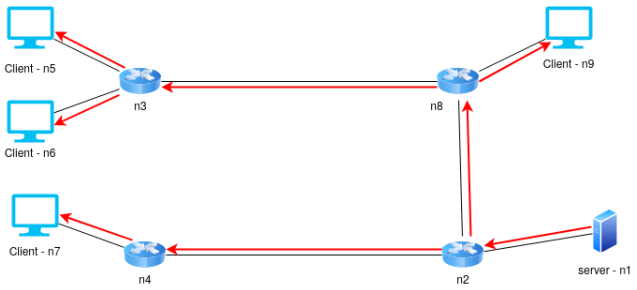



Figura 4: Rede *overlay* de teste

## 5.1. Cenários de Teste.

**Cliente N5, N6 e N7 pedem *stream*** O N5, N6 e N7 quando entram na rede vão ter de esperar para que o servidor realize um *flood* na rede de forma a as tabelas de *routing* e custo sejam atualizadas para receber os clientes. Após o *flood*, os clientes vão ativar as conexões para que estes consigam receber a *stream*. Na figura 5 encontra-se o teste realizado no core, onde podemos ver as 3 *streams* nos 3 clientes diferentes tal como pretendido.

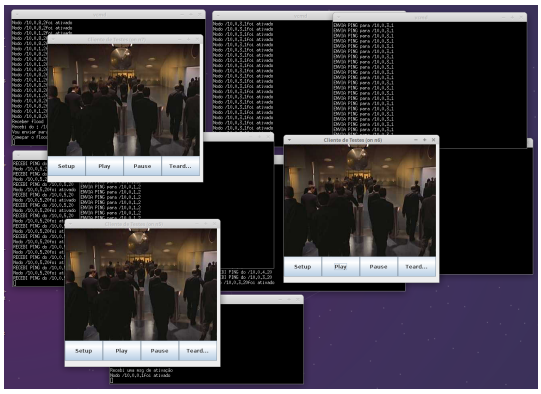


Figura 5: Teste N.º1

**Cliente N6 sai da rede** Caso o cliente N6 deixe de querer a *stream* então este sai da rede fechando a janela do vídeo. Com a saída do cliente N6, o nodo N3 deixa de receber *pings* desse cliente, desativando essa conexão. De seguida, o nodo N3 tem de averiguar se possui mais clientes a depender dele para receber a stream. Caso isto aconteça, o nodo apenas desativa a conexão do cliente que sai da rede. Caso contrário, o próprio N3 deve sinalizar ao nodo anterior (N8), que lhe envia a stream, para fazer *pruning* de N3. O processo de *pruning* ocorre assim sucessivamente caso o nodo anterior não tenha nenhum cliente a depender dele para receber a *stream*. Visto que o cliente N5 está também ligado ao nodo N3 e depende deste para receber a *stream* apenas a conexão do nodo N6 será desativada da tabela de encaminhamento de N3. Na figura 6, encontra-se o teste realizado no core, onde podemos ver as 2 *streams*, do cliente N7, N5 tal como pretendido.

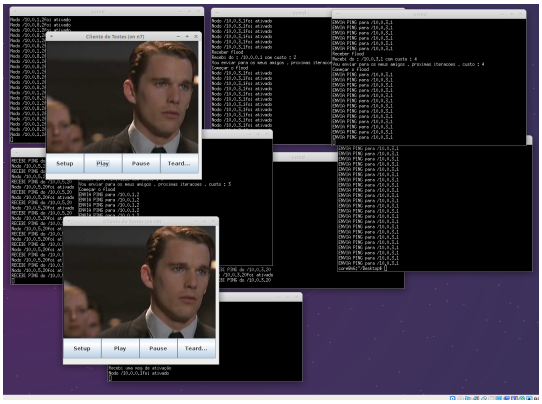


Figura 6: Teste N.º2

**Nodo N10 entra na rede *overlay*** Apesar do nodo N10 não estar inicialmente na rede da figura 4, este pode ser acrescentado a qualquer momento caso o *bootstrapper* utilizado seja alterado, mesmo durante a *stream*. O nodo 10, ao ser acrescentado na rede *overlay*, não vai alterar as tabelas de encaminhamento pois a adição do nodo 10 ia criar um fluxo com maior custo do que aquele que estava a ser usado anteriormente. Na figura 7 encontra-se o teste realizado no core, onde podemos ver as 2 *streams*, do cliente N7 e N5 tal como pretendido e o terminal do nodo 10 (rodeado a vermelho) que tal como era previsto não está a enviar *stream*.

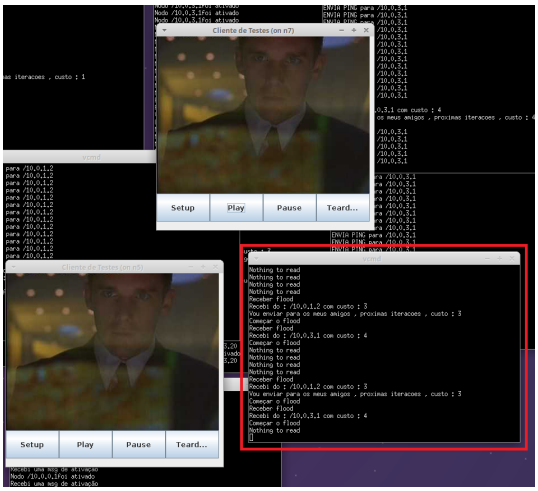
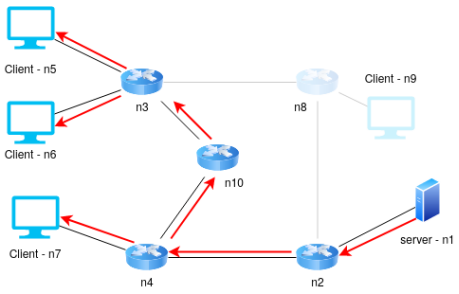
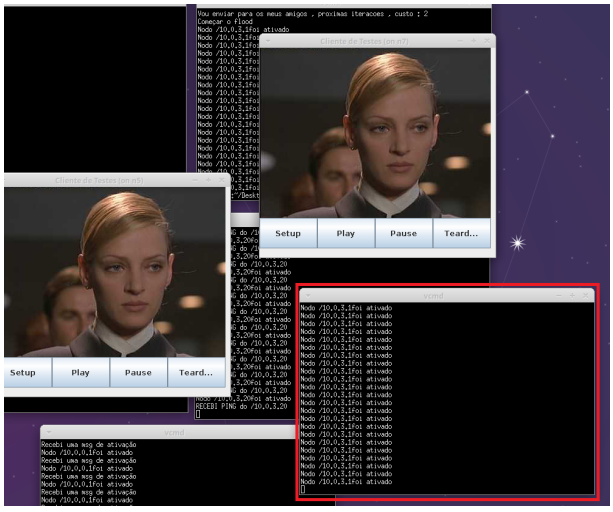



Figura 7: Teste N.º3

**Nodo N8 sai da rede** Quando o nodo N8 sai da rede é necessário que as tabelas de encaminhamento sejam atualizadas para que agora o fluxo da *stream* deixe de passar no nodo N8 e passe agora no nodo N10, criando a rede da figura 8 a). Desta forma as *streams* nos clientes N7 e N5 estagnam entre a saída do N8 e o novo *flood*. Após o *flood* a nova rota é ativada fazendo com que as *streams* nos clientes N5 e N7 retomem à transmissão. Na figura 8 b), encontra-se o teste realizado no core, onde podemos ver as 2 *streams*, do cliente N7 e N5 tal como pretendido e o terminal do nodo 10 (rodeado a vermelho), que tal como era previsto está a enviar *stream*.




(a) Rede Overlay atualizada




(b) Resultado do teste N.º 4

Figura 8: Teste N.º4

**Nodo N8 volta a entrar na rede *overlay*** Quando o N8 volta a entrar na rede então as rotas vão ser novamente atualizadas, pois o custo desta nova rota é inferior ao da rota que passa por N10. Desta forma, após o *flood* as rotas vão ser atualizadas, passando agora no nodo 8, mantendo sempre a stream em funcionamento. Na figura 9 encontra-se o teste realizado no core, onde podemos ver as 2 *streams*, do cliente N7 e N5 tal como pretendido, o terminal do nodo N10 (rodeado a vermelho) que não está a transmitir *stream* e o terminal do nodo N8 (rodeado a verde) que tal como era previsto está a enviar *stream*.

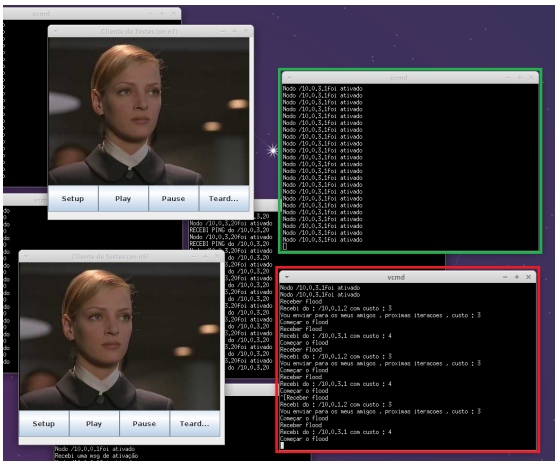



Figura 9: Teste N.º5

**Nodo N8 e N10 sai da rede *overlay*** Quando o nodo N8 e N10 saem da rede então não existe possível conexão entre o servidor e o cliente N5, ou seja, a *stream* nunca consegue chegar ao nodo N5 pois o N3 também não se consegue conectar com o servidor e por isso o vídeo estagna até haver de novo conexão entre esses nodos e o servidor. Contudo a saída deste dois nodos não impede que o cliente N7 receba a *stream*, pois rota da *stream* mais curta para chegar a N7 será n1->n2->n4->n7. Na figura 10 encontra-se o teste realizado no core, onde podemos ver as 2 *streams*, do cliente N7 e N5 tal como pretendido contudo podemos ver que a *stream* no N5 está parada enquanto que a do N7 continuou a transmissão tal como era previsto.

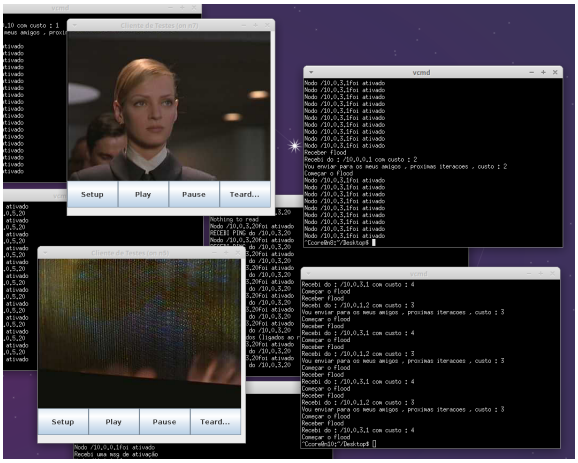



Figura 10: Teste N.º 6

## 6 Conclusões e Trabalho futuro

Dado por concluído o projeto, apresentamos uma visão crítica, refletida e ponderada do trabalho realizado.

No espetro positivo, consideramos relevante destacar o facto de termos implementado todas as funcionalidades base para o sistema proposto funcionar, tal como : o *flood*, a criação do *overlay* e a *stream* de um vídeo. Além dessas funcionalidades também foram implementadas os requisitos e mensagens de controlo necessários para que a rede se mantenha ativa e atualizada tal como, estratégias de abandono e ativação/desativação de rotas.

Em termos de melhorias ou *upgrades* no nosso projeto, consideramos que com mais tempo poderíamos facilmente implementar um sistema que permitisse ao cliente escolher que conteúdo queria assistir ao invés de apenas ter um vídeo. Em adição, de modo a conseguir monitorizar se as mensagens chegam ao destino poderia ser usado um sistema de mensagens com o auxilio de ACK's (*acknowledgements*).

Destacamos ainda como fator positvo a robustez a alterações na topologia, a eficiência da distribuição da *stream* nos nodos encaminhadores, a monitorização através de *pings* dos nodos e finalmente o facto de haver apenas 1 única *stream* de dados.

Em suma, consideramos que o balanço do trabalho é positivo, as dificuldades sentidas foram superadas e os requisitos propostos foram cumpridos.