TP2 - Serviço Over the Top para entrega de multimédia

Cláudia Silva[a93177], David Duarte[pg50315], and Laura Rodrigues[pg50542]

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
{a93177,pg50315,pg50542}@alunos.uminho.pt

1 Introdução

No âmbito da unidade curricular de engenharia de serviços em redes foi desenvolvido um protótipo de entrega de áudio/texto/vídeo com requisitos em tempo real a partir de um servidor de conteúdos para um conjunto de N clientes. Optámos por desenvolver o projeto em java pois é a linguagem mais conveniente para todos os elementos do grupo. E o protocolo de transporte usado para o overlay foi o UDP que apesar de ter maior perda de packets os mesmos não se mostram muito significantes e é um protocolo mais rápido comparativamente ao protocolo TPC.

2 Arquitetura da solução

No trabalho prático é possível destacar três classes principais. As classes Node, Client e Server. Na classe Server é criado o overlay a partir de um **boostrapper**. Esta estratégia consiste em fazer o parsing de um ficheiro com a topologia de overlay completa que é carregado quando se inicializa um servidor. Assim, após a leitura do ficheiro de configuração o servidor define quais são os vizinhos de um cada nodo com base no conhecimento que tem de todo o overlay e guarda-os num mapa para mais tarde serem usados. O servidor controla toda a informação que depois é distribuída pelos nodos/routers de modo a chegar aos clientes corretamente e assim efetuar o streaming.

No caso do Node este recebe os pedidos dos clientes e encaminha-os até ao servidor e vice-versa. É nos nodos que são calculados os melhores caminhos com base no delay no envio de mensagens e com base no número de saltos dados desde a saída de um pacote até à chegada ao próprio.

Quando um Client se tenta conectar à rede, o servidor vai informá-lo do nodo da rede overlay que estiver mais próximo deste com base no ip de ambos. Assim, o cliente poderá então pedir ao seu nodo que inicie o serviço de streaming, de modo a que este pedido chegue ao servidor e as comportas até este cliente sejam abertas.

Cada nodo poderá ter vários clientes ligados a si, mas cada cliente apenas se poderá ligar a um nodo. Esta rede de nodos overlay é **estática** de modo a garantir a segurança na estrutura da rede e previnir imprevistos. Os clientes podem ligar-se e desligar-se da rede dinamicamente.

3 Especificação do(s) protocolo(s)

De acordo com o que era pedido no enunciado foi necessário criar protocolos aplicacionais. A classe RTPpacket fornecida pelos docentes foi aproveitada de modo a poder enviar os dados relativos a streaming, no entanto, para poder manter uma rede ativa e bem estruturada houve a necessidade de criar um novo tipo de pacotes que nos permitisse enviar outras informações relativas à rede, flooding, etc. Assim surgiu a classe Packet.

Esta classe é, assim, responsável pela conexão de nodos e clientes à rede, envio de informações do servidor para ambos, criação da árvore de distribuição de conteúdos, monitorização de clientes, gestão de rotas de encaminhamento com base no número de saltos e delay das ligações, etc.

3.1 Formato das mensagens protocolares

Cada pacote tem um identificador, o custo, um LocalTime que é a hora a que este sai da sua origem e um campo vizinhos de tamanho indefinido onde são enviados InetAddress relevantes ao tipo de pacote. Assim os nossos pacotes teriam um tamanho dinâmico que dependeria do tamanho do campo vizinhos.

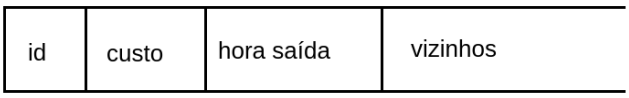

Fig. 1. Estrutura do Packet

O **identificador** permite saber o tipo de pacote que está a ser recebido. Assim, temos:

- **Tipo 0** : Quando um nodo se conecta à rede envia este ao servidor de modo a avisar a sua presença
- **Tipo 1** : Ao receber um aviso de presença de um nodo ou de um cliente o Servidor responde-lhes, respetivamente, com os nodos vizinhos e com o nodo de overlay a que se deve conectar
- **Tipo 2** : Pacote enviado deste o Servidor até todos os nodos de modo a fazer inundação controlada e assim criar a árvore de distribuição. Graças a este e a todos os relógios estarem sincronizados, é possível calcular o custo e o delay totais de envio de um pacote desde a origem até ao seu destino. É também aqui que é feito o povoamento das tabelas de encaminhamento.

- **Tipo 3** : Quando um Cliente se pretende conectar à rede, envia um ping ao servidor de modo a descobrir qual é o nodo da topologia de overlay que se encontra mais próximo de si e assim poder conectar-se a ele.
- **Tipo 4** : Após se conectar à topologia, o Cliente envia pedidos ao seu Nodo de modo a abrir as comportas de streaming e assim poder receber o vídeo. Estes pedidos são encaminhados até ao servidor para, assim, abrir todas as comportas do percurso que permite chegar ao cliente.
- **Tipo 5** : Quando um Nodo deteta que um cliente já não está ativo, envia até ao Servidor este pacote de modo a fechar as comportas de streaming até este
- **Tipo 6** : O nodo faz a gestão dos seu clientes através do envio periódico deste pacote. Graças ao envio contínuo de pacotes do tipo 4 pelo cliente, um nodo sabe assim quais dos seus clientes ainda estão ativos. Se após 3 pacotes do tipo 6, o Nodo não obtiver nenhuma resposta do Cliente é enviado um pacote do tipo 5 de modo a fechar as comportas de streaming.

O campo **custo** é enviado a zero na maior parte dos pacotes. No entanto, no pacote do tipo 2 é campo funciona como um acumulador que permite saber o número total de saltos até chegar a um dado nodo. Este campo e o seguinte permitem calcular qual a melhor e mais rápida rota para o envio de streaming.

Para a maior parte de pacotes o campo vizinho será enviado a *null* de modo a poupar o tamanho do pacote enviado e agilizar o envio de informações de monitorização. Apenas pacotes do tipo 1 tem este parâmetro preenchido.

3.2 Interações

Na figura 2 podemos observar como se desenrolam as iterações entre um cliente, um nodo e um servidor.

Ao querer conectar-se à topologia o Nodo envia um **Packet 0** ao Servidor de modo a avisá-lo da sua presença. Em resposta recebe um **Packet 1** que o informa dos seu vizinhos, ou seja, dos outros nodos da topologia com os quais pode comunicar. No caso de um Cliente se querer conectar deve enviar um **Packet 3** ao Servidor de modo a descobrir através de um **Packet 1** a qual nodo se deve conectar. Periódicamente é enviado recursivamente um **Packet 2** para toda a topologia de modo a realizar a inundação controlada, ou seja, *flooding*. No flooding o servidor manda um **Packet 2** para os nodos ligados a ele. Por sua vez, cada um destes nodos vai enviar para os seus vizinhos, com exceção de quem lhe mandou. Esta forma de envio que não permite o retrocesso de mensagens cria a árvore de distribuição. Logo após o Cliente se conectar ao Nodo começa a enviar continuamente **Packets 4**. Estes não só permitem informar os Nodos e Server que devem abrir as comportas de streaming como também informa o Nodo que este Cliente ainda se encontra ativo. Após receber o pedido do Cliente o server inicia o streaming através dos **RTPpacket**. Periodicamente cada Nodo envia aos seus Clientes **Packets 6**. Se após 3 pacotes destes não receber nenhum **Packet 4** assume que o Cliente se desconectou e envia até ao Servidor um

Packet 5 de modo a verificar (em função de haverem mais clientes ou não) se as comportas de streaming devem ser fechadas.

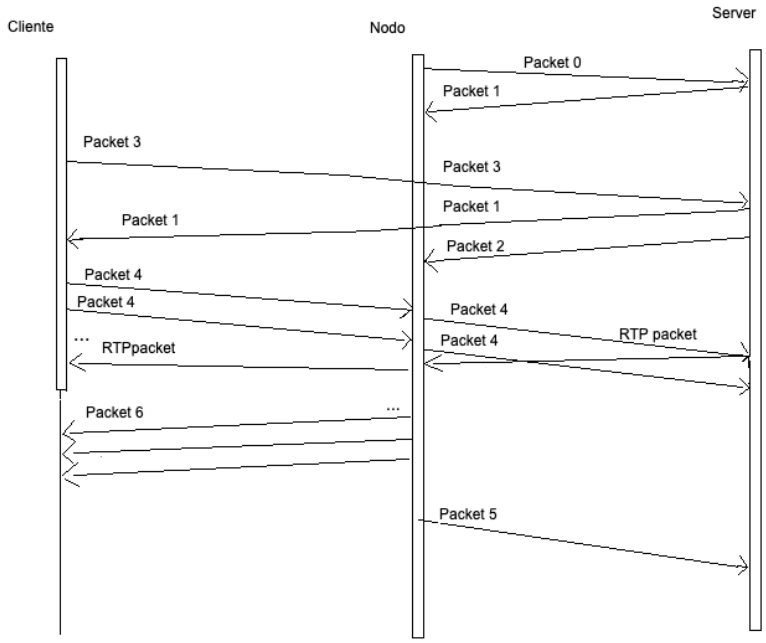

Fig. 2. Interações entre pacotes

4 Implementação

4.1 Etapa 1 - Construção da Topologia Overlay

Começámos por construir a topologia overlay. Para tal optámos pela segunda estratégia de usar um bootstrapper onde alguns nodos da rede underlay são selecionados. Após o servidor efetuar a leitura de um ficheiro de configuração onde estão definidos os nodos e os seus respetivos vizinhos estes são guardados num mapa para depois serem usados.
Definimos assim, que graças a este ficheiro seria o Servidor que teria conhecimento de todo o overlay e informaria cada nodo dos seus vizinhos. Deste modo, todos os nodos conectados a rede sabem os seus vizinhos e o servidor conhece a topologia toda.

4.2 Etapa 2 - Serviço de Streaming

Depois da primeira etapa, optámos pela estratégia 1 e construímos o serviço de streaming com base no código fornecido pelos professores. Nesta etapa conseguimos fazer um servidor capaz de ler o vídeo e enviar os pacotes do mesmo e um cliente capaz de receber pacotes da rede overlay com um número de sequência e reproduzir o vídeo numa janela.

4.3 Etapa 3 - Monitorização da Rede Overlay

Para monitorizar as redes overlay, o server realiza o flooding de 30 em 30 segundos de modo a ter um conhecimento atualizado da sua topologia. Nesta mensagem incluímos o número de saltos que a mensagem deu e o instante temporal em que a mesma foi enviada, para que pudesse ser calculado o atraso sofrido desde que o seu envio até à sua receção.

Quando um nodo recebe um packet com o id=2 ele verifica quanto tempo demorou entre o envio e a receção de mensagens calculando assim o delay. Por fim, são enviados packets para o resto da rede com o custo a ser constantemente incrementado. Por fim, o caminho escolhido é o que apresenta o menor delay. Em caso de o tempo demorado ser igual o caminho que apresenta o menor custo é o escolhido.

4.4 Etapa 4 - Construção de Rotas para a Entrega de Dados

Tal como mencionado anteriormente, cada nó de overlay considera a métrica que o grupo considerou mais favorável. No nosso caso, é escolhido o menor atraso, e para atrasos idênticos, o menor número de saltos.

Optámos assim pela estratégia 1, na qual o servidor de streaming envia anúncios periódicos. Ao receber a mensagem cada nó constrói uma tabela de encaminhamento, uma com os custos e outra com o atraso nas ligações. Cada nó que receba a mensagem atualiza-a e envia-a a todos os seus vizinhos, usando inundação controlada. Quando um cliente se liga envia uma mensagem de ativação da rota, pelo percurso inverso, que abre as comportas de streaming (colocando nas tabelas de encaminhamento a string "on"). Quando não há clientes, as rotas existem nas tabelas, mas estão inativas (o que é registado nas tabelas através da string "off"), não havendo tráfego.

4.5 Etapa 5 - Ativação e Teste do Servidor Alternativo

Uma vez que os nossos servidores enviam constantemente mensagens de flooding é possível manterem-se atualizados sobre as condições da topologia. De modo a saber se a ligação entre um servidor e um nodo foi prejudicada seria necessário, ou que os servidores comunicassem entre si, ou que os nodos, em vez de enviarem os pacotes pelo caminho inverso, verificassem, como antes, a ligação com o menor atraso. À data de escrita deste relatório esta etapa está a ser desenvolvida mas esperamos conseguir implementar a segunda opção.

4.6 Detalhes, Parâmetros, Bibliotecas de funções, etc.

Neste projeto utilizámos bibilotecas disponibilizadas pelo Java. Das várias usadas destacamos as menos comuns:

- **java.time.temporal.ChronoUnit** : que nos permitiu calcular o delay das ligações entre os vários nodos, servidores e clientes.
- **javax.swing.Timer** : Usado pelo código fornecido pelos docentes.
- **java.util.concurrent.locks.ReentrantLock** : Para proteger informação quando várias threads a tentam aceder ao mesmo tempo.
- **java.time.LocalTime** : Para obter a hora a que um pacote é enviado ou recebido.

5 Testes e resultados

Realizamos os seguintes testes de modo a demonstrar como funciona o stream.No programa demonstrado há apenas dois nodos um cliente e um servidor apenas. Nas imagens apresentadas o servidor inicia o overlay e realiza o flooding por todos os nodos da rede.

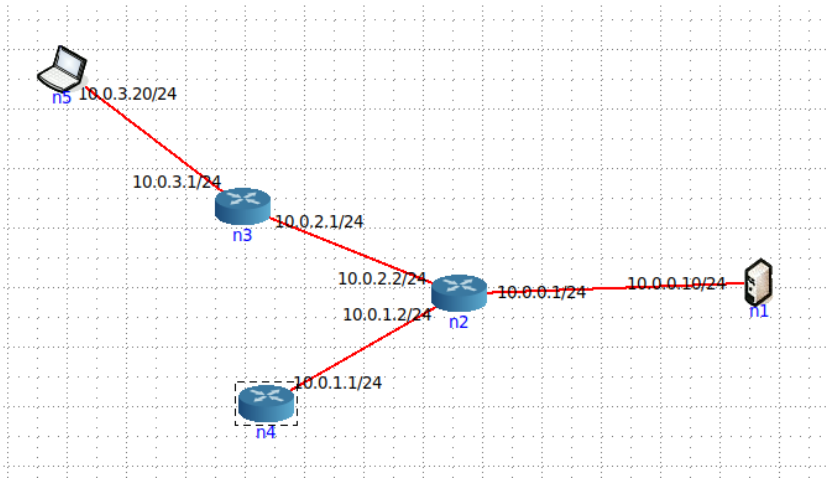

Fig. 3. Topologia simples com apenas um cliente e um server

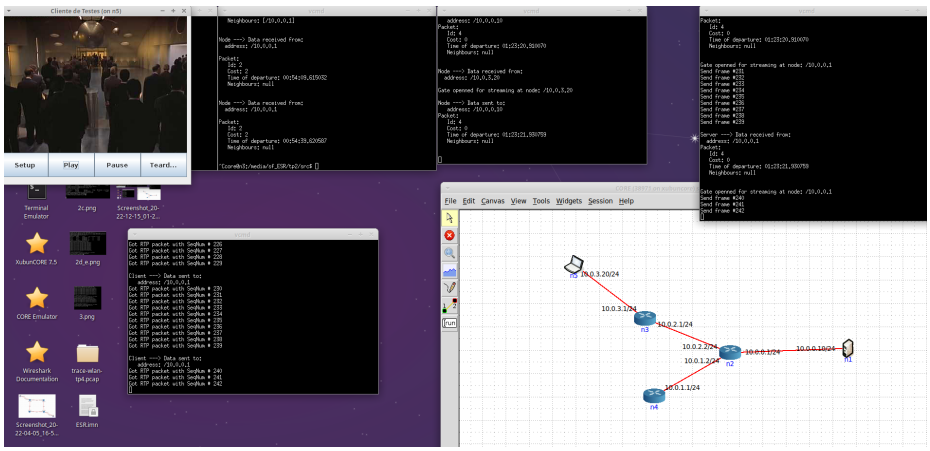

Fig. 4. Demonstração da execução na topologia simples

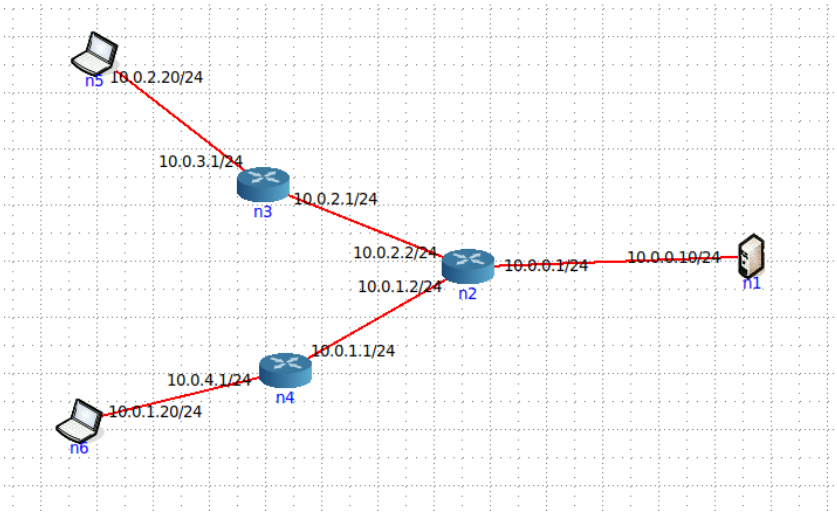

Fig. 5. Topologia com dois clientes - streaming.imn

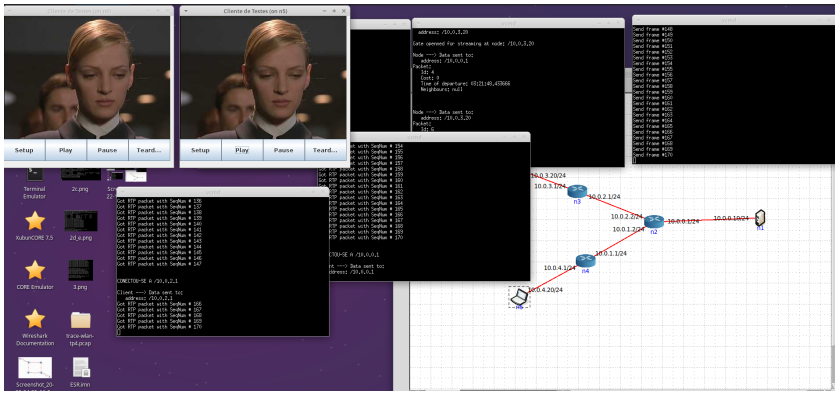

Fig. 6. Demonstração do streaming nos dois clientes

Na imagem abaixo, quando um dos clientes se desconectou o nodo detetou graças aos vários pacotes do tipo 6 e enviou para o nodo anterior um pacote 5. Podemos ver isto no terminal da esquerda que enviou o packet 5.

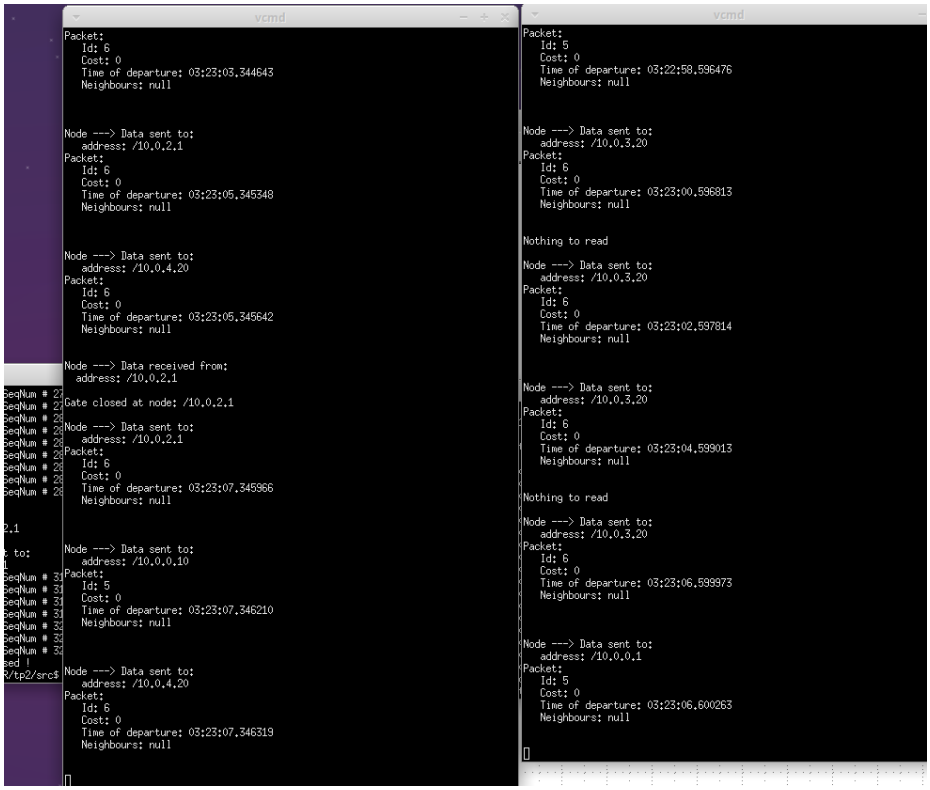

Fig. 7. Demonstração da desconexão de um cliente

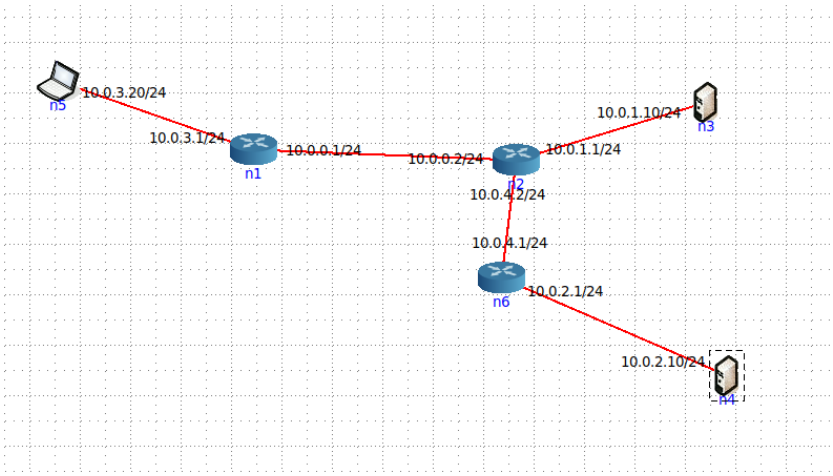

Fig. 8. Topologia com mais do que um server

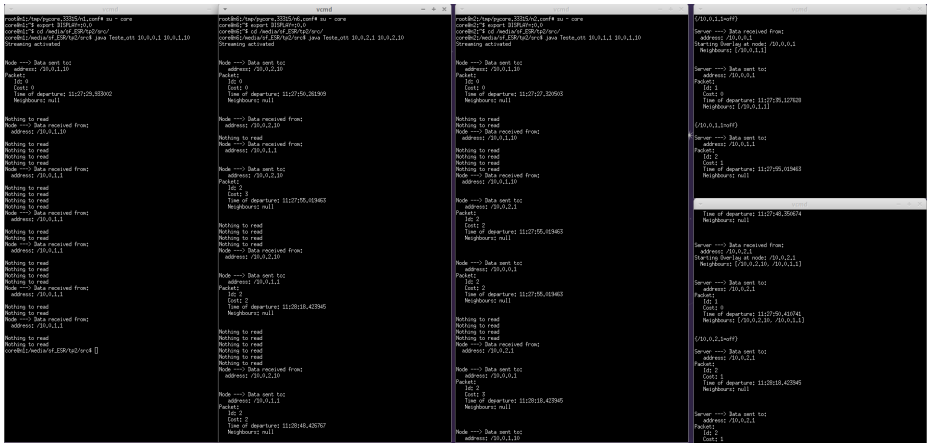

Fig. 9. Terminais respetivos à topologia com mais do que um server

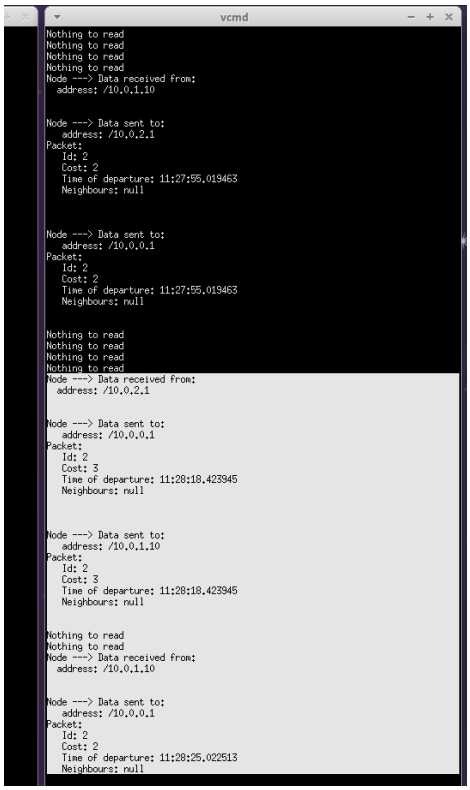

Fig. 10. Nodo 2 apresentado na figura 8 - Mudança de encaminhamento pelo flooding

Acima vemos que passa a ser o flood vindo do segundo server a ser enviado para o nodo 1 da figura 8.

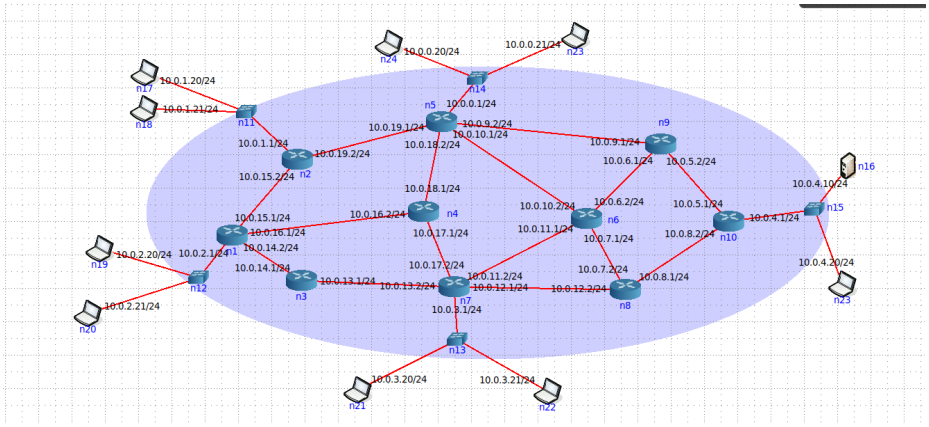

Fig. 11. Topologia mais complexa

À esquerda podemos ver todos os terminais dos nodos de overlay, no canto superior direito o servidor e abaixo deste 3 clientes.

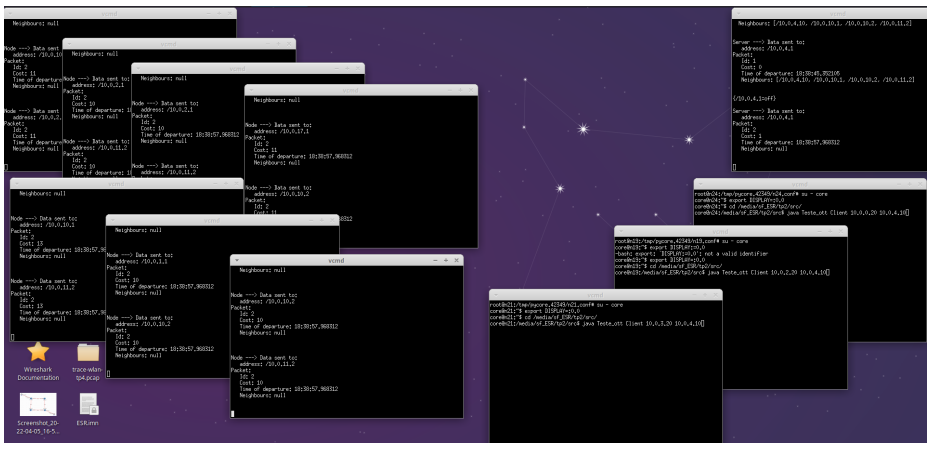

Fig. 12. Topologia mais complexa

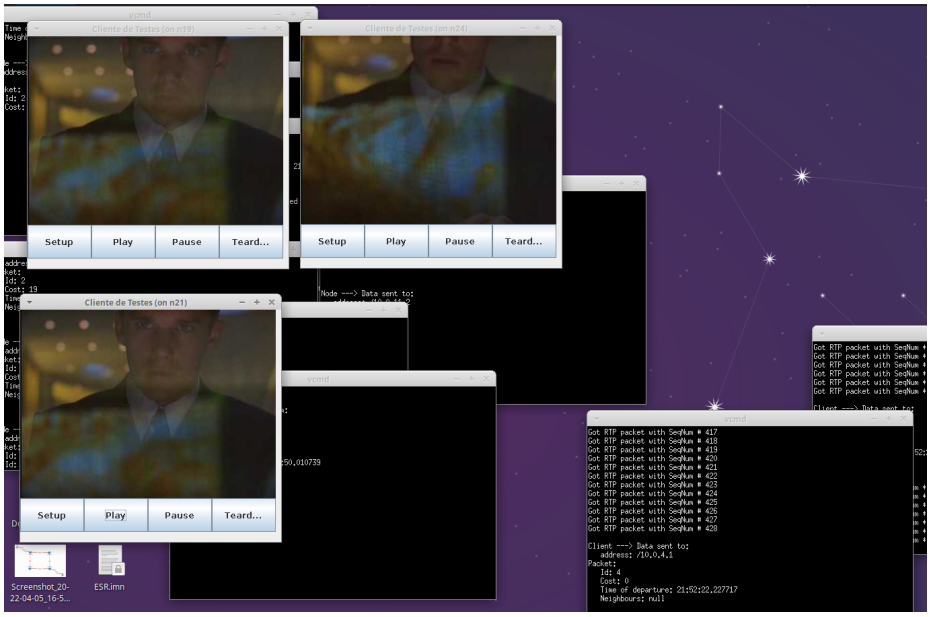

Fig. 13. 3 clientes a transmitir em simultâneo numa topologia mais complexa

6 Conclusões e trabalho futuro

Em modo de conclusão, consideramos que implementámos as funções mais importantes para o desenvolvimento do streaming pelos diferentes nodos criando corretamente o overlay, flooding, stream de um vídeo, a abertura e fecho das comportas e monitorização de clientes. No entanto, consideramos que com mais tempo poderíamos ter implementado uma rede dinâmica onde fosse possível conectar e desconectar nodos e assim prevenir várias falhas ou atrasos que acontecem numa topologia estática. Pensamos, também, que as maiores dificuldades sentidas durante a realização do mesmo foram ultrapassadas.