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

Rodrigo José Teixeira Freitas[1 [pg54197]] and João Paulo Peixoto Castro[1 [pg53929]]

Universidade do Minho

Abstract. O presente relatório visa sumarizar as estratégias arquiteturais e de conceção de um serviço de streaming em tempo-real numa rede de overlay aplicacional.

1 Introdução

Neste trabalho iremos desenvolver um serviço implementado sobre a camada aplicacional usando protocolos de transporte previamente conhecidos como o TCP. Neste serviço serão selecionados e configurados alguns nós que fazem parte da rede overlay para fornecer uma entrega de conteúdo de multimédia eficiente, visando contornar problemas de congestão e limitação de recursos, assim como de largura de banda e latência temporal. Desta forma, iremos usar o emulador do core para simular a rede física e conceber um protótipo de entrega de vídeo a um conjunto de clientes, baseando, mais uma vez, em métricas em tempo real.

2 Arquitetura da solução

Antes de proceder ao desenho arquitetural e à implementação dos módulos foi necessário compreender o domínio do problema. Em primeiro lugar, identificámos os seguintes componentes:

1. **Cliente** - componente que terá uma interface para se comunicar com a aplicação de streaming.
2. **Servidor** -componente que irá fornecer os conteúdos de vídeo nele armazenados para o RP, assim como mensagens em unicast.
3. **RP (Rendezvous Point)** - componente responsável por gerir a árvore da topologia, correr algoritmos de melhores caminhos para distribuição de conteúdos baseados em métricas, receber conteúdos de servidores e escolher o melhor servidor para um dado conteúdo pedido por um cliente.
4. **Nó Intermédio** - componente responsável por reencaminhar o conteúdo recebido para o próximo nó ou nós dependendo do melhor caminho e do número de clientes presentes.

Assim, apresentámos de seguida, um pequeno diagrama com os componentes a serem implementados.

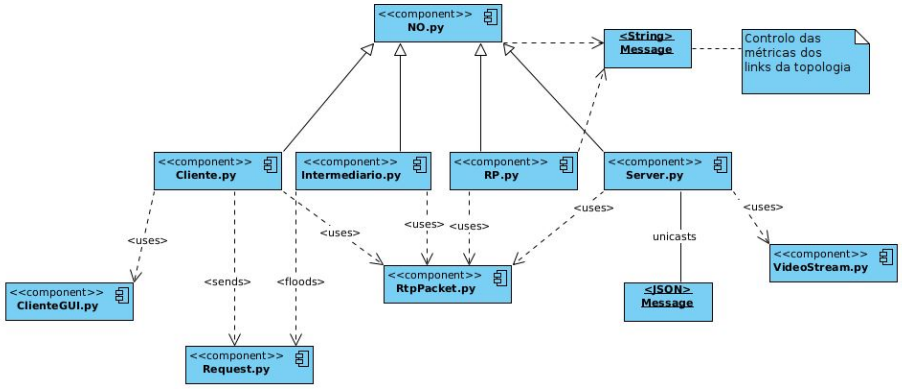

Fig. 1. Diagrama de componentes

Como é possível averiguar, iremos proceder a uma abstração destes componentes para um componente NO que será no fundo uma generalização dos mesmos. O papel de cada um irá ser distinguido de acordo com a informação disponibilizada no ficheiro de configuração de cada um destes nós (papel que desempenha na topologia).

De um modo geral, o módulo ClienteGUI irá ser usado pelo módulo Cliente para interagir com a aplicação de stream de vídeo. Enquanto que o módulo Request será um objeto que irá representar os pedidos do cliente e que se irá propagar pelos nós e no worst case scenario irá chegar ao RP e posteriormente será descodificado e tratado de acordo com o pedido realizado. Por outro lado, a maioria dos componentes irá usar o módulo RtpPacket, objeto que será criado na camada aplicacional e irá transmitir os dados de vídeo em cada frame. Para além disso temos algumas instâncias como mensagens JSON que serão enviadas dos servidores para o RP, de forma a ter informações atualizadas do conteúdo disponível. E para além disso, uma outra instância de mensagens de controlo das condições da topologia.

3 Especificação do(s) protocolo(s)

3.1 Formato das mensagens protocolares

De modo a que o RP tivesse informações das condições dos vários caminhos foi necessário em primeiro lugar, construir a topologia em árvore, de seguida implementar um algoritmo recursivo que devolve uma lista de caminhos entre dois endpoints, e por fim, fazer reencaminhar **mensagens protocolares** por cada um desses caminhos de forma a escolher a melhor trajetória baseado numa métrica de **timestamp**. Assim, a mensagem protocolar possui os seguintes campos:

1. **ini**: timestamp inicial de quando o RP envia a mensagem para o próximo salto (ns).

2. **start**: IP do RP
3. **start_port**: Porta do RP
4. **allpath**: Lista com o caminho do path (a cada hop é feito o pop do primeiro elemento)
5. **received**: timestamp final devolvido pelo último nó do path (ns).

Assim, para testar os scores de cada caminho propaga-se essa mensagem codificada com delimitadores para fazer o parsing necessário no próximo salto. Deste modo, quando o nó é executado este vai estar a ler do socket numa thread adicional, vai verificar o primeiro elemento da lista, retirando-o e enviar para esse mesmo nó. Quando a lista estiver vazia, a mensagem é enviada para o RP (start,start_port) e é feito o cálculo da diferença de timestamps. O caminho com score mais baixo (positivo) é o escolhido (quando existe um nó desativado fazemos catch e o score = -1 para esse caminho).

3.2 Interações

Interações entre os nós no flood do request
Adotando a estratégia de que cada nó terá o seu respetivo ficheiro de configuração JSON, este ao ser inicializado irá à chave neighbors (com a lista de nós dos vizinhos) e para cada um declara um descritor que será necessário para carregar a informação dos ficheiros dos nós adjacentes (cada ficheiro é designado pelo nome do nó). Assim, para cada vizinho, é criado um socket que se conecta a cada um dos nós. Para além disso, guardámos esses sockets numa lista para posteriormente fazermos uma travessia da mesma para enviar o pedido enviado pelo cliente (caso este chegue).

Interações entre os nós na escolha do melhor caminho
Após o cálculo de todos os possíveis caminhos para um dado cliente, procede-se iterativamente ao envio das mensagens protocolares do capítulo anterior. Para isso cada um dos nós terá um thread que estará à escuta numa porta de teste correspondente. Estas mensagens são enviadas usando obviamente o protocolo de transporte TCP pois queremos efetivamente avaliar o tempo de reenvio de pacotes e latências,etc. Como a mensagem tem a lista do caminho cada um dos nós sabe a quem se conectar e enviar a mensagem, sendo que posteriormente o primeiro elemento da lista é removido antes de chegar ao próximo. O RP por sua vez também fica à espera da mensagem de teste final enviada pelo último nó do caminho (cliente) quando este é levantado. Logo, quando executamos o cliente este numa thread adicional irá reenviar para o RP os scores de cada um dos caminhos.

Interações entre os nós no envio de pacotes RTP
Estas interações resultam na reescrita dos ficheiros de configuração de cada um dos nós, já que cada um destes terá de ter informação de qual nó é que deverá reencaminhar os pacotes. Logo, antes de começar o envio é feito posteriormente um setup do caminho e só depois é que começa a transmissão. Esta transmissão

é feito sob o protocolo de transporte TCP. Assim, cada nó terá de consultar o seu ficheiro de configuração para saber o próximo hop para qual deve enviar os pacotes.

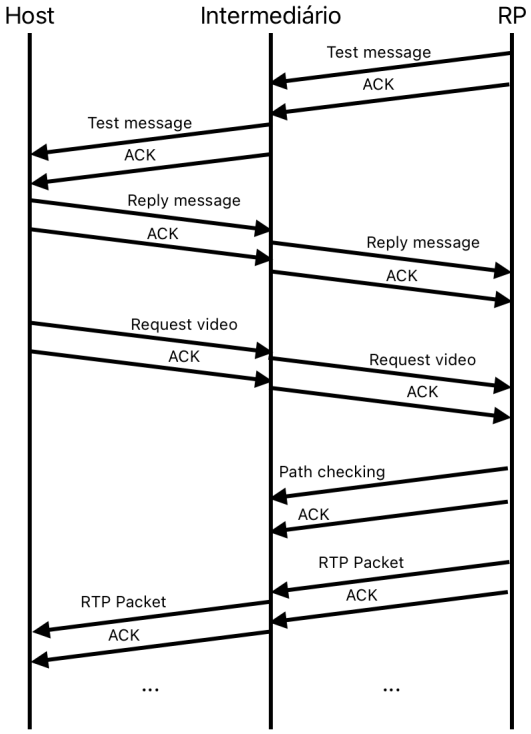

Fig. 2. Interações entre os hosts, RP e nodos intermédios, para a escolha do melhor caminho e envio de vídeo

4 Implementação

4.1 1º Etapa - Preparação das atividades

Nesta primeira etapa foram tomadas as seguintes decisões:

1. **Linguagem de Programação**: Python, uma vez que permite o desenvolvimento de código mais modular o que se torna muito útil para a organização do código em unidades independentes e permite uma conceção mais ágil. Por outro lado, é uma linguagem que possui várias bibliotecas que serão pertinentes para o desenvolvimento deste serviço como o threading, o socket, o json e ferramentas como o tkinter, entre outras.

2. **Protocolo de transporte no overlay**: Numa primeira fase tínhamos optado por implementar o envio de pacotes RTP através do protocolo de transporte em UDP. No entanto, verificámos que estávamos a perder grande parte dos pacotes o que estaria a comprometer a streaming de vídeo (apenas aproximadamente 1/8 das frames estavam a chegar ao cliente). Assim, optámos por TCP, o que também é vantajoso pois é um protocolo reliable já que garantimos a entrega de pacotes e um controlo de erros e para além disso a sua ordem de chegada, porém existe uma maior latência em relação ao UDP.
3. **Implementar um cliente/servidor (simultâneo)**: O desenvolvimento deste modulo foi pertinente para reutilizá-lo no componente do RP e do nó intermédio. De modo a realizarmos esta tarefa criámos duas threads, uma que irá realizar um método que irá aceitar conexões para o seu respetivo (ip,porta) e ler bytes do seu socket. Já a outra thread irá escrever dados no socket para um nó vizinho com um dado (ip, porta).

4.2 2º Etapa - Construção da Topologia Overlay com Árvore Partilhada

Para implementar a nossa rede Overlay, decidimos optar pela segunda estratégia, baseando-se assim em ficheiros locais. Quando o programa é iniciado, os ficheiros de leitura são lidos pela aplicação **Node**, que para além de identificar o tipo de nodo que se está a conectar (ex. router, RP, host, servidor) fica a conhecer também os vizinhos desse nodo. Posteriormente o RP, cria uma árvore que contem os vizinhos de todos os nodos a partir de si mesmo, onde, em seguida, cria um dicionário com todos os caminhos possíveis para alcançar um host a partir do RP. Todos estes caminhos são classificados com um score que é atribuído após mandar um timestamp pelo mesmo. Caso um dos nodos se desconecte esse caminho passará a ter um valor negativo no seu score, para não ser escolhido pelo RP.

4.3 3º Etapa - Serviço de Streaming

Nesta etapa criámos uma classe Request que irá representar o pedido do Cliente. Esta classe terá os seguintes atributos: rtype (tipo do request: pode ser PLAY, PAUSE ou QUIT); vídeo (nome do conteúdo do vídeo a visualizar); três atributos que se vão atualizando a cada salto, uma vez que o pedido é fundido pela rede (flooded menos da interface de onde veio para evitar ciclos) entre estes: Name-nome do nó; Port-nº da porta do nó, IP - ip do nó. Para além desses atributos colocámos mais teres que correspondem à porta de stream que o cliente vai estar à escuta, assim como o ip e por fim o nome do cliente.
No que toca à sincronização da stream (assumindo que esta é feita em real-time) tivemos que adotar algumas estratégias:

– Relativamente à sincronização da streaming de vários clientes colocámos a frame como uma variável partilhada, desta forma, conseguimos saber qual é o número de sequência atual e apresentar o mesmo conteúdo ao mesmo tempo visto que diferentes threads vão ter a mesma cópia da variável;

- Implementámos bufferização dos pacotes RTP, uma vez que o cliente poderá querer fazer Pause e retomar a visualização a partir do momento em que pausou. A partir do momento que o cliente faz um pedido PLAY, a thread irá retirar da fila de espera sempre o primeiro elemento até ficar vazia;
- De forma a disponibilizar o video ao cliente descodificamos os dados recebidos do socket, passámos esses bytes para o construtor do RtpPacket, fazemos o get do Payload e criámos a imagem jpg dessa frame e com recurso ao Tkinter abrimos esse ficheiro. Assim, conforme vão chegando novas frames, as imagens do vídeo vão-se atualizando na interface do cliente.

4.4 4º Etapa - Monitorização dos Servidores de conteúdos

Na etapa 4 o script dos servidores irá ter uma thread a enviar periodicamente (10 em 10 segundos) um unicast para o rp com uma mensagem em JSON que tem os seguintes campos: "fonte": (ip do servidor), "port":(porta do servidor), "timestamp": (tempo em nano-segundos de quando o servidor envia a mensagem) , "conteúdos": (lista de conteúdos multimédia que possui), e por fim, "Estado": (estado do servidor).
No script do RP colocámos uma thread que irá ficar à escuta de unicasts dos servidores e sempre que é aceite uma nova conexão lançámos uma nova thread que irá gerir essa mesma conexão, já que o envio vai ser periódico. De seguida, é feito o cálculo da diferença entre o tempo de chegada e o timestamp recebido e armazenámos todas estas conexões num dicionário em que a chave é o ip do servidor em questão (o que é mais fácil para depois atualizarmos a entrada sempre que uma nova conexão do servidor chega).
Esta dicionário irá funcionar como uma cache onde o RP poderá gerir as conexões e verificar se os servidores estão ativos ou não. Para isso implementámos uma thread adicional que vai fazer uma travessia do dicionário e verificar se a diferença dos timestamps das conexões que estão armazenadas na cache passam dos 10s do tempo atual; em caso afirmativo, a conexão é tagged com "desativado" pois o servidor já não está a enviar unicasts.
Por fim, no RP foi feito um método que vai ás conexões ativas e procura pela conexão com menor duração, para saber assim qual o servidor que faz transmissão de um dado conteúdo em menor tempo possível.
Para além disto, o server terá uma thread extra que estará à espera de que eventualmente o RP o peça para começar a enviar pacotes RTP de um dado conteúdo de multimédia. Em caso afirmativo, o servidor irá proceder à invocação do construtor do VideoStream usando essa instância para obter as várias frames do vídeo e colocar esses bytes em pacotes RTP para ir enviando para o RP até já não existirem mais frames de vídeo.

4.5 5º Etapa - Construção dos Fluxos para Entrega de Dados

Quando não existe nenhuma stream a ser transmitida, o setup do envio da transmissão é importante na aplicação de streaming. Após a escolha do melhor caminho pelo RP, o mesmo instruí a cada nodo a escrever as informações sobre o

próximo salto no seu ficheiro de configuração. Isto permite que em cada nó o vídeo seja enviado pelo caminho certo. Por outro lado, quando um novo cliente pretende iniciar a vizualização, é feito um flood do pedido através da rede até que seja encontrado um nodo que esteja a transmitir conteúdo. Em situações ideais, o sistema localiza rapidamente um nodo disponível e inicia a transmissão a partir do mesmo. Caso não seja possível encontrar um nodo que esteja a transmitir conteudo após o flooding inicial, o RP assume a responsabilidade de criar uma nova stream unicast para esse cliente.

5 Limitações da Solução

Como já anteriormente foi referido, a nossa solução baseia-se numa comunicação TCP, como tal, ao contrário do protocolo UDP é menos eficiente o que implica que a stream sofra um notável quebra de qualidade de visualização devido à existência de algum lag, porém com este protocolo garantímos que todas as frames são enviadas durante o stream de vídeo. Para tal tentamos tomar partido do uso de threads e de um buffer para mitigar ao máximo este problema de latência.
Por outro lado, no que toca à recuperação em caso de falhas a streaming de vídeo fica comprometida caso um dos nós falhe, pelo que num trabalho futuro, o RP adicionalmente teria de gerir a atividade dos nós do caminho escolhido e por sua vez reencaminhar os pacotes por um outro caminho que foi calculado previamente através do algoritmo recursivo de encontrar todos os paths.

6 Testes e resultados

Para a execução de testes procedemos a duas topologias, uma de menor dimensões, para efetivamente testar a transmissão de vídeo e posteriormente, outra onde se assemelha a um caso real com vários nós intermédios e hosts. Para iniciar qualquer nodo, basta executar o comando *python3 NO.py*

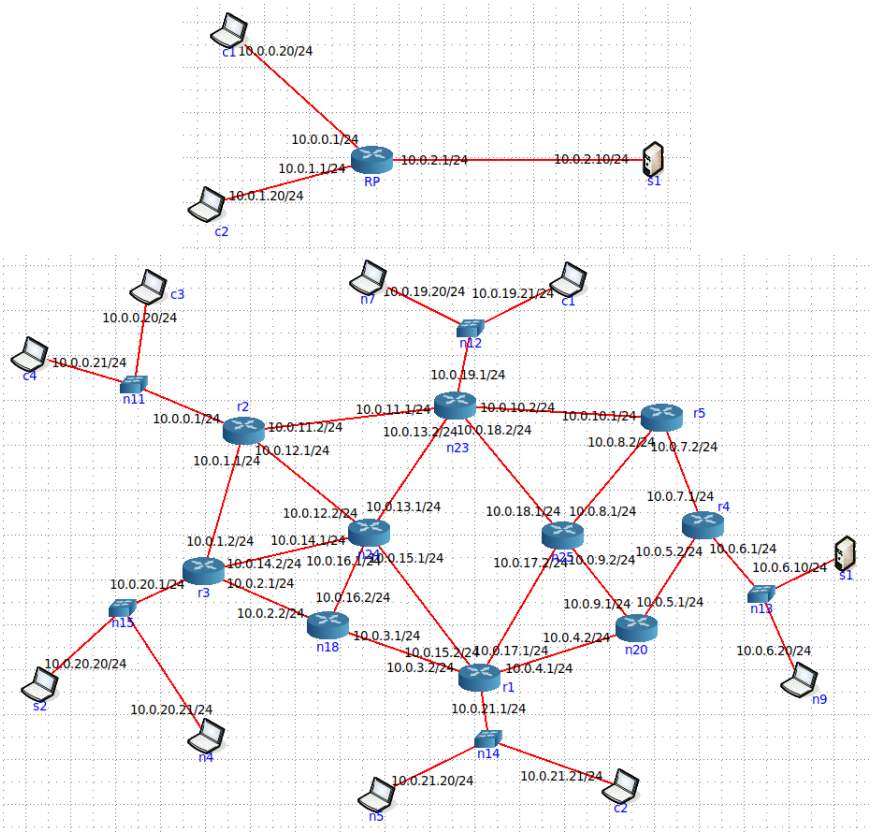

Fig. 3. Topologias usadas para testar a nossa solução

Para demonstrar o funcionamento da nossa aplicação de streaming, vamos optar por utilizar a topologia de maior dimensão, uma vez que a outra tem a mesma lógica, mas mais simples. Após iniciar todos os nodos com sucesso, podemos verificar que após o RP estar ligado este cria automaticamente uma árvore com os nodos dos vizinhos de cada um dos componentes da topologia.

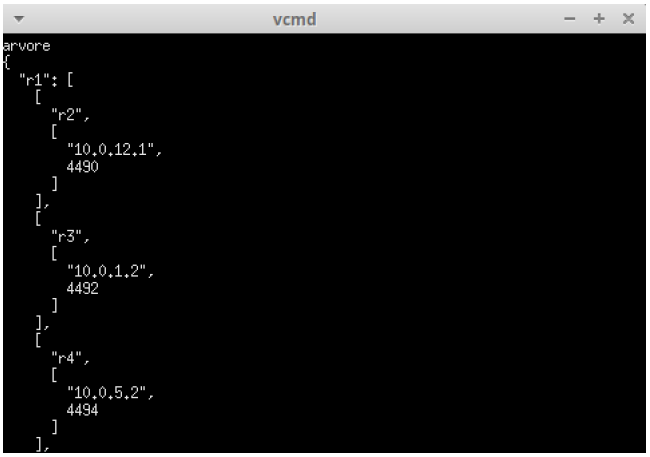

Fig. 4. Árvore dos vizinhos

Para que a nossa aplicação seja um verdadeiro streaming de vídeo, necessitámos de servidores que possuam o conteúdo desejado distribuído (CDN - content delivery network). Na seguinte demonstração vamos procedemer ao levantamento de dois servidores e analisar o trafego gerado pelo wireshark.

Como foi já referido, podemos confirmar o envio de mensagens em unicast por parte dos servidores, de modo que o RP irá posteriormente fazer um controlo dos servidores de onde se encontra o conteúdo distribuído.

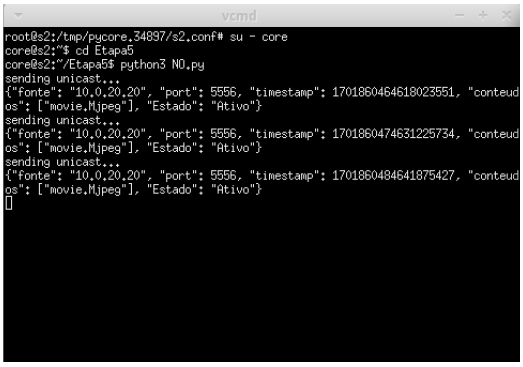

Fig. 5. Mensagens unicast do servidor

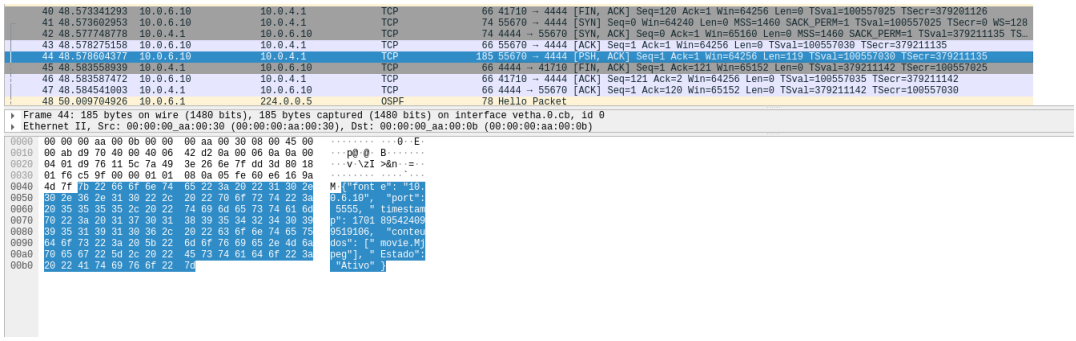

Fig. 6. Captura de uma das mensagens em unicast do S1

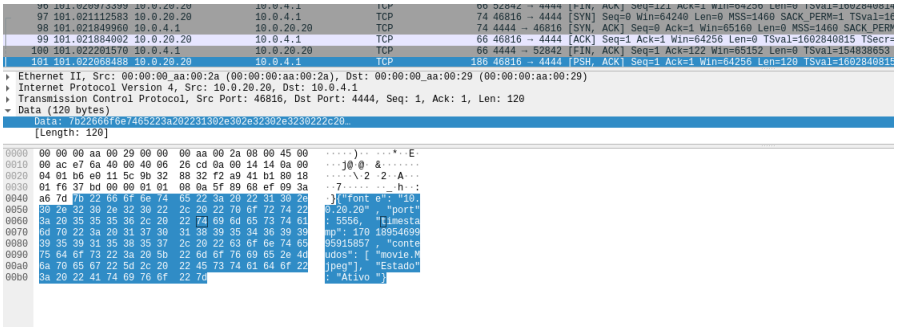

Fig. 7. Captura de uma das mensagens em unicast do S2

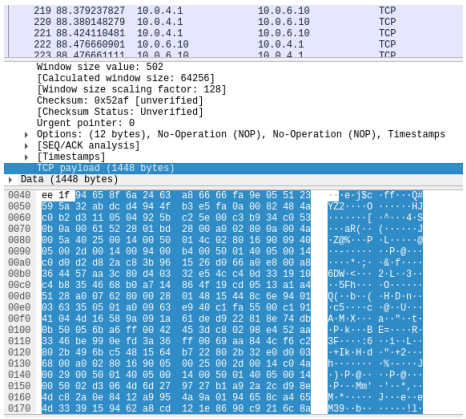

Fig. 8. Bytes do pacote RTP

Neste caso, o servidor escolhido para servir o cliente foi o S1, e de seguida podemos visualizar o conteúdo dos unicasts pelo Wireshark, assim como o con-

teúdo dos dados RTP. Por fim, procedemos à transmissão de vídeo, para tal iniciamos um cliente e procedemos ao play da stream.

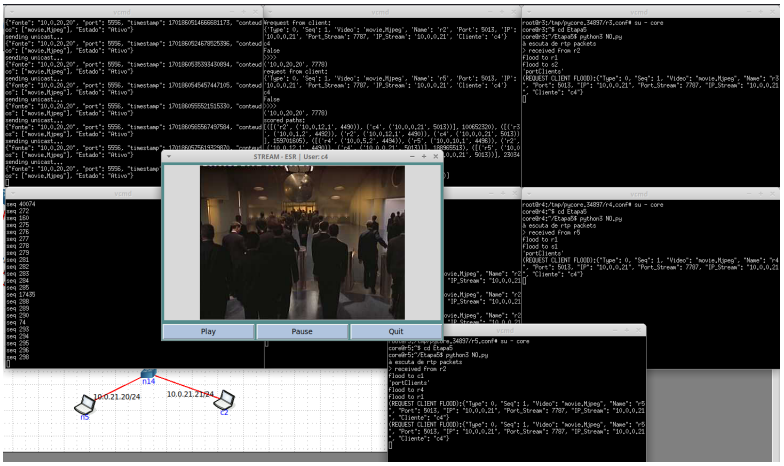

Fig. 9. Um cliente a receber a stream

Adicionalmente, poderíamos ainda levantar mais clientes e todos receberiam a stream em real-time.

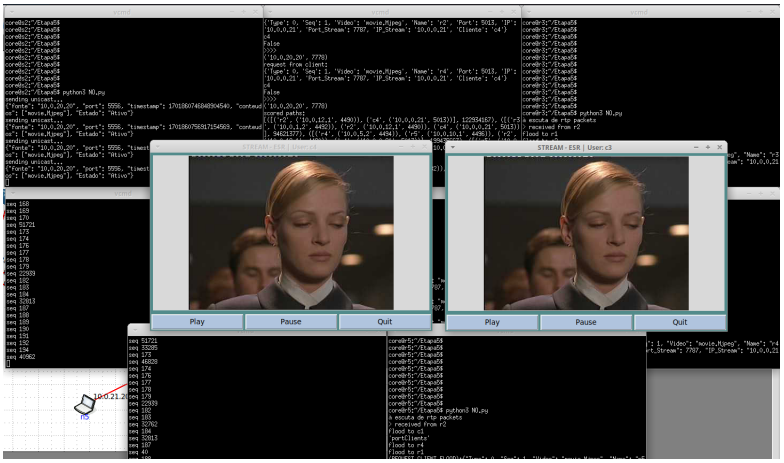

Fig. 10. Dois clientes a participar na stream

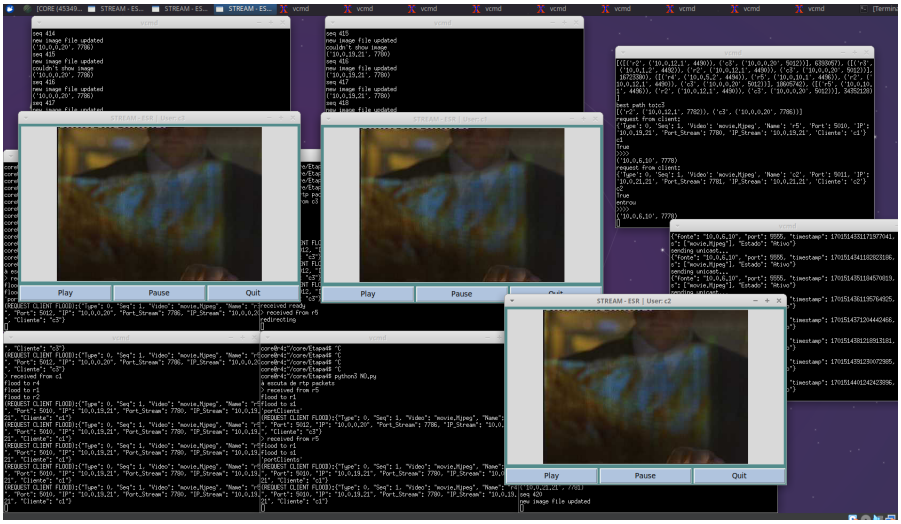

Fig. 11. Três clientes a participar na stream

7 Conclusões e trabalho futuro

Com a realização deste trabalho, pensamos ter alcançado uma solução satisfatória e consistente. Como trabalho futuro, poderiam ser implementadas soluções como uma árvore dinâmica, onde não fosse necessário a existência de ficheiros de configuração, uma stream mais fluida para o cliente. Dado o tempo e o número de elementos do grupo, pensamos ter alcançado a maior parte dos objetivos deste trabalho.