TP2: Serviço Over the Top para entrega de multimédia

Catarina Vieira, Fábio Olivera, and Rita Gomes

[1] Universidade do Minho, Departamento de Informática
[2] Engenharia de Serviços em Rede
[3] 4710-057 Braga, Portugal
[4] {pg50723, pg47107, pg50001}@alunos.uminho.pt

Abstract. O presente relatório concebido como apoio ao trabalho prático 2 da Unidade Curricular de Engenharia de Serviços de Rede tem como objetivo a criação de um protótipo de entrega de áudio, vídeo e texto em tempo real a partir de um servidor de conteúdos para N clientes. O emulador CORE servirá como bancada de testes para o protótipo concebido. A ideia base é a criação de uma rede overlay aplicacional formada por um conjunto de nós que podem ser usados como intermediários para reenviar dados, de forma mais otimizada.

Keywords: Streaming · Rede Overlay · TCP · UDP · Multicast.

1 Introdução

Nos últimos anos a Internet sofreu enormes alterações ao nível do seu paradigma. Cada vez mais são consumidos, de forma arrasadora e a todo o instante, conteúdo de tipos diversificados, o que coloca grandes desafios à infraestrutura IP de base que a suporta. A solução adotada para combater esta entrega massiva de conteúdos passa pelas CDNs (redes sofisticadas de entrega de conteúdos) e com serviços OTT (Over the Top) que são desenhados a nível aplicacional. Para o cliente final obter entregas sem perda de qualidade e em tempo real, estes serviços OTT usam uma rede overlay aplicacional configurada e gerida de modo a arranjar a solução para a congestão e limitação de recursos presente na rede de suporte. A Netflix e outros serviços de streaming, formam uma rede overlay própria sobre protocolos aplicacionais (HTTP) e de transporte (TCP ou UDP), que vai correr sobre a rede IP pública.

O objetivo deste trabalho é criar um protótipo de um serviço semelhante ao fornecido por estas empresas, tendo em conta a eficiência e a otimização de recursos para melhor qualidade de experiência do utilizador.

2 Arquitetura da solução

Primeiramente, o grupo teve de tomar algumas decisões tais como definir qual a linguagem de programação a ser usada e também qual o protocolo de transporte que vai ser usado no overlay (TCP ou UDP).

Devido ao facto de os membros do grupo se sentirem mais confortáveis com a linguagem de programação **Python** tomou-se a decisão que seria esta a linguagem utilizada para a conceção deste projeto.

Quanto ao protocolo definido para a comunicação entre Nodos/Clientes/Servidor, de modo que seja possível descobrir o melhor caminho entre Servidor-Cliente para entregar os pacotes com os dados pedidos, o grupo optou por **TCP**. Esta decisão recai sobre o facto de o TCP não só garantir que as mensagens são entregues como esta entregue é feita de forma ordenada, ao contrário do UDP. Apesar de o TCP não alcançar uma velocidade de transmissão como a que se poderia alcançar com o protocolo UDP, importante em cenários de streaming, consideramos que o TCP é suficiente para o trabalho e permite-nos demonstrar as capacidades pretendidas.

Para além disso, garantimos que caso um nodo seja removido ou adicionado, os melhores caminhos são atualizados, a não ser que seja retirado um nodo crítico, isto é, o único nodo que estabelece conexão a uma secção da rede. Nesse caso não há restabelecimento de conexão. Tudo isto é atualizado nas route tables de cada nodo.

Organizamos os diferentes programas a executar (Cliente, Nodo e Servidor) em diferentes pastas. Existem também outros módulos que são usados em mais do que um destes programas: InfoPacket, RouteTable e RTP; sendo portanto também organizados nas suas próprias pastas.

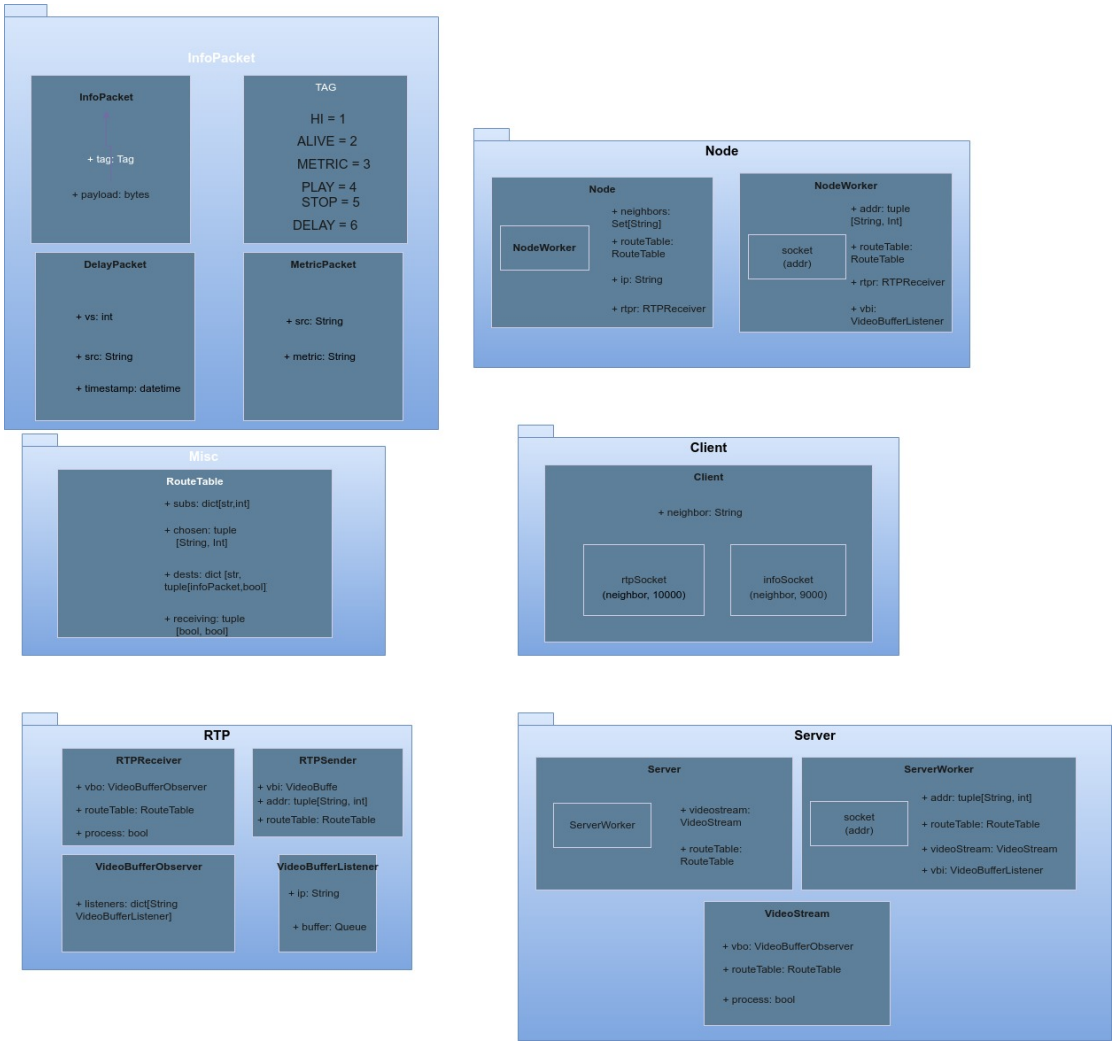

Fig. 1. Arquitetura da solução

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 InfoPacket, Tag e RTPPacket que, como iremos ver de seguida, implementam as mensagens protocolares de controlo e streaming.

Em particular, podemos associar os protolos aplicacionais de controlo a protocolos para gestão de processos de: criação da camada 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 o protocolo de transmissão de dados do streaming entre servidor, nodos e clientes, usamos principalmente o RTPPacket.

3.1 Formato das mensagens protocolares

Routing

- **InfoPacket**: As mensagens protocolares de routing entre nodos são enviadas na forma de um InfoPacket, objeto que contém uma tag para identificar a natureza do pedido ou resposta e um payload que indica possíveis dados complementares ao pacote. Nesta classe também existe a versão dos pacotes com a tag PLAY e STOP, que necessitam do ip do servidor, para os nodos saberem a quem pedir o video.
- **Tag**: A tag pode ser um **"ALIVE"**, usado para informar os nodos que continuam "vivos", **"HI"** que é a primeira mensagem enviada de modo a facilitar com que um nodo saiba os seus vizinhos, **"METRIC"** para enviar aos nodos vizinhos o número de saltos necessários para chegar ao nodo que enviou, **"PLAY"** para ativar uma determinada rota para um cliente, **"STOP"** para a desativar e **"DELAY"** para enviar aos nodos vizinhos o atraso em ligações da rede.
- **DelayPacket**: O DelayPacket é um pacote que é enviado de 30 em 30 segundos pelos servidores, que contém um timestamp para depois, o cliente fazer o calculo do delay.
- **MetricPacket**: O MetricPacket é o pacote responsável de enviar a metrica perante cada servidor, o nodo ao receber esse pacote, vê a métrica que o vizinho lhe enviou e de que servidor vem, para saber que RouteTable atualizar.

Video

- **RTPPacket**: Tendo em conta que os vídeos que serão transmitidos terão de ser divididos em N packets para posteriormente mandar pelo caminho determinado, o RTPPacket representa cada um desses packets. Contém uma flag "more" que representa se há mais packets a enviar e um payload que representa a data do vídeo a enviar ou a receber. 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.

3.2 Interações

NodeWorker: Thread que trata da comunicação entre Nodos, Clientes e Server. Contém o endereço do Node/Client/Server para onde vai enviar/receber InfoPackets, a RouteTable do Node em questão e socket para estabelecer conexões com outros Node/Client/Server. Quanto ao envio de mensagens protocolares

aquando streaming, cada Nodo cria um novo NodeWorker quando é estabelecida uma nova conexão. Um NodeWorker contém um RTPReceiver, cuja source é o chosen, que é partilhado pelos outros NodeWorkers e pode conter um RTPSender.

RTPSender: É uma thread responsável por estabelecer conexão com um determinado nodo e enviar RTPPacket para o mesmo. Consegue-o fazer pois contém a address desse nodo e um VideoBufferListener que por sua vez conterá o packet para enviar desejado numa queue de packets.

RTPReceiver: É uma thread responsável por estabelecer conexão com um Nodo, receber e descodificar o packet recebido e adicionar o packet que recebeu ao seu VideoBufferObserver e consequentemente aos VideoBufferListeners respectivos. Contém uma variável boleana que se estiver a True sabemos que já está a receber dados de algum lado.

VideoBufferListener: Classe que recebe os packets de vídeo e coloca os mesmos numa queue garantindo que o envio dos packets posteriormente é ordenado.

VideoBufferObserver: Classe que gere os VideoBufferListeners de um Nodo e permite adicionar um packet de vídeo a todos os VideoBufferListeners ativos de um Nodo.

VideoStream: Classe que lê de um determinado ficheiro vídeo e cria os respetivos RTPPackets.

4 Implementação

Decidimos guardar na estrutura RouteTable uma variável chosen que é a ligação escolhida (ligação com melhor métrica) caso haja ativação de rotas, nessa variável guardamos um tuplo com o ip da source e a métrica. Devido ao facto de existir mais do que um servidor, cada nodo vai ter um dicionário em que a key é o ip de um dos servidores e no campo value irá ter uma route table. Quando há mudanças no sistema, o nodo sabe qual das duas route tables alterar, porque o pacote da métrica leva também a source (que é o ip do servidor). Temos também uma variável subs que é um dicionário que contém as outras sources do nodo e as suas respetivas métricas para o caso do chosen morrer podermos atualizar o novo melhor chosen ou se, por exemplo, for adicionado um novo nodo e a sua métrica não for melhor, será adicionada à variável subs. Por último, temos um dicionário dests que guarda os ips dos destinos do nodo como chave e um tuplo InfoPacket e um boleano que simboliza se a rota está ativa ou não como value. Quando um nodo entra na topologia, este manda uma InfoPacket com a tag “METRIC”, ”DELAY” ou “HI” (dependendo se já tem acesso ao servidor ou

não, respetivamente) para os seus vizinhos, estabelecendo conexão com estes. Após isto, este começa a enviar uma InfoPacket com a tag “ALIVE” de segundo em segundo enquanto estiver vivo. Os vizinhos respondem com a mesma tag, se não tiverem outra InfoPacket de tag diferente para mandar. Caso a resposta seja com a tag “METRIC” ou ”DELAY”, a RouteTable desse nodo é atualizada.

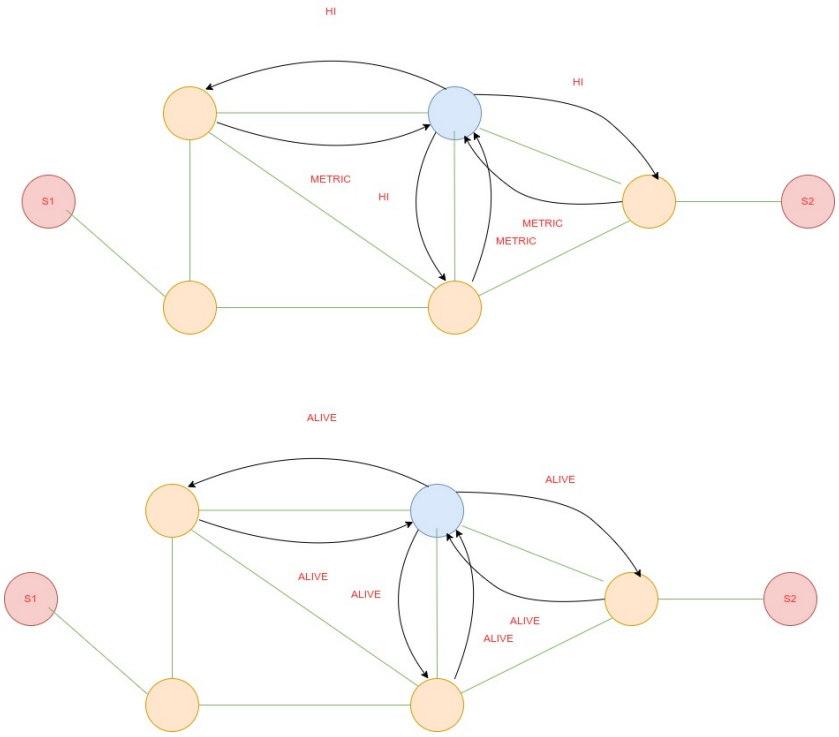

Fig. 2. Tag’s respetivas à entrada de um novo nodo na topologia e Tag’s enviadas para os vizinhos saberem que o nodo está vivo

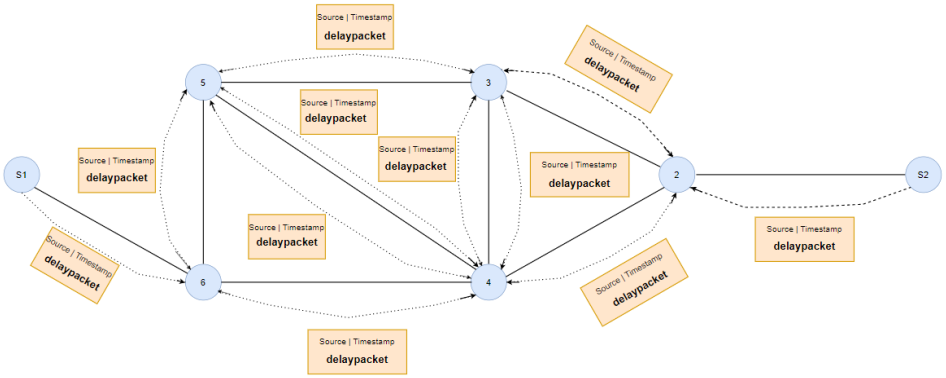

Fig. 3. Delay Packet

Quando é adicionado um novo nodo, os seus vizinhos vão atualizar as suas RouteTables. Os vizinhos do nodo adicionado enviam-lhe as suas métricas ("METRIC" e "DELAY") e o novo nodo atualiza a variável chosen na sua RouteTable com base no vizinho com menor métrica (menor número de saltos e menor atraso sofrido desde o seu envio até à sua receção). Os seus vizinhos fazem o mesmo, se a métrica que o novo nodo enviar for melhor e atualizam as suas variáveis chosen com base no mesmo, se não, atualizam o seu dest com o novo nodo. O mesmo acontece sucessivamente pelos seus vizinhos se a métrica for melhor.

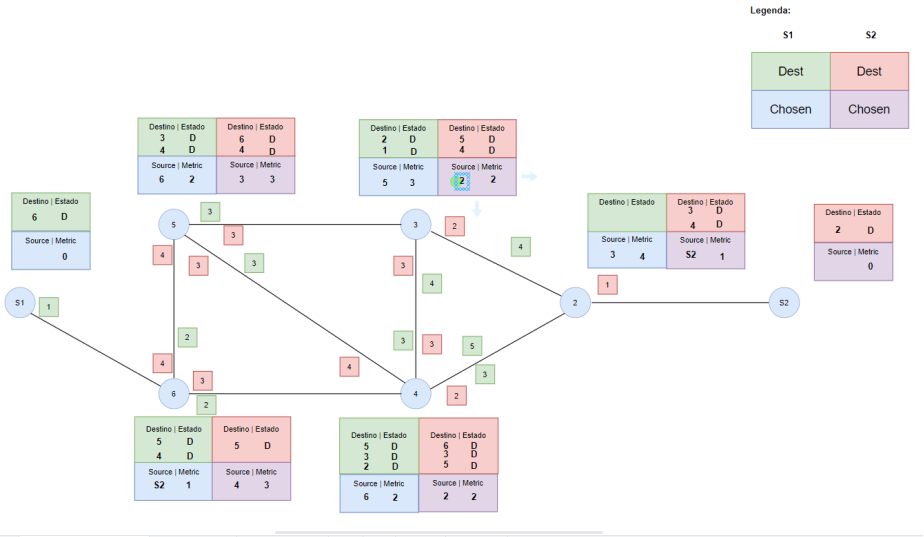

Fig. 4. Topologia antes de adicionar um nodo

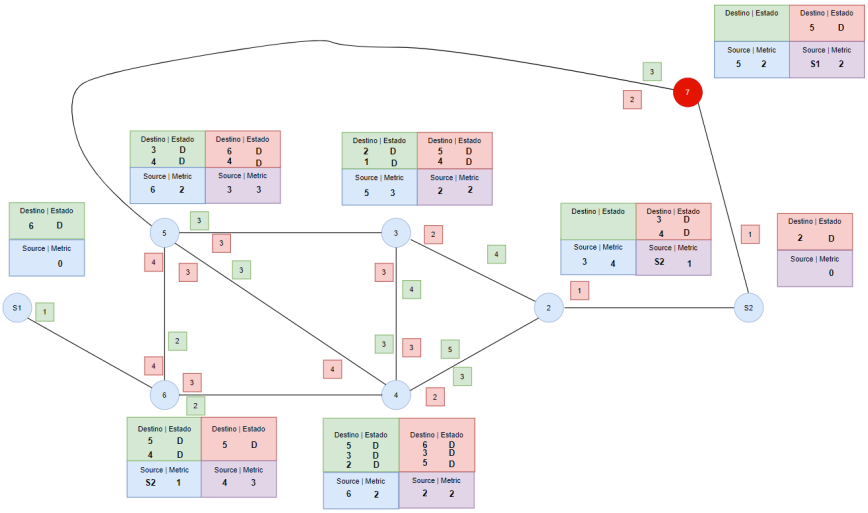

Fig. 5. Topologia depois de adicionar um novo nodo

Quando um nodo deixa de responder, o nodo é removido da topologia e as route tables dos seus vizinhos são atualizadas.

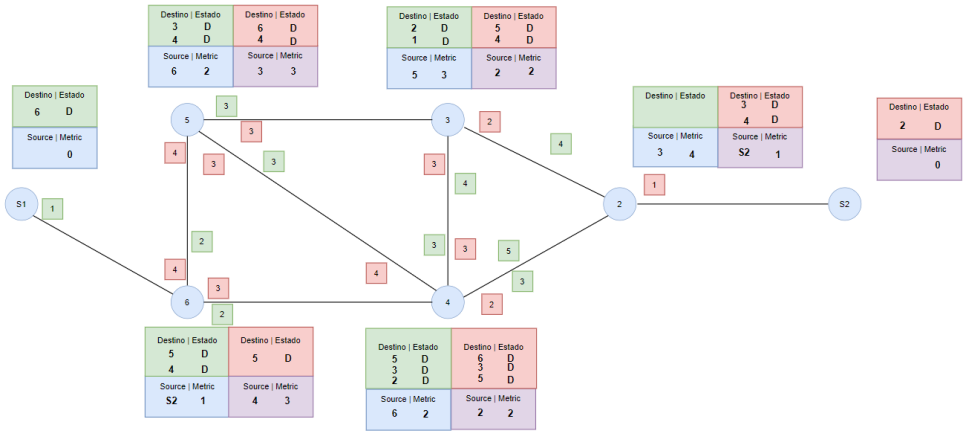

Fig. 6. Topologia antes de um nodo morrer

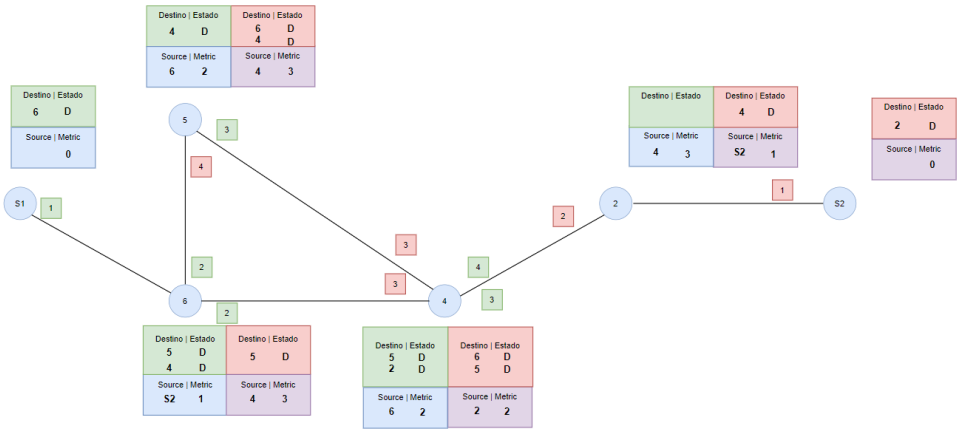

Fig. 7. Topologia depois do nodo removido

Quando um cliente quer receber dados, este informa o nodo a que está ligado que quer receber dados pela tag "PLAY", mal ele dá essa informação a ligação entre o cliente e o nodo passa a estar ativa. O nodo que recebeu a tag, informa ao seu chosen que quer receber dados, passando essa ligação também a estar ativa e assim sucessivamente até ao servidor.

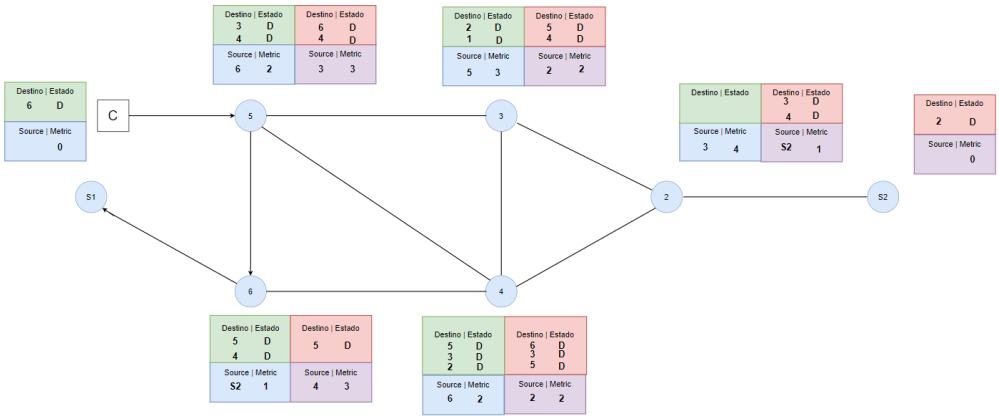

Fig. 8. Topologia antes de receber a tag "PLAY"

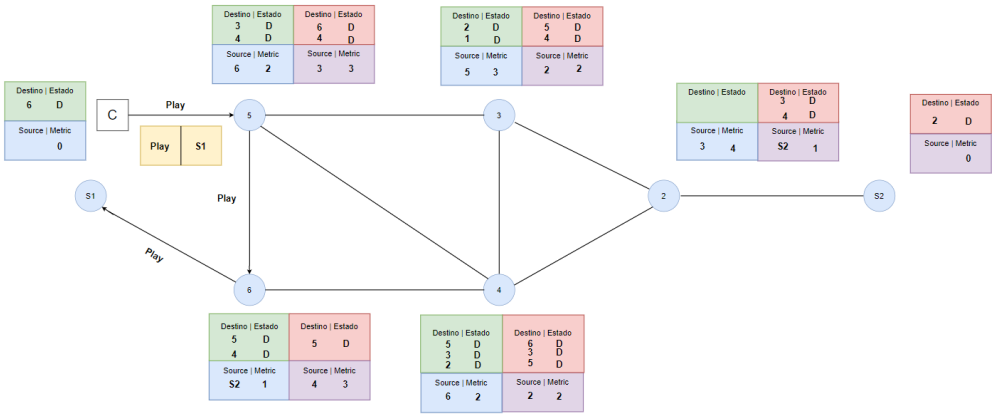

Fig. 9. Topologia depois de receber a tag "PLAY"

Quando o cliente quer deixar de receber dados o processo é equivalente ao de receber dados, mas neste caso, ele envia a tag "STOP" para o nodo a que está ligado e essa ligação passa a estar desativada. O mesmo acontece para os chosen dos nodos seguintes caso esta seja a única rota ativa do nodo.

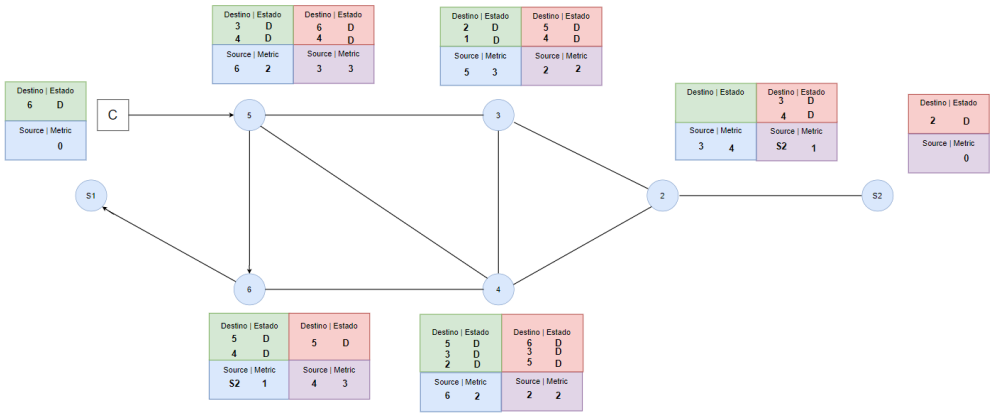

Fig. 10. Topologia antes de receber a tag "STOP"

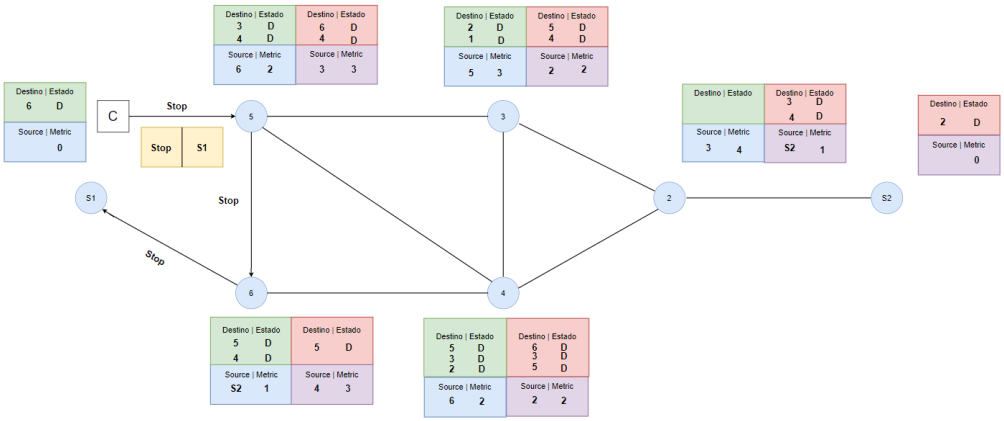

Fig. 11. Topologia depois de receber a tag "STOP"

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 a seguir encontra-se a topologia underlay que será usada em todos os testes.

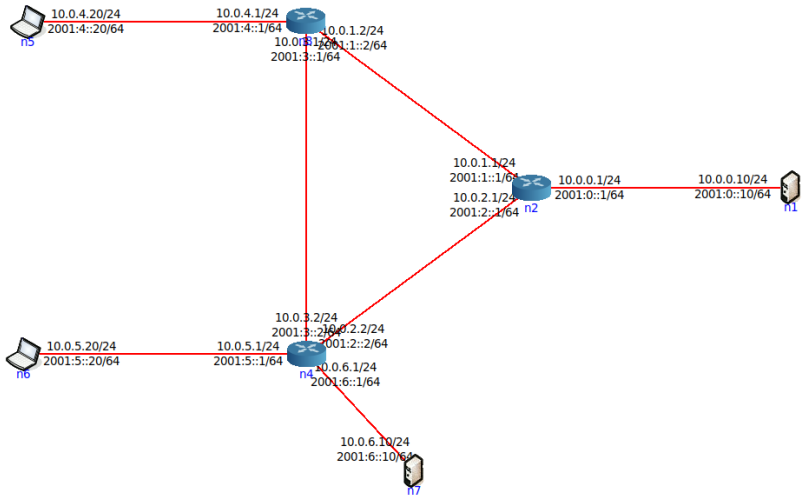

Fig. 12. Topologia utilizada para cenários de testes

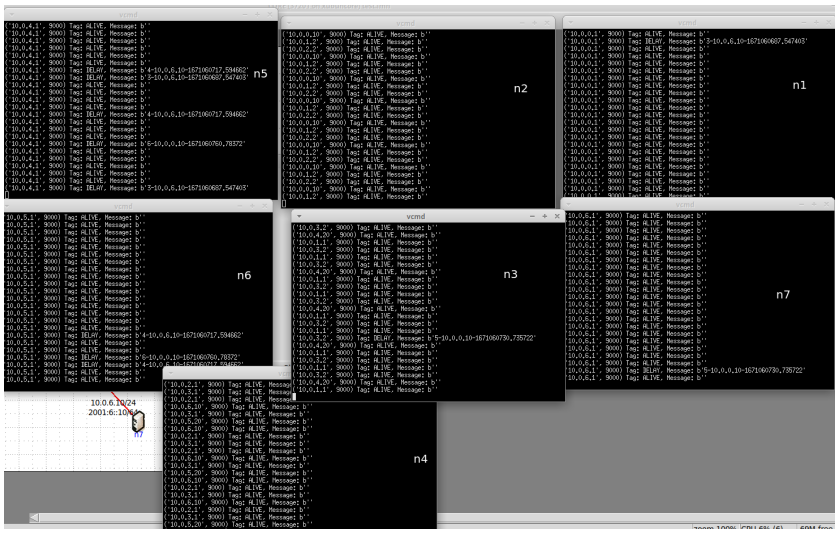

Fig. 13. Conexão estabelecida

6 Conclusões e trabalho futuro

Com a realização deste trabalho, o grupo conseguiu consolidar todo o conhecimento referente a protocolos e serviços de streaming tanto multimédia como mulicast adquirido ao longo da Unidade Curricular.

De uma forma geral, o sistema é capaz de detetar e corrigir alguma rota que deixe de funcionar e os servidores são capazes de transmitir o vídeo, simultaneamente, para quaisquer Clientes da topologia bastando apenas uma transmissão unicast para um nodo adjacente ao invés de fazer unicast para cada um dos Clientes, utilizando para isto retransmissão e replicação nos nodos intermédios da topologia.

O grupo reconhece que existem alguns erros que devem ser corrigidos em trabalho futuro, tais como:

- Quando um Cliente indica que quer parar de receber os dados, os restantes clientes também param, o que não deve acontecer pois as transmissões de cada cliente devem ser independentes das restantes.
- Durante a transmissão do conteúdo do vídeo, se um nodo intermédio presente no caminho de transmissão parar, a transmissão não é recuperada por outra rota, apesar de a rota ser recalculada. Este ponto vai de encontro ao ponto anterior.

Para além disso, há espaço para melhorias, e como trabalho futuro seria interessante implementar uma transmissão de pacotes de vídeo mais rápida, recorrendo ao uso do UDP.

A maior dificuldade do grupo foi encontrar uma maneira de manter os nodos informados sobre a atividade dos restantes nodos – se estão vivos ou não - de modo a atualizar as suas rotas.

Em suma, o grupo considera que o balanço do trabalho é positivo e os requisitos básicos propostos foram cumpridos.