**Escola de Engenharia**

Mestrado em Engenharia Informática

Engenharia de Serviços em Rede

TP3 - Serviço Over the Top para entrega de multimédia

# Grupo 37

# Pedro Veloso (pg47578)

# Carlos Preto (pg47089)

Braga, Dezembro de 2021

# Introdução

Ao longo dos últimos anos observou-se uma mudança relativa ao paradigma da internet. A utilização de serviços de comunicação extremo-a-extremo deixou de ser o serviço mais procurado, sendo ultrapassado pelo consumo de serviços em tempo real, que teve um crescimento enorme. Assim, de modo a suportar as exigências que este novo tipo de consumo demanda, criaram-se algumas estratégias, como por exemplo, a criação de redes sofisticadas de entrega de conteúdos (CDNs) específicos, desenhadas sobre a camada aplicacional, e por isso ditas *Over the Top* (OTT).

Assim, este trabalho tem como objetivo criar uma rede de servidores (*Overlay*) que permitem disponibilizar conteúdos de *streaming* (em tempo real) aos clientes conectados, contornando os problemas de congestão e limitação dos recursos, associados à rede de IP pública.

# Arquitetura da solução

Para disponibilizar o serviço aos clientes deve ser desenvolvido um servidor central, denominado por *bootstrapper*, que é responsável por conhecer a topologia e reagir a eventos realizados. Posteriormente, devem ser desenvolvidos servidores *ott* (responsáveis por reencaminhar conteúdo), *streamer* (responsáveis por realizar a *stream* e coordenar a criação das rotas do *overlay*) e servidores “cliente” (responsáveis por visualizar a *stream* a decorrer).

Em primeiro lugar foi desenvolvido o servidor *bottstrapper*. Este servidor foi pensado como responsável por controlar a conexão e desconexão dos restantes servidores. Assim, sempre que um nodo se conecta à rede *overlay*, é da sua responsabilidade conectar-se, primeiramente, com o *bootstarpper*, que por sua vez deve indicar-lhe os nodos com quem pode comunicar. Também deve indicar a cada um dos nodos vizinhos a chegada do novo nodo. Por outro lado, quando um nodo se desconectar (voluntária ou involuntariamente) é responsabilidade do *bootstrapper* perceber o ocorrido e avisar os nodos que dependiam dele.

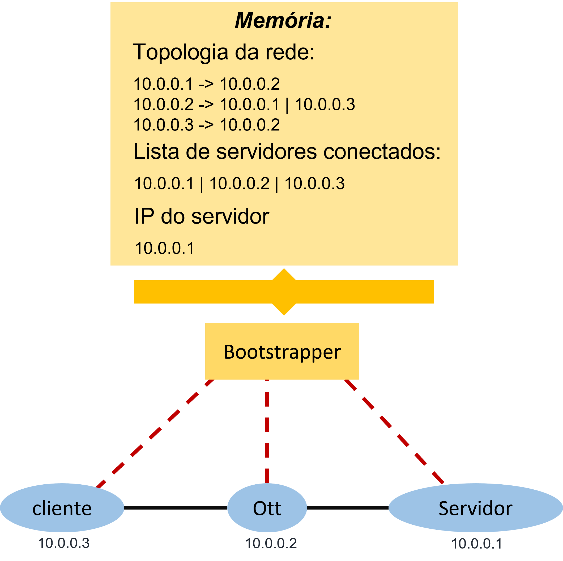


Figura 1: Exemplo dos dados em memória de um servidor bootstrapper

Posteriormente, foram desenvolvidos os servidores *Ott*, que têm como função distribuir, para os seus vizinhos, os pacotes recebidos do servidor de *streaming*, para os clientes que pretendam visualizá-los. Assim, tal como referido acima, a primeira tarefa destes servidores deve ser conectar-se com o *bootstrapper*, criando uma tabela com os seus vizinhos associados á necessidade de retransmitir-lhes, os pacotes de *streaming* recebidos. Com esta informação em memória é possível realizar a inundação dos pacotes que contém o conteúdo da *streaming*. Este processo consiste em receber os pacotes (enviados em primeiro lugar pelo servidor) e retransmiti-los para todos os vizinhos que possuem a interface ativa na tabela de rotas.

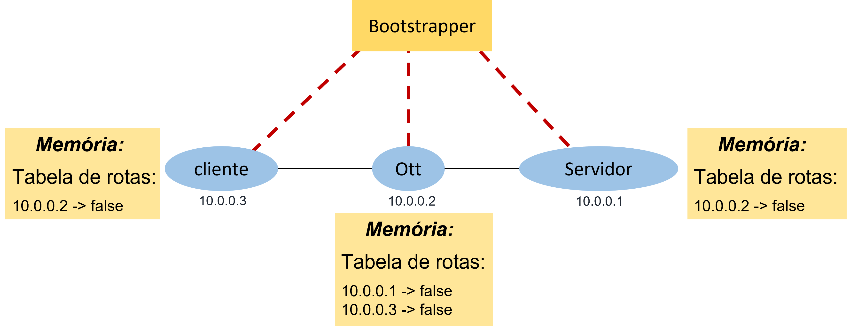


Figura 2: Exemplo de tabelas de rotas em cada servidor (obtidas após a conexão com o bootstrapper)

Após esta fase, é necessário conhecer o vizinho que permite chegar com o menor custo ao servidor. Para tal, é necessário que cada *ott* receba e realize a inundação (controlada) das rotas iniciada pelo servidor, depois de atualizar o custo recebido. Este processo permite adicionar à memória o *ip* da interface que aproxima o nodo do servidor, bem como o custo dessa ligação e o endereço do servidor.

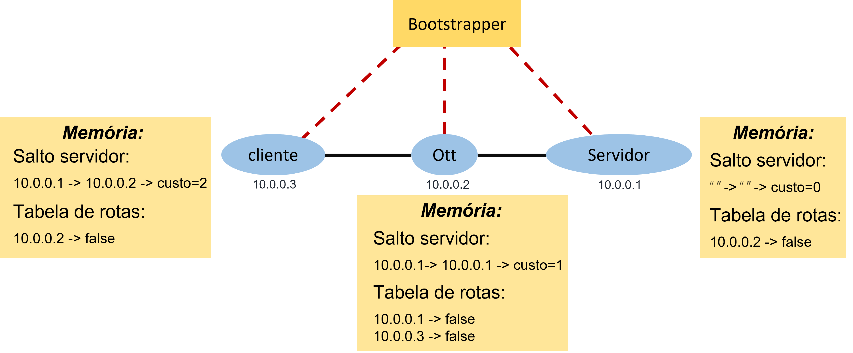


Figura 3: Exemplo de dados de routing em memória de cada servidor

Quando estes servidores possuírem esta informação em memória, é possível receber pedidos de vizinhos para ativar (corresponde a colocar como verdadeiro a interface desse vizinho na tabela de rotas) e desativar (colocar como falso a interface desse vizinho) o serviço de *streaming* – estes tipos de pedidos são iniciados pelos clientes.

Em terceiro lugar, desenvolveram-se os servidores de *streaming* responsáveis por iniciar o processo de inundação das rotas, bem como transmitir pacotes com os *frames* do vídeo que está a transmitir. Assim, a primeira tarefa deste servidor é conectar-se com o *bootstrapper* e obter a sua tabela de rotas (figura 3). Posteriormente, deve transmitir aos seus vizinhos uma mensagem para atualizarem os seus melhores caminhos. O envio desta mensagem é desencadeado pelo início do servidor de *streaming* ou qualquer outro nodo da rede. Porém, também pode ser enviada quando existe a desconexão de um nodo, ou periodicamente, de modo a manter a rede atualizada. Por fim, os *frames* do vídeo devem ser enviados para todos os vizinhos com a interface ativa (na tabela de rotas), após o início da *stream*.

Em último lugar, desenvolveu-se o servidor para os clientes. Estes servidores são responsáveis por transmitir o vídeo ao utilizador. Para tal, devem enviar mensagens de ativação ou desativação de rotas, ao nodo que melhor o aproxima do servidor, bem como receber e tratar as mensagens que possuem os dados da *stream*.

# Especificação do(s) protocolo(s)

De modo a suportar a solução acima idealizada, foram criados três protocolos distintos – *Overlay*, *Routing* e *Streaming Protocol*. Estes protocolos recorrem a protocolos de transporte TCP e UDP respetivamente.

## *Overlay Protocol*

Este protocolo é utilizado na comunicação entre os servidores e o *bootstapper*. Assim, como o *bootstrapper* deve monitorar o estado dos servidores (se estão ativos ou desconectados), utiliza-se como protocolo de transporte o TCP. Uma vez que o TCP é orientado à conexão, ao estabelecer um princípio de sessão, permite determinar, em qualquer um dos lados a desconexão do outro interveniente.

### Mensagens protocolares

Uma vez que o as mensagens trocadas entre os nodos e o *bootstrapper* são utilizadas para anunciar a lista de vizinhos, criou-se os seguintes campos para as mensagens protocolares:

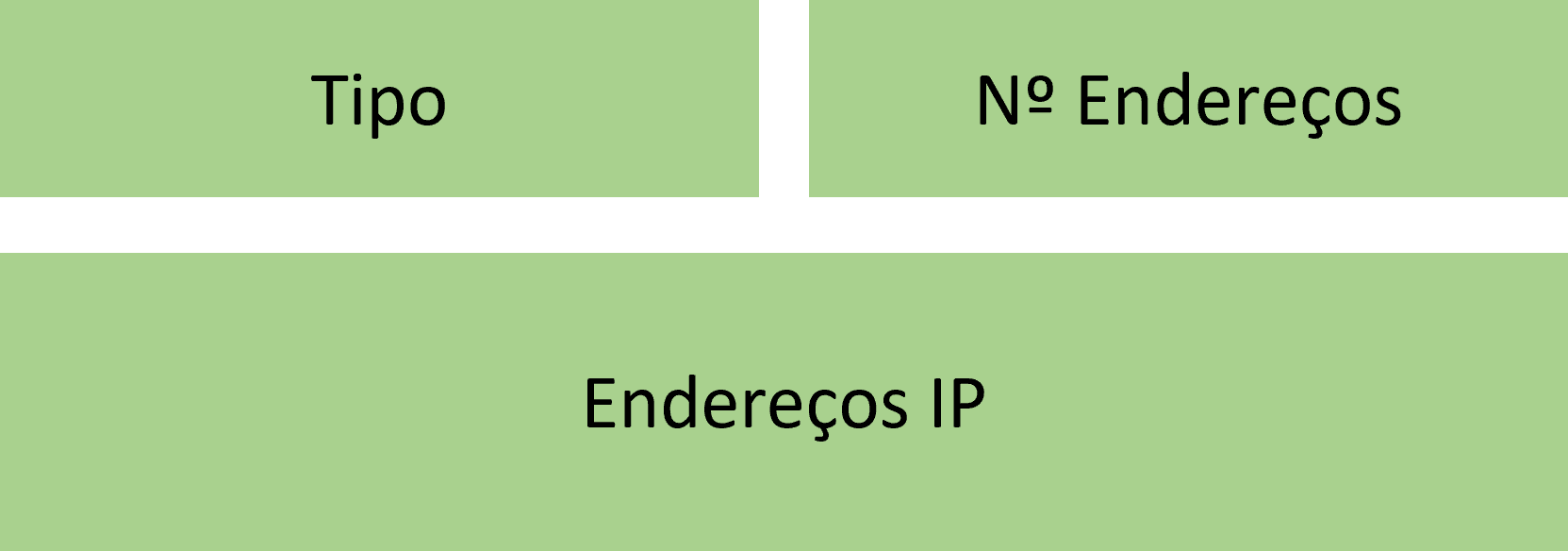


Figura 4: Mensagem protocolar “Overlay”

Tal como se observa na figura acima, existe um campo onde é indicado o tipo de mensagem, um outro campo onde é indicado o número de endereços *ip* que a mensagem transmite e por fim no último campo transmite-se a lista com os endereços *ip*.

O campo “Tipo”, de modo a simplificar o desenvolvimento, é representado por um caracter, utilizando desta forma 4 bytes, porém, uma vez que apenas são necessários 4 tipos de mensagens diferentes (para cada sentido da conexão), seria possível utilizar 3 bits para a sua representação. Posteriormente, utiliza-se mais 4 bytes (num inteiro) para a representação do número de endereços e x\*y bytes na representação da lista de endereços (sendo x = nº de endereços e 15 >= y >= 7).

### Interações

Sempre que um servidor cliente/*ott*/*streamer* se conecte deve indicar ao *bootstrapper* a sua conexão. Para tal, envia uma mensagem do tipo <caractere, 0, []> onde o caractere pode ser C (no caso de ser um cliente), S (no caso de ser um servidor de *streamer*) ou I (no caso de ser um *ott*).

Posteriormente, o *bootstrapper* envia uma mensagem para o nodo que se conectou com a sua lista de vizinhos ativos (<V, número de vizinhos, lista com os vizinhos>) e uma mensagem para cada um dos vizinhos desse nodo (<N, 1, *ip* do servidor que se conectou>) a informar que agora tem um novo vizinho. Caso a conexão seja de um cliente ou de um *ott* o *bootstrapper* também envia uma mensagem para o servidor (<C, 0, []>), a informar que deve começar a inundação da rede com a mensagem de atualização das rotas.

Por fim, quando um servidor se desconecta, o *bootstrapper* deve enviar uma mensagem para cada um dos seus vizinhos, a informar a ocorrência (<D, 1, *ip* do servidor que se desconectou>).

## *Routing Protocol*

Este protocolo foi desenvolvido para transportar mensagens que permitem aos servidores determinar o vizinho que tem o menor custo para atingir o servidor de *streaming*. Na solução implementada, utilizou-se o UDP como protocolo de transporte.

### Mensagens protocolares

De modo que os servidores realizem o apuramento da melhor rota para o servidor, criou-se mensagens protocolares com os seguintes campos:

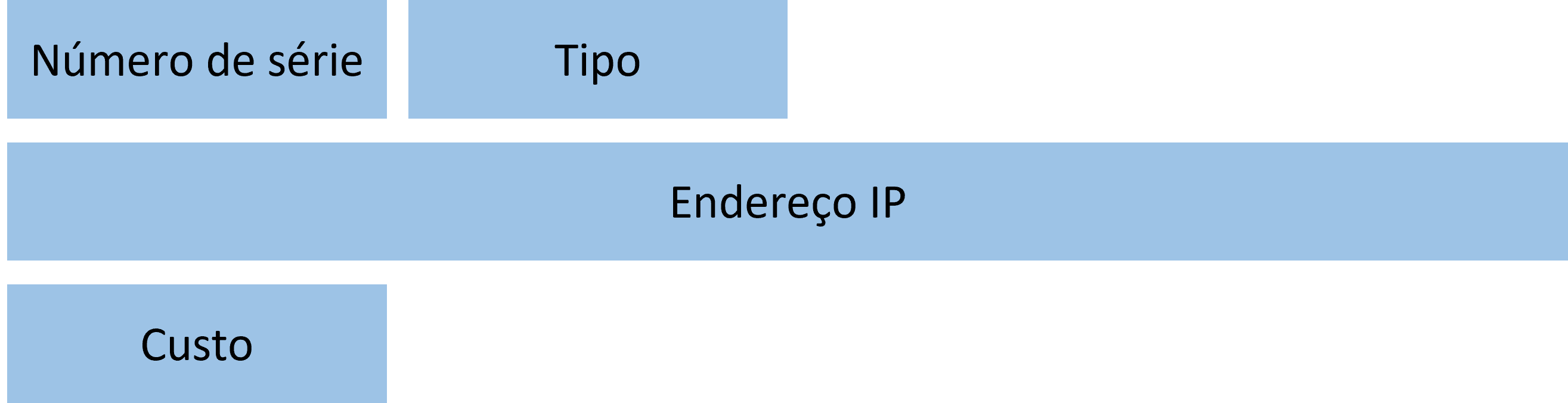


Figura 5: Mensagem protocolar “Routing”

Ao observar a figura é possível perceber que existe um campo utilizado para identificar de forma única cada mensagem (nº de série), representado como um inteiro (4 bytes). Este campo é utilizado pelos servidores, de modo a determinar se a mensagem que estão a receber não é repetida, ignorando-a sempre que tal não se evidenciar. Posteriormente, utiliza-se o campo “Tipo” para determinar o tipo de mensagem. Neste tipo de mensagens existem três tipos distintos, o tipo referente ao transporte de rotas e os respetivos custos, o tipo relativo a mensagens para ativar interfaces e o tipo para desativar interfaces (0, 1 e 2, respetivamente). Assim, é apenas necessário utilizar 2 bits para representar este campo, contudo, uma vez mais de modo a simplificar, utiliza-se um inteiro como tipo de dados (4 bytes). Por fim, utilizam-se 15 bytes para representar o endereço *ip* e 4 bytes (inteiro) para representar o custo da rota.

Desta forma, a utilização destas mensagens protocolares implica o transporte de 27 bytes entre servidores.

### Interações

De modo a contruir as rotas, o servidor deve mandar uma mensagem (<id, 0, *ip* do servidor, 0>) que passará por todos os restantes servidores. Cada servidor, ao receber esta mensagem, deverá atualizar o custo e retransmiti-la de forma controlada (evitar criar ciclos, de modo, a que a mensagem não circule na rede de forma persistente) para os seus vizinhos.

Quando um servidor pretender começar a receber conteúdo deverá enviar uma mensagem (<0, 1, *null*, *null*) para ativar a sua interface em todos os nodos no caminho até ao servidor. Por outro lado, se o servidor pretender de deixar de receber conteúdo, deve enviar uma mensagem que irá fechar a sua interface nos respetivos nodos do caminho (<0, 2, *null*, *null*>).

## *Streaming Protocol*

De modo a transportar o *frame* atual do vídeo, utiliza-se este protocolo, que divide a totalidade do *frame* em x linhas e y colunas, enviando x\*y mensagens para os servidores vizinhos ativos. A divisão do *frame* em partes mais pequenas, permite que no caso de se perda de alguma mensagem, não se perca a totalidade do *frame*. Para além disto, envia-se as linhas todas de forma aleatória, reduzindo assim a probabilidade de se perder várias linhas/colunas seguidas, o que podia levar a uma disrupção de um *chunck* grande do *frame*.

Por fim, estas mensagens protocolares utilizam como protocolo de transporte o UDP.

### Mensagens protocolares

Para a realização destas mensagens protocolares (figura 6) é necessário indicar o número do *frame* que se está a transmitir, seguido da linha e do offset a que os dados correspondem. Para cada um dos parâmetros identificados utiliza-se como estrutura de dados um inteiro, utilizando por isso 4 bytes para cada um (12 bytes no total). Por fim, indica-se o tamanho dos dados a transportar (num inteiro – 4 bytes) e os dados do *frame*.

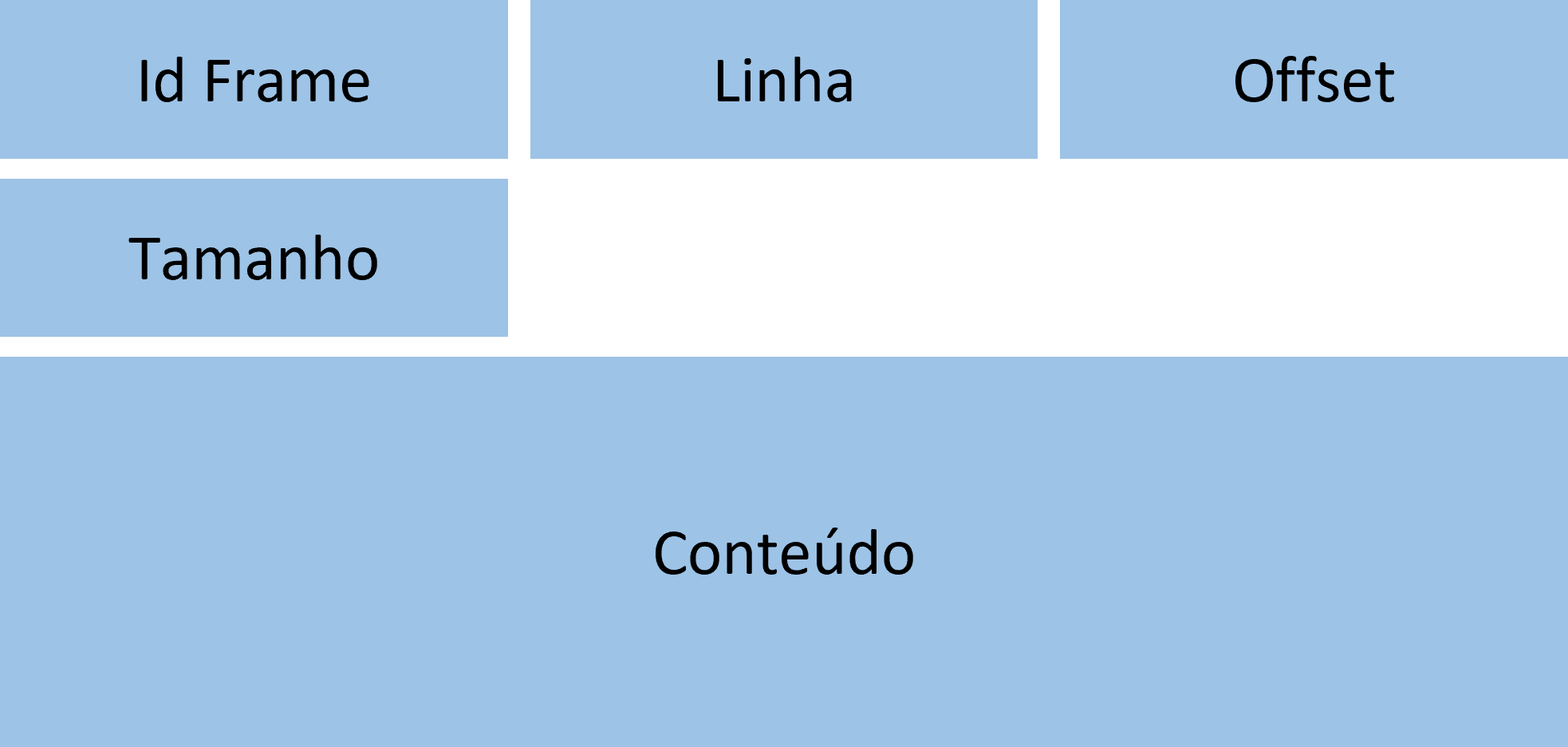


Figura 6: Mensagem protocolar “Streaming”

### Interações

Para cada *frame* o servidor de *streaming* deve enviar n mensagens do tipo: <id do *frame*, número da linha, número da coluna, número de bytes que vai transportar no campo dados, conteúdo do *frame*>, para todos os vizinhos que tenham a sua interface ativa na tabela de rotas do servidor. Posteriormente, todos os servidores que não sejam cliente devem receber esta mensagem e enviá-la para todos os endereços *ip’s*, dos seus vizinhos que também tenham as interfaces ativas.

# Implementação

Durante a fase de implementação, desenvolveu-se a solução apresentada acima na linguagem de programação JAVA. Assim, criou-se quatro programas diferentes (funções *main* distintas), para cada um dos tipos de servidores (*bootstrapper*, *streamer*, *ott* e cliente). Deste modo, para utilizar o servidor do *bootstrapper* apenas é necessário executar o programa, enquanto para todos os outros é necessário indicar (pelos argumentos do programa) o endereço *ip* do *bootstrapper*. Para além disto, no programa relativo ao servidor, indica-se o caminho para o ficheiro TXT que contém o conteúdo do vídeo.

Por fim, no desenvolvimento do *bootstrapper*, considerou-se a existência de um ficheiro denominado “*overlay*” na mesma diretoria do executável (ex: .jar) que contém, na primeira linha o endereço do servidor e nas restantes uma associação entre um nodo e as suas aresta (ex: 10.0.12.10:10.0.11.10,10.0.15.10).

Refere-se também a opção de utilizar vídeos em formato de texto, para a qual se desenvolveu uma simples interface gráfica (utilizando a biblioteca java swing), que permite visualizar o vídeo, bem como realizar o play/stop do vídeo.

Uma imagem com texto

Descrição gerada automaticamente

Figura 7: Interface gráfica criada (modo branco)

Durante a fase de implementação, decidiu-se que cada *frame* deve ser mostrado ao utilizador assim que se receba conteúdo do *frame* seguinte. Desta forma, o cliente está apenas alguns milissegundos dessincronizado com a *stream*.

# Testes e resultados

Para realizar testes ao serviço criado, desenvolveu-se (no *core*) a topologia da rede representada na figura seguinte (à esquerda), que corresponde à rede, da direita, no *overlay*.

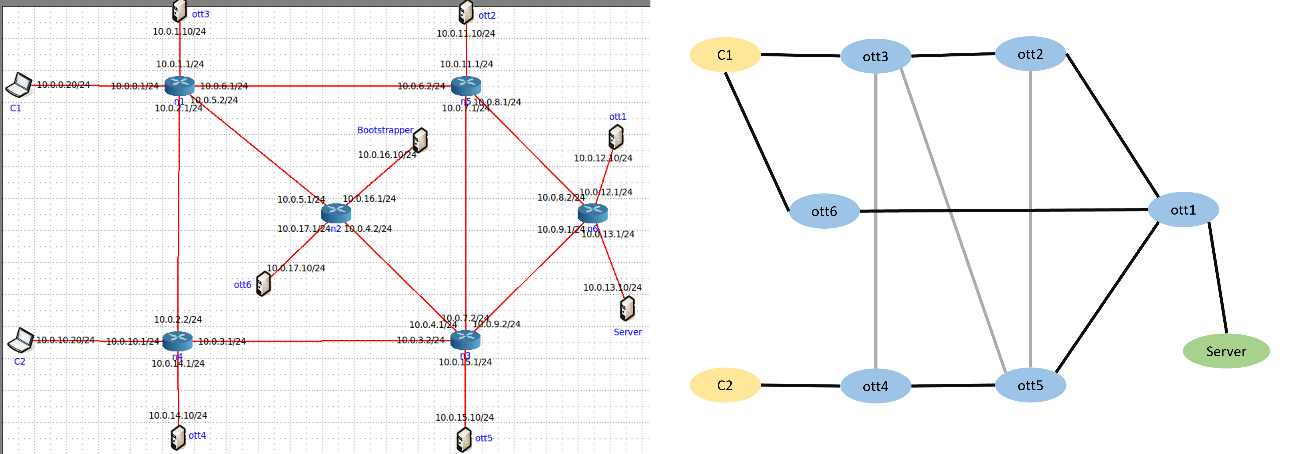


Figura 8: Topologia criada para testes (versão UNDERLAY & OVERLAY)

Com a utilização desta rede realizaram-se vários testes, utilizando o conjunto total da rede ou apenas partes da mesma. Alguns dos testes realizados estão apresentados nos subcapítulos seguintes.

## Funcionamento esperado

Para testar o correto funcionamento do serviço, executou-se os servidores de modo a verificar, se o cliente recebia a transmissão do vídeo (de notar que o vídeo é uma *streaming* em tempo real, o que implica, que se um cliente entrar depois de outro, ambos devem visualizar o mesmo conteúdo), bem como as respetivas tabelas de rotas e melhores caminhos para o servidor.

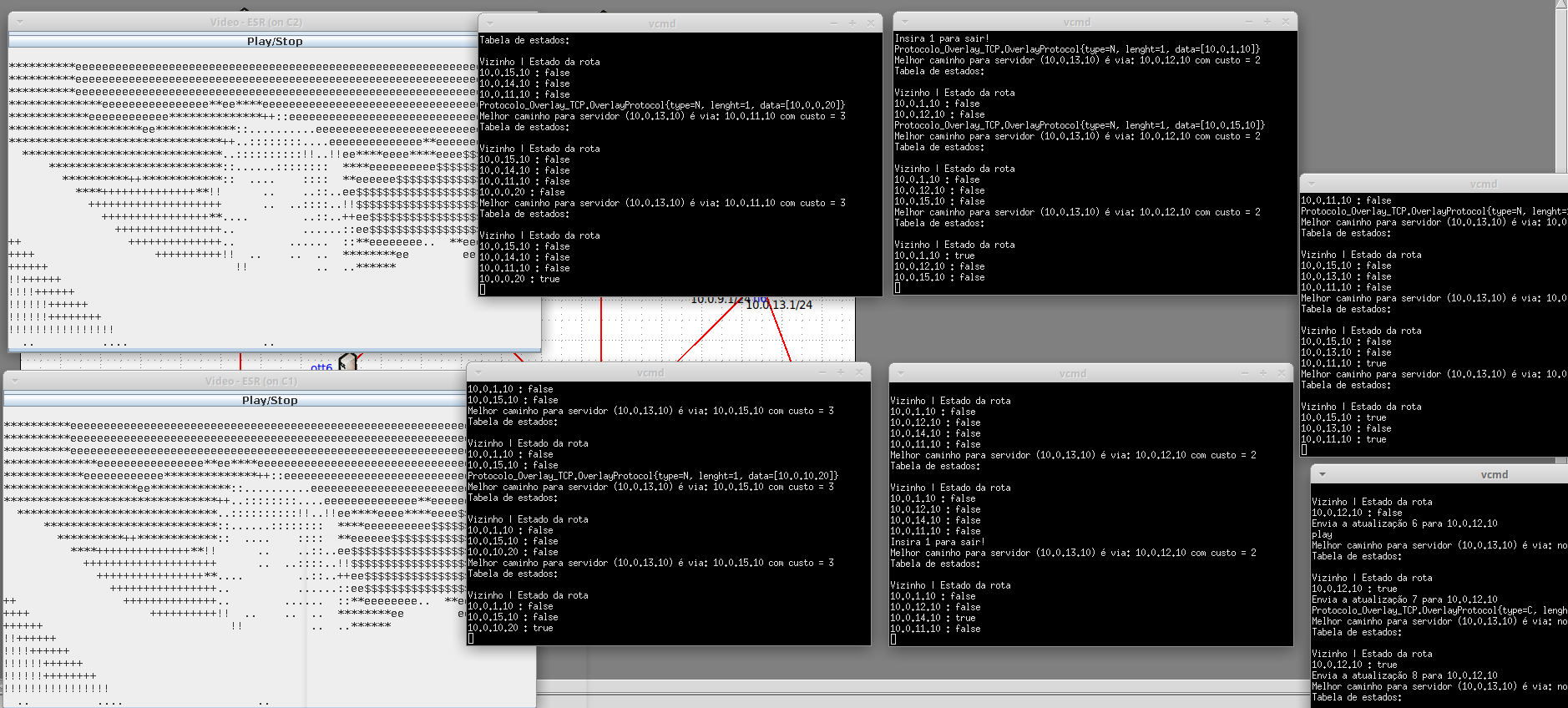


Figura 9: Teste de funcionamento

Posteriormente, recorreu-se ao *wireshark* para verificar o fluxo de mensagens tantos nos servidores envolvidos, como noutros servidores (se não pertencem à rota não devem receber mensagens de *streaming*).

Uma imagem com mesa

Descrição gerada automaticamente

Figura 10: Teste de funcionamento com o wireshark

## Alteração do melhor caminho durante execução

Uma vez que, a rede depois de estar operacional, não deve parar o seu funcionamento para evoluir, deve ser possível adicionar novos servidores (*ott*) durante o seu funcionamento. Contudo, no caso desses novos servidores melhorarem as rotas para o servidor de *streaming* a rede deve evoluir, alterando as suas rotas. Assim, testou-se o caso em que se adiciona um novo servidor após existirem nodos ativos.

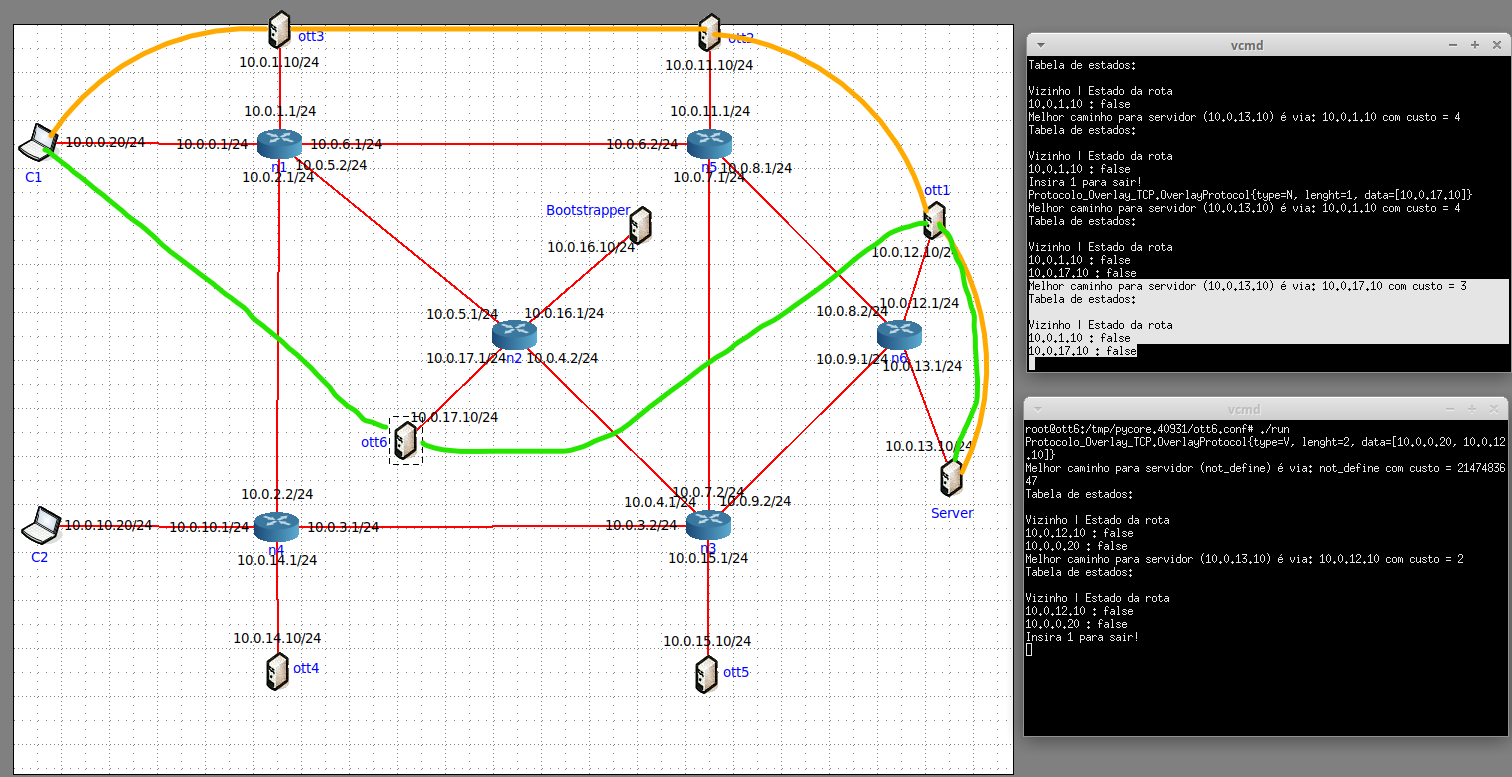


Figura 11: Correção da topologia em runtime

Posteriormente, verificou-se o caso desse novo nodo influenciar o melhor caminho de um cliente que está a receber conteúdo por outro servidor.

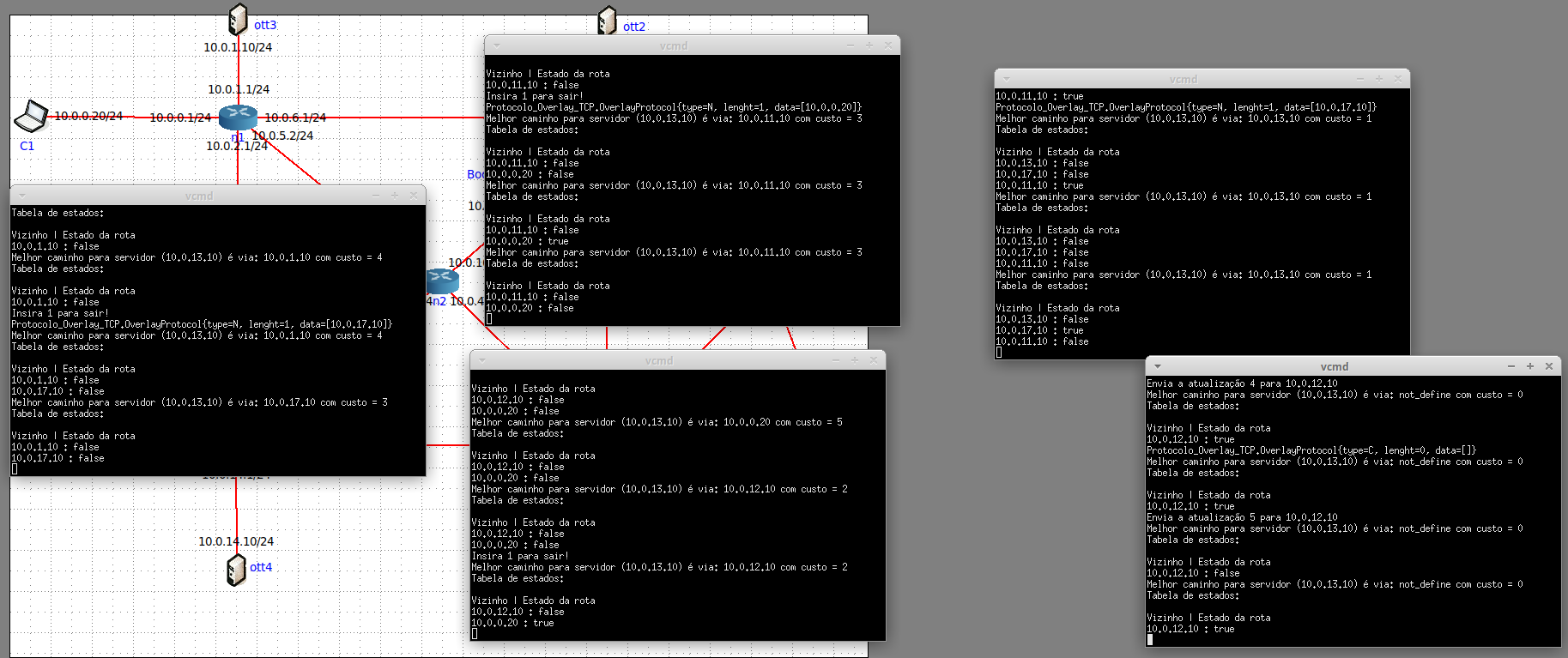


Figura 12: Correção da topologia em runtime e em transmissão

Assim, verifica-se que a rota é modificada, bem como o serviço passa a fluir pela nova rota, desligando completamente a rota antiga (pode ser desligada até ao servidor – no caso em que nenhum nodo dessa rota está a servir um cliente – ou até um nodo intermediário).

## Tolerância a falhas da rede

Ao contrário do caso anterior, testou-se a capacidade da rede recuperar à desconexão de um nodo.

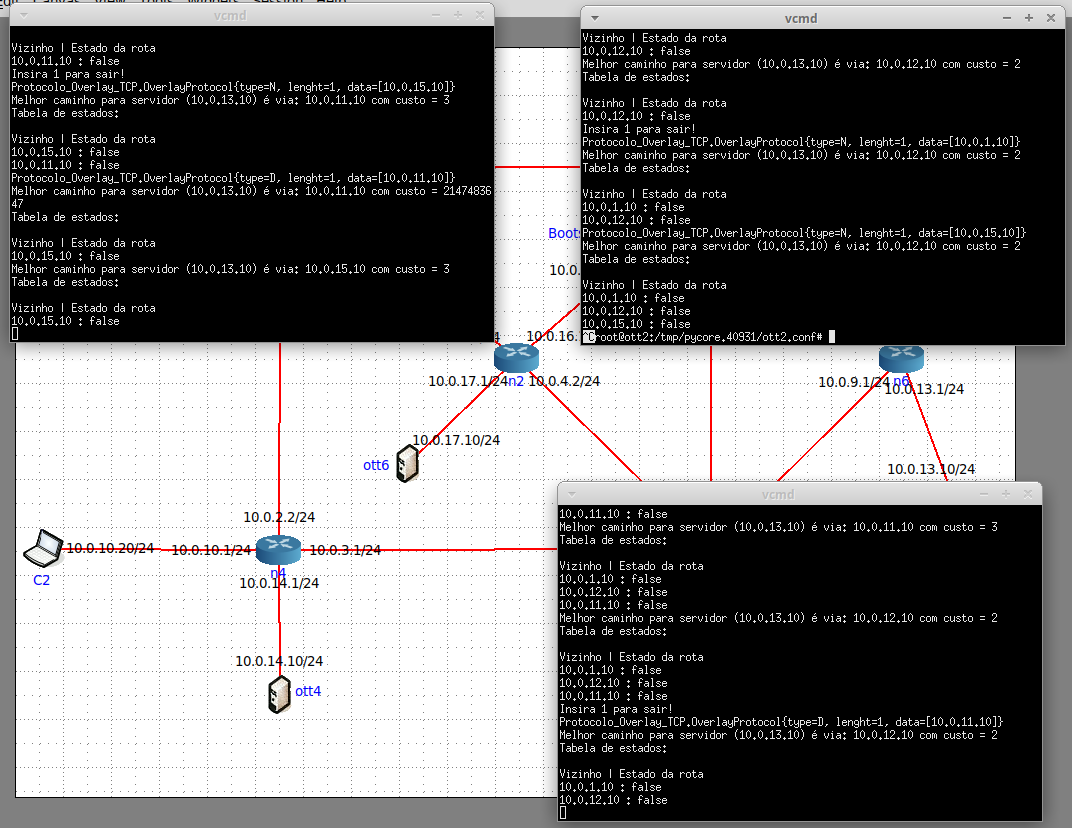


Figura 13: Correção da topologia em runtime (falhas)

Verificou-se, que a melhor rota para o servidor de *streaming* foi alterada em todos os nodos impactados, bem como as suas listas de vizinhos. Por fim, verificou-se o impacto da desconexão de um servidor que estava a retransmitir para um cliente.

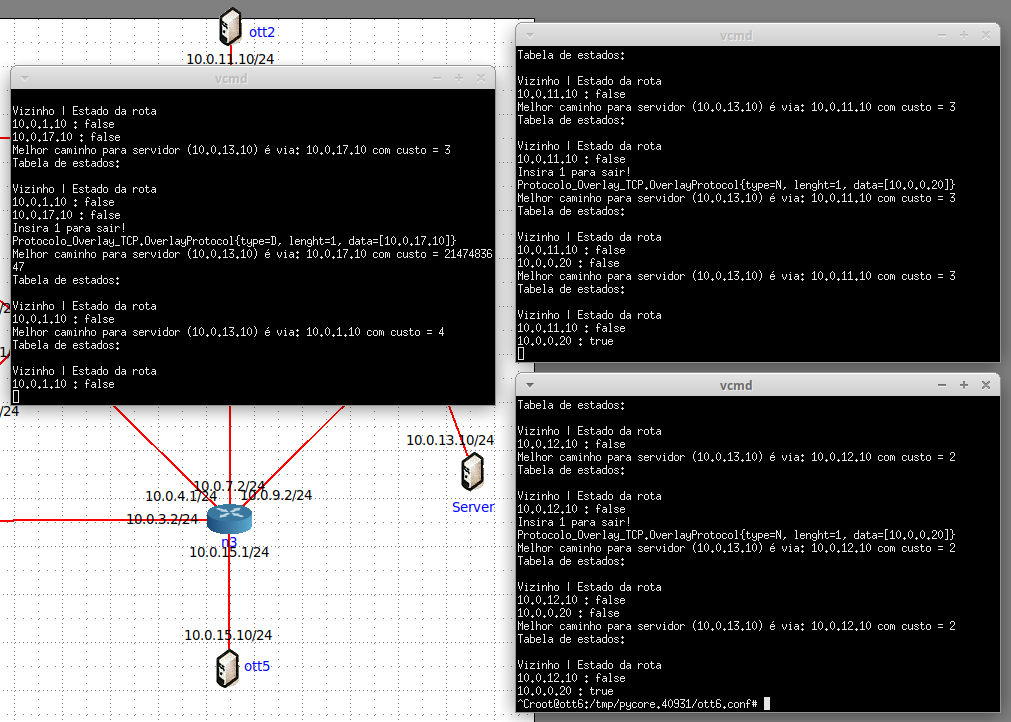


Figura 14: Correção da topologia em runtime e em transmissão (falhas)

Observou-se que, apesar da perda de algumas partes do *frame,* o serviço não se interrompeu, utilizando outra rota para obter os seus dados.

## Stop do serviço (lado do cliente)

Por fim, testou-se a capacidade de um cliente parar e recomeçar a transmissão. Assim, sempre que um cliente pretender, pode parar a sua transmissão que não influencia na transmissão de outros clientes que partilhem nodos intermediários. Posteriormente, o cliente pode retomar a *stream*, perdendo apenas as partes em que o vídeo esteve parado.

Uma imagem com texto

Descrição gerada automaticamente

Figura 15: Teste do stop/play

# Conclusões e trabalho futuro

A realização deste trabalho permitiu criar uma rede *overlay*, que através dos seus vários servidores presta um serviço de *streaming* em tempo real. Com a utilização desta rede *overlay* pretende-se diminuir o tráfego da rede, uma vez que se duplica a menor quantidade possível de pacotes. Por fim, também se pretende aumentar a eficiência no transporte de pacotes, uma vez que se calcula a melhor rota entre cada servidor e o servidor de *streaming*.

Assim, no trabalho realizado, considerou-se como escolha de melhor rota o número de saltos, o que não é a melhor medida para determinar os melhores custos das rotas. Desta forma, em trabalho futuro, deve-se alterar esta métrica para uma outra mais eficaz, como por exemplo o *round trip time*.

Considerou-se também o *streaming* de vídeos em formato ASCII em que se descarta a utilização de cores ou o som da *stream*. Posteriormente este apeto poderia ser evoluído, ao acrescentar às mensagens de *streaming* informação acerca destes valores (cor dos caracteres e som do vídeo), e uma vez que já existe uma interface gráfica, a sua implementação na solução é facilitada.

Um outro aspeto que requer um trabalho mais aprofundado é a recuperação de falhas, uma vez que se um nodo crítico (exemplo: ott1 da figura 8) deixar de responder, torna-se impossível continuar o serviço. Desta forma, poderão ser implementados mecanismos, por exemplo, no servidor de *bootstrapper* que recalculem o grafo da topologia de modo a impedir que nodos (cliente ou servidor de *stream*) fiquem isolados. Uma outra estratégia possível para mitigar este problema, é a comunicação direta com o servidor (uma vez que todos os nodos conhecem o endereço *ip* do servidor), sempre que não exista uma rota disponível para o mesmo.

Por fim, deverão ser implementados mecanismos que garantam a segurança no transporte de dados (impossibilitar a verificação/alteração do conteúdo das mensagens trocadas) e a segurança na autenticidade de novos *ott’s* e clientes (garantir que ninguém não autorizada consegue fazer-se passar-se por um servidor autorizado, tendo assim acesso ao conteúdo).