

# Serviço Over the Top para entrega de multimédia Engenharia de Serviços em Rede

Diogo Rebelo<sup>[pg50327]</sup>, João Caldas<sup>[pg50494]</sup>, and Teresa Fortes<sup>[pg50770]</sup>

<sup>1</sup> Minho's University, Department of Informatics, 4710-057 Braga, Portugal

<sup>2</sup> email: {pg50327,pg50494,pg50770}@alunos.uminho.pt

**Abstract.** A utilização de topologias Over the Top (OTT) é um modelo de distribuição de conteúdo em que o provedor de conteúdo é entregue diretamente ao usuário final através da internet, sem a necessidade de intermediários ou redes de distribuição de media tradicionais. O Protocolo RTP é amplamente utilizado em aplicações OTT de streaming multicast, pois permite a transmissão de áudio e vídeo em tempo real de maneira eficiente e confiável através da internet. A combinação do RTP com o RTCP também é importante para garantir a qualidade do streaming em aplicações OTT. Assim sendo, o programa desenvolvido permite a entrega de conteúdos de áudio, vídeo e texto através de um servidor e um ou mais clientes, utilizando o protocolo de Streaming com Multicast aplicacional. A rede de overlay aplicacional criada otimiza a entrega dos conteúdos, garantindo a eficiência e reduzindo o atraso e a largura de banda necessária.

**Keywords:** Over The Top · Streaming · Overlay · UDP · Sockets · Threads · Nodes · Server · Client

## 1 Introdução

Neste projeto de Engenharia de Serviços em Rede, desenvolvemos um programa para serviços de *Streaming* com *Multicast aplicacional*, a partir de um servidor e para um ou mais clientes. Assim, pretende-se conceber um protótipo que envolve entrega de vários tipos de tecnologias, nomeadamente, áudio, vídeo e texto. Este serviço apresenta vários requisitos de tempo real, que se relacionam com as várias etapas, desde a preparação da topologia, até à entrega propriamente dita. Importa referir que foi necessária a formação de uma rede de *overlay* aplicacional, cuja criação e manutenção deve estar otimizada para a missão de entregar os conteúdos de forma mais eficiente, com o menor atraso e a largura de banda necessária. Neste âmbito, existem objetivos primordiais que se devem ver concretizados no final do presente trabalho, nomeadamente:

- Desenvolvimento de um protocolo adequado, que defina as regras de comunicação entre as várias entidades da topologia;
- Desenvolvimento de entidades aplicacionais distintas, mas que se relacionam entre si e que apresentam uma certa dependência, das quais: cada Nodo, cada Servidor e cada Cliente;
- Compreensão dos diferentes protocolos de comunicação, a nível de Transporte, principalmente TCP e UDP, percebendo as suas vantagens e desvantagens de utilização;
- Lecionar conteúdos que envolvem o nível aplicacional, e compreender de modo mais aprofundado a gestão que existe com outras camadas da pilha protocolar;
- Aplicação prática de estratégias de roteamento - sempre em sintonia com o protocolo especificado- e de (des)serialização - nas mensagens trocadas entre entidades;

## 2 Arquitetura da solução

Para desenvolvimento da solução, optamos pela utilização da linguagem *Python*, pois mostrou-se uma linguagem familiar aos membros do grupo e com bom suporte para realizar *threading*.

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 bootstrapper 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, juntamente com o 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.

Assim, a solução arquitetural concebida pelo grupo apresenta-se na figura seguinte, sendo que a especificação e justificação de cada protocolo de transporte utilizado surge de seguida.

Em relação à figura seguinte, observa-se 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, como tal, possuem as suas próprias tabelas de encaminhamento. Então, consideramos a existência de um único servidor que será encarregue de distribuir a informação para as diferentes entidades, e ainda efetuar o *streaming* de um dado conteúdo. Os clientes são nodos que irão pedir e 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, representados pela entidade *oNode*, têm como objetivo encaminhar as mensagens que recebem do nodo bootstrapper ou do nodo cliente.

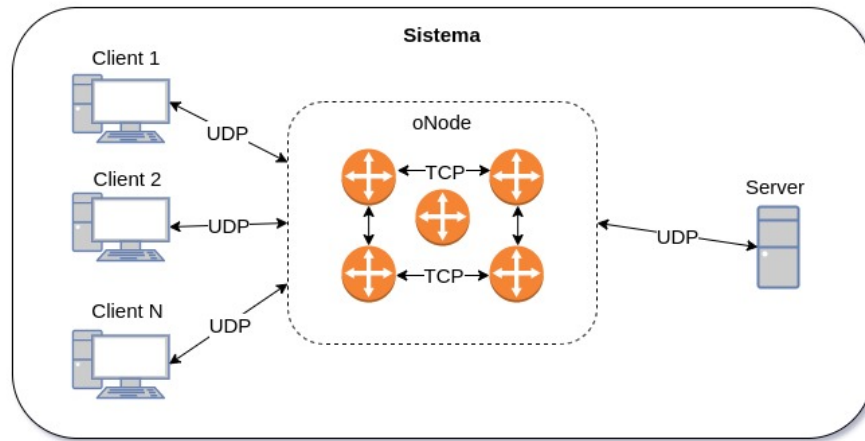


Fig. 1. Arquitetura da solução

### 3 Especificação dos protocolos

Na figura anterior, faz-se referência a dois tipos de protocolos diferentes, TCP e UDP. Tendo em conta as características de cada tipo, consideramos utilizá-los de acordo com o processo em causa: efetuamos comunicações entre nodos utilizando o protocolo de transporte TCP, já o serviço de streaming propriamente dito recorre ao protocolo UDP. Optamos pelo TCP para as várias comunicações por ser um protocolo confiável no que diz respeito à entrega de informação, ou seja, à garantia de que as mensagens trocadas por cada nodo, indispensáveis para o funcionamento da rede, chegam ao destino pretendido. Em contra partida, recorreremos ao UDP, para entrega de conteúdos de streaming, pela sua rapidez no envio de pacotes, quando comparado com o protocolo TCP. É sabido que, ao utilizar UDP, pode haver perda de pacotes, todavia, estas perdas não são muito significativas, já que perder um frame ou outro não causa muito impacto.

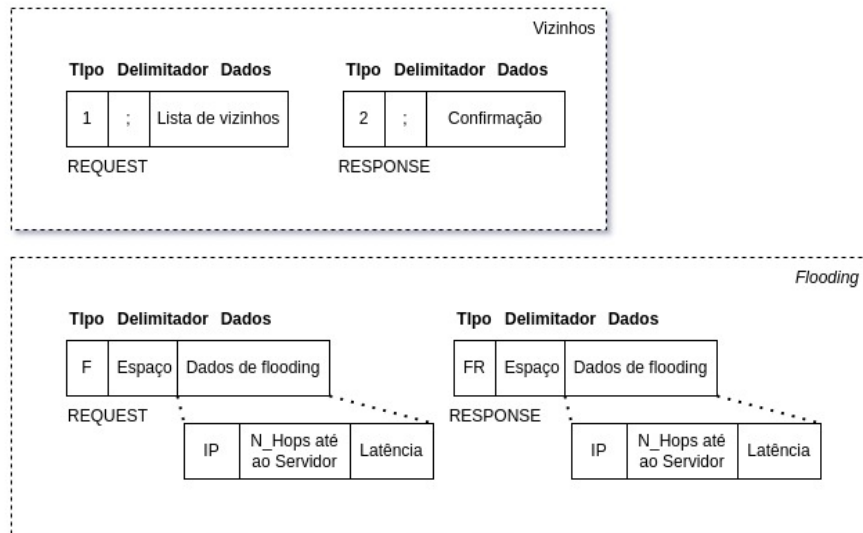
De acordo com os requisitos estabelecidos no enunciado do projeto, criamos protocolos aplicacionais de controlo e um protocolo de *streaming*, pelas razões que também definimos anteriormente. Para as mensagens de comunicação entre os diferentes nodos, não criamos uma classe especificamente para isso, ou seja, recorreremos à criação de strings, com certos delimitadores, que depois eram codificadas e enviadas. Já para as mensagens que se relacionam com streaming, recorreremos às classes *RTPpacket* e *Streaming*. Então, podemos associar os protocolos aplicacionais de controlo a protocolos que gerem processos de: criação da camada de *overlay*, criação da árvore de entidades de distribuição de conteúdos, monitorização de nodos, atualização de parâmetros de encaminhamento, etc.

### 3.1 Formato das mensagens protocolares

**Mensagens de Controlo Aplicacional** De modo a conseguir controlar o fluxo das mensagens de controlo e de dados que na topologia seguimos uma estrutura de mensagens protocolares que se apresenta de seguida. Estas mensagens não apresentam muita complexidade, apenas têm duas componentes principais, o “Tipo” - que serve para identificar o tipo da mensagem protocolar - e os “Dados” - que guardam o conteúdo da própria mensagem. Estes dois campos surgem separados por um delimitador.

- tipo 1 : caso seja uma mensagem de overlay, ou seja, quando um nodo entra na topologia overlay, alerta o bootstrapper que quer saber quais são os seus vizinhos;
- tipo F : caso seja um pedido de flooding, ou seja, quando estamos a povoar as tabelas de encaminhamento e de custos de cada nodo; item tipo FR : caso seja uma resposta de flooding, ou seja, uma resposta do nodo com a sua tabela (com o seu ip, número de saltos até ao servidor e latência);
- tipo:

Como forma de representação dos vários tipos de campos e hipóteses, observe-se a seguinte figura:



**Fig. 2.** Especificação das mensagens de controlo aplicacional.

**Mensagens de Streaming: RTPpacket** Para a transmissão de conteúdos de streaming pela topologia, usou-se o RTPpacket, que funcionará como um Real-time Transport Protocol, algo muito usado hoje em dia em aplicações de tempo real sobre IP. Esta classe foi fornecida pela equipa docente de forma a apoiar a realização do trabalho prático, contudo segue-se uma explicação dos vários campos das mensagens deste protocolo:

Version	Padding	Extension
Marker	Payload Type	Sequence Number
Ssrc	Header	Payload Size
Header Size		

Table 1. Especificação das mensagens de RTP.

## 4 Implementação

Esta secção contém o conjunto de classes e as funcionalidades que elas vêm implementar. Assim sendo, surgem listadas abaixo as classes principais, a sua constituição e a forma como contribuíram para ver cada funcionalidade implementada.

- Existem algumas diretorias que dizem respeito a ficheiros de configuração, ou seja, ficheiros que contém os respetivos endereços ip de cada nodo e os seus vizinhos. A estrutura deste ficheiro segue o seguinte formato:

```
10.0.1.1|10.0.1.2
10.0.0.2|10.0.0.1 10.0.2.10 10.0.1.1
10.0.2.10|10.0.2.1
10.0.0.1|10.0.0.2
```

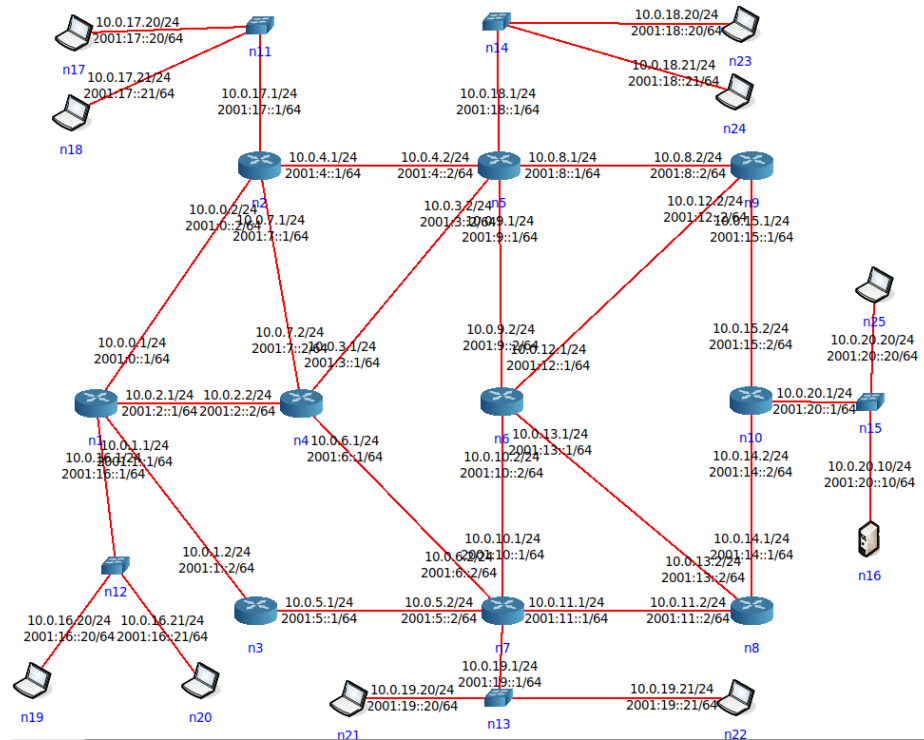
Reparamos que para cada ip, se usa como delimitador o careter ‘|’ para separar dos vizinhos, sendo que estes surgem separados por um outro careter, um espaço (‘ ’). Para além disso, há diretorias que contém os ficheiros a ser trocados para streaming e cada topologia de cada cenário utilizado como teste;

- A classe **oNode** visa tratar de tudo o que se relaciona com cada nodo, seja em termos de troca de mensagens entre nodos, seja ao nível do flood e, possivelmente, em termos de monitorização dos seus vizinhos;
- A classe **Server** procura tratar dos pedidos que são efetuados pelas entidades cliente e nodo. Independentemente do tipo de pedido que seja, o servidor serve como entidade que está à escuta e que, quando requisitado, responde com a informação requerida, podendo ser streaming, ou informações de rotas, etc;

- A classe `RTPpacket` foi formada pela equipa docente e alterada pelo grupo para atender aos objetivos e à implementação solucionada pelo grupo e contém a formação de cada mensagem e métodos para a obtenção de informações relevantes dos pacotes recebidos/enviados. Do mesmo modo, a classe `VideoStream` também fornecida pela equipa docente, auxilia o envio de frames do ficheiro pedido. Neste âmbito são utilizadas algumas bibliotecas do python para dar uma interface gráfica mais intuitiva ao cliente.

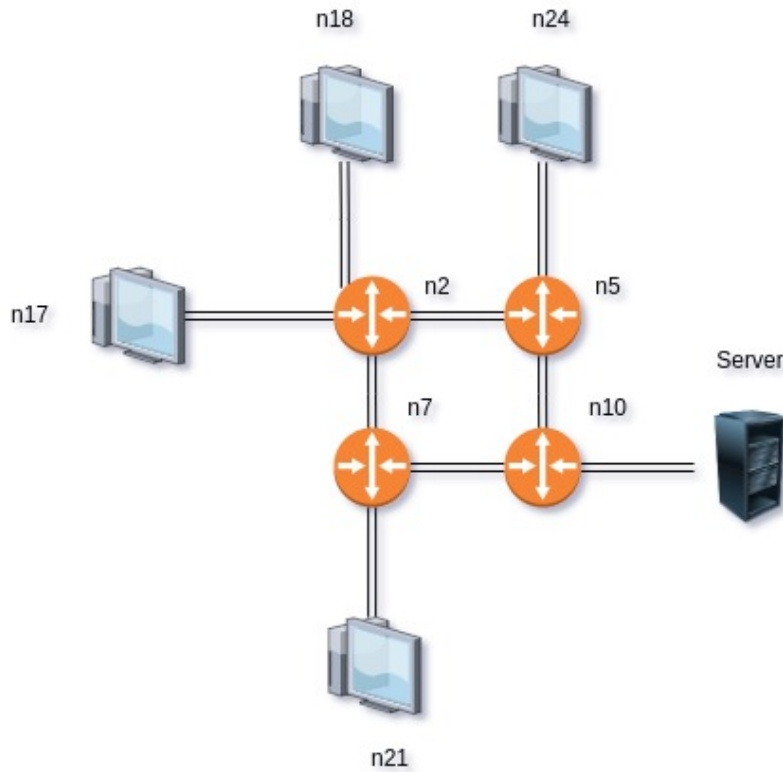
## 5 Testes e resultados

Esta secção apresenta os resultados de alguns testes realizados na topologia do core de uma rede, a fim de verificar se todas as funcionalidades foram implementadas corretamente. A figura abaixo mostra a topologia underlay que foi utilizada nestes testes. Esta topologia utilizada é exatamente igual à do próprio enunciado.



**Fig. 3.** Topologia de underlay utilizada.

Passamos a executar os nodos que consideramos indispensáveis para o funcionamento da rede, ou seja, os nodos fundamentais para iniciar a execução do sistema na rede. Então, obtemos a rede de overlay que se segue, logo após o flood ser efetuado.



**Fig. 4.** Topologia de overlay a utilizar em cenários de teste, é apenas um exemplo.

O grupo possui numa diretoria específica outros cenários que também foram criados, de acordo com a sugestão do enunciado.

### 5.1 Cenário de Teste (Novo)

Em relação a esta parte, seguem-se o que o grupo conseguiu implementar e alguns detalhes da implementação. O cenário utilizado foi o cenário (2) seguinte, já que possibilita a perceção total sobre as funcionalidades do programa.



O fluxo pode ser descrito da seguinte forma: primeiro, o servidor é iniciado, passando-lhe a flag `-s`, para dizer que se trata do servidor, e o número do cenário em questão, para que ele perceba qual ficheiro de configuração deve utilizar para ter conhecimento de topologia. Este servidor fica, então, à escuta de conexões por parte de novos nodos. Inicia-se, de seguida um nodo adjacente, passando o nome do ficheiro de cenário a utilizar (deve ser o mesmo que o servidor utilizou), o endereço ip do bootstrapper (que no nosso caso é o servidor), e a respetiva porta na qual o servidor está. Este nodo, assim que é iniciado, envia um pedido de `VIZINHOS` ao servidor, o qual responde com uma mensagem do tipo `VIZINHOS_ANN`, a confirmar a receção do request. O Nodo recebe os seus vizinhos e guarda essa informação, começando a realização de flooding. O flooding foi implementado para que seja possível conhecer a topologia e saber qual é a maneira de contactar o servidor a partir de qualquer nodo. A nossa implementação do flooding baseia-se no envio de mensagens (strings) de um nodo para todos os seus vizinhos na topologia com a mensagem "`F; < ip >`", ou seja, a mandar um pedido dizendo que está ativo e se quer conectar ao servidor. Os vizinhos que estão acordados por sua vez respondem com outra mensagem ("`F; R; < IP >, NR.SALTOS, LATENCIA`"), que mostra ser uma resposta com o seu IP, o número de saltos até ao servidor e um timestamp. Para saber qual o caminho mais rápido, guardamos o nodo para onde deve ser o próximo salto na estrutura *routing*, que guarda o ip desse próximo nodo, o número de saltos e a latência. Caso haja um caminho mais rápido, esta estrutura é atualizada com o respetivo nodo vizinho desse novo caminho.

Este fluxo de ideia, pode ser observado pelas imagens seguintes que mostram a conexão sucessiva dos nodos `n3`, `n6`, `n4`, `n1`:

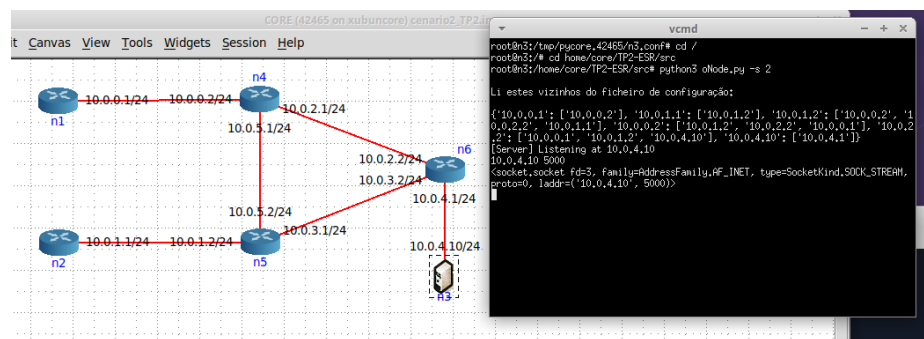
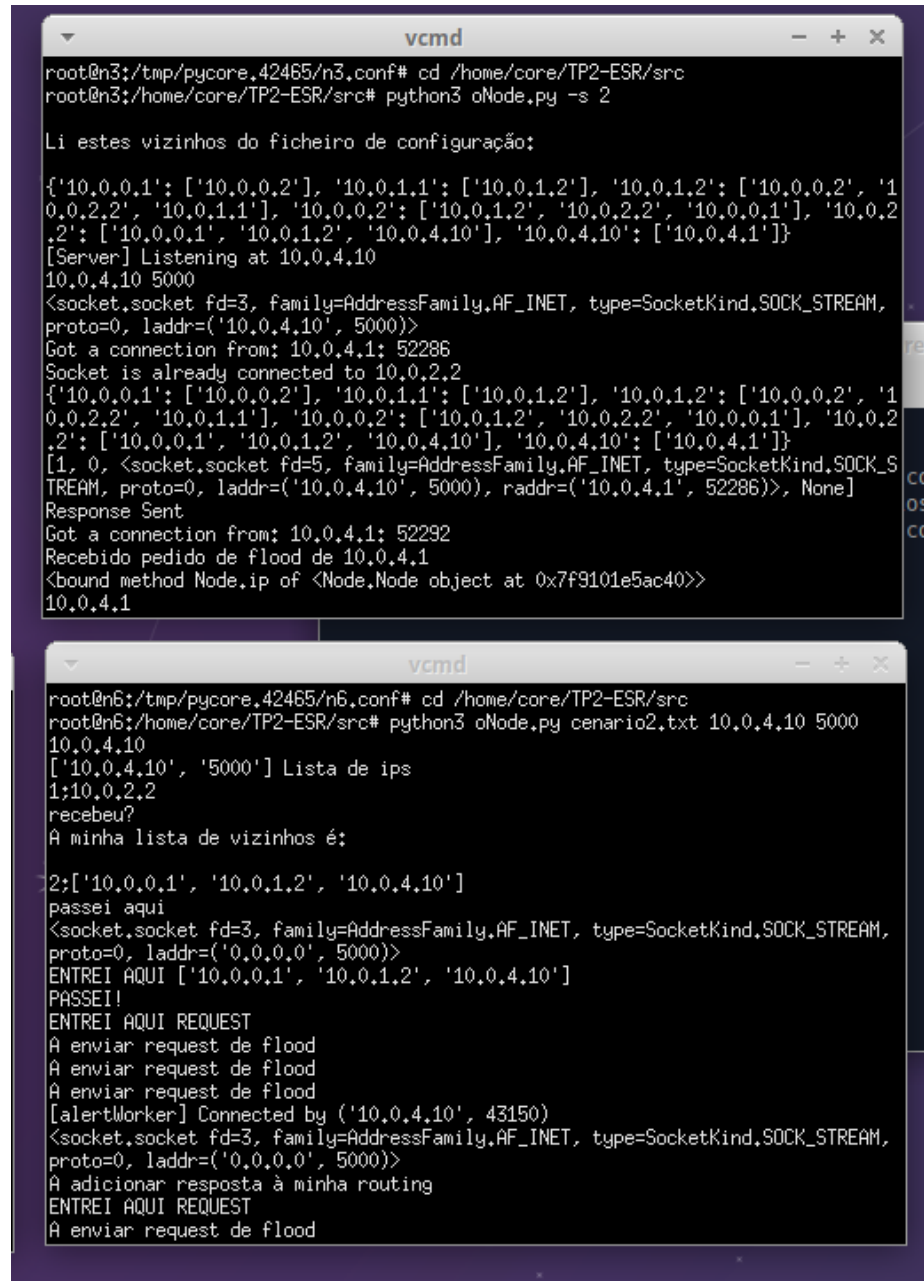


Fig. 5. Início do servidor à escuta.



The figure shows two terminal windows. The top window, titled 'vcmd', shows a user running a Python script 'oNode.py' with the argument '-s 2'. The script outputs a list of neighbors for the IP '10.0.4.10' and then listens for connections. It receives a connection from '10.0.4.1: 52286' and sends a response. The bottom window, also titled 'vcmd', shows the user running 'oNode.py' with the argument 'cenario2.txt 10.0.4.10 5000'. The script outputs a list of IPs, receives a flood request, and then sends a flood request to the neighbors.

```

vcmd
root@n3:/tmp/pycore.42465/n3.conf# cd /home/core/TP2-ESR/src
root@n3:/home/core/TP2-ESR/src# python3 oNode.py -s 2

Li estes vizinhos do ficheiro de configuração:

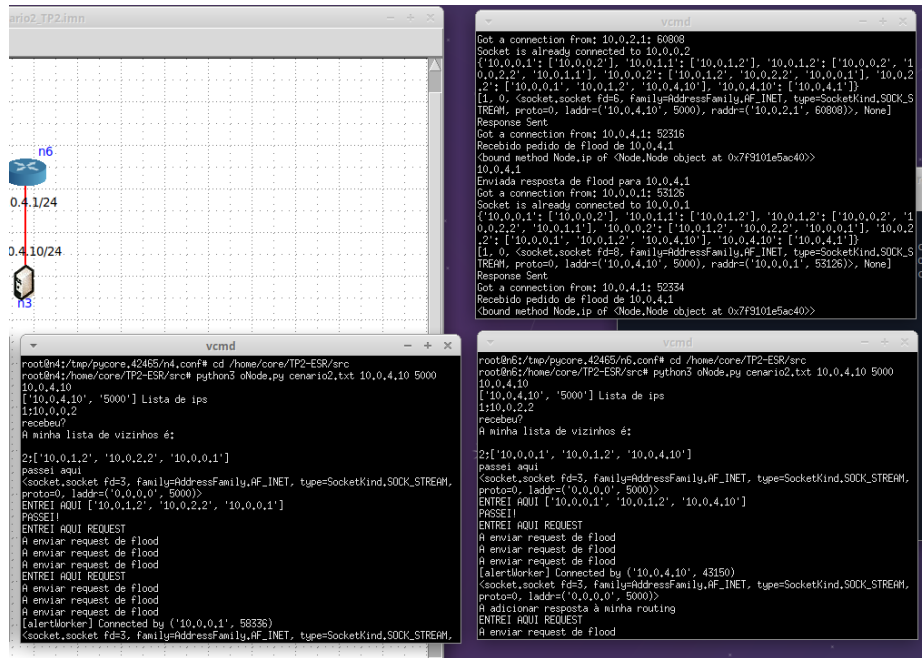
{'10.0.0.1': ['10.0.0.2'], '10.0.1.1': ['10.0.1.2'], '10.0.1.2': ['10.0.0.2', '10.0.2.2'], '10.0.1.1': ['10.0.0.2'], '10.0.2.2': ['10.0.0.1', '10.0.1.2'], '10.0.4.10': ['10.0.4.1']}
[Server] Listening at 10.0.4.10
10.0.4.10 5000
<socket.socket fd=3, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('10.0.4.10', 5000)>
Got a connection from: 10.0.4.1: 52286
Socket is already connected to 10.0.2.2
{'10.0.0.1': ['10.0.0.2'], '10.0.1.1': ['10.0.1.2'], '10.0.1.2': ['10.0.0.2', '10.0.2.2'], '10.0.1.1': ['10.0.0.2'], '10.0.2.2': ['10.0.0.1', '10.0.1.2'], '10.0.4.10': ['10.0.4.1']}
[1, 0, <socket.socket fd=5, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('10.0.4.10', 5000), raddr=('10.0.4.1', 52286)>, None]
Response Sent
Got a connection from: 10.0.4.1: 52292
Recebido pedido de flood de 10.0.4.1
<bound method Node.ip of <Node.Node object at 0x7f9101e5ac40>>
10.0.4.1

vcmd
root@n6:/tmp/pycore.42465/n6.conf# cd /home/core/TP2-ESR/src
root@n6:/home/core/TP2-ESR/src# python3 oNode.py cenario2.txt 10.0.4.10 5000
10.0.4.10
['10.0.4.10', '5000'] Lista de ips
1:10.0.2.2
recebeu?
A minha lista de vizinhos é:

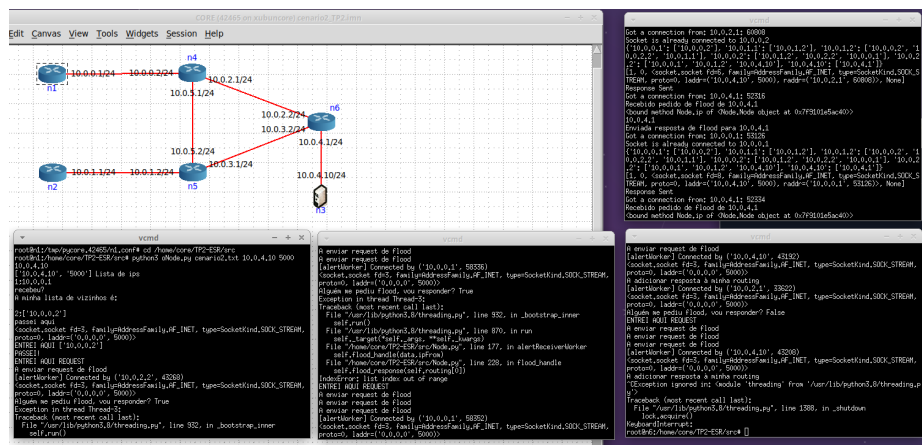
2:['10.0.0.1', '10.0.1.2', '10.0.4.10']
passei aqui
<socket.socket fd=3, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('10.0.0.0', 5000)>
ENTREI AQUI ['10.0.0.1', '10.0.1.2', '10.0.4.10']
PASSEI!
ENTREI AQUI REQUEST
A enviar request de flood
A enviar request de flood
A enviar request de flood
[alertWorker] Connected by ('10.0.4.10', 43150)
<socket.socket fd=3, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('10.0.0.0', 5000)>
A adicionar resposta à minha routing
ENTREI AQUI REQUEST
A enviar request de flood

```

**Fig. 6.** Servidor recebe pedido de vizinhos do novo novo, responde-lhe, este recebe a resposta e efetua o flooding.



**Fig. 7.** Junta-se mais um nodo, o servidor responde com os vizinhos, o novo novo efetua flooding, através da conexão com o seu adjacente (nodo que se ligou anteriormente). O Servidor recebe ambos os pedidos de flooding e responde.



**Fig. 8.** Junta-se mais um nodo e ocorre o processo anterior. As tabelas de routing vão sendo atualizadas, pelo método descrito na secção acima.

## 6 Conclusões e trabalho futuro

As principais tarefas foram realizadas, ou seja, fez-se a implementação de todas as funcionalidades básicas para o funcionamento do sistema proposto, incluindo: o flood, a criação do overlay e a stream de um vídeo. Também foram implementados os requisitos e mensagens de controle necessários para manter a rede ativa e atualizada. Em termos de melhorias, com mais tempo, seria possível implementar um sistema que permitiria ao cliente selecionar o conteúdo a ser assistido, desenvolver a tolerância a falhas e melhorar o desempenho em termos de escalabilidade. Em geral, o projeto foi bem-sucedido e os requisitos essenciais foram cumpridos.