TP2: Serviço *Over the Top* para Entrega de Multimédia

Danilo Oliveira[pg46528], Joana Alves[pg50457], and Vicente Moreira[pg50799]

Universidade do Minho
Departamento de Informática
Engenharia de Serviços em Rede

Resumo Neste trabalho, no âmbito da unidade curricular de Engenharia de Serviços em Rede, pretende-se implementar um sistema de difusão e entrega de vídeo e áudio, com requisitos de tempo real, a partir de um servidor de conteúdos para um conjunto de clientes. Para tal, serão criados e utilizados vários nós na rede IP, de modo a formar uma nova rede de nível mais elevado, a rede de *overlay*. Esta rede de *overlay* tem o objetivo de otimizar a entrega de conteúdo da maneira mais eficiente, com o menor atraso e largura de banda necessária de modo a proporcionar a melhor qualidade de experiência ao utilizador.

Keywords: UDP · Overlay · OVR · Nodo · Servidor-Cliente · RTP

1 Introdução

A Internet como hoje percecionamos, permite uma variedade de serviços de entrega de conteúdos, no entanto, na sua conceção inicial, esta tinha como seu principal objetivo o envio de ficheiros e informação simples de tamanho reduzido, através do encaminhamento de pequenos pacotes pela rede. No entanto, conforme os ficheiros multimédia de maiores dimensões se tornaram comuns e populares, assim como também o conceito de *streaming*, ou seja, a entrega "em tempo real" deste tipo de ficheiros multimédia, a implementação da entrega deste tipo de conteúdo enfrentou vários problemas devido à dimensão dos dados a serem transmitidos, assim como a complexidade necessária para criar sistemas de redundância e de baixo atraso na entrega.

Deste modo, foi necessário o desenvolvimento de protocolos de *streaming* eficientes e fiáveis, o que acarreta vários desafios no seu desenvolvimento ao longo de todas as suas fases, como um bom planeamento, estruturação do mesmo e implementação sólida. No entanto, com o desenvolvimento de tecnologias como fibra ótica e o aumento da largura de banda disponível, os serviços de *streaming* cresceram exponencialmente, sendo atualmente um dos serviços mais comuns e utilizados no mundo, havendo a necessidade de criar protocolos de *streaming* ainda mais eficientes.

Assim, sendo este o objetivo do projeto, será desenvolvido um protocolo de *streaming* capaz de satisfazer todos estes requisitos. Ao longo deste relatório, vão ser apresentados os vários passos e caminhos seguidos pelo grupo para o desenvolvimento deste projeto, assim como a arquitetura e comportamento da solução final.

2 Arquitetura da Solução

2.1 Estratégias adotadas

Protocolo de Transporte: Para a implementação deste protocolo de *streaming*, optamos por utilizar **UDP** como base para as comunicações entre todas as entidades visto que este, não sendo orientado à conexão, tem maior facilidade de uso e permite a transmissão de dados a débitos mais elevados.

Etapa 1 - Construção da Topologia *Overlay*: Para esta fase, optamos por implementar um ***BootStrapper***, sendo este uma componente/entidade adicional de um dos nós do *overlay*, a ser definido por topologia. Esta estratégia é relativamente simples de implementar e permite centralizar a informação da topologia num só nodo, evitando potenciais dificuldades e erros no iniciar dos vários nós do *Overlay*.

Etapa 2 - Serviço de *Streaming*: Utilizando por base o código fornecido pelos docentes, implementamos um servidor e cliente capazes de codificar e descodificar uma *stream* de vídeo. Para o Cliente decidimos criar uma GUI mais simplista, utilizando o terminal para selecionar as opções de um menu, abrindo apenas uma janela para apresentar a *stream*.

Etapa 3 - Monitorização da Rede *Overlay*: Para este projeto decidimos que a monitorização da rede *Overlay* poderá ocorrer tanto ao nível do Servidor como ao nível de vizinhança dos nós. Os nodos também são capazes de indicar, a pedido do utilizador, a velocidade do tráfego presente no momento.

Etapa 4 - Construção de Rotas para a Entrega de Dados: Para a construção de rotas, decidimos que estas deveriam ser definidas pelo servidor, através de *Floodings* na rede *Overlay*, com o objetivo de "anunciar" o servidor, seguido de pacotes de teste de velocidade periódicos, que definem o melhor servidor para a entrega de dados.

Etapa 5 - Ativação e Teste do Servidor Alternativo: Para suportar servidores alternativos, os nodos do *Overlay*, contêm uma lista de servidores e as suas fontes, assim como a métrica de atraso, no qual o nodo deverá escolher o melhor servidor disponível.

Etapa Opcional - Definição do Método de Recuperação de Falhas: Para a recuperação de falhas, os nós testam periodicamente a sua conexão ao vizinho fonte, e, caso esta possua um atraso elevado ou deixe de responder, o nó altera o servidor ou a sua fonte, modificando a árvore de difusão.

2.2 Arquitetura do Sistema

Nesta secção será apresentada a arquitetura geral do sistema desenvolvido, assim como todas as suas subcomponentes, acompanhado de uma breve descrição do seu papel no funcionamento do sistema.

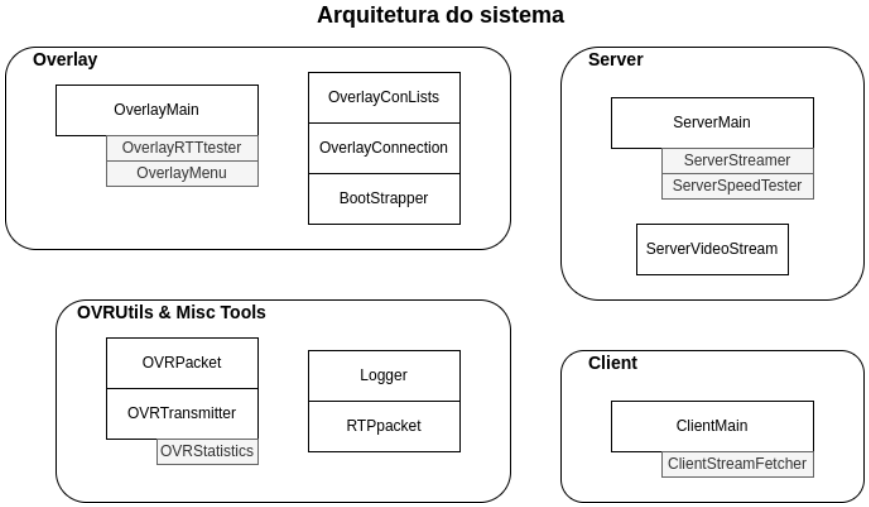
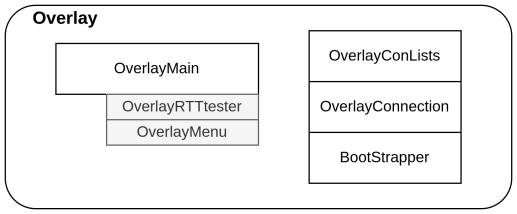


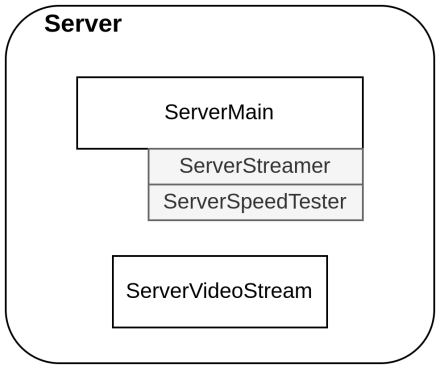
Fig. 1. Arquitetura do sistema

Overlay



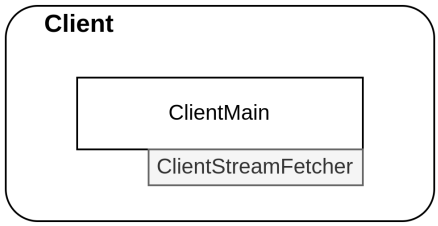
- **OverlayMain:** Leitura de argumentos, *setup* do *Bootstrapper* ou contacto com este e inicialização de *Threads*. Contém o *loop* de Leitura, Processamento e Resposta de *Packets*.
- **OverlayRTTtester:** *Stand-by Thread*. Contacta a sua fonte periodicamente para testar atrasos/falhas de conexão.
- **OverlayMenu:** *Stand-by Thread*. Apresenta um menu ao utilizador com várias opções de visualização do estado atual do nó, como o estado das conexões, a sua velocidade e acionar um mecanismo de "FakeLag".
- **BootStrapper:** Potencial *Stand-by Thread*. Caso o IP do *bootstrapper* for "localhost", o *Overlay* toma o papel de *bootstrapper*, inicializando uma *thread* à escuta numa porta distinta que, ao receber um pedido de inicialização, prepara e envia uma resposta a esse nodo baseado nos ficheiros de configuração fornecidos.
- **OverlayConnection:** Elemento básico do Overlay. Este define o estado de uma "conexão", assim como se este destino está ativo e, caso seja uma fonte, o seu atraso e número de saltos até ao servidor.
- **OverlayConList:** Elemento principal do Overlay, contém a lista de conexões dos vizinhos, a lista de servidores conhecidos e lista de destinos. Contém toda a lógica de criação, gestão, modificação, *rerouting* e eliminação de conexões.

Server



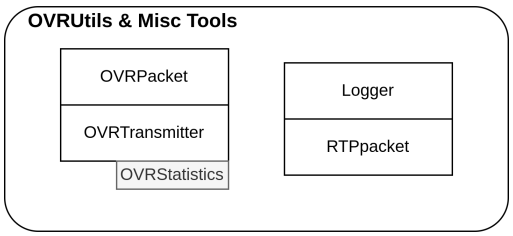
- **ServerMain:** Lógica principal do servidor. Começa por juntar-se à rede Overlay, procedendo ao *flooding* desta. De seguida testa a sua velocidade e, depois de iniciar a *stream*, permanece em *stand-by*, à escuta de pacotes de ativação ou teste de velocidade.
- **ServerStreamer:** *Thread* ativa de *Stream*. Processa as várias *frames* ao ritmo de 25fps (40ms/frame) num *loop* infinito, quando a *stream* estiver ativa, envia o dados da *frame* para a rede *overlay*.
- **ServerSpeedTester:** *Stand-by Thread*. Testa periodicamente o atraso do servidor na rede *Overlay*.
- **ServerVideoStream:** Código fornecido pelos Docentes para leitura do ficheiro de vídeo e processamento das suas *frames*.

Client



- **ClientMain:** Lógica principal do Cliente. Junta-se à rede Overlay, e apresenta um Menu com a opção de ativar a *stream*.
- **ClientStreamFetcher:** *Thread* de apresentação da *stream*. Quando chamada, abre a janela com a *stream*, recolhendo os dados das *frames* em tempo real.

OVR Utils & Misc Tools

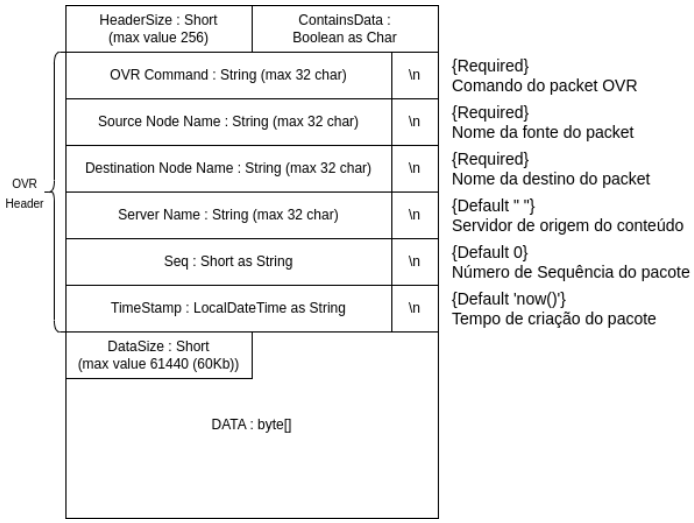


- **Logger:** Classe estática de Logging. Cria um Log específico da máquina, e fornece funções de escrita customizada neste, assim como logging de pacotes. Fornece o nome da máquina.
- **RTPpacket:** Código fornecido pelos docentes pafa criação e leitura de pacotes RTP.
- **OVRPacket:** Classe de representação dos pacotes do protocolo OVR. Contém toda a informação de uma pacote, assim como funções de serialização e desserialização dos pacotes.
- **OVRTransmitter:** Classe responsável por enviar e receber pacotes entre nós. Contém funções de redirecionamento e de envio de determinados comandos/pacotes e os argumentos requeridos desse comando. Responsável por gravar a quantidade de *bytes* recebidos/enviados para cada nodo, para efeitos estatísticos.
- **OVRStatistics:** *Stand-by Thread*. Responsável por ler a quantidade de *bytes* enviados/recebidos por cada nodo, calcular a velocidade das conexões e gravar os resultados destas para serem apresentadas.

3 Especificação dos Protocolos

Neste capítulo, serão apresentadas as especificações do protocolo desenvolvido, incluindo o protocolo dos pacotes utilizados entre os nós de *Overlay*, assim como definir as várias interações que ocorrem na rede *Overlay*.

3.1 Formato das mensagens protocolares - OVR Packet



Formato OVR Packet

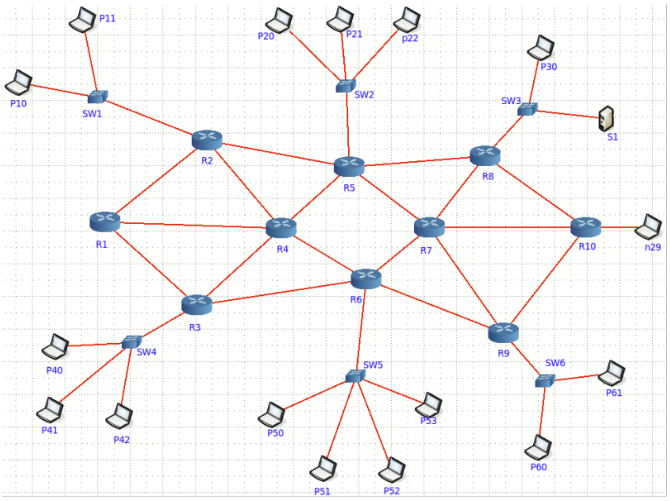
OVR Command	Descrição
INITREQUEST	Pacote de inicialização dos Nodos OVR. Responsável por pedir os vizinhos ao BootStrapper.
FLOOD	Pacote de Flooding. Usado para anunciar a presença de um servidor.
SETASDESTINATION	Pacote de criação de rota Usado para gerar a árvore de difusão
SPEEDTEST	Pacote de Teste de Velocidade ao Servidor. Usado para medir o tempo de atraso ao Servidor.
RTT	Pacote de Teste de Velocidade ao nodo de fonte. Usado para medir o tempo de atraso da fonte.
JOIN	Pacote de inicialização de Clientes/Servidor. Pedido de conexão à rede Overlay.
LEAVE	Pacote de abandono de Clientes/Servidor. Pedido de desconexão à rede Overlay.
ACTIVATE	Pacote de ativação de stream.
DEACTIVATE	Pacote de desativação de stream.
DATA	Pacote de dados genérico.

Comandos OVR Packet

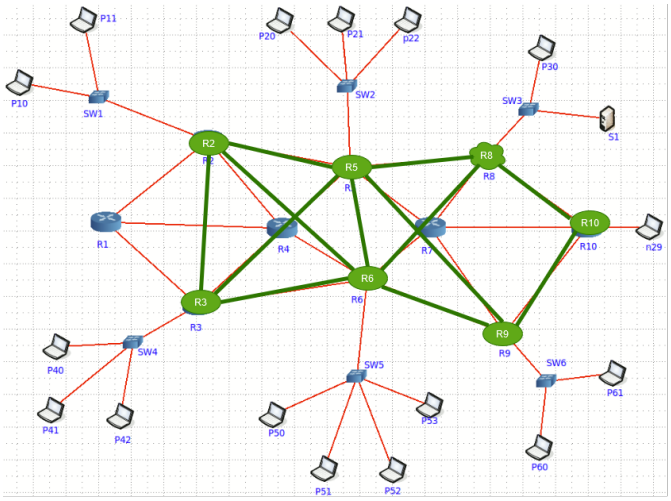
3.2 Interações

Para as várias interações, decidimos dividir e ordenar estas na sua ordem natural de ocorrência, começando por demonstrar a definição de uma das redes *Overlay* numa topologia.

Topologia de exemplo (Topologia 3)



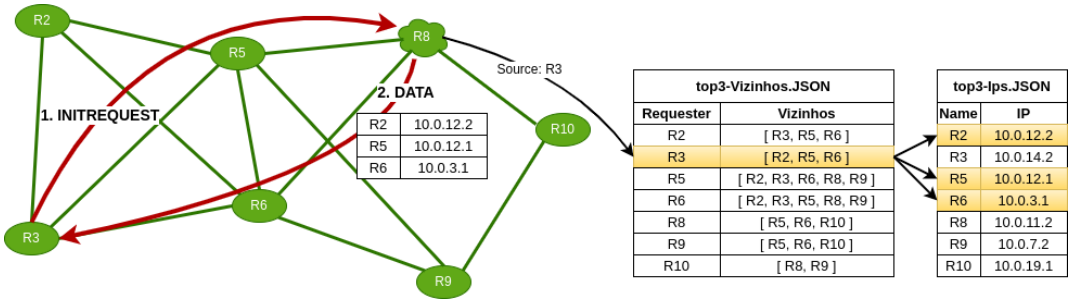
Topologia da Rede.



Rede *Overlay*.

BootStrapper (R8)

Para a inicialização de um nó da rede *Overlay*, este começa por contactar o *Bootstrapper* através do IP fornecido na sua execução. O *Bootstrapper*, através do campo 'Source Node Name', prepara uma resposta personalizada para esse nodo, através da informação presente nos ficheiros de configuração.



R2	10.0.12.2
R5	10.0.12.1
R6	10.0.3.1

top3-Vizinhos.JSON

Requester	Vizinhos
R2	[R3, R5, R6]
R3	[R2, R5, R6]
R5	[R2, R3, R6, R8, R9]
R6	[R2, R3, R5, R8, R9]
R8	[R5, R6, R10]
R9	[R5, R6, R10]
R10	[R8, R9]

top3-Ips.JSON

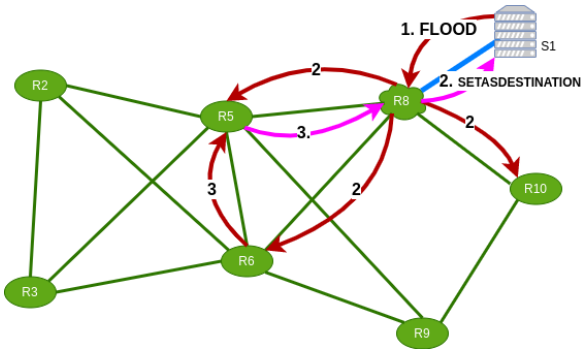
Name	IP
R2	10.0.12.2
R3	10.0.14.2
R5	10.0.12.1
R6	10.0.3.1
R8	10.0.11.2
R9	10.0.7.2
R10	10.0.19.1

Flooding e Árvore de Difusão

Após a inicialização da rede *Overlay*, passamos à inicialização dos servidores de *streaming*. Estes juntam-se à rede e procedem ao *flooding* da mesma de modo a anunciar a sua presença.

Quando um nó recebe um pacote FLOOD, este toma conhecimento do servidor em questão e procede ao FLOOD dos seus vizinhos. Adicionalmente, caso ainda não possua um nó "fonte" para esse servidor, envia um pacote SETASDESTINATION para o nó emissor do pacote FLOOD, de forma a montar a árvore de difusão. Caso o nó já conheça o servidor anunciado, este FLOOD é ignorado.

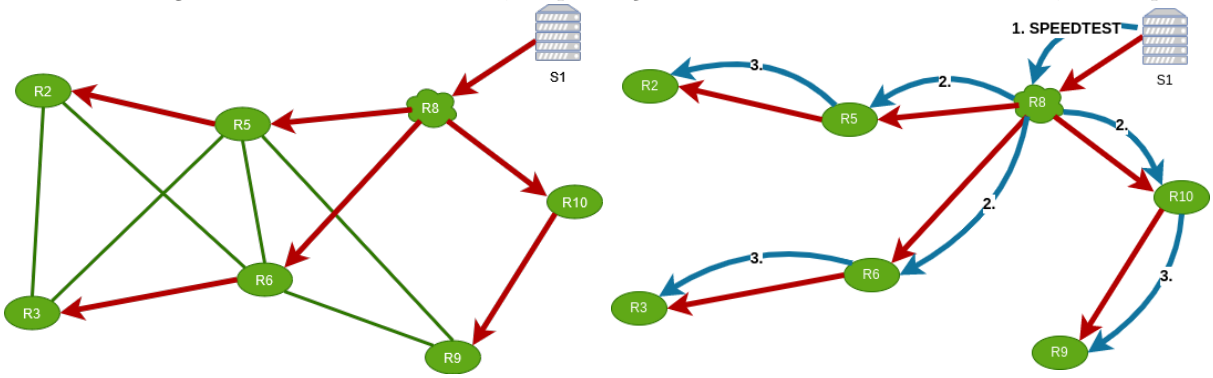
No diagrama seguinte, apresentamos um pequeno exemplo (incompleto) que realça as interações nos nós R5 e R8 de forma a representar esta lógica de forma mais simples.



Iteração	Descrição
1	**S1 -** Envia Flood para R8; **R8 -** Toma conhecimento do servidor S1
2	**R8 -** Envia SetAsDestination para S1, definindo este como fonte. **R8 -** Envia Floods a todos os seus vizinhos, exceto S1; **R5,R6,R10 -** Toma conhecimento do servidor S1.
3	**R5 -** Envia SetAsDestination para R8, definindo este como fonte. **R6 -** Envia Floods a todos os seus vizinhos, exceto R8; **R5 -** Recebe Flood de R6, mas ignora pois já conhece S1.

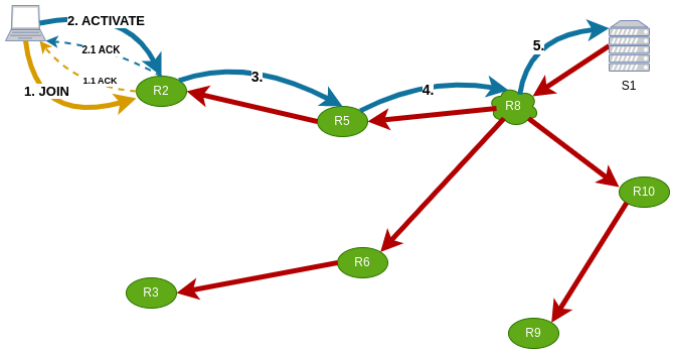
Teste de Velocidade ao Servidor

Depois da árvore de difusão estabelecida (Figura à esquerda), o servidor efetua periodicamente testes de velocidade, tendo cada nó de gravar o atraso ao servidor e, na presença de um servidor com menor atraso, trocar para este.



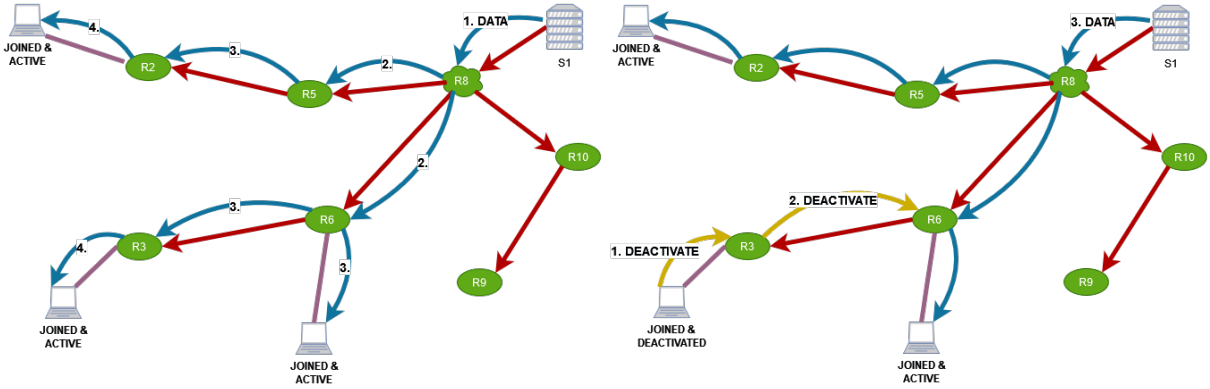
Join e Ativação dos Clientes

Quando um cliente pretende conectar-se à *stream*, este começa por enviar um JOIN ao nó *overlay* mais próximo, fornecido no início da sua execução. Quando este retornar um *Acknowledgement*, o Cliente considera-se conectado à rede *Overlay*, na qual poderá enviar um pacote ACTIVATE para iniciar a *stream*. Caso o cliente receba outro *Acknowledgement*, indicando que o envio de dados se irá iniciar, poderá então apresentar a *stream* ao utilizador.



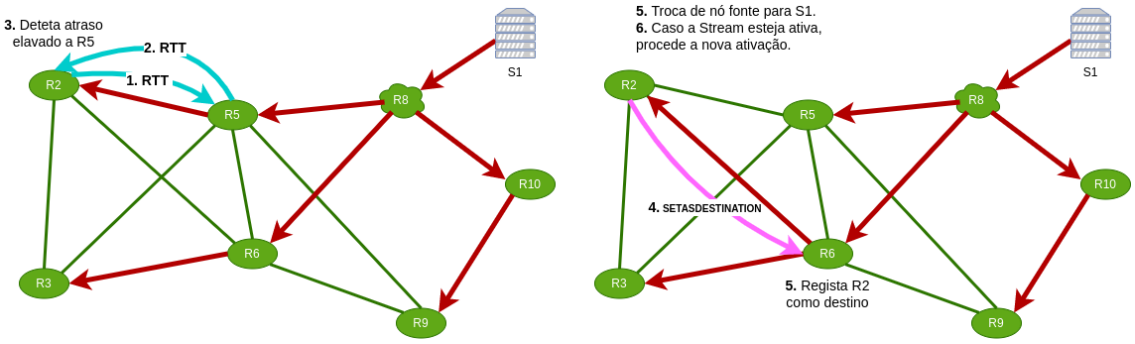
Envio de Dados e Desativação de Clientes

Na figura à esquerda, apresentamos um exemplo do *flow* de Dados quando vários clientes estão ativos na rede *Overlay*. Na figura à direita, apresentamos um exemplo de desativação de um dos Clientes, e como os vários nós se vão desativando ao longo da árvore de difusão, conforme a inexistência de clientes.



Reconexão de Nós

De seguida apresentamos as várias interações que ocorrem quando um nó de *Overlay* desaparece ou sofre de problemas de atrasos. Começamos por apresentar na figura à esquerda o método de deteção destas falhas/atrasos, através de *pings* periódicos RTT (Round Trip Time) ao nodo fonte. Quando é detetado uma falha o nó procede à troca de Servidores ou, no caso de não haver melhores opções, procura trocar de nó fonte para uma conexão vizinha que não esteja na lista de destinos.



4 Implementação

Nesta secção, o grupo irá explorar a implementação do protocolo descrito, assim como as várias decisões e desafios encontrados nesta fase. Esta secção será dividida de acordo com as componentes principais desenvolvidas, explicando o funcionamento mais detalhado destas e indicando as funções mais importantes.

4.1 OverlayMain

Esta classe representa a *Thread* principal dos nós da rede *Overlay*. Esta começa por ler o IP fornecido no início da sua execução (se não for fornecido a execução é terminada) e, caso este represente um *loopback address* (*localhost*), o nodo irá criar uma *Thread* adicional que irá correr toda a lógica do BootStrapper.

Independentemente do IP fornecido, o próximo passo do nodo será a inicialização do *transmitter* (**OVRTransmitter**), responsável pela troca de pacotes, que irá ocupar uma *socket* UDP na porta 5010 (**overlayPort**), e de seguida, o nodo inicializa um objeto da classe **OverlayConList**, responsável pela gestão da lista de conexões entre nodos, servidores e clientes. Esta classe também contém a função de contacto ao Bootstrapper, que irá correr executar nesta fase, sendo a MainThread apenas responsável por assegurar que esta ocorre sem problemas.

Depois do BootStrapper contactado e a informação dos nodos vizinhos inserida, a *Thread* principal gera duas *Threads*: **OverlayRTTtester** e **OverlayMenu**, entrando de seguida no seu *loop* infinito principal, onde são atendidos e tratados todos os pacotes da rede *Overlay*. Esta lógica é composta de um "receive packet" seguido de um *switch* case para os vários comandos do pacote a serem tratados.

BootStrapper: Esta *Thread* é responsável por "informar" os nodos da rede *Overlay* acerca dos seus vizinhos e os respetivos IP's destes. Para isso, na sua inicialização, este lê dois ficheiros de configuração estáticos, um que indica para cada nome do nodo o seu respetivo IP e outro que indica, para cada nodo, uma lista do seus nodos vizinhos. Assim, quando contactado na sua porta 5000 (**bootstrapperPort**) através de um pacote INITREQUEST, o nome presente no campo *srcNodeName* do pacote é utilizado para preparar e enviar uma resposta personalizada desse nodo, em formato JSON.

OverlayRTTtester: Esta *Thread* envia periodicamente um pacote de teste de atraso RTT (Round Trip Time) ao nodo fonte, com o propósito de testar a sua conexão. Para evitar problemas de concorrência na utilização do *transmitter*, esta classe contém o seu próprio *transmitter*, que utiliza uma *socket* na porta 5011 (**overlayPort**+1) e em intervalos de de 5s (**RTTRefreshRateMS**), envia um pacote RTT ao nodo fonte, registando que este foi enviado (**awaitingRTT**). Caso não seja enviada nenhuma resposta a este RTT, esta *Thread* é responsável por assinalar a falha de conexão, para que a MainThread acione os protocolos de redirecionamento.

OverlayMenu: *Thread* de apresentação e lógica do menu de *overlay*. Responsável por ler o input do utilizador e apresentar o estado do *overlay*, assim como acionar um mecanismo de "FakeLag", que irá introduzir um tempo de atraso artificial na lógica de leitura e resposta de pacotes da MainThread.

4.2 OverlayConLists

Esta classe contém a lógica principal acerca do tratamento e gestão de rotas e conexões. Possui a informação de todos os vizinhos, assim como servidores conhecidos e destinos possíveis gravados em vários objetos Map.

getNeighboursInfo: Função responsável por contactar o BootStrapper e, no caso de resposta, efetuar o tratamento da resposta e inserir os vizinhos nas estruturas correspondentes.

getBestServer: Função que analisa a lista de servidores conhecidos e os seus atrasos, escolhendo o melhor (servidor com o menor atraso).

setConnectedServer: Função de troca de servidor preferido.

activateServerConnection: Função de ativação do servidor escolhido.

findNewPathToServer: Função de mudança de fonte. Começa por procurar uma possível fonte ao servidor e, caso encontre, trata de efetuar a desconexão da fonte anterior e conectar com a nova, assim como ativar a *stream*, caso seja necessário.

4.3 ServerMain

Esta classe representa a *Thread* principal do Servidor. Esta começa por ler o IP fornecido no início da sua execução, assim como o nome do ficheiro de vídeo a ser transmitido. De seguida é inicializado o *transmitter*, que também utilizará a porta 5010 (**overlayPort**). Depois de inicializado, são feitos os primeiros contactos com a rede *Overlay*, sendo efetuado um JOIN, seguido de um FLOOD para anunciar a sua presença e a montagem da árvore de difusão. Após um pequeno intervalo para permitir que o *flood* ocorra com sucesso, é efetuado um primeiro teste de velocidade e são inicializadas as *Threads* **ServerStreamer** e **ServerSpeedTester**. Finalmente, o servidor entra num *loop* semelhante de leitura e tratamento de pacotes, responsável por ativar e desativar a *stream* conforme pedido.

ServerStreamer: Esta *Thread* contém o *loop* de processamento de *frames* do vídeo fornecido. Esta classe processa as *frames* a um ritmo de 25 *frames* por segundos, de forma a manter uma velocidade constante e aproximada do vídeo original. Caso a *stream* esteja ativa, a informação da *frame* processada é enviada para a rede *Overlay*.

ServerSpeedTester: Esta *Thread* é responsável por, periodicamente, a cada 10s (**speedTestRateMS**), enviar um pacote de SPEEDTEST de forma a atualizar o tempo de atraso ao servidor.

4.4 ClientMain

Esta classe representa a *Thread* principal do Cliente. Esta começa por verificar se o *Display* foi definido e por ler o IP fornecido e, de seguida, inicializa o *transmitter*, que também utilizará a porta 5010 (**overlayPort**). Depois de inicializado, são feitos os primeiros contactos com a rede *Overlay*, sendo efetuado um JOIN, sendo este respondido por um ACK. Caso este seja respondido com sucesso, a *Thread* irá abrir um Menu onde será dada a opção para iniciar a *stream*, ou desligar a aplicação. Caso a *stream* seja iniciada, é enviado um pacote ACTIVATE, que também deverá ser respondido por um ACK, que, quando recebido, iniciará uma *Thread* **ClientStreamFetcher**, e a *Thread* principal ficará à espera que esta termine.

ClientStreamFetcher: Esta *Thread* é responsável por ler e apresentar as *frames* da *stream* recebidas da rede *Overlay*. Enquanto a janela estiver ativa, esta *Thread* recebe pacotes de dados da rede *Overlay*, convertendo estas para as *frames* de vídeo a serem apresentadas.

4.5 OVR Utils & Misc Tools

Esta secção apresenta as várias classes comuns e auxiliares entre as várias componentes do sistema OVR.

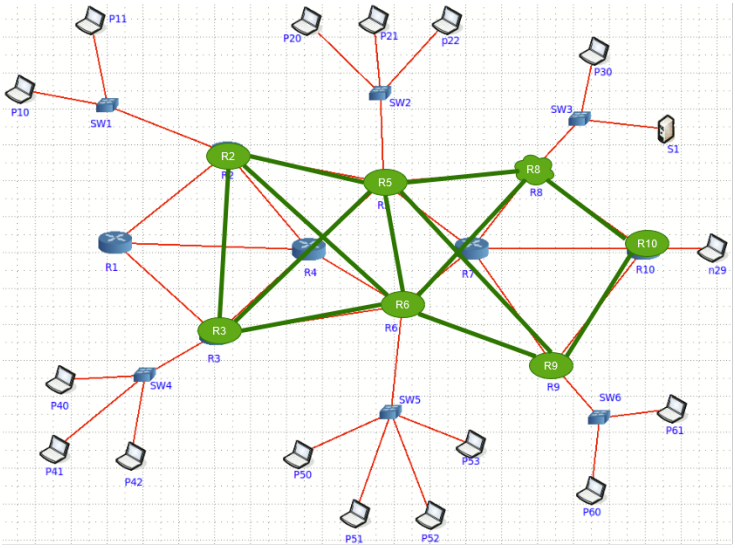
OVRPacket: Classe de tratamento dos pacotes OVR. Esta contém todos os atributos presentes na especificação do pacote OVR, assim como as verificações de limites de tamanho e argumentos inválidos. Contém, também, as funções **serialize** e **deserialize**, responsáveis por escrever a informação do pacote num *array* de *bytes* e vice-versa, respetivamente.

OVRTransmitter: Classe de empacotamento e envio de pacotes OVR, responsável por gerir a socket de receção e emissão de pacotes, contendo funções para alteração de destino dos pacotes, apresentação de estatísticas, receção de pacotes e funções para o envio dos vários tipos de pacotes OVR.

Logger: *Thread* estática de *Logging*. Na sua execução, cria um ficheiro *log*, com o nome do utilizador e máquina, onde poderão ser escritas várias mensagens. Disponibiliza funções de escrita de erros, avisos, mensagens personalizadas e adicionalmente, escrita de pacotes OVR.

5 Testes e Resultados

Para demonstrar o funcionamento da rede *Overlay*, assim como apresentar o resultado final visto pelos utilizadores, vamos usar um exemplo de execução, baseado na topologia anteriormente apresentada. Começamos por ligar todos os nodos na rede *Overlay*, começando pelo nodo R8, pois este será o BootStrapper da rede. Depois disso, iniciamos o servidor S1, que irá servir o conteúdo da *stream* aos clientes.



Rede *Overlay*. Topologia 3

```
core@R8:~$ OVRoverlay localhost
core@R8: Starting BootStrapper
core@R8: Contacting BootStrapper
core@R8: OverLay Active
Overlay Menu:
1 - Print Connection List
2 - Print Traffic Report
3 - Toggle Fake LAG (false)
```

Nodo R8 (BootStrapper)

```
core@S1:/media/sf_Mv-Shared/jars$ java -jar Server.jar 10.0.4.1
Servidor: parametro não foi indicado. VideoFileName = movie.Mjpeg
core@S1: Flooding Overlay Network...
core@S1: Pinging speed...
core@S1: Streaming...
```

Server S1

Após o Servidor efetuar o *flood* da rede *Overlay*, assim como os testes de velocidade iniciais, verificamos o estado das conexões no nó R8 de forma a verificar a árvore de difusão, assim como o tráfego neste nó. Neste último é possível verificar a existência de comunicação entre nodos (R6), devido aos pacotes RTT.

```
--------- CONNECTION LIST (R8) ---------

CONNECTED SERVER (DEACTIVATED): S1

KNOWN SERVERS:
S1: S1 Delay:6ms Jumps to Server:0

DESTINATIONS:
R10 (DEACTIVATED), R5 (DEACTIVATED), R6 (DEACTIVATED), S1 (DEACTIVATED),
POSSIBLE CONNECTIONS:
R10, R5, R6, S1,
```

Connection List R8

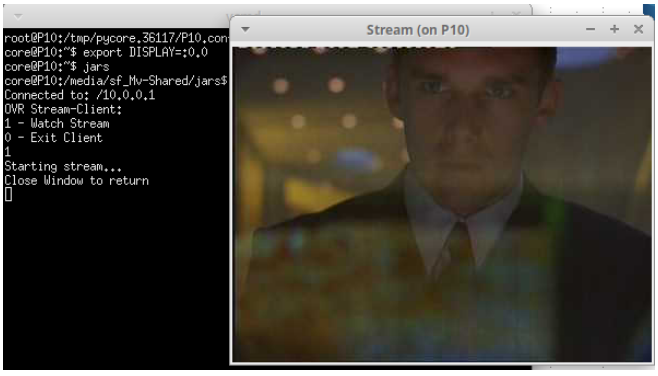
```
TRAFFIC REPORT: (Last 1 seconds)
INCOMING:
R6: 43,00 B/s,
OUTGOING:
R6: 43,00 B/s,
```

Traffic Report R8

De seguida, executamos um Cliente (P10), de forma a obter acesso à stream na rede *Overlay*:

```
core@P10:~$ export DISPLAY=:0.0
core@P10:~$ jars
core@P10:/media/sf_Mv-Shared/jars$ java -jar Client.jar 10.0.0.1
Connected to: /10.0.0.1
OVR Stream-Client:
1 - Watch Stream
0 - Exit Client
```

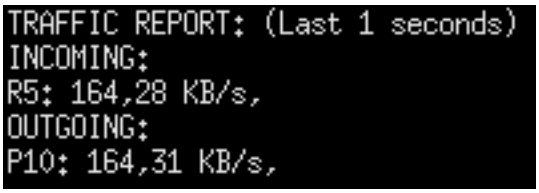
Menu Cliente P10



Stream P10

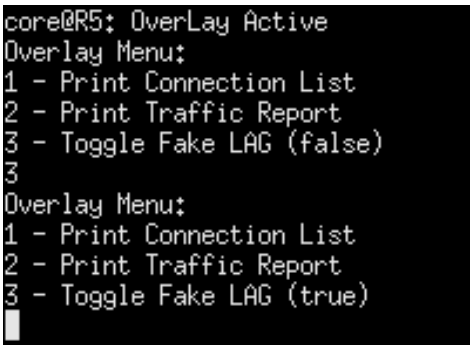
Com a Stream ativa decidimos verificar o estado do nodo R2, vizinho do Cliente P10. É possível ver na Connection List que o nodo R2 está conectado ao Servidor S1, sendo a sua fonte a este servidor o nodo R5, com um atraso de 10ms. Também é possível verificar o cliente na lista de destinos, assim como ver o tráfego presente nos nodos fonte e destino, tendo este um ritmo de tranmissão de 164KB/s.

Connection List R2



Traffic Report R2

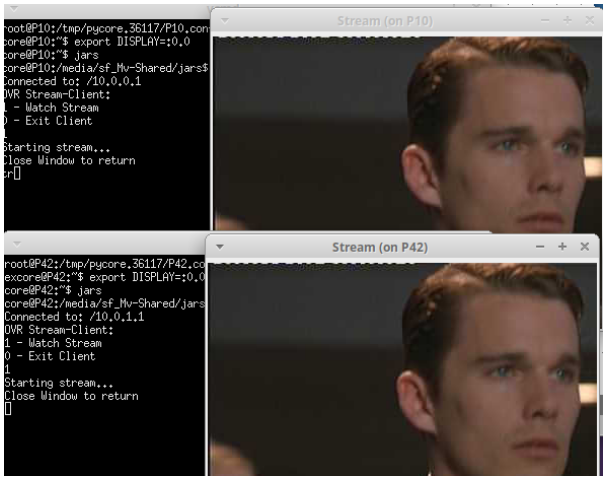
De seguida, para demonstar a capacidade de reconexão da rede, decidimos acionar o mecanismo de "FakeLag" no nodo R5 (fonte do nodo R2) de forma a testar a reação da rede *Overlay*. Após acionar o mecanismo, é possível ver que a *stream* sofre de encravos frequentes assim como saltos nas imagens, no entanto, após um pequeno intervalo, estes encravos desaparecem por completo. Voltamos a verificar a *Connection List* do R2 e podemos verificar que o nodo R2, devido aos atrasos a R5, modificou a sua fonte para um nodo vizinho com menos atraso, passando a receber a *stream* deste.



Fake Lag ON R5

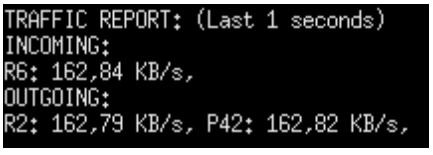
Connection List R2

Por último, decidimos iniciar outro cliente (P42), conectado ao nodo R3. Como podemos verificar, as *streams* encontram-se sincronizadas e, através do *Traffic Report*, é possível ver o processo de receção e duplicação de dados para os vários emissores.



Cliente adicional P42

Connection List R3



Traffic Report R3

6 Conclusões

Este trabalho proporcionou, sem dúvida, uma consolidação de conhecimentos sobre protocolos de *streaming* e redes *overlay*, permitindo ao grupo obter uma nova visão sobre os mesmos. Este projeto dividiu-se maioritariamente em duas fases: esboço do protocolo de *streaming* e implementação do mesmo.

Relativamente ao **esboço do protocolo**, pensamos ter desenvolvido um relativamente funcional, estando cientes dos problemas como a falta de controlo de pacotes de confirmação dentro da rede *overlay*, assim como potenciais problemas no protocolo de reconexão quando este se encontra numa rede com "bottlenecks".

No que toca à **implementação**, esta não foi desenvolvida na sua totalidade, faltando alguns aspetos definidos no esboço do protocolo como a possibilidade de conexão de vários servidores, apesar desta possuir uma base que permite uma futura implementação.

Em suma, todo o planeamento e desenvolvimento deste projeto foi útil e cativante, proporcionando uma boa abordagem à área da conexão e protocolos de *streaming*.