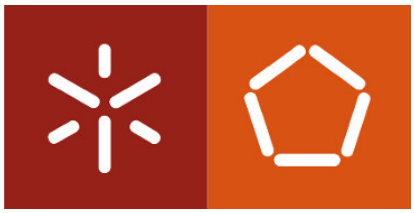


Universidade do Minho

Mestrado em Engenharia Informática

Engenharia de Serviços em Rede
Trabalho Prático 2

Rúben Samuel Alves Santos (pg50733)

25 de abril de 2023

1 Introdução

Para o segundo trabalho realizado no âmbito da Unidade Curricular de Engenharia de Serviços em Rede foi desenvolvido um sistema "Over the Top"(OTT) para entrega de multimédia. Seguem no presente relatório secções que correspondem às etapas da realização deste projeto e nas quais é demonstrado o desenvolvimento de cada uma. Finaliza-se depois com uma secção de conclusões onde o grupo faz um balanço do trabalho realizado e da experiência adquirida durante o mesmo.

2 Preparação do Projeto

Para a realização do projeto, foi necessário tomar decisões acerca de como este seria implementado, nomeadamente, teve de se escolher a linguagem de programação mais conveniente (tanto para o projeto em si como para os elementos do grupo), criar uma topologia que permitisse estabelecer diferentes cenários possíveis que devessem ser testados, determinar quais os protocolos de transporte e em que situações cada um deve ser utilizado e implementar um cliente/servidor base a partir do qual o projeto seria desenvolvido.

Devido à sua conveniência, facilidade de uso num contexto de redes, disponibilidade de código de apoio e a todos os elementos terem uma certa experiência com esta linguagem, foi decidido que a linguagem utilizada seria o Java.

A topologia estabelecida para testes foi aquela presente no enunciado, de maneira a se poder ter a certeza que se dispõe dos cenários adequados, para além de motivos de consistência e conveniência, isto é, esta topologia funcionava para fazer aquilo que era pretendido e já estava disponível, tanto que era de certa forma recomendada para este propósito. No entanto, de forma a simplificar, são feitos testes com subconjuntos de nodos desta topologia, ou seja, temos cenários de teste com diferente número de nodos e/ou ligações entre si (ver secção 6).

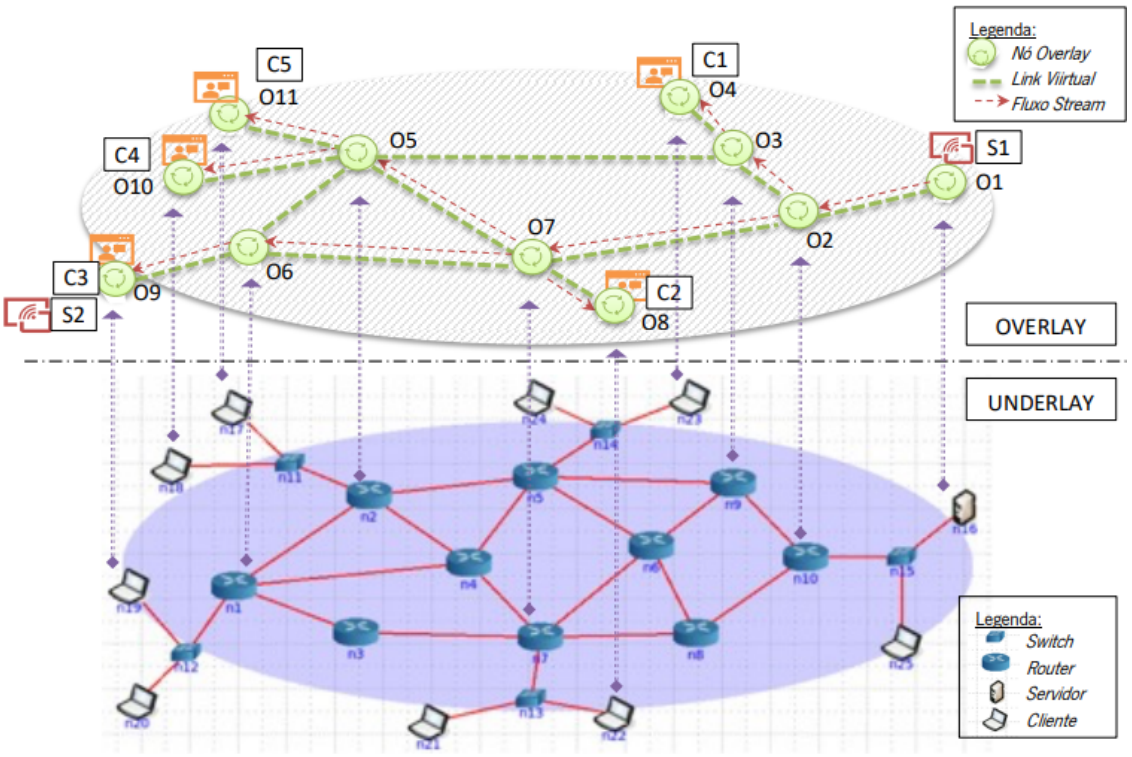


Figura 1: Visão geral de um serviço OTT sobre uma infraestrutura IP

Figura 1: Topologia utilizada no projeto.

O protocolo TCP é mais seguro, orientado a conexões e no geral é mais recomendado em casos de transmissão de dados críticos devido a melhor controlo de fluxo, deteção de erros e transmissão garantida, logo é mais indicado para estabelecimento da rede e das conexões entre os nodos.

Contudo, é mais lento e não é indicado para ser usado em tempo real, pelo que o streaming, em que não é exatamente crítico que se percam pacotes ocasionalmente e não é necessário que todos os nodos intervenientes enviem dados (geralmente um envia e os outros recebem, permitindo evitar a necessidade de haver conexão antes de cada envio) deverá ser feito através do protocolo UDP, que é menos fiável mas permite multicast e é mais rápido, sendo portanto mais indicado para o contexto em causa (streaming em tempo real).

3 Arquitetura da Solução

No âmbito deste projeto foram desenvolvidas três aplicações distintas - cliente, servidor e nodo.

3.1 Cliente

A aplicação *client* consiste numa adaptação do código disponibilizado pela equipa docente. O cliente pode dar *play*, que irá de imediato enviar uma mensagem de ativação para o nodo definido no arranque da aplicação e começar a stream (tal como vemos na figura 2). Pode também colocar a stream em pausa ou desligar completamente a stream com a opção *Teardown*. De notar que quando a stream é colocada em pausa e, posteriormente, ativamos o botão *play*, é exibido o video que estiver a ser transmitido de momento pelo servidor, ou seja,

a transmissão não é guardada, dado que nem sequer está a ser enviada para o cliente, por motivos de economização de recursos.

Figura 2: Interface da aplicação cliente.

3.2 Servidor

A aplicação servidor tem apenas duas funções: informar os nodos acerca dos seus vizinhos e enviar conteúdo multimédia para um nodo definido no arranque da aplicação. Esta última é, mais uma vez, uma adaptação do código disponibilizado pela equipa docente, com apenas algumas modificações para que seja possível transmitir o vídeo *infinitamente*. É também enviada, periodicamente, uma mensagem de *flood* para o mesmo nodo, com o intuito de atualizar as rotas da rede.

Inicialmente, as *aplicações nodo* iniciam uma conexão com o servidor. É então enviada a informação acerca da vizinhança deste nodo, obtida através de um ficheiro de configuração. Este ficheiro é apresentado na secção 5.2. É de realçar que, assim que esta fase é completada (todos os nodos receberam a informação relativa aos seus vizinhos), o servidor inicia o streaming e deixa de ser possível ligar novos nodos à rede (apenas novos clientes e servidores). Na secção de testes são apresentadas várias topologias sendo que, cada uma, terá o seu próprio ficheiro de configuração, que será indicado à aplicação servidor no seu arranque.

3.3 Nodo

A aplicação nodo é relativamente mais complexa do que ambas as anteriores. Tem várias funções, sendo que a principal é a retransmissão dos pacotes de streaming recebidos para nodos vizinhos, minimizando o número de fluxos e escolhendo a melhor rota possível. A melhor rota será a mais rápida desde o servidor e, para distinguir tempos semelhantes (10% erro), a que passe por um menor número de nodos. Desta forma deverá conseguir interpretar as mensagens descritas na secção 4 e comunicar com outros nodos vizinhos. De notar que, ao longo deste documento, quando referimos *nodo vizinho* é em relação à overlay, ou seja, nodos ligados/vizinhos na overlay podem estar, na underlay, separados por vários saltos.

Sem entrar em detalhes de implementação (discutidos na secção 5), a aplicação nodo, para além de iniciar (5.3) e manter as conexões com os seus vizinhos, retransmite mensagens de flood (com o cuidado para não entrar em ciclos). Estas mensagens permitem que cada nodo saiba qual a melhor rota até ao servidor e, como tal, qual o melhor nodo vizinho para receber a stream. Este melhor nodo é para onde enviamos todos os pacotes de ativação de stream quando já não estamos a receber o conteúdo multimédia.

4 Protocolos

O protocolo desenvolvido para dar resposta às necessidades do serviço de streaming é relativamente simples. Existem apenas três mensagens diferentes. Abaixo descrevemos estas mensagens, o seu formato e quando são utilizadas, assim como o impacto que tem no sistema.

4.1 Flood

A mensagem de flood é inicialmente enviada pelo servidor para iniciar um flood controlado de forma a manter (inicialmente obter) o conhecimento sobre o estado da rede. Esta mensagem é periodicamente difundida pela rede por iniciativa do servidor. Serão feitos tantos floods controlados como o número de servidores ativos. Se existirem dois servidores ativos então, cada um, a seu tempo, irá difundir uma mensagem de flood de forma a que todos os nodos tenham o conhecimento do estado da rede em relação a este servidor. Na tabela 1 podemos ver o formato desta mensagem.

1	serverID	jumps	serverTimestamp	route

Tabela 1: Formato da mensagem de flood.

O primeiro campo de qualquer mensagem identifica o tipo de mensagem. Neste caso o inteiro 1 está associado à mensagem de flood. De seguida temos a identificação do servidor, dado que podem existir floods de servidores diferentes. O campo *jumps* é, a cada iteração do flood, incrementado, dado que indica o número de saltos do nodo atual até ao servidor. O valor do *serverTimestamp* mantêm-se inalterado durante a retransmissão, dado que diz respeito ao instante a que a mensagem foi enviada do servidor. Este valor é utilizado para calcular a quanto tempo determinado nodo se encontra do respetivo servidor. Por fim, o campo *route* é uma lista das identificações dos nodos pelos quais esta mensagem já passou. Desta forma, antes de retransmitir a mensagem de flood, o nodo atual verifica se os seus vizinhos já receberam esta mensagem e portanto previne ciclos infinitos de trocas de mensagens.

4.2 Ativação de Stream

A mensagem de ativação de stream é enviada por um novo cliente que deseja juntar-se à rede. No entanto, caso o nodo que receba este pedido do cliente não se encontre atualmente a receber stream, o próprio nodo envia uma mensagem de ativação para o seu nodo vizinho em melhores condições para transmitir a stream. Esta informação é obtida através da inundação controlada. Esta mensagem é transmitida até ser encontrado um nodo que esteja a receber a stream. Na pior das hipóteses percorremos toda a rota deste o cliente até ao servidor. Na tabela 2 podemos ver o formato desta mensagem.

2	addressToActivate

Tabela 2: Formato da mensagem de ativação de stream.

Mais uma vez, o primeiro campo identifica a mensagem, neste caso de ativação de stream. O segundo campo é apenas utilizado pelo cliente que pede a stream. Quando o cliente envia um pacote de ativação de stream para um determinado nodo inclui na mensagem o seu endereço IP. Este endereço é então adicionado a uma lista de endereços para os quais é enviada a stream. As eventuais retransmissões da mensagem de ativação não necessitam deste campo,

dado que os nodos tem o conhecimento dos endereços dos seus vizinhos e, como tal, sabem que nodo está a pedir para que seja ativada a stream.

4.3 Desativação de Stream

A mensagem de desativação de stream é enviada por um cliente que deseja sair da rede. No entanto, caso o nodo que receba este pedido fique sem clientes interessado em receber a stream, irá enviar uma mensagem de desativação para o seu nodo vizinho que lhe está a transmitir a stream. Esta mensagem pode ser transmitida novamente caso os nodos que a recebam fiquem sem clientes interessado em receber a stream. Na pior das hipóteses percorremos toda a rota deste o cliente até ao servidor. Este mecanismos é importante pois economiza os recursos da rede. Na tabela 3 podemos ver o formato desta mensagem.

2	addressToDeactivate

Tabela 3: Formato da mensagem de desativação de stream.

Mais uma vez, o primeiro campo identifica a mensagem, neste caso de desativação de stream. O segundo campo é apenas utilizado pelo cliente que não deseja receber a stream. Quando o cliente envia um pacote de desativação de stream para um determinado nodo inclui na mensagem o seu endereço IP. Este endereço é então removido de uma lista de endereços para os quais está enviada a stream. Caso esta lista fique vazia, a mensagem de desativação de stream irá ser transmitida para o nodo que está a enviar a stream. Esta eventual retransmissão da mensagem não necessita do campo que contém o endereço a ser desativado, dado que os nodos tem o conhecimento dos endereços dos seus vizinhos e, como tal, sabem que nodo está a pedir para que seja desativada a stream.

5 Implementação

5.1 Visão Geral

Nesta secção iremos expor detalhes da implementação das três aplicações já apresentadas na secção 3. Começamos por explicar o ficheiro de configuração utilizado pelo servidor para informar os nodos acerca da topologia envolvente. De seguida analisamos o algoritmo de conexão inicial, que inclui grande parte da complexidade da aplicação servidor. Terminamos com uma visão da implementação da aplicação nodo, da forma como processa as mensagens recebidas, reenvia pacotes de streaming e organiza o seu estado interno.

Dito isto, a aplicação servidor, para além da conexão inicial (5.3), trata apenas de enviar o mesmo video, para o mesmo endereço, até ser interrompida. A aplicação cliente, exibe o vídeo numa interface gráfica e pede a ativação e desativação da stream (4).

5.2 Ficheiro de Configuração

A nossa solução parte da utilização do servidor como *bootstrapper*. De forma a agilizar este processo foi criado um ficheiro de configuração, em formato JSON, que contém a topologia da rede e irá posteriormente ser lido e interpretado pelo servidor. Foram criados vários ficheiros de configuração para os diferentes testes (secção 6). Um deles, baseado na topologia disponibilizada e apresentada na figura 1, pode ser encontrado abaixo.

Nesta configuração existem quatro nodos (O1, O2, 03 e O7) mapeados pelos equipamentos n16, n10, n9 e n7, respetivamente. Neste ficheiro, para além de incluir, por cada nodo, a sua identificação, incluímos as possíveis interfaces de saída do mesmo (*Origin*) e uma lista de

Neighbours. Esta lista inclui a identificação do nodo vizinho assim como a interface preferida para a qual se irá posteriormente comunicar. O booleano *Critical* aparece aqui pois tem impacto na leitura do ficheiro. Uma das possíveis melhorias do programa atual era iniciar as conexões e o streaming com apenas um número mais reduzido de nodos *críticos*. Isto acabou por não ser implementado mas foi mantido na configuração e o processo de conexão inicial ainda depende deste valor.

5.3 Algoritmo das Conexões Iniciais

O algoritmo que garante a correta inicialização da topologia é descrito, de forma geral, na figura 4. De notar que, por motivos de simplicidade, são apenas considerados dois nodos. Consideramos também que estes são os únicos nodos da topologia e que estão ligados (são vizinhos) na overlay.

O passo inicial é pedir ao servidor a lista dos endereços dos nodos vizinhos. Neste caso o nodo 1 faz o pedido primeiro, sendo que recebe a informação sobre os seus nodos adjacentes e fica à espera que o resto dos nodos recebam a sua. Assim que o servidor tenha enviado esta informação para todos, é enviada uma mensagem a todos os nodos com este mesmo facto. Cada nodo então cria uma thread apelidada de *StarterListener*. Esta thread irá esperar por conexões. Assim que esta thread se encontre bloqueada na chamada *socket.accept()*, cada nodo informa o servidor. Quando todos os nodos estiverem neste estado, o servidor informa os nodos. Neste momento, cada nodo entra no processo *StarterSender*. Assim que este processo e a thread textitStarterListener terminem, cada nodo informa o servidor que tem as conexões formadas. Mais uma vez, assim que todos os nodos estão conectados com os seus vizinhos, o servidor informa todos os nodos. As conexões entre os nodos e servidor são então fechadas e dá-se início ao flood inicial, seguido do streaming.

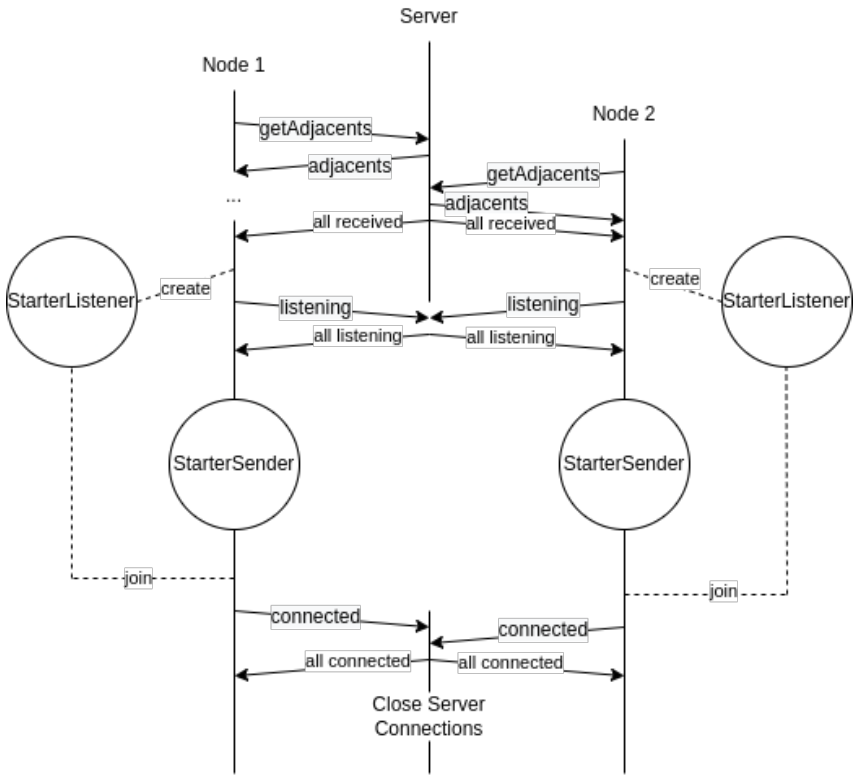


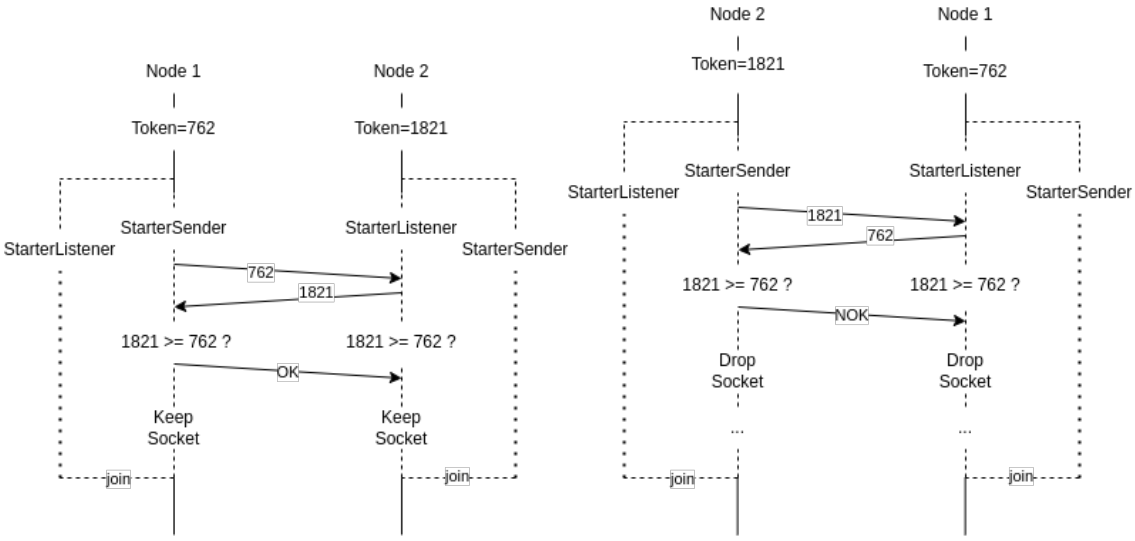
Figura 4: Visão geral do processo das conexões entre nodos.

A figura 5 tenta expor, em detalhe, o processo de sincronização entre cada nodo. É importante ter em conta que, na realidade, o *starterSender* de cada nodo está a comunicar com o *starterListener* de outro nodo, ao mesmo tempo que o *starterSender* desse mesmo nodo está a comunicar com o nosso *starterListener*. Isto acontece ao mesmo tempo para todos os vizinhos de um nodo, portanto, é expectável haver tentativas falhadas de conexão.

O processo em si é simples de entender, menos simples de implementar e corrigir. Cada nodo gera inicialmente um número aleatório. Na figura 5a o nodo 1 envia o seu número e, posteriormente, recebe o número do nodo 2. Caso o seu seja maior ao igual ao recebido, envia uma mensagem OK e a socket é mantida para futuras comunicações. No caso da figura 5b que, relembro, acontece em paralelo, os papéis invertem-se e é o nodo 2 que envia o seu número primeiro. Dado que, neste caso, o resultado não é o esperado, a socket é fechada.

Neste caso, a socket criada fica ao encargo, no nodo 1, do *starterSender* e no nodo 2, do *starterListener*. Se existirem mais nodos, no final desta sincronização, cada nodo terá sockets em ambas as threads, sendo que é ainda necessário juntar estas sockets numa lista única, na thread principal.

No final deste processo, de acordo com a figura 4, cada nodo informa o servidor que está conectado e aguarda que todos os nodos realizem as suas conexões.



(a) Conexão é bem sucedida.

(b) Conexão é fechada.

Figura 5: Algoritmo de sincronização entre dois nodos.

5.4 Nodo

Nesta subsecção partimos do ponto em que ficamos após as conexões iniciais com os nodos vizinhos. Na base da implementação da aplicação nodo está uma estrutura de dados *LinkedBlockinQueue* disponibilizada pelo *package java.util.concurrent*. Esta fila bloqueante permite que várias threads possam introduzir tarefas para que outras threads executem estas tarefas e devolvam resultados, mantendo a fila sempre mínima. O figura 6 tenta, de forma geral, identificar as entidades que tem acesso a esta fila e o que fazem com a mesma.

Analisando a arquitetura da aplicação nodo podemos desde já definir que apenas uma thread (main) tira dados da fila. O conjunto de *reading threads* é composto por V + 1 threads, sendo V o número de vizinhos. Existe uma thread que espera por novos pedidos de stream por parte de clientes.

O conjunto de *worker threads* é gerido por um *Executor Service* também disponibilizado pelo *package java.util.concurrent*. Este serviço nada mais é do que uma *thread pool*, que garante a existência de um determinado número mínimo de threads e cria mais threads caso necessário, preocupando-se depois em terminar as mesmas.

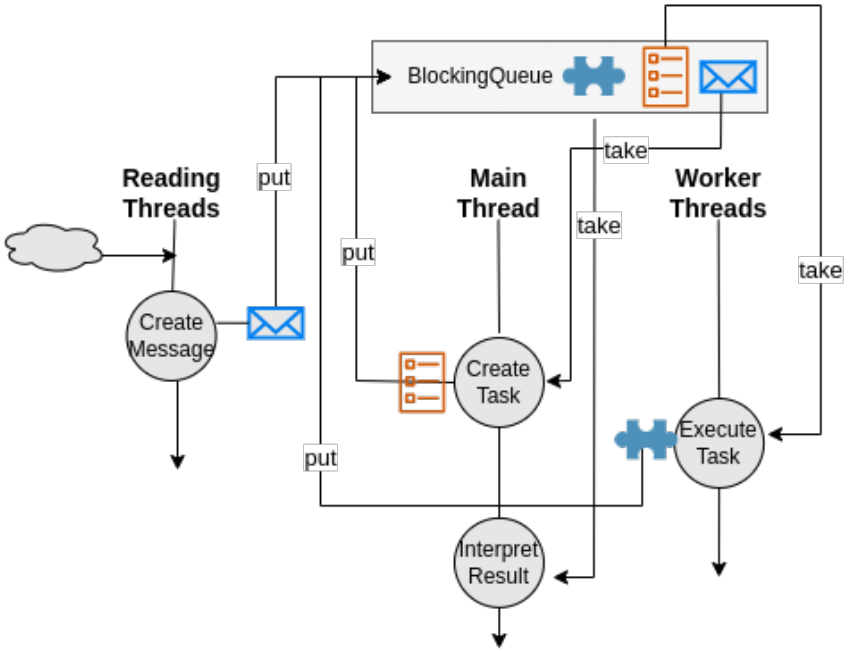


Figura 6: Arquitetura genérica da aplicação nodo.

Em relação ao fluxo de informação, são as *reading threads* que recebem informação de outros nodos ou clientes. Quando recebem esta informação, que pode ser um pedido de stream, um pacote de flood ou um pedido de desativação de stream, criam uma mensagem com a tag *OP_READ* identificada pela carta azul na figura 6. Adicionam então esta mensagem à fila bloqueante e continuam a ouvir, à espera de mais informação.

A thread principal, bloqueada a ler a fila, retira a mensagem, verifica que tipo de tarefa está a ser pedida, cria a tarefa e envia-a para o serviço de gestão das *worker threads*. Uma destas threads irá executar a tarefa e irá colocar o seu resultado novamente na fila.

O serviço de streaming é executado numa outra thread que não consta da figura acima. Esta thread tem apenas o conhecimento de uma lista para a qual tem de reenviar os pacotes de streaming. Se a lista estiver vazia, não faz nada; se a lista tiver algum endereço, aguarda por pacotes de streaming e, assim que recebe, reenvia para os endereços da lista. Este registo é manipulado pela thread principal, tendo em conta apenas as mensagens de ativação/desativação de stream.

As mensagens de ativação/desativação, quando recebidas, seguem o processo descrito anteriormente, sendo que são tratadas com as tags OP_(DE)ACTIVATE_STREAM. Na fase de interpretação de resultados, a lista de nodos para os quais se tem de enviar a stream é atualizada. Pode ser também retransmitida a mensagem de ativação/desativação, caso o nodo não esteja a receber stream ou não tenha nenhum outro nodo para enviar a stream, respetivamente.

A inundação, periódica e controlada, por parte do servidor, é tratada seguindo o fluxo já apresentado. Na fase de execução da tarefa, é criada uma nova tarefa para atualizar o estado interno do nodo, assim como outra tarefa para retransmitir a mensagem de flood. Como já

referimos anteriormente, as tarefas tem tags, neste caso, OP_CHANGE_STATE e OP_WRITE, respetivamente. Na fase de interpretação de resultados, pode ser necessário desativar uma rota que já não é ótima e ativar a melhor rota do momento.

6 Testes e Resultados

Nesta secção de testes iremos demonstrar, na medida do possível, o correto funcionamento do sistema. Para tal foi utilizado o cenário de teste completo, que podemos ver na figura 7. De realçar que, apesar de o número e as interfaces dos nodos sejam constrangidas pelo ficheiro de configuração, o número de clientes e servidores não são. Desta forma, caso desejemos lançar outro programa cliente, basta indicar o endereço de um qualquer nodo. Tanto a aplicação cliente como a aplicação servidor podem ser lançadas em qualquer máquina, inclusive na que está a correr a aplicação nodo.

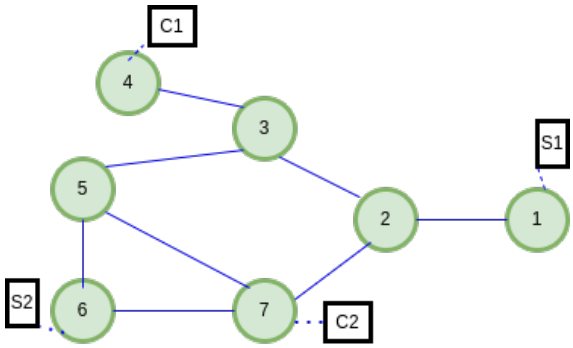


Figura 7: Cenário de teste completo.

Abaixo segue a descrição do teste completo realizado, seguido de uma pequena análise dos resultados obtidos. As capturas de ecrã relevantes encontram-se no apêndice.

1. Executar o programa servidor, seguido de todos os programas nodos nas respetivas máquinas.
2. Observar que a mensagem de flood é transmitida por todos os nodos e que o servidor começa a stream.
3. Executar o programa cliente C1. Pedir acesso à stream através do botão *Play*. Obervar que a stream segue a rota S1-O1-O2-O3-O4-C1.
4. Executar o programa cliente C2. Pedir acesso à stream através do botão *Play*. Obervar que a stream segue a rota S1-O1-O2-07-C2.
5. Adicionar 1s de delay à aresta O2-O7. Esperar pelo novo flood e verificar que apenas a rota do cliente C2 foi alterada, sendo agora S1-O1-O2-O3-O5-O7-C2.
6. Inicializar o segundo servidor S2. O cliente C2 irá agora consumir a stream do servidor S2, mudando para a rota S2-O6-O7-C2. A rota do cliente C1 deverá manter-se igual dado que não há nenhum ganho relativamente superior (10%) em consumir a stream do servidor S2.
7. Aguardar pelo flood de ambos os servidores. Nada muda.
8. Eliminar a aresta O2-O3, aguardar por novo flood. Verificar que ambos os clientes passam a utilizar o servidor S2.

Os resultados do teste são como esperado. No entanto existem duas pequenas inconveniências. No passo 6, aquando da inicialização do servidor 2, o cliente C2 consome, durante alguns segundos, a stream de ambos os servidores 1 e 2. Isto acontece pois o flood do servidor 1 ainda não aconteceu e como tal o nodo ainda não desativou esta rota. O segundo problema acontece apenas às vezes, sem causa aparente. No passo 8, após desativar a aresta O2-O3 e o flood ser difundido, o cliente 1, apesar de estar a receber os pacotes de streaming não os está a exibir na interface gráfica. O término da stream por vezes não funciona como o esperado, nomedamente com o segundo servidor ativo.

7 Trabalho Futuro

Este trabalho conta ainda com muitos aspetos que podem ser melhorados. Os pequenos erros detetados na fase de teste e que não conseguiram ser resolvidos seriam a primeira prioridade a resolver. Para além disto, a monitorização em tempo real do tempo para o servidor (algo que foi feito mas não foi integrado no projeto), poderia ajudar na decisão rápida de mudar de rota sem depender sempre da inundação controlada por parte do servidor. Atualmente o sistema depende do *flooding* para atualizar para a melhor rota. Foi experimentado um algoritmo para mudar rotas localmente sem necessidade de flood (ainda que não garantindo a melhor rota) mas, mais uma vez, não foi incluído no projeto. De resto, apresentamos uma base sólida, que serve de ponto de partida bastante válido para qualquer outra implementação de uma solução que garanta este tipo de atualização de rotas sem depender de flood.

8 Conclusão

Com este segundo trabalho *tivemos* oportunidade de aplicar soluções de streaming a pedido praticadas no trabalho prático anterior. Foi desenvolvido um protótipo de um serviço de streaming em tempo real que transmite dados desde um dado servidor para múltiplos clientes. Isto permitiu-nos consolidar os conhecimentos adquiridos durante o trabalho anterior e as aulas teóricas, bem como perceber vários aspetos acerca de serviços OTT tais como os problemas relacionados com a congestão de rede e gestão de recursos e métodos para minimizar tais problemas, nomeadamente a utilização dos protocolos mais indicados a cada situação. O grupo aprecia estes novos conhecimentos, não só como estudantes desta área mas também como utilizadores deste tipo de serviços, e espera utilizá-los durante a realização de projetos futuros.

9 Apêndice

Neste apêndice estão as capturas de ecrã relativas à secção 6.

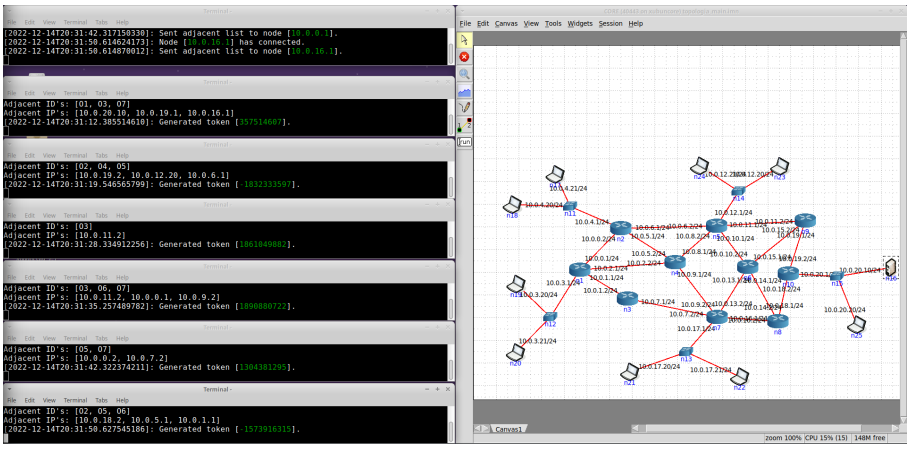


Figura 8: Inicialização da rede. Servidor e todos os nodos exceto o 1.

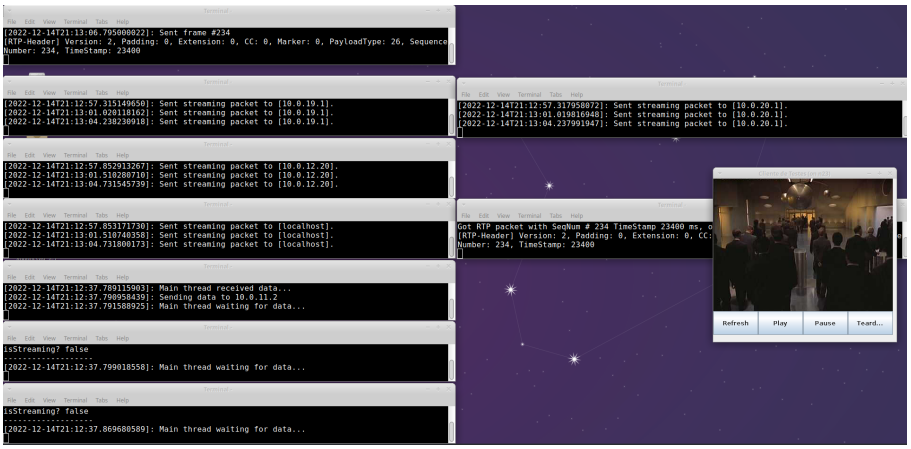


Figura 9: Passo 3 da lista de testes.

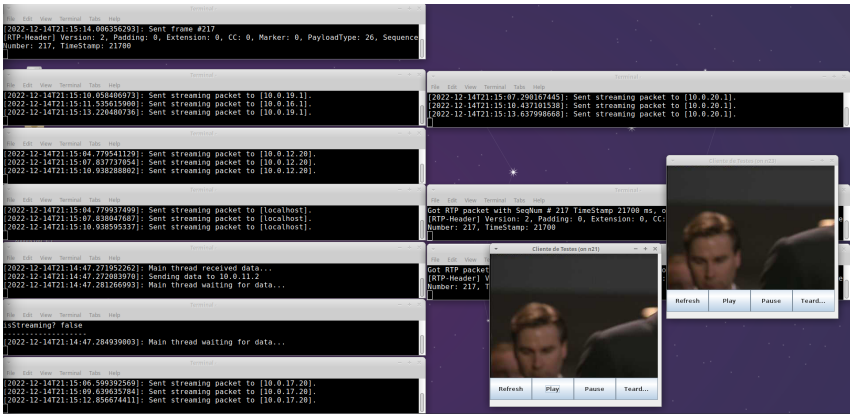


Figura 10: Passo 4 da lista de testes.

```
[
    {
        "ID": "O1",
        "Critical": true,
        "Origin": [
            "i0:10.0.20.10"
        ],
        "Neighbours": ["O2:i0"]
    },
    {
        "ID": "O2",
        "Critical": true,
        "Origin": [
            "i0:10.0.20.1",
            "i1:10.0.19.2",
            "i2:10.0.18.2"
        ],
        "Neighbours": ["O3:i0", "O7:i0"]
    },
    {
        "ID": "O3",
        "Critical": true,
        "Origin": [
            "i0:10.0.19.1",
            "i1:10.0.15.2",
            "i2:10.0.11.2"
        ],
        "Neighbours": ["O2:i1"]
    },
    {
        "ID": "O7",
        "Critical": true,
        "Origin": [
            "i0:10.0.16.1",
            "i1:10.0.13.2",
            "i2:10.0.9.2",
            "i3:10.0.7.2",
            "i4:10.0.17.1"
        ],
        "Neighbours": ["O2:i2"]
    }
]
```

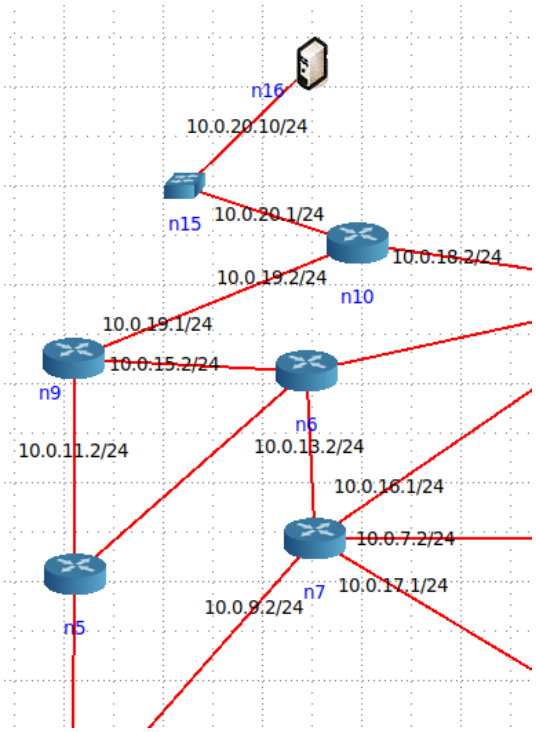


Figura 3: Configuração de topologia minimalista.