Relatório Trabalho 2 - ESR

Ana Catarina Gonçalves[pg50180] and Francisco Toldy[pg50379]

and Pedro Araújo[pg50684]

Mestrado Engenharia Informática
Universidade do Minho,Braga

Grupo 80

Abstract. O presente documento relata o trabalho realizado no âmbito da unidade curricular de Engenharia de Serviços de Rede. O trabalho tinha como objetivo conceber um protótipo de entrega de vídeo com requisitos de tempo real, recorrendo a um servidor de conteúdos para um conjunto de N clientes. Este objetivo seria alcançado recorrendo ao emulador CORE como bancada de teste e a uma topologia constituída por um conjunto de nós que formariam entre si uma rede overlay aplicacional. A criação e manutenção dessa rede deveria estar otimizada para a entrega de conteúdos de forma eficiente.

Neste documento iremos abordar a arquitetura da solução final entregue, a especificação dos protocolos usados, passando depois para a descrição da implementação e terminando nos testes e conclusão.

1 Arquitetura da Solução

Neste capítulo iremos abordar a arquitetura da solução final, utilizando como guia as várias etapas apresentadas no enunciado fornecido pelos docentes.

1.1 Construção da Topologia Overlay

A construção da topologia overlay seguiu a estratégia 2. Dependendo do tipo de nó que desejamos ter na topologia pode ser um dos seguintes:

- Servidor - é executado com os argumentos: **`server <ip_servidor> <ficheiro_de_configuração_servidor>`**
- Nodo - é executado com os argumentos: **`node <ip_nodo> <ficheiro_de_configuração_nodo>`**
- Cliente - é executado com os argumentos: **`client <ip_cliente> <ip_nodo>`**

Além destes 3 tipos de utilizadores da aplicação, ainda deverá existir um servidor master, que consiga partilhar com os nodos os seus vizinhos. Sendo que este é executado com: **`server <ip_servidor> <ficheiro_de_configuração_servidor> master`**

Todos estes tipos de nodos, exceto os clientes, executam com o ficheiro de configuração. No caso dos servidores, o ficheiro de configuração tem como conteúdo o seguinte:

```
{
  "servers":["ip_server2",...],
  "neighbours":{
    "ip_nodo1":["ip_vizinho1",...],
    ...
  }
}
```

Este ficheiro tem armazenada a informação da topologia inteira, em que cada nodo está associado aos seus vizinhos como também uma lista de servidores exceto o master. Este ficheiro foi resultado de uma simplificação que ocorreu devido ao problema da existência dse diferentes ips associados a um único nodo. O grupo escolheu uma abordagem de nodos por um único ip, podendo escolher qualquer um, deste que se mantenha para todas as instâncias de nodos.

No caso dos nodos, o ficheiro usado é o **configNode.json**, que tem como conteúdo apenas a lista de IPs de todos os servidores da topologia.

```
{
    "servers":["ip_server1",...]
}
```

1.2 Serviço streaming

No que toca à 2º etapa, o serviço de streaming, foi escolhida a 1º estratégia, sendo utilizada uma versão do código fornecido adaptada às necessidades do protocolo implementado. Mais especificamente, foi retirado por completo o protocolo rtsp, mantendo apenas o rtp que permite a reprodução do vídeo desejado.

1.3 Monitorização da rede overlay

De modo a desenvolver a monotorização, primeiramente tivemos de desenvolver o processo de flooding, para que se pudesse construir as melhores rotas para cada nodo, as quais são fixas no decorrer da aplicação. Após o flooding, cada servidor já vai ter conhecimento das melhores rotas para cada nodo, e consequentemente, vai enviar os pacotes prova ao longo da árvore criada. Finalmente, cada nodo vai fazer comparações entre os pacotes de diferentes servidores, e escolher esse servidor para o streaming nesse nodo. No entanto, à medida que ocorram alterações na topologia em termos de delays, o servidor escolhido anteriormente pode ser trocado por um que tenha um streaming em melhores condições.

1.4 Construção das Rotas de Entregas de Dados

Relativamente à construção de rotas para entregas de dados, foi escolhida a 2a estratégia. Esta consiste no envio de um pedido de streaming de um cliente a um nodo que se segue pelo envio desse pedido pela árvore criada até ao servidor. Ao longo deste caminho, todos os nodos são ativados, se ainda não estiverem e guardam também os clientes ou nodos que estão ativos. O servidor mal receba o pedido de streaming, vai enviar os pacotes de streaming para o primeiro nodo da rota até ao nodo destino. Assim, cada nodo vai saber para que vizinhos/clientes enviar. O cliente vai estar à espera dos pacotes de streaming, e finalmente, quando chegarem, vão ser apresentados numa pequena caixa de vídeo.

1.5 Ativação e Teste do servidor alternativo

Devido ao facto de existir uma monotorização regular, o servidor ótimo em cada nodo pode ser alterado. Logo, nesse momento, se esse nodo estiver ativo, vai enviar uma mensagem de desativação pela árvore até ao servidor antigo, e também vai enviar uma mensagem de ativação pela árvore ao novo servidor ótimo.

2 Especificação dos protocolos

Neste capítulo iremos abordar os protocolos utilizados e as diversas comunicações entre os vários intervenientes da solução final. As mensagens transmitidas entre servidor e nodos recorrem a transmissão UDP.

2.1 Partilha de Vizinhos

O processo de transmissão da lista de vizinhos de cada nodo é inicializado pelo mesmo. Ou seja, no momento em que um nodo é iniciado, este irá enviar uma mensagem para o servidor Master através da porta 3000. O servidor Master estará à escuta de pedidos por vizinhos dos nodos e vai enviá-los no seguinte formato JSON:

```
{
  'neighbours':["ip_vizinho1",...]
}
```

2.2 Notificação de Servidores a partir do Master

De modo a não enviar dados redundantes pela rede ao enviarmos os vizinhos de cada nodo a partir de todos os servidores, apenas o master vai receber os pedidos de vizinhos dos nodos, notificando os outros servidores de que podem iniciar o seu processo de Flooding enviando uma pequena mensagem para a porta 3000:

```
  "start"
```

2.3 Flooding

Após serem transmitidos os vizinhos para todos os Nodes da topologia, o servidor inicia o processo de flooding. O flooding irá transmitir para todos os vizinhos do servidor na porta 3000 uma mensagem UDP com o seguinte conteúdo:

```
{
    "server":<ip_servidor>,
    "from":<ip_do_nodo_que_enviou_pacote>,
    "depth":0,
    "startTime":time.time(),
    "totalDelay":0,
    "route":[<ip_servidor>]
}
```

O flooding será retransmitido por cada nodo, que irá processar cada mensagem de flood recebida, atualizando a sua informação (se necessário) da melhor rota até si próprio (através da comparação de delay) antes de continuar o flood. Essa continuação do flood exclui o vizinho que enviou o flooding e só acontecerá se a rota até si próprio alterou.

Este processo de flooding é finalizado quando um nodo não recebe nenhum pacote por 3 segundos, despoletando o envio para cada um dos servidores da sua melhor rota até esse servidor que vai estar a espera de mensagens na porta 4000.

2.4 Monitorização

A monitorização é feita recorrendo ao envio periódico de mensagens pela árvore construída na porta 4000:

```
  {
    "server":<ip_servidor>,
    "depth":<depth_da_rota>,
    "startTime":time.time(),
    "totalDelay":0,
    "path":<rota_para_nodo>
}
```

Sendo que o elemento mais importante é o path, que vai assegurar que cada nodo tem informação do próximo destino para chegar ao nodo objetivo.

Em cada nodo, é feita a verificação de se a mensagem recebida chegou ao seu destino final ou terá que ser transmitida para outro nodo. Caso não seja necessário retransmitir, é feita uma comparação com a informação previamente armazenada no nodo, sendo esta substituída se necessário.

2.5 Conexão de Cliente à Overlay

Quando um cliente inicia a sua aplicação, vai se conectar ao nodo que desejar, como também vai ser enviado uma mensagem para esse nodo a notificar desse pedido que vai estar à espera na porta 5000.

```
{
    "request":"connect"
}
```

Cada nodo vai estar à espera de pedidos de conexão, e que mal receba vai se ativar a si próprio e enviar pela árvore até ao servidor ótimo uma mensagem de ativação. No entanto, se esse nodo já estiver ativo, apenas vai adicionar esse cliente a uma lista para a transmissão dos pacotes de streaming.

2.6 Ativação de Rotas

A ativação de rotas pode acontecer em 2 diferentes situações, podendo ser:

- Cliente se conectou e nodo ainda não está ativo
- Servidor ótimo foi substituído

Esta ativação passa pelo envio da mensagem pela árvore até ao servidor.

```
{
    "route":<rota_ate_servidor>,
    "type":"active",
    "nodeActive":<ip_nodo>
}
```

Cada nodo vai estar à escuta por estas mensagens na porta 6000, porém, na passagem pelos nodos da árvore, esses também passaram a estar ativos e a guardar os IPs que vão futuramente precisar de enviar os pacotes de streaming.

2.7 Desativação de Rotas

A desativação de rotas acontece quando:

- Cliente escolhe não receber mais a stream
- Servidor ótimo foi substituído

Em semelhança à ativação de rotas, vai ser enviada a mensagem:

```
{
    "route":<rota_ate_servidor>,
    "type":"diactive",
    "nodeActive":<ip_nodo>
}
```

Cada nodo vai estar a escuta destas mensagens, também na porta 6000, e se não tiver mais nenhum nodo/cliente ativo, vai retransmitir esta mensagem para o seguinte nodo no caminho até ao servidor, de modo a evitar tráfego desnecessário.

3 Implementação

A solução final foi desenvolvida em Python, sendo que está dividida em várias classes. A classe principal sendo a oNode, executada por todos os utilizadores do sistema que vai iniciar a classe indicada dependendo do tipo de nodo a ser criado.

Em termos de implementação, uma decisão feita pelo grupo foi a forma como seriam atendidos pedidos diferentes. Em oposto a uma abordagem com uma única socket, foi decidido a criação de várias sockets para diferentes propósitos associadas a diferentes portas.

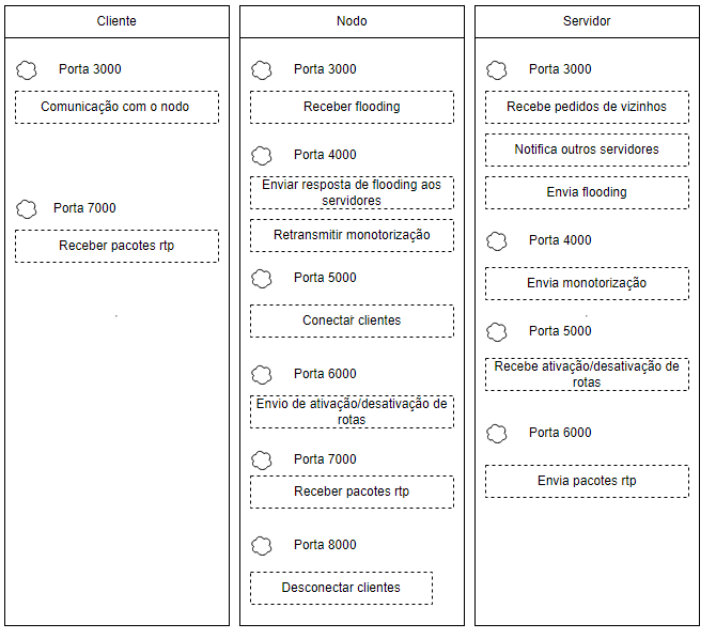

Figura 1: Tabela do uso de sockets

4 Testes e Resultados

Ao longo do desenvolvimento do projeto foram desenvolvidas várias topologias teste, partindo das mais simples para as mais complexas. Foi escolhida uma única para tirar conclusões, no entanto, a que o grupo escolheu vai permitir a testagem de todas as funcionalidades desejadas:

- Inexistência de ciclos no flooding e monotorização
- Ativação de rotas dependendo dos clientes ativos
- Transmissão de streaming correta para vários clientes
- Troca do servidor transmissor da stream se tiver condições menos favoráveis
-

Além disto, de modo a facilitar esta testagem, foi criado um script que permite facilmente iniciar cada um dos nodos indicando apenas o seu nome: **`start.sh <nome_nodo>`** criada especialmente para esta topologia.

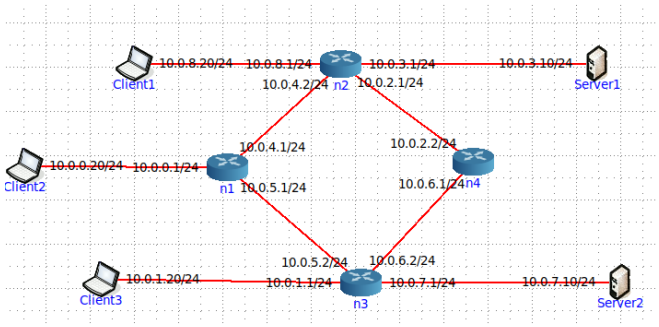

Figura 2: Topologia escolhida

Após a correr a aplicação em cada um dos nodos, e conectar os clientes ao seu nodo de conexão, é possível obter a stream em cada um dos clientes

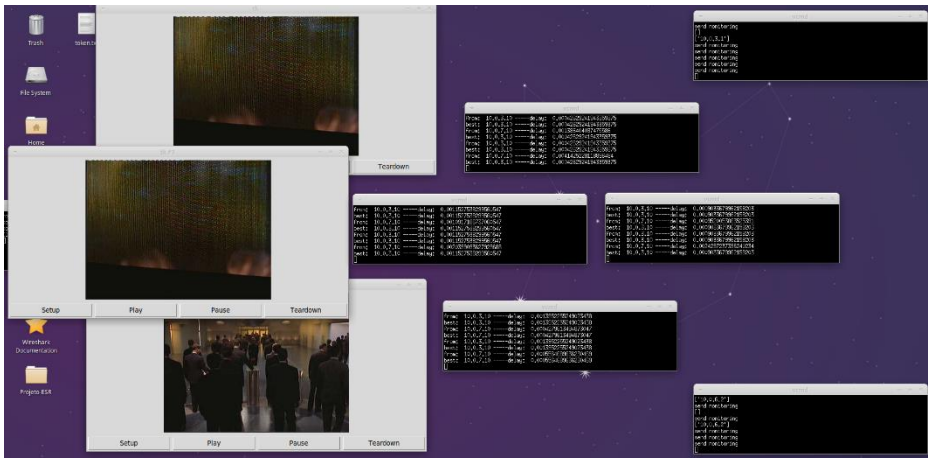

Figura 3: Verificação de 3 clientes a receberem o video

De modo a facilitar a observação de trocas de servidor, não foi feita uma sincronização da stream entre servidores. No entanto, para confirmar este facto, vamos inserir um delay sobre o servidor 2.

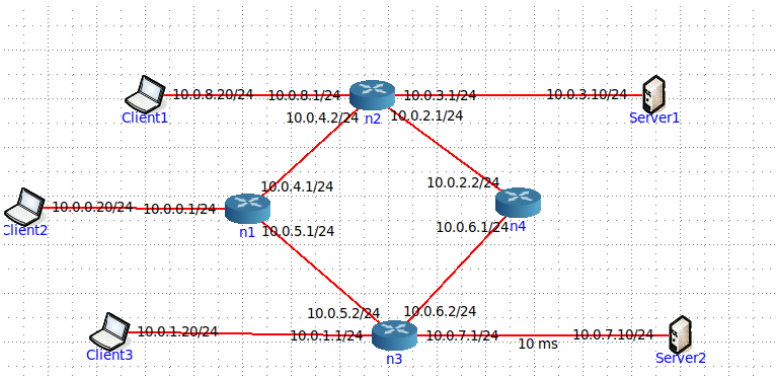

Figura 4: Adição de delay no servidor 2

Assim verificamos que os videos dos clientes passam a estar sincronizados, mas podendo ser apenas uma coincidência, é necessário verificar o tráfego na saída de cada servidor:

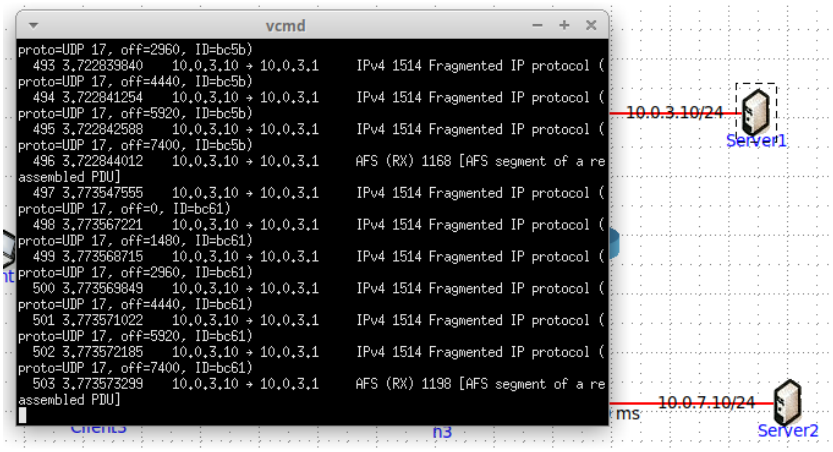

Figura 5: Envio do filme no servidor 1

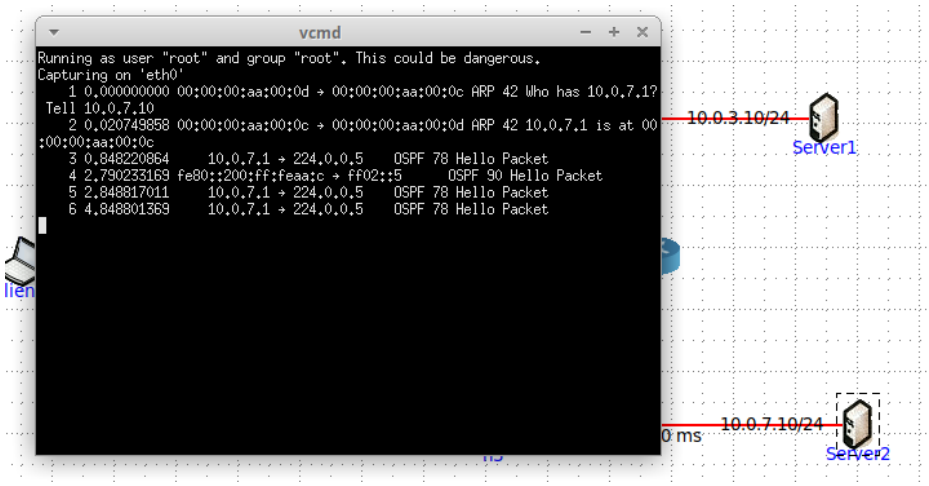

Figura 6: Servidor 2 sem nodos ativos

Devido ao delay do servidor 2 o cliente 3 já está a receber a stream também do servidor 1

Podemos também verificar que não está a ser realizado um envio de stream em broadcast, visto que só pelos caminhos mais rápidos é que o tráfego vai ser transportado.

Podemos confirmar isto na forma em que nenhum dos clientes recebe tráfego que passa pelo nodo 4:

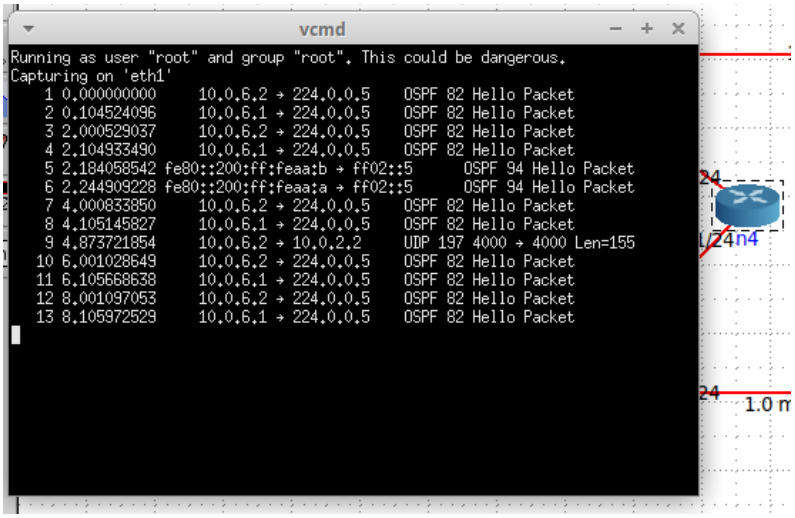

Figura 7: Nodo 4 não pertence a nenhuma rota dos clientes

5 Conclusões e Trabalho Futuro

A realização deste trabalho prático permitiu ao grupo consolidar e alguns conceitos lecionados na cadeira Engenharia de Serviços em Rede. Fomos capazes de aprofundar o conhecimento acerca dos protocolos e serviços de streaming, sendo tanto multimédia como multicast.

Ao decorrer do desenvolvimento do projeto, foram encontradas algumas dificuldades, vale realçar a etapa de streaming, visto que foram necessárias algumas alterações ao código, e consequentemente levou a um estudo mais profundo do mesmo, mas que foi superado e o grupo atingiu a solução desejada.

Portanto, o grupo considera ter realizado um bom trabalho tendo implementados os requisitos propostos no enunciado. Contudo, há possíveis melhorias para trabalho futuro a serem realizadas, referente a implementação de um método de recuperação caso ocorrer falhas durante o processo, visto que utilizamos como principal protocolo de transporte o UDP, seria adequado haver métodos que garantissem a fiabilidade da conexão.