Engenharia de Serviços em Rede
Trabalho Prático 2

Celso Rodrigues[1,2,5], Nuno Gonçalo Rodrigues[1,3,5], and Rui Braga[1,4,5]

[1] Departamento de Informática, Universidade do Minho
[2] a83655@alunos.uminho.pt
[3] pg50667@alunos.uminho.pt
[4] pg50743@alunos.uminho.pt
[5] Grupo 52

Abstract. Recentemente, tem-se observado uma mudança do paradigma da internet. A comunicação *point-to-point*, deu lugar ao consumo incessante e insaciável de conteúdo. Este novo paradigma impõe grandes desafios à infraestrutura *IP* que a suporta.
Serviços populares como a Netflix ou Hulu, fazem *streaming* sobre a rede *IP* pública, surgindo daí a designação de *"Over-the-top streaming services"*. Para o conseguirem, formam uma rede *overlay* própria, que assenta em cima de protocolos de transporte ou aplicacionais da Internet. Um serviço de multimedia *OTT* pode usar uma rede *overlay* aplicacional, devidamente configurada e mantida para contornar a congestão e limitação de recursos da rede de suporte, entregando em tempo real e sem perda de qualidade, contéudo ao cliente final.

Keywords: serviço de *streaming*, redes *overlay*, *over-the-top*

1 Introdução

Este trabaho tem como objetivo a conceção e prototipagem de um serviço de entrega de conteúdo (áudio/vídeo/texto), que seja capaz de o entregar em tempo real, para N clientes a partir de um servidor.

Com esse fim, um conjunto de nós podem servir como intermediários para o reenvio de dados, formando assim, 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 largura de banda necessária.

A forma como o *overlay* aplicacional se comporta é determinante para a qualidade de serviço e, consequentemente, proporcionar a melhor experiência possível ao utilizador.

Para conseguir esse objetivo, pretende-se usar como local de teste principal o emulador CORE, com diversas topologias de teste.

2 Arquitetura da Solução

A solução que propomos para implementar este serviço de *streaming*, assenta sobre tabelas de vetores de distância. Ou seja, cada nodo da topologia *overlay* sabe qual é a melhor *interface* a seguir para cada outro nodo da topologia e comunica-o aos seus vizinho.

Para fazer essa tabela de vetores de distância, cada nodo guarda sobre todos os outros nodos da topologia, qual é a interface a seguir até ele e qual o nodo a que ela pertence, quantos saltos e quanto estimado até ao nodo desejado.

Esta abordagem tem óbvias vantagens e desvantagens. Do lado das desvantagens, temos a demora para a rede convergir e estabilizar na fase inicial de execução, existe o risco de ocorrerem loops infinitos na partilha de certas mensagens (devíamos ter adicionado TTL a certas mensagens) e, principalmente, pode haver demasiado tráfego na rede devido a eventuais *floods* que possam ocorrer, que aliás, foi, talvez, o principal desafio com esta abordagem, mitigar a quantidade de tráfego na rede. Pelo lado positivo, exceto na fase inicial, a rede é normalmente, bastante estável, a capacidade computacional necessária para calcular rotas é irrisória e acima de tudo, é fácil de implementar e administrar.

Para além da tabela de distâncias de vetores, cada nodo da topologia *overlay* mantém também informação sobre os seus vizinhos, em quais interfaces eles estão disponíveis e qual o seu estado (se estão ligados ou não). Cada nodo sabe também quais os servidores que existem na topologia, informação essa que recebem através das mensagens de monitorização e quais as rotas de *streams* que passam por ele próprio.

No exemplo abaixo, consegue-se perceber mais explicitamente a informação que cada nodo guarda:

```
ESTADO

- Vertex:
Node: O1
State: ON
Available at: 192.168.56.101
Adjacents: O2 O3
Adjacents' State:
O2: ON
O3: ON

- Table:

Destination: O2
Via Node: O2
Via Interface: /192.168.56.102
Hops: 1
Cost: 4935000
```

```
Destination: O3
Via Node: O3
Via Interface: /192.168.56.103
Hops: 1
Cost: 360000

Destination: O6
Via Node: O2
Via Interface: /192.168.56.102
Hops: 2
Cost: 25555679

- Servers: O1

- Streams:
STREAM 1:
STREAM ID: 101
RECEIVING STREAM: O6
GOING THROUGH: O1 O2
ACTIVE: TRUE
CHANGE OCURRED: FALSE
```

Esta solução prima pela sua simplicidade, pois, por exemplo, caso um cliente deseje pedir uma *stream*, basta averiguar qual dos servidores está mais próximo e faz seguir o pedido de *stream* por essa *interface* até chegar até ao servidor desejado ou a um nodo que esteja capacitado para reproduzir a *stream* desse servidor ou, quando recebe um pacote do conteúdo a reproduzir, basta ver qual é a rota a que pertence e encaminhar para o nodo seguinte.

Então, todas as mensagens entre os nodos da topologia que estão relacionadas com o estado da topologia são enviadas por TCP, por conexões que são abertas e imediamente fechadas no momento em que a mensagem acaba de ser processada para evitar gastar recursos desnecessariamente, enquanto que o envio de pacotes de conteúdo do vídeo é feito por UDP.

No início da execução, o *bootstrapper* construi a topologia *overlay* através de um ficheiro de configuração previamente construído e cada nodo liga-se à topologia através do *bootstrapper* que lhe envia os seus vizinhos. Em baixo está um exemplo do ficheiro de configuração:

```
<overlay>
    <nodes>
        <entry n="O1">
            <address>192.168.56.101</address>
        </entry>
        <entry n="O2">
            <address>192.168.56.102</address>
        </entry>
```

```
        <entry n="O3">
            <address>192.168.56.103</address>
        </entry>
        <entry n="O4">
            <address>192.168.56.104</address>
        </entry>
    </nodes>
    <bootstrapper name="O1"/>
    <node n="O1">
        <adj>O2</adj>
    </node>
    <node n="O2">
        <adj>O1</adj>
        <adj>O3</adj>
        <adj>O4</adj>
    </node>
    <node n="O3">
        <adj>O2</adj>
    </node>
    <node n="O4">
        <adj>O2</adj>
    </node>
</overlay>
```

3 Especificação dos Protocolos

3.1 Formato das Mensagens Protocolares

- Pacote RTP (modificado)
  Pacote que remove os campos *version*, *padding*, *extension*, *cc*, *marker* e *PayloadType* e que adiciona o identificador da *stream* na qual ele será encaminhado
- HELLO

  ```
  hello
  ```

- BOOTSTRAPPER INITIAL MESSAGE

  ```
  YOU: <nome do nodo>
  You're available at: <ip1 em que o nodo está disponível>
  You're available at: <ip2 em que o nodo está disponível>
  ...
  ADJ: <nome do adjacente>: <estado do adjacente (ON / OFF)>
  Available at: <ip em que o adjacente está disponível>
  ...
  end
  ```

– PROBE INITIAL

```
probe: initial: <timestamp>
```

– PROBE REGULAR

```
probe: regular: <timestamp>
```

– NEW LINK

```
new link: <destinatário>
from: <nodo que envia>
hops: <número de hops do nodo que envia até ao destinatário>
cost: <tempo estimado do nodo que envia até ao destinatário>
end
```

– FASTEST LINKS

```
fastest links from: <nodo que envia>
link to: <nodo1>
hops: <número de hops do nodo1 que envia até ao destinatário>
cost: <tempo estimado do nodo1 que envia até ao destinatário>
link done
link to: <nodo2>
hops: <número de hops do nodo2 que envia até ao destinatário>
cost: <tempo estimado do nodo2 que envia até ao destinatário>
link done
...
end
```

– CLOSED NODE

```
node closed: <nodo que se desligou>
```

– MONITORING

```
monitoring: <servidor que iniciou o monitoring> <lista de nodos já visitados>
<PROBE REGULAR>
```

– REQUEST LINK

```
? <nodo que se está a pedir>
```

– FIXER NEW LINK

```
fixer new link: <destinatário>
from: <nodo que envia>
hops: <número de hops do nodo que envia até ao destinatário>
cost: <tempo estimado do nodo que envia até ao destinatário>
end
```

– OPEN STREAM CLIENT

```
i want a stream
```

– ASK STREAMING

```
want streaming: <nodo que está a pedir stream>
from server: <servidor específico que o nodo quer>
end
```

– OPEN UDP MIDDLEMAN

```
open UDP middleman: <nodo que pediu stream>
sent to: <nodos que já receberam mensagem (fazem parte da rota da stream)>
end
```

– ACK OPEN UDP MIDDLEMAN

```
ack open UDP middleman: <nodo que pediu stream>
sent to: <nodos que já receberam mensagem (fazem parte da rota da stream)>
end
```

– FIX STREAM

```
fix stream: <identificador da stream>
leading to: <destino da stream>
ordered by: <nodo que ordenou mudança no percurso da stream>
going through: <lista de nodos pelos quais já passou a mensagem (que vão pertencer
                à stream)>
end
```

– ACK FIX STREAM

```
ack fix stream: <identificador da stream>
leading to: <destino da stream>
ordered by: <nodo que ordenou mudança no percurso da stream>
going through: <lista de nodos pelos quais já passou a mensagem (que vão pertencer
                à stream)>
end
```

– CHANGE STREAM

```
change stream: <identificador da stream>
leading to: <destino da stream>
ordered by: <nodo que ordenou mudança no percurso da stream>
going through: <lista de nodos pelos quais já passou a mensagem (que vão pertencer
                à stream)>
end
```

– ACK CHANGE STREAM

```
ack change stream: <identificador da stream>
leading to: <destino da stream>
ordered by: <nodo que ordenou mudança no percurso da stream>
going through: <lista de nodos pelos quais já passou a mensagem (que vão pertencer
               à stream)>
end
```

– CANCEL STREAM CLIENT

```
cancel stream client
```

– CANCEL STREAM

```
cancel stream <nodos pertencentes à stream pelos quais a mensagem já passou>
```

– STREAM CHANGED COURSE

```
stream changed: <identificador da stream>
leading to: <destino da stream>
going through: <lista de nodos pelos quais já passou a mensagem (que pertenciam
               à stream)>
end
```

– STREAM BROKEN CLIENT

```
stream broken client
```

– CANCEL STREAM CLIENT

```
cancel stream client
```

– CANCEL STREAM

```
cancel stream: <percurso da stream>
```

3.2 Interações

– Ligação à rede *overlay*
Um nodo liga-se à rede overlay enviando ao *bootstrapper* um HELLO, e este responde com uma BOOTSTRAPPER INITIAL MESSAGE que tem informação sobero próprio nodo e sobre os seus adjacentes.
De seguida, envia HELLO aos adjacentes, nodo e adjacentes trocam PROBE INITIAL entre si, testando assim a ligação e adicionam-na à sua tabela de vetores, que é também, enviada aos restantes adjacentes (NEW LINK). Aí, nodo e adjacentes trocam FASTEST LINKS e depois do envio e receção de alguns NEW LINK, o nodo e o resto da rede converge e estabiliza no novo estado.

- Monitorização
  A monitorização parte dos servidores, que periodicamente enviam aos seus adjacentes uma mensagem com esse efeito (MONITORING), mensagem essa que é dispersada pela resto do topologia (cada nodo que não o servidor, envia apenas a um adjacente seu por fase de monitorização, para evitar demasiado tráfego na rede). Esta mensagem de monitorização funciona como batimento cardíaco da rede, pois é a partir desta que se verifica se algum nodo se desligou e se o estado das ligaçoes se alterou, ou por outras palavras, as condições da rede.
  Na eventualidade de se perder um nodo, é enviado um CLOSED NODE ao *bootstrapper* e esse reenvia essa mensagem aos adjacentes do nodo que se fechou. Depois, é feito *flood* na rede deste CLOSED NODE, e cada nodo remove o nodo da sua tabela e averigua se tinha alguma ligação dependente do nodo que se fechou, se assim for é lançado REQUEST LINK para os seus adjacentes, aos que estes respondem com FIXER NEW LINK caso tenham alguma ligação diferente. De resto, se o nodo pertencia a alguma *stream*, essa *stream* é dado como inativa e o nodo que se fechou como ponto de falha (a situação da rota ser reparada está mais abaixo no relatório).

- Pedir *stream*
  Quando um cliente pretende receber conteúdo, é enviado pela parte aplicacional para si próprio um OPEN STREAM CLIENT, vê qual o servidor a que, segundo a tabela de vetores de distância, consegue chegar mais rapidamente e envia um pedido de *stream* por essa interface (ASK STREAMING), que é redirecionada até um servidor ou até um nodo por onde já passe uma *stream* e esteja assim capacitado a redirecionar pacotes para o nodo que fez o pedido.
  No momento em que a mensagem chega ao servidor ou a um nodo capaz de atender o pedido, este envia um aviso para preparar um intermediário de UDP, com o identificador da futura *stream* através de um OPEN UDP MIDDLEMAN, que vai até ao nodo que pretende receber *stream* e este envia pela mesma rota por onde passou a mensagem que recebeu, um ACK OPEN UDP MIDDLEMAN, que dá o sinal para os restantes nodos para ativarem a rota.

- Reparar/Mudar rota de *stream*
  Na eventualidade de aparecer alguma ligação melhor que as atuais até ao nodo que está a pedir *stream* ou alguma ligação que "ressuscite" uma rota desligada devido a falha, o nodo que o deteta envia pela nova ligação que encontrou um CHANGE STREAM ou um FIX STREAM, respetivamente. Como acontece no pedido de *stream*, essas mensagens vão até ao destino da rota "colecionando" a nova rota até lá chegar, e é enviado um *ack*, neste caso ACK CHANGE STREAM ou ACK FIX STREAM e nesta altura, ativa-se a rota com as mudanças no percurso.

- Desligar *stream*

Se o utilizador pretender desligar a stream, é enviada pela parte aplicacional de *streaming* do nodo da topologia um CANCEL STREAM CLIENT para a parte aplicacional de controlo do *overlay* do próprio nodo que elimina essa rota e envia um CANCEL STREAM pelo percurso da rota, para os restantes nodos fazerem o mesmo.

– Mudar de servidor
Na eventualidade do cliente não receber os pacotes que pretende num certo período de tempo, a parte aplicacional de *streaming* envia à parte aplicacional de controlo do *overlay* do próprio nodo um STREAM BROKEN CLIENT, e é enviado CANCEL STREAM pela rota existente e ASK STREAMING por uma eventual melhor rota, sendo que o processa que se inicia equivale ao pedido de *streaming* já explicado acima.
Esta mensagem só é enviada a cada 2 segundos se voltar a ocorrer, para evitar causar tráfego desnecessário na rede.

4 Implementação

A implementação foi feita em JAVA. As classes NodeManager e OTTStreaming servem de parte aplicacional para a parte de controlo da topologia num nodo e para o *streaming*, respetivamente.

O estado de cada nodo é construído pela classe NodeState tem uma DistancesTable, esta constituída por NodeLink que são uma abstração do estado dos vetores de distâncias para cada nodo, e o estado tem também uma lista de StreamLink, que representa cada *stream* que passa e passou por um determinado nodo.

De resto, existem *packages* para TCP e UDP.

O *package* de UDP tem o UDPMiddleman que cada nodo mantém aberto caso tenha alguma *stream* ativa e que direciona os pacotes UDP na sua respetiva rota e o UDPServer que cada servidor mantém a correr.

O *package* de TCP tem o TCPHandler que processa as mensagens recebidas, altera o estado do nodo em conformidade com elas e cria *threads* para comunicar com outros nodos. O TCPCommunicator é uma classe que é usada em *threads*, serve só para enviar mensagens entre nodos. É chamada pelo TCPHandler através de um *int* que define o comportamento que vai ter, o estado do nodo e alguma informação extra que seja necessária.

5 Testes e Resultados

5.1 Teste 1

Neste cenário procuramos testar a capacidade de uma *stream* dependente de outra se recuperar.

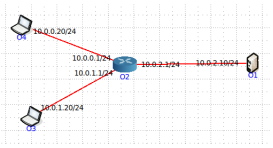

Fig. 1. Topologia de Teste 1

Para isso, temos esta simples topologia, o que fazemos neste caso é ativar a *stream* pelo nodo O3 e depois criar o pelo nodo 04. Isso resulta numa *stream* para o 03 desde o servidor (id: 101) e numa desde o intermediárop 02 até ao O4 (id: 202).

De seguida desativa-se a *stream* 101, o que deixa a *202* desamparada, pois inicia-se no O2 que não é servidor. O nodo 04 ao aperceber-se que não recebe, pede nova *stream* (id: 102), que desta vez inicia-se diretamente do servidor, pois o nodo 02 deixou de ter a rota 101 a passar nele.

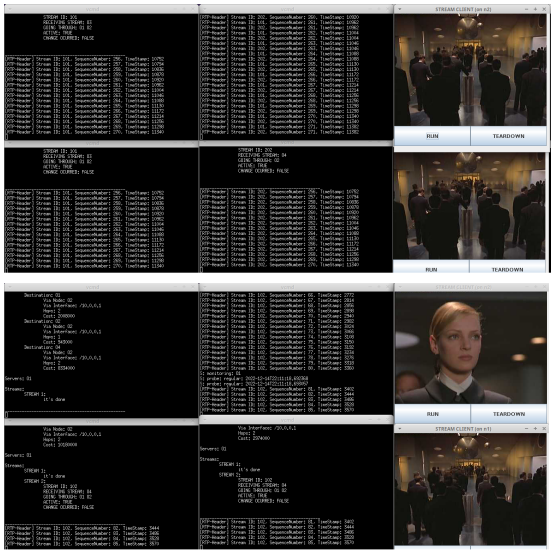

Fig. 2. Resultados do Cenário de Teste 1

5.2 Teste 2

Neste cenário procuramos testar a capacidade de uma *stream* mudar de rota.

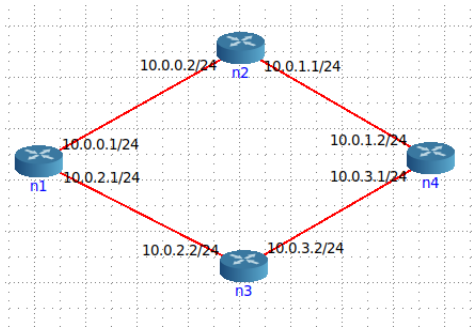

Fig. 3. Cenário de Teste 2

Ao ligar o servidor no nodo n4 da figura e o cliente no nodo n1, pedindo *stream* e adicionando *delay* na ligação entre n3 e n4, naturalmente, é preferível enviar para o n1 através de n2.

No entanto, se removermos n2 ou adicionarmos um atraso ainda maior na ligação entre n2 e n4, verificamos que n4 envia para o nodo n1 através de n3.

6 Conclusões e Trabalho Futuro

Terminado o trabalho, o grupo sente-se agradado com a *performance* que teve diante do problema proposto e de poder apresentar uma solução que implementa tudo o que foi pedido no enunciado, com algumas funcionalidades extra.

Apesar disso, resta salientar que existem alguns aspetos negativos, a solução concebida não é a mais arrojada por razões já proferidas ao longo deste relatório e existem várias arestas por limar, principalmente na organização e qualidade de código, mas também em eventuais *bugs* que possam surgir.