Universidade do Minho
Escola de Engenharia

Universidade do Minho

Mestrado em Engenharia de Telecomunicações e Informática

Serviços de Rede e Aplicações Multimédia

TP2 - Rede Aplicacional de Streaming Live

Relatório

João Carlos Teixeira Cerca da Cunha - PG47301

José Pedro Marques Macedo - PG47378

Luis Miguel Osório Torres Pereira De Carvalho - A79620

28 de julho de 2022

Índice

Índice de Imagens

1. Introdução

O mundo atual requer a troca de informação a velocidades cada vez mais elevadas. O conteúdo transmitido continua a aumentar o tamanho, e as comunicações utilizadas a aumentar a complexidade. A solução "simples"para responder a estas necessidades exige um aumento considerável dos recursos, a única aversidade é que não existem possibilidades para este aumento, pelo menos na camada física das redes.

A solução explorada neste documento procura responder unicamente aos requisitos da distribuição de conteúdo multimédia ao vivo. É realizada com o uso de uma camada intermédia entre a aplicação e a camada de transporte, isto é, acrescenta flexibilidade com o uso de software ao invés de um aumento de recursos físicos. A rede construída aqui aponta para uma redução do *end-to-end delay*.

2. Desenvolvimento

A solução que optamos por implementar foi desenvolvida em ambiente virtual com a distribuição Python 3.10.5, não foram utilizadas quaisquer bibliotecas exteriores às incluídas na mesma. c Uma secção com a discussão das estratégias escolhidas, as opções tomadas e os mecanismos adotados, incluindo eventuais otimizações;

2.1. Abordagem

Inicialmente começamos por dividir os componentes e as eventuais operações que cada um deveria efetuar e resumir a diferentes atividades básicas, assim diminuindo a complexidade para um grau mais acessível, foi estudado separadamente um sistema de construção da topologia, um protocolo comunicacional de pedidos e um programa de *streaming* simples.

Para que a topologia fosse inicializada era necessário o contacto direto com o *Service Controler (SC)*, numa troca de mensagens era transmitida a informação necessária para que os restantes elementos se integrassem na rede. Após a formação da rede overlay, cabe aos clientes a realização de pedidos, desconstruimos este protocolo para uma simples mensagem de start para os elementos vizinhos dados pelo service controler, que nesta fase ainda passam por ser uma versão rudimentar de servidores de streaming, que apenas aguardam a mensagem de start e iniciam a transmissão. Com isto conseguimos sintetizar duas grandes fases do projeto na sua forma mais básica, a construção da topologia e o *streaming*. A partir deste ponto construimos o foco do sistema, o protocolo de comunicação que conseguisse alcançar os melhores resultados de *end-to-end delay*.

O restante da elaboração passou por melhorias de qualidade de vida das diferentes partes do sistema.

2.2. Prototipagem do Sistema

Controlador de Serviço (SC)

Começamos por implementar o Controlador de Serviço que tem como objetivo iniciar o sistema, controlar a topologia utilizada, e configurar e catalogar as streams de dados disponíveis no serviço. O controlador de serviço indica também aos elementos de relay e edge quais os vizinhos com quem pode comunicar. Este elemento apenas responde a pedidos de informação dos restantes nós, pelo que nunca toma a iniciativa de comunicação.

Servidor de Streaming

De seguida foi implementado o Servidor de Streaming. Este elemento é responsável por gerar as streams multimédia. Cada stream é identificada por uma tag gerada aleatoriamente, um nome para identificação pelo utilizador, um ritmo de *frames* por segundo, um valor temporal máximo de atraso e um identificador de tipo de dados multimédia. Cada servidor envia as frames das suas streams quando são explicitamente pedidas pelos elementos relay vizinhos. A transmissão é terminada quando o servidor recebe um pedido de cancelamento dos elementos de relay.

Elementos de relay

Estes elementos existem com o propósito de retransmitir as streams ao longo da topologia de overlay. Ao arrancar, estes elementos devem contactar o SC para saber quais são os seus vizinhos. O SC responde com a informação dos vizinhos autorizados. Estes elementos devem manter a conexão com os vizinhos e o SC através dos beacons. Os relays não comunicam diretamente com os clientes aplicacionais. Mantêm também uma tabela com os pedidos de retransmissão de streams recebidos, mas ainda não ativas e das retransmissões ativas.

Elementos de edge

Os elementos de Edge permitem a retransmissão das streams entre os elementos relay e as aplicações clientes.

Elementos clientes

Estes elementos têm como principal funcionalidade permitir ao utilizador escolher uma stream para produzir de entre a lista fornecida. Como mencionado antes, a lista é obtida diretamente do SC, no entanto a stream é recebida de um elemento edge.

2.3. Otimizações

As primeiras melhorias aplicadas foram a nível de organização do código para que se tornasse mais modular e de melhor interpretação consequentemente. Implementamos 3 objetos, *Request*, *Stream* e *Node* (*figuras 1, 2 e 3* respetivamente), assim qualquer elemento que receba informação de um destes objetos consegue rapidamente agrupa-la sob uma entidade, exploramos os constituintes de cada na próxima secção. Seguindo a ideologia sistemática dos componentes, convertemos o protocolo de comunicação numa biblioteca que possa ser utilizada pelos elementos de Cliente, Edge, Relay e Streaming Server, esta consegue resolver qualquer mensagem recebida sob o protocolo e aplicar o algoritmo de flooding, termina o funcionamento com indicação ao elemento do resultado da mensagem.

```
class Request(object):
    """docstring for Request"""
    def __init__(self, request_id, stream_id, state, element, ttl = 0):
        super(Request, self).__init__()
        self.request_id = request_id
        self.stream_id = stream_id
        self.state = state
        self.element = element
        self.ttl = ttl
```

Figura 1: Objeto Request, coleção de propriedades de um pedido, integra o algoritmo de flooding para a gestão dos pedidos.

```
class Stream(object):
    """docstring for Stream"""
    """ This represents the metadata of a stream"""
    newtag = itertools.count(start=1, step=1 )
    def __init__(self, name, fps, max_delay, type_of_data, tag = next(newtag)):
        super(Stream, self).__init__()
        self.tag = tag
        self.name = name
        self.fps = fps
        self.max_delay = max_delay
        self.type_of_data = type_of_data
```

Figura 2: Representação do objeto Stream, importante para a sintetizar os metadados de cada uma das streams.

```
class Node(object):
    """docstring for Node"""
    def __init__(self, tag, type, ip, state = False):
        super(Node, self).__init__()
        self.tag = tag
        self.type = type
        self.ip = ip
        self.neighbours_list = []
        self.state = state
```

Figura 3: Objeto representativo dos nós do sistema, principalmente utilizado pelo *Service Controler*

Houve uma alteração gradual para um funcionamento *multi-threaded* com o subsistema de controlo para que possa corretamente avisar os vizinhos da saída e encerrar, o programa de streaming realiza as suas funcionalidades sem qualquer bloqueio do resto e a receção de pedidos também ocorre separadamente para que seja dada alguma prioridade à não obstrução.

Também realizamos algumas otimizações na gestão de pedidos, inicialmente realizada por *arrays* (listas no *Python*) também foi reestruturada para uma *hash table* (dicionário [*Key:Value*]) que permite acessos mais rápidos com o uso das *tags* como *keys* e os objetos *Request* como *values*. Reorganizamos o algoritmo de flooding para retirar mais proveito de acessos diretos ao dicionário de pedidos, excluindo a necessidade de algumas iterações.

3. Produto

Idealizamos uma utilização para demonstrar o funcionamento prático do projeto, uma ferramenta que distribua as horas com precisão até ao segundo de qualquer localização do mundo aquando haja um pedido para tal. O resultado é uma *stream* de 1 Hz, 1 *frame* por segundo de texto, a transmissão de valores temporais permite também perceber o impacto real da implementação e o que é percetível ao utilizador final.

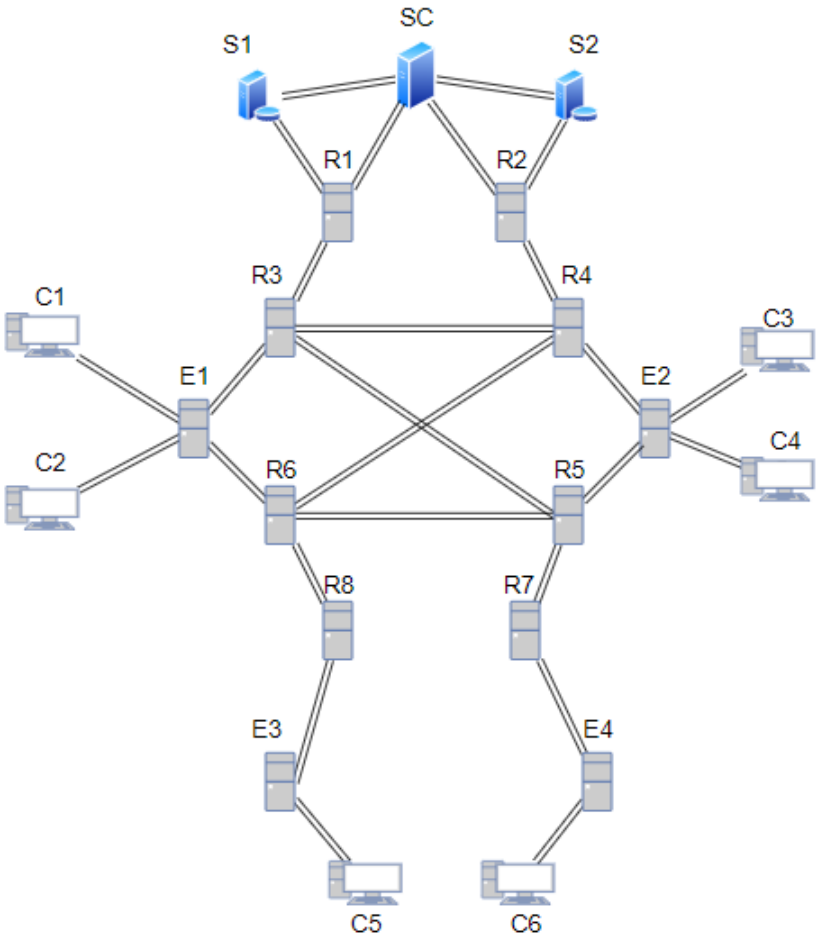

Figura 4: Representação da topologia aplicacional criada a partir da configuração no *Service Controler*

3.1. Funcionalidades

Setup de Rede Aplicacional

Com o conhecimento dos nós que podem entrar em funcionamento na rede o *Service Controler* consegue gerar qualquer tipo de rede aplicacional com o ficheiro de configuração, o sistema controla quais os nós que se encontram ativos de momento para evitar a dispersão de informação para alem da necessária. A rede adapta-se automaticamente a entrada e saída de elementos.

***Streaming* de fuso horário**

Os Servidores de *Streaming* contém uma lista com informação sobre cada *Stream*, e consegue, dado um pedido com a *tag* da stream iniciar uma thread para proceder com o envio. A stream implementada transmite as horas com precisão ao segundo no fuso horário local. A ferramenta opera no principio que os restantes elementos organizem uma boa distribuição dos dados, stream um para muitos.

Algoritmo de *Flooding*

A principal funcionalidade do projeto é o sistema de *flooding* que permite automaticamente selecionar a resposta com menos *End-to-End Delay*. Cada elemento confirma apenas a chegada da primeira *stream*, depois de uma "inundação"de pedidos à rede, o teste em cada operação não acrescenta nenhum atraso significativo e permite operar dinamicamente uma rede para o menor tempo de atraso possível.

Limitações Apesar do controlo aplicado na topologia exitem algumas falhas devido à falta de comunicação constante, um pacote num intervalo de tempo fixo pode resolver alguns problemas em falhas de comunicação extremas.

A ferramenta de *stream* implementada tem pouca versatilidade, não foi explorada a transmissão de outros tipos de *media* (video, som) apesar do sistema base ter sido construido com isso em conta.

O algoritmo de *flooding* implementado ainda traz algumas limitações, nomeadamente no que cabe à escalabilidade, um número maior de clientes causa alguma confusão, nomeadamente devido à escolha de transmissão um para muitos.

4. Implementação e testes da utilização

Os testes realizados focaram-se na realização da rota com menor atraso e com o cancelamento das restantes. Os nossos objetivos seriam testar o funcionamento com todos os nós ativos, no entanto, apenas conseguimos testar com um máximo de 11 nós, nos quais já se obtiveram alguns problemas na divisão da mesma *stream* para dois clientes.

Na figura 5 abaixo é possivel ver o sistema com 8 nós em funcionamento pleno, estes são 1 *Service Controler* (SC), 1 *Streaming Server* (S1), 1 elemento *Edge* (E1), 1 Cliente (C1) e 4 elementos *Relay* (R1, R3, R4, R6). Os terminais encontram-se relativamente próximos com a posição dos elementos onde estão a executar na topologia.

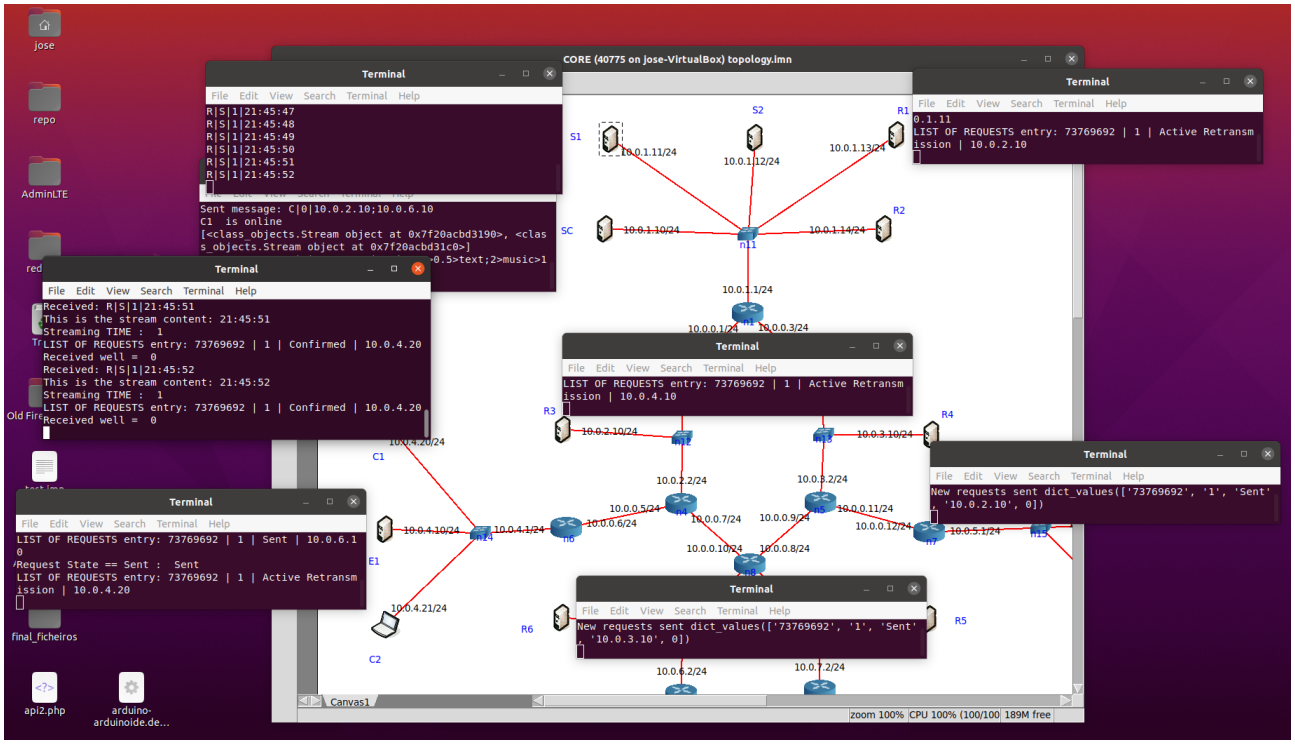

Figura 5: Sistema com uma stream ativa, com parte dos nós da topologia ativados

5. Conclusão

O resultado aqui explorado conta com algumas limitações a nível da implementação, foram priorizados os elementos fulcrais ao funcionamento e a implementação correta do algoritmo, deixando de parte algumas funcionalidades essenciais, principalmente no que toca à classificação quantitativa da *performance* da ferramenta.

No entanto, o objetivo de demonstrar o poder das redes aplicacionais para sistemas multimédia é aparente, a flexibilidade que é conseguida com estes sistemas demonstra que o resultado de uma boa aplicação dos mesmos equipara-se às grandes inovações no campo das telecomunicações e informática no geral, o segredo de conseguir mais serviço por menos recursos.