Trabalho Prático Nº2 – Serviço *Over the Top* para entrega de multimédia

Duração: 7 aulas (8 semanas)

Motivação

Ao longo do último meio século de vida da Internet (a rede das redes), observou-se uma mudança irreversível de paradigma. A comunicação extremo-a-extremo, de sistema final para sistema final, dá lugar ao consumo voraz de conteúdos de qualquer tipo, a todo o instante, em contínuo e muitas vezes em tempo real. Este novo padrão de uso coloca grandes desafios à infraestrutura IP de base que a suporta. Apesar de não ter sido originalmente desenhada com esse requisito, tem sido possível resolver a entrega massiva de conteúdos com redes sofisticadas de entrega de conteúdos (CDNs) e com serviços específicos, desenhados sobre a camada aplicacional, e por isso ditos *Over the Top* (OTT). Um serviço de multimedia OTT, pode, por exemplo, usar uma rede *overlay* aplicacional (ex: *multicast aplicacional*), devidamente configurada e gerida para contornar os problemas de congestão e limitação de recursos da rede de suporte, entregando em tempo real e sem perda de qualidade os *media* diretamente ao cliente final. Serviços bem conhecidos como o Netflix ou o Hulu, fazem *streaming* sobre a rede IP pública. Daí a designação de "*Over-the-top streaming services".* Para tal, formam uma rede *overlay* própria, assente em protocolos de transporte (TCP ou UDP) e/ou aplicacionais (HTTP) da Internet. Neste trabalho, pretende-se conceber e prototipar um desses serviços, que promova a eficiência e a otimização de recursos para melhor qualidade de experiência do utilizador.

Objetivos

Usando primariamente o emulador CORE como bancada de teste, e uma ou mais topologias de teste, pretende-se conceber um protótipo de entrega de áudio/vídeo/texto com requisitos de tempo real, a partir de um servidor de conteúdos para um conjunto de N clientes. Para tal, será eleito um *Rendezvous Point* (RP), que será responsável por receber em unicast o conteúdo a distribuir, propagando-o por um conjunto de nós que atuam como intermediários, formando entre si uma árvore de distribuição partilhada, cuja criação e manutenção deve estar otimizada para a missão de entregar os conteúdos de forma mais eficiente, com o menor atraso e a largura de banda necessária. A forma como o *overlay* aplicacional se constitui e se organiza é determinante para a qualidade de serviço que é capaz de suportar. A Figura 1 ilustra esta visão geral.

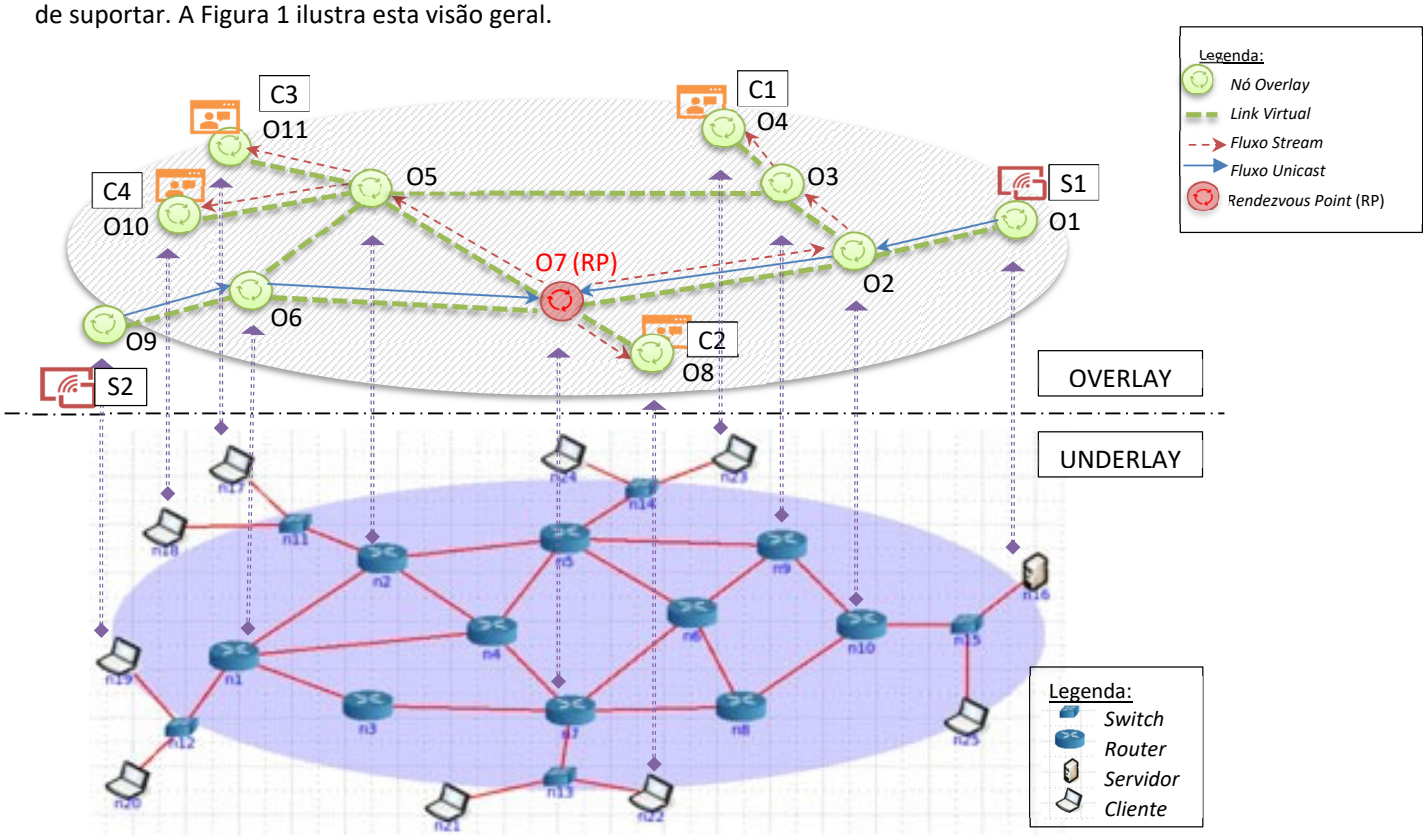


Figura 1 - Visão geral de um serviço OTT sobre uma infraestrutura

proibido o uso não autorizado

Descrição detalhada

Na Figura 1 estão representados quatro clientes (**C1**, **C2**, **C3**, **C4**) com interesse em consumir uma *stream multimédia* que estará disponível no servidor **S1 e/ou**, no servidor **S2**. Não havendo *overlay* aplicacional, cada um dos 4 clientes tem de abrir uma conexão para um dos servidores, e esse servidor tem de gerar 4 fluxos de dados idênticos, um por cada cliente. Esta solução tem problemas óbvios de escalabilidade, pois a partir de um determinado número de clientes N, o servidor ativo e/ou a rede deixam de conseguir suportar os N fluxos sem perda de qualidade. Uma abordagem mais eficiente pode ser conseguida adicionando nós de uma rede *overlay* (**O1**, **O2**, ... **O11**) construída sobre a rede *underlay* de suporte. Os nós da rede *overlay* estão instanciados diretamente em nós da rede *underlay*, pois são na verdade aplicações em execução em determinados sistemas físicos previamente selecionados. Para facilitar o cenário, vamos considerar que inicialmente apenas há um servidor de *streaming* e que esse servidor é também um nó da rede *overlay*. Ou seja, o servidor **S1** da Figura 1, é também o nó **O1** da rede *overlay*, em execução no sistema servidor **n16** da rede de suporte. Por seu lado os clientes de *streaming* (**C1**, **C2**, **C3**, **C4**), responsáveis por receber e reproduzir os conteúdos, são também nós *overlay* (respetivamente **O4**, **O8**, **O10** e **O11)**, instanciados em equipamentos da rede *underlay* (**n23**, **n22**, **n18** e **n17**).

De salientar que a rede *overlay* tem muito menos nós e ligações que a rede de suporte, pois não é imaginável nem desejável que o mapeamento possa ser de um para um. Logo, as ligações na rede *overlay* são na realidade conexões de transporte (TCP ou UDP), que atravessam uma ou mais ligações na rede física. Por exemplo, a ligação virtual entre **O3** e **O5**, vai provavelmente ser suportada pelos links **n9 ←→ n5 ←→ n2** da rede física, se essa for a melhor rota IP de **n9** para **n2** (por defeito calculada com recurso ao protocolo OSPF, no caso do emulador CORE). Na figura os nós da rede *overlay* estão instanciados em todo o tipo de equipamentos, incluindo *routers*, o que sendo possível no emulador CORE não é realista. Numa rede real, os *switches* e *routers* são equipamentos dedicados, que não executam aplicações genéricas. No CORE é possível porque todos os sistemas são *containers* Linux.

Quando um nó da rede *overlay* começa a sua execução, precisa de conhecer quais os nós da rede a quem se ligar, por exemplo, através de um arquivo de configuração. Ou, em alternativa, usar um nó bem conhecido e predeterminado (*bootstrapper ou tracker*), cuja função é ajudar a construir a árvore. Esta árvore terá como raíz o nó eleito como RP.
A construção da árvore de distribuição será feita a partir dos nós folha, através do envio de mensagens de subscrição aos seus vizinhos, que eventualmente chegará ao RP, formando assim cada um dos ramos da árvore.
Uma vez construído o *overlay*, torna-se necessário comparar as alternativas para o fluxo de dados unicast, desde cada um dos servidores até ao RP, supondo que o conteúdo está duplicado em mais de um servidor. Para tal, deverá ser implementada uma métrica baseada em diversos critérios (uma solução simples poderá apenas medir latência e perdas de dados, mas poderão ser implementadas estratégias mais robustas) e que será calculada através do seu próprio protocolo. Esta métrica deverá ser atualizada ao longo do tempo, sendo utilizada como critério de decisão quando vários servidores têm o mesmo conteúdo pedido. Como exemplo, o RP poderá implementar uma tabela com as seguintes informações: *{Fonte:S1; Métrica: 1; Conteúdos: [movie1.mp4, video4.ogg], Estado: ativa }*.

A fim de visualizar um certo conteúdo, o Cliente deverá enviar um pedido ao vizinho mais próximo. Este pedido será propagado pela árvore até alcançar um nó que já esteja a difundir este conteúdo, ou, em último caso, o RP. Ao receber um pedido, o RP deverá avaliar as condições de cada um dos servidores que possua o conteúdo, e selecionar o mais indicado. A transmissão do conteúdo deve ser otimizada para que a largura de banda seja o mais reduzida possível quando existem vários Clientes interessados em visualizar o mesmo conteúdo, usando o conceito do *multicast*. Durante a transmissão do conteúdo, o RP continuará a monitorizar o estado dos servidores, podendo alterar o servidor que está a providenciar este conteúdo.

proibido o uso não autorizado

Etapas sugeridas

Nota: as etapas não precisam ser seguidas obrigatoriamente de forma sequencial!

1. **Etapa 1**: Preparação das atividades
    - Escolher a linguagem de programação mais conveniente para todos os elementos do grupo;
    - Utilizando o CORE, preparar uma topologia para testes e criar cenários de teste nessa topologia (ver cenários sugeridos);
    - Definir qual o protocolo de transporte que vai ser usado no *overlay*: TCP ou UDP (também seria possível equacionar o uso de um protocolo de aplicação como o HTTP, mas talvez seja mais complexo);
    - Implementar um cliente/servidor simples, baseado em exemplos, para o protocolo de suporte escolhido (TCP ou UDP), que seja simultaneamente cliente e servidor;

2. **Etapa 2**: Construção da Topologia *Overlay com Árvore Partilhada*
    - Construir uma aplicação (ex: **oNode**) que é simultaneamente cliente e servidor, que atende numa porta predefinida, e que é capaz de receber e enviar mensagens em modo full-duplex; testar com envio e receção de mensagem simples de "HELLO";
    - Definir uma estratégia de construção da rede *overlay*; como alternativas pode considerar:
    - Estratégia 1: Abordagem manual. Ao executar o programa indicam-se os X vizinhos em configuração ou na linha de comando. Exemplo: **$ oNode <vizinho1> <vizinho2> <vizinho3>**. A tabela de vizinhos e respetivas conexões não se altera durante a execução. Não há necessidade de nenhum nó de controlo.
    - Estratégia 2: Abordagem baseada num ficheiro local. Ao executar o programa, é indicado o caminho onde estará um ficheiro com a informação relativa aos vizinhos deste nó. Cada nó consumirá o seu próprio ficheiro de configuração, sem necessidade de comunicação exterior.
    - Estratégia 3: Abordagem baseada num controlador. Ao executar o programa, indica-se apenas um nó como contacto para arranque da rede. Exemplo: **$ oNode <bootstrapper>**. O novo nó envia mensagem de registo, a identificar-se, e recebe como resposta uma lista de X vizinhos com os quais deverá comunicar e estabelecer conexões.
    - Outras estratégias... (sugerir ou procurar alternativas, discutindo-as com o docente);
    - Manter as ligações entre o RP e servidores ativas, implementando já alguns critérios para determinar a métrica.

3. **Etapa 3**: Serviço de *Streaming*
    - Estratégia 1: Implementar um cliente e um servidor simples com base no código do livro de apoio [1];
        - Usar o código do livro de apoio (disponível em Python e em Java) como ponto de partida;
        - Adaptar o código se a linguagem de programação escolhida não for nem Python ou Java;
        - Com base no exemplo, previamente ajustado e comentado pela equipa docente [2], fazer um servidor capaz de ler o vídeo de um ficheiro e o enviar em pacotes, numerados, para a rede *overlay*;
        - Com base no exemplo, previamente ajustado pela equipa docente, fazer um cliente capaz de receber pacotes da rede *overlay*, com um número de sequência, e reproduzir o vídeo numa janela;
        - Usar como vídeo de teste o exemplo do livro *movie.Mjpeg* [2] (trata-se de um vídeo básico, de fluxo constante, que é uma sequência simples de imagens independentes, a enviar a intervalo de tempo fixo);
    - Estratégia 2: Adaptar o código do livro de apoio para utilizar uma biblioteca alternativa, capaz de ler codecs adicionais, e recorrendo a outros vídeos para difusão.
    - Outras estratégias... (sugerir ou procurar alternativas, discutindo-as com o docente).

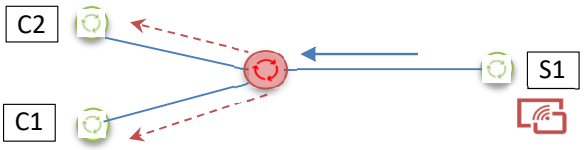
proibido o uso não autorizado

4. **Etapa 4**: Monitorização dos Servidores de conteúdos
   - Para além do que será a árvore de entrega de conteúdos, pretende-se que haja uma troca de mensagens de prova (teste) entre o RP e os Servidores, que irá permitir obter um conhecimento atualizado das condições de entrega *de cada um dos Servidores*.
     Essa mensagem será estruturada com os campos de cabeçalho que o grupo considerar relevantes para o cálculo da métrica, cujo peso e importância deverão ser justificados. Após o cálculo da métrica, o RP estará em condições para escolher o Servidor mais apropriado para a difusão de cada um dos conteúdos.
     Como teste inicial, pode considerar apenas um dos servidores (S1), e utilizando o CORE, definir atrasos de, por exemplo, 50ms por ligação, e verificar as métricas observadas pelo RP. Uma vez testada e operacional, estender a monitorização da rede ao Servidor S2, implementando diferentes condições de rede para cada um dos Servidores para que se confirme o cálculo correto da métrica.

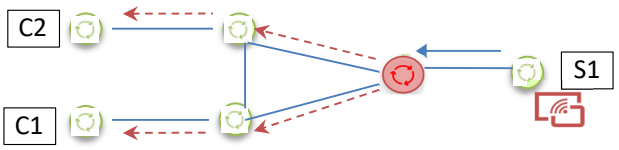
5. **Etapa 5**: Construção dos Fluxos para Entrega de Dados
   - Ao realizar um pedido de conteúdo, o cliente envia a solicitação ao nó mais próximo.
   - Caso o nó mais próximo possua um fluxo do conteúdo, ele disponibiliza o stream para o cliente. Caso contrário deverá propagar o pedido até a um nó do caminho até ao RP que já esteja a transmitir o conteúdo. No limite este nó será o RP.
   - A estratégia a implementar deverá minimizar o tráfego gerado na rede, assim como o número de fluxos de transferência de conteúdos. Garanta que só existem os fluxos de transmissão de conteúdo estritamente necessários. Os fluxos devem ser cancelados se não houver clientes interessados nesse conteúdo nesse ramo da árvore. Para tal, o término de um cliente deve gerar uma mensagem de abandono da árvore partilhada.

6. **Etapa Adicional**: Definição de um método de recuperação de falhas e entradas de nós adicionais
   - Na versão base do projeto assume-se que todos os nós estarão disponíveis desde a inicialização e que a sua composição não se altera. Este não é o caso no mundo real, pelo que podem ser implementadas estratégias de recuperação caso um ou mais nós da árvore partilhada fiquem indisponíveis.
   - Adicionalmente, poderá também ter em consideração que novos nós poderão juntar-se, havendo necessidade de alterar a estrutura da árvore.
   - Outras adições podem ser sugeridas, após discussão com o docente.

7. **Sugestões para cenários de teste:**
   - Cenário 1: *overlay* com 4 nós (um servidor, S1, dois clientes e um RP).



   - Cenário 2: *overlay* com 6 nós (um servidor, dois clientes, dois nós na árvores, e um RP).
     Posteriormente, considerar um servidor adicional S2 conectado ao RP.



   - Cenário 3: *overlay* complexo usando a topologia na Figura 1.

proibido o uso não autorizado

Avaliação

Consideram-se os seguintes itens na avaliação com respetivos pesos relativos:

1. Etapa 1 e Etapa 2 – Preparação das atividades e Construção da Topologia *Overlay com Árvore Partilhada* (10%)
2. Etapa 3 – Serviço de Streaming (10%)
3. Etapa 4 – Monitorização dos Servidores de conteúdos (10%)
4. Etapa 5 – Construção dos Fluxos para Entrega de Dados (20%)
5. Etapa 6 – Definição de um método de recuperação de falhas e entradas de nós adicionais (15%)
6. Solução final integrada (25%)
7. Checkpoints (5% + 5%)

Para cada um dos itens, consideram-se os seguintes subitens:
a. Especificação conceptual
b. Qualidade da implementação
c. Qualidade dos testes e da própria apreciação crítica da solução apresentada

Adicionalmente, na solução final (item 6) consideram-se também os subitens:
d. Qualidade da defesa/apresentação
e. Qualidade do relatório

Referências

1. Kurose, J. (2021). *Computer Networking: A Top-Down Approach* (8th edition.). Pearson. Video Streaming with RTSP and RTP Lab. Retrieved from https://gaia.cs.umass.edu/kurose_ross/programming.php
2. Exemplos do Livro anterior (adaptados): https://marco.uminho.pt/disciplinas/ESR/ProgEx.zip (*wget https://marco.uminho.pt/disciplinas/ESR/ProgEx.zip)*

Relatório

O relatório deve ser escrito em formato de artigo com um máximo de 12 páginas (recomenda-se o uso do formato LNCS - *Lecture Notes in Computer Science*, instruções para autores em http://www.springer.com/computer/lncs?SGWID=0-164-6-793341-0). Deve descrever o essencial da conceção e implementação, devidamente justificado, com a seguinte estrutura recomendada:

1. Introdução
2. Arquitetura da solução
3. Especificação do(s) protocolo(s)
   3.1. formato das mensagens protocolares
   3.2. interações
4. Implementação
   4.1. detalhes, parâmetros, bibliotecas de funções, etc.
5. Limitações da Solução
6. Testes e resultados
7. Conclusões e trabalho futuro

Submissão

Cada grupo deve fazer a submissão (*upload*) do trabalho, usando a opção de "troca de ficheiros" associada ao seu grupo nos "Grupos" do Blackboard (*elearning.uminho.pt*), da seguinte forma:

- **ESR-TP2-PLxy-Rel.pdf** para o relatório
- **ESR-TP2-PLxy-Codigo.zip** ou **ESR-TP2-PLxy-Codigo.rar** para a pasta com o código

em que **x** é o identificador do turno PL e **y** o número do grupo (e.g. ESR-TP2-PL33-Rel.pdf para o relatório TP3 do grupo 3 do PL3).

proibido o uso não autorizado