Sistemas Distribuídos

Video Streaming

Marco Pontes — Nuno Azevedo

up201308000@fc.up.pt — up201306310@fc.up.pt

4 de Janeiro de 2017

Resumo

O objetivo deste projeto é a implementação de um serviço de *streaming* de vídeo. A comunicação entre os vários elementos intervenientes assenta na tecnologia *Ice*[1].

1 Introdução

O *Ice* é uma estrutura de comunicação entre processos (*RPC framework*) que fornece vários conjuntos de ferramentas de desenvolvimento de software. Esta implementa um protocolo de comunicação próprio, *Ice Protocol* [2], que suporta tanto *TCP/IP* como *UDP* oferecendo assim uma comunicação funcional a aplicações através da internet. Este protocolo também permite usar o protocolo *Secure Socket Layer (SSL)*[3] oferecendo uma comunicação encriptada entre o cliente e o servidor.

A estrutura pode ser usada em várias linguagens de programação como *C++*, *C#*, *Java*, *Python*, entre outras. A linguagem escolhida para o desenvolvimento do serviço de *streaming* de vídeo foi *Java* devido ao conforto já adquirido com a linguagem.

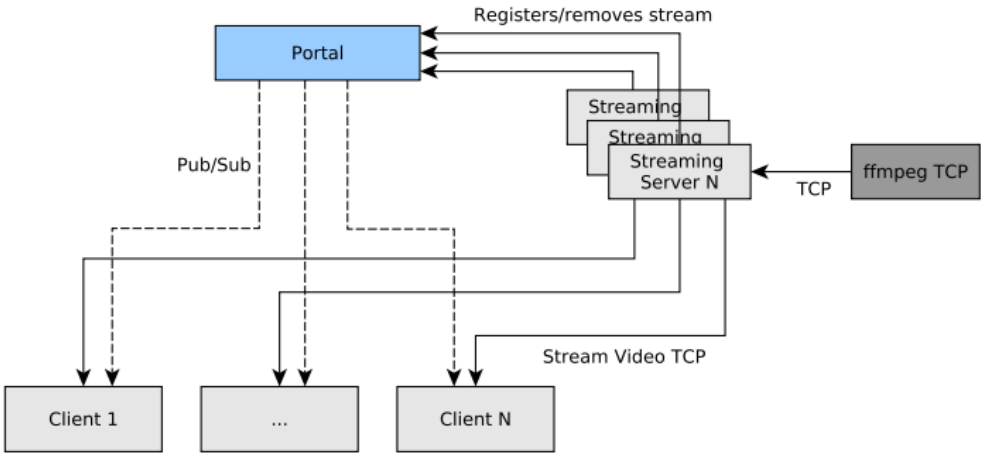
O serviço de *streaming* de vídeo é constituído por um Portal, um Cliente e um *Streamer*. O *Streamer* será o responsável por efetuar a *streaming* de vídeo, o Cliente de receber e visualizar o vídeo e o Portal de realizar a gestão dos *streamers* disponíveis e comunicar essas informações aos clientes.

[1] `https://zeroc.com/products/ice`
[2] `https://doc.zeroc.com/display/Ice36/Overview+of+the+Ice+Protocol`
[3] `https://doc.zeroc.com/display/Ice36/IceSSL`

2 Descrição

Na figura seguinte é possível visualizar a interação entre os diferentes componentes:

Figura 1: Interação entre os componentes.



Origem: `http://www.dcc.fc.up.pt/~rmartins/aulas/sd1617/trabII/trabII.pdf`

Com base na figura 1, temos as seguintes funções de cada componente:

- **Portal:**
  - Receber informações dos *streaming servers*, permitindo o registo e remoção de servidores, realizando as comunicações através do *Ice RPC*;
  - Informar todos os clientes dos *streaming servers* existentes, também através do *Ice RPC*;
  - Sempre que um novo *Streamer* se registar ou terminar, o Portal informará todos os clientes desse evento. Esta comunicação é efetuada através do *IceStorm*[4], um serviço de distribuição *Publisher/Subscriber*.
- ***Streamer*:**
  - Registar-se no Portal como um *streaming server* com todas as suas informações:
    - Nome da *stream*;
    - Endereço da *stream* constituído por tipo de protocolo, endereço IP e a porta;
    - Resolução do vídeo;
    - Velocidade de transmissão;

[4] `https://zeroc.com/products/ice/services/icestorm`

        * Conjunto de tópicos que caracterizem o vídeo.
    - Ao encerrar o *streaming server*, informa o Portal para que este atualize as suas informações.
    - Transmite o vídeo usando o programa *FFmpeg*[5] para um endereço. De seguida, cria uma socket nesse endereço com o objetivo de ler o output do *FFmpeg* para voltar a transmitir para outro endereço usando um *ServerSocket* para que múltiplos clientes possam visualizar a *stream*.

- **Cliente:**
    - Através do *Ice RPC*, pedir ao Portal a lista de *streams* existentes.
    - Registar-se no serviço de *Publisher* no Portal para passar a receber informações das novas *streams* ou de *streams* removidas.
    - Reproduzir o conteúdo disponibilizado pelo *streaming server* através do *ffplay*.

2.1 Metodologia

Para o desenvolvimento do projeto usamos o *IDE IntelliJ IDEA*[6] de forma a tornar a programação mais rápida e eficiente, além de que nos ajudou imenso a conhecer os vários métodos disponíveis nas bibliotecas usadas durante o desenvolvimento do projeto.

Pela facilidade de trabalhar em grupo, de ter registos de toda a atividade do projeto, como também nos permitir ter o código guardado de forma segura, usamos a ferramenta *Git*, neste caso o *GitLab*[7] para manter o projeto privado gratuitamente.

Quanto à divisão das tarefas do projeto, optamos por repartir os métodos de cada classe entre nós, de modo a que cada um conseguisse perceber o funcionamento de cada componente (*Cliente, Portal e Streamer*).

2.2 Interfaces/API Usada

- ***Portal Slice*:** Interface que descreve a estrutura de dados da *stream*, como também descreve as funções implementadas nas duas interfaces apresentadas a seguir.
- ***Notifier Interface*:** Utilizada para realizar a comunicação do Portal para todos os clientes através da função *inform()*.
- ***Portal Interface*:**
    - Implementa o serviço *Publisher/Subscriber* do *IceStorm* utilizada para o anúncio de alterações de *streams*.

[5] `https://ffmpeg.org/`
[6] `https://www.jetbrains.com/idea/`
[7] `https://gitlab.com/`

- Realiza uma gestão de *streams* que verifica periodicamente as que estão disponíveis, de forma a remover os *streamers* que por algum motivo encerraram a transmissão de forma inesperada.
- Contém todas as funções necessárias à manutenção dos *streaming servers*:
  * *register():* Insere um novo *streaming server* na lista de *streams* do Portal;
  * *remove():* Remove um *streaming server* da lista de *streams* do Portal;
  * *update():* Informar periodicamente o Portal de que o *streaming server* continua ativo;
  * *get():* Retorna a *stream* com o nome dado como argumento;
  * *getAll():* Devolve uma lista com todas as *streams* existentes;
  * *compare():* Verifica a igualdade dos endereços de duas *streams*.
  * *print():* Imprime as características de uma *stream* de forma estruturada.

2.3 Classes Principais e Suas Funcionalidades

- **Portal:** Classe responsável por criar uma nova instância da *Portal Interface*, inicializar o *Ice run time* e criar o adaptador capaz de realizar a comunicação com o Portal.
- ***Streamer*:** Tem como função criar a *stream* de forma a que todos os clientes possam reproduzir o vídeo.
- **Cliente:** Implementa as funções essenciais para a visualização de uma *stream*:
  - *parser():* Interpreta os comandos do utilizador verificando a sua validade;
  - *list():* Mostra todas as *streams* disponíveis;
  - *search():* Procura as *streams* relacionadas com os tópicos recebidos como argumento;
  - *play():* Responsável por reproduzir o vídeo de uma determinada *stream* utilizando o *FFplay*.

2.4 Principais Problemas Encontrados e Soluções Propostas

- **Implementação para múltiplos clientes:** Tivemos dificuldades em perceber como usar o *output* do *FFmpeg* para reproduzir vídeo em vários clientes. A solução passou por enviar o *output* do *FFmpeg* para uma *socket* e o *streaming server* retransmitir os dados recebidos dessa *socket* para tantas quanto o número de clientes conectados.
- **IceStorm:** Sendo uma tecnologia com a qual nunca trabalhamos, enfrentamos alguns problemas em criar os ficheiros de configuração do *Publisher/Subscriber*, consequentemente nos ficheiros de configuração do *Icebox*[8].
- **Fluidez da *stream*:** Mesmo com algumas implementações com o objetivo de aumentar a fluidez e conseguir as *streams* sincronizadas, continuamos com alguns problemas em *streams* em tempos diferentes.

[8] `https://doc.zeroc.com/display/Ice36/IceBox`

3 Conclusão

De uma forma geral, conseguimos implementar o serviço de *Video Streaming* entre vários clientes e *streaming servers*, embora com alguns problemas de fluidez já descritos.

A realização deste trabalho permitiu o aprofundamento do nosso conhecimento em vários temas. Nomeadamente, para perceber como um serviço de *streaming* funciona, apesar de que num nível básico, e a forma como o *Ice RPC* oferece um conjunto de ferramentas capaz de realizar a comunicação entre os componentes, como por exemplo a comunicação entre o Cliente e o Portal.

Consideramos que o tema do trabalho foi adequado para testar a aplicação desta tecnologia.