Rede peer-to-peer entre browsers

Mário Silva - A75654, Nuno Areal - A74714

Laboratórios de Engenharia Informática - MiEI
Universidade do Minho

Abstract. No âmbito da unidade curricular de Laboratórios de Engenharia Informática foram-nos propostos vários projetos. Decidimos escolher este pois pareceu-nos dos mais interessantes dentro da nossa área de especificação no mestrado. O presente relatório tem como objetivo explicitar os passos que demos desde o momento da escolha e o resultado final obtido.

1 Introdução

Numa fase inicial foi-nos indicado pelos docentes orientadores uma nova ferramenta que permitiria criar conexões entre dois *browsers* sem recurso a intermediários. Essa ferramenta era o WebRTC[1], atualmente abraçado pela Google que se empenha no seu desenvolvimento. De seguida elaboramos um plano de trabalho de forma a nos podermos organizar e conseguir realizar um prototipo final.
Tendo isto definido focamo-nos primeiramente em perceber a ideia principal que era a criação de uma rede *overlay* que permitisse o encaminhamento de tráfego entre dois quaisquer *peers*, dando a entender que estavam ligados diretamente. Esta rede viria a permitir encaminhar mensagens de texto, ficheiros e até seria possível efetuar chamadas de voz e vídeo através da mesma. Não nos foi possível realizar estes dois últimos pontos pois são os mais complexos e necessitariam de mais tempo de desenvolvimento.

[1] https://webrtc.org

Table of Contents

2 WebRTC

Lançado em 2011, o WebRTC tem vindo a ganhar cada vez mais fama desde que foi adquirido pela Google e tornado *open-source* devido à permissão que dá em estabelecer comunicações em tempo real através do *browser* e por não ser controlado pelos atuais mecanismos de *shapping* efetuados pelos ISPs, tornando-se em algo realmente útil no campo das comunicações.
Para ser possível existir essa comunicação entre dois *peers* é necessário que estes tenham conhecimento da intenção existente em estabelece-la. Para isso é obrigatória a existência de um servidor que trata do encaminhamento dessas "intenções". A nosso maior objetivo, no que é referente ao WebRTC, é a utilização mais reduzida possível do servidor durante toda a interação com a aplicação.

2.1 Sinalização

Este é o único processo onde é necessária interação com um servidor em toda a comunicação WebRTC. Este serve para que um *peer* faça chegar a outro o pedido, *Offer*, de conexão através do servidor, que guarda todas as conexões atualmente ativas. O servidor por sua vez encaminha essa *offer* até ao *peer* destino que irá responder com uma *Answer*. Estes dois tipos de mensagens contêm toda a informação sobre o tipo de ligação que cada um pode estabelecer, e através desta informação cada *browser* saberá como enviar os dados para outro *peer*.
Efetuado este processo os dois *peers* passam a trocar informações de ligação.

2.2 ICE, STUN e TURN

Para efetuar essa troca existe o protocolo ICE, Interactive Connectivity Establishment. Este tem como principal objetivo a troca de todas as formas possíveis de ligação entre os dois *peers*, recolhendo o IP da rede local, o IP público através do servidor STUN e possivelmente um IP de um servidor TURN que irá fazer de *relay* caso os dois métodos anteriores não funcionem.

STUN Este servidor é o responsável por responder somente à pergunta "Qual é o meu endereço IP?". Através dessa resposta o WebRTC poderá saber qual o IP público e utilizá-lo para o estabelecimento de conexões.

TURN Este servidor é apenas utilizado caso o endereço da rede local ou o endereço público não funcionem, o que costuma acontecer devido às *firewalls*. Este estabelece então uma ligação com os dois *peers*, passando a servir de intermediário na conexão.

2.3 *Downsides* do WebRTC

Apesar de utilizar SCTP nas comunicações, ser possível escolher se queremos uma conexão ordenada e confiável ou não, o WebRTC tem um lado menos bom,

a necessidade do uso de um servidor para estabelecer as conexões entre dois *peers*, embora isso não cause um grande impacto a nível global leva a que haja uma dependência de um meio físico sempre ligado para que seja possível efetuar ligações. Uma comunicação na sua maior parte com o servidor não é algo que seja pretendido, portanto devemos ter sempre em mente que apenas devemos utilizar o servidor para a sinalização.

3 Desenho da rede *overlay*

Tendo bem claro como funciona o WebRTC, o objetivo seguinte seria pensar numa forma de implementar a rede *overlay*. Decidimos optar por uma rede *peer-to-peer* não estruturada devido à possível saída e entrada de *peers* com elevada frequência. Baseamos desde logo no trabalho prático realizado na UC de Arquiteturas Emergentes de Redes, onde tivemos de implementar uma rede *Adhoc* que tinha o objetivo de encaminhar noticias pela rede, conhecendo apenas o seu vizinho direto e os vizinhos diretos deste, fazendo com que conhecesse-mos uma vizinhança de nível 2.
Não conseguindo comunicar com *peers* fora deste vizinhança tornou-se necessário implementar um mecanismo de descoberta de rotas até outros.

3.1 *Bootstrap* da rede

No momento em que um *peer* se liga à rede recebe uma lista com os *peers* atualmente ligados, escolhendo um aleatoriamente a partir dessa lista. É depois efetuada a conexão e recebidos os utilizadores ao quais esse *peer* está diretamente ligado de modo a que não sejam criadas conexões redundantes, algo não necessário neste caso.

3.2 Principais estruturas de dados

De forma a manter todas as informações utilizamos *Maps* do *JavaScript*, semelhante aos do *Java*, que para uma chave guardam um valor sem tipo predefinido. As estruturas *connections* e *reachable* mantêm estado das ligações diretas e dos vizinhos conhecidos, respetivamente.

```
Connections :: Map
Key         :: Peer Username (String)
Value       :: Object {name       :: String,
                       connection :: RTCPeerConnection,
                       channel    :: RTCDataChannel}

Reachable :: Map
Key       :: Destination Peer Username :: String
Value     :: Neighbour :: String/Object{destination :: String,
                                        time        :: Long}
```

Na estrutura *reachable* o valor é obrigatoriamente o nome de utilizador de uma conexão direta do *peer*.

3.3 Mecanismo de descoberta de rotas e manutenção

Quando necessitamos de enviar algo para um *peer* que não está em nenhuma das estruturas é efetuado um processo de *RouteRequest/RouteReply*. Este processo é muito semelhante ao que implementamos na UC de Arquiteturas Emergentes de Redes. Não se trata de um simples *broadcast* de mensagens a perguntar se conhecem esse IP, como é feito no ARP. No momento em que se desencadeia um *route request* é feito um *broadcast* dessa mensagem para todos os vizinhos diretos, estes vão verificar nas suas tabelas e responder consigam, caso contrário irão continuar com o *broadcast* mas desta vez não enviando para o originador da mensagem e adicionando uma entrada na estrutura *reachable*, impedindo assim retransmissões desnecessárias de pedidos iguais. Encontrado o *peer* na vizinhança de nível dois, este responde com um *route reply* que é enviado pelo caminho inverso, não efetuando desta vez o *broadcast* das respostas.
As rotas encontradas por este método apenas são válidas durante 5 minutos, procedendo-se ao mesmo processo para a sua renovação, caso seja pretendido efetuar envios para esse destino. Durante todo o período em que um cliente está online é tentado manter no mínimo duas conexões até um limite de cinco, podendo este último variar ou ser mesmo retirado consoante a aplicação.

4 Aplicação

O passo seguinte seria decidir qual seria a aplicação a construir para podermos efetivamente usar a rede *overlay*. Optamos por implementar um *chat* de grupo e também de mensagens diretas podendo dessa forma testar tanto o *broadcast* de mensagens como o seu envio pelos *peers* da rede. O resultado final foram duas funções, uma que permite o envio de mensagens e outra para o envio de ficheiros.

4.1 API

Para o envio de mensagens está disponível a função *sendMessage*. Esta recebe três argumentos o tipo da mensagem, *broadcast*, *direct* ou *file*, o destino da mensagem e a mensagem em si que tem um formato específico.
Para o envio de ficheiros existe a função *sendFile* que recebe dois argumentos. O destino do ficheiro e o ficheiro em si é tudo o que precisamos de fornecer à função para que esta consiga entregar o ficheiro no destino. De frisar que sempre antes de enviar um ficheiro é necessário enviar os dados do mesmo para que seja possível efetuar o encaminhamento pela rede.

4.2 Mensagens

Dependendo do tipo de mensagens a enviar é necessário seguir um formato específico para que a API possa fazer o seu encaminhamento de forma correta. Tanto para o envio de mensagens em *broadcast* como de mensagens diretas o formato é o mesmo e deve ser passada uma mensagem com o seguinte formato à função *sendMessage*

```
<Data>;<Utilizador origem>:<Mensagem de texto>

sendMessage("broadcast", "utilizador1",
"28-6-2018 11:56:45:539;utilizador2:Olá, tudo bem?");
```

Seguindo este formato para os dois tipos de mensagens não existirá qualquer problema. Para manter as mensagens guardadas durante uma sessão foi criada uma estrutura *messages*, que consiste num *Map*.

```
messages :: Map
Key :: String ('broadcast','<username>')
Value :: Array
    Para key=='broadcast', value==[Mensagens]
    Para key=='<username>', value==[{type: Tipo,
                                     message: Mensagem}]
```

Para o campo *type* temos dois valores, *text* e *file*, indicando que se trata de uma mensagem de texto ou de uma mensagem sobre um ficheiro. Esta última vem no formato *<Data>;<Origem>:<Nome do ficheiro>*.

No que toca ao envio de ficheiros a mensagem é um pouco diferente tendo o seguinte formato

```
<Destino>;<Data>;<Origem>;<Nome do ficheiro>;<Tamanho>;<Tipo>

sendMessage("file", "utilizador2", "utilizador1;
                    28-6-2018 11:56:45:539;utilizador1;
                    ficheiro.txt;10000;application/txt");
```

Depois do envio desta mensagem tudo estará pronto para que o ficheiro seja enviado através da rede.

5 Trabalho Futuro

Este projeto está longe de estar terminado mas encontra-se já numa fase onde é possível utilizar-lo para o envio de mensagens. O envio de ficheiros encontra-se muito prematuro, pois foi uma das últimas coisas a ser implementada, só permitindo envio de um ficheiro em simultâneo em toda a rede e apenas é possível fazer-lo em mensagens diretas não estando implementado para mensagens em *broadcast*.
Outra das coisas que falta investigar é a forma como encaminhar áudio e vídeo através da rede *overlay*, algo que após uma pesquisa ligeira se relevou trabalhoso. Uma das hipóteses seria estabelecer uma ligação de áudio e vídeo entre os *peers* da rede e encaminhar todo o tráfego por essa conexão. Isso dificultaria a distinção de pacotes ficando sem saber qual a sua origem e destino.