Serviço distribuído de Publicação e Subscrição de Ficheiros

Hugo Abreu | A76203 João Padrão | A76438
João Reis | A75372

Coordenador: Paulo Almeida

Laboratório em Engenharia Informática
Universidade do Minho

4 de Julho de 2018

1 Motivação

O desenvolvimento deste projeto advém da necessidade de partilha de ficheiros imutáveis usualmente "time-limited", disponibilizados entre grupos de pessoas. Desta forma este projeto tem um objetivo completamente contrário ao controlo de versões.

Uma partilha imutável de um ficheiro temporário que, não pode ser alterado de forma iterativa, é algo bastante natural no dia a dia, sejam estes documentos ou ficheiros pessoais. Exemplos de ficheiros imutáveis que diariamente pretendemos partilhar e disponibilizar a diferentes pessoas e grupos de utilizadores são: PDF, JPEG, PNG, MP3, MP4 e todo o tipo de multimédia não passível de modificação.

A falta de um serviço que satisfaça as todas necessidades acima referidas, apenas veio justificar o desenvolvimento deste projeto, simplificando assim a partilha local e conectada de forma a que qualquer pessoa, mesmo sem conhecimentos tecnológicos, tenha a possibilidade de transferir/enviar ficheiros de forma linear entre dispositivos e para diferentes pessoas.

2 Objetivos

Partindo da motivação acima exposta, o grupo definiu os seguintes tópicos como principais objetivos:

- Criação de grupos lógicos e agregação de utilizadores em diferentes grupos de subscrição de ficheiros;
- Publicação de ficheiros imutáveis de forma local e não conectada à rede exterior, com presença local;
- Publicação de ficheiros imutáveis de forma descentralizada e com conexão à rede;
- Utilização de super nodos de rede (vulgos servidores) para aumentar a velocidade e redundância;
- Permitir persistência nos servidores de forma temporária para suportar faltas e utilizadores indisponíveis momentaneamente;

3 Abordagem

O serviço pode ser dividido em duas partes fundamentais: Partilha de ficheiros na rede local e partilha de ficheiros com recurso a servidores bem conhecidos na rede. Para a implementação do trabalho tal como definido conceptualmente pelo grupo, houve a necessidade de implementar um protocolo P2P. Após uma análise de diferentes tipos de protocolos descentralizados de transferência de ficheiros, o protocolo BitTorrent[1] foi o que mais se ajustou ao problema apresentado, sendo abordado mais em detalhe na secção 7.

Portanto, tirando partido da natureza descentralizada do protocolo selecionado, o grupo pretende implementar um sistema funcional sem a necessidade de manter uma infraestrutura poderosa com grande processamento ou largura de banda.

[1] http://www.bittorrent.org/beps/bep_0003.html

4 Arquitetura do Sistema

A arquitetura idealizada é composta por 4 componentes:

- **Client:** Ponto de entrada para utilizadores poderem interagir com o serviço.
- **Traker:** Elemento fundamental na arquitetura Bittorrent, que atua como coordenador nas transferências de ficheiros entre utilizadores e nodo de replicação.
- **Zookeeper:** Módulo responsável pelo armazenamento distribuído de dados referentes aos utilizadores, e os respetivos grupos.
- **Front Server:** Atua como ponte de ligação entre os Clients e Trackers, e interage diretamente com a base de dados Zookeeper.

O funcionamento normal de uma transferência em modo Offline pode ser visualizada no seguinte diagrama:

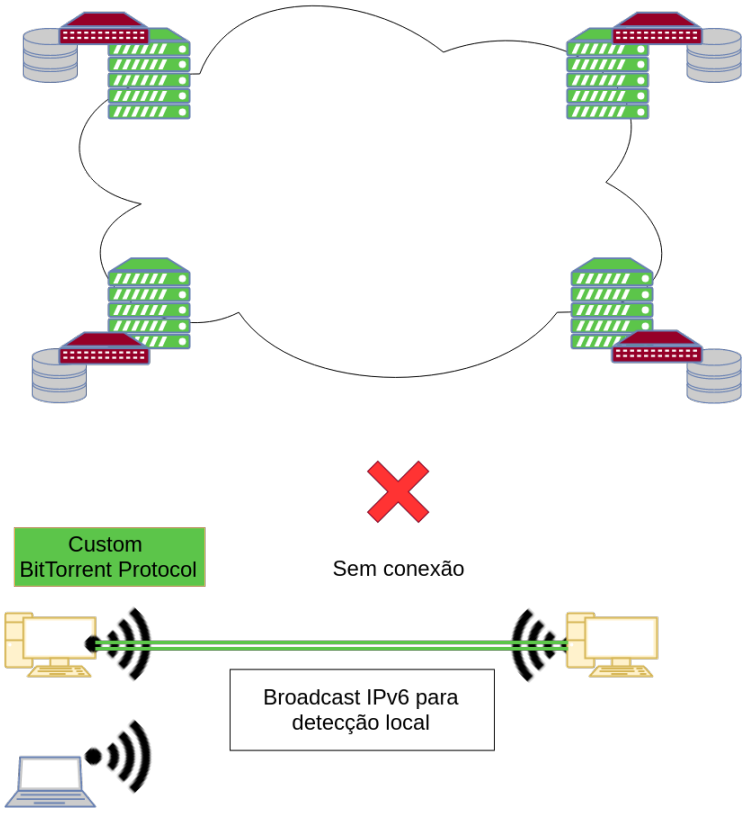

Figura 1: Arquitetura do sistema em modo Offline

Neste caso, todos os Clients anunciam-se na rede através de mensagens *multicast*. Quando um utilizadores deseja enviar um ficheiro para ou outro utilizador, ou grupo, é iniciado o protocolo de transferência entre eles.

O funcionamento normal de uma transferência Online de ficheiros entre utilizadores pode ser visualizada no seguinte diagrama:

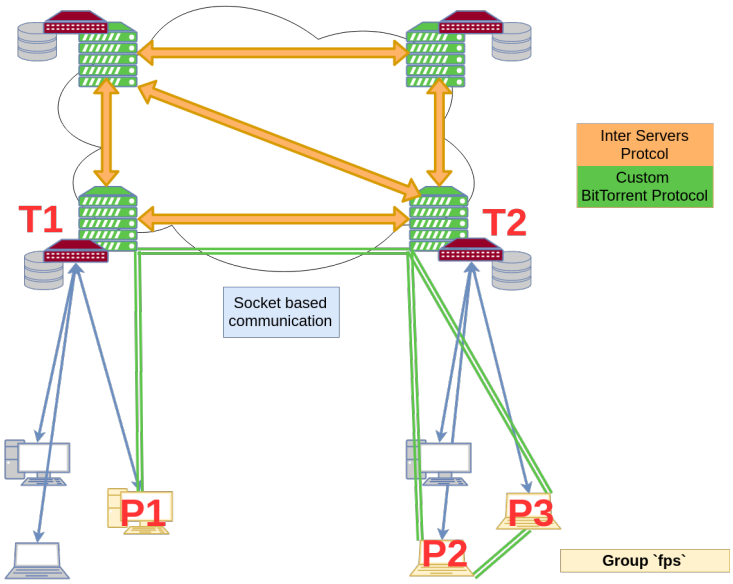

Figura 2: Arquitetura do sistema em modo Online

Neste caso concreto, os Clients P2 e P3 encontram-se ligados a um *front-server* (T2), diferente do servidor que trata do Client P1 (T1). Assim sendo, quando se dá uma partilha de um ficheiro entre estes, a ligação entre o servidor T1 e o servidor T2 deve atuar como uma ponte, pela qual passa o ficheiro em causa. Desta forma garante-se o isolamento de Clients em diferentes zonas geográficas. Esta caraterística revela-se importante pois é razoável assumir que existe grande largura de banda entre os servidores. Contudo o mesmo não se pode dizer dos Clients.

5 Zookeeper

Na arquitetura apresentada, poderão existir várias instâncias de bases de dados que terão de partilhar entre si os mesmos dados. Perante as soluções estudadas, o *Apache ZooKeeper* apresentou o melhor conjunto de características que resolviam os requerimentos do grupo - é altamente escalável e consistente, o que permite que existam vários servidores sempre com a mesma informação, e tem um comportamento semelhante a um sistema de ficheiros, tanto que um nodo tanto pode conter informações como servir de diretoria, o que se adequa muito bem à arquitetura apresentada.

5.1 Arquitetura

Sobre a raiz do *filesystem* encontram-se quatro nodos estáticos - *users*, *groups*, *trackers* e *front-fervers* e - em que cada um contem informação relativamente constantemente atualizada sobre o que lhe diz respeito.

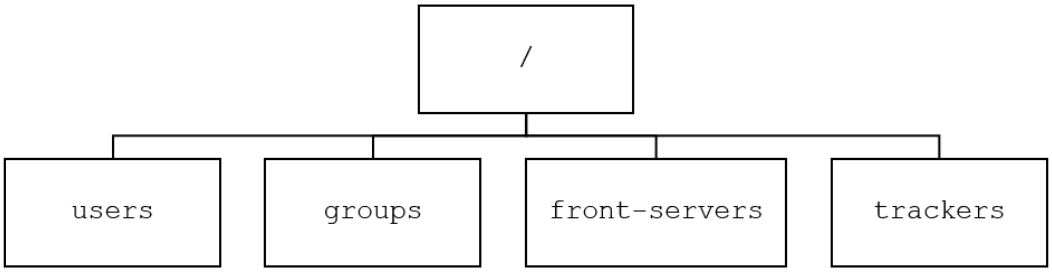

Figura 3: Arquitetura de nodos

5.1.1 Users

O nodo *users* terá associado todos os utilizadores no sistema. Consequentemente, cada nodo de utilizador conterá como identificador o *username* e contem como informação a *password*. Cada nodo de utilizador contem 5 sub-nodos:

- *name*: Corresponde ao nome do utilizador;
- *online*: Indica se o utilizador encontra-se online naquele dado momento;
- *server*: Se online, indica o servidor onde se encontra;
- *groups*: Trata-se de um diretório que contém o identificador dos grupos onde se encontra;
- *missing*: Indica os meta-dados recebidos no seu grupo quando se encontrava offline, de modo a receber o(s) ficheiro(s) quando ficar novamente online.

5.1.2 Groups

Neste nodo encontram-se todos os grupos existentes no sistema bem como as informações a ele relacionados.

- *users*: Indica os uilizadores do respetivo grupo;
- *torrents*: Nesta diretoria encontra-se os vários torrents partilhados num dado grupo
  - *torrent*: Encontra-se quantos utilizadores já receberam o ficheiro de meta-dados para iniciar a transferência;
  - *file*: Encontra-se quantos utilizadores já receberam o ficheiro de partilhado;

5.1.3 Trackers e Front-Servers

Estes nodos servem para ambos os *Trackers* e *Front-Servers* se registarem, com a particularidade de darem uso à capacidade de nodos efémeros do *ZooKeeper*. Com os nodos efémeros do *ZooKeeper*, estes existem enquanto os servidores manterem conexão com a base de dados, e, uma vez offline, são automaticamente removidos. Cada nodo contem o respetivo ID, bem como o seu endereço IP.

5.2 Garbage Collecting

Uma das principais características do serviço é o ficheiro deixar de existir no sistema quando todos os utilizadores alvo a já o tiverem. Como tal, no ZooKeeper será necessário apagar os dados dos ficheiros que já foram recebidos na totalidade. Portanto, o *Tracker* contem um contador que irá incrementando à medida que alguém concluí a transferência do ficheiro. Usando a *framework Curator* para *Java*, que se trata de uma API de alto nível para *ZooKeeper* que oferece um contador distribuído e *reliable*. Assim que contador chegar ao total de utilizadores que precisam de receber o ficheiro, a diretoria no ZooKeeper com o ficheiro será apagada.

6 Front-end

O *front-end* trata-se do intermediário entre o cliente e os restantes módulos do serviço. Tendo em atenção o elevado número de pedidos espectáveis e na alta incidência de novas conexões, usou-se Erlang para implementar este modulo do sistema dada a sua capacidade de escalar a quantidade de processos existentes.

O *front-server* foi implementado de modo a existirem várias instâncias a correr e que possam comunicar-se entre si, e arquitetura geral resume-se ao seguinte:

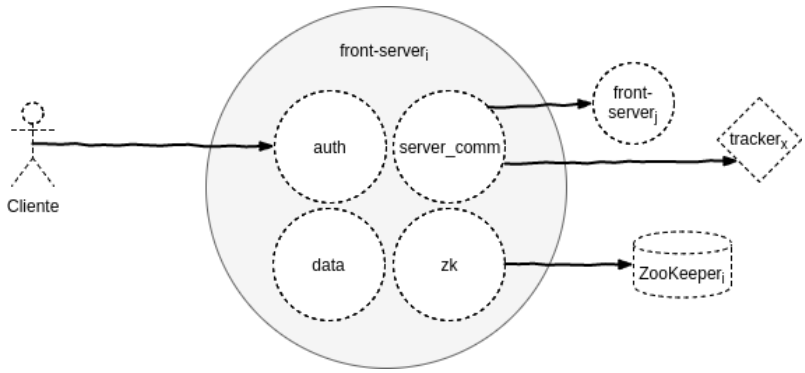

Figura 4: Arquitetura geral de uma instância *Front-Server*

Todas as comunicações - entre clientes ou outros servidores - foi feita usando *Protocol Buffers* como mecanismo de serialização de dados. Para tal usou-se a biblioteca *GPB*[2] que implementa deste mecanismo em Erlang, dado não existir suporte oficial. Usou-se também a biblioteca *erlzk*[3] para possibilitar a conexão entre Erlang e a base de dados *ZooKeeper*.

6.1 Módulos

6.1.1 auth

O módulo *auth* está ativamente à escuta de novas conexões de utilizadores. Assim que uma nova conexão é aceite, cria-se um processo para a mesma de modo a poder responder aos seus pedidos convenientemente. O cliente poderá registar-se, efetuar login, criar e juntar-se a um grupo, bem como obter uma lista de utilizadores online,

[2] https://github.com/tomas-abrahamsson/gpb
[3] https://github.com/huaban/erlzk

grupos a que pertence e anunciar a sua intenção de partilhar um ficheiro. Dado a complexidade desta última, será explorada mais à frente com maior profundidade. Quando o *login* é efetuado com sucesso, o Socket do qual é efetuado fica em loop continuamente à espera de novos pedidos.

6.1.2 data

Uma das funções deste módulo é controlar os utilizadores online naquela instância *front-server* e associa-los ao seu PID. Será útil quando for necessário disseminar uma ficheiro *torrent* pelos vários utilizadores de um grupo. Quando um utilizador efetua *login* com sucesso, é imediatamente colocado o seu utilizador e PID na base de dados *mnesia*. Porém, se a conexão fechar, da mesma maneira será removido.

Uma das outras funções será fornecer funções que escrevem, leem e removem de disco ficheiros *torrent* que poderão ser necessários persistir.

6.1.3 server_comm

Tal como o seu nome indica, este modulo comunica entre servidores - que tanto poderão ser *trackers* ou outros servidores *front-end*.

Continuamente à espera de comunicação, é criada um novo processo para cada conexão e redireciona os pedidos para os processos respetivos.

6.1.4 zk

Usando maioritariamente a biblioteca *erlzk*, este módulo efetua a comunicação dos vários processos Erlang com a base de dados *zookeeper*. Encarregue de construir caminhos válidos para funções como login e registo, ou qualquer tipo de necessidade do cliente que tenha de aceder à base de dados.

6.2 Participação na partilha de ficheiros

Quando um cliente revela intenção de partilhar um ficheiro, inicialmente terá de disseminar pelos vários utilizadores do seu grupo bem como aos devidos *trackers*, e é aqui que o front-end têm o seu importante papel.

6.2.1 Disseminação dos meta-dados

O utilizador cria os seus meta-dados em formato .torrent e gera um ID único, enviando-o para o *front-server*. Assim que o recebe, ocorre o seguinte:

1. Obtém-se a lista de utilizadores do grupo para onde irá ser enviado o ficheiro, bem como a sua localização, ou seja, a que *front-server* estão ligados ou então offline, se tal se verificar. Estes utilizadores serão os utilizadores que naquele preciso momento estão no grupo, e serão os únicos a receber o ficheiro;

2. Envia-se o ficheiro torrent para o *Tracker* associado a aquele *front-server*;

3. Guarda-se o ficheiro em disco;

4. Cria-se meta dados-sobre aquele ficheiro em *ZooKeeper*, para controlar a receção do ficheiro torrent e mais tarde o ficheiro partilhado;

5. Itera-se sobre os utilizadores anteriormente obtidos:

   (a) Se o utilizador se encontrar offline naquele dado momento, é assinalado nos dados *ZooKeeper* daquele utilizador que tem aquele ficheiro em falta;

   (b) Caso se encontre no atual *front-server*, recorrendo ao ao módulo *data* obtém-se o seu PID, e envia-se o ficheiro de modo a entregar o ficheiro *torrent*. É posteriormente assinalado como recebido no *ZooKeeper*. Se por algum motivo ficou offline entretanto, é assinalado offline como em (a);

   (c) Finalmente, se estiver noutro *front-server*, recorre-se ao *ZooKeeper* para obter a sua localização e envia-se. O *front-server* recetor efetua o mesmo que em (b).

6.2.2 Recuperação de ficheiros recebidos enquanto offline

Na eventualidade de um utilizador se encontrar offline na altura da disseminação do ficheiro, como referido em 6.2.1 5a, este ficará assinalado na base de dados *ZooKeeper*. Como tal, sempre que um cliente efetua *login* com sucesso, verificar-se-à na base de dados se tem ficheiros em falta. Se tal se conferir, verifica-se se o ficheiro existe localmente - senão é pedido uma cópia ao *Tracker*. É então enviado ao utilizador, e assinalado como recebido no *ZooKeeper*.

7 Biblioteca ttorrent

A biblioteca ttorrent[4] reutilizada e modificada pelo grupo foi selecionada pela sua simplicidade e correta implementação do protocolo pretendido.

A escolha desta biblioteca passou também pela sua versatilidade e rapidez, tal como descrito na página.

Although the write performance of the BitTorrent client is currently quite poor (10MB/sec/connected peer), it has been measured that the distribution of a 150MB file to thousands of machines across several datacenters took no more than 30 seconds, with very little network overhead for the initial seeder (only 125% of the original file size uploaded by the initial seeder).

Tendo em conta estas assunções, depreendemos que a biblioteca é ideal para o problema e, portanto foi implementada com as alterações necessárias.

7.1 Limitações

Após uma primeira análise à biblioteca, o grupo deparou-se com limitações da biblioteca no que toca ao problema abordado.

A biblioteca não suporta *Universal Plug n' Play* ou, *Hole punching (networking)*, pelo que houve a necessidade de implementar port-forwarding aquando da inicialização de um cliente pois, a grande maioria dos utilizadores, encontra-se ligado a uma NAT ou atrás de uma Firewall.

Uma outra necessidade do grupo no que toca à biblioteca e respetivo protocolo, é a definição de super-nodos com separação geográfica, isto é, cada super-nodo

[4] https://github.com/mpetazzoni/ttorrent

deve apenas receber atualizações de outros super-nodos e clientes da sua localização geográfica, portanto, clientes de secções geográficas diferentes não devem manter contacto para que, não exista desperdício de largura de banda.

7.2 Alterações efetuadas

Para colmatar as deficiências existentes foi efetuado um *fork* da biblioteca no *GitHub* e foram implementadas as soluções necessárias para superar todas as limitações encontradas, desta forma, mantendo sempre o repositório público para permitir a partilha de código útil. As alterações efetuadas no *fork* da biblioteca serão abordadas nas sub secções abaixo.

7.2.1 Inserções de super-nodos

Tal como já abordado, para ser possível efetuar a separação geográfica de *peers*, mantendo apenas ligações via super-nodos entre zonas, houve necessidade de alterar a implementação do *Tracker*.

Esta alteração visa a possibilidade de detetar a origem dos pedidos para construir respostas de acordo, isto é, quando os pedidos são oriundos de clientes, as respostas construídas apenas contêm outros clientes e o próprio super-nodo. Por outro lado, quando é o super-nodo pedir uma lista de pares, a resposta construída, para além de clientes, contém também outros super-nodos mantendo assim a ligação entre estes para envio de ficheiros inter-zona.

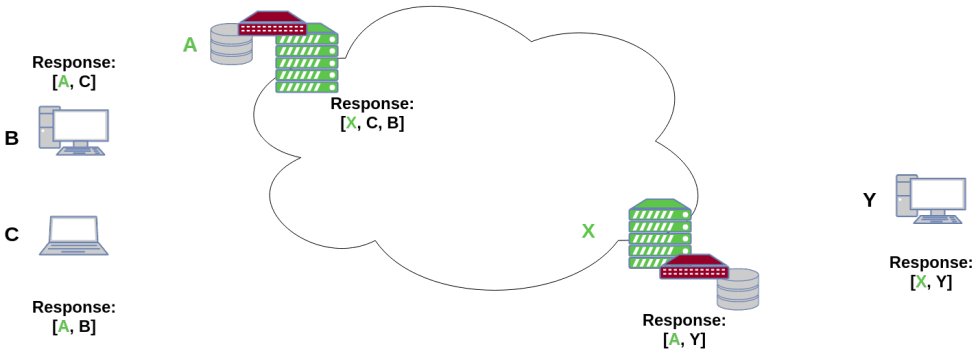

Figura 5: Construção e diferentes tipos de repostas

Como é possível verificar na figura 5, não existe qualquer tipo de conhecimento entre pares de zona diferentes, garantindo assim que não existe desperdício de largura de banda e delegando essa responsabilidade para os super-nodos.

7.2.2 Fecho unilateral de sockets

À medida que o grupo se encontrava a testar a biblioteca e implementações desta, deparou-se com o facto de que quando existe uma exceção num canal de um dado peer, típicamente este aborta a conexão com o seu vizinho e, o respetivo nodo vizinho ao verificar esta interrupção não toma medidas para eliminar esse mesmo nodo de memória. Visto que a biblioteca reutiliza peers quando recebe respostas do tracker, estas exceções podem tornar-se problemáticas, pois ao reutilizar o peer, a porta de

conexão pode não ser a correta, isto porque, normalmente as conexões podem ser efetuadas bidirecionalmente, ou seja, a porta atual do peer pode não ser a porta de *bind*, mas sim a porta da ligação anterior caso a conexão tenha sido iniciada pelo nodo vizinho.

Para resolver este problema, o grupo optou por deixar efetuar pelo menos uma tentativa de conexão quando são recebidos novos pares, se essa conexão não for bem sucedida, esse facto é guardado e, no próximo anúncio a conexão é forçosamente efetuada com a nova porta.

7.2.3 Universal Plug n' Play

Uma das mais importantes implementações e melhorias efetuadas na biblioteca foi o suporte a *Universal Plug n' Play* que, revelou-se essencial na ótica do utilizador. Típicamente um utilizador normal nunca tem acesso direto à internet, isto é, encontra-se sempre atrás de uma NAT ou Firewall, por esta razão, para efetuar *bind* de endereços e ser possível contactar este, é necessário efetuar *port-forwarding* no router.

A implementação de uma biblioteca de suporte a *UPnP* no código da biblioteca *ttorrent*, nomeadamente no cliente, permitiu abstrair essa necessidade dos utilizadores, efetuando de forma automática o *port-forwarding* do *gateway*.

7.2.4 Biblioteca weupnp

Para a correta implementação do ponto anterior foi utilizada a biblioteca weupnp[5] que permite efetuar os pedidos ao *gateway* de forma dinâmica e automática, pelo que, à medida do necessário, vamos enviando pedidos de atribuição e redirecionamento de portas até obtermos uma livre.

Por sua vez esta biblioteca também contém limitações. A limitação encontrada pelo grupo foi a necessidade específica da existência de um *gateway*, pelo que, normalmente numa VPN não existem *gateways* e, portanto, não é possível obter portas. Esta limitação poderia ser superada pois existe software P2P que efetivamente consegue efetuar *bind* numa VPN.

Estas limitações não foram superadas pelo grupo pela falta de conhecimentos acerca de *probes* específicos para *gateways* e limitações temporais, além disso, as limitações não se revelaram cruciais a implementação da biblioteca.

8 Tracker

Para a correta implementação do módulo de Tracker tal como abordado e conceptualizado no protocolo BitTorrent, foi utilizada a respetiva biblioteca ttorrent abordada na secção 7.

O tracker implementado não mantém um papel normal de um tracker BitTorrent, pelo que tem também papel de super-nodo que deve transferir, persistir e fornecer ficheiros os utilizadores de forma a reduzir a complexidade e necessidade de largura de banda nos nodos singulares.

[5] https://github.com/bitletorg/weupnp

8.1 Super-nodos

O papel de super-nodo no tracker desempenha um papel crucial na arquitetura global do projeto, pelo que, grande parte do fornecimento de ficheiros e grande parte da concetualização assenta na capacidade deste super-nodo permitir ajudar à disseminação do ficheiro graças a sua alta largura de banda e implementação do respetivo protocolo entre estes.

Um super-nodo é responsável por se inserir numa publicação de um ficheiro para que o receba e, ao mesmo tempo seja capaz de fornecer pedaços em falta aos vários clientes da sua zona geográfica que subscreveram o respetivo ficheiro.

Além do fornecimento em tempo-real do ficheiro aos clientes da sua área geográfica, o super-nodo, com recurso ao protocolo inter-servidores abordado na secção 8.3, tem a responsabilidade de partilhar com os super-nodos o ficheiro, quer para questões de replicação e persistência, quer para fornecimento local a outros clientes.

8.2 Inicialização e paragem estratégicas

Por uma questão de *performance* o super-nodo para e reinicia clientes que fornecem ficheiros, isto é, à medida que existem utilizadores a efetuar o download de um dado ficheiro o super-nodo mantém um cliente ativo a obter e fornecer o mesmo ficheiro, quando todos os clientes atuais da sua zona terminarem o download e, o super-nodo tiver a responsabilidade de persistir o ficheiro, este apenas para o cliente não eliminando qualquer tipo de ficheiros.

Quando um dado cliente anuncia ao tracker que pretende iniciar a transferência de um ficheiro, o super-nodo inicia de forma dinâmica o cliente para fornecer esse ficheiro. Desta forma apenas são consumidos recursos enquanto existirem ficheiros em circulação.

Um servidor que não tenha responsabilidade de persistir o ficheiro, ao terminar o cliente também elimina todos os ficheiros tal como será abordado na secção 8.5.

8.3 Protocolo entre servidores

Graças a separação geográfica de super-nodos (trackers), existe a necessidade de estes comunicarem entre si, esta comunicação é essencial, tanto para a persistência dos ficheiros como para a respetiva eliminação.

Com a implementação da funcionalidade de inserção de super-nodos de forma especial na biblioteca ttorrent abordada na secção 7.2.1, foi possível ao grupo efetuar conexões singulares e inter-servidores para troca e envio de ficheiros. Esta implementação permitiu que fosse possível a separação geográfica e, mesmo assim obter boas velocidades graças à largura de banda dos super-nodos.

Existem dois tipos de mensagens trocadas entre servidores, os pedidos de inserção ou injeção e os pedidos de remoção. Quando um dado super-nodo recebe um ficheiro de meta-dados, envia um pedido de inserção a todos os outros trackers da lista contida nos meta-dados. Com este pedido é possível garantir a separação de zonas e a ligação singular por ficheiro entre super-nodos.

Já a mensagem de remoção indica a um dado super-nodo que um outro terminou e, portanto, assim que for possível pode terminar o seu cliente de forma segura. Para além desta indicação, quando um super-nodo recebe uma mensagem de remoção e, efetivamente o campo do grupo se encontra preenchido significa que toda a gente do

grupo efetuou o download e, por isso, se persistiu / replicou o ficheiro neste ponto, pode de forma segura deixar de efetuar o tracking deste e pode eliminar qualquer ficheiro persistido relativo ao torrent do grupo em questão.

8.4 Replicação

No que toca a replicação o grupo tratou de dotar os super-nodos com todos os mecanismos necessários para efetuar esta.

Neste momento o único super-nodo que persiste os ficheiros é efetivamente o super-nodo principal da zona do utilizador que o publicou, a superação desta limitação apenas se encontra dependente de um protocolo de comunicação em grupo, onde se poderia efetuar um consensus para definir quais dos servidores devem persisitir ou, até mesmo a implementação de uma Distributed Hash Table. Por limitações de tempo não foi implementado nenhum protocolo específico, pelo que, aquando da implementação deste, as únicas alterações a efetuar para permitir a replicação é apenas a criação de uma função que indique se o servidor atual deve persistir ou não pois, todo o código foi efetuado com esta função conceptual.

8.5 Persistência

A persistência consiste em, quando um dado servidor recebe o ficheiro de meta-dados acerca de uma publicação se, deve persistir ou não o ficheiro que irá passar por ele. A persistência é efetuada com recurso ao ttorrent, pelo que o ficheiro é persistido peça a peça a medida que o super-nodo o recebe, quer pelo cliente quer via outro super-nodos.

Um dado super-nodo que não tenha a responsabilidade de persistir um dado ficheiro apenas o mantém enquanto houverem clientes ativos na sua zona a efetuarem o download, a partir do momento que toda a gente da sua zona termine o download então, este elimina qualquer informação acerca do ficheiro informando os outros servidores.

A persistência do ficheiro de meta-dados permite também tolerar faltas e garantir um dos requisitos do projeto que consiste em permitir clientes Offline efetuarem o download mais tarde.

9 Client

O módulo Client atua como ponto de entrada para os utilizadores no serviço, materializando a troca de ficheiros entre estes e a comunicação com os servidores. Este módulo abrange duas formas de funcionamento: Offline e Online.

9.1 Offline

A funcionalidade offline integrada no módulo Client permite que utilizadores não conectados à Internet, consigam mesmo assim partilhar ficheiros entre si. Para tal, cada Client, quando deteta que não é possível estabelecer conexão com os *front-servers*, inicia o modo de operação local.

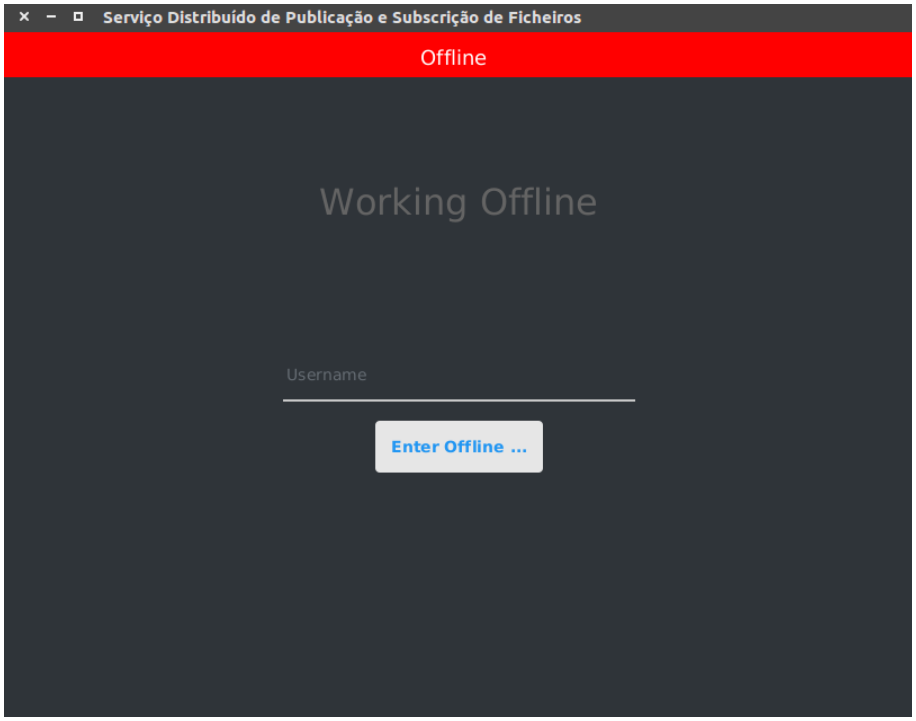

Figura 6: Interface de entrada em modo Offline

Após a definição do *username* local, o Client faz *broadcast* dos seus endereços *ipv4* e *ipv6*, e do seu *username*. O *broadcast* para a rede é conseguido através de primitivas definidas pelo protocolo *ipv6*, onde o endereço `ff02::1` está reservado para fins de *multicast* numa rede local. Paralelamente é iniciado um procedimento de escuta a outros utilizadores que estejam a correr o mesmo serviço e, desta forma, a anunciarem-se na rede.

A partilha de um ficheiro pode ser direcionada a um único utilizador ativo na rede, ou para um grupo de utilizadores definido localmente pelo emissor. Este emissor atua como *tracker* na rede e inicia também um cliente *torrent*, permitindo assim a transferência do ficheiro entre os utilizadores. Os recetores do ficheiro de meta-dados recebem uma notificação, e decidem se querem ou não recebe-lo. Se confirmarem a intenção de receber o ficheiro, iniciam um cliente *ttorrent* e efetuam *download*.

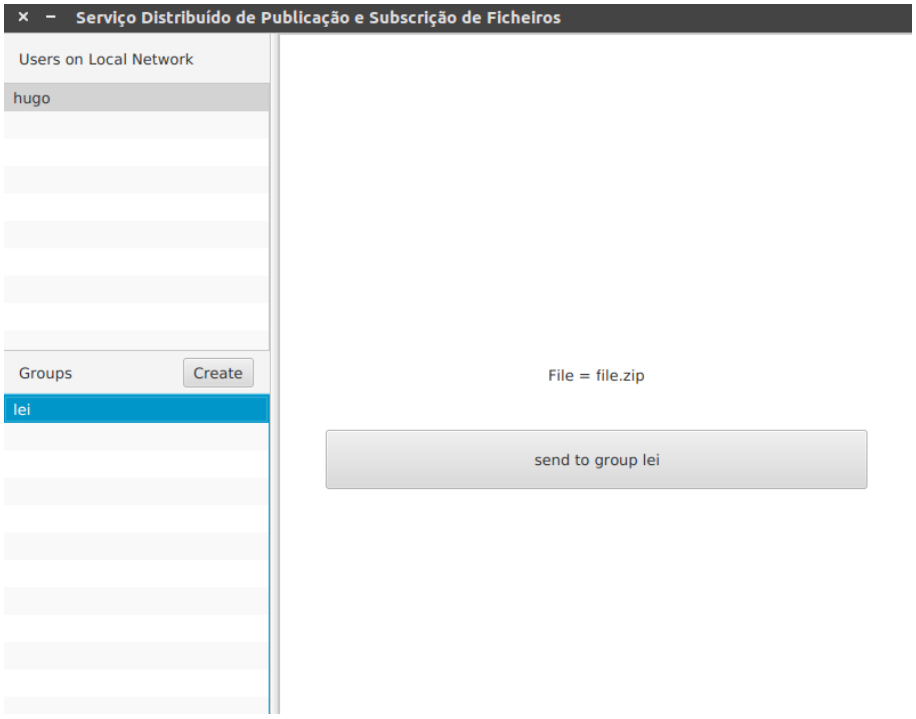

Figura 7: Envio de um ficheiro para um grupo

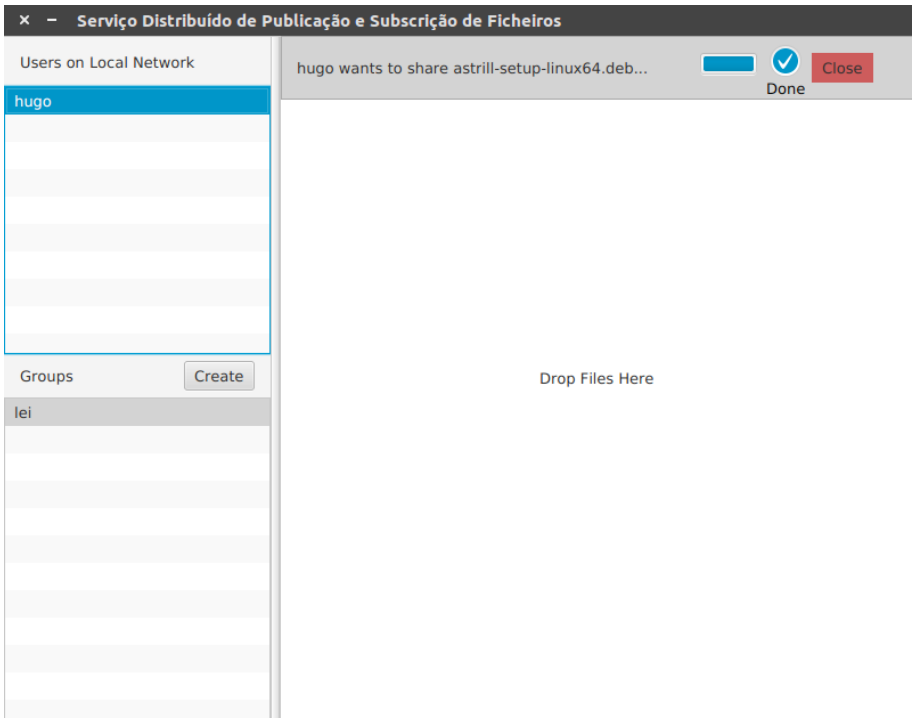

Figura 8: Término de um *download*

9.2 Online

No modo online existe conexão com os servidores do serviço, logo as funcionalidades de autenticação, partilha em grupo e ligação aos super-nodos encontram-se disponíveis.

Após o início de sessão, são reunidos os grupos aos quais o utilizador pertence. Os elementos deste grupo vão sendo atualizados à medida que outros utilizadores se juntem ou disconectem do respetivo grupo. Em cada grupo é possível ver todos os utilizadores desse mesmo grupo que se encontram Online.

Adicionalmente, o utilizador pode criar um novo grupo, ou juntar-se a um já existente de forma simples, rápida e intuitiva, recorrendo ao botões situados na borda superior da lista, tal como é possível verificar na figura 9

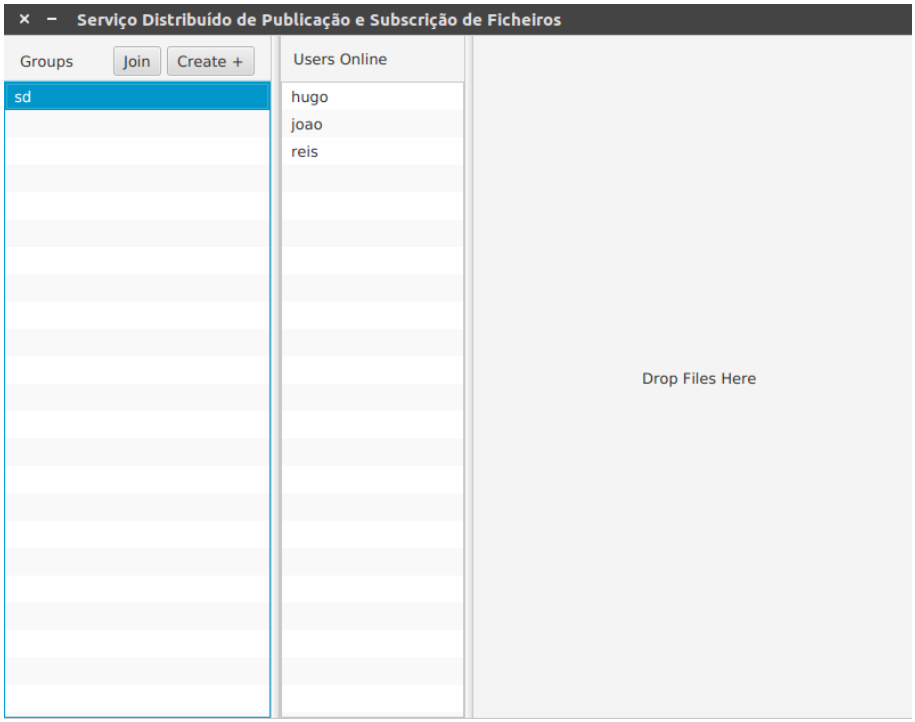

Figura 9: Interface de grupos em modo Online

Quando um utilizador deseja enviar um ficheiro para um outro utilizador, ou grupo, o client gera o ficheiro de meta-dados com recurso à biblioteca ttorrent e envia para o *front-server* ao qual se encontra atualmente conectado. Todos os destinatários recebem uma notificação sobre a qual têm possibilidade de permitir ou recusar a transferência do ficheiro. Caso estes desejem fazer *download* do ficheiro, têm a obrigação de alterar a ordem dos tendo em conta que o seu tracker mais próximo não é necessariamente o primeiro do ficheiro de meta-dados, permitindo assim que se conectem ao servidor mais perto geograficamente, tomando partido da separação geográfica e melhor largura de banda. Por fim iniciam o processo de *download* para uma pasta pré-definida.

O download do ficheiro é efetuado recorrendo ao cliente do ttorrent, alterado pelo grupo, para que automaticamente redirecione portas (secção 7.2.4) e permita que tudo funcione de forma simples, iniciando o *download* assim que possível.

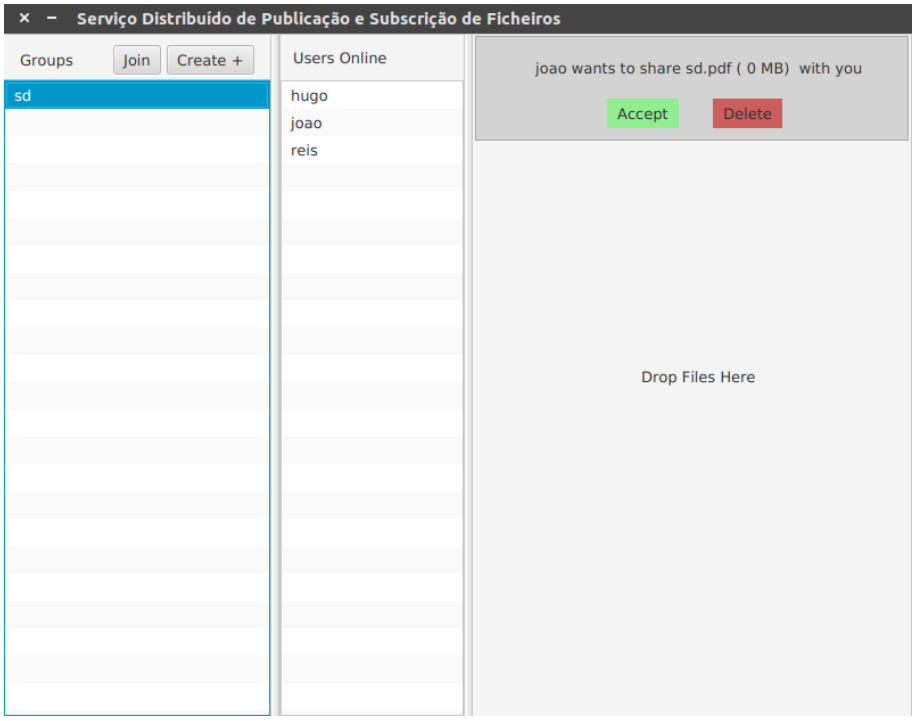

Figura 10: Notificação de partilha de um ficheiro

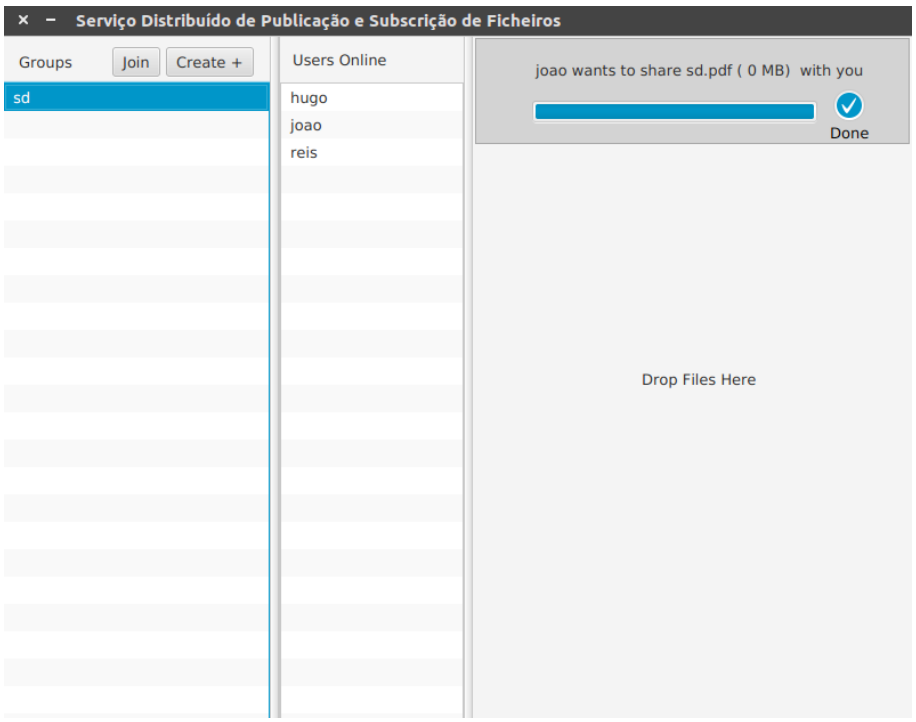

Figura 11: Término de um *download*

Todos os ficheiros de meta-dados referentes a ficheiros que não são descarregados na totalidade, são guardados para que, na próxima execução do programa, sejam carregados e descarregados pelo cliente.

Não existe comunicação contínua entre clientes e trackers, dado que o client apenas se encontra conectado ao *front-server* de forma premanente. Apenas quando

um cliente deseja efetuar o *download* de um ficheiro é estabelecida uma ligação entre o client e um tracker, para que, seja possível obter a lista dos diferentes nodos que se encontram a transferir o ficheiro e abrir as conexões necessárias para o obter de forma rápida, para isso, a biblioteca ttorrent irá tomar partido de todo o poder do protocolo BitTorrent para garantir que todos os utilizadores recebem o ficheiro o mais rápido possível, efetuando várias trocas entre vários nodos.

10 Deploy

Tendo em mente a grande possibilidade de escala do sistema, foi implementado um mecanismo simples e eficiente para iniciar e instânciar super-nodos, conjugando tracker e *front-server*. Para tal foi implementado um conjunto de *roles* em Ansible[6] que tratam da instalação das dependências necessárias, bem como a clonagem dos repositórios que contêm o código fonte do serviço e a sua compilação.

Para correta e simples implementação do deploy de servidores, foram criados 4 *roles* um para cada função. Um para instalar os diferentes pacotes necessários, como o erlang, o *rebar3* e todas as dependências necessárias, bem como a cópia de chaves ssh, um outro para tratar de clonar o repositório afeto ao projeto e instalação de todas as dependências relevantes, um *role* para compilar de forma automática o servidor de front-end e, por fim, um *role* para ser possível copiar os ficheiros do ZooKeeper necessários com as suas configurações.

Este deployment pode ser efetuado em qualquer lugar, sendo apenas necessário acesso remoto ao servidor e, preenchendo corretamente os campos necessários no *inventory.ini*, sendo depois todo o processo automático para várias máquinas em simultâneo.

11 Conclusão

Com o término do presente projeto, é admissível afirmar que a maioria dos objetivos definidos pelo grupo foram alcançados, com a exceção do melhoramento de performance da implementação do protocolo *BitTorrent* na biblioteca ttorrent, a implementação do servidor *bootstrap* responsável por obter a lista atualizada de servidores, e as suas respetivas localizações, a eliminação de ficheiros de acordo com o tempo de ocupação e a remoção de grupos e dos respetivos membros.

A qualidade do trabalho apresentado agrada bastante ao grupo que, apesar das limitações de tempo sentidas, conclui que as funcionalidades fundamentais do sistema, nas quais o grupo incidiu a maior parte do seu tempo, se encontram implementadas.

Análise do resultado final leva o grupo à conclusão que a versão completa do sistema idealizado poderia ser aplicada, por exemplo, num sistema de *backups* num *datacenter*, ou como um mecanismo de partilha de ficheiros em ambiente empresarial, abrangendo apenas a rede local. Evidentemente que uma versão do serviço disponível ao público geral também seria uma opção viável.

[6] https://www.ansible.com/