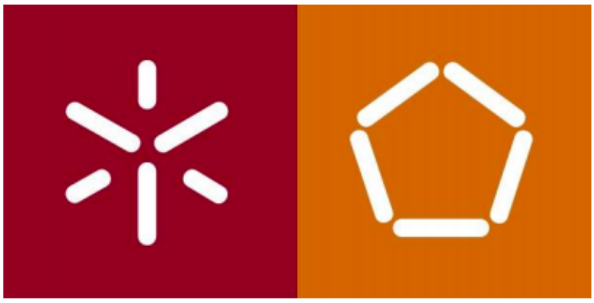

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Sistemas Distribuídos em Larga Escala

Timeline Descentralizada

Eduardo Lourenço da Conceição (A83870)
Ricardo Filipe Dantas Costa (A85851)
Rui Nuno Borges Cruz Oliveira (A83610)
Cândido Filipe Lima do Vale (PG42816)

08/06/2021
Braga

Índice

1 Introdução

O objetivo deste projeto é a criação de um serviço de *timeline* descentralizado, tendo como exemplo os serviços de redes sociais mais populares hoje em dia como o *Twitter*, *Facebook* ou o *Instragam*. Neste tipo de serviços existe uma enorme quantidade de informação a circular pela rede e o número de utilizadores também é bastante elevado. Tendo em conta este aspetos é facilmente visível que um tipo de solução centralizada para resolver este problema será bastante ineficiente, devido á inevitável existência de um *bottleneck* no servidor central, dado que seria este a tratar de todos os pedidos que lhe fossem enviados de todos os utilizadores, além de ter que reencaminhá-los também para vários utilizadores. Assim, uma solução possível para resolver este tipo de problemas é considerar uma arquitetura P2P (*Peer-to-Peer*), na qual além dos utilizadores desempenharem o papel de clientes, também podem funcionar como servidores para outros clientes e responder aos seus pedidos.

Admitindo uma solução deste tipo para o contexto em que nos encontramos, podemos pensar nos utilizadores como entidades que além de realizarem as suas funções normais neste tipo de serviço, como escrever uma publicação, também terão responsabilidades de armazenar informação de outros utilizadores e disponibilizar a outros utilizadores não só a informação referente a si mesmo, mas também estas informações provenientes de outros utilizadores. Sendo necessário também ter em conta o armazenamento disponível de cada utilizadores, faz também sentido pensar que a informação proveniente de outros utilizadores não será guardada para sempre, havendo um mecanismo que permita, por exemplo, eliminar as publicações mais antigas de outros utilizadores.

Assim, neste relatório serão abordados os aspetos tidos em consideração na escolha da arquitetura para este serviço, as várias componentes que foram implementadas e as ligações que existem entre elas.

2 Análise Inicial e Estratégia

Inicialmente efectuamos uma análise do projecto proposto de modo a perceber as características do sistema e de que modo poderíamos aplicar os conhecimentos adquiridos de forma a construir um sistema escalável mas que ao mesmo tempo mantivesse algum nível de confiabilidade.

Analisando o objetivo do projeto, criar um serviço de *Timeline* Descentralizada similar à utilizada em redes sociais como o Facebook podemos perceber que este serviço tal como o Facebook teria potencialmente, um grande número de utilizadores, nós com diferentes capacidades computacionais e uma alta taxa de entrada e saída de nós (*churn*), algo que temos de ter em consideração na nossa solução.

Foi também perceptível que o funcionamento de este tipo de sistemas se assemelha ao paradigma *publisher/subscriber* em que todos os utilizadores podem ser simultâneamente *publishers* e *subscribers*, sendo que quando publicam mensagens publicam para o grupo de utilizadores que os seguem e quando recebem mensagens, recebem-nas porque fazem parte do grupo de subscritores de um determinado utilizador, ou seja

todos os utilizadores existem um grupo associado e cada utilizador pode escolher ser parte do grupo dos utilizadores que esteja interessado em receber mensagens.

Com base nesta característica, consideramos que uma solução potencialmente interessante seria utilizar um **protocolo de comunicação em grupo** (conceito abordado noutra UC do perfil) com os conceitos aprendidos aos longo do semestre em Sistemas Distribuídos em Larga Escala para conseguir desenvolver um serviço de Timeline Distribuído que fosse escalável mas ao mesmo tempo confiável.

A utilização de um protocolo deste tipo irá permitir criar, para cada um dos utilizadores, a noção de grupo sendo que outros que estejam interessados no conteúdo desse utilizador se podem juntar a esse grupo. Isto também permitirá com relativa facilidade, cumprir o requisito definido de o conteúdo de um determinado utilizador estar disponível enquanto algum dos seus seguidores estiver online uma vez que estes se manterão no grupo enquanto estiverem online.

Para isto faremos uso do **Spread Toolkit**, um protocolo de comunicação em grupo que implementa a primitiva de *Extended View Synchrony* e que nos permitirá, para cada utilizador, estabelecer uma noção de grupo que será constituído pelos seus subscritores sendo que qualquer utilizador poderá a qualquer momento juntar-se a este grupo (subscribe) ou abandoná-lo (unsubscribe). Este protocolo permitirá, em cada momento saber que elementos de um determinado grupo estão online, ou seja, que que seguidores de um determinado utilizador estão *online* e portanto, cumprir com o requisito definido que as publicações de um determinado utilizador deve estar acessíveis a novos interessados mesmo que este esteja *offline*, desde que alguns dos seus seguidores esteja online, pois bastará a esse novo interessado juntar-se ao grupo de seguidores desse utilizador e fazer uma requisição do conteúdo a algum dos membros desse grupo que esteja *online*.

Além disto, este *toolkit* apresenta um sistema de comunicação escalável o que é muito importante neste contexto, dado exponencial crescimento da informação a circular na rede com o crescimento de número de utilizadores do nosso serviço. Tendo isto em conta, é necessário que as tecnologias utilizadas consigam lidar decentemente com este crescimento. O *Spread* também possui outras utilidades que permitirão simplificar bastante a nossa implementação, como a possibilidade de manter uma ordenação causal e FIFO das mensagens transmitidas.

Relativamente à comunicação entre diferentes grupos esta é realizada pelo *Spread* através de *daemons* aos quais os utilizadores se podem conectar.

3 Arquitetura

Com base na estratégia definida e tentando otimizar ao máximo a rede P2P do nosso serviço utilizando **Spread**, analisamos as topologias de redes P2P analisadas ao longo do semestre, DHT (Distributed Hash Tables) e Superpeers. Através desta análise percebemos que uma topologia baseada em *Super Peers* permitiria reduzir o nº de ligações entre os nós e também o nº de mensagens enviadas em cada post.

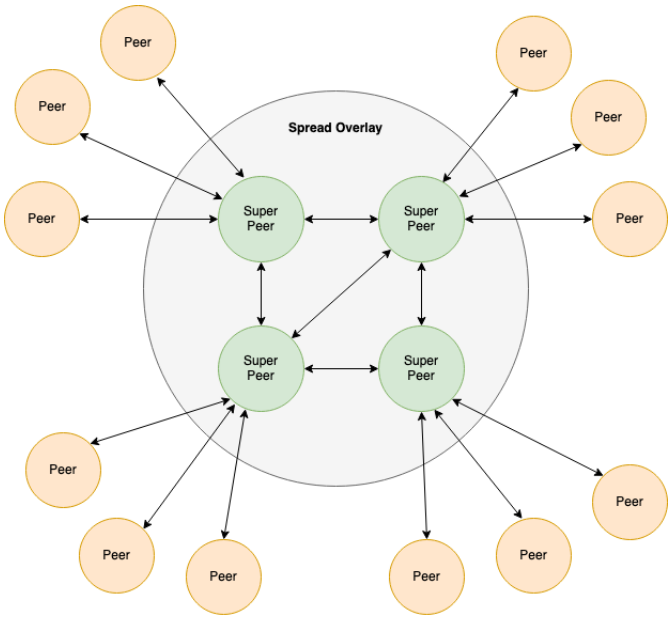

Figure 1: Arquitectura

Isto porque tendo em conta que a comunicação utilizando *Spread* é feita com base em *daemons*, caso cada user corresse o seu próprio *daemon*, e quisesse enviar mensagens para todos os seus seguidores, o seu daemon teria de comunicar com os daemons de todos os seus seguidores.

Ao definir uma arquitectura baseada em *Super Peers*, estes *Super Peers* passamos a ter nodos que para além de das funcionalidades que desempenhariam numa rede puramente P2P serão também responsáveis pelo routing e aos quais os restantes utilizadores se ligarão de modo a poderem participar na rede.

No nosso caso, esta topologia de *Super Peers* pode ser implementada elegendo Super Peers que serão nodos responsáveis por correr os *daemons* do spread aos quais os outros nós se ligarão de modo a poderem subscrever e enviar posts para outros utilizadores. Isto permite que quando um utilizador A fizer um Post, este Post é enviado para o *daemon* do seu *Super Peer* que vai tratar de fazer o routing dessa publicação, enviando-a para todos os subscribers do utilizador A para todos os outros *daemons* aos quais estejam ligados utilizadores que tinham subscrito o utilizador A. Esta topologia tem a vantagem de serem apenas necessários três hops para efetuar a entrega de uma mensagem (Daemon ao qual o Publicador está ligado - Daemon de Destino - Subscritor)

Esta topologia tem a vantagem de permitir tirar partido da heterogeneidade de nós na rede, elegendo para *Super Peers* os nós mais estáveis e com mais capacidades computacionais da rede, no entanto tem a desvantagem de ser baseada em *flooding*, o que implicará que as mensagens terão que ser potencialmente enviadas para todos os *Super Peers* o que implicará um maior *overhead* na rede.

Apesar disto, consideramos que o facto de o **Spread** ser um protocolo de comunicação escalável e de alto desempenho que garante a entrega das mensagens irá permitirá reduzir alguns dos problemas encontrados nestas topologias como por exemplo a possibilidade de termos mensagens a circular para sempre, continua a ser uma solução viável, bastante tolerante ao *churn* e que, com um rácio adequado entre *peers* e *Super Peers* pode permitir ter uma serviço com alguma escalabilidade (não tanta como nas topologias baseadas em DHT mas teríamos a vantagem de conseguirmos lidar melhor com o *churn* e a constante reconfiguração da rede).

Desta forma, definimos um serviço de timeline descentralizada com dois níveis: Super Peers que serão utilizadores potencialmente com maiores recursos e que irão executar o *daemon* do Spread e Peers que executarão a aplicação de Timeline Descentralizada e que se conectarão aos *Super Peers*.

Para permitir a entrada de novos utilizadores na rede P2P e tendo em consideração que estes podem não conhecer nenhum dos utilizadores já presentes na rede, adicionamos também um servidor de *Bootstrap*, que será o único componente centralizado da nossa infrastrutura e que funcionará como ponto de entrada na rede e que será responsável por manter os registos de todos os utilizadores e *Super Peers* na rede e pela autenticação dos utilizadores.

4 Tecnologias Utilizadas

As tecnologias que decidimos utilizar para implementar este projeto e tendo em conta a nossa arquitetura definida foram a linguagem de programação *Java*, o *Spread Toolkit* e o *Atomix*. O motivo para o qual estas tecnologias foram utilizadas será descrito de seguida para cada uma delas.

4.1 *Java*

Além da experiência existente pelos elementos do grupo nesta linguagem adquirido em unidades curriculares anteriores, consideramos também que devido às diversas bibliotecas e *frameworks* existentes para lidar com problemas no contexto de Sistemas Distribuídos, a escolha de *Java* seria uma opção viável e que nos traria confiança para realizar este projeto.

4.2 *Spread Toolkit*

Já descrito acima.

4.3 Atomix

Relativamente à comunicação com o servidor de *Bootstrap*, que é o único ponto centralizado da nossa arquitetura como foi explicado, decidimos utilizar uma biblioteca de comunicação assíncrona baseada em eventos. Isto será útil na medida em que não precisámos de estar a criar um canal de comunicação síncrono para as operações realizadas entre os utilizadores e o *Bootstrap*, dado que estas operações serão simples e

maioritariamente de autenticação dos utilizadores no serviço ou da sua respetiva saída do serviço.

5 Desenvolvimento do Serviço de *Timeline*

Para descrever o processo de implementação do serviço de *Timeline* Descentralizado, iremos começar por enumerar e explicar cada um dos componentes que faz parte deste serviço, a sua adaptação à arquitetura escolhida e numa fase posterior as relações existentes entre cada um deles e o funcionamento geral da aplicação.

5.1 Componentes

5.1.1 User

Este é o componente mais elementar do nosso projeto, já que representa a noção de um utilizador do nosso serviço. Considerando ainda que estámos perante uma arquitetura *Peer-to-Peer* este componente torna-se ainda mais relevante, dado que além de ser o componente elementar como mencionado, é também o mais importante e o que desempenha a maioria das funcionalidades do nosso sistema, dado que atua não só como cliente do serviço, mas também como servidor. Dado isto, um *User* não é mais que um *Peer* na nossa arquitetura.

5.1.2 Followers e Following

Para ser possível representar as relações existentes que um utilizador possui numa rede social, como escrever uma publicação e esta ser recebida pelas pessoas que o seguem, ou a situação contrária, foram criados dois componentes que representam as funcionalidades que um utilizador possui enquanto "pessoa que é seguida por outras pessoas" e "pessoa que segue outras pessoas".

Assim, o componente *Followers* identifica o papel do utilizador enquanto "pessoa que segue outras pessoas". Neste papel o utilizador irá armazenar temporariamente as publicações escritas por outros utilizadores que ele segue e pode também reencaminhar essas mesmas publicações para outros utilizadores.

Por outro lado, o componente *Following* representa o papel do utilizador enquanto "pessoa que é seguida por outras pessoas", ou seja, o utilizador neste caso irá escrever publicações, enviá-las aos seus seguidores e armazenar as suas publicações de forma persistente.

5.1.3 Superuser

Um *Superuser* não é nada mais do que um *User* com responsabilidades acrescidas. Neste caso, um *Superuser* irá funcionar como um intermediário entre utilizadores, no entanto, não deixa ele próprio de ser um utilizador também, podendo na mesma publicar, seguir outros utilizadores e ser seguido.

Do ponto de vista de um cliente do nosso serviço, não existe nenhuma diferença entre *Superuser* e *User*, já que esta informação não seria transparente para um cliente normal caso fosse executado o *deployment* da nossa aplicação.

5.1.4 Bootstrap

O servidor de *Bootstrap*, como foi referido anteriormente, é o único ponto centralizado do nosso sistema e tem como função permitir a autenticação dos utilizadores na rede, além de guardar informações como o número de *users* e*superusers* que estão *online* num dado momento na aplicação, para questões da gestão dos *superusers*. Importante realçar que o servidor de *Bootstrap* foi desenvolvido tendo em consideração que teria de realizar o menor processamento possível, servindo apenas para comunicar decisões aos utilizadores, como por exemplo se está autenticado ou não.

A razão pela qual foi necessário criar um ponto central foi para permitir o conceito de individualidade dos utilizadores, que é dado pelo seu *username* escolhido que tem de ser único, sendo esta informação armazenado no *Bootstrap*.

5.2 Funcionamento do Serviço

De seguida será descrito o funcionamento geral da *Timeline*, descrevendo as funcionalidades existentes na nossa aplicação, como foi realizada a gestão dos *Superusers* existentes e também as decisões tomadas em relação à persistência e à efemeridade da informação existente em cada um dos utilizadores do serviço.

5.2.1 Funcionalidades

As funcionalidades existentes na nossa aplicação são as seguintes: registar, *login*, *logout*, publicar um *post*, subscrever e cancelar a subscrição a um dado utilizador e a funcionalidade de ver as publicações perdidas enquanto esteve *offline* ou que ainda não viu.

- **Registar:** esta funcionalidade é a primeira que um novo utilizador que queira entrar na nossa aplicação tem de realizar. Se um utilizador não estiver registado não existe informação relativa a si no servidor de *Bootstrap* e como tal não irá conseguir realizar *login* e utilizar a nossa aplicação. Além disto, ao registar-se, o *Bootstrap* verifica se o nome de utilizador escolhido não existe na aplicação, dado que é o nome de utilizador que permite garantir a individualidade dos utilizadores. A partir do momento que o registo é bem sucedido, é executado o processo de *login* internamente na aplicação e o utilizador consegue aceder ao resto das funcionalidades sem ter de efetuar *login*.

- ***Login*:** Tendo um utilizador se registado anteriormente, é possível agora apenas realizar o processo de *login*. Neste caso, o utilizador comunica com o *Bootstrap* e são verificadas se as credenciais fornecidas estão corretas (*username* e *password*). Quando a autenticação é bem sucedida, o *Bootstrap* irá atribuir um *SuperUser online* ao utilizador que acabou de entrar ao qual ele se vai conectar,

ligando-se ao *daemon* respetivo desse *Superuser* e juntando-se ao grupo denominado "<*username* do *Superuser*>_SUPERGROUP".

De seguida, junta-se ao seu grupo de seguidores, denominado por "Group<*username* do *User*>" e junta-se aos grupos dos utilizadores que está a seguir, que têm a mesma nomenclatura mas com o respetivo *username*.

- ***Logout*:** Quando um utilizador pretender realizar o *logout* é enviada uma mensagem ao *Bootstrap* para diminuir no seu contador interno o número de utilizadores *online* e sai de todos os seus respetivos grupos, desconectando-se também do *daemon* que lhe tinha sido atribuído.

- **Subscrever:** Um utilizador pode subscrever a *timeline* de outro dado utilizador, passando a partir deste momento a receber as respetivas publicações que esse utilizador efetuar. Para isto, apenas necessita de entrar no respetivo grupo do utilizador que pretende subscrever, usando a nomenclatura mencionado anteriormente. Assim, um cliente apenas precisa de saber o nome do utilizador ao qual pretende subscrever.

- **Cancelar a subscrição:** Para isto é apenas necessário sair do grupo respetivo, sendo preciso apenas o nome do utilizador ao qual pretende cancelar a susbcrição. A partir do momento que sai, deixa de receber as publicações efetuadas por esse utilizador.

- ***Unseen* (publicações que não viu):** a última funcionalidade presente na nossa aplicação é a de permitir ver todas as publicações de todos os utilizadores que está subscrito que ainda não viu. O caso mais simples é, por exemplo, depois de subscrever a um dado utilizador, quer verificar quais as publicações que esse utilizador escreveu e pode utilizar esta funcionalidade para tal. Publicações que já tenha visto não serão apresentadas novamente, a menos que já não possua nenhuma publicação desse utilizador guardada localmente. Esta funcionalidade também permite, por exemplo, a um utilizador visualizar todas as publicações que perdeu enquanto esteve *offline*.

  Além disto, mesmo que o utilizador a quem está subscrito não esteja *online*, é na mesma possível obter as publicações desse utilizador, dado que este pedido é também realizado aos outros seguidores do utilizador que estamos também a seguir, dado que todos pertencem aos mesmo grupo. Neste último caso, não é garantido que a informação enviada seja a mais recente, dado que esse outro seguidor pode não ter as publicações mais recentes.

5.2.2 Gestão de *Superusers*

Para permitir a escalabilidade do nosso serviço foi necessário ter em consideração quando é que um dado *user* passaria a desempenhar também as funções de *Superuser*. Esta decisão será em parte tomada pelo *Bootstrap* mas em certas ocasiões os *Superusers* também terão responsabilidade em eleger um novo *Superuser*. Em seguida, iremos verificar todos os casos em que um dado utilizador passa a *Superuser*.

Antes de mais e tendo em conta a nossa arquitetura é fácil de perceber que pelo menos teremos sempre de ter pelo menos um *SuperUser online* no nosso sistema. Assim, assumindo que não está nenhum utilizador *online*, o primeiro *User* que entrar no serviço tornar-se-á automaticamente um *Superuser*. Esta decisão é tomada pelo servidor de *Bootstrap* que possui informações relativamente a quantos utilizadores estão *online*. No nosso caso em específico, decidimos também que o segundo utilizador a entrar, assumindo que apenas está mais um utilizador *online* no sistema, será também um *SuperUser*.

Uma outra situação em que um *User* que entra irá passar a *Superuser* é quando o rácio entre o número de *users* e *superuser* for menor que um determinado número. Para efeitos de teste, consideramos que este número fosse bastante baixo, mas num cenário normal seria necessário determinar qual o número ótimo do rácio. A única função do *Bootstrap* neste caso é calcular este rácio e requisitar a um *Superuser* aleatório para realizar um processo de eleição entre os *users* que estão conectados a si. Este processo de eleição será explicado no último caso.

A última ocasião em que um dado *User* passa a *Superuser* é quando um *Superuser* sair do sistema. Como já sabemos qualquer utilizador ao sair do sistema envia essa informação ao *Bootstrap* e este atualiza o seu contador de utilizadores *online*. No entanto, no caso em que um *Superuser* sai do sistema, além de atualizar o contador, o *Bootstrap* também retira esse *Superuser* de uma estrutura onde tem guardado todos os *Superusers online* no sistema. O próprio *Superuser* antes de efetivamente sair da aplicação irá correr um processo de eleição para eleger um dos *users* conectados ao seu *daemon* para se tornar o novo *Superuser*. Este processo de eleição é feito tendo em conta o número de *cores* de cada utilizador e a quantidade de tempo que está *online*. De notar que como estamos a correr todos os utilizadores na mesma máquina, para efeitos de teste, foi tido em consideração um valor aleatório para multiplicar pelo número de *cores* para obtermos resultados variados.

Apesar de não termos implementado, seria também importante garantir que os Super Peers nunca têm um numero de users ligados muito superiores ao valor do rácio definido. Isto é importante pois a estar continuamente ligado à rede, um determinado Super Peer pode ir acumulando conexões de peers que se vão mantendo ao longo do tempo, se isto não for controlado pode levar a que um determinado Peer tenha um nº exagerado de conexões i que torna a solução menos eficiente.

5.2.3 Armazenamento da informação

Como sabemos cada utilizador guarda localmente as publicações que realizou e as publicações que recebeu de outros utilizadores aos quais está subscrito. Antes de mais, para não perder esta informação quando efetua o *logout* do sistema, antes de efetivamente sair, é realizador um processo de serialização da informação para formato JSON e é guardada de forma persistente em ficheiro. Para cada utilizador existem 2 ficheiros, relativos às componentes *follower* e *following* já explicadas anteriormente. No ficheiro relativo à componente *following* é armazenado as publicações que já realizou e no outro componente são armazenadas os utilizadores que está a seguir no momento e também as publicações que possui guardadas de cada um desses utilizadores no momento de saída do sistema. Quando este mesmo utilizador pretender entrar novamente

na aplicação é realizado o processo oposto de desserialização e estas informações são novamente carregadas para memória. Desta forma nenhuma informação é perdida pelo utilizador.

No entanto, é necessário considerar que como estamos perante um sistema que queremos que seja escalável e que consiga suportar grandes quantidades de utilizadores e consequentemente, ainda maiores quantidades de informação, não é viável a partir de um determinado ponto o utilizador estar a guardar toda esta informação. Assim podemos considerar que há determinada informação que não há problema em ser perdida e podemos então eliminá-la. Como é de extrema importância que cada utilizador mantenha a informação relativa às suas próprias publicações guardada de forma permanente para que qualquer utilizador que o subscreve possa receber essa informação, descarta-se esta hipótese de eliminar as próprias publicações. Assim, e também tendo isto em conta, podemos eliminar a informação relativa a outros utilizadores que está guardada em nós, já que essa mesma informação continuará disponível nesses utilizadores.

Para realizar este processo, definimos um *Timer* que irá de tempo em tempo eliminar esta informação referida que seja mais antiga que uma determinada quantidade de tempo que pode ser ajustada, de forma a garantir que as publicações mais recentes permanecem armazenadas.

6 Conclusão

No geral consideramos que conseguimos cumprir os objetivos deste trabalho prático, isto é, criar um serviço de *Timeline* Descentralizado que permita aos utilizadores escreverem e receberem publicações tendo em conta se estão ou não subscritos a outros utilizadores, sem ser criado um servidor central pela qual passassem todos os pedidos, o qual, não seria bem sucedido à medida que o número de utilizadores do sistema fosse aumentando.

O uso da tecnologia *Spread Toolkit* permitiu facilitar este processo devido à conceito de grupo existente já por si neste tecnologia o qual se aplicava muito bem ao que era proposto para este projeto. Com o uso do *Spread* subscrever ou cancelar subscrição de um dado utilizador tornou-se tão simples quanto entrar ou sair de um dado grupo.

Usamos o *Spread* através um *docker* o que nos limitou o número de *daemons* que poderiamos utilizar, não permitindo realizar testes a uma maior escala como gostariamos. No futuro, seria necessário procurar por uma solução que nos permitisse correr mais *daemons* para podermos melhor testar o desempenho da solução criada.

Outro aspeto que poderiamos melhorar seria a interface, dado que esta apenas ficou como uma interface básica de terminal, sendo bastante restrita nas operações de registo e *login* por exemplo. Por exemplo, ao entrar no registo já não é possível voltar para trás, ficando aí presos se efetivamente não quisermos efetuar um novo registo. Tendo isto em conta, a própria interface do terminal poderia ser melhorada.

Por fim, outro aspeto que gostaríamos de ter realizado de forma diferente é o facto de na funcionalidade de visualizar as publicações que ainda não vimos, nós realizamos esse pedido para todos os membros desse dado grupo. Se tirássemos melhor proveito das funcionalidades de *membership* do *Spread*, conseguiríamos mandar este pedido somente ao utilizador que estamos a seguir diretamente e caso este não tivesse *online*,

aí é que escolheríamos um outro utilizador desse mesmo grupo para nos enviar essa informação.