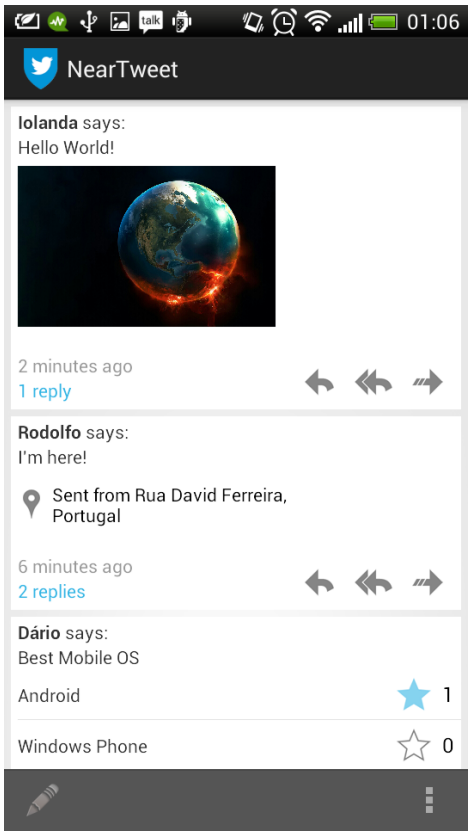
Projecto *NearTweet* – Relatório

Computação Móvel – 2012/2013

Grupo: 3
Dário Nascimento, Número: 68210, E-mail: dario.nascimento@ist.utl.pt (MERC)
Iolanda Correia, Número: 72593, E-mail: iolanda.correia@ist.utl.pt (MEIC)
Rodolfo Santos, Número: 76590, E-mail: rodolfo.santos@ist.utl.pt (MEIC)



1. Objectivos Alcançados

O projecto *Neartweet* tem como objectivo a criação de um sistema de *microblogging* para multidões através da criação de redes espontâneas.
A implementação deste projecto concretizou todos os requisitos funcionais propostos para o projecto *NearTweet*: Envio de *tweets*, resposta a *tweets*, publicação de *tweets* na rede social, *pool voting*, partilha de multimédia, recolha de localização GPS, câmara e *spam*.
Estes requisitos foram implementados quer para a arquitectura de comunicação cliente-servidor, quer para a arquitectura descentralizada, usando *Wi-Fi Direct*.

2. Especificação

O projecto desenvolvido focou-se na implementação da aplicação *NearTweet*.

A aplicação implementada permite o envio de mensagens até 160 caracteres segundo um nome de utilizador definido pelo próprio. Na primeira utilização, a aplicação sugere um nome genérico que pode ser alterado pelo utilizador em qualquer momento de execução da mesma.

Quando a aplicação se encontra a correr em *background*, o utilizador é notificado da recepção de novas mensagens através de *pull notifications*.

Uma mensagem pode conter diversos tipos de anexos, nomeadamente imagens, votações e/ou coordenadas de GPS. As imagens podem ser capturadas no momento, usando a câmara do dispositivo ou através de um URL ou a partir de qualquer galeria de imagens instalada no mesmo. As imagens em anexo podem ser vistas em *fullscreen* usando qualquer uma destas galerias. Assim como as coordenadas GPS da mensagem podem ser visualizadas no contexto de qualquer aplicação de mapas instalada no dispositivo.

A aplicação permite que os utilizadores respondam pública ou privadamente a qualquer mensagem. A *thread* de mensagens reflecte a ordem da conversação, gerindo individualmente a visibilidade das mensagens consoante a privacidade que lhes é atribuída. Acrescentou-se às funcionalidades propostas no enunciado, a possibilidade de actualizar a *thread* de conversação através dos gestos *pull down* e *release*.

O utilizador pode configurar o seu *login* do Facebook para efectuar a partilha de mensagens na rede social.

O utilizador pode denunciar mensagens de *spam* através de um mecanismo de voto. O utilizador não pode revogar o seu voto, nem votar mais do que uma vez na mesma mensagem. Após um determinado número de votos, o autor das mensagens perde a permissão de enviar mensagens.

3. Desenho

A aplicação é composta pelas componentes *User Interface*, *Network Management* e *Storage*.

A componente *User Interface* permite o acesso do utilizador às funcionalidade da aplicação, enquanto que a componente *Network Management* é responsável pela comunicação. A separação entre ambas justifica-se pela necessidade de desenvolver a aplicação segundo uma arquitectura de comunicação cliente-servidor e, posteriormente, adapta-la a uma arquitectura descentralizada. Desta forma, apenas foi necessário modificar a componente responsável pela comunicação, preservando as funcionalidades da aplicação.

A componente *Storage* permite manter o estado da aplicação em contexto *offline*, assegurando a posterior sincronização das mensagens com os restantes utilizadores da rede *NearTweet*.

3.1. Arquitectura Cliente-Servidor

Esta arquitectura, representada na figura 1, consiste na existência de um servidor central, que trata das conexões dos clientes e distribui cada mensagem recebida para todos os dispositivos clientes.

No lado do cliente, a entidade *NetworkTask*, pertencente à componente *NetworkManagement* trata das conexões, recepção e envio de mensagens de e para o servidor. Já no lado do servidor, existe uma entidade *ClientListener* que gere cada nova conexão de um cliente. Cada nova mensagem que chega ao servidor de um determinado cliente é colocada numa *queue FIFO*, e a entidade *ServerDispatcher* é responsável por fazer *multicast* de todas as mensagens presentes na *queue* para todos os clientes actualmente conectados. Além disso, o servidor mantém uma lista de todos as mensagens recebidas (na *Storage*), e quando recebe um pedido do tipo *GET*, envia todas as mensagens guardadas até ao momento para o cliente que efectuou o pedido.

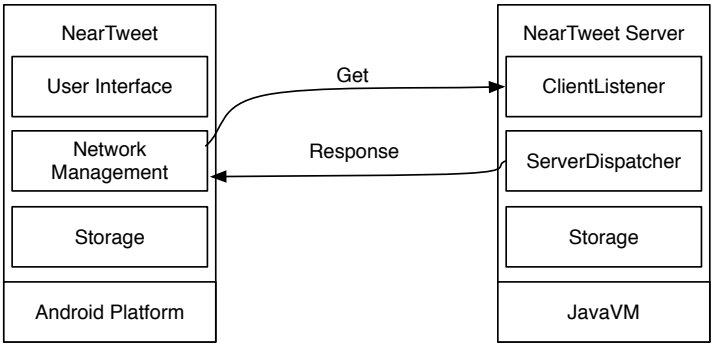


Figura 1: Arquitectura Cliente-Servidor da aplicação *NearTweet*.

3.2. Arquitectura *Peer-to-Peer*

A arquitectura *Peer-to-Peer* é suportada pela *framework WiFi Direct* que se encontra incluída em dispositivos com a versão superior ao *Android 4.0*. Esta *framework* que permite conectar directamente vários dispositivos *Wifi* sem um *Access Point* partilhado. O módulo *Network Management* implementa *WiFi Direct* com *Network Service Discovery (NSD)*.

Este módulo regista um serviço local para conectar ao *WiFi Direct* através do qual regista um serviço *NSD* com as informações da nossa aplicação e nome do utilizador.

Para se conectar a outros dispositivos, o módulo realiza a descoberta de serviços na rede local. Para cada serviço detectado, o módulo solicita detalhes do dispositivo provedor de serviço e inicia o estabelecimento de uma conexão por *WiFi-Direct*. Quando a conexão *WiFi Direct* fica disponível, o nó pode ser o *Group Owner* (GO), equivalente a um *Access Point* numa estrutura wireless comum, ou cliente. O *GO* é responsável por receber mensagens de qualquer um dos clientes da sua rede e de a reencaminhar a todos os nós clientes. Cada dispositivo guarda as conexões que tem com outros *GOs* ou com outros clientes.

Quando envia uma mensagem, o cliente faz *broadcast* para a rede, enviando-a cada um dos nós que conhece. Quando o destinatário recebe a mensagem, realiza o *broadcast* para os restantes dispositivos que também conhece, excepto para origem da mensagem. Para que o algoritmo de *flooding* convirja, o nó destinatário apenas encaminha mensagens que ele próprio desconhece.

Para evitar um cenário em que um novo nó não receberia as mensagens antigas caso estivesse conectado a um nó com estado mais recente, é utilizado um mecanismo de *GET* diferencial. Neste mecanismo, o novo

cliente efectua o *broadcast* da lista de identificadores de *tweets* que possuí. Cada destinatário compara a sua lista de *tweets* com a lista recebida, enviando os *tweets* que o requerente não tem e pedindo os que ele próprio desconhece. Deste modo, garante-se que os *peers* sincronizam as mensagens entre si, bastando uma conexão entre dois grupos para que eles se interliguem.

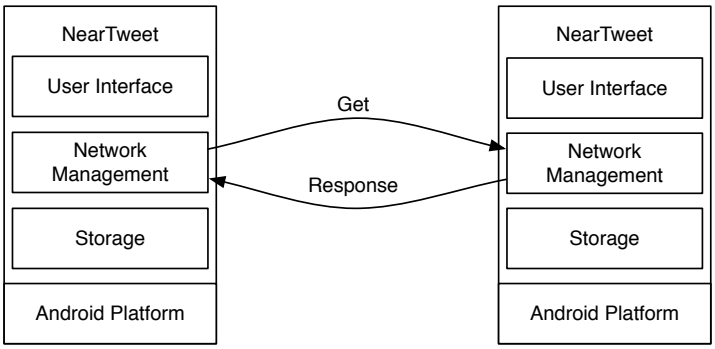


Figura 2: Arquitectura *Peer-to-Peer* da aplicação *NearTweet*.

4. Implementação

4.1. Interface

A aplicação é maioritariamente executada na sua *FragmentActivity* principal, onde é apresentada a *thread* que reflecte a ordem da conversação. Esta é composta por um *LinearLayout* vertical, contido dentro de um *PullToRefreshScrollView*, disponibilizado pela biblioteca *Android-PullToRefresh*[1]. A razão pela qual se optou pela utilização desta biblioteca, prendeu-se com o facto de estender o comportamento do *ScrollView*, disponibilizado pela API do *Android*, com a funcionalidade de actualização da *thread*, familiar ao utilizador, dada a sua utilização em diversas aplicações com o mesmo contexto.

O *LinearLayout* vertical constitui a raiz das mensagens recebidas pela aplicação. Na raiz, as mensagens são apresentadas da mais recente para a mais antiga, de forma a que as mensagens mais recentes se mantenham no topo da interface. Pelo contrário, as respostas às mensagens são apresentadas da mais antiga para a mais recente, de modo a ilustrar a ordem da conversação de uma forma mais natural.

A *view* de uma mensagem é especificada no seu próprio ficheiro XML, para que esta possa ser renderizada dinamicamente. Assim, aquando da recepção de uma mensagem, a *view* é *inflated* e todos os seus componentes são afectados de acordo com o conteúdo da mesma. Esta *view* contém os botões necessários para que o utilizador possa efectuar uma resposta pública ou privada.

A *view* de cada mensagem possui um *LinearLayout* vertical que, à semelhança do *LinearLayout* que constitui a raiz da *thread*, irá conter todas as respostas à mesma. Para que, aquando da recepção de uma resposta, seja possível identificar o *layout* onde esta deverá ser inserida, esta encontra-se referenciada com o identificador único da mensagem à qual pertence através de um *tag*. Sendo que cada resposta possui, para além do seu próprio identificador, o identificador da mensagem à qual responde. Assim é possível identificar o *layout* onde esta deverá ser inserida. O *layout* esta indentado à esquerda por forma a reflectir a conversação.

A aplicação possui um menu através do qual é possível enviar novas mensagens, alterar o nome de utilizador ou consultar informações sobre a mesma. Estas opções são disponibilizadas ao utilizador através de *popups* do tipo *AlertDialog*.

[1] Chris Banes, Android-PullToRefresh library (https://github.com/chrisbanes/Android-PullToRefresh)

4.2. *Network Management*

Para permitir a transmissão de mensagens entre dispositivos, foi utilizado o padrão de desenho com recurso a *DTOs*[2]. Através de um *DTO*, foi possível utilizar o mecanismo de *marshalling* e *unmarshalling* sobre *sockets Java*. O *DTO* é representado por uma classe abstracta *TweetObject*, que é responsável por guardar o *timestamp*, o *UUID* do utilizador que a criou e o tipo de mensagem na rede. Esta classe abstracta, é estendida pelos vários *DTOs* utilizados na transmissão de mensagens na rede, nomeadamente: *TweetGet*, *TweetMsg*, *TweetPollVote* e *TweetSpamVote*.

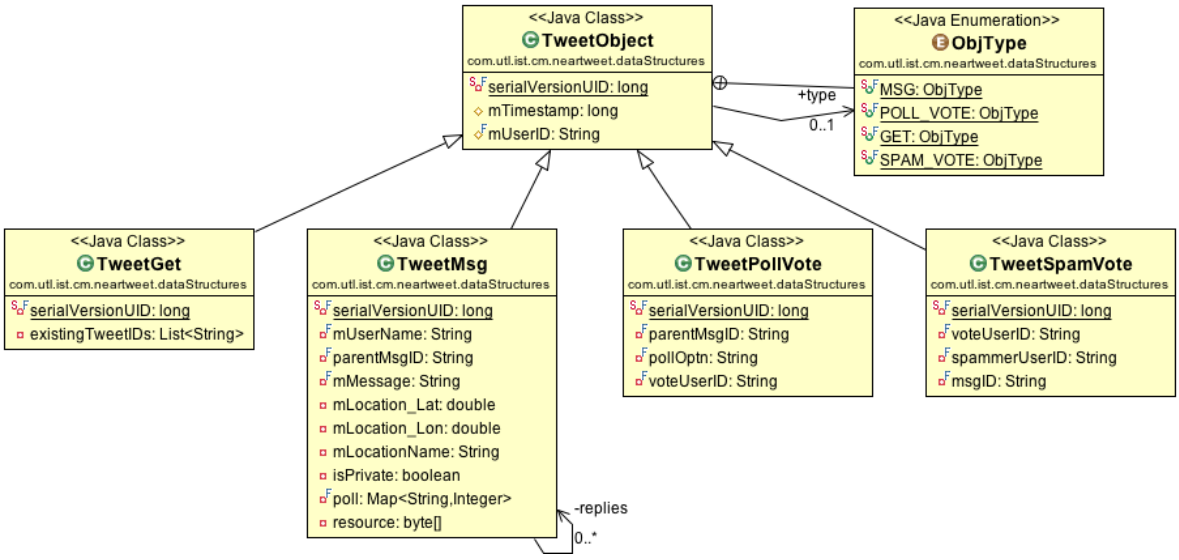


Figura 3: *Data Transfer Object - TweetObject*

Na implementação final, utilizando a arquitectura descentralizada, recorreu-se à funcionalidade *WiFi Direct*.

A maior complexidade do *WiFi Direct* reside na formação de um grupo, ou seja, na eleição de um *Group Owner* que irá gerir o espaço de endereços.

Dividiu-se este processo em três fases: serviço de descoberta, *GO negociation* e configuração de endereços. O serviço de descoberta, providenciado pela norma wireless *IEEE 802.11*, permitiu troca de pacotes para as próximas fases. A *GO Negociation* consiste na eleição do coordenador da ligação, de entre os dispositivos existentes. Esta negociação pode ser simplificada por um critério de ordenação como o endereço *MAC*. A distribuição de endereços é realizada através de um protocolo como o *DHCP*.

O *WiFi-Direct* está concebido para uma ligação ponto a ponto entre o *Client* e o *GO*. O encaminhamento entre clientes é da responsabilidade do programador. Por isso, quando é estabelecida uma ligação, se o dispositivo for eleito *GO*, é criado um *Server Socket* ao qual os clientes se ligam e são criadas *threads* individuais, que estabelecem uma conexão entre o cliente e o *GO* e tratam do processamento das mensagens. Tanto o cliente como o *GO*, guardam todas as conexões estabelecidas, por isso qualquer mensagem recebida, que não exista no dispositivo, é encaminhada para todas as conexões que o nó conhece.

Assim, implementou-se um mecanismo de *flooding* que tolera a entrada e a saída de *peers*, mantendo o estado na *Storage* e fazendo *GET* diferencial.

[2] *Data Transfer Objects*

4.3. *Storage*

A componente *Storage*, representada pela classe *DB*, permite manter o estado da aplicação em modo *offline*. Para tal, todas as mensagens enviadas e trocadas na rede são guardadas em *storage* persistente. Desta, forma o utilizador pode continuar a utilizar a aplicação, consultando as mensagens que se encontram na *thread* e enviando novas mensagens. O sistema garante que a sincronização das mensagens é efectuada no momento em que a aplicação transita para o estado *online*.
O armazenamento das mensagens é efectuado através de um conjunto de meta-dados, que guarda os identificadores de todas as mensagens. Associado a cada meta-dado é também guardada a respectiva mensagem. O armazenamento dos meta-dados é realizado com recurso à serialização de uma lista de identificadores de mensagens, ordenada pela ordem de chegada ao dispositivo. O armazenamento das mensagens é efectuado com recurso à serialização de cada *TweetObject*, identificado pelo respectivo meta-dado.
Quando ocorre uma nova conexão de rede, é feita a sincronização de estado entre os dispositivos envolventes, onde se garante que novas mensagens são enviadas ao novo dispositivo (*deviceB*), e que as mensagens do novo dispositivo são recebidas pelo dispositivo actual (*deviceA*).
Sendo A o conjunto de mensagens presentes no *deviceA*, e B o conjunto de mensagens presentes no *deviceB*, o procedimento de sincronização é realizado através do envio de um *TweetGet* com uma lista de identificadores das mensagens presentes no *deviceA*, sendo a lista identificada por A'. Depois de o *deviceB* receber a lista de identificadores provenientes do *deviceA*, este gera a sua lista de identificadores B' e efectua a diferença de conjuntos $B' - A'$, enviando todas as mensagens pertencentes ao conjunto $B - A$ para o *deviceA*. Para garantir a sincronização de mensagens também no *deviceB*, é efectuado o procedimento inverso, onde o *deviceB* efectua um *TweetGet* para o *deviceA*. O algoritmo de sincronização só termina quando todas as mensagens pertencentes a $A \cup B$ se encontram em ambos os dispositivos, ou seja, quando se verificar $A' = B' \Leftrightarrow A' - B' = 0$.

5. Avaliação Experimental

As funcionalidades implementadas foram extensivamente testadas durante a primeira fase de desenvolvimento do projecto, usando três dispositivos físicos (HTC One S, Nexus 4 e ZTE Libra) e diversos emuladores. Estes testes consistiram em trocas de mensagens, com diversos tipos de conteúdo, entre os vários dispositivos físicos e emulados. Durante a segunda fase de desenvolvimento, avaliou-se sobretudo a robustez da rede espontânea perante a troca de mensagens entre: nós estacionários; nós em movimento e recuperação; e sincronização do estado da aplicação perante a saída e reentrada de nós na rede (incluindo o nó *GO*).
Inicialmente, a avaliação foi efectuada usando os dispositivos HTC One S e Nexus 4, dado que o ZTE Libra não possui *Wi-Fi Direct*. Neste cenário, surgiram dificuldades aquando do estabelecimento da comunicação entre os dois dispositivos, nomeadamente após uma tentativa de recuperação do estado da conexão. As avaliações posteriores, efectuadas usando um Nexus 4 e dois Nexus 7, mostraram melhores resultados nos testes mencionados anteriormente, indicando que o problema pudesse estar relacionado com o facto de a *ROM* oficial da HTC não possuir o *update* mais recente do *WiFi Direct*.
Os testes revelaram que, apesar do mecanismo *Pull-to-Refresh* forçar a procura de serviços, por vezes é necessário reiniciar a aplicação para que as conexões sejam restabelecidas e o estado sincronizado.

6. Conclusões

No geral, considera-se que o projecto foi bem sucedido e que os objectivos propostos foram cumpridos, na medida em que todos os requisitos foram implementados conduzindo a soluções satisfatórias.
Os maiores obstáculos que surgiram, estão relacionados com o facto de a tecnologia *WiFi Direct* não estar ainda num estado suficientemente desenvolvido para lidar com uma arquitectura de comunicação com múltiplos nós, e na dificuldade de acesso constante a um numero adequado de dispositivos que reunissem as condições necessárias para testar os diversos cenários propostos.