Technical Documentation of BudCloAnd 1.0

Marvin Byfield
Yuwen Chen
Siyun Li

1 Introduction

Buddycloud is an open-source distributed online social network. Its protocols and specification allow anyone to host their own buddycloud and connect to other buddycloud servers and share content among their users. The server side is held by the Buddycloud Community and has a variety of different languages versions. Among them is an HTTP API server, which enables native web clients as well as mobile clients to set up their own hosting service efficiently.

In the Services of Internet Project, a basic prototype of client in platform of android for Buddycloud Http-API-Servers will be built. A couple of features are supported by this android application including authentication, display of posts and comments, commit new posts, subscription management, database management and online instant chat. Details about the implementation of these features are elaborated in the third section.

The rest parts of the documentation are organized as follows: In second section, the logic functional structure of BudCloAnd Client is elaborated. Followed is the Implementation section with detailed technical information of the implementations for each functional part. In the last section, a description of potential problems and possible optimizations will close this technical report.

2 Structure

In this section, a basic structure of functional unities is shown as in Figure 1.

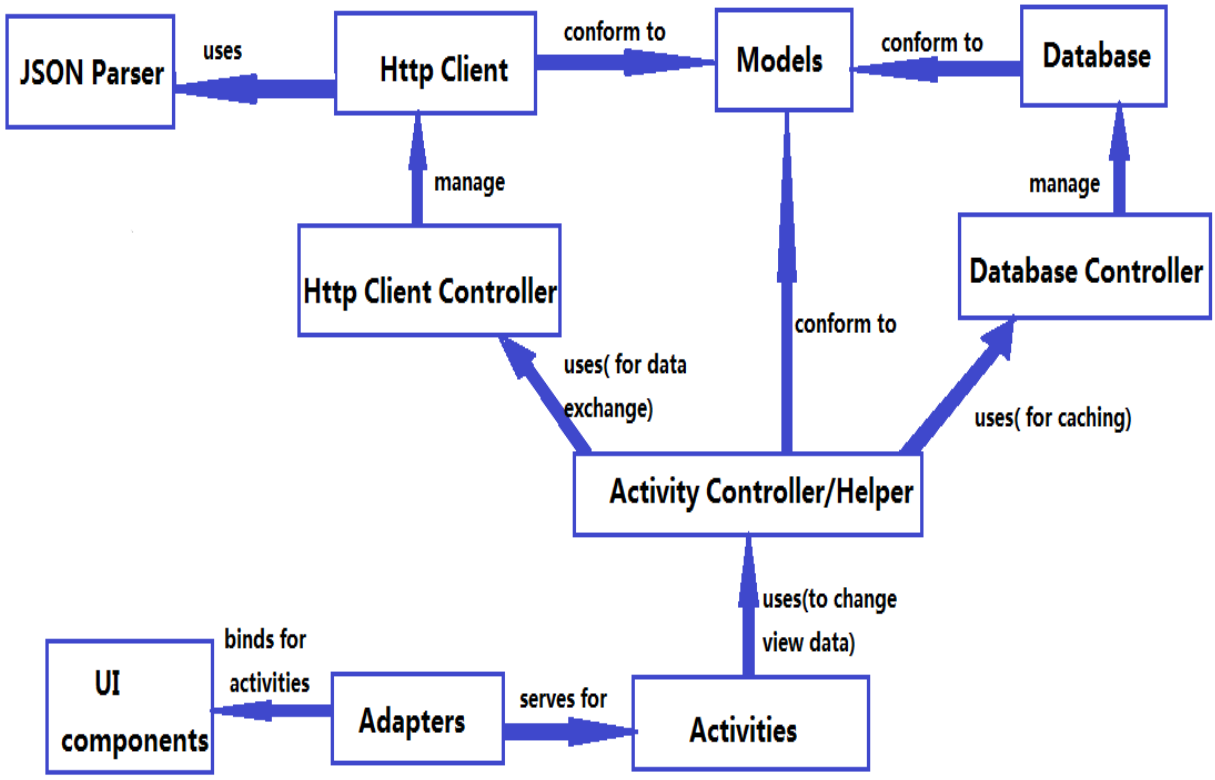

Figure 1. Structure of BudCloAnd Client

As shown in Figure 1, the basic three functional utilities(HttpClient Utility, Database Utility and Activity Set) conform to the models we design, and each of them is managed by their correspondent controller. JSON Parser is an additional unit to supporting some functions inside Http Client. Apart from above, there are adapters to bind the UI components with the data inside relevant activities.

3 Implementation

In this section, details about the implementation of BudCloAnd Client are elaborated. This section is divided into five subsections regarding the different aspects: Design of models, Activities, Controller and Helper, Data Exchange and Database. In each subsection, relevant interfaces provided and their usage are introduced as well as the principle and details of their implementation.

3.1 Design of Models

In this subsection, the principle of the design of models, which the other functional unities based on is introduced. As shown in Figure 2 below. BCItem, BCMediaData, BCMetaData and BCSubscribtion are the models corresponding to the json data format

on the server side. These classes are marked with yellow border. The database-relevant parts are marked with the color red, as shown in class channelPostsScope, CacheTimeFrame and some data fields in BC-* Models. The embedded database-relevant fields are not open to be changed manually since they are database relevant and modification of the value of them may cause a mass in the background database management. Apart from that, BCJSONObject serves as interface to providing relevant functions of JSON data, like getJSONObject() and getJSONString().

Figure 2. Definition of Models

3.2 Activities

In this subsection, the relevant activities that represent different features of BudCloAnd Client are introduced, including BasicActivity, InstantTalkActivity, LoginActivity, NewPostActivity, SearchActivity and ShowPostActivity.

3.2.1 LoginActivity

The first activity when enter the application, which communicating with the server to do the

authentication. When the account name in Email address pattern and correct password are offered, the user will be redirected into the Basic Activity.

Figure 3. Screenshot of LoginActivity

3.2.2 BasicActivity

The basic activity is based on a three-tab scroll view, which is similar as the web client of buddycloud. Left part of this view shows the channels a user subscribed to, with the home channel in top and a find channel button. The middle panel to be shown when enter the activity is a listview containing the posts of the user's home channel. The right panel contains some basic information of the current visiting channel. Several features are added to gain better interaction performance.

3.2.2.1 Pull-to-refresh-list

The post listview in middle panel is implemented as a pull-to-refresh list. This is done by adding a header view to the listview, which is hidden originally. The OnScrollListener will keep monitoring the touch event caused by user. When the list is pulled down at the top, the load-more function of the list adapter will be called to retrieve new posts in a new thread. And once the retrieving procedure ends, a message will be sent to the Main thread's handler to notify data change.

3.2.2.2 load-more button

The footer view containing a load-more button is positioned at the bottom of the whole

list. Due to memory limitation it is better not to load large set data in a single run, so this button is set to support memory management. It would be shown until all the posts in the channel is retrieved, and then removed automatically.

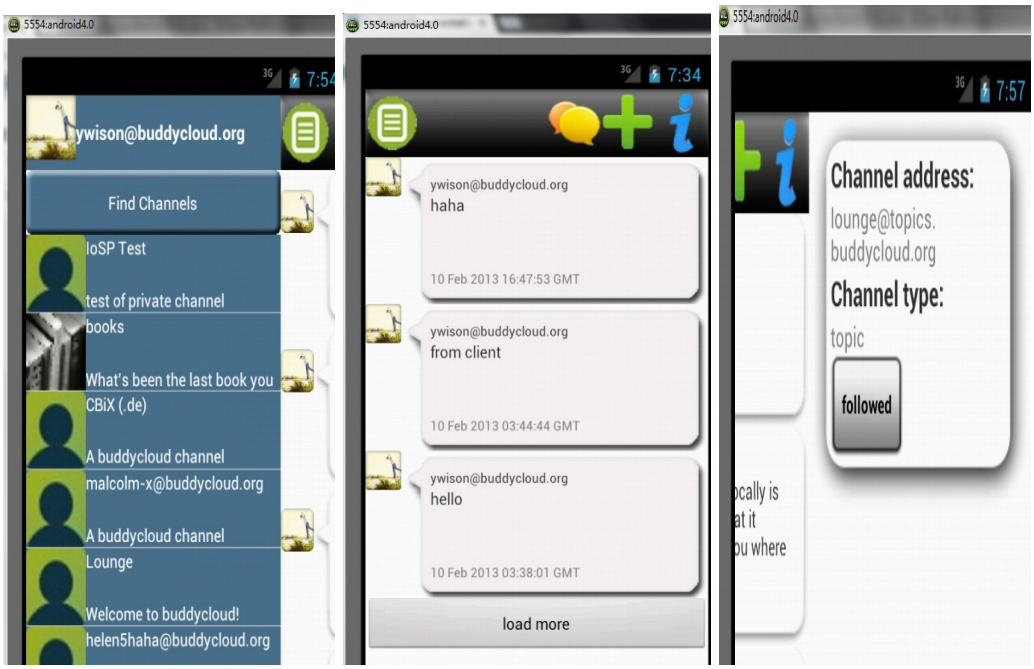

Figure 4. Screenshot of BasicActivity

3.2.3 ShowPostActivity

This activity will be started when a certain post item in BasicActivity is clicked. The two basic elements of this view are a RelativeLayout to show the post and an EditText to support posting new comments. When the chosen post has comments, they will be displayed below the post in a listview.

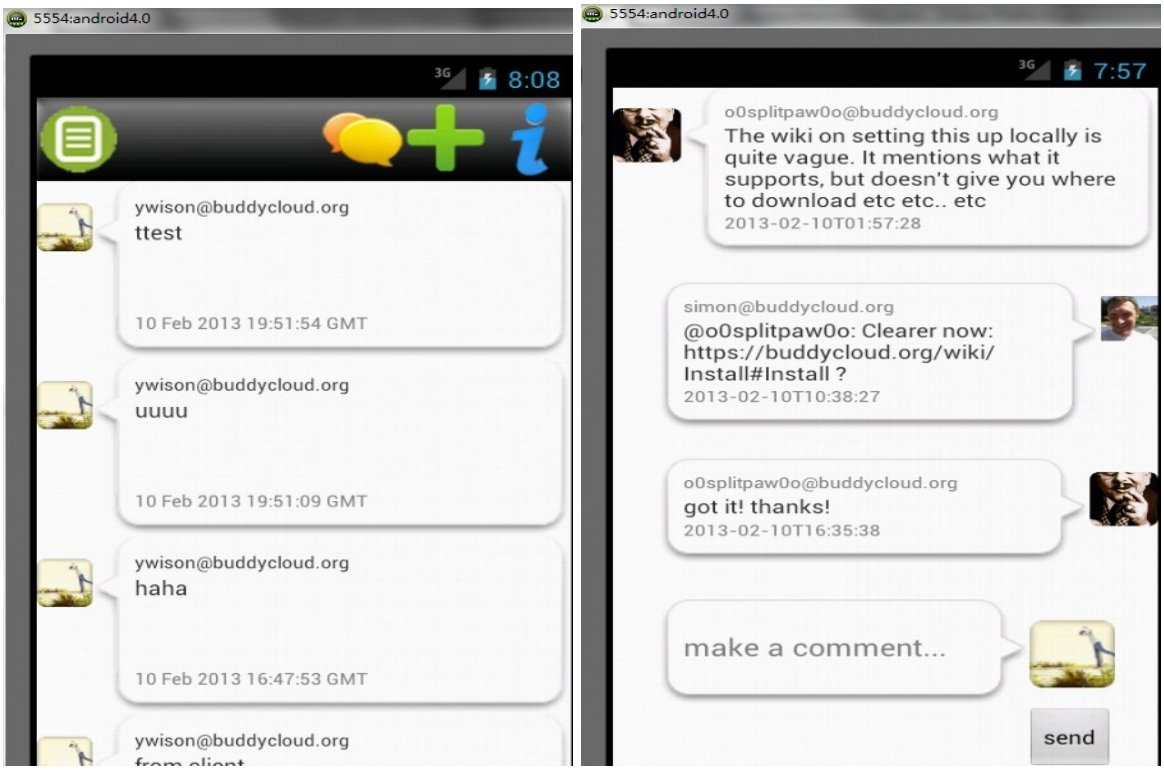

Figure 5. Screenshot of ShowPostActivity

3.2.4 NewPostActivity

A multi-line EditText similar as in ShowPostActivity forms the main part of this activity. It is started when the AddNewPost button in BasicActivity is clicked. And when the send button is clicked, this activity will be finished and return a result code to the BasicActivity indicating whether the post is posted into the server properly.

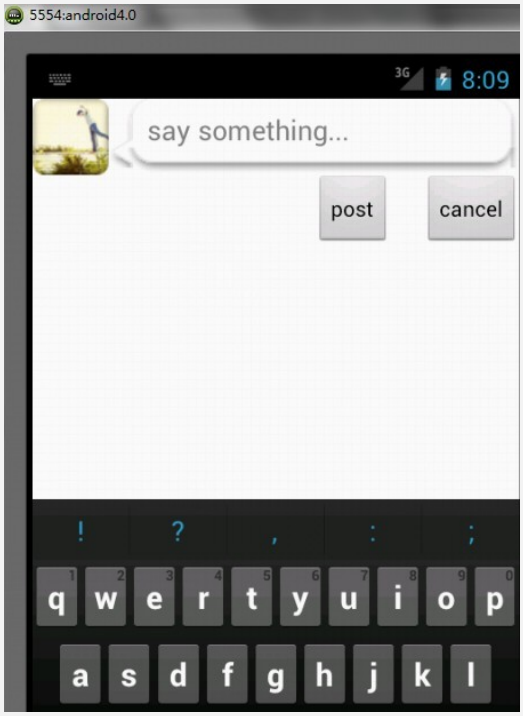

Figure 6 Screenshot of NewPostActivity

3.2.5 InstantTalkActivity

InstantTalkActivity handle the online instant chat functionality of the BudCloAnd Client. This instant talk part rely on the XMPP protocol. The workflow is shown in Figure 7.

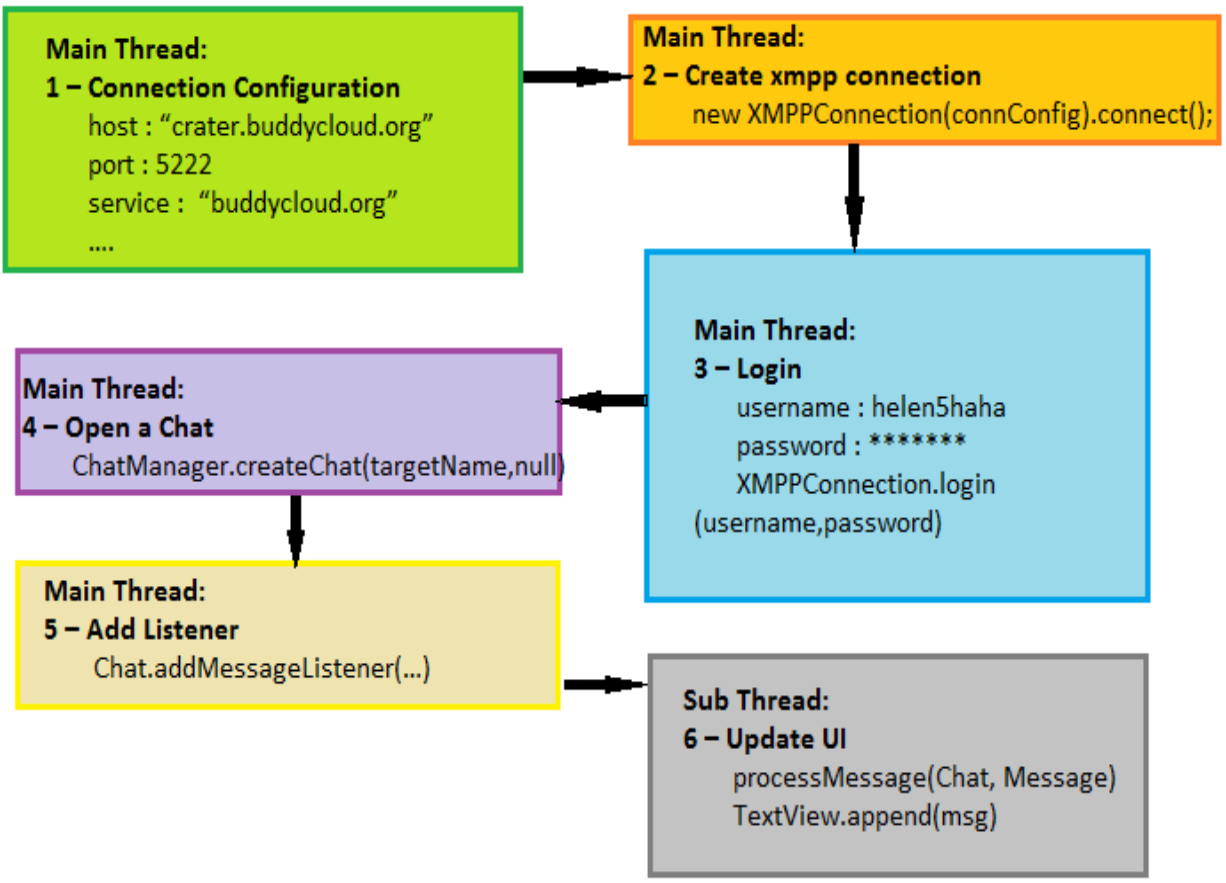

Figure 7. Workflow of instant talk

With help of APIs in external library "org.jivesoftware.smack", which can be obtained from the Internet, a simple instant chat utility can be build through the steps shown in Figure 7, including configure the connection, connect, login, open chat and chat message listening.

As a result, a screenshot of an instant chat demo is shown in Figure 8 below.

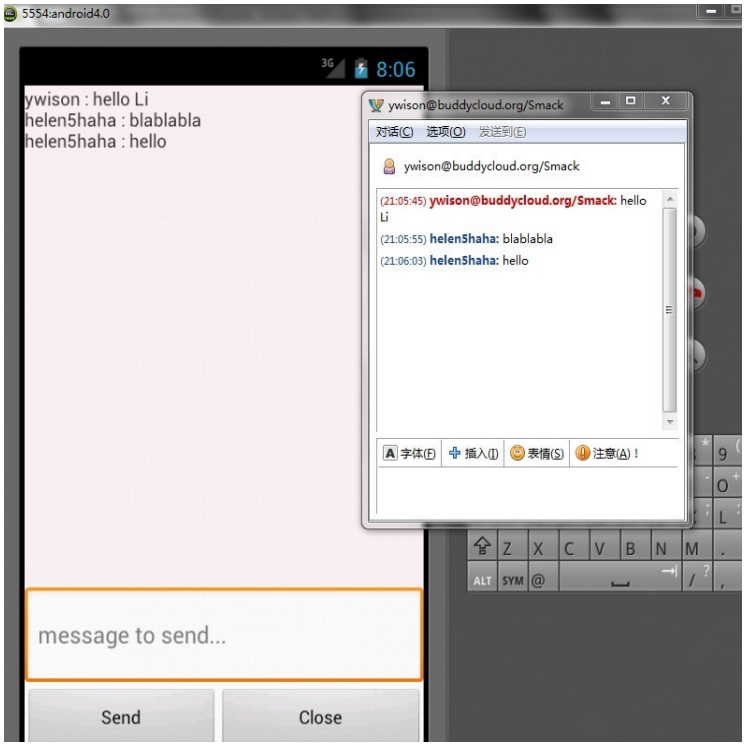

Figure 8. Screenshot of InstantTalkActivity

3.2.6 SearchActivity

SearchActivity supports users to search channels with “hints” (search keyword they type in). The result of the search process is display in form of the channel item list. User can follow the channels they are interested in by clicking the “follow” button (if it has already been followed, the button will be gray to indicating the invalid further follow operation). When valid follow-button is clicked, subscribe() is called to commit the subscription request to the server.

A screenshot of this search utility is shown in Figure 9. below.

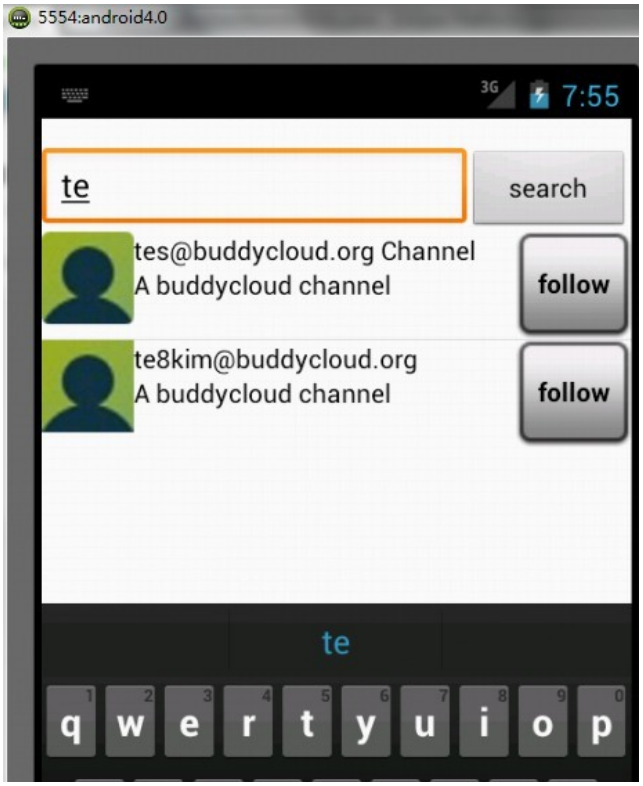

Figure 9. Screenshot of SearchActivity

To draw a conclusion, the transformation among activities is shown in Figure 10.

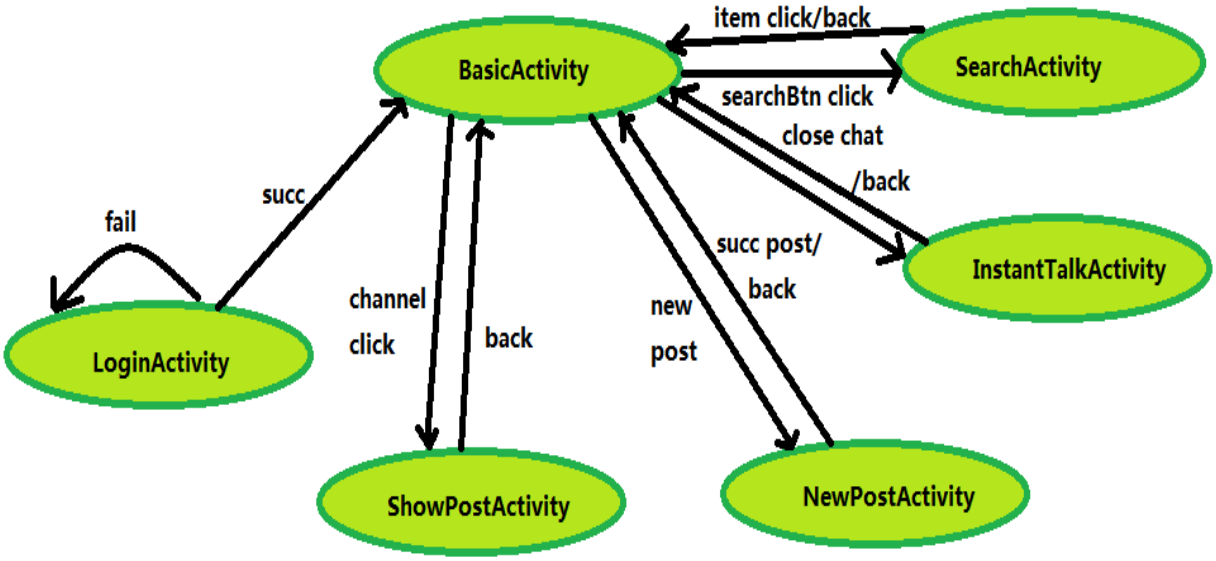

Figure 10. Transformation among Activities

As it is shown in Figure10, when the authentication return successful status in

LoginActivity, BasicActivity starts, that shows the mainly user interfaces of the BudCloAnd Client. Within this activity, user can interact with the system to trigger the other functional utilities by clicking a certain UI element. For example, when the user click the chat button in the top menu, the InstantTalkActivity will be brought to front. Another example is that, a click of any of the post item brings user to the ShowPostActivity with detailed information of the selected post.

3.3 Controller and Helper

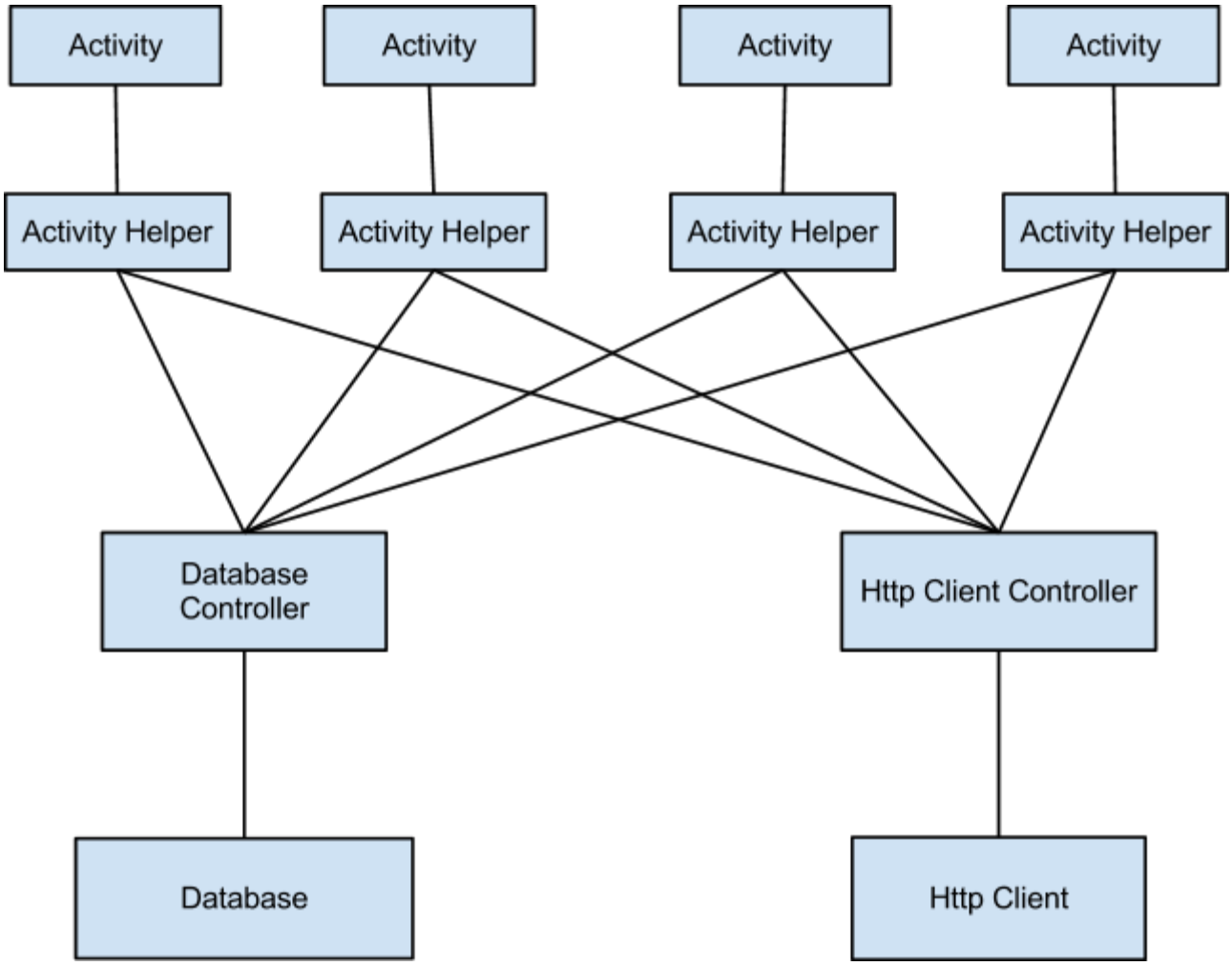

Figure 11. Controller and Helper

In our architecture each Activity is associated with exactly one ActivityHelper. The Activity Helper is used to aggregate the IO-operations in both the database and the http-client and supplies a layer that ensures data consistency. Each Activity helper bundles a set of HttpClient and database io's into convenient methods for the activities (e.g.

CachedBasicActivityHelper.loadMore(10) will cause the Cause the CBAH to load 10 more posts from the internet into the cache.)

Each helper is implementing his own interface, so it is easy to exchange the different helpers to supply new functionality or to improve performance. The Helpers also ensure, that all objects are closed when the activity ends and checks whether there is network connection or not.

The ActivityHelpers are implementing the applications caching mechanism. Right now, the application caches all posts, metadata and subscriptions in the sqlite database and uses them when there is no network connection. It also checks the cache to see if it needs to download new posts or not. It is currently not checking for updated posts(not implemented on any client anyways).

The ActivityHelpers also supply methods to post data to the buddycloud server.

For a detailed description on how to use the different helpers, please consult the Javadocs.

3.4 Rest-Client

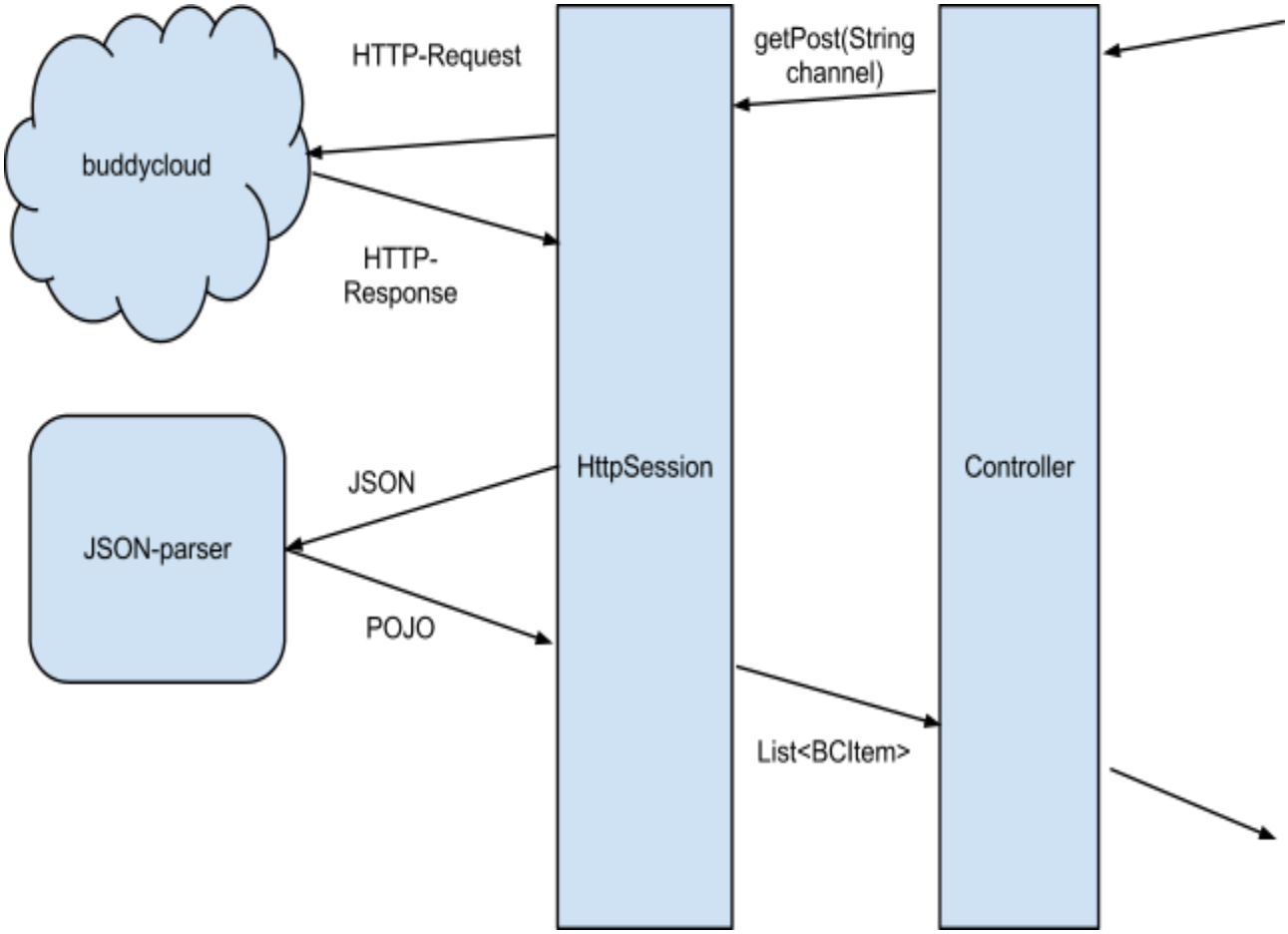

Figure 12. Rest-client

The buddycloud android client uses a controller (HttpClientHelper) to execute all requests to the users home server. The HttpClientHelper is a singleton (enum with 1 field INSTANCE and static accessible methods), that needs to be instantiated at the application start with a call to its init(String user, String pw, String home) method. This will create and instance of the actual HttpClient(BCSession) that know about the userdata and server and handles all HTTP communication. The Controller takes care of the http-clients lifecycle and handles the IOExceptions. IO to the database should always use the database controler. The Controller will also take care of creating a new session when the user changes.

The actual http-client is an implementation of the BCSession Interface, that makes sure the actual client supports a given set of IO-operations. Classes implementing this interface should make sure, they are thread safe and support ISO 8601 date-parsing, json parsing as well as encryption and http-basic authorization, but they can use their own implementation of the Rest-methods. They also need to resolve method-calls with specific parameters to the matching url for the api-endpoint at the api server.
The HttpClientHelper, also supports buffered image loading by saving soft references to pictures and downloading them in separate threads.

The DefaultClient uses HttpsUrlConnection to exchange data with the buddycloud server. It supports an unsafe constructor with and all-trusting trustmanager, as well as a constructor that allows for supplying of a ca's certificate chain via keystores. It binds the function calls to prepared url-strings and replaces the parsed function parameters.
It uses the native org.json library to supply a binding between local and remote data model.

Examples:
Retrieving data
List<BCItem> posts = HttpClientHelper.INSTANCE.getPosts(<channel-jid>);

For a full list of all supported methods, please have a look at the Javadocs.

3.5 Database Part

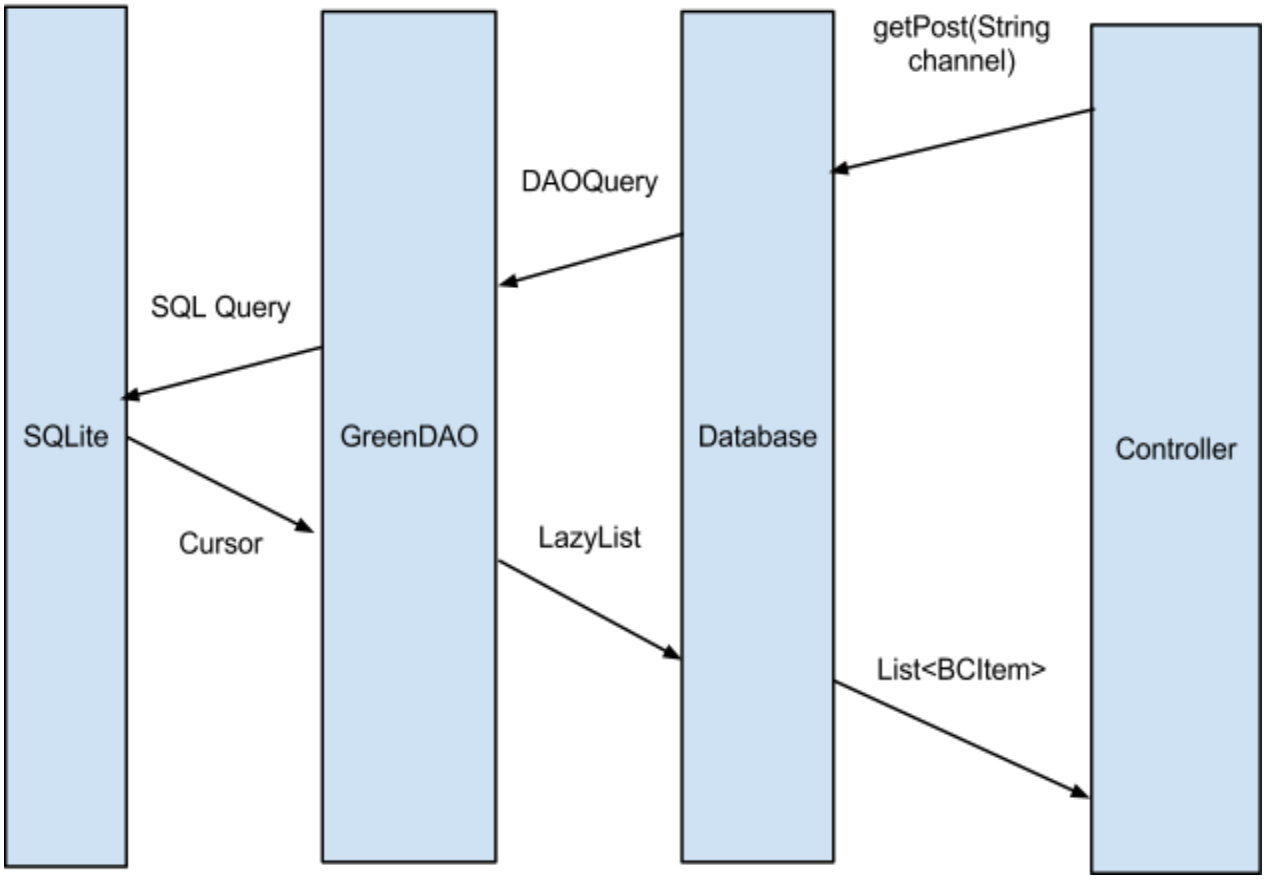

Figure 13. Flow of Database

The buddycloud android client uses a database controler (DatabaseHelper) to execute all requests to the database. The DatabaseHelper is a singleton (enum with 1 field INSTANCE and

static accessible methods), that needs to be instantiated at the application start with a call to its init() method. The Controller takes care of the database instances lifecycle and handles the IOExceptions. IO to the database should always use the database controler. The Controller will also take care of resetting the cache or creating a new database when the user changes.

The database is an instance of the Interface BCDatabase. BCDatabase ensures the actual database is supporting a given set of IO-operations such as delete, search and save entries in the database. Classes implementing the BCDatabase should make sure they are threadsave. For a list of supported IO, please consult the javadoc. Classes implementing the BCDatabase can use their own underlying database and tables, but must sure they are able to map the BCMetaData, BCItem and BCSubscribtion to the db-tables.

The DefaultDatabase is an Implementation of the BCDatabase that is using the GreenDao ORM code-generation framework to map POJO's to db-tables. The GreenDao framework is optimized for android (calls to native android api) and focuses on performance. It supports thread safe methods for querying the database and takes care of most parts of the database management.

For more information about GreenDao, please refere to the GreenDao Dokumentation (http://greendao-orm.com/documentation/). When using the GreenDao Framework, Entities with bindings to database tables may not be changed without using the GreenDAO generator and you will need the greendao library in your classpath.

To use your own implementation of BCDatabase, just change the init() method of DatabaseHelper.

Examples:
Querying the db anywhere in the application:
List<BCMetadata> metadata =
DatabaseHelper.INSTANCE.getMetadata("example@buddycloud.org");

Deleting Entries:
DatabaseHelper.INSTANCE.delete(<pojo>);

Saving entries:
DatabaseHelper.INSTANCE.store(<pojo>);

For a detailed List of supported operations please refer to the Javadocs.

4 Problems

Since the skill level of the participants is not yet professional and because the scope of this

project is limited, there is much space left for optimazation /improments .

Some parts of the code have allready been commented to indicate optimization potential, but we are going to list some additional issues here.

The Activities currently load new data in the same thread that is used to display the already cached data. If the UI thread starts a new thread to load the new data and displays the cache while doing so, loading time between different activities will decrease.

Caching: currently it is only possible to cache metadata, subscriptions and items, but it would also be possible to store user data, settings and media to the database.

External Sources cannot be displayed or opened in the web-browser.

Internal Media cannot be open since the server api is missing some endpoints.

The caching of non subscribed channels is currently leaked because of a missing parameter in the server /content endpoint.

The Client should only connect to servers with valid certificates and trusted ca's, but the buddycloud.org's ca is not supported on android right now. Currently it uses an unsafe httpsurlconnection open to man in the middle attacks to solve this issue. There is also a constructor for the DefaultSession using a keystore, but there is currently no valid keystore for StartCom/StartTLS.

The Imageloader can only load images when the server allows for https.

The InstantTalk is not saving its history and can not be used to browse conferences.

The ActivityHelpers currently don't check for the number of posts that is retrieved. if there is a huge number of comments before the next post, the ui might not show new posts even after clicking load more multiple times.

Some Endpoints in the server-api are not yet implemented. E.g. get media, post media , notify, etc

Activity Helpers could be aggregated in one single (maybe static) class.

The offline view of subscribed channels is currently bugged.

The separation between helpers and adapters should be more clear and stick tighter to mvc activities should only call adapters methods to load new posts or only call the helper method. The Helper would then call the adapters notifyOnDatasetChange() or the adapter would invoke the helper.