# Evaluation of the search mechansim in LibreSocial

Moritz Kanzler
*Email: moritz.kanzler@hhu.de*
*University of Düsseldorf, HHU - 2018 SS - Opportunistic and Peer-to-Peer Networks*

***Abstract*—Searching in a social media network is a crucial feature of connecting resources within the network. Aside from the scope of required search functionality, the implementation of such mechanisms depends on the underlying structure of the social network. This paper focuses on the currently implemented search mechanisms of the social media network "LibreSocial" and gives a quick overview about which search types are used in other networks and what functionality is needed in "LibreSocial". Its underlying Peer-to-Peer Framework determines different approaches to implement such functionality. Regarding this, the paper shows what components are used to reach that kind of search functionality, which advantages and problems exist regarding the used Peer-to-Peer structure and which metrics are built-in to evaluate those efforts. It also gives a short outlook, based on the gained results, which improvements can be made to realize better search functionality or performance and what is missing in the current state of "LibreSocial".**

## I. Introduction

Peer-to-peer (P2P) networks, formerly known for file-sharing, more and more are being seen as a solution for a distributed, self-organized, decentralized and robust network communication environment [1]. Based on that approach, P2P networks can also be used as a layer for applications which are built on top of them. Regarding different applications running above this layer, there are many different approaches needed to provide the same functionality known from client-server pendants. This paper will concentrate on the search mechanisms which are used in such a distributed environment.

When it comes to search mechanisms in a P2P network, the search is not just a component of the system for finding resources quickly and present them, but a key feature to provide routing and using data, which is stored somewhere in the distributed network. Depending on the organization of a distributed network there are many different types of P2P systems such as centralized / decentralized, structured / unstructured or hierarchical / non-hierarchical systems. According to these different types of P2P networks, the implementation of search mechanisms differ. This differences can affect particular search algorithms but also the organization and presentation of such search results [1].

For the scope of this paper, we are focusing on a decentralized, structured, non-hierarchical P2P layer which does replicate its data on different network nodes. In such a network each element has a unique ID. Nodes in the network also have an ID and are responsible for handling the data which is stored in the ID space next to them. These P2P overlays use *Distributed Hash Tables* (DHT) to enable the described functionality. The advantages of DHT based networks are good scalability and stability. Regarding search, such systems bring limited time frames for ID-lookups and reliable key-value searches for unique IDs within the network. Referred to finding objects after their unique identifier, DHTs provide a considerable advantage over other P2P solutions, which may flood the network randomly while searching an exact object or cannot guarantee to find an object in a limited time. However, because the included key-value type of search is a fundamental part of DHT, it becomes much more challenging to achieve searching for objects in between a specific range or for object attributes which are not the unique identifier [2].

Thinking about a file-sharing or -storing application on top of the described P2P layer, other types than a particular key-value search may not be necessary. However, as mentioned above, using a P2P layer like this to apply something like a platform for an online social network (OSN) upon it, brings back the need of different kinds of search.

In an OSN environment searching is a key feature regarding the different needs of the network users. Users want to search for friends, groups or even files. Such search mechanisms are known by many other OSNs as well, but they may differ from the orientation of these networks. To clarify the requirements for searching in a P2P based OSN, we first have to figure out which kinds of searches are needed or should be provided when it comes to the expectations of the user of the OSN. We also have to keep in mind that there is a separation between providing search mechanisms in the application section and the underlying P2P layer and how it can be implemented. As a first step, we do need to specify the needed search mechanisms regarding the OSN. After that, it is possible to figure out how these requirements can be collected into functional groups and how they can be implemented into an environment which initially only provides an exact identifier lookup mechanism. Following this insight, we then can examine an existing OSN with the given characteristics. For such an OSN our primary interest is what solutions to the given challenges have been made, which mechanisms are implemented and what is missing or not working yet. The rest of the paper is organized as follows.

In section II we discuss fundamental aspects of different search categories and what a user expects of a search mechanism in an OSN. We then introduce the OSN this paper is about, *LibreSocial* and also talk about some basic concepts in LibreSocial to provide different search kinds. In section III we explain how the given requirements of searching an OSN can

be mapped on the considered LibreSocial environment. We also look which search mechanisms are in fact implemented in LibreSocial, how they work, and on which data they are responsible for. In section IV we then also try to verify our insights with creating testing scenarios with the built-in test framework of the OSN. Following this scenarios, we also discuss and emphasize the results of the described test cases and put them into context. Section V is about to summarize the advantages and currently reached possibilities of searching in LibreSocial. We also discuss the problems and weaknesses of this implementation and try to give some outlook on approaches to improve the current system and get rid of some of the difficulties.

## II. Literature Review

In this section, we build the basis for understanding the key components of search mechanisms in a P2P based OSN. The first goal of this section is to understand and categorize the conceptional side of searching with its different approaches on how resources can be gathered. To apply these specifications, we also learn about an actual implementation of an OSN with underlying P2P overlay. Finally, the third goal is about to get to know some P2P related structures, which build the ground for many kinds of search functionality in a P2P environment.

### A. General search categorization

Moving on with focusing on OSNs build above the currently described P2P layer, we first try to find out which types of searches and underlying mechanisms are used in common OSNs today. Following an already done market overview, other OSNs provide searching after users, people with the same interests, fan pages, groups, events, specific criteria which matches with attributes of other users, applications within the OSN and messages [3], [4]. Search services can be divided into two types, generic and specific.

With the **generic search** networks like MySpace, Facebook, StudiVZ, and Xing offer their users a simple search field which applies the given keywords to different resources their network maintain. Networks like Facebook also offers to filter the given results after additional parameters. We call this search generic because a given keyword get applied on a not specified set of resources. Thus the OSN internally have to request different resource and datasets to calculate the shown result.

**Specific searches** are also provided by all of the OSNs mentioned above, but may come in a slightly different shape. The difference to generic searches comes mainly with defining more than just one set of keywords to search for. Instead of the generic kind, a user can specify multiple attributes like gender, age, country, university to look for. Following this, the user can expect much more accurate and organized results. Inside the system, this stricter type of searching implies to may look for much more attributes when specified, but also may a smaller range of resources, because when looking precisely after one type of record the search mechanism can dismiss other resource types.

When looking at the compared services, it can be recognized that the primary motivation for using a search mechanism in social media is to find resources that other users in the network have to provide. After summarizing the possibilities of other networks, we have to look up what the specific requirements for searching a P2P based OSN are, to keep up with the perceived conditions.

### B. Specific search categoriztion

Related to the research of section II-A, we have figured out two main search types, an OSN should provide to the user: generic and specific searches. When coming to the actual search functionality in OSNs, these two terms do not describe explicit enough the needed mechanisms. So we have to break down these search types in smaller demands and reassign them to concrete implementation ideas regarding a P2P layer which builds the ground for feasible distributed OSN.

Breitlauch did such categorization in his work about distributed data structures [5]. He named different search mechanisms related to a decentralized, structured, non-hierarchical P2P framework and outline ideas to implement them into such a system. However, they can also be seen as a general categorization for searching data in any network dealing with metadata. Following his approach and combining it with the as yet researched knowledge we get five different search mechanism categories to deal with: *Exact search*, *Keyword search*, *Substring search*, *Multiple attribute search*, and *Range search*.

**Exact search:** What exact search means is a mechanism which let us look up a single resource in a P2P overlay by its unique identifier. This is what already is a fundamental function of a structured, DHT based, P2P overlay like LibreSocials P2P framework. Its build the basis for every lookup request in the network and is reliable, scalable and fault tolerant. Exact searches in a distributed OSN environment describe such searches where the unique identifier of a user or resource is already known.

**Keyword search:** Keyword searches can be characterized by searching for attributes of a resource without knowing its unique identifier. An example for this is a user who wants to search for all OSN members who live in ”Dsseldorf”. The search mechanisms in a keyword search must provide a performant lookup with a response of unique identifiers of users which match the requested criteria without knowing them before the search. Breitlauch cited a solution in which distributed data structures like lists or sets can provide a collection of all corresponding unique identifiers for a given keyword. The search mechanism only needs to look up the responsible data structure and can retrieve all searched unique identifiers as a search result. The same approach is mentioned by [6] regarding organizing a search in the app market of the P2P based OSN *LibreSocial*.

**Substring search:** Known from other OSNs, a user may not want to search after an exact keyword, but after a part of it. An example for that can be looking for OSN members who live in a city beginning with ”Ber”. We can map this problem with the

two known search categories. Because of that, Breitlauch says that this can be solved with a range search approach, but also with a modified keyword search mechanism. He claims that tools from the computer linguistic fields can help us to use distributed list or sets to provide substring search. The idea behind that is to divide a given keyword into smaller parts, known as *n-gramms*, and save them in the same manner as the full keyword. Following this approach, the same mechanisms used by a keyword search can now produce also results for a part of a keyword - the substring search. Breitlauch noted here that an ordering of results for the different n-gramm weight is needed before using the results of such a search. However, this poses no problems, because all needed information to order the results by relevance is provided to this on the requesting machine.

**Multiple attribute search:** The goal of a multiple attribute search is merely clear after setting up the requirements in section II. The idea behind a multiple attribute search is, to not just look after one keyword, but use multiple keywords to specify a search. Breitlauch did not give an idea of implementing such searches in a distributed OSN environment, but we will come back to it later. Multi attribute search is also called multi keyword search in some cases.

**Range search:** Range searches represent a category of searches, where one or more attributes not only need to fit a given search term but to be located within a given range. A typical example for OSNs is searching for users between ages, like all users from 18 to 30 years old. Breitlauch cited that these type of searches could not be mapped with the formerly discussed data structures or mechanisms. He proposes a distributed overlay on top of the used P2P overlay to manage such searches. *Prefix Hash Tables* (PHT) are such a layer which can be applied on a DHT network. We will also come later to the usage of PHT in a concrete distributed OSN.

When we look at the described search categories above, we may see that it is not easy to map them on the defined search types of generic and specific search. It is much more like that both of the search types need at least more than one search category to rely on. It is becoming more evident, when seeing our defined search types, as what we want to provide to the user. These types are much more related to what resources we include in a lookup and how we present the results. However, it is also necessary that we look deeper into the newly discovered search categories and how they are used in distributed OSNs.

### C. LibreSocial as distributed OSN

When we talk about implementing a full-featured search into an OSN build on top of a structured P2P overlay, it is not sufficient to only consider concepts about this topic. In order to apply the recently discussed ideas on searching P2P based OSN, it is much more valuable to argue about a current implementation of such an OSN. That is the reason why we look for a real application fits the requirements set up to now. Such an application is *LibreSocial* [7].

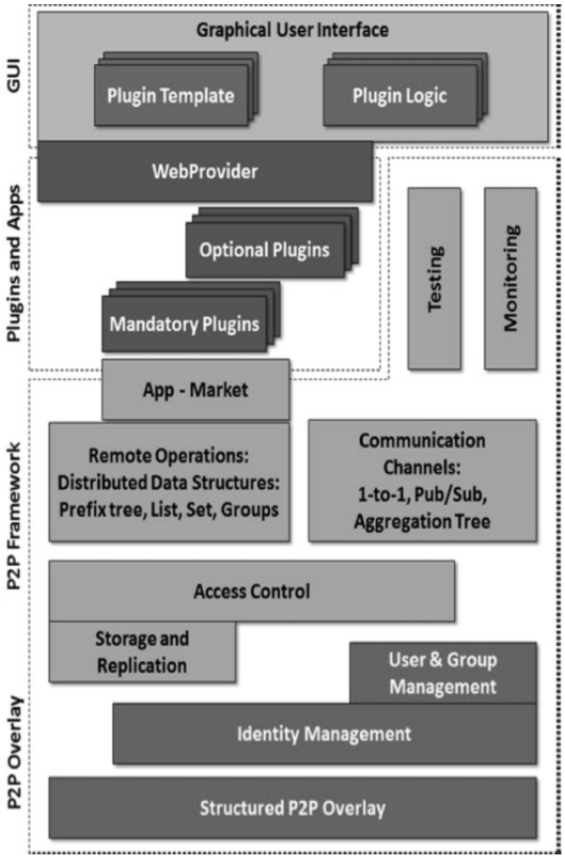



Fig. 1. Structure of the LibreSocial stack in its current implementation [8]

LibreSocial was developed since 2007 with the clear goal of providing an OSN like user experience based on a decentralized P2P overlay [3], [7], [9]. It should also be used as a platform for further researches and a test environment for new ideas related to P2P based applications [7]. LibreSocial was built upon a structured P2P overlay. On top of it, there are different extensions for that overlay, such as storage management, access control, data replication or distributed data structures. These extensions add additional functionality to the chosen overlay and altogether named as *P2P Framework* in the LibreSocial environment. Fig. 1 shows the complete structure of LibreSocial as it is implemented now.

Going through this figure from the bottom up, the structured P2P overlay is represented by implementation of *Pastry* [10], called *FreePastry*[1] [3], [9], [11]. Pastry is a non-hierarchical, structured P2P overlay, which uses DHT to manage the network topology and access the resources and the participating nodes. To complete a full workable framework for different applications, the access control, and the identity management is solved by using different asymmetric private and public key pairs, but also symmetric keys for some group functionality. It is important to notice that the FreePastry implementation was significantly extended by the LibreSocial developers to enable

[1]https://www.freepastry.org/

the described functionality in the *P2P Overlay* of Fig. 1 from the ground up.

To finish the overview on the LibreSocial P2P framework, it is also worth mentioning that there are different communication protocols and a testing and monitoring framework [11], [12], [13], [14]. LibreSocial uses a tree-based monitoring approach to aggregate results from testing and monitoring the P2P network with high precision and a low network traffic overhead [14]. We use the described monitoring and testing tool, called *SkyEye.KOM*, for running the later shown test cases and generating the presented metrics in this work.

Coming to the application part of LibreSocial on top of the *P2P Overlay*, the different functions of it are separated in *Plugins*. *Plugins* can provide a related GUI interface, loaded at runtime and communicate with each other and the underlying P2P framework through the OSGi[2] framework.

After getting in touch with the architecture of LibreSocial, the system which builds the basis for the research of search mechanisms in P2P based OSNs in this work, we want to step into how search is achieved in the current implementation of LibreSocial, what type of search mechanisms are realized, and whats missing until now.

### *D. Distributed Storage structures and their benefit for searching*

As mentioned in section II-B, distributed data structures are a fundamental component when it comes to providing different search categories in LibreSocial. To get to know how these works we now take a closer look at how they are designed and got implemented into the OSN. Distribute and store data inside the P2P network is one main topic of the framework LibreSocial is built on. As explained in [5], the idea of creating such data structures is based on known local data structures which are extended to work in a distributed scenario.

The approach of distributing data through the network is, to use the *objectIDs* of the DHT as pointers to the location of persistently stored data or parts of them. Through constructions like linked lists or graphs, many of these references can be easily stored and received by a user. After that, they can access the nodes with the searched data over their retrieved objectID. The report [11] describes the three used data structures within LibreSocials P2P framework: distributed sets, distributed linked lists, and prefix hash trees.

**Distributed sets / link lists:** Both of these distributed data structures are based on the Java Class `Set`. To allow access to all members of the network, they are stored on a specific node in the network and get replicated and distributed to other nodes. The distribution of the data structures happens through buckets, which represent a part of the whole data. All buckets have a fixed size *s*.

Handling data from such a Set appears to the user like using a locally stored set. In the background, the requested node runs remote commands to the node where the used bucket lies. This remote node executes them on their bucket locally and sends qualified answers back to the requesting node. Linked lists work similar to Sets but offer more functionality.

**Prefix hash trees:** Following our research so far, range searches are one of the main challenges for structured P2P overlays to provide. The data structures above allow only searching for particular keywords or tags in the DHT. The answer to such keyword searches always gives back a single node responsible for the hash looked after. A range search instead has often to return multiple nodes, which are all corresponding for parts of the searched range.

To perform range searches in this environment, additional techniques are needed. As mentioned in the section II-B, enabling range searches in DHT based P2P overlays need some abstraction layer on top of the P2P overlay. Breitlauch [5] provides *prefix hash trees* (PHT) [15] as possible solution for implement such an abstraction layer which can be used for tasks like range searches.

PHTs enable a reliable and fault tolerance search mechanism on top of a DHT. The PHT implementation in the P2P framework of LibreSocial is described in [11]. With having PHT implemented in LibreSocial and also a concept for range searching with them, the author notices that no multi-dimension searches or multi-keyword searches are possible at the moment, even with the use of PHTs. Nevertheless having PHTs as a data structure in LibreSocial, has not yet reached to implement a current range search functionality in it. According to that the OSN still lack of a concrete realization of range searches.

## III. Search mechanisms in LibreSocial

In this section, we discuss concretely how searching in LibreSocial is implemented. Beginning with the first concept for providing search functionality in LibreSocial, we then learn about the current implementation details of it. When looking at the required types and categories of search according to section II, we also start to check if and how these search categories are integrated in the current LibreSocial implementation. Finally, we can make a statement about which functionality is fully integrated, which can be improved and which may be missing in the current architecture.

### *A. First concept of searching in LibreSocial*

In the already mentioned work of Podrajanski [3] we can find the first attempt of formulate specific search requirements for LibreSocial. As the originator of the *Search Plugin*, Podrajanski proposes to create a way which should enable a lookup for users for specific user attributes like username, the city of living, country of living, email address, or gender. A result of such a search should show all user profiles which match the given criteria. He also claims that files which are stored by a user within LibreSocial could be looked up from other OSN members.

Following the concept of this version of the search plugin, it can be seen as a collection of data structures for searchable user attributes. It maintains the data management of the meta

[2]https://www.osgi.org/

information and manages the search. Other plugins should also be able to register new attributes to current collections or defining even new collections of attributes. With logging in the user to the network, the search plugin should receive at least the unique username for further indexing additional information. He defines these collections as *criteria lists*, which represented as distributed data structures within the P2P overlay. There is an example for the attribute ”Berlin”: so all unique usernames are saved in a criteria list under the key ”Berlin”, who have set Berlin as their living city in their profile. Combining the explanation above, criteria lists are collections of unique usernames for every attribute value exists in the network. Through generated and assigned, so-called, *storage keys* criteria lists can be found, just like any other resource in the network, with a fast and reliable exact search. Retrieving a criteria list after searching an attribute also representing the, before defined, result out of a list of usernames which match this attribute. Finally, with the approach of criteria lists, the proposed search plugin gives a solution for keyword searches in LibreSocial.

The concept of the search plugin in [3] also provides a possible concept for range searches in LibreSocial based on the described criteria lists. The first step to do so, is to divide that a given range of attributes into smaller sections. Then, each of this sections builds a criteria lists responsible for its part of the range. This sections should also be defined overlapping. When doing a range search on such an attribute, the requested range area can be matched with the criteria lists that managing the related sections of the data. With this approach the search plugin allows the P2P framework of LibreSocial to offer range searches, with just extending given mechanisms.

What we discussed here is the first approach to addressing a particular search functionality in LibreSocial. We can see that the proposed search plugin in [3] focuses on searching user profiles after given profile attributes. His concept follows the idea of bringing keyword search and range search on top of LibreSocials P2P framework to the OSN.

### *B. Current state of search mechanism in LibreSocial*

In section III-A we got in touch with a concept of providing search in LibreSocial. Next, we want to find out what specific search categories are implemented in LibreSocial yet and whether some of the previously discussed ideas have been integrated in it.

In [13], the group of researchers and developers of LibreSocial explain the current implementation details of search mechanisms in the OSN, like it can be found in 2014. The authors state that up to now a search plugin is integrated in LibreSocial. This search plugin provides users a search mechanism to find other registered users within LibreSocial. A search can be executed after different predefined attributes, which matches the profile information any user can release.

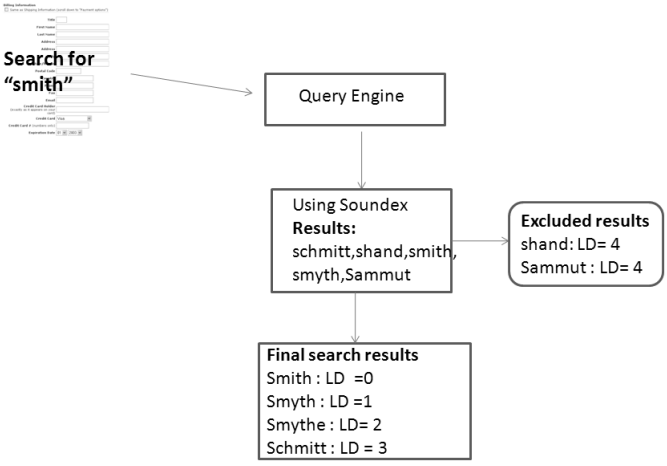



Fig. 2. Rollout of a search request initiated by the user from [13] in the current implementation of searching for user attributes in LibreSocial.

LibreSocial has a GUI to enable the users doing such searches. This GUI has a generic search field which looks up all considered profile information and specific search fields for exact information searches on single attributes. More precisely [13] divide the whole search process into two parts: *indexing* user attributes and *searching* for attributes.

**Indexing user attributes:** A new user registration in LibreSocial leads to a process, where all given profile information gets indexed for search initially. All of that information is saved into *query attributes* and are also processed into *Soundex*[3] [16] values. Both, the real values and the Soundex generated values, are grouped in a *SearchCointainer* through a *join query engine*. The detailed functionality of different join query approaches is discussed in [2], [17]. This indexing step will be returned whenever any profile information get changed in the future.

**Searching for attributes:** When a user is entering a search term in the search GUI of LibreSocial, it gets processed as Soundex value. The real value and the Soundex values pass over to a *join query engine*. Then they are queried over the nodes of the P2P overlay. After getting an answer from corresponding nodes, the search results get verified by relevance. This is necessary because some Soundex results may not be so relevant for the user than other related to their phonetic perception. To order the search result after their ”meant-by-the-user” relevance, the results are weighted with the *Levenshtein distance algorithm* [18]. After performing the algorithm on the results, the user receives them ordered by their Levenshtein distance weight. An example is shown in Fig. 2. Looking at the searching in detail, the first step in the process is to pass the search value to the *join query engine*. The *join query engine* starts a parallel request to all clients that have relevant information about the search. Therefore it uses the advantages of the distributed data structures in LibreSocial. The requested clients return data matching to

[3]https://www.archives.gov/research/census/soundex.html

the search as serialized strings. After getting all the data, the Levenshtein algorithm is performed on the strings. After that, they get sorted and filtered and being passed back to the user as a final result.

After we get to know the detailed implementation of the provided search mechanisms for user searches, the report [13] also claim some problems with the given solution.

First, the current integrated functionality ignores the fact that some profile information can be set to private. It also ignores blocked users. This result in finding users by information they do not want to make public or appearing in results of users they have blocked previously in the OSN.

Second, the developers have made a performance compromise in the implementation. Search requests are performed in parallel and remotely processed and then deliver results back to the requester. The requesting node can not know when these processes are finished or even if there is not a proper result to the search. Because of this and for a responsive GUI look and feel, the developers set a timeout for each request after 5 seconds. When testing the solution, search results often last nearly 5 seconds to respond a result. This time span does not feel like a fast responding GUI to the user, as the authors designed it.

When we compare the concrete implementation of [13] with the given concept of [3], it is getting more evident that the implementation follows the basic ideas of the concept shown in section III-A. The implementation provides a mechanism to extract the user information out of the profile into a distributed data structure. It also looks through these data structures when such a searched attribute is requested. Briefly, the discussed implementation, just like Podrajanskis concept, provides a keyword search related to the definition in section II-B. Additional to that the current implementation is also enabling substring searches on keywords as they are outlined in [5]. This substring search is achieved by the usage of Soundex as a *n-gramm* integration into LibreSocial. The idea of [3] enabling range searches with divided keyword sections has not been integrated according to [13].

Finally, the improvement and problem remarks in [13] give us the insight, that all of the described search functionality is only working for user profiles and their additional information. The report states that the search mechanism cannot search for groups, messages, or files yet. There is also no reference about any endeavor to implement range searches or multi-attribute searches in it, or how a concept should look like for these cases.

## IV. Experimental analysis of search in LibreSocial

Until now we only have examined LibreSocials search mechanism design and proposed implementation. To verify our impressions on what we have researched so far, we also want to see how a real LibreSocial environment behaves when it comes to searching for information in the network. To do so, we first show which possibilities LibreSocial offers to run tests in a real-world scenario. After that, we describe the indeed tests we have created to get some results for searching related topics in LibreSocial. Finally, we analyze and discuss the generated test results.

### A. LibreSocials test and monitoring plugins

LibreSocial provides a test and a monitoring functionality through the *Test plugin* and *SkyEye.KOM* on the whole network [14][19].

The *Test plugin* operates over a defined test plan, which includes different OSGi commands. These OSGi commands get executed on all nodes which are part of the test and contain actions for the different plugins of LibreSocial. This commands can get called by the *Test plugin* by so-called *test cases*. These test cases represent a clearly structured sequence of actions done in the OSN. In every test is acts one node as a master node, the rest as ”slave” nodes. The master node spread the defined test plan through all slave nodes and has an overview over the registered test users and the progress of the running test [19].

*SkyEye.KOM* represents the monitoring tool of LibreSocial. It can be used to monitor a running LibreSocial environment to keep track of the load balancing or the resource distribution within the P2P layer. However, it is also used to create regular evaluations on the network. These generated metrics can also be gathered after a passed through test scenario. The noteworthy feature of SkyEye.KOM is that it aggregates the single benchmark results from every node in the networks and creates a constant overview over the whole network, not only a part of it [19]. In a P2P overlay scenario, this is a very useful feature to get to know about the overall performance of such an overlay, so we do to get the metrics for our test scenarios.

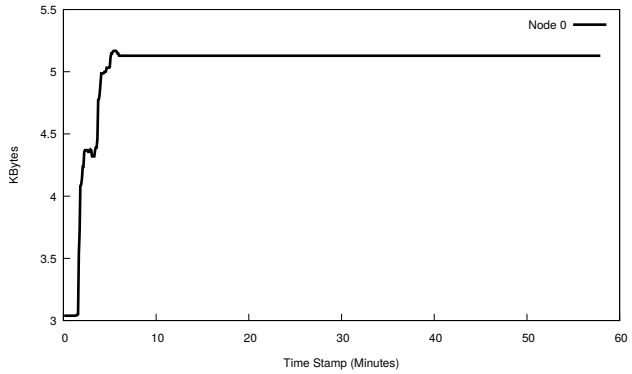
### B. Experimental setup

To test LibreSocial in a scenario as realistic as possible, we use the PC-Lab of the university. The PC-Lab offers eight machines each with 4 Gb of main memory, which is connected through a gigabit ethernet switch. Following the settings and experiences of [14], we know that we can surely run 8 instances of LibreSocial on each of the machines. One machine needs to be used solely as a master node for the Test plugin. So we have in fact 7 machines with 8 instances per unit to create a network running LibreSocial.

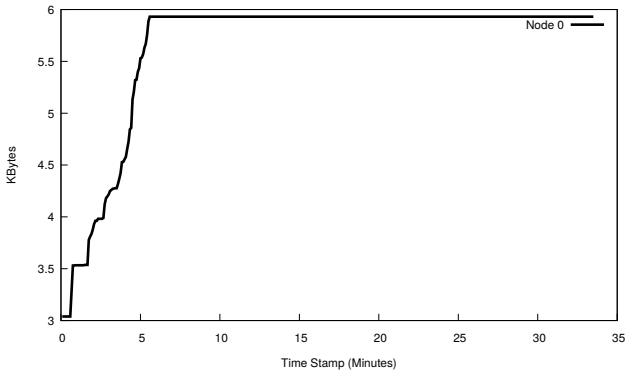
With this 56 instances LibreSocial and the predefined environment settings of [14], we create different test plans and scenarios to get an overview of what search does in the OSN.

We now describe two test scenarios we run in the PC-Lab under the described circumstances.

The **first** test scenario is about to create other data than just the user profile data into our LibreSocial environment before we start a search. Therefore we implement a scenario where we first create some photo albums and upload some photos to them. This is done




(a) Results of a test scenario where no other data than user information are stored in the OSN.




(b) Results of a test scenario with stored user information, photos and messages through the whole OSN.

Fig. 3. Globally stored search data over the whole P2P overlay generated while running two test scenarios with a experimental setup of a LibreSocial network.

with the test cases `TestCaseCreatePhotoAlbum` and `TestCaseUploadPhoto`. We also add some messages to the OSN by using `TestCaseSendMessage`. As far as we know, only profile information get indexed with the *Search plugin*. This scenario should verify our learning about the current implementation of search mechanisms so far. In order to be able to compare the results of this scenario, we also let run a test with no data besides the user information and some rounds of the `TestCaseSearch` tests. By comparing these two scenarios, we can evaluate if the additional amount of data in the OSN results in a different outcome in the metrics. The conclusion of this comparison should tell us, if the current implementation of LibreSocial may index and search for more resources than the so far known user information.

In the **second** test scenario is about to run the test case `TestCaseChangeProfileInformation` before running different `TestCaseSearch` tests. The goal of this test scenario is not primary to check differences in the monitoring output. It is more about to look at the console output during the scenario run to check what the `TestCaseSearch` test case exactly search for.

### C. Analysis of the test results

Our **first** test scenario should compare the results of searches in a LibreSocial environment where we run different `TestCaseSearch` test cases while no data is stored in the OSN beside the one of the current registered users. We then run a similar test scenario but providing data like photos and messages in the network. In Fig. 3 we see a resulting metric of both of the test cases. It shows how much search-related data is stored in the network. This data represents the produced search index references, we got to know as *criteria lists* or *query attributes* in section III. Fig. 3(a) shows the result of the test run with no additional data, Fig. 3(b) the ones of the opposite run. Both tests have to wait first to ensure an established network. In this time span, the users get registered one after another. The first test case from our test plan has been executed after 5 minutes. As we can see in both figures, the network first stores data for every newly registered user. This behavior matches to the *indexing* step mentioned in section III-B. After indexing all new registered users and their information, both LibreSocial environments do not index any other information until the test runs stop. With keeping in mind, that the test scenario, which adds data like photos or messages after the 5-minute barrier, does not change the amount of indexed data in Fig. 3(b), we can conclude that all other data in the OSN get ignored by the *Search plugin* and is dismissed in any search operation in LibreSocial. With this result, we can verify our so far researched knowledge, that only user information get indexed and searched in LibreSocial.

We also want to mention the slightly different amount of data stored in the two shown test cases in Fig. 3. We already identify that the statement of the comparison is evident because there are no changes in the amount of data after the 5-minute barrier. The different amount of data is explainable through the different point of time the test cases have run. Because the machines in the PC-Lab did not only run LibreSocial and the aggregation process of the monitoring can differ slightly in every test run, the results are not the same, but show the same quit clear trend.

For the **second** test scenario we focus on the output of the *Test plugin* and also on the code of the *TestCaseSearch*. While running the described test scenario all console outputs regarding the *Search plugin* outputs only searches after unique usernames, cities, and countries. All three attributes are information a user can provide to the own profile. To clarify this impression we look into the concrete implementation of this test case. For this we check the responsible implementation of the *TestCaseSearch* scenario in LibreSocial. A code review shows us that the search test case chooses randomly one of the three attributes mentioned above. There are corresponding files for the test plugin which each contain a various amount of usernames, city names, and country. The *TestCaseSearch* test case choose for every performed test one out of the three given categories. It then also chooses a random value from the corresponding example file and look it up in the P2P

framework.

With this insight, we again can confirm our research on which resources in the LibreSocial environment can get looked up through the search mechanism. It additionally shows us that the written test case to evaluate search is also limited to three defined user attributes and not even covering all current implemented profile information.

## V. Discussion

In this section, we start by aggregate the so far generated knowledge about search mechanisms in LibreSocial. We also draw a conclusion on what is possible in LibreSocial regarding search functionality. At the end we give a short outlook on how searching in LibreSocial can be improved and new functionalities may be implemented.

When we look at the big picture of our research, we now can summarize the possibilities of the search mechanisms and implementations of LibreSocial. According to the situated search categories we have seen LibreSocial has implemented 3 out of the 5 defined search categories, which are useful in an OSN environment: **exact search**, **keyword search**, **substring search**. There is also a concept for providing **range searches** on top of current implementations of keyword searches. We also found out that all of the currently implemented search mechanisms only work with a minor section of all searchable resources LibreSocial do provide to the user. By now, only the profile information of a user is searchable. Resources like messages, photos or files are not considered in the current search functionality. When we evaluate the search capabilities with the test framework of LibreSocial, the monitoring tool outputs a lot of different metrics, but only a minor part related to search. In our opinion adding more metrics for searching in the future is necessary, to evaluate the indexing and processing steps for a single request more detailed. We now take a view of what concepts are around to help to close the found out gaps in LibreSocials search functionality.

As possible solutions for range searches, PHTs are provided by [5]. They get also already implemented as a distributed data structure in LibreSocial. An additional approach to achieve range searches in a structured P2P overlay is one of distributed indexing and different join strategies proposed in [2]. The basic idea of this concept is to index additional attributes of a resource and distributed it over the network nodes. All the additional information have references to the unique ID of the resource these pieces of information belong to. The approach of indexing is pretty similar to what happens in the *Search plugin* described in section III-B. The ability to range search the data comes with different implementations of the mentioned *Join query alorithm*. In [2] the concepts of *NetLocJoin* and *OptimizedRangeJoin* start parallel routing requests and assemble the result in distributed manner to lower the computation work in the whole overlay. The *indexing and join* approach can lead to a more straightforward implementation of range searches because the basics of indexing and joining requests are already integrated in LibreSocial and are a notable alternative to the *PHT-approach*.

Thinking about multi-keyword searches [11] makes the statement, that multi-keyword searches are not possible within the current components LibreSocial provides. In [17] different mechanisms of achieving multi-keyword searches are discussed. His work is also based on usage in a FreePastry overlay and provides, among the already known *join query alorithms*, also a solution based on a publish/subscribe-queue [20]. The publish/subscribe-queue proposed there is called *Scribe*[4][21] and enables good scaling over a different amount of keywords per query and also is well integrated in the Pastry environment. The fact that *Scribe* is already added in LibreSocial [11] makes this approach even more suitable to test in LibreSocial to achieve a multi-keyword search in the OSN.

## VI. Conclusion

The approach of this paper is to evaluate the integrated search mechanisms in LibreSocial, an OSN based on a structured P2P overlay. After researching for the current implementation ideas and details of LibreSocials search functionality, we then tried to work out which kinds of search functionality is well integrated and supported in this OSN, and which search functionalities are missing or not working as expected.

In summary, we have found out, that LibreSocial supports 3 of 5 search categories, an OSN usually provide. Because of its distributed roots, many implementations of search mechanisms are not that easy compared to client-server-based solutions. We have discussed some concepts and ideas of providing search kinds like multi-criteria searches or range searches which may be applied to the system in the future. However, we also stated that many of the currently implemented search features, like the keyword and substring search, are not integrated consequently over the whole system. All of the built-in search functionalities are only focusing on providing a user-searches-for-user solution. The other resources like groups, messages or files cannot be searched in LibreSocial with the already integrated mechanisms. So implementing new search kinds is not the only goal for future development in LibreSocial, but also to apply already built-in solutions to all resources of the OSN. Along with this understanding, also the given test cases should be improved and extended to cover all of the functionality the search provides to the users and maybe also give additional output in metrics for search-related statistics.

## References


[1] X. Li and J. Wu, "Searching techniques in peer-to-peer networks," *Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks*, 2006.

[2] C. J. Parekh, "Meta-data based Search in Structured Peer-to-Peer Networks," 2013.

[3] S. Podrajanski, "A peer-to-peer based framework for large-scale social networking platforms," 2008.

[4] K. Jagannath, "User models and requirements," Universität Paderborn, Tech. Rep., 2011.

[5] D. Breitlauch, "Distributed data structures for online social networks," Universität Paderborn, Tech. Rep., 2011.


[4]http://www.freepastry.org/SCRIBE/default.htm

[6] P. Alavi, L. Bremer, D. C. Veena, J. Janiuk, S. K. Gurumurthy, A. Luqman, Y. Mexin, A. Morast, B. M. Jazayeri, J. Parekh, A. Petkau, J. M. Sander, and H. Zhao, "Project Group 2012 Documentation," Universität Paderborn, Tech. Rep., 2013.
[7] K. Graffi, "A P2P-Framework for Social Networks," Düsseldorf, 2013.
[8] P. Group, "A peer to peer Framework for Social Networks." Paderborn: Universität Paderborn, 2014.
[9] B. Menges, "A security-framework for peer-to-peer based online community platforms," 2008.
[10] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Middleware*, (November) 2001.
[11] P. Group, "Project Group 2011 Final Report," Universität Paderborn, Tech. Rep., 2012.
[12] P. Group, "A Peer-to-Peer Framework for Social Networks PG 2012/2013," Universität Paderborn, Tech. Rep., 2013.
[13] C. Jovita, B. Arockiaraj, K. Balachandran, N. Barua, B. Gao, T. Gour, C. Holomek, T. Jacobs, S. Kumar, and B. Kaneriya, "Final Report," Universität Paderborn, Tech. Rep., 2014.
[14] I. Dlikman, "Systematic Benchmarking of a Fully Distributed Communication System," 2018.
[15] S. Ramabhadran, J. M. Hellerstein, S. Ratnasamy, and S. Shenker, "Prefix hash tree: An indexing data structure over distributed hash tables," ... *Principles of Distributed* ..., 2004.
[16] V. P. Parmar and C. Kumbharana, "Study Existing Various Phonetic Algorithms and Designing and Development of a working model for the New Developed Algorithm and Comparison by implementing it with Existing Algorithm(s)," *International Journal of Computer Applications*, vol. 98, no. 19, 2014.
[17] D. Breitlauch, "Search Algorithms for Multi-Criterial Queries in Peer-to-Peer Networks," 2013.
[18] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, Feb. 1966.
[19] S. Kaneriya, "Testing Of A Peer-To-Peer Framework-based Social Network," 2015.
[20] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, 2003.
[21] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," in *IEEE Journal on selected areas in communications*, (October) 2002.