A Log Based Update of Replicated Profiles in Decentralized Social Networks

Lyes Badis, Djamil Aïssani
LaMOS Research Unit, Faculty of Exact Sciences
Bejaia University, Algeria
badis.lyes@yahoo.fr
lamos_bejaia@hotmail.com

Mourad Amad
LaMOS Research Unit of Bejaia University
Bouira University, Algeria
amad.mourad@gmail.com

Journal of Digital Information Management

ABSTRACT: *Decentralized Social Networks are proposed to provide more privacy guarantee to the users. They avoid storing data in central servers. The profiles are important and critical objects of this type of network. They are stored in the user's devices. In order to ensure more availability, every profile is replicated in several devices. The different copies of any given profile stored in different devices are correctly updated with all the writes that happened in any single copy. Since the profiles undergo frequent writes, an important network load is generated during the update process. In this paper, we propose an update model based on the use of a log file to store the writ s. This reduces the size of the data exchanged between the devices storing the same profiles. Simulation results show that the proposed model is globally satisfactory.*

Subject Categories and Descriptors
K.4.2 [Social Issues]; C.2 [COMPUTER-COMMUNICATION NETWORKS]; Open Systems Interconnection reference model

General Terms:
Social Networks, User Profile, Logs

Keywords: Decentralized Social Network, Profile Replication, Consistency, Update profile, Log files

DOI: 10.6025/jdim/2018/16/5/230-245

Received: 18 April 2018, Revised 22 May 2018, Accepted 2 June 2018

1. Introduction

The online social networks (OSN) are a very powerful tool of information sharing and communication. The users' data are stored in central servers. This allowed the OSN's providers to ensure high performances and to develop very advanced features (social information retrieval [21] [36], recommender systems [22]). The users' data are also available to the third parties to develop and plug their applications into the site [18]. The social network data are often analyzed in different kinds of investigations [30].

In fact, the users lose control of their own information once they publish them in a social network. Also, the publications or friends suggested by the recommender systems show that the OSN providers may know about the users more than they want. All data and relations are managed by a service provider, which involves the user privacy [14].

Decentralized Social Networks (DOSN) are alternative with high privacy guarantee. This is due to the distributed storage and management of the users' profiles and the security policies based on encryption and access control [2].

The main concept of the DOSN is to avoid the storage of users' data on untrusted servers. The scheme according to which the profiles are stored affects the system performances.

The first parameter directly concerned by this choice is

the profiles availability. To give the users the possibility to manage their own data, the first idea is to store the data in the user's devices. In this situation, the profiles availability is conditioned by the availability of the own device.

To improve the profiles availability, the replication mechanism is introduced. So, each profile is duplicated in many devices. That improves the accessibility of one copy at least. Different replication schemes were proposed in literature. Some are inherited from the replication techniques in traditional P2P systems by storing in super peers (eg. supernova [10]) or using DHT (DECENT[9] and LifeSocial.Kom[6]). Others systems explore the social relations (friends [11][12], communities [19][32], ..) to ensure safe replication.

The replication improves clearly the profiles availability, but it can degrade the network performances. The copies of the same profile should be updated in order to serve coherent contents. The update mechanism can lead to a network overload situation. Especially when we consider the particularity of social network profiles compared to the data stored in classical P2P systems: frequent changes and a growing size.

Another aspect to observe is the load balancing between the devices. The data is stored on the users' devices which have reduced performances. Devices that store many popular profiles can fail under users' requests.

In this paper, we propose a new model to manage the profiles replication in a decentralized social network. We propose to store every profile in many users' devices, including the device of the profile's owner. To ensure the consistency of all the copies of the same profile, we propose to apply two update mechanisms:

• Synchronous update based on direct synchronization messages.

• Asynchronous update based on a log that stored all the events happened on a given profile.

The lookup service is based on a priority list that allows balancing the load between the devices storing the same profiles. The main objectives of the proposed solution are:

• Improve the profile availability,

• Ensure the consistency of the replicated profiles,

• Reduce the overhead generated by the update process; this is due to the use of the log of events,

• Load balancing between the collaborating devices.

The rest of this paper is organized as follows: Next section gives a background and related works. The proposed model for updating profiles is detailed in section 3. In section 4, the evaluation of the proposed solution is investigated. Finally, we will conclude and give some future directions.

2. Background and Related Works

In this section, we discuss the main concepts of the replication techniques in the decentralized architectures for social networking. An analysis of related works is given.

2.1 Background

In this subsection, we give a small description of social networking and in particular the decentralized social networks. The P2P social networks are introduced. The federated social networks and the hybrid model are illustrated at the end of this subsection.

2.1.1 Social Network

A social network is defined in **[17]** as a webbased services that allow individuals to:

• Construct a public or semi-public profile within a bounded system,

• Articulate a list of other users with whom they share a connection,

• View and traverse their list of connections and those made by others within the system.

According to this definition, a social network is composed of a set of profiles linked with different kind of relations. The profile contains user information's and its posts. The common social networks define their own rules about the nature of the posts. They can be a short text or pictures or even videos. The same rules can be applied to the comments.

The users interact with comments and like mentions. This interaction is a very powerful tool to evaluate the posts. The users are linked by three types of relations [3]: symmetrical links (*friendship*), asymmetric links (*subscription*) or via user groups.

2.1.2 Decentralized Social Network

A DOSN is a system that provides social network functionalities without a central service provider. They are proposed to protect the user privacy: the users have more control of the data they share on the network.

The first challenge of DOSN is the data storage. The users decide where to store their data: on their own devices, on the friends' devices, or in trusted servers.

In order to achieve the main goal (privacy), access control policies are introduced. They are based on access policies, encryption schemes or a combination of both techniques [2][34].

The communication management is a real challenge. Some architectures are based on P2P connections. Others still use central servers. According to these three aspects,

the DOSN are classified into three classes [2]:

• **P2P-Social Networks:** The system is built on a top of P2P network (*in general DHT based P2P network such* as HPM[29]). The user's devices are used to store the data. The DHT can manage the data replication also. The user's interaction is released by direct communication between peers. To protect the data from unwanted access, it is encrypted before the storage in the selected Peers. As an example of this class of DOSN, we can list: Safebook[11], DECENT[9], LifeSocial.Kom[6] and SocialGate[33].

• **Federated-Social Networks:** In this class, a set of storage servers is proposed to the user. Then, the user can select where to store his data. In addition to the access control lists hosted on the servers, encryption is used. All data objects exchanged between users are end to end secure [25]. The communication is managed by the hosting nodes. As an example, we can refer to Diaspora [24], Sonet [25].

• **Hybrid (*federated-P2P*) Social Network:** In these DOSN, both P2P mechanisms and the federated servers are used. This hybridization is used in different ways to benefit from the central server capacities and keep the users at the top of his data management. Confidant [12], Vis-à-Vis [26] and Polaris [27] are considered as hybrid DOSN.

Figure 1 presents a simplified architecture of a node in a P2P social network.

2.2 Related Works

The authors in [1] present S-DATA (Structured approach for Diurnal Availability by Temporal Assemblage); a time based grouping and content replication protocol that exploits the cyclic diurnal availability pattern in user uptime. They also introduce the concept of Beta availability (at least Beta members of a replication group will be online at any given time). The peers are clustered in groups according to their diurnal availability pattern in a manner that ensure the Beta-availability. The evaluation shows that the 2-availability level ensures satisfactory availability with an optimal overhead. The consistency model is not treated in this work. In addition, the replica placement is chosen only according to availability, the authors don't consider other criteria such as trust and capacity.

In [23], a dynamic storage system is proposed. The profiles are stored in a subset of friends only. The friends' devices chosen to store a copy of the profile are changed dynamically. The Subset of friends that participate in the replication changed also according to the interactions between friends. The availability ensured by this system is interesting. However, the authors ignore the network overload caused by the change of replica placement. They assume that profile pages are generally small. This assumption can't remain true for a long time. The profile size grows and the cost of copying the whole profile several times can't be neglected.

User-driven replication is proposed in [20]. The users intervene in the selection of the trusted friends that store the replicas. This set of friends called TPS (*Trusted Proxy Set*) is then optimized according to two parameters: geographical location and online time period. For the update process, we should copy only the new events. This minimizes the network load, but the proposed algorithm needs overlapping the online time period of the nodes to synchronize. Unfortunately, this is not always guaranteed.

Ivy [15] a multi-user read/write peer-to-peer file system is proposed. It manages the concurrent writes separately. Every user saves his modifications in a log stored and replicated based on a DHT. The user, when reading,

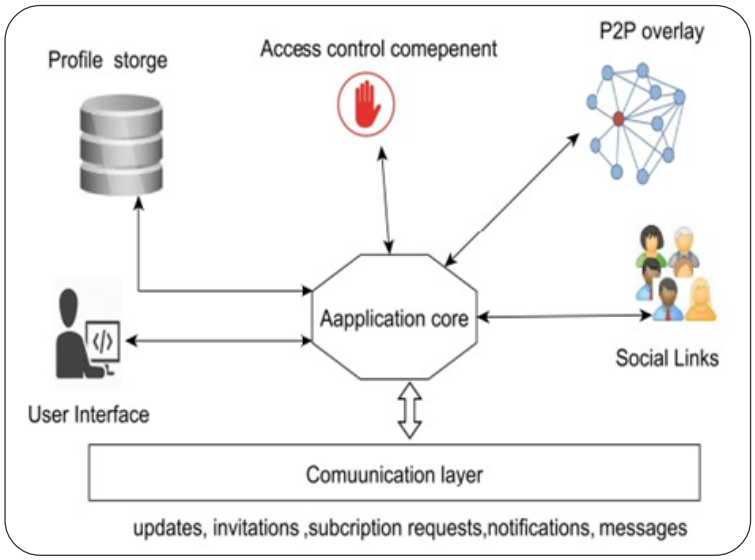


Figure 1. Node functionalities in P2P social network

collects all the logs and creates a consistence version. This approach avoids the use of any lock to protect data and doesn't need any trust on the other writers (*the users can choose the logs to use or to ignore*). We can't apply this system in the context of social network. This is due to the temporal dependence among the users writes: the order of writing is crucial for the consistency of the whole discussion. Also, in a social network, the user can freely express their opinions and the reader is not supposed to exclude some contributions.

Logoot is another collaborative editing system presented in [4]. It is designed based on P2P architectures, allowing a large number of editors and ensuring CCI consistency (*Casualty, Convergence and Intention preservation*). The system deals with text files organized in couples (*line, text*) and it allows two operations (*insert line and delete the line*) with position specification. The main problem treated is how to collect parallel update operations from various replicas in order to create a consistent final version. Wikipedia is used for tests. Logoot-undo [5] is an extensible version that supports undo operations.

The same idea is treated in another decentralized editing system [8] where the position of insertion or suppression is represented with a real number in a way that prevent the generation of the same position twice. These collaborative editing systems can be adapted to the decentralized social networks. Contrariwise, the eventual consistent model is not suitable to the ubiquitous nature of the social network content (*post and comments*).

Lilliput [31] is a replication scheme that separates the storage of static bulk data (*videos and photo albums*) from the essential social glue (*e.g. basic profile information, frequent updates, notifications, and personal messages*). It aims to increase availability and to reduce the overhead of the update operations. The bulk data is supposed to be rarely used. It is encrypted and stored on a central server. The recent data and the social graph are replicated in the P2P overlay. Every user creates its own overlay and invites other nodes to join it. This solution increases clearly the availability, but the users' privacy is not completely guaranteed since the choice of replicas does not consider friendship or trust. In addition, the central server manages the data and can build an idea about the users and their exchanges [35].

In this work, we manage the profiles replication in a decentralized social network. We propose to organize the network in replication groups. To ensure the consistency of the replicated profiles, we use a log inspired from the collaborative editing systems. This log allows reducing the cost of the update operation. Also, the use of log file ensures to transmit the updates without a necessity to overlap the connection periods. The proposed solution is detailed in the next section.

3. Proposed Model

In this section, we propose a new model to manage the replication of profiles in decentralized social networks. To ensure high availability, the users' devices are grouped in replication groups. The users' profiles are then duplicated in all the devices belonging to the group. We use two models to update the different copies of the same profile:

• Synchronous model based on direct update messages. This model is applied to synchronize the on- line devices.

• Asynchronous model to update the devices after a disconnection. It is based on a file called log of events".

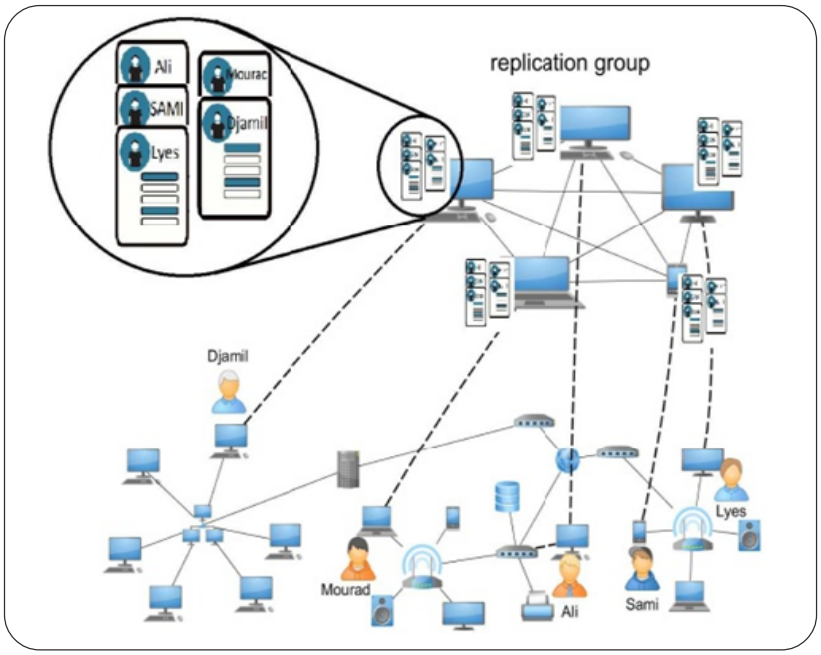


Figure 2. Profiles replication

We propose to store, all the writing undergoes by a profile in this file. These update models avoid copying the whole profile in every update process. Only, the new writes are exchanged. The lookup process is based on a priority order within the replication group in order to balance the load between the devices.

3.1 Profiles Replication

We propose to duplicate the users' profiles to increase their availability. Each profile is stored in the user's device and has other copies distributed in other users' devices.

The users' devices are grouped in replication groups. Each device can belong to one group only. The symmetric replication [28] is then adopted: if the users ($U1, U2,.., Un$) construct a replication group; every device in this group stores a copy of every associated profile ($P1, P2, .. , Pn$).

Each profile is then available if at least one device of the replication group is online.

According to this architecture, a profile is available, if at least one device from the replication group is online. Also, if one device is online, all the profiles affiliated to the group are available. In Figure 2 a replication group is represented.

As an example, the replication level is limited to 5. The users *Ali, Sami, Lyes, Mourad* and *Djamil* collaborate to store their profiles.

For example, the profile of *Ali* is primarily stored in *Ali*'s device and has 4 copies in the devices of the other members. *Ali* profile can be reached if one of the five nodes is online. Also, *Ali* stores in his device a copy of each profile belonging to the group. The replication group construction is crucial. It may consider different criteria's: performances, trust, and the connection habits during the day.

3.2 Lookup Profile

To consult or interact with a given profile, the users should know the devices responsible of its storage. The lookup service directs the users to the replication group responsible for the desired profile. A copy of the profile is stored in each device in the group. Many nodes can be online at the same time. The choice of the device to consult the profile has an important effect on the system performances.

Two strategies can be considered:

• With a random choice, the same profile can be consulted and modified in different nodes simultaneously. This will generate concurrent writes in the event logs. The asynchronous update is more complicated.

• A second issue is to define a priority between the nodes. If two or more devices are online, the device with the highest priority serves the users. With this strategy, the contribution of the different devices in the replication group will not be balanced: A device with high priority and long connectivity serves the users more than the other devices.

Given this, we propose to define a priority order for each profile. A device with the highest priority to serve a given profile should have a low priority for another profile. A priority table is created and filled during the construction of the group. Table1 illustrates this priority order for the example presented in Figure 2.

Device of	Profile of Ali	Profile of Mourad	Profile of Djamil	Profile of Lyes	Profile of Sami
ALI	1	5	4	3	2
Mourad	2	1	5	4	3
Djamil	3	2	1	5	4
Lyes	4	3	2	1	5
Sami	5	4	3	2	1

Table1. Example of priority distribution in the replication group

Each column of the table defines the device priority to serve the profile. The devices are contacted according to the priority defined in this table. Low value of priority designs a high priority.

For example, to consult the profile *Lyes* we contact the device of *Lyes* (*priority 1*). If this device is offline, we contact the device of *Sami* (*priority 2*). In this case, the device *Djamil* will serve the profile *Lyes* if only all the other devices are offline.

The device priority is changed for each profile in a way that a device with important priority to serve a profile has a weak priority to serve the other profiles.

The table creation and filling can be summarized in the algorithm 1.

With this strategy, both reads and writes on a given profile at a given time take place in the same device and the load is balanced between the different devices of the replication group.

The priority table is then exploited to generate an ordered list for each profile. The elements of this list contain links to the devices storing the profile. Figure 3 illustrates the ordered lists generated from Table1.

For each profile, we store only the ordered list of devices (*it's not necessary to store the entire priority table*). Since there is no central server or index, we propose to store these lists in a DHT structure. The profile identifier is used as a key in the hash function. And for each key (*Profile*), the list of devices storing the profile is associated.

```
Algorithm 1 : Priority table
Begin
//P: Pro_le
//R: Replication degree
//T: Priority table
//Each line represents a device and each
//column represents a profile
For P=1 to R do
//high priority (1) to the owner device
      T[P][P]=1;
//increase priority until the end of the column
      Priority=2;
For i=P+1 to R do
            T[i][P]=Priority ;
            Priority=Priority+1 ;
      EndFor
//decrease priority until the begin of the column
      Priority=R;
      For i=P-1 down to 1 do
            T[i][P]=Priority ;
            Priority=Priority-1 ;
      EndFor
EndFor
End
```

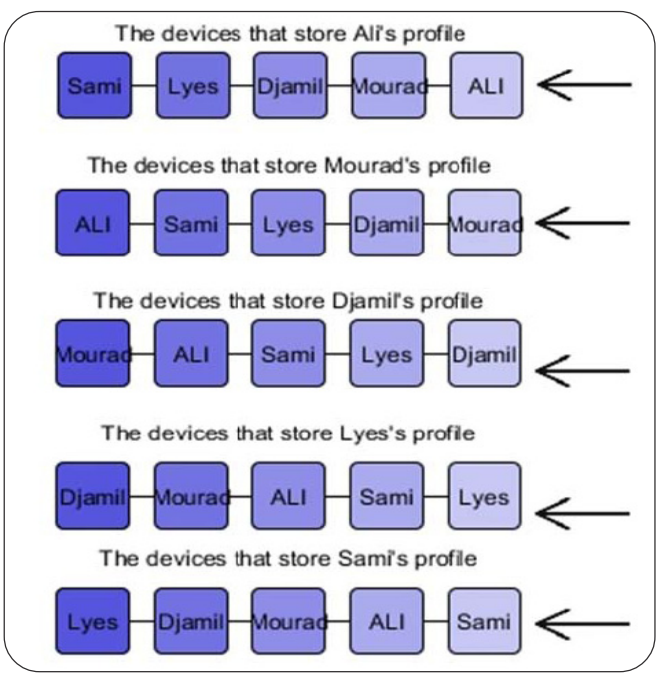


Figure 3. The ordered lists of devices

In the connection phase, the users communicate the identifier of the desired profile to the DHT. The DHT return the list of the devices storing each one a copy of the profile. Then, the users will contact these devices (*if they are online*) to consult and interact with the profile. Figure 4 illustrates an example of the lookup service.

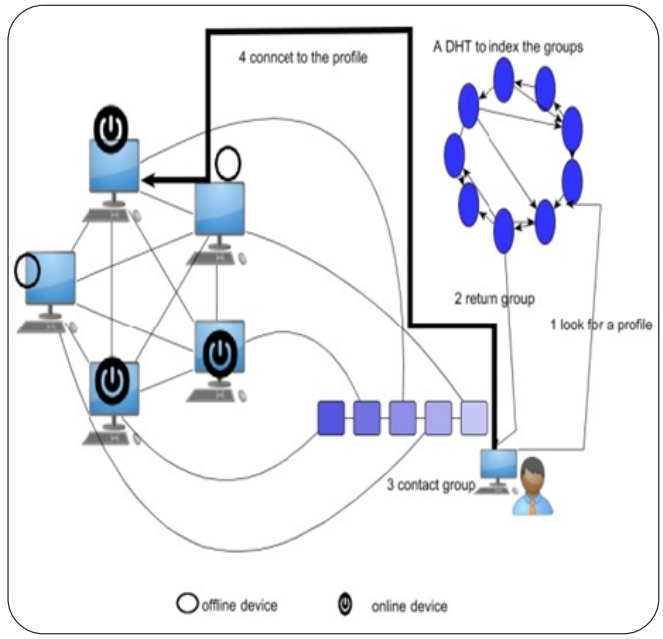


Figure 4. Example of Lookup service

As shown in Figure 4, the user requesting a profile, contacts a DHT overlay to retrieve its replication group (*step1*). The hash function uses the profile identifier to find its replication group. The response to this request is an ordered list (*step2*). Each element of the list is a link to a device that stores a copy of the profile. The user brows the list and contact the devices, according to the fixed order (*step 3*). Once, a contacted device is online, the user connects to the profile (*step 4*). The list browsing is then stopped. In the example, the user contacts 3 devices to connect to the first online one. The following algorithm explains the lookup process.

```
Algorithm 2: Lookup
Begin
List=DHT.Lookup(ID_Profile)
FOUND=FALSE;
While (List !=NULL) and (Not FOUND)do
      Contact List.Device
      Wait (time out)
      If ACK then
            Connect to List.Device;
            Found=true;
      Else
```

```
        List=List.NEXT
    EndWhile
    If (Not found) then
        Write ("The profile is
        currently unavailable");
    End
```

3.3 Events log

The different devices of a replication group should communicate to ensure the consistency of the different copies of the same profile. They exchange the different writes happen in the profiles. A direct communication (*in synchronous mode*) is possible when two devices are online. Contrariwise, if two devices are online in separate periods, they can't be updated correctly. So, we propose to use a log file to store all the writes happened on each profile. This log (*we call it log of events*) allows an asynchronous communication within the replication group. It is used to update the profiles' copies with the writes happened while the devices are offline.

The events log is a file associated to each user profile. It is used to store all the events happening in the online nodes. It will be used later to update the versions stored on disconnected nodes. This log file is introduced to avoid the duplication of the whole profile in every update operation. This mechanism may reduce the overhead due to the replication. The log size is very small compared to the profile size. The events reside for a sufficient time to be copied in all versions.

To ensure the coherence of the profile content, the events are time-stamped. The update mechanism may insert the different events according to the time noted in the log. The events log is a series of structured events that concerns one profile. Every event contains the following information:

- **Type:** Post, comment, like,...
- **Writer:** The identifier of the user who generated the event.
- **Time and date:** To insert the event in a correct order in the profile.
- **The content:** In general a text.
- The comment or the post concerned by the event.

We present some examples of the events stored in the log file.

3.3.1 Post Event

```
<event type="post ">
<date> 11/11/2016 </date>
<time> 14:55:25 </time>
<writer> IDuser1 </writer>
<idpost>idpost2 </idpost>
<post>"this is a new post "</post>
</event>
```

3.3.2 Comment Event

```
<event type="comment ">
<date> 11/11/2016 </date>
<time>14:30:25 </time>
<writer> IDuser2 </writer>
<idpost>idpost1</idpost>
<idcomment> idcomment </idcomment>
<comment>" my comment " </comment>
</event>
```

3.3.3 Like Event

```
<event type="Like">
<date> 11/11/2016 </date>
<time> 14:35:25 </time>
<writer> IDuser1 </writer>
<idpost>idpost1</idpost>
<idcomment> idcomment </idcomment>
</event>
```

A sample of a complete log is illustrated in figure 5.

The log file contains the following information:

- Information about the corresponding profile,
- Information about the devices that store the replicas: the device identification and the time of the last use of the log by this device,
- A set of time-stamped events.

3.4 Log Storage

Based on the hypothesis that the log size is smaller than the profile size, the log file is stored in a larger replication scheme. We aim that a disconnected node can be informed about all the events happened during its disconnection. A very high availability should be ensured for the log file. This is necessary to synchronize two nodes that never meet.

In our case, we choose to construct an overlay based on a DHT algorithm (*see Figure 6*). We use a hash function that calculates a primary location to store the log.

However, the replication nodes are automatically deduced [16]. The same hash function is then used to find the log

```
<log>
<profile>ID_user1</profile>
<replicas>
<replica>
<device>iddevice1</device>
<Last_log_use_time>
<date> 11/11/2016 </date>
<time> 11:25:00 </time>
</Last_log_use_time>
</replica>
<replica>
<replica>
<replica>
<replica>
</replicas>
<events>
<event type="comment ">
<date> 11/11/2016 </date>
<time>14:30:25 </time>
<writer>IDuser2 </writer>
<idpost>idpost1</idpost>
<idcomment> idcomment </idcomment>
<comment>" my comment " </comment>
</event>
<event type="Like">
<date> 11/11/2016 </date>
<time> 14:35:25 </time>
<writer> IDuser1 </writer>
<idpost>idpost1</idpost>
<idcomment> idcomment </idcomment>
</event>
<event type="post ">
<date> 11/11/2016 </date><time> 14:55:25 </time>
<writer> IDuser1 </writer>
<idpost>idpost2 </idpost>
<post>"this is a new post "</post> </event>
</events>
</log>
```

Figure 5. A Sample of a Log file

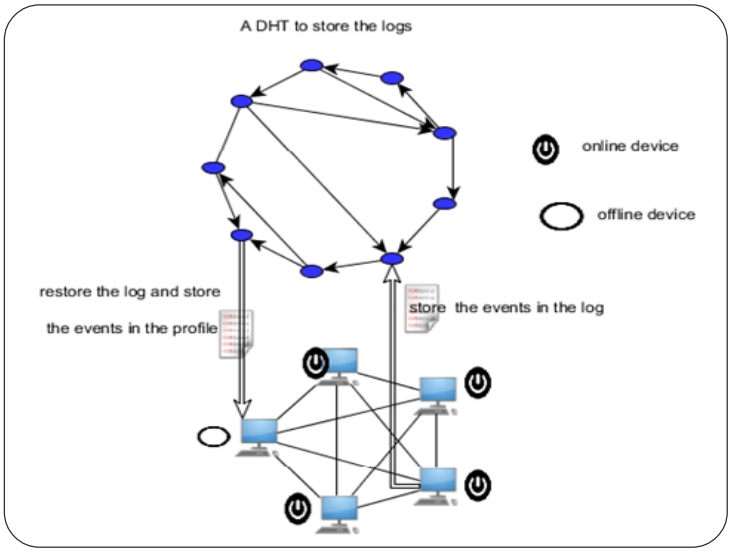


Figure 6. Log files storage

file in order to update the profile. In our case, the hash function uses the profile identifier as an entry. Any DHT algorithm can be used without significant effect on the system performances. The proposed solution is generic regarding the P2P underlying architecture.

3.5 Profile Update Algorithms

Every version of a user profile evolves according to three update algorithms. The situation of the storing node against the users' interactions defines which algorithm to apply.

• **Direct Update:** On the node receiving a modification directly from a user.

• **Synchronous Update:** Applied to the online nodes of the replication group.

• **Asynchronous Update:** Executed after the return of a node to the network after a disconnection.

3.5.1 Direct Update

This algorithm is executed on the active node (*the node used to consult the profile*). It is launched when a user (*reader, friend, visitor or the profile's owner*) tries to publish something or react with an old content. The modification is noted directly in this profile version and the other nodes should be informed about it. When a user performs a write, it is directly applied to the online version of the profile. The node storing this version should inform the other members of the replication group about it. The online nodes receive a message containing the event. The event is also stored in the log events to be considered by the disconnected nodes. The lookup service should define a priority between the devices storing the same profile. At any given time a profile is read and written through one device. This omits concurrent writes to a log file.

```
Algorithm 3: Direct Update
Begin
1: Store the event on the profile;
2:Send the event to all the nodes in the
replication group;
3:Store the event in the events log;
4:Last_update_time=Current_time;
End
```

3.5.2 Synchronous Update

To consult a profile, the users can be directed to an online node of the replication group. So, all the online nodes should store the same version of each profile. After receiving a new write (*eg. new post, comment...*), the active node informs the other online nodes about this event (*see Algorithm 3 step 2*). This event is noted immediately by running the synchronous update algorithm. The online nodes of a replication group exchange messages containing the writes happened on the profiles that they serve. When receiving an event, the node applies it on its version. It is not necessary to store it in the log file. This operation is done by the first node that receives the write (*step3 of Algorithm 3*).

```
Algorithm 4: Synchronous Update
Begin
1:Store the write on the profile;
2:Last_update_time=Current_time;
End
```

3.5.3 Asynchronous Update

This algorithm should be a part of the join procedure. After the join process, a node becomes online. But, it can't serve users until all the profiles stored are updated. The asynchronous update is not performed in the profile replication group, but it uses the events log stored in the underlying P2P architecture.

After a certain period of disconnection T, the profiles stored in a node have other versions probably modified. The asynchronous update process informs the reconnecting node about these modifications. This algorithm is executed for each profile stored in this node. Only the events that happened after the last update instant are considered. The instant of these
asynchronous update is stored.

```
Algorithm 5: Asynchronous Update
Begin
For each profile stored in the node do
      1:Open the log file;
      2:Extract all the events timestamped
      after
      Last_update_time ;
      3:Store the events on the profile;
      4:Last_update_time=Current_time;
      5:Store in the log Last_log_use_time=
Current_time;
EndFor
End
```

3.6. Illustration of the Update Model

In this sub-section, we present a scenario of the execution of the different update algorithms. We suppose that only two users: *Ali* and *Sami* are online. All the five profiles are

consulted throw *Ali* and *Sami*'s devices.

• If *Ali* posts a new publication this post is saved using the direct update algorithm in *Ali* 's device. So, *Ali* stores the post in his version and sends the event to *Sami*.

• *Sami* launches the synchronous update algorithm.

• Also, *Ali* stores this post as a new event in the events log corresponding to his profile (*see figure 7*).

• After a few times, both *Sami* and *Ali* disconnect and *Mourad* joins the network.

• *Mourad* collects the log events of all the profiles that he stores and uses the asynchronous update algorithm.

As a result, the version of *Ali*'s profile stored in *Mourad* device contains the last post of *Ali* even if it happened during his disconnection on the device of *Ali* and *Ali* and *Mourad* have never been online simultaneously.

3.7. Log Maintenance

The events log stores all the writes that happened on the different versions of the profile. The oldest events should be eliminated to reduce the log size. We should only delete the useless events. They are events already applied by the offline nodes.

During reconnection, the joining node applies the asynchronous update and stores the current time in the log: **Last_log_use_time** (*algorithm 5, step5*). Then, the log contains a **Last_log_use_time** value for every version of the profile.

The log maintenance procedure deletes all the events that happened before the oldest **Last_log_use_time**.

4. Performance Evaluation

In this section, we discuss the performances of the proposed model. We focus principally on two major parameters:

• **The Profile Availability:** Every profile is stored in many copies in various users' devices. A profile is considered available if at least one storing device is online. We measure the profiles availability in a period of 24 hours by varying two factors: the replication degree (*number of devices storing the profile*) and the connection period (*the duration of connectivity in 24 hours*).

• **The Load Balancing between the Devices:**
We test the lookup strategy presented in (*algorithm 2*).


```
<log>
 <profile>Ali</profile>
<replicas>
<replica>
<replica>
<replica>
<replica>
<replica>
 </replicas>
<events>
<event type="post ">
<date> 08/08/2018 </date><time> 14:00:25 </time>
<writer> Ali </writer>
<idpost>idpost12500 </idpost>
<post>"What a beautiful summer day"</post>
 </event>
<event type="comment ">
<event type="comment ">
<event type="Like">
<event type="post ">
<event type="comment ">
 </events>
 </log>
```

Figure 7. New event writing in the log file

We aim that all the collaborating devices contribute by the same charge when storing profiles and serving users.

4.1. Profiles Availability

We observe the profiles availability according to two factors: the replication degree and the average connection time of different devices. We vary the replication degree between 4 and 8 replicas. Then, the connectivity time for 24 hours is measured with the assumptions that all the devices are online for the same duration. This connection duration also varies from 4 to 12 hours per day and it is distributed in a random way in an interval of 24 hours. According to our proposed architecture, a profile is available if at least one node from the replication group is online. The Figure 8 illustrates the simulation result.

It is clear that the availability rate increases if the replication group is greater. More devices storing the profile increase the chance that the profile is reachable. However, a large replication group risks overloading the network by the update messages. So we look for other criteria that can ensure a sufficient availability even with a smaller replicas number.

We observe that the average connection time is more effective: with a short connection period, adding new replicas hasn't important effect. We found that the choice of the devices to construct a replication group is more important than creating a group with a large number of devices.

4.2 Load Balancing in Replication Groups

We observe here the effect of the lookup strategy within the replication group on the load balancing between the collaborating nodes. In Figure 9, we determine the responsible node for each profile.

With a replication group of 5 devices that connect 8 hours a day, profiles are assumed to be online for 86% of the time. We observe that the profiles are served by the different devices. For example, the profile of *Ali* is consulted from *Ali*'s device more than *Djamil*'s profile.

To illustrate the load balancing between the different devices, we observed the number of profiles served simultaneously by the different devices in the replication group. The results are shown in Figure 10.

We vary the connection duration per day and we calculate for each device, the average number of the profiles served simultaneously. We observe that the lookup strategy balances the load between the devices. For any given connection period, the device serves the same number of profiles. It is clear that the load decreases when the connection period increases.

4.3 Discussion

We evaluate the profiles availability and the load balancing according to the replication group size and the average duration of connection. The data are randomly generated because of the absence of a real data from decentralized social network (*most of the proposed approaches are academic works*). Even the data sets returned by the centralized social networks such as Facebook or Twitter do not give a complete view of the users' behavior (*in general because of privacy considerations*).

The availability rate increases according to the number of replicas. The effect is clear for smaller groups: adding a new device improves the availability rate up to 10%. The effect of adding a new device decreases if the group is large. We may get to the point that adding a new device may not quite improve the availability: the connection period of the added device is already covered by the other devices.

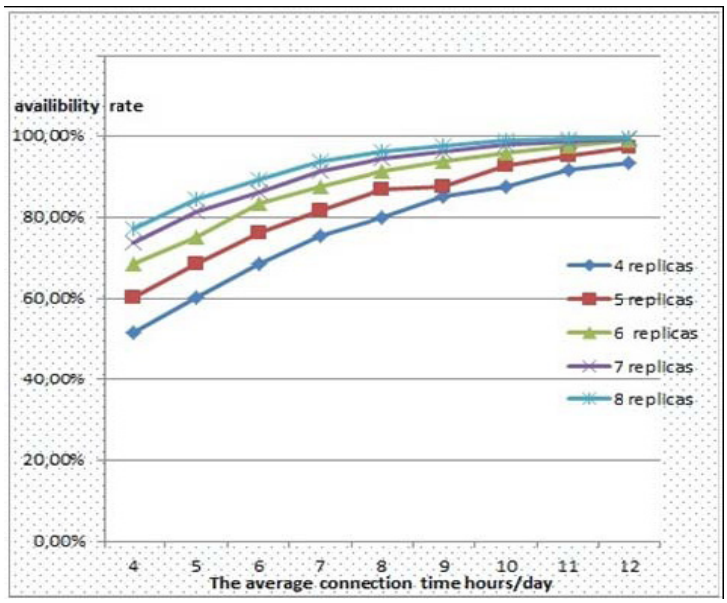


Figure 8. Profiles availability

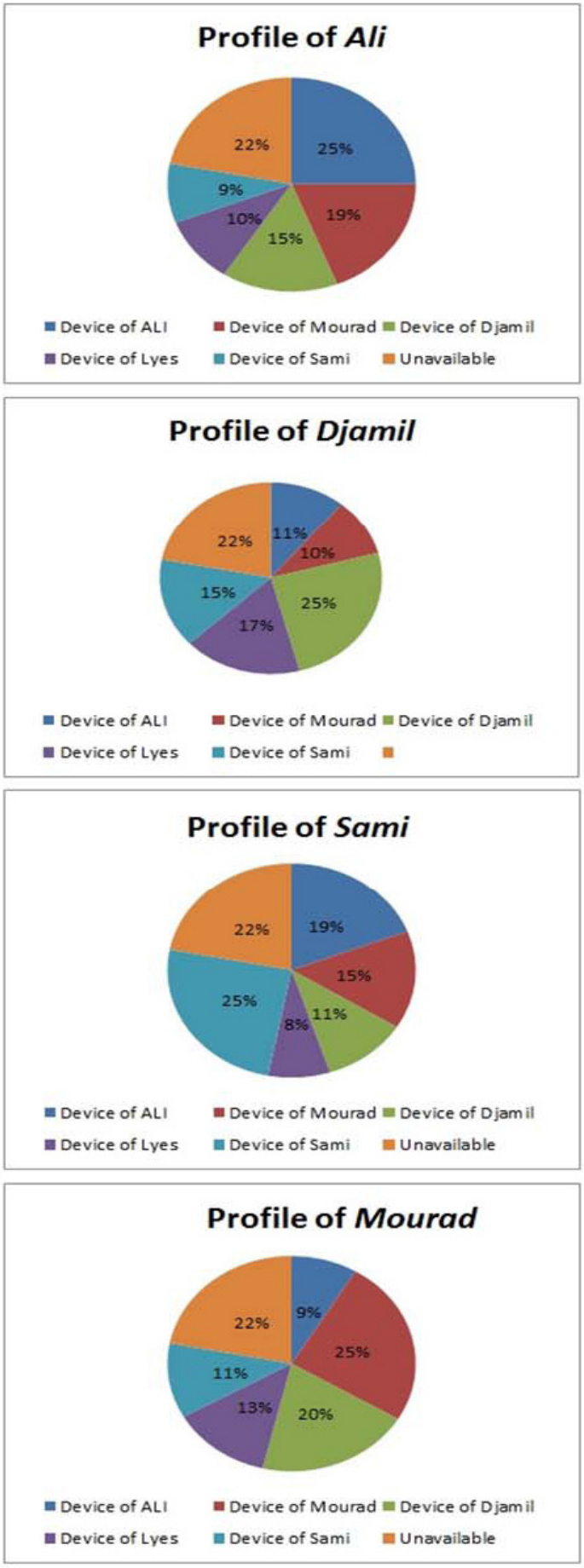
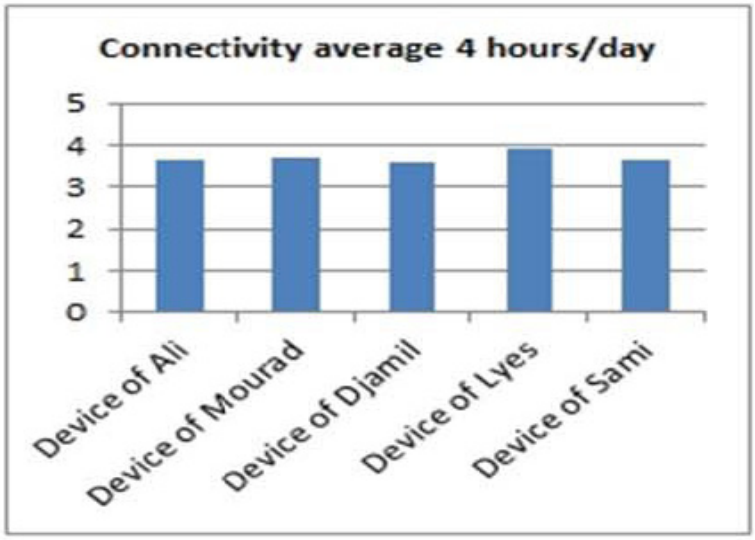
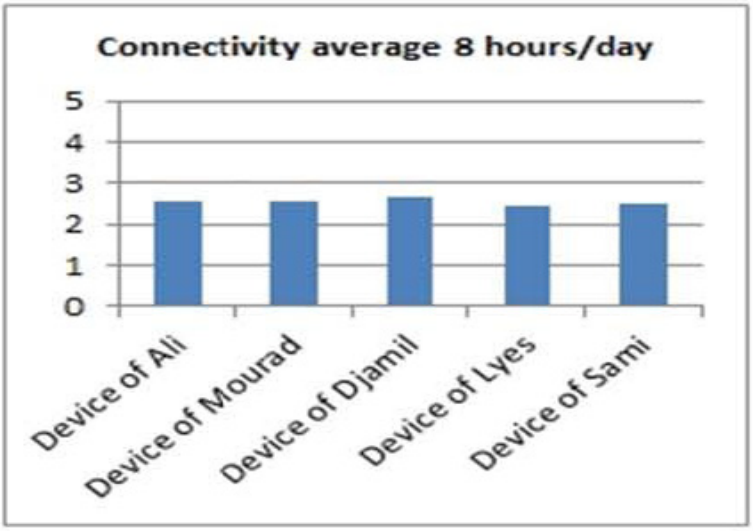


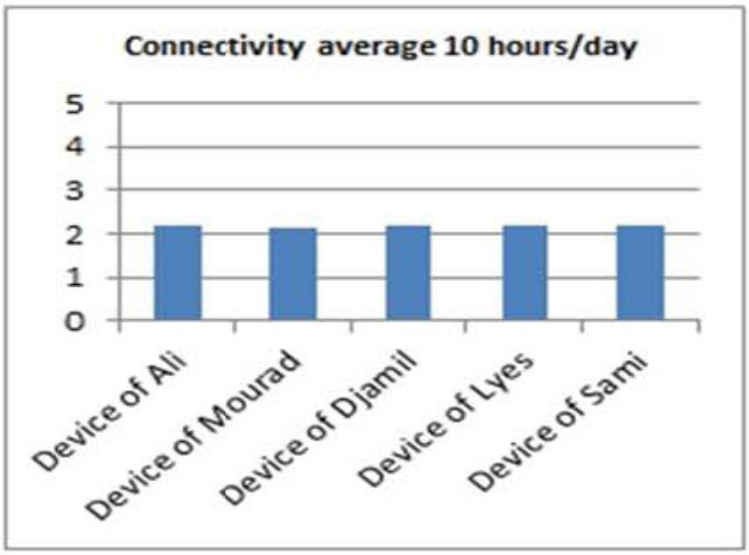
Figure 9. The profiles access

The second factor that affects the availability is the connection duration (*the time that the users' devices are online*). It has been noticed that if the users have a sufficient duration of presence, the availability rate is acceptable even in the case of a small replication group. We conclude that the creation of the replication group should be based primarily on connectivity information.

It is inadvisable to try to expand the replication group. This has a small effect on the availability. In addition, it may degrade the network load. Every update operation on a user profile generates a synchronization message sent to all the online devices. Also, the devices should look for the log files every session start. They have to restore a log file for each stored profile.







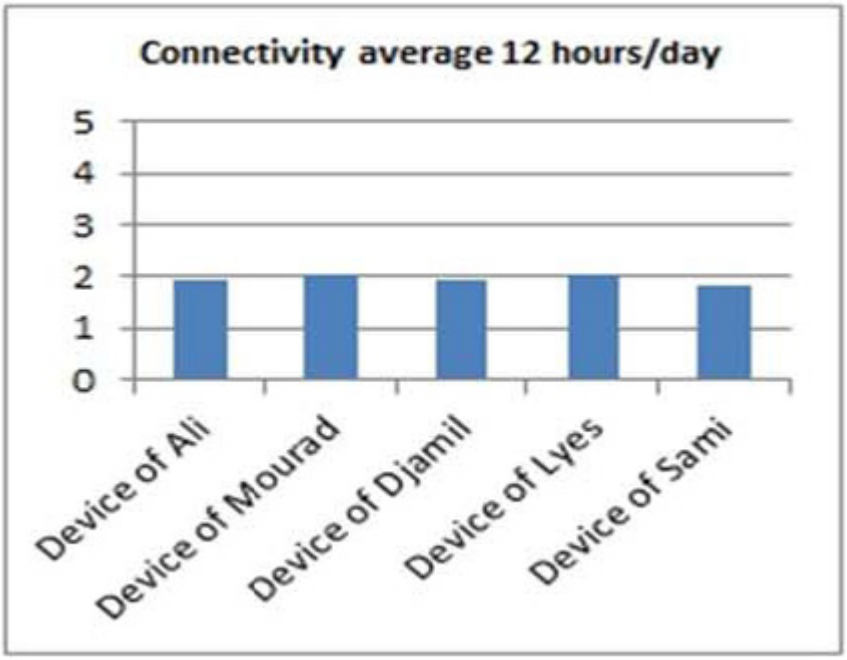


Figure 10. Number of profiles served per device

The load balancing is observed to evaluate the lookup strategy. The simulation shows that the cooperating devices contribute with a balanced load. The load is quantified as the number of profiles served simultaneously. Although the devices are responsible of storing a number of profiles, they rarely act as a server for all these profiles. We admit that the devices serving more popular profiles may support an extra load. We advise to define a load balancing strategy that considers the popularity of profiles or publications.

5. Conclusion and Future Works

In this paper, we proposed a model to update replicated profile in a Decentralized Social Network. Every member in the SN stores his profile on his own device. Then this profile may be replicated in other users' devices in order to improve its availability.

Many replication schemes are proposed. We have focused on the network load generated when updating different copies of the same profile. We proposed to use a log file to store all the writes happened on a given profile. Every device that stores a copy of this profile should consult this log file after every disconnection period.

Thanks to this log, the new writes are copied in all the replicas without copying the whole profile (*an important size after a while*). In addition, the log file allows synchronizing to replicas stored in two devices that never meet (*never been online simultaneously*).

In order to balance the load between the devices storing the same set of profiles, we introduced a priority order. This order regulates the number of profiles served simultaneously by any device.

The strategy according to which the devices are selected to create a replication group wasn't discussed in this paper. It may be the object of a future work. Also, we envisage introducing other consistency model to optimize the update process. It will be interesting to evaluate our updating algorithms basing on real data extracted from the existing OSN or by implementing the proposed architecture.

References

[1] Shahriar, N., Chowdhury, S. R., Sharmin, M., Ahmed, R., Boutaba, R., Mathieu, B. (2013). Ensuring beta-availability in p2p social networks. *In: I*EEE 33rd International Conference on Distributed Computing Systems Workshops., p. 150-155.

[2] Paul, T., Famulari, A., Strufe, T. (2014). A survey on decentralized online social networks. *Computer Networks*, vol.75, p. 437-452.

[3] Badis, L., Amad, M., Aissani, D., Bedjguelal, K., Benkerrou, A. (2016). ROUTIL: P2P routing protocol based on interest links, *In:* International Conference on Advanced Aspects of Software Engineering (ICAASE), p. 118- 122, 2016.

[4] Weiss, S., Urso, P., Molli, P. (2009). Logoot: a scalable optimistic replication algorithm for collaborative editing on p2p networks, *In:* 29th IEEE International Conference on Distributed Computing System ICDCS'09, p. 404-412, 2009.

[5] Weiss, S., Urso, P., Molli, P. (2010). Logoot-undo: Distributed collaborative editing system on p2p networks. *IEEE Transactions on Parallel and Distributed Systems*, vol.21(8), p. 1162-1174.

[6] Graffi, K., Gross, C., Stingl, D., Hartung, D., Kovacevic, A., Steinmetz, R. (2011). LifeSocial. KOM: A secure and P2P-based solution for online social networks, *In*: 2011 IEEE Consumer Communications and Networking Conference CCNC, p. 554-558.

[7] Druschel, P., Rowstron, A. (2001). PAST: A large-scale, persistent peer-to-peer storage utility, *In*: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, p. 75-80.

[8] Ahmad, M., Imine, A. (2015). Decentralized Collaborative Editing Platform, *In*: 16th IEEE International Conference on Mobile Data Management, Vol. 1, p. 323-326.

[9] Jahid, S., Nilizadeh, S., Mittal, P., Borisov, N., Kapadia, A. (2012). DECENT: A decentralized architecture for enforcing privacy in online social networks, *In:* Pervasive Computing and Communications Workshops (PERCOM Workshops), p. 326-332.

[10] Sharma, R., Datta, A. (2012). Supernova: Super-peers based architecture for decentralized online social networks, *In* Fourth International Conference on Communication Systems and Networks (COMSNETS 2012), p. 1-10.

[11] Strufe, T. (2009). Safebook: A privacy-preserving online social network leveraging on real-life trust, *IEEE Communications Magazine*, vol 95.

[12] Liu, D., Shakimov, A., Cceres, R., Varshavsky, A., Cox,L. P. Confidant: protecting OSN data without locking it up. *In:* ACM/IFIP/USENIX International Conference on *Distributed Systems Platforms and Open Distributed Processing*, p. 61-80, Springer Berlin Heidelberg. 2011.

[13] Liu, Y., Vlassov, V. (2013). Replication in distributed storage systems: State of the art, possible directions, and open issues. *In: I*nternational Conference on CyberEnabled Distributed Computing and Knowledge Discovery (CyberC), p. 225-232.

[14] Van Eecke, P., Truyens, M. (2011). Privacy and social networks. *Computer Law & Security Review*, 26 (5) 535-546.

[15] Muthitacharoen, A., Morris, R., Gil, T. M., Chen, B. (2002). Ivy: A read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review*, 36 (SI) 31-44.

[16] Rowstron, A., Druschel. P. (2001). Pastry: Scalable, distributed object location and routing for large-scale Peer-to-peer systems, *In:* Proc. IFIP/ACM Middleware 2001, Heidelberg, Germany, (November).

[17] Boyd, D., Ellison, N. (2010). Social network sites: definition, history, and scholarship, *IEEE Engineering Management Review*, vol 3(38), p. 16-31.

[18] Datta, A., Buchegger, S., Vu, L. H., Strufe, T., Rzadca, K. (2010). Decentralized online social networks*, In: Handbook of Social Network Technologies and Applications*. p. 349-378. Springer US.

[19] De Salve, A., Guidi, B., Ricci, L. (2017). Evaluation of Structural and Temporal Properties of Ego Networks for Data Availability in DOSNs, *Mobile Networks and Applications*, p. 1-12.

[20] Narendula, R., Papaioannou, T. G., Aberer, K. (2012). A decentralized online social network with efficient user-driven replication. *In:* Privacy, Security, Risk and Trust (PASSAT), International Confernece on Social Computing (SocialCom), pp. 166-175. (September).

[21] Bouadjenek, M. R., Hacid, H., Bouzeghoub, M. (2016). Social networks and information retrieval, how are they converging? A survey, a taxonomy and an analysis of social information retrieval approaches and platforms, *Information Systems*, vol. 56, p. 1-18.

[22] Xu, G., Wu, Z., Zhang, Y., Cao, J. (2015). Social networking meets recommender systems: survey, International Journal of Social Network Mining, 2 (1) 64-100.

[23] Conti, M., De Salve, A., Guidi, B., Pitto, F., Ricci, L. (2014). Trusted Dynamic Storage for Dunbar-Based P2P Online Social Networks, *In: OTM Conferences*, p.400-417.

[24] Diaspora. [Online]. Available: *https://joindiaspora.com/*

[25] Schwittmann, L., Boelmann, C., Wander, M., Weis, T. (2013). SoNet–Privacy and replication in federated online social networks*, In:* Distributed Computing Systems Workshops (ICDCSW) 51-57.

[26] Shakimov, A., Varshavsky, A. Landon, L.P., Caceres, R. (2009). Privacy, cost, and availability tradeoffs in decentralized osns, *In:* Proceedings of the 2nd ACM workshop on Online social networks, p.13-18.

[27] Wilson, C., Steinbauer, T., Wang, G., Sala, A., Zheng, H., Zhao, B. Y. (2011). Privacy, availability and economics in the polaris mobile social network, *In:* Proceedings of the 12th Workshop on Mobile Computing Systems and Applications,ACM, p. 42-47.

[28] Ktari, S., Zoubert, M., Hecker, A., Labiod, H. (2007). Performance evaluation of replication strategies in DHTs under churn. *In:* Proceedings of the 6th International conference on Mobile and ubiquitous multimedia, ACM. p. 90-97.

[29] Amad, M., Meddahi, A., Aissani, D., Zhang, Z. (2012). HPM: a novel hierarchical peer-topeer model for lookup acceleration with provision of physical proximity, *Journal of Network and Computer Applications*, vol.35 (6) 1818-1830.

[30] Castiglione, A., Cattaneo, G., Cembalo, M., Petrillo, U. F. (2013). Experimentations with source camera identification and online social networks. *Journal of Ambient Intelligence and Humanized Computing*, 4 (2) p. 265-274.

[31] Paul, T., Lochschmidt, N., Salah, H., Datta, A., Strufe, T. (2017). Lilliput: A Storage Service for Lightweight Peer-to-Peer Online Social Networks, IEEE 26th International

Conference on Computer Communications and Networks (ICCCN),

[32] GUIDI, Barbara, MICHIENZI, Andrea, et ROSSETTI, Giulio. (2017). Dynamic community analysis in deentralized online social networks, *In:* European Conference on Parallel Processing, Springer, Cham, p. 517-528, .

[33] Koll, D., Lechler, D., Fu, X. (2017). SocialGate: Managing large-scale social data on home gateways, *In*: 2017 IEEE 25th International Conference on Network Protocols (ICNP) p. 1-6.

[34] DE SALVE, Andrea, MORI, Paolo, et RICCI, Laura, (2018). A survey on privacy in decentralized online social networks. *Computer Science Review*, vol. 27, p. 154-176.

[35] Greschbach, B., Kreitz, G., Buchegger, S. (2012). The devil is in the metadata—New privacy challenges in Decentralised Online Social Networks. *In*: IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), p. 333- 339, 2012.

[36] JOMSRI, Pijitra. (2016). A Combination Indexing for Image Social Bookmarking System to Improve Search Results, Journal of Digital Information Management, 14 (6).