LDConnect - A SoLiD Compliant Interface for the Facebook Social Graph

By Happy S. Enchill

Submitted to the
Department of Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

At the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2015

© Massachusetts Institute of Technology 2015. All rights reserved.

Author...........................................................................................................................................
Happy S. Enchill
Department of Electrical Engineering and Computer Science
August 18, 2015

Certified by.................................................................................................................................
Prof. Tim Berners-Lee
3COM Founders Professor of Engineering
Thesis Advisor
August 18, 2015

Accepted By.................................................................................................................................
Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee
LDConnect - A SoLiD Compliant Interface for the Facebook Social Graph

By Happy Enchill

Submitted to the Department of Electrical Engineering and Computer Science

August 18, 2015
In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT
LDConnect is a service that allows Crosscloud applications to easily interact with Facebook data. This paper describes the design of LDConnect and the benefits that it provides Crosscloud developers over other methods of interacting with Facebook data. LDConnect’s live translation of JSON data to RDF and vice-a-versa makes it easy for developers to run CRUD applications on Facebook elements using the same access methods they would use to access Crosscloud backends.

Thesis Advisor: Professor Tim Berners-Lee
Title: 3COM Founders Professor of Engineering
# Table of Contents

1. **Introduction** .......................................................................................................................... 5

2. **Accessing the Facebook Social Graph** ................................................................................. 7
   2.1 Facebook Graph API ......................................................................................................... 7
   2.2 Facebook Open Graph API .............................................................................................. 8

3. **Design and Implementation** ............................................................................................... 9
   3.1 Access Tokens .................................................................................................................. 10
   3.2 Handling Requests .......................................................................................................... 11
   3.3 Structure of Data ............................................................................................................. 11
   3.4 Objects and Representation ............................................................................................ 14
      3.4.1 Main Containers ....................................................................................................... 14
      3.4.2 Posts ......................................................................................................................... 16
      3.4.3 Photos ....................................................................................................................... 16
      3.4.4 Comments and Likes .............................................................................................. 17
      3.4.5 Friendship .............................................................................................................. 19
   3.5 Design Decisions .............................................................................................................. 20
      3.5.1 RDF Ontologies ....................................................................................................... 20
      3.5.2 Programming Language .......................................................................................... 21
      3.5.3 Representation of Likes and Comments .................................................................... 22
      3.5.4 Leveraging Pod Access Control ............................................................................... 23
      3.5.5 Scraping Scripts ....................................................................................................... 23

4. **Performance Analysis** ....................................................................................................... 24

5. **Future Work** ....................................................................................................................... 26
   5.1 Extending Support for Other Facebook Elements ............................................................. 27
1 Introduction

Despite the large number of social applications that exist today, users often have huge limitations on the applications that they can choose to use, and their mobility between applications. Most users find themselves limited to one primary application which stores their data and relationships. This system makes it difficult for new social applications to compete with existing applications, and for users to switch to new applications. The need for more freedom and mobility for users is what motivated the Crosscloud Project. The goals of the Crosscloud Project [1] are to:

1. Make it easy to develop multi-user (‘social’) software using only front-end development and respecting the rights and privacy of users.

2. Allow users the freedom to move easily among applications, hardware platforms, and social networks, while keeping their data and social connections.

To achieve these goals, the project proposes a system whereby data is stored in user defined spaces called pods, and accessed by front-end applications as needed. With this approach, application developers are no longer responsible for maintaining a backend storage system; addressing the first goal of the Crosscloud Project. New applications will have immediate access to a user base which includes users, their data, and their connections. By separating user data from front-end applications, users will have the option to use any application they prefer to view or access their data.

With several applications creating and modifying the same data, certain conventions to limit data corruption and promote application interoperability must be agreed upon. SoLiD is a
proposed set of conventions that aims to tackle this interoperability issue. SoLiD leverages existing W3C standards to provide conventions for data storage, data creation, data modification, identity management, and several other aspects of decentralized social applications [2].

The second goal of the Crosscloud project seeks to give users the freedom to use any application they desire, and still keep their data and connections. However, Crosscloud suffers from the same issue that it hopes to solve. Users that choose to switch to Crosscloud applications will not be able to easily interact with their data on non Crosscloud applications or leverage the social connections that they have built through non Crosscloud applications. For Crosscloud to achieve the level of freedom and mobility that it seeks for users, there must be a way to keep Crosscloud users connected to their data on other platforms and systems. One approach would be to create services that provide a SoLiD compliant interface to interact with data on non Crosscloud applications. LDConnect is an example of such a service.

LDConnect provides a SoLiD compliant mapping for Facebook data; it allows individuals who switch to Crosscloud applications to access their data that is stored by Facebook, and still interact with their connections that have yet to switch from Facebook. Although LDConnect does not support all SoLiD conventions right now, it supports a large subset of the conventions specified, and aims to support all applicable SoLiD conventions in the near future.

In the next few sections, this paper will discuss LDConnect and the features it provides, and some of the design decisions made when building it. Section 2 will discuss the current method of accessing and modifying Facebook data. Section 3 will expand on the design and implementation of LDConnect. Section 4 will analyze the performance overhead added by accessing Facebook data through LDConnect rather than using the Facebook API. Section 5 will
look at possible future work on LDConnect and on services that will provide SoLiD compliant interfaces for accessing data on non Crosscloud platforms.

2 Accessing the Facebook Social Graph

There are two main methods for getting data in and out of the Facebook social graph: the Facebook Graph API [3] and the Facebook Open Graph API [4]. The Facebook Graph API is described as “the primary way to get data in and out of the Facebook social graph.” [3]. By contrast, the Open Graph API allows applications to share custom stories on Facebook. Though both of these interact with linked data on some level, they do not provide expansive support for linked data applications, and do not promote interoperability between applications.

2.1 Facebook Graph API

The Facebook Graph API provides HTTP based methods to allow applications to query data in the Facebook Social Graph as well as insert and modify data in the graph. As an added feature, data retrieved from any graph queries can be returned as linked data [5] through http content negotiation. Although this is useful, it requires developers to have special knowledge of the Facebook Graph API. Also, applications cannot insert data into the graph by sending linked data through an HTTP Post method to an API endpoint. This results in the need for application developer to write user data twice: once to the user’s pod as linked data, and another to a Graph API endpoint as JSON.

In addition to this inability to post linked data directly to the graph, there are additional factors that render the Facebook Graph API incompatible with the decentralized social
applications created through the Crosscloud Project. First, the Graph API and the Facebook Social Graph are significantly different from the data stored by Crosscloud applications, and how that data is accessed. SoLiD conventions promote a storage structure that does not fit that of the Facebook Social Graph. The difference in structure creates a discourse between Facebook and Crosscloud applications, and this leads to large setup costs for application developers hoping to allow their users to interact with Facebook data. Second, data that is posted by an application to Facebook using the Graph API can only be updated by that application. Unfortunately this does not bode well in a decentralized environment, where it is necessary for applications to share data. The inability to modify data created by other applications means that data is still owned by an application rather than by the user. Should the user switch applications, they will be unable to update data that they created using any other application.

2.2 Facebook Open Graph API

The Facebook Open Graph API allows applications to share custom “stories” on Facebook. These stories often reflect the objects on your site or in your application, and the actions that users perform on those objects. An example of an object on one’s site would be a newspaper article and an action could be reading that article. The Open Graph API allows applications to share this information on Facebook as a story detailing that a user performed some action on some object on their application. In our newspaper example, when User A reads an article on your website (MyApp), your application can post the action to an Open Graph API endpoint, and it will appear on Facebook as “User A read an article on MyApp” with an accompanied image specified by the app developer.
To create an Open Graph story, an app developer would need to register their app with Facebook, as well as register any custom actions or objects on their site, then go through an approval process. To create an object, the developer would need to insert Open Graph HTML Markup, which is similar to RDFa, on their webpages. Open Graph meta tags are inserted into the head of the page to create an object for that page. The Open Graph API will then scrape the meta tags from where the content is hosted when an action is published linking a user to that object. If an application wants to update an object, it would modify the page where the data is hosted, then call an Open Graph API endpoint and specify that the content should be scraped again. Once an object is connected to a user through an action, it appears on the user’s timeline, and the appropriate feeds.

Though the Open Graph API has some support for writing linked data to the Facebook social graph, the process of creating custom stories seems cumbersome, and it still suffers from the same issues as the Graph API. Data that is posted by an application can only be modified by that application, the API does not support SoLiD conventions, and there is a setup cost and bureaucracy that developers must navigate. In addition, the stories that are posted are often not in the voice of the user, but of the application.

3 Design and Implementation

LDConnect presents a Linked Data Platform (LDP) [6] interface through which users can access their Facebook data. It exposes its own url structure and abstracts methods to access Facebook data by allowing application developers to read and post linked data to Facebook. To
the application LDConnect presents itself as an LDP server, and applications can access user’s data using similar methods.

### 3.1 Access Tokens

In order to query a user’s data using the Facebook Graph API, an access token that permits accessing and modifying that user’s data is needed. Obtaining this access token requires forwarding the user to a Facebook webpage where they will need to sign in, and indicate that they are willing to permit LDConnect to make requests on their behalf. Applications seeking to use LDConnect to access a user’s Facebook data will need to forward their users to https://he1.crosscloud.org/facebook_token where the user will be asked to permit LDConnect to access their data. The forwarding url accepts an “after” parameter that allows applications to specify where the user should be forwarded once the access token has been obtained. An example request would be:

https://he1.crosscloud.org/facebook_token?after=myapp.com/home

Once the access token has been obtained it will be stored by LDConnect, and used for future requests. The token is automatically renewed so the user will only have to perform this process once.

The process for obtaining access tokens also involves having the user specify whether they would like LDConnect to use web scraping to perform actions that are not possible through the Facebook API. Should the user choose to enable this setting, LDConnect will request the user’s Facebook username and password, and store it along with a copy of the user’s access_token on the user’s pod. The password and access_token will be accessible by
LDConnect, and any application that the user authenticates to. This leverages the access control methods of the user’s pod to protect the user’s Facebook login information and access token to a point. Poor judgment on the part of the user could lead to their information being leaked.

### 3.2 Handling Requests

The workflow for LDConnect begins with an application sending a request to the LDConnect service. Upon receiving the request, the service processes the request, and then builds its own HTTPS request to send to the Facebook Graph API. Once the request is built, LDConnect sends it to the appropriate endpoint, and awaits a response. When a response arrives, LDConnect processes the response, then builds an appropriate linked data response to send back to the requesting application. If no access token exists for the authenticated user, then a 403 status code will be sent back to the requesting application with information requesting that the user visits he1.crosscloud.org/facebook_token to obtain an access token.

On a GET request, the response received from the Graph API is used to build an RDF graph which is then serialized and sent back to the requesting application. The expected output of a GET request is described in Section 3.3. On a POST request, LDConnect parses the body of the request into an rdf graph then extracts out the necessary fields to build a request to send to the Facebook API. If the data is malformed, then a 400 status code will be sent back to the requesting application.

### 3.3 Structure of Data

The directory/container structure for LDConnect can be found in Figure 1. LDConnect supports the retrieval and creation of Albums, Posts, Photos, Comments, and Likes from the
Facebook social graph. It does so by using an access token specific to that user. This ensures that only data that the user has permission to view is returned.

On a high level, the url for accessing each user’s Facebook data is rooted at he1.crosscloud.org/{user_id} where user_id refers to the user’s Facebook id. The special uri he1.crosscloud.org/me refers to the authenticated user who is accessing the information. Within this container, you can find the friends resource, the fbprofile resource, the albums container, and the posts container; each of which is described in the following sections. A user’s root url will be referred to as the {base_uri}\(^1\) throughout this section.

The {base_uri}/albums/ endpoint is a container that holds the list of Facebook albums that this user has. Each album element is itself a container, and its uri is {base_uri}/albums/{album_id}. The albums container holds a special album called user_photos, which refers to the photos a user uploads, and the photos in which that user is tagged. The {base_uri}/albums/({album_id}|user_photos)/ endpoint will contain information about the album as well as the photos within the album, and the comments and likes on those photos. Each photo resource in the album has a likes resource and comments resource where the likes on that photo and the comments on that photo can be found respectively. Comments and likes on a photo can be found at the location {base_uri}/albums/{album_id}/comments_{photo_id} and {base_uri}/albums/{album_id}/likes_{photo_id} respectively. Uploading a photo involves sending an HTTP Post to its album uri, and creating an album is likewise done by sending an HTTP Post to its parent container {base_uri}/albums/.

\(^1\) {base_uri} refers to https://he1.crosscloud.org/{user_id} where {user_id} is the Facebook user id of the user whose data is being viewed.
The posts container holds all posts relevant to a user. This includes posts the users create themselves, posts others made to the user’s feed, and posts in which the user is tagged. The location of the posts endpoint is /base_uri/posts. This endpoint is structured similar to that of an album endpoint in that the posts endpoint holds post resources as well as their likes and comments resources. Each post resource is located at

he1.crosscloud.org
    /({user_id}|me)/
    /fbprofile
    /friends
    /albums/
        /({album_id}|user_photos)/
            /{photo_id}
            /likes_{photo_id}
            /comments_{photo_id}
        /likes_{album_id}
    /posts/
        /{post_id}
        /likes_{post_id}
        /comments_{post_id}

**Figure 1: Structure of LDConnect**

Each directory listed above is a container that holds resources, other containers or both. The /user_id has the same substructure as the /me container.

/base_uri/posts/{post_id} and each post’s respective likes and comments resources are located at /base_uri/posts/likes_{post_id} and /base_uri/posts/comments_{post_id}. New posts can be created by sending HTTP Post requests to the posts endpoint, and specifying the linked data properties described in Section 3.4.

It is often necessary for applications to augment a user’s personal information with data available on Facebook. This information can be found in the fbprofile resource located at
Here a user’s personal information located on Facebook, such as their name and profile picture, can be found. A user’s Facebook relationships (their friends) can be found inside the friends resource located at /friends.

LDConnect currently supports HTTP Get, Post, and Delete on resources for which the current user has the correct permissions. Section 3.4 further describes the resources and containers described above. Additionally, this section provides more detail on how to create these resources, the properties to specify when creating these resources, and the properties developers can expect when fetching these resources.

### 3.4 Objects and Representation

The data returned from querying each LDConnect endpoint has different SoLiD compliant representations in LDConnect. The following sections describe these representations as they currently exist, but what is described is not fixed and will be expanded on in the future as needed.

#### 3.4.1 Main Containers

Containers in LDConnect include /albums/, /posts/, and /albums/[album_id]/. As shown in Figure 2, these containers have an RDF:type\(^2\) of SIOC:Container, and have four main properties: DCTERMS:created, DCTERMS:creator, DCTERMS:title, and DCTERMS:description\(^3\). A GET request on a container will return a description of the container (as noted in Figure 2), and a summary of the elements in it. When

---

\(^2\) The structure RDF:type maps an rdf namespace to a property or class. In this example RDF is the namespace and ‘type’ is the property.

\(^3\) Appendix 1 shows the full list of namespaces used in this project and their abbreviations.
creating a container, the RDF:type must be specified as SIOC:Container and the DCTERMS:title must be specified.

All containers, except for albums, are automatically created by the service. To create an album you can send an HTTP Post request to the albums container, and specify rdf data that notes that the RDF:type of the data is a SIOC:Container, and specifies the DCTERMS:title. Failure to specify those two properties will lead to the request being rejected. The DCTERMS:description property is optional, the DCTERMS:creator property is inferred from the user that is creating the album, and the DCTERMS:created field is created by Facebook when the data is posted using the Graph API. The response from LDConnect for an album creation includes headers that specify the location of the resource and the location of the likes and comments resources.

<table>
<thead>
<tr>
<th>RDF Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RDF:type</strong></td>
<td>SIOC:Container - the type of the data</td>
</tr>
<tr>
<td>DCTERMS:created</td>
<td>When the container was created</td>
</tr>
<tr>
<td>DCTERMS:creator</td>
<td>The person who created the post</td>
</tr>
<tr>
<td><strong>DCTERMS:title</strong></td>
<td>Name of the container</td>
</tr>
<tr>
<td>DCTERMS:description</td>
<td>A description of the container</td>
</tr>
<tr>
<td>LDP:contains</td>
<td>The resources and containers that can be found in this container.</td>
</tr>
</tbody>
</table>

**Figure 2**: Shows a RDF properties used to represent the posts, albums, and album containers. **--** refers to a property that is required on a creation
3.4.2 Posts

Facebook Posts, Links, and Stories, are all represented as SIOC:Post in LDConnect. All three are similar in definition and have many common properties. Figure 3 lists the RDF properties supported by LDConnect for a Facebook post. Stories posted on an individual’s Facebook feed holds a story field which is represented as a SIOC:note, and a link field which is represented as a SIOC:attachment. Links posted on an individual’s feed usually have a link field which I also represent as a SIOC:attachment. Both Links and Stories have an optional message field which is represented as a SIOC:content in LDConnect. A general Facebook post, such as a status update, usually has a message field which would also be represented as a SIOC:content. Because of how the three types of posts are related, in order to create a new post element, the developer must specify at least one of either the SIOC:note property, the SIOC:attachment property, or the SIOC:content property. Failure to specify one of these properties would lead to the request being rejected and no post being created. Like containers, the DCTERMS:creator property is inferred from the user making the post, and the DCTERMS:created property is defined by Facebook when the request is sent through the Graph API.

3.4.3 Photos

Facebook photos are represented as DCTYPE:Image in LDConnect. Figure 4 shows the attribute mapping between RDF properties and Facebook properties for a photo. The actual image is specified by the DCTERMS:source which holds the url of the image. The DCTERMS:source is required when uploading a photo. This means the image you are trying to upload would need to be stored somewhere on the web; ideally a user’s pod. In the future, it would be helpful to extend the application to accept multipart form data, and generate a
<table>
<thead>
<tr>
<th>RDF Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCTERMS:creator</td>
<td>The person who created the post</td>
</tr>
<tr>
<td>**SIOC:note</td>
<td>The Facebook story attached to the note</td>
</tr>
<tr>
<td>**SIOC:content</td>
<td>The message in the post</td>
</tr>
<tr>
<td>**SIOC:attachment</td>
<td>The link added to the post</td>
</tr>
</tbody>
</table>

**Figure 3:** Shows the RDF properties used to describe a Facebook Post, Link, or Story. ** - at least one must be specified.

<table>
<thead>
<tr>
<th>RDF Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>**RDF:type</td>
<td>DCTYPE:Image</td>
</tr>
<tr>
<td>DCTERMS:created</td>
<td>When the image was uploaded</td>
</tr>
<tr>
<td>DCTERMS:creator</td>
<td>The person uploading the image</td>
</tr>
<tr>
<td>**DCTERMS:source</td>
<td>The url of the actual image</td>
</tr>
</tbody>
</table>

**Figure 4:** Shows the rdf properties for a Facebook photo. ** - required when creating a photo

Facebook url for the photo. Like other LDConnect resources, the DCTERMS:created and DCTERMS:creator are not required on an image upload.

**3.4.4 Comments and Likes**

LDConnect supports commenting and liking of albums, photos, and posts. The Comments and Likes on each of these resources are stored in the same container as the resource itself. Comments and Likes are named `comments_{resource_id}` and `likes_{resource_id}`, where
{resource_id} refers to either the album id, photo id, or post id. The uri for comments and likes are returned in the link header of an HTTP response to a Get or Post request from an application.

A like on an object is represented as a triple that relates a resource uri to the hash uri #user_{user_id}, where user_id refers to the Facebook id of the user that liked the object. This relation is made using an LDConnect ontology, LDConnect:liked_by. The hash uri #user_{user_id} is of type SIOC:UserAccount, and following the uri will provide more information regarding the user such as their name. To like an object, all that is required is to send an HTTP Post to the likes endpoint of that object. There is no need to add a body to the request as all of the information needed, such as the person liking the object, can be inferred.

```
@prefix ldconnect, sioc, rdf, foaf
<{resource_uri}> ldconnect:liked_by <#user_user_id>
<#user_user_id> rdf:type sioc:UserAccount
   foaf:name “User Name”
```

**Figure 5**: Shows the structure of the likes resource. The resource uses the ldconnect, sioc, rdf, and foaf ontology which are described in Appendix A.

Comments on resources in LDConnect are each represented as SIOC:Post. Figure 5 lists the RDF properties expected for a comment. Comments can be created by making an HTTP Post request to the comments_{resource_id} endpoint for that particular resource. The request should specify that comment described by the data has SIOC:Post as its RDF:type, and if desired, can have the DCTERMS:references property as a back link to the post that the comment is about. Either a SIOC:content property or a SIOC:attachment property is however required else the
creation request will be rejected. The SIOC:content is the message in the comment, and the SIOC:attachment allows you to attach an image to the comment.

### 3.4.5 Friendship

The friends resource is similar to that of the likes resource. It uses the has_friends property from the LDConnect namespace. Figure 7 shows the structure of

<table>
<thead>
<tr>
<th>RDF Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RDF:type</strong></td>
<td>SIOC:Post. The rdf class for a comment</td>
</tr>
<tr>
<td>DCTERMS:created</td>
<td>The date the comment was created.</td>
</tr>
<tr>
<td>DCTERMS:creator</td>
<td>The person creating the comment</td>
</tr>
<tr>
<td>*SIOC:content</td>
<td>The message of the comment</td>
</tr>
<tr>
<td>*SIOC:attachment</td>
<td>The url of an image to attach to the comment</td>
</tr>
<tr>
<td>DCTERMS:references</td>
<td>The uri of the post being commented on.</td>
</tr>
</tbody>
</table>

**Figure 6:** Shows the rdf properties of a comment.  
** - required on a post else the post is rejected  
* - one must be specified else the post is rejected.

```
@prefix ldconnect, sioc, rdf, foaf
<resource_uri> ldconnect('has_friend') <#user_user_id>
<#user_user_id> rdf('type') sioc('UserAccount')
foaf('name') "User Name"
```

**Figure 7:** Shows the structure of the friends resource. The resource uses the ldconnect, sioc, rdf, and foaf ontology which are described in the appendix section.
each friendship in the resource. For each user, the *friends* resources shows a triple that maps the user to a hash uri of another Facebook user that they are friends with. Limitations in the Graph API makes it such that users can only view their list of friends that have permitted LDConnect to access their Facebook data. Limitations in the Graph API also restrict a user from adding a friend using the Graph API. However, LDConnect still allows users to add friends by using an automated script to perform the action of adding a friend. This option can only be performed by users that have specified that they want LDConnect to use alternate means to perform actions not supported by the Facebook Graph API.

### 3.5 Design Decisions

While designing LDConnect, several decisions were made that require clarification. These include what namespaces to use, how likes and comments should be represented, the language in which the service should be written, storing user information, and using automated scripts to perform actions not supported by the Facebook Graph API. In making these decisions, the main goal was to ensure that the service was very useful to Crosscloud developers, and that it was compatible with current and future decentralized social applications.

#### 3.5.1 RDF Ontologies

LDConnect uses popular rdf namespaces like SIOC (Semantically Interlinked Online Communities) Core Ontology, and the Dublin Core Metadata Initiative, along with its own namespace and a few other ontologies to describe the Facebook resources being accessed. This decision was made because I felt that reusing existing ontologies would make the program more compatible with other Crosscloud applications. It was also made with consideration for future
extensions of the program. Should the service be extended to support other external applications, it would be easier to extend the currently model, and have the data be compatible, than to create new namespaces for each new external application that we provide support for.

There is a Facebook Open Graph ontology [7] that I could have used, but the future of the project is currently unknown. As specified by the Open Graph documentation, the future of the project is dependent on its reception by the development community. Currently, the Open Graph API is not widely used, therefore it seems unwise to rely on its existence in the future.

3.5.2 Programming Language

Available resources was a main consideration when choosing the language in which the service should be written. Currently, there are many resources, such as rdflib.js⁴ and ldnode,⁵ that make it easier to work with linked data in node.js. In addition, there is a lot of effort being placed into developing these tools and others that support an all JavaScript development environment. In addition, rdflib.js has methods which simplify the creation and serialization of RDF graphs into other RDF formats. This ensures that the service can serve applications that work with different RDF formats.

One possible extension of this project would be to cache data rather than forwarding every request to Facebook. Doing this would improve performance, by reducing the number of requests to the Facebook API, and allow Crosscloud applications to share popular resources. Ldnode would make caching an easy feature to add. The combined advantage provided by

---

⁴ documentation on rdflib.js can be found at https://github.com/linkeddata/rdflib.js/
⁵ documentations on ldnode can be found at https://github.com/linkeddata/ldnode
ldnode, and rdflib.js, along with my familiarity of the JavaScript language made node.js the ideal choice for building this service.

### 3.5.3 Representation of Likes and Comments

Deciding on how comments and likes should be handled by the service was not trivial. One of the challenges with comments and likes is deciding where they should be stored. A popular approach to dealing with comments and likes in the decentralized environment that this project functions in would be to store the comments and likes on the personal space of the individual making the comment or liking the resource. Another approach would be to store the comment or like in the same space as the original resource. LDConnect focuses on the latter.

As described in Section 3.4.4, comments and likes are stored in the same container as the resource to which they refer. This makes it easier for an application to fetch resources and their comments and likes separately. This approach also makes it less difficult to paginate the results in the event that you have an object that has a large number of comments and likes on it. This paging approach [8] is preferred because it’s part of the LDP standard. Comments can be retrieved by sending a get request to the `comments_{resource_uri}`. In the event that there are more results than a set limit, the request will be redirected to the first page of the results, and next and previous pages of the result set is provided in the link header of the response. Liking an object involves sending a post request to the `likes` endpoint. It follows the same pagination rules as comments.
3.5.4 Leveraging Pod Access Control

LDConnect does not store any information about users. All user relevant information such as access tokens are stored on the user’s pod. This decision was made to be compliant with SoLiD conventions. In addition to complying with SoLiD conventions, this allows LDConnect to leverage the access control architecture used by the pods.

3.5.5 Scraping Scripts

Although the Facebook Graph API gives the ability to make many modifications in the social graph, it has its limitations. The API usually falls short on API calls that would allow applications to create certain elements or modify certain elements in the graph. For such situations, LDConnect tries to log into Facebook as the user and perform the necessary task. This feature was added due to the need to ensure that any data that the user can access from the Facebook website should be accessible in LDConnect. Friend requests are one feature that uses phantom.js\(^6\) to perform actions on Facebook. The Graph API does not allow you to send a friend request, but LDConnect will do so by using phantom.js. In order to use this feature, the user’s Facebook username and password must be stored on their pod. LDConnect will read the information from there then use it to perform the necessary action.

---

\(^6\) Phantomjs is a headless browser with a Javascript API. Documentation for Phantom JS is at http://phantomjs.org/documentation/
4 Performance Analysis

To analyze the performance of LDConnect, we compare the amount of time it takes to send an HTTP Request to the Facebook Graph API and receive a response, to the amount of time required to send a similar request through the LDConnect service. This allows us to measure the performance overhead of using LDConnect. The tests were done using jQuery\(^7\) to make ajax requests to the Facebook Graph API and the LDConnect service. The system time was recorded before the request was sent and when the response was received.

A resource read in LDConnect involves sending an HTTP Get request to LDConnect which then sends an http request to the Facebook Graph API, converts the response to linked

<table>
<thead>
<tr>
<th>OP</th>
<th>Facebook API (ms)</th>
<th>LDConnect (ms)</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>getImage</td>
<td>239.3</td>
<td>2260.3</td>
<td>9.4x</td>
</tr>
<tr>
<td>getPosts</td>
<td>766.3</td>
<td>2605.4</td>
<td>3.4x</td>
</tr>
<tr>
<td>getPost</td>
<td>360.4</td>
<td>2179.2</td>
<td>6.0x</td>
</tr>
<tr>
<td>getComments</td>
<td>309</td>
<td>2134.1</td>
<td>6.9x</td>
</tr>
<tr>
<td>getLikes</td>
<td>350.3</td>
<td>2174.8</td>
<td>6.2x</td>
</tr>
<tr>
<td>getAlbum</td>
<td>287.8</td>
<td>2238.2</td>
<td>7.8x</td>
</tr>
<tr>
<td>getAlbums</td>
<td>294.9</td>
<td>2165.4</td>
<td>7.3x</td>
</tr>
</tbody>
</table>

**Figure 8:** shows the round trip time for making requests to the Facebook Graph API and the LDConnect service.

---

\(^7\) jQuery is a javascript library that exposes convenient methods for performing ajax calls. [http://api.jquery.com/](http://api.jquery.com/)
data, then sends it back to the user. As shown in Figure 8, this leads to an average increase of 6.7 times the amount of time that it takes to send and receive responses to similar requests. This increase can be accounted to network delays, and the added overhead of the webid authentication. The average round trip time (rtt) for sending packets to graph.facebook.com from the testing machine was 10.7ms, and the average round trip time for sending packets to hel.crosscloud.org is 78.4ms. What these numbers show is that requests to hel.crosscloud.org are on average 7.3x slower than requests to graph.facebook.com. However, the rtt for sending a packet to graph.facebook.com from hel.crosscloud.org is 1.2ms; making that step in the workflow relatively negligible.

A write in LDConnect involves sending linked data to LDConnect which will in turn parse it for relevant data then send a request to the Facebook API. Figure 9 shows the performance reduction in sending requests to LDConnect rather than directly to Facebook. As shown, the decrease is 1.75x, but using alternate methods of sending linked data to the Facebook Graph will prove to be significantly slower.

It’s important to recognize that there are limitations to the approach used in comparing LDConnect and the Facebook Graph API. First, the accuracy of the time it takes to make a request is obscured by the amount of data that gets cached after each request. To reduce the effect of cached data, each run of the test was done in a fresh browser in private browsing mode. This made the data relatively consistent. Second, LDConnect is currently not optimized for production which makes it such that the time for writing and retrieving data is sub optimal.

In terms of using automation scripts to add friends, the average time to send a friend request was approximately 5 seconds. This is largely due to the overhead of executing the script
itself, and the setup overhead of launching a headless browser. Future optimizations in this area could prove helpful in migrating LDConnect towards using the less restrictive scraping method of accessing user data.

<table>
<thead>
<tr>
<th>OP</th>
<th>Facebook API</th>
<th>LDConnect</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>writePost</td>
<td>1458.3</td>
<td>2301.4</td>
<td>1.6x</td>
</tr>
<tr>
<td>writeComment</td>
<td>1320.4</td>
<td>2300.9</td>
<td>1.7x</td>
</tr>
<tr>
<td>uploadImage</td>
<td>1225.8</td>
<td>1989.2</td>
<td>1.6x</td>
</tr>
<tr>
<td>createAlbum</td>
<td>1001.7</td>
<td>2114.6</td>
<td>2.1x</td>
</tr>
</tbody>
</table>

**Figure 9:** Shows the performance reduction for write operations to LDConnect and the Facebook Graph API.

## 5 Future Work

LDConnect is an introductory project that offers an initial investigation into a much more complex and extensive field of research and development. There are many ways of extending the project. These extensions include adding support for other Facebook objects beyond Posts and Images, extending the service to cover other applications, adding a caching system for improved performance, and implementing access control methods that can better define how a user’s data should be treated. One of the main issues that face new social applications is this idea of a ghost town. If only a small number of individuals are using an application, they’ll often leave due to the fact that no one else is using it. Early adopters tend to abandon the application if their data and connections do not follow them to the new application. One way to address this
problem is by giving users that choose to begin using Crosscloud applications a way to still connect to their data on other applications. Currently, LDConnect focuses on data stored in the Facebook graph, but this feature could be extended across different non Crosscloud applications.

5.1 Extending Support for Other Facebook Elements

LDConnect only supports friendships, posts, and photos, but there several other objects and relationships in the Facebook graph, such as, pages, events, videos, groups, check-ins, etc. One way to extend the service would be to provide support for more of these objects and relationships. The structure of LDConnect would permit it to support these extensions relatively easily. The biggest challenge lies in determining how each of these elements should be represented in linked data. In addition to expanding the list of supported objects, we can also expand the number of scripts that scrape data from Facebook for actions not supported by the API.

5.2 Supporting Other Applications

Facebook is not the only application that users would benefit from being able to access through a linked data interface. LDConnect could be extended to retrieve data from other applications, or other services could be created that would retrieve user data from non Crosscloud applications and converted to linked data so that Crosscloud applications can leverage that data. This type of extension is a significant undertaking and its difficulty will be based on the resources available for the application that the service is attempting to connect to. Facebook’s API makes it easy to interact with Facebook data, but not all applications expose an API. Applications that do not provide methods or an API to access the data they store will be
much harder to create a Linked data interface for. However, the introduction of more scraping scripts will make it possible to still access the necessary data.

### 5.3 Caching

Currently, LDConnect does a live translation of data that is fetched from Facebook, and data that is written by Crosscloud applications. This means all requests require interacting with the Facebook API, which adds at least one more round trip cost to the request. It would be beneficial to cache some of the data rather than sending all requests to Facebook. In this system, reads on a resource could be performed on a local copy that gets refreshed periodically, and writes could be sent to the Facebook Graph immediately. Adding this caching to LDConnect would impact the freshness of the data, but would not corrupt the data since the program would not have to deal with simultaneous edits to a resource. Most operations in a social application are read operations, and therefore the cache would be very beneficial.

### 5.4 Access Control

Another way to extend LDConnect would be to add access control support. Facebook uses different access control rules from Crosscloud applications. LDConnect could do a translation/mapping of access control policies to ensure that these policies are upheld in all applications that use that data. This feature would mostly be beneficial for data being inserted into the Facebook graph. It could allow Crosscloud applications to write to an acl resource that would be translated when creating new objects in the Facebook Graph. The access control feature would also be necessary if LDConnect began caching data. Currently, access control
checks are done by Facebook when data is fetched, but if data is cached, there must be a way to check if the user trying to access the data has access to it or not.

## 6 Conclusion

LDConnect presents an interface that provides developers of decentralized applications with a way to allow their users to access their data in non-decentralized applications. This service leverages the Facebook Graph API to access a user’s Facebook data, and present it in a format that is compliant with the conventions promoted by SoLiD. This compliance makes the data automatically accessible to all decentralized applications that are cognizant of SoLiD conventions, with an added benefit that these applications will not have to be modified to access Facebook directly.

Creating LDConnect has shown that it is possible to build an interface that provides users a way to access their data stored in centralized social applications. It has also shown that the best way to access this data would be to create automation scripts that scraped the data from the website. Although this gives maximum control over what data can be accessed, it adds a performance overhead that cannot be ignored. However, a combination of scraping scripts, and leveraging open APIs improves the experience of using an intermediary to access data that is stored in centralized applications in a decentralized environment.
Acknowledgements

Thanks to Olivia Thayer for reviewing an earlier copy of this paper, and to members of the Crosscloud team for providing invaluable advice and assistance throughout the design and implementations of LDConnect.
References


Appendix A

RDF Namespaces

FOAF - http://xmlns.com/foaf/0.1/
RDF - http://www.w3.org/1999/02/22-rdf-syntax-ns#
DCTERMS - http://purl.org/dc/terms/
SIOC - http://rdfs.org/sioc/ns#
DCTYPE - http://purl.org/dc/dcmitype/
LDCONNECT - https://henchill.databox.me/fb#