Recommendation System for Diaspora Social Network

Nikhil.M Chandrappa
Syracuse University

1 INTRODUCTION

Social media is being used extensively in our daily lives resulting in peta bytes of user data in photos and posts. Traditional social media websites like facebook, Twitter have huge centralized backend systems supporting enormous amount of data being generated and one important thing to notice here is all the private information of the users are stored in those centralized servers which may be used for targeted marketing without any knowledge of the users. In this project, I am working on a social networking platform called Diaspora* which emphasizes on providing freedom and privacy to the end users.

Diaspora* is an open source distributed social networking platform with three key philosophies in decentralization, freedom and privacy. Instead of a single central server, diaspora supports a distributed network of small servers called pods. A user can select a pod near to their location/community or create their own pod and connect to existing network of pods thus providing greater ownership and privacy to the end users. In this project, I want to understand the communication architecture used in diaspora for distributed computing and also the design of the system in aggregating the feeds for the users. Also in doing so, I want to build an extension to diaspora network and I have decided to build a recommendation system as part of my term project.

Currently, Diaspora does not have a recommendation system to suggest friends which is actually the motivation for my project. I had previously worked on the development of recommendation system for facebook and twitter users and I want leverage the knowledge to develop a friend recommendation system for distributed social network.

2 BACKGROUND

Before we design the recommendation system, we have to understand the features and object model supported by the diaspora platform for retrieving the necessary data for the recommendation system.

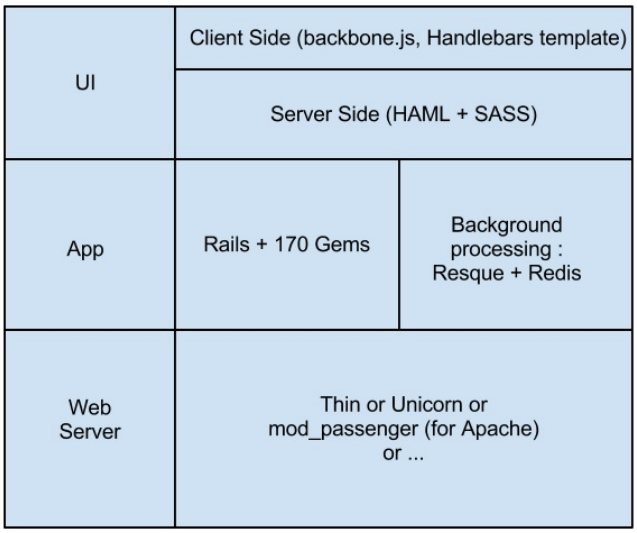


Figure 1: Diaspora Software Stack

Diaspora posts is built on the notion of asymmetric sharing, if you start sharing posts with someone, you have decided to send them posts however they may or may not choose to share there posts with you. Diaspora is driven by features, aspects and hashtags. On creating account, diaspora asks the users to select a list of hashtags which will be used for displaying related posts in the user feed. Features are the user interest based on the hashtags they have subscribed. Aspects are way of organizing the friends in different groups, based on the aspects user can restrict the visibility of posts. User features and hashtags can be used in generating the recommendation which will be discussed in later part of the report.

Diaspora is built using ruby on rails framework, its view templates are built using Haml templating language and css is written in Sass. Like any other rails project, di-

aspora follows the standard project structure and we are interested app/models in order to understand the object model of the diaspora. Diaspora persistance layer can be configured either using with Mysql db or a PostgreSQL db to store all the user information.

Diaspora consists of following objects models - User, Person, Profile, Contact, Request, Aspect and Post. User, person and profile are responsible for holding the personal details of the user. Contact contains the type of relation between the user and his friends. Request object is that one which will be sent to remote pod when a friend request is sent by the user. Post object contains the details of the posts created by the users. Diaspora communication is supported by a distributed federation protocol which is responsible for communicating the user posts and updates among the diaspora pods. Detailed explanation of the communication framework is presented in the section 3.

2.1 Recommendation System

Over the years there has been extensive research performed on recommendations systems[1][2][3]. Collaborative filtering is extensively used in Recommendation system, however many of the algorithm implementations assumes the candidates nodes reside on centralized database, there is no selection of the candidates over a distributed network. In case of diaspora social network which is distributed in nature and philosophy of securing personal data, traditional algorithms will not suffice. In order to identify the candidates, there has to be some mode of communication between the distributed systems. Therefore, I want to address this by developing a recommendation algorithm which is distributed in nature. Friend recommendation algorithm will be based on collaborative feature analysis and link prediction approach explained in [1].

3 Diaspora Architecture

High level architecture of Diaspora is depicted in the Figure 2, Diaspora network consists of standalone pod machines which are connected over the internet. Diaspora pods consists of two types of roles, users and pod admins. Users are the people who are registered in the pod and all the details regarding the users are stored in local database. Pod Admins, as the name suggests have administrative privileges required for troubleshooting the diaspora pods.

Every request goes through the rails routing mechanism, every URL is mapped to an action inside a controller. Lets consider a request to better understand the architecture of the diaspora for eg, When a user writes a new post from the Browser:

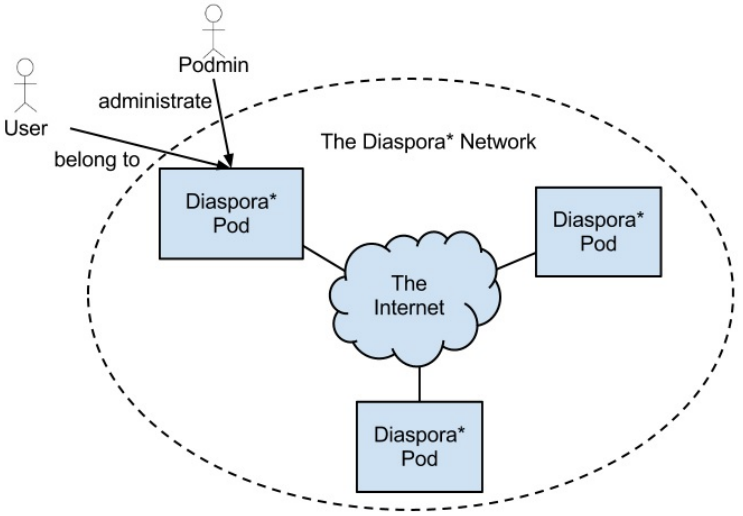


Figure 2: Diaspora Network

POST yourpod.org/status_messages invokes StatusMessagesController#create

first a local copy of the message will be saved in local database and corresponding table entries are updated then message is transmitted to other pods using the federation protocol. Diaspora communication framework is based on push model, Each and every post, comments and status update will be communicated to other pods over the internet. As diaspora performs large amount of network communications with other servers, it is essential for supporting background processing of the jobs. Diaspora uses Resque processes to perform the background jobs. As you understand the working of diaspora social network, it becomes obvious that diaspora is dogged by large amount of communication between the distributed pods which takes us to over next section, Federation Protocol which is backbone of diaspora social network.

3.1 Federation Protocol

Before we dwell in to the working of Federation protocol, we need to understand the events the that trigger the communication among the diaspora pods, following are the list of events as mentioned in diaspora specifications,

- during discovery of users on other system
- during sending information to users whom your are sharing with. Information includes,
  - Notification that you have begun sharing with them.
  - Posts that you have made.

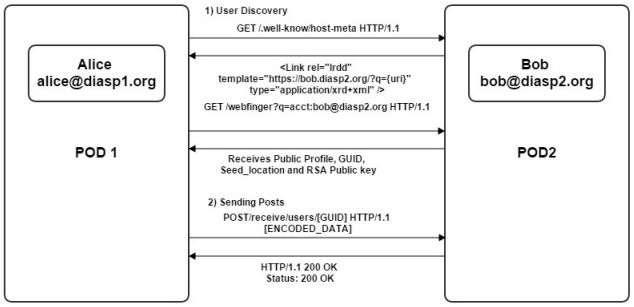


Figure 3: Communication between pods during user discovery, sending posts

- Comments that have been made (by you or others) on one of your posts.
- "Like"s that have been made (by you or others) on one of your posts.
- Conversations (each thread in the inbox has an object representing it)
- Messages (each individual message in a Conversation)
- Profile information
- Retractions of posts
- Retractions of likes/comments

Federation protocol consists of two phases, user discovery and sending & receiving posts.

3.1.1 User Discovery

When user is first registers on the pods, he will not have any contacts in his aspects so user discovery phase is used to search users on other pods based on the diaspora handle. Diaspora uses webfinger servers for users discovery. Webfinger servers can be hosted as part of the diaspora pod or can be hosted as standalone server. User discovery consists of following steps,

Step 1: In our example consider Alice with webfinger address alice@alice.diasp1.org wants to discover bob@bob.diasp2.org, alice has to perform a Webfinger lookup of bobs address. Alice's pod first receives host meta file from bob's web finger address which contains the url of the pod on which bob's reside.

```
<Linkrel="lrdd"template="https://bob@bob.
diasp2.org/?q={uri}"type="application/xrd+
xml"/>
```

Step 2: Alice's pod replaces the uri with bob webfinger address and it makes a GET request to get bobs webfinger profile.
Step 3: Bob responds with Alice webfinger profile which contains host of information including bob's public profile information and important of all its contains GUID and RSA public key of bob which will be used in all the future communications.

```
<Linkrel="http://joindiaspora.com/
guid"type="text/html"href="((guid))"/>

<Linkrel="diaspora-public-key"type=
"RSA"href="((base64-encodedrsapublickey)
)"/>
```

Step 4: If Alice decides to add bob in her aspect, all the information including GUID and RSA public key of bob will be stored in corresponding tables, Contacts and Profile.

Now Pod on which Alice resides knows the existence of bob and Alice decides to share with bob which enables the communication between two pods. Diaspora pods make use of salmon envelop message for communication.

3.1.2 Sending Posts

Communication of posts and other messages are based on the following steps,
Step 1: Construction the message
In Diaspora, all the message communication over the network are encrypted. Even if pod uses http to transmit the messages, there is a guarantee of privacy over tranmission on network. construction of message consists of three stages

- Construct the encryption header
- prepare the payload message
- construct a Diaspora salmon magic-envelope.

Encrypted header is constructed using two AES-256-cbc ciphers. First AES key referred to as "inner" key is placed in the header along with the initialization vector (IV) and second key called the "outer" key is used to encrypted the plain text header. Encrypted header and outer key are placed in JSON object and is encrypted using public key bob which was retrieved in user discovery phase. Hence there is three layer of encryption for header which provides guarantee over transmission on network.

Payload messages are encrypted using inner key which was generated in previous step. Now we two encrypted blocks, 1. encrypted header 2. encrypted payload message.

Final Step is to generate salmon magic-envelope which contains encrypted message and payload, also this message will be signed using RSA-SHA256 algorithm and stored in the salmon magic envelope for verification at bobs end.

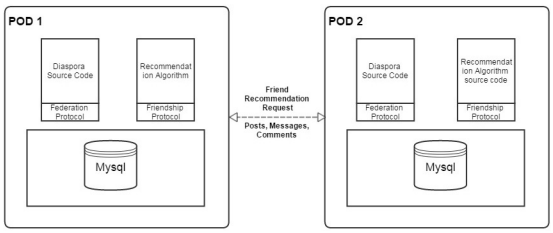


Figure 4: High level Architecture of Diaspora pod with newly developed Friend recommendation system

step 2: Construct the URL of bob's salmon endpoint Based on the pod url on which bob resides that was retrieved in user discovery phase will be used to generated salmon endpoint url.

Step 3: Post the message to bob If Alice's pod receive an HTTP 202 Created, or a 200 OK, your salmon slap has been accepted. If message delivery was not successful, Alice's pod try to push the request 3 more times after which it decides the remote host is not actively accepting requests. Apart from excessive communication, we can also note that there is not guarantee that all the updates will delivered to remote pods.

Before concluding this section, we have one important observation on diaspora social network. If we take away the distributed federation protocol there is no way for the users to get posts from other pods and it just behaves as any other centralized social network application. hence federation protocol is the backbone of diaspora which provides the distributed aspect to it.

4 PROBLEM STATEMENT

As part of the term project, I will install the development version of diaspora* platform on my local machine. I will analyze implementation of distributed communication framework and aggregation of posts. And also I will develop a distributed friend recommendation prototype system for diaspora social network.

5 Friend Recommendation System

As part of final project, I have designed a Friend recommendation system which consists two modules, Recommendation Algorithm module and Friend Recommendation Protocol. Recommendation protocol handles the communication of Friend Request messages between the diaspora pods. High level architecture of the system is as show in the Figure 4.

Both the diaspora source code and recommendation algorithm source code get the required model data from underlying backend supporting the pod which can be either MySQL DB or a Postgres DB. Friend recommendation system is multi-threaded concurrent application. Recommendation System consists of three stages, 1. construction of Friendship graph 2. Identifying candidate and sending recommendation requests 3. Results calculation.

5.1 Construct Friendship Graph

In order to properly model a friendship network we have to develop a friendship graph with consists of a connected graph where edge is present between two nodes if they are friends. Diaspora has a unique behavior where contacts are organized in to different aspects which are nothing but groups like Friends, Family, Acquaintance.... However it doesn't affect the structure of our friendship graph, as aspects are client side features.

$$G = (V,E)$$

v = set of friends, E = relationship between them.

For building graph we make use of data from Users, contacts, profiles and people table to retrieve user GUID's, Friends GUID, Friends POD URL and Diaspora Handle. When Recommendation Algorithm is run for the first time, we iterate through all the local users GUID's and retrieve the contacts of each user to build a sparse interconnected graph. It is possible to configure this stage on server start up which will considerably reduce the process time of displaying the recommendation results. Each edge will be have information about Friends GUID, Pod URL and Diaspora Handle.

5.2 Identifying candidate and sending recommendation requests

As per[1], there are several alternatives to generate the candidate edges (u,v)

1. Uniform at random
2. From a query distribution modeled on the frequency of vertices at a given graph distance
3. Exhaustively within a specified radius

Since we have a friendship graph its efficient to retrieve the candidates by traversing the graph in a breadth first manner. We can retrieve friends of friends and so on based on the radius of the graph. Also it is possible to sample the users not in the connected graph by random sampling on the same diaspora pod based on the interests. This will be implemented in subsequent iteration of the recommendation system.

In recommendation system there are two kinds of message, Friend Request Message and Result Message,

friend request message contains the details of the requester, remote user and data required for feature analysis. Response message will contain key value pair of Recommendation candidate GUID and weighted score based on feature analysis.

In diaspora, friends of users can reside in same pod or in a remote pod, its is possible to obtain the behavior of people residing in the same pod however in order to perform collaborative feature analysis on remote users we either have to retrieve details of remote users or send a recommendation request message to perform collaborative feature analysis and return the scores. Former method is a breach to ideology of diaspora, of not sharing personal information hence we employ a protocol of sending the requests where the recommendation algorithm originates

Identifying the candidates works in two phases, first we traverse the friendship graph with dfs having a default radius of 1. We build a list with local and remote users. Collaborative feature analysis of the local Users are performed by querying the database. For remote user, we build a friend recommendation request which contains information on user account that initiated the recommendation request, information will include user GUID, user tags, user location and GUID of the remote users. Details of the implementation is documented in section 6, please refer to the section for better understanding

5.2.1 Data Collection

To perform the collaborative feature analysis, we need to identify set of features which are needed for evaluating the likelihood of sending a friend request by a user, in other words the mutual interests between the candidates. I categorized the data collection in to

- HashTags Subscribed to
- HashTags mined from the user activity
- location of the user

All the data required for performing collaborative analysis is stored in following tables - Users, Contacts, Profiles, People, tags and tag_followings

5.3 Results Calculation

When a pod receives a recommendation request, it reads the request to determine the users on its pod and also stores the tags, location of the user which initiated the recommendation which will be used for calculating the collaborative feature score. Each feature will be weighted based on the Rank order centroid which ranks the features and determines the weights. Weighted score for tags is 0.7 and for location is 0.3.

Now algorithm will determine the candidates by performing a bfs search on friendship graph of this pod and start node of bfs search will be the requester friend who is from this pod. Now algorithm matches the tags of the requester with candidates and calculates the score of based on the following formula,

$$score = (no.of matching tags)(0.7) + (location match)(0.3)$$

Once the algorithm calculates the scores of all the users. It sends a response to requester with details on recommendation GUID, score, which will be collected by the requester and sort the results to display recommendation in increasing order of likelihood of being a friend.

5.4 Weighted Score

In order the rank the predictions, it is essential to provide weights to different features and data collected from mining the profile. A trivial approach can be to give a certain value to each of the different attributes however to obtain more accurate results there has to be scientific approach to select weights. After going through literature, I finalized on using Rank order Centroid (RoC)[5] to give weights

$$W_i = (1/M) \sum_{n=i}^{M} \frac{1}{n}$$

where M is the number of items and W_i is the weight for i^{th} item.

Roc is based on providing ranks to different attributes under consideration. Following ranks are assigned in recommendation system,

Feature	rank
similarity in interest	1
Location	2

6 Implementation Details

Recommendation system is a multi-threaded concurrent application that consists of four modules running on their own thread which are Recommendation Algorithm client, Request Processor, receiver and Sender. Module diagram of the recommendation system is shown in figure 5.

In real world diaspora pod, recommendation system will be invoked when a user logs into diaspora page and navigates to aspect section. However in our simulation environment, we assume that user has successfully logged and we have corresponding GUID. For the simulation, I have configured a test user logged in and we have his GUID. Lets look at the flow of execution when a recommendation system is started,

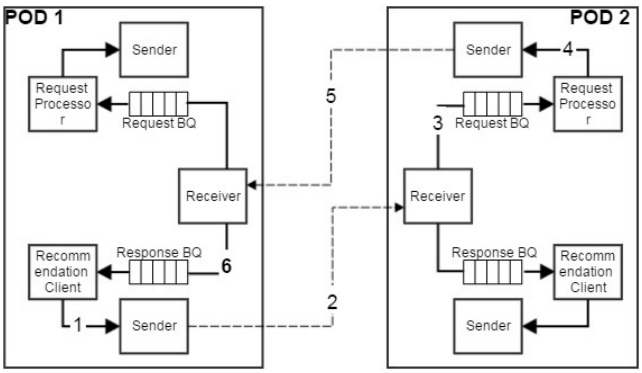


Figure 5: Recommendation System Module Diagram

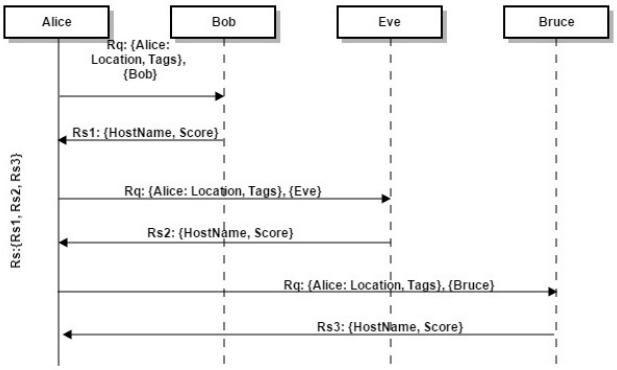


Figure 6: shows the interaction of the recommendation algorithm, messages that gets communicated between the diaspora pods

On system start up all the modules are invoked. Request processor and Receiver blocks waiting to read messages from the blocking queue.

Step 1: Recommendation Client at POD 1 starts the process by building the friendship graph as explained in the previous section and identifies the list of candidates by traversing the friendship graph. It then iterates through the list, to generate a set of candidates who are present in remote pods. It then sends the Friend recommendation requests and blocks, waiting for the results.

Step 2: Receiver at POD 2, receives the message and determines whether its a request message or a result message.

Step 3: As its a request message, it enqueues the message in to Blocking queue which it shares with request processor.

step 4: Request process reads the request message and retrieves the GUID of local users. It determines the candidates by walking the friendship graph and calculates the weighted score. It generates a key value pair of candidate GUID and score and sends the result back to requester.

step 5: Receiver at Pod 1 receives the message, as its a result message it will enqueue the message to blocking queue it shares with recommendation client.

Step 6: Recommendation client which was waiting for the result will read the message and aggregate the results. Once all the responses are received, it sorts the results list based on the score and displays the results in increasing order of candidate likely hood.

7 Experimental Setup and Results

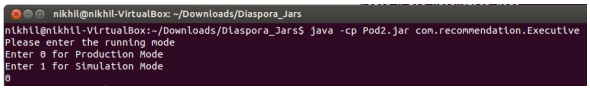
In order to test the distributed algorithm, I need two diaspora pod installations. However, it is not feasible to create production pods and connect them to diaspora network. Hence I have created two implementations of algorithm, one version which queries the diaspora development backend to perform collaborative feature analysis and other version which queries the test data simulating the diaspora back end. I have installed a diaspora machine on my laptop which is running on mysql backend.

In simulation environment, Pod 1 will start the recommendation algorithm and pod 2 will process the request and send the scores. Working of recommendation system is present in Figure 7

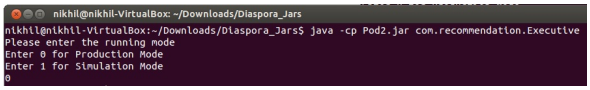
8 Limitations

As the recommendation system relies on the communication of the messages between diaspora pods to perform collaborative analysis, it is essential for algorithm to receive friend request and result messages. Any loss of message will result in a inconsistent recommendation algorithm. Based on the prototype implementation of the algorithm, I was able to gain insight into critical issues that may plague the production system if current design of the distributed recommendation system is implemented on diaspora network,

- Performance of recommendation algorithm will be hampered due to excessive communication
- Scenario where remote friends in turn have friends on remote pods will cause original requester to endlessly wait for response
- Current implementation of recommendation system uses collaborative feature analysis and link prediction. Its can be improved by implementing the link prediction based collaborative filtering.

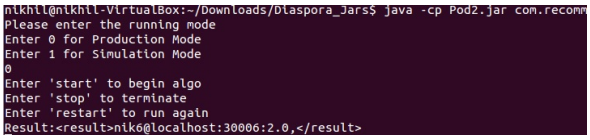


(a) Pod 2 - Start

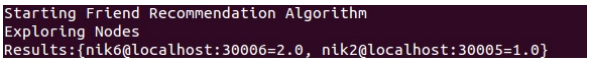


(b) Pod 1 - Start

(c) Pod 1 - Begin Recommendation System



(d) Pod 2 - Process request and send scores



(e) Pod 1 - Aggregate and display sorted results

Figure 7: Working of Recommendation System

- random sampling of the users of the same pod should be considered to diversify the user aspects group

Apart from the above, the broader issue affecting the diaspora network like security threat due to presence of malicious pod, also affects our recommendation system. Among all the issues faced by the diaspora network, there are is serious one in compromising privacy of the user personal information which is affecting one of the philosophies of diaspora.

As explained during communication of the posts to users on different pods, all the status updates and posts of the users gets transmitted to remote pods and copy of them will be stored on the remote pod database, unknowingly user data is being stored in pods other than pod on which he is registered. Trouble comes when user deletes the profile, when user deletes his profile a delete account message is sent to all the pods however there is not guarantee that messages were successfully delivered which might result in copy of user personal information on a remote pod even if user has excited from the diaspora network. It is one of scenario's among many other violations of user privacy reported by diaspora community.

9 Conclusion

As part of this project I was able to understand the architecture of diaspora social network, implementation of communication framework based on federation protocols. It gave me an insight in to concepts of distributed computing in diaspora network and understand the major limitation that are dogging the diaspora network. By developing a recommendation system prototype, I was able to understand the intricacies of developing a distributed applications, gave me an insight to design of a distributed applications and how important are communication framework for success of distributed applications. Also, it helped me understand how a generic algorithm can be retrofitted to distributed applications.

10 Contribution

Name	Value
Number of Codelines	approx. 2300
Scale of Performance Study	N/A
External Review	N/A

Before I could start with recommendation algorithm, I had to understand the object models and architecture of Diaspora system. Diaspora source code is poorly documented, had to analyze the source code and development database to determine the extraction of the data required for collaborative feature analysis.

References

[1] W. H. Hsu, A. King, M. S. R. Paradesi, T. Pydimarri, T. Weninger. Collaborative and Structural Recommendation of Friends using Weblog-based Social Network Analysis. In Proceedings of the 2006 AAAI Spring Symposium on Computatational Approaches to Analyzing Weblogs (CAAW 2006).

[2] Michael Moricz , Yerbolat Dosbayev , Mikhail Berlyant, PYMK: friend recommendation at myspace, Proceedings of the 2010 international conference on Management of data, June 06-10, 2010, Indianapolis, Indiana, USA

[3] http://cs229.stanford.edu/

[4] Mahmuda Rahman, Qinyun Zhu, Edmund Szu-Li Yu. TRECT: A Hashtag Recommendation System for Twitter. SeRSy Workshop (2013)

[5] http://www.tcrponline.org/PDFDocuments/tcrp_rpt_131AppF.pdf

[6] https://wiki.diasporafoundation.org/Federation_protocol_overview

[7] Figure 1 and Figure 2 - Diaspora community page