Implementing Federated Social Networking: Report from the Trenches

Gabriel Silva
Faculty Gama (FGA)
University of Brasília
Gama, Brazil
gabrielssilva.sw@gmail.com

Larissa Reis
Colivre
Salvador, Brazil
larissa@colivre.coop.br

Antonio Terceiro
Colivre
Salvador, Brazil
terceiro@colivre.coop.br

Paulo Meirelles
Faculty Gama (FGA)
University of Brasília
Gama, Brazil
paulormm@unb.br

Fabio Kon
FLOSS Competence Center
University of São Paulo
São Paulo, Brazil
kon@ime.usp.br

ABSTRACT

The federation of social networks aims at integrating users by means of a decentralized structure, enabling the interoperability among multiple social networks in a transparent way. Despite a few isolated initiatives in federating open social networks, there is no adoption of any standard, which hinders the emergence of new, effective federated systems. To understand the difficulties in the development and standardization of federated services, we have conducted research on existing specifications and implementations of interoperability among social networks. We have developed a federation proof of concept within the Noosfero platform, implementing a subset of the Diaspora protocol to federate users and public content, in addition to complementary specifications, such as Salmon and WebFinger. In this work, we introduce our results to federate Noosfero with Diaspora networks, pointing the required steps before further development. We aim to implement the Diaspora protocol within Noosfero, finishing its specification and improving its documentation, encouraging more projects to adopt this protocol.

ACM Classification Keywords

C.2.6 Internetworking: Standards

Author Keywords

Federation; Social Networks; Noosfero; Diaspora; Free Libre Open Source Software.

OpenSym '17, August 23–25, 2017, Galway, Ireland

ISBN 978-1-4503-5187-4/17/08…$15.00

DOI: https://doi.org/10.1145/3125433.3125455

INTRODUCTION

Social networks, or social media, can be defined as platforms that allow individuals to connect to others, share personal information, and provide content [5]. The information that flows in these networks is not always public, but usually takes place by means of private infrastructure that belongs to service providers.

In the context of computer security, privacy can be defined as someone's capacity of controlling which information related to them can be consumed and stored, and with whom it can be shared [17]. In addition to guaranteeing their users' privacy, social networks must ensure the confidentiality of the information they host, i.e. making sure that users' private information is not exposed to unauthorized individuals.

It can be argued that a central provider having all the control over the flow of private information would put user privacy in jeopardy, for a few reasons. First, the central provider is a single point of failure, and a security breach in its infrastructure exposes every user of the service. Second, users are left with no option other than blindly trusting the service provider to not misuse their personal information. In the case of a commercial entity, that implies trusting the company *but also any other company that may acquire it in the future* to not secretly violate its publicly available privacy policy, and to not revise the privacy policy itself to include new terms that are not favorable to its users.

The concept of decentralized networks is an alternative to the centralization of private data flow. Decentralization is even one of the original characteristics of the Internet, and a common organization pattern in communication networks. Communication networks can be centralized or distributed [2]. Centralized networks rely on a central node to mediate communication between all the other nodes. Decentralized networks have multiple mediating nodes, while, in fully distributed networks, nodes can communicate directly in any pattern that is possible and/or necessary. As illustrated by Baran [2] in Figure 1.

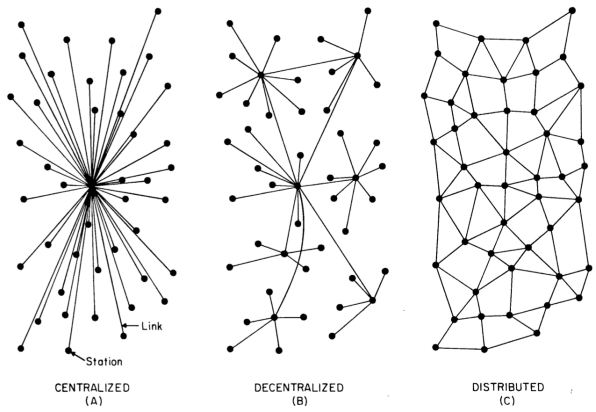

Figure 1. Conceptual models for communication networks (Source [1]).

Centralized social networks, even if provided by a decentralized infrastructure with the goal to improve performance and efficiency, conceptually still follow a centralized network model. The central node represents the service provider while the rest represents the millions of users. Adopting a decentralized model means dividing users among connected, intermediary providers that offer the same service, guaranteeing interoperability, so that a user that is served by provider "A" can still interact with a user that is served by provider "B".

A set of interconnected servers that seamlessly provide a service is defined by some authors as a federated network [15], which is similar to the definition of decentralized networks shown in [2], but that is also defined as a set of interoperable implementations that follow a client-server model [3]. This definition is important because it generalizes the concept of federation to other types of communication systems that are not computer networks and indicates the property of extension independence — any entity that guarantees interoperability can be part of the federation without the need of previous coordination with its existing members.

Service federation contradicts the very existence of a single provider, common in centralized social networks like Facebook and Twitter. Decentralization removes information storage restrictions to a single provider. More importantly, extension independence allows new providers to independently appear in the federation as long as they respect the interoperability criteria, what encourages autonomy. The decentralization of private information distributes the responsibility for maintaining confidentiality, which is no longer dependent on a single entity with arguably concealed intentions.

For a federated network to work, there is a need of interoperability among systems. The probable scenario includes communication among vastly distinct systems, in which protocols and standards are essential for successful communication. This requirement tends to increase the complexity of the applications, as it introduces problems such as data replication and consistency management.

In this work, we study federation in the context of social networks, investigate which aspects are involved in the interoperability of this type of media, and report on the state of the art of the standardization and interoperability. We then present a case study of implementation of federation features in Noosfero, a free software [1] platform for social networking, discussing the design and implementation of this proof of concept.

The remainder of this work is organized as follows. Section II lists some federation alternatives that obtained popularity among open social network projects, describing the basic flow and used technologies. Section III enumerates a number of related works on the development of decentralized social networks and federation infrastructures. Section IV presents the research design and defines our case of study. Sections V and VI report the implementation of a proof of concept and the results we obtained, and Section VII concludes the paper highlighting its main contributions and pointing paths to future works.

BACKGROUND

The absence of a standard protocol for the federation of social networks hinders the development of interoperable applications, as it leads to the adoption of divergent technologies. This segmentation affects the network effect and does not help on the emergence of a *de facto* standard.

To discuss the standardization of federated systems, it is interesting not only to analyze the community effort to discuss and adopt specifications, but also to identify the reasons that make it difficult to reach a consensus. The subject is incipient in the literature, so we had to resort to reviewing existing community discussion in the relevant mailing lists. Given the nature of the World Wide Web Consortium (W3C) as the standards body of the Internet, we used the archives of the W3C's Federated Social Web Community Group[2].

In the next subsections, we present a documentation of the existing initiatives and attempts to establish protocols for the federation of social networks, describing the used technologies and the state of practice in different projects.

OStatus

OStatus[3] is a protocol suite to enable real-time interaction between social networks. It was proposed by Evan Prodromou for the implementation of StatusNet, a federated microblogging network that later originated the GNU Social[4] project.

The project obtained visibility in the community, and had its W3C community group[5] created in 2012, one of the few formal attempts to propose a federation protocol. However, the group did not produce any results so far. The OStatus specification received a fair amount of criticism over its limitations, and did not advance into a standard accepted as good enough for all participating projects.

[1] Free/Libre/Open Source Software (FLOSS)
[2] http://w3.org/community/fedsocweb
[3] http://w3.org/community/ostatus
[4] http://gnu.io/social
[5] http://w3.org/community/ostatus/

OStatus is defined as a combination of protocols, each covering a part of the interactions that make federation possible. It incorporates standards that already existed when it was proposed, such as Atom/RSS, used for syndicating content from different sources. OStatus proposes to arrange these standards and protocols in a suite for real-time interactions between servers, automating the flow of social interactions among users.

The specialized protocols that are included in the OStatus specification are described below.

PubSubHubbub

PubSubHubbub specifies a distributed system for publishing and subscribing [10]. It makes use of central hubs used by services to publish new contents and updates, and by other services to subscribe, receiving real-time notifications on new activities. It is an extension to technologies such as Atom/RSS, that depend on users manually fetching for updates.

WebFinger

WebFinger is a protocol used for discovering information about entities based on standard identifiers [13]. It aims to solve the problem of sharing identities among servers.

The specification requires every resource to be identified by a URL. WebFinger servers should respond to HTTP requests, providing information in JSON or XML formats. The only information required for other servers to retrieve information about any entity is its resource URL, which can be generated with public information using a LRDD process [8].

ActivityStreams

The ActivityStreams specification defines a format to represent activity in social networks, such as "Bob started following Alice", "Alice posted a new message", etc. It aims to standardize entity formats, making it easier to share and consume those objects from different servers [1]. The latest specification proposes the use of JSON objects with a set of predefined attributes, including the action type, involved users, and content data.

Salmon

Combining solutions like Atom and PubSubHubbub allow publishing and updating contents across servers in real-time. However, to allow a content to be updated – for example, adding a comment to it – from any of those servers, it is also important to make sure the content state will always be the same in the entire network.

Salmon is a message exchange protocol that provides a way to change and track the state of contents across servers [14]. It proposes a standard process to securely exchange information among servers, merging all modifications in a stream of messages.

OStatus Current State

The OStatus project was an important step towards the standardization of a federation protocol, what is clear given the creation of a W3C work group. However, the specification was not complete enough to fulfill the requirements of most projects, mainly due to its limitation to public content and the lack of mechanisms to handle privacy.

In 2012, Evan Prodromou announced the development of *pump.io*, another approach for federating social networks, with more modern technologies. It contributed to the weakening of the OStatus community, which is not currently active. The StatusNet project is also no longer active.

Beyond there being no active community to maintain or develop the project, most of the technologies used in OStatus suite evolved. OStatus still specifies the use of technologies considered by many as legacy, such using Atom (instead of JSON) for ActivityStreams messages.

Despite using technologies said to be outdated, OStatus is still used on the development of federated social networks, such as GNU Social[6], which describes itself as a continuation of StatusNet, and the Mastodon project[7], which promises compatibility with GNU Social.

Diaspora

The Diaspora project[8] proposed the implementation of decentralized social networks in response to concerns raised on privacy and freedom in centralized social platforms. Its first version was released on September, 2010 as a result of a crowdfunding campaign. By August, 2012, a community of users started to maintain it.

Diaspora's main selling point is avoiding content centralization, with a lack of central control over user data, by building a network of personal servers, called pods. Each Diaspora pod stores only the data of its own users, and allows them to interact with users on other servers through a federated network.

The Diaspora federation protocol was created to converge with OStatus as soon as the latter supported private contents, what did not happen so far. Most of the protocol specification relies on the concept of remote users, and on an exchange mechanism.

Remote Users

A fundamental concept to implement federated networks using the Diaspora protocol is taking into consideration the existence of remote users. Most applications only recognize local users, that is, those who have their credentials and profiles saved in the local database. However, to enable interaction with users from other networks, at some point it is necessary to handle users that do not exist locally.

Diaspora classifies its users in two categories: local and remote. While local users follow the classic implementation, remote users only interact with the application through federation mechanisms. Having two category of users may affect the project architecture, and this concern should ideally be taken into consideration on early design stages.

---

[6] http://gnu.org/s/social/

[7] https://github.com/Gargron/mastodon

[8] http://diasporafoundation.org

Message Exchange

In the Diaspora protocol, messages are exchanged using a subset of the Salmon protocol. Essentially, the Diaspora specification describes how the message should be built, encrypted, and sent to the Salmon endpoint of the destination pod.

The payload of a Salmon message is an object that represents an activity in a pod. Those objects are called entities and can represent content publications, private messages, comments, retractions, or even notifications of follow and unfollow actions.

Every message is sent using HTTP to certain endpoints in the destination server. The endpoint depends on the nature of the entity. For example, private messages are sent to user-specific salmon endpoints, while public contents are sent to public endpoints.

Protocol Flow

Besides using a subset of the Salmon protocol to exchange messages, Diaspora also uses WebFinger and hCard standards to discover identities and fetch public profiles from remote servers. It is also possible to use ActivityStreams and PubSubHubbub to provide public feeds.

The basic flow is to discover users, fetch their public information, then create and send Salmon messages according to the desired interaction. Diaspora pods will also send Salmon messages to subscribed servers, notifying about publications and updates.

Diaspora Current State

Both the project and the protocol specification remain in active development. The latest specification dates from July, 2016. There is also a protocol implementation written as a Ruby library, used in the main Diaspora project, that includes most of the message exchange mechanism. That library can be reused by other projects that want to implement federation with the Diaspora network.

The visibility gained by the Diaspora project has provided it with a significant following, including adoption of its specifications by other projects such as Friendica[9] and Hubzilla [10], which support some level of integration with Diaspora pods. That visibility was not enough, though, to trigger a formal standardization process.

RELATED WORK

Decentralized social networks are well established in early works. Au Yeung et al. [11] describes decentralized models for social networks and proposes the use of technologies such as linked data. Chao et al. [7] further explores this method describing seamless interaction and single point of access to the network, important concepts for federated systems.

In addition to reference models, several works propose implementations for decentralized infrastructures. Buchegger et al. [6] propose an infrastructure for P2P networks, exploring issues related to decentralized social networks, such as exchanging messages over a decentralized architecture and encryption. They also provide insights on asynchronous message exchange and the use of a centralized storage mechanism to avoid content replication.

There are other works that propose implementations of decentralized networks. Boelmann et al. [16] explore privacy and security issues. Aberer et al. [12] work on content replication, exploring the impacts of availability and performance on the propagation of messages.

Bielenberg et al. [4] explore the growth of Diaspora networks, but more importantly, they give an overview of privacy in the given platform. The authors show that, at the time, users preferred to join popular and reliable servers, indicating that they do not host their own data, even though this is the proposal of decentralized networks.

Finally, Fan et al. [9] propose a different approach to develop a decentralized social network, presenting a proof of concept with a large network. The proposed approach is to, instead of building a decentralized network, decentralize ordinary social networks by providing a layer to relay contents from one network to another.

Even though there are some implementation proposals, we choose to explore the approach of the Diaspora project. The discussion regarding the implementation of federation protocols in existing projects is still incipient, and constitutes a motivation to proceed with the proof of concept we propose.

Our goal is not to present another federation infrastructure, but to investigate the implementation of existent protocols in projects that were not designed to be federated, and differently from the related works, to report our experiences implementing the said proof of concept.

RESEARCH DESIGN AND STRATEGIES

Given the non-existence of standards for federated social networks, we define the following research questions to guide this work.

RQ1: *What are the protocols in use for federation of social networks and what is the status of their development and adoption?* During the development of this research, we explored the protocols proposed and being used to approach the implementation of federated social networks, reporting the current status of the alternatives and projects that use it.

RQ2: *What engineering problems need to be solved in order to add support for federation in a platform that was not conceived with federation as a requirement?* There is virtually no literature on the implementation of federation support on existent projects. Most of the previous work describe the design and implementation of federated systems, without assuming a previous, non-federated system. We wanted to investigate which aspects of a project can affect the implementation of federation support.

Design Alternatives

There are currently two possible strategies for implementing federation. Some of the discussion on this topic is recorded

[9] http://friendi.ca/

[10] https://project.hubzilla.org

in W3C mailing lists[11] and we have organized them to be presented in this subsection.

We could have the development of a system or protocol that should be supported by all applications interested in joining the federation, a method characterized by a sole entity acting as the common denominator among all networks, or alternatively by a massive coordination effort among loosely-coupled, independent projects working on the topic.

The alternative is every application explicitly implementing the protocol of every other application it wants to connect with. This strategy does not depend on a standard protocol, or on a central authority to define what needs to be implemented and how. It can be called the "polyglot strategy".

The polyglot strategy brings some restrictions to the federation of systems, since it depends on every application responding to the protocol of every other application in the network. This strategy seems to support the segmentation of specifications as opposed to what would happen when using a single protocol as the common denominator. On the other hand, the adoption of a common denominator depends on a protocol capable of covering the peculiarities of all possible networks, including types of relationships among users, content sharing mechanisms, and privacy policies.

The criticism over the lack of a notion of privacy in OStatus[12] is an example of the barriers caused by these differences. The project only supports public content, so networks with more complex privacy definitions are not completely covered. In the meantime, other projects were created to cover these flaws, such as Diaspora and Friendica.

A specification that meets the requirements of all possible social networks would require a very large design and development effort. An alternative would be a protocol that establishes a subset of policies as a base to the implementation of more complex or specific features. It can be argued that, eventually, these policies would have to support incompatible concepts, given the difficulty of finding ways to satisfy all systems.

Even if such an agreement could be found, every application would have to solve its own needs based on the restrictions of the common denominator. It would possibly require projects to be heavily modified to adopt such a protocol, which does not contribute to the viability of that strategy.

The tendency to adopt a polyglot strategy indicates that the difficulty in finding a universal common denominator surpasses the interoperability benefits. On the other hand, if projects keep implementing integration with individual projects we could eventually find some kind of convergence, leading to a network effect, and identifying a common denominator for a subset of applications.

[11] In the following thread from the federated social networks group http://lists.w3.org/Archives/Public/public-fedsocweb/2013May/0058.html

[12] The discussion regarding OStatus privacy can also be found in W3C mailing lists, http://lists.w3.org/Archives/Public/public-fedsocweb/2013May/0061.html.

For now, federation can only be achieved by a polyglot strategy, adopting the specifications of individual projects, which are usually a suite of well established protocols. A good start is to choose a project considering its community activity and the adherence of its standard with your particular needs.

Case study: Noosfero

Noosfero[13] is our free software platform that can be used to build social and collaboration networks, providing a platform with blogs, CMS, and feeds. It is written in Ruby, using the Rails framework, is licensed under the Affero GNU General Public License version 3, and has an active development community.

The need for supporting federation is a long-standing issue for the Noosfero project. Started in 2007, when the ideas around federation were still incipient, Noosfero ended up evolving a large set of features required by the different organizations that use it before a proper plan for adding federation could be laid out.

We decided to spearhead the development of federation support in Noosfero, with the main goal of allowing Noosfero sites to integrate with other social network providers, whether they also use Noosfero, or not. Our proposal was to use the Diaspora protocol.

Even though the objective is to eventually support the entire protocol, the first step was producing a proof of concept that implements a subset of it. This would help identifying the required modifications in the Noosfero code, and designing the rest of the federation infrastructure. It will provide an initial level of federation that can already be used by end users.

To define the scope of this proof of concept we took into consideration roadmaps that were result of previous discussions in the community. We derived the following requirements from the features we believe will help the most to build the federation infrastructure, mainly the users discovery and message exchange mechanisms.

1. Remote users must be located via the Noosfero people search. The implementation must follow the discovery standard used by Diaspora, based on WebFinger. It must also be possible to find Noosfero users from any other Noosfero or Diaspora site.

2. Users from a Noosfero and a Diaspora site must be able to follow each other. Both sides must be aware of the relationship.

3. The Noosfero server should receive and handle publications sent by Diaspora servers, making it possible to consume contents from remote servers. Sending these messages from Noosfero will make it possible for local users to be followed, what is also the scope of the proof of concept.

4. Noosfero sites should also send publications and comments to any other server that hosts a user that follows the activity of local profiles. This adds symmetry to the previous feature, providing reciprocal interaction between the servers.

[13] http://noosfero.org

IMPLEMENTATION

The first step to implement interoperability through the Diaspora protocol is to offer a mechanism to discover and provide users and public profiles, in this case using WebFinger and hCard.

The second step is to implement a message exchange mechanism to provide the means to talk to third-party servers. The Diaspora protocol proposes the exchange of messages containing entities that represent contents and interactions. For now, Noosfero should handle the following entities:

- Profiles creation, update, and removal
- Publications and comments creation and removal
- Content subscriptions and unsubscriptions
- User relationships (follow and unfollow)

The following subsections also describe technical specifications of the Diaspora protocol, and could be used as reference for future implementations.

User Discovery

The Diaspora protocol proposes the use of WebFinger to fetch user identities and hCard to share public profiles. The implementation used in the Diaspora project still uses XML to format WebFinger payloads, what is considered legacy.

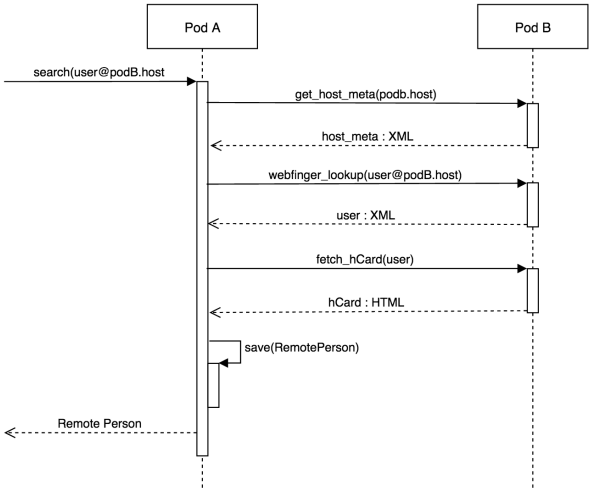

Figure 2. Sequence diagram of the user discovery process.

The discovery process is described in Figure 2, where the remote server is queried with an identifier in the format *user@host*. The discovery endpoint can be found in the server metadata, also obtained via WebFinger, on a standard endpoint. After the identity lookup, the remote server is queried again for the public profile, which in turn is obtained via hCard in HTML format.

As the discovery implementation is based on WebFinger, it is possible to find users on any application that responds to this format.

Message Exchange

To follow an external user, it is necessary to send a private Salmon message to its endpoint. Receiving the message, the Diaspora server creates a local profile to represent the remote user locally — only after querying the Noosfero server for its server metadata and the user's public profile. This process is shown in Figure 3, and includes part of the discovery process.

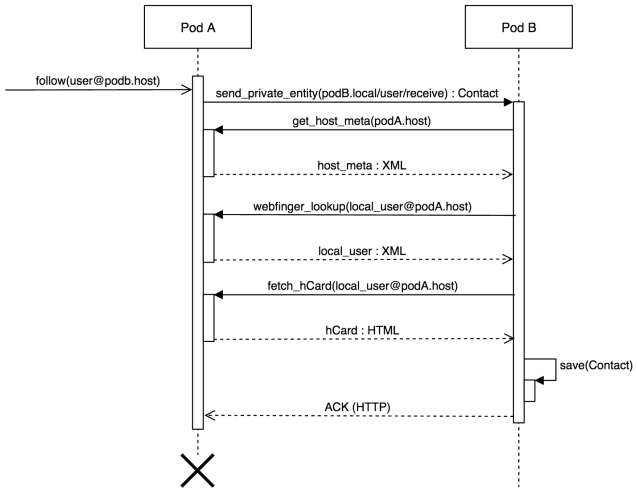

Figure 3. Sequence diagram of the contacts sharing process.

It is crucial to make sure that the message was successfully delivered, otherwise the servers will not share the same state. The server sending the message should offer some kind of reliability, retrying or even undoing actions when facing communication problems. When trying to follow a remote user, Noosfero will retry a predefined number of times, and then destroy the relationship locally if the remote server did not respond with a success status.

Private salmon messages are encrypted with RSA to ensure confidentiality, which requires every user to have a pair of RSA keys. The proposed implementation for Noosfero uses the OpenSSL Ruby bindings to generate a key pair for every user, as soon as that is necessary.

Security is an important aspect that still have to be considered more carefully. We are aware that the server should not hold keys in behalf of users, but for now the keys are serialized and stored in the database. To avoid storing plain text values, the private keys are encrypted using a symmetric key and the *Advanced Encryption Standard*. Ideally, we should be able to find a solution where each user holds its own keys.

The last step is to handle incoming entities representing user contents. As shown in Figure 4, new contents are sent by Diaspora to every server that is involved in the interaction — in this case, the origin of users following the author of the new content. At the time of writing, Noosfero supports only public content.

After receiving the publication notification in a public Salmon message, Noosfero saves the data locally. It is also necessary

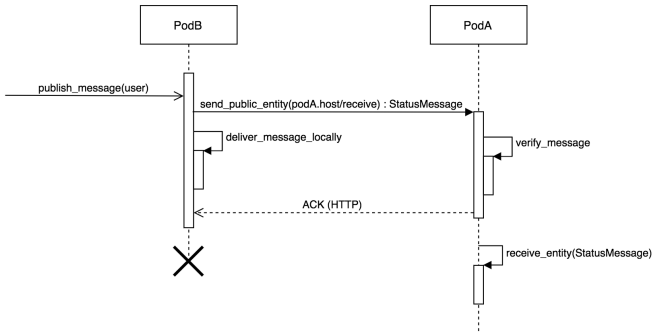

Figure 4. Sequence diagram of the publication sending process.

to include a GUID (Globally Unique IDentifier), required as identifier for all entities sent across servers.

It is also important to handle retraction entities to maintain the same state in all servers. These entities are sent every time a content or user profile is removed, and have the same visibility as the original content. Handling a retraction involves querying for a related local entity that matches the one that has been removed remotely, so that the removal can be replicated locally.

RESULTS

After the implementation described in the last section, it was possible to achieve initial federation functionality that showcases the possibilities of federation using the Diaspora protocol. This proof of concept counts with the following features.

Remote users can be found using Noosfero search using the user identifier and the remote pod host name (Figure 5). Remote profiles are created to store and display the public info in the local server.

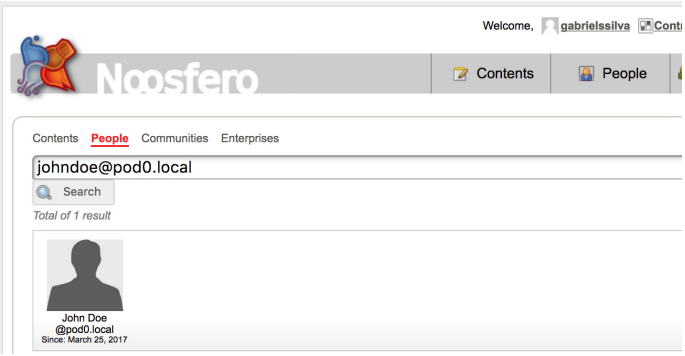

Figure 5. Diaspora user displayed in Noosfero search results.

Noosfero users can also be found on the Diaspora side, since Noosfero servers now respond to WebFinger and hCard requests (Figure 6).

Remote users can be followed by users of a Noosfero server. When following a remote user, Noosfero notifies the remote server, which will handle the message by creating the relationship and notifying the related users. Figure 7 shows a

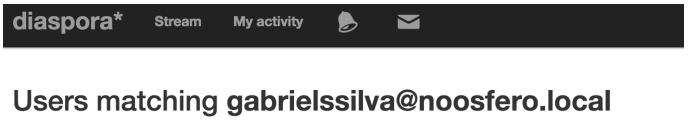

Figure 6. Noosfero users displayed in Diaspora search results.

notification displayed for a Diaspora user after he was discovered and followed by a remote Noosfero server.

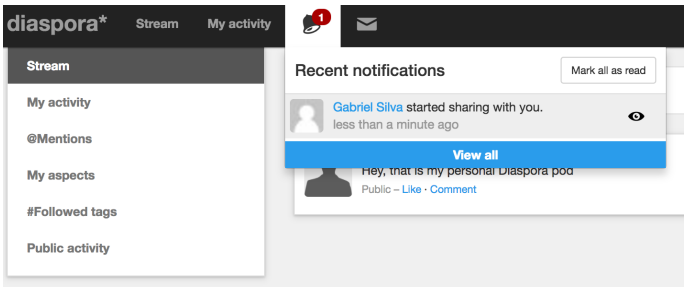

Figure 7. Notification of remote activity in Diaspora.

Any content created by Diaspora users being followed by someone in the local server will be sent to Noosfero, that will handle them by creating the content locally. The publication is displayed in the remote user local profile, but notifications can also be sent to all related users (Figure 8).

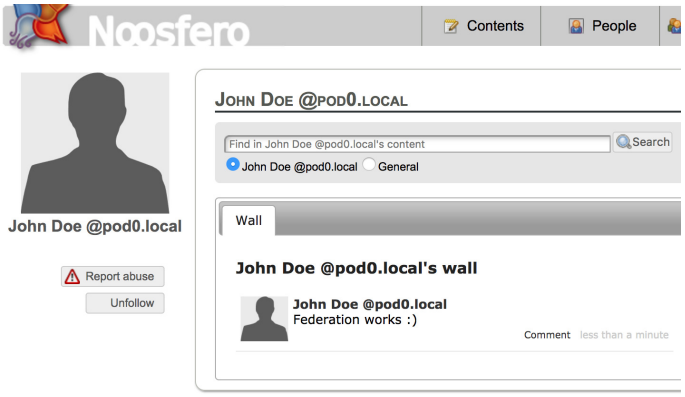

Figure 8. Content created by Diaspora users displayed in Noosfero.

As users are able to reply existing publications, comments must also be sent to remote servers. *Noosfero will create comments locally, and send comments to other servers* when remote users are subscribed (Figure 9). Publications and comments are both important for a first set of federation features.

The information sent by other servers is usually saved to the local database, so there will be an amount of data replication, requiring further attention to maintain coherence across the network. Users can delete publications or comments, edit their

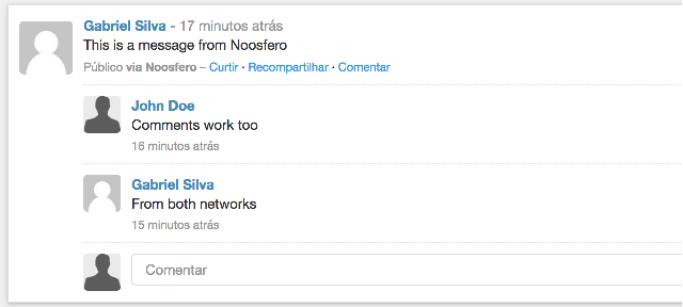

Figure 9. Comments from Noosfero sites are visible in Diaspora pods.

personal information, unfollow other users and even destroy their own account. These tractions are also represented as entities that are handled by Noosfero, which also sends them accordingly.

These features will work between Noosfero and any application that supports the Diaspora protocol, not only Diaspora itself. User discovery will work with any server that supports the WebFinger and hCard standards as well.

CONCLUSIONS

Our report on the existing federation initiatives helped to identify *what are the protocols in use for federation of social networks and what is the status of their development and adoption*. We described several standards that are used to build decentralized social networks, where we can highlight the Diaspora protocol, and OStatus and derived projects, such as Mastodon. While evaluating the adoption of these technologies, we could verify the lack of standardization, identifying a scenario where different projects choose to follow distinct strategies to better suit individual needs.

The Diaspora protocol provides a reliable process for finding users and exchanging messages among servers. By implementing it in Noosfero, we were able to federate an instance with both Noosfero and Diaspora servers, showing that it is feasible to implement federation in an existing application.

It is important to note that the Diaspora protocol suits only a subset of the existing projects, from what follows that it is most probably not a "silver bullet" for building a larger federation.

Even though we can point projects that successfully implemented a decentralized network, such as Hubzilla [14], which also supports the Diaspora protocol, these are usually designed for a decentralized context and aim to provide federation out of the box. An additional challenge lies with projects that were not designed as such since the beginning but still want to support federation.

Our proof of concept helps to answer *what engineering problems need to be solved in order to add support for federation in a platform that was not conceived with federation as a requirement?* To start with, our implementation would require more effort if Noosfero features were not already somewhat compatible with the specification. For example, Noosfero already supported asymmetric relationships ("follows") besides symmetric ones ("friends"). If that was not the case, the implementation would turn harder than it was.

There are a couple points that we want to list regarding the implementation of the Diaspora protocol in existing social networks. These are either challenges that can also be faced or aspects that need attention when adding federation support in projects that were not designed to be federated. These findings also offer some insight on how the Diaspora protocol can be used, and we take the opportunity to point some limitations.

Lessons learned on supporting a federation protocol

Beforehand, it is important to take into account that your data model will have to be modified to contemplate Diaspora entities. For instance, all exchanged entities must have a global identifier (GUID), and each user requires a pair of RSA keys to properly send and read private Salmon messages.

The protocol may also affect the way the project handles users and contents, mostly because all remote data must be stored in the local database, what includes remote profiles and contents. It most likely means that the application must foresee the existence of remote users and provide some level of interaction, like publishing contents and sharing with the rest of the network.

There are also some points that can help to exchange entities among servers. When sending or receiving these entities, besides the message format, it is important to identify the endpoints to be used for the HTTP requests. Remote servers will use analogue endpoints to send its own entities when establishing the communication, so it will be easier to start by choosing a small set of entities and sending them from a remote server, implementing the local receiver endpoints as needed. Contacts and status messages are a good choice of entities to begin with.

Sending and Handling entities should be done in background, since it involves network usage and opening and creating Salmon envelopes, which can be computationally expensive. Whereas part of the work is to build and exchange Salmon messages, one should also consider using some tool that implements it, such as the diaspora federation Ruby library [15].

The reliability of the message sending must be assured by each server. It may be interesting to keep track of the statuses of each pod, for example storing timestamps and number of trials of failed requests. Pods do not inform the network of their status, so everything is still made based on the HTTP responses status. Considering that the failure to deliver messages affects the coherence of the information in the network, the protocol should be reviewed to handle scenario formally.

It is also important to point that a server may implement several federation protocols, i.e. servers A and C support two different federation protocols, and server B wants to join both networks. This use case will affect the implementation, since public interactions from users in B with A or C may be potentially replicated.

[14] https://github.com/redmatrix/hubzilla

[15] https://github.com/diaspora/diaspora_federation

Consider that a user from B interacts with both A and C servers, the state of public contents will need to be the same across all networks. In this case, A and C do not acknowledge each other, so the responsibility to relay the modifications lies with B. Also because A and C are isolated, the *relayability* restriction is not described in any of the specifications, and it is up to the local server to decide whether or not the messages should be relayed.

The relaying problem is also present when all servers support the same protocol, and it is somewhat covered by Diaspora's specification. The idea is that relaying messages to pods that are in some way related to the interaction, it will be easier to find references to new pods, helping the servers to establish new connections. Since every federated feature depends on a reference to the remote pod, it is important to provide the means to broaden the network.

Limitations and future work

Regarding the support of the Diaspora protocol, contents with limited visibility were not in the scope of this proof of concept, and it is an important step to extend the federation features turning the network privacy aware. Although we already handle private messages, we need to match Noosfero circles with Diaspora aspects, making sure the messages will only be visible for the appropriate users.

For now, only plain text messages are being shared across servers. Since Noosfero is also a CMS, it is our interest to federate rich contents, such as articles with embedded HTML and images. The protocol currently offers the means to share static images, so it will probably be the case to extend its custom entities.

It is also important to support direct messages, events and post reactions (such as a "Like"), showing some convergence with the Diaspora interaction model, and increasing the use cases of the federated network. Besides supporting the current state of the protocol, we must attend to the security issues stated previously, possibly following good practices such as using different pairs of keys to sign and encrypt the exchanged messages.

Whereas we were able to implement the protocol and join a network of Diaspora pods, it is clear the specification has to evolve in the ways it deals with content replication and reliability of the messages delivery. A standard approach to exchange messages between servers, such as a mechanism built over HTTP, would address those concerns and be a great contribution to the Diaspora protocol.

REFERENCES

1. M. Atkins, R. Dolin, C. Messina, W. Norris, and M. Wilkinson. 2011. *Atom Activity Streams 1.0*. Technical Report. http://activitystrea.ms/specs/atom/1.0/
2. Paul Baran. 1964. On Distributed Communications Networks. *IEEE Transactions of the Professional Technical Group on Communications Systems* (January 1964).
3. Solon Barocas, Dan Boneh, Arvind Narayanan, Helen Nissenbaum, and Vincent Toubiana. 2012. A Critical Look at Decentralized Personal Data Architectures. *CoRR* abs/1202.4503 (2012). http://arxiv.org/abs/1202.4503
4. Ames Bielenberg, Lara Helm, Anthony Gentilucci, Dan Stefanescu, and Honggang Zhang. 2012. The growth of Diaspora - A decentralized online social network in the wild.. In *INFOCOM Workshops*. IEEE, 13–18. http://dblp.uni-trier.de/db/conf/infocom/infocom2012w.html#BielenbergHGSZ12
5. Danah m. Boyd and Nicole B. Ellison. 2007. Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication* 13, 1 (2007), 210–230. DOI: http://dx.doi.org/10.1111/j.1083-6101.2007.00393.x
6. Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. 2009. PeerSoN: P2P Social Networking: Early Experiences and Insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems (SNS '09)*. ACM, New York, NY, USA, 46–52. DOI: http://dx.doi.org/10.1145/1578002.1578010
7. Wu Chao, Yike Guo, and Bo Zhou. 2012. Social networking federation: A position paper. In *Computers and Electrical Engineering*, Vol. 38. 306–329.
8. E. Hammer-Lahav. 2010. *LRDD: Link-based Resource Descriptor Discovery*. Internet-Draft 06. RFC Editor. 1–16 pages. https://tools.ietf.org/id/draft-hammer-discovery-06.txt
9. Pili Hu, Qijiang Fan, and Wing Cheong Lau. 2014. SNSAPI: A Cross-Platform Middleware for Rapid Deployment of Decentralized Social Networks. *CoRR* abs/1403.4482 (2014). http://arxiv.org/abs/1403.4482
10. J. Genestoux M. Atkins, B. Fitzpatrick and B. Slatkin. 2014. *PubSubHubbub Core 0.4 – Working Draft*. Technical Report. http://pubsubhubbub.github.io/PubSubHubbub/pubsubhubbub-core-0.4.html
11. Ching man Au Yeung, Ilaria Liccardi, Kanghao Lu, Oshani Seneviratne, and Tim Berners-lee. 2009. Decentralization: The future of online social networking. In *In W3C Workshop on the Future of Social Networking Position Papers*.
12. R. Narendula, T. G. Papaioannou, and K. Aberer. 2012. A Decentralized Online Social Network with Efficient User-Driven Replication. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*. 166–175. DOI: http://dx.doi.org/10.1109/SocialCom-PASSAT.2012.127
13. M. Jones P. Jones, G. Salgueiro and J. Smarr. 2013. *WebFinger*. RFC 7033. RFC Editor. 1–28 pages. https://tools.ietf.org/rfc/rfc7033.txt
14. John Panzer. 2009. Salmon Protocol Summary. http://www.salmon-protocol.org/salmon-protocol-summary. (2009). Accessed: 2016-07-15.

15. Stijn Peeters. 2013. Beyond distributed and decentralized: what is a federated network? http://networkcultures.org/unlikeus/resources/articles/what-is-a-federated-network/. (2013). Accessed: 2016-07-13.

16. L. Schwittmann, C. Boelmann, M. Wander, and T. Weis. 2013. SoNet – Privacy and Replication in Federated Online Social Networks. In *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*. 51–57. DOI: http://dx.doi.org/10.1109/ICDCSW.2013.20

17. William Stallings. 2010. *Cryptography and Network Security: Principles and Practice* (5th ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.