Take Control of Your Data: A Distributed Overlay for Online Social Services with Data Ownership and Availability

Andres Ledesma Hariton Efstathiades George Pallis Marios Dikaiakos
Computer Science Department
University of Cyprus
{aledesma, h.efstathiades, gpallis, mdd}@cs.ucy.ac.cy

***Abstract*—Online Social Networks, Cloud Storage and Microblogging services are increasingly popular. Companies provide these services to the public without an apparent cost. In order to generate revenue, companies monetize user data in the form of marketing campaigns while privacy is subject to policies that often disclose private information. As a consequence, researchers have taken the challenge to design decentralized alternatives. These efforts are missing fundamental attributes such as data availability and ownership. We present an overlay that provides these attributes and does not require the user to have technical expertise to enjoy provided services. We implement and test our design on a low-cost device using Twitter statistics. The results show that our proposed overlay is feasible for the majority of Twitter users while preserving data availability and ownership.**

I. Introduction

The use of online social networks has increased dramatically over the past years. With Facebook leading the market with over a billion users [18], these services are likely to thrive in the near and distant future. These services are provided to the public without an explicit cost. However, these companies inherently need to develop a business strategy in order to maintain their technological infrastructure and generate revenues. To this extent, it is understandable, although not acceptable, that companies monetize personal user data. By doing so, privacy violations occur in the form of information disclosure for marketing purposes. Additionally, surveillance and censorship are constant factors at play when the user delegates control of data to a third-party. No mechanisms exist to avoid the misuse of data from companies hosting these services [7].

With this motivation in mind, researchers have taken the challenge of designing decentralized alternatives to centralized social networks and cloud storage services. These alternatives have also attracted the attention of the open source community. Currently, several projects aim to provide alternatives under various architectures such as peer-to-peer (P2P) systems and server federations. These alternatives lack a compromise between the user technical skills and the availability and ownership of data.

Currently, no approach has succeeded in providing data ownership to the user while maintaining data availability. In this paper, we present existing attempts to address these issues, identify their limitations and extract the main challenges behind decentralizing online social services. After presenting the research problem, we introduce our proposed overlay that considers data storage and retrieval while preserving ownership and privacy. We built an implementation of the overlay and tested its performance on a Raspberry PI, a low-cost device with a Linux OS. Using statistics from Twitter, we evaluate the system and extract performance metrics to substantiate its feasibility. We conclude with future directions on the potentials of the presented overlay and further research on the field of decentralization of online services.

II. Problem Definition

In this paper we take a close look at the challenge of **designing a decentralized overlay that preserves user privacy and data ownership while providing high quality online services.**

- The purpose is to decentralized the online service existing at the moment in order to give the user control of the data and overcome privacy issues.
- We assume that the cost of the required hardware is affordable and that the user does not need to have technical skills to setup a server or install a series of applications. To that extent, we address a significant challenge.

We analyze the existing approaches and identify their limitations in order to design and motivate our proposed architecture as a solution that can address the problem.

A. Approaches and Their Drawbacks

Currently, researchers and open source communities aim to decentralized online service with three different approaches.

The first approach is a P2P system where users hold their data. P2P system implement one of the many existing algorithms such as Kademlia, Pastry, Chord or others. An example of this approach is the application RetroShare [1]. The drawback is the availability of the data once a user closes the application. Placing replicas in the computers of other users goes against preserving data ownership. This problem is explained further by Buchegger et al. [10].

The second approach relies on Distributes Hash Tables (DHT) in which replicas are stored in a group of nodes and coordinated by a root. Data availability increases in the underlying overlay while at the same time unnecessary replicas

are discarded. This research work is presented by Legtchenko et al. [15]. The drawback is that the replicated data is stored in a nodes that are not owned by the original user. Users can copy or use data they do not own, without the consent of the author. Additionally, a group of nodes with replicas cannot completely guarantee availability because these they could all leave the network at a given time. The replication would need to spread to more nodes, increasing the risk of misuse.

Similar to a central service, federation management is comprised of a group of servers that are responsible for a portion of the data. Paul et al. [16] name this approach 'Distributed Servers' and it is a reputation system that ranks servers according to users' perception of trustworthiness. An example of this type of data management is Diaspora [3] which consists of a federation of servers called 'pods'. This approach has two drawbacks. First, the reputation system can be tempered if a server initially acquires a high level of trust in order to attract users. Afterwards, the administrator of the server can take advantage of the users that decided to trust in that server. The administrator can then read private information about the user. This problem is described in more detail by Paul et al. [16]. Furthermore, the time required to rank the servers might not reflect these immediate privacy violations and thus new victims will trust a compromised server. The second drawback is that users do not own their data since they have to trust a server that they do not control, similar case as in the previous approach.

The problem we aim to address has not been solved by existing approaches. As we explained in this section, data ownership and availability appear to be orthogonal. The next section describes our approach to address this problem and provide ownership and availability.

III. Design Overview

Low-cost devices are credit-card size computer able to run server applications efficiently, examples include the Raspberry PI, UDOO, Banana PI and Arduino. These devices have a Linux Operating System providing a platform to deploy a wide range of services.

The distributed messaging system that we built aims to leverage from limited hardware resources and provide message exchange between users including sensors supporting HTTP communication. The assumption is that low-cost devices are affordable by the public and the computation power is enough to deploy an overlay that exchanges messages respecting privacy and ownership. Privacy is ensured by exchanging messages with a controlled group of users. Additionally, encryption mechanisms are possible and they can be a combination of synchronous and asynchronous encryption. Data ownership is provided because the low-cost device holds user data and it is owned by the user. The user can remove data from the devices and cannot be censored by a central authority because the server instance is owned by the user.

The messaging system does not transfer messages, it sends notifications that content is available locally. This guarantees data ownership as well. The message exchange is the backbone

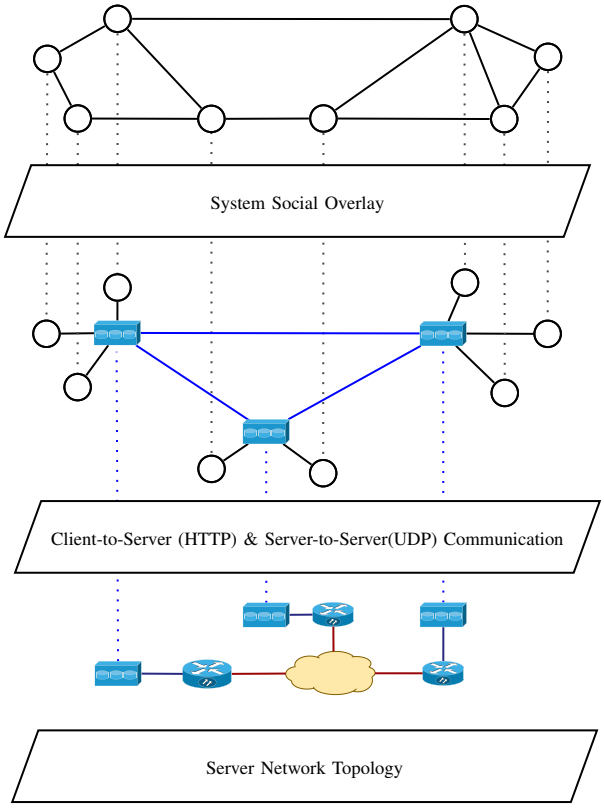

Fig. 1: Communication overlays depict the network topology, client-to-server and server-to-server exchange underlying the online social interactions.

of the Online Social Overlay. Messages can be micro-blogging posts, instant messaging with friends, content sharing, sharing actions ("likes", "recommendations" or links) and also information collected by sensors. The latter is becoming increasingly common with the Internet of Things (IoT). Web cameras, room temperature sensors and actuators increasingly support HTTP communication.

A. Overlays

a) Server Network Topology: The network topology of a typical user has the server instance running in a low-cost device connected to the router and/or modem providing Internet access. The convenience of a device is that the user plugs it next to the router and forgets about it. The service executes and provides the message exchange functionalities transparently.

b) Client-to-Server and Server-to-Server Communication: The communication takes place at two levels: client-to-server and server-to-server. The first level of communication is a traditional client server communication. We chose Hypertext Transfer Protocol (HTTP) because it can incorporate devices as part of the Internet of Things that can communicate to a server in order to aggregate or retrieve data automatically. The second level of communication is the notification service between the servers. The notifications are messages that informs other servers that a user hosted in that instance is allowed

to visit newly available content. The notifications include the source where the new data is available, either as IP addresses or domain names. User Datagram Protocol does not have communication overheads which makes it efficient but at the same time less reliable. The implementation section describes the motivation behind using each protocol in more detail.

c) System Social Overlay: The social interactions between the users of the system are represented as circles in the figure 1. This layer represents the interaction when the servers are abstracted from the system. This represents the perception of the user who interacts with others regardless of the client-to-server and server-to-server communication.

IV. Implementation

The distributed message service is implemented using several technologies. Client-to-server communication uses HTTP and a Representation State Transfer or RESTful architectural style as described by Fielding [12]. The data exchange format JSON with UTF-8 encoded serialization. The communication between the servers uses UDP datagrams. The servers establish their connections using public IP address and ports or domain names known *a priori*. The notification stream service uses WebSocket or a Long Polling mechanism, depending on the browser support.

A. Server to Server

We use UDP for server-to-server notification because it does not have communication synchronization overheads. The notification is a message that informs interested users about new available content. A notification that is not delivered does not impact greatly in the system. Interested users can poll the servers any time to request new available information.

B. Client to Server

The message system serves data to clients using a RESTful API. The protocol is TCP and HTTP because we aim to include any devices that has a Web browser. The Internet of Things has pushed HTTP as the protocol of choice for data exchange. We continue with this trend so that we can incorporate devices, sensors and actuators using HTTP.

1) RESTful API: Representation State Transfer defines four different mechanisms to communicate with an HTTP server. The GET mechanism is equivalent to a read operation, where a client requests information from the server without altering its state. The PUT mechanism is equivalent to a create operation in which a new entry is added to the system. The POST mechanism is an edit operation that changes the value of existing records in the system. The last mechanism is DELETE which removes an entry from the system. The choice of using the RESTful architectural style is its simplicity and compatibility over HTTP and the Internet of Things.

2) Notification Stream: The notification stream service is the real-time communication between the client and the server. After a server receives a UDP notification, interested clients receive an HTTP push message. The server uses a channel or "socket" connected to the client in order to send information as soon as it becomes available. Two mechanisms enable this communication. The first is a WebSocket which is an HTTP communication channel that most modern browsers support. The second is the Long Polling technique which mimics a socket by stalling an HTTP request until the server issues a response whenever content becomes available to the client. These two techniques are known as "HTTP Push" or "reverse AJAX". Notification streaming is more efficient than polling the server periodically because the later results in a large number of HTTP requests returning the same results over a period of time.

3) Database: We selected a document-based or NoSQL database, MongoDB. The unstructured storage engine does not have overheads regarding key management, meta-data and structure verification of the insertions. Messages and notifications can contain variations in their structure. The wide range of devices participating in the message system produce data with different structures. The message system must cope with this diversity with the least possible overhead. MongoDB is an adequate choice considering these factors.

C. Compatibility

We implemented the messaging service with Node.js because it runs in Operating Systems supporting V8 JavaScript Engine. The JavaScript Engine was developed by Google [5] and it is supported in Windows (XP or newer), Mac OS X (10.5 or newer), and Linux systems using IA-32, x64, or ARM processors. The MongoDB database is also supported on these operating systems and architectures. The messaging service is highly compatible and can run on other low-cost devices such as UDOO, Arduino, Banana PI and more.

V. Evaluation

The server stores the data generated by the user that owns the server instance. That data is shared with the interested users. We evaluate the feasibility of the retrieval of data using the RESTful interface. The consumers of the interface are HTTP clients requesting data from the server.

We tested the message service using "Siege", an HTTP benchmark tool that allows the simulation of concurrent connections and reports performance metrics. Test sessions last 5 minutes because after a repeating the experiments with increments of one minute, we found that there is no substantial difference if the test sessions last 5 or 10 minutes, the performance metrics did not change substantially. After the test session is complete, the tool reports the total number of completed transactions during the session, the transactions processed per second, the average response time of the transaction, the shortest and longest response time and the amount of Megabytes transfered per second.

The hardware platform running the message service is a Raspberry PI Model B with 512 MB of RAM and an ARM CPU of 700 MHz. The Operating System is Raspbian, an ARM variation of the Linux distribution Debian. We chose this hardware platform because its price is affordable and it is possible to configure its services before the final user switches

		No Delays			3 Second Delay			6 Second Delay		
Number of Connections		100	208	500	100	208	500	100	208	500
Average Response Time	seconds	0.72	2.11	6.24	0.05	1.09	5.18	0.03	0.07	3.49
Transaction Rate	transactions/second	81.59	79.42	73.48	64.54	87.87	74.11	33.33	67.24	76.42
Shortest Response Time	seconds	0.24	0.59	0.87	0.01	0.09	0.52	0.01	0.02	0.10
Longest Response Time	seconds	1.21	3.62	37.26	0.72	1.89	7.43	0.40	0.53	5.16

TABLE I: Performance evaluation of the message system simulating 100, 208 and 500 concurrent connections with no delays, 3 and 6 second delay. The server application is deployed in a Raspberry PI Model B.

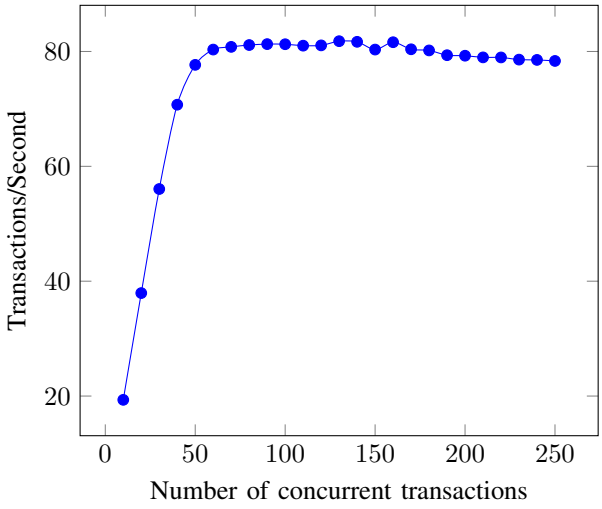

Fig. 2: Plot showing the number of transactions per second given a number of concurrent connections without delays.

the device on for the first time. Internet Service Providers can ship these devices along with the modem and router so as to provide an additional service that can spread the adoption of the system.

Figure 2 shows the performance of the service under stress. The x-axis shows the number of concurrent connections with no delays. The performance metric that is represented in the graph is the number of transactions per second. We tested the server in increments of 10 connections until we reached 250 because the number of transactions per second did not change significantly after 5 consecutive increments in the number of connections. The average value stayed at 80 transactions per second despite the increment on the number of concurrent connections. The performance decreases when the number of concurrent transactions increases because more resources are needed to process an increasing number of requests. Those requests that wait to be processed are placed in a queue increasing memory usage of the server application.

We evaluate the system with Twitter statistics collected by Beevolve [2], a company that specializes in analyzing Social Media. According to the statistics, the average number of followers is 208 and an estimate of 87.4% of the users have less than 101 followers. An additional 9.2% of the users have between 101 and 500 followers. We found after experiments that the limit of the performance is reached fairly soon with no delays in the concurrent connections. The tests with concurrent connections constantly requesting data does not represent the behavior of the user. After the server issues a response, the user takes time to read the data and send a new request. This delay is calculated in an experiment conducted by Stephen Blackwell from "deathandtaxes" News (SpinMedia) [9]. The experiment reveals that it takes 6.6 seconds in average to read a single tweet.

Table I shows the performance metrics with 100, 208 and 500 concurrent connections and no delays. The results show a performance bottleneck with 500 connections in the longest response time, 37.26 seconds. However, we introduced a 6 second delay and the results show that in this scenario, the server is substantially more responsive. The test sessions represent the worst case scenario because they simulate all the followers requesting data at the same time.

We tested the performance of the service instance for 100 followers representing 87.4% of the total Twitter users. In this test we used the delay of 6 seconds to verify the performance under a more realistic scenario. The results show that the performance is acceptable for this test case. When 500 concurrent connections request data from the server, the performance does not achieve the same result. Popular users would require more hardware resources. However, for the majority of Twitter users, the experiments demonstrate the feasibility of our message system running on a low-cost device.

VI. Related Work

Decentralizing online social services is an existing challenge in the research and open source community. Proposed solutions are classified in Peer-to-peer (P2P) systems and server federations. Currently, none of the existing approaches provide data availability and ownership. Server federations have limitations derived from the client-server model, in which the server is neither owned nor controlled by the user. P2P systems rely on replication to increase availability, however replicated data is no longer owned or controlled by the user.

A. Peer-to-Peer

P2P systems are comprised of nodes which are instances of software applications. Users contribute storage and network resources to satisfy the network demand. Communication and storage depends on the number of nodes and on the network demand. Research efforts are mainly focused on P2P algorithms. Persona[8] is a distributed online social network that focuses on user privacy using encryption mechanisms. Homeviews[13] is a P2P middleware that based on pre-created views to share user data. PeerSon[10], Safebook[11] and LotusNet[6] use Distributed Hash Tables (DHT) in order to design decentralized Online Social Networks. Data availability is accomplished using replicas. When data is replicated, ownership is given up to nodes that are not controlled by the user. These approaches store data in random nodes. This leads to performance, availability and security issues identified by Legtchenko et al. [15].

Another option is to store data in nodes owned by friends. Cutillo et al. propose Safebook[11] using the concept of *Matryoshkas*, a set of concentric rings where the user is in the center and each ring represents a level of connection with the core. The first ring contains nodes directly connected to the core, the second ring nodes connected to the first ring (friends of friends). Nodes on the first ring hold replicas and nodes in the outer ring act as entry points. A scalability problem occurs with users that have a large number of friends. Matryoshkas use the actual social graph of the user to generate the rings. High degree nodes will have a large amount of replicas slowing propagation of updates and expired information. Sala et al. [17] demonstrates that users in a social network have a wide range of connectivity. Users with small social graphs will not have enough replicas and thus data availability would be limited.

P2P social networks have limitations with data dissemination. In large dense networks, messages are not sent directly, instead relay nodes forward them through the network. Nodes can act maliciously as intermediates and interfere in the delivery process. Malicious users may acquire sensitive information by analyzing message meta-data. The problem is described in more detail by Greschbach et al. [14].

B. Server Federations

Open source projects such as Daispora [3] and Friendica [4], rely on community hosted servers connected in a network. These servers exchange information forming a federation. Likewise, Shakimov et al. [19] propose a personal cloud service hosted by the community. A server federation provides the data availability that P2P approaches cannot. However, Baden et al. [8] points out that server administrators can access foreign sensitive data. The proposes solution is to encrypt the data but meta-data can still be analyzed. In server federations, users must trust system administrators, thus the same privacy issues of centralized approaches apply to this approach. Priv.io is another approach proposed by Zhang and Mislove [20] that allows the user to deploy a server application using a cloud provider. In this approach, data ownership is not satisfied because cloud providers own the hardware and may have access to user data or enforce their policies affecting the user.

VII. Conclusion and Future Work

We presented the design and implementation of an overlay with data availability and ownership. The implementation of the overlay demonstrates its feasibility and deployment on a low-cost device. We used Twitter statistics to simulate the demand on the server application and determined that the performance is sufficient for 87.4% of Twitter users. The source code is available in GitHub along with the performance tests and the client applications used to simulate the users. The project can be found at the following URL: https://github.com/isocial-itn/distributed-messaging. The proposed overlay considers users sharing data via a device running a server application. This devices is intended to stay at home, next to the modem and router. The application is the interacting hub that converges data exchange among users and sensors as part of the IoT.

The overlay potential comprises research and practical situations such as personalized recommendation systems based on users personal data, situation-aware online interactions, health monitoring and smart home services. Distributed agents can provide knowledge about current situation, from supermarket sales to disease spread and epidemics. The analysis of data generated from different sources, provides knowledge can improves the quality of life of its users while keeping the data private and at home.

References

[1] Retroshare. Technical report, Retroshare Team, 2009. http://retroshare.sourceforge.net/, accessed August 2014.

[2] An Exhaustive Study of Twitter Users Across the World. Technical report, Beevolve Inc, 2012. http://www.beevolve.com/twitter-statistics/, accessed August 2014.

[3] Federation protocol overview. Technical report, Diaspora Foundation, 2013. https://wiki.diasporafoundation.org/Federation_Protocol_Overview, accessed August 2014.

[4] Friendica. Technical report, Friendica Open-Source Community, 2013. http://friendica.com, accessed August 2014.

[5] V8 JavaScript Engine. Technical report, Google, Inc, 2014. https://code.google.com/p/v8/, accessed August 2014.

[6] L. M. Aiello and G. Ruffo. Lotusnet: Tunable privacy for distributed online social network services. *Comput. Commun.*, 35(1):75–88, Jan. 2012.

[7] M. Aspan. How Sticky Is Membership on Facebook? Just Try Breaking Free, 2008. http://www.nytimes.com/2008/02/11/technology/11facebook.html?_r=0, accessed August 2014.

[8] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: An online social network with user-defined privacy. *SIGCOMM Comput. Commun. Rev.*, 39(4):135–146, Aug. 2009.

[9] S. Blackwell. It Would Take One Person 10 Years to Read Every Tweet Posted on Twitter in a Single Day, 2011. http://www.deathandtaxesmag.com/56791/it-would-take-one-person-10-years-to-read-every-tweet-posted-on-twitter-in-a-single-day/, accessed August 2014.

[10] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta. PeerSoN: P2P Social Networking: Early Experiences and Insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, SNS '09, pages 46–52, New York, NY, USA, 2009. ACM.

[11] L. Cutillo, R. Molva, and M. Onen. Safebook: A distributed privacy preserving online social network. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pages 1–3, 2011.

[12] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[13] R. Geambasu, M. Balazinska, S. D. Gribble, and H. M. Levy. Homeviews: Peer-to-peer middleware for personal data sharing applications. In *IN PROC. OF SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*, 2007.

[14] B. Greschbach, G. Kreitz, and S. Buchegger. The devil is in the metadatanew privacy challenges in decentralised online social networks. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 333–339. IEEE, 2012.

[15] S. Legtchenko, S. Monnet, P. Sens, and G. Muller. RelaxDHT: A Churn-resilient Replication Strategy for Peer-to-peer Distributed Hash-tables. *ACM Trans. Auton. Adapt. Syst.*, 7(2):28:1–28:18, July 2012.

[16] T. Paul, B. Greschbach, S. Buchegger, and T. Strufe. Exploring decentralization dimensions of social networking services: Adversaries and availability. In *Proceedings of the First ACM International Workshop on Hot Topics on Interdisciplinary Social Networks Research*, HotSocial '12, pages 49–56, New York, NY, USA, 2012. ACM.

[17] A. Sala, H. Zheng, B. Y. Zhao, S. Gaito, and G. P. Rossi. Brief announcement: Revisiting the power-law degree distribution for social graph analysis. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 400–401, New York, NY, USA, 2010. ACM.

[18] A. Sedghi. Facebook: 10 years of social networking, in numbers, 2014. http://www.theguardian.com/news/datablog/2014/feb/04/facebook-in-numbers-statistics, accessed August 2014.

[19] A. Shakimov, H. Lim, R. Caceres, L. Cox, K. Li, D. Liu, and A. Varshavsky. Vis-a-vis: Privacy-preserving online social networking via virtual individual servers. In *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, pages 1–10, 2011.

[20] L. Zhang and A. Mislove. Building confederated web-based services with priv. io. In *Proceedings of the first ACM conference on Online social networks*, pages 189–200. ACM, 2013.