Sweets: A Decentralized Social Networking Service Application Using Data Synchronization on Mobile Devices

Rongchang Lai(✉) and Yasushi Shinjo

Department of Computer Science, University of Tsukuba,
Tsukuba, Ibaraki 305-8573, Japan
yas@cs.tsukuba.ac.jp
http://www.softlab.cs.tsukuba.ac.jp/~yas/

Abstract. Conventional Social Networking Services (SNSs) or Online Social Networks (OSNs) are implemented based on a centralized architecture, and this centralization causes privacy problems. This paper describes Sweets, a decentralized SNS application that synchronizes tweets among users' own mobile devices and enables users to retain ownership of their data. Sweets provides user authentication and access control without centralized servers. Sweets improves the data availability using an indirect replication scheme. Experimental results show that Sweets has a feasible performance over a 4G cellular network.

Keywords: Decentralized Social Networking Services (DSNSs) · Decentralized Online Social Networks (DOSNs) · Data synchronization · Decentralized user authentication · High data availability · Access control

1 Introduction

Most current popular Social Networking Services (SNSs) or Online Social Networks (OSNs) are constructed based on a centralized architecture and all of the user data are hosted on centralized servers. Service providers can easily access user data to improve the user experience and commercial benefits through targeted advertising. Obviously, from the user's perspective, such centralized architecture causes a violation of privacy. People lose control of their own data and are tracked by SNS providers. Some recent events have proved the existence of such concerns [10]. Finally, if service providers shut down their services, users may lose their data and can no longer retrieve them.

Due to these drawbacks of current centralized SNSs, a number of decentralized SNS approaches [1,5,6,8] have been proposed. However, these approaches suffer from multiple problems, such as permanently available data storage required, lack of revocation support, and high overheads. In addition, to the best of our knowledge, no existing decentralized SNS approach is compatible with mobile devices, which is the major mode by which people use SNS applications.

S. Wang and A. Zhou (Eds.): CollaborateCom 2016, LNICST 201, pp. 188–198, 2017.
DOI: 10.1007/978-3-319-59288-6_17

To address these problems, we are implementing a decentralized SNS application using data synchronization. We call our application Sweets, which stands for Sync Tweets. The goal of Sweets is to realize a decentralized SNS application that runs on mobile devices.

Sweets achieves basic SNS functionality with data synchronization using Couchbase Lite [3] and supports revocation of tweets. Users can also define and enforce a fine-grained access control policy to restrict the dissemination of their data. We implement decentralized user authentication in Sweets using OpenID Connect [14].

Since we implement an indirect replication scheme, the application provides high data availability regardless of whether users are online or offline. For the indirect replication, we implement access control with Attribute Based Encryption (ABE) [9].

2 Related Work

Twister [8] is a decentralized microblogging application over a Distributed Hash Table (DHT). However, the DHT network is unsuitable for mobile devices. Since a device must serve all the other devices in the DHT network, this quickly depletes battery life and data volumes can become very large. Furthermore, Twister does not support revocation. Unlike Twister, we implement a decentralized SNS application without DHTs, and enable users to delete or update their posted tweets.

Vegas [6] accomplishes decentralization using a federated server architecture. In this research, data synchronization is achieved through a component called datastores. A datastore represents the abstract concept of a public user-writable storage space with world-readable access. Datastores can be deployed on a cloud storage service like Google Drive or Dropbox. Compared with Vegas, our research realizes this component on users' mobile devices.

Persona [1] is a privacy-enhanced decentralized SNS application that provides flexible, user-defined access control with ABE. Similar to Vegas, Persona's design requires permanently available data storage. The data hosted on data storage systems are encrypted by ABE. Sweets differs from Persona in respect to data storage, as it stores user data on their own devices instead of permanently available data storage. Sweets combines data synchronization and ABE, so that encrypted data can be transferred among mobile devices and relayed by friends who have no permission to access the data.

3 Implementing Basic SNS Functionality on Mobile Devices

We implement our SNS application, Sweets on Android mobile devices. Figure 1 shows a screenshot of Sweets. The user interface of the application is similar to current popular SNS applications and gives users a similar user experience.

A user can post tweets with texts and photos from an album or camera. Users can pull the latest tweets from their friends by pulling the main view down. Each tweet item has a number of buttons that allow users to comment on a tweet or give it a thumbs-up. If their friends pull these changes back, they can retrieve these comments and thumbs-up updates. When a user deletes a posted tweet, Sweets forces other friends who have previously pulled this tweet to delete it from their devices immediately.

3.1 Basic SNS Functionality by Data Synchronization

We implement Sweets based on data synchronization among mobile devices. Figure 1 shows the high level architecture of Sweets. Each user runs Sweets on a mobile device. Sweets consists of the user interface, the identity provider, and an embedded database. When a user posts tweets and photos, they are first stored on the local database. Next, Sweets synchronizes the local database with other users' databases using peer-to-peer TCP connections over Social Virtual Private Networks (VPNs) [7]. During the synchronization process, Sweets pulls new or modified data from remote databases and stores them on the local database. Finally, Sweets shows the user tweets, photos, and friends' comments in the local database.

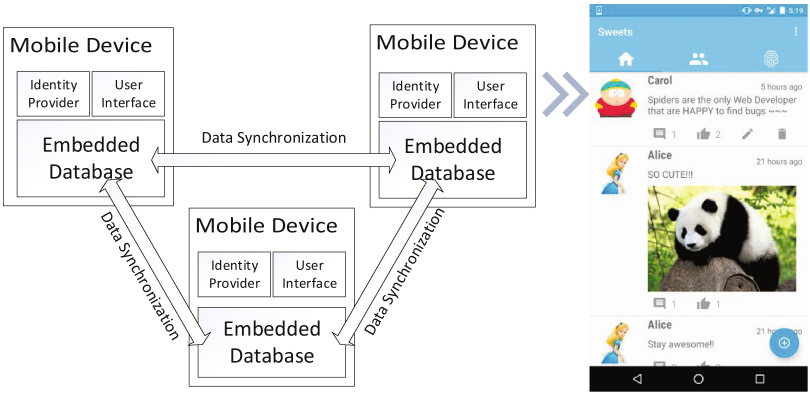

Fig. 1. Overview of sweets

Data synchronization is executed asynchronously as a background task. Users are unaware of the time consumption during data synchronization and the user interface avoids becoming frozen due to data synchronization.

3.2 Couchbase Lite

We use Couchbase Lite for data synchronization among mobile devices. Couchbase Lite is an embedded document-oriented database management system

(DBMS). It uses a technique called Multiversion Concurrency Control (MVCC) to manage data replication in a distributed environment, and documents in Couchbase Lite are automatically versioned. With the MMVC technique, Couchbase Lite enables developers to process the revision history of a document.

We integrate Couchbase Lite into Sweets. With Couchbase Lite, any data element that should be replicated is versioned, and any data update is tracked. Sweets only replicates missing updates during data synchronization. Most of the time, Sweets pulls updates rather than pushing them. By this means, data updates are only synchronized if some other users are willing to retrieve them. As a result, Sweets minimizes data synchronization traffic and confines the resource consumption, which is sensitive for mobile devices.

Data synchronization among mobile devices comes with some inherent problems, including data conflict and deletion synchronization. The following sections describe how we resolved these problems.

3.3 Conflict Resolution

In Sweets, posted tweets are replicated to remote databases, and friends can comment on them. A tweet can be updated on multiple databases. We want to gather all comments and resolve this conflict; namely, we need to merge these branches. Unfortunately, Couchbase Lite provides a simple conflict detection and resolution mechanism, selecting a winner revision; this requires us to drop a number of comments.

To avoid this, we solve the conflict by other means. Every time the conflict occurs during synchronization, we aggregate all conflicting revisions of a tweet document in the device on which the user has posted the tweet. Next, we merge all these contents, and create a new revision for the conflicting tweet document that incorporates all the comments obtained from other friends. Next, we set the new revision as the current revision of the document. Finally, we delete all obsolete conflicting revisions. Once someone pulls this tweet again, the latest revision will be pulled and the obsolete revision in her/his database will be updated. By this way, she or he can also retrieve the latest comments from other users.

3.4 Implementing Revocation

Revocation is another problem we need to solve. We want to ensure that deleted tweet documents are deleted from other databases as soon as possible.

To realize such rapid tweet message revocation, we use a special revision object called *tombstone revision* and a different replication scheme. When a user deletes a tweet from her/his own device, a tombstone revision object, which indicates that the deleted document will be created for this document. We use a push rather than a pull scheme to synchronize the tombstone revision. Once the tombstone revision is set as the current revision, the corresponding document will be deleted from other remote databases.

4 User Authentication and Access Control in Data Synchronization

One of the fundamental features of SNSs is their ability to share selected pieces of information with selected friends or groups. To accomplish this fundamental feature, we implement access control in our decentralized SNS without central authority.

4.1 Implementation of Decentralized User Authentication with OpenID Connect

OpenID Connect is the third generation of OpenID technology [13]. It is an authentication layer on top of OAuth2.0. OpenID Connect utilizes the OAuth 2.0 semantics and flows to allow applications to access the user's identity, which is encoded in a JSON Web Token (JWT) called *ID token*. The design of OpenID Connect protocol is flexible. The Identity Provider (IDP) can not only comprise central Internet services such as Google and Yahoo but also many other types of IDP, including IDPs that run on users' own mobile devices. The latter are called *self-issued OpenID providers*.

Based on OpenID Connect, we implement decentralized user authentication for Sweets. We run self-issued OpenID providers on users' devices, and provide users with ID tokens as their identity. While a typical centralized application uses a user name and password to log in, our SNS application uses this ID token. When a user opens the Sweets, the IDP running on the same mobile device is called automatically. Following user authentication, the IDP sends back her/his ID token to the SNS application. This ID token will be sent to a remote device in data synchronization. The remote node will verify the received ID token thought the OpenID Connect procedure, and perform access control using the user ID in the ID token.

When two users establish the social relationship, they exchange their user identities with ID tokens in a face-to-face way rather than over untrusted servers. Due to the features of mobile devices, users can exchange their identities conveniently using Quick Response (QR) codes and cameras, Bluetooth, and Near Field Communication (NFC) networks.

4.2 Access Control Model for Direct Replication

The access control model used in direct data synchronization is similar to that for files in the Microsoft Windows operating system [12]. In this access control model, any user data have a corresponding access control list (ACL) that identifies the users and groups with allowed or denied access to the data element. When a user tries to pull a data element, the application steps through the ACL for each document and checks Access Control Entry (ACE) until it obtains the permission defined by the user.

We integrate our access control model with filtered replication in Couchbase Lite. *Filtered replication* examines documents through a filter function [3]. A filter function is executed when a remote user tries to synchronize data with the local database. Once the remote user tries to pull data from the local database with her/his ID token, the filter function steps through documents and determines whether the document can be pulled based on the ACL.

5 Indirect Replication

To improve data availability, we enable users to pull the data of offline friends indirectly from mutual friends. For this indirect replication, we realize another access control scheme based on Attribute Based Encryption (ABE).

5.1 Access Control for Indirect Replication

With the aim of providing access control during indirect replication without an access control list, we integrate Attribute-Based Encryption (ABE) into Sweets and implement a novel access control scheme using ABE.

In an ABE cryptosystem, ciphertexts are labeled with sets of attributes and secret keys (SK) are associated with access structures that control which ciphertexts a user can decrypt. In particular, we utilize Ciphertext-Policy Attribute Encryption (CP-ABE) [2] in our application. In this cryptosystem, a user's secret key is associated with an arbitrary number of attributes expressed as strings. With ABE, we implement a hybrid encryption scheme for access control in indirect replication.

In indirect replication, we use two types of documents: tweet documents and encrypted tweet documents. An encrypted tweet document comprises the encrypted content of a tweet, which is encrypted using Advanced Encryption Standard (AES) [4] with a random key. We encrypt the AES key with the given access policy using ABE and store the encrypted AES key within the encrypted tweet document.

At initialization time, when two users establish a social relationship, they determine each other's attributes within their social network (e.g., family, friend, etc.), and exchange public keys (PK) and secret keys of ABE.

In Fig. 2, Bob is the mutual friend of Alice and Carol. Alice publishes a tweet and Sweets creates a tweet document and the corresponding encrypted tweet document. In direct replication, Sweets checks the ACL. If Bob is given permission to access the tweet, he can pull both documents. However, since Alice forbids Bob from accessing the tweet, Bob pulls only the encrypted tweet document. Finally, Alice goes offline.

When Carol tries to pull the tweet from Alice directly, Sweets is aware that Alice has been offline; thus, Sweets pulls the encrypted tweet document from Bob. After replicating the encrypted tweet document, Sweets decrypts the AES key within the document using the ABE secret key. If Alice allows Carol to

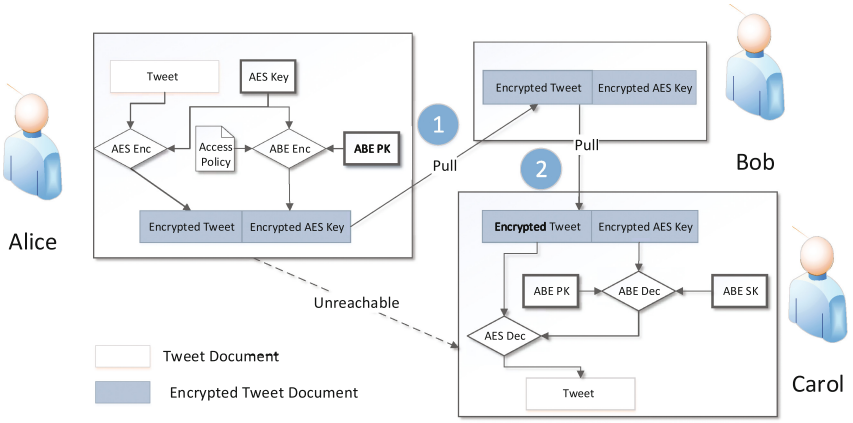

Fig. 2. Access control implementation with ABE

access the tweet, then she can decrypt it and obtain the AES key successfully. With the AES key, Carol can decrypt the tweet content.

As a result, Sweets achieves high data availability with this access control scheme. Furthermore, our implementation comes with some other merits. First, the access control is executed on the receiver side. Relaying friends cannot be aware who has the permission. Sweets allows users to grant fine-grained access to their data without replicating the access control list to other users. Second, with the hybrid encryption scheme, users can encrypt the tweet content with its key again after update, e.g., making comments and thumbing-up. Obviously, the new revision is also access limited under the same access policy. The author of the tweet can receive the comment indirectly.

5.2 Mirror Selection

As described in Sect. 3.2, Couchbase keeps the revision history of a document, and increases the sequence number of a database for each update. Our implementation uses this sequence number for mirror selection in indirect replication. In particular, Sweets selects the most eligible mutual friend as the mirror by comparing their sequence numbers when they pull encrypted tweet documents from offline friends. Our mirror selection scheme enables users to obtain almost the latest tweet documents of offline friends. Moreover, we can rank mirrors in terms of objective reputation scores which are calculated by reputation measurement [11]. The reputation score of a mirror represents the overall performance, and it can be determined not only by the sequence number, but also the response time, availability, etc.

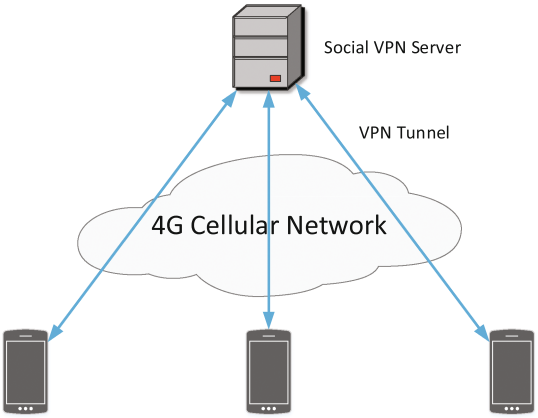

Fig. 3. Performance evaluation experiment configuration

6 Evaluation

In this section, we evaluate Sweets. Sweets is operational using the Social SoftEther VPN over a 4G cellular network. Social SoftEther VPN [15] is an implementation of Social VPN based on SoftEther VPN [16]. SoftEther VPN is our open-source VPN software. In our experiment, we established the connection between several Android devices over the cellular network using Social SoftEther VPN. Social SoftEther VPN allows Sweets to identify friends and connect their devices with domain names that include friends' identifiers.

6.1 Qualitative Aspect

So far, we have achieved the following goals. First, compared with current centralized SNS applications, Sweets preserves user privacy by discarding centralized storage servers and enabling users to store their data on their own devices. Second, Sweets runs on mobile devices, which is the major mode by which people participate in SNSs. Third, Sweets provide strong access control schemes in both direct and indirect replication.

The current implementation has some limitations. First, Sweets depends on a social VPN. We use SoftEther VPN to realize the social VPN, and users must trust VPN servers and their owners. If the VPN server is cracked, Sweets can leak private information. Second, due to the features of the distributed network, Sweets does not support certain functionality, e.g., search.

6.2 Performance

We measured the performance of Sweets. The experiment configuration and experiment environment is as follows (Fig. 3):

- Mobile device: Nexus 5X (Android 6.0, 1.8 GHz Snapdragon 808 CPU, 2 GB RAM)
- Mobile network operator: Softbank (4G)
- Average latency over VPN: 101 ms
- Bandwidth over VPN: 1.73 Mbps

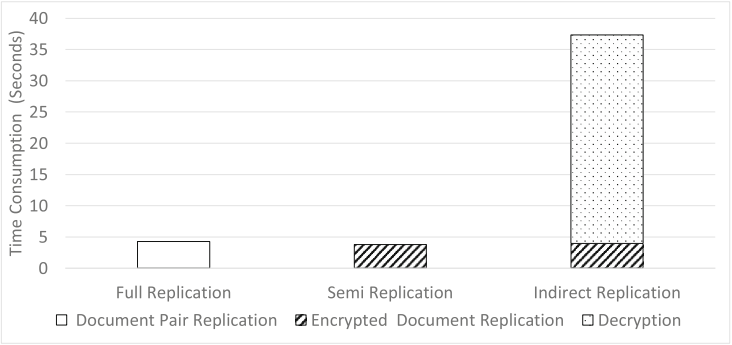

Fig. 4. Execution time of replicating 50 tweets

For the comparison, we have measured the performance in a LAN. However, due to the implementation of an Android Wi-Fi module, which limits the Wi-Fi connection performance, we obtained a worse performance on a LAN than on the 4G cellular network. Hence, we omitted the LAN performance results.

We measured the time consumption to transfer 50 tweets in direct and indirect replication. Figure 4 shows the results. Full replication means direct replication when the receiver has the permission for the tweets. In this case, both tweet documents and encrypted tweet documents are replicated. Semi replication means direct replication, which replicates only encrypted tweet documents. The indirect replication consists of two phases. The first phase entails pulling encrypted tweet documents. After the first phase, Sweets decrypts these encrypted tweet documents.

Sweets took approximately 4 s to pull 50 tweets in direct replication and approximately 35 s in indirect replication. These execution times are adequate for exchanging short tweets among a small number of friends over the 4G network. If a user has 100 friends, and they post 10 tweets per day, Sweets handles 2,000 tweets per day. Sweets can handle these messages within a few minutes per day in direct replication and 20 min per day in indirect replication. Obviously, tweet document decryption is the bottleneck in the indirect replication. We noticed that AES decryption only took 30 ms for 50 tweet documents, and the replication time can be substantially shorten by reusing AES keys [1].

Data synchronization is conducted as a background task asynchronously after as soon as Sweets is launched. When users refresh the time line through the user interface, the application simply retrieves the data from the local database and

shows them to users. Hence, we believe that Sweets is feasible for mobile device users and can provide almost the same user experience as that provided by current popular centralized SNS applications.

7 Conclusions and Future Work

This paper has described a decentralized SNS application, Sweets. We implemented Sweets using data synchronization among mobile devices. Sweets also provides user authentication and access control without centralized servers. Furthermore, we improved the data availability of Sweets using an indirect replication scheme. We realized basic social networking functionality on Android devices. Finally, we have shown that Sweets has a feasible performance over a 4G cellular network.

As future work, we plan to realize a better mirror selection scheme in indirect replication and achieve higher data availability.

References

1. Baden, R., Bender, A., Spring, N., Bhattacharjee, B., Starin, D.: Persona: an online social network with user-defined privacy. In: The ACM SIGCOMM 2009 Conference on Data Communication, pp. 135–146 (2009)
2. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, pp. 321–334 (2007)
3. Couchbase Lite. http://www.couchbase.com/nosql-databases/couchbase-mobile
4. Daemen, J., Rijmen, V.: The Design of Rijndael: AES-The Advanced Encryption Standard. Springer Science & Business Media, Heidelberg (2013)
5. Datta, A., Buchegger, S., Vu, L.H., Strufe, T., Rzadca, K.: Decentralized online social networks. In: Furht, B. (ed.) Handbook of Social Network Technologies and Applications, pp. 349–378. Springer, Heidelberg (2010)
6. Durr, M., Maier, M., Dorfmeister, F.: Vegas-a secure and privacy-preserving peer-to-peer online social network. In: 2012 International Conference on Privacy, Security, Risk and Trust (PASSAT), pp. 868–874. IEEE (2012)
7. Figueiredo, R.J., Boykin, P.O., Juste, P.S., Wolinsky, D.: Integrating overlay and social networks for seamless P2P networking. In: Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 93–98. IEEE (2008)
8. Freitas, M.: Twister-a P2P microblogging platform. arXiv preprint (2013). arXiv:1312.7152
9. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 89–98. ACM (2006)
10. Greenwald, G.: No Place to Hide. Metropolitan Books, New York (2014)
11. Huang, L., Wang, S., Hsu, C.H., Zhang, J., Yang, F.: Using reputation measurement to defend mobile social networks against malicious feedback ratings. J. Supercomput. **71**(6), 2190–2203 (2015)
12. Microsoft Windows access control. https://msdn.microsoft.com/en-us/library/windows/desktop/aa374860(v=vs.85).aspx
13. OpenID. http://openid.net/

14. OpenID Connect. http://openid.net/connect/
15. Shinjo, Y., Kunyao, X., Kainuma, N., Nobori, D., Sato, A.: Friend news system: a modern implementation of Usenet over social VPNs. In: 7th IEEE International Conference on Social Computing and Networking, pp. 432–440 (2014)
16. Softether VPN Project. http://www.softether.org/