Twitterize: Anonymous Micro-Blogging

Jörg Daubert* and Leon Böck* and Panayotis Kikiras† and Max Mühlhäuser* and Mathias Fischer*
* CASED / Telecooperation Lab
Technische Universität Darmstadt
{daubert, tim.grube, max, mathias.fischer}@tk.informatik.tu-darmstadt.de
† AGT International
pkikiras@agtinternational.com

***Abstract*—Privacy, in particular anonymity, is required to increase the acceptance of users for the Internet of Things (IoT). The IoT is built upon sensors that encompass us in each step we take. Hence, they can collect sensitive, privacy-invading data that can be used to establish complete user profiles. For this reason, sensing in the IoT needs to provide means of privacy-protection. In this paper, we discuss an approach for sharing smartphone sensor data and user-generated content in a privacy-protecting manner via the Micro-blogging platform (MbP) Twitter. For that, we discuss privacy needs of users in Micro-blogging platforms (MbPs) and that privacy should not only ensure confidentiality but also anonymity. We discuss related work and systems along these requirements and conclude that anonymity is hardly considered. We introduce our construction Twitterize that integrates well with the MbP Twitter and allows users and sensors to share information normally as well as privacy-preserving with a single application. Twitterize establishes overlay networks for hashtags over Twitters' social network and neither depends on additional infrastructure nor peer-to-peer communication.**

I. Introduction

Micro-blogging is one of the most popular social Internet applications. For instance, the well known Micro-blogging platform (MbP) Twitter claimed more than 200 million active users during a single month with more than 400 million messages published every day[1].

With micro-blogging, users publish messages to groups: either publicly to all users, or privately to a predefined group. To structure public messages, MbPs offer means to organize messages along topics. For example, Twitter established the notion of *hashtags* as topics annotation. Users follow such hashtags, may even forward messages further, and therefore become part of the global community [1].

Besides standard micro-blogging, platforms like Twitter can also disseminate sensor data from smartphones to a large audience. However, data dissemination via micro-blogging brings up privacy challenges. In standard micro-blogging it is easy to identify users, so that the risk of defacement and even political repression is high. This risk becomes evident when considering that MbPs played a major role during the Arab spring movement in Egypt[2]. Moreover, even dissidents are known to use MbPs[2]. When using micro-blogging platforms to disseminate smartphone sensor data, the identity of the devices and thus also the user identity needs to be concealed. Otherwise, sensor data from different sources can be connected to establish complete user profiles that would result in the end of all privacy. Thus, appropriate privacy protection is required when using MbPs either for user generated content or for disseminating sensor data.

In this paper, we propose the privacy-preserving micro-blogging application Twitterize for Android that does not only protect confidentiality but anonymity as well. Twitterize integrates normal Twitter operation with privacy-preserving Twitter. For that, we transfer our mechanism for anonymous overlay construction [3] to the application domain of MbPs. Twitterize establishes a P2P overlay via tweets and therefore does not depend on other servers or communication channels.

The remainder of this paper is structured as follows: first, we discuss requirements to MbPs, related work, and applications in Section II. Then, we introduce Twitterize as a solution in Section III, followed by an evaluation in Section IV. Finally, we wrap up our contribution in Section V and point out future work.

II. Requirements and Related Work

In this section, we introduce requirements to privacy-preserving micro-blooging and analyze related work in the area of privacy with respect to these requirements and the applicability to MbPs.

A. Requirements

We use the following privacy requirements to evaluate privacy-preserving MbPs.

- **Anonymity:** Participants should remain anonymous within sufficiently large anonymity sets [4]. In particular, their group memberships should be concealed within an anonymity set. This requirement should protect against global adversaries, e.g., the MbP itself, as well as inside adversaries, e.g., when devices and secrets get compromised.
- **Confidentiality:** Micro-blog entries, like tweets, should be kept and transmitted secretly between sender and receivers.

In addition to these non-functional requirements, a privacy-preserving MbP should also comply to the functional requirements of a normal MbP and impose low additional overhead. We focus on the following requirements.

[1]https://blog.twitter.com/2013/celebrating-twitter7
[2]http://www.technologyreview.com/news/422735/an-app-for-dissidents/

- **Understandable privacy:** The system should indicate the privacy-protection applied to each message. The user should be able to chose the desired level of privacy-protection.
- **Easy group management:** The process of adding and removing participants should be integrated in the system, quick to perform, and easy to understand. Every participant should be able to easily determine his current group memberships.
- **Low overhead:** The privacy-protection should cause only mild overhead. That is signaling overhead, memory consumption, and battery drain on mobile devices.

B. Related Work

First, we summarize related anonymization systems. Second, we discuss privacy-preserving micro-blogging applications. Third, we discuss privacy-preserving group communication applications.

a) Anonymization Protocols: Such protocols hide senders and receivers of messages within a set of participants, the anonymity set [4]. This can be achieved by relaying messages through other nodes to remove the link between message and sender/receiver. MIX networks, onion routing, and Crowds are the most common protocols to implement this scheme.

MIX networks were first introduced in 1981 [5]. In a MIX network, a sender forwards a message through several MIXes to the sender. Each MIX collects multiple messages, and forwards them in shuffled order to the next MIX. Therefore, each MIX conceals the link between incoming and outgoing messages, rendering senders unlinkable to receivers.

Onion routing, first published in 1998, extends MIX networks and is today predominantly know from The Onion Router (Tor) [6]. In the Tor system, the sender creates a virtual circuit consisting of several Tor nodes. The sender adds layers of encryption around the message, each layer only decryptable by a specific Tor node and only revealing the address of the next node in the circuit.

In Crowds [7], introduced at the same time as onion routing, participants organize themselves in groups, so called crowds. The sender forwards the message to another crowd member. Each crowd member may forward to yet another crowd member or to the receiver. Thus, onion routing and Crowds also remove the link between sender and receiver of a message.

These systems offer particular benefits while suffering from drawbacks. MIX networks provide good message decoupling but require high latency or cover traffic (overhead). Onion routing offers low latency but requires to establish circuits first, in particular for both-sided privacy-protection. Crowds does not depend on circuits but rather forwards probabilistically. Members have to maintain crowd sets. However, attacks on all protocols are possible: [8], [9].

Off-the-Record Messaging (OTR) describes another protocol for privacy-preserving communication [10]. Compared to the previous three protocols, OTR provides confidentiality with plausible deniability rather than anonymity. Goldberg et al. extend OTR with group communication: mpOTR [11]. They claim that mpOTR can be combined with an anonymization service to ensure confidentiality, deniability, and anonymity. However, the three anonymization service presented here only support point-2-point communication and thus do not fit the MbP scenario.

Senftleben et al.[12] propose a micro-blogging service based upon Crowds and mobility of users. Senders transfer messages to other crowd members using ad-hoc P2P communication if possible. Crowd members spread the message further or store the message at a central server in case participant mobility is insufficient to bridge crowds. However, a full protocol specification and evaluation is still missing.

In summary, the presented anonymization schemes do not fit the MbP scenario or impose overhead by presuming a P2P overlay approach.

b) Privacy-preserving Twitter Applications: Secure Twitter applications haven been proposed in recent years: Hummingbird [13] mimics Twitter functionality, encrypts Tweets, provides access control, and allows users to follow hashtag via oblivious matching and therefore conceals interests. Twister [14] mimics a pseudonymous Twitter. Senders register a pseudonym, their nickname which is bound to a key pair via the Bitcoin [15] protocol. Pseudonyms enforce non-repudiation and cause all actions from the sender to be linkable. Senders publish signed tweets via the BitTorrent [16] protocol. Receivers anonymously follow senders and hashtags. Twitsper [17], [18] uses a second central server besides Twitter to realize private group communication. Senders notify receivers via direct Twitter messages, the receivers then obtain further information from the Twitsper server. Other approaches [19] focus on the anonymization of tweets and metadata using techniques based upon k-anonymity [20]. However, such approaches are only applicable for bulk data release, i.e., after tweets have already been posted.

In summary, these applications focus on confidentiality and provide at best pseudonymity, but no anonymity.

c) Privacy-preserving Group Communication: Privacy-preserving Instant Messaging (IM) applications mimic behavior similar to MbP, i.e., send messages to groups as compared to followers and hashtags.

With Threema [21], users communicate confidentially but with a phone number as identity. Threema stores encrypted and signed messages on a central server. Group membership is established via a list of at most 20 participants. Threema establishes authenticity via an out-of-band verification. TextSecure from Open WhisperSystems [22] uses a modified version of OTR for confidential messaging. TextSecure uses a central server to distribute messages and a phone number as optional identity, too. FireChat [23] uses ad-hoc communication to exchange messages. FireChat only distinguishes between global and a locality-based group and thus misses the flexibility of self-defined groups and hashtags.

All of the presented products suffer from one or more drawbacks: dependence on additional central servers, lack of

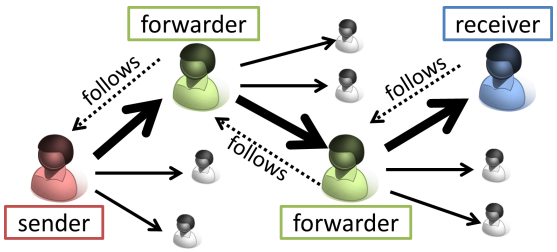

Fig. 1. Twitter social graph with follow relation and tweet flow. The bold arrows represent the overlay for a hashtag on top of the social graph underlay.

scalable group communication, and lack of build-in anonymity protection besides confidentiality.

III. Twitterize

As the related work does not provide anonymity in microblogging scenarios, we designed Twitterize to overcome this shortcoming. Twitterize builds upon the MbP Twitter and encrypt tweets to achieve confidentiality, forwards and mixes tweets via several participants to achieve anonymity of sender and receivers, and is implemented as an Android application to meet the usability requirement.

For anonymity, we establish overlays networks that connect sender and receivers, but do not require ad-hoc or P2P communication. Twitterize creates one such overlay per hashtag. Each overlay contains additional nodes called *forwarders* that are not interested in the hashtag, i.e., they are neither sender nor receiver for that hashtag, to increase the size of the anonymity set.

We use the Twitter social graph as underlay network. That is, Twitter users represent *nodes* and followers represent neighbors with *directed edges* for the flow of tweets. Each Android application takes the role of a node, maintains the local overlay state per hashtag, and exchanges messages via tweets and re-tweets. Therefore, Twitterize can maintain normal Twitter functionality while also providing the option to tweet anonymously and confidentially at the same time using solely tweets.

Figure 1 depicts an example of a Twitter social graph. Users follow each other (dotted arrows) and thus receive status update tweets (solid arrows) from the users they follow. Twitterize exploits this social graph as an underlay and constructs hashtag overlays on top. In this example, the two forwarders in the center forward tweets from sender to receiver. Therefore, sender and receiver remain decoupled and no single forwarder can link sender and receiver together.

In Section III-A, we describe the theory of the overlay construction. We transfer this theory in Section III-B to Twitter and Android.

A. Anonymous Overlay Construction

We use the paradigm publish-subscribe (pub-sub) with advertisements [24] to construct overlay networks for hashtags following our prior work [3]. We briefly summarize this work and adapt it to a tweet-based underlay in the following paragraphs *key generation and secrets*, *flooding of advertisements*, and *hashtag discovery and subscriptions*.

Summarizing this approach, users first exchange secrets via an out of band channel. We implement two variations of this out of band channel (cf. Section III-B). Then, senders (*publishers*) flood the the social graph (*underlay*) with routing information about hashtags. Upon receiving these routing information, receivers (*subscribers*) subscribe to hashtags and thereby establish fast and efficient routing paths (*overlay*). Afterwards, publishers can send tweets anonymously and confidentially to subscribers. Throughout the following paragraphs we describe what secrets we use, how they protect confidentiality, how knowledge about hashtags is flooded through the underlay, how users subscribe to hashtags, and how tweets are distributed in the system.

a) Key generation and Secrets: The first publisher creating a new confidential hashtag has to create key material. We use a symmetric key K_x for hashtag x to encrypt and decrypt information. An optional asymmetric key pair can be used to sign information and thus also ensures integrity and authenticity. The publisher uses K_x to encrypt and hash the hashtag and thus creates as pseudonym: $t_x := H(\{x\}_{K_x})$. Now the publisher can annotate all tweets that relate to x with t_x without revealing x itself. The tuple (x, K_x) constitutes the secret and has to be shared with subscribers of hashtag x.

b) Flooding of Advertisements: To distribute knowledge about a hashtag x, the publisher uses our private flooding mechanism [3]. For that, the publisher creates an advertisement tweet (t_x, h) and floods the underlay with it. Here, h is an element of a hash chain: $h_1 = nonce, h_2 = H(h_1), \ldots, h_{n+1} = H(h_n)$, where $H()$ denotes a hash function. The hash chain serves two purposes. First, it acts as a transaction pseudonym for flooding processes as each publisher selects its own nonce. Second, each node increments the hash chain element by one when forwarding and thus enables nodes to determine the shortest path and avoid routing loops. As a result of the flooding process, every node establishes a table containing tipples (t_x, h, v) where v is the last sender or forwarder of the advertisement tweet. This flooding scheme treats hashtags confidential and protects anonymity as neither global identifiers nor pseudonym are required.

Figure 2 illustrates this process with Twitter. A publisher p tweets the advertisement as a status update to her timeline (step 1). As forwarder f follows p, the advertisement tweet will automatically show up on f's timeline (step 2). Forwarder f the forwards this advertisement by incrementing the hash element and posting another status update to her timeline (step 3). Finally, subscriber s receives the advertisement tweets as s follows f.

c) Hashtag Discovery and Subscription: Each participant w in possession of the secret (x, K_x) compares elements (t_y, h, v) from its triple table with t_x and subscribes in case t_y and t_x are equal. To do so, w sends a direct tweet (t_y) to the sender of the advertisement tweet v. A direct tweet in Twitter starts with $@user_v$ and will only be visible to $user_v$.

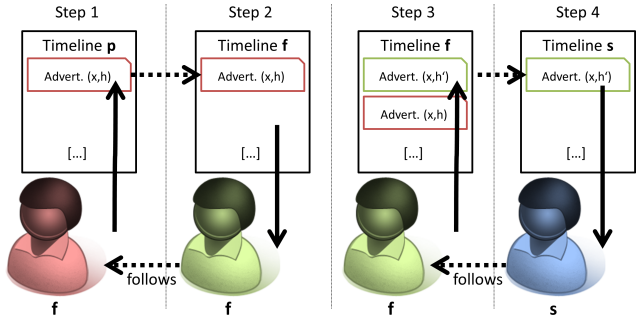

Fig. 2. Advertisement tweets propagate in Twitter over timelines to followers. If an advertisement is new for a follower, the follower increments the hash element and posts it to her own timeline. Hence, the follower becomes a forwarder.

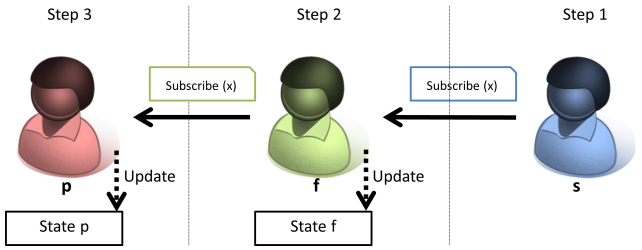

Fig. 3. Subscription tweets propagate via direct tweets in Twitter. users have to main a routing table as state outside of Twitter for the Twitterize protocol.

Node v remembers this message as tuple (t_y, w) in a second table and forwards the subscription towards the origin of the advertisement.

Figure 3 illustrates this process from right to left. The subscriber s sends a subscription via a direct tweet to f (step 1). Forwarder f then updates its internal state, the routing table, and forwards the subscription to p (step 2). Finally publisher p receives the subscription and updates her internal state as well. As we use direct tweets here, the social graph may remain directed.

d) Summary: The flooding of advertisements in combination with hashtag discovery and subscriptions forms an overlay network for each hashtag. The two routing tables at each node describe the flow of tweets (notifications). The key material ensures confidentiality of these tweets. Furthermore, the mechanisms introduced so far do not require global identifiers. Thus every participant is limited to its local view on the message flow an does not learn origin and destinations, similar to MIX networks (cf. Section II).

B. Proof of Concept

We implemented Twitterize as a proof of concept (POC) using Android. We first provide an overview of the majors components. Next, we elaborate on the functionality using a top-down approach: we start with the User Interface (UI), cover how overlay management functions get translated into tweets (*serialization into tweets*), and finally demonstrate the whole *notification flow*. The Twitterize architecture is comprised of the following components: *Twitter services*, *data storage*, *UI*, *cryptography* and *encoding*.

Twitter services are based on the twitter4j library [25] and provide functionality to log in, retrieve user information and send or receive tweets. The services periodically obtain new tweets from Twitter. Data storage consists of an Android SQLite DB and shared preferences. This Database (DB) stores the tweets, subscriptions, cryptographic keys and notifications sent by the application itself. The shared preferences maintain the runtime state of the Twitterize protocol and application.

The UI displays this data in form of timelines: A home timeline and a new Twitterize timeline for each hashtag. Additionally, users can customize the behavior of Twitterize via shared preferences. These offer options such as automatic synchronization, the amount and frequency of tweets to pull during synchronization, and the time frame for protocol recovery, e.g., re-processing of old tweets in case data has been lost.

a) Tweet Serialization: To protect confidentiality, Twitterize encrypts messages sent via Twitter service with AES 128bit in Cypher Block Chaining mode. For that, Twitterize maintains the key K_x for every hashtag x.

Twitter services are bound by Twitter restrictions, e.g., no binary data and only 140 characters per message. Thus, the ciphertext of notifications, the t_x of an advertisement and the hash chain element h must be transformed into a non-binary representation. Furthermore, data structures like JSON are too verbose to structure messages. Therefore we use Base64 encoding that converts ciphertexts and hashes into UTF-8 representable strings. We use rarely utilized UTF-8 symbols to structure messages and distinguish message types.

b) Notification Flow: The combined information flow, covering application components, serialization into Tweets, and Twitter itself is depicted in Figure 4. For this example, we assume that publisher p created hashtag x by generating key material K_x. The publisher then invited subscribers by sharing K_x via QR code and NFC. Furthermore, p announced x via an advertisement, and thus Twitterize constructed an overlay for x.

In Figure 4, p creates a notification N for hashtag x encrypted with K_x and tweets it to her home timeline. Forwarder f is a follower of p and therefore obtains N from her home timeline via the *PullTweetTask* class. The task identifies N as a Notification and checks if it has been seen and forwarded before. It then passes N on to the *SendTweetTask* class which re-tweets N to f's home timeline and marks it as forwarded in the DB. As f does not subscribe to x, f does not gain any knowledge about N besides length, immediate predecessor, and successors.

Subscriber s pulls N from her home timeline. As s subscribed to x, her PullTweetTask decrypts N with the key K_x obtained from p and store it inside the DB. The plaintext of N is now shown in both the home timeline and the timeline of hashtag x. Figure 5 shows some screenshots of the final POC.

This POC allows users to read and tweet regular as well as privacy-preserving tweets. Users can quickly create new hashtags and invite new users via near field communication

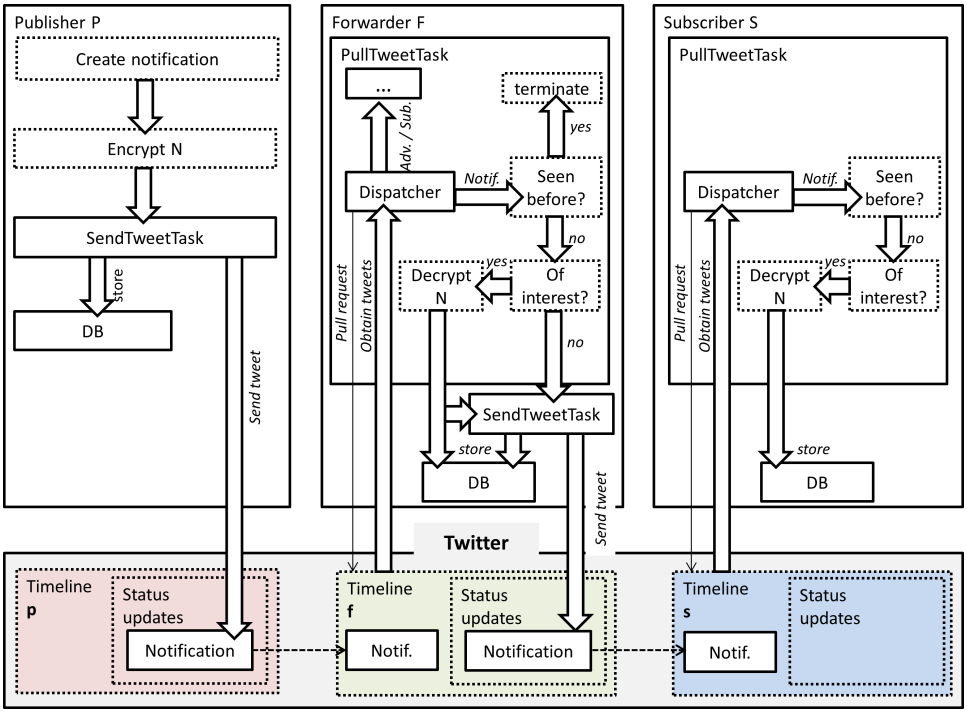

Fig. 4. Information flow of a notification. A publisher creates an encrypted tweet (top left) and posts it as status update to his home timeline (bottom). The forwarder (top center) follows the publisher and thus received this tweet, processes it, and posts it to its timeline (bottom center) as status update so that the subscriber (top right) will receive it.

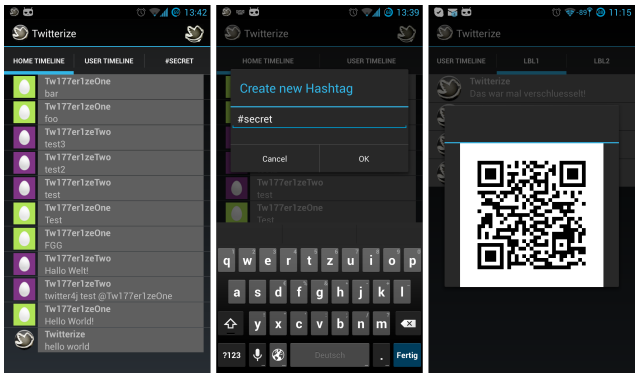

Fig. 5. Screenshots of Twitterize. Left: home timeline with normal tweets and one privacy-preserving tweet at the bottom. Middle: minimal user interface to create new hashtags. All protocol functions remain in the background. Right: One out-of-band channel to share secrets for a hashtag.

(NFC) as well as QR codes. Hence, only few seconds of physical closeness of participants are necessary. Mandatory user confirmation prevent accidental and malicious joining of hashtags. Moreover, QR codes can be used via a secure out-of-band channels to add new users as well.

IV. Evaluation

In this section, we discuss how the Twitterize meets the requirements, outline implications of our platforms decisions, and analyze the overhead of Twitterize.

c) Requirement Discussion: The symmetric cryptography used in Twitterize is assumed to be secure. Thus, Twitterize protects confidentiality if tweets in transmission as well.

Finally, we constructed the security of Twitterize to be easy to understand for the user. For that, every tweet carries iconographic and textual representations of the applied privacy protections. Furthermore, we designed the key management in an easy and familiar manner by leveraging NFC gestures and QR codes.

d) Empirical Usage: To analyze the overhead (cf. Section II-A) that Twitterize imposes over normal smartphone operations, we measured CPU load and battery drain. For that, we used a LG Nexus 5 smartphone running Android OS version 4.4.3 with the *Trepn Profiler* from Qualcomm[3] and compared measurements for Twitterize with measurements for Twitter version 5.13.1 as reference application. We set the Twitterize pull rate to $r = 1/60$ (once per minute), measured over 15 minutes in wake state with no user interaction, and obtained 8850 samples each.

Twitter caused a mean normalized CPU load of 7.739% (standard deviation of 3.906%) whereas Twitterize caused a load of 8.454% (standard deviation of 5.142%). Moreover, Twitter caused a mean power consumption of $2,117mW$ (standard deviation of $571mW$) compared to Twitterize with $2,216mW$ (standard deviation of $622mW$). Thus, Twitterize caused only moderate overhead compared with Twitter.

[3]https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler

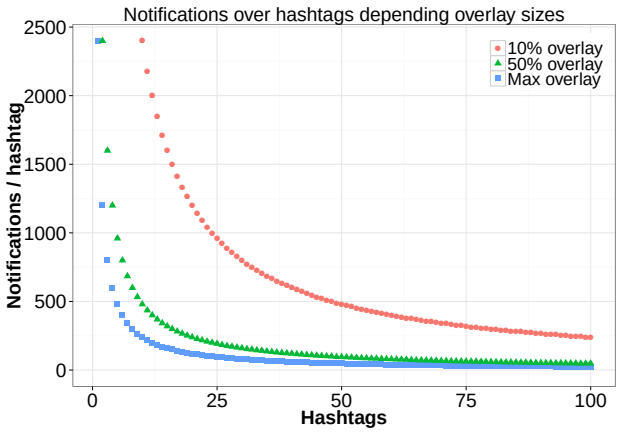

Fig. 6. Tweets per hastag over the number of hashtags in the system. The system degrades quickly with increasing overlay sizes, e.g., max overlay size (blue squares). However, with an overlay membership rate of 10% (red circles), the system can still handle one tweet per hour for $1,000$ hashtags.

The advertisement flooding process introduced in Section III-A requires even non-interested users to main some state information as forwarders and potential forwarders. However, our implementation requires to store 48*Byte* per hashtag. Hence, every participant can manage more than $21,000$ hashtags per megabyte of storage space.

The routing through a hashtag overlay causes delay of messages compared to normal Twitter usage. According to [26], several samples of the Twitter social graph indicated a degree of separation of up to 4.71 users in 2012. Using this parameter for the Twitter underlay and pulling once per minute, we can calculate the average delay $d_{avg} = 4.71/(2\times r)$ for a tweet with $d_{avg} = 142$ seconds ($d_{max} = 300$ seconds upper boundary) until delivery to all receivers.

In summary, Twitterize causes only mild processing overhead and negligible storage on Android systems. However, secure tweets get delayed by several minutes.

e) Twitter and Android Limitations: We share some of the shortcomings we encountered during prototype construction with both the Android platform and the Twitter API. Twitter applies several usage constraints. General limitations are $2,400$ tweets/day, 250 direct messages/day, access to only the $3,200$ latest tweets, users can only follow $2,000$ other users, and the API allows only 15 requests per 15 minute window.

The limit of 250 direct messages per day is insufficient to run the whole protocol. This leaves us with the $2,400$ tweets/day which have to be shared between regular and Twitterize tweets. Given that notifications will be re-tweeted this greatly limits the frequency of Twitterize tweets or the amount of participants. Figure 6 visualizes this limitation.

The retrieval limit of $3,200$ tweets allows us to limit the processing of message, e.g., one day into the past, since we also cannot re-tweet more than $2,400$ notifications at once. This period is still sufficient to compensate for short offline periods.

For our protocol to work properly, we need the participants to be always online. This drawback is mitigated by the use of an Android application compared to a web page in a browser. Our application runs continuously in the background of an Android device connected to the Internet, whereas browser windows might be closed at some point. Our empirical evaluation showed that the battery drain of Twitterize is sufficiently low. The use of Twitter as underlay still offers potential redundant connections to propagate notifications in the presence of node failure.

V. Conclusion

In this paper, we motivate the need for anonymity besides confidentiality to protect privacy in Micro-blogging platforms (MbPs). Our discussion of related work and systems reveals that most approaches do not ensure anonymity or depend upon an additional infrastructure. Hence, we adapt our construction method for privacy-preserving P2P overlay networks on top of MbPs. We evaluate this construction via an Android POC implementation *Twitterize* for the Twitter MbP. This POC neither depends on additional infrastructure nor P2P communication but tweets. Furthermore, the POC integrates normal tweets with privacy-preserving tweets.

Future work will address overlay robustness as future requirements, for instance quick detection and recovery from failed nodes, as well as increase efficiency of the tweet serialization.

Acknowledgments

Authors would like to thank Daniel Joaquin Gonzalez Nothnagel, Patrick Müller, Tobias Hamann, and Jonas Mantel for supporting the POC implementation.

References

[1] A. Java, X. Song, T. Finin, and B. L. Tseng, "Why we twitter: An analysis of a microblogging community," in *WebKDD/SNA-KDD*, ser. Lecture Notes in Computer Science, H. Zhang, M. Spiliopoulou, B. Mobasher, C. L. Giles, A. McCallum, O. Nasraoui, J. Srivastava, and J. Yen, Eds., vol. 5439. Springer, 2007, pp. 118–138.

[2] G. Lotan, E. Graeff, M. Ananny, D. Gaffney, I. Pearce *et al.*, "The arab spring— the revolutions were tweeted: Information flows during the 2011 tunisian and egyptian revolutions," *International Journal of Communication*, vol. 5, p. 31, 2011.

[3] J. Daubert, M. Fischer, S. Schiffner, and M. Mühlhäuser, "Distributed and anonymous publish-subscribe," in *Network and System Security*, ser. Lecture Notes in Computer Science, J. Lopez, X. Huang, and R. Sandhu, Eds. Springer Berlin Heidelberg, 2013, vol. 7873, pp. 685–691.

[4] A. Pfitzmann and M. Köhntopp, "Anonymity, unobservability, and pseudonymity - a proposal for terminology," in *Workshop on Design Issues in Anonymity and Unobservability*, ser. Lecture Notes in Computer Science, H. Federrath, Ed., vol. 2009. Springer, 2000, pp. 1–9.

[5] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–88, 1981.

[6] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in *USENIX Security Symposium*. USENIX, 2004, pp. 303–320.

[7] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, pp. 66–92, 1998.

[8] M. Wright, M. Adler, B. N. Levine, and C. Shields, "An analysis of the degradation of anonymous protocols," in *NDSS*. The Internet Society, 2002.

[9] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. F. Syverson, "Users get routed: traffic correlation on tor by realistic adversaries," in *ACM Conference on Computer and Communications Security*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 337–348.

[10] N. Borisov, I. Goldberg, and E. A. Brewer, "Off-the-record communication, or, why not to use pgp," in *WPES*, V. Atluri, P. F. Syverson, and S. D. C. di Vimercati, Eds. ACM, 2004, pp. 77–84.

[11] I. Goldberg, B. Ustaoglu, M. V. Gundy, and H. Chen, "Multi-party off-the-record messaging," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 358–368.

[12] M. Senftleben, M. Bucicoiu, E. Tews, F. Armknecht, S. Katzenbeisser, and A.-R. Sadeghi, "Mop-2-mop – mobile private microblogging," in *Financial Cryptography and Data Security*, 2014.

[13] E. D. Cristofaro, C. Soriente, G. Tsudik, and A. Williams, "Tweeting with hummingbird: Privacy in large-scale micro-blogging osns," *IEEE Data Eng. Bull.*, vol. 35, no. 4, pp. 93–100, 2012.

[14] M. Freitas, "twister peer-to-peer microblogging," sep 2014. [Online]. Available: http://twister.net.co

[15] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system (whitepaper)," sep 2014. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[16] B. Cohen, "The bittorrent protocol specification," sep 2014. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html

[17] I. Singh, M. Butkiewicz, H. V. Madhyastha, S. V. Krishnamurthy, and S. Addepalli, "Twitsper: Tweeting privately," *IEEE Security & Privacy*, vol. 11, no. 3, pp. 46–50, 2013.

[18] I. Singh, M. Butkiewicz, H. Madhyastha, S. Krishnamurthy, and S. Addepalli, "Enabling private conversations on twitter," in *ACSAC*, R. H. Zakon, Ed. ACM, 2012, pp. 409–418.

[19] A. Singh, D. Bansal, and S. Sofat, "An approach of privacy preserving based publishing in twitter," in *SIN 2014*. International World Wide Web Conferences Steering Committee / ACM, 2014.

[20] L. Sweeney, "k-Anonymity: A model for protecting privacy," *Ieee Security And Privacy*, vol. 10, no. 5, pp. 1–14, 2002.

[21] Threema, "Threema messenger," sep 2014. [Online]. Available: https://threema.ch/

[22] M. Marlinspike, R. Orbits, M. Corallo, C. C. Moran, F. Jacobs, and T. Reinhard, "Open whisper systems textsecure," sep 2014. [Online]. Available: https://whispersystems.org/

[23] M. Benoliel, S. Shalunov, and G. Hazel, "Opengarden firechat," sep 2014. [Online]. Available: http://opengarden.com

[24] P. T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[25] Y. Yamamoto, "Twitter4j java library for the twitter api," sep 2014. [Online]. Available: http://twitter4j.org/en/index.html

[26] M. Watanabe and T. Suzumura, "How social network is evolving?: a preliminary study on billion-scale twitter network," in *WWW (Companion Volume)*, L. Carr, A. H. F. Laender, B. F. Lóscio, I. King, M. Fontoura, D. Vrandecic, L. Aroyo, J. P. M. de Oliveira, F. Lima, and E. Wilde, Eds. International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 531–534.