Resource Efficient Privacy Preservation of Online Social Media Conversations

Indrajeet Singh[1], Masoud Akhoondi[1], Mustafa Y. Arslan[2], Harsha V. Madhyastha[3], and Srikanth V. Krishnamurthy[1(✉)]

[1] UC Riverside, Riverside, USA
{singhi,makho001,krish}@cs.ucr.edu
[2] NEC Labs, 4 Independence Way, Suite 200, Princeton, NJ, USA
marslan@nec-labs.com
[3] University of Michigan, Ann Arbor, USA
harshavm@umich.edu

Abstract. On today's online social networks (OSNs), users need to reveal their content and their sharing patterns to a central provider. Though there are proposals for decentralized OSNs to protect user privacy, they have paid scant attention to optimizing the cost borne by users or hiding their sharing patterns. In this paper, we present *Hermes*, a decentralized OSN architecture, designed explicitly with the goal of hiding sharing patterns while minimizing users' costs. In doing so, *Hermes* tackles three key challenges: 1) it enables timely and consistent sharing of content, 2) it guarantees the confidentiality of posted private content, and 3) it hides sharing patterns from untrusted cloud service providers and users outside a private group. With extensive analyses of *Hermes* using traces of shared content on Facebook, we estimate that the cost borne per user will be less than $5 per month for over 90% of users. Our prototype implementation of *Hermes* demonstrates that it only adds minimal overhead to content sharing.

1 Introduction

Today, leakage of information from OSN servers [5,6], coupled with the need for OSN providers to mine user data (e.g., for targeted advertisements), have concerned users [12]. While posting encrypted data on OSNs [15,23] can work in theory, it compromises the profit motives of an OSN if done at scale. Alternatively, one could share private content with OSN friends by storing data outside the OSN provider's control. Prior works that follow this approach either store private content in the cloud [4,13,29] or across client machines [24,27]. The former simply leaks private information to the cloud providers in lieu of the OSN providers, and also increases user costs. The viability of an approach based on the latter depends on the availability of consistent access to client machines.

Our Contributions: In this paper, we design a decentralized OSN architecture, *Hermes*, with cost-effective privacy in mind. *Hermes* seeks to ensure that both the content shared by a user and her sharing habits are kept private from both the

© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2015
B. Thuraisingham et al. (Eds.): SecureComm 2015, LNICST 164, pp. 233–255, 2015.
DOI: 10.1007/978-3-319-28865-9_13

OSN provider and undesired friends. In doing so, *Hermes* seeks to (i) minimize the costs borne by users, and (ii) preserve the interactive and chronologically consistent conversational structure offered by a centralized OSN.

Hermes uses three key techniques to meet these goals. First, it judiciously combines the use of compute and storage resources in the cloud to bootstrap conversations associated with newly shared content. This also supports the high availability of the content. Second, it employs a novel cost-effective message propagation mechanism to enable dissemination of comments in a timely and consistent manner. It identifies and purges (from cloud storage) content that has been accessed by all intended recipients. Lastly, but most importantly, *Hermes* carefully orchestrates how fake postings are included in order to hide sharing patterns from the untrusted cloud providers used to store and propagate content, while minimizing the additional costs incurred in doing so. A key feature of *Hermes* is its flexibility in deployment; it can either be implemented as a stand alone distributed OSN or as an add-on to today's OSNs like Facebook (while maintaining the decentralized nature of content sharing). To summarize, our contributions are:

Design of *Hermes*: As our primary contribution, we design *Hermes*. It utilizes extremely small amounts of storage, bandwidth, and computing on the cloud to facilitate real-time, consistent and anonymous exchange of private content. Importantly, *Hermes* ensures that cloud providers cannot discover the users involved in private conversations and is robust to the intersection attack [18].

Analyzing OSN Data to Determine Resource Requirements: Based on 1.8 million posts crawled from Facebook, we 1) perform an analysis to determine key parameters for implementing *Hermes*, and 2) conduct realistic simulations to show that (a) *Hermes* effectively anonymizes users' sharing patterns and (b) *Hermes*'s use of cloud resources is low enough to facilitate its practical deployment. Our analysis suggests that, for 90% of users, *Hermes* would typically require 1) cloud storage of much less than 5 MB, and 2) a compute instance on the cloud that is active for roughly 4 days every month. This corresponds to a monthly cost of less than $5 per user. With this budget, *Hermes* ensures that cloud service providers are unable to guess the members or the group size of any private conversation. If the cloud provider attempts to randomly guess the group members, it is correct less than 15% of the time.

Implementation and Evaluation: We implement a prototype of *Hermes* as a rudimentary add-on to Facebook. Our evaluations show that *Hermes* incurs low cost, and the user experience, in terms of delays, is similar to that with Facebook.

Scope: The privacy preserving features of *Hermes* can be used in conjunction with a centralized component that can be used for posts that are not intended to be private. In fact, our prototype of *Hermes* as an add on to Facebook achieves just that; private posts are directed to *Hermes* while other content is shared in the traditional way. We wish to also point out that we do not explicitly consider mobile users; however, *Hermes* can be used in such contexts, and across multiple devices.

2 Related Work

Improving Privacy in OSNs: Several systems propose to post encrypted content on OSNs to protect privacy (e.g., [15,16,21]). However, encryption precludes OSN providers from interpreting posted content and/or hides users' social connections from OSNs. These are not in the commercial interests of OSN providers, who may thus disallow such postings. *Hermes* does not post any encrypted content on an OSN; it uses either cloud storage or users' personal devices to do so. Further, it does not use a centralized OSN framework to inform users of new content; doing so also informs the OSN provider of the specifics of ongoing conversations.

Distributed OSNs: Other efforts propose storing private shared data on devices other than OSN servers [4,24,27,29]. However, unlike *Hermes*, they either expose user sharing patterns to cloud providers [4,29] or degrade user experience in terms of timely and consistent sharing. Systems that store private data in the cloud do not control either storage or bandwidth costs which increase over time as the volume of shared data grows. While other systems store the data on users' personal machines [24,27] to reduce costs, the low availability of these machines (they may be turned off when not used) reduces the timeliness of conversations and compromises data consistency. *Hermes* combines resources on cloud services (within limit) with that on users' personal machines to support cost-effective sharing that is held privy from cloud providers.

Priv.io [33] is a new decentralized OSN that aims to minimize the cost incurred for facilitating private content sharing. However, Priv.io critically relies on support for advanced messaging APIs from cloud services, which restricts the generality of Priv.io's architecture. In contrast, *Hermes* only requires cloud storage services to offer a minimal PUT, GET, DELETE interface. Most importantly, due to Priv.io's reliance on messaging APIs offered by cloud services, unlike *Hermes*, it does not attempt to hide sharing patterns (i.e., whom does a user share data with) from cloud providers.

Other Related Work: Efforts [21,22,30] that secure the data stored on untrusted servers or the cloud do not try to account for OSN-specific characteristics (e.g., hiding content sharing patterns). Unlike *Hermes*, these solutions would either significantly increase cost or degrade timeliness. Moreover, *Hermes* enables anonymity in OSN conversations without requiring all members of a conversation to be simultaneously online.

3 Goals and Threat Model

Goals and Challenges: Our over-arching goal is to design a decentralized, private OSN architecture. In doing so, we have the following three objectives.

- *High availability, timeliness, and consistency:* First, we seek to preserve the desirable properties enabled by a central provider. Specifically, (a) users should always be able to access content shared with them, (b) content shared by a

user should be received by the intended recipients in a timely manner, so as to preserve the interactive comment "threads" associated with content shared on OSNs, and, (c) all users involved in a conversation should receive comments in the same causally consistent order. How do we preserve these desirable properties despite the fact that content is stored in a decentralized manner in *Hermes*?

- *Protect the privacy of content and sharing patterns:* While *Hermes* lacks any central OSN provider, cloud services used to store and disseminate content may be able to monitor conversations. How do we preserve the privacy of shared content from cloud providers and prevent them from discovering the participants in any conversation?
- *Minimize cost:* Finally, we seek to minimize the storage, bandwidth, and compute costs incurred by users in *Hermes*'s use of cloud services. This is made particularly challenging due to the previous two goals. For example, one could enable timely dissemination of comments if every user were to maintain her own compute instance in the cloud at all times. Similarly, the members of any particular conversation can be hidden from cloud providers by having all users constantly exchange fake comments with each other. However, such measures will result in high cost.

Threat Model: We assume that all service providers (of cloud services or of a centralized OSN) preserve the integrity and availability of the data that users store on them. This may be either in fear of bad publicity or because users pay for the service. However, we assume that all service providers may benefit from inferring information associated with private conversations. Thus, we treat all service providers as "curious but honest", as in [33]. Moreover, if cloud providers discover the members of private conversations, this information may leak. Therefore, we seek to ensure that, when a group of users are involved in a private conversation using *Hermes*, no one outside the group learns either the size or membership of this group. Here, we assume that cloud providers can perform network-level traffic analysis (e.g., a provider can map the IP addresses from which it is accessed, to user identities). The use of anonymity networks such as Tor [19] would not scale to meet the traffic demands of a large-scale OSN. Lastly, ensuring the privacy of a users' conversation group via fake messages (as in *Hermes*) requires that the user has a sufficiently large set of friends; if a user has very few friends (e.g, < 5), preserving the anonymity of a private conversation group is hard. We assume that users have friends of the order of hundreds, as is typical on OSNs [3]; however, we assume the sizes of private conversation groups to be much smaller.

4 Hermes Architecture

In this section, we describe the *Hermes* architecture with a simple running example.

Consider an OSN user (Alice), who wishes to share some content (say a photo) meant only for her friends Bob and Chloe. To ensure that neither the private content nor the intended recipients are exposed to anyone other than the

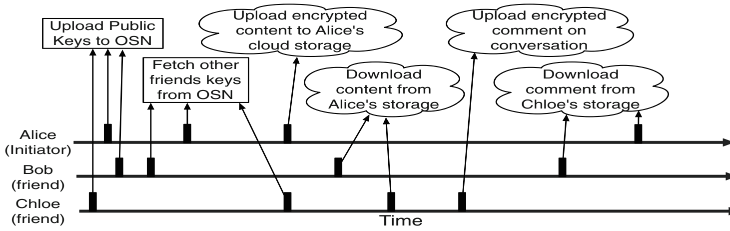


Fig. 1. Illustration of conversation timeline.

intended recipients, Alice encrypts the photo with an appropriate key (known only to Bob and Chloe) and shares it using resources in the cloud. There are four main issues that we need to address to enable this: 1) how do Bob and Chloe discover this content *and* the associated key to decrypt it?, 2) how can comments on the content, posted by Alice, Bob, and Chloe, be disseminated in a timely manner?, 3) how do we prevent the cloud provider from inferring the members of this private exchange?, and 4) how to minimize costs incurred by Alice, Bob, and Chloe? We next describe how *Hermes* tackles these questions.

Sharing New Content: As shown in Fig. 1, every user (including Alice) first posts her public key component to enable an ECDH key exchange (details of ECDH can be found in [25]) on her OSN profile[1], which is visible to all of her friends. Any user can thus, fetch the public key components from her friends' profiles and derive pairwise keys with any of her friends.

To share a photo, Alice's *Hermes* client chooses a new *group key* and creates two encrypted copies (using a cipher such as AES) of this key, one copy encrypted using her pairwise key with Bob and the other using her pairwise key with Chloe. Alice's client then stores these encrypted group key copies in Alice's cloud store. The client also puts the photo, encrypted with the group key (again using AES), in her cloud storage.

Bob's and Chloe's *Hermes* clients periodically check Alice's cloud store for new content shared with them. When new content exists, they fetch their respective encrypted group key copies from Alice's store (the process is discussed later) and extract it using their respective pairwise keys with Alice. Bob's and Chloe's clients then store the extracted group key locally on their personal devices. The clients can fetch and decrypt the photo using this group key.

Enabling OSN-Like Conversations: We next describe how *Hermes* enables OSN like conversations with low cost.

Disseminating Comments: After Bob and Chloe discover Alice's photo, the three of them may post comments on it. Our goal is to ensure that these comments are disseminated in a timely and consistent manner, as is the case with a centralized OSN. If all users involved in the conversation are always online, whenever a user posts a comment, that user's client can establish secure

[1] This could be on her favorite OSN or *Hermes*'s servers depending on the implementation.

connections with the clients of the other members of the conversation and inform them of the new comment. However, in practice, Alice, Bob, and Chloe may come online at different times. Thus, there has to be a common arbitrator that enables a user to discover comments posted when she is not online and facilitates the chronological ordering of posted comments.

For this, we propose that the user who initiates the conversation (Alice) uses a computing instance in the cloud to act on her behalf as the arbitrator. Today, there are many such online computation resources available (e.g., Google App Engine [7], Heroku [9], and Amazon EC2 [1]). Alice's instance acts as a proxy for her.

Reducing Compute Instance Costs for Alice: However, keeping the compute instance active at all times is not cost-effective for Alice. Thus, by default, Alice's *Hermes* client terminates her instance following a preset period after Alice has shared any new content (discussed later in Section 6). However, there may be users who come online much after the instance has been terminated. To deal with such cases, *Hermes* uses log files called *ufiles* (update files). Every user maintains a *ufile* in her cloud store for each friend; these *ufiles* are created and the location of the *ufiles* are exchanged between friends either when a user installs the *Hermes* client or when the user adds a friend. Thereafter, whenever a user (Alice) posts a new piece of content relevant to a specific friend (Chloe), Alice's *Hermes* client adds an entry to the *ufile* for Chloe. In all subsequent discussion, for the purposes of clarity, we only provide a high level description of how *ufiles* are used and defer a detailed description to an appendix.

If Bob comes online after Alice's compute instance has been terminated, his client retrieves her *ufile* for him and locates any new updates. This allows Bob to retrieve any content or comments shared by Alice. His client then indicates that the content has been retrieved in his own *ufile* for Alice. Upon checking this entry when Alice comes online, her *Hermes* client deletes the original entry in her *ufile* for Bob.

Note that *ufiles* also enable a user to discover comments without waiting for the initiator of a conversation to come online. For example, if Bob is also Chloe's friend, Bob's *ufile* for Chloe will indicate that he has commented on Alice's photo. Chloe can thus retrieve the comment and associate it with the original photo received from Alice (based on an associated conversation ID).

Ensuring Consistency of Comments with ufiles: The above framework allows a user who comes online after the instance is terminated to retrieve the object and reconstruct the conversation associated with it (i.e., put the comments in chronological order using vector timestamps [26]) as long as all the group members are his friends. However, if a group member (say Chloe) is not Bob's friend, Chloe is unable to read Bob's *ufiles*; in fact, such a file for Chloe will not exist in Bob's cloud storage, since *ufiles* are only maintained for friends. This violates the structure of an OSN conversation.

To deal with such cases, Alice relays the locations of the comments associated with her content via her own *ufiles* for each member of the conversation (who

are her friends since she initially shared the content with them). Since there may be delays in relaying these locations (in rare cases where multiple users come online much after the compute instance is terminated), there may be temporary loss in the chronological consistency for a user who comes online at a late stage. There is an inherent trade-off here; the longer Alice's compute instance is active, the less likely is that there is such a loss in consistency. However, this will incur a higher cost.

Reducing Storage Costs: Finally, Alice cannot store her photo (or for that matter, Bob cannot store his comment) on the cloud forever. This would result in a monotonic growth in the consumed storage and thus, the associated cost. Instead, with *Hermes*, content is removed from cloud storage after a certain time (the duration can be set by Alice, but we discuss what might be appropriate in Section 6). A simple way of ensuring that all group members have seen the content before it is purged is for Alice to check if they have indicated this to be the case in their $ufiles$ for her. If a user (say Bob) comes online after a prolonged absence (much after when the content was removed from the cloud), he may still learn of its existence via Alice's $ufile$ meant for him. Via his own $ufile$ for Alice, Bob's client then requests Alice for the purged content. When Alice comes online next, her client then copies the requested content back on to the cloud. In fact, Bob can request the purged content from any or all of the group members of that conversation (information on the group can be embedded as metadata in the encrypted content) to restore the content on the cloud for him. Once a group member (say Chloe) restores the content, Bob's $ufiles$ can be updated to indicate that the content is no longer needed from other members.

This process increases the complexity of *Hermes*'s design, and thus, is not currently implemented in our prototype; however, as we show in Section 6, such cases are rare if one looks at typical content sharing on Facebook. Here, we also point out that *Hermes* enables users to access their content from multiple devices; however, we omit the details of how this is made possible due to space constraints.

5 Hiding Users' Sharing Patterns

Next, we discuss how *Hermes* ensures that cloud providers cannot determine any of the following: a) "when" a private conversation is occurring, b) the group size of any given conversation, and c) the individual members taking part in that conversation.

5.1 Hiding the Membership Information within Each Private Conversation

First, let us consider a single private conversation initiated by Alice. Our goal here is to ensure that the identities of the members of this private group and the size of the group are not exposed to anyone outside the group.

Strawman Approach: To hide the group members in a given conversation initiated by Alice, one simple approach is to make $ufiles$ indistinguishable across

all of Alice's friends. Whenever Alice's *Hermes* client needs to insert an entry into the *ufile* for a particular friend, it can also insert dummy entries into the *ufiles* for all of Alice's remaining friends; the entries in the *ufile* for any particular friend are encrypted with the shared pairwise key between Alice and that friend, thus preventing the cloud provider from inferring which entries are fake. Thus, based on the writes to and reads from the *ufiles* in Alice's cloud storage, the cloud provider will not be able to determine which subset of Alice's friends are involved in ongoing private conversations.

However, this simple approach has two limitations. First, it results in high storage, bandwidth, and operational query costs for Alice, because a large number of fake entries will need to be stored by Alice and accessed by Alice's friends. Second, the cloud provider may still be able to infer the members of Alice's private conversation by observing which of Alice's friends insert updates into the *ufiles* in their own storage space; group members will post comments, but friends who are not part of the group will not. We next discuss how we address both of these issues in *Hermes*.

Obfuscating Group Size: Instead of making the *ufiles* for all of Alice's friends indistinguishable, *Hermes* attempts to hide the group members ($\mathbb{G}$) among a subset of Alice's friends ($\mathbb{D}$), where $\mathbb{G}$ is a subset of $\mathbb{D}$ (referred to as the anonymity set). Whenever an entry has to be added to the *ufile* for any user in $\mathbb{G}$, dummy entries are also added to the *ufiles* for those users in ($\mathbb{D} - \mathbb{G}$). The number of users in ($\mathbb{D}-\mathbb{G}$) follows an exponential distribution, with its minimum, mean, and maximum values set to α, $|(\mathbb{N} - \mathbb{G})|/4$ (rationale in Section 6), and $|(\mathbb{N} - \mathbb{G})|$ [2], where $\mathbb{N}$ contains all of Alice's friends. The parameter α allows us to handle small groups and is set to $\max(15, |\mathbb{G}|)$.

The effect of these parameters is that the size of the anonymity set is always at least double that of the private group. As a result, random guessing as to whether a particular user in the anonymity set is a member of the group will be correct with a probability of at most 50%. For small groups of size less than 15, randomly guessing as to whether a user in $\mathbb{D}$ is a group member succeeds with probability $|\mathbb{G}|/(|\mathbb{G}| + 15)$. In addition, the exponential distribution biases the anonymity set towards smaller sizes. This reduces the additional storage and bandwidth costs incurred for providing anonymity, as compared to a uniform distribution that chooses the size of the anonymity set at random from the range $[\alpha, |\mathbb{N} - \mathbb{G}|]$. Lastly, note that it is insufficient to determine the size of the anonymity set simply by inflating the group size by a fixed factor (since this clearly reveals the group size).

Preventing Inference of Group Membership Based on Comments: So far, Alice has been able to share content with $\mathbb{G}$ without revealing $\mathbb{G}$ or its size ($|\mathbb{G}|$). However, since only members of $\mathbb{G}$ will post comments on the shared content, the cloud provider will be able to distinguish the users in $\mathbb{G}$ from all those in $\mathbb{D}$. Thus, the additional fake members in $\mathbb{D}$ must also post fake comments

[2] Since private group sizes are typically small, we assume $|(\mathbb{N} - \mathbb{G})| > |\mathbb{G}|$.

as part of the conversation (these fake comments are discarded upon retrieval by group members).

A naive approach would require all the additional members in $\mathbb{D}$ to post as per either some random distribution or based on their previous posting habits. However, it is hard to provide any anonymity guarantees with such an approach. Moreover, since we assume that the source code for the *Hermes* client is publicly accessible, cloud service providers will have access to any distributions hard-coded into the client software.

Instead, our approach for posting of dummy comments works as follows. We divide time into slots, where all the members of a conversation can derive the slot boundaries based on the time at which the conversation was initiated (see Figure 2). We refer to each time slot as a round. In each round, every member of the conversation who is online during that period posts at least one comment, at a random point in time during that round. Those group members who have no real comments to post in a particular round—this includes both the users in $(\mathbb{D} - \mathbb{G})$ and the users in $\mathbb{G}$ who have no comments to post during that round—post at least one dummy comment during that round. All entries added to any $ufile$ are padded to a fixed size in order to hide the number of comments being posted by a user; this is necessary because a user who posts real comments may post multiple comments in a single round.

Importantly, every user in $\mathbb{D}$ posts either real or fake comments at only one particular time during each round. This ensures that the cloud provider cannot distinguish between users in $\mathbb{G}$ and those in $(\mathbb{D} - \mathbb{G})$, since it observes the same pattern of writing to and reading from $ufiles$ for all users in $\mathbb{D}$. Thus, when all users in $\mathbb{D}$ are online, the cloud provider has only a $\frac{\mathbb{G}}{\mathbb{D}}$ probability of correctly inferring whether a particular user in $\mathbb{D}$ is indeed a member of the private group $\mathbb{G}$.

Selecting the Length of a Round: A key design decision in instantiating the approach described above is to determine how time should be divided into rounds. Shorter rounds lead to more timely dissemination of comments. This is because when one user posts a comment in a particular round, another user can respond to this comment only in the next round; note that every user can post comments only once in each round. In contrast, longer rounds result in lower cost since fewer fake comments are posted, but compromise timeliness. Based on this trade-off, we split the timeline of a conversation into rounds as follows.

Our design is based on the observation that the commenting activity associated with most conversations is high when the conversation initially begins. After this initial period, the conversation goes stale and users may have few new comments.

Given this, to reduce the costs incurred to guarantee anonymity (hiding user sharing patterns), we partition any conversation into two phases. The first phase is when the conversation is fresh and one is likely to expect a comment in the near future. In this phase, the timeliness of comment dissemination is important, and therefore we keep a round's length short. Once several rounds with no real comments are observed, the conversation transitions to the second phase. The

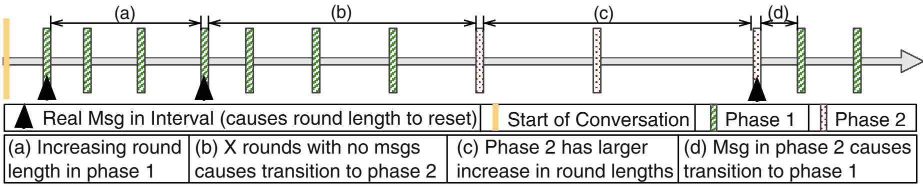


Fig. 2. Round structure in *Hermes*.

second phase aims to capture that phase of a conversation where no user has posted a comment for a while and there is a low probability of new comments. In this phase, we want to limit the cost associated with the conversation by minimizing the number of fake comments. The key property we exploit in this second phase is that, since the conversation is already stale, the timeliness of straggler comments posted during this period is not of concern.

In the first phase, all rounds are of equal length as long as at least one real comment is posted in each round. When there are no real comments in a particular round, we increase the length of the round by a multiplicative factor. The round length in the first phase is reset to its original value when a real comment is posted in the previous round. After a certain number of consecutive rounds with no real comments, the conversation transitions to the second phase. We model round durations in the second phase as a geometric series also, but use a larger multiplicative factor to increase round durations as compared to that used in the first phase. When a real comment is posted in the second phase, the conversation is reset to the first phase, but a fewer number of rounds of inactivity transitions the conversation back to the second phase in this case.

Note that the users who are in $\mathbb{D}$ but not in $\mathbb{G}$ cannot distinguish between real and fake comments; this is intentional, since we seek to hide group membership not only from cloud providers but also from users who are not in $\mathbb{G}$. Therefore, in every round, every user in $\mathbb{G}$ broadcasts to all of her friends who are also in $\mathbb{D}$ as to whether a real comment was posted in the previous round or not. Every user in $\mathbb{D} - \mathbb{G}$ who receives this notification relays this on to all of the user's friends who are in $\mathbb{D}$, exactly once. Thus, a user who receives a notification cannot distinguish between whether this was an original broadcast or a relayed broadcast. Once a user receives this information, she can independently determine what the length of the next round will be and when the transition between phases is triggered. Note that, from the cloud provider's perspective, these notification messages that convey whether a real comment was posted in a particular round are indistinguishable from real and fake comments. Moreover, though the cloud provider may be able to infer when real comments are posted based on when inter-comment spacings decrease, *no one other* than the users in $\mathbb{G}$ can determine *which* users posted the real comments.

5.2 Hiding Users' Conversation Patterns by Handling Intersection Attacks

Thus far, we have only considered hiding the identities of group members within a conversation. Unfortunately, the above approach is insufficient in completely hiding a users' sharing patterns across conversations. If fake users (in $\mathbb{D} - \mathbb{G}$) are chosen randomly from the user's friends ($\mathbb{N}$), the cloud provider can infer that users who appear repeatedly in different conversations are likely to indeed be real members of private groups.

To prevent such intersection attacks [18], we need to preserve anonymity *across* conversations. For this, we seek to ensure that a consistent group of $\mathbb{K}$ friends ($\mathbb{K} \subset \mathbb{N}$) appear across the conversations initiated by a user (Alice); we refer to this group as the Top $\mathbb{K}$ group. Thus, if a private, repetitive, group initiated by Alice is of size $\mathbb{G}$, the provider can only randomly guess if a user in the group of $\mathbb{K}$ friends ($\mathbb{K} >> \mathbb{G}$) is a true repetitive member with probability $\frac{\mathbb{G}}{\mathbb{K}}$. In essence, this provides $|\mathbb{K}|$-anonymity [31].

Our approach to form the Top $\mathbb{K}$ group (algorithmically depicted below) is to (1) tune the membership of $\mathbb{D}$ and (2) use fake conversations. We identify the friends with whom Alice consistently has private conversations (say $\mathbb{K}_1 \supset G$) and include them in the Top $\mathbb{K}$ group. We then fill the remainder of the Top $\mathbb{K}$ group with other friends with whom Alice rarely initiates private conversations (say $\mathbb{K}_2$).

Stage 1 Learn user habits

1: **for** Next M_1 conversations **do**
2: $\{\forall x \in \mathbb{G} : x.count+ = 1\}$
3: set $\mathbb{D} = \mathbb{N}$ and start conversation with entire friends list.
4: **end for**
5: Select $\mathbb{K}_1$ users with highest count values
6: Select $\mathbb{K}_2$ random friends s.t. $\{\forall x \in \mathbb{K}_2 : x \in \mathbb{N} \wedge x \notin \mathbb{K}_1\}$
7: reset count Values
8: **return** $\mathbb{K} = \mathbb{K}_1 \cup \mathbb{K}_2$

Stage 2 Use learned habits

1: **for** Next M_2 conversations **do**
2: Select size for $|\mathbb{D}| = \alpha + Exp(\frac{|\mathbb{N}|-|\mathbb{G}|}{4})$
3: $\forall x \in (\mathbb{N} - (\mathbb{G} \cup \mathbb{K})) : \mathbb{P}(x \in \mathbb{D}) = p$
4: Fill $\mathbb{D}$ from $\mathbb{K} - \mathbb{G}$ with probability of $x \in \mathbb{D} \propto Max(c - x.count, delta)$, where $c = Max(\forall x \in \mathbb{K} : x.count)$
5: $\forall x \in \mathbb{D} : x.count{+}{=}1$
6: schedule $\lceil \frac{\mathbb{K}-\mathbb{D}}{\mathbb{D}} \rceil$ fake conversations with $\mathbb{G} = \emptyset$ in current M_2 conversations
7: **end for**

Tuning the Membership of $\mathbb{D}$: As the first step, we need to determine which of Alice's friends consistently belong in private conversations. While doing so, in order to preserve anonymity, we simply use the naive approach wherein all of her friends are included in all conversations. This is referred to as the *first stage* or the *learning stage* of anonymizing conversations (Stage 1). This stage is executed

for M_1 (tunable parameter) conversations. During this stage, the *Hermes* client learns of the user's posting habits and with which friends the user is more likely to privately exchange information (set $\mathbb{K}_1$). It then forms the Top $\mathbb{K}$ group as described above.

In the second stage (Stage 2) which is then executed for the subsequent M_2 (tun able parameter) conversations, we reduce the total cost incurred by a user (Alice) by only consistently including the Top $\mathbb{K}$ group in private conversations. In each true conversation initiated by Alice, we now form the group $\mathbb{D}$ for that conversation as follows. First, all the user's friends that are neither part of the conversation group $\mathbb{G}$ nor the Top $\mathbb{K}$ group (determined in the first stage) are considered as candidates for inclusion in $\mathbb{D}$. Each of these candidates is included in $\mathbb{D}$ with a very small fixed probability p. This ensures that friends outside of Alice's Top $\mathbb{K}$ group, i.e., users with whom she rarely exchanges private content, are included with a small probability; this protects against the server correctly identifying true rare inclusions of such friends. Subsequently, Alice's friends that are part of her Top K group but not in $\mathbb{G}$, are considered for inclusion. The probability that a particular user (say Chloe) in the Top $\mathbb{K}$ group is selected is proportional to the difference between the maximum number of conversations any member of Top $\mathbb{K}$ group is involved in (both true or fake roles), and the number of conversations that Chloe is involved is involved in (both true or fake roles). This ensures that all of the members of the Top $\mathbb{K}$ group are consistently involved in conversations.

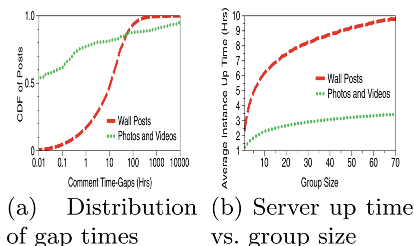
Using Fake Conversations: In spite of filling the groups as above, it is possible that real users appear more often than fake users. To address this, we schedule $\lceil \frac{\mathbb{K}-\mathbb{D}}{\mathbb{D}} \rceil$ *fake* conversations (with fake comments) where $\mathbb{G} = \emptyset$ (since each real conversation already includes $\approx \mathbb{D}$ members from the Top $\mathbb{K}$ group). The groups, $\mathbb{D}$, for such fake conversations are filled exactly as the real conversations are filled. Together, the above two steps of stage two ensure that every member of the Top K group is in (approximately) the same number of conversations on average.

To cope with the dynamics of Alice's sharing behaviors (she could converse more often with Bob and Dave at some point in time, and at a different time, exchange more private content with Chloe and Eve), we return to the first stage periodically to recompute the Top $\mathbb{K}$ group. Here, we take care to ensure that only minimal changes are made to the group $\mathbb{K}_2$ to prevent the server from identifying these as fake users.

Finally, instead of using fake conversations, to reduce costs one can think of suppressing an initiator's conversations with particular users with whom she is conversing too frequently. We do not explore this option as it violates our goal of ensuring timely sharing as in a traditional OSN.

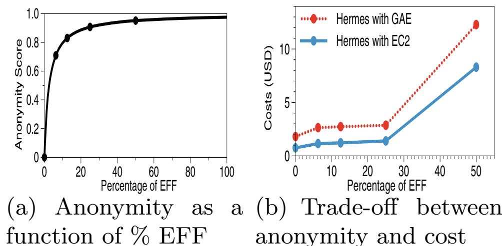
6 Quantifying Cost, Anonymity, and Timeliness Trade-offs

In order to tune *Hermes*'s configuration, we seek to understand the trade-offs between anonymity, timeliness, and cost. To do this, we crawl a large dataset



(a) Distribution of gap times (b) Server up time vs. group size

Fig. 3. Analyses with Facebook data



(a) Anonymity as a function of % EFF (b) Trade-off between anonymity and cost

Fig. 4. Anonymity trade-offs per conversation

from Facebook, and use the posting habits seen to perform a trace-driven simulation of *Hermes*.

Understanding the Temporal Nature of Conversations: We first seek to understand how long a posting is likely to be of interest to a user's friends, in the common case. Our particular interest is the *time gap* between when specific content was posted by a user and when the friends of that user lose interest in viewing it (the interested friends have already viewed it with high probability). However, it is impossible to accurately determine this duration without access to Facebook's server-side logs. Therefore, we instead use users' comments on a post as a proxy for their interest in the post. Though all users who find a posting to be of interest may not comment on it, previous studies have shown that the number of friends that see a post and the number of friends commenting or liking it are positively correlated [17]. Thus, we ask the question: for those postings that have associated comments, what is the time gap between the instant when the posting was made and when the last associated comment was posted?

Due to the lack of a publicly available dataset on users' posting habits, we crawled the profiles of 68,863 Facebook users using a combination of FQL (Facebook query language) and RestFB. Our crawled dataset, which spans a month, roughly comprises 1) 1.8 million wall posts and associated comments, and 2) 40K posts of either photos or videos with $\approx$ 35K associated comments. Remarkably, 70% of the 1.8 million posts did not have any associated comments. Thus, we look at the other 30% and the photos/videos to determine the time-gap between when the initial content was posted and when the last associated comment was seen. Based on Fig. 3a, we set the duration for which a user caches data on her cloud storage to 3 days; 90% of posts do not receive new comments beyond this period. In outlier cases, where content is sought long after it was posted, we sacrifice timeliness for resource thriftiness as discussed earlier.

A Simulation of *Hermes*: Next, we build a simulator to capture user interactions with *Hermes* in a large-scale setting; the simulation provides both 1) an understanding of how *Hermes* may perform, and 2) a validation of *Hermes*'s ability to provide anonymity with limited resources (small volumes of storage and bandwidth, few operational queries, and short uptimes for a user's

computing instance). To the best of our knowledge, there does not exist a simulator that mimics user interactions on an OSN.

Determining Simulation Parameters: The first input required by our simulator is a measurement of how often users come online. This dictates the expected time for disseminating content across *Hermes* clients, and thus, impacts how long the computing instance, or data stored on the cloud, will need to be active. Note that the *Hermes* client on a user's device does not need her to interact with it to fetch new content. Thus, the only time of interest is when the device is powered on and connected to the Internet. Here, we use data from [32], which provides the time per day for which users' devices are active. We assume that most powered on devices today are connected to the Internet. The weighted average of this time for desktops is 9.7 hours a day. The weighted average of online time for portable computers is comparable at 8.3 hours a day [32].

Second, to determine when a friend retrieves a private posting made by a user, we compute the relative time-gap between when the user is online and when the friend comes online later. We assume that users in similar time-zones are online during similar periods; if users are in time zones far apart, this time gap may be larger. Unfortunately, we were unable to access the location information of users in our data set; Facebook does not allow programmatic access to this information. Hence, we use two approximations to characterize the distribution of when the friends of a user come online. 1) We assume that users come online at random instances uniformly distributed over a 24 hour period, and stay online for a uniformly distributed period with an average of 9.7 hours; we believe that this model represents the likelihood that a user's friends are distributed all over the globe. 2) We consider a best case scenario wherein all of a user's friends are in her time-zone; here, we assume that the user and her friends come online within a 12 hour period. Again, the time at which the user comes online is uniformly distributed within this period, and the duration for which one stays online is chosen as before.

Third, to accurately represent a user's posting habits, we replay the posts in our Facebook data set. Since the posts we crawl are those shared by a user with all her friends, we obtain an estimate for the expected *private* group size from [20] and [3]; these studies suggest that while the social group size of a user is about 190, the more intimate size of a social group is 12. On this basis, we consider expected group sizes of 15, and use a uniform distribution with variance 10 (to cover group sizes from 5 to 25).

Selecting System Parameters: To simulate *Hermes*, we also need to choose the parameters that control how the system trades off timeliness for anonymity. The two phases of a conversation, as discussed in Section 5, depend on four factors, namely the initial length of a round (l), the multiplicative growth rate in phase one (A_1), the growth rate in phase two (A_2), and the number of rounds with no real comments in phase 1 (X), after which a conversation transitions to phase 2. To set X, we observe from the Facebook data that 95% of the time, the time gap between two consecutive comments is less than 24 hours. In

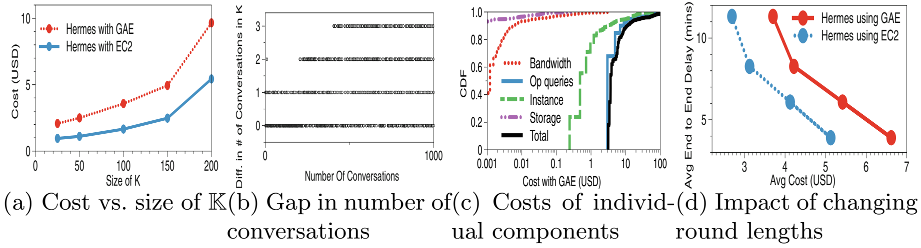
other words, a conversation is unlikely to be of interest to friends if there are no comments for about 24 hours. Hence, we transition a conversation from the first to the second phase if we do not see a comment for 24 hours. Later we vary l and A_1 in our simulator and examine the effects on average cost and timeliness. For phase 2, we seek an exponential growth, but want to simultaneously keep in check the delay incurred in retrieving straggler comments; thus we set $A_2 = 2$.

Simulator Design: Our simulator captures all the features of *Hermes* described in Sections 4 and 5. In our simulation, every user initiates conversations and posts comments as per her posting activity on Facebook. For each private conversation initiated by a user, we select a randomly chosen subset of the user's friends based on her posting habits and the expected group size considered. We consider the size of every shared photo as 2 MB and the size of all other private posts as 0.5 KB (these numbers are much larger than what we got from our crawled data). Since a user's comments in our crawled data may be on posts made by users outside our crawled population, we post any comment by a user to a pre-existing randomly chosen conversation that she is involved in.

Results and Interpretations: Our metrics of interest are (i) the time for which a user's compute instance needs to be active, (ii) the anonymity (likelihood of guessing if a friend is a true group member) a conversational group is provided, (iii) the total cost incurred, and (iv) the loss of timeliness due to users receiving stale data.

Compute Costs: First, we seek to determine the time for which the instance associated with any object (post) needs to be active. Recall that a *Hermes* client of a group member obtains new content as soon as she comes online. Considering the two approximations discussed above for when users come online, Fig. 3b plots the distribution of the time it takes for all the members of a private group to access the object. This is the time for which the compute instance has to be up. To handle the common cases where the group size is small (< 15), the compute instance needs be up for 6–7 hours even if a user's friends are globally distributed; if the friends are all local, it needs to be up for ≈ 4 hours. One may expect that in a typical case (when a user has both global and local friends), the compute instance will have to be up for a duration somewhere in between 4 and 7 hours; we conservatively choose the duration to be 10 hours. In rare cases where not all members access a posted object within the 10 hours, we trade-off timeliness in serving the content for lower cost. Based on this, our simulations indicate that, for $\approx 90\%$ of the users, *Hermes* will need to keep their instances active for less than 100 hours (or 4 days) in a month, in order to privately exchange *all the Facebook data* that they shared in the entire month. In comparison, prior solutions for OSN privacy [4,29] require every user to persistently have a compute presence in the cloud.

Quantifying Anonymity: Second, we seek to quantify the anonymity provided to a conversational group. First, we consider each conversation individually. We define the *anonymity score* to be the probability that the server is unable to correctly identify a group member as true or fake (as discussed in Section 5 this



(a) Cost vs. size of $\mathbb{K}$ (b) Gap in number of conversations (c) Costs of individual components (d) Impact of changing round lengths

Fig. 5. Anonymity across conversations and system costs

is $(1 - \frac{|\mathbb{G}|}{|\mathbb{D}|})$). In Figure 4a, we plot the anonymity score while varying the number of fake group members. The x-axis represents the percentage of the initiator's friends outside the group, who are added as fake members (denoted as external fake friends or EFFs). Specifically, $\mathbb{D} = \mathbb{G} \cup \text{EFF}$ where, $\text{EFF} \subseteq \{\mathbb{N} - \mathbb{G}\}$. The y-axis represents the anonymity score. Since the likelihood that the server is able to guess correctly is approximately proportional to the ratio of the number of true members (fixed) to the size of the composite group (with true and fake members), the anonymity score steeply increases as the size of the composite group increases initially. Beyond a certain point, we reach a point of diminishing gains, wherein the increase in the anonymity score is less significant with an increase in the composite group size. To achieve an anonymity score of about 0.9, we need to add 25% of the friends outside the true group as fake members in each conversation.

Per Conversation Anonymity vs. Cost Trade-Off: Next, in Figure 4b, we depict the expected (total) cost incurred as a function of the percentage of EFFs. We obtain the per-resource costs for different contributing factors from [1,2,8], and multiply this with the amount of resources consumed. For an anonymity score of 0.9 (% of EFFs = 25), we see that the expected *total* monthly cost per user is relatively low (< $ 4) with both Google App Engine (GAE) and Amazon EC2 (storage and computing are from the same provider). Thus, an EFF of 25% (or $|\mathbb{N} - \mathbb{G}|/4$) presents the best trade-off between per conversation cost and anonymity in *Hermes*.

Handling Intersection Attacks: In the scenario described above, we only considered the anonymity in a given conversation; the cost due to fake conversations (included for protection against intersection attacks) was not considered. Next, we present results that capture the effect of these conversations, which appear at a rate of $\lceil\frac{\mathbb{K}-\mathbb{D}}{\mathbb{D}}\rceil$ (recall Stage 2). The parameter $\mathbb{K}$ determines the level of anonymity provided across conversations as discussed earlier. The value of $\mathbb{G}$ is specific to each user and varies from 5 to 25; we fix an EFF of 25 % based on our previous results. In Figure 5a, we change $\mathbb{K}$ by varying $\mathbb{K}_2$ ($\mathbb{K}_1$ is estimated in the first stage of the process for each user as discussed in Section 5). We immediately see that $\mathbb{K} \approx 150$ yields the highest anonymity at a reasonable cost. This is only marginally higher than the value of $\mathbb{D}$ (with the chosen EFF value). This implies

Table 1. Cost for various values of A_1

A_1	**Avg. Delay** (min)	**Avg. Ops Cost** ($)
1.025	2.8867	5.13
1.05	3.096	2.80
1.10	22.52	2.32

Table 2. *Hermes*'s resource consumption on GAE

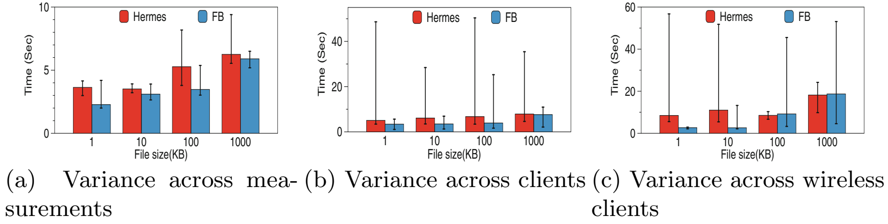
Operation	**MS/Req**	**Bytes Rx**	**Bytes Sent**
Posts links	130	78	4
Check and retrieve update	50	22	35
Add comment	45	42	2

that, on average, we need only one fake conversation for every real conversation in order to thwart the intersection attack. In Figure 5b, we plot the difference in the maximum and minimum number of conversations that the members of Top $\mathbb{K}$ group have participated in. We see that this difference is no greater than 3 at all times; this demonstrates the high degree of anonymity within the Top $\mathbb{K}$ group.

Cost Breakdown: Figure 5c shows the distribution (across users) of the total costs due to the various components required by *Hermes*, viz., storage, bandwidth, operational queries, and the computing instance, with GAE. We see that, for about 95% of the users, the total cost is $<$ \$10 a month. In comparison, if a compute instance is always active, the cost of this alone would be $>$ \$60 per month. We also see that the cost due to operational queries and the instance are the biggest contributors to the total cost. This is expected since storage and bandwidth are relatively cheap, especially since *Hermes* purges the cloud storage regularly. Operational costs are *relatively* high since storing, retrieving, or even checking for content, incurs a cost [2,8]. The total cost with EC2 is slightly lower than that with GAE ($<$ \$9 for 95% of the users) but the trends in the cost components are similar; we do not plot the results here due to space considerations.

Timeliness vs. Cost: In Figure 5d, we plot the expected delay incurred in accessing posts (a measure of timeliness) versus the expected (total) cost. We again use GAE. If the length l of each round (recall Section 5) is reduced, the operational costs are increased, but the timeliness is improved as well. If instead, we increase l, the timeliness suffers but queries are made less often (to check for content) and thus, cost decreases. Even if the desired expected delay is as low as 5 minutes, the incurred cost with GAE is no more than \$5 per month. With EC2, this cost is even lower (\$3). These results demonstrate that good timeliness is possible with *Hermes*, with fairly low cost.

Timeliness vs. Anonymity: Next, we quantify the impact of varying A_1 on timeliness for a fixed l (set to 5 minutes for our experiments). We measure this impact in terms of average costs and the average delay over all conversations in the simulation. In Table 1 we show a representative subset of our results that is of interest. As evident, increasing A_1 decreases average cost but increases the average delay of message propagation. From the table we see that when A_1 is decreased to 1.05



(a) Variance across measurements (b) Variance across clients (c) Variance across wireless clients

Fig. 6. End-to-end delays on *Hermes* and on Facebook

from 1.10, the marginal reduction in delay is significant; however, the additional reduction is marginal when A_1 is further reduced to 1.025. The cost growth is almost linear. These results suggest that setting A_1 (by default) to 1.05 provides the best trade-off between delay and cost.

7 Prototype Implementation and Evaluations

Implementation: We prototype *Hermes* in Java as an add-on to Facebook. We use the Facebook front end and a user's profile therein is used for making her public key component available. *Hermes* runs as a middleware and intercepts posts classified as private. Dropbox and Google App Engine (GAE) are used for storage and computation. Upon installation, the *Hermes* client requests OAuth 2.0 [10] access tokens from both Facebook and Dropbox and stores these locally for later use. The client crawls the list of the user's friends on Facebook and creates one $ufile$ for each of them on Dropbox. The client also initializes a web-based application on GAE on behalf of the user.

The implementation of *Hermes* essentially follows the design as described in the previous sections. ECDH is used to establish pairwise keys between the initiator and each of her friends and these are then used to establish conversation specific group keys. Then, the *Hermes* client uploads the content encrypted with the group key, to the user's space on Dropbox and requests Dropbox for the public URLs for these files.[3] When the user shares content, the *Hermes* client also invokes a GAE instance and uses the GAE data store to save the encrypted public URL to the file on Dropbox. All communications with the instance are over HTTPS. For efficiency, the instance shuts itself down after a configurable time has elapsed after creation (10 hours by default).

Both true and fake members access content as described earlier. The client devices of fake users are provided with a group key whose prefix indicates that it is a fake member; the server cannot detect a fake member, since the fake group key is encrypted using the pairwise keys. The client of a fake member simply

[3] With typical file sharing on Dropbox, when Alice shares a file with Bob, the shared file is counted towards the storage capacity of both Alice and Bob. In our implementation, public links are simply pointers to Alice's files; the files are then directly accessed by Bob.

discards all content retrieved with respect to the conversation (both the original posting as well as comments).

Storage Overhead Associated with Shared Content: Our implementation uses AES (256 bit key) to encrypt data and ECDH to establish a symmetric key (using the P-256 curve defined in [11]). The parameters (such as the curve and p) are defined in P-256 and available to all users.

Hermes adds overhead to shared content in three ways. (1) As described in Section 4, each $ufile$ entry occupies 16 bytes for a hash value and $\approx$ 20 bytes for an encrypted URL on Dropbox. (2) Each $ufile$ entry also includes the group key encrypted with the pairwise key of the sender and its corresponding receiver, with information for associating the entry with the receiver and authenticating her. In our implementation, the size of each *password* tuple is 62 bytes. (3) *Hermes* stores information about the uploaded files and the access tokens for writing to a user's Dropbox account on the user's GAE instance. In our implementation, every post accounts for 440 bytes of space on the GAE instance. In essence, (1) *Hermes*'s storage overhead is a few KB for sharing data of any size; as an illustration, storage overhead is 82, 820, and 1640 bytes for group sizes of 1, 10, and 20 members (including fake), and (2) storage overhead of *Hermes* increases linearly with the composite group size. Note that we expect private groups to be typically small [3,20].

Efficiency of *Hermes*: We next evaluate our prototype by comparing the delay incurred in sharing data with *Hermes* to that with Facebook. We share files of different sizes and measure the total delay between when a user shares a file and when a recipient completes receiving that file. To mimic the overhead seen by real users, the receiver program contacts 250 compute instances (250 is the average number of friends on Facebook [14]) to check for new content.

Fig. 6a shows the variance in delays incurred (the minimum, median, and maximum values across 5 trials) in the above experiment. The overhead imposed by *Hermes* as compared to sharing and receiving data on Facebook, especially for delay-sensitive sharing of small files, is within reason (a few seconds). Delays on *Hermes* are higher than delays on Facebook because *Hermes* not only posts the shared content on Dropbox, but also sends the links to these files to the user's GAE instance. Furthermore, *Hermes* uploads and downloads $ufiles$ in addition to the content being shared.

We repeat the experiment on 6 PlanetLab [28] nodes, two on the US west and east coasts, and one each in Europe, Asia, Australia, and South America, and with 10 clients that access *Hermes* and Facebook via WiFi. Figs. 6b and 6c show the variance in median access times across the PlanetLab nodes and across the wireless clients. We see that the access times with *Hermes* are comparable to that of direct data sharing on Facebook.

Resource Usage: Next, we measure the compute and bandwidth resources consumed by the three *Hermes* client operations that require interactions with the compute instance: (i) post links to newly shared content to the instance, (ii) serve requests from friends who check if anything new has been shared with

them (and download new comments, if any), and (iii) receive the link to a recipient's comment and post it to the conversation. We perform each operation 1000 times and examine GAE's reports for resource usage. Table. 2 shows that the compute time and incoming/outgoing network traffic incurred on average, for each operation is low.

8 Conclusions

We design and implement *Hermes*, a practical, cost-effective, OSN architecture for private content sharing. *Hermes* intelligently uses limited storage and computing resources on the cloud to facilitate timeliness and high availability, while minimizing resource usage. A key property of *Hermes* is that neither the cloud providers nor other friends of a user can infer the membership of a private group. Via an analysis of mined Facebook data and exhaustive simulations, we show that *Hermes* greatly reduces costs compared to alternative solutions while ensuring the anonymity of the private group.

Acknowledgement. This work was supported by Army Research Office grant 62954CSREP.

References

1. Amazon EC2 micro-instance. amzn.to/14fxKbM
2. Amazon S3 pricing. amzn.to/1dRGFuz
3. Anatomy of Facebook. on.fb.me/1az2axi
4. The Diaspora project. diasporaproject.org/
5. Facebook fixes security glitch after leak of Mark Zuckerberg photos. lat.ms/14fx4mC
6. Facebook says it fixed leak that opened info to third-parties. wapo.st/12UidOW
7. Google App Engine. bit.ly/117kPXo
8. Google App Engine Pricing. bit.ly/1cclPzm
9. Heroku. www.heroku.com/
10. OAuth. oauth.net/
11. Recommended elliptic curves for federal government use. 1.usa.gov/14fwPYI
12. Some quitting Facebook as privacy concerns escalate. bit.ly/15pqQmK
13. Syme. getsyme.com
14. Your Facebook friends have more friends than you. wapo.st/11I58Mj
15. Baden, R., Bender, A., Spring, N., Bhattacharjee, B., Starin, D.: Persona: an online socialnetwork with user-defined privacy. In: SIGCOMM 2009 (2009)
16. Beato, F., Kohlweiss, M., Wouters, K.: Scramble! your social network data. In: Fischer-Hübner, S., Hopper, N. (eds.) PETS 2011. LNCS, vol. 6794, pp. 211–255. Springer, Heidelberg (2011)
17. Bernstein, M.S., Bakshy, E., Burke, M., Karrer, B.: Quantifying the invisible audience insocial networks. In: CHI 2013 (2013)
18. Danezis, G., Serjantov, A.: Statistical disclosure or intersection attacks on anonymity systems. In: Fridrich, J. (ed.) IH 2004. LNCS, vol. 3200, pp. 293–308. Springer, Heidelberg (2005)

19. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. In: Security 2004 (2004)
20. Dunbar, R.: The ultimate brain teaser. bit.ly/17Floky
21. Feldman, A.J., Blankstein, A., Freedman, M.J., Felten, E.W.: Social networking with frientegrity: privacy and integrity with an untrusted provider. In: Security 2012 (2012)
22. Feldman, A.J., Zeller, W.P., Freedman, M.J., Felten, E.W.: Sporc: group collaboration usinguntrusted cloud resources. In: OSDI 2010 (2010)
23. Guha, S., Tang, K., Francis, P.: Noyb: privacy in online social networks. In: WOSN 2008 (2008)
24. Jahid, S., Nilizadeh, S., Mittal, P., Borisov, N., Kapadia, A.: DECENT: a decentralized architecture for enforcing privacy in online social networks. In: IEEE SESOC 2012 (2012)
25. Koblitz, N., Menezes, A., Vanstone, S.: The state of elliptic curve cryptography. Designs, Codes and Cryptography (2000)
26. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM (1978)
27. Liu, D., Shakimov, A., Cáceres, R., Varshavsky, A., Cox, L.P.: Confidant: protecting OSN data without locking it up. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, pp. 61–80. Springer, Heidelberg (2011)
28. Peterson, L., Anderson, T., Culler, D., Roscoe, T.: A blueprint for introducing disruptivetechnology into the internet. In: HotNets 2002 (2002)
29. Shakimov, A., Lim, H., Caceres, R., Cox, L., Li, K., Liu, D., Varshavsky, A.: Vis-a-vis: privacy-preserving online social networking via virtual individual servers. In: COMSNETS 2011 (2011)
30. Stefanov, E., Shi, E., Song, D.X.: Towards practical oblivious ram. In: NDSS 2012 (2012)
31. Sweeney, L.: K-anonymity: A model for protecting privacy. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. (2002)
32. Urban, B., Tiefenbeck, V., Roth, K.: Energy consumption of consumer electronics in US homes in 2010. bit.ly/10NMqOn
33. Zhang, L., Mislove, A.: Building confederated web-based services with priv.io. In: COSN 2013, New York, NY, USA (2013)

A Propagating Updates via *ufiles*

The $ufile$ contains tuples $< id, c_{url}, up, data >$, where id is a monotonically increasing counter (eventually wraps around), c_{url} is the unique URL pointing to the folder of the owner of conversation c, up indicates the type of update, and $data$ is the data associated with a specific update status (explained later). A user's *Hermes* client creates a $ufile$ for each of the user's friends when the user first begins using *Hermes* (and whenever the user adds a new friend thereafter). The first time that a pair of friends engage in a private conversation, they use the computing instance to exchange pointers to the $ufiles$ they have created for each other. The links to the $ufiles$ are then stored on their respective cloud storage permanently.

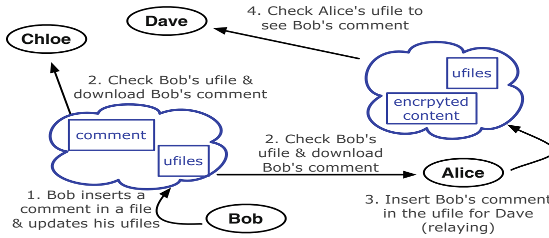


Fig. 7. Illustration of comment propagation in *Hermes*.

Consider a new bootstrapped conversation between Alice and all her friends. We consider the scenario where Bob wants to post a comment (or reply) to content that was originally shared by Alice.

Step 1: To comment on the content posted by Alice, Bob writes an update to the $ufiles$ that he maintains for his friends Alice and Chloe. Bob's *Hermes* client writes this update only to his $ufiles$ for Alice and Chloe, and not his other friends, since they are the members of the group. This update contains the tuple $< id, c_{url}, 1, link >$, where '1' is an integer code to indicate that a new comment in conversation c is in $link$,which is owned by Bob.

Step 2: When Alice and Chloe come online, they download their respective $ufiles$ from Bob's storage and learn of the new comment in c. They individually retrieve Bob's comment and create new tuples of the form $< id, c_{url}, 2, link >$ in their own $ufiles$ for Bob. Here, '2' is an integer code indicating that they have received the last comment made by Bob in conversation c.

Step 3: When Bob comes online again and his client downloads the corresponding $ufiles$ from Alice and Chloe, it realizes that all his relevant friends have read his latest comment. It then deletes the prior update $< id, c_{url}, 1, link >$ from his $ufiles$; it also purges the corresponding comment from his cloud storage. By doing this, the space occupied by the comment and $ufiles$ do not simply grow over time, thus drastically decreasing *Hermes*'s cloud storage requirements. Upon returning online, Alice and Chloe notice that Bob's original entry is deleted from his $ufiles$. This implicitly tells them that Bob has received their update and hence, they delete their update tuples from their $ufiles$ for Bob.

Step 4: While Alice and Chloe get Bob's comment, Dave is not Bob's friend and hence, does not receive it (Bob does not even maintain a $ufile$ for Dave). To allow Dave to see all the comments in a conversation he is part of (as with Facebook), *Hermes* leverages the fact that Alice is a friend to all group members and incorporates an additional step (shown in Fig. 7). When Alice notices Bob's update (step 2), she checks whether there exist group members who are not friends with him. For each such member (e.g., Dave), Alice inserts an update tuple $< id, c_{url}, 3, rc >$ in their respective $ufiles$; here '3' is a code for the relaying of a comment. rc refers to the relayed comment included in the tuple.

Upon coming online, Dave downloads Alice's $ufile$ for him, finds the comment, and notifies Alice of the receipt of this update. Alice and Dave then purge the associated updates from their respective $ufiles$ (steps 2-4 as before).

Note that the above scheme for distributing comments also works for other types of notifications (e.g., 'Likes' on Facebook) by simply having different update codes for different types of notifications.