MobiRate: Making Mobile Raters Stick to their Word

Daniele Quercia, Stephen Hailes, Licia Capra
Department of Computer Science, University College London, London, WC1E 6BT, UK.
{D.Quercia, S.Hailes, L.Capra}@cs.ucl.ac.uk

ABSTRACT
To share services, portable devices may need to locate reputable in-range providers and, to do so, they may exchange ratings with each other. However, providers may well tweak ratings to their own advantage. That is why we have designed a new decentralized mechanism (dubbed MobiRate) with which portable devices store ratings in (local) tamper-evident tables and check the integrity of those tables through a gossiping protocol. We evaluate the extent to which MobiRate reduces the impact of tampered ratings and consequently locates reputable service providers. We do so using real mobility and social network data. We also assess computational and communication costs of MobiRate on mobile phones.

ACM Classification Keywords
D.2.0 Software Engineering: General—*protection mechanisms*; C.2.4 Computer-Communication Networks: Distributed Systems—*distributed applications*

General Terms
Algorithms, Security

Author Keywords
Distributed Reputation (Trust) Systems, Mobile Systems

INTRODUCTION
In-range portable devices may collaborate for enabling new applications. Such applications might include, for example: cooperative device localization that increases the precision of street map software and of location-based services; synchronization of timers in playing mobile multi-player games; Web content caching that avoids monetary costs of cellular or wireless providers.

To enable those applications, researchers have proposed general infrastructures with which portable devices opportunistically trade various services with each other [6, 20, 21]. At the heart of those designs is a *centralized* reputation system that stores ratings about service providers. For locating reputable providers, portable devices have no choice but to go through this central server. Given this problem, researchers have conceded that, on scalability and mobility grounds, the reputation system needs to be decentralized [6, 21].

One way of decentralizing a reputation system is to have each user run a reputation model on her device [25]. A reputation model is a piece of software that keeps track of which devices are reputable for providing services and which are not. To see how a reputation model works, consider two devices A and B, and say that, after receiving a service from B, A rates B. When deciding whether to trust B, other devices may find that rating useful, so the rating should be stored and made available to other devices. A simple way of making the rating available is to have A send the rating to B, B acknowledge it, and both of them store the rating locally (B may do so to prove its trustworthiness to other devices by showing credentials, and A to keep track of the nodes it has interacted with). This approach is simple yet yields to these threats: A *lies* (it supplies fake ratings); B pretends to be *ignorant* (it does not acknowledge the rating); or B is *malicious* (it tampers with the rating).

In the next section, we will see that previous work has focused on the first problem (lying recommenders) and has not thoroughly explored the problem of ignorant and malicious individuals. We set out to fill this void by making the following contributions:

- A new and secure way of storing and exchanging ratings for portable devices (described in Section "Our Proposal: MobiRate"). The key idea is that A and B store the ratings in their local tables. They then obtain verifiable evidence that the counterpart actually did so by running a commitment protocol. They send those commitments to other devices, which become witnesses. Whenever they interact with other devices, witnesses check whether they store commitments about those devices and, if so, they verify whether those commitments are honored. Eventually, honest individuals that run MobiRate: *suspect* any device that has not acknowledged a rating sent to it; *expose* any device that has tampered with at least one of its ratings; or *trust* any other honest device.
- Evaluation of the robustness of MobiRate on real mobility and social network data (Section "Robustness").
- Evaluation of its communication and computational overheads on mobile phones (Section "Overheads").

EXISTING SOLUTIONS: UNFIT FOR MOBILE USERS

There has been rapid progress in recent years in understanding how to combine user ratings in both Web and peer-to-peer applications in the presence of malicious users.

Early work by Chris Dellarocas [8] explored the use of robust statistics in aggregating user ratings on reputation websites (e.g., on eBay). Those statistics aimed at safeguarding the reliability of those websites in the presence of potentially deceitful buyers and sellers. At the same time, Ralph Levin designed a way for identifying reputable members in the Advogato free software developer community [19]. He did so by creating a network whose nodes are members and whose links are trust relationships. To assign a trust value to each member, he applied the max flow algorithm to that network. The idea of max flow is that between any two nodes (members), the quantity of trust flowing from one node to another cannot be greater than the weakest rating somewhere on the path between the two nodes. Therefore, the less connected a member, the lower its trust value. Since malicious nodes happen to be the least connected nodes, they receive the lowest trust value.

In peer-to-peer networks, EigenTrust [16] assigns a trust value to each peer similar to how Google's PageRank [23] ranks web pages. Trust values are then used by peers to exclude untrustworthy peers (which send inauthentic files) and to select peers from whom to download files. As a consequence, the number of inauthentic files in the network decreases.

In short, there is now a substantial literature on how Web and peer-to-peer users aggregate ratings. However, there has been little work on how mobile users would do so. One may well wonder why. Here is a possible explanation: a portable device collects and stores only a tiny part of the ratings in the system and does not necessarily have enough knowledge to keep track of who is trustworthy and who is not. By contrast, a Web or peer-to-peer application is able to collect and aggregate all the ratings in the system [2, 8].

It thus seems that a different way of storing ratings for portable devices is needed. But what sort of method should we use? Ideally, one needs a distributed mechanism with which ratings are made both available (to counter limited knowledge) and tamper-evident (to detect tampering).

OUR PROPOSAL: MOBIRATE

We design one such mechanism and call it MobiRate:

1. *What it is:* MobiRate allows portable devices both to store ratings locally and to make them tamper-evident.

2. *Where it works:* MobiRate is applicable to various scenarios where there are many opportunities of relatively long-lived interactions between in-range devices. An example is a local coffee shop in which individuals (few of which are regulars) spend tens of minutes in relatively close proximity of each other. Another example is a train in which passengers commute to work daily. In those locations, individuals may be willing to share digital content (e.g., music files) within their physical neighborhood. They may want to do so to improve their downloading performance, to reduce monetary costs of downloads over cellular links, or, in the long-term, to interact with their peers using Mobile 2.0 applications in which users are not only consumers of content but also producers [5] (so much so that they have been aptly called "prosumers" [30]). However, sharing content introduces the problem of how to find reputable sources of downloads. One way of doing so is to rate sources, and then store and exchange those ratings using MobiRate. To ease illustration, we will henceforth refer to an abstract situation: devices A and B come into range, B makes available digital content, A downloads it and eventually rates the completed experience.

3. *When it works:* MobiRate works under the assumption that each device has a *unique* identifier in the form of a public key. If identifiers are not unique, malicious individuals can escape responsibility by assuming a different identity (i.e., by launching Sybil attacks [9]). In Section "Discussion", we will look at the impact of Sybil attacks.

4. *How it works:* A device that runs MobiRate implements four operations (which we gradually introduce in this section): (1) Maintain ratings locally; (2) Commit to those ratings; (3) Gossip those commitments; and (4) Act on the commitments it receives. By implementing those operations, MobiRate guarantees that every malicious device (that tampers with the ratings it stores) will be detected *eventually* and *likely*. "Eventually" because we allow for some delay since devices are mobile and can communicate with each other only sporadically; and "likely" because MobiRate offers *probabilistic* guarantees. In "Evaluation", we will show that those guarantees are high and met in brief simulation time under the assumption that people have few regular encounters. That is the case in reality because of the so-called familiar strangers: people who we do not know (strangers) but meet regularly (familiar) on, say, the way to work [13].

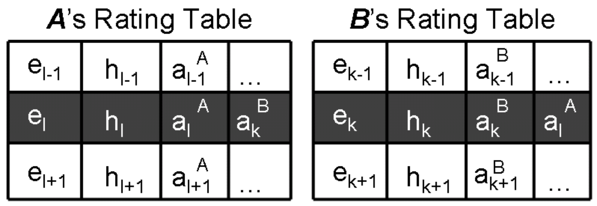
$P_B(Y)$	Public-key encryption of data Y using B's public key.
$S_B(Y)$	Public-key encryption of data Y using B's secret key.
$H(Y)$	Hash value of data Y.
$X\|\|Y$	Bit-string concatenation of data X and Y.
a_k^B	Authenticator of B's k^{th} entry signed by B(using S_B).
r_k	k^{th} rating.
s_k	Sequence number of r_k.
h_k	Hash chain up to r_k.
t_k	Additional information about r_k.
e_k	k^{th} entry made up of s_k, r_k, and t_k.

Table 1. Symbols used to present MobiRate.

Rating Tables

To begin with, consider that device A receives content from B and produces a rating for the completed interaction. A then stores its rating (e.g., "$A \rightarrow B$: 2") locally and sends a copy of it to B. After receiving the rating, B stores it as well.

Both A and B should now make their rating tables *tamper-evident* (i.e., any deliberate altering of them is detectable).



A's Rating Table

e_{l-1}	h_{l-1}	a_{l-1}^{A}	…
e_{l}	h_{l}	a_{l}^{A}	a_{k}^{B}
e_{l+1}	h_{l+1}	a_{l+1}^{A}	…

B's Rating Table

e_{k-1}	h_{k-1}	a_{k-1}^{B}	…
e_{k}	h_{k}	a_{k}^{B}	a_{l}^{A}
e_{k+1}	h_{k+1}	a_{k+1}^{B}	…

Figure 1. A's and B's Rating Tables. Take B's table. In it, B appends its ratings in a way that makes them tamper-evident. For example, the entry e_k contains B's k^{th} rating and is made tamper-evident by adding authenticator a_k^B and hash value h_k (which chains back to the previous entry). The last column in both tables are populated after A (B) runs the commitment protocol.

They do so in a way similar to how an existing technique builds tamper-evident logs [28]. This technique dates back to 1998, and, until recently, has inspired successful mechanisms for detecting faults in distributed systems [7, 12]. More specifically, A and B make their entries tamper-evident by associating with each of their entries a statement (called "authenticator"). Upon verifying it, any device is able to determine whether the entry is authentic - if the entry has been tampered, the authenticator gets compromised. To see why, having the notation in Table 1 at hand, consider that B produces an entry e_k containing the rating and appends it to its table. The entry (highlighted in Figure 1):

- Consists of a sequence number s_k, a rating r_k of the form "$A \rightarrow B$: 2", and additional information t_k. For short, the entry $e_k = (s_k, r_k, t_k)$. In general, it is useful to know the type of content a rating refers to (e.g., music file, video file) [27], and that is why we specify it in t_k - a statement signed by B that ensures that A has sent a certain type of content to B. Sequence numbers do not have to be contiguous but must be increasing; a timestamp could be used. Also, sequence numbers must be increasing, but corresponding interactions do not need to be serialized - one does not need to end one interaction before starting the next. That is because interactions can end without any predefined order and, only once they end, they are rated and associated with sequence numbers.
- Includes a recursively defined hash value $h_k = H(h_{k-1}||s_k||H(e_k))$. The hash value is recursive in the sense that it "chains back" to the previous one (h_{k-1}). That makes ratings tamper-evident - if any rating is altered, then the hash chain gets compromised.
- And finally includes an authenticator $a_k^B = S_B(s_k, h_k)$ - a signed statement with which B commits to having stored entry $e_k = (s_k, r_k, t_k)$ whose hash value is h_k.

In Section "Discussion", we will see that this way of constructing a table makes sure that tampered ratings are detected, that each device keeps only one table, and that ratings can change over time.

Making Commitments

After storing the rating, A and B must obtain verifiable evidence that the counterpart actually did so. That is done by using the *commitment protocol*, which consists of two steps:

1^{st} **Commitment:** A stores the rating in its table and sends its authenticator to B. The authenticator is a verifiable evidence that A stored "$A \rightarrow B$: 2" in its table. As we will see shortly, the authenticator comes with additional information used for verifying it.

2^{nd} **Commitment:** Similarly, B stores the rating in its table and sends its authenticator (plus additional information) to A.

At this point, both devices possess verifiable evidence that the counterpart stored the rating. Let us describe these two steps in more detail:

- A creates an entry $e_l = (s_l, r_l, t_l)$ in its table (r_l is of the form "$A \rightarrow B$: 2"). It then creates the authenticator a_l^A, attaches additional information to verify it (h_{l-1} and e_l), and sends the result to B.

$$1^{st}\ \textbf{Commitment:}\ A \rightarrow B : \{a_l^A, h_{l-1}, e_l\}$$

- B has now enough information to compute h_l and, consequently, to verify a_l^A. If the signature in a_l^A is not valid, B discards r_l. Otherwise, B creates its own table entry (s_k, r_k, t_k) (with $r_l = r_k$) and returns an acknowledgment with the authenticator a_k^B plus additional information to verify it (h_{k-1}, and e_k).

$$2^{nd}\ \textbf{Commitment:}\ A \leftarrow B : \{a_k^B, h_{k-1}, e_k\}$$

If the protocol ends correctly, it ensures that B cannot modify A's rating without being detected. But, what if the protocol fails? That is: what if there is no interaction between A and B but, nonetheless, A makes up a rating for it? what if A refuses to take part in the protocol? what if either A or B does not store the corresponding authenticators? We answer those questions next.

Gossiping Commitments

Up to now, A and B have committed to having stored the rating and have gossiped verifiable evidence of it. This evidence takes different forms depending on how the commitment protocol ended (or failed):

- 1^{st} Commitment *does not take place* (i.e., A does not send its rating): Nothing is done since we leave recommenders free to choose whether to rate or not. However, if A does not rate B, it should not be able to claim otherwise later. That is, A must be able to rate B only if it has received content from B. To ensure that, we have considered that A can rate B only if it possesses a statement t_k with which B certified that it has sent content to A. To make that statement unique, B associated a timestamp with it. Associating a timestamp is feasible since it only requires that devices in the system are *loosely* synchronized, and it guarantees that A cannot make up a rating if A has not received any content from B.

- 2^{nd} Commitment *does not take place* (i.e., B does not acknowledge the rating it has received from A): A gossips the answer it sent to B to other devices. Again, A cannot make up ratings simply because, to release a rating, it has to receive t_k from B. So, if we name one of those devices W, then:

  A's Type 1 gossip: $A \rightarrow W : \{a_l^A, h_{l-1}, e_l\}$

  The gossip contains enough evidence for challenging B to acknowledge A's rating and for convincing any device (including W) that if B were an honest device, it would be able to answer the challenge. We name this gossip Type 1 gossip and will see how W acts upon it in the next Section.

- *Both steps run correctly*: A and B gossip each other's commitment (reply) to other devices. We name this kind of gossips Type 2 gossips:

  A's Type 2 gossip: $A \rightarrow W : \{a_k^B, h_{k-1}, e_k\}$
  B's Type 2 gossip: $B \rightarrow W : \{a_l^A, h_{l-1}, e_l\}$

To see how witnesses are selected, consider that not all devices should care to store gossips about (to act as witnesses for) B. Were that not be the case, devices would be swamped by massive storage and communication overheads. In a content sharing scenario, the devices that care to store gossips about B are those that come into contact with B and share interests with it. That is because they are the only devices that are willing to interact with B (as they share interests with B) and able act upon those gossips (as they are colocated with B). So we consider that W is a good witness for B if both conditions hold:

- W has met B before. Past encounters help in predicting future ones because individuals are creatures of habits. They meet the same people regularly. For example, they meet their friends and their familiar strangers [24]. Familiar strangers are those people who we do not know but we meet regularly, say, on the way to work or at local coffee shops. So, if W has met B, it is likely that W will again do so in the future [13].

- W shares interests with B. That introduces the problem of how W and B would know whether they have common interests (e.g., whether they like similar music genres). They may do so by simply having their Bluetooth interface on: in so doing, they can advertise and match their interests by encoding them into the undefined attributes of the Bluetooth Service Discovery [1]. Being part of the discovery mechanism, undefined attributes are shared without any additional communication cost.

Therefore, witnesses are what we call *like-minded familiar strangers* - people who we do not know (strangers) but we meet very regularly (familiar) and we share interests with (like-minded).

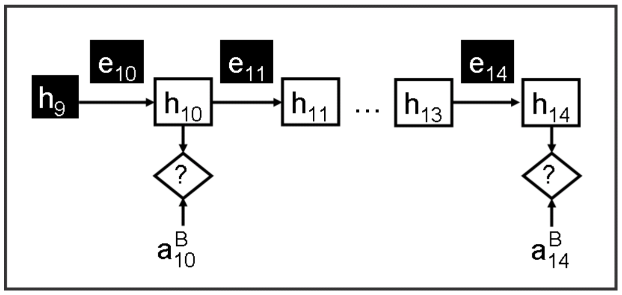


Figure 2. Verifiable Hash Chain. During the audit protocol on B's entries $(e_{10}, \ldots, e_{14})$, W attempts to form a chain with B's answer (highlighted) and to match that chain with two authenticators a_{10}^B and a_{14}^B. If the match succeeds, one can conclude that B has not tampered with any of the ratings in its entries $(e_{10}, \ldots, e_{14})$.

Acting on Gossips

To see what B's witnesses should do, let us call one of them W. W acts on the gossips about B only if it needs to interact with B, and the way it will do so depends on the gossips it stores, which are of two types:

Type 1 gossip: *B not acknowledging a rating.* If it has received a gossip of Type 1, W checks whether B deliberately failed to acknowledge the rating or was unable to do so (simply because it was slow or moved away). So, before interacting with B, W runs the *commitment protocol* for the unacknowledged rating. If B does not respond, W marks B as *suspected* and gossips its suspicion (again, a Type 1 gossip). By contrast, if B responds, W simply *trusts* B, and the gossip dies out. The main idea is that if B is honest and has not received a certain rating yet, it accepts the rating now and returns an acknowledgement. By contrast, if B has already received the rating but the rater has failed to receive any acknowledgement, then B can simply re-send the earlier acknowledgment to witness W, which will eventually send it back to the rater. In Section "Evaluation", we will see that these messages entail little communication cost.

Type 2 gossip: *B committing to ratings.* If W has collected gossips about B and now needs to interact with B, then, before interacting, W runs the *audit protocol*. This protocol checks the consistency of B's ratings. For example, consider that W has collected 3 authenticators - say, $\{a_{10}^B, a_{11}^B, a_{14}^B\}$. In that case, W checks the signature on the authenticators with the lowest and highest sequence numbers (on a_{10}^B and on a_{14}^B) and sends them to B. By doing so, W challenges B to produce the corresponding ratings $(r_{10}, \ldots, r_{14})$. If B is:

- unable to do so, then W marks B as *suspected* and gossips its suspicion (as a pair of Type 2 gossips).
- able to do so, then B has not tampered with any of the corresponding ratings. That is because the hash values of those entries form a chain connecting a_{10}^B to a_{14}^B. To see how, consider that B returns the entries $(e_{10}, \ldots, e_{14})$ plus the preceding hash value h_9. By construction, B's answer should form a chain against which both authenticators can be matched (see Fig-

ure 2 for a sketchy representation, and Theorem 2 in the Appendix for a formal explanation). If B's answer forms a chain, W *trusts* B of not having tampered with those ratings, and the gossip dies out. By contrast, if it does not form a chain, W marks B as *exposed* and gossips evidence of it (a pair of Type 2 gossips).

Let us rephrase those two steps more formally. If it runs successfully, the audit protocol consists of a challenge and of a response:

W's Challenge: $W \rightarrow B : \{a^B_k, a^B_m\}$

B's Response: $W \leftarrow B : \{h_{k-1}, e_k, \ldots, e_m\}$

By contrast, if the protocol fails, W sends a pair of Type 2 gossips to other devices (one of which we now call D):

W's Pair of Type 2 gossips:

$W \rightarrow D :$

$\{a^B_k, h_{k-1}, e_k\}$

$\{a^B_m, h_{m-1}, e_m\}$

To summarize, depending on the type of gossips, W runs either the commitment protocol or the audit protocol. By doing so, W *suspects* B, if B does not acknowledge any message from W; *exposes* B, if B has tampered with at least one of its ratings; or *trusts* B otherwise. We now turn to evaluating to what extent MobiRate is able to suspect and expose malicious devices in practice.

EVALUATION

The goal of MobiRate is to make it difficult for users to tamper with ratings. To ascertain the effectiveness of MobiRate at meeting this goal, our evaluation ought to answer two questions:

- **Robustness:** How effectively does MobiRate protect ratings? More specifically, is MobiRate robust against malicious individuals (see Section "Robustness")?
- **Overhead:** What time, storage, and communication overhead does MobiRate impose on a mobile phone (see "Overhead")?

Robustness

Ideally, we would evaluate the robustness of MobiRate by carrying out a field experiment. We would find a large number of mobile users who run applications for sharing services and have them exchange ratings either using state-of-the art protection or MobiRate. We would also "introduce" users who maliciously modify their ratings. As users share content on the move, we would measure the fraction of *compromised interactions* (i.e., decisions of interacting made by honest individuals on fake ratings).

Unfortunately, we do not have a deployment of mobile content sharing applications. So we need to set up a realistic simulation. We do so in "Simulation Setup" and use empirical observations about how individuals move, when they interact, and which interactions get compromised. Then, while running our simulations, we evaluate the robustness of MobiRate by keeping track of the fraction of compromised interactions (which we call f) - interactions between an honest device A and a malicious device B in which, unbeknownst to A, B supplies forged ratings, and A decides to interact upon those ratings. By doing so, we assess: to what extent MobiRate reduces f (see Section "Reducing f"); and how fast it does so (see "f Over Time").

Simulation Setup

The simulation setup is based on observations about:

- *How individuals move.* We use the mobility traces produced by the Reality Mining project at MIT [10]. That project tracked how 96 individuals moved while carrying their mobile phones for 9 months.
- *When they interact.* To model *when* people interact (i.e., exchange digital content), we reasonably consider that two individuals interact if they come into range and have interest in common. Our mobility traces tell those who come into range, so we simply need to model those who have interests in common. To do so, we model interests as categories of digital content (e.g., music genres) and distribute those categories across individuals. We do so by assigning categories at *random*. However, that does not reflect reality on two counts. The first is that some categories are more popular than others. More specifically, category popularity often follows a Zipf distribution [3]. The second count is that one may well befriend similar people since homophily (i.e., love of "similar others") has been found to play a starring role in human society [17]. Therefore, to assign categories, we need to account for those two aspects. We are in the position of doing so because the Reality Mining project not only tracked how individuals moved but also recorded their social network ("who knows whom"). We have exploited this added knowledge to realistically distribute interests so that:
  - *Friends share more categories than unknown individuals do.* We obtain that by following links in the social network, selecting one node at the time (step 1), and assigning categories to that node (step 2). In so doing, we bias the way we assign categories over time - one friend after the other. We have quantitatively measured to what extent friends' categories overlap by computing the Jaccard similarity for each pair of friends (Jaccard similarity is a statistic used for comparing the similarity of two sets [29]). Compared to random, the realistic way of distributing interests shows an average Jaccard similarity 3.2 times higher.
  - *Category popularity follows a Zipf distribution.* We do so by preferentially assigning popular categories in step 2. More formally, the probability of assigning a category is proportional to that category popularity. That is, given the choice between two categories, one with twice as many people assigned as the other, it

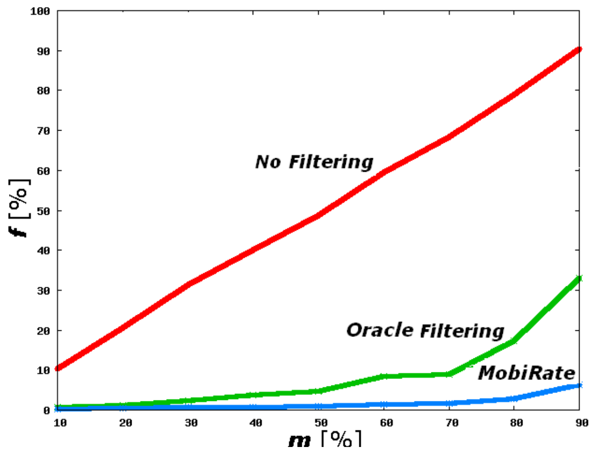


Figure 3. Fraction of compromised interactions f versus the percentage of malicious individuals m.

is twice as likely that we assign the more popular category.

To sum up, any two individuals interact if they share content categories. We assign c categories (out of a possible t) to each individual. We do so in two ways: *random* and *realistic*. By changing c and t, the results (fraction f of compromised interactions) differ little. So we will report below only the results for $c = 2$ and $t = 10$ and will study which factors (other than c and t) impact those results.

- *Which interactions get compromised.* Finally, to determine which interactions get compromised, we need to determine which individuals are malicious and consider that, whenever one interacts with any of those individuals, the completed interaction gets compromised. There are clearly many possibilities of how to select malicious individuals and we consider two extremes. We pick malicious individuals either uniformly at random or by preferentially selecting among those least-social (have the least number of friends). Again, we will refer to those two ways of choosing malicious individuals as *random* and *realistic*, respectively.

To compare MobiRate to other approaches, we consider that simulated devices either exchange recommendations or rely on MobiRate's tamper-evident tables. When devices exchange recommendations, we further consider two extremes: devices either do not filter recommendations or perfectly filter them (they are oracles who know which recommendations are fake and filter them out). In short, we consider that devices use either "No Filtering", "Oracle Filtering", or "MobiRate". Using "Oracle Filtering", a device can still interact with malicious nodes about which it lacks direct experiences or recommendations.

Reducing f

One would expect that how well MobiRate performs (lowers f) mainly depends on the fraction of malicious individuals (which we call m). Figure 3 plots f against m. f increases linearly with m when recommendations are not filtered ("No Filtering" curve), as one expects: the more malicious individuals (the higher m), the more users are to fall victim (the higher f). Both "Oracle Filtering" and MobiRate significantly reduce f. The latter is robust to higher percentages of malicious individuals - it shows negligible f up to 80% of malicious individuals. That is because, in the presence of malicious individuals, it is preferable to rely not on exchanging recommendations but on locally stored ratings which are tamper-evident.

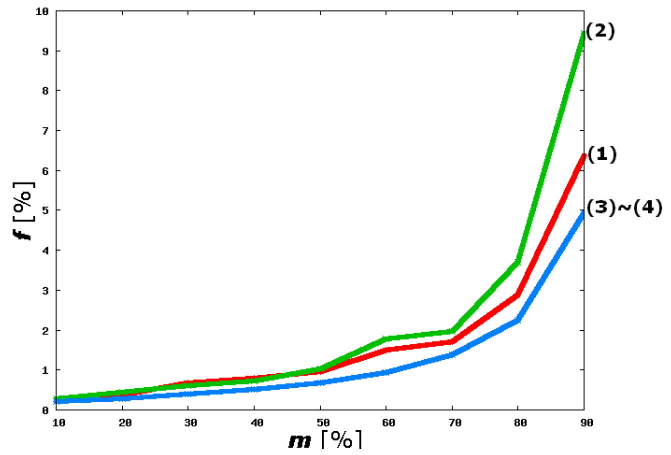


Figure 4. Fraction of compromised interactions f in four simulation setups. By categorizing setups along the two dimensions ($way_{\text{interests}}$, $way_{\text{malicious}}$), we identify four of them: (1) (*realistic*, *realistic*); (2) (*realistic*, *random*); (3) (*random*, *realistic*); and (4) (*random*, *random*). For example, (1) corresponds to a setup in which the way we distribute interests and the way we pick malicious individuals are both realistic. "$(3) \sim (4)$" means that setups (3) and (4) show the same results.

One would also expect that f may be affected by the way we have distributed interests across users ($way_{\text{interests}}$) and the way we have picked malicious individuals ($way_{\text{malicious}}$). We have seen that either of this way can take two extremes (*random*,*realistic*). So we have four possible simulation setups. Figure 4 depicts how f varies across those setups - f shows the same trend for all of them - as one expects, it increases with m. However, the results quantitatively vary across setups. To find out whether the way we pick malicious individuals or the way we distribute interests impacts the most, we run a 2-factorial experiment [14] (whose results have a confidence interval of 95%). We found that the way we pick malicious individuals has the least impact (24%) on f. That is intuitively confirmed by Figure 4 in which (3) is superimposed on (4) - those setups vary in the way we pick malicious individuals ($way_{\text{malicious}}$), and they show the same results - so $way_{\text{malicious}}$ matters little.

f Over Time

We have seen that MobiRate considerably reduces the fraction of compromised interactions. But does MobiRate take considerable time to show those good results? Figure 5 shows how f evolves over (simulation) time if there are 70% of malicious individuals[1]. For "No Filtering", after an initial assessment, f remains constant - if the percentage of malicious

[1]For 30% of malicious individuals, f follows the same trends.

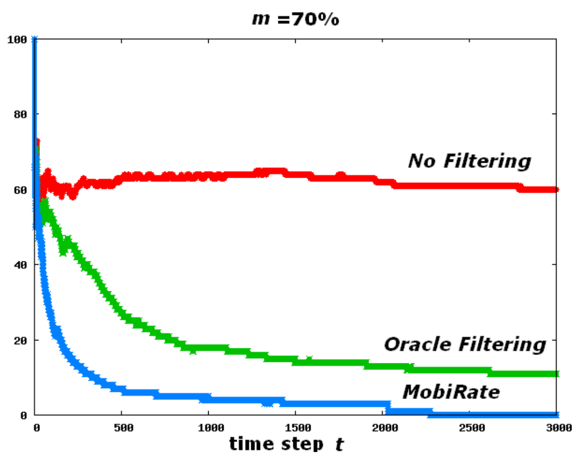


Figure 5. Fraction of compromised interactions f over (simulation) time. 70% **of malicious individuals in the system.**

individuals remains the same, the number of compromised interactions roughly remains the same. For both "Oracle Filtering" and MobiRate, f exponentially decreases. MobiRate shows a steeper decline because, again, it does not rely on exchanging recommendations (limited in the presence of malicious individuals) but relies on locally stored and tamper-evident ratings.

Overhead

The overhead a mobile phone would see in using MobiRate largely depends on two cryptographic techniques: public key encryption and collision-resistant hashing. These techniques can be implemented by different algorithms and, among them, we should pick those two that are both secure and fast. To attain a minimum level of security, America's National Institute of Standards and Technology (NIST) recently suggested SHA-1 (for collision-resistant hashing) and RSA (for public key encryption) with a key of at least 1024 bytes [4]. However, the use of RSA may slow down current models of mobile phones. So, we consider a second public key encryption algorithm - ECDSA [15]. We do so because, security level being equal, compared to RSA, ECDSA uses smaller keys and, consequently, signs messages faster [11].

Storage Overhead. A device that runs MobiRate stores the code for a set of cryptographic algorithms, a rating table, and the gossips it receives from other devices. In the worst case, all this requires about 106KB. That is because: the cryptographic algorithms in JAVA require 53KB for ECDSA or 60KB for RSA [31], a rating table that contains 100 ratings (which is pessimistically high given that expired ratings are discarded) takes 15.4KB, and a set of 100 gossips also requires 15.4KB. So the amount of storage required is negligible, mostly because mobile phones come with GBs of storage nowadays.

Communication Overhead. As for communication overhead experienced by a device, if the device runs: (1) the commitment protocol (at either end), it transmits 158 bytes[2]; (2) the audit protocol as challenger, it transmits 256 bytes; or (3) the audit protocol as responder, it transmits $20 + n \cdot 10$ bytes (where n is the number of ratings being audited). In theory, using Bluetooth version 1.2, devices can transfer 434 kb/s . In practice, environmental conditions (e.g., human bodies that interfere with Bluetooth's frequencies) lower that speed. Still, at a speed as low as 100 kb/s, a mobile phone can carry out the audit protocol (which requires the most number of bytes) in 2.5 milliseconds.

[2]That is because a table row is transmitted. A row takes 158 bytes as it consists of an entry (10 bytes), an authenticator (which requires 128 bytes in the worst case of RSA signature), and an hash value (20 bytes).

Protocol	Nokia 6600 RSA	Nokia 6600 ECDSA	Ericsson P900 RSA	Ericsson P900 ECDSA
Commitment (Rater)	4,41	**1,10**	2,9	**0,6**
Commitment (Rated)	4,55	**2,35**	3,87	**1,45**
Audit (Challenger)	0,28+ $n{\cdot}0,34$	**0,28** + $n \cdot 0,34$	1,94+ $n \cdot 0,18$	**1,70** + $n \cdot 0,18$

Table 2. Execution time (seconds) of MobiRate's protocols on two mobile phones. We compare two public key encryption algorithms: RSA and ECDSA. Using the latter (whose execution time is highlighted), both protocols run faster.

Communication Complexity. MobiRate's protocols use two types of gossips. These gossips have both $O(a)$ message complexity, where a is the average number of "like-minded familiar strangers". As we mentioned at the end of Section "Gossiping Commitments", those individuals are a subset of familiar strangers (a *limited* group of people who one does not know but meets regularly [13]).

Computational Overhead: CPU. Researchers have extensively implemented and evaluated the cryptographic algorithms that MobiRate uses [31]. Based on their results, we compute the execution time of our protocols on two mobile phones (Table 2). The overhead on either phone is acceptable. As one would expect, ECDSA signs messages faster than RSA. However, Moore's Law lends a word of caution on those results: they are meant to demonstrate the feasibility of MobiRate but not to hold in the future - in few months' time mobile phones will become more powerful, and MobiRate is expected to run seamlessly on them. Furthermore, one may observe that the two cryptographic operations used are well-established, and one may consequently imagine a (near) future in which those operations will be partially or fully implemented in hardware for higher performance.

DISCUSSION

Based on the previous results, we now discuss some open questions.

Privacy Concerns. By exchanging their ratings, users reveal people with whom they have interacted, and some users may not feel comfortable doing so for privacy concerns [18]. Our design partially alleviates these concerns because it supports anonymous tokens for identifying users.

Additional attacks. Malicious individuals may not limit themselves to simply being malicious or ignorant, but may also carry out other attacks:

- *Sybil Attackers.* MobiRate is a way of collecting and storing ratings. Those ratings may come not only from honest individuals but also from Sybil attackers (Sybils). A Sybil is a user who takes on multiple identities and pretends to be multiple, distinct users who highly rate each other [9]. To reduce the impact of Sybils, one needs an algorithm that filters out their ratings. We have recently proposed a distributed algorithm for portable devices that does so by arranging ratings in a social network of recommenders [26]. The idea is that the more connected a recommender, the higher the importance of her ratings. Since Sybils happen to be poorly-connected, our algorithm has been proven experimentally robust to their ratings.

- *Dynamic Attackers.* User A may be ignorant or malicious when interacting with B but not so with C. The result is that B singles out A, while C keeps on interacting with A. That situation does not create any conflict between B and C as there is no need for them to share the same opinion on any device (including A).

- *Lying Recommenders or Gossipers.* MobiRate is orthogonal to most existing work that focuses on designing algorithms for filtering lying recommenders. Interestingly, we have shown that even assuming *ideal* filtering algorithms, devices can still fall victim of malicious devices whenever they lack information about them. By contrast, using MobiRate, devices do not need to collect recommendations but rely on locally stored ratings and on gossips to verify the integrity of those ratings. Devices cannot make up gossips simply because any gossip that is cryptographically corrupted is automatically discarded.

- *Slackers.* Recommenders who are willing to contribute and see their ratings forged will likely feel frustrated, and that may turn them into slackers. A slacker is a term for people who escape their fair share of work by not supplying ratings. Intuitively, MobiRate may encourage contribution since it allows its users to rate without being discouraged by forgers. However, this argument needs further research - it should be treated as a testable hypothesis rather than an established fact.

Design of Rating Tables. The way we have designed rating tables guarantees that:

- *Tampered ratings are detected.* In Section "Acting on Gossips", we have seen that if B tampers with its entry e_k, any device is able to detect the alteration by challenging B to produce that entry, thereby obtaining enough evidence to conclude that B has tampered with the entry.

- *Each device keeps only one table.* B may attempt to keep more than one rating table. For example, B may keep one table for ratings by C and another for ratings by D. However, if B were to do so, it would be unable to answer audit challenges (Type 2 gossips). That is because, as we have seen in Section "Acting on Gossips", those challenges require B to return a "*linear* hash chain" of ratings, which is infeasible if ratings come from different tables, as we will formally prove in Theorem 2 of the Appendix.

- *Ratings can change over time.* If A rates B more than once, B appends more than one rating in its table. To limit storage, B may well want to stop its table from growing over time. It may do so in two ways. First, by summarizing ratings using standard optimizations such as Bloom Filters [22]. Second, by discarding expired ratings - ratings older than some specific expiration time. That would make sense since the value of ratings decays over time. By simply assigning each gossip with an expiration time, a device can delete the entries whose gossips have expired but cannot do so for those whose gossips are still valid - witness devices should still be able to scrutinize them. Given that B interacts with some devices more frequently than others, B chooses an expiration time (e.g., few months) that will be large enough to accommodate all of the devices it has interacted (and will interact) with.

CONCLUSION

We proposed a mechanism (MobiRate) that makes it possible for mobile users to run collaborative applications by storing and exchanging tamper-evident ratings. MobiRate is effective in singling out malicious individuals (it is robust up to 90% of malicious individuals in the system) and scales (it entails reasonable storage, communication, and computational overhead).

To further evaluate MobiRate, we are studying how underground passengers (of the order of 1 million) happen to be co-located and how their mobility patterns would impact the performance of MobiRate. To evaluate whether users understand MobiRate's underlying concept and whether they can use it, we are also designing a user study.

Acknowledgments: We thank the anonymous reviewers and our shepherd, Rene Mayrhofer, for their helpful comments. We also thank: Bence Pasztor, Anders Lindgren, and Liam McNamara for their guidance on mobility traces; Shinichi Honiden for his support; Eric Platon, Neal Lathia, and Costin Raiciu for their constructive feedback; and Microsoft Research Cambridge for its financial support.

REFERENCES

1. *Bluetooth SIG. Core Specification v2.1 + EDR*. July 2007.
2. K. Aberer and Z. Despotovic. Managing Trust in a Peer-to-Peer Information System. In *Proc. of the* 10^{th} *ACM International conference on Information and Knowledge Management*, pages 310–317, Atlanta, USA, November 2001.
3. A. L. Barabasi. *Linked: How Everything Is Connected to Everything Else and What It Means*. Penguin, 2003.
4. E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for key management. In *NIST Special Publication. Revised Mar 2007*.
5. A. Bassoli, J. Brewer, K. Martin, P. Dourish, and S. Mainwaring. Underground Aesthetics: Rethinking Urban Computing. *IEEE Pervasive Computing*, 6(3):39–45, 2007.

6. R. Chakravorty, S. Agarwal, S. Banerjee, and I. Pratt. MoB: A mobile bazaar for wide-area wireless services. In *Proceedings of the 11th ACM International Conference on Mobile Computing and Networking (MobiCom)*, page 228242, 2005.
7. B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *Proc. of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 189–204, Stevenson, Washington, USA, 2007.
8. C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC)*, pages 150–157, Minnesota, US, 2000.
9. J. R. Douceur. The Sybil Attack. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems*, pages 251–260, Cambridge, USA, 2002.
10. N. Eagle and A. S. Pentland. Reality mining: sensing complex social systems. *Personal Ubiquitous Computing*, 10(4):255–268, 2006.
11. V. Gupta, M. Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. C. Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet. In *Proc. of the 3rd IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, pages 247–256, Washington, DC, USA, 2005.
12. A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: practical accountability for distributed systems. In *Proc. of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 175–188, Stevenson, Washington, USA, 2007.
13. P. Hui, J. Crowcroft, and E. Yoneki. BUBBLE Rap: Social Based Forwarding in Delay Tolerant Networks. In *Proc. of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, HongKong, 2008.
14. R. K. Jain. *The Art of Computer Systems Performance Analysis*. Wiley.
15. D. B. Johnson and A. J. Menezes. Elliptic curve dsa (ecsda): an enhanced dsa. In *Proc. of the 7th Conference on USENIX Security Symposium (SSYM)*, pages 13–13, San Antonio, Texas, 1998.
16. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proc. of the 12th ACM International Conference on World Wide Web (WWW)*, pages 640–651, Budapest, Hungary, May 2003.
17. G. Kossinets and D. J. Watts. Empirical Analysis of an Evolving Social Network. *Science*, 311(5757):88–90, 2006.
18. N. Lathia, S. Hailes, and L. Capra. Private Distributed Collaborative Filtering using Estimated Concordance Measures. In *Proceedings of ACM Recommender Systems (RecSys)*, 2007.
19. R. Levien and A. Aiken. Attack-resistant trust metrics for public key certification. In *Proc. of the 7th USENIX Security Symposium*, pages 229–241, Berkeley, USA, January 1998.
20. H. Luo, R. Ramjee, P. Sinha, L. E. Li, and S. Lu. UCAN: a unified cellular and ad-hoc network architecture. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 353–367, New York, US, 2003.
21. L. McNamara, C. Mascolo, and L. Capra. Media Sharing based on Colocation Prediction in Urban Transport. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking (MobiCom)*, San Francisco, US, 2008.
22. M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2005.
23. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford University Technical Report, 1998.
24. E. Paulos and E. Goodman. The familiar stranger: anxiety, comfort, and play in public places. In *Proc. of ACM Conference on Human Factors in Computing Systems*, pages 223–230, 2004.
25. D. Quercia, S. Hailes, and L. Capra. B-trust: Bayesian Trust Framework for Pervasive Computing. In *Proc. of the 4th International Conference on Trust Management (iTrust)*, pages 298–312, Pisa, Italy, May 2006. LNCS.
26. D. Quercia, S. Hailes, and L. Capra. Lightweight Distributed Trust Propagation. In *Proc. of the 7th IEEE International Conference on Data Mining (ICDM)*, Omaha, US, October 2007.
27. D. Quercia, S. Hailes, and L. Capra. TRULLO - local trust bootstrapping for ubiquitous devices. In *Proc. of the 4th IEEE International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, Philadelphia, US, August 2007.
28. B. Schneier and J. Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proc. of the 7th Conference on USENIX Security Symposium (SSYM)*, pages 4–4, San Antonio, Texas, 1998.
29. P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.
30. D. Tapscott and A. D. Williams. *Wikinomics: How Mass Collaboration Changes Everything*. Portfolio, Penguin, 2006.
31. S. Tillich and J. Grossschaedl. A Survey of Public-Key Cryptography on J2ME-Enabled Mobile Devices. In *Proc. of the 11th International Symposium of Computer and Information Sciences*, pages 935–944, 2006.

APPENDIX

Audits

Theorem 1. If B tampers with its entry e_k, A is able to detect that tampering by running a commitment protocol on e_k.

Proof. Assume to the contrary that:

1. B tampers with the rating in its k^{th} entry e_k (we indicate the tampered entry as $\tilde{e}_k = (s_k, \tilde{r}_k)$).
2. A and B run the commitment protocol for the entry e_k, and A does not detect the tampering.

After the commitment protocol, A obtains $a_k^B = S_B(s_k, h_k)$, $\tilde{e}_k = (s_k, \tilde{r}_k)$, and h_{k-1}. It then computes $\tilde{h}_k = H(h_{k-1}||s_k||H(\tilde{r}_k))$, arranges $(s_k, \tilde{h_k})$, and compares it to (s_k, h_k) (extracted from a_k^B). During this comparison, A runs into a mismatch and concludes that B has tampered with e_k. That is a contradiction.$\square$

Theorem 2. If B tampers with its l^{th} entry e_l (where $k < l < m$), A is able to detect that tampering by running a commitment protocol on the segment $(e_k, \ldots, e_m)$.

Proof. Assume to the contrary that:

1. B tampers with the rating in its l^{th} entry e_l (we indicate the tampered entry as $\tilde{e}_l = (s_l, \tilde{r}_l)$).
2. A and B run the commitment protocol for the segment $(e_k, \ldots, e_m)$, and A does not detect the tampering.

After the commitment protocol, A obtains $a_k^B = S_B(s_k, h_k)$, $e_k = (s_k, r_k)$, and h_{k-1}. It then computes:

- $h_k = H(h_{k-1}||s_k||H(r_k))$
- $h_{k+1} = H(h_k||s_{k+1}||H(r_{k+1}))$
- $\ldots$
- $h_{l-1} = H(h_{l-2}||s_{l-1}||H(r_{l-1}))$
- $\tilde{h}_l = H(h_{l-1}||s_l||H(\tilde{r}_l))$ - from here, the chain gets compromised
- $\ldots$
- $\tilde{h}_m = H(h_{m-1}||s_m||H(\tilde{r}_m))$

A then arranges $(s_m, \tilde{h}_m)$, and compares it to (s_m, h_m) (extracted from a_m^B). During this comparison, A runs into a mismatch and concludes that B has tampered with at least one of the entries in the segment $(e_k, \ldots, e_m)$. That is a contradiction.$\square$

Completeness

Theorem 3. Eventually, every ignorant device is suspected by each of its like-minded familiar strangers (defined as per Section "Gossiping Commitments").

Proof. Assume to the contrary that:

1. B is ignorant.
2. There is a device C that does not suspect B at time $t' > t, \forall t$

The first assumption (B is ignorant) means that there is a device D that sent a message to B, B did not acknowledge it, and D sent a Type 1 gossip. Eventually, all reached devices (B's like-minded familiar strangers) started suspecting B.

Let t_1 be the first time C receives D's gossip. Given the second assumption, C does not suspect B at time $t_2 > t_1$. But that may only happen if B has engaged in a commitment protocol with C and has correctly acknowledged B's message. That contradicts the first assumption (B is ignorant).□

Theorem 4. Every malicious device is eventually exposed or forever suspected by each of its like-minded familiar strangers (defined as per Section "Gossiping Commitments").

Proof. Assume to the contrary that:

1. B is malicious.
2. There is a device C that does not suspect (or exposes) B at time $t' > t, \forall t$

The first assumption (B is malicious) means that there is a device D that has proof of B being malicious and that distributes evidence as Type 2 gossips. Eventually, all reached devices (B's like-minded familiar strangers) start suspecting B.

Let t_1 be the first time C receives D's gossip. Given the second assumption, C does not suspect B at time $t_2 > t_1$. But that may only happen if B has engaged in an audit protocol with C and has correctly acknowledged C's message. That contradicts the first assumption (B is malicious).□

Accuracy

Theorem 5. No honest device is forever suspected by an honest device.

Proof. Assume to the contrary that there is an honest device B that is forever suspected by A after time t_1.

Let $t_2 > t_1$ be the first time A receives a gossip about B. A runs either the commitment protocol or the audit protocol. However, being B an honest device, B constructs a valid response. But A still suspects B after t_2. That is a contradiction.□

Theorem 6. No honest device is ever exposed by an honest device.

Proof. Assume to the contrary that there is an honest device B that is exposed by another honest device A. Since A is an honest device, the only reason for A exposing B is that A has a proof of B misbehaving. But B, being honest, has not misbehaved. As such, A cannot have any proof of misbehavior and cannot expose B. That is a contradiction.□