Trust Models for Mobile Content-Sharing Applications

Daniele Quercia

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University College London.

Department of Computer Science

University College London

January 2009

For Vilma (in memory)

I, Daniele Quercia, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Using recent technologies such as Bluetooth, mobile users can share digital content (e.g., photos, videos) with other users in proximity. However, to reduce the cognitive load on mobile users, it is important that only appropriate content is stored and presented to them.

This dissertation examines the feasibility of having mobile users filter out irrelevant content by running trust models. A trust model is a piece of software that keeps track of which devices are trusted (for sending quality content) and which are not. Unfortunately, existing trust models are not fit for purpose. Specifically, they lack the ability to: (1) reason about ratings other than binary ratings in a formal way; (2) rely on the trustworthiness of stored third-party recommendations; (3) aggregate recommendations to make accurate predictions of whom to trust; and (4) reason across categories without resorting to ontologies that are shared by all users in the system.

We overcome these shortcomings by designing and evaluating algorithms and protocols with which portable devices are able automatically to maintain information about the reputability of sources of content and to learn from each other's recommendations. More specifically, our contributions are:

1. An algorithm that formally reasons on generic (not necessarily binary) ratings using Bayes' theorem.
2. A set of security protocols with which devices store ratings in (local) tamper-evident tables and are able to check the integrity of those tables through a gossiping protocol.
3. An algorithm that arranges recommendations in a "Web of Trust" and that makes predictions of trustworthiness that are more accurate than existing approaches by using graph-based learning.
4. An algorithm that learns the similarity between any two categories by extracting similarities between the two categories' ratings rather than by requiring a universal ontology. It does so automatically by using Singular Value Decomposition.

We combine these algorithms and protocols and, using real-world mobility and social network data, we evaluate the effectiveness of our proposal in allowing mobile users to select reputable sources of content. We further examine the feasibility of implementing our proposal on current mobile phones by examining the storage and computational overhead it entails. We conclude that our proposal is both feasible to implement and performs better across a range of parameters than a number of current alternatives.

Acknowledgements

Ignorance is the first requisite of the ideal PhD student. In that respect at least I was eminently qualified to embark upon this research. My ignorance made doing research a fun experience. For really I hardly remember “working”. I remember only feeling absurdly pleased with my nonsense, for which I am eager to spread some credits (while of course remaining accountable for all blame).

Thanks to Steve Hailes and Licia Capra whom I am proud to call my supervisors. They thoughtfully guided my research from nonsense to sense. Thanks to Steve for relentlessly bulling me into clarifying the purpose and execution of my research ideas. He also patiently let me disappear for many months while I was working on my PhD. Everyone should have a supervisor as generous as Steve. Licia first got me thinking about applying trust models to portable devices with her remarkable writings on the subject. Licia’s work remains a touchstone for me. She has commented all my papers with amazing tact. Her red pen was both a mighty sword and an artist’s brush. She also weathered my minor crisis of confidence (and a major one). Thanks, Licia.

Special thanks goes to Cecilia Mascolo for always “being in my corner” and for asking me everyday (from day one) when my thesis would be done. Thanks to UCL folks who supported the creation of a research blog and to Neal who was my partner in crime in creating and managing the blog. A printed shout goes as well to the following mates who provided crucial assistance, the right fragment at the right time. At UCL: Bozena, Clovis, Costin, Chris, Elias, Franco, Genaina, Ida, James, Manish, Matteo, Mike, Piers, Stef, Socrates, and Torsten. At NII: Eric (Platon), Eric (Tschetter), Christian, Fuyuki, Prof. Honiden, and Soo Ling. At IIIA: Andrea, Carles, Claudio, Jordi, and Ronny. At PoliTO: Laura, Paolo and Tania.

Conducting research takes money and feedbacks. Thanks to Microsoft Research Cambridge for its financial support. Thanks to the anonymous reviewers who carefully read and commented on my publications. There are fewer errors thanks to you. I am so proud to be part of this community.

I could not have asked for better friends than the following. Andre was one of the first people I met when I moved to London, and my experience of living here is inextricable from our friendship. In (enchanted) Torino: Antonello, Andrea, Chiara, Enrico, and Sonia (thanks for being at the ends of a phone when needed - I love you all!); and Max - I am in debt with him, and he knows why. In Barcelona: Antonietta, Chema and Lorenzo (thanks for your tireless talk; Lorenzo: how about dancing on tables, sporting Aussie Bum, and drinking caipirinhas “en el chiringuito” again next year?). In Karlsruhe: Flavio and Davidino (thanks for generously sharing your eloquence and intellect). In Tokyo: Tom (your

generosity defies description).

“It is sheer good fortune to miss somebody long before they leave.” I miss Vilma more than ever. This thesis is for her (in memory).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Researchers are realizing that mobile devices may engage people in many different ways. For example, mobile devices allow people to take photos or shoot videos and distribute them to their local communities at very low cost. Content distribution may help to engage people in, for example, urban planning or creative expression [RBE$^+$06, Rhe02]. But what happens if everybody is distributing content? In that case, to paraphrase Italo Calvino, we would live in an unending rainfall of content [Cal96].

To avoid content overload, we need new ways of filtering content (of deciding which content to accept). To this end, researchers have proposed general infrastructures with which portable devices collaboratively filter content [CABP05, LRS$^+$03, MMC08]. At the heart of those designs is a *centralized* reputation system that stores ratings about content producers. For locating reputable producers, 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 [CABP05, MMC08].

One way of decentralizing a reputation system is to have each user run a trust model on her device [QHC06a]. A trust model is a piece of software that keeps track of who provides quality content and who does not.

What We Mean by Trust

Before designing a trust model, one needs to clarify what the word 'trust' means. To this end, let us start from a commonly accepted definition [Gam98]: *" [Trust] (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before [we] can monitor such action (or independently of his capacity of ever be able to monitor it) and in a context in which it affects [our] own action".* From this definition, three properties of trust emerge: subjectiveness, context-dependence, and dynamism. The same behavior may lead to different trust levels in different trusting entities, hence *subjectiveness* qualifies trust. As trust (e.g., in giving good advice) in one context (e.g., academia) does not necessarily transfer to another context (e.g., industry), we add *context-dependence* to the list of trust properties. Finally, the fact that trust increases after successful observations, while it decays over time, exemplifies its *dynamism*. As a result, trust evolution must embed the notion of *time*.

By adapting this general definition of trust to the case of two mobile users A and B who share digital content, one concludes that:

"A trusts B" translates into "A relies on B as a producer of quality content".

The three basic properties of trust are still pertinent: A *subjectively* decides whether to rely on B, it does so only for content in specific *categories* (e.g., music genres), and ages its reliance over *time* to, for example, prevent B from unduly capitalizing on past good interactions/exchanges.

This generic definition of trust can then be translated into practice by considering two examples.

Example 1 - Electronic Ads in Antique Markets

Consider a typical antique market in the United Kingdom. Visitors look for items they are interested in and have difficulty in finding them because a typical market is huge - thousands of visitors and hundreds of sellers. The use of portable devices may help the visitors' quest. On the one hand, stall holders could use their devices to disseminate electronic ads and promote their wares, while visitors could collect those ads to make up their minds about which stalls to visit. Then, to avoid being overloaded by ads, visitors could accept only those ads that are of relevance to their users. They would do so by running trust models - pieces of software that filter ads by keeping track of which sellers send relevant ads.

Example 2 - Songs of Emerging Musicians

To get some free publicity, emerging artists upload their latest tracks into publicly-available WiFi hotspots and add the date of their next gig as a note to the track [BBM+07, Lia07]. Mobile users then discover these songs through custom searches or through their profiles (users specify their likes and dislikes beforehand, and their devices search for them). The problem is that the amount of information that matches a user's query may be overwhelming. To give end-users a good pervasive experience, content should be ranked so that, the more reputable the source that produced it, the higher up its ranking. Trust models may be used to dynamically assess a source's reputation - the extent to which a source produces quality content.

Research Hypothesis

In those two examples, people share content, and that content may be of poor quality simply because its producers are either ignorant (not knowledgable enough to make quality content available) or malicious (motivated to disrupt content sharing communities). Trust models promise to isolate such individuals. More specifically:

We posit that, by running trust models, mobile users increase the relevance of the content they find.

This implies: protecting users from maliciously created content (*robustness*); and delivering content that matches users' interests (*accuracy*). We will test our research hypothesis along both dimensions of robustness and accuracy.

1.2 Research Goals

To understand our research problem, consider that whenever A has to decide whether to rely on B, A should: identify B; set its trust for B; and decide whether to rely on B. We are interested in one of these three steps: how A sets its trust for B. We propose that A does so by reasoning both on personal experiences and on third-party recommendations. That is, A should go through four steps: (1) reason on personal experiences; (2) reason across categories; (3) reason on recommendations; and (4) store recommendations.

Our research goal is to design algorithms for portable devices that carry out those four steps (see Figure 1.1).

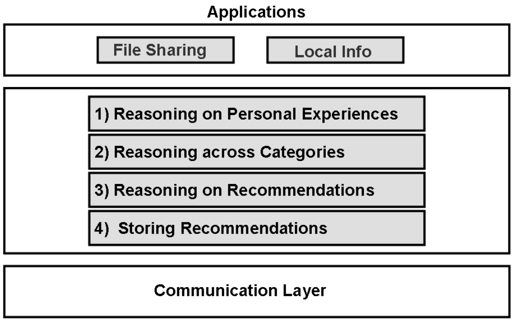

Applications
File Sharing
Local Info
1) Reasoning on Personal Experiences
2) Reasoning across Categories
3) Reasoning on Recommendations
4) Storing Recommendations
Communication Layer

Figure 1.1: Our Research Goal: Four algorithms with which portable devices set up trust.

Research Goal 1: Reasoning on Personal Experiences

Design and evaluate an algorithm that reasons on personal experiences.

Whenever A receives content from B, A needs to rate content quality and accordingly update its trust. As we will see in Chapter 2, existing algorithms are informal in the sense that they update trust using hand-crafted formulae. Plus, if A uses those algorithms to set its trust for B, A has to input a binary rating (good or bad); however, A may well wish to judge B on a more fine-grained scale. So our first goal is to design an algorithm that reasons on personal experiences of direct interactions, accepts generic ratings (not necessarily binary) of those experiences and, in contrast to most existing solutions, is based on formal foundations. It is often preferable to use well-grounded algorithms as opposed to hand-crafted ones because well-grounded algorithms are ready-made tools based on sound principles (founded on strong theoretical basis) that have been peer-reviewed, widely used and tested. By contrast, new formulae are often crafted by researchers who, being human, are error-prone beings.

Research Goal 2: Reasoning across Categories

Design and evaluate an algorithm that "ports" ratings across categories.

Say that A received "financial news" from B and found it interesting. B now produces "economic news". From its past experience, can A conclude that the economic news is also of good quality? A may well conclude so since "economics" and "finance" are (semantically) similar. In general, A needs to reason across categories, that is, to “convert” its trust from one category to the other. To do so, A needs to determine to what extent categories are similar. To decide whether two categories are similar, existing algorithms typically use an ontology (e.g., a taxonomy of content categories). However, in so doing, they require that the same ontology is shared by all users (which is hardly the case in reality) and that the ontology does not change over time. Our second goal is to design an algorithm that learns category similarities directly from ratings without using an explicit ontology.

Research Goal 3: Reasoning on Recommendations

Design and evaluate an algorithm that reasons on recommendations.
After collecting recommendations, A should reason on them for making trust-informed predictions. A robust way of doing so is to have A collect trust ratings and arrange them in a network called “Web of Trust”. A web of trust is a network of trust relationships: we trust (link to) only a handful of other people; these people, in turn, trust (link to) a limited number of other individuals; overall, these trust relationships form a network (a web of trust) of individuals linked by trust relationships. Upon the “Web of Trust”, A may run *trust propagation* algorithms to accurately predict trust ratings that are missing. In Chapter 3, we will show that existing ways of propagating trust cannot be readily applied in mobile computing because they are usually designed to work on a centrally stored Web of Trust and to run on high-end machines. Our next goal is then to design an algorithm with which portable devices reason on recommendations in a fully decentralized way.

Research Goal 4: Storing Recommendations

Design and evaluate a set of protocols that securely collect and store recommendations.
In addition to learning from their own experiences, devices learn from each other by storing and exchanging recommendations. However, devices may well be *selfish* - they refuse to exchange recommendations - or *malicious* - they lie and make available untruthful recommendations. Previous work suggests that, to counter selfish devices, recommendations should be replicated across devices and, to counter malicious devices, algorithms that filter away untruthful recommendations should be designed [ARH00, Cap04, LI04]. Both of those propositions may be unrealistic because: (1) replicating recommendation introduces traffic to which mobile networks may yield; and (2) filtering away untruthful recommendations requires a critical mass of recommendations, and devices often cannot carry out such filtering because they lack sufficient knowledge for doing so (Chapter 4). Our second goal is thus to design a set of protocols that collect and store recommendations and that, while doing so, they are robust to both selfish and malicious users.

1.3 Assumptions

In meeting these four goals, we assume that portable devices have already ways of:

- *Discovering content or services.* For example, to discover services, devices may could use: (1) Chakraborty and Finin’s proposal [CF06] with which they cache service advertisements and forward service requests in a fully decentralized (peer-to-peer) way; or (2) Sailhan and Issarny’s solution with which part of the devices cooperatively act as service directories, and they do so in a scalable way [SI05].

- *Rating experiences.* Devices may rate the content they receive by explicitly asking users for ratings (e.g., in the form of thumbs up or thumbs down) or by inferring ratings from users' behavior (e.g., by monitoring whether users have permanently stored or deleted content).

- *Identifying other devices.* To protect user anonymity, devices may use anonymous tokens (e.g., public keys) to identify themselves. However, if a device is malicious, it may take on multiple tokens (identities) and pretend to be multiple, distinct users. In Chapter 6, we discuss this problem and propose a remedy for it.

1.4 Contributions

Under those assumptions we will make five contributions. The first four consist of designing four algorithms (the four numbered blocks in Figure 1.2), and the fifth is the evaluation of those algorithms as a whole.

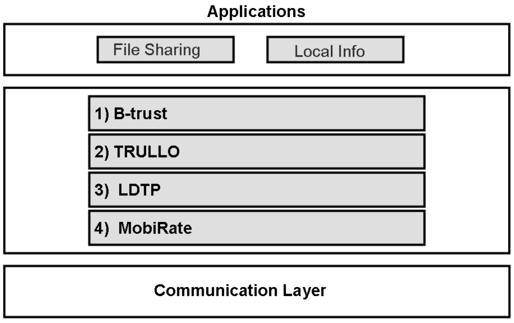


Figure 1.2: Our Research Contribution: Four algorithms with which portable devices establish trust.

Contribution 1: Reasoning on Personal Experiences (B-trust)

In Chapter 2, we propose an algorithm with which portable devices reason on their own personal experiences. The ratings of those experiences are generic - they are not binary but are expressed on a generic scale of n. Reasoning consists of a formal Bayesian process applied both on input of ratings and over time. Reasoning over time is necessary to discount old experiences (the older the reputation assessments, the lower their value).

Contribution 2: Reasoning across Categories (TRULLO)

In Chapter 2, we also propose an algorithm with which portable devices "port" their ratings (of personal or third-party experiences) across categories without using an explicit ontology. The idea is that devices learn from their ratings statistical properties upon which they then discover which categories are similar. Upon learned similarity, devices are then able to infer accurate trust values in unknown categories.

Contribution 3: Reasoning on Recommendations (LDTP)

In Chapter 3, we put forward a way with which devices set their trust for unknown devices, and they do so from third-party recommendations. Each device organizes ratings of content producers that it knows and trusts in a graph (called "Web of Trust"). It then uses a graph-based learning technique to form opinions about content producers with whom it has never interacted before. This way of aggregating recommendations allows for subjective trust. That is, it makes it possible for two individuals to have different opinions about the trustworthiness of the same person.

Contribution 4: Storing Recommendations (MobiRate)

In Chapter 4, we propose a new set of protocols with which portable devices store and exchange recommendations in the presence of selfish users (who do not share their recommendations) and of malicious users (who make untruthful recommendations available). The idea is that devices store recommendations in (local) tamper-evident tables and check the integrity of those tables through a gossiping protocol.

Contribution 5: Evaluation of Our Algorithms

Finally, in Chapter 5, we evaluate those four contributions as a whole. Using real mobility and social network data, we show that our proposal is robust against both malicious and selfish individuals and runs reasonably quickly on current mobile phones.

1.5 Publications

We have published both results that are strictly related to this thesis (Table 1.1) and results about problems that revolve around it (Table 1.2).

Dissemination of this Thesis

	Contribution	**Publication(s)**
Chapter 2	A Bayesian model for reasoning on personal experiences (expressed as ratings on a generic scale - not necessarily binary). An algorithm for porting ratings across categories.	D. Quercia, S. Hailes, and L. Capra. B-trust: Bayesian Trust Framework for Pervasive Computing. In *Proceedings of the* 4^{th} *International Conference on Trust Management (iTrust)*, pages 298-312, Pisa, Italy, May 2006. LNCS. D. Quercia, S. Hailes, and L. Capra. TRULLO - local trust bootstrapping for pervasive devices. In *Proceedings of the* 4^{th} *IEEE Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous)*, Philadelphia, USA, August 2007.
Chapter 3	Algorithms for reasoning on recommendations.	D. Quercia, S. Hailes, and L. Capra. Lightweight Distributed Trust Propagation. In *Proceedings of the* 7^{th} *IEEE International Conference on Data Mining (ICDM)*. Ohama, USA, October 2007.
Chapter 4	Protocols for storing and sharing recommendations in a secure and distributed way.	D. Quercia, S. Hailes, and L. Capra. MobiRate: Making Mobile Raters Stick to their Word. In *Proceedings of the* 10^{th} *ACM Conference on Ubiquitous Computing (Ubicomp)*, Seoul, South Korea, September 2008.
Chapter 5	Evaluation of our algorithms on real-world data.	D. Quercia, S. Hailes, and L. Capra. StrongTrust: Defending Mobile Communities. In *Submission*, 2008.

Table 1.1: Publications of This Thesis

Further Published Work

Chapter 6 a) Identification	A distributed identification scheme that ensures unique and anonymous identifiers.	D. Quercia, S. Hailes, and L. Capra. TATA: Towards Anonymous Trusted Authentication. In *Proceedings of the* 4^{th} *International Conference on Trust Management (iTrust)*, pages 298-312, Pisa, Italy, May 2006. LNCS.
b) Deciding whether to trust	A trust-informed model for deciding among more than two actions (not only "trust/don't trust") grounded on economics literature.	D. Quercia, S. Hailes. MATE: Mobility and Adaptation with Trust and Expected-utility. *International Journal of Internet Technology and Secured Transactions, 2007*. D. Quercia and S. Hailes. Risk Aware Decision Framework for Trusted Mobile Interactions. In *Proceedings of the* 1^{st} *IEEE/CreateNet International Workshop on the Value of Security (SECOVAL).*
c) Applications	Preliminary examples of using trust models for sharing bandwidth among portable devices.	D. Quercia, M. Lad, S. Hailes, L. Capra, and S. Bhatti. Survivable wireless networking - autonomic bandwidth sharing in mesh networks. *BT Technology Journal.* Springer Verlag. 2006. D. Quercia, M. Lad, S. Hailes, L. Capra, and S. Bhatti. STRUDEL: Supporting Trust in the Dynamic Establishment of peering coaLitions. In *Proceedings of the* 21^{st} *ACM Symposium on Applied Computing (SAC).* April 2006. Dijon, France.

Table 1.2: Publications Outside of the Thesis' Scope but Related to it

1.6 Structure of the Thesis

Chapter 1 has introduced the research problem of how co-located mobile users share content and has touched on why existing solutions fall short.

Chapter 2, Chapter 3, and Chapter 4 will propose four algorithms with which portable devices effectively share digital content.

Chapter 5 will evaluate the robustness of our algorithms against eight attacks to which content-sharing mobile applications are vulnerable.

Chapter 6 will summarize our work, point out its limitations, and draw a research agenda.

Chapter 2

Reasoning on Personal Experiences Across Categories

The main goal of trust models is to keep track of reputable sources of content. To do so, trust models need to record their experiences and produce trust assessments from them. Algorithmically, this translates into rating experiences and feeding the ratings into a *trust updating* algorithm (first block in Figure 2.1). Then, since digital content spans different content categories, users express ratings in specific categories and, consequently, their devices should be able to handle ratings across categories. One way of doing so is for devices to port ratings (both of personal experiences and of recommendations) from one category to another. That may be done by determining the extent to which the two categories are similar: similar ratings may hold in similar categories.

Problem Statement: How a mobile user A aggregates ratings about user B to produce trust assessments, and how it "ports" ratings across categories. By port, we mean the ability of converting ratings of one content category into ratings of another category.

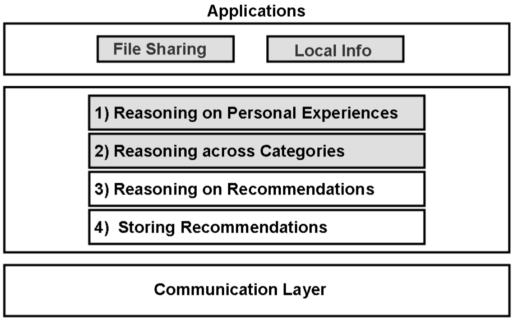

Applications
File Sharing
Local Info
1) Reasoning on Personal Experiences
2) Reasoning across Categories
3) Reasoning on Recommendations
4) Storing Recommendations
Communication Layer

Figure 2.1: Research Goals 1 and 2: Reasoning on Personal Experiences and Across Categories.

2.1 Existing Solutions to Reasoning on Personal Experiences

The body of work on updating trust is littered with mechanisms that are often based on social (human) considerations in that they evolve trust based on direct experiences (as is commonly the case in society), and they integrate the classical *three dimensions* of context, subjectiveness, and time, which Chapter 1 introduced. Although foundational, most of existing approaches are ad-hoc in the sense that they update trust with *hand-crafted formulae*.

A broader concept of trust in the computer science arena began with work in the early 1990s by Marsh [Mar94]. Abdul-Rahman and Hailes first proposed the use of recommendations for managing trust [ARH97] and then introduced a comprehensive distributed trust model for online environments [ARH00], based on Marsh's model. From direct experiences and recommendations, each entity forms its trust and, as such, is also able to exclude untrustworthy devices. Although foundational, the previous approach suffered from being rather too architectural in style: they lacked processes for trust evolution, for example.

Within the SECURE project, Carbone et al. proposed a formal model for trust formation, evolution, and propagation based on a policy language [CNS03]. They also have thrown light on a previously unexplored aspect: the distinction between trust levels and trust confidence.We regard such distinction as fundamental and, thus, preserve it in our Bayesian formalization of trust evolution. The design of frameworks resistant to attacks is not a common occurrence in literature. The most felicitous example we find in Liu and Issarny's work [LI04]. They proposed a (reputation-based) trust framework that is robust to both defamation and collusion attacks.

Ideally, one would prefer formal algorithms to hand-crafted ones because the former have been peer-reviewed, widely tested and founded on strong theoretical basis. That is why Mui *et al.* [MMA$^+$01] proposed a Bayesian formalization. However, two issues remained unsolved: they considered only binary input ratings of personal experiences and did not discount them over time. By doing so, they made it easier to use Bayesian reasoning but they oversimplified reality by assuming that: (1) one cannot profit from expressing ratings on a generic scale (not necessarily binary); and (2) old and new ratings are worth to the same extent. However, intuitively: (1) generic ratings are more expressive and, as such, produce better-informed trust assessments [Cap04]; and (2) one may well prefer new ratings to old ones [BB04]. Buchegger and Le Boudec [BB04] tackled the issue of old ratings but not that of fine-grained ratings: they proposed a Bayesian trust model with which portable devices keep track of who cooperates and, while doing so, they age reputation information; however, they are able to accept binary ratings only in input.

Furthermore, users express rating in specific content categories (e.g., political news, financial news), and existing trust algorithms exploit this fact: they decide which categories are similar to then port ratings across categories. To decide whether two categories are similar, those algorithms typically use an ontology (e.g., a taxonomy of content categories). To see how, consider two recent approaches that do so: the first [Cap05, LI04] defines similarity between any two categories in an ontology as the distance between the two corresponding nodes; the second approach [KR03] draws category similarity based on a direct graph of categories (a less-constrained structure than a tree) whose weights have to be, however, manually set by device users. The researchers who proposed the first approach have acknowledged that the idea that the same ontology is shared by all users is unrealistic and that it is also improbable that those users will continue to agree on that ontology for all time (i.e., the ontology is not supposed to change over time). Those of the second approach concede that, on poor usability grounds alone, their solution has to be automated.

What is missing: A *formal* way of updating trust that takes in input ratings of outcome that are *generic* (not necessarily binary) and that are handled across categories *without relying on an ontology* shared by all users.

2.2 Our Proposal for Reasoning on Personal Experiences: B-trust

There is a literature at the intersection of philosophy, statistics, and economics that explores the rationality of making decisions [Ber85, CH04, Sim79]. This literature has showed that the way people process information and make subjective assessments is accurately modeled by Bayes' theorem [GGT06]. In machine learning, this theorem has also been widely used to draw strong assessments from sparse data [Bis06]. Since we need to draw trust assessments from sparse interactions, we introduce a new model that is based on Bayes' theorem - hence its name B-trust [QHC06a] (first block in Figure 2.2).

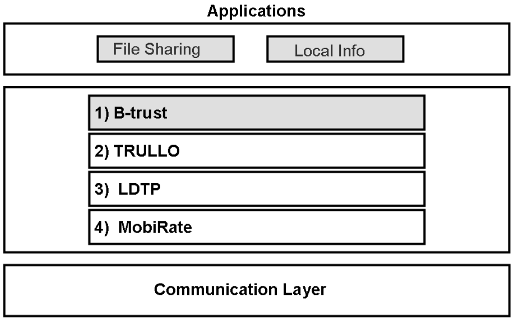


Figure 2.2: Contribution 1: Reasoning on Personal Experiences (B-trust).

2.2.1 Binary Updating

To see how B-trust works, consider that, after rating B's content, A should update its trust for B. To do so, A uses Bayes' theorem. As an example of application of the theorem, consider that B has sent **g**ood content to A, and that A has to consequently updates its probability $p_{\mathbf{t}}$ of "**t**rusting B". It does so by taking the old probability $p_{\mathbf{t}}$ and multiplying it by $L_{\mathbf{g}|\mathbf{t}}$ - the likelihood that **g**ood content comes from **t**rusted sources. If we leave out a proportionality constant at the denominator, the updating is:

$$p_{\mathbf{t}} \propto p_{\mathbf{t}} \cdot L_{\mathbf{g}|\mathbf{t}} \tag{2.1}$$

More formally, the posterior probability $p_{\mathbf{t}}$ is proportional to the product of the prior probability ($p_{\mathbf{t}}$) and the likelihood ($L_{\mathbf{g}|\mathbf{t}}$). For example, upon receiving **g**ood content from B, A updates its probability of B being trusted $p_{\mathbf{t}}$ and of B being untrusted $p_{\mathbf{u}}$ as follows.

$$p_{\mathbf{t}} \propto p_{\mathbf{t}} \cdot HighValue$$
$$p_{\mathbf{u}} \propto p_{\mathbf{u}} \cdot LowValue$$

In words, the new probability of B being trusted/untrusted is equal to the old probability multiplied by the likelihood of good content coming from trusted/untrusted sources: the likelihood of good content coming from trusted sources ($L_{\mathbf{g|t}}$) is high, while that of good content coming from untrusted sources ($L_{\mathbf{g|t}}$) is low.

How does A set likelihood $L_{\mathbf{g|t}}$? Common sense would suggest that **g**ood content usually comes from **t**rusted sources. As such, $L_{\mathbf{g|t}}$ is high. However, A does not set likelihoods according to common sense, but learns them while interacting. It does so by counting the number of times what type of content comes from what type of source (e.g., counting the number of times good content comes from trustworthy sources). Next, we will see how A does so by extending Formula 2.1 to the case of ratings on a generic scale of n (not necessarily binary).

2.2.2 Updating on a generic scale

Content cannot always be of good quality - it could also be of bad quality. More generally, content ratings are on a scale of n. To accommodate this, we define: $p_{\mathbf{t}}$ as the probability of "rating B at level t" and $L_{\mathbf{q|t}}$ as the likelihood that content rated as q comes from sources rated as t (where t and q are discrete numbers in $[1, n]$). So, whenever A receives content of quality **q** from B, it updates its probability $p_{\mathbf{t}}$ of "rating B as t" by taking the old $p_{\mathbf{t}}$ and multiplying it by likelihood $L_{\mathbf{q|t}}$. The updating is:

$$p_{\mathbf{t}} \propto p_{\mathbf{t}} \cdot L_{\mathbf{q|t}} \tag{2.2}$$

For example, if A's rating of B is 8 ($q = 8$) on a discrete scale $[1 - 10]$, A updates its profile on B $(p_1, p_2, \ldots, p_9, p_{10})$ as follows:

$$p_1 \propto p_1 \cdot L_{8|1}$$

$$\ldots$$

$$p_{10} \propto p_{10} \cdot L_{8|10}$$

That is, to update the probabilities in B's profile, A multiplies the old probabilities p_i's by the corresponding likelihoods. A then updates its likelihoods by keeping track of which type of content comes from which type of sources. A does so by recording $o_{\mathbf{q|t}}$ (the number of times content rated as q has come from sources rated as t). For example, whenever A rates content as 8, it updates $o_{\mathbf{8|t}}$. More specifically, $\forall t \in [1, l]$, A sets $o_{\mathbf{8|t}} = o_{\mathbf{8|t}} + p_{\mathbf{t}}$. By doing so, A is then able to compute its likelihoods:

$$L_{\mathbf{q|t}} = \frac{o_{\mathbf{q|t}}}{\sum_i o_{\mathbf{q|i}}} \tag{2.3}$$

However, if A is new to the system, it has not set any of its $o_{\mathbf{q|t}}$, but needs to do so to apply our formulae. To see how A may reasonably set its initial $o_{\mathbf{q|t}}$, consider that quality content (high q) goes hand in hand with trusted producers (high t): quality content often comes from trusted producers. That is, small $|q - t|$ differences are more likely than big differences. That is why we consider that A sets its $o_{\mathbf{q|t}}$ as follows:

$$\begin{cases} \text{high } o_{\mathbf{q|t}}, & \text{if } |q - t| \text{ is low} \\ \text{low } o_{\mathbf{q|t}}, & \text{if } |q - t| \text{ is high} \end{cases}$$

More formally, A sets $o_{\mathbf{q}|\mathbf{t}}$ as being inversely proportional to $|q-t|$:

$$o_{\mathbf{q}|\mathbf{t}} = \frac{init}{|q-t|+1} \tag{2.4}$$

For example:

$$o_{\mathbf{q}|\mathbf{t}} = \begin{cases} o_{1|1} = & init \\ \cdots & \\ o_{1|10} = & \frac{init}{10} \end{cases}$$

One needs to set $init$ depending on what extent the initial ratings should persist: the smaller the $init$, the longer the impact of initial ratings. In our evaluation, we will set $init$ to a fixed value. However, other choices are possible - one may learn the optimal $init$ by using, for example, reinforcement learning. Reinforcement learning algorithms could learn by themselves in a trial-and-error fashion by iterating through possible values of $init$ and remembering the best ones. Reinforcement learning has been successfully applied to problems in operations research, control and robotics [KLM96].

Also, to avoid excessive capitalization on past good behavior, one should place greater emphasis on new experiences [BB02, LI04]. Conveniently, Bayesian reasoning does not weight old and new ratings in the same way: new ratings impact more than old ones. That is because old ratings are all summarized into a profile, and each new rating impacts the whole profile. People make subjective assessments in a similar way - they are likely to remember new experiences more than old ones [BLDDJ99, FBT$^+$99]. That is one of the reasons why Bayesian reasoning effectively models human reasoning [GGT06].

2.3 Our Proposal for Reasoning Across Categories: TRULLO

So far we have considered that portable devices reason on personal experiences in a fixed content category (they process ratings about producers of, say, only "economic news"). However, in reality, devices need to reason on more than one category - they need to process ratings about producers of, say, "economic news" and "financial news". That is why we now design an algorithm (called TRULLO) with which devices learn the similarity between two categories by extracting (statistical) similarities between the two categories' ratings [QHC07b]. More specifically, given a matrix of ratings across categories, we need to learn similarities between categories. The problem is that the matrix would likely contain a mixture of signal and noise. That is why we choose to use the Singular Value Decomposition (hereafter SVD or 'the decomposition'). SVD has been repeatedly shown to improve similarity measures (e.g., the similarity of time series [KP00] and of words in documents [FDD$^+$88]) and to be robust to noisy input (i.e., to capture the signal and remove the noise [HDWC99]).

2.3.1 Assumption

We assume that it is given a way of differentiating one category from another. Devices do not need to organize categories in a taxonomy but they simply need to distinguish categories. One simple way of doing so is to manage a directed graph whose nodes are categories and whose edges are keywords [CCDG05]. By selecting the transitions that correspond to the keywords in the category description, one ends up in the node that represents the category.

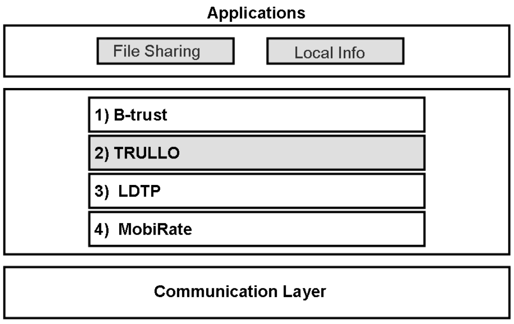


Figure 2.3: Contribution 2: Reasoning across Categories (TRULLO).

2.3.2 Brief Description

To see how TRULLO works, consider that we have a set of ratings in certain categories (that we know from past experiences or from recommendations) and, on those ratings, we need to initialize *one unknown* rating in a new content category. TRULLO does so by using the decomposition. The decomposition expresses the known ratings as linear combination of numeric “features”. Then, those features are combined to initialize the unknown rating. A desirable property of the decomposition is that numeric features are not specified by users but are automatically extracted. That is why we say that TRULLO bootstraps trust *latently*.

More formally, consider that A places its ratings into a matrix N: ratings of n_p producers of content in n_c categories. Each element N_{iy} is A's rating for B (the i^{th} producer) for sending content about “economics” (the y^{th} category). Applying SVD on the matrix N means decomposing it into three other matrices (see Figure 2.4):

$$N = O \cdot P \cdot Q^T.$$

That is, the $n_p \times n_c$ matrix N is decomposed into:

- a producer-by-feature matrix O $[n_p \times n_f]$. In Figure 2.4, O's first row consists of the trust ratings decomposed across features.
- a diagonal feature-by-feature matrix P $[n_f \times n_f]$. The diagonal lists (in *descending* order) the extent to which each feature impacts (matters for) trust assessment[1].
- a feature-by-category matrix Q^T $[n_f \times n_c]$. In Figure 2.4, Q's second column is the extent to which each feature matters when assessing trust in category c_x.

To see how TRULLO uses those matrices, let us assume, for the sake of argument, that A does not know its rating for B in “economic news” (N's highlighted value in Figure 2.4). A predicts this rating by extracting features from the ratings it knows (by decomposing N into three matrices). From the decomposition, for each feature, A learns the feature's importance (P's diagonal) and the feature's

[1] By definition, the number of feature is $n_f = min\{n_p, n_c\}$ [FMM77]; however, in most applications $n_p > n_c$ and thus $n_f = n_c$.

$$N=\begin{array}{c|cccc} \text{producers} \downarrow \; \text{categories} \rightarrow & c_w & c_x & c_y & c_z \\ \hline B & \mathbf{0.67} & \boxed{\underline{\mathit{0.71}}} & \mathbf{0.49} & \mathbf{0.97} \\ C & 0.37 & 0.92 & 0.8 & 0.82 \\ D & 0.3 & 0.9 & 0.21 & 0.42 \\ E & 0.23 & 0.98 & 0.24 & 0.32 \end{array}$$

Unknow rating

$$O=\begin{array}{c|cccc} \text{producers} \downarrow \; \text{features} \rightarrow & & & & \\ \hline & \boxed{0.5629} & \boxed{-0.5690} & \boxed{0.5701} & \boxed{-0.1853} \\ & 0.5958 & -0.2133 & -0.7673 & 0.1043 \\ & 0.4071 & 0.4534 & 0.2904 & 0.7378 \\ & 0.4030 & 0.6520 & 0.0446 & -0.6406 \end{array}$$

Trust for B across features

$$P=\begin{array}{c|cccc} \text{features} \downarrow \; \text{features} \rightarrow & f_1 & f_2 & f_3 & f_4 \\ \hline & \mathbf{2.4972} & 0 & 0 & 0 \\ & 0 & \mathbf{0.6136} & 0 & 0 \\ & 0 & 0 & \mathbf{0.3330} & 0 \\ & 0 & 0 & 0 & 0.0169 \end{array}$$

$$Q^T=\begin{array}{c|cccc} \text{features} \downarrow \; \text{categories} \rightarrow & c_w & c_x & c_y & c_z \\ \hline f_1 & \mathbf{0.3253} & \boxed{\underline{\mathit{-0.2838}}} & \mathbf{0.5869} & \mathbf{-0.6849} \\ f_2 & \mathbf{0.6844} & \boxed{\underline{\mathit{0.7282}}} & \mathbf{0.0118} & \mathbf{0.0334} \\ f_3 & \mathbf{0.3743} & \boxed{\underline{\mathit{-0.3222}}} & \mathbf{-0.7892} & \mathbf{-0.3650} \\ f_4 & 0.5344 & \boxed{-0.5341} & 0.1803 & 0.6297 \end{array}$$

Figure 2.4: Singular Value Decomposition (SVD). TRULLO gathers known ratings in a producer-by-category matrix N, upon which it then applies the decomposition. Based on the three resulting matrices, it then reconstructs the rating to be initialized. The resulting matrices are: producer-by-feature matrix O (trust for producers across features), diagonal feature matrix P (feature contribution to trust assessment), and transposed category-by-feature matrix Q^T (feature relevance across categories).

relevance to "economic news" (Q's second column). A makes its prediction by weighting its trust (O's first row) by all features' importance and relevance.

2.3.3 Bootstrapping Steps of TRULLO

Having introduced the main idea behind TRULLO, we are now ready to detail TRULLO's algorithmic steps. To do so, we consider that A has to bootstrap a trust rating t for B in category c_x ("economic news"). To bootstrap t, TRULLO running on A carries out the following steps:

Step 1. Determine the categories in which A has previously received content from B. For example, those categories are c_w, c_y, and c_z.

Step 2. Determine the producers from whom A has received content in the previously identified categories plus that to be bootstrapped (i.e., in c_w, c_x, c_y, and c_z). For example, those producers are C, D, and E. In general, TRULLO considers n_p producers in n_c categories (with $n_p \neq n_c$). In this example, $n_p = n_c = 4$. Then, if TRULLO has limited information (i.e., either n_p or n_c equals 1), it bootstraps t depending on the following cases:

1. $n_p = 1$ and $n_c = 1$ (A has received no content - TRULLO has just been installed on A). TRULLO has no information, but has at least one default recommender: its user.

So TRULLO bootstraps t according to its user's risk attitude, which existing mechanisms (e.g., [QH05, LI04]) may elicit. We describe one such mechanism in Chapter 6 by introducing a risk-aware decision model (Section 6.3). We will also see that portable devices that run our architecture use MobiRate (Chapter 4) and LDTP (Chapter 3) to boostrap trust through third-parties's recommendations.

2. $n_p = 1$ and $n_c > 1$ (A has received content only from B and has done so in n_c categories). TRULLO has to formulate a hypothesis of "how trustworthy B is" in a new category. It does so by setting t as the median of its known ratings for B. We choose the median instead of, for example, the mean, because the median gives less weight to outliers [SA04].

3. $n_p > 1$ and $n_c = 1$ (A has received content from n_p producers in the bootstrapping category, but not from B). TRULLO bootstraps t in a way similar to how people initialize their trust. In the most-cited model of generalized trust by Hardin [Har93], initial trust is based on trusting disposition, which, in turn, is based on accumulated experiences *within* a particular category. In this vein, TRULLO bootstraps t as the median of the ratings for the producers known in the bootstrapping category, which may be interpreted as the "typical behavior" for content producers in that category.

In the above cases (very limited information), a simple median may appear to be a reasonable choice. However, in the presence of more information ($n_p > 1$ and $n_c > 1$), Section 2.4.3 on "Simulation Results" demonstrates that the results obtained by extracting features are significantly more accurate than simply using the median (even for small n_p and n_c). Next, with Figure 2.4 at hand, we will detail how TRULLO extract those features.

Step 3. Populate a matrix $N_{n_p \times n_c}$. In our example, this matrix contains A's ratings for the four producers in all categories. N is, however, incomplete - A's trust for B in c_x is the unknown value and is missing. As SVD does not compute on incomplete matrices, we insert the row's average to fill the gap. Given that the row's average is an arbitrary value, we will use it only to apply SVD and not to reconstruct the unknown rating (see step 6), as Troyanskaya *et al.* [HTBA01] suggested. In Figure 2.4, TRULLO populates N and assigns the row's average 0.71 to the rating to be initialized (underlined).

Step 4. Apply SVD on N thus obtaining $N = O \cdot P \cdot Q^T$ (as Figure 2.4 shows).

Step 5. From the three resulting matrices, extract the elements P_{kk} and Q^T_{kj}, $\forall k \in [1, \mu]$ and $j \in [1, n_c]$, where: P_{kk} is the k^{th} feature's influence on trust assessment; Q^T_{kj} is the k^{th} feature's relevance in the j^{th} category; and μ is the number of relevant features[2]. In Figure 2.4, with $\mu = 3$, TRULLO extracts the elements in bold from P and Q^T, and also those highlighted from the latter.

Step 6. For each l^{th} category in which A has received content from B (in our case, $l \in \{w, y, z\}$), regress A's trust in B (in the i^{th} producer) in the l^{th} category against the relevances in that category

[2] The number of features that the decomposition extracts from a matrix ($n_p \times n_c$) is $min\{n_p, n_c\}$. The number of categories A knows is $(n_c - 1)$. Therefore, the number μ of relevant features is $min\{n_p, n_c, (n_c - 1)\}$, i.e., $min\{n_p, (n_c - 1)\}$.

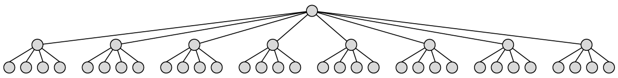


Figure 2.5: Ontology of Reference. The simulation models the number of categories and their relationships with the above ontology. This reflects a typical ontology of the digital content on Amazon.

of all μ features: $N_{il} = b_1 P_{11} Q^T_{1l} + \ldots + b_\mu P_{\mu\mu} Q^T_{\mu l}$. As we neither regress N_{ix} nor consider any feature in the bootstrapping category (Q^T's column corresponding to c_x), the row's average in step 3 has little influence in the regression. That regression then results in μ correlation coefficients $\{b_1, \ldots, b_\mu\}$. For example, in Figure 2.4, TRULLO regresses the elements in bold of N against those of P and Q^T, and obtains the following correlation coefficients: $\{0.5886, -0.4578, 0.5734\}$.

Step 7. Having the correlation coefficients, the elements in bold of P, and those underlined of Q^T, finally compute $t = N_{ix} = b_1 P_{11} Q^T_{1x} + \ldots + b_\mu P_{\mu\mu} Q^T_{\mu x}$. In words, A's trust for B in category c_x equals the *weighted* combination of A's rating for B across features $(b_1, \ldots, b_k, \ldots, b_\mu)$. The weighting factors for each k^{th} feature are the feature's influence on trust assessment (P_{kk}) and the feature's relevance in the x^{th} category (Q^T_{kx}). In the example of Figure 2.4, TRULLO would set t to 0.8037.

That concludes the description of the bootstrapping steps. We now turn to evaluating TRULLO.

2.4 Evaluation of B-trust and TRULLO

The goal of TRULLO is to *initialize* ratings on *handheld* devices and that of B-trust is to update those ratings. To ascertain whether these two algorithms meet their own goals, our evaluation ought to answer two questions:

- *How effective are our algorithms in setting and updating ratings?* To answer this question, we setup a set of simulations (Section 2.4.1), run them (Section 2.4.2), and comment their results (Section 2.4.3).

- *What computational overhead do our algorithms impose on a mobile phone?* We implement TRULLO and B-trust in J2ME and run it on a Nokia phone (Section 2.4.4)

2.4.1 Simulation Setup

To measure the effectiveness of our algorithms, one should measure the extent to which the producers our algorithms select actually send quality content. To do so, we refer back to antique markets (first example in Chapter 1) and model them by making specific choices. To ease reading, we list those choices first, and only then we will justify them in detail.

1. A set of content categories (see point 1 below).

2. A set of content publishers and of consumers. Publishers have categories of expertise in which

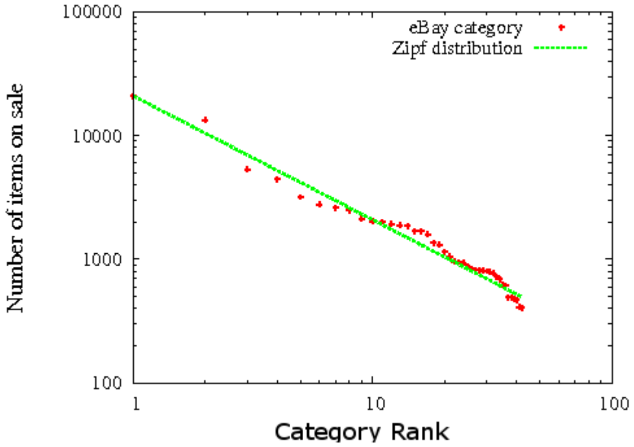


Figure 2.6: Popularity of Antique Categories in eBay. The categories are ranked by the number of antiques on sale. The resulting popularity follows a Zipf distribution.

they send quality e-ads of their products. Expertise is allocated in a way that category popularity[3] is Zipf. Consumers have categories of interest for which they receive e-ads. Again, interests are distributed in a way that category popularity[4] is Zipf.

3. Consumers who select content by bootstrapping trust for publishers. They do so using TRULLO plus two other popular techniques discussed in the literature (see point 3 below). For each of these three ways of bootstrapping trust, we keep track of its utility - the extent to which it select producers who send quality content.

4. Consumers who accept content and rate it depending on its quality. Quality content is made available by publishers in their categories of expertise.

5. Consumers who update trust based on their ratings using B-trust.

A detailed account of how we have realized these five points follows. To ease reading, one may however skip to Section 2.4.2, in which we directly put the five points to use for running simulations.

1) Ontology Definition. TRULLO does not use any ontology. However, we do need to craft one for the purposes of simulation, that is, for modeling the characteristics of antique markets - number of categories, and how producers and consumers distribute their interests across them. We craft an ontology of 40 categories: 1 root, 8 children, each of which has 4 grandchildren (Figure 2.5). This ontology mirrors Amazon's and eBay's ontologies for antiques: they are flat in the sense that they have few levels and most of the nodes lie at lower levels.

2) Category Allocation. The ontology defines the categories we consider. Now, we should determine in which of those categories producers are expert (send quality content) and in which consumers

[3] The popularity of a category is the number of *publishers* who are *expert* in that category.

[4] In this case, the popularity of a category is the number of *consumers* who are *interested* in that category.

are interested. To do so, we again turn to eBay. By ordering the 42 antique categories in eBay by the number of items on sale, we find out that category popularity follows a Zipf distribution (Figure 2.6), as one would expect from literature. Zipf-like distributions are rooted in the dynamics of sending content: the more content a source sends up to a certain moment, the more likely it is that the source will send other content in the future. This, which is a form of preferential attachment, is known to lead to power-law distributions, as shown in economics and complex networks [Bar03]. Therefore, we assign categories to producers and to consumers such that category popularity follows a Zipf distribution. While there is good reason to suppose that the distributions of producers to categories, and interested devices to categories, are all Zipf-like (power-law with parameter *close* to unity), there is no real reason to suppose that they follow the 'strict' Zipf distribution (power-law with unitary parameter). However, slightly changing such a parameter (1 ± 0.2) demonstrated little effect on our simulation results; we thus report results for a unitary parameter. We then consider that each producer is expert in one of the categories we assigned her to and is knowledgable in the remaining categories. Then, to account for real-life unpredictability, we allow producers to randomly change their expertise with probability p (see Section 2.4.3).

3) Bootstrapping Methods. While running our simulations, we consider that a consumer selects content (e-ads) by initializing its trust for producers using three methods. One is TRULLO, and the other two are:

- *Initial disposition bootstrapping:* A device sets the initial trust to be a fixed value in a range $[0, 1]$ that reflects its user's disposition to trust. For example, that value might be 0.2 if the user is risk-averse, or 0.8 if she is risk-prone. In our simulation, such a way of initializing trust is the worst-case scenario because setting the same initial trust (whatever that is) for all producers is equivalent to a device randomly choosing a producer.
- *Recommendation-based bootstrapping:* A device initializes its trust by collecting recommendations from other devices. We consider that recommenders are wholly truthful and do not suffer from any ontology misalignment. Under such assumptions, and given that producers do not change their expertise over time, recommendation-based bootstrapping is the ideal case. To see why, consider that a recommendation about B is a record of how well B has performed. Since B does not change its expertise, the recommendation is also a record of how well B *will* perform. However, that assumes the absence of ontology misalignment and of fake recommendations, which is unrealistic but will serve as a yardstick (best case scenario) to evaluate TRULLO.

4) Rating Content. Whenever they receive content, consumers are unaware of its quality but need to rate it. A consumer rates an ad as follows: $ad's\ relevance = w - dist(c_a, c_p)$, where c_a is the ad's category, c_p is the producer's category of expertise, and w is the maximum distance

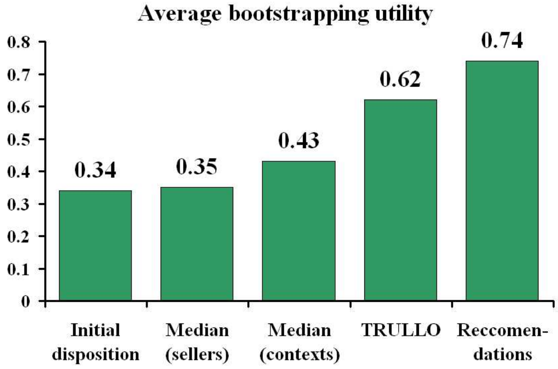


Figure 2.7: Average Utility for Five Ways of Bootstrapping Trust. Average Bootstrapping Utility for TRULLO, two standard bootstrapping methods, and the median of ratings about producers and of ratings across categories. We average all utilities across all content consumers, and each utility lies within $[0, 1]$. Recommendation-based bootstrapping relies on the ideal case of truthful recommenders (no fake recommendations) among which there is no ontology misalignment.

between any two categories (in our ontology, $w = 4$). In words, the closer the ad's category to its producer's expertise, the higher the ad's relevance. The relevance goes from 0 (c_a and c_p are farthest) to a maximum of w (c_a and c_p are the same). We compute the distance $dist()$ between two categories as the minimum number of edges between them[5].

5) Updating Trust. Upon rating a producer, consumers need to update their trust ratings. They do so by using B-trust [QHC06a]. For our simulations, we set the forgetting factor $init$ to a unitary value, and we study the impact of changing the number of trust levels l from 2 (binary) to 4 (scale [1,4]).

2.4.2 Simulation Execution

We are now ready to combine these five points to perform full simulations. We do so by dividing up the execution of the simulation into two phases: (1) content consumers build initial knowledge, (2) upon which they then bootstrap trust.

- *Build initial knowledge.* Producers send content and interested consumers receive it. To simulate this, in each round, each producer sends content in the categories we have assigned her to. Consumers who are interested in those categories receive that content and rate it as described above.
- *Trust bootstrapping.* After this initial phase, whenever consumers wish to receive content in a given category, they choose the best producer(s) in that category. To model this, each

[5] Of course, other measures of distance might be defined. Indeed, we have also used a more complex distance function, whereby siblings of lower levels are considered closer than siblings at higher levels. This did not lead to statistically significant changes in the results.

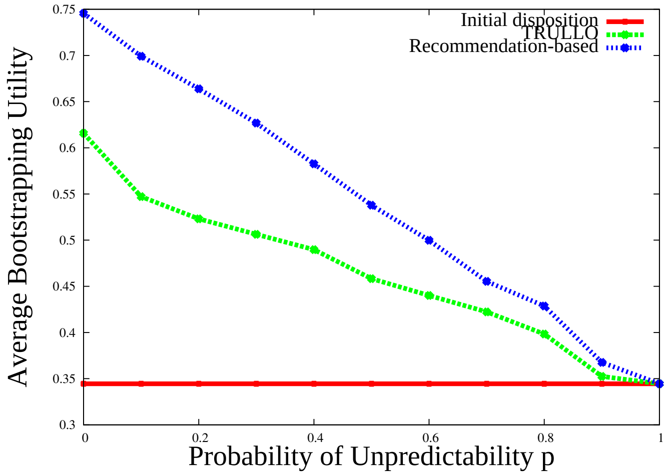


Figure 2.8: Average Utility of Three Bootstrapping Methods. Average utility for TRULLO and two standard bootstrapping methods as a function of the probability p of producers randomly changing their expertise.

consumer selects producers in the categories she is interested. More specifically, for each category of interest c_j, a consumer:

1. Bootstraps its trust for producers in c_j;

2. Selects the producer with the highest initial rating;

3. Updates its utility: $utility\_sum = utility\_sum + \frac{1}{w}(w - dist(c_j, c_p))$. In words, the closer c_j to the producer's expertise (c_p), the higher the consumer's utility. To obtain an average utility within $[0, 1]$, we then normalize the sum.

2.4.3 Simulation Results

We have run a number of those simulation executions and we now discuss their results.

TRULLO compared to two standard bootstrapping methods. We compare TRULLO to both initial disposition bootstrapping and recommendation-based bootstrapping. We simulate a typical antique market that consists of 100 content producers (sellers who make available e-ads) and of 1000 consumers (visitors). As we have seen, the simulation consists of two parts: build initial knowledge and bootstrap trust. We run the first part of the simulation only twice. As we will see, after just two updates, ratings converge (i.e., their confidence is maximum). That is because producers do not change their expertise. We then run the second part and average the utilities for each bootstrapping method across all consumers. With a confidence level of 95%, TRULLO's utility is within a confidence interval of $[0.51, 0.73]$. Figure 2.7 shows that TRULLO's average utility (0.62) is much closer to (the ideal) recommendation-based bootstrapping's (0.74) than to (the baseline) initial disposition bootstrapping's (0.34). Even if it relies on truthful recommendations with no ontology misalignments, recommendation-based bootstrapping does not reach the maximum average utility of 1 because some categories have no specialized producers (by selecting specialized producers, one obtains utility equal to 1).

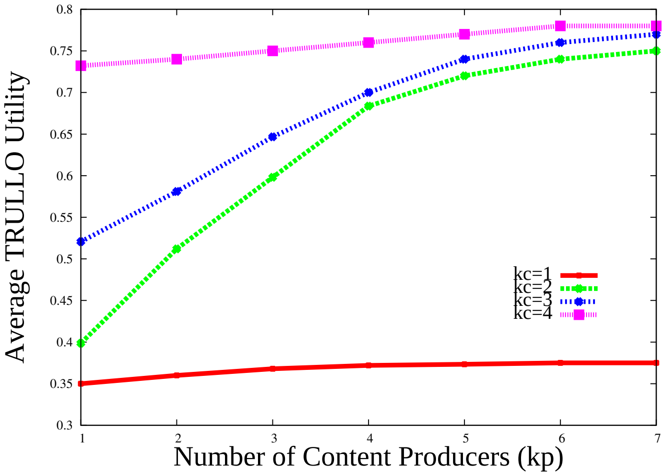


Figure 2.9: Average Bootstrapping Utility for TRULLO Versus its Input Size. Each utility refers to TRULLO bootstrapping trust in a category in which k_p content producers are known in k_c categories. We average across all consumers.

TRULLO compared to a simple median. One may now ask whether using a simple median instead of TRULLO would yield similar results. To test whether this is the case, consider a consumer who initializes its trust for producer B in c_x. We distinguish two cases:

- If the consumer knows k_s producers in c_x, it initializes its trust as the median of the ratings for those producers (median across producers).
- If the device knows B across k_c categories, it initializes its trust as the median of its ratings for B across those categories (median across categories).

Figure 2.7 shows that TRULLO's average utility is much higher than the median across producers and the median across categories. In particular, both medians perform slightly better than initial disposition. Still, for any number of categories/producers, both medians' utilities are less than 0.45 - well below TRULLO's utility.

Factors affecting effectiveness. Having these preliminary results, we now study how the following simulated factors might have affected TRULLO's utility:

- *B-trust.* One would expect that how well TRULLO performs partly depends on the granularity of B-trust's input ratings. The previous results refer to a number of trust levels l equal to 4. That is, B-trust has processed ratings on a scale [1,4]. However, during those experiments, we also measured the average utility for users who run a version of B-trust that processed:
  - binary ratings only (i.e., $l = 2$) - the utility decreased from 0.62 to 0.54.
  - ratings on a scale $[1, 6]$ (i.e., $l = 6$) - the utility remained unchanged (0.62).

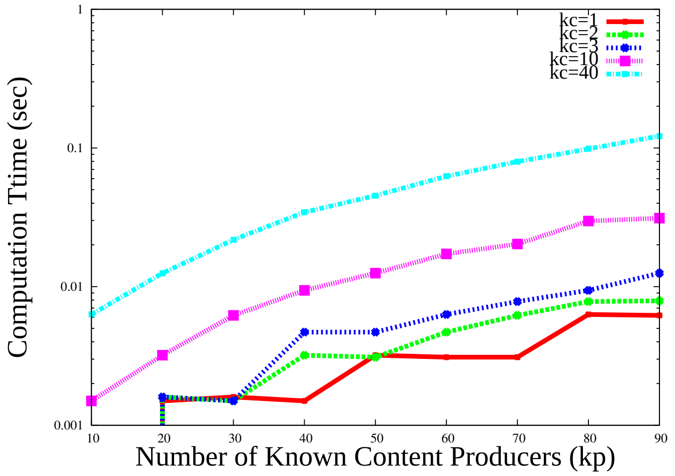


Figure 2.10: TRULLO's Computational Overhead. TRULLO's performance (seconds) versus TRULLO input size, as means of 10 runs.

This tells us that, by accepting more fine-grained ratings (e.g., $l : 2 \rightarrow 4$), B-trust is able to significantly increase its user's utility.

- *Confidence in the ratings.* If one decreases the number of rounds of the first part of the simulation, consumers would receive less content and, as a result, their ratings would be updated less frequently - they would be more uncertain. We decrease the number of rounds and find that TRULLO's average utility does not change. That is because producers do not change their expertise and, as such, ratings about them converge quickly. To capture real-life unpredictability, we now allow producers to send content whose relevance reflects either their actual expertise with probability $(1 - p)$ or a randomly chosen expertise with probability p. Figure 2.8 shows that as p increases, the average utilities for TRULLO and recommendation-based bootstrapping decrease, as expected. However, even for high values of p, both bootstrapping methods perform better than initial disposition. Also, their utilities decrease linearly. The reason is that for 'misplaced' content (i.e., content that a producer sends from a category other than that of her expertise), the utility decreases by a certain amount. The amount of 'misplaced' content increases as p increases and hence the average utility decreases *proportionally*.

- *Knowledge.* One would expect that the more ratings (across producers and categories) a user knows, the higher her utility in using TRULLO. To verify that, during the simulation execution, whenever TRULLO bootstrapped trust, we kept track of its utility and of the corresponding number of producers and categories known, and we then averaged all utilities. Figure 2.9 shows that TRULLO performs better than initial disposition bootstrapping even if it only knows one producer in a single content category. As one might expect, as the number of categories and the number of producers increase, so does TRULLO's average utility.

2.4.4 Overhead

Given that the integration of TRULLO and B-trust effectively bootstraps trust, it is now worth verifying whether it is usable on a mobile phone. We implemented TRULLO and B-trust in J2ME and ran the code on a Nokia 3230 mobile phone (whose features include: Symbian operating system 7.0, 32 MB of memory, 32-bit RISC CPU). Figure 2.10 shows the code's performance, given as the mean of 10 runs, for varying input $(k_s \times k_c)$ matrix sizes. We minimized background activities by shutting down all applications other than TRULLO and B-trust. The computational overhead is very low. That is because the input matrix only contains the ratings of a single user. For example, to bootstrap rating in a category in which 20 producers are known in 10 categories (maximum input in the previous experiments), the code takes just 3.2 milliseconds.

2.5 Discussion

Based on those evaluation results, we now discuss various open questions.

Correlation of how producers perform across categories. One inherent property of TRULLO is that one can extract statistical features from ratings. Unfortunately, we do not have any dataset coming from real mobile applications that corroborates this property. However, we do have Internet web sites reporting user ratings across categories. Take hostels.com: it reports customer ratings of hostels across categories, namely character, security, location, staff, fun, and cleanliness. By sampling parts of these trust ratings and applying the singular value decomposition, we learned that they do correlate across categories - for example, trust ratings about “character” and “staff” share roughly the same statistical features. That, however, does not guarantee that this would be the case in mobile applications. For this reason, our simulation setup has made no explicit assumption on whether trust correlates across categories, and it has shown how some simulated factors have affected the results. More specifically, we have showed that if content producers randomly change their expertise, TRULLO still outperforms existing ways of bootstrapping trust.

Choosing the right bootstrapping method. The choice of the right bootstrapping method is application-dependent. More concretely, consider the following aspects that are usually critical in mobile computing:

- *Device computational cost.* If the computational cost must approach zero and bootstrapping accuracy does not matter, then initial disposition bootstrapping may be a fair choice. Otherwise, one might use TRULLO, which runs on a mobile phone at modest computational cost.
- *Device communication overhead.* In a fully distributed setting, asking for recommendations might increase data traffic among devices. To avoid that, one may use TRULLO, since it is more effective than initial disposition bootstrapping and relies solely on local information

(no device communication required) or may use MobiRate, which robustly collects recommendations with little communication overhead (Chapter 4).

- *Threats.* Modeling hostile environments is an important research question, on which we will focus in Chapter 5 but on which now we ponder briefly. Consider environments that may be deemed hostile because of ontology misalignments among users, or of presence of fake recommendations, or of users behaving very differently across categories. In the presence of the first two problems (which are indeed very likely), recommendation-based bootstrapping may suffer as it relies on third party information that, in this case, would be made unreliable (because of ontology misalignments and fake recommendations). In such a situation, one may run MobiRate to counter fake recommendations or run TRULLO as it relies only on local information. However, if producers perform very differently across logically close/related categories, then TRULLO may turn to be a poor choice because no statistical property about content producers would exist. Thus, in the presence of all three problems, one has to resort to initial disposition bootstrapping.

2.6 Conclusion

TRULLO and B-trust effectively bootstrap trust upon personal experiences as they perform close to how exchanging recommendations would do in an ideal (though unrealistic) world, one in which recommenders are wholly truthful and, furthermore, share the same ontology. Plus, our J2ME implementation does not impact the usability of a Nokia 3230 mobile phone.

To produce trust assessments, portable devices should not limit themselves to reason on ratings of their experiences, but they may well rely on third-parties' recommendations. The next chapter is about how devices reason on recommendations.

Chapter 3

Reasoning on Recommendations

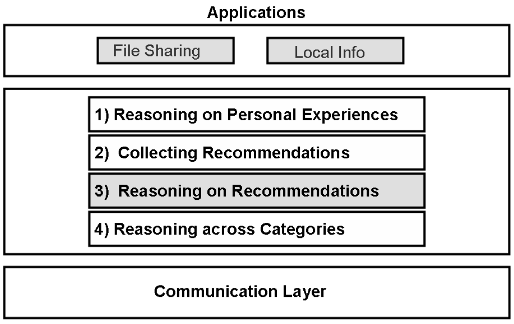

Applications
File Sharing
Local Info
1) Reasoning on Personal Experiences
2) Collecting Recommendations
3) Reasoning on Recommendations
4) Reasoning across Categories
Communication Layer

Figure 3.1: Research Goal 3: Reasoning on Recommendations.

To set its trust for B from recommendations (third-party ratings), A may compute the median of recommendations [SA04], or it may weight recommendations according to whether the corresponding recommenders have been found useful in the past [Cap04, LI04]. Those ways of aggregating recommendations suffer, however, from different attacks. For example, they suffer from sybil attackers who create multiple identities and pretend to be multiple, distinct people. To see why attackers profit from creating multiple identities, consider that an attacker authenticates herself with a pseudonym, misbehaves, creates a new pseudonym, authenticates herself with the new pseudonym (pretending to be a new user), and misbehaves again. As a result, the attacker misbehaves without being traceable. Resnick and Friedman [FR01] formally laid down such a problem, presenting a game theoretical model for analyzing the social cost of allowing users of reputation systems to freely change identities. They concluded that, if users generate pseudonyms by themselves, all unknown users should be regarded as malicious. The end result is that sybil attacks undermine trust relationships and, thus, cooperation.

Recently, researchers proposed one way of aggregating ratings that is robust to Sybil attacks. This way consists of maintaining a web of trust [GKRT04, ZL04] of content producers. A web of trust is a network of trust relationships: we trust (link to) only a handful of other people; these people, in turn, trust (link to) a limited number of other individuals; overall, these trust relationships form a network (a web of trust) of individuals linked by trust relationships. Upon this web of trust, individuals may form opinions of other individuals (in technical parlance, they *propagate trust* in other individuals) from whom they have never received content before. Individuals then decide whether to accept content according to these opinions.

One may represent the statement "C trusts D" as a web-of-trust of two people and one trust relationship going from C to D (Figure 3.2(a)). The relationship may also be labeled with a rating representing the extent to which C trusts D in a given range of trust values (for example, in Figure 3.2(a), C rates its trust for D as 2 in a discrete range $\{1, 2, 3\}$ - for simplicity's sake, our examples use a trust scale $\{1, 2, 3\}$, but our model by no means requires this as it works on any scale). Now, consider the web of trust in Figure 3.2(b), where we have four people A, B, C and D. Not everyone has interacted with everyone else. For example, A and B have never interacted (no link between them). For the sake of argument, suppose that A now wishes to interact with B and, as a consequence, it has to form an opinion about B. A may do so by predicting its trust for B.

Problem Statement: How a mobile user A may form an opinion about another user B without prior interaction.

3.1 Existing Solutions to Reasoning on Recommendations

To form its initial trust for B, most literature suggests that A creates a web of trust from third-party ratings and, based on that, *sets* its trust (*propagates its trust*) for B. There is substantial literature on *how to* propagate trust. That literature breaks roughly into two camps. In the first, techniques assign a *global* trust value to each user. That is, A's trust in B corresponds to a global trust value in B. By global, we mean a trust value that is accepted and shared by all users. In peer-to-peer networks, EigenTrust [KSGM03] assigns a global trust rating to each peer similar to how Google's PageRank [PBMW98] ranks web pages. Global ratings 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 a free software developer community, Advogato [LA98] assigns a global trust to each community member. It does so by arranging ratings in a web of trust and by composing ratings between members using max flow from designated trusted members. The idea of max flow is that between any two nodes, 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. This way of

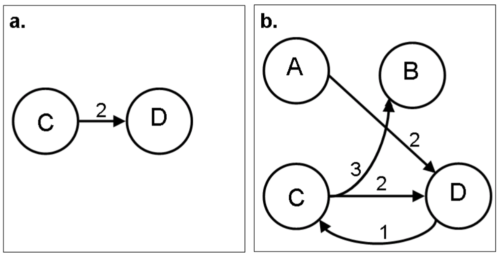


Figure 3.2: Webs of Trust. **(a)** A simple web of trust representing the statement "C rates 2 its trust in D". **(b)** A web of trust of four people connected by their trust relationships.

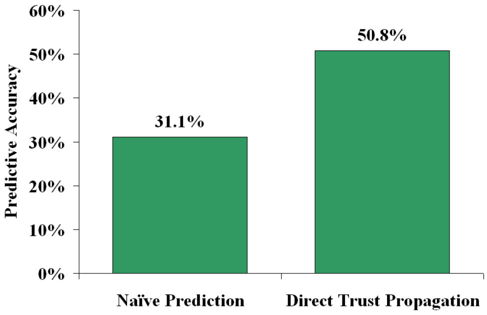


Figure 3.3: Predictive Accuracy. The fraction of correct predictions for naive prediction (random guess) and direct trust propagation.

composing ratings has proved to be attack resistant - it successfully isolates unreliable members. More recently, Ziegler and Lausen [ZL04] proposed to rank all users by spreading "activation models" (by arranging ratings in a matrix and finding the principal eigenvector), while Dell'Amico [Del06] focused on peer-to-peer networks and proposed to rank peers by using link-analysis techniques in a fully distributed setting.

By contrast, in the second literature camp, techniques assign a pairwise (*local*) trust rating to each pair of users. That is, A assigns a personalized trust rating in B. Personalizing trust is beneficial because it makes it possible for two individuals to have different opinions about the trustworthiness of the same person (which may well happen in reality). In 2003, Golbeck *et al.* [GPH03] proposed different algorithms for propagating trust in this way. For example, they proposed a variation of max flow that accounts for path length. To Domingos *et al.* [RAD03] and Guha *et al.* [GKRT04] goes the merit of presenting the first comparative studies of different trust propagation algorithms in which pairwise ratings are computed. These algorithms have been evaluated against Epinions [epi], a large collection of binary ratings[1]. By binary we mean that each rating simply expresses whether an individual trusts another individual or not. Despite being evaluated on binary ratings, these algorithms are general in the sense that they can take discrete ratings (not necessarily binary).

Most of the work on assigning pairwise trust ratings is based on a simple, yet effective mechanism: A finds all paths leading to B; for each path, A then concatenates the ratings along the path; A finally aggregates all path concatenations into a single trust rating for B. Algorithmically, this is equivalent to A arranging trust ratings into a matrix and, over a series of iterations, propagating trust by, for example, *direct* propagation: if A trusts C and C trusts B, then trust propagates from A to B. The resulting matrix values are then rounded into a single trust rating. Unfortunately, this way of propagating trust suffers from two main limitations:

- Literature has showed direct trust propagation to be extremely effective, but it has done so only on data sets of *binary* ratings. There is no published work on how direct propagation would perform on a *large* data set of *discrete* ratings, not necessarily binary[2]. An individual may express whether

[1] In Epinions, quality assessments of products are on a scale $\{1, \ldots, 5\}$, but user ratings are binary.

[2] One may point out the existence of PGP's web of trust [Zim95]. Unfortunately, that web of trust is not suitable for what we

she trusts another individual or not, and, if she does, she may then express the extent to which she trusts by a discrete value. For example, in Advogato [LA98], users express their trust with 3-level ratings (ratings in $\{1,2,3\}$). Given that, one may wonder how direct trust propagation would perform against the Advogato data set. We have evaluated direct trust propagation on Advogato as Section 3.3 will describe. Unfortunately, as Figure 3.3 shows, the predictive accuracy (fraction of correct predictions) is low. More precisely, considering a large sample of Advogato's trust ratings (55,455 ratings), direct trust propagation correctly predicts 50.76% of the ratings (as a reference, consider that a naive prediction (random guess) correctly predicts 31.1% of the ratings[3]). Therefore, direct trust propagation does not predict the actual rating in roughly half of the cases. That is because direct trust propagation assumes that trust is transitive - it maintains that if A trusts B and B trusts C, then A should trust C. However, that is not necessarily true.

- Direct trust propagation does not scale on mobile devices. Direct trust propagation is meant for Web applications in which centralized servers store full webs of trust upon which trust is then propagated by multiplying vectors and matrices whose dimensions are extremely high. As a consequence, it is computationally expensive and would not run on any existing portable device. Moreover, mobile devices would only know a very small subset of the web of trust at any given time (it is unrealistic to assume complete knowledge) because of, for example, network partition, device (un)availability, and limited resources.

What is missing: A way of propagating trust with which (*resource-constrained*) mobile phones make accurate predictions in a *distributed* way.

3.2 Our Proposal: LDTP

We here present a novel way of propagating trust that works in distributed settings (e.g., P2P networks) and runs on (resource-constrained) mobile phones. We call it LDTP, which stands for "Lightweight Distributed Trust Propagation" [QHC07a].

3.2.1 Assumptions

- We consider the web of trust to be a network in which the small world phenomenon holds (i.e., the distance between two people is short). We argue this is a perfectly plausible assumption given that the web of trust is a social network, for which the small world phenomenon emerges.

- We consider that anonymous pseudonyms identify users. A pseudonym may take the form of a public key and is anonymous in the sense that it is not linked to its user's real identity. This may protect user anonymity but makes the system vulnerable to Sybil attacks (whose impact is discussed in Section 3.4).

call trust propagation (i.e., computation of *single* pairwise ratings). On that web of trust, it makes only sense to *combine* pairwise ratings (as PGP does) - each of these ratings cannot be propagated but must be set by a user. Any research on PGP (including that in mobile settings) solves a problem different from ours, i.e., how to make it possible to combine (not to compute) pairwise ratings.

[3] Its predictive accuracy is 31.1% and not 33% because ratings are not uniformly distributed across the three possible discrete values.

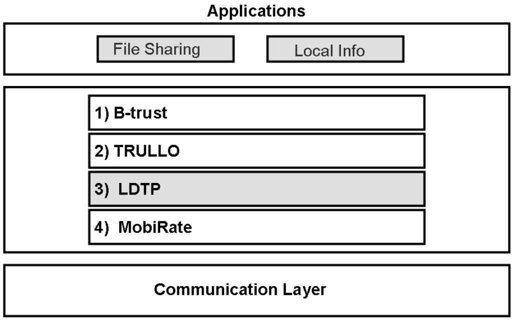


Figure 3.4: Contribution 3: Reasoning on Recommendations (LDTP).

3.2.2 Brief Description

We use a class of semi-supervised learning techniques that have been proved to be effective when applied to various problem domains (e.g., to predict movie reviews [GZ06], to recognize digits [HPW05], to classify text [ZGL03]).

These techniques work on a graph (for example, that of Figure 3.5(b)) in which: *nodes* (not links) are either rated (e.g., x_j) or unrated (e.g., x_h), and those nodes are then connected to each other if they are *related* (i.e., if their ratings are likely to be similar). Informally, these techniques exploit knowledge already present in the graph (rated nodes) to construct a function that is capable of predicting unrated nodes. To choose the most *effective* function (the function with the highest predictive accuracy), the techniques impose that: on input of each rated node in the graph, the chosen function returns the node's actual rating (this serves to choose a function that is *consistent* with existing ratings); given two connected nodes x_i and x_j, the chosen function returns $f(x_i)$ similar to $f(x_j)$ (this serves to choose a function that assigns similar ratings to connected nodes).

Let us now imagine a graph whose nodes are trust relationships (we call this graph the *'relationship graph'*). A node is rated if the corresponding relationship in the web of trust is known and rated (e.g., in Figure 3.6(a), $C \rightarrow B$ is known and rated, so that, in the relationship graph, the node that represents this relationship will be rated). By contrast, a node is unrated if the corresponding relationship is unknown and has to be predicted (e.g., $A \rightarrow B$ in Figure 3.6(a)). Predicting a trust relationship thus means finding a function that effectively rates the corresponding node. Finding such a function is exactly the problem solved by the semi-supervised learning techniques described above.

To describe our model, we refer to our running example of how A may propagate its trust in B given the web of trust shown in Figure 3.6(a). The following four steps are required, each of which is described in the following sections:

1. A determines the trust relationships that our propagation scheme may find relevant for predicting A's trust in B (Section 3.2.3).

2. A restricts its attention to the subset of the web of trust that it knows and that includes those relationships (Section 3.2.4).

3. From this subset, A builds a relationship graph (Section 3.2.5).

4. A finally applies the machine learning technique to determine a function that predicts $A \to B$ (Section 3.2.6).

3.2.3 Determining Relevant Trust Relationships

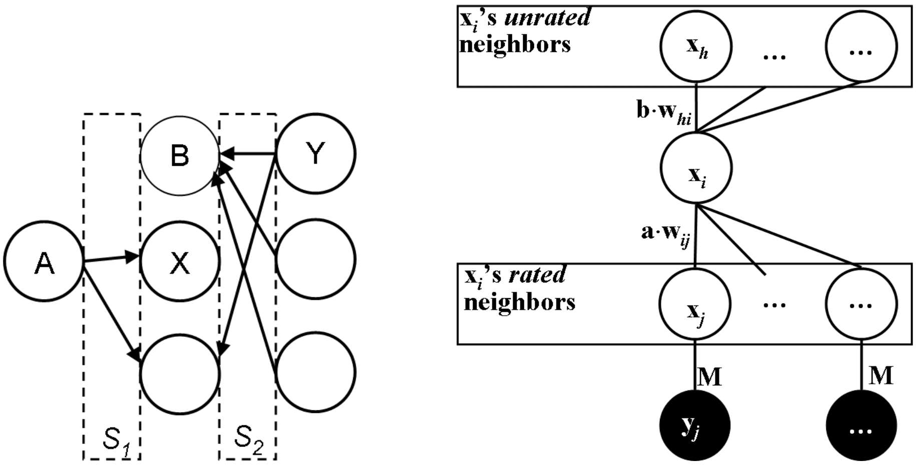
To begin with, A determines the trust relationships that our propagation scheme may find relevant for predicting A's trust in B, that is, those relationships *related* to $A \to B$. As defined previously, *two relationships are related if their ratings are similar.* Hence we consider related any:

- *Two relationships with the same rater.* For example, $A \to B$ and $A \to D$ are related as they have the same rater A, and the more alike B and D *perform*, the more related $A \to B$ and $A \to D$ are. We call this kind of relation *performing relation*, and denote it as $rel_p(B, D)$. More formally, the weight between the i^{th} trust relationship "A trusts B" and the j^{th} trust relationship "A trusts D" is

$$w_{ij} = o \cdot rel_p(B, D), \tag{3.1}$$

  where o is the weight given to performing relations. How do we quantify the performing relation between B and D? We may do so by considering that the more alike B and D perform, the closer the ratings about them. Along these lines, we: *(1)* take all people who have rated both B and D. In our example (Figure 3.6(a)), this is only C. *(2)* For each such person, compute the absolute difference between her rating in B and that in D. In our example, the absolute difference between $C \to D$ and $C \to B$ is $3 - 2 = 1$. *(3)* Normalize those differences in $[0, 1]$, and aggregate them. We will shortly describe two ways of aggregating differences. For the time being, consider that, in our example, there is only one person who has rated both B and D, thus only one difference (so no need for any aggregation). Given that ratings are given in a discrete scale $[1, 3]$, the normalized difference value is $(3 - 2)/(3 - 1) = 1/2$. Having the aggregated value, we may then say that the bigger this is, the less alike B and D perform (the lower $rel_p(B, D)$). For this reason, we define $rel_p(B, D) = 1 - v$, where v is the aggregated value. In our example, we had only one difference so we did not need any aggregation at the third step. In general, we may have more than one person who has rated both B and D, thus more than one difference (for example, let us assume we have two normalized differences $1/2$ and $2/2$). In that case, we need to aggregate those differences. The simplest way to do so is to compute the average difference (in our example, that is $\frac{1/2+2/2}{2} = 3/4$). However, whether the average is a good choice depends on the number of the differences across which we average and their variance. The average is a good choice when there are many differences whose variance is very low. However that is not necessarily true. Therefore, we consider a second way of aggregating that accounts for both number of differences and their variance: we compute the confidence interval of the average (with 95% of confidence) and take the higher extreme. We will evaluate which aggregation leads to the highest predictive accuracy in Section 3.3.1.

- *Two relationships in which the same person is rated.* $A \to B$ and $C \to B$ are an example, as they both rate B. People may differ in the way they rate. We thus have to compute a *judging relation*



(a) (b)

Figure 3.5: A's View of the Web of Trust. **(a)** A's view of the web of trust. **(b)** A schematic representation of a relationship graph. On that graph, our algorithm predicts x_i's rating.

for which, the more alike two raters (A and C) have judged the same person (B), the higher their relation. We denote this as $rel_g(A, C)$. The corresponding weight between the two relationships is

$$w_{ij} = s \cdot rel_g(A, C), \tag{3.2}$$

where s is the weight given to judging relations. To compute $rel_g(A, C)$, we take all people who have been rated by both A and C; for each person, we compute the difference between A's and C's ratings; finally, we aggregate all differences. Again, we aggregate by computing the average difference and its confidence interval.

3.2.4 Taking Part of the Web of Trust

To use our propagation scheme for predicting its trust for B, A needs to know part of the web of trust. To see which part is needed, we must consider the two steps through which A predicts its trust for B. In so doing, we will refer to Figure 3.5(a) and consider the following sets: S_1, that is, the set of ratings of A's outgoing relationships; and S_2, that is, the set of all ratings from nodes that have rated B.

Step 1. A determines the trust relationships related to $A \rightarrow B$. Those relationships take the form $A \rightarrow X$ and $Y \rightarrow B$ (where X and Y are generic persons different from A and B). To determine those relationships, A needs the ratings of its outgoing relationships (set S_1) plus the ratings of B's incoming relationships (relationships in S_2 going into B).

Step 2. A determines the extent to which each pair of those edges are related. More specifically:

- A determines the extent to which any pair of edges $A \rightarrow X$ and $A \rightarrow B$ are related. This requires to quantify the (performing) relation between B and X — to quantify how alike B and X perform. To do so, A needs the ratings of the relationships in S_2 plus those in S_1.

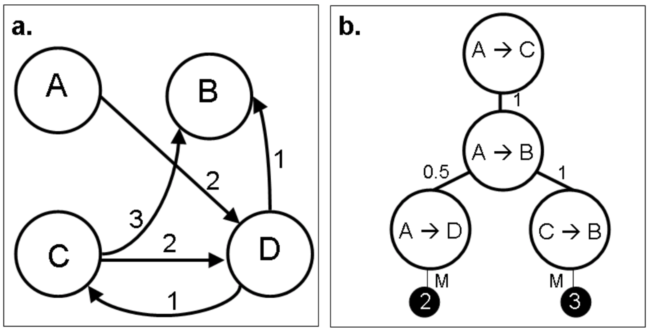


Figure 3.6: Web of Trust and Learning Graph. **(a)** A web of trust and **(b)** the corresponding relationship graph for predicting $A \rightarrow B$'s rating.

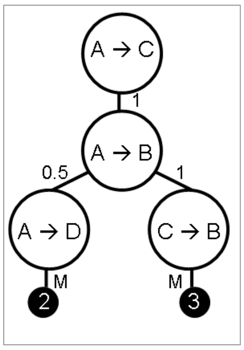
- A determines the extent to which any pair of edges $Y \rightarrow B$ to $A \rightarrow B$ are related. This requires to quantify the (judging) relation between Y and A — to quantify how alike Y and A rate. Again, to do so, A needs the ratings of the relationships in S_2 plus those in S_1.

Overall, to use our propagation scheme for predicting its trust in B, A needs the ratings of the relationships in S_1 and in S_2. In the next chapter, we will propose a scheme (called MobiRate) with which A collects those relationship. In this scheme, the former relationships (in S_1) are readily available, as we can assume that A stores locally the ratings it has produced. And the latter are received from B. In fact, those ratings have been generated by the people who have rated B; therefore, B may have all the ratings of the relationships in S_2, as it has received them from its raters. As a result, each user stores a very small subset of the web of trust — she stores the ratings she generates plus those generated by her raters.

3.2.5 Building the Relationship Graph

At this point, A knows which trust relationships are related to the one that has to be propagated, and the extent to which they are so; A can then build a relationship graph. The key idea is to create a graph whose nodes include the trust relationship to be predicted plus related relationships. Figure 3.6(b) shows one such graph whose nodes are: $A \rightarrow B$ (trust relationship to be predicted), $A \rightarrow C$, $A \rightarrow D$, and $C \rightarrow B$ (related relationships). Nodes are linked and the label on a link expresses the extent to which the two linked nodes are related.

More generally, there are n trust relationships $x_1, \dots, x_n$, of which r are rated $(x_1, y_1), \dots, (x_r, y_r)$ and u are unrated $x_{r+1}, \dots, x_{r+u}$ (an 'unrated' relationship is a relationship that has no rating on the web of trust). The numerical ratings are defined as being $y_1, \dots, y_r \in \{l_1, \dots, l_p\}$, with $l_1 < \dots < l_p$. Our problem is now to build a connected graph $G = (V, E)$ with nodes V corresponding to the n trust relationships. We do so step by step with reference to an example (Figure 3.6(b)) and to a general representation (Figure 3.5(b)). To understand the rationale behind this construction, we must remember our end goal of finding a predictive function $f : V \rightarrow \mathbb{R}$ on G capable of assigning ratings to unrated nodes (note that f assigns a real value to a trust relationship; this value will then be mapped to the nearest discrete rating in $\{l_1, \dots, l_p\}$). In particular, let x_i be the trust relationship we wish to predict



$$y=\begin{pmatrix}2\\3\\0\\0\end{pmatrix} \quad H=\begin{pmatrix}M&0&0&0\\0&M&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}$$

$$\bar{W}=\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0.5&1&1&0\end{pmatrix} \quad f=\begin{pmatrix}2\\3\\2.7\\2.7\end{pmatrix}$$

Figure 3.7: A Relationship Graph and Corresponding Matrices. In the relationship graph, there are two rated nodes $x_1 : A \rightarrow D$ and $x_2 : C \rightarrow B$; and two unrated nodes $x_3 : A \rightarrow C$ and $x_4 : A \rightarrow B$. The prediction algorithm populates the rating matrix **y**, the diagonal dongle matrix H, and the weight matrix $\bar{W}$. Then, it computes the predictive function **f**.

(e.g., $A \rightarrow B$ in Figure 3.6(b)) among the unrated ones.

- The node x_i is connected to its k most related nodes that are *rated*. We call these nodes x_i's rated neighbors. Connecting x_i to those nodes serves to impose that the function f rates x_i and its rated neighbors with similar ratings. Let x_j denote one of x_i's rated neighbors. The weight of the edge between x_i and x_j is $v \cdot w_{ij}$. The coefficient v is the weighting factor for rated neighbors, and w_{ij} is determined as per formula (3.1) or (3.2) (depending on whether the relation between x_i and x_j is a performing or a judging relation). For example, in Figure 3.6(b), $A \rightarrow B$ is connected to $A \rightarrow D$ and to $C \rightarrow B$ and the corresponding weights are 0.5 and 1, respectively.

- The node x_i is also connected to the k' most related nodes that are *unrated* (e.g., in Figure 3.6(b), $A \rightarrow B$ is connected to $A \rightarrow C$). We call these nodes x_i's *unrated* neighbors. Connecting x_i to those nodes serves to impose that the function f rates x_i and its unrated neighbors with similar ratings. Let x_h denote one of those unrated nodes. The weight of the edge between x_h and x_i is $z \cdot w_{hi}$. The coefficient z is the weighting factor for unrated neighbors, and w_{hi} is determined as per formula (3.1) or (3.2).

- Each rated node x_j is connected to an "observed node" (dark circles in both reference figures) whose value is the rating y_j. The observed node is a 'dongle' because it connects only x_j. The edge weight is a large number M. Setting M to a large number serves to pull $f(x_j)$ towards the true rating y_j (in particular, if $M \rightarrow \infty$ then $f(x_j) = y_j$). That corresponds to the first condition for choosing an effective function f (Section 3.2.2): making the function consistent with existing ratings.

The coefficients in a relationship graph are thus M, v, z, k, and k'. M is set to an arbitrary large number (10^6). The remaining coefficients will be set by cross validation (Section 3.3.1).

3.2.6 Finding a Predictive Function on the Graph

Having the relationship graph, A now has to find a function that predicts all unrated nodes in that graph, including that of interest (e.g., $A \rightarrow B$).

Let us formalize the problem and the construction of its solution. We have a graph G of n nodes $x_i, i \in [1, n]$, r of which have ratings y_i, and the remaining u $(x_{r+1}, \ldots, x_{r+u})$ are unrated. The set R contains the indices of the *rated* nodes and U those of unrated nodes. Our problem is to seek a function f that rates each of the *unrated* nodes (the nodes whose indices are in U). Section 3.2.2 mentioned that to choose a function that rates *effectively*, one has to impose that: on input of each rated node in the graph, the chosen function returns the node's actual rating (that has been done in the previous Section 3.2.5 by setting the coefficient M to a large number); on input of either of two connected nodes, the chosen function's outputs are similar.

The latter condition is equivalent to saying that (refer to Figure 3.5(b)): for any pair of related nodes x_i and x_j, the difference of their ratings $f(x_i)$ and $f(x_j)$ should be minimum. In other words, $(f(x_i) - f(x_j))^2$ should be minimum. This expression represents the rating difference *over one edge*. One may compute the difference *over the graph* by summing the rating differences of all edges. We denote this difference as $\mathcal{L}(f)$. Such a difference depends on the chosen function f. Our problem is to find the function f for which the difference over the graph is minimum. We do so in Appendix B and find that such a function is $\mathbf{f} = \Big(H + F\Big)^{-1} H\mathbf{y}$. Let us now introduce the notation and the three steps of the algorithm for computing $\mathbf{f}$ (refer to Figure 3.7):

1. Populate 3 matrices:

   - The rating matrix $\mathbf{y} = (y_1, \ldots, y_r, 0, \ldots, 0)^\top$. This is an $n \times 1$ matrix whose first r rows contain the ratings of the rated nodes, and any remaining row contains zero. For example, Figure 3.7 shows a relationship graph and the corresponding rating matrix.
   - The diagonal dongle weight matrix H. The elements that correspond to rated nodes contain M (otherwise 0):
   $$H_{ii} = \begin{cases} M, & i \in R \\ 0, & i \in U \end{cases}$$
   In Figure 3.7, the first two nodes are rated.
   - The $n \times n$ weight matrix $\bar{W}$:
   $$\bar{W}_{ij} = \begin{cases} v \cdot w_{ij}, & j \in RN(i) \\ z \cdot w_{ij}, & j \in UN(i) \\ 0, & \text{otherwise} \end{cases}$$
   $RN(i)$: set of indices of i's *rated* neighbors;

   $UN(i)$: set of indices of i's *unrated* neighbors.

2. Compute: the symmetrized version of $\bar{W}$: $W = max(\bar{W}, \bar{W}^\top)$; the diagonal degree matrix $I_{ii} = \sum_{j=1}^{n} W_{ij}$ (we consider a node's degree to be the sum of its edge weights); and the combinatorial Laplacian matrix $F = I - W$.

Factor	Description	Tuning Range
$f_r = \frac{k}{\lvert R\rvert}$	Fraction of rated nodes used as neighbors.	{0.05, 0.1, 0.15, 0.2, 0.3 }
$f_u = \frac{k'}{\lvert U\rvert}$	Fraction of unrated nodes used as neighbors.	{0.05, 0.1, 0.15, 0.2, 0.3}
$\beta = \frac{z}{v}$	Relative weight between rated and unrated nodes.	$\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$
$\gamma = \frac{s}{o}$	Relative weight between judging and performing relations.	$\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$
α	How to compute $rel(A, B)$, i.e., the extent to which A and B perform or judge alike.	Two ways: average difference or confidence interval

Table 3.1: Parameters of Our Prediction Algorithm. To find the optimal values, we tune the parameters in the reported ranges.

3. Finally, compute $\mathbf{f} = \Big(H + F\Big)^{-1} H\mathbf{y}$. This has the form $\mathbf{f} = (f(x_1), \ldots, f(x_n))^\top$, and its last u elements are the predicted ratings for the nodes in U. Appendix B shows that the so computed **f** minimizes the rating difference over the relationship graph. For example, in the relationship graph of Figure 3.7, $A \rightarrow B$'s rating is predicted to be $f_4 = 2.7$.

That concludes the description of our model. In the next section, we turn to evaluating it.

3.3 Evaluation of LDTP

The goal of our algorithm is to predict trust ratings on portable devices. To ascertain the effectiveness of our algorithm at meeting this goal, our evaluation ought to answer three questions:
(1) *(Predictive Accuracy)* How accurate is our algorithm in predicting trust ratings?
(2) *(Prediction Robustness)* What is the impact of uncooperative users upon the algorithm's accuracy?
(3) *(Overhead)* What time, storage, and communication overheads does our algorithm impose on a mobile phone?

To see whether our algorithm effectively predicts trust and whether it is usable on portable devices, we need a *large-scale deployment*. In this way we can separate statistically significant answers from the plausible insights gained by a small-scale deployment. Moreover, a deployment needs to be evaluated in the *long-term* to see whether our algorithm is robust against, for example, uncooperative users.

Unfortunately, we do not have a long-term evaluation of a large-scale mobile computing deployment. We do, however, have a large rating data set from the Advogato community that represents more than a decade's history[4]. Using this data set (described next), we evaluate whether our algorithm is effective in predicting real trust ratings (Section 3.3.1). Then, to evaluate how robust our algorithm is, we emulate how users may rationally turn to be uncooperative (Section 3.3.2). Finally, we implement our algorithm to assess whether it is usable on a mobile phone (Section 3.3.3).

[4] Since webs of trust tend to contain private information, only a limited number are publicly available, and those few are quite small (they do not allow for thorough evaluation). The only exception is the Advogato data set.

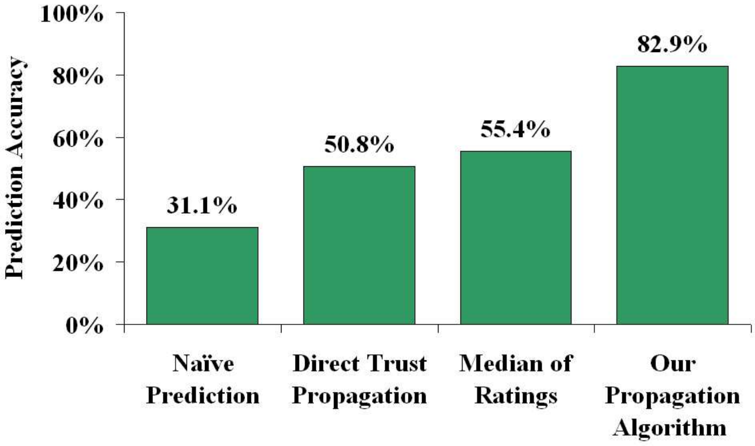


Figure 3.8: Predictive Accuracy of Four Algorithms.

To begin with, let us describe the Advogato data set. Advogato is a community discussion board for free software developers. Using the Advogato's trust metric [LA98], each user has a single (global) trust value computed by composing other users ratings. There are three possible ratings: apprentice, journeyer, and master. Global trust is used to control access to the discussion board: 'apprentices' can only post comments, whereas 'journeyers' and 'masters' are able to post both stories and comments. From this community, we have extracted 55,455 trust relationships.

3.3.1 Predictive Accuracy

We evaluate the predictive accuracy of our algorithm by using leave-one-out cross validation.

Validation Execution. The cross validation unfolds as follows. We take Advogato's web of trust. We mask one trust relationship and then predict the relationship's rating in four different ways. We repeat this on all relationships. In doing this, we measure the *predictive accuracy*, i.e., the fraction of correct predictions.

The four ways of predicting the rating of a masked relationship $A \rightarrow B$ that we have compared are: *naive prediction* (random guess); *median of ratings* about B; *direct trust propagation*; and *our algorithm*. Since we cannot assume complete knowledge in mobile settings, we consider that our algorithm does not know the whole web of trust, but predicts $A \rightarrow B$ on input of only the ratings known by A (as described in Section 3.2.4). Instead, we allow distributed trust propagation to know all paths (and corresponding ratings) between A and B because it is the only way it can be carried out.

Our propagation algorithm has five parameters. We tune each parameter in the range shown in Table 3.1. This leads to 1250 possible combinations. For each combination, we compute the predictive accuracy.

Validation Results. For our algorithm, we first computed the optimal parameters (parameters for which the accuracy is highest) as described above: they turn out to be $\gamma = 1$, $\alpha =$"confidence interval", $f_r = 0.1$, $f_u = 0.2$, $\beta = 0.1$, and $k = k' = 20$ (a relationship graph should contain at most 20 edges). We expect that these values will apply in other domains; that is because they, informally speaking, specify

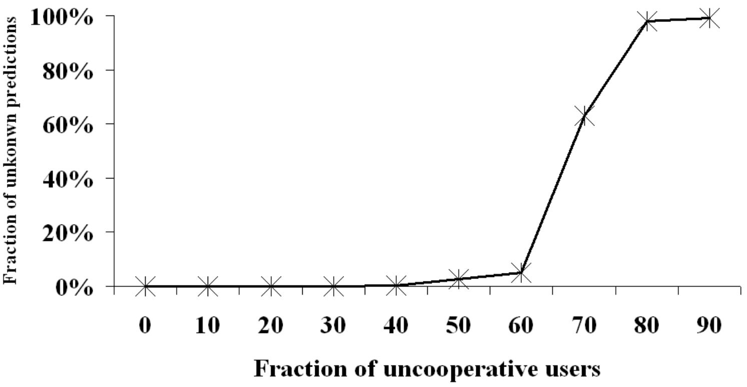


Figure 3.9: Fraction of unknown predictions as a function of uncooperative users (users who are not willing to make their ratings available).

reasonable and intuitive choices. More specifically, they indicate that to predict the rating of a trust relationship, the learning algorithm has: to consider *relevant* those relationships that include people who perform or rate alike ($\gamma = 1$); to estimate relevance by aggregating ratings using the confidence interval of their average ($\alpha =$"confidence interval"); to consider a small fraction of those relevant relationships ($f_r = 0.1$, $f_u = 0.2$); to weight the *actual* ratings more than the ratings it predicts during the learning process ($\beta = 0.1$).

This preliminary analysis allows us to move on to answer the key question: how would the predictive accuracy of our algorithm compare to that of any of the algorithms previously mentioned. Figure 3.8 shows that direct trust propagation performs better than naive prediction, but is comparable to a median of ratings. It also shows that our algorithm's accuracy is as high as 82.9% (with a confidence level of 95%, our algorithm's accuracy is within a confidence interval of $[78.5\%, 87.3\%]$.). In all cases in which our algorithm failed to predict (17.1%), the actual rating and the predicted rating differed by one only (on a scale $[1, 3]$).

3.3.2 Prediction Robustness

Uncooperative Users. All trust propagation techniques rely on knowing ratings. In mobile computing, this translates into users making available their ratings. For privacy reasons, some users may well decide not to do so. That is why we now evaluate how our algorithm would cope if different fractions of users did not disclose their ratings. Again, we measure predictive accuracy by cross validation: we mask one trust relationship and then predict its rating upon the limited knowledge of the nodes subject of the trust relationship; we do so for all relationships that link any pair of cooperative users (users willing to make their ratings available). That is because we consider that users willing to be subject to prediction are also willing to cooperate.

Figure 3.9 shows the fraction of predictions for which a relationship graph is not defined (percentage of unknown predictions) as a function of the fraction of uncooperative users. If at most 60% of the users are not willing to make their ratings available, the remaining users can still propagate their trust, and they do so with a high predictive accuracy (82.9%). However, as Figure 3.9 shows, if the number of uncooperative users reaches a critical point (if it is higher than 60%), the remaining users are abruptly unable to form a relationship graph. In other words, if at least 40% of the users make their ratings available, those users can still effectively propagate trust. For one possible explanation of this result, consider that the web of trust is a social network and that social networks are robust because they are scale-free. Albert *et al.* [AJB00] studied the fraction of nodes that must be removed at random from a scale-free network to break it into pieces: they "removed as many as 80% of all nodes and the remaining 20% still hung together, forming a highly interlinked cluster" [Bar03]. In short, our scheme does not work if more than 60% of the nodes are removed. The reason is that, to propagate trust from A to B, our scheme does not simply need a connecting path between A and B (which may likely exist even if 80% of the nodes are removed), but needs the set of links that are necessary for propagating trust (as per Section 3.2.4). And those links likely disappear after removing more than 60% of the nodes.

Sybil Users. Given that users' identifiers correspond to anonymous identifiers, one may rightly point out that user-identifier bindings need to be certified to avoid sybil attacks [Dou02], in which a malicious user takes on multiple identities and pretends to be multiple, distinct users. In mobile computing, user-identifier bindings cannot be certified by a central authority. However, those bindings may be statistically guaranteed by mechanisms similar to SybilGuard [YKGF06]. If those mechanisms prove ineffective, one might be interested in knowing whether our model is robust against sybil attacks. We evaluate the predictive accuracy of our algorithm as follows: we mask one trust relationship $A \rightarrow B$; we create n sybil identities who highly rate B; we then predict $A \rightarrow B$'s rating. We do so for all trust relationships. Regardless of n, the prediction accuracy remains unchanged (82.9%). The reason for this result is that sybil users are not connected in the same way as real users are and, as a consequence, their ratings do not influence trust propagation.

3.3.3 Overhead

Communication and Storage Overhead. Both communication overhead and storage overhead are minimal. As described in Section 3.2.4, any device stores the ratings of its outgoing relationships plus those of its incoming neighbors in a table. Each tuple of this table corresponds to a trust relationship, i.e., to two identifiers (of the connected persons) and one rating. Hence, say that the size of a tuple is roughly 10B. Even with 50 incoming and 50 outgoing edges (which is pessimistically high), the table size is 30KB. Also, for a single trust propagation, the data to be sent is less than 30KB.

Computational Overhead. We ran a J2ME implementation of our algorithm on a Nokia 3230 mobile phone whose features include: Symbian operating system 7.0, 32 MB of memory, 32-bit RISC CPU (123 MHz). In Section 3.3.1, we evaluated that a relationship graph should contain 20 nodes at most.

We run our algorithm in this worst case scenario. We minimized background activities by shutting down all applications other than our algorithm. The computational overhead, given as the mean of 10 runs, is as low as 2.8 milliseconds.

3.4 Discussion

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

Privacy Concerns. By exchanging their web of trust, users reveal their social ties (people with whom they have interacted), and some users may not feel comfortable doing so for privacy concerns [LHC07]. Our design alleviates these concerns for two reasons. First, users are identified by pseudonyms. Second, one inherent property of distributed trust propagation is that users' ratings are not made available on public servers, but each user discloses her ratings whenever she finds it convenient to do so. Moreover, during our evaluation, we found that if at least 40% of the users make available their ratings, those users can still propagate their trust without relying on any other user (Section 3.3.2).

Distrust. Previous work has shown that introducing distrust may be beneficial to trust propagation [GKRT04]. Despite not explicitly modeling distrust, we may introduce it by simply adding an additional rating level. For example, if we have 3 possible ratings, we may simply add a fourth rating representing distrust.

Dynamic Ratings. Since ratings may change over time, we have allowed for rating tables to be updated either reactively or proactively and the storage overhead and communication overhead of doing so are minimal (Section 3.3.3).

3.5 Conclusion

LDTP allows mobile users to predict their trust for content producers from whom they have never received content before. It does so by reasoning on recommendations. It scales (it entails minimal storage and communication overhead) and is effective (its predictive accuracy on a large data set is as high as 82.9%). That accuracy remains unchanged even if most of the users were not to make available their ratings. LDTP effectively runs on portable devices - a J2ME implementation spends at most 2.8ms for one propagation on a Nokia phone.

LDTP makes the assumption that the recommendations on which it reasons are available on portable devices. How devices store recommendations and make them available is the subject of the next chapter.

Chapter 4

Storing Recommendations

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 in a fully decentralized way 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 it yields to these threats: A may *lie* (it may supply fake ratings); B may pretend to be *ignorant* (it may not acknowledge the rating); or B may be *malicious* (it may tamper with the rating).

Problem Statement: How mobile users exchange and store ratings in a secure and distributed way.

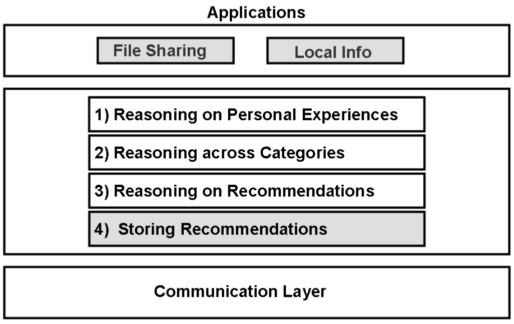

Applications
File Sharing
Local Info
1) Reasoning on Personal Experiences
2) Reasoning across Categories
3) Reasoning on Recommendations
4) Storing Recommendations
Communication Layer

Figure 4.1: Research Goal 4: Collecting Recommendations.

4.1 Existing Solutions to Collecting Recommendations

Previous work [BB02, KSGM03, LA98] has focused on the first problem (lying recommenders) and has not thoroughly explored the problem of ignorant and malicious individuals. 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 [Del00] explored the use of robust statistics in aggregating ratings of website users (e.g., eBay's sellers). Those mechanisms aimed at safeguarding the reliability of reputation websites in the presence of potentially deceitful buyers and sellers. At the same time, Raph Levin designed a way for identifying reputable members in the Advogato free software developer community [LA98]. His solution is to arrange members and their ratings in a network upon which to then run the max flow algorithm [CSRL01]. That is done to assign a trust value to each member. 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 [KSGM03] assigns a trust value to each peer similar to how Google's PageRank [PBMW98] 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 ubiquitous users would do so. One may well wonder why. Here is a possible explanation: ubiquitous devices aggregate ratings relying on *limited* knowledge. In Web applications, ratings are aggregated by one server that knows all the ratings in the system [Del00]. Instead, in ubiquitous computing, they are aggregated by each device that knows only few ratings [BB02]. Being less-informed, ubiquitous devices do not necessarily have enough knowledge to filter out fake ratings and, consequently, aggregate ratings far less effectively than Web applications [HRB$^+$08]. However, one may rightly point out that ratings can be stored on the Internet, which devices may easily access. The problem is that this solution may not scale, does not work if devices have no Internet connection, and assumes that privacy-concerned people trust specific reputation websites.

It thus seems that a different way of storing ratings for ubiquitous devices is needed. But what sort of method should we use?

What is missing: A distributed mechanism with which ratings are made both available (to counter limited knowledge) and tamper-evident (to detect tampering).

4.2 Our Proposal: MobiRate

We design one such mechanism and call it MobiRate [QHC08].

4.2.1 Assumption

MobiRate works under two assumptions:

- Each device is equipped with a short-range connectivity mechanism (e.g., infrared, wire, bluetooth). This mechanism works only if two users are within a "secure range" of each other and are able to confidentially exchange secure passcodes. This assumption makes sense in our scenarios of proximity-based content sharing, is in line with the literature [vHB03], and allows us to rule out denial-of-service and man-in-the-middle-attacks. Of course, another way of ruling out those attacks would be to authenticate every message that two devices exchange, but that would introduce

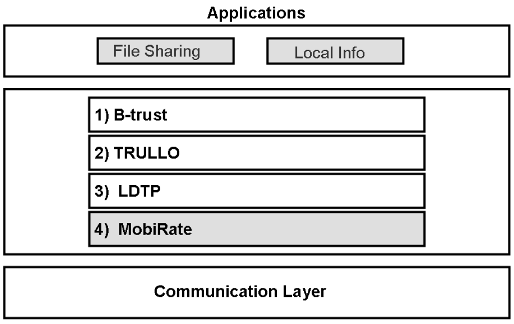


Figure 4.2: Contribution 4: Collecting Recommendations (MobiRate).

considerable overhead.

- Each device has a public/private key-pair bound to a *unique* identifier (e.g., a public key). Unique identifiers are required for avoiding malicious individuals to escape responsibility by assuming a different identity (i.e., by launching Sybil attacks [Dou02]). Binding keys to identifiers is easily done by additional infrastructures. The problem is that these infrastructures, such as admission control and public key servers, are expensive and difficult to implement in any network, let alone in a *mobile* network. That is why we sketch an alternative way of binding identities in Chapter 6.

4.2.2 Brief Description

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 experience, 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 [BBM$^+$07] (so much so that they have been aptly called “prosumers” [TW06]). 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 [Dou02]). In Section 4.4, we will look at the impact of Sybil attacks.

$P_B(Y)$	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 4.1: Symbols Used to Present MobiRate

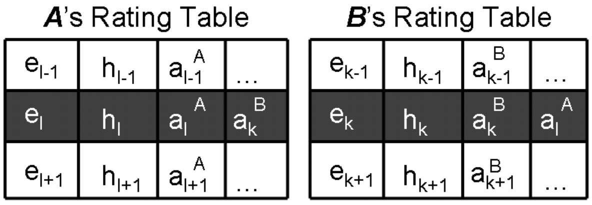
4. *How it works:* A device that runs MobiRate implements four operations (which we gradually introduce in this chapter): (1) Maintain ratings locally; (2) Commit to those ratings; (3) Gossip those commitments; and (4) Act on the commitments it receives. By implementing those operations, MobiRate guarantees that every malicious device (that tampers with the ratings it stores) will be detected *eventually* and *with high probability*. "Eventually" because we allow for some delay since devices are mobile and can communicate with each other only sporadically; and "with high probability" because MobiRate offers *probabilistic* guarantees. In Section 4.3, we will show that those guarantees are high and met in brief simulation time under the assumption (which has been found to hold in reality [CHD$^+$07, EP06, GHB08, MMC08]) that people have few regular encounters with the so-called familiar strangers: people who we do not know (strangers) but meet regularly (familiar) on, say, the way to work [PG04].

4.2.3 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 should be detectable). They do so in a way similar to how an existing technique builds tamper-evident logs [SK98]. This technique dates back to 1998, and, until recently, has inspired successful mechanisms for detecting faults in distributed systems [CMSK07, HKD07]. 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 with, the authenticator is compromised. To see why, having the notation in Table 4.1 at hand, consider that B produces an entry e_k containing the rating and appends it to its table. The entry (highlighted in Figure 4.3):

- Consists of a sequence number s_k, a rating r_k of the form "$A \rightarrow B$: 2", and additional information



***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 4.3: 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.

t_k. In 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) [QHC07b], 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. We consider that A accepts content from B only if A has received t_k from 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 are associated with sequence numbers only after they end.

- 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}). As we will see in this chapter, 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 4.4, 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.

4.2.4 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 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.

4.2.5 Gossiping Commitments

Up to now, A and B have committed to having stored the rating and have produced 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 C, then:

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

The gossip contains enough evidence for challenging B to acknowledge A's rating and for convincing any device (including C) 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 C 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:

$$\begin{array}{lll} A\textbf{'s Type 2 gossip:} & A & \rightarrow C : \{a_k^B, h_{k-1}, e_k\} \\ B\textbf{'s Type 2 gossip:} & B & \rightarrow C : \{a_l^A, h_{l-1}, e_l\} \end{array}$$

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 the case, devices would be swamped by massive storage and communication overheads. In a content sharing scenario, the devices that 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 are able to act upon those gossips (as they are co-located with B). So we consider that C is a good witness for B if both:

- C 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 [PG04]. 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 C has met B regularly before, it is likely that C will again do so in the future [HCY08].

- C shares at least one interest with B. That introduces the problem of how C 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 [blu07]. Being part of the discovery mechanism, undefined attributes are shared without any additional communication cost [MMC08].

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).

4.2.6 Acting on Gossips

To see what B's witnesses should do, let us call one of them C. C 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 C has received a gossip of Type 1, it 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, C runs the *commitment protocol* for the unacknowledged rating. If B does not respond, C marks B as *suspected* and gossips its suspicion

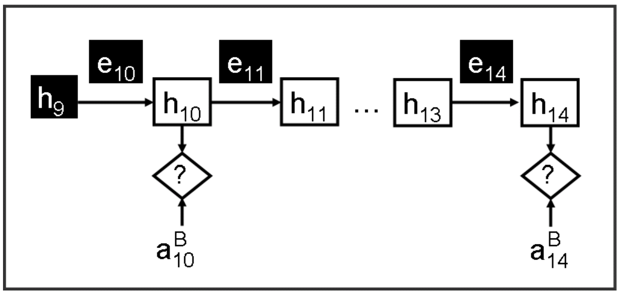


Figure 4.4: Verifiable Hash Chain. During the audit protocol on B's entries $(e_{10}, \ldots, e_{14})$, C attempts to form a chain with B's answer (highlighted) and to match that chain with two authenticators a^B_{10} and a^B_{14}. If the match succeeds, one can conclude that B has not tampered with any of the ratings in its entries $(e_{10}, \ldots, e_{14})$.

(again, a Type 1 gossip). By contrast, if B responds, C 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 C, which will eventually send it back to the rater. In Section 4.3, we will see that these messages entail little communication cost.

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

- unable to do so, then C 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^B_{10} to a^B_{14}. 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 Figure 4.4 for a sketchy representation, and Theorem 2 in the Appendix for a formal explanation). If B's answer forms a chain, C *trusts* B of not having tampered with those ratings, and the gossip dies out. By contrast, if it does not form a chain, C 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:

$$C\textbf{'s Challenge:} \quad C \quad \rightarrow B : \{a^B_k, a^B_m\}$$

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

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

C's Pair of Type 2 gossips:

$C \rightarrow D :$

$\{a_k^B, h_{k-1}, e_k\}$

$\{a_m^B, h_{m-1}, e_m\}$

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

4.3 Evaluation of MobiRate

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 (Section 4.3.1)?
- **Overhead:** What time, storage, and communication overhead does MobiRate impose on a mobile phone (Section 4.3.2)?

4.3.1 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 to interact made by honest individuals on fake or missing ratings).

Unfortunately, we do not have a deployment of mobile content sharing applications. So we need to set up a realistic simulation. We describe this in the section below 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 Section "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 [EP06]. 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 consider that two individuals interact if they come into range and have interests 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 could do so by simply 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 [Bar03]. 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 [KW06]. Therefore, to assign categories, we need to account for those two aspects. We are in the position to do 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 do unknown individuals.* 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 [TSK05]). 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 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

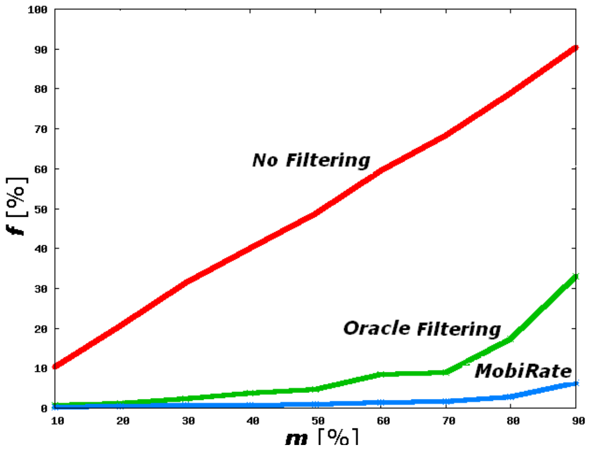


Figure 4.5: Fraction of Compromised Interactions f Versus the Percentage of Malicious Individuals m.

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 (with 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 exchange recommendations either in traditional ways or using MobiRate. When devices exchange recommendations in traditional ways, 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 4.5 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 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

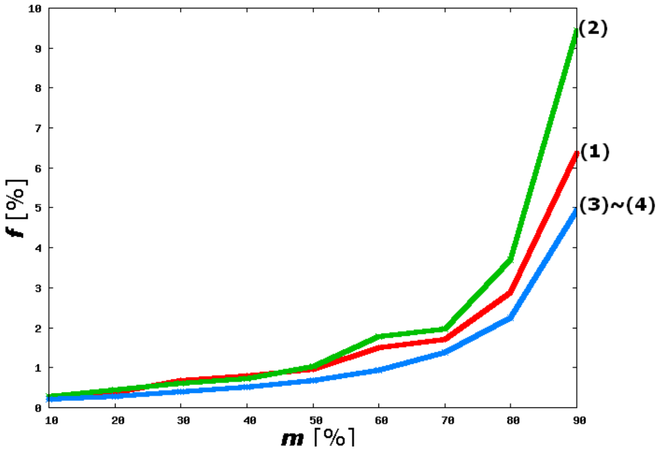


Figure 4.6: Fraction of Compromised Interactions f in Four Simulation Setups. By categorizing setups along the two dimensions $(way_{\text{interests}}, way_{\text{malicious}})$, we identify four of them: (1) (*realistic*, *realistic*); (2) (*realistic*, *random*); (3) (*random*, *realistic*); and (4) (*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.

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 available when needed.

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 4.6 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 [Jai] (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 4.6 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 less.

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 4.7 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 individuals remains the same, the number of compromised interactions roughly remains the same. For both "Oracle Filtering" and MobiRate, f

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

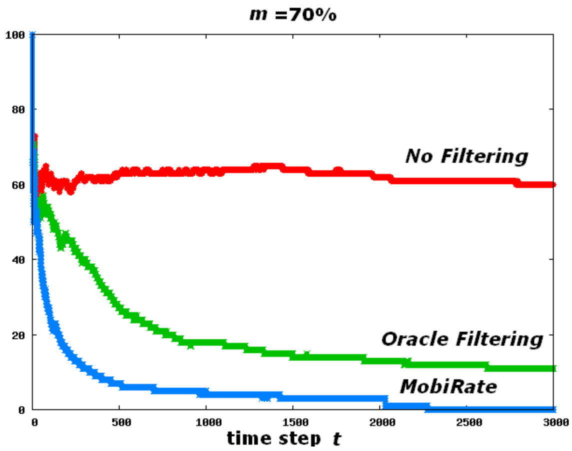


Figure 4.7: Fraction of Compromised Interactions f Over (Simulation) Time. 70% of malicious individuals in the system.

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.

4.3.2 Overhead

The overhead a mobile phone would suffer from 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 [BBB$^+$]. However, the use of RSA may slow down current models of mobile phones. So, we consider a second public key encryption algorithm - ECDSA [Don98]. We do so because, security level being equal, compared to RSA, ECDSA uses smaller keys and, consequently, signs messages faster [GMF$^+$05].

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 [TG06], 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. 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).

2. The audit protocol as challenger, it transmits 256 bytes (two authenticators each of which takes 128 bytes).

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 KBps . 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.

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 4.2.5, those individuals are a subset of familiar strangers (a *limited* group of people who one does not know but meets regularly [HCY08]).

Computational Overhead: CPU. Researchers have extensively implemented and evaluated the cryptographic algorithms that MobiRate uses [TG06]. Based on their results, we compute the execution time of our protocols on two mobile phones (Table 4.2). 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 a 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.

4.4 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 [LHC07]. Our design partially alleviates these concerns because it supports anonymous identifiers. Furthermore, users' ratings are not made available on public servers, but each MobiRate's user discloses her ratings whenever she finds it convenient to do so.

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

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 4.2: 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.

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 [Dou02]. To reduce the impact of Sybils, one needs an algorithm that filters out their ratings. In Chapter 3, we have shown that one such algorithm is LDTP, which arranges ratings in a social network of recommenders. The idea is that the more connected a recommender is, the higher the importance of her ratings becomes. 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 (i.e., oracle filtering), devices can still fall victim to 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 but, without MobiRate, they would see their ratings forged and would 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:

- *Ratings that have been tampered with are detected.* In Section 4.2.6, 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 manages two tables, then its like-minded familiar strangers would be able to detect that - they would receive contrasting evidence about B managing more than one rating table with a certain probability (as we will formally prove in Theorem 2 in the Appendix). In our evaluation of the Reality Mining dataset, we have experimentally shown that this probability is extremely high.
- *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 [MU05]. 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 the devices it has interacted (and will most likely interact) with.

4.5 Conclusion

MobiRate makes it possible to run collaborative mobile applications by storing and exchanging tamper-evident ratings. In our evaluation, MobiRate proved to be effective in singling out malicious individuals who tweak with the ratings they store - it is robust up to 90% of malicious individuals in the system. It also scales - it entails reasonable storage, communication, and computational overhead.

By introducing MobiRate, we ran down the last of the four components that made up our proposal. Now it is time to evaluate the robustness of our proposal.

Chapter 5

Evaluation of Our Algorithms

To defend mobile content-sharing communities, we have proposed and independently evaluated four algorithms, which now need to be evaluated as a whole.

5.1 Two Questions to be Answered

We need to carry out an evaluation that answers two questions:
(1) **Robustness:** How effectively do users of our algorithms protect themselves from attacks? (Section 5.3)
(2) **Overhead:** What time, storage, and communication overhead do our algorithms impose on mobile phones (Section 5.4)?

To answer the first question, we scope the ways in which communities of mobile users may be attacked within this dissertation (Section 5.2), and we then run simulations that test the robustness of our algorithms (Section 5.3). Finally, to answer the second question, we assess the extent to which J2ME implementations of our algorithms are usable on mobile phones (Section 5.4).

5.2 Threat Model

We crafted simulations similar to those we ran to evaluate MobiRate (Chapter 4). The basic idea of those simulations was that any two individuals interacted (exchanged content) if they came into range and shared interests, and that individuals could also interact with malicious people who tampered with their ratings (Section 4.3). The simulations we will now carry out are based on the same idea. The only difference is that now malicious individuals do not restrict themselves to one attack (tampering with ratings) but carry out one of eight attacks.

To introduce those attacks, we should consider that this dissertation focuses on the problem of how *A sets its trust* for B based on two types of evidence: personal experiences and recommendations. These two types of evidence may be compromised by malicious individuals (attackers) who are motivated to disrupt the communities they are in. To see how they may do so, in this section, we model attackers and, upon that, we then list the attacks they can carry out.

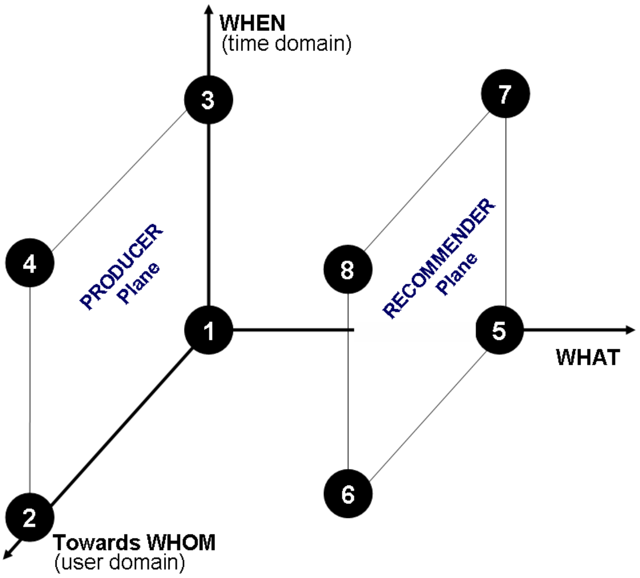


Figure 5.1: Attack Space. Malicious individuals who disrupt content-sharing communities are either producers of content or recommenders.

5.2.1 Modeling Attackers

To model plausible attackers, we list three of their most relevant traits. This list is not exhaustive, and researchers will likely add other traits to it, but it gives us a preliminary grasp of the kind of traits that we think necessary for modeling attackers. The traits are: *what* is the role of the attacker, *when* and *towards whom* does she misbehave:

- *What is the role of the attacker?* In content sharing communities, attackers are either producers of content - they send bad (e.g., low-quality, spam) content - or they are recommenders - they send untruthful recommendations[1].

- *When does she misbehave? (Time Domain)* If one differentiate attackers in the *time* domain, then attackers either misbehave at all times or misbehave only at some times (in the latter case, they do so hoping to remain undetected while causing damage).

- *Towards whom does she misbehave? (User Domain)* If one considers the *user* domain, then attackers misbehave either towards everybody or only towards targeted victims.

5.2.2 Attacks

From this three-dimensional model of attackers, we now determine how attackers operate. More specifically, we determine how they compromise the two types of (trust) evidence: personal experience and recommendations.

[1] In general, the same user may hold two roles - she can be consumer or producer of content. That is the case, for example, of a mobile user who makes available her files and, at the same time, downloads files from others. Also, we consider that whether content is of quality is decided by users. Consequently, quality assessments are subjective.

Bad Experiences. Bad experiences come from attackers who make bad content available. That is why, in our three-dimensional model of attackers, we now fix the first dimension (set "$what$ is the role of the attacker" to "producer"), and vary the two remaining dimensions: *when* and *towards whom* does the producer misbehave. By doing so, we obtain the first four attacks of Figure 5.1:

Attack 1: Always, Everybody. The attacker *always* supplies bad content. In our example of antique markets, that is the case of sellers or visitors who decide to inject fake e-ads at all times.

Attack 2: Always, Targets. The attacker supplies bad content only to targeted victims. One sees this attack in antique sellers who team up (collude) to gain reputation and increase visits to their stalls.

Attack 3: At times, Everybody. The attacker supplies bad content only at times. That may be epitomized by sellers who inject fake e-ads and do so only once in a while hoping to remain undetected.

Attack 4: At times, Targets. The attacker supplies bad content only to targeted victims and only at times. For example, sellers may send good ads only to regular customers, while they send fake ads to the remaining visitors and do so only once in a while.

Untruthful Recommendations. Recommenders may well lie and make available recommendations that are untruthful. They may do so to frame good parties or to boost the reputation of their friends. To list the attacks recommenders may carry out, in our three-dimensional model of attackers, we fix the first dimension (set "$what$ is the role of the attacker" to "recommender"), vary the two remaining dimensions (*when* and *towards whom* recommenders misbehave), and obtain four additional attacks (Attacks 5-8 in Figure 5.1):

Attack 5: Always, Everybody. The attacker always sends untruthful recommendations. For example, malicious visitors may befriend poorly-evaluated sellers and may promote them by sending very good recommendations.

Attack 6: Always, Targets. The attacker sends untruthful recommendations only to targeted victims. For instance, malicious visitors may send fake recommendations only to newcomers hoping to drive them away from the ad-sharing community.

Attack 7: At times, Everybody. The recommender sends untruthful recommendations only at times. Malicious visitors who make available fake recommendations may do so only at times hoping to go undetected.

Attack 8: At times, Targets. The recommender sends untruthful recommendations only to targeted victims and she does so only at times. To disrupt the ad-sharing community, malicious visitors may send fake ads only to newcomers and only at times.

To make our discussion tractable, we have limited the number of attacks to eight, and we have done so in two ways. First, we have considered a simple behavioral model for attackers. This model should

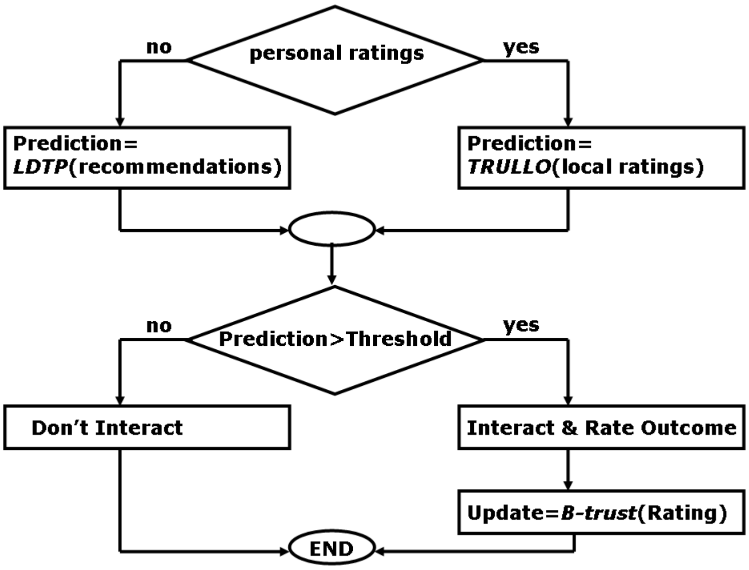


Figure 5.2: Execution Order of Our Algorithms

be extended by researchers in the security community who would put forward new formal threat models. Second, we have focused on the attacks that concern devices establishing trust. In so doing, we have ignored other attacks - for example, we have left out attacks that originate from unsecured identification of devices. One such attacks called sybil attack is very popular, and we will discuss it in Chapter 6. To recap, here are the attacks we will examine:

- A malicious content producer sends bad content always (Attack 1), or only to targeted victims (Attack 2), or only at times (Attack 3), or only to targeted victims and only at times (Attack 4).
- A malicious recommender sends untruthful recommendations always (Attack 5), or only to targeted victims (Attack 6), or only at times (Attack 7), or only to targeted victims and only at times (Attack 8).

We still need to clarify what we mean by attackers who misbehave “only at times” and “only towards targeted victims”. For the purpose of our evaluation, we consider that the attackers who misbehave:

- “only at times” are those who make available bad content/recommendations whenever they deal with specific content categories.
- “only towards targeted victims” are those who misbehave only towards strangers and stop being malicious only when they deal with their friends[2].

5.3 1^{st} Question: Are Our Algorithms Robust to Attacks?

5.3.1 Simulation Execution

To test the extent to which our algorithms defend mobile communities against those eight attacks, we compare our algorithms to existing protection mechanisms. That is, we consider that honest devices run

[2] Recall that we use the Reality Mining dataset that consists not only of mobility traces but also of a social network.

either:

- *No Protection.* If a pair of devices come into range and share interests, then they exchange content without reasoning any further.

- *State-of-the-art Protection.* To exchange content, a pair of devices not only have to come into range and share interest but they also have to decide whether the counterpart is not malicious. They do so by using a state-of-the-art proposal recently put forward by Yan Sun *et al.* [SHL08]. The idea is to have devices independently keep track of whose content and whose recommendations have been useful. By accepting content and recommendations only from sources who have been useful, devices effectively isolate attackers. They decide who has been useful by keeping two separate pairs of counters for each source - one pair for content (number of times good and bad content has been sent), and the other for recommendations (number of good and bad recommendations).

- *Our Algorithms.* Instead of using Yan Sun *et al.*'s proposal, devices run our algorithms. More specifically, Figure 5.2 shows the order with which devices run the algorithms. To predict their trust for B, a device gets the ratings from B and checks the integrity of those ratings using MobiRate; it then produces trust assessments by aggregating ratings either with TRULLO (if it has had personal experiences with B) or, otherwise, with LDTP (upon the ratings it has got from B). If the device then decides to interact, it rates the interaction and updates its trust for B using B-trust.

We consider that malicious devices carry one of the eight attacks. Then, while running our simulations, we keep track of the extent to which honest individuals manage to avoid exchanging content with malicious ones. We do so by keeping track of the number of good interactions - interactions between two honest individuals - and by then computing the precision for each protection mechanism:

$$Precision = \frac{|\text{Good interactions}|}{|\text{Interactions}|}$$

A protection mechanism that carries *only good interactions* shows a perfect precision score of 100%. Whenever confidence intervals are shown in our plots, the confidence level on these intervals is 95%.

5.3.2 Results

Our Proposal. From the simulations, the first interesting result we learn is that our proposal shows the same $Precision$ for the following groups of attacks:

Group 1: Attacks 1 and 2. Under those attacks, it shows a $Precision$ of 99.1%. These attacks differ from having malicious individuals misbehave either towards everybody (Attack 1) or towards strangers (Attack 2).

Group 2: Attacks 3 and 4. Under those attacks, our proposal shows a $Precision$ of 84.9%. In these attacks, malicious individuals misbehave only at times, and they do so either towards everybody (Attack 3) or towards strangers (Attack 4).

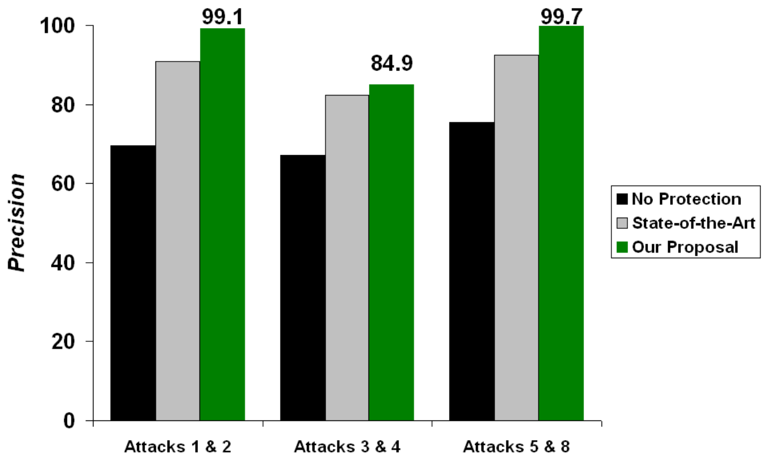


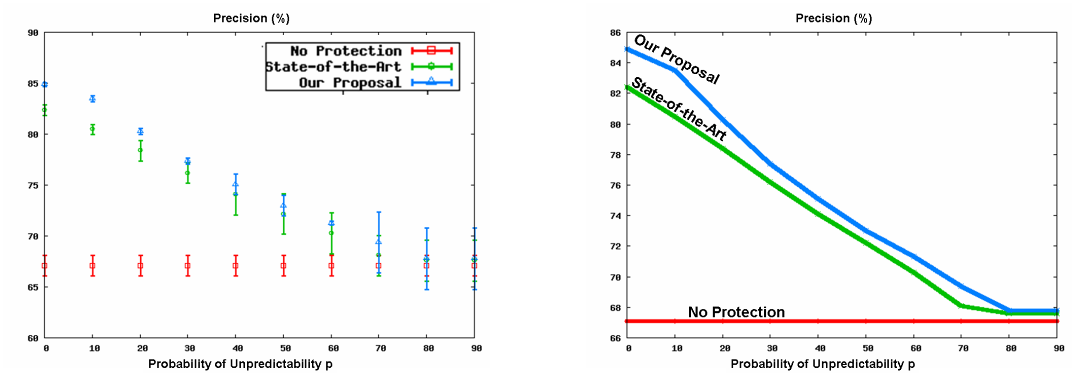
Figure 5.3: $Precision$ for Three Protection Mechanisms with a Fraction of Malicious Users $m = 30\%$. The protection mechanisms are: No Protection (black), State-of-the-art Protection (white), and Our Proposal (green).

Group 3: Attacks 5, 6, 7, and 8. Under those attacks, the $Precision$ is 99.7%. Those are the four attacks that *recommenders* can carry out.

Those groupings lead to the following preliminary comments. Groups 1 and 2 point to one interesting result: it does not matter whether attackers target everybody or only few people - our proposal is able to handle both cases. That is because, by using trust propagation (LDTP), our proposal makes it possible for two individuals to have different opinions about the trustworthiness of the same person. By contrast, traditional *reputation* systems (e.g., EigenTrust [KSGM03]) do not cope with this attack because, by definition, they assign a single reputation value to each user, and that value should be accepted and shared by all users in the system.

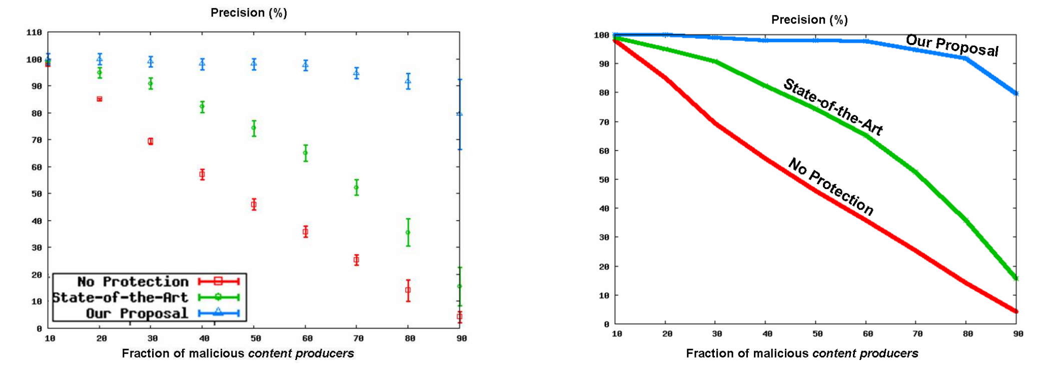
Group 3 suggests that no matter how sophisticated a malicious recommender is (e.g., no matter that a recommender sends fake recommendations only to strangers and only at times), our proposal performs in the same way. That is because, by using MobiRate, it is able to identify who tamper with the ratings it is storing. Plus, by running LDTP, it is able to filter out untruthful ratings.

Comparing Our Proposal. Figure 5.3 compares our proposal to "No Protection" and "State-of-the-Art Protection". Again, we group attacks by our proposal's $Precision$ and take the best $Precision$ for the two other protection mechanisms. The results show that the three protection mechanisms are all threatened the most by (perform worst under) Attacks 3 and 4. That means that malicious individuals who misbehave only at times are the most effective attackers. The more so, the more unpredictable they are. Figures 5.4(a) and 5.4(b) plot $Precision$ versus the probability p of attackers being unpredictable: attackers make available bad content in specific categories with probability $(1-p)$ or in randomly chosen categories with probability p. As one expects, the higher p, the lower the $Precision$ for both our proposal and state-of-the-art protection. Compared to state-of-the-art protection, our proposal performs slightly



(a) (b)

Figure 5.4: $Precision$ for Three Protection Mechanisms with a Fraction of Malicious Content Producers $m = 30\%$. Producers randomly change their behavior with probability p. $Precision$ shown with: (a) error bars; and (b) continuous lines.

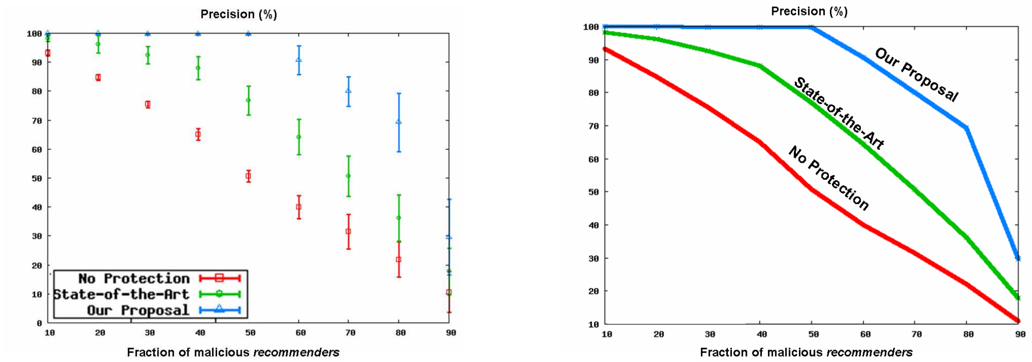


(a) (b)

Figure 5.5: $Precision$ of Three Protection Mechanisms Versus Fraction of Malicious Content Producers ($m\%$) - Attacks 1-2. $Precision$ shown with: (a) error bars; and (b) continuous lines.

better. That is because TRULLO ports ratings across categories and, as such, one is able to express, across categories, different ratings for the same individual. But, as p increases, randomness increases, and TRULLO falls short.

Growing Fraction of Malicious Individuals. One would expect that how well protection mechanisms perform mainly depends on the fraction of malicious individuals in the system (which we call m). We plot the $Precision$ for the three protection mechanisms versus the fraction of malicious *content producers* (Figures 5.5(a) and 5.5(b)) and versus the fraction of malicious *recommenders* (Figures 5.6(a) and 5.6(b)). As m increases, the two lines of $Precision$ for "No Protection" and "State-of-the-Art Protection"' decay sharply. By contrast, $Precision$ for our proposal is high for considerable fractions of malicious individuals. That is in line with the results reported in previous chapters. In Chapter 2, we showed that B-trust and TRULLO are able to select honest content *producers* by effectively isolating malicious ones. In Chapters 3 and 4, MobiRate has been proven to be robust against malicious *recom-*



(a)

(b)

Figure 5.6: $Precision$ of Three Protection Mechanisms Versus Fraction of Malicious Recommenders ($m\%$) - Attacks 5-8. $Precision$ shown with: (a) error bars; and (b) continuous lines.

menders who tweak with ratings, and LDTP has shown a high predictive accuracy against a large dataset of real ratings.

5.4 2^{nd} Question: What is the time, storage, and communication overhead?

To estimate the overhead of our proposal, we refer back to the results reported from Chapter 2 to Chapter 4, summarize them in Table 5.1, and now briefly comment on them.

Overhead	**MobiRate**	**B-trust + TRULLO**	**LDTP**	**Our Proposal**
Communication	256B (25ms)	0	30KB (300ms)	325ms
Storage	106KB	$\times n_c$	-	$106KB \times n_c$
Computational	2s	3.2ms	2.8ms	$\sim$ 2s

Table 5.1: Overhead of Our Proposal. n_c is the number of categories.

Communication Overhead. Neither B-trust nor TRULLO exchanges messages so they show no communication overhead. By contrast, LDTP needs to exchange rating tables, and MobiRate needs to exchange messages as part of its protocols.

- **MobiRate.** The heaviest of MobiRate's protocols is the audit protocol. If we consider a conservative Bluetooth speed of 100 kb/s, the audit protocol is carried out in 2.5 milliseconds.

- **LDTP.** LDTP needs to store ratings in the form of tuples. Even with 100 tuples (which is pessimistically high), the table size is 30KB. Therefore, over a slow Bluetooth connection of 100 kb/s, it would take 300ms to transfer an entire rating table.

Storage Overhead. As we have anticipated towards the end of Chapter 4, for each content category, the amount of storage required is 106KB, and that is negligible mostly because mobile phones come with GBs of flash storage nowadays.

Computational Overhead. J2ME implementations of TRULLO, B-trust and LDTP run seamlessly on mobile phones - they are executed in less than 3ms in the worst case. Only MobiRate has non-negligible computational demands. The heaviest of its protocols needs to run for 2 seconds on standard Nokia and Ericsson phones. This execution time is not however prohibitive - the cryptographic operations MobiRate uses are well-established and, consequently, they are likely to be partially or fully implemented in hardware for higher performance in the near future.

5.5 Conclusion

Our proposal defends mobile users from eight plausible attacks and often outperforms existing solutions. Importantly, it is robust even if a large fraction of the users in the system happens to be attackers, and, in so doing, it scales - it entails minimal storage and communication overhead.

More generally, from our experiments, three facts have emerged. If content producers or recommenders attack their communities, and they do so:

- *By tampering with ratings*, then it is essential to provide ways of storing ratings in a secure way and making those ratings readily available. We have done so by putting forward MobiRate, which effectively detects mobile users who maliciously tamper with their ratings.

- *By misbehaving only at times*, then it is promising to learn *when* attackers misbehave. In this chapter, we considered attackers who misbehave only when they send content/recommendations in specific categories, and we saw that TRULLO is able to distinguish an attacker who is honest in some categories and malicious in others. That is because TRULLO allows devices to assign different ratings to the same device - the device may be trustworthy for “financial news” but not for “literature reviews”.

- *By targeting only few individuals*, then it is important to make it possible for two individuals to have different trust ratings for the same person. As opposed to reputation systems that assign a single rating to each user, LDTP can assign multiple ratings to each users.

Chapter 6

Conclusion

It is now time to summarize the work we have done (Section 6.1); why we have done it (Section 6.2); and what are its limitations and future directions (Section 6.3).

6.1 Contributions

The goal of this thesis has been to design algorithms with which portable devices automatically keep track of reputable sources and learn from each other's recommendations. To this end, we have designed and evaluated:

Chapter 2 (first contribution): *An algorithm that produces trust assessments from user ratings.* After rating the content they receive, mobile users need algorithms that translate those ratings into trust assessments. That is why we have designed one such algorithm based on Bayes' theorem and called it B-trust. As opposed to existing solutions, it updates trust *formally* and takes in input ratings that are *generic* (not necessarily binary).

Chapter 2 (second contribution): *An algorithm that builds trust across (content) categories.* Say that device A received financial news from B and found them interesting. B now produces economic news. From its past experience, A may conclude that B's economic news are also of good quality since "economics" and "finance" are (semantically) similar. In general, to decide whether two categories are similar, existing algorithms typically use an ontology (e.g., a taxonomy of content categories). However, in so doing, they require that the same ontology is shared by all users (which is hardly the case in reality) and that those users agree on that ontology for good (i.e., the ontology is not supposed to change over time). That is why we designed an algorithm (called TRULLO - TRUst bootstrapping by Latently Lifting cOntext) that does away with these two problems. The idea is that devices learn statistical properties from ratings and upon those properties they then find out which categories are similar. We have showed that TRULLO effectively determines category similarity using simulations based on eBay's data and that its implementation is reasonably fast on mobile phones.

Chapter 3: *A distributed algorithm that builds trust on recommendations.* We designed a way with which devices set their trust from third-party recommendations and called it LDTP, which stands for

Lightweight Distributed Trust Propagation. Each device organizes ratings of content producers that it knows and trusts in a graph (called "Web of Trust"). It then uses a graph-based learning technique to form opinions about content producers with whom it has never interacted before. LDTP shows high predictive accuracy on a large real-world dataset, and, in contrast to existing approaches, it is fully decentralized. Plus, its JAVA implementation runs reasonably fast on mobile phones.

Chapter 4: *A set of protocols with which portable devices collect and store recommendations.* To learn from each other's recommendations, portable devices need to collect and store recommendations first. But mobile networks suffer from content providers who tweak recommendations to their own advantage. That is why we have designed a new decentralized mechanism (dubbed *MobiRate*) with which portable devices store recommendations in (local) tamper-evident tables and check the integrity of those tables through a gossiping protocol. Using real mobility and social network data, we have showed that MobiRate considerably reduces the impact of "tweaked" recommendations. We have also showed that MobiRate runs on mobile phones with little computational and communication costs.

6.2 Who Cares

To understand the potential impact of this research, consider that our algorithms and protocols can be used:

- By *mobile users* to run applications that, for example, automatically find: songs of emerging musicians - who upload their latest tracks into publicly-available WiFi hot-spots and add the date of their next gig as a note to the track for some free publicity; prices of outlets - that embed their latest offerings or discounts or seasonal menus within their clickable logos displayed on navigational maps; or impromptu street performances - advertised by disseminating electronic flyers.

- By *software developers* as ready-made tools when designing Web 2.0 mobile applications (whose market is forecasted at $22.4bn in 2013 [Jun08]).

6.3 Limitations and Future Work

We have put forward solutions to the problem of how portable devices produce trust assessments. Those solutions are limited, and there is still significant work to be done. We have recently worked around some limitations of our work concerning what A needs to do *before* and *after* making trust assessments on B: before, A needs to identify B; after, A needs to decide whether to trust B - whether to accept content from B. We now describe those two contributions and then point to future directions of research.

Identifying Portable Devices

Problem Statement: To identify B, A should associate an identifier with B and should make that identifier both:

- *Anonymous* (the identifier does not reveal B's real identity). The anonymity makes it difficult to profile mobile users (i.e., to match users' service requests with their real identities) and may encourage contributions (e.g., users rate without fearing retaliations from poorly-rated individuals).
- *Unique* (only one pseudonym identifies B). We have assumed that users' identifiers are anonymous tokens. However, by being identified through tokens, users suffer from sybil attacks [Dou02]. In this type of attack, a malicious user takes on multiple identities and pretends to be multiple, distinct users.

Existing Solutions. Over the course of nearly five years, distributed trust management and identification have begun to diverge: identification has relied on central authorities, while trust management has migrated to decentralized solutions. Only recently, trust models started to use decentralized forms of identification. Trust models commonly identify devices with pseudonyms. Pseudonyms facilitate anonymity, yet hinder cooperation in the absence of a central authority. To see why, consider individuals who cooperate. If each individual authenticates herself with an anonymous pseudonym, then she does not have to disclose her real identity and, thus, she can remain anonymous. However, she may profit from ease of creating pseudonyms. For example, she may authenticate herself with a pseudonym, misbehave, create a new pseudonym, authenticate herself with the new pseudonym (pretending to be new user), and misbehave again. As a result, she misbehaves without being traceable. Resnick and Friedman [FR01] formally laid down such a problem, presenting a game theoretical model for analyzing the social cost of allowing people to freely change identities. They concluded that, if people generate pseudonyms by themselves, new users should all be regarded as malicious. To avoid mistreating new users, they proposed the use of free but unreplaceable (once-in-a-lifetime) pseudonyms, which a *central* authority certifies through blind signature. A couple of years later, Doucer discussed the attack in the context of P2P networks in which users use multiple identities and named the attack "Sybil attack" [Dou02]. He concluded with a critical take on decentralized authentication support: in the absence of a trusted central identification authority, Sybil attacks undermine trust relationships and, thus, cooperation. EigenTrust [KSGM03] (a distributed trust model) then tried to prevent Sybil attacks by making it costly for a user to obtain multiple identities: to get a new identity, a user must perform an entry test (e.g., must read a text off a JPEG) and must send the result to a server. As such, it will be costly for a simple adversary to create thousands of users, but the presence of a server is still necessary.

More recently, researchers put forward a distributed mechanism to prevent Sybil attacks in peer-to-peer networks and named it SybilGuard [YKGF06]. To understand how SybilGuard identifies Sybil nodes, imagine that every person exchanges keys with a limited number of well-known trusted friends. The sum of these social networks shows that an attacker will only have a limited number of friends, and so will its Sybil nodes. As a consequence, Sybil nodes are identifiable. This mechanism has been validated in peer-to-peer networks and it is not meant to work in mobile networks.

> **What is missing:** A *decentralized* way for *mobile* devices to issue identifiers that are *unique* and *anonymous*.

Our Proposal: TATA. In a paper titled "Towards Anonymous Trusted Authentication (TATA)" [QHC06b], we have proposed a scheme in which A identifies B with a pseudonym. The scheme guarantees that the pseudonym is anonymous and unique and, unlike existing work, it does so in a (partially) decentralized way. We say partially because we still use public key certificates, but we do so only when identifiers are generated (i.e., only once in each identifier's lifetime). Briefly, the scheme consists of two protocols:

- The *induction protocol*, with which B obtains its pseudonym from a group of at least t devices. More specifically, B proves its identity to at least t devices; then, those devices check whether B is actually a newcomer: each device checks with $(n-2t+1)$ other devices whether they have issued a pseudonym for B. If not, B is a newcomer and so the devices create and issue a pseudonym, i.e., they blindly sign a newly generated public key. The blind signature allows the public key to be signed without it being revealed. As a consequence, the devices are not able to associate the key with B's identity.

- The *authentication protocol*, with which B sends its pseudonym to A, and A verifies whether the signature of the pseudonym is valid.

To work properly, TATA makes two assumptions:

- To generate identifiers, one needs a public key certificate. Then, to use the identifier, one does not need any public key certificate.

- To certify identifiers, one needs at least t devices to cooperate. Plus, the number of malicious individuals who forge identifiers is bounded (less than t).

Conclusion. This scheme is a decentralized way for portable devices to issue identifiers that are anonymous and unique. The problem is that the scheme needs to use public key certificates, which are difficult to distribute in mobile networks. That is why we are currently working on a way of certifying identities that exploit both mobility and social networks.

Deciding to Trust

Problem Statement: After identifying B and assessing its trust for B, A has to decide whether to rely on B (whether to accept content from B). So, the next problem is how A, on input of its trust assessments, decides whether to rely on B.

Existing Solutions. Initially, literature proposed that, to decide whether to rely on B, A has two available actions (rely / don't rely) and discriminates between the two based on its trust for B being above a fixed threshold. Of course, this solution does not work whenever A has more than two actions to choose from. In some cases, A may wish to decide among three possible actions: the usual two - whether to rely on B or not - and a third - whether to ask its user for further guidance, for example. That is why researchers proposed decision models based on risk analysis [CJGmS03, Dim03, JP04]. The idea is that, in deciding whether to rely on B, A lists all (potential) actions; weighs in the utility of each action and its risks; and chooses the action with the highest utility. This idea is promising but has been realized in ad-hoc ways (for example, solutions have not been properly grounded in the economics literature).

What is missing: A way for A to decide among *more than two actions* that is *well-grounded* on economics literature.

Our Proposal: MATE. In a paper titled "MATE: Mobility and Adaptation with Trust and Expected-utility" [QH08], we have designed a new decision model that lists possible actions and corresponding risks; assigns utility values to all actions (depending on trust ratings); and chooses the action with the highest utility. These three steps make it possible for A to discriminate among three or more actions and are well-founded on economics (more specifically, on the literature of expected utility). More specifically, here are the three steps that MATE runs on A:

- List the possible actions and corresponding risks. For example, the actions might include *"rely on B's ads"*, *"do not rely"*, and *"ask user"*. And the risks might be *"B's ads are of bad quality"*, *"of fair quality"*, and *"of good quality"*.
- Assign utility values to all actions. Utilities depend on how likely some risks are, which in turn depend on trustworthiness. For example, action *"rely on B's ads"* is of high utility if the risk *"B's ads are of bad quality"* is unlikely (if B is trustworthy).
- Choose the action with the highest utility.

As for the assumptions, MATE works only if it knows all possible actions beforehand (it does not learn new, unspecified actions on-the-fly) and if it is able to quantify (elicit) the utility its user has in carrying out each action.

Conclusion. MATE is a model of decision-making for portable devices. By applying the expected utility theory, MATE combines its user's risk attitudes and trust assessments to determine which content producers to trust. However, MATE still shows one major limitation - it assumes that users are able to numerically express their utilities in, for example, receiving good/bad content or being interrupted for further guidance. Whether human beings are effectively able to express rational utilities is debatable and needs to be tested by further research.

Future Work

We have seen that our dissertation and ongoing work show important limitations. These limitations call for new ways of:

- *Distinguishing content categories.* To handle on ratings across content categories, devices should be able to differentiate categories first. In Chapter 2, we suggested that devices could distinguish categories by using a directed graph whose nodes are categories and whose edges are keywords [CCDG05]. The idea is that a set of keywords dictates the transition from one category to the other. The problem is how to automatically learn those sets of keywords. Given the recent popularity of tagging systems, a promising way of automatically learning keywords could be tagging systems: one could learn the keywords associated with a category by analyzing how users tag content in that category [QCZ].

- *Predicting ratings.* Trust models rely on users who are willing to input ratings on their portable devices. However, to reduce the burden on the user, one may design algorithms that predict ratings: instead of explicitly requiring a user to express a rating, one such algorithm might recognize how often the user visits an antique stall and infer that it is a favorite. Predictive algorithms would likely work only in specific cases and, consequently, it is also important to research their limitations and recognize when user input would still be required.

- *Defending against sybil attackers.* Collaborative applications can be severely disrupted by a sybil attack to the point of being unusable. Our protection mechanism based on blind threshold signature shows constraints that limit its applicability to real settings. That is why it will be worth researching new decentralized defences for portable devices. We are currently experimenting with one such defence based on social networks. The idea is that each device manages two small networks in which it stores information about the devices it meets: its *network of friends* contains honest devices, and its *network of foes* contains suspicious devices. By reasoning on these two networks, the device should then be able to determine whether an unknown individual is carrying out a sybil attack or not.

Appendix A

Proofs of MobiRate's Protocols

A.1 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.□

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.□

A.2 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).□

A.3 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.□

Appendix B

Optimization of LDTP's Rating Function

Consider the graph shown in Figure 3.5 of Chapter 3. The loss $\mathcal{L}(f)$ over the whole graph is

$$\begin{aligned}
\mathcal{L}(f) \quad = \quad & \sum_{j \in R} M \cdot (f(x_j) - y_j)^2 + \\
& + \sum_{i \in U} \sum_{j \in RN(i)} v \cdot w_{ij} \cdot (f(x_i) - f(x_i))^2 + \\
& + \sum_{i \in U} \sum_{h \in UN(i)} z \cdot w_{hi} \cdot (f(x_h) - f(x_i))^2
\end{aligned} \tag{B.1}$$

where:

R: set of indices of *rated* nodes;

U: set of indices of *unrated* nodes;

$RN(i)$: set of indices of i's *rated* neighbors;

$UN(i)$: set of indices of i's *unrated* neighbors.

Expression (B.1) can be written as:

$$\mathcal{L}(f) \quad = \quad (\mathbf{f} - \mathbf{y})^\top H (\mathbf{f} - \mathbf{y}) + \mathbf{f}^\top F \mathbf{f} \tag{B.2}$$

Being F and H symmetric, the gradient is:

$$\frac{\delta \mathcal{L}(f)}{\delta \mathbf{f}} = 2H(\mathbf{f} - \mathbf{y}) + 2F\mathbf{f}. \tag{B.3}$$

To solve the optimization problem $min_f \mathcal{L}(f)$, we set the gradient to zero, $\frac{\delta \mathcal{L}(f)}{\delta \mathbf{f}} = 0$, and obtain:

$$\mathbf{f} = \Big(H + F\Big)^{-1} H\mathbf{y} \tag{B.4}$$

Bibliography

[AJB00] Reka Albert, Hawoong Jeong, and Albert Laszlo Barabasi. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, July 2000.

[ARH97] Alfarez Abdul-Rahman and Stephen Hailes. Using Recommendations for Managing Trust in Distributed Systems. In *Proc. of IEEE Malaysia International Conference on Communication*, Kuala Lumpur, Malaysia, November 1997.

[ARH00] Alfarez Abdul-Rahman and Stephen Hailes. Supporting Trust in Virtual Communities. In *Proc. of the* 33^{rd} *IEEE Hawaii International Conference on System Sciences*, volume 6, page 6007, Washington, USA, January 2000.

[Bar03] Albert Laszlo Barabasi. *Linked: How Everything Is Connected to Everything Else and What It Means*. Penguin, 2003.

[BB02] Sonja Buchegger and Jean-Yves Le Boudec. Performance analysis of the CONFIDANT protocol. In *Proc. of the* 3^{rd} *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 226–236, Lausanne, Switzerland, 2002.

[BB04] Sonja Buchegger and Jean-Yves Le Boudec. A robust reputation system for P2P and mobile ad-hoc networks. In *Proc. of the* 2^{nd} *Workshop on the Economics of Peer-to-Peer Systems*, Cambridge, USA, June 2004.

[BBB$^+$] Curt Barker, Elaine Barker, Bill Burr, Tim Polk, and Miles Smid. Recommendation for key management. In *NIST Special Publication. Revised Mar 2007.*

[BBM$^+$07] Arianna Bassoli, Johanna Brewer, Karen Martin, Paul Dourish, and Scott Mainwaring. Underground Aesthetics: Rethinking Urban Computing. *IEEE Pervasive Computing*, 6(3):39–45, 2007.

[Ber85] James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, New York, 1985.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[BLDDJ99] B. Bontempi, C. Laurent-Demir, C. Destrade, and R. Jaffard. Time-dependent reorganization of brain circuitry underlying long-mark distributed memory traces and ensure that hippoterm memory storage. *Nature*, 400:671–675, 1999.

[blu07] *Bluetooth SIG. Core Specification v2.1 + EDR.* July 2007.

[CABP05] Rajiv Chakravorty, Sulabh Agarwal, Suman Banerjee, and Ian Pratt. MoB: A mobile bazaar for wide-area wireless services. In *Proceedings of the* 11^{th} *ACM International Conference on Mobile Computing and Networking (MobiCom)*, page 228242, 2005.

[Cal96] Italo Calvino. *Six Memos for the Next Millennium.* Harvad University Press, 1996.

[Cap04] Licia Capra. Engineering human trust in mobile system collaborations. In *Proc. of the* 12^{th} *International Symposium on Foundations of Software Engineering*, pages 107–116, Newport Beach, USA, November 2004. ACM Press.

[Cap05] Licia Capra. Reasoning about Trust Groups to Coordinate Mobile Ad-Hoc Systems. In *Proc. of the* 1^{st} *IEEE Workshop on the Value of Security Through Collaboration*, Athens, Greece, September 2005.

[CCDG05] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[CF06] Dipanjan Chakraborty and Tim Finin. Toward distributed service discovery in pervasive computing environments. *IEEE Transactions on Mobile Computing*, 5(2):97–112, 2006.

[CH04] C. Janie Chang and Joanna L. Y. Ho. Judgment and Decision Making in Project Continuation: A Study of Students as Surrogates for Experienced Managers. *Abacus*, 40(1):94–116, 2004.

[CHD$^+$07] Augustin Chaintreau, Pan Hui, Christophe Diot, Richard Gass, and James Scott. Impact of Human Mobility on Opportunistic Forwarding Algorithms. *IEEE Transactions on Mobile Computing*, 6(6):606–620, 2007.

[CJGmS03] Yong Chen, Christian Damsgaard Jensen, Elizabeth Gray, and Jean marc Seigneur. Risk Probability Estimating Based on Clustering. In *Proc. of the* 4^{th} *IEEE Anual Information Assurance Workshop*, pages 229–233, New York, USA, June 2003. IEEE Systems, Man and Cybernetics Society.

[CMSK07] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *Proc. of* 21^{st} *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 189–204, Stevenson, Washington, USA, 2007.

[CNS03] Marco Carbone, Mogens Nielsen, and Vladimiro Sassone. A Formal Model for Trust in Dynamic Networks. In *Proc. of the* 1^{st} *IEEE International Conference on Software Engineering and Formal Methods*, pages 54–63, Brisbane, Australia, September 2003.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[Del00] Chrysanthos Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the* 2^{nd} *ACM Conference on Electronic Commerce (EC)*, pages 150–157, Minnesota, USA, 2000.

[Del06] Matteo Dell'Amico. Neighbourhood Maps: Decentralised Ranking in Small-World P2P Networks. In *Proc. of the* 3^{rd} *International Workshop on Hot Topics in Peer-to-Peer Systems*, Rhodes Island, Greece, April 2006.

[Dim03] Nathan Dimmock. How much is 'enough'? Risk in Trust-based Access Control. In *Proc. of the* 12^{th} *IEEE International Workshop on Enabling Technologies*, page 281, Washington, USA, June 2003.

[Don98] Don Johnson and Alfred Menezes. Elliptic curve DSA (ECSDA): An Enhanced DSA. In *Proc. of the* 7^{th} *Conference on USENIX Security Symposium (SSYM)*, pages 13–13, San Antonio, Texas, 1998.

[Dou02] John R. Douceur. The Sybil Attack. In *Proc. of the* 1^{st} *International Workshop on Peer-to-Peer Systems*, pages 251–260, Cambridge, USA, 2002.

[EP06] Nathan Eagle and Alex (Sandy) Pentland. Reality mining: sensing complex social systems. *Personal Ubiquitous Computing*, 10(4):255–268, 2006.

[epi] http://www.epinions.com/.

[FBT$^+$99] P. W. Frankland, B. Bontempi, L.E. Talton, L. Kaczmarek, S. G. Anagnostaras, S. Maren, and M.S. Fanselow. The involvement of the cortex in remote contextual rally graded retrograde amnesia of contextual fear after hippocam-fear memory. *Science*, 304:881–883, 1999.

[FDD$^+$88] G. W. Furnas, S. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings of the* 11^{th} *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 465–480, Grenoble, France, 1988.

[FMM77] George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler. *Computer Methods for Mathematical Computations*. Prentice Hall, 1977.

[FR01] Eric J. Friedman and Paul Resnick. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10(2):173–199, 2001.

[Gam98] Diego Gambetta. Can we trust trust? *In D. Gambetta, editor, Trust, Making and Breaking Cooperative Relations*, pages 213–237, 1998.

[GGT06] Thomas Griffiths, Thomas L. Griffiths, and Joshua B. Tenenbaum. Optimal predictions in everyday cognition. *Psychological Science*, 17:767–773, 2006.

[GHB08] Marta C. Gonzalez, Cesar A. Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, June 2008.

[GKRT04] Ratan Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *Proceedings of the* 13^{th} *International Conference on World Wide Web*, pages 403–412, New York, USA, May 2004.

[GMF$^+$05] Vipul Gupta, Matthew Millard, Stephen Fung, Yu Zhu, Nils Gura, Hans Eberle, and Sheueling Chang Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet. In *Proc. of the* 3^{rd} *IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, pages 247–256, Washington, DC, USA, 2005.

[GPH03] Jennifer Golbeck, Bijan Parsia, and James Hendler. Trust Networks on the Semantic Web. In *Proc. of the International Conference on Cooperative Intelligent Agents (CoopIS)*, Helsinki, Finland, August 2003.

[GZ06] Andrew B. Goldberg and Xiaojin Zhu. Seeing stars when there aren't many stars: Graph-based semi-supervised learning for sentiment categorization. In *Proc. of the International Workshop on Graph-based Algorithms for Natural Language Processing*, New York, USA, June 2006.

[Har93] Russell Hardin. The street-level epistemology of trust. *Politics and Society*, 21(4):505–529, December 1993.

[HCY08] Pan Hui, Jon Crowcroft, and Eiko Yoneki. BUBBLE Rap: Social Based Forwarding in Delay Tolerant Networks. In *Proc. of the* 9^{th} *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, HongKong, 2008.

[HDWC99] K Hermus, I Dologlou, P Wambacq, and D V Compernolle. Fully Adaptive SVD-Based Noise Removal for Robust Speech Recognition. *International Journal of Signal Processing*, 3(4), 1999.

[HKD07] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: practical accountability for distributed systems. In *Proc. of* 21^{st} *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 175–188, Stevenson, Washington, USA, 2007.

[HPW05] Mark Herbster, Massimiliano Pontil, and Lisa Wainer. Online learning over graphs. In *Proc. of the* 22^{nd} *International Conference on Machine Learning (ICML)*, Bonn, Germany, August 2005.

[HRB$^+$08] Z. Hayat, J. Reeve, C. Boutle, M. Field, and P. Tuson. Distributed Security for Decentralized Information Sharing. 2008.

[HTBA01] Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B. Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, June 2001.

[Jai] Raj Jain. *The Art of Computer Systems Performance Analysis*. Wiley. 1991.

[JP04] Audun Jøsang and Stéphane Lo Presti. Analysing the Relationship between Risk and Trust. In *Proc. of the* 2^{nd} *International Conference on Trust Management*, volume 2995, pages 135–145, Oxford, UK, March 2004. LNCS.

[Jun08] JuniperResearch. Share, Collaborate, Exploit - Defining Mobile Web 2.0. 2008.

[KLM96] L.P. Kaelbling, M.L. Littman, and Andrew Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[KP00] Eamonn J. Keogh and Michael J. Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series databases. In *Proceedings of the* 4^{th} *Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications (PADKK)*, pages 122–133, London, UK, 2000.

[KR03] Michael Kinateder and Kurt Rothermel. Architecture and Algorithms for a Distributed Reputation System. In *Proc. of the* 1^{st} *International Conference on Trust Management*, pages 48–62, Crete, Greece, May 2003. LNCS.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proc. of the* 12^{th} *ACM International Conference on World Wide Web (WWW)*, pages 640–651, Budapest, Hungary, May 2003.

[KW06] Gueorgi Kossinets and Duncan J. Watts. Empirical Analysis of an Evolving Social Network. *Science*, 311(5757):88–90, 2006.

[LA98] Raph Levien and Alexander Aiken. Attack-resistant trust metrics for public key certification. In *Proc. of the* 7^{th} *USENIX Security Symposium*, pages 229–241, Berkeley, USA, January 1998.

[LHC07] Neal Lathia, Stephen Hailes, and Licia Capra. Private Distributed Collaborative Filtering using Estimated Concordance Measures. In *Proceedings of ACM Recommender Systems (RecSys)*, 2007.

[LI04] Jinshan Liu and Valrie Issarny. Enhanced Reputation Mechanism for Mobile Ad Hoc Networks. In *Proc. of the* 2^{nd} *International Conference on Trust Management*, volume 2995, pages 48–62, Oxford, UK, March 2004. LNCS.

[Lia07] Liam McNamara, Cecilia Mascolo and Licia Capra. Content Source Selection in Bluetooth Networks. In *Proc. of the* 4^{th} *International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, Philadelphia, USA, August 2007.

[LRS$^+$03] Haiyun Luo, Ramachandran Ramjee, Prasun Sinha, Li (Erran) Li, and Songwu Lu. Ucan: a unified cellular and ad-hoc network architecture. In *Proceedings of the* 9^{th} *ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 353–367, New York, USA, 2003.

[Mar94] Stephen Marsh. Formalising Trust as a Computational Concept. Ph.D. Thesis. Department of Mathematics and Computer Science, University of Stirling, 1994.

[MMA$^+$01] Lik Mui, Mojdeh Mohtsahemi, Cheewee Ang, Peter Szolovits, and Ari Halberstadt. Ratings in Distributed Systems: A Bayesian Approach. In *Proc. of the* 11^{th} *Workshop on Information Technologies and Systems*, New Orleans, USA, December 2001.

[MMC08] Liam McNamara, Cecilia Mascolo, and Licia Capra. Media Sharing based on Colocation Prediction in Urban Transport. In *Proceedings of the* 14^{th} *ACM International Conference on Mobile Computing and Networking (MobiCom)*, San Francisco, US, 2008.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford University Technical Report, 1998.

[PG04] Eric Paulos and Elizabeth 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.

[QCZ] Daniele Quercia, Licia Capra, and Valentina Zanardi. Selecting Trustworthy Content Using Tags. In *Proceedings of the International Conference on Security and Cryptography*.

[QH05] Daniele Quercia and Stephen Hailes. Risk Aware Decision Framework for Trusted Mobile Interactions. In *Proc. of the* 1^{st} *IEEE/CreateNet International Workshop on The Value of Security through Collaboration*, Athens, Greece, September 2005.

[QH08] Daniele Quercia and Stephen Hailes. MATE: Mobility and Adaptation with Trust and Expected-utility. *International Journal of Internet Technology and Secured Transactions*, 2008.

[QHC06a] Daniele Quercia, Stephen Hailes, and Licia Capra. B-trust: Bayesian Trust Framework for Pervasive Computing. In *Proc. of the* 4^{th} *International Conference on Trust Management*, pages 298–312, Pisa, Italy, May 2006. LNCS.

[QHC06b] Daniele Quercia, Stephen Hailes, and Licia Capra. TATA: Towards Anonymous Trusted Authentication. In *Proceedings of the* 4^{th} *International Conference on Trust Management*, pages 298–312, Pisa, Italy, May 2006. LNCS.

[QHC07a] Daniele Quercia, Stephen Hailes, and Licia Capra. Lightweight Distributed Trust Propagation. In *Proc. of the* 7^{th} *IEEE International Conference on Data Mining*, Omaha, USA, October 2007.

[QHC07b] Daniele Quercia, Stephen Hailes, and Licia Capra. TRULLO - local trust bootstrapping for ubiquitous devices. In *Proc. of the* 4^{th} *IEEE International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, Philadelphia, USA, August 2007.

[QHC08] Daniele Quercia, Stephen Hailes, and Licia Capra. MobiRate: Making Mobile Raters Stick to their Word. In *Proceedings of the* 10^{th} *ACM International Conference on Ubiquitous Computing (UbiComp)*, Seoul, Korea, September 2008.

[RAD03] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust Management for the Semantic Web. In *Proc. of the* 2^{nd} *International Semantic Web Conference*, Sanibel Island, USA, October 2003.

[RBE$^+$06] Sasank Reddy, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani B. Srivastava. Participatory sensing. In *Proc. of the* 1^{st} *ACM Workshop on World Sensor Web (ACM Sensys)*, pages 117–134, Boulder, USA, October 2006.

[Rhe02] Howard Rheingold. *Smart Mobs: The Next Social Revolution*. Perseus Books Group, 2002.

[SA04] Andrei Serjantov and Ross Anderson. On dealing with adversaries fairly. In *Proc. of the* 3^{rd} *Annual Workshop on Economics and Information Security*, Minnesota, USA, May 2004.

[SHL08] Yan Sun, Zhu Han, and Ray Liu. Defense of trust management vulnerabilities in distributed networks. *IEEE Communications Magazine*, 46(2):112–119, 2008.

[SI05] Francoise Sailhan and Valerie Issarny. Scalable Service Discovery for MANET. In *Proceedings of the* 3^{rd} *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 235–244, Hawaii, USA, 2005.

[Sim79] Herbert A Simon. Rational decision making in business organizations. *American Economic Review*, 69(4):493–513, September 1979.

[SK98] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proc. of the* 7^{th} *Conference on USENIX Security Symposium (SSYM)*, pages 4–4, San Antonio, USA, 1998.

[TG06] Stefan Tillich and Johann Grossschaedl. A Survey of Public-Key Cryptography on J2ME-Enabled Mobile Devices. In *Proc. of the* 11^{th} *International Symposium of Computer and Information Sciences*, pages 935–944, 2006.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[TW06] Don Tapscott and Anthony D. Williams. *Wikinomics: How Mass Collaboration Changes Everything*. Portfolio, Penguin, 2006.

[vHB03] Srdjan Čapkun, Jean-Pierre Hubaux, and Levente Buttyán. Mobility Helps Security in Ad Hoc Networks. In *Proceedings of the* 4^{th} *ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, pages 46–56, Annapolis, Maryland, USA, 2003.

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. SybilGuard: defending against sybil attacks via social networks. In *Proc. of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 267–278, Pisa, Italy, May 2006.

[ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proc. of the* 20^{th} *International Conference on Machine Learning*, Washington, USA, August 2003.

[Zim95] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

[ZL04] Cai-Nicolas Ziegler and Georg Lausen. Spreading activation models for trust propagation. In *Proc. of the IEEE Conference on e-Technology, e-Commerce and e-Service (EEE)*, pages 83–97, Taipei, Taiwan, March 2004.