B-trust: Bayesian Trust Framework for Pervasive Computing

Daniele Quercia, Stephen Hailes, Licia Capra

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

Abstract. Without trust, pervasive devices cannot collaborate effectively, and without collaboration, the pervasive computing vision cannot be made a reality. Distributed trust frameworks may support trust and thus foster collaboration in an hostile pervasive computing environment. Existing frameworks deal with foundational properties of computational trust. We here propose a distributed trust framework that satisfies a broader range of properties. Our framework: (i) evolves trust based on a Bayesian formalization, whose trust metric is expressive, yet tractable; (ii) is lightweight; (iii) protects user anonymity, whilst being resistant to “Sybil attacks” (and enhancing detection of two collusion attacks); (iv) integrates a risk-aware decision module. We evaluate the framework through four experiments.

1 Introduction

Significant commercial benefits are predicted from the deployment of new services that pervasive computing will enable. These benefits are, however, theoretical in the absence of appropriate security. Fundamental to the creation of security are mechanisms for assigning trust to different pervasive devices. Also, it is in the nature of such devices that security mechanisms must be automatic - they must operate without the need for users to intervene. To make commercial benefits true, distributed trust frameworks may be employed as they provide security by automatically managing trust among pervasive devices.

To design a general distributed trust framework, one needs to identify its desirable properties first. From literature (e.g., see work by Liu and Issarny [9], and by Suryanarayana and Taylo [17]), those properties are: (i) be distributed; (ii) protect user anonymity, whilst providing accountability; (iii) be lightweight in terms of both required storage and scalability; (iv) minimize bandwidth demand; (v) be robust to common attacks; (vi) evolve (social) trust as humans do (e.g., trust evolves based on reputation information); (vii) support both types of recommendations (good and bad ones); (viii) incorporate the three classical dimensions of computational trust: context, subjectiveness, and time; (ix) be integrated with a decision module; (x) have a trust metric that is expressive, yet tractable.

A common limitation to many existing trust frameworks is that they deal with only a very narrow subsets of these properties. Abdul-Rahman and Hailes [1] were the first to propose the use of recommendations. Carbone *et al.* [5] then integrated more advanced aspects in a formal trust model. More recently, Liu and Issarny [9] focused on designing a (reputation-based) trust framework that integrates additional trust aspects, including robustness to some attacks.

Our contribution lies in designing and evaluating a distributed trust framework with the above ten properties in mind. Our framework: (i) uses a generic n-level discrete trust metric that is expressive (more than existing 2-level Bayesian solutions), yet tractable; (ii) incorporates the trust dimensions of subjectiveness, time and context; (iii) is lightweight in terms of required storage and bandwidth: as the number of its peering devices increases, its data structures grow linearly, and the computation and bandwidth demand remain flat; (iv) supports anonymous authentication, whilst being resistant to "Sybil attacks" [7]; (v) enhances detection of two collusion attacks; (vi) evolves trust embedding social aspects, in that : trust evolves from both direct experiences and (positive and negative) recommendations; evaluation of recommendations depends on their originator's trustworthiness and ontology view; finally, the trust metric embeds the distinction between trust levels and trust confidence; (vii) integrates a well-founded decision module. We have evaluated the framework through four experiments.

We structure the paper as follows. Section 2 introduces existing research and how our framework enhances it. As our trust evolution process is based on reputation information, section 3 defines trust and reputation. Section 4 then dwells on describing the whole trust management framework. Section 5 presents an experimental study. Section 6 concludes.

2 Related work

The body of work in distributed computational trust is littered with frameworks that are often based on social (human) considerations, sometimes attack-resistant, rarely integrated with well-founded decision modules.

Foundational distributed trust frameworks were already based on social trust considerations, in that they evolved trust based on direct experiences and recommendations, and they integrated the classical trust dimensions of context, subjectiveness, and (only later) time. Abdul-Rahman and Hailes first proposed the use of recommendations for managing context-dependent and subjective trust [1]. Although foundational, the previous approach suffered from, for example, the lack of a process for trust evolution. To fill the gap, Mui *et al.* [10] proposed a Bayesian formalization for a distributed rating process. However, two issues remained unsolved: they considered only binary ratings and did not discount them over time. Buchegger and Le Boudec [4] tackled the latter issue, but not the former: they proposed a Bayesian reputation mechanism in which each node isolates malicious nodes, ages its reputation data (i.e., weights past reputation less), but can only evaluate encounters with a binary value (i.e., encounters are either good or bad). Using a generic n-level discrete trust metric, our Bayesian framework addresses the issue. Furthermore, it discounts its trust beliefs over time (i.e., it decreases the confidence level it has in its trust beliefs). This avoids excessive capitalization on past good behavior and allows discarding old reputation information (contributing to make the framework lightweight).

Recent frameworks account for advanced social trust aspects. For example, Carbone *et al.* [5] have proposed a *formal* model for trust formation, evolution, and propagation based on a policy language. 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 [9]. They proposed a model robust to both defamation and collusion attacks. Although foundational, their work suffers from other attacks, such as privacy breaching (the lack of user anonymity protection). Of the relatively small body of academic work published in anonymity protection, Seigneur and Jensen [16] proposed the use of disposable pseudonyms. Such approach facilitate anonymity, yet hinder cooperation in the absence of a central authority due to "Sybil-attacks" [7] (attacks resulting from users who maliciously use multiple identities). Our framework enhances the detection of defamation and collusion attacks, and it tackles "Sybil attacks" (we will name the first two attacks as bad mouthing and ballot stuffing collusion attacks, respectively).

Trust frameworks' integration with decision-making mechanisms, though fundamental, is rare. Within the SECURE project, a trust model's output feeds a decision-making mechanism [6]. More recently, Quercia and Hailes [11, 12] proposed a decision model for trust-informed interactions that, on input of trust assessments, estimates the probability of potential risks associated with an action, based on which it decides whether to carry out the action. Our framework combines trust assessments in a way that such model is easily integrable.

3 Trust definition, trust properties, and reputation

We now define the concept of trust and highlight some of its properties. We will then stress trust dependence on reputation. Let us first define trust with a commonly accepted definition [8]: " *[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 advices) 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*.

Reputation relates to trust, as the following definition suggests [10]: "*Reputation [is the] perception that an agent creates through past actions about its intentions and norms"*. Actions build up reputation (the perception about intensions and norms). Direct experiences and recommendations about one entity describe the entity's past actions, which, thus, create the entity's reputation (i.e., the perception about entity's intentions and norms).

Reputation is not to be confused with trust: the former only partly affects the latter. Other factors affect trust, and they include disposition to rely more on personal experiences rather than on recommendations, disposition to forget past experiences, risk, and motivation.

4 Trust management framework

We now present our distributed trust management framework. We first provide a general overview. We discuss authentication support. We then introduce the data structures containing reputation information. After that, we describe the processes of trust evolution (i.e., updating the reputation data structures), trust formation (i.e., trustworthiness assessment), and trust decision (i.e., contemplating whether to carry out an action based on the trust formation process and on local policies).

4.1 General description of the framework

Here, we describe our framework's main processes: trust formation and trust evolution. In so doing, we resort to an abstract situation: a *trustor* p_x (trusting peer) interacts with both a *trustee* p_y (trusted peer) and a *recommender* p_r. We finally describe our trust metric.

First, p_x forms its trust in p_y by: (i) assessing the part of trust, also called *direct trust*, stemming from evaluations of its past direct experiences with p_y ; (ii) assessing the part of trust, also called *recommended trust*, from others' recommendations about p_y; (iii) combining the previous assessments to obtain the *overall trust*. We keep separated direct trust and recommended trust so that two types of collusion attacks can be detected, as we will describe in this section. Note that when p_x assesses trust (as it does in the first two steps), it just retrieves reputation data and process it.

Second, p_x evolves its trust in p_y upon obtaining new reputation information, which consists of direct experience's evaluations and recommendations. After a direct experience with p_y, p_x evaluates the corresponding outcome, and consequently evolves its direct trust in p_y. After receiving a recommendation about p_y from p_r, p_x assesses recommendation reliability, and it consequently evolves its recommended trust in p_y.

Finally, consider our trust metric. The random variables of direct trust, direct experience evaluation, recommendation and recommended trust are *discrete*: they can assume any of the following n levels $\{l_1, ..., l_n\}$. For example, with four levels ($n = 4$), we may have the following semantics for the different levels: l_1 means *'very untrustworthy'*, l_2 means *'untrustworthy'*, l_3 means *'trustworthy'* , and l_4 means *'very trustworthy'*. Since the random variables describing direct trust, recommended trust, and overall trust are discrete (i.e., they assume one of n discrete values $\{l_1, \ldots, l_n\}$), our framework has numerous advantages: (i) the random variable distributions emerge as a consequence of updates and are not fixed *a priori*, as existing models impose; (ii) a generic n-level metric is more *fine-grained* than a binary metric (for which an entity is either completely trustworthy or completely untrustworthy), as existing models impose; (iii) discrete metrics are more computationally *tractable* than continuous metrics (e.g., they do not involve the computation of integrals).

Throughout this section, we will use the following notation. $DT_{x,y}$ is a random variable expressing p_x's direct trust in p_y (($DT_{x,y} = l_\alpha$) is the event 'p_x *deems* p_y *deserves a level* l_α *of direct trust*'). $DE_{x,y}$ is a random variable expressing p_x's evaluations of direct experiences with p_y (($DE_{x,y} = l_\beta$) is the event 'p_x *evaluates the direct experience with* p_y *at a* l_β *satisfaction level*'). $RT_{x,y}$ is a variable expressing p_x's recommended trust in p_y (($RT_{x,y} = l_\alpha$) is the event 'p_x *deems* p_y *deserves level* l_α *of recommended trust*'). Finally, $SR_{r,x}$ is a variable expressing the recommendations p_r sent p_x (($SR_{r,x} = l_\beta$) is the event 'p_r *sent* p_x *a recommendation whose level is* l_β').

4.2 Authentication support

We consider that peers using our framework authenticate themselves by means of once in a lifetime anonymous pseudonyms.

To support anonymous authentication resistant to Sybil attacks, we propose the use of distributed blind threshold signature. Consider the situation in which p_x has to authenticate p_y. To protect p_y's user anonymity, the piece of information used to authenticate p_y has to be anonymous. Generally, such piece is a public key randomly generated by p_y. However, to protect against Sybil attacks, p_y has to have the limitation of possessing one and only one valid public key. We enforce such a limitation with public key certification that is both distributed (to match the distributed nature of our framework) and blinded (to protect anonymity). We propose a detailed scheme in [14].

4.3 Reputation data structures

The peer p_x stores reputation evidences locally: p_x solely relies on its local data structures to produce *subjective* trust assessments, thus being suitable for pervasive computing environments, in which peers frequently enter, leave, or simply disconnect from network domains. p_x maintains reputation-related evidence in the following sets:

C $= (c_1, \ldots, c_q)$ is the set of contexts known to p_x.

P $= (p_a, \ldots, p_z)$ is the set of peers that p_x has interacted with.

Direct Trust Set (DTS) stores direct trust levels. It contains p_x's direct trust levels in other peers. For each context c_k and peer p_y, an n-tuple $d = (d_1, \cdots, d_n)$ exists, where d_j is the probability that p_x has a l_j direct trust level in p_y (i.e., $p(DT_{x,y}) = l_j$). The relation DTS is defined as $DTS \subseteq C \times P \times D$, where $D = \{(d_1, \cdots, d_n)\}$.

Direct Experience Set (DES) stores data from which p_x assesses one of its direct trust prior beliefs. From it, p_x computes the probability $p(DE_{x,y} = l_\beta | DT_{x,y} = l_\alpha)$ for all $\beta = 1, \ldots, n$ and $\alpha = (1, \ldots, n)$, as Subsection 4.5 will discuss. DES is defined as $DES \subseteq C \times P \times EC$, where $EC = \{(EC_1, \ldots, EC_n)\}$. For each context c_k and peer p_y, n ordered sets of $n-$tuple exist: $EC_\beta = (ec_{1\beta}, \ldots, ec_{n\beta})$. To see what a single member $ec_{\alpha\beta}$ means, consider p_x deciding whether to interact with p_y. p_x has direct trust in p_y exclusively at level l_α; it decides to interact; it then evaluates the just completed direct experience with p_y at level l_β; it records such an experience by just increasing one of the member in EC: as it acted upon a l_α direct trust level and then experienced a level l_β, p_x increases the counter $ec_{\alpha\beta}$. Therefore, after each interaction with p_y, p_x does not store the interaction outcome, but it simply increases one of the counter associated with p_y. For example, if $n = 4$, p_x aggregates into 16 counters all the direct experiences with p_y.

Recommended Trust Set (RTS) stores recommended trust levels. This contains trust levels solely based on other peers' recommendations. For each context c_k and peer p_y, an n-tuple $r = (r_1, \cdots, r_n)$ exists, where r_j is the probability that p_x has l_j recommended trust in p_y (i.e., $p(RT_{x,y} = l_j)$). $RTS \subseteq C \times P \times R$, where $R = \{(r_1, \cdots, r_n)\}$.

Sent Recommendation Set (SRS) stores data from which p_x assesses one of its recommended trust prior beliefs. From it, p_x computes the probability $p(SR_{r,x} = l_\beta | RT_{x,y} = l_\alpha)$, as subsection 4.5 on trust evolution will discuss. $SRS \subseteq C \times P \times RC$, where $RC = \{(RC_1, \ldots, RC_n)\}$. For each context c_k and recommender peer p_r, n ordered sets of $n-$tuple exist: $RC_\beta = (rc_{1\beta}, \ldots, rc_{n\beta})$. To clarify the meaning of a single member $rc_{\alpha\beta}$, consider that p_x has built up a recommended trust in p_y at level l_α from all the recommendations received. It then receives an additional recommendation about p_y from p_r, which recommends a trust level l_β. p_x records how far p_r's recommendation is from other peers' recommendations by increasing one member in RC: as it had a l_α recommended trust level and received a l_β recommendation level, p_x increases $rc_{\alpha\beta}$. Thus, after receiving a recommendation from p_r, p_x does not store it, but increases one of the n counters corresponding to p_r.

The data structure design minimizes the overhead imposed on p_x, thus leading to a *lightweight* framework. All of these data structures increase linearly with the number of peers with which p_x has interacted with or with the number of contexts p_x has experienced. We thus do not require large amounts of data to be processed as we aggregate reputation-related information each time p_x either carries out a new direct experience or processes a new recommendation.

Data structure bootstrapping If peer p_x meets p_y for the first time, p_x's beliefs about p_y distributes uniformly. That is, for the peer p_y and the context c_k, p_x has: $D = (\frac{1}{n}, \ldots, \frac{1}{n})$; $R = (\frac{1}{n}, \ldots, \frac{1}{n})$; $ec_{\alpha\beta} = \Delta_d$, for $\alpha \in [1, n]$ and $\beta \in [1, n]$; and $rc_{\alpha\beta} = \Delta_r$, for $\alpha \in [1, n]$ and $\beta \in [1, n]$. In other words, to express maximum uncertainty in the initialization phase, p_x's prior beliefs equal a uniform distribution. The counter of direct experiences (recommendations) equals a constant Δ_d (Δ_r). The choice for the constant should consider that the greater its value is, the more the bootstrapping phase persist over time.

4.4 Trust formation

Whenever the trustor p_x contemplates whether to interact with a trustee, it has to assess the trustee's trustworthiness, i.e., it has to carry out the process of *trust formation*. As our model considers three types of trust, p_x carries trust formation out in three steps: (i) direct trust formation; (ii) recommended trust formation; (iii) overall trust formation.

Direct trust formation To determine its direct trust in p_y in the context c_k, p_x obtains the relation (c_k, p_y, d) from DTS. The j^{th} member of $d = (d_1, \ldots, d_n)$ is the probability that p_x has a l_j direct trust level in p_y: $p(DT_{x,y} = l_j) = d_j$.

The tuple d describes the distribution of p_x's direct trust in p_y in context c_k. For example, assuming both $n = 4$ and the semantics in subsection 4.1 on trust metric, a tuple $d = (0.8, 0.2, 0, 0)$ suggests that p_x deems p_y *'very untrustworthy'*, whereas with a tuple $d = (0.1, 0.1, 0.2, 0.6)$, p_x places more trust in p_y.

As a trustor can only have a partial knowledge about a trustee, trustor's assessments contain a level of uncertainty and have, consequently, a confidence level. In particular, the confidence level that p_x places in its direct trust assessment equals d's variance: $dtc_{x,y} = \frac{\sum_{j=1}^{n}(d_j-\mu)^2}{n-1}$, where the mean $\mu = \frac{\sum_{j=1}^{n} d_j}{n}$. As $\sum_{j=1}^{n} d_j = 1$ (i.e., the probabilities sum up to 1), then $\mu = \frac{1}{n}$. The confidence level ranges from 0 to $(1 - \frac{1}{n})$. Note that we compute the confidence level (the variance) dividing by $(n-1)$ (and not by n) because the variance we are estimating is of an unknown distribution (and not of a known one) - in general, dividing by $(n-1)$ provides an unbiased estimation of the variance of an unknown distribution.

As d's variance decreases, direct trust levels tend to become equally probable, and p_x hence places less and less confidence in its direct trust assessment. For example, assuming $n = 4$, the uncertainty of $d = (0.25, 0.25, 0.25, 0.25)$ is maximum, its variance zero, and, thus, the associated confidence level has to be minimum.

Recommended trust formation To determine its recommended trust in p_y in context c_k, p_x first obtains the relation (c_k, p_y, r) from RTS. The j^{th} member of $r = (r_1, \ldots, r_n)$ represents the probability p_x has a l_j recommended trust level in p_y: $p(RT_{x,y} = l_j) = d_j$.

For instance, assuming both $n = 4$ and the semantics in subsection 4.1 on trust metric, $r = (0, 0, 0, 1)$ suggests that the recommenders (that p_x considered so far) deem p_y totally trustworthy.

Similarly to direct trust, p_x associates a confidence level with its recommended trust: $rtc_{x,y} = \frac{\sum_{j=1}^{n}(r_j-\mu)^2}{n-1}$, where the mean $\mu = \frac{1}{n}$ and the confidence level ranges from 0 to $(1 - \frac{1}{n})$.

Overall trust formation The overall trust combines direct trust and recommended trust. For example, the probability p_x totals its overall trust in p_y at a level l_j is the weighted sum of the probabilities that p_x values both its direct trust and recommended trust in p_y at a level l_j.

Hence, to determine its overall trust in p_y in context c_k, p_x obtains both the relation (c_k, p_y, d) from DTS and the relation (c_k, p_y, r) from RTS, where $d = (d_1, \ldots, d_n)$ and $r = (r_1, \ldots, r_n)$. It then computes $\forall j \in [1, n] : p(T_{x,y} = l_j) = \sigma \cdot d_j + (1 - \sigma) \cdot r_j$, where the weighting factor σ holds the importance p_x places on direct experiences over others' recommendations. This increases as two factors increase: (i) the confidence level $dtc_{x,y}$ over $rtc_{x,y}$; (ii) p_x's subjective reliance on its own personal experiences rather than on on others' recommendations.

Similarly to direct and recommended trust, the confidence level p_x associates with its overall trust is: $tcl_{x,y} = \frac{\sum_{j=1}^{n}(p(T_{x,y}=l_j)-\mu)^2}{n-1}$, where $\mu = \frac{1}{n}$ and the confidence level ranges from 0 to $(1 - \frac{1}{n})$.

4.5 Trust evolution

The process of trust evolution updates both direct trust and recommended trust. In so doing, it incorporates social aspects of trust. Recommended trust evolves based on both good and bad recommendations that are weighted according to recommenders' trustworthiness and recommenders' subjective opinion - to account for honest and dishonest recommenders and to resolve the different ontological views of the world honestly held by different peers. Both direct and recommended trust evolutions: (i) incorporate the time dimension both to prevent peers from capitalizing excessively on good past behavior and to discard old reputation from data structures; (ii) and are based on Bayes' theorem which has "far-reaching ... implications about scientific inference and how people process information" [2].

Trust evolution through direct experience evaluation Consider p_x contemplating whether to have a direct experience with p_y in context c_k. *Before* the direct experience, p_x has the following prior beliefs (probabilities):

1. p_x has a direct trust belief in p_y. For context c_k and peer p_y, p_x finds the relation (c_k, p_y, d) from DTS, where $d = (d_1, \ldots, d_n)$ expresses p_x's direct trust belief distribution;
2. p_x has a belief that a direct experience will show a certain level of satisfaction. More formally, for context c_k and peer p_y, p_x finds the relation (c_k, p_y, EC) from DES, where $EC = (EC_1, \ldots, EC_n)$.
   From $EC_\beta = (ec_{1\beta}, \ldots, ec_{\alpha\beta}, \ldots, ec_{n\beta})$, p_x computes, for all $\beta = 1, \ldots, n$, the probability which the first row of figure 1 shows.

After interacting, p_x evaluates the direct experience with a, say, l_β satisfaction level. Based on that:

1. p_x updates its Direct Experience Set (DES). It updates EC_β (i.e., the experience counter of a l_β direct experience level) as follows: $\forall \alpha \in [1, n] : ec_{\alpha\beta} = ec_{\alpha\beta} + d_\alpha$;
2. p_x evolves its direct trust according to Bayes' Theorem as the second row of figure 1 shows.

Trust evolution through recommendation evaluation Consider now that p_x gets a recommendation from p_r about a peer p_y in context c_k and that the recommendation level is l_β. *Before* receiving the recommendation, p_x has the following prior beliefs (probabilities):

1. p_x has a recommended trust belief in p_y. For context c_k and peer p_y, p_x finds the relation (c_k, p_y, r) from RTS, where $r = (r_1, \ldots, r_n)$ and expresses p_x's recommended trust belief distribution;
2. p_x has beliefs that p_r will send certain recommendation levels. More formally, for context c_k and recommender peer p_r, p_x finds the relation (c, p_r, RC) from SRS, where $RC = (RC_1, \ldots, RC_n)$.
   From $RC_\beta = (rc_{1\beta}, \ldots, rc_{\alpha\beta}, \ldots, rc_{n\beta})$, p_x computes, for all $\beta = (1, \ldots, n)$, the probability which the third row of figure 1 shows.

After receiving a recommendation whose level is l_β:

1. p_x updates its Sent Recommendation Set (SRS). It updates RC_β (i.e., the recommendation counter associated with a recommendation level equal to l_β) as follows: $\forall \alpha \in [1, n] : rc_{\alpha\beta} = rc_{\alpha\beta} + r_\alpha$;

$$p(DE_{x,y} = l_\beta | DT_{x,y} = l_\alpha) = \frac{\text{\#events } DE_{x,y} = l_\beta \text{ given } DT_{x,y} = l_\alpha \text{ took place}}{\text{\#events } DT_{x,y} = l_\alpha} = \frac{ec_{\alpha\beta}}{\sum_{\gamma=1}^{n} ec_{\alpha\gamma}}$$

$$d_\alpha^t = \frac{d_\alpha^{(t-1)} \cdot p(DE_{x,y} = l_\beta | DT_{x,y} = l_\alpha)}{\sum_{\gamma=1}^{n} d_\gamma^{(t-1)} \cdot p(DE_{x,y} = l_\beta | DT_{x,y} = l_\gamma)}$$

$$p(SR_{r,x} = l_\beta | RT_{x,y} = l_\alpha) = \frac{\text{\#events } SR_{r,x} = l_\beta \text{ given } RT_{x,y} = l_\alpha \text{ took place}}{\text{\#events } RT_{x,y} = l_\alpha} = \frac{rc_{\alpha\beta}}{\sum_{\gamma=1}^{n} rc_{\alpha\gamma}}$$

$$r_\alpha^t = \frac{r_\alpha^{(t-1)} \cdot p(SR_{r,x} = l_\beta | RT_{x,y} = l_\alpha)}{\sum_{\gamma=1}^{n} r_\gamma^{(t-1)} \cdot p(SR_{r,x} = l_\beta | RT_{x,y} = l_\gamma)}$$

Fig. 1. Formulae that evolve prior and posterior beliefs about both direct trust and recommended trust.

2. p_x evolves its recommended trust according to Bayes' Theorem as the forth row of figure 1 shows.

In the forth row, the portion $p(SR_{r,x} = l_\beta | RT_{x,y} = l_\gamma)$ weights p_r's recommendations according to either p_r's reliability as recommender or p_r's ontological view.

Trust evolution over time As time goes by, direct trust' and recommended trust' confidence levels decrease.

Let us first see how direct trust evolves over time. As we said, the tuple $d = (d_1, \ldots, d_n)$ shows p_x's direct trust in p_y. Let t be the time elapsed from the last d's update. If $t \to \infty$ (i.e., a very long time goes by before a new update), d converges to a uniform distribution (i.e., to its bootstrapping values). To age its direct trust values, p_x decreases some of d's members while it increases others over time, so that all members sum to 1. In particular, it increases the members below $\frac{1}{n}$ (d's mean when uniformly distributed), whilst increasing the members above. More formally, let I be the indicator function, $n_d = I(d_\alpha > \mu)$ be the number of members p_x decreases, and $n_i = I(d_\alpha < \mu)$ be the number of members p_x increases. If $d_\alpha < \mu$, $d_\alpha = (d_\alpha + \delta)$. If $d_\alpha > \mu$, $d_\alpha = d_\alpha - (\frac{n_d \cdot \delta}{n_i})$.

Same considerations apply for recommended trust. The tuple $r = (r_1, \ldots, r_n)$ represents p_x's recommended trust in p_y. To age its information, p_x increases some of r's members (those below $\frac{1}{n}$), while decreasing others (those above $\frac{1}{n}$).

If some tuples, as a consequence of evolution over time, converge to the bootstrapping value, then we delete them. This saves storage space without any reputation information loss.

Trust evolution and attack detection We here expose how our framework protects against two types of collusion in certain cases, whilst enhancing their detection in the rest of the cases.

Let us first describe the two types of collusion. The first is the *bad mouthing collusion* attack. A collection of attackers colludes in that each of them spreads negative recommendations about the same benevolent entity. After evaluating those unanimous recommendations, recipients build a negative trust in the benevolent entity.

Hence, the attackers lower the benevolent entity's reputation without harming their own. For example, some peers decide to team up against peer p_y: they start spreading negative recommendations about p_y (e.g., p_y is a bad packet forwarder) so to damage its reputation. The second type of attack is the *ballot stuffing collusion* attack. Here we have a collection of colluding attackers: some offer services and others increase the remaining attackers' reputations as recommenders. The last subset of attackers (the good recommenders) send positive recommendations about those in the subset of service providers. Based on the positive opinions, a victim selects the providers. They then offer a low quality of service. The victim lowers its trust level in the abusing service providers only, whereas it still deems trustworthy the remaining attackers. To clarify, consider a peer p_y boosting its own reputation by means of colluding with three other peers p_{c1}, p_{c2}, and p_{c3}. p_{c1} sends positive recommendations about p_{c2}'s and p_{c3}'s trustworthiness as recommenders. p_{c2} and p_{c3} then send positive recommendations about p_y. Based on those, the victim (p_x) chooses as packet forwarder p_y, which drops all the packets.
The rule for re-evaluating trust assessments based on recommendations protects against both collusion types. To clarify, let us see how p_x evolves its recommended trust in p_y from a set of recommendations. p_x uses a Bayesian evolution rule that weights similar recommendations more, whilst filtering out extreme ones. If the number of false recommendations (i.e., those received from any of the collusions above) are less than honest recommendations, then the evolution rule protects against those collusion attacks.
However, if p_x receives recommendations mainly from colluding sources, the evolution rule is no more collusion-resistant.
In such cases, separating direct trust from recommended trust helps detecting both collusion attacks. In the presence of either collusion, p_x's direct trust in p_y significantly differs from its recommended trust in p_y. In particular, direct trust depicts a more trustworthy p_y than does recommended trust in case of bad-mouthing (p_y offers good direct experiences and is just subject to bad mouthing), whereas the reverse is true in case of ballot stuffing (p_y offers bad experiences, even though colluding recommenders assures p_x to the contrary).

4.6 Trust decision

To take better-informed decisions, a peer has to be able to integrate a well-founded decision module with its distributed trust framework. The trust framework produces trust assessments. p_x then uses such assessments to decide the best action to be carried out (e.g., to decide whether to forward a packet). We thus integrate our framework with a decision module that Quercia and Hailes recently proposed [12]. Such a model, local to a peer p_x, selects an action that maximizes p_x's utility. User-specified local policies influence p_x's utility.

For integration purposes, any trust framework has to adapt its output to what the decision module takes on input. Quercia and Hailes's module takes on input a single trust value and the value's confidence. On the other hand, the trust framework produces a single confidence value, but not a single trust value: it produces a distribution of trust levels (represented with the random variable T). We thus extract one single value from the distribution by means of a weighted sum of the values of each trust levels. Weighting factors increase as the corresponding trust levels increase. The condensed trust value $t_{x,y}$ (that p_x has in p_y) hence takes the form: $t_{x,y} = (\sum_{j \in [1,n]} p(T_{x,y} = l_j) \cdot \frac{j}{n})$. For example, with $n = 4$, the weighting factor for level l_1 (very untrustworthy) is $\frac{1}{4}$, while the factor for level l_4 (very trustworthy) is 1.

5 Experiments

We here describe the experimental setup and the four experiments we have conducted.

Goal: The objective of this set of experiments is to determine the impact of our trust management framework on successful packet delivery in a network configuration where part of the peers act maliciously. Such a configuration refers to a scenario in which a set of peers pool their resources so to share their Internet connectivity [13]. Benevolent peers share their connectivity, whereas malevolent ones exploit others' connectivity without actually sharing their own.

Simulated configuration: As we are interested in analyzing the local impact of our framework at a peer level, we simulate a configuration consisting of a peer p_x and a set of corresponding next-hops. These are connected directly to Internet. We consider p_x forwarding packets to its next-hops, which make available their connectivity. p_x selects a next-hop either randomly or through two types of trust-informed decisions (discussed later). The next-hop acts according to the behavioral model to which it belongs.

Next-hop behavioral models: A next-hop belongs to one of the following four behavioral models: fully malicious, malicious, benevolent, and fully benevolent. Depending on its behavioral model, a next-hop offers the following packet loss ratios if it was selected for the whole simulation duration: 100% for a fully malicious next-hop, 70% for a malicious one, 30% for a benevolent one, and 15% for a fully benevolent one. Both fully malicious and malicious next-hops drop packets randomly, whereas both benevolent and fully benevolent do it according to a *Gilbert model* [3]. To understand why, consider that the next-hops are connected directly to Internet. As a consequence, packet losses through (fully) benevolent next-hops depend on Internet congestion, which is bursty. A Gilbert model reproduces such burstiness. We have thus implemented the model whose parameters varied according to packet loss ratios it simulated (either 30% or 15%).

Next-hop selection methods: A peer p_x chooses its next-hops in three different ways. The first is *random* selection, i.e., it selects each of its next-hops with equal probability. The second is *pure trust-informed* selection, i.e., it selects the most trustworthy next-hop. The third is *probabilistic trust-informed* selection, i.e., p_x selects its next-hop p_y with a probability P_y that is directly proportional to p_x's trust in p_y: $P_y = \frac{t_{x,y}}{\sum_j t_{x,j}}$, where j represents each of p_x's next-hops. As we will see, we introduce the latter selection method as a better load balancing alternative to the pure trust-informed method.

Simulation execution: A simulation consists of several executions of an experiment. An experiment duration is of 100 time units. At each time unit, p_x selects one of its next-hops and sends it a stream whose size is 10 packets. Based on the number of packet losses, p_x computes its satisfaction and consequently evolves its trust. We collect the overall number of packet losses at each time unit. We run each experiment 10 times and the results of all runs are averaged.

Experiment metrics: We consider two metrics. The first is p_x's average fraction of successfully sent packets. The second is the load distribution among p_x's next-hops.

We now describe four different experiments. For each, we describe goal, setup, and results.

Experiment A

Goal: To understand whether a more-fine grained trust metric gives a greater average fraction of successfully sent packets.

Setup: We simulate p_x with four next-hops, one for each next-hop behavioral model. p_x first uses a framework whose trust metric is binary ($n = 2$). It then uses a more fine-grained metric, i.e., $n = 4$. The next-hop selection method is pure trust-informed.

Results: Switching from the binary trust metric ($n = 2$) to one that is more fine-grained ($n = 4$), p_x improves its average fraction of successfully sent packets from 67% to 83%. Figure 2 shows that the more fine-grained trust metric outperforms the binomial one.

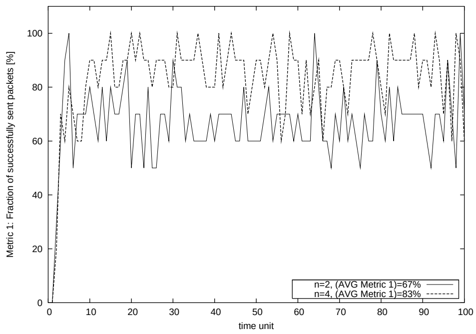

Fig. 2. Experiment A. Fraction of successfully sent packets in the case of p_x using a framework based on pure trust-informed selection with a binomial trust metric $n = 2$ (continuous line) and with a more fine-grained one $n = 4$ (dashed line).

Experiment B

Goal: To understand whether pure trust-informed selection gives a greater average fraction of successfully sent packets than random selection.

Setup: We simulate a peer p_x with four next-hops, one for each next-hop behavioral model. We first consider p_x using random next-hop selection. We then consider p_x using pure trust-informed selection. For both cases, $n = 4$.

Results: When using pure trust-informed selection, p_x successfully sent 84% of the packets on average, in contrast to 42% when using random selection.

Experiment C

Goal: To understand whether probabilistic trust-informed selection gives a better load distribution than pure trust-informed selection, whilst showing a greater fraction of successfully sent packets than random selection.

Setup: We simulate a peer p_x with five next-hops, one for each next-hop behavioral model plus an additional benevolent next-hop. The additional next-hop may lead to more interesting results for the discussion about load balancing. With a constant $n = 4$, p_x applies in turn the three next-hop selection methods.

Results: From figure 3, we note that (i) pure trust-informed selection shows an unbalanced load share: the fully benevolent next-hop (fb) has a 96% of such a share; (ii) probabilistic trust-informed selection shows a better load share, whilst penalizing malicious next-hops: the fully malicious (fm) one has received 9% of the traffic in contrast to 29% of a fully benevolent (fb). However, probabilistic selection leads to an average fraction of successfully sent packets of 60%, that is worse than pure trust-informed selection (83%), but better than random selection (47%).

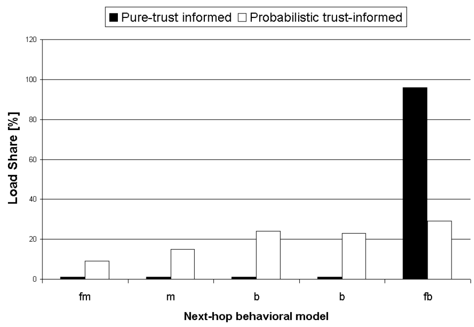

Fig. 3. Experiment C. Load share among p_x's next-hops, which include: one fully malicious (fm), one malicious (m), two benevolents (b), and one fully benevolent (fb). p_x uses both pure trust-informed (filled bars) and probabilistic trust-informed (empty bars) selections.

Experiment D

Goal: To understand which factors have an effect on the average fraction of successfully sent packets. We consider two factors, each with two extreme levels. The first factor is n whose levels are 2 and 4. The second factor is the next-hop selection method p_x uses: its levels are probabilistic and pure trust-informed.

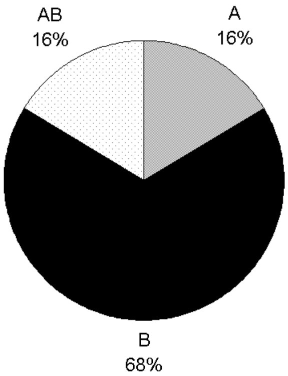

Fig. 4. Experiment D. The impact on the average fraction of successfully sent packets of: (i) the change of trust metric (factor A); (ii) whether the trust framework is used (factor B); (iii) the combination of both (factor AB).

Setup: We simulate a peer p_x with four next-hops, one for each next-hop behavioral model. We set $n = 2$. We first consider p_x using random selection. We then consider p_x using pure trust-informed selection. We then set $n = 4$ and repeat what we did before after setting $n = 2$.

Results: Figure 4 shows that the change of trust metric (from n=2 to n=4) has a positive impact (16%) on the average fraction of successfully sent packets. It also confirms the intuition that the use of the trust framework has the most significant impact (68%).

6 Conclusion

We have presented a distributed framework that produces trust assessments based on direct experience evaluations and on (both good and bad) recommendations. All of this is based on a Bayesian formalization, whose generic n-level trust metric improves on existing Bayesian solutions (which use binary metrics). The framework is lightweight and integrates a well-founded decision module. Furthermore, it supports user anonymity by means of pseudonyms, whilst being robust to "Sybil attacks". It also enhances detection of two types of collusion attacks. Finally, we have conducted four experiments which shows that the use of our framework and a more fine-grained trust metric have a considerable positive impact on packet delivery in a network where part of the peers act maliciously.

As part of future work, we plan to design mechanisms for trust bootstrapping (i.e., how to set the initial trust in an unknown entity).

References

[1] A. Abdul-Rahman and S. Hailes. Using Recommendations for Managing Trust in Distributed Systems. In Proceedings of IEEE Malaysia International Conference on Communication, Kuala

Lumpur, Malaysia, November 1997.

[2] Amir D Aczel. Chance. High Stakes, London, 2005.

[3] M. Arai, A. Chiba, K. Iwasaki. Measurement and modeling of burst packet losses in Internetend-to-end communications. In Proceedings of the IEEE International Symposium on Dependable Computing, pages 260-267, Hong Kong, 1999.

[4] Buchegger and J.-Y. L. Boudec. A robust reputation system for p2p and mobile ad-hoc networks. In Proceedings of the 2^{nd} Workshop on the Economics of Peer-to-Peer Systems, Cambridge, MA, USA, June 2004.

[5] M. Carbone, M. Nielsen, and V. Sassone. A Formal Model for Trust in Dynamic Networks. In Proceedings of the 1^{st} IEEE International Conference on Software Engineering and Formal Methods, pages 54-63, Brisbane, Australia, September 2003.

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

[7] J. R. Douceur. The Sybil Attack. In Proceedings of the 1^{st} International Workshop on Peer-to-Peer Systems, pages 251-260, Cambridge, U.S., March 2002. Springer-Verlag.

[8] D. Gambetta. Can we trust trust? . In D. Gambetta, editor, Trust, Making and Breaking Cooperative Relations, pages 213237. Basil Blackwell, Oxford, 1998.

[9] J. Liu and V. Issarny. Enhanced Reputation Mechanism for Mobile Ad Hoc Networks. In Proceedings of the 2^{nd} International Conference on Trust Management, volume 2995, pages 4862, Oxford, UK, Mar. 2004. LNCS.

[10] L. Mui, M. Mohtsahemi, C. Ang, P. Szolovits, and A. Halberstadt. Ratings in Distributed Systems: A Bayesian Approach. In Proceedings of the 11^{th} Workshop on Information Technologies and Systems, New Orleans, Louisiana, USA, December 2001.

[11] D. Quercia, S. Hailes. MATE: Mobility and Adaptation with Trust and Expected-utility. To appear in the International Journal of Internet Technology and Secured Transactions.

[12] 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 through Collaboration, Athens, Greece, September 2005.

[13] D. Quercia, M. Lad, S. Hailes, L. Capra, S. Bhatti. STRUDEL: Supporting Trust in the Dynamic Establishment of peering coaLitions. Proceedings of the 21^{st} ACM Symposium on Applied Computing. April 2006. Dijon, France.

[14] D. Quercia, L. Capra, S. Hailes. TATA: Towards Anonymous Trusted Authentication. In Proceedings of the 4^{th} International Conference on Trust Management, Lecture Notes in Computer Science, Pisa, Italy, May 2006. Springer-Verlag.

[15] S. Ross. A first course in probability. Macmillan College Pub. Co., 1994.

[16] J.-M. Seigneur and C. D. Jensen. Trading Privacy for Trust. In Proceedings of the 2^{nd} International Conference on Trust Management, volume 2995, pages 93107, 2004. Springer-Verlag.

[17] G. Suryanarayana and R. N. Taylo. A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications. ISR Technical Report number UCI-ISR-04-6, University of California, 2004.