Detecting and reacting to changes in reputation flows

Sini Ruohomaa, Aleksi Hankalahti, and Lea Kutvonen

University of Helsinki, Finland
sini.ruohomaa@cs.helsinki.fi,
WWW home page: http://cinco.cs.helsinki.fi/

Abstract. In inter-enterprise collaboration, autonomic services from different organizations must independently determine which other services they can rely on. Reputation-based trust management in the Pilarcos open service ecosystem combines shared experience information on the actors' past behaviour and the decision context to estimate the risks of a collaboration. The trust decision process is semi-automatic, with selected decisions forwarded to a human user. A particularly interesting feature of the decision process is incongruity, that is, unexpected changes in service performance. In the classical example, a previously well-behaved service turns malicious to cash in its good reputation as ill-gained monetary profit. If the reputation system swiftly reacts to such changes, it protects its user more efficiently and deters misbehaviour. We present a new model for detecting and reacting to incongruities in a reputation-based trust management system. The model is based on the concept of reputation epochs, dividing an actor's reputation into periods of internally consistent behaviour. In contrast to earlier approaches, this model provides the necessary flexibility for the trust management system to adjust to constantly changing business situations.

1 Introduction

In inter-enterprise collaboration, services from different organizational domains join together to fulfil a mutual goal. In the open service ecosystem, the services are autonomous and there is no centralized control on the collaboration process. Each service must independently determine which other services it can rely on, both in the sense of making its own resources available, and in expecting that the other collaborators do their part in realizing the joint goal. To support these decisions, it collects experience information on the past performance of other services, both first-hand and shared by other actors in the ecosystem. This experience information forms the reputation of a service. Due to the decentralized nature of open service ecosystems, reputation is subjective to the actor who has collected and analyzed the information; it is not globally agreed upon by all.

Reputation-based trust management supports making trust decisions on different services, in the context of a given collaboration and possibly a specific set of transactions within it. The decision process consists of two parts [6]: First,

reputation information is combined with a model of the decision context in order to estimate the risks of a positive decision. Second, this estimate is compared to a risk tolerance policy for the particular type of decision in order to determine the outcome: yes, proceed; no, withdraw; or, if the result falls on a gray area between the two, forward the decision to a human user.

A semi-automatic decision process is necessary due to the complex and constantly changing business environment. Automation is perfect for handling routine cases, which we expect to form the majority of decisions. Unforeseen situations and uncertainty, on the other hand, cannot be comprehensively dealt with by exact rules, and are left for a human user to resolve. Due to this division, the trust management system must be able to detect when a decision is not routine.

Incongruities, i.e. sudden changes in service performance, are a challenge for the automated decision process. In the classical example, a previously well-behaved service turns malicious to cash in its good reputation as ill-gained monetary profit. As reputation largely depends on shared experiences, information sources can also change their behaviour, and begin to spread misinformation.

We require a reputation system to swiftly react to such changes, both to limit the losses of the decision-maker service, and to discourage misbehaviour by minimizing the gains from it. At the same time, an honest actor's reputation should not be irreparably tarnished due to a momentary outage or dishonest negative feedback. These conflicting requirements are impossible to meet with a single universal, or fixed, set of rules; instead, reputation-based trust management systems must have support for appropriately handling incongruity through policies that can be configured — and re-configured — according to the business situation.

This paper presents a new model for detecting and reacting to incongruities in a reputation-based trust management system. We build on our existing Pilarcos trust management system, presented in earlier work [7]. Our model separates reputation analysis policies from trust decision policies in order to be able to use the same reputation information in a variety of decision-making contexts. Similarly, we separate the detection of behaviour changes from how their existence should affect upcoming trust decisions.

The model is based on the novel concept of *reputation epochs*, periods of internally consistent behaviour. A detected change in the behaviour pattern causes a change of reputation epoch. To illustrate the concept, we compare a few simple algorithms for identifying a pattern and detecting when it is broken. For this, we draw inspiration from the field of anomaly detection. The power of the concept goes beyond our examples, however, and it provides unprecedented flexibility for handling incongruity in reputation evolution.

The paper is organized as follows: Section 2 presents the problem context and related work, Section 3 presents reputation epochs, and evaluates the costs and benefits of introducing them. Section 4 presents three example epoch detection policies for use in three different scenarios, and Section 5 presents simulation results on how the policies behave in practice for each scenario. Section 6 concludes the paper.

2 Background

In this section, we first introduce the context in which we utilize reputation flows to for trust management in Pilarcos. We then present related work on the topic of adjusting to changes and uncertainty in reputation flows, and relate our proposal to the existing research.

2.1 Reputation-based trust management in Pilarcos

The reputation-based trust management system we propose has been implemented as a part of the Pilarcos open service ecosystem [5]. Besides trust management, the Pilarcos collaboration management tools provide automated support for setting up collaborations, including interoperability checking and contract negotiation, and runtime monitoring, including contract breach detection and recovery. For signing contracts, actors are required to have persistent identities.

The trust management process can be divided into two parts: the trust decisions, and the evolution process of the reputation information. Both are governed by their own, separate policies; this separation is necessary in order to be able to use the once-collected reputation information in different decision contexts.

A trust decision is triggered at specific points of the collaboration process, whenever resources are being committed and risk evaluation is needed. To evaluate the risk of proceeding with the collaboration, we predict the outcome it would have on different assets based on previous experiences, stored in reputation information [6]. For the purposes of this paper, we only consider the effects on the monetary asset. These effects are divided into a scale of minor or major, positive or negative effects on the asset, no effect, and unknown effect for cases when the outcome cannot be observed for all assets.

To complete the trust decision, the risk estimate is compared to a risk tolerance policy, which essentially categorizes the risk into three options: acceptable (proceed with the collaboration), unacceptable (withdraw) or uncertain (forward the decision to a human user). The risk tolerance policy defines a minimum level of certainty required for an automated decision; certainty is influenced by a set of factors, measuring the amount and quality of reputation information used in the decision. A central measure of the quality of the information is its credibility — reputation based on local observations is ultimately credible, while the credibility of external reputation information varies [6, 8].

The evolution of reputation information allows the system to adjust to changes in partner behaviour. New experiences from collaborations are stored into the system from two kinds of sources: a flow of first-hand experiences from local monitors, observing parameters such as quality of service and detecting any breaches of contract, and flows of external experiences, reported by other actors through reputation networks. A reputation network is the combination of a reputation system, defining the processes of dissemination, calculation and evolution of reputation scores, and the actors using it. We expect that third-party reputation information relevant to a specific service will be scattered over a number of different reputation networks. The local trust management system will merge the

reputation information from different sources, taking into account how relevant it is and how credible the source is considered to be.

The epoch detection process we are proposing will take place after reputation flows from different networks have been analyzed and merged into a single external reputation flow, and compared to the flow of local experiences. The ordering is significant in that an epoch change must be based on credible information; errors should be minimized to avoid situations where outlier experiences trigger unnecessary epoch changes. An epoch change affects both local and external reputation streams simultaneously.

2.2 Related work on changes and uncertainty in reputation flows

While there exists a notable body of research on reputation systems [8, 2], the evolution of reputation remains a largely uncharted area. A typical approach to addressing changes in behaviour is time-based discounting of old experiences [2], which ensures that newer experiences gain a greater weight in decision-making. While this does mean that bad behaviour is reflected in the actor's reputation value somewhat faster, an increase in reaction speed also directly increases the speed in which old information is lost — and any old transgressions forgotten.

Simulation experiments, used for comparing reputation metrics (e.g. [10, 4]), commonly assume consistent behaviour from both the well-behaved actors and any attackers. While it is quite reasonable to expect that an attacker intelligently adjusts its behaviour to the policies in place, the simulated attacker behaviour that we have observed so far is far from optimal. It is revealing that the most simple baseline attack against reputation systems, one where an attacker first gains reputation through good behaviour and then uses it up with bad behaviour, is dubbed the "disturbing" attacker model by Schlosser et al. [10]. The name reflects how poorly the compared systems could handle strategic changes in behaviour. Our proposal directly addresses this issue.

A promising example of related work, TrustGuard, proposes a reputation calculation formula which can react to incongruity [11]. In the model, decisions on a given actor are based on a trust value calculated as a weighted sum of three factors: 1) the current experiences, i.e. reports received in a given time period, 2) the (optionally weighted) average of accumulated experiences overall, and 3) a "derivative" of the current reports, representing whether there has been a recent change for the better or worse in the reported experiences.

The TrustGuard model is one-dimensional: it assumes the experience report values are real numbers between 0 and 1, and the result is a single real number. The Pilarcos model, in contrast, is multidimensional, with reputation represented as a vector of observed outcomes in relation to different assets and a set of certainty measures. In other words, the trust score alone does not fix the decision.

TrustGuard has no concept of certainty; the calculated trust value is the sole basis of a trust decision. The first factor represents the idea that current experiences should be emphasized in the decision. However, as other time-based discounting systems, it discounts all other information equally, independent of their information content or context. The second, accumulated average factor

can emphasize the time-based discounting, or treat stored experiences equally. The derivative, on the other hand, could capture sudden changes in behaviour. The larger the change, the larger the derivative value. Given a high weight, the factor could speed up reaction to a current drastic change considerably. On the other hand, it would equally strongly reward any change for the better, partially overriding even the accumulated reputation or lack thereof. Information about past changes in behaviour is not retained.

A few other reputation systems introduce certainty measures for the resulting decision. For example REGRET [9] defines a certainty measure based on the amount of information used, the variation in experience values, the subjective confidence of the recommender in the information they give, and social relationships. If the certainty of the calculated reputation value is too low, the model includes an option for calculating reputation using different approaches, such as falling back from actor-specific experiences to the reputation of the social group the actor belongs to. The amount of information stored is tracked also by e.g. Travos [12]. SECURE, on the other hand, equates certainty with the particularity of information [1]: it defines trust values as ranges between 0 and 1, and the wider the range is, the less certainty there is, as it excludes fewer values as incorrect.

Pilarcos has a computational measure for the degree of certainty [7]. As a result, specific trust decisions can be identified as needing human attention. We can also take advantage of knowing how inconsistent an actor's behaviour has been in the past. If an actor changes its behaviour all the time without a discernible pattern, a reputation system should be able to detect that the information it has is insufficient for the task at hand — it cannot predict the unpredictable actor's future behaviour.

The option to selectively involve an external human actor based on certainty is generally not used in existing reputation systems, which are designed to either be fully autonomous or to only provide supporting information for a human decision-maker [8]. Even for a fully automated system, certainty could be used for activating further levels of computation, such as REGRET does. Indeed, it is natural to divide decision processes in partially supervised systems to multiple levels as well, with a human intervention as the last option if other attempts fail. We have discussed different levels of policy in earlier work [7].

3 Tracking changes with reputation epochs

When analyzing experience and reputation information for decision-making, a central concern is whether the information is up-to-date, i.e. describes the current behaviour of the actor in question well enough to be useful in trying to predict its future behaviour. When faced with ten positive experiences and two negative, it can make quite a difference if the negative experiences are ancient and the actor's recent behaviour has been spotless, or if the good experiences are mostly older and the two negative are the most recent experiences available of the actor.

Whether it is possible to detect the order or timing of actions depends on the reputation information model. Storing the ordering of experiences sets heavy requirements on the way experiences are stored and processed: treating each experience as a unique object with a timestamp, or with a position in a queue of experience objects, creates large data structures, which take an increasing time to process as the number of experiences grows. While a theoretical model of reputation can accommodate for infinite amounts of experience information, practical models must adapt to time constraints at decision-making time: either information must be compressed, which loses information, or old experiences must be purged after a while.

We have chosen to compress experience items into outcome counters. The compression tradeoff loses timing information. The basic reputation data structure places equal weight to all experiences, independent of the time they were gathered. While it would be possible to discount old information through e.g. aging factors, we find that such methods in practice steadily lose information: they form a kind of fixed-size window to the past. The main problem with fixed windows, in turn, is that they cannot be easily adjusted at the time of the decision: data is already lost while it is gathered. For example, past transgressions can be completely erased from such systems by simply flooding the network with new experiences from low-value real transactions or false experiences produced by colluding partners.

Considered against the goal of reacting to changes in behaviour, we find that time is actually *not* the optimal measure for determining the weight or value of a unit of experience at all. Instead, we should measure whether the experience brings new information; something we did not already know.

We propose to divide reputation information into reputation epochs, groups of abstract periods of a given type of behaviour. While the latest turn of behaviour is the most interesting, it is also typical that there is very little experience on it; hence information from older epochs should also be included. The weight given to the current epoch determines the speed in which the system reacts to changes in behaviour. As an example, a pessimistic decision could even be based on the worst ever observed behaviour. The number of epochs also provides a measure of the consistency of the trustee: if experiences on the trustee are divided into a large and constantly increasing number of epochs, it indicates that the trustee's behaviour is not stable — or that it is not entirely fitting into any behavioural categories the system can detect.

While reputation epochs do allow us to give less weight to old information, they are superior to time-based constant discounting in two aspects: First, as epochs are based on behaviour changes rather than strict time periods, they fit the purpose of detecting when information is outdated in the sense of not being useful for predicting future behaviour. Second, the weighing policy between new and old information can be dynamically changed, and as no information is actually discarded, the oldest experiences remain available for later analysis: the reputation system can be configured to never forget anything without straining the time-constrained decision-making process.

We assume that the number of epochs will not grow without limit, and can therefore be processed in real time. It is also possible to set up epoch pruning processes that ensure the number of distinctly stored epochs remains under control. For example, if it turns out that an actor's behaviour regularly fluctuates between two types of epochs, older epochs can be regularly merged, as the fluctuations then actually represent a different type of consistent behaviour in the long term.

4 Detecting changes in service performance

As a baseline risk tolerance policy, we will use a simple additive policy that gives more weight to negative than positive outcomes (by a factor of 3), and further weight on major effects as opposed to minor effects (another factor of 3) [7]. As an example, an actor with 10 experiences stored, of which 4 represent major positive outcomes, 2 minor positive, 2 minor negative and 1 major negative, would receive a *trust score* of $(4 * 3) + 2 + 2 * (-3) + 1 * (-3) * 3 = -1$. We set the trust score threshold for a positive decision to 0, so this would result in a negative decision. Changes in behaviour are then addressed through epoch change policies: for simplicity, our simulations will use only the current epoch to calculate this trust score.

In the general case, detecting changes between reputation epochs can be approached through the reasonably well-studied problem of anomaly detection [13]. Anomaly detection algorithms are often based on an example set of "normal" values learned from earlier data, which are used to form a model of normal behaviour; anything deviating from the model is then an anomaly.

In more specific cases, even quite simple epoch change policies can be suitable. We present two example policies to achieve two different goals:

Load balancing: a service provider usually provides good service, but occasionally the service quality varies depending on the number of incoming requests. The first example policy should quickly react to a drop in the quality of service, as it also indicates a need for load balancing.

Oscillation detection: a service provider oscillates between good and malicious behaviour: first it collects good reputation, then it cuts corners in as many service transactions as it can. Whenever there is a fixed decision policy in use that is known or can be deduced by experimenting, the optimal attacker targeting the reputation system will collect just enough positive reputation to not be shut out of the community, which makes this kind of behaviour relevant to address. The second example policy should quickly react to this kind of change for the worse, but also take advantage of the service returning to normality.

In the load balancing example, we apply a simple dynamically learning algorithm: a window of n previous experiences is stored by the epoch change detector, and whenever a new experience falls outside the values present in the existing filled window, a new epoch is created. As normal service quality is indicated by

the vast majority of experiences, the window is typically filled with such experiences. At the first drop in reputation, a new epoch and a new, empty learning window are created. While the disturbance goes on, the window is filling up with negative (or less positive) experiences. During this learning phase, when the epoch contains less than n experiences, new epochs are not created. In our simulations in the following section, we set $n = 10$.

If the window (n) is set to be shorter than a typical disturbance, it will be full of negative experiences by the time the service returns to normal load, and a new epoch is started when the first positive experience arrives. This leads to a swift return to the service provider when it is no longer overloaded. For a more pessimistic, slow recovery, the window (n) can be chosen to be longer than a typical disturbance, which means that reputation is slowly regained within the newest epoch. Again, once the window fills up with normal experiences, the first sign of a negative experience causes a new epoch to be started. A limitation of this policy is that if the experiences indicating normal or overloaded states have some natural variation, new epochs may be created too easily.

In the oscillation detection example, the difference between good and malicious behaviour is simple to observe, as the experiences will be polarized: positive or negative. To allow greater variation in behaviour than the previous policy, we apply a static, specification-based epoch detection algorithm. We define two behaviour profiles: "good" and "evil". The good profile covers positive experiences, the evil profile negative. Neutral experiences, or those representing unknown outcomes, fall in neither category.

Given these profiles, we define each ongoing epoch to be either good or evil, and the epoch changes if an incoming experience matches the opposite profile rather than the current one. Neutral or unknown outcomes do not change the epoch, as they match neither. Again, the ongoing epoch can in principle be given full weight in decision-making. On the other hand, the attacker may respond by oscillating on every service request: cooperate, defect, cooperate, defect. To withstand this kind of behaviour, the number of epochs or the number of experiences in the current epoch should play a part in choosing a better weight division between the current and previous epochs, or indicate that the decision should really be delegated to a human user due to high uncertainty in the reputation information.

The two above policies perform at their best when the central source of reputation information is either first-hand experience, or a single highly credible reputation network. On the other hand, sometimes experiences on an actor are only available through a low-credibility reputation network, where there may be errors in the experience information — either intentional misinformation or due to e.g. differences in measurement standards. To cover this scenario, we extend the oscillation detection case above with an additional requirement:

Conservative oscillation detection: a potentially oscillating service provider is only known through a reputation network where some experience reports are incorrect. The third example policy should be cautious in trusting rep-

utation information that is out of the ordinary, and treat it as an outlier unless it is backed up by additional information.

For this requirement, we apply the idea of sequential hypothesis testing [14], which can be used to limit the probability of overreacting in anomaly detection [3].

In sequential hypothesis testing, a single experience out of the ordinary does not yet change the epoch. It only strengthens the hypothesis that the epoch should be changed, by a constant measure i; we set $i = 1$ for the purposes of this text. Similarly, an experience supporting the current epoch weakens the hypothesis by 1. Again, neutral and unknown experiences cause no effect. For the epoch to change, either the change must amass support exceeding a given threshold k, or during a period of t consecutive experiences there must be more support for changing it than there has been for continuing the current epoch. For our experiments in the following section, we set $k = 5$ and $t = 10$.

Combined to the oscillation detection policy, we get the following algorithm:

```
% Initialize and reset support and timer variables to 0.
for each round:
  if experience and epoch match: % (both good / both evil)
    if timer == 0:
      skip to next round;
    else:
      support--;
  if experience and epoch mismatch: % (one good, one evil)
    support++;
  timer++;
  if support >= k:
    change_epoch();     % Overwhelming support for change.
    reset_variables();
  if support < 0:
    reset_variables();
  if timer >= t:
    if support > 0:
      change_epoch();   % Majority of t votes supports it.
    reset_variables();
```

Sequential hypothesis testing can be similarly combined to the window-based load balancing algorithm, to test the need to change epochs once the learning window has been filled. However, this modification alone will not stop outliers in incoming reputation information from being stored as examples of normal behaviour during the learning process. Therefore, the algorithm would remain vulnerable to any noise in reputation flows.

5 Comparison of epoch detection policies

We illustrate the behaviour of the policies proposed in the previous section by applying them to four scenarios with different experience input. These reputation flows are manually constructed to illustrate the kind of a situation each policy is designed to handle. For realistic applications, a combination of detection policies is likely to be better suited than any specialized single policy.

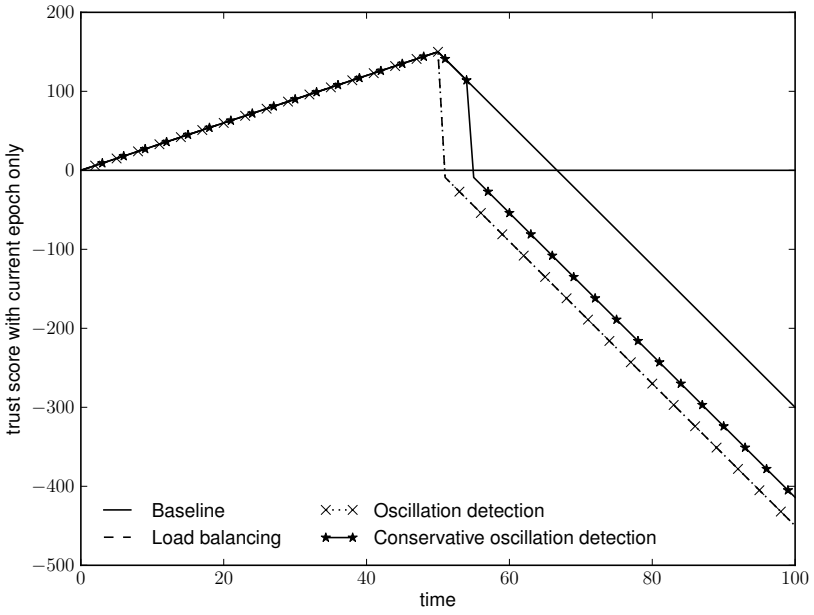

Fig. 1. Scenario 1: A change for the worse.

In the first scenario, a service behaves well (major positive experiences) for the first half of the time period, then misbehaves (major negative experiences) for the latter half; this situation could be caused e.g. by a service going permanently offline, getting hacked or otherwise becoming untrustworthy. For a traditional additive trust decision policy, its trust score (positives – negatives) would remain positive to the end of this simulation, the trusting service suffering considerable losses during that time. As discussed in the previous section, however, our baseline policy for comparison is already more strict: it gives more weight (x3) to negative than positive experiences. Our threshold for a positive trust decision is set to 0, i.e. when the calculated trust score drops below 0, trust decisions become negative.

The plots only follow the trust score calculated from the current epoch. As seen in Fig. 1, the two first epoch detection algorithms (Load balancing and Oscillation detection) react instantaneously to a change in behaviour; their trust scores are equal. The more conservative version of oscillation detection follows closely behind; the sequential hypothesis tester waits until 5 consequtive evidences of the change in behaviour have arrived, then triggers an epoch change. The baseline algorithm, on the other hand, patiently waits until the old good experiences are "used up" before turning negative. Even though it gives triple weight to negative experiences, it lags behind the epoch change policies in reaction speed (by 17 rounds).

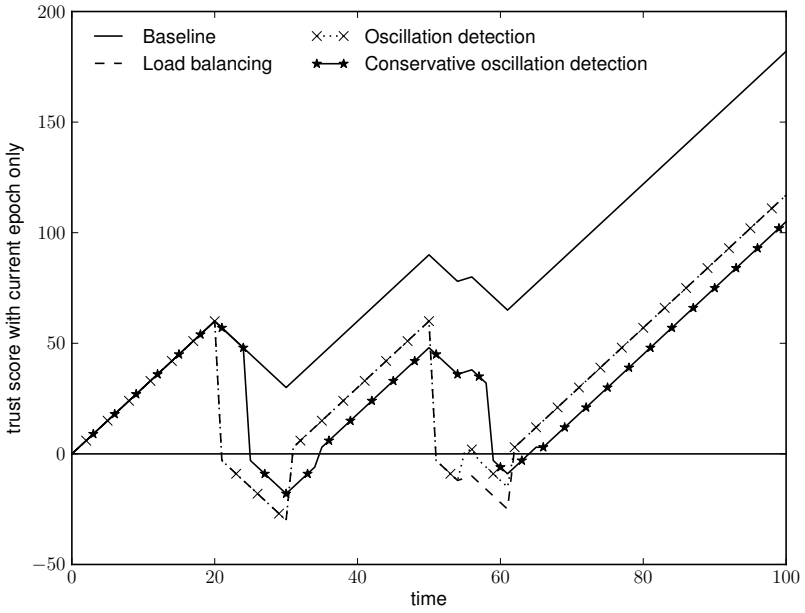

Fig. 2. Scenario 2: A need for load balancing.

In the second scenario, the monitored service generally behaves well (major positive experiences), but suffers two drops in service quality due to overload — at rounds 20 and 50, the flow outputs *minor* negative experiences to reflect e.g. a slow response time. The first disturbance lasts for 10 rounds, after which the service recovers. The second disturbance is less predictable, with two minor positive experiences arriving in the middle of it — in other words, the response time was reasonable, but the service had not actually recovered and was congested again soon after. The scenario is depicted in Fig. 2.

Using the baseline policy, the service has a positive trust score throughout the simulation. The weight on negative experiences is cancelled out due to their lesser impact: positive experiences are major, while the service quality drops produce minor negative experiences.

The epoch detection policies react quickly to the first outage; again, the Load balancing and Oscillation detection algorithms behave the same way for the first service quality drop, and immediately recover as service quality returns to normal. In the second disturbance, however, the 10-round-long learning window of the Load balancing policy is only filling up when the temporary recovery occurs, and it does not change epochs in the middle of its learning phase. On the other hand, the Oscillation detection algorithm reacts somewhat too quickly to the temporary change to positive. As before, the Conservative oscillation detection algorithm reacts with a delay while it accumulates evidence of the change; the inconsistency during the second disturbance simply delays its epoch change by two rounds.

The third scenario represents the "disturbing" behaviour pattern [10], an actor who oscillates between good and bad: the service collects just enough good reputation to not be shut out of the system when it denies service. This can also be interpreted as an opportunistic or "lazy" service provider who is ready to violate contracts to cut costs, but takes care to not burn up its reputation entirely. The length of the different oscillation phases is optimized for a given trust decision policy. In the case of our baseline policy, it behaves well 75% of the time (minor positive experiences), and misbehaves 25% of the time (minor negative experiences); as a result, its trust score is never negative with the baseline policy. This scenario is depicted in Fig. 3 on the next page.

The Oscillation detection and Load balancing algorithms react immediately to the first transgression. Their differences become apparent when the quality of service increases again: The policy optimized for regular oscillation changes the epoch in order to take advantage of the period of good service, up until the next disturbance. Meanwhile, due to the shortness of the period of low service quality (8 rounds), the Load balancing algorithm is still in its learning phase when the change for better occurs; it only changes epoch at the next drop in service quality. By then its window has filled with positive experiences (at round 56), and the current epoch's reputation has just climbed to zero, resulting in a mirror plot of the baseline policy. Finally, the Conservative oscillation detection algorithm's reactions are slightly delayed versions of those observed for the basic Oscillation detection.

The strategic oscillation presented here is only an optimal attack against the baseline policy. The Oscillation detection policy would appear to severely discourage misbehaviour, although it is also instantly forgiving. This combination of epoch detection and trust decision policies is altogether only feasible as an example to build on; it reacts too absolutely to a single negative or positive experience.

In the fourth scenario, noise is introduced to a basic oscillation pattern, i.e. experience reports which disagree with the norm. We expect this kind of pattern

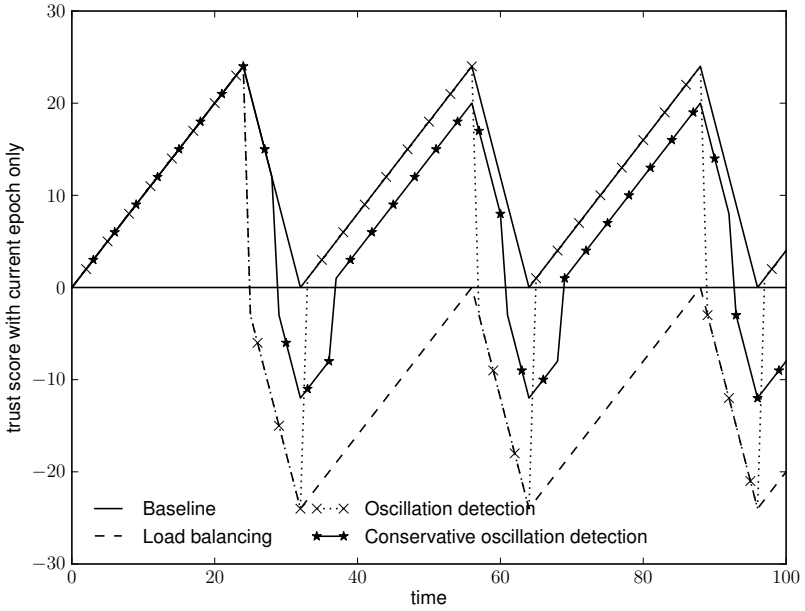

Fig. 3. Scenario 3: Oscillating behaviour.

to be more realistic than reputation flows consisting of unanimous agreement between sources, which makes it interesting to see how the different policies can handle it. The scenario is depicted in Fig. 4 on the next page. First, two neutral experiences ("no effect") interrupt a pattern of minor positive experiences. In the second "oscillation cycle", the series of minor negative experiences is interrupted by two minor positive experiences. Finally, both the positive and negative halves are affected, by three minor negatives on the positive side, and two major positives on the negative side.

The Load balancing algorithm becomes somewhat unstable in this scenario. It reacts to the introduction of neutral experiences in round 12; due to its full learning window not containing any of them from before, it is the only policy strongly affected by this kind of noise. It changes epochs again when the negative half of the oscillation begins, and again at the second drop in service quality. During the second drop, minor positive experiences are introduced in the learning window, and therefore the policy does not change epochs at the third rise, but only at the third drop — which, ironically, zeroes out the already negative trust score before it drops again. A final epoch change is introduced as the last oscillation cycle turns downwards. Similarly, the Oscillation detection algorithm reacts strongly to every change in behaviour except the neutral experiences, zigzagging between positive and negative trust scores.

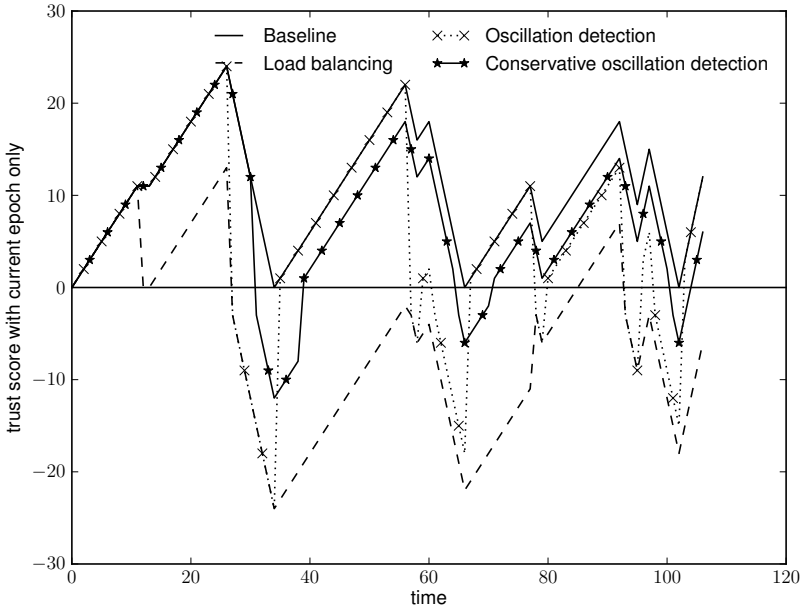

Fig. 4. Scenario 4: Oscillating behaviour with noise.

The Conservative oscillation detection policy endures the introduced noise, and only changes epochs once per each longer change in behaviour. It first deviates from the baseline policy once it has observed 5 minor negative experiences in the first cycle; it similarly ignores the bumps in the second and third cycle. The cost of this stability is the same as observed in all these scenarios: a slightly delayed reaction to changes, as the algorithm awaits for supporting evidence.

6 Conclusion

An important strength of reputation-based trust management, as opposed to more static, certification-based trust management, is in its ability to adjust to changes in behaviour. This strength has gone underutilized in the past, as proposed reputation systems provide quite limited means of detecting or reacting to incongruity in reputation information.

In contrast, the epoch-aware reputation model we have proposed provides the necessary flexibility for our trust management system to adjust to constantly changing business situations.

A central design cornerstone has been to keep separate the policies for updating reputation information, and making decisions based on it: this allows us to use the same reputation data in multiple decision contexts, without having

to collect it repeatedly. As a logical extension, we also separate the detection of behaviour changes, i.e. epoch change policies, from the reaction to such changes, i.e. policies on weighing different epochs and detecting when behaviour is too inconsistent to warrant an automated decision. The first process is governed by reputation update policies, and the latter by trust decision policies.

Trust management for inter-enterprise collaboration is inherently policy-driven; it must be designed to be reconfigured at runtime as the need arises. As Pilarcos is a general inter-enterprise collaboration management infrastructure, it must be able to cater for very different needs. We have demonstrated in this paper how different scenarios warrant different approaches, and proposed suitable policies for four example scenarios. These scenarios also demonstrate the expressive power of the epoch concept; it can be adjusted according to the situation, while for example time-based discounting can only be set up once.

Addressing changes in the behaviour of external sources of reputation information remains an important item of future work. Spreading misinformation into reputation networks must have a negative impact on the source's own reputation. To fulfil this requirement, we are currently researching contractually governed reputation systems based on objective, verifiable experiences.

Acknowledgements

This work has been performed in the CINCO group at University of Helsinki. We thank the anonymous reviewers for their encouraging and helpful comments on improving the manuscript.

References

1. Cahill, V., et al.: Using trust for secure collaboration in uncertain environments. Pervasive Computing 2(3), 52–61 (Aug 2003), http://ieeexplore.ieee.org/iel5/7756/27556/01228527.pdf
2. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decision Support Systems: Emerging Issues in Collaborative Commerce 43(2), 618–644 (Mar 2007), http://dx.doi.org/10.1016/j.dss.2005.05.019
3. Jung, J., Paxson, V., Berger, A.W., Balakrishnan, H.: Fast portscan detection using sequential hypothesis testing. In: Proceedings of the IEEE Symposium on Security and Privacy. pp. 211–225 (May 2004), http://dx.doi.org/10.1109/SECPRI.2004.1301325
4. Kinateder, M., Baschny, E., Rothermel, K.: Towards a generic trust model - comparison of various trust update algorithms. In: Herrmann, P., Issarny, V., Shiu, S. (eds.) Proceedings of Trust Management: Third International Conference, iTrust 2005, Paris, France, May 23–26, 2005. LNCS, vol. 3477, pp. 177–192. Springer-Verlag (Apr 2005), http://www.springerlink.com/link.asp?id=qh7db1k6uqr3bl76
5. Kutvonen, L., Ruokolainen, T., Ruohomaa, S., Metso, J.: Service-oriented middleware for managing inter-enterprise collaborations. In: Global Implications of

Modern Enterprise Information Systems: Technologies and Applications. pp. 209–241. Advances in Enterprise Information Systems (AEIS), IGI Global (Dec 2008), http://www.igi-global.com/reference/details.asp?id=9648

6. Ruohomaa, S., Kutvonen, L.: Making multi-dimensional trust decisions on inter-enterprise collaborations. In: Proceedings of the Third International Conference on Availability, Security and Reliability (ARES 2008). pp. 873–880. IEEE Computer Society, Barcelona, Spain (Mar 2008), http://dx.doi.org/10.1109/ARES.2007.123
7. Ruohomaa, S., Kutvonen, L.: Trust and distrust in adaptive inter-enterprise collaboration management. Journal of Theoretical and Applied Electronic Commerce Research 5(2), 118–136 (Aug 2010), http://www.jtaer.com/aug2010/ruohomaa_kutvonen_p7.pdf
8. Ruohomaa, S., Kutvonen, L., Koutrouli, E.: Reputation management survey. In: Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES 2007). pp. 103–111. IEEE Computer Society, Vienna, Austria (Apr 2007), http://dx.doi.org/10.1109/ARES.2007.123
9. Sabater, J., Sierra, C.: Reputation and social network analysis in multi-agent systems. In: AAMAS '02: Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems. pp. 475–482. Bologna, Italy (2002), http://doi.acm.org/10.1145/544741.544854
10. Schlosser, A., Voss, M., Brückner, L.: On the simulation of global reputation systems. Journal of Artificial Societies and Social Simulation 9(1) (Jan 2006), http://jasss.soc.surrey.ac.uk/9/1/4/4.pdf
11. Srivatsa, M., Xiong, L., Liu, L.: TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks. In: WWW '05: Proceedings of the 14th International Conference on the World Wide Web. pp. 422–431. ACM Press, New York, USA (May 2005), http://doi.acm.org/10.1145/1060745.1060808
12. Teacy, W.T.L., Patel, J., Jennings, N.R., Luck, M.: TRAVOS: Trust and reputation in the context of inaccurate reputation sources. Autonomous Agents and Multi-agent Systems 12(2), 183–198 (Mar 2006), http://www.springerlink.com/content/2h56k13n37qk0274/
13. Viljanen, L.: A survey on application level intrusion detection. Tech. rep., University of Helsinki, Department of Computer Science (2005)
14. Wald, A.: Sequential tests of statistical hypotheses. The Annals of Mathematical Statistics 16(2), 117–186 (1945), http://dx.doi.org/10.1214%2Faoms%2F1177731118