Evidence-Based Trust Metrics in Web Services

Hamman W. Samuel

Dept. of Computing Science, University of Alberta, Edmonton, Alberta, Canada
{hwsamuel@cs.ualberta.ca}

Abstract

In a service ecosystem of complementary and competing web services, clients have many options but, at the same time, determining which one to trust can be a challenge. While there have been several proposals in literature about web services trust measurement, none has been actually adopted. Even the notion of trust itself is not clearly established in the context of the web-services' stack of standards. In this paper, we propose a service that collects different types of service-quality measurements from clients, as well as evidence in the form of relevant request/response headers, in order to produce aggregate trust metrics of service providers. Our experimental analysis using simulations shows that the proposed trust-aggregator service framework is feasible and effective in measuring trust metrics.

1 Introduction and Background

Consider a traveler looking for an inexpensive and convenient flight, who discovers an on-line service that offers an option that appears to meet his criteria of price, dates and number and length of intermediate stops. Feeling quite successful, our traveler may book the ticket only to discover later that the service is unable to confirm the booked ticket and has to issue a refund.

It turns out that the selected service, although it appears to aggregate and compare flights for multiple providers, does so only periodically; the actual booking process relies on staff, manually booking flights through the relevant airline offices, after receiving a client request. Clearly, this underlying process is likely to make this service appear less trustworthy than a competitor with real-time data aggregation. More importantly, it would be desirable to enable clients (users and organizations alike) to rank (or somehow comment on) the trustworthiness of services, their processes and data, so that new clients can make more informed decisions with regard to service selection in the future.

Considering this scenario within the broad context of web services and Service-Oriented Architectures (SOA), we argue that, in an ecosystem that includes a number of alternative services comparable in terms of functionality and Quality of Service (QoS) indicators, potential clients may rely on "trustworthiness" metrics to select among them. The question then becomes about defining "trust" and the factors that influence it. There are many definitions of trust; according to Child [1], the general consensus is that trust involves an entity willingly interacting with another, while holding the belief that the interaction will be at least self-beneficial, and in the best case mutually beneficial to all entities involved. While there is no guarantee that this belief is correct, trust is essential for interactions to happen. Fundamentally, trust enables action even when given limited or non-existent knowledge about another (group of) partner(s). It enables a positive valuation of actions before the actions are performed, thereby helping deal with uncertainty about the future or about the reactions of collaborators.

The research literature in the field of SOA and web services suggests that the notion of trust is not clearly understood.

Firstly, trust is often confused with security [2]. Security can be classified as hard and soft security: "hard security" deals with low-level securitization, such as encryption, while "soft security" includes trust. And while security may be a criterion for characterizing the system as trustworthy, this is only one contextual definition of trust, namely "identity trust". "Provision trust", on the other hand, refers to the reliability and quality of services [3]. There have been efforts towards incorporating the notion of trust in the context service selection. The majority of the existing literature suggests that clients rate services for various trust metrics, related to QoS properties. However, evidence-based methodologies are lacking, and the suggested instrumentation approaches rely on monitoring.

Also, most implementations have considered all clients as equally credible, while others have suggested development of trust models similar to [4], based on tokens such as security certificates, as well as bootstrapping for unknown entities. Our approach takes a cue from research work on behavioral trends of clients [5].

The objective of our work is to address some of these shortcomings by developing a loosely coupled SOA solution to enable measurement of provision of trust metrics. Consequently, we propose a *trust-aggregator* web service, to which clients and providers can subscribe, for providing evidence for, and querying about, trust-reputation ratings of service providers. The trust-aggregator service is responsible for combining client evidence and ratings to create trust profiles of service providers.

To evaluate our proposal, we developed a simulation through which we aim to demonstrate the validity of the service ratings, in the presence of typical behavior as well as in the presence of Sybil-like attacks, with "bot" clients attempting to dilute the ratings, by providing misleading ratings [6, 7]. The properties that we investigate in our evaluation include the trustworthiness of the client, the quality of the provider, and the presence/absence of evidence. The client can be giving undeservedly good ratings, undeservedly bad ratings, or deserved ratings; the provider may be giving a good, bad average, or unstable service; there may be valid evidence provided by the client, or invalid evidence, or none provided. There are various permutations of these three variables possible.

The rest of the paper is organized as follows. Section 2 looks at related literature, while Section 3 describes the design of the proposed trust-aggregator service. In Section 4, evaluation experiments of the proposed system are presented.

2 Related Work

Trust as a computational concept was presented by Marsh in the context of multi-agent systems modeling human notions of trust [8]. Since then, the state-of-the-art in web services trust is summarized here. Resnick and Zeckhauser [5] looked at a case study of the eBay reputation system, which is among the earliest web reputation systems. The study performed analysis of historical eBay ratings of buyers and sellers, wherein trust and reputation were seen to be counterparts. Because eBay is a virtual online marketplace, trust is very important to facilitate transactions and sales. Trust in a seller is correlated to their reputation rankings, and distinctions between trustworthy and non-trustworthy sellers is based on their reputation, which in turn was built upon ratings given by clients. Since then, many systems have used reputation as a means of quantifying trustworthiness.

Jindal and Liu [7] looked into trustworthiness of online opinions. They identified the issue of review spam, which arises because anyone online can post comments and reviews about products and services. While these reviews are actually meant to help potential customers, spam reviews/opinions are a common occurrence whereby users or automated systems post many misleading or biased reviews. They identified three categories of spam reviews, Type 1 (untruthful opinions), Type 2 (reviews on brands only), and Type 3 (non-reviews). Jindal and Liu concluded that Type 1 reviews are difficult to identify. They also looked into the effects of duplicate and outlier reviews on the overall product perception.

Next, Kovač and Denis set up a theoretical framework to model trust in service-oriented environments, which supersede web services [9]. Architecturally, they propose incorporating a trust engine external to the service broker. They also provide formalizations on computing trust as social interactions among agents. Trust is modeled in terms of trust relations between agents, where agents provide a trust degree as an opinion about the relations, where the opinion is based on a set of trust values within the trust domain.

This leads to a trust matrix of all interactions between agents on which trust operations can be performed. The two fundamental operators identified are *Insert* and *Query*. We make use of an implementation of the two operators for our trust-aggregator service. A similar approach was proposed by Lin et al. [10] in the context of supply-chain performance. In order to facilitate better selection of suppliers, Lin et al. demonstrated a trust-measuring simulation. An important contribution of their work was the formulation of a research framework for mathematically defining ability, benevolence, and integrity. Our proposed approach also looks at various aspects of trustworthiness related to clients and providers, in terms of algorithms that can better predict trustworthiness and incorporate evidence.

Malik and Bouguettaya presented the RATEWeb framework in 2009 with the promise of creating trust within service-oriented systems [11]. They approach trust via client ratings, which are aggregated to determine a provider's reputation, and in turn, trust. In their model, clients and providers interact in a peer-to-peer manner, without the need for any central trust verification authority. In our approach, we separate functional concerns of clients and providers from verification of trust. Instead of having client/provider track trustworthiness, our proposed approach delegates this responsibility to an independent third-party service, and the client only needs to provide rating-based feedback, which leads to a more practical and less tightly coupled architecture. They incorporate client/rater credibility into the calculation of reputation. Our approach extends the notion of client weight by also incorporating karma, i.e. penalty or reward based on the diversity of client ratings. They use *k*-means clustering to compute the majority opinion among clients. We extend the approach by incorporating the number of ratings the provider receives. Consequently, our approach considers the quantity, quality and diversity of ratings. It is also noteworthy that Malik and Bouguettaya use the notion of communities for grouping ratings, and identification of providers with communities is compulsory. Our approach is less stringent, and our algorithm can be extended to query the service repository for finding categories that would assist in ranking providers within categories, or without. Ultimately, our approach provides absolute ratings in terms of aggregated ratings, and in addition gives relative ratings via our provider ranking algorithm.

More recently, Aljazzaf [3] complemented and surveyed and synthesized various previous studies in the literature on trust, services selection and discovery, in her work on trust-based service selection. An important contribution of the study was the distinction between security, privacy and trust, which are sometimes used interchangeably in the literature. For instance, in the WS-Security and WS-Trust standards, Certification Authorities (CA) can provide identity trust, but service requesters might also be interested in metrics for provision trust. WS-Trust can verify that the service provider is who they say they are, but cannot tell whether this service provider is good or bad. In order to handle various shortcomings with existing specifications and frameworks, Aljazzaf proposed the Trust Mediator trust framework, added to the SOA broker as a service. The Trust Mediator provides a ratings repository that stores QoS trust metrics.

We extend the notion of ratings to evidence-based ratings, and also look into reputation computation via clustering to provide a representative rating. Furthermore, part of the Trust Mediator's job is to monitor QoS values claimed by the provider. In contrast, our proposal puts the onus on the provider to satisfy the client requirements instead of claims of trustworthiness. Also, we separate the functional requirements of the service broker to discover web services from our trust-aggregator service. The bulk of the work of the trust-aggregator service is in reasoning about the trust ratings provided by clients and finding the best representation of the ratings to assist in trust decisions.

Finally, it is worth noting that, in the most recent SOA implementations, there is tight coupling of low-level security mechanisms such as the Public Key Infrastructure (PKI) systems with business functions, and SOA systems are usually secured at end-points, leading to hard-coded security functions [12, 13]. To address this issue, various studies have suggested extending the SOA architecture to incorporate an independent trust web service, which is the approach we adopt in our work also.

3 The Trust Service

We present the detailed design and inner workings of the trust-aggregator service. Figure 1 shows the service architecture and interactions needed for rating submissions.

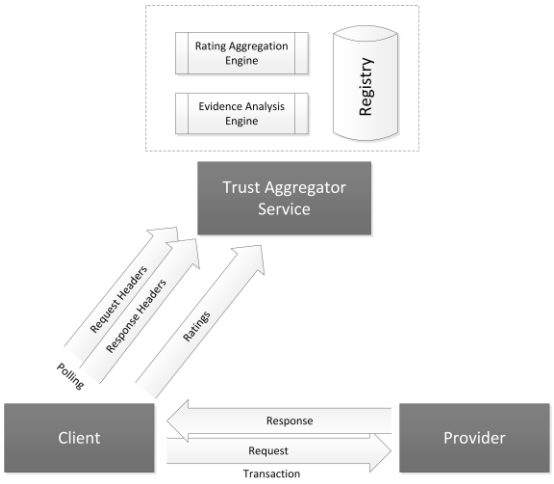

Figure 1. High-level Architecture and Interactions of Trust-Aggregator Service

In the architecture, clients send ratings to the trust-aggregator service. Evidence is polled and sent as headers when the client engages a provider. Related evidence and ratings are analyzed together by using heuristics, leading to an assessment of Weight of Evidence (WoE). Both client and provider can query existing ratings; the clients are likely to be interested in adopting a service where the providers are interested in their own trust reputation.

3.1 Rating Trust Metrics

A trust metric is a measurable quantification of how much one entity trusts another [14]. We focus on the following four trust metrics as representatives of subjective and objective QoS ratings. It is important to distinguish that QoS metrics give an empirical measurement of quality, while trust metrics give a valuation of the truthfulness of these QoS values.

- Correctness/accuracy (*c*): How accurate were the results returned?
- Availability (*a*): How responsive was the server when requested?
- Timeliness (*t*): How quickly were the results delivered?
- Satisfaction (*s*): How satisfied was the client overall?

The first three ratings, *c*, *a*, *t*, are objective performance metrics and we use a five-point Likert scale [1 … 5] as the rating mechanism. The client's rating *R* for a provider can be represented as a tuple ([*c*, *ec*], [*a*, *ea*], [*t*, *et*]), where *e** represents the evidence supporting each quality rating. We also include an additional subjective rating, namely satisfaction, *s*, as a proxy for the utility that the client perceives through the use of the service, likely dependent on the other three performance metrics. Clients may query the satisfaction rating for a provider by supplying weights per performance rating, or looking at various scenario profiles, in similar fashion to the hypothetical equivalents-inequivalents method suggested by [15]. Generally, we are using the two-operator *Insert* and *Query* approach by Kovač and Denis, ratings are submitted (i.e. insertion), and ratings are retrieved (i.e. querying) [9].

3.2 Interaction Sequence Diagram

To better illustrate the overall scenario, Figure 2 depicts the sequence of operations. As part of the agreement for subscribing to the trust-aggregator service, when a client invokes the provider's service, the request and response headers are forwarded to the trust-aggregator service via a polling mechanism. Polling essentially takes the headers from the client-provider transaction and sends them to the trust-aggregator service. It keeps these headers in its repository to be used as evidence in support of the aggregate ratings.

Clearly not all headers have to be sent to the trust-aggregator service as this would fundamentally undermine the scalability of the system. A combination of random sampling of which headers to store and forgetting old headers could be used to manage the overall number of headers stored as evidence by the aggregator.

3.3 Evidence

The trust-aggregator service receives request and response headers as trust-related evidence, submitted by the client along with ratings. These request and response headers are the result of the interactions between the client and provider. Because our aim is to develop a loosely coupled solution, the trust-aggregator service is not an intermediary between the client and service provider.

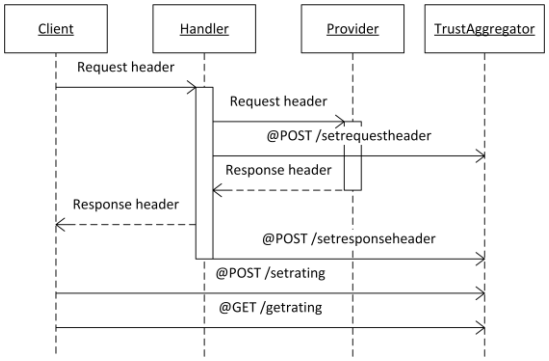

Figure 2. Trust-Aggregator Service Sequence

Consequently, the headers must be relayed to the trust-aggregator service, after the client-provider transaction has been completed. To that end, we propose a polling API that must be implemented by the clients who wish to participate to the trust-aggregator service to allow it to collect all request and response headers of the client. The implementation of the polling mechanism involves (a) the collection of request/response headers and (b) operations to selectively submit these headers to the trust-aggregator service upon request. The first functionality is likely to rely on message handlers, which typically sit between the port where the middleware listens and the recipient class. Handlers can inspect, process and record messages between the client and provider, and map them in pairs so that they can be used as evidence. Later on, when the client is submitting a rating for a provider, the relevant request and response headers related to the client and provider can be selected by the client from the trust-aggregator service's repository.

3.3.1 Server- and client-side polling

There are generally two places where message handlers can be placed: (a) provider-side, (b) client-side.

In the first scenario, the provider would have to use our polling API so that any incoming requests and outgoing responses can be monitored and the relevant headers extracted. In the second scenario, the client would need to use our polling API so that all requests to the provider and responses from the provider are routed through our message handler. Our proposal is to use a client-side polling mechanism to enhance the chances of adoption of the trust-aggregator service because providers may not be willing to have monitors installed on their servers.

In the context of WS-* services, a server-side polling mechanism can be implemented as a handler in Apache Axis2, for instance, or other similar Simple Object Access Protocol (SOAP) API for clients [16]. Programming models that provide application handlers enable the client to manipulate an inbound or outbound message. Handlers such as those provided by Apache Axis2 can be added to the runtime for performing additional tasks on request and response messages, such as logging. The handler can intercept messages and perform automated tasks, and it can also forward the response and request messages onward (i.e. polling) to the trust-aggregator service.

There is no similar standard mechanism for RESTful services, but a possible solution would be to parse server logs and extract headers; alternatively clients of RESTful service could implement standard APIs to submit this meta-level information about their interactions with the services. For demonstration purposes, in our simulation, web-browser extensions are used to implement client-side polling mechanism. Browser extensions expand the functionality of a web browser, and can monitor inbound and outbound traffic, as well as perform seamless actions on requests and responses being exchanged. Extensions can act as an intermediary between client and provider.

3.3.2 Header properties

The trust-aggregator service presents to the client the set of relevant polled headers, and the client can select the appropriate one as potential evidence of the transaction for which the rating is being given. Once the client chooses their potential evidence, the trust-aggregator service can then extract relevant properties from the headers as evidence. The following properties are queried as potential evidence [17, 18].

- Date: The timestamp for when the message was sent
- Status: The response status
- Warning: Potential problems with message body

These properties are used to determine, using heuristics, if the ratings match the information about the interaction inferred from the polled headers, as discussed in Section 3.6.3. The lag time between a request and response is also noted.

3.4 Ratings Repository

The trust-aggregator service stores all client ratings in a repository, along with identifiers to service providers and clients. The repository stores tuples (*cid*, *pid*, *R*) consisting of the client identifier *cid*, the provider identifier *pid*, and the rating *R*. The polled headers are also stored in the repository. These tuples are matched against the polled headers during the evidence analysis step, as discussed in Section 3.6. The trust-aggregator service extracts the relevant properties from the polled headers and saves these to the repository. The repository model is shown in Figure 3.

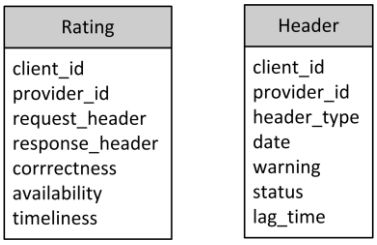

Figure 3. Trust-Aggregator Service Repository Data Model

Notably missing from the repository are satisfaction ratings. It should be clarified that satisfaction is not implemented as a rating provided by the client, but rather as a utility function based on other ratings, so there is no need to store it in the repository.

3.5 Service Interface

The trust-aggregator service exposes the following methods to the client. We chose to describe it as a RESTful service for reasons of simplicity in the development of the simulation, but it could equally be developed in the WS* style.

- `setRequestHeader(cid,pid,reqhead)`
- `setResponseHeader(cid,pid,resphead)`
- `setRating(cid,pid,reqhead,resphead,c,a,t)`
- `getRating(pid)`
- `getRating()`

The *getRating* method without parameter arguments returns the ratings of all providers with entries the repository. Retrieving side-by-side multiple provider ratings helps in comparative analysis. The methods map to the following POST methods respectively.

- `@POST /setrequestheader?cid=&pid=&reqhead=`
- `@POST /setresponseheader?cid=&pid=&resphead=`
- `@POST /setrating?cid=&pid=&reqhead=&resphead=&c=&a=&t=`
- `@GET /getrating?pid=`
- `@GET /getrating`

The aggregation algorithms are invoked in sequence via the *getRating*, as discussed in Section 3.7. The evidence analysis algorithms are invoked with *setRating*, as outlined in the following Section 3.6. The evidence analysis steps are presented first.

3.6 Evidence Analysis

The evidence-analysis process involves three steps: (a) header polling, (b) property mapping, and (c) heuristic rating validation. Polled request and response headers are parsed to extract properties that can be useful as evidence for the ratings, as discussed in Section 3.6.1. Once header properties have been extracted, they can be presented to the client as possible evidence. Also, ratings can be validated via heuristics, as discussed in Section 3.6.3, while evidence as properties can be aggregated to create provider profiles. Moreover, evidence analysis and inference can lead to data mining scenarios.

3.6.1 Header Polling

On SOAP-based clients, handlers can perform the required polling mechanism to automatically forward the headers a client gets from any provider they interact with. These are all saved in the trust-aggregator service's repository to be looked up later as possible evidence. For the purpose of demonstrating our RESTful-based service, we use a web browser and an extension. The polling mechanism would require installation of the extension that can monitor and forward headers.

3.6.2 Property Mapping

The property-mapping step transforms request-response headers to quality-based trust metrics. The mapping of properties to ratings is given in Figure 4. Correctness is related to warning and status codes because certain codes exclude the possibility that the request was completed successfully. For instance, a status code between 400 and 600 implies that an internal client or server error was encountered while attempting to service the request, thus the response could not have been correct [17, 18]. Availability is related to response status codes, since certain status codes above 500 can give an indication of whether the server was up or not [17, 18]. Also, timeliness is related to the time elapsed between the request being sent and the response being received. Also, the rationale for these mappings is further explained using examples in the proceeding Section 3.6.3.

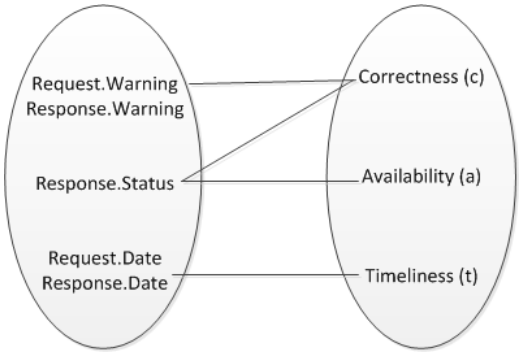

Figure 4. Header Properties as Rating Evidence

3.6.3 Rating Validation Heuristics

The header properties can be analyzed against the client rating via heuristics to validate the rating. For instance, a low rating on *a* (availability) might imply a response *Status* code other than 200.

Similarly, a high rating for *t* (timeliness) implies that the time elapsed between the request and response is short, which can be inferred by inspecting the *Date* properties of request and response headers. Based on these heuristics, a Weight of Evidence, *WoE* = [0.1, 1] can be incorporated with the rating, i.e. $c = woe \times c$, $a = woe \times a$, $t = woe \times t$, when the rating is deemed trustworthy, *WoE* = 1, and no dilution of ratings occurs. If the rating is suspicious based on the evidence, then it is diluted and its value reduced.

Simply put, dilution implies that the *WoE* < 1, and the value for *WoE* is adjusted depending on the scenario. As an example, suppose the response time is greater than 1 day and the rating is greater than 3. A simple heuristic would be to set *WoE* = 0.5 for this scenario because the response time is seemingly slow, but the rating is relatively high.

3.7 Ratings Congregation

The ratings-congregation process follows the evidence analysis and is responsible for the ratings composition through the following steps: (a) rating normalization; (b) client ranking (karma); (c) rating factoring; and (d) rating aggregation. After a */getrating* query is issued, all ratings for the specified provider are retrieved and a rating selection algorithm is applied to validate and prune client ratings.

3.7.1 Rating Normalization

This algorithm is invoked whenever ratings are requested. First, two weights are incorporated in the ratings: temporal damping factor and Weight of Evidence (WoE). Temporal damping can also be referred to as the decay of the rating variable, and naturally occurs because older ratings should have lower weight in the overall assessment about a provider's trustworthiness. Services (their implementation details and deployment configurations) evolve and older ratings are obviously unaware if these changes.

We use a threshold, τ that determines 'how old is old'. If the difference in the present date and the rating's date exceeds τ then it is dampened by a numerical factor, *tdf* = [0.1, 1]. A simple heuristic could be setting τ to 3 months and any rating that is older than that would get scaled down by a factor of 0.1 for each month-age. Similarly, the evidence heuristics can provide a value for *WoE* as explained previously.

3.7.2 Client Ranking (Karma)

For each rating, a "karma" is assigned to the client. Karma, in this context, is conceived as a penalty or a reward given to the client, based on their 'deeds', i.e., their ratings to providers. Karma is implemented as a multiplier weight, w_{cid}, based on the client, i.e. $c = c \times w_{cid}$, $a = a \times w_{cid}$, $t = t \times w_{cid}$.

This weight serves as a meta-trust metric. Clients who give more ratings have a higher weight, because more is known about them than about clients that give very few ratings.

We also factor in the distribution of the ratings by having standard deviation of the client's rating. Suppose for client *cid*, the total number of ratings is n_{cid}, while the total number of ratings for all clients in the trust-aggregator service's repository is T_c. Also, let std_{cid} be the standard deviation of the ratings by client *cid*. Then

$$w_{cid} = [(\alpha \times (n_{cid}/T_c)) + (\beta \times std_{cid})] / (\alpha + (maxsd \times \beta))$$

where $maxsd = 2$, and α, β determine the relative importance of the number of client ratings or the distribution of ratings. The intuition behind this formula is that a client who is more active in giving ratings is more known, hitherto more trustworthy, than another client who is less active. Also, a client who consistently gives similar ratings to different providers could be seen as less trustworthy than a client who gives more diversified ratings.

Consequently, the activity of the client is rewarded or punished as karma. The next step is to apply the rating aggregation algorithm. It should be noted that *maxsd* needs to be set to 2 to normalize the value of w_{cid} so that its range is always [0, 1], given that $max(n_{cid}/T_c) = 1$ and $max(std_{cid}) = 2$ [19].

3.7.3 Rating Factoring

When a client has provided multiple ratings for the same provider, these ratings are averaged, so that only one representative rating per client is used in the provider profiling. Also, the client-provided evidence is checked: in the case that a client has not provided evidence for a rating, that rating is discarded.

3.7.4 Rating Aggregation

The purpose of the algorithm is to calculate an overall satisfaction rating, based on the *c*, *a*, and *t* ratings. The rating-aggregation algorithm works as follows. First, a representative value for each of the trust metrics is computed as the median of values from the previous step. This steps eliminates outlier ratings that may be too high or too low, and therefore not representative.

Next, the satisfaction-rating profiler algorithm is applied, to compute the satisfaction rating as a utility function. Different weights, w^*_{si} are assigned to *c*, *a*, *t* and the results averaged, i.e. $s = (wc_{s1} \times c + wa_{s1} \times a + wt_{s1} \times t)/(wc_{s1} + wa_{s1} + wt_{s1})$. For each variation of the weights, a different profile, *si* of satisfaction is created. We explore the following multipliers for *s* profiles, listed in Table 1. The actual selection of which profile is more suitable is subjective and depends on the preference. For instance, if a client thinks availability is very important, then *S5*, *S8* and *S11* would be pertinent.

Profile	c	a	t
S1	1.00	0.50	1.00
S2	0.50	1.00	1.00
S3	1.00	1.00	0.50
S4	0.50	0.50	1.00
S5	0.50	1.00	0.50
S6	1.00	0.50	0.50
S7	0.25	0.50	1.00
S8	0.25	1.00	0.50
S9	0.50	0.25	1.00
S10	1.00	0.25	0.50
S11	0.50	1.00	0.25
S12	1.00	0.50	0.25
S13	1.00	1.00	1.00

Table 1: Multipliers for Satisfaction (s) Profiles

As noted earlier, the concept of the profiler is based on the work by Srivastava and Sorenson [15], the weights shown in Table 1 are configured for diversity in options. The different weights allow different aspects of the ratings to be highlighted, such as correctness, availability, or timeliness.

3.8 Additional Potential Features

The trust-aggregator service could be enhanced with a number of additional features.

3.8.1 Provider Profiles

The extracted header properties can also be aggregated to create profiles of providers. For instance, a provider's average response time, typical status response, and common response warnings can be determined from the header submissions.

3.8.2 Inference

The trust-aggregator service could also reason about the meaning of ratings via data mining methods, such as association rule mining. For instance, an empirical understanding of slowness can be determined from looking at low *t* ratings and comparing the response times involved. Similarly, low *c* ratings can be associated with some header or response body properties, to help validate ratings.

3.8.3 Provider Rankings

Clients could also comparatively review ratings for more than one provider. We assign a weight, w_{pid}, based on how many ratings a provider gets, n_{pid}, relative to the total number of ratings given to all providers, T_p. Providers with more ratings have a higher quantitative weight. Also, we consider only one rating per client, and in the situation there are more than one, they are counted once. To rank the ratings qualitatively, we also incorporate the aggregated ratings per provider from the previous rating aggregation algorithm and rank providers by normalization. The normalization ranking algorithm works as follows. The aggregated ratings for each metric are summed to give sum_{pid}. Each provider is then given a normalized weight, nw_{pid}, using the grand total of all provider ratings from the previous aggregation algorithm, S_p, as a fraction $nw_{pid} = sum_{pid}/S_p$. Then, $w_{pid} = ([\alpha \times (n_{pid}/T_p)] + [\beta \times nw_{pid}])/(\alpha + \beta)$, where α, β adjust the importance we can place on the quantity vs. quality of ratings.

4 Evaluation

We implemented a prototype of the proposed trust-aggregator service and tested it using a simulation engine. The trust-aggregator service was implemented using JAX-WS running in Apache Tomcat web server, while its repository was implemented using SQLite. We also built a simulation engine in Visual Basic.NET that can generate text of HTTP headers as well as simulate large amounts of ratings and associate them with appropriate headers.

4.1 Experimental Design

Our experimental design involved the simulation of a variety of scenarios that explored different types of (a) client trustworthiness, (b) provider quality, and (c) the presence/absence of evidence. The client could be giving undeservedly good ratings, undeservedly bad ratings, or deserved ratings. The provider may have given a good, bad average, or unstable service. Also, there may be valid evidence provided by the client, or invalid evidence, or none provided. We explored various possible combinations of these three types of parameters.

In our experiments, we defined predetermined characteristics of the providers and the clients, and evaluated the trust-aggregation algorithm in terms of the validity of the trust metrics it provides. Essentially, the question we set out to address was whether or not the trust-aggregation service would be able to provide useful trust metrics which clients could rely on. We also examined how the algorithm reacts to outlier ratings, both invalid and valid ones. As an example, suppose a bogus client gave consistently good ratings to a provider who is bad, the question is to determine whether other clients could be made aware of this in terms of this provider's absolute and relative rating aggregations. Finally, we investigated the response of the trust-aggregator service to Sybil attacks, which involve clients or spambots providing misleading ratings [6].

4.2 Simulation Engine

The simulation engine allows configuration of the nature of HTTP request and response headers needed. The engine can generate these headers and save them in the format that the trust-aggregator service needs. A screenshot of the engine is shown in Figure 5.

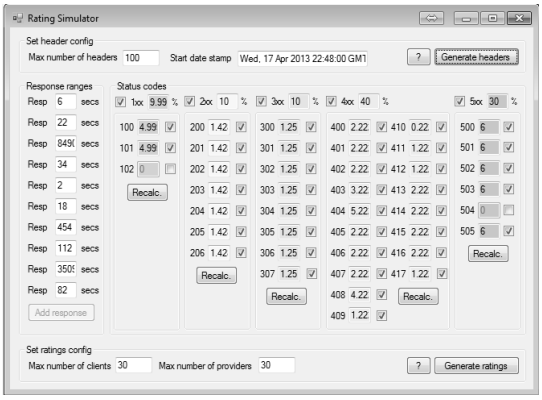

Figure 5. Simulation Engine

The engine interface allows the user to input into the array of textboxes the frequency of each status code in the final output of headers, as percentages of the total occurrences. In addition, the user can configure different ranges of response times, total number of headers to generate, total number of unique clients and providers.

The number of response codes in each category, e.g. 2xx, is determined based on the total number of headers needed and the percentage of these codes' occurrence, as input by the user. For each response code selected, a random number of request warning codes are generated, along with a random response time. The type of warning codes and duration of response times are based on the values input by the user. Only those warning codes and response times that were selected are used.

The engine has two functionalities. First, it can generate the headers to simulate the polling mechanism. The generated headers have the option of varying timestamps to simulate temporal damping, as well as control over the distribution of HTTP *Status* codes that are generated. Based on the number of headers that are required, a random *Warning* status code and a random *Date* are assigned based on the configuration of the response ranges option. All the selections follow a Normalized distribution with a pre-configured mean and standard deviation.

The normal distribution provides a better simulation of real-world scenarios where the overall ecosystem of web services has an average positive rating. It should be noted that in a real-world system, these codes and values would be extracted from raw HTTP headers via parsing in the evidence analysis stage, and then saved to the repository. Second, the engine can simulate ratings by multiple clients to various providers. Ratings for *c*, *a*, *t* are randomly generated but again follow a Normal distribution, as shown using a sample simulation in Figure 6.

The simulator also generates multiple ratings by the same client for each provider. A sample distribution for providers is shown in Figure 7, where providers are given unique ratings per client as well as multiple ratings from the same clients. The degree to which a client is more likely to give multiple ratings to the same provider is based on a randomized provider bias value.

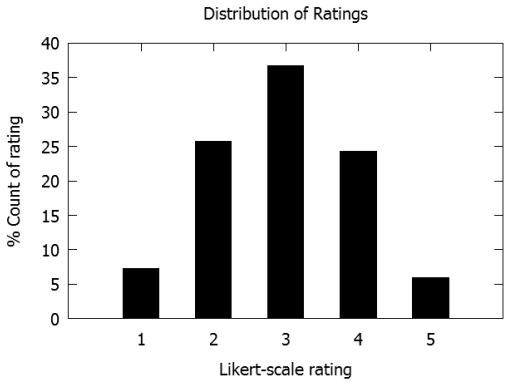

Figure 6. Ratings Distribution

For each rating, a header is assigned as evidence. Furthermore, in order to simulate Sybil-like attacks, each client is assigned a randomized threat-level value based on which the client's ratings are adjusted to either match or mismatch the evidence. For instance, a client considered as a high-threat level giving a good rating would imply assigning weak evidence to these ratings. This scenario simulates the phenomenon where a malicious client assigns weak ratings even though the actual experience with the service is satisfactory. The trust-aggregator requires clients to substantiate their ratings with evidence, which makes this service more robust to unfair, malicious ratings.

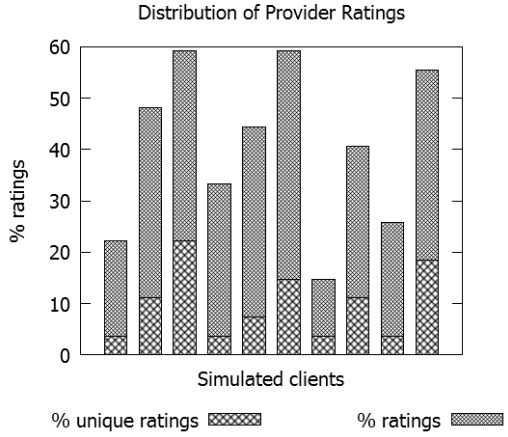

Figure 7. Provider Ratings Distribution

A sample distribution of threat levels is given in Figure 8 for a sub-section of 15 clients, with four classifications of threat, depicted by the horizontal, from top to bottom: severe, high, medium, low.

It should be noted that these levels are classifications of data generated by the simulation engine, and not implicitly known by the aggregation algorithms.

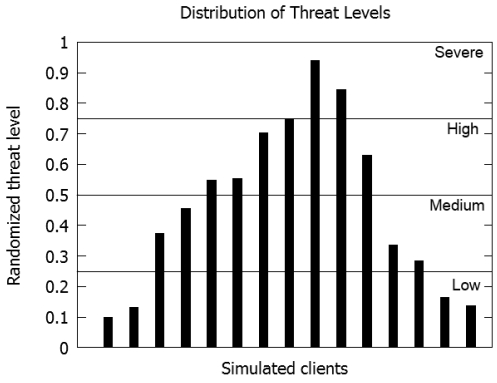

Figure 8. Threat Level Distribution

Once the simulation engine generates the required data, it can be plugged into the trust-aggregator service to query rating aggregations. The simulation engine essentially replaces the *setRequestHeader*, *setResponseHeader*, *setRating* methods so that large amounts of data can be fed into the trust-aggregator service. Ultimately, our experimental analysis involves testing the *getRating* method, which is the core of the trust-aggregator service. It should be noted that the algorithms for evidence analysis can be run when a rating is first submitted, or when it is being queried. For demonstration purposes, we compute *WoE* on the fly when ratings are queried.

4.3 Header Polling Browser Extension

For the polling mechanism demonstration, we implemented a proof-of-concept Google Chrome extension that allows the relaying of headers to the trust-aggregation service. The Chrome Extensions API exposes various aspects of the browser via JavaScript and AJAX. Generally, RESTful services are invoked programmatically by applications that form the HTTP requests and parse the XML/JSON responses. The same action is possible with a browser because JavaScript-based technologies make invocation of REST services from browsers possible.

The headers for the currently loaded page are retrieved and forwarded to the trust-aggregator web service via XMLHttpRequest.

4.4 Findings

We look at the response of the trust-aggregator web service to Sybil-like attacks, which generally involve clients diluting the ratings of the community by providing bogus ratings. In our simulations, various client were given different degrees of threat level, which equate to the likelihood of the clients carrying out a Sybil-like attack. For instance, a client can be giving undeservedly good ratings, undeservedly bad ratings, or deserved ratings; the provider may be giving a good, bad average, or unstable service; there may be valid evidence provided by the client, or invalid evidence, or none provided. We configure three scenarios labeled as Scenario A, Scenario B, and Scenario C. Scenarios B and C approximately correspond to the Type 1 opinion spam category as defined by Jindal and Liu [7].

4.4.1 Scenario A: Temporal damping

In this scenario, we examine the effect of temporal damping. The trust-aggregator service should adjust ratings with older timestamps, in comparison to newer ratings. Figure 9 shows aggregate ratings for one of the metrics, with varying timestamps, for a given providers. After a while, the overall rating for the provider starts falling from just above 2 eventually to about 1.5, even though no new ratings are being given, as can be seen from the number of clients giving ratings.

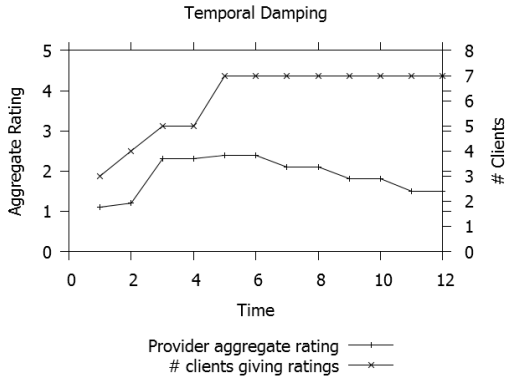

Figure 9. Temporal Damping

4.4.2 Scenario B: Single-provider attack

In this scenario, we examine the behavior of the trust-aggregation service when a client targets a single provider and gives it bogus ratings. These ratings can be overly positive or negative. This scenario, which we characterize as a single provider attack, implies that a client is purposefully trying to boost or bring down a provider's reputation. Figure 10 shows that the trust-aggregator service is able to mitigate against such attacks. In the area of interest between 125 and 220, the number of ratings is increasing but the aggregate rating of the provider stays stable. The increase in ratings is caused by the same client, and it is being balanced out with the factoring algorithm.

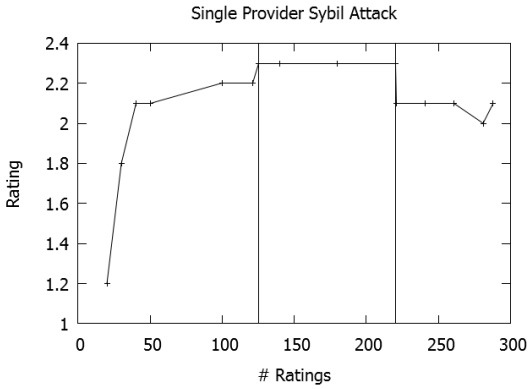

Figure 10. Single Provider Attack

4.4.3 Scenario C: Flooding attack

In this scenario, we examine the case of a client giving the same ratings to multiple providers. Again, these could be negative or positive ratings. Figure 11 shows the result of a client flooding three providers with automated ratings. In the marked area between 140 and 220, the numbers of ratings are increasing because of the Sybil attack of a client. However, the corresponding aggregated ratings per provider is not changing. This is because the client ranking algorithm step takes into account the standard deviation of a client's ratings. With the proper configuration of β, the algorithm can deal with Sybil flooding.

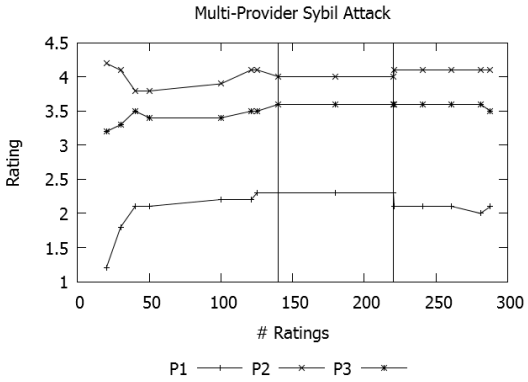

Figure 11. Multi-Provider Flooding Attack

4.5 Discussion

The simulation results show that the proposed trust aggregation can handle Sybil-like attacks, which includes client bias towards a single provider, or flooding all providers with bogus data. In addition to the anti-attack mechanisms, the trust-aggregator service can provide a good mechanism for measuring and aggregating trust metrics to present a client with information to make an informed choice. However, it is noted that the simulations require tweaking and proper configuration of various control variables within the algorithms, such as α, β. Nevertheless, we believe that the strength of the trust-aggregator service is in the simplicity of the evidence polling and RESTful ratings submission protocols.

Providers may have some concerns in adopting our proposed service because of possible performance impacts. We note that with server-side polling, the client would not experience any lags because of the trust-aggregator service, because polling would be happening asynchronously, independent of the services by the provider. With client-side polling, there might be performance impacts for the client, depending on their system, network, or browser specifications.

5 Conclusions and Future Work

Related literature suggests that the notion of trust is not clearly understood within web services. Although there have been efforts at incorporating trust in the context of web-services selection, there is no agreement on how exactly trust metrics should be established and used.

Our contributions to this research area include extending trust to evidence-based trust, as well as providing absolute and relative trust metrics of providers based on meta-trust characteristics of clients. Furthermore, we propose evidence analysis as a way to make inferences about ratings and provider services. We also conform to the self-reinforcing notion of trust by incorporating karma that adjusts client credibility. We also propose the notion of Weight of Evidence (WoE) from heuristic analysis of evidence. We created a loosely coupled trust-aggregator service prototype that enables measurement of provision trust metrics. Our experimental analysis used various permutations of ratings from clients and providers of varied quality and trustworthiness.

An important lesson learnt from this study is that while there are many good ideas in the literature about measuring web services trust, actual adoption of these proposals is not prevalent. Our empirical results show that the proposed trust-aggregator service framework is feasible and effective in measuring trust metrics. Our proposed evidence-collection polling strategies are also workable. Our contributions included extending trust to evidence-based trust. Moreover, we incorporate the state-of-the-art concepts from literature within this area of research.

For future work, a comparative ranking of providers can be retrieved with the inclusion of categories, so that like-providers are compared using the provider ranking algorithm proposed here. To further improve rankings, we plan to extend the trust-aggregator service to incorporate provider categories from the service broker. Moreover, using the proposed provider profiles feature, we plan to incorporate inference and trust bootstrapping via data mining to reason about the evidence and ratings and make inferences about the computational meaning of concepts such as slow service, bad service, and dissatisfaction.

Acknowledgements

The author would like to thank Dr. Eleni Stroulia and Dr. Osmar R. Zaïane for their insights and critique on the presentation and formulation of this research work. Dr. Stroulia is the Industrial Research Chair on Service Systems Management for the Natural Sciences and Engineering Research Council (NSERC)/Informatics Circle of Research Excellence (iCORE), and professor of Computing Science at University of Alberta. Dr. Zaïane is the Scientific Director of the Alberta Innovates Center for Machine Learning (AICML), and professor of Computing Science at University of Alberta. The author is very grateful to the Alberta Innovates Center for Machine Learning (AICML) for funding this research.

About the Author

Hamman Samuel completed his master's degree in computing science at the University of Alberta, Canada, where he is currently a doctoral student in computing science. He received his bachelor's degree in computer science (magna cum laude) from the American University of Nigeria. Hamman is presently serving as webmaster of the ACM SIGWEB special interest group.

References

[1] J. Child. Trust - The Fundamental Bond in Global Collaboration. Organizational Dynamics, 29(4):274288, 2001.

[2] P. Hallam-Baker, V. M. Hondo, H. Lockhart, B. R. Martherus, O. H. Maruyama, A. Nadalin, I. B. M. Nataraj Nagaratnam, D. Platt, and D. Waite. Web Services Trust Language (WS-Trust). 2004.

[3] Z. M. Aljazzaf, Trust-Based Service Selection. The University of Western Ontario, 2011.

[4] A. Abdul-Rahman. The PGP Trust Model. In *EDI-Forum: the Journal of Electronic Commerce*, 1997, vol. 10, pp. 27–31.

[5] P. Resnick and R. Zeckhauser. Trust Among Strangers in Internet Transactions: Empirical Analysis of eBay's Reputation System. 2002.

[6] Reputation System. Wikipedia. 01-Mar-2013.

[7] Nitin Jindal and Bing Liu. Opinion Spam and Analysis. In Proceedings of the International Conference on Web Search and Web Data Mining, page 219-230, 2008.

[8] S. P. Marsh. Formalising Trust as a Computational Concept. University of Alberta, 1994.

[9] D. Kovač and D. Trček, Qualitative Trust Modeling in SOA. Journal of Systems Architecture, vol. 55, no. 4, pp. 255–263, 2009.

[10] F.-R. Lin, Y.-W. Sung, and Y.-P. Lo. Effects of Trust Mechanisms on Supply-Chain Performance: A Multi-Agent Simulation Study. International Journal of Electronic Commerce, vol. 9, no. 4, pp. 9–112, 2003.

[11] Z. Malik and A. Bouguettaya. RATEWeb: Reputation Assessment for Trust Establishment among Web services. *The VLDB Journal*, vol. 18, no. 4, pp. 885–911, 2009.

[12] L. Boursas, M. Bourimi, W. Hommel, and D. Kesdogan. Enhancing Trust in SOA Based Collaborative Environments. *Systems and Virtualization Management Standards and the Cloud*, pp. 94–102, 2010.

[13] F. Moyano, C. Fernandez-Gago, and J. Lopez, Service-Oriented Trust and Reputation Architecture. In *Doctoral Symposium of the International Symposium on Engineering Secure Software and Systems*, 2012, pp. 41–46.

[14] Trust Metric. Wikipedia. 09-Jan-2013.

[15] A. Srivastava and P. G. Sorenson. Service Selection Based on Customer Rating of Quality of Service Attributes. 2010, pp. 1–8.

[16] Apache Axis2 - JAX-WS Guide. [Online]. Available: http://axis.apache.org/axis2/java/core/docs/jaxws-guide.html#Handlers. [Accessed: 21-Apr-2013].

[17] List of HTTP Header Fields. Wikipedia. 01-Apr-2013.

[18] The TCP/IP Guide - HTTP General Headers. [Online]. Available: http://www.tcpipguide.com/free/t_HTTPGeneralHeaders-3.htm. [Accessed: 21-Apr-2013].

[19] M. F. Al-Saleh and A. E. Yousif. Properties of the Standard Deviation that are Rarely Mentioned in Classrooms. *Austrian Journal of Statistics*, vol. 38, no. 3, pp. 193–202, 2009.