Trust and Security in Spatial Messaging: FoxyTag, the Speed Camera Case Study

Michel Deriaz and Jean-Marc Seigneur
University of Geneva, Switzerland
{firstname.lastname} [at] unige.ch

Abstract

Current speed cameras alerting systems heavily rely on humans to check the trustworthiness of information sent by their users. Hence, these systems are often either expensive or suffer from drawbacks, such as incomplete information, for example, concerning mobile speed cameras. We propose an application called FoxyTag to address most of the previous issues by using a computational trust engine instead of human checks. FoxyTag lets any driver equipped with a Java/GPS-enabled mobile phone post a virtual tag about a speed camera to notify other equipped drivers who can confirm or deny the (short-lived) presence of the (mobile) camera. The novel aspect of our trust engine is that it must be location and time aware to automatically compute the trustworthiness of the given tag. We have validated FoxyTag both in real-life settings and with a simulator for large-scale scenarios. The validation showed that our novel time-patterned trust metrics are appropriate.

1. Introduction

A few European countries have been multiplying the number of speed cameras in order to reduce the number of car accidents. To further assist the drivers, more and more companies sell information systems in order to warn the driver nearby a critical zone. Because these systems heavily rely on humans to check the trustworthiness of the information sent by their users, these systems are often either expensive or suffer from drawbacks. They have incomplete information, for example, they do not deal with mobile speed cameras.

We propose a new system called FoxyTag that allows any driver equipped with a Java/GPRS-enabled mobile phone with access to a GPS (for example, we currently use affordable external Bluetooth GPS modules) to easily signal a speed camera or to signal that a former one has been removed. This is done by posting to our central server via GPRS virtual spatial messaging tags at the critical points, so that other drivers can be alerted on time by querying this server by affordable GPRS connections.

The novelty of our system is that it does not require human checks to decide about the trustworthiness of the posted tags because our system uses a computational trust engine to automatically make this decision.

Due to the fast context changing aspect of this application domain – mobile speed cameras are short lived – previous work on computational trust was challenged and we had to engineer novel time-patterned trust metrics.

This paper is organized as follows. First, we survey the related work. In section 3, we present the requirements for an efficient speed camera tagging system. Section 4 describes FoxyTag an application whose aims is to fulfill the requirements of Section 3. Section 5 explains why and how our new trust metrics go beyond the state-of-the-art. Section 6 describes the simulator that we have created and used for large-scale testing. Section 7 gives the results of our validation of the FoxyTag prototype in real life settings with real Java/GPS/GPRS-enabled mobile phones and real information from related online services. Section 8 validates our work with simulation results and completes the validation in the field. Section 9 highlights future work and concludes.

2. Related Work

The design of FoxyTag is based on the use of spatial messaging (virtual tags) and the quality of the service relies on a trust engine. There are many approaches in defining trust and there are many spatial messaging systems. In this section, we first survey the related work on computational trust and then the related work on spatial messaging. Finally, we give a summary of other related speed camera alerting systems.

2.1. Computational Trust Survey

In the human world, trust exists between two interacting entities and is very useful when there is uncertainty in result of the interaction. The requested entity uses the level of trust in the requesting entity as a mean to cope with uncertainty, to engage in an action in spite of the risk of a harmful outcome. There are many definitions of the human notion trust in a wide range of domains, with different approaches and methodologies [27]: sociology, psychology, economics, pedagogy… These definitions may even change when the application

domain changes. However, McKnight and Chervany have convincingly argued that these divergent trust definitions can fit together [20].

Interactions with uncertain result between entities also happen in the online world. So, it would be useful to rely on trust in the online world as well, as in our speed camera collaborative alerting application.

A computational model of trust based on social research was first proposed by Marsh [13]. In social research, there are three main types of trust: interpersonal trust, based on past interactions with the trustee; dispositional trust, provided by the trustor's general disposition towards trust, independently of the trustee; and system trust, provided by external means such as insurance or laws [20]. Trust in a given situation is called the *trust context*. Each trust context is assigned an importance value in the range [0,1] and utility value in the range [-1,1]. Any trust value is in the range [-1,1). In addition, each virtual identity is assigned a general trust value, which is based on all the trust values with this virtual identity in all the trust contexts. Dispositional trust appears in the model as the basic trust value: it is the total trust values in all contexts in all virtual identities with whom the trustor has interacted so far. Risk is used in a threshold for trusting decision making.

2.1.1. Evidence-based Trust Value Computation. A computed trust value in an entity may be seen as the digital representation of the trustworthiness or level of trust in the entity under consideration. The trustcomp online community [21] defines *entiTrust*, that we consider in this paper as a trust value, as a non-enforceable estimate of the entity's future behavior in a given context based on past evidence. The EU-funded SECURE project [18] represents an example of a trust engine that uses evidence to compute trust values in entities and corresponds to evidence-based trust management systems. Evidence encompasses outcome observations, recommendations and reputation. Sabater and Sierra remark that "direct experiences and witness information are the 'traditional' information sources used by computational trust and reputation models" [23]. Depending on the application domain, a few types of evidence may be more weighted in the computation than other types. When recommendations are used, a social network can be reconstructed [24]. A *trust metric* [30, 23, 14] consists of the different computations and communications which are carried out by the trustor (and his/her network) to compute a trust value in the trustee.

Figure 1 gives an overview of a trust engine. The decision-making component can be called whenever a trusting decision has to be made. The Entity Recognition (ER) module [14] bridges the gap between identity management and reputation by recognizing the entities involved in the interactions with attack resilience and privacy protection considerations. The decision-making of the trust engine uses the trust module to dynamically assess the trustworthiness of the requesting entity and evaluates the risk involved in the interaction based on the available trust and risk evidence in the evidence store.

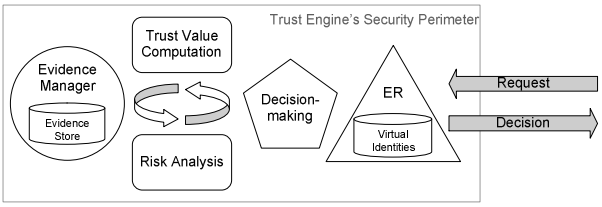

Figure 1. Overview of a Trust Engine

A common decision-making policy is to choose (or suggest to the user) the action that would maintain the appropriate cost/benefit. In the background, the evidence manager component is in charge of gathering evidence (for example, recommendations, comparisons between expected outcomes of the chosen actions and real outcomes…). This evidence is used to update risk and trust evidence. Thus, trust and risk follow a managed life-cycle.

2.1.2. Time in Computational Trust. However, Dimmock, who took care of the risk module in the SECURE project, concludes in his PhD thesis that more work with regard to the risk of the situation must be done and especially with regard to the time element of risk: "one area that the framework does not currently address in great detail is the notion of time" [25].

Concerning the trust context [27], most previous trust engines have focused on the domain of trustworthiness under consideration (for example, trustworthiness in writing good security books or trustworthiness in recommending good doctors) or the virtual identity of the requester/candidates. It seems that another type of trust from social research, that is, the "situational decision to trust […, which means that the trust engine's owner] has formed an intention to trust every time a particular situation arises", has been overlooked, even if it may be considered as imbricate with dispositional trust.

In our speed camera application domain, our trust engine needs to filter the candidate speed camera alert tags based on:

- the location, as explained in Section 4 our GPS-based measurement technique does not introduce a high level of uncertainty;
- the time of the camera installation and removal, which cannot be directly captured and is the reason to use a trust engine and collaborative recommendations to cope with the high level of uncertainty that it introduces.

Guha [24] argues to have built a generic open rating system based on a trust engine, which means that anybody is allowed to rate anything in the system,

including the ratings of contents. The contents may be considered as speed camera tags. However, even Guha argues that his work is limited with regard to time: "these [content/tag rating systems] are dynamic systems which can change quite rapidly. Understanding the time-dependent properties of such systems and exploiting these properties is another potentially useful line of inquiry" [24].

In the Semantic Web application domain, the TriQL.P trust engine [26] is used to decide how much information found on the Web should be trusted. The main types of evidence are the context, which includes who and when, and content. Although their policy language allows the programmers to specify time dependent policies, no trust metric including the notion of time is given. Marsh [13] underlined the role of time as being relevant to each of the variables used in his trust model but again no specific time-dependent metric was given. The SECURE trust engine's method to compute a trust value takes more than the identity context into account but no specific time-sensitive metric implementation is given.

Most of previous trust metrics consider from a time point of view that a trust value is updated only when a user manually resets [23, 30] a trust value in another entity or when there is the outcome of a previous interaction with this entity by means of direct observations or recommendations.

The few trust metrics that takes further time into consideration simply proposes that the trust values decay over time, even if there is no interaction. Mezzetti [28] assumes that trust values decay as time passes and his metric consists of decreasing the trust value by multiplying the trust value at time *t* by a factor between zero and one, which is the result of a transitive aging function taking the elapsed time since *t*. Similarly, in recent Bayesian-based trust metrics [31, 32], the trust values are aged and converge towards their bootstrapping value over time but still the choice of the aging factor is rather arbitrary. It may work in rather continuous application domains but mobile speed cameras are short-lived: they can be quickly installed and removed. When a speed camera is removed, the trustworthiness in the tag alert must drop promptly and trust metrics decreasing gracefully over time are not appropriate. It is the reason we have had to engineer and validate our novel time-sensitive trust metrics, as we report in this paper in Section 5, for the spatial messaging application domain.

2.2. Spatial Messaging Survey

Spatial messaging, also called digital graffiti, air graffiti, or splash messaging, allows a user to publish a geo-referenced note so that any other user that attends the same place can get the message. Different usage scenarios can be found in the following projects that focus on spatial messaging and in [1].

2.2.1. E-Graffiti. E-Graffiti [3] is a spatial messaging application that allows a user to read and post geo-localized notes. These notes can be either public or private, meaning that only the set of people defined by the author are able to read the note. E-Graffiti has been designed to study the social impacts on spatial messaging. 57 undergraduate students were given a laptop with E-Graffiti for a semester. All their activity has been logged and studied. An issue was that most of the posted notes were not related to their position. For example, a number of people posted notes to advertise a website, which underlines the need for tag trustworthiness.

2.2.2. GeoNotes. GeoNotes [4] has more functionalities than E-Graffiti. While posting a note, the user can choose how he is going to sign it (for privacy reason the user can write any text he wants as a signature), decide whether people are allowed to comment it, and decide whether anyone can remove this message. For the readers, the graphical interface of the application provides some interesting functionalities like showing all the neighboring messages or sort them according to different criteria. Each user maintains also a friends list, which can be used as a filter. But the trust and security aspects are not strong because it is easy to usurp someone's identity and post funny notes. An analysis of a GeoNotes log made during a real-use study showed that 6% of the messages have been signed using someone else's identity.

2.2.3. ActiveCampus Explorer. ActiveCampus Explorer [5] goes a step further by displaying also where other users are. Every user holds a PDA and its location is determined by comparing the signal strength of different wireless access points. Thus, the system knows the position of all its users, and communicates this information to all of them who are close enough. Like E-Graffiti and GeoNotes, it is also possible to tag objects.

2.2.4. SocialLight. Socialight [17] allows a user to post some data to a specific place, intended for himself, for his friends, or for everybody. Meta-data containing keywords and geographical coordinates are attached to the posted data, in order to facilitate searches. Tags are called Stickyshadows and can be viewed with some specific mobiles phones (equipped with a positioning system) via the Socialight Mobile application, or by browsing the Socialight website. A nice feature that they provide consists in showing Stickyshadows on maps.

2.2.5. Context Watcher. Context Watcher [16] is a mobile phone application written in Python for Nokia Series 60 based on the MobiLife framework [19]. The first version of this application already uses the notion of confirmed buddy for security and trust purposes. It may be expected that the following version will use the MobiLife trust engine as specified in the MobiLife framework for other purposes.

2.3. Speed Camera Services

As the number of speed cameras increases on European roads, we find more and more services that help the driver to avoid expensive pictures. We will talk neither about illegal means (for the majority of European countries), like the radar detectors provided by RadarBusters [6], nor about non-technical means like phone centrals providing vocal information. We will concentrate here only on information systems that inform drivers about speed camera positions, which is legal according to the law of most European countries.

2.3.1. SmartSpeed. SmartSpeed [7] is an application running on Windows Mobile that informs the driver about dangerous zones, traffic jams, and speed cameras. Working with all NMEA compatible Bluetooth GPS, the program compares the current position with the "events" to come and informs the user through a voice synthesizer. Maps and "events" files can be downloaded in advance, and a GPRS connection allows the user to get recent information. An interesting functionality allows any user to send a new event to the server, which will in turn inform all the users. A typical use consists in signaling mobile speed cameras to other drivers. Even if presented differently, it is clearly a way of doing spatial messaging. The light version a SmartSpeed is relatively cheap (30 € including free updates for one year) if you possess already a smartphone and a Bluetooth GPS. However, messages sent by other users to signal mobile speed cameras are not verified and are available only for one hour. And users are not really motivated to post such messages since they have nothing to gain in signaling a new "event". SmartSpeed seems more adapted to signal fix speed cameras than mobile ones.

2.3.2. Coyote. Coyote [8] is an independent system sold as a little box containing a GPS. When the driver approaches a speed camera, Coyote informs her/him orally about the remaining distance to this camera. To signal a new speed camera (or a new position for a mobile one), the user can simply press once the button on the top of the box. To signal a speed camera on the opposite direction, the user presses twice the button. This information is then sent to the server thanks to an included GPRS card, where a human operator verifies (previous messages of that user, comparison with other users, using another speed camera information service...) the plausibility of the information before broadcasting it to all users. Despite it is very simple to use, Coyote remains an expensive system (699 € for 2 years with unlimited use and including communication fees), which limits its user base. And if there are too few users, then the chance that a user is the first to discover a mobile speed camera (by being flashed!) is high.

2.3.3. Natel-Futé. Natel-Futé [9] is a system that informs its users about traffic jams and mobile speed cameras via SMS. The user gets only textual information about the positions of the speed cameras. It is therefore not possible to be automatically informed when the users approach a critical point, like it is done by SmartSpeed or Coyote. And the price is quite expensive too, since the users have to pay about 170 € for one year.

2.3.4. InfoRad. Autonomous and easy to use, InfoRad [10] beeps when the driver enters a "risky area". All the risky areas, materialized with a speed camera, are stored in the on-board database. It works thus only with fix speed cameras and it is not possible to signal a new one to other drivers. It allows however a user to add its own risky areas for personal use. Their website provides time-to-time updates of risky areas. The device with an unlimited access to their database costs about 200 €.

2.3.5. POIplaces. A POI (Point Of Interest) is a geo-referenced item that presents a particular interest, like a restaurant, a petrol stations, or a car park. Written in standard formats, POI lists can be used by most navigation systems. POIplaces [11] is a website where people can share their own POIs. One successful topic is speed cameras. In the same way than for restaurants or petrol stations, users can download for free the list of all speed cameras. Their navigation system can then be configured to emit a sound when they approach a POI. Free for everyone who already owns a GPS and a navigation device, this solution is however far from perfect. Since everybody can publish his/her POIs without any control, the speed cameras database is incomplete (lots of speed cameras are missing), redundant (several POIs for the same speed camera), incoherent (speed cameras have been found in a forest...), and mobile speed cameras are not taken into consideration.

2.3.6. GpsPasSion. GpsPasSion [12] provides active forums about the different topics of the GPS world. Some of them are specialized in the speed cameras domain and aim to collect information about their positions. They provide from time-to-time an update of their POIs file that can be freely downloaded. Compared to POIplaces, the list is smaller (lots of speed cameras are missing), but is more consistent since they check the information before updating their list. Members that submit new positions have also access to a list containing the preferred places for mobile speed cameras.

3. Towards an Efficient Speed Camera Tagging System

We saw in section 2.3 - "Speed Camera Services" that there is no ideal solution for a reasonable price. Systems like Coyote are very expensive. Natel-Futé provides only textual information instead of coordinates. A few systems

are more specifically (like Smartspeed) or even exclusively (like InfoRad) designed for fix speed cameras. And finally a few suffer from inconsistent, missing, incomplete and untrustworthy data (like POIplaces or GpsPasSion). We propose to build up on the previous work on computational trust described in Subsection 2.1 to achieve an efficient and safe application that informs drivers about speed cameras.

In order to validate our ideas for the development of trusted spatial messaging, we have been developing GeoVTag. GeoVTag [1][2], which is a framework for reading and posting trusted spatial messages. It is designed to run on a Java/GPRS-enabled mobile phone coupled with a Bluetooth GPS. The architecture is centralized. Each server manages vTags (virtual tags) of a specific subject and each of them is identified by a different URL and works independently from the others. Any user can obtain anonymously all the vTags in his neighborhood just by querying the server. To become a member, and therefore be able to review vTags (add comments to an existing vTag) or be able to create new vTags, a user has to register. The registration process allows you to choose a pseudonym and returns a key pair (as in many previous frameworks [14]) that will be used each time to reconnect to the server. Technically, posting virtual tags is quite simple. During a vTag edition, the application gets also the GPS position and adds it as a meta-data. The whole is then sent via Internet (using the HTTP protocol) to a vTag server. For the vTag reading, the principle is similar. The mobile user sends its current position to the server (still using the HTTP protocol) which returns all the available vTags at the given position and its neighborhood. The size of the neighborhood is specified during the request. GeoVTag serves as the vehicle for further research in trust for spatial messages. The goal of GeoVTag is to provide a generic framework for the development of trust mechanisms and models for spatial messages. A reference implementation, called GeoVTagRI, will be sufficiently generic to suit several different services. A main advantage over other spatial messaging systems is the trust engine that automatically computes the trustworthiness of a vTag. Each vTag contains different trust values computed according to the current context, the marks given by reviewers, the reputation of the author, and the friends list of the reader. When the users participate, they become friends of users who rate like them and their alerts become personalized: this incentive to participate is a main advantage over SmartSpeed, which provides no incentive for users as mentioned in Subsection 2.3.1. A user can then easily determine how trustworthy a given vTag is. Based on the GeoVTag platform and in order to validate the trust mechanisms, we designed and implemented an advanced speed camera application: the FoxyTag application.

4. The FoxyTag Application

In this section we present our novel application which aims to solve most of the issues described in Section 2.3.

4.1. Usage Scenario

Our application, built on top of GeoVTag, works as follows: a driver runs the client application (or simply *client*) on her/his mobile phone. Coupled to a GPS or to another positioning device, the client knows permanently its position, expressed in latitude and longitude. According to its context (position and heading), and to the trust relations with his friends, the driver gets a personalized list of all the speed cameras she/he is susceptible to cross within the next few minutes. The client is then responsible to warn her/him when she/he approaches a critical point. We call *protected zone* the zone that is covered by the speed camera list. From time-to-time, the client connects to the server to check if there are changes in the protected zone. It contacts also the server when it is close to leave the current protected zone, so that the server defines a new one and provides the corresponding speed camera list.

When a driver gets a false alarm (she/he is alerted about a speed camera that does not exist), or when she/he crosses a speed camera, she/he signals it to the server, which can then update the user's trust relationships. The more a user contributes, the more her/his trust relationships will evolve and become precise: the approach is based on computational trust as presented in subsection 2.1. In return, the user beneficiates of more correct alerts (closer to the reality).

4.2. Initiating a First Connection

To simplify the trust model and to avoid that the system becomes a Sybil attack [15] victim, we impose that a single user owns only one pseudonym. There are many means to achieve this goal [14]. It will depend on the identity scheme used and the GeoVTag model will be generic enough to accommodate any of these schemes by following the Entity Recognition (ER) approach [14]. One of them (the one that we have chosen for FoxyTag) simply consists in sending a reverse billing SMS (there are many phone company partners that provide such a service) containing a password. Then, each time the user connects to the server, she/he will identify herself/himself with her/his pseudonym and the corresponding password. Thus, any Sybil attack would be very costly (and unprofitable) due to the cost of the SMS at pseudonym creation time.

4.3. Exchanged Messages

Since Internet communications through a mobile phone are still quite expensive, our model tries to minimize them. A client sends only five different

messages (we do not take into consideration the messages exchanged to establish the connection):

- **CPZ, latitude, longitude**: CPZ is an abbreviation for Check Protected Zone. The message indicates to the server that the driver is currently at the position defined by the given latitude and longitude, and that she/he wants the protected zone to be checked. Checking is done in the following way: if there are changes in the current zone, or if the user is close to the border of the zone, then a new protected zone is computed. Otherwise, the server just answers that the current protected zone is still OK.
- **CAN, latitude, longitude, heading**: CAN is an abbreviation for cancel. The message indicates that there is no, or not anymore, any speed cameras at the given position and for the given heading. The heading is important, it avoids that a user cancels a speed camera in the opposite direction.
- **MSC, latitude, longitude, heading**: MSC is an abbreviation for Mobile Speed Camera. The message indicates that there is a mobile speed camera at the given position and for the given heading.
- **FSC, latitude, longitude, heading**: FSC is an abbreviation for Fix Speed Camera. The message indicates that there is a fix speed camera at the given position and for the given heading.
- **OTC, latitude, longitude, heading**: OTC is an abbreviation for Other Type of Camera. The message indicates that there is a camera that does not measure the speed of the driver (like the ones used to record registration numbers) at the given position and for the given heading.

Headings are usually positive numbers between 0 and 360 (0° = North, 90° = East, 180° = South and 270° = West), but it is possible to add a minus sign to indicate that the measurement has been taken from the opposite direction (for example a driver sees a mobile speed camera in the opposite direction). The server will therefore compute the correct heading, and, during trust computation, take into account that the precision of the positioning is not optimal.

A message from the server is either an acknowledgement, to say for example that there is no change in the protected zone, or a list of speed cameras as well as information about the space covered by the new protected zone.

4.4. Defining Protected Zones

As a first approach, we thought that the ideal shape and size of a protected zone can be computed mathematically according to the context. For example, we guess we are in a city if the speed is low. In a city, the speed camera density is high and the users are susceptible to change often the heading, thus we would choose a small circle. Another example may be to guess that we are on a highway because the speed is high. We would then choose a thin (the chance that the car changes suddenly its heading is low) and long (the speed camera density is low) rectangle. But we were wrong. The speed and the heading is only a part of the context. For example, on a highway speed cameras usually form a line along the road (since highways are most of the time in the countryside), so a wide zone contains not necessarily more speed cameras than a thin one. Computing the ideal zone according to the context is far more complex than we could initially think, and this issue is therefore out of the scope of this paper.

Since mathematics could not help us to define the best protected zones, we set up dozens of different hypothetical scenarios and deduced experimentally the best parameters. It came out that a circle with a diameter of 12 km suits in most of our scenarios. All the numbers presented in the rest of this paper are also deduced experimentally.

When the user connects to the system (at time t_0), the server defines a protected zone as a 12-km-diameter circle centered at the user's current position (see Figure 2). Every 5 minutes, or when the distance between the driver and the center of the circle is higher than 5.5 km, the client connects to the server and sends him a CPZ (Check Protected Zone) message. For example, in Figure 3, a driver follows a given path. At time t_1, the driver is at 5.5 km from the center (or he reached the internal circle in the figure). The client connects to the server which computes a new protected zone. To do that, the server draws a line between t_0 and t_1, prolongs it for 5.25 km, and defines the end of this line as the center of the second circle (see Figure 3). When the driver reaches t_2 (5 minutes after t_1), the client sends a CPZ message. Since there is no change in the protected zone, the client keeps the same one. And finally, the process repeats when the driver reaches the internal circle of the second protected zone, at time t_3.

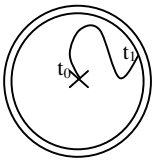

Figure 2

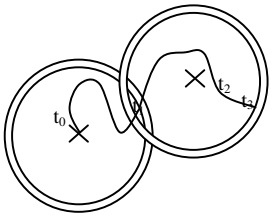

Figure 3

If a user requests two new protected zone within a minute, the second one is centered on the driver's current position. This could happen for example when the driver changes his/her heading just after the computation of a new protected zone, or if the driver goes round of a point that is between two protected zones.

4.5. Specific Rules

The client must follow a few additional rules in order to behave in a similar way as the others, and therefore get the same chances in building good trust relationships. The first rule concerns speed cameras inside shadow zones (zones that prevent the client from knowing its current position, like in a tunnel). The rule says that if a speed camera is in a shadow zone, then the warning (message signaling the presence of a speed camera) must be posted at the shadow zone entry. Technically, the application always remembers the last position that it gets from the GPS, and uses it as the position of the shadow zone entry when the GPS becomes unable to make a fix. The second rule concerns speed cameras close to shadow zone exits. In order to leave enough time between the moment the device acquires a new positioning signal and the time the driver crosses a speed camera, the second rule imposes to use the coordinates of the shadow zone entry when posting a new message within ten seconds after exiting the shadow zone.

5. Time-sensitive Trust Metrics

As explained above, mobile speed cameras can be installed and removed very quickly and we cannot directly measure their time of installation and removal as for the location measured by the GPS module. Thus, there is high uncertainty concerning the time aspect of context. The users can collaborate (but also collude) to confirm or deny the presence of a speed camera. However, the correctness of their vote strongly depends on time and worse not in a continuous manner. As soon as the speed camera is removed, a vote that confirmed that this camera exists drops from true to false, correct to incorrect. A decay function continuous over time or with mere a priori fixed discrete levels based on the elapsed time is not relevant. As surveyed in Subsection 2.1, the previous trust metrics that take time into considerations proposed such simple aging functions and are not suitable for our fast context changing application domain.

To tackle our application domain, trust metrics with a more fine-grained integration of time than the time elapsed since the last interaction are needed. The first step in this direction is to timestamp each interaction and evidence. Once each piece of evidence has a timestamp, the trust metric can use this information for more complex analysis integrating time. For example, patterns in the evidence update can be detected and used to revise the trust value because the patterns trigger themselves revisions of evidence. For example, if a speed camera alert has been removed due to five users who have denied the presence of a camera against two who confirmed its presence, the outcome is that the two users were not correct. However, if the next user is flashed shortly afterwards, it may mean that the five denying users were malicious users. In this case, the outcome must be revised, which means that the two users were right and that their trust value should be revised. The second step for more fine-grained time-sensitive trust metrics is to allow the trust engine to use discrete time-based functions rather than arbitrary continuous decay function. For example, the continuous flow of user confirmations should not conceal a few recent consecutive denials that should lead to a quick removal of the alert.

To set up a reference time-sensitive trust metric, we assumed a world where all users make very few mistakes when reporting their speed camera observation (in reality, it may happen when a user pushes the wrong mobile phone button) and are not malicious. In such a world, we assumed that when a user reports a new speed camera, the application should create a new alert because the cost of being flashed is much greater than to be disturbed by an alert. Then, if one user denies having seen a speed camera after an alert, the application should remove the alert, independently of the number of users who had confirmed the presence of the speed camera before.

If we assume a world where the majority of users makes few mistakes (or are not malicious) and all users have approximately the same level of participation, this first discrete rule-based trust metric should be a quite good reference trust metric with quite good results. It is why we set up the trust engine with the following basic discrete TIme-Patterned (TIP) trust metric:

- If a user sees and mentions a new camera, then a new tag is created. The default value of the tag trust value equals 0.
- If a user sees and mentions an existing camera (one that was signalized by a tag), then the corresponding tag trust value is set to 1.
- If a user gets an alarm about a camera that does not exist anymore and mentions it, the trust value of the corresponding tag is decreased by 1.
- A tag whose trust value reaches -1 becomes an inactive alert.

The main idea behind these rules is that if a user signals by mistake a new speed camera, then the next user can alone cancel the message, but if a second driver confirms the existence of a speed camera, then we need two people to remove a tag. Our simulation results in Section 8 confirm that when most of the users are not malicious, the assumption holds but when a few users are not able to use the application correctly (maybe they mixed the meaning of each button when they read the user's guide) this first trust metric does not give decent results.

To isolate such a minority of users who do not vote as the majority of users, we designed another trust metric that would probabilistically estimates the trust value of a

user based on the outcomes of his/her past actions from the point of view of the whole community of users. Another requirement for our second trust metric was to approximately give results as good as the basic metric in the other cases. According to Ziegler and Lausen trust metrics taxonomy [30], this trust metric is a Global Centralized (GC) trust metric and we call it the TIP Probabilistic (TIPP) GC trust metric. In this TIPP GC trust metric, the future actions of the users who have a trust value below a threshold between 0 and 1 are ignored. The users cannot rebuild their trust and we assume it is fine because we said that we assume that a majority of users are able to correctly understand and use the application. The users who are not able to correctly report their observations can still receive the correct alerts anyway.

However, due to the fact that a probabilistic approach requires a good deal of evidence before becoming accurate enough, we introduced a threshold number of outcome observations. When a user reports a new speed camera or votes for an existing alert (confirming or denying the presence of the camera), the trust engine decides to use either the basic trust metric if the number of direct observations about this user is below the threshold or the probabilistic trust metric if enough observations have been made to rely on probabilities. The default value that we used for this threshold is four. To further avoid the costly situations of not setting up an alert when a new real speed camera is reported, we set another threshold to only ignore a report of a new speed camera if the reporting user's trust value is strictly below (by default 0.3). A final higher threshold is set up for the less costly CAN reports: the CAN report is ignored if the user who reports a CAN has a trust value strictly below this final threshold (by default 0.5).

If an alert is created, the alert is alive until two following CANs are not ignored.

The trust engine stores in P the count of the positive action outcomes of each user and in N the count of the negative outcomes. The trust value of the user is initially set to 0.5 and is computed as follows:

$$TrustValue = \frac{P}{N+P}$$

The outcome of a user's action (creation of a new alert or confirmation/denial of a live alert) is computed when the trust engine detects the following chronologically ordered actions patterns:

- $Action_1$, $Action_2$, … , $Action_N$: this line means that N actions have been done from $action_1$ to $action_N$ in a chronological order.
- Alert Creation, CAN, CAN: apparently, the first user created an alert about a speed camera that is not present. N of the alert creator is increased (although she/he might have been unlucky to create an alert on a speed camera that was removed just after creation; the time between creation and refutation may be taken into account in an extension of this metric).
- Alert Creation, CAN, Confirmation, CAN, CAN: apparently, the first user created an alert about a speed camera that is not present. N of the creator is increased.
- Alert Creation, Confirmation: apparently, the creator was right: P of the alert creator is increased.
- Confirmation, Confirmation: P of the first confirming user is increased.
- CAN, Confirmation, CAN, CAN: apparently, the user who confirmed the presence of the speed camera was wrong. N of the confirming user is increased and the Ps of all the three denying users are increased.
- CAN, Confirmation, Confirmation: apparently, the first denying user was wrong and his/her N is increased.

Figure 4 summarizes the default TIPP GC trust metric process.

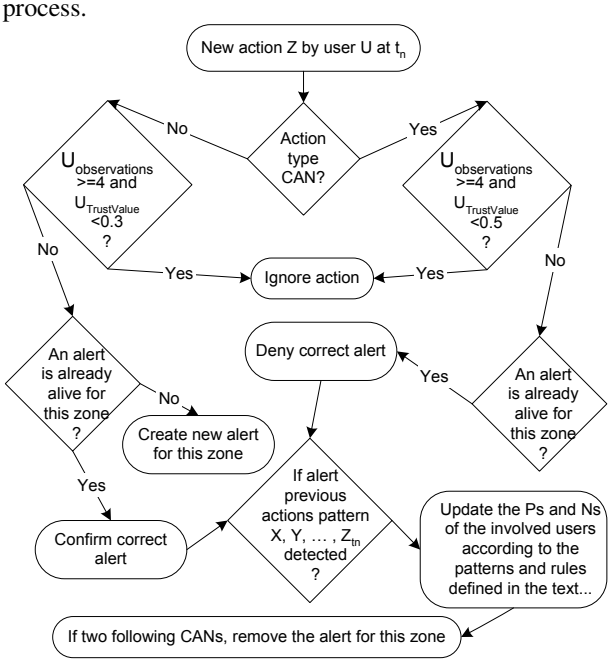

Figure 4. TIPP GC Trust Metric Summary

The final TIPP trust metric that we have developed consists of a Local Centralized (LC) trust metric [30]. The TIPP LC trust metric corresponds to the TIPP GC trust metric until the alert is presented to the requesting user. Just before the alert is presented to the user, an additional test is done: if the evidence supporting the alert comes from enough users who have had observations similar to the observations made by the requesting user, the alert is presented to the user; otherwise, the alert is ignored. The goal of the TIPP LC trust metric is to allow the users to have a personalized trust metric that is more resilient to collusion attacks as it has been done in the related work [31, 32]. Although this adjunct technique is a further line of defense against collusion, due to space limitation and

no extra novelty in how we have reused this technique covered in the related work, we do not detail further this TIPP LC trust metric.

6. Our Simulator

In order to validate our new time-sensitive trust metrics, we set up a Java-based car traffic simulator that randomly positions speed cameras on a road and runs the user's cars according to given scenario parameters. An additional user, whose behavior can also be completely specified, logs its observations and returns the number of true positives (alarm: yes, camera: yes), false positives (alarm: yes, camera: no), true negatives (alarm: no, camera: no) and false negatives (alarm: no, camera: yes).

We model our road as a single way on a highway. Exits are numbered between 1 and n. Between two exits there is only one speed camera, numbered between 1 and n-1. So the camera c1 is between exits e1 and e2, the camera c2 is between exits e2 and e3, and so on. For example:

```
e1    c1    e2    c2    e3    c3    e4
 +-----*-----+-----*-----+-----*-----+
```

This model does not make any difference between a mobile, a fixed or another type of camera (MSC, FCS and OTC are considered as the same message). This model seems to be very simplistic. It is however sufficient to validate our trust metrics. Of course, we do not take into account contextual information, like shadow areas or what happens when the user posts a message concerning the opposite direction. These are more technical issues that need to be validated in the field and that is what we actually did with a real device in a real car as detailed in Section 7. Since we can define the behavior of every user (where they enter and exit, how reliable they are by signaling speed cameras...) as well as the behavior of each speed camera (frequency of turning on, for how long...), we can precisely define which user drives in which area and how many speed cameras she/he is meant to cross on average.

Our simulator accepts an input file that looks like this:

```
cam;1-4;8;15  // three times a day, for 15 min.
cam;5-5;24;2  // once a day, for 2 min.
cam;5-5;240;3 // once every 10 days, for 3 min.
usr;1-10;1-5;24;95;90  // 95% cp, 90% cn
usr;1-1;3-5;240;80;75  // 80% cp, 75% cn
usr;11-15;1-10;1;10;10 // hacker!
usr;11-11;1-10;0;20;25 // hacker!
col;5-7;1-11;6;10;100  // 10% cp, 100% cn
scn;100;2;run(24);pas(1,10);act(1,10,50,60)
scn;10;4;run(2400);pas(3,5);run(1);act(1,10,100
```

- In the first line, "cam;1-4;8;15" means that cameras 1 to 4 have one chance out of 8 to become active in an hour, and when one becomes active then it stays active for 15 minutes.
- The two next lines define two different behaviors for the camera 5.
- In the fourth line, "usr;1-10;1-5;24;95;90" means that users 1 to 10 entry the highway at 1 and exits it at 5, that they run once a day, and that they vote 95% of the time correctly when they signal the presence of a speed camera (cp, or correct positive), and 90% of the time correctly when they cancel a camera (cn, or correct negative). More precisely, this means that when a user sees a camera, she/he will send a MSC message 95% of the time, and a CAN message 5% of the time. And when she/he gets an alarm about a camera that does not exist (anymore), then she/he sends a CAN message 90% of the time, and a MSC message 10% of the time. In the two cases, the user always votes. We do not take into account users that would "ignore" a vote, since a user that does not vote is like a user that does not exist, according to the system.
- In the collusion line, "col;5-7;1-11;6;10;100" we deduce that users 5 to 7 are colluding by entering all at the same time on entry 1, exiting on exit 11, and voting (all similarly) about all 6 hours with 10% of correct positives and 100% of correct negatives. In other words, this collusion tries to cancel speed camera alerts.
- The first scenario, "scn;100;2;run(24);pas(1,10); act(1,10,50,60)" contains 100 big loops and 2 small loops. The scenario itself will be executed twice, then the trust engine is initialized, and then we re-execute the scenario twice. And so on (100 times). In other words, the scenario is repeated 200 times, and the trust engine is initialized every 2 times.
- run(t) means that the system will run for t hours (simulation time). Each minute, the go method of each camera and each user is called, allowing them to act according to their behavior.
- pas(e1, e2) means that our test user will passively drive once from exit e1 to exit e2. Passively means that he does not vote. His observations are logged and printed.
- act(e1, e2, cp, cn) means that our test user will actively drive once from exit e1 to exit e2 and has cp (correct positive) chances (in %) to vote correctly if he sees a speed camera, and cn (correct negative) chances (in %) to vote correctly when she/he tries to cancel a speed camera that does not exist (anymore). Her/his observations are logged and printed.
- Comments is everything after a // and they are ignored.

7. Results of the In-The-Field Validation

The validation has been done by analysing the log files that where created while driving in different places in Switzerland. The client application runs on a Sony-Ericsson K750 mobile phone, paired with a GlobalSat Bluetooth GPS (see Figure 5 below).

Figure 5. FoxyTag Prototype Photograph

We did essentially three main validations: protected zones, shadow areas, and precision of tags positions.

To validate the concept of the protected zones computations, we used Google Earth and printed the different zones. The results were as expected. To estimate the costs of our system, we logged also the number of tags that were downloaded each time the client got a new protected zone. It came out that a 2-hour drive in a mixed area (cities and highways) required only 15 ko of data transfer. To overestimate our costs, we also computed that a user who would drive at 150 km/h in the Geneva city (Switzerland) would only require about 1000 bytes/minute. According to the current GPRS prices in most European countries (2006), this corresponds to less than 1 € for a 2-hour trip.

The shadow areas validation has simply be done by posting message from inside tunnels and checking via Google Earth that the corresponding tags were placed at the entry of these tunnels. The results were as expected.

To estimate the precision of the tags positions, we did this last validation by driving twice on the same road. The first time we stopped at different points for a few seconds and measured the average position. The second time we placed the tags without stopping, using the *2-second* rule. This rule has been set up experimentally and stipulates that, since the correct position is given two seconds too late (GPS satellites send data only once a second, refresh rate of the application...), a user must count in his head two seconds before posting the message. A former implementation counted these two seconds automatically, but our tests showed us that it is safer to leave two seconds more for the driver to press the button. The following Figure 6 shows a very dangerous road (holes, bumps...) in Wallis (Switzerland), where we decided to record all the critical points using our application.

Driving at 80 km/h and using the 2-second rule, we observe that the second records (dangers_202 - dangers_204) are very close to the first ones (dangers_102 - dangers_104). Actually the biggest error is between danger_102 and dangers_202, where we have only 18 meters (the GPS's precision is less than 10 meters).

Figure 6. Results on a Dangerous Road in Wallis

8. Results of the Simulations

To obtain a sufficiently high number of observations in order to compare with statistical confidence our basic TIP trust metric with our TIPP GC trust metric, we set up much more speed cameras than we would find in a real case. For example, we can compare for each trust metric the number of true positives (alarm: yes, camera: yes) with the number of false negatives (alarm: no, camera: yes), which represents in a way the money that the user saves. To simplify the reading of the results, we present them in tables and we use the following abbreviations: BASIC refers to the trust engine with the basic TIP trust metric, PROB for the trust engine with the TIPP GC trust metric, "TP yy" means true positives (alarm: yes, camera: yes), "FP yn" means false positives (alarm: yes, camera: no), "TN nn" means true negatives (alarm: no, camera: no) and "FN ny" means false negatives (alarm: no, camera: yes).

Scenario 1:

```
cam;1-10;24;180
usr;1-100;1-11;24;95;95
scn;100;100;run(24);act(1,11,95,95)
```

We have 10 speed cameras that are turned on once a day (on average) for 3 hours, and 100 users who cross all of them once a day (on average) and who signal 95% of the time correctly what they observe. The scenario works as follows. The system runs for 24 hours and then our test user (the one that counts the TP, FP, TN and FN values) drives to all the speed cameras. This is repeated 100 times (small loop) before the trust engine is reinitialized. And the whole is repeated 100 times (big loop). This first basic scenario, containing only reliable users, can be considered as a reference for the next ones.

Scenario 1	TP yy	FP yn	TN nn	FN ny
BASIC	10100	1845	87114	941
PROB	10091	1860	87140	909

We observe a diminution of 3.4 % in FN between BASIC and PROB. FN is the most important/costly value since in this case you cross a speed camera without being previously informed.

Scenario 2:

```
cam;1-10;24;180
usr;1-100;1-11;24;95;95
usr;101-200;1-11;24;75;75
scn;100;100;run(24);act(1,11,95,95)
```

In this second scenario we added one hundred users that vote less precisely (75% instead of 95%).

Scenario 2	TP yy	FP yn	TN nn	FN ny
BASIC	10345	1011	87861	783
PROB	10410	1020	87885	685

Compared to scenario 1, it is interesting to note that adding 100 users, even if they are not very reliable (75%), helps to decrease the numbers of false positives (FP) and false negatives (FN). We observe also a diminution of 12.5 % in FN when we compare BASIC with PROB.

Scenario 3:

```
cam;1-10;24;180
usr;1-100;1-11;24;95;95
usr;101-101;1-11;1;10;95
scn;100;100;run(24);act(1,11,95,95)
```

We took scenario 1 and added a malicious user (n° 101) that votes for all the speed cameras once an hour, with 10% of correct positives and 95% of correct negatives. In other words, this user tries to cancels as much as possible tags that mention speed cameras.

Scenario 5	TP yy	FP yn	TN nn	FN ny
BASIC	9624	1292	87473	1611
PROB	10191	1565	87121	1123

We observe a diminution of 30.3 % in FN between BASIC and PROB.

Scenario 4:

```
cam;1-10;24;180
usr;1-100;1-11;24;95;95
usr;101-200;1-11;1;10;95
scn;100;100;run(24);act(1,11,95,95)
```

Same as scenario 3, but with one hundred malicious users.

Scenario 7	TP yy	FP yn	TN nn	FN ny
BASIC	1472	2	88887	9639
PROB	2513	6	88861	8620

There is a diminution of 10.6 % in FN between BASIC and PROB.

Scenario 5:

```
cam;1-10;24;180
usr;1-100;1-11;24;95;95
usr;101-200;1-11;1;50;95
scn;100;100;run(24);act(1,11,95,95)
```

Same as scenario 4, but we increased the number of correct positive votes (50% instead of 10%).

Scenario 9	TP yy	FP yn	TN nn	FN ny
BASIC	7679	25	88717	3579
PROB	8503	29	88751	2717

We observe a diminution of 24.1 % in FN between BASIC and PROB.

Scenario 6:

```
cam;1-10;24;180
usr;1-100;1-11;24;95;95
usr;101-200;1-11;24;100;100
scn;100;100;run(24);act(1,11,95,95)
```

A scenario with only good guys: 100 of them vote correctly 95% of the time, and the other 100 vote correctly 100% of the time. This scenario is interesting when compared to the next one.

Scenario 10	TP yy	FP yn	TN nn	FN ny
BASIC	10848	818	87896	438
PROB	10673	946	87935	446

We observe an increase of 1.8 % in FN between BASIC and PROB.

Scenario 7:

```
cam;1-10;24;180
usr;1-100;1-11;24;95;95
usr;101-200;1-11;24;100;100
col;101-200;1-11;6;10;100
scn;100;100;run(24);act(1,11,95,95)
```

Similar to scenario 10, but users 101 to 200 collude 4 times a day (in addition to their normal behavior). They vote similarly on all speed cameras with 10% of correct positive votes and 100% of correct negative votes. They try therefore to cancel valid tags.

Scenario 11	TP yy	FP yn	TN nn	FN ny
BASIC	10395	862	88100	643
PROB	10507	805	88107	581

We observe a diminution of 9.6 % in FN between BASIC and PROB.

Scenario 8:

```
cam;1-10;8;180
usr;1-100;1-11;24;95;95
usr;101-110;1-11;6;5;5
scn;100;100;run(24);act(1,11,95,95)
```

In this scenario we observe what happens when 10% of the users mix the MSC and CAN messages (this can happen for example if the user's manual is not clear enough, or if the user tries the system without reading it). Note that in order to get higher figures and therefore be able to make a more precise comparison, we increased the frequency of the speed cameras (three times a day instead off ones).

Scenario 12	TP yy	FP yn	TN nn	FN ny
BASIC	21876	3935	69043	5146
PROB	24591	4554	68350	2505

We observe a diminution of 51.3 % in FN between BASIC and PROB.

9. Conclusion

The TIPP GC trust metric fulfils the requirements that we have set. First, in case there is a minority of users who do not act as the majority of users (either because they are not able to use the mobile phone application correctly or they are malicious), these users are more isolated than with the basic TIP trust metric. Second, in the other cases covered, the results of the TIPP GC trust metric are never dramatically worse than the basic TIP trust metric. Our work underlines that time has been overlooked in trust metrics and that previous trust metrics with simple time-based decay function are not sufficient for an application domain with fast changing context such as FoxyTag.

The FoxyTag application maintains trustworthy information without the cost of human manual checks and seems to be an affordable alternative to current speed camera alerting systems.

In future work, we intend to extend and apply our work to other types of spatial messaging applications.

10. References

[1] M. Deriaz, "Trust and Security for Spatial Messaging", OSG technical report, 2005.

[2] M. Deriaz, "GeoVTag", OSG technical report, 2005.

[3] Burrell, Jenna, Gay, Geri K., "E-graffiti: evaluating real-world use of a context-aware system", in Interacting with Computers, 2002.

[4] Persson, P., Espinoza, F., Fagerberg, P., Sandin, A., and Cöster, R. GeoNotes, "A Location-based Information System for Public Spaces", in Höök, Benyon, and Munro (eds.), Readings in Social Navigation of Information Space, Springer, 2000.

[5] William G. Griswold, Patricia Shanahan, Steven W. Brown, Robert S. Boyer, Matt Ratto, R. Benjamin Shapiro, Tan Minh Truong, "ActiveCampus: Experiments in Community-Oriented Ubiquitous Computing", IEEE Computer, 2004.

[6] Website: http://www.radarbusters.com/

[7] Website: http://www.smartspeed.fr/

[8] Website: http://www.moncoyote.com/

[9] Website: http://www.natel-fute.ch/

[10] Website: http://www.gpsinforad.co.uk/

[11] Website: http://poiplace.oabsoftware.nl/

[12] Website: http://www.gpspassion.com/forumsen/topic.asp?TOPIC_ID=21763

[13] S. Marsh, "Formalising Trust as a Computational Concept", PhD Thesis, University of Stirling, 1994.

[14] J.-M. Seigneur, "Trust, Security and Privacy in Global Computing", PhD Thesis, Trinity College Dublin, 2005.

[15] John R. Douceur, "The Sybil attack", in Proceedings of the IPTPS02 Workshop, 2002.

[16] Website: http://www.lab.telin.nl/~koolwaaij/showcase/crf/cw.html

[17] Website: http://socialight.com/

[18] Website: http://secure.dsg.cs.tcd.ie/

[19] Website: http://www.ist-mobilife.org/

[20] McKnight, D., and Chervany, N. L., "The Meanings of Trust", MISRC 96-04, University of Minnesota, 1996.

[21] Website: http://www.trustcomp.org/

[22] Sabater, J., & Sierra, C., "Review on Computational Trust and Reputation Models", Artificial Intelligence Review, Kluwer, 2005.

[23] Golbeck, J., & Hendler, J., "Accuracy of Metrics for Inferring Trust and Reputation in Semantic Web-based Social Networks", 2004.

[24] Guha, R., "Open Rating Systems", 2004.

[25] Dimmock, N., "Using trust and risk for access control in Global Computing", PhD thesis, University of Cambridge, 2005.

[26] Bizer, C., Cyganiak, R., Gauss, T., & Maresch, O.: The TriQL.P Browser, "Filtering Information using Context-, Content- and Rating-Based Trust Policies", paper presented at the Semantic Web and Policy Workshop at the 4th International Semantic Web Conference, 2005.

[27] J.-M. Seigneur., "AmbiTrust? Immutable and Context-Aware Trust Fusion", OSG technical report, 2005.

[28] N. Mezzetti, "A Socially Inspired Reputation Model", in Proceedings of EuroPKI, 2004.

[30] C.-N. Ziegler and G. Lausen, "Spreading Activation Models for Trust Propagation", in Proceedings of the International Conference on e-Technology, e-Commerce, and e-Service, IEEE, 2004.

[31] S. Buchegger and J.-Y. Le Boudec, "A Robust Reputation System for P2P and Mobile Ad-hoc Networks", in Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems, 2004.

[32] Quercia, D., Hailes, S., and Capra, L., "B-trust: Bayesian Trust Framework for Pervasive Computing", in Proceedings of the 4th International Conference on Trust Management, LNCS, Springer, 2006.