The Uncertainty of the Truth

Michel Deriaz
University of Geneva, Switzerland
{firstname.lastname} [at] unige.ch

Abstract

How to trust without knowing the truth? This is probably the key question that arises while designing applications using virtual tags. A virtual tag is a geo-referenced note that is visible for all the people that are in a specific place. But what if you see a tag about an event or an object that is not here? How to know if you are facing a spam attack, or if the tag is simply outdated? And, how to update the trust values of the author and the other people that confirmed the tag, since you do not know if they are honest? To answer these questions, we designed and implemented FoxyTag, a free and collaborative system which consist in posting virtual tags over speed cameras in order to warn the other drivers. We used it to test our new generic trust engine and got very promising results.

1. Introduction

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 reaching the same place gets the message. For example, let us consider the community of the Mt-Blanc mountain guides. The members would like to inform their colleagues about dangers in specific places. One guide publishes a geo-referenced message that informs about a high risk of avalanches, and any other guide that goes to the same place will get the warning, and comment it if necessary. Spatial messaging is a kind of blog in which editors and readers share the same physical place.

There are many reasons to believe that spatial messaging will become a widespread concept in a nearby future. Today, people use the connection capabilities of their mobile phone mostly in one way, to download information. But in the same way that people passed from television to Internet, the next generation of users will probably become more "active" and create new content with their mobile phones. We already observe this tendency today for specific cases, like sharing pictures or videos recorded by mobile phones and published on some websites. If we remember how fast the computer power and the communication capabilities of these little devices increase, we can easily paint a glorious future for mobile technology.

We will see in the related work section that lots of non critical applications are already running on mobile technology. We insist here on the "non critical" aspect; it clearly implies that there is today no third party that proposes any serious application using virtual tags.

To our view the reason is simple: We cannot trust their tags. We do not talk about POI (Points Of Interest), data that is usually provided by a unique source and copied in the devices. We talk here about virtual tags, pieces of information that can be posted by unknown users and modified by other unknown users. These virtual tags are posted in a collaborative way, like it is done in the Google Earth Community [1] where every user can post any geo-referenced information. But, we observe then that we cannot trust this information. Security tools are not sufficient; even if you can be sure about the identity of an author, it is useless if you do not know him and therefore cannot trust the content of his message.

And trusting virtual tags is not that easy. We will see that applying "conventional" trust algorithms doesn't work. One reason is because of what we call the "Uncertainty of the Truth". For instance, a user that sees a tag that warns about a danger of avalanche in mid-summer doesn't know if it is spam (in which case he must decrease the trust value of the tag's author) or if the tag is simply outdated.

2. Related Work

At least to our knowledge, we are the first to study the trust aspects in spatial messaging. Actually, even if we type only "spatial messaging" in Google [2], the first results point directly to our former papers. We find also some people that use our definition to describe it, like for instance in the alvafilm website [3]. So if we add the trust component to spatial messaging, we reduce even more the chances of finding some parallel work.

Since we couldn't find any similar work, we divided this section in three parts. The first part describes other work done for spatial messaging. The second part gives a state of the art in the trust domain. And finally, since we used a speed cameras warning system in order to test our models, the third part gives a list of other warning systems.

2.1. Spatial Messaging

Before starting this section, we would like to precise the difference between spatial messaging and LBS (Location Based Services). In short, LBS is a kind of spatial messaging in which the user can only get data, and not post it. Lots of LBS applications for augmented cities (tourists get information, in their mother tongue, about their current place) or augmented museums (visitors get information about what they are looking at) have already been implemented. We are clearly interested in spatial messaging in general, where users also post information.

2.1.1. E-Graffiti. E-Graffiti [4] 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. And the results are far from encouraging. At the end of the semester, it came out that a user logged into the system only 7.6 times in average (std dev: 12.6), and that actually most of the users stuck to initial test messages. Another disappointment 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. The system was designed so that the user could only get messages available at his current position, but it was possible to post a new message at any place from anywhere.

Technically, the position of the user is determined by the wireless access point to which the device is connected. The precision is therefore limited to the building in which the user is.

2.1.2. GeoNotes. GeoNotes [5] 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. Inspired by the E-Graffiti evaluation, GeoNotes discarded the remote authoring of tags as well as the possibility to "direct" notes to certain users.

The main interest of the GeoNotes authors seems to be the navigation problems in the virtual messages space. How to find a specific note? How to select only relevant messages? One answer of these questions consists in giving to the readers the possibility of ranking the notes. Each user maintains also a friends list, which can be used as a filter. But the trust and security aspects have not been taken into account. It is easy to usurp someone's identity and post false 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.

2.1.3. ActiveCampus Explorer. ActiveCampus Explorer [6] 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 the all of them that are close together. Like E-Graffiti and GeoNotes, it is also possible to tag objects.

2.1.4. Socialight. Socialight [7] 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 (and equipped with a positioning system) via the Socialight Mobile application, or by browsing the Socialight website. A nice feature they provide consists in showing Stickyshadows on maps.

2.1.5. Context Watcher. Context Watcher [8] is a mobile phone application written in Python for Nokia Series 60 based on the MobiLife framework [9]. The first version of this application already uses the notion of confirmed buddy for security and trust purposes. They have a part that they called trust engine in their architecture but a closer look at it shows that it is actually only an access control system. Policies and profiles are used to decide who can access what data and under what condition, but there is no trust mechanism that informs how reliable a requested information is.

2.1.6. Summary. These projects don’t seem to be successful. E-Graffiti and GeoNotes have been abandoned shortly after their launch. Socialight is still active, but there are seldom new posts. We believe that the lack of success is related to the lack of interest... in publishing notes just for publishing notes! Spatial messaging would probably have more chance to emerge if we focus on specific communities, with real problems that could be solved by this concept, rather than imposing the system to students without giving them any good reason to use it. But then we need a trust mechanism to exclude malevolent users. In GeoNotes people may stay anonymous, but we saw that users then usurped others’ identities; it is therefore not possible to trust a message. In E-Graffiti users reveal their real identity, but it useless to know that a message has been posted by a certain "John" if you do not know John.

Commercial systems usually implement all the conventional security tools (username, password), but there is no trust engine that informs about the reliability of a given message. It means that it is always a human that plays the role of the trust engine and that excludes what he thinks are malevolent users.

However, in widely deployed systems (like for instance our FoxyTag [10] application that informs about speed cameras in all Europe) where there is only very little human interaction, only a trust engine can ensure a high quality of the data.

2.2. Trust

Trust is a very active research domain. It started by providing solutions for centralized systems (for instance the reputation system in eBay where seller and buyer can rate one another after a transaction), and then quickly switched to peer-to-peer systems. Peers rate each other and the combination of all the values informs about the reputation of the peer. The challenge here is where to store trust values, as there is no central server. Among the proposed solutions, we mention here a few of them:

In EigenTrust [11] each peer has a set of mother peers responsible for storing its trust value, and therefore each peer acts also as a mother peer for others. It resists to an attack even when up to 70% of the peers are colluding in order to subvert the system. Peers are anonymous.

An interesting system that is similar to EigenTrust, but in which peers store their own trust value locally, is called Elicitation-Storage [12]. The Elicitation-Storage protocol is used to protect cryptographically the trust value. The requester gets the IP address of the former requesters and checks with them the authenticity of their vote.

The Secure project [13] aimed to describe in a formal way what trust is, staying as close as possible to the human notion of trust. The motivation for the project was that the number of entities in Internet systems is becoming very large. Consequently, it was important to develop security models that allow nodes to measure the risk involved in interacting with other nodes that they have not met before. The secure project implementation has been tested with a mail application: A proxy between the peer and his mailbox was analyzing the behavior of the user (for instance if he moved a message in his spam folder) and updated the trust values according to it. The reputation system allowed the different peers to share their information in order to exclude faster the spammers.

Kinateder and Rothermel [14] present a peer-to-peer system that provides trust and recommendations about different categories of topics. Similar to sites like Epinion.com or the rating system that we find in eBay, but peer-to-peer.

The TrustMe protocol [15] builds trust in peer-to-peer networks. The trust value of a specific peer is anonymously stored on another peer. Communications are encrypted using sets of private/public keys. The drawback is that all peers have to connect to a bootstrap server when they join and when they leave the network (in order to transmit the hosted trusted values to another peer).

Anwitaman Datta, Manfred Hauswirth and Karl Aberer present in [16] how P-Grid can be used to implement a distributed PKI (Public Key Infrastructure), enabling c2c (customer to customer) services like eBay but without any centralized system. Unlike PGP that uses the web of trust approach to access a particular public key, this system uses a statistical method; many peers are queried, and the information is rejected if a quorum a peers cannot be obtained.

Very interesting and promising decentralized solutions like the EigenTrust algorithm made the community to forget one aspect that is only seldom taken into account: time. In practice time is important. Someone you trusted a long time ago is perhaps not trusty anymore. Even people with a very high reputation can become malevolent afterwards. Since in human communities the trust is very time dependent, we believe that this component should also be included in trust engines and particularly in the spatial messaging context where posted information can simply become obsolete after a while.

Guha [17] built a generic trust engine allowing people to rate the content and the former ratings. He recognized however that in case of highly dynamic systems (like in spatial messaging where tags can appear and disappear very quickly), "Understanding the time-dependent properties of such systems and exploiting these properties is another potentially useful line of inquiry." Most existing trust metrics update their trust values only after a specific action, like a direct interaction or the reception of a recommendation. The few trust engines that take the time component into consideration simply suggest that the trust value decreases with the time. Mezzetti's trust metric [18] consists in multiplying the trust value at time *t* by a constant between 0 and 1. In Bayesian-based trust metrics [19, 20], the trust value converges to its initial value over time. All these models work in situations where the changes occur slowly, but are challenged in short-lived cases. Unlike the spatial messaging community that seems to be less and less active, the trust community seems to grow and commercial applications are more and more interested in their work. We find for instance some attempts to add trust in Wikipedia articles, like it is presented in a paper from Pierpaolo Dondio, Stephen Barrett, Stefan Weber and Jean-Marc Seigneur [21]. However, we haven't found yet any work on trust in the spatial messaging domain.

2.3. Speed Cameras Warning Systems

As the number of speed cameras increases on European roads, we find more and more services that help the drivers avoiding expensive pictures. We will talk neither about illegal means (for the majority of European countries), like the radar detectors provided by RadarBusters [22], 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 completely legal according to the law of most European countries.

2.3.1. Mogoroad. Mogoroad [23] is a well-known system in Switzerland to announce traffic perturbations, police controls, and of course fixed and mobile speed cameras. It works on most mobile phones. They collect their information from different partners, like radio stations and newspapers, as well as from their own users that can either signal an event by phone or through an application running on mobile phones. There is no trust engine to validate the quality of the data. According to their CEO, Roberto Marra, it is the experience of the employees that collect the data that is used to differentiate useful and correct information from spam. In practice this works quite well since the covered area is small. However, such a system could not easily be extended to work worldwide while providing the same quality of information. The cost of this service is (in 2008) about 110 € per year.

2.3.2. SmartSpeed. SmartSpeed [24] 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 fixed speed cameras than mobile ones.

2.3.3. Coyote. Coyote [25] is an independent system sold as a little box containing a GPS. When the driver approaches a speed camera, Coyote informs 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) that not everybody can afford. And if there are too few users, then the chance that you are the first that discover a speed camera (by being flashed!) is high...

2.3.4. InfoRad. Autonomous and easy to use, InfoRad [26] 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 fixed 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 €.

3. Trusting Virtual Tags

Lots of work has already been done in the trust context (see 2 - "Related Work"), and the question that arises is why not just using well-known trust models and apply them to virtual tags? The answer is simply that it will not work. Indeed, traditional trust models are mainly designed with file sharing or auction applications in mind. In this case, people are rating each other and when user *A* wants to download a file (or buy an item) from user *B*, he questions the system in order to determine how trustworthy user *B* is. Currently, commercial systems (like eBay) are using very basic centralized systems, and the academics are suggesting solutions to transform such systems into peer-to-peer architectures.

But spatial messaging is noticeably different from file sharing or auctioning and needs therefore a new trust model. The key difference is that in spatial messaging it is difficult to increase its own trust without making a significant contribution. For instance, to post a new tag that will be confirmed by others (in order to create a trust link), a user will have to be physically there (to make the observation that deserves a tag). In a similar way, the user that deletes an outdated tag makes also a significant contribution. So, even if a user wants to increase his trust value in order to harm the system later, his former contribution will compensate his future bad behavior. And this is an interesting difference that will be used in this work in order to construct our trust engine. It is, at least to our knowledge, a novelty in the trust domain and can be considered as the key point of this work. In "traditional" trust systems, it is always possible to easily increase one's own trust value in order to subvert the system later. For instance, it is easy to sell honestly a few goods in eBay in order to increase one's trust value. It is also easy to provide a few good files in a file-sharing system and then use the resulting good reputation to send Trojan horses.

But in spatial messaging, a user can increase his trust value only in return of a significant contribution. We will also see in 3.2 - "Updating trust values" how we can make it impossible for a user to switch regularly between good and bad behavior in order to keep a minimum trust value, and how to avoid that a user that behaved correctly for a long time and became malevolent afterwards uses its long-term good reputation to harm the system.

3.1. The Uncertainty of the Truth

In traditional computational trust, we usually agree over a set of axioms and hypothesis. For instance, the "truth" is a notion that is common to all. A corrupted file is seen as corrupted by everybody. In spatial messaging however, the truth is context dependent. The truth becomes a subjective and temporal notion. Something that is true for one user is not necessarily true for the others. Something that is true at a certain time is not necessarily true later. We call this new notion the "uncertainty of the truth". If user *A* posts a tag saying "Dangerous path", user *B* only knows that user *A* finds this path dangerous. But *A* is perhaps just a tourist and the path is in no way dangerous for user *B*, which can be a confirmed mountain guide. Or this path was maybe dangerous because of the snow, which melt away by the time.

To our view, trust is not only a tool that can be used to exclude malevolent users from a given system. Trust is also a way of creating relationships between users that behave in a similar way. Like in real life, each user has its own definition of what the truth is. The aim is therefore to create trust relationships between people that share the same definition.

3.2. Updating Trust Values

A traditional way to store and update a trust value consists in counting the number of positive outcomes *P*, the number of negative outcomes *N*, and to define the current trust value as $T = P / (P + N)$. It is a simple model that fits very well to file sharing applications where a good file is simply considered as a positive outcome and a corrupted file as a negative one. In spatial messaging however, defining a positive and a negative outcome is more complicated. And since we have to deal with what we called previously the "uncertainty of the truth", we need to define a model that is specific for spatial messaging.

A model that can be used in case people are honest is one that uses data mining techniques in order to determine how reliable a given tag is, in a given situation for a given person. Data mining consists in picking up relevant information in large data sets. A good definition can be found at [27]. Basically, when you rate a tag, you increase the trust links with all the people that reviewed it in the same way, and decrease the trust links with all the people that rated it differently. While requesting tags, data mining algorithms are then able to determine how "close" you are with each reviewer according to the situations where you previously interacted with these people, and take this into account to determine how pertinent this tag is to you.

This model is however challenged when malevolent users take part in the system. For instance, an attack would consist in rating automatically and positively all new tags so that the next reviewers increase the malevolent user's trust value. And then this user will use his high value to post "reliable" false tags. A solution to this consists in increasing only the trust value of the author of a tag, since posting randomly interesting tags (if they are not "interesting", nobody will rate them positively) is almost impossible.

In applications where it is possible to scan all the tags, and rate them automatically, it seems easy to cheat the system. It is difficult in some cases to differentiate a normal behavior from a malevolent one. For instance, if you see a tag warning about a specific danger and you do not see this danger, you do not know if the author is a spammer (and you need to decrease his trust value) or if the danger simply disappeared (and then you should not decrease his trust value). We need to determine how much a trust value must be decreased when we rate negatively a tag, so that an honest user is not too much penalized, but so that a spammer can be excluded from the system in a reasonable delay. It means that even if the system is generic, it needs a high comprehension of the application domain in order to determine what are the right rules and parameters. For instance a rule will define how much we must decrease the trust value of someone that doesn't vote like us and a parameter will define what the minimum trust value is.

Like in the human world, trust varies not in the same way when it increases than when it decreases. Trust takes time be built, but can be destroyed very fast. And this non-linear way of handling trust is certainly necessary to protect ourselves. If you lent 10 times 2 € to someone that always paid you back, you will probably stop to trust him before 10 times when he stops refunding you. The reason is even more accentuated in a digital world where people can act in an automatic way, thus very fast. If we use our former $P / (P + N)$ example, it is easy for a user to behave correctly (most probably in an automatic way) for a certain time, and then use its high trust value to subvert the system. A first idea consists in representing a trust value as a single value. A good behavior increases it, a bad behavior decreases it. But the maximal value is limited. It means that even if someone behaves very well for years, his trust value is not that high, and can quickly become negative in case of a big bad behavior, or a succession of a few bad behaviors. Another important point is that trust increases in a linear way but decreases exponentially. An exponential function varies very slowly at the beginning and then increases endlessly. Like in the human model, we accept to forgive seldom and small misbehaviors, but we break our trust relationships if you we face a big misbehavior or a succession a small misbehaviors.

4. A Generic Trust Engine for Virtual Tags

4.1. Overview

The main idea of our generic trust engine, called GenTE, is to remember only important or recent information, like it is done in human communities. The virtual tags (called vTags or simply tag) and the users keep a history of their last or important transactions.

To know whether a tag must be shown to the user, the trust engine checks the *n* last reviews done by trustworthy users. A user is trustworthy if his combined trust value, computed as a mix of the trustor's opinion (based on former direct interactions) and the opinions of the trustor's friends (who ask their own friends, and so on until a certain level), is above a certain threshold. A trustor calls "friend" every user with who he has a good trust relationship, or better said, each user with a local trust value higher than 0.

When a user rates a tag, he updates the trust values of the author and the former reviewers according to rules and parameters that depend on the application. In certain cases, a review can be done on both directions. For instance an author can update the trust value of every reviewer that gives a positive rating, since they seem to share the same opinion about the tag.

4.2. A vTag in GenTE

A vTag contains different information, like its position and its content, as well as a history. The history is a two-column table containing pairs of user ID and corresponding vote. An example is given in figure 1.

ID	Vote
8	0
3	1
4	1
2	1

Figure 1. The history of a vTag

The lines are ordered in an inverse chronological order, meaning that the last user that voted for this tag is user 8. The vote can be either a "1", if the user confirms the tag, or a "0" if the user denies it. So we see that users 2, 4 and 3 agreed with the content of the tag, but later user 8 disagreed with it. The reasons can be either because user 8 is a malevolent user that wants to delete the tag, or, more probably, that the tag is outdated and needs therefore to be removed. If a user that is already in the history votes again for this tag, then his line is moved at the top of the table and the corresponding vote is updated.

When a user requests tags in a given area, the trust engine checks the vote of the two last friends (remember that a friend is someone in which we have a local trust value higher than 0) and if at least one of them voted "1", the tag is sent to the user. It means that even if someone denied the tag by mistake, the tag is still returned to people that are asking for it. This choice implies that we suppose that the price of a false positive (a tag that should not be sent is sent) if lower than the price of a false negative (a tag that should be sent is not sent), which seems to be the case in all the practical applications we thought about.

When the two last users denied the tag (they voted "0"), the tag gets a request-to-delete order. It means that the tag remains for the same amount of time than elapsed since its creation before being deleted by the trust engine. A tag that has been created a long time ago needs therefore more time to be deleted than a recent one. However, to avoid that an "old" tag needs too much time to be deleted we have a maximum delay. And, to avoid that malevolent users scan the network and deny the tags as soon as they appear, we added also a minimum delay. Since then, each new tag is at least present for a certain amount of time (the minimum delay), so even if malevolent users deny these tags, honest users will have time to confirm them (which will cancel the request-to-delete order) and by the same time decrease the trust value of the malevolent deniers.

4.3. A user in GenTE

A user is represented by an ID and a trust table. The trust table is a three-column table containing pairs of user ID and corresponding trust values. We differentiate the AT trust (author trust) which indicates how reliable a given user is to post or to confirm an existing tag and the DT trust (denier trust) which indicates how reliable a given user is to deny tags that are outdated or false. An example is given in figure 2. We see that this user has in his trust table two friends (users 3 and 7), one user he doesn't trust (user 8), and one user in who he has the same trust as for an unknown one (user 13).

ID	AT Trust	DT Trust
3	5	4
7	2	3
8	-3	-4
13	0	0

Figure 2. The trust table of a user

After modifying the trust value of a user, the corresponding line is placed on top of the list, so that there are sorted in an inverse chronological order. Each trust value is simply an integer in the range [t_{min}, t_{max}] so that $t_{min} < 0 < t_{max}$. GenTE allows specifying rules to describe how a trust value must be changed according to a given situation. A typical case is to have a linear way to increase a value (for instance adding *n* when you agree with a tag) and an exponential way to decrease a value (for instance multiplying by *m* a negative trust value). And if $-t_{min}$ is much bigger than t_{max} (for instance t_{min} =-50 and t_{max} =5), then we imitate the human way of handling trust [28]: Trust takes time to be built, we forgive some small misbehaviors (exponential functions moves slowly at the beginning), but when we loose trust in someone (one big disappointment or lots of small disappointments) then it becomes very difficult to rebuild a good trust relationship. We avoid that malevolent users switch between good behaviors (in order to increase their trust value) and bad behaviors (in order to subvert the system).

It is important that our system forgives small mistakes in cases where the truth is unknown. Imagine that a user sees a tag, but the tagged object does not exist anymore. He will disagree with the author of the tag as well as with all the people that agreed. He will therefore decrease their trust values since they are perhaps spammers. But, most likely, the object simply disappeared in the meantime and they are not spammers. Our model is built to forget easily such mistakes, as long as they do not happen too often, but to decrease quickly the trust values of malevolent users. The combined trust value of a user is relative and is computed by the following function:

$$combined\_trust = q * myOpinion + (1-q) * friendsOpinions , \quad q=[0..1]$$

It is a recursive function where *myOpinion* is the local trust value and *friendsOpinions* is the average opinion of the *n* first friends (where local trust > 0). These friends apply the same function, so they return a mix between their own opinion and the average opinion of their own friends. And so on until we reached the specified depth. This way of processing is fast (all the values are centralized) and gives a good idea of the global reputation of a user. Typically, if we choose *n*=10 (number of friends) and a depth level of 3, then we have already the opinion of $10^0 + 10^1 + 10^2 + 10^3 = 1111$ reliable people including ourselves, with more importance given to close friends. The higher is *q*, the more the user gives importance to his own value. In situations where people are susceptible of making mistakes, this value is usually quite small.

4.4. Trust updates

When a user votes for a tag, he puts his ID and his vote at the first line of the tag's history. This newly updated history is then analyzed by the trust engine, and the trust values of the users (that are in the history) are update according to their votes. For instance, if a user votes "1" and the two previous voters voted "0", the confirmer will decrease the trust value of the deniers. And perhaps increase the trust value of the author. The trust engine proposes a default behavior for each situation that can be adapted by the application developer in order to better meet the requirements of his application.

4.5. Rules

The rules that define the trustworthiness of a tag for a given user, as well as the rules that define how the trust values must be updated, are written by the application developer. To test our trust engine, we chose a speed camera warning system and wrote the following rules for a tag request:

History	Rules
Ø (empty)	if I trust the author, return true; return false;
1	if I trust the author, return true; if I trust the confirmer, return true; return false;
1-1	return true;
0	if I trust the author, return true; return false;
0-0	if I trust booth deniers, return false; if I trust the author, return true; return false;
1-0	if I trust the author, return true; if I trust the confirmer, return true; if I trust the denier, return false; return true;
0-1	if I trust the author, return true; if I trust the confirmer, return true; if I trust the denier, return false; return true;

We chose for this case that the size of the history is 2. We therefore keep, for each tag, the author ID as well as the two last votes. For instance, the notation 0-1 means that the last user denied the tag (he voted "0") and the last but one user confirmed it (he voted "1"). If we need to be more precise, we use also the notation $0(U_2)$-$1(U_1)$ meaning that user U_1 confirmed the tag, followed by user U_2 who denied it.

These rules decide whether a given tag must be returned to the requester. We execute the rules one by one until a condition make us to execute a "return true", in which case we return the tag, or a "return false", in which case we do not return the tag.

We then defined also how the trust values must be updated. The next two ables show the current history and shows how the trust tables of the author, the current user and the people in the history are updated according to the current vote ("1" or "0").

To show how we modify the trust values in each case, we define two functions. The first updates the AT trust value and is written like: UAT(U_1, U_2, *a*, *b*, *c*, *d*). It means that U_1 updates the local trust he has in U_2 as following: If the current trust of the trustee is equal or greater than 0, it multiplies the current trust by *a* and adds *b*, and if the trust of the trustee is negative, then it multiplies the current value by *c* and adds *d*. In a similar way, we define UDT(U_1, U_2, *a*, *b*, *c*, *d*) to update the DT trust. Finally we add also two functions, UAT(U_1, U_2, *a*, *b*, *c*, *d*, *C*) and UDT(U_1, U_2, *a*, *b*, *c*, *d*, *C*), where *C* is a specific condition that must be true in order to update the trust.

For instance, if the current user U_c votes 1 and the history is empty, then this user will increase the author's trust value if the condition *C* is met. In our case, *C* returns true only if there are at maximum *N* voters that already voted for this tag.

History	**Rules if vote = 1(U_c)**
Ø (empty)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
1(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
1(U_2)-1(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)

0(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_1, 1, -1, 1.3, -1)

0(U_2)-0(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_1, 1, -3, 2, -3)
	UDT(U_c, U_2, 1, -3, 2, -3)

1(U_2)-0(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_1, 1, -1, 1.3, -1)

0(U_2)-1(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_2, 1, -1, 1.3, -1)

History	**Rules if vote = 0(U_c)**
Ø (empty)	UAT(U_c, U_a, 1, -1, 1.3, -1)

1(U_1)	UAT(U_c, U_a, 1, -1, 1.3, -1)
	UAT(U_c, U_1, 1, -1, 1.3, -1)

1(U_2)-1(U_1)	UAT(U_c, U_a, 1, -1, 1.3, -1)
	UAT(U_c, U_1, 1, -1, 1.3, -1)
	UAT(U_c, U_2, 1, -1, 1.3, -1)

0(U_1)	UAT(U_c, U_a, 1, -1, 1.3, -1)
	UDT(U_c, U_1, 1, 5, 1, 5)
	UDT(U_1, U_c, 1, 5, 1, 5)

0(U_2)-0(U_1)	UAT(U_c, U_a, 1, -1, 1.3, -1)

1(U_2)-0(U_1)	UAT(U_c, U_a, 1, -1, 1.3, -1)
	UAT(U_c, U_2, 1, -1, 1.3, -1)

0(U_2)-1(U_1)	UAT(U_c, U_a, 1, -1, 1.3, -1)
	UAT(U_c, U_1, 1, -1, 1.3, -1)
	UDT(U_c, U_2, 1, 5, 1, 5)
	UDT(U_2, U_c, 1, 5, 1, 5)

4.6. Additional rules

Rule 1: If you are in the first place of the history and you vote the same as previously, do nothing (no trust update and no modification of the history).

Without this rule a single user could delete a tag (by voting twice "0"). However, it is important to note here that this rule mentions explicitly that the two votes are the same. If you vote differently, the trust tables and the history are updated normally. We could thing that if someone votes differently, it was a mistake the first time and we can simply remove the former vote in the history and replace it by the new one. However, this behavior opens the door to a structured attack: The hacker finds a tag whose history is 0(U_2)-1(U_1), and then simply votes alternatively "0" and "1". He first votes "0", so he increases his DT trust with U_2. Then he votes 1, which would erase his last vote, and then he votes again 0, which will again increase his DT trust with U_2. And so on. In short, this would allow anyone to get the maximum DT trust value.

Rule 2: If you are in the first place in the history and voted "0", then the tag is not returned.

This rule avoids that users are disturbed by an object that disappeared. For instance, if a user tagged an object, then you need two different users to give a request-to-delete order to this tag. But if you are the only one that votes for this tag, you will never be able to delete it, and the tag will always be returned to you.

Rule 3: If an author denies his tag, and if the history is either empty or contains a single "0", then the tag is removed immediately.

If you post a tag and nobody sees it, or if you post a tag by mistake and want to remove it, this rule avoids keeping a wrong useless tag. We see also that if the only person that voted for this tag denied it ("0"), then it is a good idea to remove the tag immediately. However we do not remove the tag if the history equals 0-0. The reason is because a malevolent user can set up a structured attack in order to increase his AT trust: He authors a new tag, wait for a while so that people confirming the tag increase his trust value, and then with the help of a friend denies the tag (0-0) and then revokes it. Since the tag disappears, he can post a new one at the same place and again benefit from the trust increases given by the *N* first users that will confirm the new tag.

4.7. Validation process

We chose a speed camera tagging application to validate our trust engine. The first reason is because the topic is quite complex and interesting. Speed cameras can appear and disappear at any time, and it is not always possible to know if a false alarm is due to spammers or if it is actually the speed camera that just disappeared. The second reason is that it was very easy to find volunteers to test our system. We set up a simulator that allowed us to test different scenarios (spammers, users that try to delete all the tags...) as well as a widely deployed application used to confirm the results of the simulator. This application is FoxyTag [10], a worldwide free and collaborative system to signal speed cameras. The idea of FoxyTag consists in posting tags over speed cameras in order to warn the other drivers. Users are also motivated to confirm existing speed cameras; by doing so, they create trust links with the author and the other users that confirmed the camera, allowing them to get more reliable information in the future. More information about FoxyTag can be found on the website of the project [10].

5. Simulator

Our simulator randomly positions speed cameras on a road and simulates user's cars navigating 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. Figure 3 shows a road model.

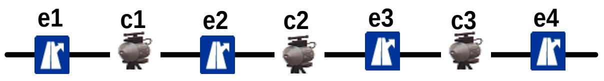

Figure 3. The road model

This model seems to be very simplistic. It is however sufficient to validate our trust metrics. Of course, we do not take into account some contextual information, like shadow areas (tunnels, urban canyons...) or what happens when the user posts a tag for the user driving in the opposite direction. These are more technical issues that need to be validated in the field and it is what we actually did with a real device in a real car. 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 he is meant to cross on average. Our simulator accepts an input file that looks like this:

```
cam;1-4;8;15,10
cam;5-5;24;2,0
cam;5-5;240;3,30
usr;1-10;1-5;24;95;90
usr;1-1;3-5;240;80;75
usr;11-15;1-10;1;10;10
usr;11-11;1-10;0;20;25
col;5-7;1-11;6;10;100
spm;20-23;1-10;1
scn;100;2;run(24);pas(1,10);act(1,10,50,60)
```

- In the first line, "cam;1-4;8;15,10" means that cameras 1 to 4 have one chance out of 8 to become active within an hour, and when one becomes active then it stays active for 15 minutes. After it stays inactive (paused) for at least 10 minutes. Note that these cameras will on average become active less than 3 times a day, since they cannot switch to active while there are already active or paused. Precisely, these cameras will become active every 8+(15+10)/60 = 8.42 hours on average.
- The next two lines define two different behaviors for 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, and 90% of the time correctly when they cancel a camera.
- 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 true positives and 100% of true negatives.
- In the spam line, "spm;20-23;1-10;1", we deduce that users 20 to 23 spam by entering all at the same time on entry 1, exiting on exit 10, and voting 1 about every hour at every speed camera place.
- The scenario, "scn;100;2;..." 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).
- 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 specified behaviors.

- 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, tp, tn) means that our test user will actively drive once from exit e1 to exit e2 and has tp (True Positive) chances (in %) to vote correctly if he sees a speed camera, and tn (True Negative) chances (in %) to vote correctly when he tries to cancel a speed camera that does not exist (anymore). His observations are logged and printed.
- Everything after a // is a comment and is ignored by the parser.

6. Results

We compare here our GenTE trust engine with one called BasicTE, which simply adds a tag when a user posts such a request and remove it when a user denies it (there is in fact no trust engine). This permits to the reader to appreciate the efficiency of the GenTE trust engine. We tested it once with fixed speed cameras (Gen_F), and once with mobile speed cameras (Gen_M). The only difference is that in Gen_M the tags are automatically removed after 6 hours.

Scenario 1

```
cam;1-10;0;9999999;0
usr;1-100;1-11;24;100;100
usr;101-105;1-11;1;0;100
scn;100;100;run(24);act(1,11,100,100)
```

Scn 1	tp	fp	tn	fn
Basic	43030	0	0	56970
Gen_F	99948	0	0	52
Gen_M	92022	0	0	7978

Scenario 1 tests our trust engine when malevolent users try to remove all the tags. We have 10 speed cameras that are always turned on (they are fixed speed cameras), a hundred users that behave always correctly and five users that systematically try to cancel all speed cameras they cross. Each hacker runs on average 24 times more often than an honest user. In the results table we compare the Basic and the GenTE trust engines. We used also the following abbreviations: "tp" means true positives (alarm: yes, camera: yes), "fp" means false positives (alarm: yes, camera: no), "tn" means true negatives (alarm: no, camera: no) and "fn" means false negatives (alarm: no, camera: yes).

With the BasicTE trust engine, we see that there are more false negatives (alarm: no, camera: yes) than true positives (alarm: yes, camera: yes). This is normal since the malevolent users are driving more than the honest ones. But our GenTE trust engine eliminates quite well these malevolent users, since less than 0.06% (52 / 99948) of the speed cameras where not tagged when we mentioned them as fixed ones (Gen_F).

Scenario 2

```
cam;1-10;9999999;0;0
usr;1-100;1-11;24;100;100
spm;101-105;1-11;1
scn;100;100;run(24);act(1,11,100,100)
```

Scn 2	tp	fp	tn	fn
Basic	0	20820	79180	0
Gen_F	0	925	99075	0
Gen_M	0	840	99160	0

Scenario 2 tests how the trust engine reacts against a spam attack. This time the cameras are always turned off and the malevolent users vote "1" for each speed camera position. Again, we observe a significant improvement with our new trust engine.

Scenario 3

```
cam;1-10;48;360;720
usr;1-100;1-11;24;100;100
scn;100;100;run(24);act(1,11,100,100
)
```

Scn 3	tp	fp	tn	fn
Basic	8705	143	90767	385
Gen_F	8759	748	90146	347
Gen_M	8787	245	90619	349

In scenario 3 we have 10 speed cameras that are turned on every 66 hours (48 + (360 + 720) / 60) for 6 hours, and 100 users that vote always correctly. We have of course more false positives since we need two users to remove a tag (against only one in BasicTE). But if we tag the cameras as mobile ones (Gen_M), we observe an interesting improvement for the number of false positives.

Scenario 4

```
cam;1-10;48;360;720
usr;1-100;1-11;24;95;95
scn;100;100;run(24);act(1,11,95,95)
```

Scn 4	tp	fp	tn	fn
Basic	8423	294	90472	811
Gen_F	8806	802	89990	402
Gen_M	8488	277	90856	379

In scenario 4 the users are voting incorrectly 5% of the time. This figure is clearly overrated (according to the tests realized with FoxyTag where this number is less than 1% in practice), but it let us to prove that our trust engine is tolerant with unintentional incorrect votes made by honest users.

Scenario 5
cam;1-10;48;360;720
usr;1-100;1-11;24;100;100
usr;101-105;1-11;1;0;100
scn;100;100;run(24);act(1,11,100,100)

Scn 5	tp	fp	tn	fn
Basic	3845	76	90801	5278
Gen_F	8765	719	90102	414
Gen_M	8761	262	90591	386

In scenario 5 we added 5 deniers that try to remove all the tags they cross. The honest users are behaving correctly 100% of the time. We have clearly more false positives than for the BasicTE trust engine. This is normal since the deniers removed all the tags, whether there is a camera or not. If we compare the results with the ones from scenario 4 (for Gen_M), we see that our trust engine eliminates efficiently deniers.

Scenario 6
cam;1-10;48;360;720
usr;1-100;1-11;24;95;95
usr;101-105;1-11;1;0;100
scn;100;100;run(24);act(1,11,95,95)

Scn 6	tp	fp	tn	fn
Basic	3612	60	91000	5328
Gen_F	8637	795	90109	459
Gen_M	8679	267	90604	450

In scenario 6 the users vote incorrectly 5% of the time. Unfortunately, we observe for Gen_M that the number of false negatives increases (compared to scenario 5). It seems that 5% of incorrect votes is a critical limit for this scenario.

Scenario 7
cam;1-10;48;360;720
usr;1-100;1-11;24;100;100
spm;101-105;1-11;1
scn;100;100;run(24);act(1,11,100,100)

Scn 7	tp	fp	tn	fn
Basic	8781	17824	73124	271
Gen_F	8073	3073	87754	1100
Gen_M	8420	1345	89435	800

In scenario 7 we replaced the deniers by a spammer team, who votes "1" at every speed camera position. The other users are voting correctly 100% of the time. We observe quite bad numbers for GenTE. We first thought of a weakness in our trust engine, but further investigations concluded that it is actually the simulator that presents a weakness. The problem is that the positions of the cameras are always the same (which is not the case in reality), and that sometimes, by chance, a spammer really signal a new speed camera, which generously increases its trust value. In reality this would not be a problem, since signaling randomly a real speed camera at the right place is almost impossible.

Scenario 8
cam;1-10;48;360;720
usr;1-100;1-11;24;95;95
spm;101-105;1-11;1
scn;100;100;run(24);act(1,11,95,95)

Scn 8	tp	fp	tn	fn
Basic	8595	18699	72115	591
Gen_F	7878	3471	87498	1153
Gen_M	8085	1403	89695	817

In scenario 8 the honest users are voting incorrectly 5% of the time. We face the same weakness as in scenario 7. We got therefore a bit worse results, since the honest users are less reliable.

7. Conclusion

This paper presented a generic trust engine to manage virtual tags. We saw that we couldn't simply use existing trust algorithms, since virtual tags have some particularities that need to be handled in a specific way. For instance we faced what we called the "uncertainty of the truth" problem, or how to rate a user if we cannot be sure if he is honest or not. We saw that this situation can happen in presence of an outdated tag. A user that sees a tag about an object or an event that is not present is either victim of a spam attack, in which case he should decrease the trust value of the tag's author, or he simply sees a tag that is outdated, in which case the author shouldn't be too much penalized.

We designed and implemented a trust engine called GenTE, which is able to exclude malevolent users but which is sufficiently tolerant with honest users, even if they do sometimes little mistakes. Since these mistakes are inevitable in spatial messaging (due to the uncertainty of the truth issue but also due to environmental ones, like a tag over a partially hidden object), GenTE is able to forgive small misbehaviors so that frequent users are not penalized.

We personalized GenTE through rules and parameters in order to adapt it for a speed cameras warning system called FoxyTag. We chose FoxyTag to test GenTE because the speed camera topic is quite complex (cameras can appear and disappear at any time, some are partially hidden...), and because it was easy to find volunteers to test our application. We got very promising results.

8. References

[1] Google Earth Community website, visited the 5th of May 2008: http://bbs.keyhole.com/

[2] Google website, visited the 5th of May 2008: http://www.google.com

[3] Tiny tiny blog, visited the 5th of May 2008: http://www.alvafilm.ch/blog/tinytiny/?cat=3

[4] Burrell, Jenna, Gay, Geri K. (2002): E-graffiti: evaluating real-world use of a context-aware system. In Interacting with Computers, 14 (4) p. 301-312

[5] 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)

[6] 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 37(10): 73-81 (2004)

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

[8] R. Guha, "Open Rating Systems", 1st Workshop on Friend of a Friend, Social Networking and the Semantic Web, 2004.

[9] 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.

[10] FoxyTag website, visited the 5th of May: http://www.foxytag.com

[11] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigen-Trust Algorithm for Reputation Management in P2P Networks. 2003.

[12] Prashant Dewan. Peer-to-Peer Reputations. Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04) IEEE.

[13] V. Cahill, et al. Using Trust for Secure Collaboration in Uncertain Environments. IEEE Pervasive Computing Magazine, July-September 2003.

[14] Michael Kinateder, Kurt Rothermel. Architecture and Algorithms for a Distributed Reputation System. 2003.

[15] Aameek Singh, Ling Liu. TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems. Proceedings of the Third International Conference on Peer-to-Peer Computing (P2P'03). IEEE.

[16] Anwitaman Datta, Manfred Hauswirth, Karl Aberer. Beyond "web of trust": Enabling P2P E-commerce. Proceedings of the IEEE International Conference on E-Commerce (CEC'03).

[17] R. Guha, "Open Rating Systems", 1st Workshop on Friend of a Friend, Social Networking and the Semantic Web, 2004.

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

[19] 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.

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

[21] Pierpaolo Dondio and Stephen Barrett and Stefan Weber and Jean Marc Seigneur, Extracting Trust from Domain Analysis, a Study Case on the Wikipedia Project 3rd International Conference on Autonomic and Trusted Computing (ATC 2006) LNCS 4158, Wuhan, China, 2006, L.T. Yang et al., 4158, Lecture Notes in Computer Science, pp. 362--373, sep, Springer-Verlag

[22] Radar buster website, visited the 5th of May: http://www.radarbusters.com/

[23] Mogoroad website, visited the 5th of May: http://www.mogoroad.ch

[24] Smart speed website, visited the 5th of May: http://www.smartspeed.fr/

[25] Coyote website, visited the 5th of May: http://www.moncoyote.com/

[26] Inforad website, visited the 5th of May: http://www.gpsinforad.co.uk/

[27] Data mining according to Wikipedia website, visited the 5th of May 2008: http://en.wikipedia.org/wiki/Data_mining

[28] Book: "Trust Rules: How to Tell the Good Guys from the Bad Guys in Work and Life (Hardcover)", by Linda K. Stroh, Praeger Publishers (August 30, 2007), 184 pages, ISBN: 978-0275998646