# A Node-Centric Reputation Computation Algorithm on Online Social Networks

**JooYoung Lee and Jae C. Oh**

**Abstract** In online social networks, reputations of users (nodes) are emerged and propagated through interactions among the users. These interactions include intrinsic and extrinsic consensus (voting) among neighboring users influenced by the network topology. We introduce an algorithm that considers the degree information of nodes (users) to model how reputations spread within the network. In our algorithm, each node updates reputations about its neighbors by considering the history of interactions and the frequency of the interactions in recent history. The algorithm also captures the phenomena of accuracy of reputations deteriorating over time if interactions have not occurred recently. We present the following two contributions through experiments: (1) We show that an agent's reputation value is influenced by the position of the node in the network and the neighboring topology; and (2) We also show that our algorithm can compute more accurate reputations than existing algorithms especially when the topological information matters. The experiments are conducted in random social networks and Autonomous Systems Network of the Internet. In addition, we show the efficacies of each component in our algorithm and present their effects on the algorithm.

**Keywords** Reputation management · Multi-agent systems · Propagation of information · Trust in autonomous systems

J. Lee (✉) · J.C. Oh
Department of Electrical Engineering and Computer Science Syracuse University, Syracuse 13244, New York
e-mail: j.lee@innopolis.ru, jlee150@syr.edu

J.C. Oh
e-mail: jcoh@ecs.syr.edu



P. Kazienko and N. Chawla (eds.), *Applications of Social Media and Social Network Analysis*, Lecture Notes in Social Networks,
DOI 10.1007/978-3-319-19003-7_1

## 1 Introduction

Accurate reputation information about nodes in social networks improves services provided by the networks. For example, reputations can be calculated and updated for web sites and servers to identify malicious nodes and connections. The ability to observe and analyze propagations of reputations within a large social network structures is also important.

There are centralized and decentralized approaches in reputation computations. Centralized reputation management systems are often used in commercial applications, such as *eBay* and *Amazon*, where a centralized authority is relatively explicit, but such approaches fail in environments that lack a central authority. In online networks such as the Internet, social networks, and other multi-agent systems environments, a distributed reputation computation algorithm is naturally more suitable.

Several distributed reputation algorithms including *AFRAS* [1], *REGRET* [2] and *HISTOS* [3] exist. However, these algorithms do not consider frequencies and *velocity* of interactions; frequency of interactions is an important measure of reputation of the users involved. Velocity of interactions measures the second order information of frequency, i.e., the rate of changes in frequency. Existing algorithms also lack the consideration of topological information of the networks and the subjectivity of reputations (i.e., two different nodes may perceive the reputation of a node differently).

This article presents a new reputation management model that addresses the above issues. Our algorithm considers frequencies and velocity of interactions online. Because our algorithm is developed by modeling the behavior of social networks, we show the algorithm can be used as an analytical tool for studying social network behaviors as well as a query tool for retrieving reputation values for specific nodes at a given time. Through experiments, we show that how reputations emerge and propagate in random social networks and how the model captures the idea of dynamic reputation propagations from one part of the network to another. We also show experiments on real *Autonomous Systems Networks* (*ASN*) of the Internet for identifying malicious ASN nodes through our model. We compare our results with an existing reputation values computed by another well accepted ASN reputation management system. The results show that our algorithm computes similar reputation values as the values provided by the existing algorithm. However, we show that our algorithm is better suited to find many malicious ASN nodes while the compared method is good for finding the worst malicious node only. Finally, we extend the previous work presented in [4] to test the effectiveness of each components in computing reputation values.

The paper is organized as follows. Section 2 presents motivations and contributions of the work. Section 3 presents some of the well known reputation computation algorithms. In Sect. 4 we propose our reputation computation algorithm in detail. Then in Sect. 5, we explain experiment settings as well as the results. Following this, in Sect. 6, we discuss the contributions of the algorithm. Finally, in Sect. 7, we conclude and propose future works.

## 2 Motivations and Contributions

Reputation is an estimate or prediction of an agent's real behavior. Reputation values should dynamically reflect past behaviors of an agent and also tolerate bad behaviors which may have been caused by mistakes. The proposed algorithm has the following uniquely novel features:

- Because reputation is subjective, as it is one's estimate of another, we assume that reputation values of other agents are a private opinion of an agent. Each agent uses a recursive voting algorithm to incorporate opinions of other nodes for a more objective view of a target agent.
- Many social network analysis show that the degree of a node is an important indication of the node's influence as well as the topological information of the networks [5–7]. In the voting process, degrees of neighboring nodes are used to compute the weight of the votes. In this article, we assume that each node knows the degrees of its neighbors and the total number of nodes in the network. In many cases, including *HISTOS* [3], the topology of the network is assumed to be known to enable aggregations of reputation.
- In many times, how frequently interactions occurs within a limited period can be an important evidence about reputations of the nodes. The algorithm incorporates frequencies of interactions as well as velocity. Existing distributed algorithms usually only consider the total number of interactions.
- If two agents have not interacted for a long time, even if the total number of interactions is large, the reputation values for each other many not be up to date. Our algorithm applies a time decaying function to balance the previous reputation value and the evaluation of the most recent interaction.

## 3 Related Work

In this Section, we introduce some of the most well known reputation management algorithms, *AFRAS* [1], *REGRET* [2] and *HISTOS* [3]. According to the classifications discussed in [8], *AFRAS*, *REGRET* and *HISTOS* are distributed algorithms and combine direct interactions as well as the witness information to determine reputation. *HISTOS* considers only a given number of recent interactions and recursively aggregates reputation values following paths to the target agent. It also weights opinions from others with their reputation values. *AFRAS* uses fuzzy sets to represent reputation with truth levels. It combines the previous reputation and the current satisfaction level to compute a new reputation. It introduces a dynamic memory factor as a weight, updated with the similarity between the previous reputation and the current satisfaction level. *REGRET* uses the three dimensionality of reputation, namely *individual, social* and *ontological* dimensions. The *individual* dimension computes reputations based on direct experiences while *social* dimension col-

lects witness information. In addition, the *ontological* dimension adds possibility of combining different aspects of reputation.

The common idea behind the algorithms discussed above is that updating reputation involves weighted aggregation of the previous reputation and current feedbacks. The previous reputation is weighted with a factor of time or memory, and current feedbacks are computed through evaluation methods. The difference lies on how the algorithms interpret current interactions with respect to the given information about the target agent. In addition to that, we incorporate network degree information to measure the weight of the node's own opinion about the target node. For example, an agent gives more weight to its own evaluation than opinions from others if it has a relatively small number of neighbors in a network while it assigns more weight on others' opinions if it has more connections.

The reputation function in *HISTOS* algorithm focuses on estimating reputation values when an agent asks for reputation of another at distant, meaning it recursively aggregates ratings given by neighbors following the shortest paths. If the agent is directly connected to the target agent, it doesn't need to combine ratings from other nodes. On the other hand, our algorithm is more focused on updating reputations after interactions. After having a direct interaction with a target agent, agents still combines their direct experience with indirect experiences (through voting). This enables agents to keep personalized reputations of other agents that are not biased by unfortunate wrong impressions.

Other reputation systems include but not limited to *FIRE* [9] which identifies dishonest and mistaken agents, *TRAVOS* [10] which attempts to determine the credibility of witnesses when combining opinions and *PeerTRUST* [11] which addresses the bootstrapping problem in peer-to-peer systems.

## 4 ReMSA: Reputation Management for Social Agents

In this Section, we present the proposed algorithm—*ReMSA: Reputation Management for Social Agents* [4]. A reputation value is computed when an interaction between two agents occurs. In an interaction, an agent can be an observer, observee, or both. After an interaction, the observer evaluates the interaction to compute the reputation of the observee. If both are observers, they will compute reputations of each other. The more interactions occur between two agents, the more accurate the reputations are computed for the agents. As interactions occur within the network over time, reputations of agents in one part of the network will propagate to another part of the network, much similar to what is happening in real-world social networks. At any given time, any agent can query about reputation of an arbitrary agent. Note that the returned value to the agent may be different from the result of the query initiated by another agent. When there's enough interactions among overall agents in the network, reputations will propagate to the entire network, and more homogeneous views of reputations of agent will emerge.

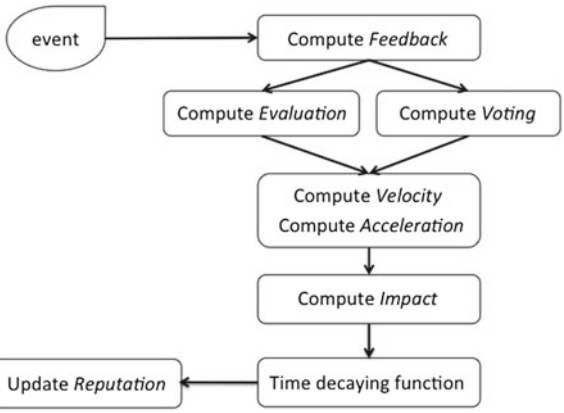


**Fig. 1** The sequence of processes agents take to update reputation values

Figure 1 shows the flowchart of reputation computation for an observer node when an event occurs. Following subsections explain each process in Fig. 1.

## *4.1 Events*

We define an event as an interaction between two agents with time information. There are two types of events in terms of who owns the event. When an interaction happens between two agents, if both agents can observe each other's behavior, then both own the event. If only one of the agents can observe the other's behavior, only the observer is the owner of the event. All the following computations are based on the observer agent's point of view.

- The set of agents and events are defined as follows.

$$A = \{a_1, a_2, ..., a_n\}$$

$$E_i = \{e_1, e_2, ..., e_m\}$$

$$e_j = (t_j, a_j)$$

  where, $a_i$ is an observer agent, $E_i$ is a set of events that $a_i$ as an observer, and $e_j$ is an event which consists of its time, $t_j$, and the observee agent, $a_j$.
- Given a network, we define $\alpha$ and $\beta$ where nodes are agents and edges are relationships.

$$\alpha_i = \frac{d_i}{\max_{m \in A}\{d_m\}}$$

$$\beta_{il} = \frac{d_l}{\sum_{k \in N_i} d_k}$$

where, $N_i$ is a set of $i$'s neighboring agents, and $d_a$ is the degree of agent $a$. $\alpha_i$ is a ratio of $i$'s degree and the maximum degree in the network. $\alpha_i$ is used to weight $i$'s opinion (i.e., the observer's) since $\alpha_i$ represents $i$'s position in the network. It implies that we can infer an agent's confidence from its relative degree information. Given $i$ and it's neighbor $l$, $\beta_{il}$ is a ratio of $l$'s degree and the sum of $i$'s neighbors' degrees. $\beta_{il}$ represents $l$'s relative credibility from $i$'s perspective. $i$ will use $\beta_{il}$ to collect opinions from its neighbors. The neighbors of each $l$ will recursively compute $\beta_{l*}$ in turn until one of the three terminating conditions is met as explained in Sect. 4.2.3. In the voting algorithm shown in (1), $i$ evaluates voting weights for each of it's neighbor $l$.

## *4.2 Compute Feedback*

*Feedback* process consists of two subprocesses, *Evaluation* and *Voting*. *Feedback* is a part of reputation updating function in Sect. 4.6.

### 4.2.1 Compute Evaluation

After each interaction, agents that were involved in the interaction evaluate the interaction according to their own evaluation methods. Therefore, *Evaluation* of interactions is subjective and can be implemented depending on applications. $Evaluation$ of an event $e$ is represented as a function, $Eval(e)$.

### 4.2.2 Compute Voting

While the evaluation function is computed by each agent, agents collect opinions from other agents before updating the reputation of the target agent through a voting process to combine diverse opinions. If $a_i$ had an interaction with $a_l$, $a_i$ can take a vote about $a_l$ to its neighbors to obtain more objective views. The neighbors of $a_i$ can either return a vote to $a_i$ with their own evaluations (if they don't have neighbors other than $a_i$ or $a_l$) or they can spread the vote to their neighbors. $V^e_{a_i a_l}$ is the weighted sum of voting results and we define it as follows.

$$V^e_{il} = \sum_{k \in N_i} \beta_{ik} \times F^e_{kl} \tag{1}$$

$\beta_{ik}$ represents $i$'s delegation trust towards $k$ and is multiplied by $F^e_{kl}$ which is a weighted sum of $i$'s evaluation of $l$ on the event $e$ and collected feedbacks from

$k$'s neighbors about $l$. $Feedback$ process is represented with the function $F^e_{il}$ which recursively calls the voting function, $V^e_{il}$.

$$F^e_{il} = \alpha_i \times Eval_i(e) + (1 - \alpha_i) \times V^e_{il} \tag{2}$$

$\alpha_i$ implies self-confidence of $i$ and it is multiplied by $Eval_i(e)$, the self evaluation on event $e$.

As shown in formula (1) and (2), *Feedback* and *Voting* processes are defined recursively.

#### 4.2.3 Stoping Criteria

To avoid infinite loops or circular voting processes, we need to specify a few restrictions. First, when an agent takes a vote to its neighbors, it excludes itself. Since the agent's opinion about the target agent is already included in the evaluation function, it only needs to hear from its neighbors. Second, for the voters to avoid casting duplicate votes, each agent keeps history of votes which it has already participated. This is beneficial to the voters so that they don't waste their own resources on duplicate votes. Third, the base condition of the recursive voting is: (1) when an agent has only one neighbor which originated the voting process, (2) when an agent has two neighbors one being the originator and the other being the target agent and (3) when an agent has already participated in the current vote. In the first two cases, (1) and (2), the voter agent returns its reputation value of the target agent.

### *4.3 Compute Velocity and Acceleration*

We also consider the frequency of events to compute reputation since an event with a dormant period should be treated differently from frequent ones. We define the velocity for each event to compute the acceleration of events. Then the acceleration of an event influences *Feedback* value of the event through the *Impact function*.

Velocity of an event is defined as follows.

$$Vel(e) = \frac{1}{t_e - t_{e'}} \tag{4}$$

where $e'$ is the most recent previous event.

It is obvious that there needs to be at least two events to compute a velocity, otherwise the velocity is zero. Also, since we consider time with increasing positive integers, $t_e - t'_e > 0$ and $Vel(e) \in [0, 1]$.

Now, we can compute the acceleration of event $e$ to identify if its velocity is increasing or decreasing through $Acc(e)$.

$$Acc(e) = Vel(e) - Vel(e') \tag{5}$$

### 4.4 Compute Impact

We introduce *Impact function* to calculate the influence of $Acc(e)$, defined in (5), on the feedback, $F^e$.

$$I(Acc(e), F^e) = |Acc(e)| \times F^{e\,3^{\frac{-Acc(e)}{|Acc(e)|}}} + (1 - |Acc(e)|) \times F^e \tag{6}$$

The magnitude of $Acc(e)$ determines the curve of the function which decides how much to increase or decrease from $F^e$. When $Acc(e) > 0$, $I$ increases the original feedback value, $I(Acc(e), F^e) > F^e$, and when $Acc(e) < 0$, $I$ decreases the original feedback value, $I(Acc(e), F^e) < F^e$. If $Acc(e) = 0$ then $I(Acc(e), F^e) = F^e$ which means that when there is no change in the velocity, no impact is made to the feedback value, $F^e$.

### 4.5 Time Decaying Function

Time decaying function is an essential part of the reputation computation since it captures the temporal nature of information; old reputation value may not be as accurate as a new one. Intuitively, an interaction shortly after the previous one can make more use of the built-up reputation (current reputation) while an interaction after a long inactive period should rely more on the current feedback values since the built-up reputation is not up to date. As discussed in [12], time decaying function should to be designed carefully, based on the context (e.g. a periodic function) so that it can adjust time sensitivity weights when computing reputations.

We use an exponential decay function to capture the idea. Our time decaying function relies on the elapsed time since the last interaction.

$$D(x) = e^{-x}$$

where $x$ is $t_e - t_{e'}$.

### 4.6 Update Reputation

Finally, we are now ready to explain the reputation update function that utilizes the functions discussed so far. A new reputation value is computed when a new event occurs. The new reputation is a weighted sum of the current reputation and the feedbacks. The current reputation is weighted by the time decaying function and *Impact function* is applied to the feedbacks. Finally, we formally define the reputation update function as follows.

**Reputation update function**:

$$R_{il}^{t_e} = d \times R_{il}^{t_{e'}} + (1 - d) \times I(Acc(e), F_{il}^{e})$$

where $d = D(t_e - t_{e'})$.

### 4.7 Ask Function

In our algorithm, each agent keeps a list of reputation values of the neighbors. However, in some occasions, an agent might wonder about another agent's reputation other than the neighbors. Therefore, we implement *Ask function* to query a target agent's reputation who is not a neighbor. *Ask function* is the same as *Feedback* function except the agent does not have its own evaluation of the target agent. *Ask function*, then, goes through the same processes as in *Voting* function as in (1).

$$Ask_{il} = \begin{cases} R_{kl} & l \in N_k \\ \sum_{k \in N_i} \beta_{ik} \times Ask_{kl} & \text{otherwise} \end{cases}$$

## 5 Experiments

In this section, we present two sets of experiments. First, we compute reputations of Autonomous Systems (AS) in the Internet using our algorithm. And we compare our results with *AS-CRED* [13], which is a well-known reputation service for AS network. We use subsets of the real AS networks obtained from *RouteViews* [14]. Second, we study the emergence and propagation of reputations within random social networks generated by Graph-Generator [15]. Graph-Generator is a small Java program to create random graphs in which the number of nodes, edges and the maximum degree are specified.

## 5.1 Experiment 1: Computing Reputations of Autonomous Systems

We apply *ReMSA* algorithm to compute reputation values of Autonomous Systems. An Autonomous System is a collection of connected Internet Protocol (IP) routing prefixes under the control of one or more network operators. Since behaviors of each AS represent human interests, we consider AS network as social network. We analyze Border Gateway Protocol (BGP) updates data, which is the standard communication protocol for interconnecting ASes, to evaluate validity of activities among ASes in the network and compute reputation values based on the evaluations. The reputations of ASes could be incorporated for the routes deciding algorithm for each AS since reputations of ASes directly represent the behaviors of ASes.

### 5.1.1 Network Sampling Algorithm

As shown in Table 1, the number of nodes in the original AS network is very large because it represents all ASes in the Internet. However, only less than 10 % of ASes appear in each day's BGP update data we use. Therefore, for the tractability of the experiments, we extract two representative, scaled down, sub-networks from the original AS network. If we sample sub-networks from the original AS network using existing algorithms, most of the ASes in sampled sub-networks don't appear in the BGP data. Instead of using the algorithms discussed in [16], we designed a context-based network extraction algorithm in order to sample meaningful sub-networks which contain most of the autonomous systems appearing in BGP update data. Therefore, we extract ASes that appeared in the BGP data so that we can compute reputations of the ASes. For each randomly chosen *ASPATH* in BGP update data on January 1, 2010, we add ASes which appear in the *ASPATH* to the sub-network and repeat the process until the desired number of nodes for the sub-network is reached (in this case, 5,000).

In order to measure whether the sub-networks represent the original AS network reasonably, we evaluate the sub-networks by the metrics defined in [16].

Table 1 shows the number of nodes and edges of the original network and two sampled sub-networks. The number of nodes of sub-networks (5,000) is approximately 15 % of the real network (33,508) which is enough to match the properties shown in Table 2 [16]. Table 2 shows five different distributions of two sample networks measured using Kolmogorov-Smirnov D-statistics. *D*-statistic measures the agreement

**Table 1** Nodes and edges information of networks

| | Original | Sub-network1 | Sub-network2 |
|---|---|---|---|
| # Nodes | 33,508 | 5,000 | 5,000 |
| # Edges | 75,001 | 19,953 | 22,379 |

**Table 2** Sampling criteria for sub-networks

| | In-deg | Hops | Sng-val | Sng-vec | Clust | AVG |
|---|---|---|---|---|---|---|
| Sub-network1 | 0.2743 | 0.3500 | 0.1883 | 0.1180 | 0.2346 | 0.2399 |
| Sub-network2 | 0.0703 | 0.3500 | 0.1234 | 0.0357 | 0.1944 | 0.1547 |

between the true and the sample network property [16]. In the last column, we show the average of the results. Lower average values means more agreement between original network and the sub-network. Some good average values discussed in [16] were 0.202 and 0.205. Therefore, both of the sub-networks qualify for scaled-down sub-networks well representing the original.

### 5.1.2 Evaluation of BGP

We use BGP (Border Gateway Protocol) update data from *RouteViews* [14] dated from January, 2010. Also, we use the same analysis of AS-behavior discussed in [13] and [17] to compute feedbacks of BGP activities in order to compare our reputation computation results with *AS-CRED*. In order to use our algorithm, we need to define *events* and associated *observer* and *observee* in the problem domain of AS reputation computation. In BGP update data, for each update message sent from say, $AS0$ to $AS1$, $AS1$ analyzes the message as an observer and evaluates $AS0$'s behavior. Such a message is an event as we defined in Sect. 4.1. And an event can be a non-empty subset of the set {*AS-Prefix behavior, AS-Path behavior, AS-Link behavior*}, which represents the *observee's* behaviors. Each behavior can be evaluated to be positive or negative and then the result of the evaluation is accumulated for that message. In other words, each message will have an associated score representing the behavior of the *observee.* We describe how each behavior is evaluated below.

- *AS-Prefix behavior*: For the observee AS and its prefix $p$, we compute two temporal metrics, *persistence and prevalence.* These two metrics can represent positive or negative behavior of the observee AS. We will not discuss the details of the metrics because they are beyond the scope of this paper. The value of the persistence and prevalence are compared against a set of thresholds mentioned in [17] and feedback is provided. For good behaviors, evaluation of 1 is provided and otherwise $-1$.
- *AS-Path behavior*: We use AS relationship data from [18] to evaluate the valley free property of AS $paths$. None of the ASes in the AS $path$ should form a valley. The observee AS provides an AS-Path. If a valley is found in the provided AS-Path and the first AS forming the valley is the observee, it gets evaluated by its observer with $-1$.
- *AS-Link behavior*: For each link in the AS-Path provided by the observee, we compute persistence and prevalence values, then these are compared with the threshold

discussed in [17]. If the classification of the behavior is good, an evaluation of 1 is provided, otherwise the link is unstable, therefore, the evaluation is −1.

### 5.1.3 Results

We compute reputations for each AS appeared in BGP update data for each day between January 1, 2010 and January 31, 2010. We average the reputations computed by *ReMSA* with two different sub-networks. We exclude ASes with good behaviors (positive reputation values) to compare the result with *AS-CRED* which accumulates reputation values when bad behavior occurs (zero is considered the best reputation in *AS-CRED*).

In Fig. 2, we show the distribution of reputation values of ASes computed by *AS-CRED* for January 1, 2010. In Fig. 3, we show the distribution of reputation values of ASes compute by *ReMSA* for the same day. The ranges of reputation values shown in the figures are different as they represent the raw reputation values computed by *AS-CRED* and *ReMSA*. Since *AS-CRED* computes centralized reputation values, we averaged reputation values computed for each AS by *ReMSA*. The purpose of each algorithm is implied by the distribution of reputation values shown in the figures. *AS-CRED* computes reputation values of ASes to detect globally malicious ASes while *ReMSA* computes distributed reputation values for each AS to measure the trustworthiness of relationships between neighbors. Therefore, *ReMSA* allows each AS to have its private perception of its neighbors based on the history of interactions and the witness information rather than to rely on global computed values.

Figure 4 shows average reputation values of ASes computed by *AS-CRED* and *ReMSA* over one month. The non-zero reputation values of ASes are added and averaged from our sub-networks. Similarly, the reputation values of respective nodes from *AS-CRED* were averaged. We normalized values computed from *AS-CRED*

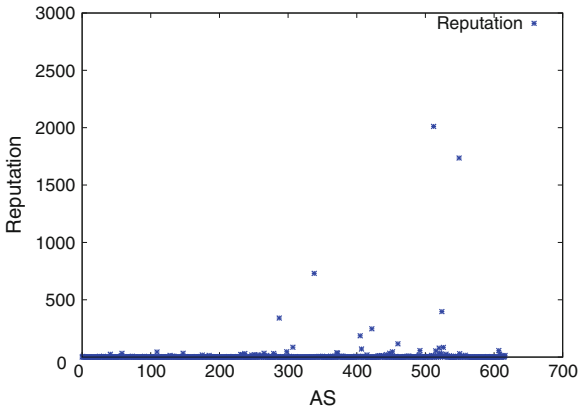


**Fig. 2** Reputations computed by *AS-CRED*

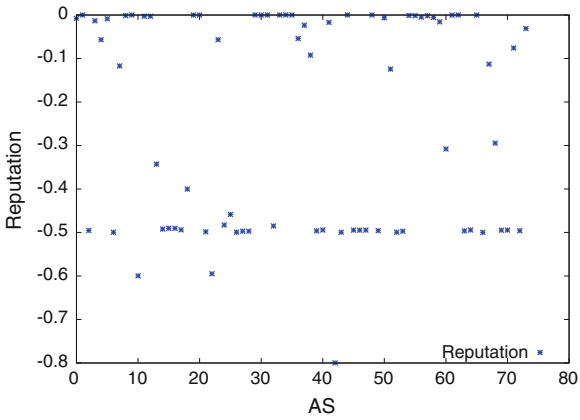


**Fig. 3** Reputations computed by *ReMSA*

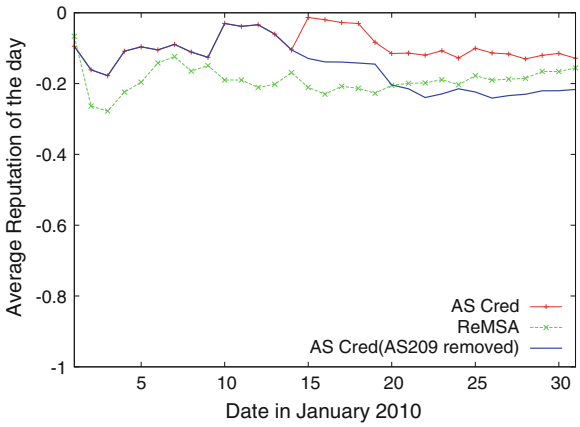


**Fig. 4** Reputations computed by *AS-CRED* and *ReMSA*

since zero is the best reputation value and the higher values represent the worse reputation in their algorithm.

The two lines from *AS-CRED* differs in that the one below doesn't include *AS209*, which has an extremely bad reputation with the raw value 2755.52. We investigated the differences shown in Fig. 4 and found out that whenever there are big gaps, e.g., on the 14th day, there was an AS with extremely bad reputation (i.e. $AS209$) in *AS-CRED*. Therefore, when we normalized the reputation values, since the worst reputation value in *AS-CRED* becomes $-1$, it makes other reputation values negligibly small. Consequently, the normalized average for *AS-CRED* is smaller than our algorithm's average. For example, on the 14th day, $AS209$ received an extremely bad reputation (the raw value 2755.52) when most of other ASes received less than 10. Such a huge difference among reputation values makes other reputation values negligible which enables moderately malicious ASes to get hidden under an extremely malicious AS.

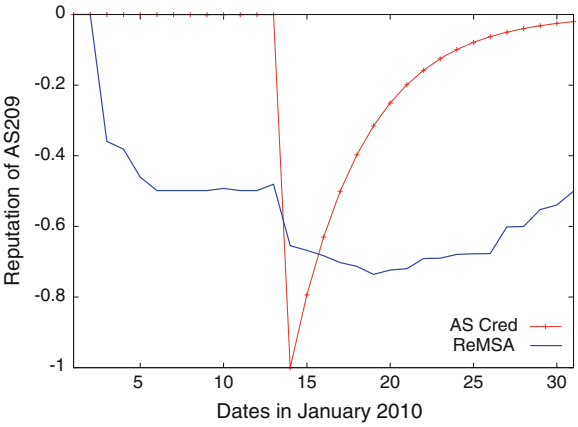


**Fig. 5** Reputation values computed by *AS-CRED* and *ReMSA* for *AS*209

Now let's observe *AS209* more closely. Figure 5 shows the reputation value of *AS*209 computed by *AS-CRED* and our algorithm. In the figure, we normalized reputation values from *AS-CRED* with the largest value *AS*209 had, which was on January 14th, 2010. *AS*209 had bad behaviors before the 14th, but because of the huge variances among the reputation values over time in *AS-CRED*, the reputation values of *AS*209 on other days except the 14th became almost zero after normalization. *AS-CRED* may be useful to identify the most malicious AS, but it may lose other important information such as how reputation value changes over time.

## 5.2 *Experiment 2: Propagation of Reputation on Random Networks*

In addition to the performance evaluations on a real social network (ASN) presented in Sect. 5.1, we test the propagation of reputation values in random networks. We study a *sparse* and a*dense* network in order to show how topology of the networks (degrees of nodes) influence propagation of information (reputation values). Table 3 shows the statistics of the two random networks.

For each network, we pick an observee node, $a_0$, and observe how reputation computed by its neighbors change over time. We also pick two observers distant from $a_0$, say $A$ and $B$, in order to show $a_0$'s reputation values obtained by each observer. Note that each observer will obtain a subjective reputation value about $a_0$. We generated random events that are associated with time information, an observee node and the evaluation of event. Each node has behavioral probability (positive or negative) and the observer evaluates behaviors of observee nodes.

The straight line in Fig. 6 shows the true behavior of $a_0$ based on its behavioral probability, $p_{a_0}$. We define $p_{a_0}$ as the probability that $a_0$'s behavior is evaluated to 1.

**Table 3** Statistics of two random networks

| | Sparse network | Dense network |
|---|---|---|
| # Nodes | 5,000 | 5,000 |
| # Edges | 10,000 | 50,000 |
| Average degree | 4 | 20 |
| Density | 0.001 | 0.004 |
| Diameter | 11 | 4 |
| Average path length | 6.624 | 3.129 |

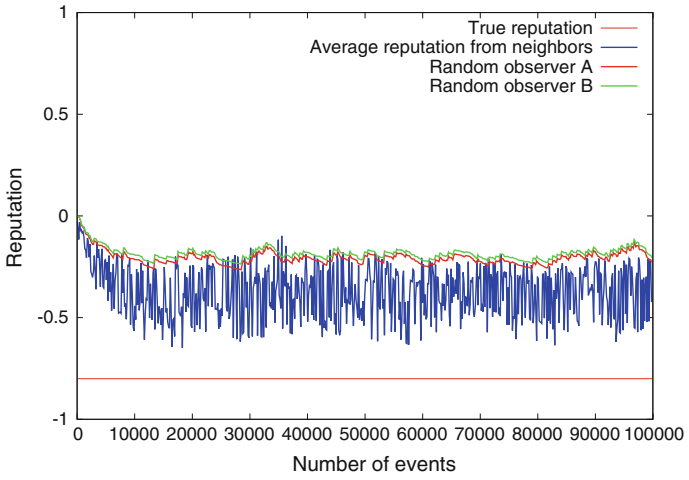


**Fig. 6** Propagation of reputation in a sparse network

In this case, the probability was 0.1 and therefore the true reputation (true reputation $= 2 * p_{a_0} - 1$) is $-0.8$ since the reputation value is between $-1$ (when $p_{a_0} = 0$) and 1(when $p_{a_0} = 1$). The neighbors have interactions with $a_0$ and update reputations based on the probabilistic random behaviors of $a_0$. The average reputation values of $a_0$ computed by the neighbors lie right above the true reputation value in Fig. 6. The two other lines on top represent the reputations values seen by $A$ and $B$. For each time the neighbors' reputation values of $a_0$ are computed, $A$ and $B$ query reputation of $a_0$ using $Ask$ function presented in Sect. 4.7. As we can see in Fig. 6, it is not hard to believe that direct interactions with $a_0$ help compute more accurate reputation values of $a_0$ compared to the reputation values received only by collecting opinions from other nodes. Also we can see that the changes in reputation values become more stable as nodes learn $a_0$'s true behaviors and share subjective reputation values of $a_0$ through voting processes. Since $A$ is 4-hop-away from $a_0$ and $B$ is 6-hop-away form $a_0$, we can see that $A$ has closer values to the true value than $B$.

We repeat the process on the dense network and the results are shown in Fig. 7. We set the behavioral probability of the observee, say $a_1$, the same. $A$ and $B$ both were

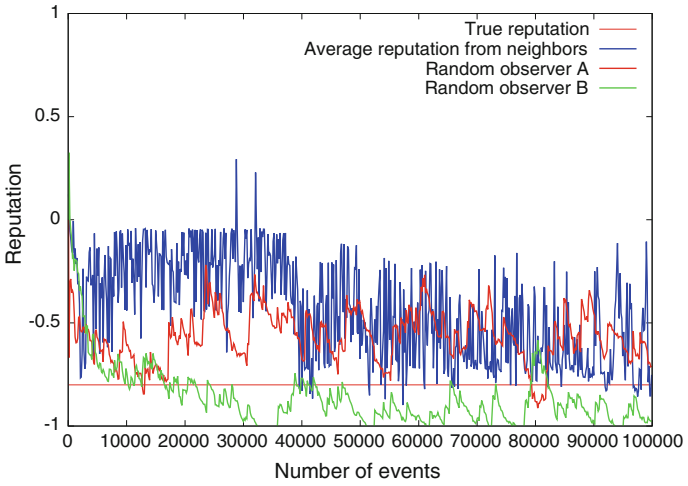


**Fig. 7** Propagation of reputation in a dense network

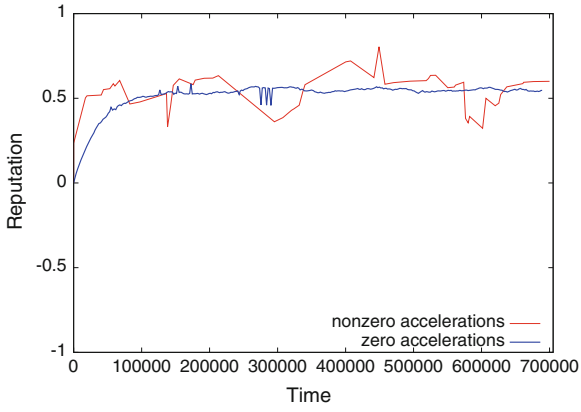


**Fig. 8** Reputations computed with constant and increasing velocity

3-hop-away from $a_1$. The average reputation values computed by $a_1$'s neighbors converge closer to the true value. On the dense network, reputation values seen by the two observers fluctuate because, as shown in Table 3, the network is so dense and the reputation of the target agent is influenced by many different agents through *Ask Function*.

In Fig. 8, we show how velocity of events influence reputation values. As discussed in Sect. 4.4, the *Impact* function adjusts the computed feedback values based on the acceleration of the event. On a random network, we pick a node with behavioral probability 0.8 and observe reputation values from a neighbor changing over time when the velocity of interactions is constant and when the velocity of interactions increases. As we can see in Fig. 8, the reputation computed without acceleration becomes stable as it reaches close to the node's true reputation, 0.6, while the reputa-

tion computed with nonzero accelerations fluctuates more. Since the *Impact function* emphasizes the influence of accelerations of events, the reputation values become more sensitive to the current feedbacks when the rate of events is more frequent.

## 5.3 *Effects of Voting, Time Decaying Function and Impact Function*

In this Section, we show the effect of each mechanism in the reputation update formula, introduced in Sect. 4.6. We create a scale-free network using *Albert-Barabási* algorithm with 5,000 nodes and 50,000 edges. This network was used in the following experiments. We discuss the results of experiments using a sample agent picked randomly which represents the typical behavior of agents in general. The reputation values are computed from the neighbors of the sample agent.

### 5.3.1 Effects of Voting

Voting process is introduced in Sect. 4.2.2. Through the voting, one can aggregate others' opinion so that the computed reputation values are objective. The balance between direct experiences (self opinion) and indirect experiences (others' opinions) is automatically controlled by the degree of each agent as discussed in Sect. 4.2.2.

Figure 9 shows the reputation computed with and without the voting process which means that the *feedback* function is replaced by *Evaluation* (self opinion) only. The straight line is the true reputation of an agent. The reputation values computed with

**Fig. 9** Effects of Voting

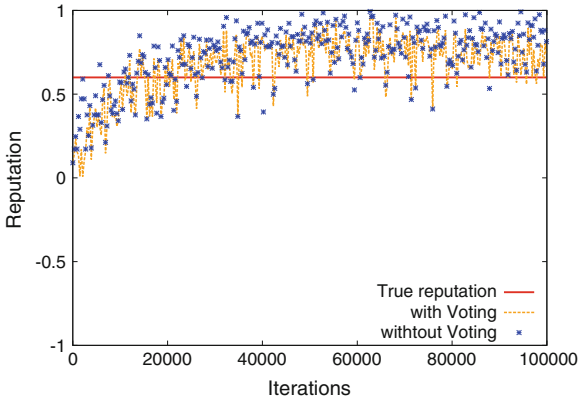

**Table 4** Average distance from the true reputation

| Node | 1453 | 4999 | 3387 | 4102 |
|---|---|---|---|---|
| Degree | 10 | 11 | 33 | 57 |
| With voting | 0.247 | 0.187 | 0.156 | 0.142 |
| Without voting | 0.311 | 0.291 | 0.234 | 0.175 |

voting, which considers others' opinions, are closer to the true reputation value compared to the reputation values computed without voting. Table 4 shows the average distance from the true reputation value over the iterations for four sample nodes. The average distance from the true reputation is lower when the reputation values are computed with voting. We also observe that as the degrees of node increases, the average distance from the true reputation decreases since having more neighbors lead to getting more opinions.

### 5.3.2 Effects of Time Decaying Function

The time decaying function, discussed in Sect. 4.5, utilizes the frequencies of interactions to balance the current reputation and the feedbacks. In some applications, the steady-state reputation is more valuable , while in other cases, reputation values need to be adaptive so that they reflect the up-to-date information. In *ReMSA*, this is automatically controlled by the time decaying function.

Figure 10 shows four different reputation values. The straight line shows the true reputation value of the agent under observation. The line shows reputation values computed using standard *ReMSA*. We also show steady-state reputation values as well as adaptive reputation values, plotted with square points and star points respectively.

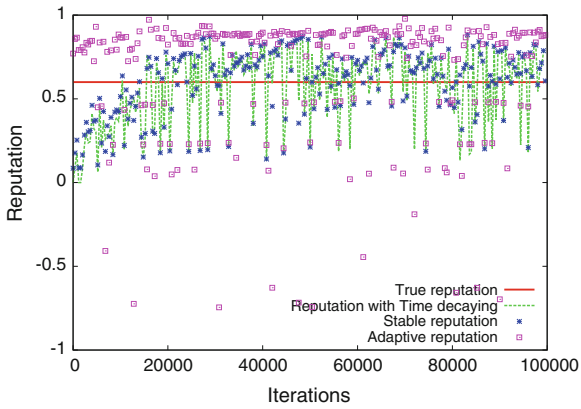


**Fig. 10** Effects of time decaying function

As discussed before, new reputation is computed as a weighted sum of current reputation and new feedback. Steady-state reputation has more weight on current reputation while adaptive reputation has more weight on new feedback. For the comparison, we weight current reputation with 0.9 and new feedback with 0.1 for steady-state reputation and vice versa for adaptive reputation. Intuitively, if the current reputation is weighted more than the new feedback, the reputation value is stable meaning it does not fluctuate much. On the other hand, if the new feedback is weighted more than the current reputation, the reputation value is more adaptive since the updated reputation value reflects more of the current behavior of an agent than the history of the past behaviors. As shown in the Fig. 10, adaptive reputation values are distributed near 1, 0, or $-1$, which are the raw feedback values. Steady-state reputation values are distributed near reputation values computed by standard *ReMSA*; this is because the interactions are randomly generated and the frequencies of interactions don't vary too much.

### 5.3.3 Effects of Impact Function

The purpose of *Impact function* is to reflect accelerations of repeated interactions to the *feedback*. In real social networks, repeated interactions in a short period of time may imply greater closeness of two entities involved. Therefore, we emphasize sudden increases of interactions rate using *Impact function*. In Fig. 11a, reputation values of interaction with increasing accelerations are shown and in (b), reputation values of interactions with decreasing accelerations are shown. Since the time information of the interactions are randomly generated integers, the acceleration values are small. For example, if three interactions with time 10, 20, 25 occurs, the velocity of the second and third interactions are 0.1 and 0.2 and the acceleration of the third interaction is 0.1. Therefore, even the acceleration of the third interaction is positive and emphasized by *Impact function*, the difference is not big. With positive accelerations, positive *feedback* values are rewarded and negative *feedback* values are punished according to the acceleration. In Fig. 11a, reputation values below the red line show that reputation values computed with *Impact function* are lower than the ones computed without *Impact function*. Since the acceleration is positive, negative feedbacks were punished (decreased) by *Impact function*. On the other hand, in Fig. 11b, reputation values of interactions with negative accelerations are shown. Reputation values below the red line shows that reputation values computed with *Impact function* are higher than the ones computed without *Impact function* since negative *feedbacks* with decreasing accelerations are rewarded (increased) according to the acceleration values.

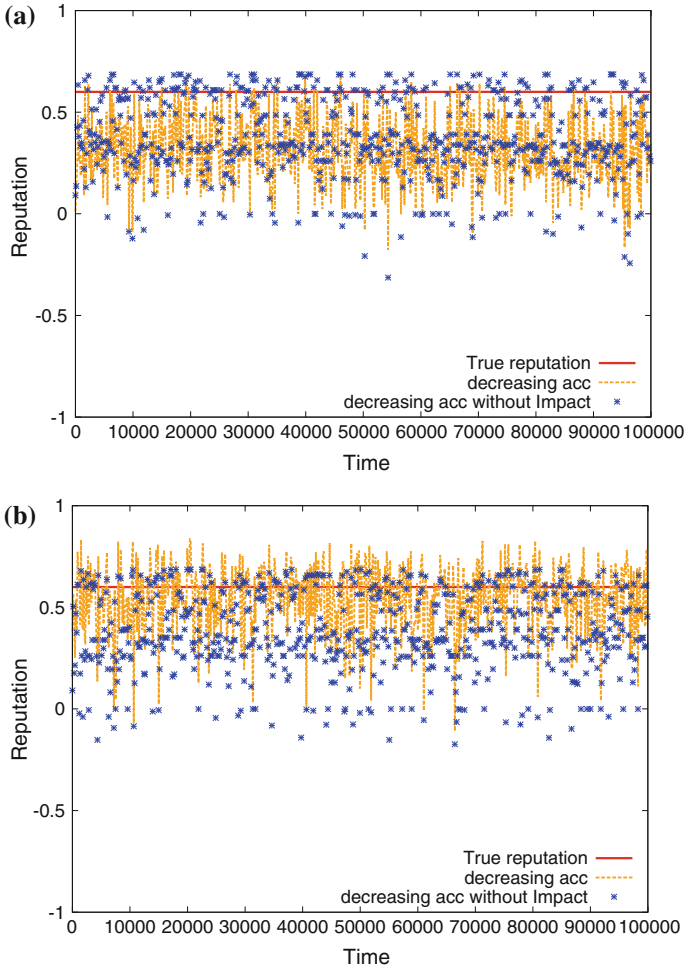


**Fig. 11** Effects of Impact function.

## 6 Discussions

A natural way of measuring an agent's reputation is to accumulate interaction history in some form of weighted sum. We started with this idea and incorporated frequency and velocity of interactions as well as the topological information of the network. Generally, reputation computation takes an approach that combines the previous reputation and new experience.

Our algorithm assumes that each agent is aware of its neighbors and its position in the network and makes use of the information. Therefore if the topology of network changes, an agent perceives its new sociological information and its reputation

updating function changes accordingly. This is explained by human behavior; people tend to act with more responsibly when there are more observers (i.e., a higher degree). Instead of using neighbor's reputation value as a weight (as in *HISTOS*), we take advantage of neighbor's relative degree as a weight which is more objective. Also, in our algorithm, when an agent takes a vote of its neighbors, the neighbors can recursively take votes of their neighbors.

Through the experiments, we show that our algorithm can compute quality reputation values by comparing the results with an existing reputation computation algorithm (*AS-CRED*). The algorithm can also successfully model propagation of reputations within the network. We show that in a dense network, reputations travel much quicker and diffuse wider given the same number of interactions. In addition, we show how frequencies of interactions can influence the reputation values. Since a higher rate of interactions implies greater significance, we believe that the velocity of interactions is an important parameter in our algorithm.

Since *ReMSA* is designed for distributed environments, the algorithm is employed within each agent. Therefore the time complexity of *ReMSA* for each agent is dependent on the number of events and the number of neighbors each agent has. Then the time complexity of the algorithm is $O(d_i * |E_i|)$ for each agent $i$.

## 7 Conclusions and Future Work

In this paper, we developed a new reputation computation algorithm, *ReMSA*, in which reputation values are considered subjective and reputation computation utilizes the topological information of the given network as well as velocity of interactions among agents. *ReMSA* also employes a distributed voting process to overcome biases that an agent might have towards others. To our best knowledge, no reputation computation algorithm considers velocity of events. Instead, most of the algorithms use the total number of interactions occurred. We believe that by taking velocity into consideration, it is possible to detect some abnormal activities since they tend to happen in a short period of time.

We use some of the degree distribution of the given network to take advantage of the topological information of the network. Many studies that involve social networks assume the second degrees (degrees of the neighbors) are known either because the degree distribution of agents contains high information. We are currently relaxing this assumption and trying to estimate the second degree distribution using bayesian probabilistic methods. We assume that each agent start the estimation with the same degree of its own [6]. The degree distribution of a given social network is assumed to follow a power-law distribution and we let each agent apply power-law to the estimated second degree distribution. By estimating the second degrees we do not rely on the information from outside since each agent can estimate all the information needed which will also prevent possible deceptions from outside sources.

Our algorithm is an online algorithm that can be deployed to observe ASes in real-time and can compute reputations of ASes as in *AS-CRED* [13]. According

to [19], centralized methods have limitations in BGP security and no solution has yet provided complete security. Unlike *AS-CRED* [13], *ReMSA* incorporates social dimensions and velocity of events that are important features of the *AS* network.

Also, as *ReMSA* was originally developed for social networks, we plan to apply it to more general social networks where reasoning about private friendship is an important value.

## References

1. Carbo J, Molina JM, Davila J (2003) Trust management through fuzzy reputation. Int J Coop Inf Syst 12(01):135–155
2. Sabater J, Sierra C (2001) Regret: reputation in gregarious societies. In: Proceedings of the fifth international conference on autonomous agents. AGENTS '01. ACM Press, New York, pp 194–195
3. Zacharia G (2000) Trust management through reputation mechanisms. Appl Artif Intell 14:881–907
4. Lee JY, Oh JC (2013) A model for recursive propagations of reputations in social networks. In: Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining. ASONAM '13. ACM Press, New York, pp 666–670
5. Galan J, Latek M, Rizi S (2011) Axelrod's metanorm games on networks. PLoS One 6(5):e20474
6. Galeotti A, Goyal S, Jackson MO, Vega-Redondo F, Yariv L (2010) Network games. Rev Econ Stud 77(1):218–244
7. Szabo G, Fath G (2007) Evolutionary games on graphs. Phys Rep 446(4–6):97–216
8. Pinyol I, Sabater-Mir J (2011) Computational trust and reputation models for open multi-agent systems: a review. Artif Intell Rev, 40(1):1–25
9. Huynh TD, Jennings NR, Shadbolt NR (2004) Fire: an integrated trust and reputation model for open multi-agent systems. In: Proceedings of the 16th european conference on artificial intelligence (ECAI), pp 18–22
10. Teacy WTL, Patel J, Jennings NR, Luck M (2006) Travos: trust and reputation in the context of inaccurate information sources. J Auton Agents Multi-Agent Syst 12:2006
11. Xiong L, Liu L (2004) Peertrust: supporting reputation-based trust for peer-to-peer electronic communities. IEEE Trans Knowl Data Eng 16:843–857
12. Lee J, Duan Y, Oh JC, Du W, Blair H, Wang L, Jin X (2012) Social network based reputation computation and document classification. j-jucs 18(4):532–553
13. Chang J, Venkatasubramanian KK, West AG, Kannan S, Lee I, Loo BT, Sokolsky O (2012) As-cred: reputation and alert service for interdomain routing. IEEE J Syst, 1:99
14. University of Oregon RouteViews project. http://www.routeviews.org/
15. Graph-generator. https://code.google.com/p/graph-generator/
16. Leskovec J, Faloutsos C (2006) Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '06, ACM Press, New York, pp 631–636
17. Chang J, Venkatasubramanian K, West A, Kannan S, Loo B, Sokolsky O, Lee I (2011) As-trust: a trust quantification scheme for autonomous systems in bgp. In: McCune J, Balacheff B, Perrig A, Sadeghi AR, Sasse A, Beres Y (eds) Trust and trustworthy computing. Lecture notes in computer science, vol 6740. Springer, Heidelberg, pp 262–276
18. The CAIDA AS Relationships dataset, January 1, 2010—January 31, 2010. http://www.caida.org/data/active/as-relationships/
19. Butler K, Farley T, McDaniel P, Rexford J (2010) A survey of bgp security issues and solutions. Proc IEEE 98(1):100–122