Trust management for vehicular networks

Renan Greca
Departamento de Informática
Universidade Federal do Paraná
Curitiba, Brazil
Email: rdmgreca@inf.ufpr.br

Luiz Carlos Albini
Departamento de Informática
Universidade Federal do Paraná
Curitiba, Brazil
Email: albini@inf.ufpr.br

***Abstract*—**

I. Introduction

This demo file is intended to serve as a "starter file" for IEEE conference papers produced under LaTeX using IEEEtran.cls version 1.8b and later. I wish you the best of success.

mds

August 26, 2015

A. Subsection Heading Here

Subsection text here.

1) Subsubsection Heading Here: Subsubsection text here.

II. Previous work

III. Algorithm

A. Tarjan's strongly connected components algorithm

An important aspect of [1] is the use of Tarjan's strongly connected components algorithm [2]. This allows a large graph to be abstracted into a smaller graph, which therefore reduces the input for further algorithms. Given a directed graph $T = (V, O)$, a strongly connected component is defined as a group of nodes in which, for any pair of nodes $u, v \in V$, there exists a path from u to v and a path from v to u. For the purposes of trust management, this definition is extended to accept only paths of edges with weight 1. Every node of the input graph T must belong to a component.

In the implementation used, $index$, $lowlink$, $count$ and $stack$ are global variables accessible from every call of the function. $index$ and $lowlink$ are arrays indexed by node IDs (predefined unique identifiers), $count$ is an integer and $stack$ is a last-in-first-out data structure.

The algorithm works by performing a depth-first search, adding nodes to $stack$ as they are visited. If two nodes are present on $stack$, then there is a path from the first node to the second one (in the order they were added to $stack$). Each node has two attributes assigned to it during the execution of the algorithm: $index$ is used to number the nodes in the order they are visited, while $lowlink$ is the lowest indexed node reachable from each node.

In the call that visits a node u, the algorithm must loop through each node v trusted by u (that is, $u \rightarrow v$ exists and has value 1). If node v has not yet been visited, the algorithm is called for v. The $lowlink$ of u is then calculated as the smallest value between $lowlink[u]$ and $lowlink[v]$, because any node reachable from v is also reachable from u. After the loop, if $lowlink[u]$ is equal to $index[u]$, it means that u is the lowest indexed node reachable from itself and that it is the root of a component. Therefore, nodes must be popped from the $stack$ until u is found. Each node popped, including u, is a member of a strongly connected component.

The number of components is, at most, $|v|$. In a worst-case scenario, each node is put into its own component; however, this would not be the case for most useful input graphs.

Algorithm 1 shows the general structure of Tarjan's algorithm [2] [3]. The complexity of the algorithm is $O(|V|+|O|)$ for a graph $T = (V, O)$.

With the results of Tarjan's algorithm, an undirected component graph $C = (V', O')$ is formed. Each $v' \in V'$ is the abstraction of one component identified by Tarjan's algorithm, while the edges $o' \in O'$ are edges from T between nodes that do not belong in the same component.

Algorithm 1 Tarjan's strongly connected components algorithm

```
1:  function TARJAN(vertex u)
2:      index[u] = count
3:      lowlink[u] = count
4:      count ← count + 1
5:      push u to stack
6:      for v in neighbors of u  do
7:          if weight of u → v is 0 then
8:              continue
9:          if index[v] = −1 then    // v has not been visited
    yet
10:             Tarjan(v)
11:         lowlink[u] ← min(lowlink[u], lowlink[v])
12:     if lowlink[u] = index[u] then
13:         repeat   // unstack nodes until u is found
14:             pop w from stack
15:             add w to component
16:         until w = u
```

B. Graph coloring with minimum colors

The algorithm proposed in [4] is an efficient approach to graph coloring, a classic graph theory problem. Graph coloring is one of the heuristics used in [1] to detect malicious nodes after the generation of the component graph using Tarjan's

algorithm. Since it was the heuristic that presented the best results, it has been chosen as the heuristic for this current study as well.

The process of graph coloring consists of labeling each node with a color so that no two neighboring nodes share the same color. This problem has been studied in Computer Science since, at least, 1972 [5] and has been studied as a classic mathematics problem for even longer [6]. It has been proven mathematically that any planar graph can be colored with at most four colors [7], but discovering the smallest number of colors necessary to color an arbitrary graph (called the graph's chromatic number) is an NP-hard problem [8].

In [4], the authors propose to color a graph using the minimum possible amount of colors. Although they do not prove that their algorithm always uses the smallest possible amount of colors, the output is always a correct coloration and the algorithm is nevertheless efficient. The complexity of the algorithm is $O(|E|)$ for an undirected graph $G = (V, E)$. As a comparison, the classic DSATUR algorithm for graph coloring has complexity $O(|V|^2)$ [9]. For the purposes of this study, it is not necessary to prove that the coloring algorithm's output uses the minimum possible number of colors.

Algorithm 2 shows the general structure of the graph coloring algorithm [4] [3].

A limitation of this algorithm is that the edges must be sorted according to node indexes beforehand. It does not matter which nodes get assigned which indexes, but once they are assigned those numbers, the algorithm must follow the edges in numerical order. This is demonstrated in [3].

Algorithm 2 Graph coloring with minimum colors

```
1: function COLORING(graph G)
2:     color all nodes of G with 0
3:     d → 0
4:     for e = (u, v) in edges of G do
5:         if u and v have the same color then
6:             if color[v] = d then
7:                 d → d + 1
8:             color[v] → d
```

C. Malicious Node Identification Algorithm

The basis of the presented trust model is the Malicious Node Identification Algorithm (MaNI) proposed in [1], [3]. The authors present a malicious node identification scheme based on strongly connected components and graph coloring. The model is proposed for complex networks in general, but is not suited for VANETs because it is designed only for static networks. Furthermore, the algorithm is executed by a global observer which has information about the complete network.

The input graph $T = (V, O)$ is a static, connected, and directed graph containing all trust relationships in the network. Such relationships are binary, so there are no varying degrees of trust: either one node trusts another completely (edge value is 1), or it distrusts the other completely (edge value is 0). The relationships are also directed, meaning that if the value of $u \to v$ is 1, $v \to u$ is not necessarily 1.

The process for identifying malicious nodes within T is as follows:

First, T is separated into strongly connected components using Tarjan's algorithm [2], which is described in detail in subsection III-A. For each node in a component, there is a path formed by edges of weight 1 to each other node in the same component. In other words, within a single component, all nodes trust one another directly or indirectly; nodes that do not satisfy this condition are separated into different components. Each of these components becomes a node of a component graph $C = (V', O')$.

The creation of the graph C simplifies the remaining computation. Since each node of C is a vertex $v' \in V'$ and each vertex v' is a component of T in which all nodes trust each other, for the purposes of identifying malicious nodes, all nodes within each of those components can be treated as one. They can either be benign nodes which legitimately trust one another, or malicious nodes colluding with each other. After the formation of C, one or more heuristics can be used to classify the nodes as benign or mal icious.

The article describes the coloring heuristic, which uses a graph coloring algorithm, either DSATUR [9] or the algorithm described in subsection III-B. After running either algorithms with graph C as input, the color whose nodes in C represent the most nodes in T is classified as correct, and all others are classified as malicious. Once this information in C is brought back to graph T, it is trivial to label the nodes in T as either benign or malicious based on their components' classifications.

In the experiments shown in [3], the coloring heuristic shows the most promising results, identifying a high ratio of the malicious nodes in the network. Other heuristics were experimented with, but were either less effective in detecting malicious nodes, or provided too many false positives. Therefore, for the purposes of this research, only the coloring heuristic is considered.

Two types of experiments were made in each network: first, all malicious nodes inverted the edge weights leading to their neighbors; second, malicious nodes randomly inverted or not the weights. In the first scenario, the results show excellent precision in most networks, detecting nearly every malicious node. Experimenting with the second scenario, the results are less precise, however still promising: with up to 20% of malicious nodes in the network, the error rate is under 7%, while with the worst case, 50% of the network being malicious, the error rate is approximately 15%.

The authors suggest running the algorithm repeatedly after removing the malicious nodes from the network. By doing this, nearly all malicious nodes are detected by it even when randomly changing edge weights.

IV. Trust management for a vehicular network

The MaNI algorithm works well and is efficient for static graphs. However, some important modifications had to be

made to accommodate dynamic graphs in a decentralized fashion. The main changes made are described below.

First of all, in order to work with a dynamic graph, snapshots of the graph must be considered. A snapshot $G_t = (V, E_t)$ represents the topology of the network at a timestamp t. As the network is dynamic, it is expected that each G_t is different from G_{t-1}, but also that the changes follow a determined mobility model.

Furthermore, it was necessary to decentralize the algorithm. The MaNI algorithm is executed by a supervisor of the network with complete access to nodes' trust values, which is not viable in a vehicular ad-hoc network. Therefore, each node must maintain its own vision of the network surrounding it. Every node u has a trust graph $T^u = (V^u, O^u)$. Since T^u changes over time, there is a T^u_t for every snapshot t.

Another important change is on the trust values themselves. MaNI uses binary trust (either 0 or 1). This has been replaced by a floating-point number between 0 and 1; a predefined threshold defines which values are deemed trustworthy or not. This was not the case in the first versions of the trust model; it was added in order to avoid abrupt changes to opinions. Since the network is different on each iteration of the algorithm, there were instances in which a change to the topology caused a radical change to the output of the graph coloring algorithm. Using a trust value that gradually changes over time, output variations between two iterations became small and gradual.

// A prerequisite of this algorithm is the existence of a test that correctly classifies a neighboring node as benign or malicious. (preciso mencionar isso em algum lugar mas não sei bem onde)

At the start of an execution, a node knows only about itself. From that point, iterations of the algorithm will run in a set interval of l seconds. In each iteration (labeled t after the timestamp it occurs on) and for every node u, the following steps are executed:

1) Node u checks who are its neighbors (nodes within communication range). Newly discovered nodes and newly formed edges are added to T^u_t Edges are created with weight 0.5.
2) Node u tests all of its neighbors to discover which ones can be directly trusted or not. New trust values are computed for the edges using the average between the previous value and either 1 (if the neighbor is trustworthy) or 0 (otherwise).
3) If a neighbor v is trustworthy, u merges T^v_{t-1} into T^u_t, adding nodes and edges that were present on T^v_{t-1} but not on T^u_{t-1}.
4) Tarjan's algorithm is executed to identify the strongly connected components of T^u_t, resulting in a component graph C^u_t. Tarjan's algorithm uses a threshold value between 0 or 1 to determine whether or not to take edges into consideration.
5) The graph coloring algorithm is executed on C^u_t and nodes are identified as benign or malicious according to the same rules as the MaNI algorithm.

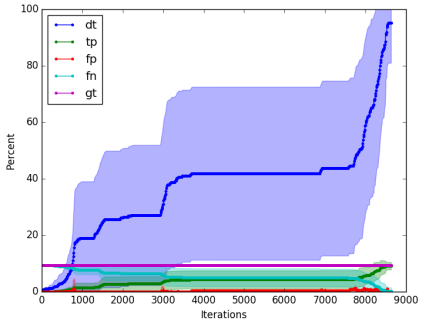

Fig. 1. Simulation without random nodes and range of 10m.

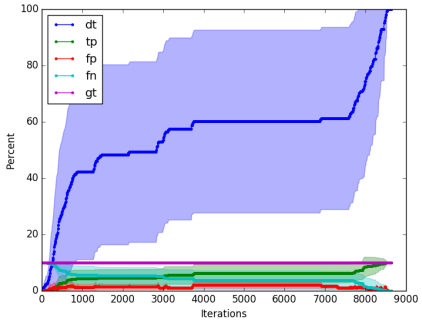

Fig. 2. Simulation without random nodes and range of 30m.

V. Results

In order to test the trust model, simulations were made using an implementation of the algorithm in Python. To generate the input graphs with node mobility, the ONE simulator [10] was used in conjunction with the Working Day Movement Model [11], which provides a reasonable imitation of vehicle movement in real life. Snapshots of the network were taken every 10 simulated seconds, and these snapshots were used as input for the algorithm.

Most of the parameters for the simulator were taken from the article detailing the Working Day Movement Model. The simulation ran for 86400 seconds (24 hours), with a work day length of 28800 seconds and a standard deviation of departure time of 7200 seconds. Nodes move between 7 and 10 m/s in an area of approximately 14 km^2 based on a section of the map of Helsinki. Simulations had either 150 nodes moving according to the Working Day Movement Model, or a combination of those same 150 nodes plus 10 nodes moving randomly across the map (to simulate vehicles that don't conform to standard daily paths). Since this simulation is for vehicles instead of pedestrians, there are no buses in the model and every node is guaranteed to own a vehicle and travel by car. The parameters regarding offices, meeting spots and shopping were kept intact.

VI. Conclusion

The conclusion goes here.

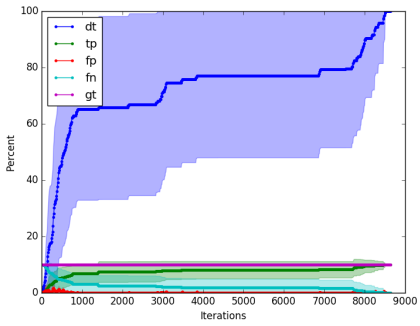

Fig. 3. Simulation without random nodes and range of 50m.

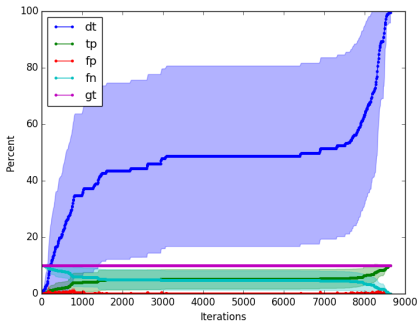

Fig. 4. Simulation with random nodes and range of 10m.

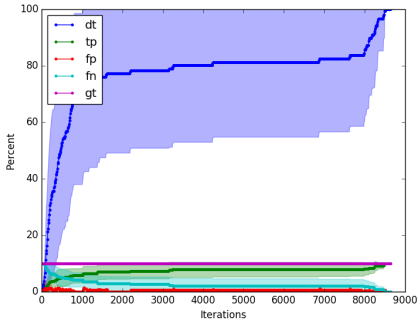

Fig. 5. Simulation with random nodes and range of 30m.

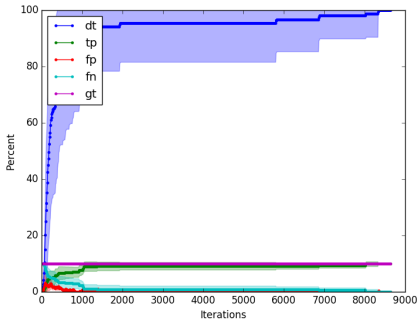

Fig. 6. Simulation with random nodes and range of 50m.

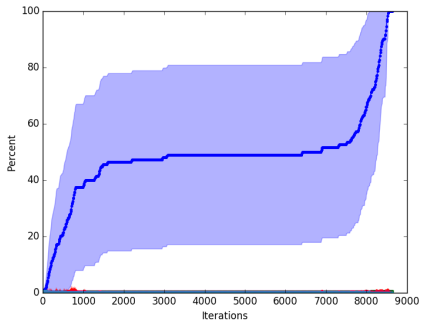

Fig. 7. Simulation with random nodes, range of 10m and 1% malicious.

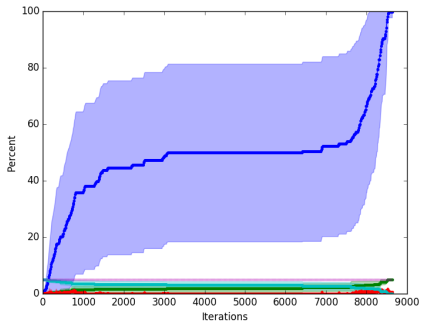

Fig. 8. Simulation with random nodes, range of 10m and 5% malicious.

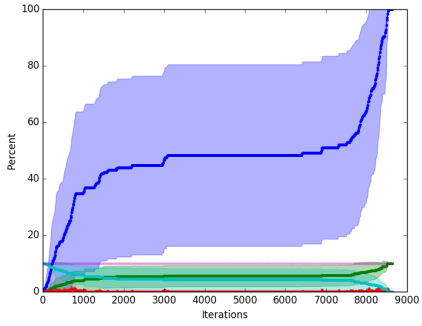

Fig. 9. Simulation with random nodes, range of 10m and 10% malicious.

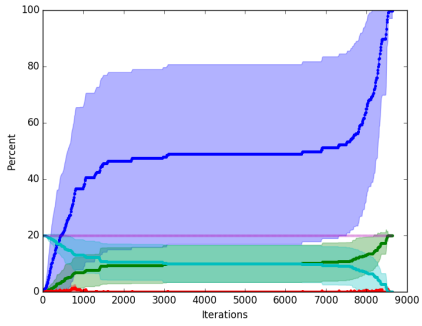

Fig. 10. Simulation with random nodes, range of 10m and 20% malicious.

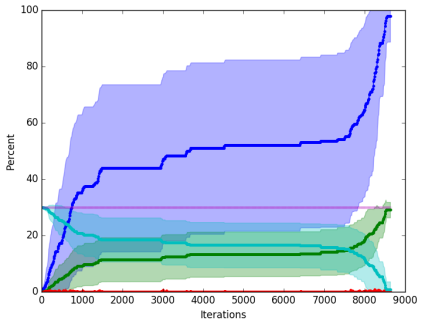

Fig. 11. Simulation with random nodes, range of 10m and 30% malicious.

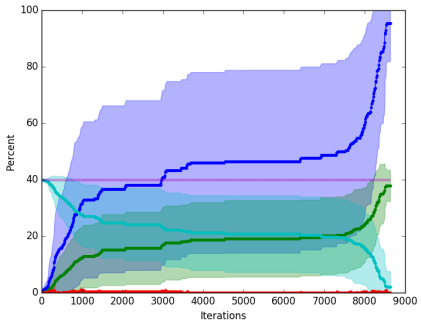

Fig. 12. Simulation with random nodes, range of 10m and 40% malicious.

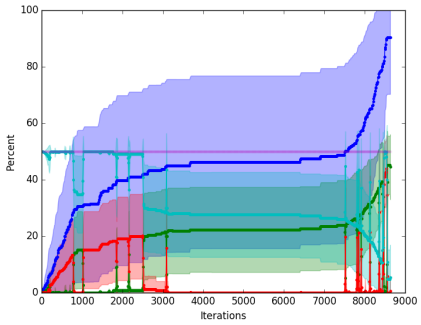

Fig. 13. Simulation with random nodes, range of 10m and 50% malicious.

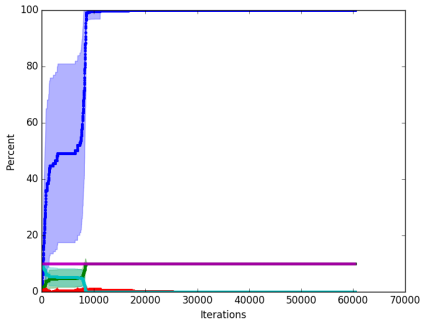

Fig. 14. Simulation with random nodes, range of 10m and 10% malicious during 7 days.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] G. Vernize, A. L. P. Guedes, and L. C. P. Albini, "Malicious nodes identification for complex network based on local views," *The Computer Journal*, vol. 58, no. 10, pp. 2476–2491, 2015.

[2] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[3] G. Vernize, "Identificação de nós maliciosos em redes complexas baseada em visões locais," MSc dissertation, Universidade Federal do Paraná, 2013.

[4] A. Mittal, P. Jain, S. Mathur, and P. Bhatt, "Graph coloring with minimum colors: An easy approach," in *Communication Systems and Network Technologies (CSNT), 2011 International Conference on*. IEEE, 2011, pp. 638–641.

[5] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[6] A. B. Kempe, "On the geographical problem of the four colours," *American journal of mathematics*, vol. 2, no. 3, pp. 193–200, 1879.

[7] K. Appel, W. Haken, and J. Koch, "Every planar map is four colorable," *Bull. Amer. Math. Soc*, vol. 82, no. 5, pp. 711–712, 1976.

[8] A. Sánchez-Arroyo, "Determining the total colouring number is np-hard," *Discrete Mathematics*, vol. 78, no. 3, pp. 315–319, 1989.

[9] D. Brélaz, "New methods to color the vertices of a graph," *Communications of the ACM*, vol. 22, no. 4, pp. 251–256, 1979.

[10] A. Keränen, J. Ott, and T. Kärkkäinen, "The one simulator for dtn protocol evaluation," in *Proceedings of the 2nd international conference on simulation tools and techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 55.

[11] F. Ekman, A. Keränen, J. Karvo, and J. Ott, "Working day movement model," in *Proceedings of the 1st ACM SIGMOBILE workshop on Mobility models*. ACM, 2008, pp. 33–40.