Misbehavior Detection in VANETs

Manish Kumar Dash, Rodney Stephen Rodrigues, M. Krishnananda Singh, Longkiri Bey
(140101087, 174101036, 174101005, 140101035)
Computer Science and Engineering
Indian Institute of Technology, Guwahati

***Abstract*—Vehicular Ad hoc networks(VANETs) communicates through exchange of information among different nodes (vehicular and infrastructural) in the network. For the network to function correctly, reliable and real time information must be procured, which depends on node cooperation and trust. Presence of malicious nodes in the network can have disastrous consequences. It is therefore essential to detect misbehaving nodes and defend the VANET against them. In this paper, firstly, we explore various solutions present in the literature. The mesh-network between the nodes in VANETs is leveraged and the time-series nature of the generated data is exploited to present a solution that is very accurate and robust to the real-life noisy data. It can be extended to both inter-vehicular communications and vehicle-to-infrastructure.**

Keywords—VANETs, Misbehavior detection, LSTM, decision trees, random forests, SUMO

I. Introduction

Vehicular Ad-hoc Networks (VANETs) are ad-hoc networks, a special case of Mobile Ad-hoc Networks (MANETs) involving vehicles. VANETs have become a hot topic in network research with the advent of smart cars and more powerful embedded devices. The main goal is to provide the drivers real time information about the road conditions, which would increase travel safety and comfort.

VANETs need nodes to cooperate with each other to exchange information and ensure trust between nodes to make sure that the information is credible. But due to characteristics such as high mobility, real-time constraints, privacy, and scalability, maintaining cooperation and trust in VANETs is a challenging task. It is possible that malicious nodes may try to sabotage the network to maximize their own personal benefit at the cost of overall network performance. They may use the network provided information and contribute nothing in return, or give wrong information to the network. These nodes may also launch various attacks like black hole, Sybil, DoS, Timing, etc.

Towards this end, it is imperative to devise a solution that can detect such misbehaving nodes and discard them from the network to avoid degradation in performance. In this paper, we propose a technique that can be used in inter-vehicle, inter-infrastructural as well as vehicle-to-infrastructural communications. Our contributions are as follows:

- A large dataset for misbehaving nodes have been generated using the popular Simulation of Urban Mobility (SUMO) simulator.
- A comprehensive analysis of the existing solutions is presented.
- An online, deep neural network is proposed which is capable of processing the time series data produced by VANETs.

This paper is organized as follows: Section II explains our motivation in choosing this particular problem. Section III discusses the past work that has been done to tackle misbehavior in VANETs. Section IV gives a brief overview of our solution approach. Section V discusses the data generation process using SUMO. Sections VI and VII presents the various solutions that have been analyzed. Section VIII holds the conclusion and future work. Sections IX and X hold the acknowledgment and references.

II. Motivation

With the growth in the artificial intelligence and automation industries the drive towards “smart” cars and road traffic has intensified. VANETs have especially attracted a lot of attention. Intelligent vehicles that communicate with each other to exchange information can make road travel a lot safer. We can use the power of artificial intelligence and robust networking in VANETs to reduce traffic accidents, regulate traffic in the urban cities and make road travel a lot more safer and comfortable.

In such cases, it is of utmost importance that the network performs correctly and do not make the wrong decisions that could potentially endanger human lives. VANETs are mostly distributed and are dependent on the cooperation and trustworthiness of the nodes. Even a single node behaving maliciously can completely damage the whole network. There can be many reasons for a node to misbehave. There may be selfish reasons like to decrease congestion in a particular segment of the road, malicious intent to sabotage the network by intentionally sending false information, or just failure of equipment due to natural causes. No matter what the reason, it is imperative that the misbehaving nodes are detected and the false information ignored by the system to prevent unwanted results.

III. Related Work

VANETs have become a hot topic in network research over the years. Current research has focused on location privacy, maintaining authenticity of data and revocation of certificates and secret credentials. This in turn needs the presence of a Trusted Authority (TA) who issues and revokes certificates for the nodes. [1],[2],[3],[4] are surveys on security challenges in VANETs. Authentication and signature schemes have been studied extensively like ECMV[5] and PASS[6]. Detection of malicious nodes and revocation of certificates have been studied in [6]. All these methods come under the category of node-centric misbehavior detection.

In this paper we assume that the nodes misbehave mostly for natural or selfish reasons, i.e., their intent is never "evil". They do not sabotage the network only to cause traffic accidents. Each node normally sends valid information. In isolated cases, when the nodes malfunction or want to optimize their personal goals, do they provide false information to the network. As [7] points out, in these cases if a node-centric technique revokes a node's certificate then all the useful information sent out by that node will be ignored. Thus we do not need to classify nodes as purely malicious or normal, but we want to identify correct and false information sent out by a node in real time.

This approach is known as a data centric misbehavior detection. Accurately predicting whether current information is correct or not is very important, as it makes the network robust against new attackers and unforeseen complications due to natural causes. In [8] Raya considers trust on the information rather on the source of information. In [9] the author penalizes the nodes whenever they produce false information but do not remove them from the network.

In [10], Golle et al. create a model of the network. This model is the set of all possible events that can take place in the network. So every event is observed by another node and checked with the model. If the model validates the event then it is correct, otherwise not. This solution is just a proof-of-concept. The author has not specified how the model can be created or maintained for larger networks. For a general VANETs, which can have a large number of nodes, maintaining a global model or dataset storing all possible events can be very expensive and impractical.

In [11], Ghosh et al. investigate post crash scenarios. They compare simulation results in cases where a node misbehaved or functioned normally. They dealt particularly in the case of vehicular nodes sending Post Crash Notifications(PCN). But their approach assumes that a malicious node always gives correct information about its position. Moreover, accurately modelling a real world vehicle trajectory involving an accident is a challenge.

In [12] Vulmiri et al. propose a probabilistic misbehavior detection approach. They leverage the secondary notifications, i.e., responses to the primary notification to verify the reliability of the primary alerts received by the vehicles. Harit et al.[13] further improved this by reducing approximation errors. In [14] Ruj et al., the system monitors the actions of vehicles after alert messages has been sent out. Then reported and estimated positions of the vehicles are used to make future decisions. In both these approaches, the systems do not learn from past mistakes and the detection algorithms are scenario and application dependent. In [15,16] Grover et al. have employed machine learning techniques to detect misbehaving nodes. They have used classical classifiers like J-48, Random forests, Naïve Bayes and AdaBoost1. Moreover, majority voting is used in [16] to further improve the accuracy of detection of misbehaving nodes.

The data generated by the VANET nodes can be modeled as a time-series sequence. This makes our problem very similar to the time-series forecasting that has been well explored in other fields. It is worth noting that the nodes do not depend on each other linearly, and there exists a non-linear relationship between a misbehaving node and its normal neighbors. To make an accurate model, this non-linear relationship must be captured. This is further discussed in Section

IV. Approach

Our solution approach depends on the availability of a large amount of time-series data about network statistics in the presence and absence of misbehaving data, and then to correctly process this data and predicting the estimated behavior of normal nodes correctly. In this way our work can be seen in two phases:

A. Data generation

Despite the large amount of work done in the field of VANETs and even in the area of detecting misbehaving nodes, there is a surprising absence of a good dataset containing network traces of misbehaving nodes. We decided to generate our own dataset using the SUMO simulator and this is further discussed in Section 5.

B. Model Architecture

We noticed the time-series nature of the problem and hence decided to first explore the already existing solutions for time-series forecasting. As mentioned before, there exists a non-linear relationship between the nodes and it is imperative for the model to capture this. Towards this end, we explored three classes of solutions:

- Linear models
- Non-linear models
- Deep neural networks (DNNs)

We cross-examined these three classes of solutions with respect to both detection accuracy as well as efficiency of the algorithm. The detailed analysis is presented in Section

V. Dataset Generation

There are many popular network simulators available which can be used to generate many kinds of data and network traces for VANETs. Some of the most popular of them are the Simulation of Urban Mobility (SUMO), OMNet++, Veins and VANET-simulator.

We chose the SUMO simulator because of the extensive documentation support and the presence of an active and huge community. Moreover, in this paper we are only concerned with the information generated by the nodes and do not consider any networking or communication overhead. As SUMO only produces mobility traces, this works perfectly for out case.

Before generating the data, there were a few considerations that we had to take care of:

A. *Type of Misbehavior*

In data centric misbehavior detection, the technique has to be robust enough to handle different types of communication paradigms. But in essence it has to learn how to differentiate between correct and false information being sent by the same node. In this paper we decided to go with simple infrastructural nodes that provide network statistics per road or per lane.

We chose these types of nodes because they provide an overview of entire network and essential VANET functions like traffic regulation, traffic re-routing depend on the information provided by these nodes. An attack that causes these nodes to misbehave can seriously cripple the network. These nodes form the backbone of the VANETs and hence their correct functioning has to be ensured.

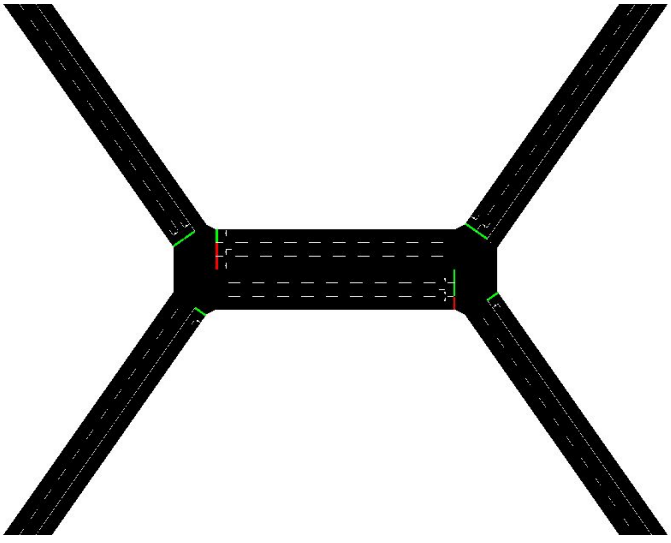

Fig 1. The road network

B. *Road network*

We have tested our model on a simple road network consisting of 5 road segments. This small network captures the essence of the prediction problem: we want to estimate the nodes output on basis of the data from the neighboring nodes.

C. *Infrastructural nodes*

SUMO provides a variety of detector nodes that can be deployed in the simulation. We have chosen the entry-exit detector provided by SUMO. This detector monitors the specified segment of road and generates network statistics about:

- the number of vehicles that are currently in the segment.
- the number of haltings in the segment
- the average speed of vehicles that cross the segment

The number of haltings and the average speed together act as a measure for traffic congestion in the road segment.

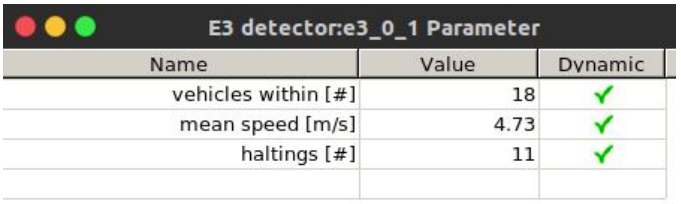

Fig 2. Parameters recorded by the E3 detector

D. *Simulation*

SUMO is a very user friendly simulator that can generate huge mobility traces very quickly. It deals mainly with ".xml" files. We first defined our road network in the main .net.xml file. Then five entry-exit detectors were placed in each of the five road segments through the .add.xml file. Then we initialized a number of vehicles with randomly generated routes that are uniformly distributed across the network. These routes were specified in the .rou.xml file.

SUMO internally maintains "good" behavior in the vehicular and infrastructural nodes. This means that we cannot simulate conditions where a node is misbehaving directly in the simulation. To counter this, we decided to tackle the problem in the opposite direction. We monitored the nodes' outputs for the four outer roads and tried to predict the real-time output of the central fifth road. As all traffic originates at the network's endpoints, the traffic statistics of the fifth road must depend linearly or non-linearly on the four neighboring nodes. This can be easily seen as the fifth road is neither a source nor a sink when considering traffic flow, and

hence the overall traffic flow through it must depend on its neighbors.

With the above consideration in mind, we ran our simulation for 2,000,000 time steps, where a new vehicle was introduced into the network at every 10^{th} time step. Our detectors generated aggregated outputs at every 30 time steps. This aggregation is important to allow the detectors to accumulate enough vehicles in the road segment to provide meaningful traffic statistics. This resulted in about 266,000 data points generated by the network.

So our final dataset consisted 266,000 datapoints, each datapoint being a combined vector of 8 features, 2 features each from the four detectors (as mentioned in part *C*). Our models' goal is to estimate the congestion statistics of the central fifth road, for which we have the simulation's groundtruth statistics. In this way we have created a supervised prediction dataset for our network.

The simulation code with proper documentation is available in the git repository:

https://github.com/manishdash12/Misbehaviour-VANETs

VI. Understanding misbehavior

A lane detector is believed to be misbehaving if it either reports more or less congestion (halt-time) than what it should ideally report. In real time scenario, this deviation (D) can be represented as the difference between reported (H_R) and expected (H_E) halt time per vehicle.

$$D = || H_R - H_E ||$$

At any time we can observe the actual reported value H_R from a node. But the network can not be sure of the correctness of this reported value. Without proper validity checks, if a reported, yet incorrect value is used for network decisions like switching a traffic signal or a smart car making a turn, it may lead to traffic accidents and even loss of life. So it becomes very important to simulate such scenarios in advance and design and train models that can tackle such situations.

Unfortunately, simulating every possible real-life scenario is impractical. Moreover, most simulation environments do not give a lot of flexibility to model misbehaviors. To counter this we suggest tackling the problem the other way around: if we can accurately estimate the expected value (H_E) that a node should report, then if the deviation D crosses a threshold, we can correctly deduce that the node is misbehaving in some way.

As mentioned in Section V, we decided to choose the entry-exit detectors in SUMO as our VANET nodes. These nodes record and report the average traffic statistics for a particular segment of road. This closely reflects the real-world scenario where traffic lights at junctions have sensors that can pick up how many vehicles are passing into a road(entry) or moving out (exit). Moreover, it is evident that these nodes' readings are related to each other as vehicles moving from one road segment has to move to an adjoining road segment.

Thus a node's readings can be predicted from the readings of it neighbor nodes. All the nodes produce readings periodically, and hence the readings of the past time-stamps also contribute to the current reading. Keeping this in mind, we decided to concatenate all the readings of the neighbors of a node as the evidence data. After repeated experiments we found that a lag of 5 past observations is very effective for producing accurate predictions. Thus, the input to our models would be a concatenation of the past 5 readings of the neighbors readings.

VII. Analysis of models

Just like any regression problem, any loss function is at heart based on the difference between label and prediction. We use mean squared error (MSE) as our loss function while training our models. For a given batch size, mean squared error is the average of square of difference between label and prediction.

$$Mean\_squared\_error = avg\ (\ ||\ label - prediction\ ||^2\ )$$

After deciding on the format of the training data and the loss function, we implemented and compared various models. These solutions can be divided into three categories:

A. Linear models

Classical linear regression models are very popular and often produce decent results. Four popular linear models: linear regressor, Lasso, Ridge and Elasticnet [17] regressor. The results are given in Table I.

Table I shows the inability of linear models to accurately learn the non-linear relationships between the nodes' readings. Lasso and Ridge are regularized versions of linear regression. They tend to shrink the coefficients of the features that have redundant information. We observe that Ridge, Lasso and Elasticnet perform better than simple linear regression. This is due to the L1 and L2 regularization used in Lasso and Ridge respectively, while Elasticnet uses both L1 and L2. This prevents overfitting and gives better performance over the test data. Resource wise, the linear models are very efficient. But the poor regression performance is more than enough to reject linear models as an effective solution for our problem.

TABLE I. COMPARISON OF LINEAR MODELS

Linear model	Metrics		
	R^2 *score*	*MSE*	*Time (s)*
Linear regressor	0.288	4846.98	0.17
Lasso	0.425	4006.93	3.5
Ridge	0.466	3721.82	0.05
Elasticnet	0.418	4058.76	4.6

B. Non-linear models

The non-linear relationship of the nodes would require more complex models than simple linear regression. We experimented with a large number of non-linear models and the results are summarised in Table II.

From Table II, we can immediately notice the success of non-linear models over their linear counterparts. The minimum R^2 score is 0.755 which is much higher than the maximum of the linear models. This is evidence to the fact that the nodes indeed have a non-linear relationship and the non-linear models were able to capture that.

We started with a simple Decision tree. Decision trees (DTs) make decisions in a tree-like manner, partitioning the dataset into parts which are as pure as possible, based on the value of a particular feature at that node. The path from the root to the leaf combines many such leaf decisions, which captures the non-linear relationship between the features. We then extend the decision trees through two ensemble techniques: *boosting* and *bagging*.

Boosting is an ensemble technique where the model learns many *weak learners*, each new learner trying to improve upon the previous learner and thus improves performance. In Adaboost DTs [18], after each weak learner is trained the algorithm increases the weights of the error datapoints, and reduce the weights of the correctly predicted datapoints. This helps the overall *strong learner* to be much better than the weak learners.This is evident from the boost in R^2 score compared to simple DTs. Gradient Boosting regressor [20] is another boosting algorithm which performs well with time-series data. Instead of modifying weights of the datapoints like Adaboost, the new learner is trained on the residual errors of the past learner. This makes sure that the newer models learn from the mistakes of the older ones. As we are now learning many small learners, the time to train also increases.

Bagging is another ensemble technique where the dataset is randomly and with replacement partitioned into several subsets. Individual models are trained on these subsets and the prediction is the average across these models. This allows the model to learn small data trends which may be missing the whole dataset, but evident in the smaller subsets. We observe that the Bagging decision trees [19] outperform Adaboost DTs. Similarly to Adaboost, the training time increases due to repeated model trainings.

KNN is a popular supervised classification technique, which can be extended to regression. KNN is different from decision trees as KNN learn from the data directly without forming a classification model, much unlike the tree-structure of DTs. This gives KNN flexibility to adapt to new data boundaries than the DTs. But during testing, DTs can predict results very fast due to simple rule-based branch switching. KNNs have to compute a new value for every training instance which makes it slower. This would not be a great fit for our problem as the VANET should be able to take decisions fast.

Random forests (RFs) [21] performed the best among the models we tested. RFs train a number of fully-grown DTs, forming a forest. Each tree is trained on a slightly different subset of the training data. Our RF consisted of 20 trees which is the reason for the large 6.5 seconds of training time. XGBoost (extreme gradient boosting) [22] is well tuned extension of the Gradient boosting algorithm. It performs well but is beaten by the RF.

TABLE II. COMPARISON OF NON-LINEAR MODELS

Non-linear model	Metrics		
	R^2 *score*	*MSE*	*Time (s)*
Decision Tree	0.755	1709.67	0.32
Adaboost Decision Tree	0.808	1358.88	3.67
Bagged Decision Tree	0.882	821.91	5.05
Gradient Boosting Regressor	0.829	1186.36	11.294
KNN	0.801	1384.90	0.518
Random Forests	0.890	766.0	6.515
XGBoost	0.824	1223.75	4.77

These experiments show that the non-linear models are capable of accurately capturing the data trends and producing very good results. But the recent achievements of deep neural networks warrant some additional experiments.

C. Deep neural networks (DNNs)

Our training data is a time-series sequence. At each time step *t*, we know the reported halt-time value per vehicle, for each lane detector node. Estimating the expected halt-time value per vehicle at heart a regression problem. We know the correct value for previous time steps {0,1,..,*t-1*} and we use it to estimate H_E

We want a model that can learn incrementally, learning as new conditions are met. Towards that end, with scope to learn both linear, and non-linear functions *f(x)*, Deep Neural Networks (DNNs) outperform Support vector Regressor (SVR) and other incremental learning ML algorithms. The

ability to incrementally learn from data allows the online model to consume less time to generate predictions and can be deployed in servers with real-time stream data flow.

Working on the principle of "learning never stops", our model first validates each new entry and then if permissible, trains on it. Avoiding re-training with entire dataset, the model tweaks its weights by performing backpropagation using single entry, thereby handling stream of data. We trained 2 DNNs with different characteristics as given in Table III.

TABLE III. COMPARISON OF DNNS

DNN	Metrics		
	R^2 *score*	*MSE*	*Time (s)*
online-MLP	0.815	1400.51	119.2
LSTM	0.885	801.78	213.6

Our first model is an extension of the multi-layer perceptron model. The MLP model will first be trained on a batch of data. The model is light enough to be then deployed in a node, where it could be trained online using the "leaning never stop" paradigm. The MLP model consist of three layers: an input layer containing 250 units, two hidden layers of 500 units and 100 units respectively, and an output layer of 100 units. It was trained for 500 iterations, optimized with Adam with a learning rate of 0.001.

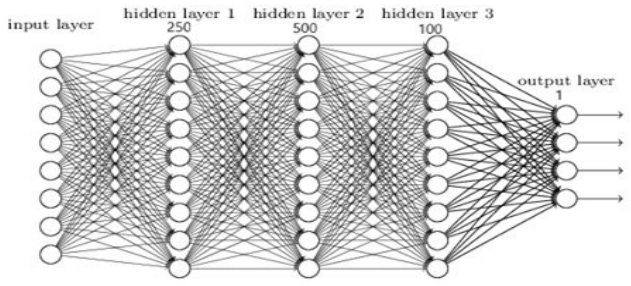

Fig 3. The MLP architecture

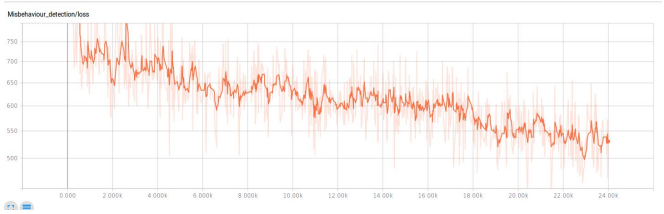

Fig 4. The MLP loss

As evident from Table III, the MLP model does not perform on par with the non-linear model. This can be explained by the simple structure of the DNN. The MLP model, though, can be deployed on the nodes itself to detect incorrect data sent by the neighbors and then can learn in an online fashion.

On the other hand, a model that can remember the past sequences of data and identify trends in the periodic data is ideal. Recurrent neural networks (RNNs) are designed to apply a feedback of the current output as input to the model in the next iteration. This allows RNNs to remember past predictions and mistakes and the model learns the trends in periodic data much better. LSTMs [23] are a special type of RNN that function by maintaining a *cell state* for each LSTM cell. This cell state allows the LSTM to freely pass information between cells even a large distance apart, while storing information in the cell at the same time. This allows LSTMs to remember long sequences easily and thus they perform very well in time-series forecasting.

Our LSTM model consisted of 50 LSTM cells in the first later, followed by another 100 cells in the second layer and finally one dense layer. A dropout of 50% was implemented in between the layers for regularization. Relu was used as the activation function. It was trained for 100 epochs, optimized with Adam with a learning rate of 0.0001.

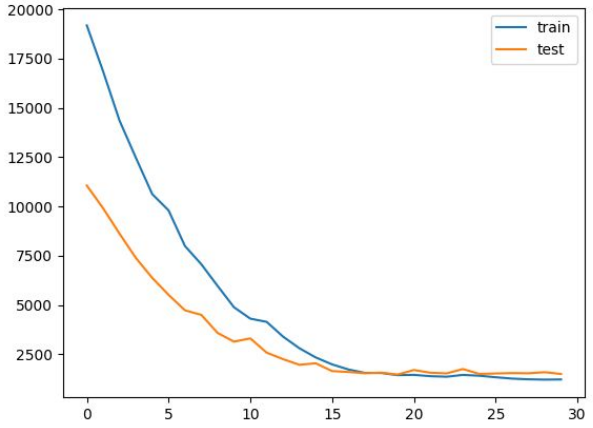

Fig 5. The LSTM loss

The LSTM model performs almost as better as Random Forests. This shows the ability of LSTM to identify and remember trends in periodic data. But it is a lot heavier than the MLP model as the increased training time shows.

VIII. FUTURE WORK

The model can be further extended to include dynamic vehicular nodes, where the neighbors keep changing constantly. The DNN models can be further tuned to allow it to figure out that the net traffic flow remains the same in a junction. This may lead to better accuracies.

The non-linear models were analysed independently but further tests can be run with a stacked regressor: a collection

of different models that improve each other. Similarly, combining a statistical model with a DNN can make the model more robust and accurate.

IX. Acknowledgement

We would like to thank Dr. Rashmi D. Baruah for providing us the opportunity to formulate and explore this problem. We would also like to extend our heartfelt gratitude towards Ms. Sonia Sharma and all the TAs for their guidance and being there for discussing our trivial questions.

X. References

[1] P. Papadimitratos, L. Buttyan, T. Holczer, E. Schoch, J. Freudiger, M. Raya, Z. Ma, F. Kargl, A. Kung, and J. p. Hubaux. Secure vehicular communication systems: design and architecture. In In IEEE Wireless Communication Magazine, pages 100–109, 2008

[2] F. Kargl, P. Papadimitratos, L. Buttyan, M. Mter, E. Schoch, B. Wiedersheim, T. v. Thong, G. Cal, A. Held, A. Kung, and J. p. Hubaux. Secure vehicular communication systems: Implementation, performance, and research challenges. In In IEEE Wireless Communication Magazine, pages 110–118, 2008

[3] Bryan Parno and Adrian Perrig. Challenges in security vehicular networks. In HotNets-IV, 2005.

[4] Subir Biswas, Md. Mahbubul Haque, and Jelena V. Misic. Privacy and anonymity in vanets: A contemporary study. Ad Hoc & Sensor Wireless Networks, 10(2-3):177–192, 2010.

[5] Albert Wasef, Yixin Jiang, and Xuemin Shen. Ecmv: Efficient certificate management scheme for vehicular networks. In GLOBECOM, pages 639– 643. IEEE, 2008.

[6] Yipin Sun, Rongxing Lu, Xiaodong Lin, Xuemin Shen, and Jinshu Su. An efficient pseudonymous authentication scheme with strong privacy preservation for vehicular communications. IEEE Trans. on Vehicular Technology, 59(7):3589–3603, 2010

[7] Ruj, S., Cavenaghi, M. A., Huang, Z., Nayak, A., & Stojmenovic, I. (2011, September). On data-centric misbehavior detection in VANETs. In *Vehicular technology conference (VTC Fall), 2011 IEEE* (pp. 1-5).

[8] Maxim Raya. Data-Centric Trust in Ephemeral Networks. Ph D Thesis. EPFL, Lausanne, 2009.

[9] Ahren Studer, Mark Luk, and Adrian Perrig. Efficient mechanisms to provide convoy member and vehicle sequence authentication in vanets. In SecureComm, pages 422–432, 2007.

[10] Philippe Golle, Daniel H. Greene, and Jessica Staddon. Detecting and correcting malicious data in vanets. In Kenneth P. Laberteaux, Raja Sengupta, Chen-Nee Chuah, and Daniel Jiang, editors, Vehicular Ad Hoc Networks, pages 29–37. ACM, 2004

[11] Mainak Ghosh, Anitha Varghese, Arobinda Gupta, Arzad Alam Kherani, and Skanda N. Muthaiah. Detecting misbehaviors in vanet with integrated root-cause analysis. Ad Hoc Networks, 8(7):778–790, 2010.

[12] Vulimiri, A., Gupta, A., Roy, P., Muthaiah, S.N., Kherani, A.A.: Application of secondary information for misbehavior detection in VANETs. IFIP. LNCS, vol. 6091, pp. 385–396. Springer, Berlin (2010)

[13] Harit, S.K., Singh, G., Tyagi, N.: Fox-hole model for data-centric misbehavior detection in VANETs. In: 3rd International Conference on Computer and Communication Technology (ICCCT), pp. 271–277 (2012)

[14] Ruj, S., Cavenaghi, M.A., Huang, Z., Nayak, A., Stojmenovic, I.: On data-centric misbehavior detection in VANETs. In: Vehicular Technology Conference (VTC Fall), IEEE, pp. 1–5 (2011)

[15] . Grover J., Prajapati N.K, Laxmi V., Gaur M.S: Machine learning approach for multiple misbehavior detection in VANET. In: CCIS, vol. 192, pp. 644–653. Springer, Berlin (2011)

[16] Grover J., Laxmi V., Gaur M.S. Misbehavior detection based on ensemble learning in VANET. In: ADCONS. LNCS, vol. 7135, pp. 602–611. Springer, Berlin (2011)

[17] Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2), 301-320.

[18] Quinlan, J. R. (1996, August). Bagging, boosting, and C4. 5. In *AAAI/IAAI, Vol. 1* (pp. 725-730).

[19] Freund, Y., & Mason, L. (1999, June). The alternating decision tree learning algorithm. In *icml* (Vol. 99, pp. 124-133).

[20] Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, *38*(4), 367-378.

[21] Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5-32.

[22] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794). ACM.

[23] Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.