Analysis and Detection of Outliers due to Data Falsification Attacks in Vehicular Traffic Prediction Application

Raj Mani Shukla* and Shamik Sengupta†
Department of Computer Science and Engineering, University of Nevada, Reno, USA
Email: *rshukla@unr.edu, †ssengupta@unr.edu

***Abstract*—This paper presents an analysis of the vehicular traffic prediction system under the data falsification attack. The proposed analysis shows the effect of orchestrated attacks on Deep Neural Network (DNN) based traffic prediction algorithm. We provide an efficient model for traffic data distribution and management. Based on the proposed model, we investigate traffic gradient based anomaly detection algorithm against data falsification attacks. The proposed anomaly detection method uses dynamic time window to find the traffic gradient at different time. The presented approach finds the presence of anomaly and also accurately detects the type of attack launched by an adversary. The simulation results show that the proposed algorithm identifies data falsification attacks with an accuracy of $\approx$ 70% within a very short interval of attack time and about $\approx$ 95% for the longer interval. Furthermore, the algorithm detects the attack type with an accuracy of $\approx 90\%$.**

***Index Terms*—Smart and connected communities, Data falsification attacks, Anomaly detection, Traffic prediction**

I. Introduction

The idea of Smart and Connected Communities (SCC) incorporate smart cities, smart transportation, smart homes, smart learning, and smart agriculture, all tied together through the Internet connection [1], [2]. The SCC is enabled due to the recent developments in technologies like RFID tags or nano-technology based sensors. These sensors can accurately monitor the environmental parameters and also occupy very small space [3]. Thus, they can be easily installed on any device or at any place. The measured parameters from the distributed set of sensors are transferred to the centralized system like the cloud, edge, or SDN controller for processing where it is used for making analytical decisions [1], [4].

One such example is the sensors, loop detectors, or cameras installed throughout the streets of a city and used for gathering information about the traffic count and the speed of the vehicles at different locations [5]. The collected data from sensors is sent to the centralized unit where it is processed to enable making analytical decisions for providing efficient SCC based services. Such as, the trend and patterns in traffic flow can be observed and based on that accurate forecasting of traffic behavior assists in pro-actively making better travel decisions, traffic congestion control, electric vehicle charging, and intelligent transportation [6].

978-1-5386-7693-6/18/$31.00

There have been several traffic prediction methods in literature to estimate the future traffic count [7]. However, the researchers have used the clean data to determine the future traffic pattern and assumed that the sensors are providing accurate information. The impact of the prediction algorithm when it is fed with the incorrect data has not been investigated in the literature. The measured traffic parameters can be corrupt due to the injection of data falsification attacks. Data falsification attacks are the kind of vulnerabilities where some measured values can be intelligently altered such that the associated system behaves abnormally [8]. Such attack can severely hurt the performance of the prediction method and affect the proactive intelligent decision making by SCC based service providers.

Data falsification attack can either be limited to few sensors or an orchestrated attack on several sensors can be launched. Since the traffic behavior is spatially and temporally correlated, an attacker can intelligently target correlated group of sensor nodes. The attack can be launched by nefarious adversaries, like business rivals or organized criminals [9], seeking long-term benefits. These adversaries are equipped with the resources that can bypass the cryptographic security mechanism [8]. For example, an adversary can alter predicted traffic count near some electric vehicle charging station by changing historical traffic count values in the correlated group of sensors. This can lead to mismatch between energy supplied from the smart grid and actual energy demand at a charging station. Also, the estimation of the wait time for an electric vehicle at a specific charging station may go wrong resulting in charging service provider giving incorrect information to the customers. Albeit, the customer's trust in that service provider can be affected resulting in revenue loss for that company.

The problem of determining anomalous data points due to data falsification attack in traffic count is complicated because of several reasons. The amount of data from the sensors is of enormous quantity such that manually analyzing the data is infeasible. Additionally, the possible types of attacks that an attacker can introduce is generally not known in advance. Moreover, the traffic count is affected by external factors like climatic conditions, events, construction, or emergency situations. These conditions may cause the traffic count to behave abnormally. Therefore, distinguishing whether the abnormal number is due to data falsification attack or external factors

complexifies detection of such vulnerabilities.

This paper addresses the issue of data falsification attack for the traffic prediction system. Since deep learning has been found as an efficient technique to predict both short and long-term traffic accurately, [10], [11], [7], we use a multilayer neural network as our traffic prediction algorithm. We propose a hierarchical data flow and data management model consisting of sensors, edge server, and cloud. The proposed model describes how an adversary can affect the applications hosted at the edge server. We investigate the impact of data falsification attack on the performance of the deep neural network algorithm. We also discuss the amount false injection that an attacker should introduce to cause any noticeable error in prediction performance. Further, the paper provides a novel gradient-based anomaly detection method to detect the time for which a sensor is under data falsification attack. The proposed detection method also finds the type of the attack injected by an adversary.

The rest of this paper is organized as follow. Following this section, in section II, a brief literature survey has been described. Section III presents the proposed system model. System IV explains the prediction performance. In section V, we describe the problem statement and proposed algorithm. In section VI, simulation experiments and results are presented and in section VII we conclude this paper.

II. Literature Survey

There has been work done in the literature regarding anomaly detection. Supervised learning is one of the technique to differentiate between anomalous and non-anomalus data points. In [12], Malhotra et al. has proposed an LSTM network-based anomaly detection in time series data. The given network is trained for the non-anomalous data and is used as a predictor. The obtained error is used to find the anomalous behavior. Cheng et al. has described the anomaly detection for BGP traffic using MS-LSTM network. The author has explained that processing of the traffic data over a suitable time frame can significantly improve the traffic prediction accuracy. Although, LSTM network-based approach is suitable for capturing features of a time series model, they use labeled set of data for training the network. Often the labeled set of anomalous data is not known, and hence the approaches as mentioned above are valid only when the training set data is pre-known in advance.

Principal Component Analysis (PCA) is also a popular technique for anomaly detection. Netflix has proposed an outlier detection method using the PCA [13]. Hendrycs et. all has proposed the detection method of adversarial images using PCA in [14]. The work has shown that the adversarial examples put more focus on the low ranked principal components. However, the PCA based methods are based on the assumption that dominant vectors in high-dimensional data are orthogonal to each other. The assumption cannot be always true, and hence, PCA cannot be used for all kinds of data-set.

Clustering is another popular technique to detect anomaly as shown by Li et al. in [15] and Pandeeserai et al. in [16]. In [15], Li et al. has investigated a novel feature representation approach, viz. cluster center and nearest neighbor (CANN). The method outperforms than the traditional k-NN or SVM based approach. In [16], Pandeeserai et al. has proposed an anomaly detection method using a hybrid of Fuzzy C-Means clustering algorithm and Artificial Neural Network (FCM-ANN). The presented method outperforms than the Naïve Bayes classifier and classic Artificial Neural Network (ANN) even for low frequency attacks. However, as presented by Ahmed et al. in [17], clustering method is not always suitable as any new valid data may also be considered anomaly.

This paper proposes a gradient based technique to detect anomaly in vehicular traffic. The presented method uses a dynamic window size for identifying outliers in the data set. In contrast to the aforementioned works, the proposed work use gradient-based information and does not rely on the orthogonality of the data set. Moreover, rather than supervised learning approach like LSTM networks, the proposed method does not use pre-known anomalous dataset for training.

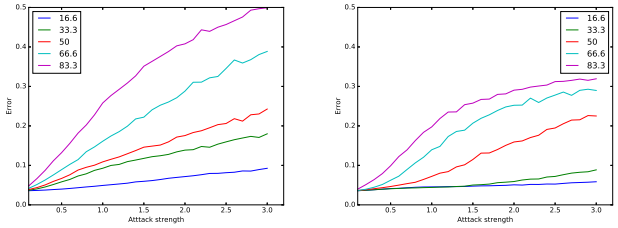
III. System model

System model consists of en-route vehicles in a specific part of the city. The road network is divided into zones based on their spatial location. Loop detectors are installed along the road to measure traffic volume and speed of the vehicles. The loop detectors continuously measure the data and the aggregate data over a periodic time interval τ is sent to the centralized edge server. Edge server brings the cloud computing capabilities near the user and is used for analysis and storage of the data [18], [19]. The proposed edge server has analytical modules like advance traffic prediction that predicts the future traffic counts and thus supports applications like charging station allocation for PEV charging [20], efficient traffic routing, or traffic speed management [21]. The processed information is transmitted either to the vehicle's dashboard or the user's mobile phone. The filtered information from the edge server is sent to the remote cloud for making long-term analytical decisions like city planning and management.

An adversary may target a group of loop detectors over a time interval Γ_a such that data sensed from the loop detectors is corrupted, and the incorrect information is sent to the edge server. Because of that, the analytics hosted at the edge platform performs adversely affecting the performance of different services. Additionally, the small anomalies added at the sensors get cumulated at the cloud and may also impact the long-term decision making at the cloud. To address this issue, we propose an anomaly detector that is placed in the edge server. The incoming data from loop detectors is first analyzed by the anomaly detector to find if it is a clean data. After analysis, filtered data is sent to the analytic applications for assisting different services.

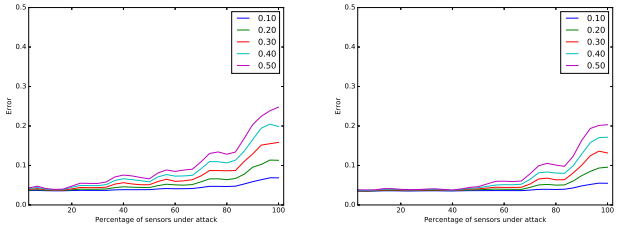
IV. Prediction performance

We evaluate the performance of the Deep Neural Network (DNN) under the condition that an adversary modifies the



(a) Additive attack (b) Deductive attack

Fig. 1: Effect of attack as magnitude of attack is varied



(a) Additive attack (b) Deductive attack

Fig. 2: Effect of attack as percentage of values in input vector are varied

sensor data. DNN has been widely used for traffic road-side traffic prediction [7]. We use multi-layer perceptron based DNN trained using backpropagation algorithm. The given neural network uses the historical traffic from a group of sensors to predict future traffic at a particular location. The network provides an accuracy of 95% when clean data is used as an input to the neural network. We modify the input to the DNN, during its operation, randomly, either by a positive (additive attack) or negative (deductive attack) value. We assume that the attack is not introduced during the training phase of the network. The attack is injected while the neural network is used for the prediction application. We examine the error variation at the output node as attack strength or percentage of values in the input vector is varied.

Figure 1 shows the performance of the DNN for the case when attack strength is varied for different percentage of values in the input vector. Here, it should be noted that the input vector contains the data coming from the loop detectors (that may be under attack). As expected the prediction error increases for the higher percentage of input vector values under attack and greater attack strength. Figure 1a and 1b show that to cause significant variation in error either attack strength has to be high or a large percentage of values in the input vector have to be altered. For example, in Figure 1a if an adversary injects an attack of strength 0.5 (normalized value) per input value then if 83.3% of values in the input vector are changed the error goes above 10%. Otherwise, for the low percentage of input vector values under attack, the prediction error is in the range of 5%.

Figure 2a and 2b present the scenario when the percentage of sensors under attack are varied for different attack strength. The figure depicts that irrespective of attack strength, the error does not increase for less percentage of input vector values under attack. However, for both the additive and deductive attacks, as the number of sensors increases above a threshold point, the error suddenly starts increasing. Specifically, for additive attacks, at least 20% of input vector values feeding input to the DNN need to be targeted. Similarly, for the deductive attack, the number of input vector values targeted should be around 40%.

The mentioned analysis reveals that the point anomaly such that anomaly affecting few values are insignificant for vehicular traffic prediction application. Since, DNN for traffic prediction takes input from spatially correlated sensors, to cause considerable traffic prediction error an adversary need to target a large number of sensors or few sensors for a significant time interval, thereby affecting the high percentage of values input to the DNN. Correspondingly, we develop a technique to detect the anomalies in data values over a considerable time interval from a sensor. The proposed method can be repeated for sensors located at different locations to find the spatially separated anomalies.

V. Methodology

This section describes the proposed gradient-based anomaly detection method. Before proceeding to the technique, we describe gradient and threshold determination that we use in the process.

A. Preliminaries

We divide the time into slots where the length of each slot is τ. Every loop detector sends the aggregate traffic measured in a time interval of τ and transfers the information to the edge server. We use the gradient of traffic flow between different time slots. However, rather than finding the traffic gradient between two consecutive time slots, we use a window of dynamic length l where l varies between $1, 2, ...L$.

The traffic count depends on several factors that include weekday, weekend, the point of interest, external weather conditions, and other road-side factors. Therefore, the traffic at a specific location can be either high or low depending on multiple factors. For example, Figure 3 shows the traffic count at four different locations in the same highway segment. The values of the traffic count at different locations differ to a large extent. Also, in a same location, the traffic deviates to a considerable extent. For instance, the peak and trough values vary considerably within a location. Furthermore, the behavior of traffic is also different at different points. For example, at location 4 traffic is highly fluctuating near peak values but the curve has some flatness near location 1.

In order to avoid the effect of above inherent characteristic in anomaly detection technique, we use traffic gradient for different window size l. We use variable d_i^l to represent gradient at the time slot i for window size l. The value of d_i^l is found using equation 1. Equation 1 determines the absolute difference between mean traffic for time slot $i \rightarrow i + l$ and $i + 1 \rightarrow i + l + 1$. Here, n_i is the true traffic count value at time slot i.

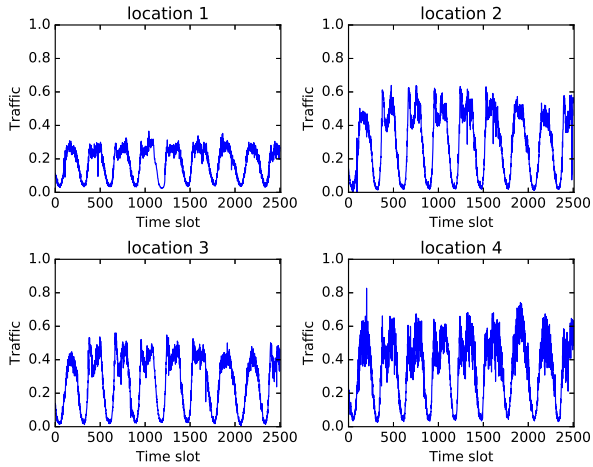


Fig. 3: Traffic at a particular location

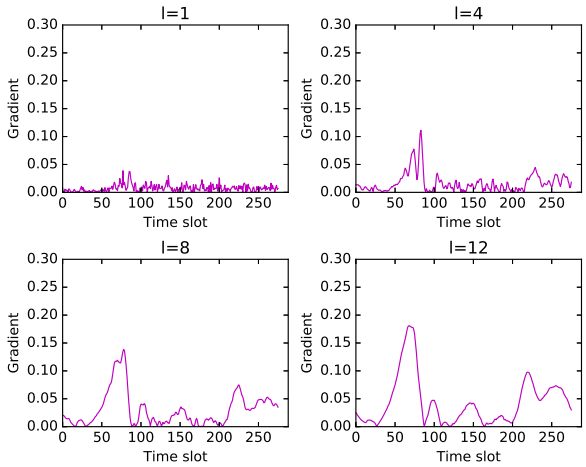


Fig. 4: Gradient plot for different window size

$$d_i^l = \frac{|\Sigma_{i+1}^{i+l+1} n_i - \Sigma_i^{i+l} n_i|}{l} \tag{1}$$

Figure 4 shows the traffic gradient for different values of window size l. The figure depicts that the gradient values are low for lower window size showing traffic changes very slowly between different time slots. As window size is increased, the gradient also increases, but the curve becomes smoother. Thus, for the lower window size, although the gradient is small, its behavior is random. For the large window, the gradient becomes large, and its curve becomes smoother.

We use traffic gradient information at different locations for every window size to find the threshold values with regard to the traffic changes. Then, based on the threshold values we find the anomalous data points in the traffic patterns. Furthermore, since for lower l values, the gradient is smaller, a lot of true values can also be counted as anomaly. In contrast, for large l value some anomaly can be escaped due to higher gradient value. Therefore the gradient size is determined dynamically between window size of 1 to L at the runtime.

B. Threshold values

We determine the threshold values for each window size l. For every l value, we determine the threshold of the gradient based on different locations. We use the concept that if traffic gradient is more than the threshold value, then there is a possibility of an attack at that point of the time.

To find the the threshold values, we use a tuple $diff$ where $diff_l$ is the l_{th} list in $diff$ and $diff_i^l$ is the i_{th} element of $diff_l$. Value of $diff_i^l$ is given by the equation 2.

$$diff_i^l = |\frac{\Sigma_1^D \Sigma_i^{i+l} n_i - \Sigma_1^D \Sigma_{i+l+1}^{i+l+1+l} n_i}{D \times l}| \tag{2}$$

Here, D are the total number of days considered to take the average value. For every $diff_l$, we calculate the maximum value of the set $(\max(diff_l))$ and append it to threshold lists $thList_l$. We consider a tuple $thList$ with $thList_l$ is the l_{th} set of tuple $thList$ and $thList_i^l$ is the i_{th} element in set $thList_l$. We append such maximum values obtained for every location p (where loop detector is installed) that are sending data to the edge server. Once we have searched $\max(diff_l)$ for all positions, the threshold for window l is obtained as in equation 3.

$$th_l = \frac{\Sigma_{i=1}^P thList_i^l}{P} + std(thList_l) \tag{3}$$

Here, P are the total number of locations and $std(thList_l)$ is the standard deviation of the elements in list $thList_l$. We use the threshold values th_l to dynamically decide the possible attack points and their type.

C. Problem statement

Given the true traffic counts $n_1, n_2, ... n_N$, an adversary can inject the noise $\delta_i, \delta_{i+1}....\delta_{i+j}$ such that the traffic between time slots i and j is added or deducted by the quantity $\delta_i, \delta_{i+1}....\delta_{i+j}$. The traffic after an attack is represented by $\eta_1, \eta_2.....\eta_N$. The adversary may launch data falsification attack by adding noise to the true traffic count at a particular location within a time interval Γ_a. The attack starts at the time slot $i = \nu_s$ and ends at the point $j = \nu_e$ and its type is κ_a.

The problem statement is to detect start point ϑ_s, end point ϑ_e, interval Γ_d, and attack type κ_d while optimizing equation 4, s.t. the constraints given by the equation 5.

$$\begin{aligned} min(\overline{|\vartheta_s - \nu_s|}) \ \& \ min(\overline{|\vartheta_e - \nu_e|}) \ \& \\ min(\overline{|\Gamma_a - \Gamma_d|}) \ \& \ min(\overline{|\kappa_a - \kappa_d|}) \end{aligned} \tag{4}$$

s.t.

$$|\vartheta_s - \nu_s| < m, \ |\vartheta_e - \nu_e| < m, \ |\Gamma_a - \Gamma_d| < 2m \tag{5}$$

Here, m is a hyper-parameter which we call as the error margin such that if the difference between actual attack time slot and detected time slot is less than m then the anomaly point is considered to be detected.

Algorithm 1 Anomaly detection algorithm

Input: $< \eta_1, \eta_2 ... \eta_N >$- Set of a traffic counts
$< th_1, th_2 ... th_L$- Set of threshold values
Output: Γ_d, κ_d, ϑ_s, ϑ_e

1: **for** $l \leftarrow 1$ **to** L **do**
2: **for** $1 \leftarrow 1$ **to** N **do**
3: **if** $d_i^l > th_l$ **then**
4: $A_l.append(i+1)$
5: $S_l.append(sign(d_i^l))$
6: **end if**
7: **end for**
8: $Replace\ by\ Centroid$
9: $Sign\ Processing$
10: **end for**
11: Find $X \in A$ with $X.cardinality = 2$ and $X.window\_size$ is minimum
12: Find $Y \in S$ with same window size as X
13: **if** $X! = Null$ **then**
14: $\vartheta_s = X[0], \vartheta_e = X[1]$
15: $FindAttackType(Y)$
16: **else**
17: Find $X \in A$ with $X.cardinality = 3$ and $X.window\_size$ is minimum
18: Find $Y \in S$ with same window size as X
19: r = Random(0, 1)
20: **if** $r <= 0.5$ **then**
21: $\vartheta_s = X[0], \vartheta_e = X[1]$
22: $FindAttackType(Y)$
23: **else**
24: $\vartheta_s = X[1], \vartheta_e = X[2]$
25: $FindAttackType(Y)$
26: **end if**
27: **end if**

D. Anomaly detection

Algorithm 1 explains the steps for finding the anomaly in a traffic data where an adversary may have modified the real data. The algorithm takes the traffic count and threshold values for different window size and provides interval Γ_d, time slot at which attack starts ϑ_s, time slot when the attack ends ϑ_e, and attack type κ_d.

In algorithm 1, steps 2-9 are repeated for all window size ranging from length $1 \rightarrow L$. Steps 3-6 are repeated for every time slot values. For every l value, we determine if the traffic gradient at that level is greater than the threshold value for the same level. For this purpose, we use a tuple A where $A_l \in A$. The set A_l contains the potential list of points where an anomaly can be present. To find the actual start and end point of an anomaly, we use the tuple A. We compare the d_i^l with th_l for every time slot $i \in 1 \rightarrow N$ (step 3). If the d_i^l is greater than the th_l then the point i is appended to the list A_l such that the point i may have the attack possibility (step 4). In step 5, we determine the sign (positive or negative) of d_i^l and append it to the list S_l that is used for determining the attack type. Hereby, S is a tuple such that $S_l \in S$ and stores sign of traffic gradients.

Steps 8 and 9 processes set A_l and S_l to remove the redundant values. In step 8, set A_l and S_l are processed according to values in set A_l. Step 9 processes the A_l and S_l according to the values in S_l.

Since we use a window size of l, an attack that starts (or ends) at a certain time slot ν_s (or ν_e) affects the gradient before $\approx l$ and after $\approx l$ slots (such that from time slot $\approx \nu_a - l \rightarrow \approx \nu_a + l$). We use approximate values because due to the noise some extra time slots may be added or dropped in the set. Therefore, we replace the occurrence of consecutive values in set A_l that differ by one to their centroid point (step 8). For example, if A_l has the elements $\{2, 3, 4, 8, 9, 10, 11, 150\}$, then after the replacement process the set will be modified to $\{3, 9, 150\}$. We do not drop single points, and for even number of groups, the lower centroid is kept. We also process the S_l by keeping the corresponding elements as in processed set A_l and dropping others. For example, for every element in processed set A_l, we keep the element from S_l having the same index as the element has in original set A_l (before processing). For the mentioned example, if S_l before processing has elements $-1, -1, 1, 1, 1, -1, -1, 1$ then after processing it will contain $\{-1, 1, 1\}$. Hereby, it should be noted that the length of the set A_l will always be equal to the length of the set S_l.

In step 9, we further process the sets A_l and S_l according to the values in the set S_l. We determine the consecutive occurrence of values in S_l with the similar sign and replace them by a single value. In set A_l, the values in the corresponding group with the highest index is kept. In given example, the values in the set $\{-1, 1, 1\}$ are updated after processing to $\{-1, 1\}$. Here, the consecutive occurrence of one is replaced by the single value. The set A_l is updated to $\{3, 50\}$. Thus, after processing set S_l contains values such that the consecutive values have opposite sign. The set A_l contains the possible time slots where an attack may have started or ended. The given processing of sets A_l and S_l based on their values (first by A_l and then by S_l) is done for every set $A_l \in A$ and set $S_l \in S$. To determine the anomaly and its type the processed sets A_l and S_l are used.

In step 11, the list of sets A_l in the tuple A having cardinality of 2 and lowest window size l (set X) is determined. Selecting set with lowest window size has an advantage because even small anomalies can be easily detected (due to the low threshold value). We find the corresponding list from set S having same window size as set X and assign it to the variable Y (step 12). X is used to determine the start and end points of attack such that ϑ_s and ϑ_e values respectively. Y is used to determine the attack type κ_d. If the set X exists (step 13) then, the detected start point ϑ_s is the 0_{th} element of X and end point ϑ_e is the 1^{st} element in X (step 14). In step 15, the attack type κ_d is determined from the corresponding variable Y. If the value of Y is $\{1, -1\}$, the attack type is additive signifying first the gradient increases and after the attack ends gradient decreases. Otherwise, if the value of Y is $\{-1, 1\}$ the attack is deductive in nature.

If set A_l with length two does not exist in the tuple A then we determine the anomalous values from the set having the

cardinality of 3 and with the lowest window size (steps 17 and 18). The start point of attack ϑ_s is chosen between 0_{th} and 1^{st} element with a probability of 0.5. In step 19 and if block 20-26, the variable r is a random variable and used to select either 0_{th} or 1_{st} element as the start point with a probability of 0.5. If 0_{th} element is the start point ϑ_s then 1^{st} element is the endpoint ϑ_e of the attack (step 21). Step 22 finds the attack type In this case, if the set Y contains $\{1, -1, 1\}$ then the attack is additive otherwise if set elements are $\{-1, 1, -1\}$ then the attack is deductive. On the other hand, if 1^{st} element is the start point, the 2^{nd} one is chosen as the endpoint (step 24). In tep 25, the sign of corresponding elements in Y determines attack type.

The above process is repeated for different locations to find the anomalies for the sensors located at that specific location.

VI. Results

A. Simulation parameters

We evaluate the performance of the proposed algorithm using data obtained from the Performance Measurement System (PeMs) that gathers freeway traffic of California Highways [5]. We collect the traffic for October 2017 for the freeway segment of I-680 at 136 locations. We develop python based frameworks both for the evaluation of proposed anomaly detection method. The obtained traffic data is normalized between the zero and one using the min-max method.

Since the labels for anomalous data was not available, we simulate it by randomly generating attack data. The attack is simulated by randomly selecting the location, day, and the time interval of the attack. The performance of the proposed method is evaluated both for additive and deductive types of attacks.

B. Anomaly detection

Table I-IV presents the performance of attack detection method for different values of attack strengths χ and error margin m. Table I and II show the results as attack strength χ is varied for additive and deductive attack respectively for fixed error margin m of 10. Table III and IV presents the performance for different error margin for additive and deductive attacks for fixed attack strength of 0.20. In each table, the first column represents the value of the variable parameters such that the attack strength χ or error margin m. $\overline{Fs}$ and $\overline{Fe}$ end provides average percentage of time the algorithm detects start and end time incorrectly. $\overline{Fs.Fe}$ represents number of times both start and end values are detected incorrectly. The start or end values are considered to be incorrect if the difference between the actual time slot at which attack starts or ends differs the detected values by more than the error margin m. $\overline{|\vartheta_s - \nu_s|}$ presents the average absolute difference between actual and detected start slot of the attack. Similarly, $\overline{|\vartheta_e - \nu_e|}$ presents the average value of the absolute difference between actual and detected end slot of the attack. We refer $\overline{|\vartheta_s - \nu_s|}$ and $\overline{|\vartheta_e - \nu_e|}$ as *Detection margin* values. $\overline{|\kappa_a - \kappa_d|}$ presents the average percentage of time the attack type is incorrectly detected.

TABLE I: Attack detection for different amount of injected attack strength for additive attack

χ	$\overline{Fs}$	$\overline{Fe}$	$\overline{Fs.Fe}$	$\overline{\lvert\vartheta_s - \nu_s\rvert}$	$\overline{\lvert\vartheta_e - \nu_e\rvert}$	$\overline{\lvert\kappa_a - \kappa_d\rvert}$
0.15	27.1	31.1	16.5	0.57	0.51	1.7
0.20	20.6	29.0	11.9	0.56	0.45	2.1
0.25	21.7	28.7	12.3	0.5	0.41	1.8
0.30	20.5	28.3	11.9	0.44	0.29	1.2
0.35	20.4	27.2	10.0	0.44	0.3	1.1
0.40	19.7	26.8	11.3	0.41	0.25	2.0
0.45	21.3	26.1	10.7	0.35	0.26	1.0
0.50	18.3	25.7	9.1	0.36	0.24	0.7
0.55	19.9	28.8	10.5	0.33	0.19	1.1
0.60	21.4	27.6	11.5	0.42	0.18	0.9

TABLE II: Attack detection for different amount of injected attack strength for deductive attack

χ	$\overline{Fs}$	$\overline{Fe}$	$\overline{Fs.Fe}$	$\overline{\lvert\vartheta_s - \nu_s\rvert}$	$\overline{\lvert\vartheta_e - \nu_e\rvert}$	$\overline{\lvert\kappa_a - \kappa_d\rvert}$
0.15	32.3	40.2	25.7	0.35	0.48	17.2
0.20	22.5	32.3	17.5	0.36	0.39	8.9
0.25	19.6	31.5	14.9	0.35	0.5	5.3
0.30	19.5	28.8	14.9	0.31	0.41	3.0
0.35	19.4	30.1	12.8	0.32	0.41	1.3
0.40	17.5	28.2	13.0	0.28	0.28	1.3
0.45	18.7	29.4	12.9	0.3	0.37	0.8
0.50	16.0	27.4	12.1	0.19	0.42	0.8
0.55	18.6	29.7	13.6	0.29	0.34	0.8
0.60	17.0	26.3	11.9	0.23	0.32	0.3

As depicted in table I and II, the percentage of false start varies between 16% to 32.3%, that is detection accuracy ranges 67.7% to 86% for false start values. Regarding false end, the percentages are in the range of 59.8%-73.9%. Here, it should be noted that although accuracy seems to low, the error margin is also considered to be the low value of 10. If we look at the fourth column of both table, the accuracy is much higher at around 85-90%. This portrays that even if the algorithm may detect the false start or endpoints with $\approx 70\%$ accuracy, the detection accuracy of both false start and end points is more than 90%. That is the proposed method can accurately determine at-least start or end of the attack with very high accuracy.

Furthermore, the absolute average detection margin as shown in column 5 and 6 is less than a one-time slot. The low detection margin indicates that the attack is detected within a single time slot. Column 7 shows the attack type accuracy. Here, the algorithm identifies the attack type with a very high accuracy of more than 97.9%. The results deduce that the detection margin and attack type are found with very high accuracy. Additionally, if the error margin is kept very low, the percentage of false start or endpoint detection may be significant, the percentage of both false start and end is low showing that at least one point is detected accurately.

We further evaluate the performance as the error margin m is changed for the fixed attack strength of 0.20 in table III and IV. Results reveal that the percentage of both false start and end point of attack decreases considerably as we increase the margin. Henceforth, the accuracy reaches around 10% for larger values of error margin. However, as we can see in column 4 and 5, the detection margin is within an acceptable range. For example, in table III for the error margin value of

TABLE III: Attack detection for different error margin for additive attack

m	$\overline{F_s}$	$\overline{F_e}$	$\overline{F_s}.\overline{F_e}$	$\overline{\lvert\vartheta_s - \nu_s\rvert}$	$\overline{\lvert\vartheta_e - \nu_e\rvert}$	$\overline{\lvert\kappa_a - \kappa_d\rvert}$
10	21.1	28.4	10.8	0.44	0.42	1.5
20	18.7	25.9	10.1	1.39	0.99	1.9
30	14.0	24.8	8.6	2.67	1.45	2.2
40	10.0	21.3	5.8	3.66	3.1	3.7
50	6.7	19.6	4.2	5.17	3.82	3.8
60	5.1	17.1	3.2	6.39	5.47	3.1
70	4.7	15.8	3.3	7.91	6.23	3.3
80	3.4	14.8	3.3	8.15	7.52	3.9
90	2.6	12.7	2.5	8.01	10.51	3.7
100	1.9	10.0	1.7	8.27	12.42	4.2

TABLE IV: Attack detection for different error margin for deductive attack

m	$\overline{F_s}$	$\overline{F_e}$	$\overline{F_s}.\overline{F_e}$	$\overline{\lvert\vartheta_s - \nu_s\rvert}$	$\overline{\lvert\vartheta_e - \nu_e\rvert}$	$\overline{\lvert\kappa_a - \kappa_d\rvert}$
10	25.5	33.3	20.2	0.3	0.6	9.8
20	21.7	30.3	17.7	0.86	1.05	10.0
30	18.4	27.3	13.6	1.35	2.81	10.1
40	14.7	21.6	10.7	2.46	3.86	10.3
50	13.5	21.5	10.0	3.67	5.33	12.6
60	12.1	15.8	6.7	3.75	7.99	11.1
70	10.1	13.4	5.3	5.37	9.52	12.8
80	7.4	10.0	3.7	6.91	11.95	13.3
90	5.8	8.9	2.9	7.29	11.75	12.5
100	5.1	7.4	2.9	7.96	12.78	12.5

50, the average number of false start and false end points are 6.7% (93.3% accuracy) and 19.6% (80.4% accuracy). The combined accuracy is 95.8%. Interestingly, the detection margin is only 5.17. Thus, the anomaly is detected with very high accuracy within about five time slots. Henceforth, we can deduce that by increasing the error margin, we can achieve very high accuracy. Also, increasing the error margin does not deviate the detection margin to a substantial value. A considerable error margin (100) enables the algorithm to detect attack with very high accuracy (around 95%), but the detection margin is 7.97 and 12.78 for start and end value respectively (Table IV).

VII. Conclusions

This paper has emphasized the importance of finding a detection method for orchestrated data falsification attack. We provided a novel gradient-based anomaly detection method that does not rely on labeled dataset or orthogonal vectors in high dimensional points. The given methods perform with decent accuracy when error margin is kept low. The accuracy increases on relaxing the error margin while still maintaining the detection margin not to deteriorate significantly. In the future, we plan to analyze our model for other possible anomalies. The steps taken to subsidize the values of errors to make the correct prediction in the presence of attack also need to be investigated.

Acknowledgement

This research is supported by NSF Award #1723814.

References

[1] R. Shukla, S. Sengupta, and M. Chatterjee, "Software-defined network and cloud-edge collaboration for smart and connected vehicles," in *Proceedings of the Workshop Program of the 19th International Conference on Distributed Computing and Networking*, ser. Workshops ICDCN '18, 2018, pp. 6:1–6:6.

[2] S. Vakilinia, I. Vakilinia, and M. Cheriet, "Green process offloading in smart home," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, October 2017, pp. 1–5.

[3] M. Wu, T. Lu, F. Ling, J. Sun, and H. Du, "Research on the architecture of internet of things," in *Proc. of IEEE International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5, August 2010, pp. V5–484–V5–487.

[4] R. Shukla, S. Sengupta, and A. Patra, "Smart plug-in electric vehicle charging to reduce electric load variation at a parking place," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, January 2018, pp. 632–638.

[5] Pems. [Online]. Available: http://pems.dot.ca.gov/

[6] B. Pan, U. Demiryurek, and C. Shahabi, "Utilizing real-world transportation data for accurate traffic prediction," in *IEEE 12th International Conference on Data Mining*, December 2012, pp. 595–604.

[7] N. G. Polson and V. O. Sokolov, "Deep learning for short-term traffic flow prediction," *Transportation Research Part C: Emerging Technologies*, vol. 79, pp. 1 – 17, 2017.

[8] S. Bhattacharjee, A. Thakur, and S. K. Das, "Towards fast and semi-supervised identification of smart meters launching data falsification attacks," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18, 2018, pp. 173–185.

[9] Ami system security requirements. [Online]. Available: https://www.energy.gov/sites/prod/files/oeprod/DocumentsandMedia/14-AMI_System_Security_Requirements_updated.pdf

[10] J. Guo, Z. Wang, and H. Chen, "On-line multi-step prediction of short term traffic flow based on gru neural network," in *Proceedings of the 2nd International Conference on Intelligent Information Processing*, ser. IIP'17, 2017, pp. 11:1–11:6.

[11] Y. Wu, H. Tan, L. Qin, B. Ran, and Z. Jiang, "A hybrid deep learning based traffic flow prediction method and its understanding," *Transportation Research Part C: Emerging Technologies*, vol. 90, pp. 166 – 180, 2018.

[12] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*. Presses universitaires de Louvain, 2015, p. 89.

[13] Rad - outlier detection on big data. [Online]. Available: https://medium.com/netflix-techblog/rad-outlier-detection-on-big-data-d6b0494371cc

[14] D. Hendrycks and K. Gimpel, "Early methods for detecting adversarial images," *arXiv preprint*, 2016.

[15] W.-C. Lin, S.-W. Ke, and C.-F. Tsai, "Cann: An intrusion detection system based on combining cluster centers and nearest neighbors," *Knowledge-based systems*, vol. 78, pp. 13–21, 2015.

[16] N. Pandeeswari and G. Kumar, "Anomaly detection system in cloud environment using fuzzy clustering based ann," *Mobile Networks and Applications*, vol. 21, no. 3, pp. 494–505, Jun 2016.

[17] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19 – 31, 2016.

[18] M. Patel, Y. Hu, P. Hédé, J. Joubert, C. Thornton, B. Naughton, J. R. Ramos, C. Chan, V. Young, S. J. Tan, D. Lynch, N. Sprecher, T. Musiol, C. Manzanares, U. Rauschenbach, S. Abeta, L. Chen, K. Shimizu, A. Neal, P. Cosimini, A. Pollard, and G. Klas, "Mobile-Edge Computing," in *ETSI White Paper*, September 2014.

[19] R. Shukla and A. Munir, "An efficient computation offloading architecture for the internet of things (iot) devices," in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2017, pp. 728–731.

[20] R. Shukla and S.Sengupta, "A novel software-defined network based approach for charging station allocation to plugged-in electric vehicles," in *Proc. of IEEE International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA, 2017, pp. 437–441.

[21] A higher technical standard for the austrian autobahn. [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/security/us-asfinag-case-study.pdf