2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS) | 978-1-6654-0386-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/RTAS52030.2021.00027

Real-Time Adaptive Sensor Attack Detection in Autonomous Cyber-Physical Systems

Francis Akowuah and Fanxin Kong
Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse NY
feakowua@syr.edu, fkong03@syr.edu

***Abstract*—Cyber-Physical Systems (CPS) tightly couple information technology with physical processes, which rises new vulnerabilities such as physical attacks that are beyond conventional cyber attacks. Attackers may non-invasively compromise sensors and spoof the controller to perform unsafe actions. This issue is even emphasized with the increasing autonomy in CPS. While this fact has motivated many defense mechanisms against sensor attacks, a clear vision on the timing and usability (or the false alarm rate) of attack detection still remains elusive. Existing works tend to pursue an unachievable goal of minimizing the detection delay and false alarm rate at the same time, while there is a clear trade-off between the two metrics. Instead, we argue that attack detection should bias different metrics when a system sits in different states. For example, if the system is close to unsafe states, reducing the detection delay is preferable to lowering the false alarm rate, and vice versa. To achieve this, we make the following contributions.**

In this paper, we propose a real-time adaptive sensor attack detection framework. The framework can dynamically adapt the detection delay and false alarm rate so as to meet a detection deadline and improve the usability according to different system status. The core component of this framework is an attack detector that identifies anomalies based on a CUSUM algorithm through monitoring the cumulative sum of difference (or residuals) between the nominal (predicted) and observed sensor values. We augment this algorithm with a drift parameter that can govern the detection delay and false alarm. The second component is a behavior predictor that estimates nominal sensor values fed to the core component for calculating the residuals. The predictor uses a deep learning model that is offline extracted from sensor data through leveraging convolutional neural network (CNN) and recurrent neural network (RNN). The model relies on little knowledge of the system (e.g., dynamics), but uncovers and exploits both the local and complex long-term dependencies in multivariate sequential sensor measurements. The third component is a drift adaptor that estimates a detection deadline and then determines the drift parameter fed to the detector component for adjusting the detection delay and false alarms. Finally, we implement the proposed framework and validate it using realistic sensor data of automotive CPS to demonstrate its efficiency and efficacy.

***Index Terms*—autonomous cyber-physical systems, security, physical attacks, real-time, detection**

I. Introduction

Autonomous Cyber-Physical Systems (CPS), such as self-driving cars and unmanned aerial vehicles (UAV), are becoming an integral part of our daily lives. For example, Amazon's Prime Air service seeks to use drones to deliver orders up to five pounds in 30 minutes or less and has already demonstrated its feasibility in [1]. UAVs have also been seen in applications such as aerial photography [2], policing and surveillance [3] [4], infrastructure inspections [5], construction site management [6] and many others. Self-driving cars continue to attract huge investments from big companies and they are expected to be in common use in the near future [7] [8].

Due to the safety-critical roles that they play, autonomous CPS security continues to be an essential requirement for its safe functioning. However, due to the tight integration of information technology with physical processes, autonomous CPSs have become susceptible to both cyber and physical attacks. Cyber attacks refer to attacks against the computing and communication CPS components. Physical attacks compromise the physical environment of the CPS to allow injecting malicious signals into sensors and actuators. There are numerous works addressing cyber attacks such as memory isolation [9], software and firmware techniques [10]–[13], control-flow integrity [14] [15], and so on.

Such conventional cybersecurity techniques, however, are inadequate to address physical attacks. This is especially emphasized by non-invasive sensor attacks. These attacks do not require physical access to the target component and have been shown to be easy (requiring a modicum of knowledge) and inexpensive (requiring cheap equipment to execute). Rutkin [16] showed how non-invasive attacks enabled malicious signals to be injected into GPS sensors, and in the end misguided a yacht off course. Similarly, Shoukry et al. [17] demonstrated how non-invasive attacks on wheel speed sensors influenced Anti-lock Braking Systems (ABS) of a vehicle to malfunction. Petit et al. [18] also showed how an automotive CPS camera and LiDAR can be attacked remotely. In addition, the consequences of sensor attacks will be even exaggerated as the autonomy increases.

The urgent need to protect autonomous CPS from physical sensor attacks has motivated a lot of research efforts such as attack-resilient sensor fusion [19]–[21], model-based attack detection [22]–[24], and data-based detection [25]–[30]. However, the timing and usability of attack detection have not been adequately addressed in existing works. This timing constraint is the detection deadline, before which attacks must be detected. The usability refers to the false alarm rate, and a lower (higher) rate means a better (worse) usability. Existing works tend to minimize the detection delay and false alarm rate at the same time. However, the goal is deemed to be unachievable because of the clear trade-off between the two metrics, i.e., lower delay coming with higher false alarm rate, and vice versa [24], [31], [32]. Hence, we believe that attack

978-1-6654-0386-3/21/$31.00 ©2021 IEEE
DOI 10.1109/RTAS52030.2021.00027

detection should have a preference on different metrics when a system runs in different states.

To realize this, we propose a real-time adaptive sensor attack detection framework that can dynamically adjust detection delay and false alarms. The key rationale behind this framework is as follows.

(i) Why real-time? Given safety-critical CPS, timing is important, as untimely defense, that is, detection of an attack after consequences occur, is just as damaging. For example, consider the cruise control function under a speed sensor spoofing attack that changes the true measurement to a smaller value. Then the vehicle is misled to accelerate so that the real speed can be much higher than the desired. This attack needs to be detected before the vehicle crashes into the front car. This timing constraint is referred to as the *detection deadline*, before which attacks must be detected.

(ii) Why adaptive? On the one hand, a shorter detection delay is not always favorable. In the end, we can have an attack detector that raises an alert at every control period. The detector will discover an attack once it occurs, and thus the detector has the shortest detection delay. However, this will give an unmanageable number of false alarms and thus unacceptably low usability. On the other hand, an alert can be raised after monitoring multiple control periods to ascertain the occurrence of an attack. However, this can lead to increased detection delay. Hence, we argue that there is a need to adapt the attack detection so that it can make the appropriate trade-off. For example, if the system is already close to unsafe states and thus the detection deadline is stringent, reducing the detection delay will be preferable to lowering the false alarm rate, and vice versa.

To enable real-time adaptive detection, our attack detection framework consists of three necessary components: attack detector, behavior predictor, and drift adaptor, as shown in Fig. 1. The technical contribution for each component is as follows.

(i) Attack Detector. As the core of our framework, this component detects anomalies using a CUSUM algorithm that monitors the cumulative sum of residuals between the nominal (estimated by the behavior predictor) and observed sensor values. The algorithm will raise an alarm when the cumulative sum of the residuals is greater than a predefined threshold. Importantly, we augment this algorithm with a drift parameter that governs both the detection delay and false alarms. That is, the algorithm can adjust the two metrics by changing the drift parameter.

(ii) Behavior Predictor. This component estimates nominal sensor values that are fed to the core component. It uses a deep learning (DL) model that is offline extracted through uncovering and exploiting both the local and complex long-term dependencies in multivariate sequential sensor measurements. Thus this model depends on little knowledge of the physical system (e.g., dynamics). Further, this model leverages convolutional neural network (CNN) and recurrent neural network (RNN) to capture non-linear aspects in sensor data and uses autoregressive models to capture linear aspects. This combination results in high robustness and scalability in handling the sequential sensor data.

(iii) Drift Adaptor. The third component is a drift adaptor that estimates a detection deadline and then determines the drift parameter. The detector component uses this parameter for adjusting the detection delay to ensure timely detection as the detection deadline varies over time.

We implement our framework and validate it using realistic sensor data of automotive CPS from the AEGIS Big Data Project [33]. The results demonstrate that our framework can detect attacks in a real-time manner. One key insight here is that tuning the drift parameter has little impact on false negatives while the detection deadline can be effectively satisfied.

The rest of this paper is organized as follows. Section II presents a background and system design overview. Sections IV, III, and V detail the design for each component respectively. Section VI validates the proposed framework. Section VII gives further discussions on the applicability of our framework. Section VIII presents the related work. Section IX concludes the paper.

II. Background and System Overview

In this section, we first present the system and threat model, and then briefly describe our real-time detection framework.

A. System and Threat Model

The CPS model we consider in the paper is a physical system, also called a plant, controlled by a controller. The controller operates at every δ unit of time, where $\delta > 0$ is called a control period. At the beginning of every control period, the controller first reads the output of the plant or sensor measurements. Then using a control algorithm, the controller computes the control signals or inputs that are sent to the actuators. The actuators will apply the control inputs to the plant in the current step.

We consider attack scenarios, where the attacker is able to compromise the integrity and availability of sensor data of autonomous CPS, as shown in Fig. 1.

(i) Integrity of Sensor Measurements. The adversary is able to modify the sensor measurements by launching spoofing attacks in the CPS's physical environment such as introducing noise or interference in the signals that the sensor is perceiving. The attacker may also undertake replay attacks to compromise data integrity. A successful replay attack enables an attacker to send previously captured data to the CPS. While the replayed data was valid data at a particular point in the past, it does not reflect the current state of the CPS.

(ii) Availability of Sensor Measurements. The adversary is able to delay the controller from receiving the sensor values. The received values are out-of-date and reflect a historical state of the system. Denial of service (DOS) attacks belong to this kind of attack, where the delay is infinite. Signal jamming is one typical DOS attack that the attacker can execute in the CPS's physical environment.

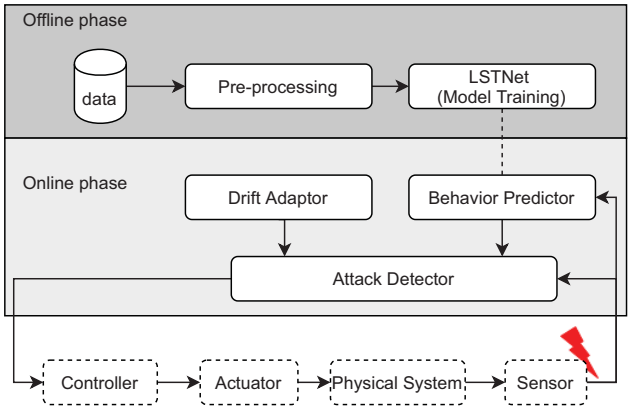


Fig. 1: Design overview of the real-time adaptive sensor attack detection framework.

This paper is focused only on sensor the attacks mentioned above. We thus assume that the adversary does not compromise the controller, the actuator, or other cyber components of the system (cyber attacks). We do not restrict the maximum number of sensors that can be compromised by an attacker but assume that the attacker has no knowledge of our attack detector.

B. Overview of System Design

Fig. 1 shows the overview of the proposed adaptive real-time attack detection framework. It has two phases: an offline training phase and an online detection phase.

The offline phase consists of components that function together to learn the nominal behavior of the system through training a deep learning model. It leverages both the local and complex long-term dependencies that exist among sensor data. To achieve this, the pre-processing component first screens out sensors that are correlated with each other by calculating their pairwise correlations. Then, the Long- and Short-Term Time-Series Network (LSTNet) component captures a consistent pattern among the correlated sensors, which is referred to as the nominal behavior.

The online phase handles the real-time attack detection and is made up of three components. The Behavior Predictor uses the learned model to predict nominal sensor values. In the presence of attacks, sensor measurements (observed) will be different from the predicted values. This difference, called the residual, is tracked by the Attack Detector to identify anomalies. It will raise an alarm when the cumulative sum of residuals becomes larger than a pre-defined threshold. The Drift Adaptor ensures a usable detection result before the detection deadline. The deadline may vary over time as the physical environment changes. This component can dynamically adjust the detection delay to meet the deadline via the drift parameter. To be clear, we state the workflow of the online phase as follows. At each control period, the Behavior Predictor and Drift Adaptor first produce nominal sensor values and the drift value, respectively. Then the Attack Detector uses these values to identify anomalies.

III. Design of Attack Detector

In this section, we present the detailed design of the core component, Attack Detector, in our framework. This component needs predicted sensor values and the drift parameter from Behavior Predictor and Drift Adaptor respectively. The latter two components will be detailed in the subsequent sections.

A. Problem Formulation

We formulate the attack detection problem as follows. Given the predicted nominal sensor value $\hat{y}_t \in \mathbb{R}^n$, observed sensor value $y_t \in \mathbb{R}^n$ and the drift parameter λ, the problem is to determine the appropriate time to raise an attack alert t_{alarm} when the observed sensor values deviate from the expected values such that it exceeds a threshold τ:

$$t_{alarm} = \mathcal{C}(y_t, \hat{y}_t, \lambda) > \tau, \tag{1}$$

where $\mathcal{C}$ is a change detection mechanism.

B. Attack Detection

There are two main strategies that can be used to realize Eq. (1), that is, to determine the appropriate time to raise alarm: stateless and stateful. (i) In a stateless strategy, it is confined to monitor every single period's residual, and an alarm is raised for every single deviation, that is, if the residual exceeds a pre-determined threshold τ i.e, $r_t > \tau$. This kind of strategy has been shown to have increased false positives [24]. (ii) A stateful strategy, on the other hand, calculates the statistic S_t that keeps track of the historical changes of r_t. It raises alarm when there is a *persistent* deviation over time, i.e. $S_t > \tau$. This kind of strategy has been demonstrated to have decreased false positives [24].

We thus choose to develop a stateful strategy in our framework due to its lower false positive rates. There are usually two kinds of stateful strategies: time window and cumulative sum (CUSUM). (i) In a time-window-based method, the detector looks at the residuals within a time window of multiple control periods. (ii) A CUSUM-based method, on the other hand, efficiently tracks the cumulative sum of residuals of the whole history. The authors in [23] demonstrate that a CUSUM-based approach tends to be faster and more accurate than a time-window-based approach. Further, the former is more robust to attacks that are hard to be detected by other approaches such as attacks hidden in-between time windows and other stealthy attacks.

Hence, we present a CUSUM-based attack detection approach. The algorithm is augmented with a drift parameter, by tuning which the detection delay and false alarms can be changed. The algorithm outline is shown in Algorithm 1. We briefly explain the algorithm as follows.

Line 1 initializes the cumulative sum to zero. Line 2 calculates the residual between the observed sensor value y_t and the predicted sensor value $\hat{y}_t$ obtained from the Behavior

Algorithm 1: The CUSUM Algorithm.

Input: threshold τ, drift λ, observed sensor value y_t, predicted sensor value $\hat{y}_t$

Output: alarm time t_{alarm}

1 **Initialize:** $S_0 = 0$;
while $t > 0$ **do**
2 $r_t = y_t - \hat{y}_t$; // the residual of control period t.
3 $S_t = [S_{t-1} + |r_t| - \lambda]^+$; // the cumulative sum; // $[a]^+ = max\{a, 0\}$.
4 **if** $S_t > \tau$ **then**
5 $t_{alarm} = t$;
6 $S_t = 0$;
7 **end**
// the cumulative sum is greater than the threshold; at period t an alarm is raised; rest the sum.
8 **end**

Predictor. That is, this difference indicates how deviated the observed value is from the nominal estimate. Line 3 calculates the cumulative sum S_t at control period t, which is a non-negative value. Basically, it equals the cumulative sum at period $t-1$ plus the absolute value of the residual at t minus the drift parameter. The drift parameter is decided by the Drift Adaptor. As mentioned, selecting the appropriate drift parameter is an important aspect of the algorithm. It can impact both the detection delay and the number of false positives. Line 4-7 checks if the cumulative sum is larger than the pre-defined threshold. If yes, an alert t_{alarm} is raised, and S_t is reset to zero.

IV. Design of Behavior Predictor

In this section, we present the detailed design of behavior predictor. This component builds a data model of the system that captures physical invariants for the purpose of predicting sensor measurements.

Physical invariants are properties of the physical system that should always hold. The invariants are guarded by physical laws. One method to capture physical invariants is to use a physical system model. One disadvantage of this method is the requirement of adequate knowledge of accurate system dynamics, which may not be easy to attain.

In this paper, we approximate physical invariants using a deep learning technique instead. The approximated physical invariant will be used as the nominal behavior of the system. This technique treats the system as a black box and explores the correlation of multivariate sensor data. Our insight is that if the system operates normally and obeys physical laws, then

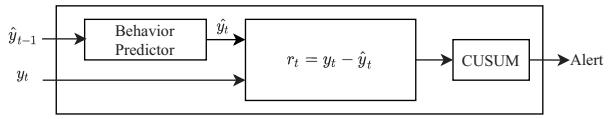


Fig. 2: Dataflow in attack detector.

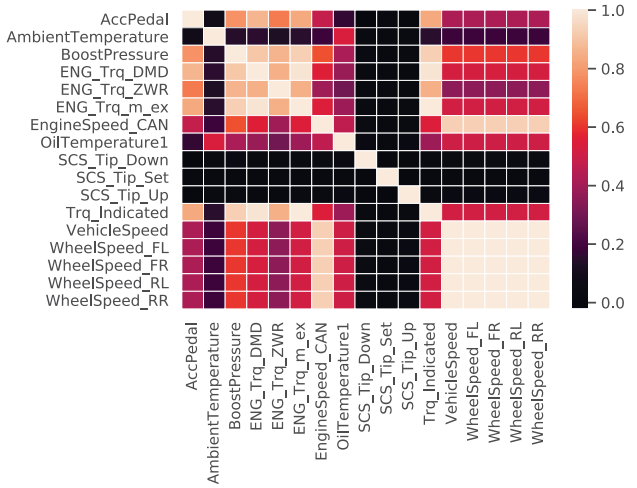


Fig. 3: Example confirming the wheel speed sensors in the dataset has strong correlation with the wheel speed, engine speed and boost pressure sensors.Table I shows the available sensors in the dataset.

the sensor data obtained from the CPS also indirectly obeys physical laws. Hence, with little knowledge of the system dynamics, our deep learning approach enables us to learn the behavior of the system in order to make accurate predictions.

A. Problem Formulation

In order to perform the non-trivial task of predicting nominal system behavior, we formulate the problem as a multivariate time series forecasting problem.

Given a fully observable system with n correlated sensors $Y = \{y_1, y_2, ..., y_T\}$ where $y_t \in \mathbb{R}^n$, we want to extract the natural redundancy that exists among the correlated sensors using a deep learning model $\mathcal{D}$, so that we can learn the nominal behavior of the system such that we can predict future sensor values $\hat{y}_{T+1}$. It is assumed that $\{y_1, y_2, ..., y_T\}$ will always be available whenever we predict $\hat{y}_{T+1}$. The input to the behavior predictor at time step T is formulated as $X_T = \{y_1, y_2, ..., y_T\} \in \mathbb{R}^{n \times T}$

$$\hat{y}_{T+1} = \mathcal{D}(X_T) \tag{2}$$

B. Pre-processing

Sensors on automotive CPS exhibit physical sensor correlation or natural redundancy [25]. We need to ensure the DL model is trained using only sensor data that are correlated. This component uses a statistical method to observe the natural redundancy in the dataset and also finds sensor data that are correlated but may not be obvious from domain knowledge.

The pre-processing component builds a correlation matrix based on Pearson's Correlation Coefficient (PCC) algorithm. Data variables or features are said to have a positive correlation when both variables move in tandem. That is, if one variable increases, the other variable also increases. A positive

correlation also holds when one variable decreases and the other variable decreases as well. Conversely, two variables have a negative correlation when one increases and the other variable decreases, and vice versa. PCC indicates a strong positive correlation with coefficient values that are close to +1.0 whereas a strong negative correlation has coefficients that are close to -1.0. Coefficient values close to 0 signifies that the two variables do not have any correlation. We select dataset features whose PCC values are either greater than 0.5 or less than -0.5 as input to model training. For example, to observe the sensors that have natural redundancy with the wheel speed sensor in the AEGIS CAN dataset (we describe this dataset in section VI-B), we created the heatmap shown in Fig. 3 based on the PCC values. In the figure, we observe the wheel speed sensors have a strong positive correlation with vehicle speed, engine speed, boost pressure, engine torque and oil temperature sensors.

C. Long- and Short-Term Time-Series Network (LSTNet)

Fig. 4 is an overview of the deep learning architecture used which is based on [34]. The interested reader is referred to [34] for details, here, we briefly describe each component. Mainly, the architecture consists of a convolutional neural network (CNN), a recurrent neural networks (RNN) as well as an autoregressive linear model.

CNN Component. The first layer of the deep learning framework is a CNN without pooling. It is tasked to extract the temporal patterns and the local relationship between the correlated sensor variables. This CNN layer is made up of a number of filters of width w and height n (the number of correlated sensor variables) with each k-th filter passing through the input matrix X to output a vector h_k:

$$h_k = RELU(W_k * X + b_k) \tag{3}$$

where $*$ is the convolution operation, W_k and b_k denote the weight parameter and bias respectively. $RELU$ activation function ensures values stay between 0 and 1. Each vector h_k is zero-padded on the left of the input matrix X to have a length of T. In the end, the convolutional layer outputs a matrix of size $d_c \times T$, where d_c is the number of filters. This output matrix is inputted into the recurrent component.

Recurrent Component. The recurrent component has two sub-components namely, gated recurrent unit (GRU) and recurrent-skip.

GRU is a specialized recurrent neural network (RNN) that is suited for modeling sequential data such as sensor readings [35]. Unlike artificial neural networks (ANN), GRU is able to store past information in addition to current inputs in order to determine current outputs. The ability to store past information in GRU is enabled by the state variables that it introduces i.e. the update and reset gates. At a time t, given the input minibatch $x_t \in \mathbb{R}^{m \times l}$ (where m is the number of examples in the minibatch and l is the number of inputs) and the previous hidden state $h_{t-1} \in \mathbb{R}^{m \times s}$ (where s is the number of hidden states), the reset gate $z_t \in \mathbb{R}^{m \times s}$ and update gate $u_t \in \mathbb{R}^{m \times s}$, candidate hidden state c_t and final state h_t are computed as,

$$\begin{aligned} z_t &= \sigma(x_t W_{xr} + h_{t-1} W_{hr} + b_r) \\ u_t &= \sigma(x_t W_{xu} + h_{t-1} W_{hu} + b_u) \\ c_t &= RELU(x_t W_{xc} + r_t \odot (h_{t-1} W_{hc}) + b_c) \\ h_t &= (1 - u_t) \odot c_t + u_t \odot h_{t-1} \end{aligned} \tag{4}$$

where $\odot$ is the element-wise (Hadamard) product, σ is the sigmoid function, W_{xr}, W_{xu}, W_{xc}, W_{hr}, W_{hu}, W_{hc} are the weight parameters, and b_r, b_u, b_c are bias parameters.

The output of the GRU layer is the hidden state h_t at each time step. Note that the use of GRU in the recurrent component allows the deep learning model to discard irrelevant previous sensor information and extract only the important ones that help to learn the nominal system behavior.

The second sub-component of the recurrent component is the recurrent skip component. This feature enables the architecture to memorize the repeated historic periodic pattern (such as daily, weekly patterns) in time series data. However, since the automotive CPS sensor data do not exhibit this periodic pattern, we do not turn it on in our experiment.

The output of the recurrent component is passed to a fully connected (FC) layer as shown in Fig. 4. FC combines its input to make a prediction result h_t^D is at time step t.

Autoregressive Component. This component addresses a deficiency found in the non-linear neural network components: convolutional and recurrent components. The scale of output in neural networks is known to be insensitive to the scale of its inputs [34]. Hence, given the non-periodic nature of sensor data, this deficiency diminishes the forecasting accuracy of the neural networks. This is solved by decomposing the final prediction into a linear component by using an autoregressive (AR) model which is formulated as,

$$h_t^L = \sum_{k=0}^{q^{ar}-1} W_k^{ar} v_{t-k} + b^{ar} \tag{5}$$

where $h_t^L \in \mathbb{R}^n$ is the forecasting result of the AR component, $W^{ar} \in \mathbb{R}^{q^{ar}}$ and $b^{ar} \in \mathbb{R}$ are the coefficients of the AR model such that q^{ar} is the size of input window over the input matrix. v_{t-k} is the past series values (lagged values).

At time step t, the DL model makes a prediction $\hat{y}_t$ by integrating the outputs of the neural network part and the AR component:

$$\hat{y}_t = h_t^D + h_t^L \tag{6}$$

Objective function. We use absolute loss (L1-loss) as the objective function which is formulated as:

$$\min_{\Theta} \sum_{t \in \Omega_{Train}} \sum_{i=0}^{n-1} |y_{t,i} - \hat{y}_{t,i}| \tag{7}$$

where Θ denotes the parameter set of our model, Ω_{Train} is the set of time stamps used for training.

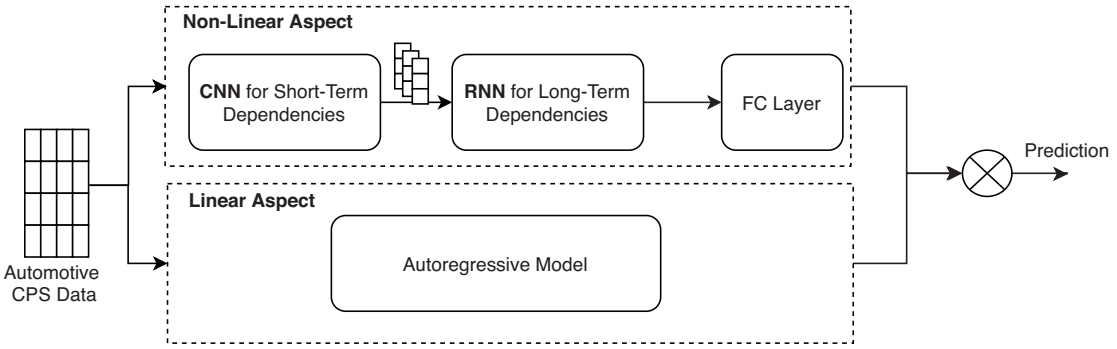


Fig. 4: Architecture of LSTNet model that learns both local and complex long-term dependencies in automotive CPS sensor values for attack detection.

Although squared error function is an option often used, experiment results in [34] indicate the absolute loss function is more robust.

V. Design of Drift Adaptor

In this section, we present the design of the drift adaptor. This component ensures attack detection occurs before a detection deadline.

The requisite detection deadline for an autonomous CPS varies with its physical environment. In other words, the deadline by which the attack has to be detected depends on the physical environment. The deadline can change as the physical environment varies. For instance, the deadline for detecting a wheel speed attack of a vehicle that is 50m away from an object it can crash into will be different from the situation where the crashing object is 200m away. Hence, there is a need for real-time attack detection that adapts its mechanism based on the physical environment or how the system is close to unsafe states, such that the detection delay will be less than the required detection deadline.

Another motivation is the trade-off between detection delay and false alarms in our experiment. The attack detector discussed above (in Section III) is augmented with a drift

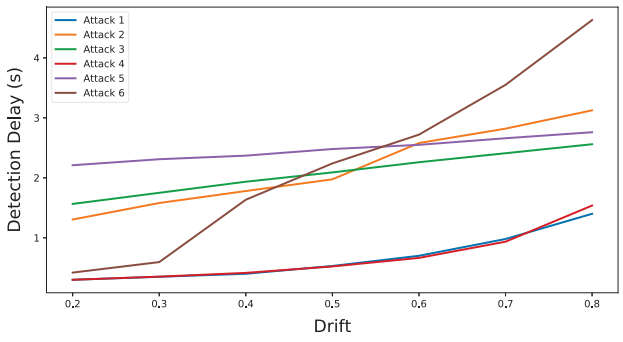


Fig. 5: The relationship between drift parameter and detection delay for various attack scenarios.

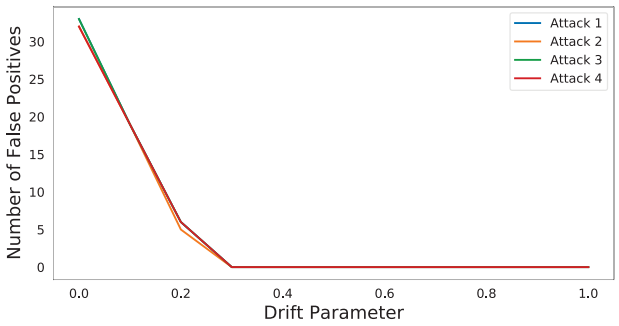


Fig. 6: Relationship between drift parameter and number of false positives.

parameter λ that can be adjusted to produce varying detection delays and false positives. Fig. 5 and Fig. 6 show how the drift parameter affects the detection delay and number of false positives. We note that as the drift parameter increases, the time to detection or detection delay increases while the number of false positives decreases. Hence, adjusting the drift parameter enables our attack detection mechanism to adapt its behavior for an appropriate trade-off while meeting the real-time constraint.

The Drift Adaptor component is made up of two sub-components: Deadline Estimator and Drift Analyzer. The deadline estimator determines the detection deadline whereas the drift analyzer determines the appropriate drift parameter.

A. Deadline Estimator

The detection deadline considered in this paper is the time in the future when the system may touch the unsafe set. We consider a time that is estimated in a conservative way, i.e., at a worst case. The authors of [36] propose a reachability-based deadline estimation method, but it requires knowing the system dynamics. By contrast, we propose a pure data-driven method towards this end.

The core idea of the proposed method is to first calculate the maximum change rate of the sensor value and then use it to estimate the shortest time when the system may touch the

unsafe set. The proposed method has two phases: offline and online.

(i) At the offline phase, we consider the collected time series of each individual sensor i, denoted as $\{y_1(i), y_2(i), ...y_T(i)\}$. The change rate of the sensor value of two adjacent periods is defined as

$$\Delta_t(i) = \frac{y_t(i) - y_{t-1}(i)}{\delta}. \tag{8}$$

Then using the collected time series, we use the following equations to calculate the maximum (Δ^+) and minimum (Δ^-) change rate.

$$\begin{aligned} \Delta^+(i) &= [max\{\Delta_t(i), 2 \leq t \leq T\}]^+, \\ \Delta^-(i) &= [min\{\Delta_t(i), 2 \leq t \leq T\}]^-, \end{aligned} \tag{9}$$

where $[a]^+ = max\{a, 0\}$ and $[a]^- = min\{a, 0\}$.

(ii) At the online phase, based on the fastest change rate given in Eq. (9), we can perform the following reachability analysis to estimate the detection deadline. At current time t, we calculate the reachable value for each sensor by

$$\begin{aligned} y_d^+(i) &= y_t(i) \times (1 + \Delta^+(i) \times \delta \times (d - t)), d > t, \\ y_d^-(i) &= y_t(i) \times (1 + \Delta^-(i) \times \delta \times (d - t)), d > t. \end{aligned} \tag{10}$$

The earliest time $D(i)$ when the value of sensor i may touch the unsafe set is

$$D(i) = min\{d | y_d^+(i) \in U(i) \vee y_d^-(i) \in U(i)\}, \tag{11}$$

where $U(i)$ is the unsafe set associated with sensor i. Finally, the detection deadline D is calculated by

$$D = min\{D(i) | 1 \leq i \leq n\}. \tag{12}$$

Note that our framework does not rely on any specific deadline estimation method, and is always applicable as long as a detection deadline is outputted.

B. Drift Analyzer

With a detection deadline D as input, the Drift Analyzer determines the best drift parameter that allows the attack to be detected before the deadline. For this component to function properly, we need to first establish the relationship between the detection delay and the drift parameter. This is achieved by performing offline profiling. Fig. 5 and Fig. 6 depict that there is a relationship among the drift parameter, detection delay and false positives. Armed with this information and the CUSUM tuning tools provided in [37], we are able to build a drift-parameter-detection delay pair that ensures we do not exceed the acceptable false positive rate. In other words, We build a lookup table based on the offline profiling results. To perform its online adaptation functionality, the Drift Analyzer simply queries the lookup table to output the drift parameter that adjusts the detection delay to meet the given detection deadline.

TABLE I: Some sensors in the dataset used in experiment.

CAN bus Sensors	GPS Sensors	IMU Sensors
ASR	Acceleration	Accelerometer_X
AccPedal	Current_sec	Accelerometer_Y
AirIntakeTemperature	Direction	Accelerometer_Z
AmbientTemperature	Distance	Body_acceleration_X
BoostPressure	Velocity	Body_acceleration_Y
BrkVoltage		Body_acceleration_Z
EngineSpeed_CAN		G_force
EngineTemperature		Magnetometer_X
Kickdown		Magnetometer_Y
MFS_Tip_Down		Magnetometer_Z
MFS_Tip_Up		Velocity_X
SteerAngle		Velocity_Y
Trq_FrictionLoss		Velocity_Z
Trq_Indicated		
VehicleSpeed		
WheelSpeed_FL		
WheelSpeed_FR		
WheelSpeed_RL		
WheelSpeed_RR		
Yawrate		

ASR = Acceleration Slip Regulation, ACC = Acceleration, BRK = Break, MFS = Misfiring System, TRQ = Torque, FL = Front Left, FR = Front Right, RL = Rear Left, RR = Rear Right, G = Gravity

VI. Evaluation

A. Implementation and Experimental Setup

We implemented our deep learning model in Python, utilizing PyTorch Deep Learning framework. We train the model on Ubuntu 18.04 64-bit with sixteen Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz CPUs, two Nvidia GeForce GTX 1080 GPUs and 64 GB RAM. We follow a 60/20/20 proportions for splitting the original dataset into training/validation/test sets. The experimental model is made up of 100 hidden CNN layers and 100 hidden RNN layers. The model was trained for 100 epochs. Metrics used for the test accuracy were Root Relative Square Error (RSE) and Relative Absolute Error (RAE). The accuracy for our experimental model was 0.0032 (RSE) and 0.0018 (RAE).

B. Dataset Description

We used the publicly-available real-world automotive CAN bus dataset from the AEGIS Big Data Project [33] [1] for our experiment. The sensor data, sampled at 20Hz, was collected during trips in the same passenger vehicle. More than 40 sensor measurements were collected including but not limited to those listed in Table I. Specifically, the data contains about 2.5 hours of driving data (about 160,000 data points).

C. Attacks

The dataset does not include any anomalous events or scenarios, hence we manually modify portions of the dataset to simulate physical attacks that achieve similar goals of a real attacker. Based on the attacks discussed in section II and the attacks in [38], we evaluate our work under (1) modification, (2) delay and (3) replay attacks.

[1] https://zenodo.org/record/3267184#.X5YtpIhKg2x

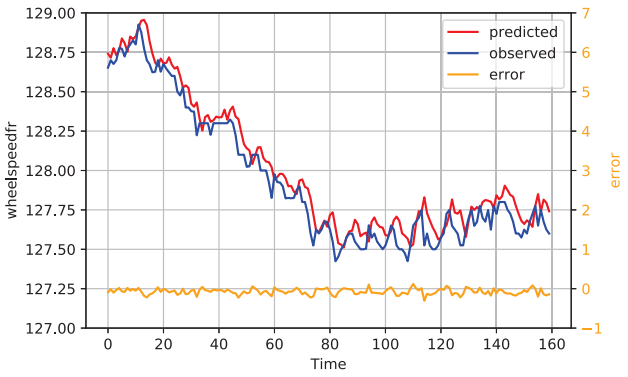


Fig. 7: Front-Left wheel speed sensor measurement showing the predicted and observed values.

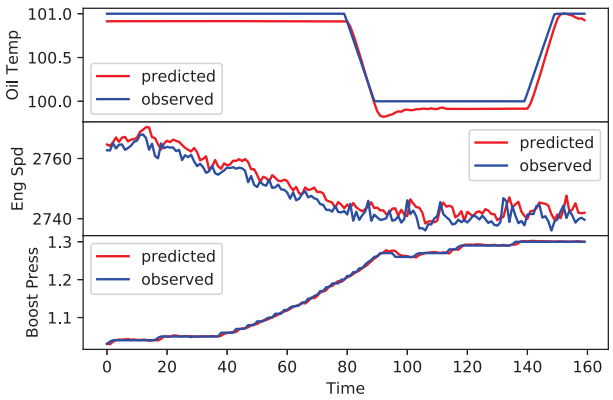


Fig. 8: Observed and predicted values for the oil temperature, engine speed and boost pressure sensors.

We simulate four attack scenarios under modification attacks and one each for the delay and replay attacks. **Attack 1** adds a fixed value to the sensor readings for a period of time. This simulates the attacker spoofing the sensor measurement as done in Fake Data Injection (FDI) attacks. **Attack 2** sets the sensor reading to a fixed value indicating a spoofing attack that spoofs sensor readings to a specific value. **Attack 3** incrementally changes sensor measurement. Here the attacker has a target value to spoof the sensor, yet he does not set the value right away. Rather, he gradually adds small values (e.g 0.01 kph) to current sensor readings until the target spoofing value is reached. In real life, an attacker might use this strategy with the intent of evading detection mechanisms. **Attack 4** sends both normal sensor values and malicious/fake sensor values alternately and repeatedly. Like Attack 3, a real attack might use this tactic to mislead attack detection mechanisms.

Attack 5 mimics delay attack where the attacker causes a 10s delay in sensor data transmission. **Attack 6** is a replay attack where the sensor data from a previous time are replayed.

D. Experiments and Results

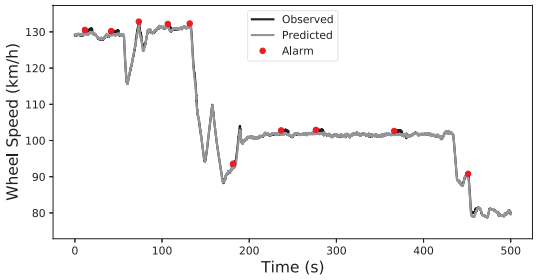
We perform various experiments to evaluate the effectiveness and efficiency of our proposed framework.

Experiment I: The first experiment tests if the behavior predictor component was able to capture the behavior of the automotive CPS accurately. Fig. 7 shows the normal behavior of the front-left wheel speed sensor. It can be seen in the figure that the behavior predictor's prediction closely matches the observed sensor measurement operating under normal conditions. A similar plot for the engine speed, oil temperature and boost pressure sensors shown in Fig. 8 further indicate the behavior predictor is able to capture the system's nominal behavior. The error shown in Fig.7 has a close to zero-average, an indication that the predictor method is not biased. Further residual analysis shows the residuals follow a normal Gaussian distribution and do not have any trend, cyclic or seasonal structure in the error plot.

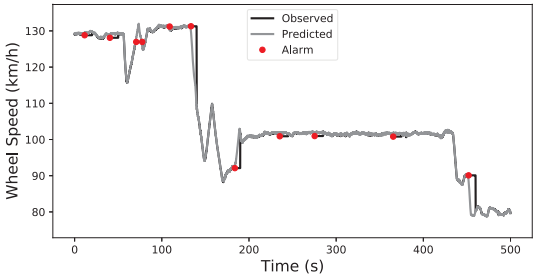
Experiment II: We evaluate the attack detector component in this experiment. The component is tasked to detect the attack scenarios described in VI-C. We randomly placed 10 simulated attacks in each case. Fig. 9 shows the proposed framework is effective in detecting various attack scenarios. In the figure, the red dots indicate the points where the CUSUM-based attack detector raises an alert for the attacks. It can also be observed that the detector raises alarm only when the abrupt change is significant and has persisted for a while thus reducing flagging transients faults as attacks. For instance, a close-up look at one of the attack points (see Fig. 10), shows the detector observed an abrupt change in speed at time 39.95s (indicated by the green arrow) but did not raise alarm immediately until 42.1s.

Experiment III: In this experiment, we measure and analyze the false-positive (FP) and false-negative (FN) rates under various drift and threshold monitoring parameters. FP is measured by inputting normal data (no attack) into the framework and counting the number of false alarms rate. We measure FN by inputting data containing simulated attacks and counting the number of attacks that the framework missed raising an alert for. Note here that *miss* means that an alert was not raised within the duration of the attack. Therefore, alarms that occurred shortly after the attack ended were not counted. The results in Fig. 11 and Fig. 12 show the FP and FN rates respectively for this experiment. In the legend of the figures, A1, A2...A6 represent the various attack scenarios discussed above, the numbers (3, 4, 5) represent the threshold monitoring parameter. For example, A1-3 represents Attack 1 being monitored with a threshold of 3. In FP analysis, we observe that the CUSUM drift parameter with value 0 produces very high FP rates. However, the FP rate plummets with drift values greater than 0. The FN rate results, on the other hand, show the drift parameter ranging from 0.2 to 0.8 produce zero rates for all attack case scenarios. This implies that for most attacks, the behavior of the detector framework can be adapted to meet attack deadlines whilst still maintaining very low FN rates. However, beyond that range (0.2 - 0.8), we observe the FN increases.

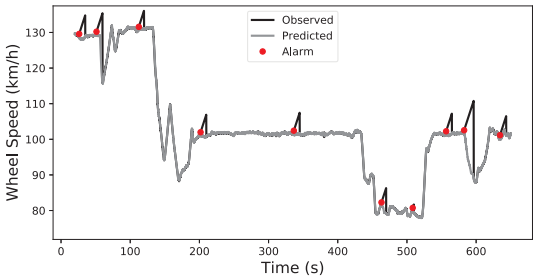
Experiment IV: This experiment measures the detection delay φ i.e. the time it takes to detect the attack after its



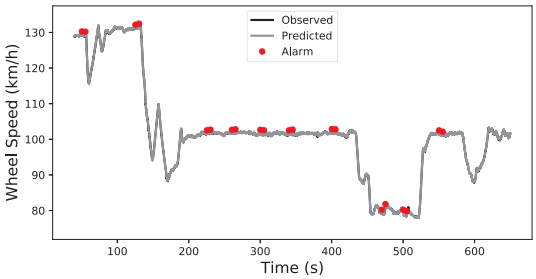
(a) Attack 1.



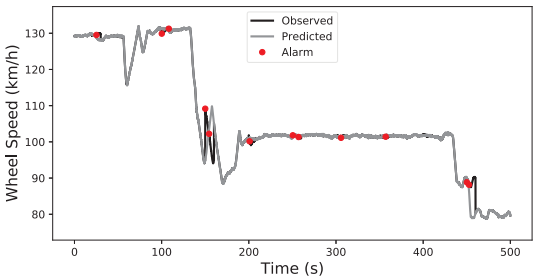
(b) Attack 2



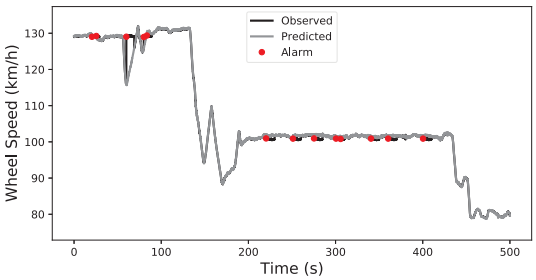
(c) Attack 3



(d) Attack 4



(e) Attack 5



(f) Attack 6

Fig. 9: Results of the attack detector's detection of various attack scenarios discussed in section VI-C

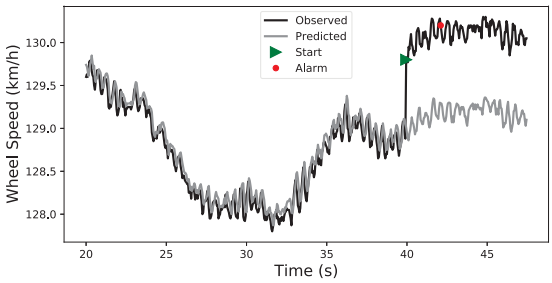


Fig. 10: A close-up look at one case of attack 1 detection.

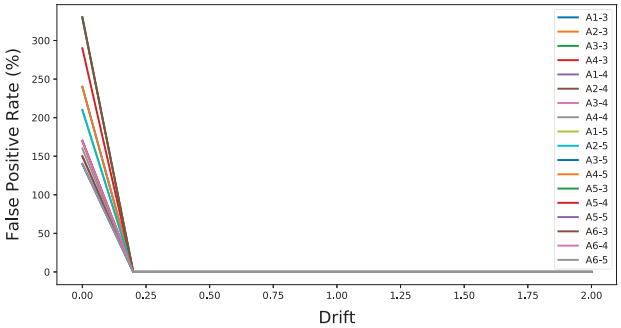


Fig. 11: False positive rate for the various attack scenarios under different monitoring parameters (drift and threshold).

launch. This metric allows us to evaluate the time needed for our attack detection framework to disclose or alert an attack. If an attack starts at time k_s, and the attack detection mechanism detects it at time k_d, φ is defined as: $\varphi = k_d - k_s$. The lower the value of φ, the better the attack detection mechanism and as such, reduces the impact of the attack.

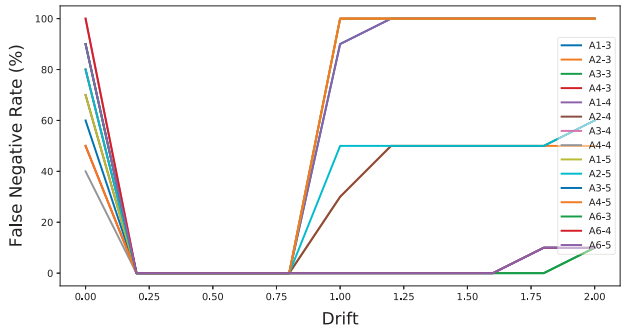


Fig. 12: False negative rate for the various attack scenarios under different monitoring parameters (drift and threshold).

Table II shows the results of the detection delay φ for the various simulated attacks (see section VI-C). Note that the simulated attacks lasted for 10s. The result in the table suggests the attacks were detected while the system was being attacked rather than after the attack ended. While this is desirable, we show in a subsequent experiment that it is more desirable and important for real-time systems to detect an attack *before a detection deadline*. Further, the result shows the relationship between the CUSUM drift parameter and the detection delay. The detection delay increases as the drift parameter increases.

Experiment V: This experiment analyses the effect of adaptive detection. For real-time systems, it is not only desirable but required that the attack detection mechanisms are able to detect attacks before the detection deadline D, $0 \leq \varphi \leq D$. Fig. 13 shows how the real-time adaptive detection enables Attack 3 to be detected under different deadlines. In the figure, we observe that, in order to meet Deadline 1 (1sec), the drift parameter has to be adjusted to a value not greater than 0.25. Though Attack 3 can be detected with a drift parameter of say 0.8, it cannot satisfy Deadline 1 because its corresponding detection delay is 1.8 seconds.

Experiment VI: This experiment compares a fixed time-window approach with our real-time adaptive attack detection approach. The goal is to show how our framework adapts it behavior based on the drift parameter in order to meet an attack deadline.

The time-window implementation used in this experiment is similar to [22]'s attack detector monitoring algorithm. It sums up the square errors between the observed and the prediction. When the time window expires, it determines if the accumulated mean square exceeds a threshold. The accumulated sum is reset when the time window expires. Note that the time-window approach can only raise an alarm after its time-window expiration.

We compare the two approaches based on the attacks described in Section VI-C and the case scenario depicted in Fig. 14. In this case scenario, an attack occurs at $40s$ with a detection deadline estimated at $45s$. The red dots in the figure represent the alarm raised by our framework whereas the

TABLE II: Detection delay in seconds for the attack scenarios. A1 refers to Attack 1, A2 refers to Attack 2 and so on.

Drift	A1	A2	A3	A4	A5	A6
0.2	0.3	1.31	1.57	0.30	2.21	0.42
0.3	0.35	1.58	1.75	0.35	2.31	0.60
0.4	0.4	1.78	1.94	0.42	2.37	1.64
0.5	0.53	1.98	2.09	0.52	2.48	2.24
0.6	0.7	2.58	2.26	0.67	2.55	2.72
0.7	0.98	2.82	2.41	0.94	2.66	3.55
0.8	1.40	3.13	2.56	1.54	2.76	4.63

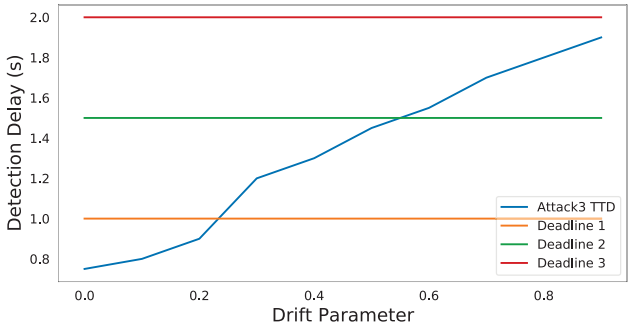
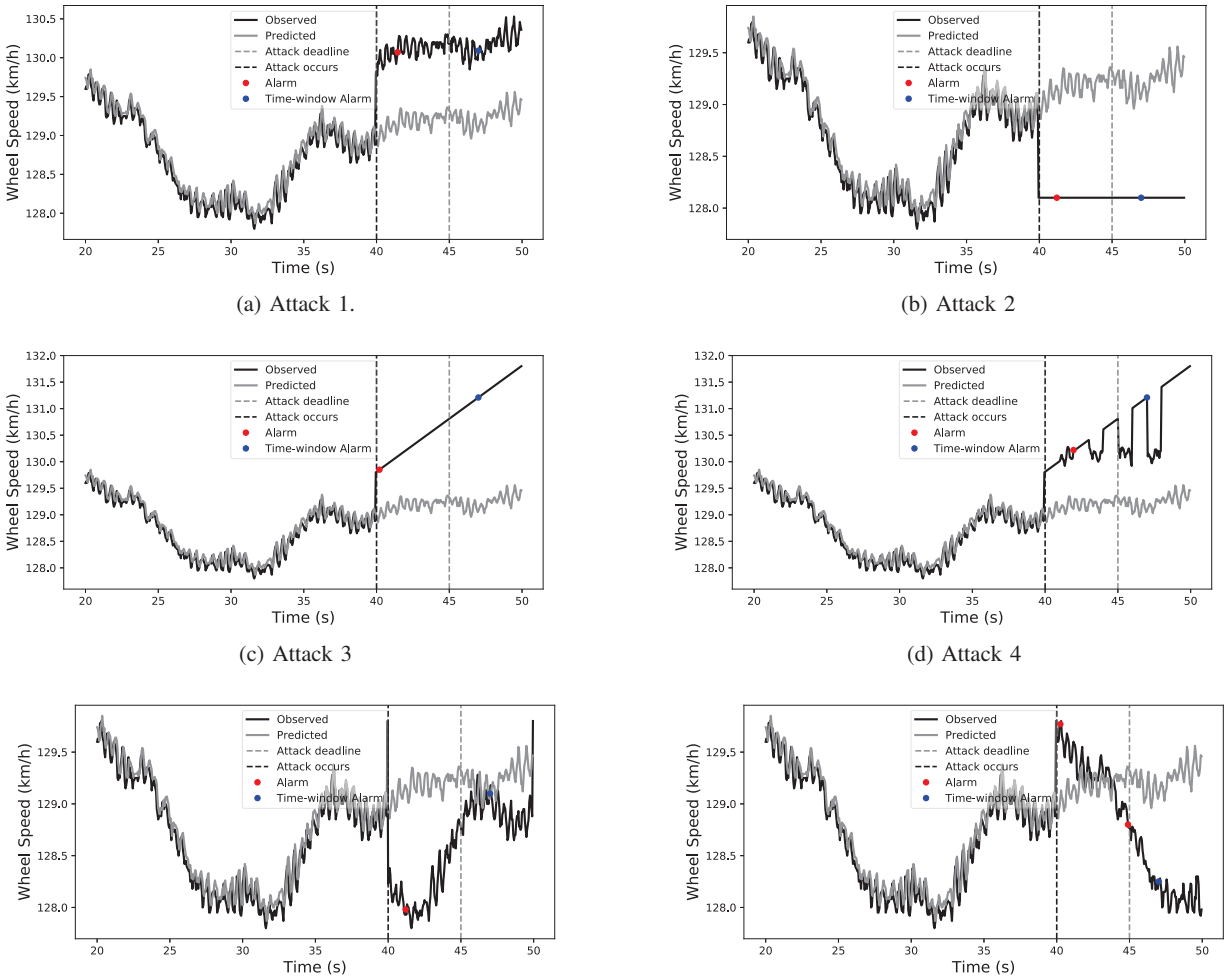


Fig. 13: Adaptive detection for Attack 3.

blue dot refers to the alarm raised by the time-window attack detector approach. We observe that the time-window approach rightly determines that an attack occurred in all cases, however, the alert is raised after the detection deadline. In a real-world situation, this would mean the attack detection alert is raised after the damage has occurred. Our framework, on the other, raises an alarm before the deadline.

Experiment VII: We compare our work with a recent anomaly detector that closely relates to our work [25]. Like our work, the researchers exploit the natural redundancy that exists among heterogeneous sensors and they also employ deep learning techniques (deep autoencoder) to detect attacks. Whereas they focus only on the rightness of attack detection, we focus on detecting attacks before a detection deadline by adapting the attack detection mechanism. More importantly, in order to decide to raise an attack alert, their approach monitors only one control period. Due to the adaptive nature of our approach to meet a deadline, the number of control periods that it monitors varies. Their detection mechanism raises an alert when the reconstruction error of the decoder is above a certain threshold. We trained a model based on [25] and tuned the hyper-parameters with our dataset for a fair comparison. The deep autoencoder attack detector is tasked to detect the same attack scenarios we subjected our approach to in Experiment II. Fig. 15 shows the results. The green marks are the data points that represent the attack. Comparing this results with our results shown in Fig. 9, the detector in [25] produces high false alarms.



(a) Attack 1.

(b) Attack 2

(c) Attack 3

(d) Attack 4

(e) Attack 5

(f) Attack 6

Fig. 14: Comparing our framework with a fixed time-window approach. In all attack scenarios (see VI-C), the attack occurs at 10s and has a deadline set at 45s.

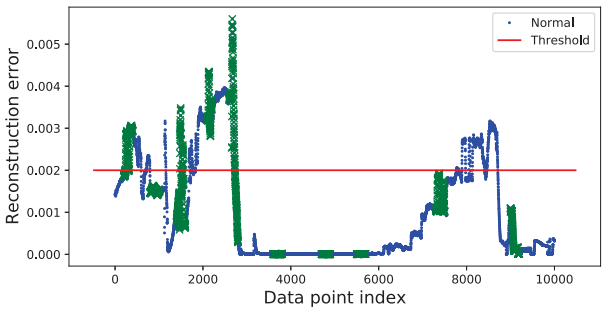


Fig. 15: Attack detection in deep automated [25] attack detector.

VII. Discussion

A. Stealthy Attacks

While our proposed system effectively detects physical attacks against automotive CPS we do not rule out completely the possibility of it being vulnerable to stealthy attacks. In this attack, the attacker spoofs sensor values that do not exceed the determined threshold, hence the attack detection system raises no alarm. Gradually, the attacker is able to deviate the CPS to his desired target. References [23] and [31] note that this weakness is also found in physics-based attack detection (PBAD) systems. In the real world, a stealthy attack is hard to launch as it requires very detailed knowledge about the system dynamics and ensuring that all the laws of physics are obeyed [22]. On one hand, our proposed detection framework provides some defense as the Behavior Predictor component learns the system dynamics from multiple heterogeneous sensors. To evade our framework, the attacker may have to launch spoofing attacks against all the heterogeneous sensors simultaneously such that it maintains the natural correlation among the sensors that the proposed framework also learned. Achieving such a sophisticated attack in the real world is hard since

each sensor attack requires specific tools and equipment to successfully launch. On the other hand, we agree with [31] that a combination of detection schemes can also be implemented to mitigate stealthy attacks. We suggest combining a deep learning approach like our work with PBAD approaches can be a viable solution against stealthy attacks.

B. State Estimation and Attack Response

Our work has focused on physical sensor attack detection without attack response. Once our framework detects an attack, the behavior predictor can also be used to predict values that can be used for state estimation. The state estimation can be forwarded to the controller for recovery control. However, there are more viable recovery solutions such as [39] that recovers the CPS from an attack so that continual functioning is attained.

VIII. Related Work

This section discusses research works related to attack detection and sensor correlation.

A. Attack detection

Existing works on cyber-physical systems attack detection can be categorized based on techniques that are used.

Redundancy-based: Works that use this approach [30], [40]–[42] detect attacks by using multiple system components. The duplicated system component may be software (e.g controller) or hardware (e.g. sensors). The states or outputs of each of the duplicated system components are cross-checked at runtime. In spite of its effectiveness, this approach leads to increased cost, weight, power, space requirement and system complexity.

Signature-based: The works that use this approach [43], [44] monitor runtime patterns and compare them with a pre-maintained dictionary that contains known attack types or attack patterns. For it to be effective, the dictionary needs to have the latest attack patterns. These methods are known to be fast and have low false positives rates, however, they are not effective in handling zero-day attacks [22], [28]

Behavioral rule-based: Behavioral rule-based techniques [45]–[47] models the normal system operations by using a specification. The program state transitions or execution time constraints are usually modeled in this approach.

Physics-based: This approach detects attacks by monitoring the physics of cyber-physical system. It is an area of research that is attracting a lot of attention. [24] surveys works that use this approach for cyber-physical systems in general, whereas [48] surveys works that use this approach specifically for autonomous vehicles. Recently, [22] and [23] applied this technique to detect physical sensor attacks. During the first of its two steps, Physics-Based Attack Detection (PBAD) approaches extract the physical invariant of the system and use it to model the system. Although we do not extract the physical invariant directly, we indirectly use deep learning to learn about them. Like other approaches, in the second step, PBAD compares the model predictions with the observed values and raise alarm when observed states exceed a threshold.

Machine/Deep Learning: Machine Learning (ML) and Deep learning (DL) techniques have been employed in many CPS attack detection works lately [25], [27], [28], [38], [49]–[56]. These solutions build a data model by using the system data to train a machine or deep learning model. The models are often used to make predictions of CPS measurements such as sensors. ML and DL methods require a large amount of data to build accurate models. While supervised ML methods require both labeled normal and attack training data, unsupervised methods often process only normal training data. Our solution uses an unsupervised deep learning approach. Our work is distinguished by the incorporation of attack detection deadline estimation and adaptive attack detection mechanism.

B. Sensor correlation

Researchers have explored the correlations among sensors in their solutions especially for anomaly detection. The authors in [57] utilize pairwise correlation between vehicle sensors to detect anomalies caused by sensor faults or attacks. [58] builds on the idea of [57] by modeling the subtle nonlinear relationships in CAN data without knowing its functional meaning. [38] demonstrates an in-vehicle intrusion detection system (IDS) for detecting data spoofing attacks. The solution uses a regression learning approach to learn CAN bus sensor data correlation. Liu et.al [59] extract causal interactions among the sub-systems of a CPS and present a spatio-temporal graphical modeling approach to detect anomalies for a heating system. Other works that explore sensor correlation include [60]–[62]. Our work also exploits the correlation among sensors, however, our work is unique. All of these works focused on detecting the attacks instead of when the attack is detected. We provide a real-time attack detection that adjusts the detection delay so that it meets the detection deadline.

IX. Conclusion

The safety-critical roles that autonomous CPS play require that they are protected from physical sensor attacks in a timely manner. In this paper, we have presented a novel deep learning-based real-time adaptive attack detection framework. The framework dynamically adjusts the detection delay via the drift parameter so that the detection deadline can be satisfied. We achieve this through the three main components: behavior predictor, attack detector and drift adaptor. Behavior Predictor uses a deep learning technique to approximate the system's physical invariant for the purpose of predicting nominal sensor values. Attack Detector employs the CUSUM algorithm for attack detection. We have introduced a detection deadline estimation method in the drift adaptor component. We evaluated our proposed framework using a real-world dataset. We compared our work with a time-window attack detector approach and a recent deep learning-based solution. The results show that while our work is not free from false alarms, it is able to adjust its behavior to meet attack deadlines.

ACKNOWLEDGMENT

This work was supported in part by NSF CCF-1720579 and NSF CCF-2028740. We would like to thank the anonymous reviewers for their constructive comments. We also would like to thank Kenneth Fletcher, Romesh Satish Prasad and Carlos Omar Espinoza Zelaya for their assistance in undertaking this project.

REFERENCES

[1] Amazon, *Amazon Prime Air*, 2020 (accessed June 29, 2020). https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011s.

[2] DroneUp, *Best aerial photography drones for business in 2020: DJI, Freefly, Skydio, and more*, 2020 (accessed June 29, 2020). https://www.zdnet.com/article/best-aerial-photography-drones/.

[3] DroneFly, *Police Drone Infographic*, 2020 (accessed June 29, 2020). https://www.dronefly.com/police-drone-infographic/.

[4] S. Haley, *How Police Forces Are Using Drones to Keep Officers Out of The Line of Fire*, 2020 (accessed June 29, 2020). https://www.govtech.com/products/How-police-forces-are-using-drones-to-keep-officers-out-of-the-line-of-fire.aspx?admgarea=mag&m=1.

[5] P. Gutierrez, *Infrastructure Inspection - UAS Are All Over It*, 2020 (accessed June 29, 2020). https://insideunmannedsystems.com/infrastructure-inspection-uas-are-all-over-it/.

[6] DroneUp, *Complete Drone Solutions*, 2020 (accessed June 29, 2020). https://www.droneup.com/.

[7] T. Higgins and M. Grossman, *Amazon to Acquire Self-Driving Startup Zoox*, 2020 (accessed June 29, 2020). https://www.wsj.com/articles/amazon-to-acquire-self-driving-startup-zoox-11593183986.

[8] K. Piper, *It's 2020. Where are our self-driving cars?*, 2020 (accessed June 29, 2020). https://www.wsj.com/articles/amazon-to-acquire-self-driving-startup-zoox-11593183986.

[9] C. H. Kim, T. Kim, H. Choi, Z. Gu, B. Lee, X. Zhang, and D. Xu, "Securing real-time microcontroller systems through customized memory view switching.," in *NDSS*, 2018.

[10] F. B. Cohen, "Operating system protection through program evolution.," *Computers & Security*, vol. 12, no. 6, pp. 565–584, 1993.

[11] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks.," in *USENIX security symposium*, vol. 98, pp. 63–78, San Antonio, TX, 1998.

[12] A. Cui and S. J. Stolfo, "Defending embedded systems with software symbiotes," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 358–377, Springer, 2011.

[13] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in *Proceedings of the 11th ACM conference on Computer and communications security*, pp. 298–307, 2004.

[14] A. A. Clements, N. S. Almakhdhub, K. S. Saab, P. Srivastava, J. Koo, S. Bagchi, and M. Payer, "Protecting bare-metal embedded systems with privilege overlays," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 289–303, IEEE, 2017.

[15] L. Pike, P. Hickey, T. Elliott, E. Mertens, and A. Tomb, "Trackos: A security-aware real-time operating system," in *International Conference on Runtime Verification*, pp. 302–317, Springer, 2016.

[16] A. H. Rutkin, "spoofers use fake gps signals to knock a yacht off course," *MIT Technology Review*, 2013.

[17] Y. Shoukry, P. Martin, P. Tabuada, and M. Srivastava, "Non-invasive spoofing attacks for anti-lock braking systems," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 55–72, Springer, 2013.

[18] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote attacks on automated vehicles sensors: Experiments on camera and lidar," *Black Hat Europe*, vol. 11, p. 2015, 2015.

[19] K. Marzullo, "Tolerating failures of continuous-valued sensors," *ACM Transactions on Computer Systems*, vol. 8, no. 4, pp. 284–304, 1990.

[20] P. Lu, L. Zhang, B. B. Park, and L. Feng, "Attack-resilient sensor fusion for cooperative adaptive cruise control," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3955–3960, 2018.

[21] R. Ivanov, M. Pajic, and I. Lee, "Attack-resilient sensor fusion for safety-critical cyber-physical systems," *ACM Transactions in Embedded Computing Systems*, vol. 15, no. 1, p. 21, 2016.

[22] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Deng, "Detecting attacks against robotic vehicles: A control invariant approach," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 801–816, 2018.

[23] R. Quinonez, J. Giraldo, L. Salazar, and E. Bauman, "Savior: Securing autonomous vehicles with robust physical invariants," *Usenix*, 2020.

[24] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A survey of physics-based attack detection in cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[25] T. He, L. Zhang, F. Kong, and A. Salekin, "Exploring inherent sensor redundancy for automotive anomaly detection," *DAC2020*, 2020.

[26] P. Guo, H. Kim, N. Virani, J. Xu, M. Zhu, and P. Liu, "Exploiting physical dynamics to detect actuator and sensor attacks in mobile robots," *arXiv preprint arXiv:1708.01834*, 2017.

[27] A. Abbaspour, K. K. Yen, S. Noei, and A. Sargolzaei, "Detection of fault data injection attack on uav using adaptive neural network," *Procedia computer science*, vol. 95, pp. 193–200, 2016.

[28] K. N. Junejo and J. Goh, "Behaviour-based attack detection and classification in cyber physical systems using machine learning," in *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pp. 34–43, 2016.

[29] J. Shin, Y. Baek, Y. Eun, and S. H. Son, "Intelligent sensor attack detection and identification for automotive cyber-physical systems," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, IEEE, 2017.

[30] J. Park, R. Ivanov, J. Weimer, M. Pajic, and I. Lee, "Sensor attack detection in the presence of transient faults," in *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, pp. 1–10, 2015.

[31] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the impact of stealthy attacks on industrial control systems," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1092–1105, 2016.

[32] R. Tunga, C. Murguia, and J. Ruths, "Tuning windowed chi-squared detectors for sensor attacks," in *2018 Annual American Control Conference (ACC)*, pp. 1752–1757, IEEE, 2018.

[33] C. Kaiser, A. Stocker, and A. Festl, *Automotive CAN bus data: An Example Dataset from the AEGIS Big Data Project*, July 2019.

[34] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long-and short-term temporal patterns with deep neural networks," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 95–104, 2018.

[35] B. A. Kwapong, R. Anarfi, and K. K. Fletcher, "Personalized service recommendation based on user dynamic preferences," in *International Conference on Services Computing*, pp. 77–91, Springer, 2019.

[36] L. Zhang, X. Chen, F. Kong, and A. A. Cardenas, "Real-time recovery for cyber-physical systems using linear approximations," in *41st IEEE Real-Time Systems Symposium (RTSS)*, IEEE, 2020.

[37] C. Murguia and J. Ruths, "Cusum and chi-squared attack detection of compromised sensors," in *2016 IEEE Conference on Control Applications (CCA)*, pp. 474–480, IEEE, 2016.

[38] H. Li, L. Zhao, M. Juliato, S. Ahmed, M. R. Sastry, and L. L. Yang, "Poster: Intrusion detection system for in-vehicle networks using sensor correlation and integration," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2531–2533, ACM, 2017.

[39] F. Kong, M. Xu, J. Weimer, O. Sokolsky, and I. Lee, "Cyber-physical system checkpointing and recovery," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 22–31, IEEE, 2018.

[40] M.-K. Yoon, B. Liu, N. Hovakimyan, and L. Sha, "Virtualdrone: virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems," in *Proceedings of the 8th International Conference on Cyber-Physical Systems*, pp. 143–154, 2017.

[41] F. Fei, Z. Tu, R. Yu, T. Kim, X. Zhang, D. Xu, and X. Deng, "Cross-layer retrofitting of uavs against cyber-physical attacks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 550–557, IEEE, 2018.

[42] M.-K. Yoon, S. Mohan, J. Choi, J.-E. Kim, and L. Sha, "Securecore: A multicore-based intrusion detection architecture for real-time embedded systems," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 21–32, IEEE, 2013.
[43] W. Gao and T. H. Morris, "On cyber attacks and signature based intrusion detection for modbus based industrial control systems," *Journal of Digital Forensics, Security and Law*, vol. 9, no. 1, p. 3, 2014.
[44] S. Kaur and M. Singh, "Automatic attack signature generation systems: A review," *IEEE Security & Privacy*, vol. 11, no. 6, pp. 54–61, 2013.
[45] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo, "Sandboxing controllers for cyber-physical systems," in *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pp. 3–12, IEEE, 2011.
[46] R. Mitchell and R. Chen, "Adaptive intrusion detection of malicious unmanned air vehicles using behavior rule specifications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 5, pp. 593–604, 2013.
[47] R. Mitchell and R. Chen, "Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 1, pp. 16–30, 2014.
[48] F. Akowuah and F. Kong, "Physical invariant based attack detection for autonomous vehicles: Survey, vision, and challenges," in *The Fourth International Conference on Connected and Autonomous Driving (MetroCAD 2021)*, IEEE, 2021.
[49] Q. Shen, B. Jiang, P. Shi, and C.-C. Lim, "Novel neural networks-based fault tolerant control scheme with fault alarm," *IEEE transactions on cybernetics*, vol. 44, no. 11, pp. 2190–2201, 2014.
[50] K. Paridari, N. O'Mahony, A. E.-D. Mady, R. Chabukswar, M. Boubekeur, and H. Sandberg, "A framework for attack-resilient industrial control systems: Attack detection and controller reconfiguration," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 113–128, 2017.
[51] M. Kravchik and A. Shabtai, "Detecting cyber attacks in industrial control systems using convolutional neural networks," in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, pp. 72–83, 2018.
[52] J. Goh, S. Adepu, M. Tan, and Z. S. Lee, "Anomaly detection in cyber physical systems using recurrent neural networks," in *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, pp. 140–145, IEEE, 2017.
[53] J. Inoue, Y. Yamagata, Y. Chen, C. M. Poskitt, and J. Sun, "Anomaly detection for a water treatment system using unsupervised machine learning," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 1058–1065, IEEE, 2017.
[54] P. Nader, P. Honeine, and P. Beauseroy, "Mahalanobis-based one-class classification," in *2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2014.
[55] O. Al-Jarrah and A. Arafat, "Network intrusion detection system using neural network classification of attack behavior," *Journal of Advances in Information Technology Vol*, vol. 6, no. 1, 2015.
[56] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*, vol. 89, Presses universitaires de Louvain, 2015.
[57] A. Ganesan, J. Rao, and K. Shin, "Exploiting consistency among heterogeneous sensors for vehicle anomaly detection," tech. rep., SAE Technical Paper, 2017.
[58] Z. Tyree, R. A. Bridges, F. L. Combs, and M. R. Moore, "Exploiting the shape of can data for in-vehicle intrusion detection," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5, IEEE, 2018.
[59] C. Liu, S. Ghosal, Z. Jiang, and S. Sarkar, "An unsupervised spatiotemporal graphical modeling approach to anomaly detection in distributed cps," in *Proceedings of the 7th International Conference on Cyber-Physical Systems*, ICCPS '16, IEEE Press, 2016.
[60] Z. Wang, F. Guo, Y. Meng, H. Li, H. Zhu, and Z. Cao, "Detecting vehicle anomaly by sensor consistency: An edge computing based mechanism," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, IEEE, 2018.
[61] F. Guo, Z. Wang, S. Du, H. Li, H. Zhu, Q. Pei, Z. Cao, and J. Zhao, "Detecting vehicle anomaly in the edge via sensor consistency and frequency characteristic," *IEEE Transactions on Vehicular Technology*, 2019.
[62] P. Sharma, J. Petit, and H. Liu, "Pearson correlation analysis to detect misbehavior in vanet," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5, IEEE, 2018.